Package: phenoscapR (via r-universe)

November 3, 2026

Title Read, Analyse, and Visualise Single-Cell Spatial Biology Data

Version 0.2.0.9001

Description Tools for importing, processing, analysing, and visualising single-cell spatial biology data from multiplexed imaging platforms. Supports cell-table quality control, marker normalisation, phenotype summaries, spatial neighbourhood statistics such as Ripley's K function (Ripley (1977) <doi:10.1111/j.2517-6161.1977.tb01615.x>) and Moran's I (Moran (1950) <doi:10.2307/2332142>), and explicit sample and coordinate-frame registries. Patient-level inference enumerates label allocations for declared contrasts (exact permutation tests; Pitman (1937) <doi:10.2307/2984124>) and retains unavailable hypotheses when adjusting for multiple testing. Grouped summaries expose missingness, eligibility, and denominator provenance. Optional canonical cell-table import uses 'cellspecR' <https://github.com/CTTIR/cellspecR>; optional Self-Organising Nebulous Growths (SONG) embeddings (Senanayake et al. (2021) <doi:10.1109/TNNLS.2020.3023941>) use 'songR' <https://github.com/CTTIR/songR>. Both optional packages are available from <https://cttir.r-universe.dev>.

License MIT + file LICENSE

URL https://github.com/CTTIR/phenoscapR, https://cttir.github.io/phenoscapR/

BugReports https://github.com/CTTIR/phenoscapR/issues

Additional_repositories https://cttir.r-universe.dev

Depends R (>= 4.1.0)

Imports data.table, dbscan, digest, ggplot2 (>= 3.0.0), methods, stats, utils, grDevices, grid, png

Suggests brand.yml, bsicons, bslib (>= 0.9.0), cellspecR (>= 1.1.0), chromote, deldir, ggtext, gridtext, isoband, jsonlite, knitr, mclust, Matrix, RANN, ragg, reactable, rmarkdown, Rtsne, S4Vectors, sass, Seurat, SeuratObject, sf, shiny (>= 1.6.0),

shinytest2, songR, SpatialExperiment, SummarizedExperiment, testthat (>= 3.0.0), thematic, uwot, withr

Config/Needs/check CTTIR/cellspecR, CTTIR/songR

Config/Needs/coverage covr, CTTIR/cellspecR, CTTIR/songR

Config/Needs/lint CTTIR/cellspecR, CTTIR/songR

Config/testthat/edition 3

Config/Needs/website CTTIR/themakR, pkgdown, CTTIR/cellspecR, CTTIR/songR

Encoding UTF-8

LazyData true

Language en-GB

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libpng-dev

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

Date/Publication 2026-11-03 15:14:48 UTC

RemoteUrl https://github.com/CTTIR/phenoscapR

RemoteRef HEAD

RemoteSha 72fb1bc9bccf210393071c7a1e2e4a4bb0f552a0

Contents

[,SpatialCellData-method . . . 6
[[,SpatialCellData-method . . . 7
$,SpatialCellData-method . . . 8
AdjustPatientFamilies . . . 8
AlterationMatrixData . . . 9
AnnotateClusters . . . 10
AnnotationTracks . . . 11
ApplyGroupedThresholds . . . 12
as_Seurat . . . 13
as_SpatialCellData . . . 14
as_SpatialExperiment . . . 14
AssemblePairedSummaries . . . 15
AssignFixedGrid . . . 17
BestSubWindow . . . 18
BootstrapMedianInterval . . . 19
BoxPlot . . . 20
BuildExactDelaunayEdges . . . 21
cell_density . . . 24
CellDensity . . . 25

CellMap . . . 25
CellMapData . . . 26
CellPositionsFor . . . 28
CellularNeighbourhoods . . . 29
ClusterLabelPositions . . . 30
CompareArrays . . . 30
CompareTables . . . 31
CompleteRegisteredValues . . . 32
ComposeFigure . . . 34
CompositeFields . . . 35
CompositionPlot . . . 36
ContrastGroupedOverlap . . . 37
ContrastGroupedRanks . . . 39
ContrastPatientEndpoints . . . 40
Coords . . . 42
CorrelateGroupedRanks . . . 42
CountDisjointEdges . . . 43
CountRadiusSets . . . 45
CreateSpatialObject . . . 48
CrossNNDistance . . . 49
CustomGradient . . . 49
DelaunayNetwork . . . 50
DensestGridWindow . . . 51
DensityLevelContour . . . 51
DensityPlot . . . 53
DifferentialAbundance . . . 54
dim,SpatialCellData-method . . . 55
DotPlot . . . 55
EmbeddingLayers . . . 56
EmbeddingLimits . . . 57
EmbeddingPlot . . . 58
Embeddings . . . 59
ExactPairedTests . . . 60
ExactPatientTests . . . 61
ExactPermutationBound . . . 63
ExpressionClusters . . . 63
FeaturePlot . . . 64
FieldCellsRanked . . . 65
FieldCentre . . . 66
FigureGrammars . . . 67
FigureGrob . . . 68
FigurePanel . . . 68
FigureText . . . 69
FindNeighbours . . . 70
FormatP . . . 71
FormatStatistic . . . 72
FormatTemplate . . . 73
FromCellspec . . . 74

GetData . . . 75
HistogramPlot . . . 75
Idents . . . 76
interaction_matrix . . . 77
InteractionMatrix . . . 78
InteractionPlot . . . 79
LeaveOnePatientOut . . . 79
LegendKeys . . . 81
LookupNestedRows . . . 82
ManuscriptPalettes . . . 83
ManuscriptStyle . . . 84
MarkerHeatmap . . . 85
Markers . . . 86
Meta . . . 87
MoransI . . . 87
NCells . . . 88
nearest_neighbours . . . 89
NearestLabelDistance . . . 90
NearestPartnerPairs . . . 91
NeighbourhoodEnrichment . . . 92
NeighbourTypeCounts . . . 93
NiceBreaks . . . 94
NMarkers . . . 95
normalise_markers . . . 96
NormaliseData . . . 97
PairCorrelation . . . 98
PairedDifferences . . . 99
PaletteContinuous . . . 99
PaletteDiscrete . . . 100
PermutationCorrelation . . . 101
phenoscapR_example . . . 102
phenotype_cells . . . 103
PhenotypeCells . . . 103
PhenotypeSummary . . . 104
PlanExactDelaunayWorkspace . . . 105
plot_cell_map . . . 106
plot_density . . . 107
plot_heatmap . . . 108
plot_interactions . . . 108
PlotAlterationMatrix . . . 109
PlotAnnotatedHeatmap . . . 111
PlotBandProfiles . . . 114
PlotBars . . . 116
PlotCellMap . . . 119
PlotDataBindRows . . . 121
PlotDataConstant . . . 122
PlotDataContrastSummary . . . 123
PlotDataCount . . . 124

PlotDataCrossShares . . . 125
PlotDataDirection . . . 126
PlotDataDominant . . . 127
PlotDataFromArrays . . . 128
PlotDataJoin . . . 129
PlotDataKeyColours . . . 130
PlotDataLabels . . . 131
PlotDataLonger . . . 132
PlotDataMembership . . . 133
PlotDataPairRatio . . . 134
PlotDataProduct . . . 135
PlotDataQuantiles . . . 136
PlotDataRankRows . . . 137
PlotDataRatio . . . 138
PlotDataRecode . . . 139
PlotDataRelative . . . 140
PlotDataSelect . . . 141
PlotDataShares . . . 142
PlotDataSummarise . . . 143
PlotDataTableCells . . . 144
PlotDataUnite . . . 146
PlotDensityOutlines . . . 146
PlotDotMatrix . . . 148
PlotEmbedding . . . 150
PlotFluorescenceField . . . 152
PlotGroupDifferences . . . 153
PlotGroupStrip . . . 156
PlotImage . . . 158
PlotIntervalForest . . . 160
PlotPairedPatients . . . 162
PlotQuantileCurves . . . 164
PlotScatterGroups . . . 166
PlotStackedBars . . . 170
PlotTerritories . . . 173
PlotTextBlock . . . 174
PlotVolcano . . . 176
ProjectSpatialTable . . . 178
PtLinewidth . . . 178
qc_filter . . . 179
QCFilter . . . 180
QCPlot . . . 181
QuadratAnalysis . . . 182
read_spatial . . . 182
ReadCosMx . . . 184
ReadFigureSpec . . . 185
ReadMatrixCoords . . . 186
ReadMERSCOPE . . . 187
ReadPlotTable . . . 188

ReadSpatial . . . 188
ReadXenium . . . 189
Reductions . . . 190
RenderFigure . . . 191
RidgePlot . . . 192
RipleysK . . . 193
run_app . . . 194
RunPCA . . . 195
RunSONG . . . 195
RunTSNE . . . 197
RunUMAP . . . 198
SaveFigure . . . 199
SchematicShapes . . . 200
ScreePlot . . . 201
SetPalette . . . 202
show,SpatialCellData-method . . . 203
spatial_clusters . . . 203
SpatialCellData-class . . . 204
SpatialClusters . . . 205
SpatialDomains . . . 206
SpatialNetworkPlot . . . 207
summarise_phenotypes . . . 208
SummarizeCellComposition . . . 208
SummarizeComparisons . . . 210
SummarizeFixedGrid . . . 211
SummarizeGroupedCountFractions . . . 212
SummarizeGroupedValues . . . 213
SummarizeNearestDistances . . . 214
TerritoryTiles . . . 216
ThemeManuscript . . . 217
TypesetLabel . . . 218
VarianceExplained . . . 219
ViolinPlot . . . 219
WriteFigure . . . 220

Index **222**

`[,SpatialCellData-method`

Subset a SpatialCellData Object

Description

Subset a SpatialCellData Object

Usage

```
## S4 method for signature 'SpatialCellData'
x[i, j, drop = FALSE]
```

Arguments

x	A SpatialCellData object. Supplying cell indices clears cached spatial results and the derived nn_distance and density metadata columns. Recompute these after filtering, reordering or duplicating cells. Marker-only subsetting preserves spatial results; dimensional reductions are subset by cell. Phenotype, cluster, neighbourhood and domain labels remain frozen annotations from the original fit for plotting and comparison; they do not describe a refit on the subset. Neither labels nor retained reductions are refitted. The metadata attribute frozen_spatial_labels records each retained spatial label's original fit row count; further subsetting preserves it. Recomputing an assignment clears its frozen-label record. Recompute spatial assignments to describe the new neighbourhoods. Marker-only subsetting does not revalidate expression-dependent cached analyses.
i	Cell indices (integer or logical).
j	Marker indices (integer, logical, or character).
drop	Ignored.

Value

A subsetted SpatialCellData object.

[[,SpatialCellData-method

Access Metadata Columns with [[

Description

Access Metadata Columns with [[

Usage

```
## S4 method for signature 'SpatialCellData'
x[[i]]
```

Arguments

x	A SpatialCellData object.
i	Column name (character).

Value

The requested metadata column.

$,SpatialCellData-method

Access Metadata Columns with $

Description

Access Metadata Columns with $

Usage

```
## S4 method for signature 'SpatialCellData'
x$name
```

Arguments

x	A SpatialCellData object.
name	Column name.

Value

The requested metadata column.

AdjustPatientFamilies *Adjust Explicitly Declared Patient-Test Families*

Description

Computes finite-only BH for historical comparison and declared-family BH/BY using every registered hypothesis as the adjustment count. Unavailable p values remain NA. Family IDs define the entire correction scope: include setting and contrast in the ID only when separate families are intended. A retrospective family registry is not prospective registration, and neither adjustment establishes biological validity or study-wide error control.

Usage

```
AdjustPatientFamilies(tests, policy, method = c("BH", "BY"))
```

Arguments

tests	Data frame containing character hypothesis_id, family_id, and numeric p_value. All declared hypotheses must be present exactly once.
policy	Explicit adjustment policy, "declared" or "finite_only".
method	Selected correction, "BH" or "BY".

Value

A list with `tests` (input columns plus all adjustment variants, selected `p_adjusted`, policy and method) and `families` (declared, testable and unavailable counts). Every NA is retained in the original row order.

Examples

```
x <- data.frame(hypothesis_id = c("h1", "h2"), family_id = "f",
                p_value = c(0.03, NA))
AdjustPatientFamilies(x, policy = "declared")
```

AlterationMatrixData *Alteration Classes for an Alteration Matrix*

Description

Turns a gene-by-sample table of alteration types (several types joined by a separator) into one class per gene and item: types are mapped to classes (for example nonsense, frameshift and splice variants to "Truncating"), and a gene with more than one class in a sample is "Multiple". Genes are selected and ordered as `first`, then genes altered in at least `min_count` samples by decreasing count (ties in table order), then `last`.

Usage

```
AlterationMatrixData(
  matrix,
  gene = NULL,
  ids,
  id_map = NULL,
  map_from = NULL,
  map_to = NULL,
  first = NULL,
  last = NULL,
  min_count = 1,
  classes = list(Truncating = c("Nonsense", "Frameshift", "Splice")),
  multiple = "Multiple",
  separator = "|"
)
```

Arguments

`matrix`	Data frame: gene column and one column per sample; empty or missing entries mean no alteration.
`gene`	Gene column (default the first column).
`ids`	Items (columns of the plot) in order: vector or data frame ordered by `rank`.
`id_map`	Optional data frame mapping items to sample columns.

map_from, map_to	Columns of id_map.
first, last	Genes always shown first and last.
min_count	Smallest number of altered samples for other genes.
classes	Named list: class to the types it collects; other types keep their name.
multiple	Class of genes with several classes in one sample.
separator	Separator of several types in one entry.

Value

A long data frame with gene, id, class (missing if none), count (altered samples of the gene in the table), percent (count relative to the number of items) and gene_rank.

Examples

```
m <- data.frame(gene = c("G1", "G2"), S1 = c("Missense", ""),
                S2 = c("Nonsense|Missense", "Deletion"))
AlterationMatrixData(m, ids = c("S1", "S2"), min_count = 1)
```

AnnotateClusters *Annotate Clusters by Their Top Markers*

Description

Labels each cluster (or any categorical grouping) by the markers most enriched in it relative to the global mean, producing human-readable names such as "CD3+CD8".

Usage

```
AnnotateClusters(
  object,
  group = "expr_cluster",
  slot = "data",
  n_markers = 2L,
  add_column = TRUE
)
```

Arguments

object	A SpatialCellData-class object.
group	Character. Metadata column holding cluster labels. Default "expr_cluster".
slot	Character. "data" (default) or "counts".
n_markers	Integer. Number of top markers per cluster name. Default 2.
add_column	Logical. If TRUE (default) also store the names in a new <group>_label metadata column.

Value

A named character vector mapping each group level to its marker label. If `add_column` the object is returned instead, with the label column added (access the mapping via `attr(., "labels")`).

Examples

```
data(phenoscapR_example)
obj <- NormaliseData(phenoscapR_example, "zscore")
obj <- ExpressionClusters(obj, k = 6)
AnnotateClusters(obj, add_column = FALSE)
```

`AnnotationTracks` *Coloured Annotation Tracks*

Description

Builds categorical annotation tracks (for example clinical group, co-mutation, architecture) for a set of items: each track takes a value column, maps values to colours by exact values or regular expressions and uses a default colour otherwise.

Usage

```
AnnotationTracks(data, id, tracks, style = ManuscriptStyle())
```

Arguments

`data`	Data frame with one row per item.
`id`	Identifier column.
`tracks`	List of tracks: `name`, `column`, `colours` (named mapping value to colour), optional `patterns` (named mapping regular expression to colour) and `default`. Colours may be hex values or palette entries such as `"groups/group1"`.
`style`	A `ManuscriptStyle()` object resolving palette entries.

Value

A long data frame with `id`, `track`, `value`, `colour` and `track_rank`.

Examples

```
d <- data.frame(p = c("a", "b"), grp = c("x", "y"), mut = c("yes", "no"))
AnnotationTracks(d, "p", list(
 list(name = "Group", column = "grp", colours = c(x = "groups/group1", y = "groups/group2")),
 list(name = "Mutation", column = "mut", colours = c(yes = "#222222"), default = "#E0E0E0")))
```

`ApplyGroupedThresholds` *Apply Supplied Group Thresholds to Observations*

Description

Looks up one supplied threshold per composite group and tests strictly greater than. Thresholds are not estimated; equality is FALSE. Eligibility and numeric availability remain explicit. No labels, units or observation universe are inferred. Unknown groups fail, including groups of ineligible observations.

Usage

```
ApplyGroupedThresholds(
  data,
  thresholds,
  groups,
  id,
  value,
  threshold,
  eligible
)
```

Arguments

`data`	Data frame of observations. Original columns, metadata and row order are preserved; inputs are not mutated.
`thresholds`	Data frame with one threshold per unique group. Extra groups are permitted. Supply an explicit NA threshold for an unavailable group.
`groups`	Nonempty plain character vector of distinct group column names.
`id`	Plain character scalar naming the observation ID column. Identity is unique within groups; IDs may recur across different groups.
`value`	Plain character scalar naming the observation value column.
`threshold`	Plain character scalar naming the supplied threshold column. Selected numeric columns must be plain integer/double, finite or NA; classed, dimensional, complex, NaN and infinite values fail even when ineligible.
`eligible`	Plain character scalar naming a plain logical eligibility column. TRUE, FALSE and NA are distinct. Data roles must not overlap; threshold must not overlap groups. Keys are plain nonblank character values without NA or ASCII 28. Selectors are dimensionless and nonblank.

Value

A data frame with all original observation columns unchanged, followed by threshold_value (double), threshold_available and value_available (logical), above_threshold (logical or NA), and threshold_status (character). Original columns with these reserved names fail. Status precedence is INELIGIBLE, UNKNOWN_ELIGIBILITY, MISSING_VALUE_AND_THRESHOLD, MISSING_THRESHOLD, MISSING_VALUE, COMPLETE. Availability flags retain numeric missingness independently of eligibility. Only eligible observations with both numbers known receive a comparison. Empty inputs retain typed outputs; nonempty observations require complete group-reference coverage. Metadata classes and the original value storage type are preserved. Units must agree upstream; no conversion is performed.

Examples

```
d <- data.frame(group = "a", id = c("x", "y"), value = c(1, 2), eligible = TRUE)
r <- data.frame(group = "a", cutoff = 1)
ApplyGroupedThresholds(d, r, "group", "id", "value", "cutoff", "eligible")
```

---

as_Seurat *Convert a SpatialCellData to a Seurat Object*

Description

Builds a **Seurat** object with raw counts, normalised data, cell metadata, and the spatial coordinates stored both in metadata (`x`, `y`) and as a `"spatial"` dimensional reduction. Cell IDs repeated across samples are made unique with `make.unique()` for the Seurat cell names; the metadata keep the original `cell_id`. Export refuses marker identifiers that Seurat would rewrite (such as underscores); use `as_SpatialExperiment()` for unchanged identifiers, or rename markers explicitly before export.

Usage

```
as_Seurat(object)
```

Arguments

`object`	A `SpatialCellData-class` object.

Value

A `Seurat` object.

Examples

```
if (requireNamespace("Seurat", quietly = TRUE)) {
  data(phenoscapR_example)
  se <- as_Seurat(phenoscapR_example)
}
```

as_SpatialCellData *Convert a Seurat or SpatialExperiment Object to SpatialCellData*

Description

Imports an object from the wider ecosystem. Coordinates are taken from `spatialCoords` (SpatialExperiment) or from `x`/`y` metadata columns or a `"spatial"` reduction (Seurat). The normalized `logcounts` assay (SpatialExperiment) or `data` layer (Seurat) is retained when present; raw counts are used as data only when it is absent. Normalized assays must contain the same cells and features as counts; named axes are aligned, and incompatible partial assays are refused. All-missing features are retained. Existing cell identifiers are kept; absent `cell_id` metadata use the source cell names when available. No normalization method is inferred or applied.

Usage

```
as_SpatialCellData(x, ...)
```

Arguments

`x`	A `Seurat` or `SpatialExperiment` object.
`...`	Unused.

Value

A `SpatialCellData-class` object.

Examples

```
if (requireNamespace("Seurat", quietly = TRUE)) {
  counts <- matrix(c(5, 0, 3, 1, 2, 7, 0, 4, 6, 1, 2, 3), nrow = 4,
                   dimnames = list(NULL, c("CD3", "CD8", "CD20")))
  obj <- CreateSpatialObject(counts, data.frame(x = c(1, 5, 9, 4), y = c(2, 8, 3, 6)))
  seurat_object <- as_Seurat(obj)
  as_SpatialCellData(seurat_object)
}
```

as_SpatialExperiment *Convert a SpatialCellData to a SpatialExperiment Object*

Description

Builds a **SpatialExperiment** with `counts` and `logcounts` assays (markers x cells), the metadata as `colData`, and the coordinates as `spatialCoords`.

Usage

```
as_SpatialExperiment(object)
```

Arguments

object	A SpatialCellData-class object.

Value

A SpatialExperiment object.

Examples

```
if (requireNamespace("SpatialExperiment", quietly = TRUE)) {
  counts <- matrix(c(5, 0, 3, 1, 2, 7, 0, 4, 6, 1, 2, 3), nrow = 4,
                   dimnames = list(NULL, c("CD3", "CD8", "CD20")))
  obj <- CreateSpatialObject(counts, data.frame(x = c(1, 5, 9, 4), y = c(2, 8, 3, 6)))
  spe <- as_SpatialExperiment(obj)
}
```

AssemblePairedSummaries

Assemble Registered Paired Summaries

Description

Aligns already computed scalar summaries to an explicit expected group registry. Computes the first side or first minus second side without pooling observations or changing the supplied estimator. Choose the value and count columns appropriate to the estimator, for example weighted_mean and n_weighted_available from SummarizeGroupedValues(). This is descriptive assembly, not an independence assessment or inferential test.

Usage

```
AssemblePairedSummaries(
  data,
  registry,
  groups,
  side,
  value,
  count,
  weight_sum = NULL,
  sides = c("A", "B"),
  contrast = c("difference", "first"),
  min_count = 1L
)
```

Arguments

data	Data frame of precomputed summaries, at most one row per group and side. Extra columns are ignored.
registry	Data frame with distinct expected group keys.
groups	Nonempty character vector of distinct group columns. Keys must be nonblank character vectors without NA or the reserved ASCII 28 separator.
side	Name of the character side column.
value	Name of the finite plain integer/double summary column, allowing NA. Classed vectors (including factors, dates and integer64) are rejected for values, counts, weight totals and min_count; names are permitted.
count	Name of the nonnegative whole-number count column, allowing NA.
weight_sum	Optional name of a nonnegative finite numeric denominator column, allowing NA. Missing-side totals remain NA, not an observed zero.
sides	Two distinct nonblank side labels, in subtraction order.
contrast	Either "difference" (first minus second) or "first".
min_count	Positive whole-number minimum for each required side.

Details

Absent sides retain NA values, zero counts and a false presence flag. Observed NA counts remain unknown. Optional weight totals are provenance only and never used to reconstruct counts or recompute supplied values. The registry defines output order; extra registry columns are ignored.

Value

Data frame in registry order with keys, first/second presence, values, counts and weight totals, value, status, contrast, side labels and minimum. Status precedence is MISSING_SIDE, UNKNOWN_COUNT, INSUFFICIENT_COUNT, UNAVAILABLE_VALUE, COMPLETE, considering required sides only. An empty registry with empty data returns a typed empty table. Unknown group/side keys, duplicates, malformed numbers and arithmetic overflow fail.

Examples

```
d <- data.frame(sample = c("one", "one"), arm = c("A", "B"),
                average = c(4, 1), n = c(3, 2))
AssemblePairedSummaries(d, data.frame(sample = c("one", "two")),
                        "sample", "arm", "average", "n")
```

`AssignFixedGrid` *Assign Physical Cell Coordinates to Fixed Square Grids*

Description

Assigns every supplied cell independently in each setting. Bins are half-open intervals: an exact boundary belongs to the bin on its positive side. Negative coordinates use floor division. No rounding or tolerance is applied. Samples remain separate even when coordinates overlap. Select eligible cells explicitly before calling; this function does not filter, classify, or infer review status.

Usage

```
AssignFixedGrid(cells, settings, frames)
```

Arguments

`cells`	Data frame with character `cell_id`, `sample_id`, `frame_id` and finite numeric `x`, `y` coordinates in micrometres. Keys are unique within each sample. An empty, correctly typed table is supported.
`settings`	Nonempty data frame with unique character `setting_id`, positive finite numeric `size_um`, and finite numeric `offset_x_um`, `offset_y_um`. Each setting applies separately to every sample.
`frames`	Data frame with unique character `sample_id`, character `frame_id` and character `unit`, which must be `"um"`. Each cell must match its sample's frame exactly. Extra frame rows are permitted.

Details

The result describes detection locations, not tissue regions. Square bin area is not observed tissue area and must not be used as an inferred tissue-density denominator. Frame identifiers bind already calibrated coordinates; no unit, origin, axis, crop or registration conversion is performed.

Value

Data frame ordered by setting, sample and cell ID (radix order), with those keys, frame ID, integer grid indices, physical bounds and grid size. Bins are keyed by `(setting_id, sample_id, grid_x, grid_y)`, never coordinates alone. Unrepresentable bounds and indices outside the R integer range fail. When the rounded division puts a coordinate outside its bin, the index is corrected against the computed bounds; bounds that still exclude an assigned cell fail. No tolerance or coordinate correction is introduced. Extra input columns are ignored; inputs are not modified.

Examples

```
cells <- data.frame(cell_id = c("a", "b"), sample_id = "s",
  frame_id = "physical", x = c(-1, 10), y = c(0, 0))
settings <- data.frame(setting_id = "g", size_um = 10,
  offset_x_um = 0, offset_y_um = 0)
```

```
frames <- data.frame(sample_id = "s", frame_id = "physical", unit = "um")
AssignFixedGrid(cells, settings, frames)
```

`BestSubWindow` *Sub-Window with the Most Cells of a Kind*

Description

Scans square sub-windows of width `sub` on a grid of spacing `step` inside a field and returns the one holding the most cells that match `filter` (first maximum in row-major grid order, x varying fastest).

Usage

```
BestSubWindow(
  cells,
  centre,
  size,
  sub,
  filter = list(),
  ranges = list(),
  step = 10,
  x = "x",
  y = "y"
)
```

Arguments

`cells`	Data frame of cells.
`centre`	Field centre, `c(x, y)` or a data frame with `x`, `y`.
`size`	Field width.
`sub`	Sub-window width.
`filter`	Named list: column to values.
`ranges`	Named list: column to `c(min, max)` (inclusive) for numeric conditions, combined with `filter`.
`step`	Grid spacing.
`x`, `y`	Column names.

Value

One-row data frame with `x`, `y`, `n`, `dx`, `dy` (offset from the field centre).

Examples

```
set.seed(1)
d <- data.frame(x = c(runif(40, 0, 100), runif(20, 60, 80)),
                y = c(runif(40, 0, 100), runif(20, 20, 40)),
                label = rep(c("a", "b"), c(40, 20)))
BestSubWindow(d, centre = c(50, 50), size = 100, sub = 20, filter = list(label = "b"))
```

`BootstrapMedianInterval` *Bootstrap Percentile Interval of a Median*

Description

Resamples the finite values of each group with replacement and reports the percentile interval of the resampled medians (quantile type 7, which is NumPy's default linear percentile). Draws come from R's Mersenne-Twister generator with a declared seed, or from supplied resampling indices (for example indices archived from another implementation), so that the interval can be reproduced exactly.

Usage

```
BootstrapMedianInterval(
  data,
  value,
  groups = NULL,
  n_boot = 10000L,
  conf_level = 0.95,
  seed = NULL,
  reseed = TRUE,
  indices = NULL,
  index_base = 1L
)
```

Arguments

`data`	Data frame in long form.
`value`	Numeric value column.
`groups`	Optional group columns; one interval per group.
`n_boot`	Number of resamples.
`conf_level`	Interval coverage.
`seed`	Optional whole-number seed. With `reseed = TRUE` every group starts from this seed (as a per-call generator would); otherwise groups share one stream in output order. The caller's random stream is restored.
`reseed`	See `seed`.

indices	Optional supplied draws: an integer matrix with n_boot rows and one column per finite value (a single group), or a named list of such matrices keyed by hypothesis_id-style group labels (group values joined by " :: "). Each row is one resample.
index_base	1 (R) or 0 (zero-based archived indices).

Value

A data frame with the group columns, n, median, lower, upper, conf_level, n_boot, seed and draws ("R_MERSENNE_TWISTER" or "SUPPLIED").

Examples

```
d <- data.frame(g = rep(c("a", "b"), each = 8), v = c(1:8, (1:8)^2))
BootstrapMedianInterval(d, "v", "g", n_boot = 2000, seed = 42)
```

BoxPlot *Box Plot of Marker Expression*

Description

Box plot showing the distribution of marker expression across phenotypes or other grouping variables.

Usage

```
BoxPlot(
  object,
  features,
  group_by = "phenotype",
  slot = "data",
  colours = NULL,
  ncol = NULL
)
```

Arguments

object	An SpatialCellData object.
features	Character vector. Marker names to plot.
group_by	Character. Metadata column to group by. Default "phenotype".
slot	Character. "data" or "counts".
colours	Named character vector or NULL.
ncol	Integer or NULL.

Value

A ggplot object.

Examples

```
counts <- matrix(c(rnorm(30, 5), rnorm(30, 1)), ncol = 2,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(30), y = runif(30))
meta <- data.frame(cell_id = 1:30, sample_id = "s1",
                   phenotype = rep(c("T cell", "Other"), each = 15))
obj <- CreateSpatialObject(counts, coords, meta)
BoxPlot(obj, features = c("CD3", "CD8"))
```

BuildExactDelaunayEdges

Build Explicitly Ordered Delaunay Graphs

Description

Constructs each complete declared geometry before selecting edges whose two endpoints are eligible. Uses `deldir::deldir()` with no fallback, coordinate rounding, jitter or population inference.

Usage

```
BuildExactDelaunayEdges(
  nodes,
  frames,
  groups,
  frame,
  id,
  x,
  y,
  point_order,
  orientation_rank,
  geometry,
  eligible,
  unit_column,
  coordinate_unit,
  edge_order,
  window_policy,
  cap_policy,
  backend_contract = NULL,
  eps,
  mode,
  caps = numeric(),
  initial_madj = 64L,
  max_retries = 32L
)
```

Arguments

`nodes`	Data frame of keys, coordinates, orders and memberships.
`frames`	Data frame of unique complete group/frame keys and units.
`groups`	Nonempty character vector of distinct grouping column names.
`frame`	Character name of the coordinate-frame key in both tables.
`id`	Character name of the node ID column. IDs are unique within frames.
`x`, `y`	Character names of finite coordinate columns in `nodes`.
`point_order`	Character name of unique positive whole within-frame ranks defining geometry input order, independently of table row order.
`orientation_rank`	Character name of unique positive whole within-frame ranks defining undirected endpoint orientation, independently of input order.
`geometry`, `eligible`	Character names of nonmissing logical memberships. Eligibility implies geometry membership. Nodes outside geometry remain in the node map; isolated eligible nodes are retained.
`unit_column`	Character name of the units column in `frames`.
`coordinate_unit`	One of `"nm"`, `"um"`, `"mm"`, `"cm"`, `"m"`. All frames must declare this unit; no conversion occurs.
`edge_order`	Required string `"orientation_rank"`. Output edges sort by UTF-8 group/frame keys, then numeric endpoint orientation ranks.
`window_policy`	Required string `"span_padding_v1"`. Bounds extend the coordinate ranges by `max(1e-6, max(x_span, y_span) * 1e-10)` on each side.
`cap_policy`	Required string `"distance_leq"`. Named caps use inclusive comparisons with `sqrt(dx^2 + dy^2)`, without a tolerance.
`backend_contract`	Deprecated and ignored. Construction uses the public `deldir::deldir()` interface, so there are no private functions or native routines to pin. Kept so that existing calls keep working.
`eps`	Explicit positive finite scalar tolerance passed to the engine.
`mode`	Explicit `"stock"`, `"bounded"` or `"auto"` workspace mode, validated with the workspace planner and recorded in the provenance. Every mode runs `deldir::deldir()`, which sizes its own workspace and enlarges it when needed; frames whose stock workspace would exceed the integer limit (`bounded_required`) are refused.
`caps`	Named positive finite numeric thresholds in `coordinate_unit`. An empty numeric vector requests no cap columns.
`initial_madj`, `max_retries`	Validated as in `PlanExactDelaunayWorkspace()`; they no longer change the construction.

Details

Supported columns are dimensionless, unclassed base character, integer/double or logical vectors as appropriate. Names are allowed. Factors, `integer64` and arbitrary classed columns are rejected without coercion. Data frames and data.tables are supported containers. Keys must be nonmissing and nonblank. Selected roles are distinct within each table. Node roles cannot be named `source_row` or `geometry_index`; group/frame keys cannot collide with output edge or frame-summary fields. Duplicate IDs across different frames are valid.

Every frame is preflighted before construction begins. A declared frame needs at least three distinct noncollinear geometry points; a declared empty frame fails. A completely empty registry and empty node table return typed empty outputs without loading or executing the backend. Integer workspace and conservative combined edge/cap bounds are checked before allocation; they do not certify available memory. A finite direct-determinant guard is deliberately conservative for extreme or nearly collinear coordinates. Unsupported geometry fails rather than being perturbed. Distances must be finite and positive.

Complete edges are checked for whole in-range nonself indices, exact endpoint coordinate reconciliation, duplicate unordered pairs, connectedness and planar cardinality bounds before eligibility induction. These necessary invariants alone are not an independent proof of Delaunay geometry. Co-circular choices depend on the `deldir` version and explicit point order. Numeric orientation ranks and UTF-8 output ordering do not infer a historical locale or ordering policy.

`deldir` is optional: a missing installation fails clearly for nonempty construction. Each frame calls `deldir::deldir(x, y, rw, eps, sort = TRUE, round = FALSE)` on the geometry points in `point_order`. Warnings and unexpected messages fail closed; messages announcing that `deldir` enlarged its workspace are counted in the frame provenance (`workspace_growth`). No incomplete tessellation is returned. The frame provenance records the `deldir` version.

Value

A list with `schema_version = "0.1.0"`, `nodes` (including input `source_row` and ordered `geometry_index`), `full_edges`, `analysis_edge_rows`, `cap_flags`, `caps`, `frames` and `provenance`. `full_edges` contains keys, endpoint IDs/geometry indices/orientation ranks, coordinates, distance, `analysis_edge` and `edge_row`. Frame counts distinguish all nodes, geometry, eligible nodes, full edges and induced edges. `frame_construction` contains keyed per-frame engine/workspace receipts in input registry order; output tables sort independently. No input is modified.

Examples

```
PlanExactDelaunayWorkspace(100, mode = "stock")$native_buffer_bytes
if (requireNamespace("deldir", quietly = TRUE)) {
  nodes <- data.frame(
    sample = "one", frame = "image", id = c("a", "b", "c", "d"),
    x = c(0, 4, 0, 3), y = c(0, 0, 3, 4), input = 1:4, orientation = 1:4,
    geometry = TRUE, eligible = c(TRUE, TRUE, TRUE, FALSE)
  )
  frames <- data.frame(sample = "one", frame = "image", unit = "um")
  graph <- BuildExactDelaunayEdges(nodes, frames, "sample", "frame", "id",
    "x", "y", "input", "orientation", "geometry", "eligible", "unit", "um",
    "orientation_rank", "span_padding_v1", "distance_leq",
    eps = 1e-9, mode = "stock", caps = c(short = 3, long = 5)
```

```
  )
  graph$full_edges
}
```

`cell_density` *Compute Cell Density*

Description

Estimates the local cell density around each cell using a circular neighbourhood of a given radius.

Usage

```
cell_density(dt, radius, target_phenotype = NULL)
```

Arguments

`dt`	A `data.table` with columns `x` and `y`.
`radius`	Numeric. Radius of the neighbourhood (in coordinate units).
`target_phenotype`	Character string or `NULL`. If provided, only cells of this phenotype are counted in the neighbourhood.

Value

The input `data.table` with an added `density` column representing the number of neighbours within the specified radius.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:50,
  x = runif(50, 0, 100), y = runif(50, 0, 100)
)
result <- cell_density(dt, radius = 20)
head(result[, .(cell_id, density)])
```

CellDensity *Cell Density (SpatialCellData)*

Description

Estimates local cell density by counting neighbours within a radius.

Usage

```
CellDensity(object, radius, target = NULL)
```

Arguments

object	An SpatialCellData-class object.
radius	Numeric. Radius of the neighbourhood.
target	Character or NULL. Restrict to a specific phenotype.

Value

An SpatialCellData-class with density added to meta_data.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 100), y = runif(50, 0, 100))
obj <- CreateSpatialObject(counts, coords)
obj <- CellDensity(obj, radius = 20)
```

CellMap *Plot Cell Map*

Description

Scatter plot of cell positions coloured by phenotype, cluster, or any metadata column. Analogous to Seurat's DimPlot but in tissue space.

Usage

```
CellMap(
  object,
  colour_by = "phenotype",
  colours = NULL,
  pt_size = 0.5,
  title = NULL,
  dark_theme = FALSE
)
```

Arguments

object	An SpatialCellData object.
colour_by	Character. Column name in meta_data to colour by. Default "phenotype". Numeric columns get a continuous colour scale.
colours	Named character vector or NULL for automatic colours via the global palette.
pt_size	Numeric. Point size. Default 0.5.
title	Character or NULL. Plot title.
dark_theme	Logical. Use a dark background (useful for tissue images). Default FALSE.

Value

A ggplot object.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 500), y = runif(50, 0, 500))
obj <- CreateSpatialObject(counts, coords)
obj <- PhenotypeCells(obj, thresholds = list(CD3 = 0, CD8 = 0))
CellMap(obj)
```

CellMapData *Cells of a Map Field or Whole Section*

Description

Prepares the input of PlotCellMap(): the cells inside a square field (or all cells for a whole-section map), the field geometry and, when requested, the interface zone, i.e. the points of a regular grid that lie within radius of a reference cell type (for example the tumour-stroma interface within 20 micrometres of a tumour cell).

Usage

```
CellMapData(
  cells,
  centre = NULL,
  size = NULL,
  centre_cell = NULL,
  id = "row",
  grid = NULL,
  offset = NULL,
  interface = NULL,
  classes = NULL,
  margin = 0,
```

```
  x = "x",
  y = "y",
  label = "label"
)
```

Arguments

cells	Data frame of cells (positions, label and further columns).
centre	Optional c(x, y) or a data frame with columns x, y.
size	Optional field width; NULL for a whole-section map.
centre_cell	Optional cell identifier giving the centre.
id	Column holding cell identifiers for centre_cell.
grid	Optional list with step and offset: the centre is moved to the nearest min(coordinate) + offset + k * step.
offset	Optional shift c(dx, dy) added to the centre.
interface	Optional list with labels (reference cell labels), radius and step (grid spacing).
classes	Optional list with column, map (named list of value to class) and to (new column name; default "class"); unmapped values become default (default NA).
margin	Extra width kept around the field (cells drawn partly inside).
x, y, label	Column names.

Details

The field centre is given directly, taken from a cell (centre_cell, the value of column id), taken from the first row of a data frame with columns x and y, snapped to a regular grid anchored at the minimum cell coordinates (grid), and finally shifted by offset.

Value

A list of class phenoscapR_cellmap with cells, interface (data frame of grid points in the zone, or NULL), centre, size, step and extent (c(xmin, xmax, ymin, ymax)).

Examples

```
set.seed(1)
d <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100),
                label = sample(c("t", "a"), 200, TRUE))
m <- CellMapData(d, centre = c(50, 50), size = 40,
                 interface = list(labels = "t", radius = 5, step = 2))
nrow(m$cells)
```

`CellPositionsFor` *Attach Cell Positions to a Table of Cell References*

Description

Looks up cells referenced by image and row (as in review ledgers and example lists) in per-image cell tables and adds their positions and further columns.

Usage

```
CellPositionsFor(
  table,
  cells,
  labels = NULL,
  source = "sample_id",
  row = "row",
  columns = c("x", "y", "label")
)
```

Arguments

`table`	Data frame with an image key column and a row column.
`cells`	List of cell tables (with column `row`), one per image.
`labels`	Image keys of `cells`, in list order (default the list names).
`source`, `row`	Columns of `table` holding the image key and the cell row.
`columns`	Columns copied from the cell tables.

Value

`table` with the added columns.

Examples

```
cells <- list(s1 = data.frame(row = 0:2, x = c(1, 5, 9), y = c(2, 4, 6), label = "a"),
              s2 = data.frame(row = 0:1, x = c(3, 7), y = c(1, 1), label = "b"))
refs <- data.frame(sample_id = c("s2", "s1"), row = c(1, 2))
CellPositionsFor(refs, cells)
```

CellularNeighbourhoods *Cellular Neighbourhoods (Niches)*

Description

Assigns each cell to a cellular neighbourhood ("niche") by clustering cells on the phenotype composition of their local spatial neighbourhood. This is the windowed-neighbourhood approach popularised for multiplexed imaging: for every cell the phenotype frequencies among its `k` nearest neighbours form a composition vector, and those vectors are clustered.

Usage

```
CellularNeighbourhoods(
  object,
  n_neighbourhoods = 8L,
  k = 20L,
  phenotype_col = "phenotype",
  seed = NULL
)
```

Arguments

`object`	A `SpatialCellData-class` object with a phenotype column.
`n_neighbourhoods`	Integer. Number of neighbourhoods to find. Default `8`.
`k`	Integer. Neighbourhood size (nearest neighbours per cell). Default `20`.
`phenotype_col`	Character. Metadata column of phenotype labels. Default `"phenotype"`.
`seed`	Integer or `NULL`. Random seed for k-means.

Value

The object with a `neighbourhood` column in `meta_data`; the neighbourhood-by-phenotype composition matrix is stored in `object@spatial$neighbourhood_composition`.

Examples

```
data(phenoscapR_example)
obj <- phenoscapR_example
obj@meta_data$phenotype <- obj@meta_data$phenotype_true
obj <- CellularNeighbourhoods(obj, n_neighbourhoods = 5, k = 15, seed = 1)
table(obj$neighbourhood)
```

ClusterLabelPositions *Label Positions of Clusters on an Embedding*

Description

The median position of the points of each cluster with at least `min_points` points, for numbering clusters on the embedding.

Usage

```
ClusterLabelPositions(
  data,
  x = "x",
  y = "y",
  cluster = "cluster",
  filter = NULL,
  min_points = 150
)
```

Arguments

`data`	Data frame with coordinates and a cluster column.
`x`, `y`, `cluster`	Column names.
`filter`	Optional named list of columns and values selecting the points used.
`min_points`	Smallest cluster that gets a label.

Value

A data frame with `label`, `x`, `y` and `n`, ordered by cluster.

Examples

```
d <- data.frame(x = c(1, 2, 3, 10), y = c(1, 1, 2, 5), cl = c(1, 1, 1, 2))
ClusterLabelPositions(d, cluster = "cl", min_points = 2)
```

CompareArrays *Compare Table Columns or Arrays with Arrays of a NumPy Archive*

Description

Compares observed vectors or matrices (taken from columns of a table or elements of a list) with reference arrays, read from an `.npz` archive with `cellspecR::cs_read_npz()` or given as a list. Shapes must agree; values are compared exactly or within an absolute tolerance.

Usage

```
CompareArrays(observed, reference, arrays, tolerance = 0, check = "arrays")
```

Arguments

observed	A data frame or named list.
reference	Path to an .npz archive, or a named list of arrays.
arrays	Named list: reference array name to the observed column(s) or element. Several columns are bound into a matrix in the given order.
tolerance	Absolute tolerance (numbers).
check	Label stored in column check.

Value

A data frame with one row per array: check, row_key (array name), array, observed_dim, reference_dim, reference_type, n_values, n_different, max_difference, tolerance, matches and status (MATCH, MISMATCH, SHAPE_MISMATCH).

Examples

```
obs <- data.frame(a = 1:3, b = c(0, 1, 0), z = c(0.5, 1.5, 2.5))
ref <- list(counts = cbind(1:3, c(0, 1, 0)), z = c(0.5, 1.5, 2.5))
CompareArrays(obs, ref, list(counts = c("a", "b"), z = "z"))
```

CompareTables *Compare a Recomputed Table with a Reference Table by Key*

Description

Joins two tables on declared key columns (which must be unique in both) and compares declared value columns: numbers within an absolute tolerance, everything else (text, logical) exactly. Missing values match only missing values. Keys present in one table only are reported.

Usage

```
CompareTables(
  observed,
  reference,
  keys,
  columns,
  tolerance = 0,
  observed_element = NULL,
  reference_element = NULL,
  observed_filter = NULL,
  reference_filter = NULL,
  reference_fill = NULL,
  check = "comparison"
)
```

Arguments

observed, reference
: Data frames (or lists holding one, see observed_element, reference_element).

keys
: Key columns: a character vector of names shared by both tables, or a named vector/list observed name = reference name.

columns
: Value columns, given like keys.

tolerance
: Absolute tolerance for numeric columns: one number, or a named vector by observed column (unnamed columns use 0).

observed_element, reference_element
: Optional element names when an input is a list of tables (for example "tests" of AdjustPatientFamilies()).

observed_filter, reference_filter
: Optional named lists of column values selecting rows before the join.

reference_fill
: Optional value used for every reference column when a key is absent from the reference (for registries that omit zero rows); such rows are flagged in reference_filled.

check
: Label stored in column check.

Value

A data frame with one row per key and column: check, the key columns (observed names), row_key, column, reference_column, observed_value, reference_value (text, 17 significant digits for numbers), difference (absolute, numeric pairs), tolerance, reference_filled, matches and status (MATCH, MISMATCH, MISSING_OBSERVED, MISSING_REFERENCE).

See Also

SummarizeComparisons()

Examples

```
a <- data.frame(id = c("x", "y"), p = c(0.1, 0.2), s = c("ok", "ok"))
b <- data.frame(key = c("y", "x"), pv = c(0.2 + 1e-12, 0.1), s = c("ok", "no"))
CompareTables(a, b, keys = c(id = "key"), columns = c(p = "pv", s = "s"),
              tolerance = 1e-10)
```

CompleteRegisteredValues

Complete Numeric Observations Against an Explicit Registry

Description

Matches one numeric observation to each caller-supplied composite registry key. The registry defines the entire row universe and its order. No Cartesian product, observed group, eligibility rule or biological absence is inferred. The required absent_value applies only to absent rows, never to observed NA.

Usage

```
CompleteRegisteredValues(data, registry, keys, value, absent_value)
```

Arguments

data	Data frame containing the selected keys and numeric value column.
registry	Data frame with every expected composite key, in output order. Additional metadata columns retain their values and classes. It must not contain the selected value column or observed, status, absent_value columns.
keys	Nonempty plain character vector of distinct key column names. Key columns must be plain dimensionless nonblank character vectors without NA or the reserved ASCII 28 separator.
value	Plain character scalar naming the value column, distinct from keys and reserved output names. Values must be plain integer/double vectors, finite or NA. Complex, classed, dimensional, infinite and NaN values fail. Names on numeric vectors are permitted.
absent_value	Required plain integer/double scalar, finite or NA_real_. No default is supplied. Choosing zero explicitly does not impute observed NA.

Details

Both tables require unique composite keys. Unknown data keys fail rather than being discarded. Registry metadata is authoritative and carried unchanged; extra data columns are ignored. Inputs are not mutated. Values are normalized to double; no arithmetic or aggregation is performed.

Value

A data frame with registry columns, the selected value (double), observed (logical), status (character), and absent_value (double policy). Status is OBSERVED for known matched values, OBSERVED_MISSING for matched NA, ABSENT_FILLED for absent rows with finite fill, or ABSENT_UNAVAILABLE for absent rows with NA fill. Row count and order equal the registry. Empty registry requires empty data and returns a typed empty result. Provenance distinguishes observed zero, filled zero, observed NA and absent NA.

Examples

```
d <- data.frame(bin = c("a", "b"), fraction = c(0.5, NA_real_))
expected <- data.frame(bin = c("a", "b", "c"), sample = "one")
CompleteRegisteredValues(d, expected, "bin", "fraction", absent_value = 0)
```

`ComposeFigure` *Compose a Multi-Panel Figure*

Description

Lays out panels, page-level texts and legend keys on a page of fixed size in millimetres, with bold panel letters and headings in the manuscript style. The result can be drawn on any device or written with `SaveFigure()`.

Usage

```
ComposeFigure(
  panels,
  height_mm,
  width_mm = NULL,
  texts = list(),
  style = ManuscriptStyle()
)
```

Arguments

`panels`	List of `FigurePanel()` objects.
`height_mm`	Page height in millimetres.
`width_mm`	Page width; defaults to the style's page width.
`texts`	Optional list of `FigureText()` objects and grobs placed with `FigureGrob()`.
`style`	A `ManuscriptStyle()` object.

Value

An object of class `phenoscapR_figure`; `print()` draws it on the current device.

Examples

```
library(ggplot2)
p <- ggplot(data.frame(x = 1:3, y = 1:3), aes(x, y)) + geom_point() +
  ThemeManuscript()
fig <- ComposeFigure(list(FigurePanel(p, c(12, 10, 50, 30), letter = "a",
                                      heading = "Example")), height_mm = 50)
```

CompositeFields *Composite Fluorescence Images from Channel Fields*

Description

Turns image fields (lists with a `channels` element of named matrices, as returned by `segmantR::sg_read_field()`) into RGB images for galleries: every view is an additive mixture of display-scaled channels, `clip((value - lo) / (hi - lo), 0, 1) ^ gamma * weight * colour`.

Usage

```
CompositeFields(
  fields,
  views,
  slide = NULL,
  levels = NULL,
  source = "source",
  gamma = 0.8,
  min_span = 1,
  style = ManuscriptStyle()
)
```

Arguments

`fields`	List of fields (`channels`, and `row` with the field's table row for column lookups).
`views`	Named list of views; each view is a list of layers.
`slide, levels`	Optional data frames with columns `channel`, `lo`, `hi` and, when fields come from several images, `source`.
`source`	Column of a field's `row` that matches `source` in the range tables.
`gamma`	Display gamma.
`min_span`	Minimum distance between the levels.
`style`	A `ManuscriptStyle()` object (resolves colour names).

Details

A layer is a list with `channel` (a channel name) or `column` (a column of the field's `row` table that holds the channel name), `colour` (hex or style colour), optional `weight` (default 1), `overlay` (paint over the mixture instead of adding) and display levels: `lo` and `hi`. `lo` is the name of a range table (`"slide"`, `"levels"`) or `"q0.01"` (a quantile of the field's own values); `hi` is one or several terms, each a table name, a table name with a factor (`"slide*0.9"`) or a quantile (`"q0.995"`); the largest term is used, and never less than `lo` + `min_span`.

Value

A list of class `phenoscapR_images` with `images` (list of RGB arrays) and `info` (data frame: `image`, `field`, `view` and the columns of each field's row).

Examples

```
set.seed(1)
field <- list(channels = list(dna = matrix(runif(100, 0, 50), 10),
                              cd3 = matrix(runif(100, 0, 20), 10)))
views <- list(merge = list(list(channel = "dna", colour = "#3366FF", lo = "q0", hi = "q0.99"),
                      list(channel = "cd3", colour = "#FF3300", lo = "q0.5", hi = "q1")))
imgs <- CompositeFields(field, views)
dim(imgs$images[[1]])
imgs$info
```

`CompositionPlot` *Stacked Bar Plot of Phenotype Composition*

Description

Stacked bar plot showing the proportion or count of each phenotype within each sample (or other grouping variable). Matches the stacked bar composition plots from the images.

Usage

```
CompositionPlot(
  object,
  group_by = "sample_id",
  proportion = TRUE,
  colours = NULL
)
```

Arguments

`object`	An `SpatialCellData` object with a `phenotype` column.
`group_by`	Character. Metadata column for the x-axis grouping. Default `"sample_id"`.
`proportion`	Logical. Show proportions instead of counts? Default `TRUE`.
`colours`	Named character vector or `NULL`.

Value

A `ggplot` object.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50), y = runif(50))
meta <- data.frame(cell_id = 1:50,
                   sample_id = rep(c("S1", "S2"), each = 25),
                   phenotype = sample(c("T cell", "B cell", "Mac"), 50, TRUE))
obj <- CreateSpatialObject(counts, coords, meta)
CompositionPlot(obj)
```

`ContrastGroupedOverlap` *Contrast Explicit Groups Within Finite-Metric Overlap Strata*

Description

Compares positive and control observations only within strata containing both labels with finite metric values. The caller supplies the policy-selected population, independent group keys, stratum assignments and binary labels. This function does not select a population, construct strata, establish independence, or infer a causal estimand or scientific eligibility policy.

Usage

```
ContrastGroupedOverlap(
  data,
  registry,
  groups,
  strata,
  id,
  positive,
  value,
  quality = NULL
)
```

Arguments

`data`	Data frame of policy-selected observations, including unavailable metric values. Rows are counted before filtering for finite metric values.
`registry`	Data frame declaring every group, including empty groups. Extra registry columns are ignored.
`groups`	Nonempty character vector of distinct group column names.
`strata`	Nonempty character vector of distinct stratum column names. Strata are nested within the complete group key and are supplied explicitly.
`id`	Character name of the observation ID column. IDs must be unique within a group, even when stratum assignments differ.
`positive`	Character name of a dimensionless logical column without NA. TRUE denotes positive observations and FALSE denotes controls.
`value`	Character name of a real numeric metric column. NA, NaN and infinite values count in policy totals but are not metric-evaluable.
`quality`	Optional named list of explicit nonnegative finite criteria: `min_positive_policy`, `min_positive_evaluable`, `min_positive_overlap`, `min_policy_coverage`, `min_evaluable_coverage`, and `min_control_ess`. Count criteria must be integers; coverage criteria must lie between zero and one (inclusive). Each criterion is optional. NULL requests no quality assessment.

Details

Group, stratum and observation keys must be nonblank, nonmissing character vectors without dimensions. Factors and numeric IDs are not coerced. All selected column roles must be distinct, and keys must not collide with output column names. Unknown groups and duplicate registry or observation keys are rejected. Row order does not define identity.

Overlap is determined separately within each group and stratum after filtering for finite metric values. In an overlap stratum, positives receive weight one and controls receive the finite positive count divided by the finite control count. Observations with unavailable values or outside overlap retain NA weights and explicit statuses. They are not discarded from the returned observation and stratum audit tables.

The contrast is the overlap-positive mean minus the weighted overlap-control mean. Both policy-positive and metric-evaluable-positive coverage denominators are reported. Effective control count is the squared sum of control weights divided by their sum of squares, computed after scaling weights to avoid avoidable overflow. Means similarly scale values and weights. Unrepresentable results or scaling/product underflow raise an error rather than silently dropping a nonzero contribution. Exact cancellation to zero is valid. Ordinary floating-point rounding still applies; this is not arbitrary-precision arithmetic.

All optional quality criteria are assessed separately. A failed criterion never suppresses an otherwise finite contrast. A PASS means only that the requested criteria passed, not that the contrast is defined or scientifically eligible; inspect `status` and the explicit denominators as well. Criteria using an unavailable denominator fail. No count population or threshold is selected implicitly. Policy, evaluable and overlap positive counts are distinct.

Value

List with `schema_version` ("1.0.0"), `summary`, `strata`, `observations`, and the canonical requested `quality` list. All tables are sorted by explicit keys; empty registries return typed empty tables. The summary contains policy/evaluable/overlap counts, stratum counts, both positive coverage fractions, positive/control evaluable fractions, `positive_mean`, `control_weighted_mean`, `contrast`, `control_weight_sum` and `control_effective_n`. A zero weight total denotes an empty matched control set; missing means and ESS remain NA, never zero-imputed. Summary status is EMPTY_GROUP, NO_EVALUABLE_VALUES, NO_OVERLAP or DEFINED; availability is UNAVAILABLE, PARTIAL or COMPLETE. Quality is NOT_REQUESTED, PASS or FAIL with semicolon-separated failed criterion names. The stratum table retains policy/evaluable counts and overlap status. The observation table retains keys, `is_positive`, `metric_value`, `metric_evaluable`, `overlap`, `weight`, and OVERLAP, NONFINITE_VALUE or NO_OVERLAP_STRATUM status. Extra input columns are not returned.

Examples

```
d <- data.frame(sample = "one", block = c("a", "a", "a", "b"),
  cell = letters[1:4], positive = c(TRUE, FALSE, FALSE, TRUE),
  score = c(4, 1, 3, NA))
registry <- data.frame(sample = c("one", "empty"))
result <- ContrastGroupedOverlap(d, registry, groups = "sample", strata = "block",
  id = "cell", positive = "positive", value = "score")
result$summary
```

`ContrastGroupedRanks` *Compare Two Distributions Within Explicit Groups*

Description

Computes descriptive Cliff's delta as the number of A-greater-than-B pairs minus A-less-than-B pairs, divided by all available cross-arm pairs. Ties contribute zero. No inferential p value or independence claim is supplied. Pair counts use sorted value frequencies without a cross-product matrix or value subtraction. Every registered group is retained, including empty arms.

Usage

```
ContrastGroupedRanks(
  data,
  registry,
  groups,
  id,
  arm,
  value,
  missing = c("propagate", "available")
)
```

Arguments

`data`	Data frame with group keys, observation IDs, arms and values.
`registry`	Data frame containing every allowed group key exactly once.
`groups`	Nonempty character vector naming distinct group columns.
`id`	Character name of the observation ID column; IDs are unique within each complete group key.
`arm`	Character name of the arm column, containing exactly `"A"` or `"B"`.
`value`	Character name of a numeric column containing finite values or explicit NA. NaN and infinities are rejected. Convert nonfinite input values to NA only through an explicit upstream policy.
`missing`	`"propagate"` suppresses delta if any supplied value is NA; `"available"` explicitly uses available values independently in each arm.

Details

Group keys and observation IDs must be dimensionless, nonmissing, nonblank character vectors. Factors and numeric IDs are not coerced. The reserved ASCII 28 separator (`intToUtf8(28)`) is rejected in keys. Column selections must be distinct, and group columns must not collide with output names. Registry keys must be unique. Every observation must belong to a registered group and have a unique ID within that complete group, including across arms. The same ID may occur in different groups. Arm values must be exactly `"A"` or `"B"`; no labels or population policy are inferred.

Delta is positive when A tends to exceed B and negative when B tends to exceed A. It equals the proportion of available cross-arm pairs with A greater than B minus the proportion with A less than B. Equal values, including signed zeros, are ties. Comparisons use represented numeric values without a tolerance. No value subtraction, cross-pair matrix, rank test or resampling is performed. Pair counts above `2^53 - 1` are rejected before multiplication rather than rounded. Sorting distinct values and counting frequencies avoids allocating a Cartesian product, but input and distinct values must still fit in memory.

Available-case analysis filters missing values separately within each arm; it is not paired-observation deletion. Propagation suppresses delta when either supplied arm contains missing values, while still reporting the available pair counts. If an available arm is empty, status is `UNAVAILABLE` under either policy. A declared group without observations is retained. Group membership and missingness policies do not establish independence or exchangeability. This function computes a descriptive effect only.

Value

Data frame sorted by group keys, preserving original key values. Supplied, available and unavailable counts are returned for each arm, together with available cross-pair, greater, less and tied counts, delta, missingness policy and status. Empty available arms yield UNAVAILABLE; otherwise propagation with missing values yields MISSING_VALUES, followed by PARTIAL or COMPLETE. Pair counts always describe available values even when delta is suppressed. Counts exceeding exact binary64 integer capacity are rejected before multiplication. An empty registry returns typed output.

Examples

```
d <- data.frame(
  g = "one", id = letters[1:4],
  arm = c("A", "A", "B", "B"), value = c(2, 4, 1, 2)
)
ContrastGroupedRanks(d, data.frame(g = "one"), "g", "id", "arm", "value")
```

`ContrastPatientEndpoints` *Exact Two-Group Contrasts for a Registry of Patient Endpoints*

Description

Runs `ExactPatientTests()` for every row of a hypothesis registry (endpoint, contrast, family) on a wide patient table (one row per patient, one column per endpoint). Contrast arms are declared as sets of patient groups. Two missingness policies are offered:

Usage

```
ContrastPatientEndpoints(
  data,
  registry,
```

```
  contrasts,
  id,
  group,
  endpoint = "endpoint",
  contrast = "contrast",
  family = "family",
  missingness = c("complete_case", "require_complete"),
  hypothesis_prefix = "h",
  max_allocations = 1000000L
)
```

Arguments

`data`	Wide patient table with the `id`, `group` and endpoint columns.
`registry`	Data frame with one row per hypothesis and the `endpoint`, `contrast` and `family` columns. Extra columns are ignored.
`contrasts`	Named list: contrast name to a list with character vectors `A` and `B` of patient groups (disjoint).
`id`, `group`	Patient identifier and patient group columns of `data`.
`endpoint`, `contrast`, `family`	Column names in `registry`.
`missingness`	`"complete_case"` or `"require_complete"`.
`hypothesis_prefix`	Prefix of the generated `hypothesis_id` (followed by the registry row number).
`max_allocations`	Passed to `ExactPatientTests()`.

Details

- `"complete_case"`: each hypothesis uses only patients with a finite value; a hypothesis whose arm has no such patient is unavailable. This reproduces tables that silently drop missing patients.
- `"require_complete"`: every patient of both arms must have a finite value, otherwise the hypothesis is unavailable (no patient is dropped).

Each hypothesis receives its own design, so a registry may repeat an endpoint-contrast pair in different families. The effect is the mean of arm A minus the mean of arm B.

Value

A data frame with the registry columns, `hypothesis_id`, `family_id` (`family :: contrast`), expected and evaluable arm sizes, `mean_difference`, `p_value`, `allocations`, `status` and `missingness`, in registry order. Pass it to `AdjustPatientFamilies()`.

Examples

```
d <- data.frame(patient = paste0("p", 1:6), arm = rep(c("x", "y"), each = 3),
                e1 = c(1, 2, 3, 6, 7, NA), e2 = c(5, 4, 6, 1, 2, 3))
reg <- data.frame(endpoint = c("e1", "e2"), contrast = "x vs y", family = "f")
ctr <- list("x vs y" = list(A = "x", B = "y"))
ContrastPatientEndpoints(d, reg, ctr, "patient", "arm")
ContrastPatientEndpoints(d, reg, ctr, "patient", "arm",
                         missingness = "require_complete")
```

Coords *Get Spatial Coordinates*

Description

Get Spatial Coordinates

Usage

```
Coords(object)

## S4 method for signature 'SpatialCellData'
Coords(object)
```

Arguments

object	A SpatialCellData object.

Value

Data frame with columns x and y.

Examples

```
data(phenoscapR_example)
head(Coords(phenoscapR_example))
```

CorrelateGroupedRanks *Correlate Paired Ranks Within Explicit Groups*

Description

Computes descriptive Spearman correlations over paired observations, using average ranks for ties. Each observation must occur once per complete group key. Include the independent sampling unit in groups; no independence, eligibility, causal interpretation or inferential p value is established. Missingness is paired: a row is available only when both values are known. The default suppresses correlation when any pair is unavailable. Explicit available-case analysis retains the supplied and unavailable pair counts.

Usage

```
CorrelateGroupedRanks(
  data,
  groups,
  id,
  x,
  y,
  min_pairs = 3L,
  missing = c("propagate", "available")
)
```

Arguments

data	Data frame with group keys, observation IDs and paired values.
groups	Nonempty character vector of distinct group columns with nonempty character values and no missing keys.
id	Character column naming observations within each group.
x, y	Distinct numeric value columns. Finite numbers and NA are allowed; NaN and infinite values are rejected.
min_pairs	Integer minimum available pair count, at least two.
missing	Missingness policy: "propagate" or explicit "available".

Value

Data frame sorted by group keys with n_supplied, n_available, n_unavailable, available-pair n_unique_x and n_unique_y, rho, status, and missing_policy. Status precedence is UNAVAILABLE, INSUFFICIENT, MISSING_PAIRS (propagation), CONSTANT_BOTH, CONSTANT_X, CONSTANT_Y, then COMPLETE or PARTIAL. Suppressed correlations are NA. Empty input returns a typed empty table; absent groups are not invented.

Examples

```
d <- data.frame(sample = "one", cell = letters[1:4],
                a = c(1, 1, 3, 4), b = c(4, 3, 2, 1))
CorrelateGroupedRanks(d, "sample", "cell", "a", "b")
```

CountDisjointEdges *Count Undirected Edges Between Disjoint Node Sets*

Description

Counts an explicitly supplied edge set using stable node IDs within complete group and frame keys. Isolated nodes and nodes in neither set remain in the supplied graph totals. No graph is constructed, no distance cap is applied, and no scientific population, radius or eligibility policy is inferred.

Usage

```
CountDisjointEdges(
  nodes,
  edges,
  registry,
  groups,
  frame,
  id,
  from,
  to,
  source,
  target
)
```

Arguments

nodes	Data frame containing every node, including isolated nodes.
edges	Data frame with one row per undirected edge. Extra columns, including distances, are ignored; filtering is the caller's responsibility.
registry	Data frame with unique group/frame keys, including desired empty frames, and character columns geometry_source and geometry_reference. The source is "frozen_import" or "supplied_edges"; the reference is a nonempty caller-declared artefact or provenance reference.
groups	Nonempty character vector naming distinct group columns shared by all three tables. Include the independent sample key as appropriate.
frame	Character name of the frame column shared by all three tables.
id	Character name of the node ID column in nodes.
from, to	Character names of endpoint ID columns in edges.
source, target	Character names of logical membership columns in nodes. Missing membership and overlapping sets are rejected. Nodes in neither set are valid.

Details

Keys and IDs must be dimensionless, nonmissing, nonblank character vectors. IDs may repeat across different groups or frames, but must be unique within the complete group/frame key. Factors and numeric IDs are not coerced. Duplicate registry keys, unknown endpoints, self edges, and duplicate undirected pairs (including reversed duplicates) raise errors. Each edge declares one shared frame, so a cross-frame relationship cannot be represented; an endpoint absent from the declared frame raises an error. The function cannot detect incorrectly declared frames or establish physical coordinate validity.

Each source-target edge contributes once regardless of orientation. Conditional neighbour means divide cross-edge count by the number of distinct interacting source or target nodes. Contact fractions divide interacting nodes by the corresponding complete source or target population. Thus isolated nodes enter contact-fraction denominators but not conditional-neighbour denominators.

A population with no contacts has contact fraction zero (DEFINED) and conditional mean NA (NO_INTERACTING_SOURCE or NO_INTERACTING_TARGET). An absent population has contact fraction NA (NO_SOURCE or NO_TARGET).

Missing membership never becomes an observed zero. Empty registries return the same typed zero-row schema.

Provenance is copied from the registry, not authenticated. In particular, `frozen_import` does not mean geometry was regenerated or its hashes verified. The result declares `geometry_recomputed = FALSE` and `geometry_authentication = "CALLER_DECLARED_NOT_VERIFIED"`. This API does not invoke or certify any triangulation, distance, review or geometry backend.

Value

Data frame sorted by group/frame keys. Schema version `"1.0.0"` has integer-valued numeric counts `n_nodes`, `n_edges`, `n_source`, `n_target`, `n_cross_edges`, `n_interacting_source`, and `n_interacting_target`; `source_neighbor_mean`, `target_neighbor_mean`, `source_contact_fraction`, and `target_contact_fraction`; and their corresponding `_status` columns. Supplied graph totals include all nodes and edges, even those contributing no source-target contact. Geometry provenance columns and `schema_version` are retained explicitly. Extra input columns are not returned.

Examples

```
nodes <- data.frame(sample = rep("one", 3), frame = rep("image", 3),
  cell = c("a", "b", "isolated"), is_source = c(TRUE, FALSE, TRUE),
  is_target = c(FALSE, TRUE, FALSE))
edges <- data.frame(sample = "one", frame = "image", from = "b", to = "a")
registry <- data.frame(sample = c("one", "empty"), frame = "image",
  geometry_source = "supplied_edges", geometry_reference = "example-v1")
CountDisjointEdges(nodes, edges, registry, groups = "sample", frame = "frame",
  id = "cell", from = "from", to = "to", source = "is_source", target = "is_target")
```

CountRadiusSets *Count Reference Sets Within Explicit Euclidean Radii*

Description

Counts reference neighbours and overlapping reference sets for each query ID within an explicitly declared coordinate frame. Query batching always searches the complete reference population of that frame. No population, coordinate transformation, graph, tissue area or scientific threshold is inferred.

Usage

```
CountRadiusSets(
  nodes,
  frames,
  groups,
  frame,
  id,
  x,
  y,
  query,
```

```
  reference,
  sets,
  unit_column,
  radius_unit,
  radii,
  search_radius,
  chunk_size = 1000L
)
```

Arguments

nodes	Data frame containing node keys, coordinates and memberships.
frames	Data frame declaring unique group/frame keys and coordinate units, including desired empty frames.
groups	Nonempty character vector naming distinct group columns shared by both tables. Include independent sample keys as appropriate.
frame	Character name of the shared coordinate-frame column.
id	Character name of the node ID column in nodes.
x, y	Character names of finite numeric coordinate columns in nodes.
query, reference	Character names of nonmissing logical query and reference memberships in nodes. Memberships may overlap.
sets	Nonempty character vector naming distinct nonmissing logical reference-set columns in nodes. Sets may overlap. Membership outside reference does not expand the reference population.
unit_column	Character name of the units column in frames.
radius_unit	One of "nm", "um", "mm", "cm", or "m". Every declared frame must use this unit; no unit conversion is performed.
radii	Distinct positive finite numeric output radii in radius_unit.
search_radius	Required finite numeric search envelope, at least every requested radius. This is part of the numerical contract, not a default.
chunk_size	Positive integer number of queries per native search batch.

Details

Keys and IDs must be dimensionless, nonmissing, nonblank character vectors. IDs must be unique within the complete group/frame key, but may repeat across independent frames. Factors and numeric IDs are not coerced. Selected roles within each input table must be distinct. Unknown frames, duplicate keys, missing memberships and malformed inputs raise errors. Extra node columns are ignored; a tile column does not partition a frame unless selected as a key.

Searches use dbscan::frNN() with explicit query mode, search = "kdtree", approx = 0 and sort = FALSE, without a fallback. Returned distances are filtered inclusively using distance <= radius. Self exclusion compares IDs within the complete frame key. Distinct coincident IDs remain neighbours.

Exact search mode does not imply arbitrary-precision real-number geometry. Binary64 boundary decisions can depend on `search_radius`: the distance from `(0, 0)` to `(3 + 2 * .Machine$double.eps, 4)` is excluded by a native search at 5, but a search at 6 returns a distance rounded to 5, which passes the output-radius filter at 5. Keep the search envelope fixed when requesting different subsets of radii or comparing runs. No padding, recentring, coordinate rounding or squared-distance replacement is applied.

Squared output and search radii must be finite and at least `.Machine$double.xmin`. Absolute coordinates must not exceed `sqrt(.Machine$double.xmax) / 4`; frame spans must support finite squared arithmetic. Unsupported extreme inputs and integer indexing/output sizes raise errors. These conservative guards do not remove ordinary rounding. Chunking bounds each native query batch, not dense-neighbourhood memory within a batch or total output size. This function does not authenticate input coordinates or historical geometry and does not infer fractions or weights.

Value

A list with `schema_version = "1.0.0"` and:

- `frame_summary`: one row per declared frame, coordinate units, `n_nodes`, `n_queries`, `n_references`, and `status` (`NO_QUERIES`, `EMPTY_REFERENCE`, or `DEFINED`).
- `query_counts`: every query by requested radius, including zero neighbours, with `n_reference_neighbors` and `status` (`EMPTY_REFERENCE` or `DEFINED`). A nonempty reference universe containing only the query itself has status `DEFINED` and zero eligible neighbours.
- `set_counts`: every query by radius by set, with integer-valued double `count` and matching status. Overlapping sets need not partition neighbours.
- `backend`: installed version, search settings and radius/unit, DESCRIPTION, R-wrapper and native-library SHA-256 fingerprints. These identify the current runtime and do not authenticate a historical backend.

Keys and IDs are retained. Tables are sorted by keys, IDs, radii and set names as applicable. Empty inputs return typed empty tables; inputs are not modified.

Examples

```
nodes <- data.frame(
  sample = "sample-1", frame = "image-1", id = c("a", "b", "c"),
  x = c(0, 0, 3), y = c(0, 0, 4),
  query = c(TRUE, FALSE, FALSE), reference = TRUE,
  set_a = c(FALSE, TRUE, TRUE), set_b = c(FALSE, TRUE, FALSE)
)
frames <- data.frame(sample = "sample-1", frame = "image-1", unit = "um")
CountRadiusSets(nodes, frames, "sample", "frame", "id", "x", "y",
  "query", "reference", c("set_a", "set_b"), "unit", "um",
  radii = c(1, 5), search_radius = 5
)
```

`CreateSpatialObject` *Create a SpatialCellData Object from Existing Data*

Description

Constructs a `SpatialCellData-class` object from a counts matrix, coordinate data frame, and optional metadata.

Usage

```
CreateSpatialObject(
  counts,
  coords,
  meta_data = NULL,
  sample_id = "sample1",
  project = "SpatialProject"
)
```

Arguments

`counts`	Real numeric matrix (cells x markers), without infinity. Missing values are allowed. Row names are optional. Empty matrices retain their marker columns; all-missing columns in nonempty inputs are dropped with a warning. Logical placeholders with no entries, as produced by zero-row or zero-marker matrices, are stored as numeric matrices.
`coords`	Data frame with finite numeric columns `x` and `y`. Zero-row logical coordinate placeholders are stored as numeric vectors; nonempty logical values and factors are refused.
`meta_data`	Data frame of per-cell metadata, or `NULL`.
`sample_id`	Character. Sample identifier. Default `"sample1"`.
`project`	Character. Project name. Default `"SpatialProject"`.

Value

A `SpatialCellData-class` object.

Examples

```
counts <- matrix(rnorm(50), nrow = 10,
                 dimnames = list(NULL, c("CD3", "CD8", "CD20", "DAPI", "PanCK")))
coords <- data.frame(x = runif(10, 0, 500), y = runif(10, 0, 500))
obj <- CreateSpatialObject(counts, coords, sample_id = "mysample")
obj
```

CrossNNDistance *Cross Nearest Neighbour Distance*

Description

Computes the nearest neighbour distance from each cell of one phenotype to cells of another phenotype. With `from` equal to `to`, the distance is to the nearest other cell of that phenotype.

Usage

```
CrossNNDistance(object, from, to, by_sample = FALSE)
```

Arguments

`object`	An `SpatialCellData-class` object.
`from`	Character. Source phenotype.
`to`	Character. Target phenotype.
`by_sample`	Logical. If `TRUE` and the object holds several samples, the statistic is computed per sample and returned as a named list. Default `FALSE`; otherwise a single sample is required.

Value

A numeric vector of distances (one per cell of the `from` phenotype).

Examples

```
counts <- matrix(rnorm(60), nrow = 30,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(30, 0, 100), y = runif(30, 0, 100))
meta <- data.frame(cell_id = 1:30, sample_id = "s1",
  phenotype = rep(c("A", "B", "C"), each = 10))
obj <- CreateSpatialObject(counts, coords, meta)
CrossNNDistance(obj, from = "A", to = "B")
```

CustomGradient *Create a Custom Gradient Palette from Anchor Colours*

Description

Creates a gradient palette function from 2 or 3 anchor colours. The returned function takes an integer `n` and returns `n` colours.

Usage

```
CustomGradient(colours)
```

Arguments

colours	Character vector of 2-3 colours (names or hex codes).

Value

A function that takes integer n and returns n hex colour codes.

Examples

```
pal <- CustomGradient(c("navy", "white", "firebrick"))
cols <- pal(100)
image(matrix(1:100, ncol = 1), col = cols)
```

DelaunayNetwork *Delaunay Triangulation Network*

Description

Computes a Delaunay triangulation of cell positions and stores edges in the spatial slot. Each sample_id is triangulated separately; it must identify one independent spatial coordinate frame. Every sample must contain at least three cells. Edge endpoints refer to cell rows in the complete object, never rows local to a sample. Of cells with identical coordinates, only the first is a vertex; the others get no edges.

Usage

```
DelaunayNetwork(object, max_edge = NULL)
```

Arguments

object	An SpatialCellData-class object.
max_edge	Numeric or NULL. Maximum edge length to retain.

Value

An SpatialCellData-class with delaunay_edges stored in the spatial slot.

Examples

```
counts <- matrix(rnorm(40), nrow = 20,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(20, 0, 100), y = runif(20, 0, 100))
obj <- CreateSpatialObject(counts, coords)
obj <- DelaunayNetwork(obj, max_edge = 30)
```

DensestGridWindow *Centre of the Densest Grid Window*

Description

Bins cells into square windows of a regular grid with origin 0 and returns the centre of the window holding the most cells (ties: the window with the smallest x index, then y index).

Usage

```
DensestGridWindow(cells, size, filter = list(), x = "x", y = "y")
```

Arguments

`cells`	Data frame of cells.
`size`	Window width.
`filter`	Optional named list: column to values; only matching cells are counted. Use `NA` among the values to count missing values.
`x`, `y`	Column names.

Value

A one-row data frame with `x`, `y` (window centre), `x0`, `y0` (window corner) and `n`.

Examples

```
d <- data.frame(x = c(1, 2, 15, 16, 17), y = c(1, 2, 5, 6, 7))
DensestGridWindow(d, 10)
```

DensityLevelContour *Density Level Region of Each Group on an Embedding*

Description

For each group (for example each patient), counts the points in a `bins` x `bins` histogram over the display limits, smooths the counts with a Gaussian kernel of `sigma` bins (mirror boundaries, kernel truncated at `truncate` sigma), and finds the density level whose superlevel region holds the fraction `level` of the smoothed mass: the smallest bin value that is still needed when bins are added from the highest down until their sum reaches `level` times the total. The region above that level is returned as filled bands and outline paths on the bin centres.

Usage

```
DensityLevelContour(
  data,
  x = "x",
  y = "y",
  group = "group",
  limits,
  filter = NULL,
  bins = 90,
  sigma = 2,
  level = 0.9,
  truncate = 4,
  groups = NULL,
  carry = NULL,
  slot_within = NULL,
  label_from = NULL,
  label_sub = NULL
)
```

Arguments

`data`	Data frame with coordinates and a group column.
`x`, `y`, `group`	Column names.
`limits`	Display limits `c(xmin, xmax, ymin, ymax)` or a data frame from `EmbeddingLimits()`; the histogram spans them exactly.
`filter`	Optional named list of columns and values selecting the points used.
`bins`	Bins per axis.
`sigma`	Kernel standard deviation in bins.
`level`	Fraction of the smoothed mass inside the region.
`truncate`	Kernel half-width in standard deviations.
`groups`	Optional group order (also the order of slots).
`carry`	Columns constant within a group copied to the output (for example the clinical group).
`slot_within`	Optional carried column: slots count groups within each of its values (for example patients within a clinical group).
`label_from`, `label_sub`	Column for the key label (default the group) and an optional regular expression replacement `c(pattern, replacement)` applied to it.

Value

A data frame with the group column (named as in `data`), `slot`, `label`, carried columns, `kind` (`"band"`, `"line"`), `piece`, `x`, `y`, `level` (the density level) and `n` (points of the group).

Examples

```
set.seed(3)
d <- data.frame(x = rnorm(400), y = rnorm(400), g = rep(c("a", "b"), 200))
head(DensityLevelContour(d, group = "g", limits = c(-4, 4, -4, 4), bins = 30))
```

`DensityPlot` *Plot Cell Density*

Description

Scatter plot with colour scaled by local cell density. Requires `CellDensity` to have been run.

Usage

```
DensityPlot(object, pt_size = 1, title = NULL, palette = NULL)
```

Arguments

`object`	An `SpatialCellData` object with a `density` column.
`pt_size`	Numeric. Point size. Default `1`.
`title`	Character or `NULL`.
`palette`	Character or character vector. Default `NULL` (global palette).

Value

A `ggplot` object.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 100), y = runif(50, 0, 100))
obj <- CreateSpatialObject(counts, coords)
obj <- CellDensity(obj, radius = 20)
DensityPlot(obj)
```

`DifferentialAbundance` *Differential Abundance of Phenotypes Across Conditions*

Description

Tests whether the proportion of each phenotype differs between experimental conditions. Proportions are computed per sample (the unit of replication), then compared across the condition groups, so the test respects the sample-level design rather than treating individual cells as independent.

Usage

```
DifferentialAbundance(
  object,
  condition,
  phenotype_col = "phenotype",
  test = c("wilcox", "t", "kruskal")
)
```

Arguments

`object`	A `SpatialCellData-class` object with several samples.
`condition`	Character. Metadata column giving each cell's condition (constant within a sample). Missing or empty conditions and sample IDs are rejected, as are conflicting conditions within a sample.
`phenotype_col`	Character. Phenotype column. Default `"phenotype"`.
`test`	Character. `"wilcox"` (default) or `"t"` for two groups; `"kruskal"` for more than two.

Value

A classed data frame (`phenoscapR_diffabund`) with one row per phenotype: per-condition mean proportions, the test `statistic`, `p_value`, and BH-adjusted `p_adj`.

Examples

```
data(phenoscapR_example)
obj <- phenoscapR_example
obj@meta_data$phenotype <- obj@meta_data$phenotype_true
# Toy condition: label the two sections as different arms.
obj@meta_data$arm <- ifelse(obj$sample_id == "tonsil_A", "ctrl", "treat")
DifferentialAbundance(obj, condition = "arm")
```

dim,SpatialCellData-method *Get Dimensions of a SpatialCellData Object*

Description

Get Dimensions of a SpatialCellData Object

Usage

```
## S4 method for signature 'SpatialCellData'
dim(x)
```

Arguments

x	A SpatialCellData object.

Value

Integer vector: number of cells, number of markers.

DotPlot *Dot Plot of Marker Expression*

Description

Dot plot where dot size represents the percentage of cells expressing a marker (above a threshold) and colour represents the mean expression. Analogous to Seurat's DotPlot.

Usage

```
DotPlot(
  object,
  features,
  group_by = "phenotype",
  slot = "data",
  threshold = 0,
  palette = NULL
)
```

Arguments

object	An SpatialCellData object.
features	Character vector. Marker names to plot.
group_by	Character. Metadata column to group by. Default "phenotype".
slot	Character. "data" or "counts".
threshold	Numeric. Expression threshold to call a cell "expressing". Default 0.
palette	Character or character vector. Colour palette for mean expression. Default NULL (global palette).

Value

A ggplot object.

Examples

```
counts <- matrix(c(rnorm(60, 5), rnorm(60, 0)), ncol = 4,
                 dimnames = list(NULL, c("CD3", "CD8", "CD20", "PanCK")))
coords <- data.frame(x = runif(30), y = runif(30))
meta <- data.frame(cell_id = 1:30, sample_id = "s1",
                   phenotype = rep(c("T cell", "B cell", "Other"), each = 10))
obj <- CreateSpatialObject(counts, coords, meta)
DotPlot(obj, features = c("CD3", "CD8", "CD20", "PanCK"))
```

EmbeddingLayers *Layered Point Data for an Embedding*

Description

Stacks the points of several layers in drawing order: each layer selects rows by column values, takes its key either as a fixed name or from a column, and may be put in random order (for example so that no cell type is systematically drawn on top). The random order is a permutation from R's default generator after set.seed(seed); the caller's random state is left unchanged.

Usage

```
EmbeddingLayers(data, x = "x", y = "y", layers, carry = NULL, seed = NULL)
```

Arguments

data	Data frame with coordinates.
x, y	Coordinate columns.
layers	List of layers, each a list with filter (named list of columns and values), key (fixed key) or key_from (column holding the key), optional recode (named mapping of keys) and shuffle.
carry	Further columns copied to the output.
seed	Seed for shuffled layers.

Value

A data frame with x, y, key, layer and the carried columns, in drawing order.

Examples

```
d <- data.frame(u = 1:6, v = 6:1, type = c("a", "b", "a", "c", "b", "a"), kept = TRUE)
EmbeddingLayers(d, "u", "v", layers = list(
  list(filter = list(kept = TRUE), key = "background"),
  list(filter = list(type = "a"), key_from = "type", shuffle = TRUE)), seed = 1)
```

EmbeddingLimits *Display Limits of an Embedding*

Description

Limits enclosing the central part of the points plus a margin: the probs quantiles of each coordinate (type 7, linear interpolation) widened by pad times their span on both sides.

Usage

```
EmbeddingLimits(
  data,
  x = "x",
  y = "y",
  filter = NULL,
  probs = c(5e-04, 0.9995),
  pad = 0.04
)
```

Arguments

data	Data frame with coordinates.
x, y	Coordinate columns.
filter	Optional named list of columns and values selecting the points used.
probs	Lower and upper quantile.
pad	Margin as a fraction of the quantile span.

Value

A one-row data frame with xmin, xmax, ymin, ymax.

Examples

```
EmbeddingLimits(data.frame(x = 1:100, y = (1:100)^2))
```

EmbeddingPlot *Plot a Dimensionality-Reduction Embedding*

Description

Scatter plot of cells in an embedding (PCA, UMAP, t-SNE, or SONG), coloured by a metadata column. Analogous to Seurat's `DimPlot`.

Usage

```
EmbeddingPlot(
  object,
  reduction = "umap",
  colour_by = "phenotype",
  colours = NULL,
  pt_size = 1,
  title = NULL,
  dark_theme = FALSE
)

DimPlot(
  object,
  reduction = "umap",
  colour_by = "phenotype",
  colours = NULL,
  pt_size = 1,
  title = NULL,
  dark_theme = FALSE
)
```

Arguments

`object`	A `SpatialCellData-class` object.
`reduction`	Character. Reduction to plot. Default `"umap"`.
`colour_by`	Character. Metadata column for colour. Default `"phenotype"`.
`colours`	Named character vector or `NULL`.
`pt_size`	Numeric. Point size. Default `1`.
`title`	Character or `NULL`.
`dark_theme`	Logical. Dark background. Default `FALSE`.

Value

A `ggplot` object.

Examples

```
counts <- matrix(rnorm(500), nrow = 50,
                 dimnames = list(NULL, paste0("M", 1:10)))
coords <- data.frame(x = runif(50), y = runif(50))
obj <- CreateSpatialObject(counts, coords)
obj <- RunPCA(obj, n_pcs = 5)
obj <- PhenotypeCells(obj, thresholds = list(M1 = 0))
EmbeddingPlot(obj, reduction = "pca")
```

Embeddings *Get a Dimensionality-Reduction Embedding*

Description

Get a Dimensionality-Reduction Embedding

Usage

```
Embeddings(object, reduction = "pca")

## S4 method for signature 'SpatialCellData'
Embeddings(object, reduction = "pca")
```

Arguments

`object`	A `SpatialCellData-class` object.
`reduction`	Character. Name of the reduction (e.g. `"pca"`, `"umap"`).

Value

A numeric matrix (cells x dimensions).

Examples

```
data(phenoscapR_example)
Embeddings(RunPCA(phenoscapR_example, n_pcs = 5), "pca")[1:3, ]
```

`ExactPairedTests` *Exact Paired Sign-Flip Tests of Within-Subject Differences*

Description

For each declared group of rows, tests whether paired differences are symmetric about zero with the exact Wilcoxon signed-rank test over all 2^n sign assignments, together with the exact two-sided sign test and, optionally, a Clopper-Pearson interval for the fraction of subjects whose difference lies in an expected direction. Leave-one-subject-out results can be requested instead of the full tests.

Usage

```
ExactPairedTests(
  data,
  value,
  groups = NULL,
  id = NULL,
  expected = NULL,
  family = NULL,
  omit = c("none", "each"),
  conf_level = 0.95,
  max_n = 500L
)
```

Arguments

`data`	Data frame in long form: one row per subject and test.
`value`	Name of the numeric difference column (for a one-sample test, the values themselves).
`groups`	Optional names of columns that define separate tests. `NULL` treats all rows as one test. Output groups follow first appearance.
`id`	Optional subject column; required for `omit = "each"`. Subjects must be unique within a group.
`expected`	Optional expected direction: `"less"` or `"greater"` for every group, or the name of a column holding `"less"`/`"greater"` (constant within a group). Adds the number of subjects in that direction and its Clopper-Pearson interval.
`family`	Optional columns defining multiplicity families; adds `family_id` (values joined by `" :: "`). `hypothesis_id` joins the group values the same way. Both are ready for `AdjustPatientFamilies()`.
`omit`	`"none"` (full tests) or `"each"`: one row per group and omitted subject (all subjects of the group, in data order), with the omitted subject in column `omitted`.
`conf_level`	Confidence level of the direction interval.
`max_n`	Largest number of nonzero differences accepted.

Details

Non-finite differences are unavailable; zero differences are excluded from the signed-rank and sign tests (they still count as available subjects and as not being in the expected direction). Ranks of absolute differences use midranks for ties. The two-sided p value is the fraction of sign assignments whose absolute signed-rank sum is at least the observed one. It is computed by dynamic programming over twice the midranks (integers), so it equals full enumeration exactly for up to 52 nonzero differences without enumerating them.

Value

A data frame with the group columns, `n_available`, `n_nonzero`, `n_zero`, `positive`, `negative`, `median_difference` and `mean_difference` (of the nonzero differences), `signed_rank_sum`, `p_value` (signed-rank), `sign_p_value`, `status`, optionally `expected`, `expected_direction`, `direction_ci_low`, `direction_ci_high`, `conf_level`, and `hypothesis_id` (plus `family_id`).

See Also

`PairedDifferences()`, `AdjustPatientFamilies()`

Examples

```
d <- data.frame(test = rep(c("a", "b"), each = 6), subject = rep(1:6, 2),
                difference = c(-3, -1, -2, -5, 1, -4, 2, -1, 3, 0, 1, 2))
ExactPairedTests(d, "difference", groups = "test", id = "subject",
                 expected = "less")
ExactPairedTests(d, "difference", groups = "test", id = "subject",
                 omit = "each")
```

ExactPatientTests *Exact Allocation Tests of Independent Patient Endpoints*

Description

Enumerates every allocation with the declared group sizes. The statistic is the mean in A minus the mean in B; the two-sided p value is the fraction with absolute statistic at least the observed absolute statistic. Observed and allocated statistics are computed alike from values centred on their mean. Two statistics count as tied when they differ by no more than the rounding of this arithmetic (relative to the largest centred value) plus the rounding of the stored values (half a unit in the last place of the largest absolute value), so p values do not depend on the scale of the values. A shift leaves them unchanged while the rounding of the shifted values stays below the spacing of distinct statistics; for example, whole numbers from 0 to 20 for up to 12 patients keep their exact p values up to a shift of 1e13, while larger shifts can merge distinct statistics into ties. If an intermediate tolerance bound overflows, the same bound is evaluated with machine precision multiplied first; finite ordinary bounds are unchanged. This requires exchangeability under the null. Observational confounding, paired designs and stratified randomisation are not addressed by this test. Repeated images must be aggregated to one endpoint per patient beforehand.

Usage

```
ExactPatientTests(
  endpoints,
  design,
  hypotheses,
  missingness = "require_complete",
  max_allocations = 1000000L
)
```

Arguments

endpoints	Data frame with character endpoint_id, patient_id, numeric value, and logical eligible. At most one row per endpoint and patient. NA values or eligibility mark unavailable observations; NaN and infinite values are rejected. Extra columns are ignored.
design	Data frame with character contrast_id, patient_id, and group (exactly "A" or "B"). Each patient occurs once per contrast; both groups must be nonempty. Patients can belong to multiple contrasts.
hypotheses	Data frame with unique character hypothesis_id and character endpoint_id, contrast_id, family_id. Every declared row, including unavailable tests, remains in the result and family count.
missingness	Only "require_complete" is supported. All expected patients must have finite eligible endpoint values. No patient is dropped.
max_allocations	Positive integer ceiling on exhaustive allocations, at most one million. Exceeding it fails rather than using an approximation.

Value

A data frame with the registry, expected/evaluable group sizes, mean_difference, p_value, allocation count, probability grid step, and status. Grid step is a resolution bound, not a guaranteed attainable minimum two-sided p value. Unavailable effects and p values remain NA.

Examples

```
endpoints <- data.frame(endpoint_id = "e", patient_id = letters[1:4],
                        value = 1:4, eligible = TRUE)
design <- data.frame(contrast_id = "c", patient_id = letters[1:4],
                     group = c("A", "A", "B", "B"))
hypotheses <- data.frame(hypothesis_id = "h", endpoint_id = "e",
                         contrast_id = "c", family_id = "f")
ExactPatientTests(endpoints, design, hypotheses)
```

ExactPermutationBound *Lower Bound of an Exact Two-Sided Permutation P Value*

Description

For an exact permutation test of two groups of sizes `n_a` and `n_b` (all allocations of the pooled values), the observed allocation always reaches its own statistic; with equal group sizes its mirror allocation reaches the same absolute difference. The smallest attainable two-sided P is therefore at least `2 / choose(n_a + n_b, n_a)` for equal and `1 / choose(n_a + n_b, n_a)` for unequal sizes.

Usage

```
ExactPermutationBound(n_a, n_b)
```

Arguments

`n_a`, `n_b`	Group sizes.

Value

Numeric lower bound.

Examples

```
ExactPermutationBound(6, 6)
ExactPermutationBound(4, 2)
```

ExpressionClusters *Expression-Based Cell Clustering*

Description

Clusters cells based on marker expression profiles.

Usage

```
ExpressionClusters(
  object,
  k,
  method = c("kmeans", "hierarchical", "gmm"),
  slot = "data",
  markers = NULL
)
```

Arguments

object	An SpatialCellData-class object.
k	Integer. Number of clusters.
method	Character. "kmeans" (default) or "hierarchical".
slot	Character. "data" (default) or "counts".
markers	Character vector or NULL. Markers to use.

Value

An SpatialCellData-class with an expr_cluster column in meta_data.

Examples

```
counts <- matrix(c(rnorm(50, 10), rnorm(50, 0)), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50), y = runif(50))
obj <- CreateSpatialObject(counts, coords)
obj <- ExpressionClusters(obj, k = 2)
table(Meta(obj)$expr_cluster)
```

FeaturePlot *Feature Plot in Tissue Space*

Description

Plots marker expression intensity on spatial coordinates. Each marker gets its own panel. Similar to Seurat's FeaturePlot but in tissue space.

Usage

```
FeaturePlot(
  object,
  features,
  slot = "data",
  pt_size = 0.3,
  ncol = NULL,
  palette = NULL,
  dark_theme = FALSE
)
```

Arguments

object	An SpatialCellData object.
features	Character vector. Marker names to plot.
slot	Character. "data" or "counts".
pt_size	Numeric. Point size. Default 0.3.
ncol	Integer. Number of columns in the faceted layout. Default NULL (auto).
palette	Character or character vector. Colour palette. Default NULL uses the global palette.
dark_theme	Logical. Dark background. Default FALSE.

Value

A ggplot object.

Examples

```
counts <- matrix(rnorm(150), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8", "PanCK")))
coords <- data.frame(x = runif(50, 0, 500), y = runif(50, 0, 500))
obj <- CreateSpatialObject(counts, coords)
FeaturePlot(obj, features = c("CD3", "CD8"))
```

FieldCellsRanked *Cells of One Type inside a Field, Ranked*

Description

Selects cells with given labels inside a square of half-width half around a centre, ordered by a column (for example the distance to the nearest tumour cell), and keeps the first n; used to place arrows on a fluorescence field.

Usage

```
FieldCellsRanked(
  cells,
  centre,
  half,
  labels,
  order_by = NULL,
  n = Inf,
  x = "x",
  y = "y",
  label = "label"
)
```

Arguments

`cells`	Data frame of cells.
`centre`	`c(x, y)` or a data frame with `x`, `y`.
`half`	Half-width of the square.
`labels`	Labels of the cells to select.
`order_by`	Optional column to sort by (ascending).
`n`	Maximum number of cells.
`x`, `y`, `label`	Column names.

Value

The selected rows of `cells`.

Examples

```
d <- data.frame(x = c(1, 2, 3, 9), y = c(1, 1, 2, 9), label = c("a", "a", "b", "a"),
                dist = c(4, 2, 1, 3))
FieldCellsRanked(d, centre = c(2, 2), half = 3, labels = "a", order_by = "dist")
```

`FieldCentre` *Field Centre as Plot Data*

Description

Resolves a field centre as in `CellMapData()` (given point, cell, grid snapping, offset) and returns it as a one-row data frame, the form taken by image readers, maps and boxes.

Usage

```
FieldCentre(
  cells,
  centre = NULL,
  centre_cell = NULL,
  id = "row",
  grid = NULL,
  offset = NULL,
  size = NULL,
  x = "x",
  y = "y"
)
```

Arguments

`cells`	Data frame of cells (positions, label and further columns).
`centre`	Optional `c(x, y)` or a data frame with columns `x`, `y`.
`centre_cell`	Optional cell identifier giving the centre.
`id`	Column holding cell identifiers for `centre_cell`.
`grid`	Optional list with `step` and `offset`: the centre is moved to the nearest `min(coordinate) + offset + k * step`.
`offset`	Optional shift `c(dx, dy)` added to the centre.
`size`	Optional field width stored in a `size` column.
`x, y`	Coordinate columns of `cells`.

Value

A data frame with `x`, `y` and, when given, `size`.

Examples

```
d <- data.frame(row = 0:2, x = c(1, 5, 9), y = c(2, 6, 10), label = "a")
FieldCentre(d, centre_cell = 1, offset = c(10, 0), size = 20)
```

FigureGrammars *Figure Plot Builders Available to Figure Specifications*

Description

Names usable in the `grammar` field of a figure specification and the exported function each one calls.

Usage

```
FigureGrammars()
```

Value

Named character vector: grammar name to function name.

See Also

`RenderFigure()`

Examples

```
head(FigureGrammars())
```

FigureGrob *Place a Grob on a Figure Page*

Description

Place a Grob on a Figure Page

Usage

```
FigureGrob(grob, at, hjust = 0, vjust = 1)
```

Arguments

grob	A 'grid' grob, for example a legend from LegendKeys().
at	Numeric c(left, top) in millimetres.
hjust, vjust	Justification relative to at.

Value

A list of class phenoscapR_grob.

Examples

```
key <- FigureGrob(grid::textGrob("Legend text", just = c(0, 1)), at = c(5, 5))
fig <- ComposeFigure(list(FigurePanel(NULL, c(5, 10, 40, 20), letter = "a")),
                     height_mm = 35, width_mm = 60, texts = list(key))
grid::grid.newpage(); print(fig)
```

FigurePanel *Describe One Figure Panel*

Description

A panel places a plot so that its data area fills box exactly; axis labels, ticks and legends extend outside the box, as in hand-laid journal figures. Boxes are measured in millimetres from the top-left corner of the page.

Usage

```
FigurePanel(
  plot,
  box,
  letter = NULL,
  letter_at = NULL,
  heading = NULL,
```

```
  note = NULL,
  fill_box = FALSE,
  typeset = TRUE
)
```

Arguments

plot	A 'ggplot2' plot, a 'grid' grob, or NULL for a text-only panel.
box	Numeric c(left, top, width, height) in millimetres.
letter	Optional panel letter.
letter_at	Optional c(left, top) of the letter in millimetres; defaults to 4 mm left of and 3 mm above the box.
heading	Optional panel heading, set bold above the box.
note	Optional statistic or note line, set small and grey directly above the box (below the heading).
fill_box	If TRUE, the whole plot (including axes) is fitted into the box instead of aligning the data area.
typeset	If FALSE, heading, note and the text drawn by the plot are not typeset (see TypesetLabel()).

Value

A list of class phenoscapR_panel.

See Also

ComposeFigure()

Examples

```
FigurePanel(NULL, c(10, 10, 40, 30), letter = "a", heading = "Overview")
```

FigureText *Place Text on a Figure Page*

Description

Place Text on a Figure Page

Usage

```
FigureText(
  label,
  at,
  size = NULL,
  colour = NULL,
  bold = FALSE,
  hjust = 0,
  vjust = 1,
  angle = 0,
  typeset = TRUE
)
```

Arguments

label	Text, typeset with TypesetLabel().
at	Numeric c(left, top) in millimetres.
size	Font size in points; default is the style's note size.
colour	Text colour; default is the style's muted colour.
bold	Bold face.
hjust, vjust	Justification relative to at; by default the text hangs below and to the right of the point. vjust = "baseline" puts the baseline of the last line at at, as figure text placed without vertical alignment in common plotting code.
angle	Rotation in degrees.
typeset	If FALSE, the label is drawn as written (see TypesetLabel()).

Value

A list of class phenoscapR_text.

Examples

```
FigureText("Median and IQR\nn = 12 patients", c(20, 50))
FigureText("Treg-like cells", c(20, 60), vjust = "baseline", typeset = FALSE)
```

FindNeighbours *Find Nearest Neighbours (SpatialCellData)*

Description

For each cell, computes the distance to the k nearest neighbours.

Usage

```
FindNeighbours(object, k = 1L, target = NULL)
```

Arguments

object	An SpatialCellData-class object.
k	Integer. Number of nearest neighbours. Default 1.
target	Character or NULL. Restrict to a specific phenotype.

Value

An SpatialCellData-class with nn_distance added to meta_data and stored in the spatial slot.

Examples

```
counts <- matrix(rnorm(40), nrow = 20,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(20, 0, 100), y = runif(20, 0, 100))
obj <- CreateSpatialObject(counts, coords)
obj <- FindNeighbours(obj, k = 3)
```

FormatP *Format P Values for Figures*

Description

Two significant digits from 0.001 upwards, scientific notation below (4.9×10^{-4} after TypesetLabel()) and "n.e." (not evaluable) for missing or non-finite values. This is the C format %.2g / %.1e of common figure code.

Usage

```
FormatP(p)
```

Arguments

p	Numeric vector of P values.

Value

Character vector; values below 0.001 use the plain form "4.9e-04", which TypesetLabel() turns into a power of ten.

Examples

```
FormatP(c(0.0039, 0.055, 0.5, 4.9e-4, NA))
```

`FormatStatistic` *Format a Patient-Level Statistic Line*

Description

Builds the short statistic text placed above a panel, for example `"11/12 patients; p = 0.0015"` or `"+3.4; p = 0.021; q = 0.18"`, before typesetting.

Usage

```
FormatStatistic(
  p,
  k = NULL,
  n = NULL,
  q = NULL,
  estimate = NULL,
  digits = 1,
  unit = "patients"
)
```

Arguments

`p`	P value.
`k`, `n`	Patients in the expected direction and evaluable patients; omit both for a contrast line.
`q`	Optional adjusted P value.
`estimate`	Optional signed effect shown first, with `digits` decimals and a true minus sign.
`digits`	Decimals of `estimate`.
`unit`	Optional text after the patient count, default `"patients"`.

Value

Character vector.

Examples

```
FormatStatistic(0.0015, k = 11, n = 12)
FormatStatistic(0.021, q = 0.18, estimate = -3.42)
```

`FormatTemplate` *Fill Text Templates from Table Rows*

Description

Replaces placeholders in braces with values of `data`, so that figure text such as `"{k|d}/{n|d} patients; p = {p|p}"` is written from plot data rather than typed by hand. Placeholders:

Usage

```
FormatTemplate(template, data = NULL)
```

Arguments

`template`	Character vector of templates. A single template is applied to every row; several templates use one row each, or all the single row of a one-row table.
`data`	A data frame (or a named list treated as one row), or `NULL` for templates without placeholders.

Details

- `{column}`: the value; whole numbers with thousands separators (`1,200`), other numbers and text as they are, missing and infinite values `"n.e."`. Negative numbers are written with a true minus sign.
- `{column|format}`: a named format: `p` (`FormatP()`), `pfixed` (four decimals below 0.01, three otherwise), `d` (rounded integer), `comma` (rounded integer with thousands separators), `f0` to `f9` (fixed decimals), `sf0` to `sf9` (fixed decimals with a leading `+` or true minus sign) and `s` (the value as text).
- `{column:%fmt}`: a `sprintf()` format such as `%.1f`.
- `{column~singular~plural}`: `singular` when the value is 1, otherwise `plural`.

A placeholder whose whole content is a column name refers to that column, so column names may contain `|`, `:` or `~`.

Value

Character vector, one element per template or per row.

Examples

```
d <- data.frame(k = 11, n = 12, p = 0.00146, cells = 27081)
FormatTemplate("{k|d}/{n|d} patients; p = {p|p}", d)
FormatTemplate("{cells} cells ({cells:%.0f}); {k~patient~patients}", d)
FormatTemplate("{x|sf1} ({share|f0}%)", data.frame(x = -3.42, share = 51.2))
```

`FromCellspec` *Import a Canonical Cell Table Without Losing Spatial Identity*

Description

Converts a validated cellspecR object to a SpatialCellData object. Each image is an independent coordinate frame: output sample_id is image_id, while source_sample_id preserves the original biological sample identifier. Cell identifiers remain unchanged and are unique jointly with image_id. Coordinates remain in micrometres and area remains in square micrometres; calibration is validated but is not applied a second time.

Usage

```
FromCellspec(x, features = NULL, project = "SpatialProject")
```

Arguments

`x`	A structurally valid cellspec object. Requires cellspecR.
`features`	Unique ordered feature identifiers, or NULL for all features.
`project`	Character scalar naming the project.

Details

Feature identifiers retain their complete compartment/marker/statistic names. No marker guessing, normalisation, filtering, imputation or removal of all-NA features occurs. Functions requiring complete intensities need an explicit downstream missing-feature policy. Existing phenotype metadata is preserved; import does not verify or authorize review decisions.

The metadata attribute cellspec_import holds an original-import snapshot of dictionary, image/channel tables, input provenance, adjacency and raw feature support. Its scope remains original_import after subsetting: adjacency and support in this snapshot are not current subset results or active graphs. No reviewed tissue window or observation area is inferred from cell centroids.

The optional **cellspecR** package (https://github.com/CTTIR/cellspecR) is available from the CTTIR R-universe; install it with:

```
install.packages(
  "cellspecR",
  repos = c("https://cttir.r-universe.dev", getOption("repos"))
)
```

Value

A SpatialCellData object with unchanged selected measurements and explicit image-local sample IDs. Read provenance via attr(Meta(object), "cellspec_import").

Examples

```
if (requireNamespace("cellspecR", quietly = TRUE)) {
  object <- FromCellspec(cellspecR::cs_example())
  Coords(object)
}
```

`GetData` *Get Expression Data*

Description

Retrieve raw counts or normalised data from a SpatialCellData object.

Usage

```
GetData(object, slot = "data")

## S4 method for signature 'SpatialCellData'
GetData(object, slot = "data")
```

Arguments

`object`	A `SpatialCellData` object.
`slot`	Character. `"counts"` for raw data or `"data"` (default) for normalised data.

Value

A numeric matrix (cells x markers).

Examples

```
data(phenoscapR_example)
GetData(phenoscapR_example, slot = "counts")[1:3, ]
```

`HistogramPlot` *Histogram of Marker Expression*

Description

Histogram of Marker Expression

Usage

```
HistogramPlot(
  object,
  feature,
  group_by = NULL,
  slot = "data",
  bins = 50L,
  colours = NULL
)
```

Arguments

object	An SpatialCellData object.
feature	Character. Single marker name.
group_by	Character or NULL. Metadata column for fill grouping.
slot	Character. "data" or "counts".
bins	Integer. Number of bins. Default 50.
colours	Named character vector or NULL.

Value

A ggplot object.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50), y = runif(50))
obj <- CreateSpatialObject(counts, coords)
HistogramPlot(obj, feature = "CD3")
```

Idents *Get Active Cell Identities*

Description

Returns the phenotype labels if set, otherwise sample identities.

Usage

```
Idents(object)

## S4 method for signature 'SpatialCellData'
Idents(object)
```

Arguments

object	A SpatialCellData object.

Value

A character vector.

Examples

```
data(phenoscapR_example)
table(Idents(phenoscapR_example))
```

---

interaction_matrix *Spatial Interaction Matrix*

Description

Computes a pairwise interaction matrix between phenotypes based on observed versus expected neighbour frequencies within a given radius.

Usage

```
interaction_matrix(dt, radius)
```

Arguments

dt	A data.table with columns x, y, phenotype, and sample_id.
radius	Numeric. Radius for defining spatial neighbourhoods.

Value

A data.table in long format with columns from, to, observed, expected, and interaction_score (log2 observed/expected). Positive values indicate spatial attraction; negative values indicate avoidance.

Examples

```
set.seed(42)
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:100,
  x = runif(100, 0, 500), y = runif(100, 0, 500),
  phenotype = sample(c("CD3+", "CD8+", "Tumour"), 100, replace = TRUE)
)
interactions <- interaction_matrix(dt, radius = 50)
interactions
```

`InteractionMatrix` *Interaction Matrix (SpatialCellData)*

Description

Computes pairwise spatial interaction scores between phenotypes.

Usage

```
InteractionMatrix(
  object,
  radius,
  method = c("analytic", "permutation"),
  n_perm = 100L,
  seed = NULL
)
```

Arguments

`object`	An `SpatialCellData-class` object.
`radius`	Numeric. Radius for defining spatial neighbourhoods.
`method`	Character. `"analytic"` (default) compares observed neighbour counts to an expectation under random mixing; `"permutation"` builds the null by shuffling phenotype labels within each sample, adding `z_score` and `p_value` columns.
`n_perm`	Integer. Number of label permutations when `method = "permutation"`. Default `100`.
`seed`	Integer or `NULL`. Random seed for the permutation null.

Value

A data frame with columns `from`, `to`, `observed`, `expected`, and `interaction_score`; the permutation method adds `z_score` and `p_value`.

Examples

```
set.seed(42)
counts <- matrix(rnorm(200), nrow = 100,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(100, 0, 500), y = runif(100, 0, 500))
meta <- data.frame(cell_id = 1:100, sample_id = "s1",
  phenotype = sample(c("CD3+", "CD8+", "Tumour"), 100, replace = TRUE))
obj <- CreateSpatialObject(counts, coords, meta)
InteractionMatrix(obj, radius = 50)
```

InteractionPlot *Plot Spatial Interaction Heatmap*

Description

Heatmap of pairwise spatial interaction scores between phenotypes.

Usage

```
InteractionPlot(interactions, title = NULL)
```

Arguments

interactions	Data frame as returned by InteractionMatrix.
title	Character or NULL.

Value

A ggplot object.

Examples

```
interactions <- data.frame(
  from = rep(c("CD3+", "CD8+"), each = 2),
  to = rep(c("CD3+", "CD8+"), 2),
  observed = c(50, 30, 25, 40),
  expected = rep(35, 4),
  interaction_score = log2(c(50, 30, 25, 40) / 35)
)
InteractionPlot(interactions)
```

LeaveOnePatientOut *Exact Patient Omission Diagnostics for Declared Hypotheses*

Description

Computes full tests once through ExactPatientTests, then evaluates explicitly declared patient omissions through the same API. An incomplete full hypothesis cannot become testable by omitting a missing or ineligible patient. No clinical groups, independence, omission set or multiplicity policy is inferred.

Usage

```
LeaveOnePatientOut(
  endpoints,
  design,
  hypotheses,
  omissions,
  missingness = "require_complete",
  max_allocations = 1000000L
)
```

Arguments

endpoints, design, hypotheses	Inputs with the ExactPatientTests schemas. Selected columns must be dimensionless vectors. Extra columns are ignored.
omissions	Data frame with character hypothesis_id and patient_id. Each pair must be unique; the patient must belong to that hypothesis's contrast design. Zero rows are allowed with typed character columns.
missingness	Only "require_complete" is supported.
max_allocations	Allocation ceiling passed unchanged to full and reduced exact tests. No random approximation or changed numerical rule is used.

Value

List with full_tests (the complete exact-test result) and omissions (one row per supplied omission in its original order). Omission rows retain hypothesis keys and full results, reduced expected/supplied/evaluable counts, the omitted arm, an explicit empty_arm flag, reduced results and effect direction diagnostics. reduced_status is UNAVAILABLE_FULL_INCOMPLETE, UNAVAILABLE_EMPTY_ARM or EXACT, in that priority order. Reduced allocations, grid step, effect and p value are NA when no reduced test is performed. Supplied counts include present endpoint rows regardless of eligibility; evaluable counts require a finite value and nonmissing TRUE eligibility. Full-incomplete omissions do not execute reduced tests. Zero effects have direction ZERO; direction preservation is NA when either effect is missing. Arithmetic overflow in an effect change fails rather than returning Inf.

Examples

```
endpoints <- data.frame(endpoint_id = "e", patient_id = letters[1:5],
                        value = c(1, 2, 4, 6, 7), eligible = TRUE)
design <- data.frame(contrast_id = "c", patient_id = letters[1:5],
                     group = c("A", "A", "B", "B", "B"))
hypotheses <- data.frame(hypothesis_id = "h", endpoint_id = "e",
                         contrast_id = "c", family_id = "f")
omissions <- data.frame(hypothesis_id = "h", patient_id = letters[1:5])
out <- LeaveOnePatientOut(endpoints, design, hypotheses, omissions)
out$omissions[, c("omitted_patient", "mean_difference", "p_value")]
```

`LegendKeys` *Legend Keys as a Grob*

Description

Draws a compact legend without a frame, in one row or several columns, for placement with `FigureGrob()`. Text is measured when the legend is drawn. Keys are points, open circles, crosses, squares, diamonds, patches (filled rectangles of the key width), lines or point-and-line keys.

Usage

```
LegendKeys(
  labels,
  colours,
  shape = "point",
  ncol = length(labels),
  size = NULL,
  key_mm = 2.2,
  marker_size = NULL,
  row_mm = NULL,
  text_gap_mm = 0.8,
  column_gap_mm = 0.8,
  byrow = TRUE,
  linewidth = NULL,
  typeset = TRUE,
  style = ManuscriptStyle()
)
```

Arguments

`labels`	Character labels, typeset with `TypesetLabel()`.
`colours`	Key colours, recycled.
`shape`	One of `"point"`, `"open"` (open circle), `"cross"`, `"square"`, `"diamond"`, `"patch"`, `"line"` or `"pointline"`, recycled per key.
`ncol`	Number of columns; default is one row.
`size`	Text size in points; default is the style's legend size.
`key_mm`	Key width in millimetres.
`marker_size`	Optional marker diameter (square and diamond side) in points; by default it follows the text size.
`row_mm`	Optional row pitch in millimetres (default 1.35 times the text size).
`text_gap_mm`	Gap between key and label (mm).
`column_gap_mm`	Gap after the widest label of a column (mm).
`byrow`	Fill rows first (`TRUE`) or columns first (`FALSE`).
`linewidth`	Optional line width (points) of line keys.

typeset	If FALSE, labels are drawn as written.
style	A ManuscriptStyle() object.

Details

The layout defaults suit short legends; legends of common plotting code are reproduced with a marker diameter (marker_size), the row pitch (row_mm), the gaps around the label and column-major filling (byrow = FALSE, earlier columns take the extra rows).

Value

A 'grid' grob.

Examples

```
grid::grid.draw(LegendKeys(c("Untreated", "Responder"), c("#009E73", "#0072B2")))
grid::grid.newpage()
grid::grid.draw(LegendKeys(c("a", "b", "c"), "#7B2FBE", shape = c("point", "diamond", "patch"),
                           ncol = 2, byrow = FALSE, marker_size = 4))
```

LookupNestedRows *Look Up Values Stored on Nested Row Subsets*

Description

Per-cell results are often stored only for a subset of a subset of the rows of a sample (for example neighbourhood labels for the defined cells among the counted cells). For records that refer to rows of the full sample table, this returns the stored value, or missing where the row lies outside the subsets.

Usage

```
LookupNestedRows(
  data,
  sample = "sample",
  row = "row",
  outer,
  inner,
  values = NULL,
  samples,
  outer_name = "counted",
  inner_name = "defined_rows",
  value_name = "cn",
  missing = -1L,
  base = 0L,
  to = "value"
)
```

Arguments

`data`	Data frame of records with a sample and a row column.
`sample`, `row`	Column names; rows are counted from `base`.
`outer`	List (one entry per sample, in the order of `samples`) of logical vectors over all rows, or lists holding such a vector named `outer_name`.
`inner`	List of logical vectors over the outer rows, or lists holding one named `inner_name` together with the values `value_name`.
`values`	Optional list of value vectors over the inner rows (default taken from `inner`).
`samples`	Sample names matching the entries of the lists.
`outer_name`, `inner_name`, `value_name`	Element names inside list entries.
`missing`	Value for rows outside the subsets.
`base`	Index of the first row (0 for arrays from Python).
`to`	Name of the new column.

Value

`data` with the new column.

Examples

```
d <- data.frame(s = c("a", "a", "b"), r = c(0, 2, 1))
LookupNestedRows(d, "s", "r", outer = list(c(TRUE, FALSE, TRUE), c(TRUE, TRUE)),
                 inner = list(c(TRUE, TRUE), c(FALSE, TRUE)),
                 values = list(c(5L, 6L), 7L), samples = c("a", "b"))
```

`ManuscriptPalettes` *Manuscript Palettes*

Description

Named colour palettes used by `ManuscriptStyle()`. Names are generic cell types, groups, markers and alteration classes; a study maps its own labels onto them.

Usage

```
ManuscriptPalettes()
```

Details

- `groups`: Okabe-Ito colours for three clinical groups (`group1` to `group3`).
- `lineage`: one colour per cell lineage for legends and embeddings.
- `subsets`: highlight colours for cell maps (focus subsets on grey background cells) and the tumour interface tint.
- `coarse`: six coarse classes for whole-section maps.
- `neighbourhood`: an ordered palette for cellular neighbourhoods.
- `fluorescence`: display colours of common immunofluorescence channels.
- `alteration`: genomic alteration classes.
- `qc`: four colours for exclusion classes.
- `sequential`, `diverging`: endpoints for continuous scales.
- `clusters`: twenty colours for many unordered clusters.

Value

A named list of named character vectors.

Examples

```
ManuscriptPalettes()$groups
```

ManuscriptStyle *Manuscript Figure Style*

Description

Returns the typography, colours, line widths and page settings of a compact journal figure style: DejaVu Sans at 6 pt, 8 pt bold lower-case panel letters, 6.4 pt bold panel headings, thin grey axes without top and right lines, a 180 mm page width and 450 dpi raster output. Every plot builder and `ComposeFigure()` reads its settings from this object, so one change here restyles all figures.

Usage

```
ManuscriptStyle(...)
```

Arguments

`...`	Named values replacing top-level entries or, for list entries, individual elements (for example `size = list(base = 7)`).

Details

The family is used as named by devices that read system fonts (`cairo_pdf()`, 'ragg', Cairo PNG). For R's plain `pdf()` and `postscript()` devices an unknown family is registered with Helvetica metrics, so plots still draw there (in Helvetica).

Sizes are in points, line widths in points (converted for 'ggplot2' with `PtLinewidth()`) and page dimensions in millimetres. Palettes hold generic names; map study labels to them in study configuration, not in code.

Value

A list of class `phenoscapR_style` with elements `family`, `size`, `colour`, `line`, `page` and `palettes`.

See Also

`ThemeManuscript()`, `ManuscriptPalettes()`, `ComposeFigure()`

Examples

```
style <- ManuscriptStyle()
style$size$letter
ManuscriptStyle(size = list(base = 7))$size$base
```

MarkerHeatmap *Plot Marker Heatmap*

Description

Heatmap of mean marker intensity per phenotype. Analogous to Seurat's `DoHeatmap`.

Usage

```
MarkerHeatmap(
  object,
  markers = NULL,
  slot = "data",
  palette = NULL,
  title = NULL
)
```

Arguments

`object`	An `SpatialCellData` object with a `phenotype` column.
`markers`	Character vector or `NULL`. Markers to include. If `NULL`, all markers are shown.
`slot`	Character. Data slot to use: `"data"` (default) or `"counts"`.
`palette`	Character or character vector. Default `NULL` (global palette).
`title`	Character or `NULL`.

Value

A ggplot object.

Examples

```
counts <- matrix(c(rnorm(30, 1), rnorm(30, 0), rnorm(30, 0),
                   rnorm(30, 1)), ncol = 2,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(60), y = runif(60))
meta <- data.frame(cell_id = 1:60, sample_id = "s1",
                   phenotype = rep(c("A", "B"), each = 30))
obj <- CreateSpatialObject(counts, coords, meta)
MarkerHeatmap(obj)
```

Markers *Get Marker Names*

Description

Get Marker Names

Usage

```
Markers(object)

## S4 method for signature 'SpatialCellData'
Markers(object)
```

Arguments

object	A SpatialCellData object.

Value

Character vector.

Examples

```
data(phenoscapR_example)
Markers(phenoscapR_example)
```

Meta *Get or Set Cell Metadata*

Description

Get or Set Cell Metadata

Usage

```
Meta(object)

## S4 method for signature 'SpatialCellData'
Meta(object)
```

Arguments

object	A SpatialCellData object.

Value

Data frame of cell metadata.

Examples

```
data(phenoscapR_example)
head(Meta(phenoscapR_example))
```

MoransI *Moran's I Spatial Autocorrelation*

Description

Computes Moran's I statistic for a numeric variable.

Usage

```
MoransI(
  object,
  feature,
  radius,
  slot = "data",
  weights = c("binary", "row", "idw"),
  n_perm = 0L,
  seed = NULL,
  by_sample = FALSE
)
```

Arguments

object	An SpatialCellData-class object.
feature	Character. Name of the marker or metadata column.
radius	Numeric. Neighbourhood radius for spatial weights.
slot	Character. "data" (default) or "counts".
weights	Character. Spatial weighting within the radius: "binary" (default; 1 for every neighbour), "row" (row-standardised, each cell's weights sum to 1), or "idw" (inverse-distance, 1 / d).
n_perm	Integer. Number of label permutations for the p-value. 0 (default) uses the analytic normal approximation, which is exact only for "binary" weights; any other weighting auto-enables a permutation test.
seed	Integer or NULL. Random seed for the permutation test.
by_sample	Logical. If TRUE and the object holds several samples, the statistic is computed per sample and returned as a named list. Default FALSE; otherwise a single sample is required.

Value

A list with I, expected, variance, z_score, p_value, and method ("analytic" or "permutation").

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 100), y = runif(50, 0, 100))
obj <- CreateSpatialObject(counts, coords)
MoransI(obj, feature = "CD3", radius = 30)
MoransI(obj, feature = "CD3", radius = 30, weights = "idw", n_perm = 199)
```

NCells *Get the Number of Cells*

Description

Get the Number of Cells

Usage

```
NCells(object)

## S4 method for signature 'SpatialCellData'
NCells(object)
```

Arguments

object	A SpatialCellData object.

Value

Integer.

Examples

```
data(phenoscapR_example)
NCells(phenoscapR_example)
```

---

nearest_neighbours *Compute Nearest Neighbour Distances*

Description

For each cell, computes the distance to the nearest neighbouring cell, optionally restricted to specific phenotypes.

Usage

```
nearest_neighbours(dt, target_phenotype = NULL, k = 1L)
```

Arguments

dt	A data.table with columns x and y.
target_phenotype	Character string or NULL. If provided, distances are computed only to cells of this phenotype. Requires a phenotype column.
k	Integer. Number of nearest neighbours to consider. Default 1.

Value

The input data.table with an added nn_distance column (mean distance to the k nearest neighbours).

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:20,
  x = runif(20, 0, 100), y = runif(20, 0, 100)
)
result <- nearest_neighbours(dt, k = 3)
head(result[, .(cell_id, nn_distance)])
```

`NearestLabelDistance` *Distance to the Nearest Cell of Given Labels*

Description

Adds the Euclidean distance from every cell to the nearest cell whose label is in `labels` (exact nearest-neighbour search). Cells carrying one of these labels get `self` (0 by default) instead of the distance to another such cell.

Usage

```
NearestLabelDistance(
  cells,
  labels,
  column = "distance",
  x = "x",
  y = "y",
  label = "label",
  self = 0
)
```

Arguments

`cells`	Data frame of cells.
`labels`	Labels of the target cells.
`column`	Name of the new column.
`x`, `y`, `label`	Column names.
`self`	Value for target cells; `NULL` keeps the distance to the nearest other target cell.

Value

`cells` with the new column.

Examples

```
d <- data.frame(x = c(0, 3, 10), y = 0, label = c("t", "a", "a"))
NearestLabelDistance(d, "t", column = "dist")
```

NearestPartnerPairs *Cells Paired with Their Nearest Partner Cell*

Description

For selected cells (by identifier) finds the nearest cell with a partner label and returns both positions, their distance and the midpoint, the input of close-up fields that show a cell next to its nearest neighbour of another type.

Usage

```
NearestPartnerPairs(
  cells,
  ids,
  partner,
  id = "row",
  x = "x",
  y = "y",
  label = "label"
)
```

Arguments

cells	Data frame of cells.
ids	Identifiers (values of id) of the focus cells, in output order.
partner	Labels of the partner cells.
id, x, y, label	Column names.

Value

A data frame with id, x, y, partner_id, partner_x, partner_y, distance, mid_x and mid_y.

Examples

```
d <- data.frame(row = 0:3, x = c(0, 4, 10, 30), y = 0, label = c("f", "p", "f", "p"))
NearestPartnerPairs(d, c(0, 2), "p")
```

`NeighbourhoodEnrichment` *Neighbourhood Enrichment Analysis*

Description

Tests whether specific phenotype pairs are spatially enriched or depleted relative to a random permutation baseline.

Usage

```
NeighbourhoodEnrichment(
  object,
  radius,
  n_perm = 100L,
  seed = NULL,
  by_sample = FALSE
)
```

Arguments

`object`	An `SpatialCellData-class` object.
`radius`	Numeric. Neighbourhood radius.
`n_perm`	Integer. Number of permutations. Default `100`.
`seed`	Integer or `NULL`. Random seed.
`by_sample`	Logical. If `TRUE` and the object holds several samples, the statistic is computed per sample and returned as a named list. Default `FALSE`; otherwise a single sample is required.

Value

A data frame with columns `from`, `to`, `observed`, `mean_expected`, `z_score`, and `p_value`.

Examples

```
set.seed(1)
counts <- matrix(rnorm(200), nrow = 100,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(100, 0, 500), y = runif(100, 0, 500))
meta <- data.frame(cell_id = 1:100, sample_id = "s1",
  phenotype = sample(c("A", "B"), 100, replace = TRUE))
obj <- CreateSpatialObject(counts, coords, meta)
NeighbourhoodEnrichment(obj, radius = 50, n_perm = 10)
```

NeighbourTypeCounts *Count Neighbour Types over a Delaunay Graph with an Edge-Length Cut*

Description

Triangulates the cell centroids of one section (Delaunay), removes edges longer than `max_edge`, and counts for every cell how many of its remaining neighbours fall in each declared type group. This is the per-cell neighbourhood composition input of cellular-neighbourhood analyses.

Usage

```
NeighbourTypeCounts(
  cells,
  x = "x",
  y = "y",
  type = "type",
  levels,
  max_edge = Inf,
  engine = c("geos", "deldir"),
  duplicates = c("error", "isolate"),
  backend_contract = NULL,
  eps = 1e-09,
  index_base = 1L
)
```

Arguments

`cells`	Cell table of one section.
`x`, `y`, `type`	Column names of the coordinates and cell type.
`levels`	Named list of type groups (group name to cell types) or a character vector (each type its own group). Every cell type must belong to exactly one group. The order defines the count columns.
`max_edge`	Inclusive maximum edge length (same unit as `x`, `y`); `Inf` keeps all edges.
`engine`	`"geos"` or `"deldir"`.
`duplicates`	`"error"` or `"isolate"`.
`backend_contract`	Deprecated and ignored (see `BuildExactDelaunayEdges()`).
`eps`	Tolerance passed to the `"deldir"` engine.
`index_base`	Base of the integer `type_index` column (`1` or `0`).

Details

Edge length is computed with the C library `hypot()` (via `Mod()`) on the double coordinates and compared inclusively (`<= max_edge`), as `numpy.hypot` does. In general position the Delaunay triangulation is unique, so every correct engine gives the same edges; with four or more co-circular points engines may choose different diagonals. Duplicate coordinates are refused by default; `duplicates = "isolate"` keeps the first occurrence as the graph vertex and gives later copies no neighbours (as Qhull's coplanar-point handling does).

Engines: `"geos"` (default) uses GEOS through **sf** (`st_triangulate(..., bOnlyEdges = TRUE)`); input coordinates are mapped back to cells exactly. `"deldir"` uses `deldir::deldir()` as the exact graph builder does; its run time grows roughly quadratically with the number of cells, so it is meant for cross-checks on small sections.

Value

A data frame with one row per cell, in input order: `type_group`, `type_index`, `x`, `y`, `degree` and one integer count column per group. Attributes `edges` (kept undirected edges), `full_edges` (all Delaunay edges) and `engine`.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  set.seed(1)
  cells <- data.frame(x = runif(60, 0, 100), y = runif(60, 0, 100),
                      type = sample(c("a", "b", "c"), 60, replace = TRUE))
  out <- NeighbourTypeCounts(cells, levels = list(A = "a", BC = c("b", "c")),
                             max_edge = 20)
  head(out)
  attr(out, "edges")
}
```

`NiceBreaks` *Tick Breaks of Common Plotting Code*

Description

Tick positions as placed by the automatic tick locator of common plotting libraries (a "maximum number of intervals" locator): the smallest "nice" step from `steps` (times a power of ten) that divides the axis range into at most `nbins` intervals, with ticks restricted to the limits. For example limits 0 to 7 give 0, 2, 4, 6 rather than 0, 3, 6.

Usage

```
NiceBreaks(
  limits,
  nbins = 9,
  steps = c(1, 2, 2.5, 5, 10),
  min_n = 2,
```

```
  length_pt = NULL,
  tick_size = 5.5,
  axis = c("y", "x")
)
```

Arguments

limits	Axis limits (as drawn, including any margins).
nbins	Maximum number of intervals, or "auto".
steps	Nice step multipliers between 1 and 10.
min_n	Minimum number of ticks inside the limits.
length_pt	Axis length in points (for nbins = "auto").
tick_size	Tick label size in points (for nbins = "auto").
axis	"y" or "x" (for nbins = "auto").

Details

With nbins = "auto" the number of intervals follows the axis length as in those libraries: the axis length divided by twice (y axis) or three times (x axis) the tick label size, between 1 and 9.

Value

Numeric vector of tick positions.

Examples

```
NiceBreaks(c(0, 7))
NiceBreaks(c(-3.9, 82), nbins = "auto", length_pt = 51, tick_size = 5.5)
```

NMarkers *Get the Number of Markers*

Description

Get the Number of Markers

Usage

```
NMarkers(object)

## S4 method for signature 'SpatialCellData'
NMarkers(object)
```

Arguments

object	A SpatialCellData object.

Value

Integer.

Examples

```
data(phenoscapR_example)
NMarkers(phenoscapR_example)
```

`normalise_markers` *Normalise Marker Intensities*

Description

Normalises marker intensity columns using one of several methods. Inputs must be real numeric values without infinity. Z-scores use the sample standard deviation; z-score and min-max calculations retry in bounded power-of-two units if direct arithmetic overflows, underflows, or yields a subnormal variance whose precision is reduced. The definitions and ordinary finite calculations are unchanged.

Usage

```
normalise_markers(
  dt,
  method = c("zscore", "minmax", "quantile"),
  markers = NULL
)
```

Arguments

`dt`	A `data.table` as returned by `read_spatial()`.
`method`	Character string. Normalisation method: `"zscore"` (default), `"minmax"`, or `"quantile"`. Missing values are preserved. Quantile normalisation scales average ranks to `[0, 1]`; a sole observed value is assigned 0.5, as for a constant marker with several observations.
`markers`	Character vector or `NULL`. Marker columns to normalise. If `NULL`, all detected marker columns are normalised.

Value

A `data.table` with normalised marker intensities.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:20,
  x = runif(20), y = runif(20),
  CD3 = rnorm(20, 500, 100),
  CD8 = rnorm(20, 300, 80)
)
norm_dt <- normalise_markers(dt, method = "zscore")
head(norm_dt)
```

`NormaliseData` *Normalise Marker Intensities (SpatialCellData)*

Description

Normalises marker intensities and stores the result in the `data` slot. Raw counts remain unchanged. Z-scores use the sample standard deviation. When direct finite-input arithmetic overflows, underflows, or yields a subnormal variance, calculations retry in bounded power-of-two units with the same definitions; ordinary finite calculations are unchanged.

Usage

```
NormaliseData(
  object,
  method = c("zscore", "minmax", "quantile"),
  markers = NULL
)
```

Arguments

`object`	An `SpatialCellData-class` object.
`method`	Character. `"zscore"` (default), `"minmax"`, or `"quantile"`. Missing values are preserved. Quantile normalisation scales average ranks to `[0, 1]`; a sole observed value is assigned 0.5, as for a constant marker with several observations.
`markers`	Character vector or `NULL`. Markers to normalise. If `NULL`, all markers are normalised.

Value

An `SpatialCellData-class` with updated `data` slot.

Examples

```
counts <- matrix(rnorm(40, 500, 100), nrow = 20,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(20), y = runif(20))
obj <- CreateSpatialObject(counts, coords)
obj <- NormaliseData(obj, method = "zscore")
```

PairCorrelation *Pair Correlation Function*

Description

Computes the pair correlation function g(r), the derivative of Ripley's K, measuring clustering/inhibition at specific distances.

Usage

```
PairCorrelation(
  object,
  r_seq = NULL,
  dr = NULL,
  target = NULL,
  by_sample = FALSE
)
```

Arguments

object	An SpatialCellData-class object.
r_seq	Numeric vector of radii, or NULL for automatic.
dr	Numeric or NULL. Ring width. Default is derived from r_seq.
target	Character or NULL. Restrict to a phenotype.
by_sample	Logical. If TRUE and the object holds several samples, the statistic is computed per sample and returned as a named list. Default FALSE; otherwise a single sample is required.

Value

A data frame with columns r and g.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 200), y = runif(50, 0, 200))
obj <- CreateSpatialObject(counts, coords)
PairCorrelation(obj)
```

PairedDifferences *Build Paired Differences from Wide Subject Tables*

Description

Turns pairs of columns of one or several subject tables (one row per subject) into long paired differences `x - y`, ready for `ExactPairedTests()`.

Usage

```
PairedDifferences(data, id, pairs, name = "endpoint")
```

Arguments

`data`	A data frame, or a named list of data frames (for example several summary tables of the same subjects).
`id`	Subject column, unique within each table; not named `x`, `y` or `difference`.
`pairs`	List of pair specifications, each a list with `name` (label of the difference), `x` and `y` (column names; `y = NULL` keeps `x` unchanged) and, when `data` is a list, `table` (element name). Further scalar fields (for example `expected = "less"`) become columns.
`name`	Name of the output label column.

Value

A data frame with `name`, the subject column, `x`, `y`, `difference` and any extra fields, pairs in the given order and subjects in table order.

Examples

```
d <- data.frame(subject = c("s1", "s2", "s3"), a = c(5, 7, 9), b = c(6, 6, 10))
PairedDifferences(d, "subject", list(list(name = "a - b", x = "a", y = "b",
                                          expected = "less")))
```

PaletteContinuous *Generate a Continuous Colour Palette Function*

Description

Returns a function that maps numeric values to colours based on the current or specified palette. Works with `ggplot2` via `scale_colour_gradientn` / `scale_fill_gradientn`.

Usage

```
PaletteContinuous(n = 256L, palette = NULL)
```

Arguments

n	Integer. Number of colours to generate. Default 256.
palette	Character or character vector. Palette specification (see SetPalette). If NULL, uses the global palette.

Value

A character vector of n hex colours.

Examples

```
cols <- PaletteContinuous(10)
plot(1:10, col = cols, pch = 19, cex = 3)

cols <- PaletteContinuous(10, palette = c("blue", "white", "red"))
plot(1:10, col = cols, pch = 19, cex = 3)
```

PaletteDiscrete *Generate Discrete Colours*

Description

Generates n evenly spaced colours from the current or specified palette. Useful for colouring phenotypes, clusters, or samples.

Usage

```
PaletteDiscrete(n, palette = NULL)
```

Arguments

n	Integer. Number of distinct colours needed.
palette	Character or character vector. Palette specification (see SetPalette). If NULL, uses the global palette.

Value

A character vector of n hex colours.

Examples

```
cols <- PaletteDiscrete(5)
barplot(rep(1, 5), col = cols)

cols <- PaletteDiscrete(8, palette = c("#1b9e77", "#ffffff", "#d95f02"))
barplot(rep(1, 8), col = cols)
```

`PermutationCorrelation` *Permutation Test of a Rank or Linear Correlation*

Description

Computes Spearman's rho (Pearson correlation of midranks) or Pearson's r for the complete pairs of each group, with a two-sided permutation p value: Monte Carlo with a declared R seed, `(b + 1) / (B + 1)` where `b` counts permuted absolute correlations at least the observed one, or exact enumeration of all `n!` orderings (`mean` of the same indicator).

Usage

```
PermutationCorrelation(
  data,
  x,
  y,
  groups = NULL,
  method = c("spearman", "pearson"),
  permutations = c("monte_carlo", "exact"),
  n_perm = 99999L,
  seed = NULL,
  min_n = 4L,
  max_exact_n = 9L,
  tolerance = 1e-12
)
```

Arguments

`data`	Data frame.
`x`, `y`	Numeric columns.
`groups`	Optional group columns.
`method`	`"spearman"` or `"pearson"`.
`permutations`	`"monte_carlo"` or `"exact"`.
`n_perm`	Number of Monte Carlo permutations.
`seed`	Optional whole-number seed for Monte Carlo draws; every group starts from it. The caller's random stream is restored.
`min_n`	Smallest number of complete pairs tested (at least 2).
`max_exact_n`	Largest `n` accepted for exact enumeration.
`tolerance`	Absolute tolerance in the comparison with the observed correlation.

Value

A data frame with the group columns, `n`, `estimate`, `p_value`, `method`, `permutations`, `n_perm` (number of orderings used), `seed` and `status`.

Examples

```
d <- data.frame(x = c(1, 2, 3, 4, 5, 6), y = c(2, 1, 4, 3, 6, 5))
PermutationCorrelation(d, "x", "y", permutations = "exact")
PermutationCorrelation(d, "x", "y", n_perm = 999, seed = 1)
```

---

phenoscapR_example *Synthetic multiplexed-imaging example dataset*

Description

A small, fully synthetic two-sample SpatialCellData object used throughout the examples, tests, and vignettes. It is simulated to resemble a multiplexed immunofluorescence assay of tonsil tissue, with realistic spatial niche structure so that the spatial statistics return meaningful, non-trivial results.

Usage

```
phenoscapR_example
```

Format

A SpatialCellData object with 640 cells across 2 samples and 8 markers (`CD3`, `CD4`, `CD8`, `CD20`, `CD68`, `PanCK`, `FoxP3`, `Ki67`). The `meta_data` slot carries `cell_id`, `sample_id`, `cell_area`, and a `phenotype_true` column recording the ground-truth population for each cell. Coordinates lie within a 1000 x 1000 (arbitrary unit) imaging window.

Details

The data contain two samples (`"tonsil_A"`, `"tonsil_B"`), each built from six cell populations placed in distinct spatial niches:

- a tight **B-cell follicle**,
- an overlapping **T-cell zone** (helper, cytotoxic, and a few regulatory T cells),
- an **epithelial/tumour** region, and
- scattered **macrophages**.

Per-population marker intensities are drawn from log-normal distributions anchored on canonical lineage markers, so unsupervised phenotyping and neighbourhood-enrichment analyses recover the planted structure.

Source

Simulated by `data-raw/make_example_data.R` with `set.seed(2024)`. Entirely synthetic; not derived from any real specimen.

Examples

```
data(phenoscapR_example)
phenoscapR_example
table(phenoscapR_example$sample_id, phenoscapR_example$phenotype_true)
```

phenotype_cells *Phenotype Cells by Marker Thresholds*

Description

Assigns phenotype labels to cells based on marker intensity thresholds. A cell is considered positive for a marker if its intensity meets or exceeds the given threshold.

Usage

```
phenotype_cells(dt, thresholds, labels = NULL)
```

Arguments

dt	A data.table as returned by read_spatial() or normalise_markers().
thresholds	A named list where names are marker column names and values are numeric thresholds. Example: list(CD3 = 0.5, CD8 = 0.3).
labels	A named character vector mapping phenotype signatures to labels, or NULL for automatic labelling. When NULL, phenotypes are labelled by concatenating positive marker names with "+".

Value

The input data.table with an added phenotype column.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:6,
  x = runif(6), y = runif(6),
  CD3 = c(0.8, 0.1, 0.9, 0.2, 0.7, 0.05),
  CD8 = c(0.1, 0.6, 0.7, 0.05, 0.8, 0.02)
)
result <- phenotype_cells(dt, thresholds = list(CD3 = 0.5, CD8 = 0.5))
table(result$phenotype)
```

PhenotypeCells *Phenotype Cells (SpatialCellData)*

Description

Assigns phenotype labels to cells based on marker intensity thresholds. Uses the normalised data slot.

Usage

```
PhenotypeCells(object, thresholds, labels = NULL)
```

Arguments

object	An SpatialCellData-class object.
thresholds	Named list of thresholds (marker name = value).
labels	Named character vector mapping signatures to labels, or NULL for automatic labelling.

Value

An SpatialCellData-class with a phenotype column in meta_data.

Examples

```
counts <- matrix(c(rnorm(10, 5), rnorm(10, 1)), ncol = 2,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(10), y = runif(10))
obj <- CreateSpatialObject(counts, coords)
obj <- PhenotypeCells(obj, thresholds = list(CD3 = 3, CD8 = 3))
table(Meta(obj)$phenotype)
```

PhenotypeSummary *Phenotype Summary (SpatialCellData)*

Description

Computes phenotype counts and proportions per sample.

Usage

```
PhenotypeSummary(object)
```

Arguments

object	An SpatialCellData-class object.

Value

A data frame with columns sample_id, phenotype, count, and proportion.

Examples

```
counts <- matrix(c(rnorm(15, 5), rnorm(15, 1)), ncol = 2,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(15), y = runif(15))
obj <- CreateSpatialObject(counts, coords)
obj <- PhenotypeCells(obj, thresholds = list(CD3 = 3, CD8 = 3))
PhenotypeSummary(obj)
```

PlanExactDelaunayWorkspace

Preflight Integer Workspace Bounds for Exact Delaunay Construction

---

Description

Computes buffer lengths and routing without loading a backend, constructing a graph or allocating the planned native vectors.

Usage

```
PlanExactDelaunayWorkspace(
  n_points,
  mode = c("auto", "stock", "bounded"),
  initial_madj = 64L,
  max_retries = 32L,
  retain_tessellation = FALSE
)
```

Arguments

`n_points`	Whole scalar point count, at least three.
`mode`	`"auto"`, `"stock"` or `"bounded"` workspace route.
`initial_madj`	Whole bounded adjacency capacity, at least 20. The request is clipped to the largest safe capacity; no unsafe integer coercion occurs.
`max_retries`	Nonnegative whole adjacency-growth retry ceiling.
`retain_tessellation`	Logical scalar. `TRUE` is rejected for a bounded plan, which is edge-only and cannot retain an incomplete tessellation.

Details

All values must be dimensionless unclassed base vectors. Every native argument length is checked against signed integer capacity, with one complete spare adjacency row. The byte estimate includes native buffers only, not R objects, copies or runtime overhead, and is not an available-memory check. Successful arithmetic preflight does not qualify a large native execution.

`BuildExactDelaunayEdges()` constructs every frame with `deldir::deldir()`, which allocates the stock workspace and enlarges it itself when needed. It refuses frames for which `bounded_required` is `TRUE`; the bounded figures are arithmetic only.

Value

A list containing requested/selected route, stock and selected adjacency capacity, segment capacity, named native lengths, integer headroom, retry ceiling and `native_buffer_bytes`. `native_execution` is always `FALSE`.

See Also

`BuildExactDelaunayEdges()`

Examples

```
PlanExactDelaunayWorkspace(100)
# Pure arithmetic at the stock spare-row boundary; no large allocation:
PlanExactDelaunayWorkspace(799800)$bounded_required
```

`plot_cell_map` *Plot Cell Map*

Description

Creates a scatter plot of cell positions, coloured by phenotype or cluster.

Usage

```
plot_cell_map(
  dt,
  colour_by = "phenotype",
  colours = NULL,
  point_size = 0.5,
  title = NULL
)
```

Arguments

`dt`	A `data.table` with columns `x` and `y`.
`colour_by`	Character string. Column name to colour cells by. Default `"phenotype"`.
`colours`	Named character vector of colours, or `NULL` for automatic colours.
`point_size`	Numeric. Size of plotted points. Default `0.5`.
`title`	Character string or `NULL`. Plot title.

Value

A `ggplot` object.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:50,
  x = runif(50, 0, 500), y = runif(50, 0, 500),
  phenotype = sample(c("CD3+", "CD8+", "Negative"), 50, replace = TRUE)
)
plot_cell_map(dt)
```

`plot_density` *Plot Density Map*

Description

Creates a scatter plot with point size or colour scaled by local cell density.

Usage

```
plot_density(dt, point_size = 1, title = NULL)
```

Arguments

`dt`	A `data.table` with columns `x`, `y`, and `density` (as returned by `cell_density()`).
`point_size`	Numeric. Base point size. Default `1`.
`title`	Character string or `NULL`. Plot title.

Value

A `ggplot` object.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:50,
  x = runif(50, 0, 100), y = runif(50, 0, 100),
  density = rpois(50, 5)
)
plot_density(dt)
```

plot_heatmap *Plot Marker Heatmap*

Description

Displays a heatmap of mean marker intensities per phenotype.

Usage

```
plot_heatmap(dt, markers = NULL, title = NULL)
```

Arguments

dt	A data.table with marker columns and a phenotype column.
markers	Character vector or NULL. Markers to include. If NULL, all marker columns are used.
title	Character string or NULL. Plot title.

Value

A ggplot object.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:60,
  x = runif(60), y = runif(60),
  CD3 = c(rnorm(30, 1), rnorm(30, 0)),
  CD8 = c(rnorm(30, 0), rnorm(30, 1)),
  phenotype = rep(c("CD3+", "CD8+"), each = 30)
)
plot_heatmap(dt)
```

plot_interactions *Plot Interaction Heatmap*

Description

Visualises the spatial interaction matrix as a heatmap.

Usage

```
plot_interactions(interactions, title = NULL)
```

Arguments

interactions	A data.table as returned by interaction_matrix().
title	Character string or NULL. Plot title.

Value

A ggplot object.

Examples

```
interactions <- data.table::data.table(
  from = rep(c("CD3+", "CD8+"), each = 2),
  to = rep(c("CD3+", "CD8+"), 2),
  observed = c(50, 30, 25, 40),
  expected = c(35, 35, 35, 35),
  interaction_score = log2(c(50, 30, 25, 40) / 35)
)
plot_interactions(interactions)
```

PlotAlterationMatrix *Alteration Matrix*

Description

Tiles in a row-by-column grid, coloured by a category (for example the alteration class of a gene in a tumour, or a clinical annotation of a patient); missing combinations are drawn in a light grey. Tiles can have rounded corners; a value per row (for example the share of altered tumours) can be printed right of the matrix, and row labels can be set in italics (gene symbols).

Usage

```
PlotAlterationMatrix(
  data,
  row = "row",
  column = "column",
  value = "value",
  fill_column = NULL,
  rows = NULL,
  columns = NULL,
  row_order_by = NULL,
  column_order_by = NULL,
  row_labels = NULL,
  column_labels = NULL,
  colours = NULL,
  missing_colour = "#F0F0F0",
  tile_width = 0.8,
```

```
  tile_height = 0.72,
  radius = 0.12,
  row_size = 5.3,
  row_pad = 3.5,
  row_italic = FALSE,
  show_rows = TRUE,
  show_columns = FALSE,
  column_angle = 55,
  column_size = 5,
  column_pad = 3.5,
  column_colours = NULL,
  right_text = NULL,
  right_format = "%.0f%%",
  right_size = 4.6,
  right_colour = "muted",
  right_offset = 0.2,
  legends = NULL,
  texts = NULL,
  values = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Long data: one row per matrix cell.
`row`, `column`	Columns with the row and column keys.
`value`	Column with the category of each tile.
`fill_column`	Optional column with explicit tile colours (overrides `colours`).
`rows`, `columns`	Row order (top to bottom) and column order (left to right): character vectors or data frames (ordered by a `rank` column if present); default first appearance.
`row_order_by`, `column_order_by`	Optional numeric columns of `data` giving the row or column order.
`row_labels`, `column_labels`	Display labels: named mapping or vector.
`colours`	Named colours per category.
`missing_colour`	Colour of missing cells.
`tile_width`, `tile_height`	Tile size in column and row units.
`radius`	Corner radius in data units (0 for square tiles).
`row_size`, `row_pad`, `row_italic`	Row label size, distance and italics.
`show_rows`, `show_columns`	Draw row or column labels.
`column_angle`, `column_size`, `column_pad`	Column label rotation (degrees), size and distance from the matrix (points).

column_colours Named text colours of column labels.

right_text Optional column with one numeric value per row, printed right of the matrix with right_format.

right_format, right_size, right_colour, right_offset
Format, size, colour and position (column units beyond the last column centre).

legends Optional list of legends, each as the legend of PlotEmbedding() (shape "rect" gives patches); colours default to colours.

texts Optional texts at panel fractions (list of label, xy, size, colour, bold, hjust, vjust, angle); labels may use FormatTemplate() placeholders filled from values.

values Optional data frame for text templates.

style A ManuscriptStyle() object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(row = rep(c("g1", "g2"), each = 3), column = rep(c("s1", "s2", "s3"), 2),
                value = c("Missense", NA, "Deletion", NA, "Truncating", NA),
                share = rep(c(67, 33), each = 3))
PlotAlterationMatrix(d, colours = c(Missense = "#008300", Truncating = "#222222",
                                    Deletion = "#2a78d6"), right_text = "share",
                     row_italic = TRUE)
```

PlotAnnotatedHeatmap *Annotated Heatmap*

Description

Draws a matrix of coloured cells, optionally with the values printed in the cells, coloured annotation boxes left of the rows, coloured or boxed column labels and a colour bar. Colours follow a 256-step ramp between the anchors of a named scale ("blues", "rdbu_r", "rdbu", "greys") or of given colours, as image heatmaps of common figure code; values can be coloured on a log10(x + 1) scale while the printed numbers stay raw.

Usage

```
PlotAnnotatedHeatmap(
  data,
  row = "row",
  column = "column",
  value = "value",
  rows = NULL,
  columns = NULL,
```

```
  row_order_by = NULL,
  column_order_by = NULL,
  row_labels = NULL,
  column_labels = NULL,
  scale = "blues",
  limits = NULL,
  transform = c("identity", "log10p1"),
  fill_column = NULL,
  missing_colour = "#EEEEEE",
  drop_columns_below = NULL,
  cell_text = NULL,
  row_colours = NULL,
  row_box = c(-1.05, 0.4, 0.7),
  column_colours = NULL,
  column_fills = NULL,
  column_fill_alpha = 0.55,
  column_angle = 0,
  column_size = 5.1,
  column_pad = 3.5,
  row_size = 5.3,
  row_pad = 3.5,
  row_italic = FALSE,
  show_rows = TRUE,
  show_columns = TRUE,
  x_label = NULL,
  x_label_size = 5,
  x_label_pad = 2,
  colourbar = NULL,
  texts = NULL,
  values = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Long data: one row per matrix cell.
`row`, `column`, `value`	Column names.
`rows`, `columns`	Row order (top to bottom) and column order (left to right): character vectors or data frames (ordered by a `rank` column if present); default first appearance.
`row_order_by`, `column_order_by`	Optional numeric columns of `data` giving the row or column order.
`row_labels`, `column_labels`	Display labels: named mapping or vector.
`scale`	Colour scale name or vector of anchor colours.
`limits`	Colour limits (on the transformed scale); default the range.
`transform`	`"identity"` or `"log10p1"` (colour by `log10(x + 1)`).

`fill_column` Optional column with explicit cell colours (overrides `scale`).

`missing_colour` Colour of missing cells.

`drop_columns_below`
Drop columns whose largest value is below this.

`cell_text` Optional list printing values in cells: `min` (smallest value printed), `format` (`"fixed"` with `digits`, or `"count"`: thousands separator and `"12k"` from `abbreviate` upwards), `white_above` (raw values above it are printed in `light_colour`), `size` (points).

`row_colours` Named colours of row annotation boxes.

`row_box` Box geometry `c(left, width, height)` in column units, left measured from the centre of the first column.

`column_colours` Named text colours of column labels.

`column_fills` Named colours of rounded boxes behind column labels.

`column_fill_alpha`
Opacity of these boxes.

`column_angle`, `column_size`, `column_pad`
Column label rotation (degrees), size and distance from the matrix (points).

`row_size`, `row_pad`, `row_italic`
Row label size, distance and italics.

`show_rows`, `show_columns`
Draw row or column labels.

`x_label`, `x_label_size`, `x_label_pad`
Title below the column labels.

`colourbar` Optional list: `box` (`c(left, bottom, width, height)` in panel fractions), `title`, `breaks`, `labels`, `tick_size`, `title_size`.

`texts` Optional texts at panel fractions (list of `label`, `xy`, `size`, `colour`, `bold`, `hjust`, `vjust`, `angle`); labels may use `FormatTemplate()` placeholders filled from `values`.

`values` Optional data frame for text templates.

`style` A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- expand.grid(row = c("a", "b"), column = c("x", "y", "z"), stringsAsFactors = FALSE)
d$value <- c(85, 3, 9, 15, 2, 67)
PlotAnnotatedHeatmap(d, limits = c(0, 80), cell_text = list(min = 1, white_above = 45),
                     row_colours = c(a = "#e34948", b = "#eda100"))
```

PlotBandProfiles *Profiles over Ordered Categories*

Description

Two forms for values over ordered categories such as distance bands. Without `lower` and `upper`: one thin line per unit (patient) coloured by its group, a thick summary line of the medians, and values of `-Inf` (for example the log ratio of an empty band) drawn as open triangles at `floor`. With `lower` and `upper`: one summary line per series with vertical interval segments, series side by side within each category.

Usage

```
PlotBandProfiles(
  data,
  x = "band",
  y = "value",
  series = "patient",
  group = "group",
  x_order = NULL,
  x_labels = NULL,
  group_colours = NULL,
  floor = NULL,
  floor_label = NULL,
  floor_size = 4.6,
  linewidth = 0.6,
  line_alpha = 0.7,
  marker_size = 1.8,
  summary = TRUE,
  summary_colour = "ink",
  summary_linewidth = 1.6,
  summary_marker = 3,
  lower = NULL,
  upper = NULL,
  series_order = NULL,
  colours = NULL,
  dodge = 0.12,
  interval_linewidth = 1,
  interval_alpha = 0.6,
  reference_y = NULL,
  y_limits = NULL,
  y_breaks = NULL,
  y_nbins = 9,
  x_label = NULL,
  y_label = NULL,
  tick_size = NULL,
  box_mm = NULL,
```

```
  style = ManuscriptStyle()
)
```

Arguments

`data`	Tidy data: one row per unit (or series) and category.
`x`	Category column.
`y`	Value column.
`series`	Unit (or series) column.
`group`	Group column of the units (profile form).
`x_order`	Category order; `x_labels` their tick labels.
`x_labels`	Tick labels.
`group_colours`	Named colours per group.
`floor`	Value at which `-Inf` is drawn; `floor_label` is written to the right of the last category at that height.
`floor_label`, `floor_size`	Text and size of the floor note.
`linewidth`, `line_alpha`, `marker_size`	Unit lines (points; marker diameter as for line markers with a 1 pt edge).
`summary`	Draw the median line (profile form).
`summary_colour`, `summary_linewidth`, `summary_marker`	Median line style.
`lower`, `upper`	Interval columns (summary form).
`series_order`, `colours`	Series order and named colours (summary form).
`dodge`	Offset between series within a category.
`interval_linewidth`, `interval_alpha`	Interval segment style.
`reference_y`	Optional horizontal reference line.
`y_limits`, `y_breaks`, `y_nbins`	y axis (see `PlotQuantileCurves()`).
`x_label`, `y_label`	Axis titles.
`tick_size`	Tick label size.
`box_mm`	Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).
`style`	A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(patient = rep(c("p1", "p2", "p3"), each = 3),
                group = rep(c("A", "B", "A"), each = 3),
                band = rep(c("near", "mid", "far"), 3),
                value = c(1, 0, -1, 2, -Inf, -2, 0.5, 0.2, -0.5))
PlotBandProfiles(d, x_order = c("near", "mid", "far"), floor = -3,
                 group_colours = c(A = "#009E73", B = "#0072B2"), floor_label = "none")
```

PlotBars *Bars per Unit*

Description

Vertical or horizontal bars per unit (patient), single, side by side or stacked by a series column, with tick labels coloured by group, an optional reference line with a label from summary data, text at the end of each bar and a corner note.

Usage

```
PlotBars(
  data,
  x = "x",
  y = "value",
  series = NULL,
  x_order = NULL,
  x_labels = NULL,
  fills = "#D9D9D9",
  series_order = NULL,
  position = c("identity", "dodge", "stack"),
  width = 0.74,
  edge_colour = NA,
  edge_linewidth = 0,
  horizontal = FALSE,
  log_y = FALSE,
  tick_group = NULL,
  group_colours = NULL,
  tick_angle = 0,
  tick_size = NULL,
  stats = NULL,
  reference = NULL,
  reference_label = NULL,
  reference_label_dy = 1.5,
  reference_label_size = 4.6,
  bar_label = NULL,
  bar_label_d = 2,
  bar_label_scale = 0,
  bar_label_size = 4.4,
```

```
  bar_label_colour = "muted",
  note = NULL,
  note_xy = c(0.02, 0.97),
  note_size = 4.8,
  note_hjust = 0,
  note_vjust = 1,
  value_limits = NULL,
  value_breaks = NULL,
  value_labels = NULL,
  value_expand = NULL,
  value_nbins = 9,
  x_label = NULL,
  y_label = NULL,
  title_size = 5.6,
  grid = TRUE,
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Tidy data: one row per unit (and series).
`x`	Unit column; `y` the bar value.
`y`	Value column.
`series`	Optional series column (bars side by side or stacked).
`x_order`	Unit order (left to right, or top to bottom when horizontal).
`x_labels`	Tick labels.
`fills`	Colour (one, or named per series).
`series_order`	Series order (stacking from the bottom; side by side from the left).
`position`	`"identity"`, `"dodge"` or `"stack"`.
`width`	Total bar width per unit.
`edge_colour, edge_linewidth`	Bar outline.
`horizontal`	Horizontal bars.
`log_y`	Logarithmic value axis.
`tick_group`	Optional group column colouring the unit tick labels.
`group_colours`	Named group colours.
`tick_angle`	Rotation of unit tick labels.
`tick_size`	Unit tick label size.
`stats`	Optional one-row data frame for the reference line and the note.
`reference`	Column of `stats` giving a reference value line.
`reference_label`	Template written above the right end of the line.

`reference_label_dy`
: Its offset in value units.

`reference_label_size`
: Its size.

`bar_label`
: Optional template written at the end of each bar.

`bar_label_d`
: Offset in value units.

`bar_label_scale`
: Further offset as a fraction of the bar value.

`bar_label_size`
: Size.

`bar_label_colour`
: Colour of the bar labels.

`note`
: Optional template (first row of `stats`) at `note_xy`.

`note_xy`, `note_size`, `note_hjust`, `note_vjust`
: Note placement.

`value_limits`, `value_breaks`, `value_nbins`
: Value axis.

`value_labels`
: Optional tick labels for `value_breaks`.

`value_expand`
: Optional upper value limit as a multiple of the largest bar (the lower limit is then zero).

`x_label`, `y_label`
: Axis titles (x is the unit axis).

`title_size`
: Axis title size.

`grid`
: Grid lines along the value axis.

`box_mm`
: Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).

`style`
: A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(id = c("p1", "p2", "p3"), v = c(20, 55, 70), g = c("A", "A", "B"))
PlotBars(d, x = "id", y = "v", tick_group = "g", group_colours = c(A = "#009E73", B = "#0072B2"),
         stats = data.frame(m = 48), reference = "m", reference_label = "Mean, {m|f0}%")
```

PlotCellMap *Cell Map*

Description

Draws cells as points in layers (by label or any column condition) on an optional interface-zone tint, with frame, scale bar, field label, title above the map and boxes. Point sizes follow the area convention of common scatter plots (`size` is the marker area in square points); larger markers get a thin edge. Without a field size the whole section is drawn and the scale bar sits below the map.

Usage

```
PlotCellMap(
  data,
  layers = list(),
  rest = list(colour = "#E3E3E3", size = 1.2),
  exclusive = TRUE,
  point_extra = 0,
  title_align = "left",
  interface_colour = "subsets/interface",
  limits = NULL,
  expand = 0,
  scale_bar = NULL,
  frame = TRUE,
  frame_colour = "frame",
  frame_width = NULL,
  label = NULL,
  label_colour = "white",
  label_size = 5.5,
  title = NULL,
  subtitle = NULL,
  title_colour = "ink",
  title_size = 5.6,
  labels = NULL,
  groups = NULL,
  label_id = NULL,
  labels_key = "patient",
  title_column = "patient",
  subtitle_column = "text",
  group_column = "group",
  group_colours = NULL,
  boxes = NULL,
  box_colour = "ink",
  box_width = 0.5,
  box_size = NULL,
  style = ManuscriptStyle()
)
```

Arguments

data	A CellMapData() result or a data frame of cells.
layers	List of layers (see Details).
rest	Layer for all remaining cells, or NULL.
exclusive	If TRUE, a cell is drawn by the first layer that selects it only; with FALSE every matching layer draws it.
point_extra	Points added to every marker diameter: sub-pixel markers of some renderers look denser than their nominal size (about 0.055 matches 'matplotlib' output at 450 dpi).
title_align	"left" (title at the left edge, subtitle after it) or "centre" (title centred above the map).
interface_colour	Colour of the interface zone.
limits	Optional c(xmin, xmax, ymin, ymax); default the field or the data range.
expand	Fraction of the data range added on each side when no field size is given (0.05 imitates automatic axis margins).
scale_bar	NULL or a list: length, label, x (start, as a fraction of the map width from the left), y (distance of the bar from the lower map edge as a fraction of the width; negative values place it below the map), label_offset (signed distance of the label from the bar, as a fraction of the width; negative values hang it below), linewidth (points), size (points), colour and relative_to ("limits", the default, or "data": fractions of the data extent).
frame	Draw a frame around the map.
frame_colour, frame_width	Frame colour and width (points).
label	Optional text in the top-left corner of the map.
label_colour, label_size	Label colour and size.
title, subtitle	Optional title (bold) and subtitle (muted) above the map; taken from labels when label_id is given.
title_colour	Title colour.
title_size	Title size (points); the subtitle is 1 point smaller.
labels	Optional data frame of titles, with label_id selecting a row by labels_key.
groups	Optional data frame giving the group of label_id (column labels_key and group_column); default labels.
label_id, labels_key	Row selection in labels.
title_column, subtitle_column, group_column	Columns of labels.
group_colours	Named colours by group_column value for the title.
boxes	Optional data frame or list of squares to outline (x, y, size), drawn with box_colour and box_width.

box_colour, box_width
: Box colour and width (points).

box_size
: Box width used when boxes has no size column.

style
: A ManuscriptStyle() object.

Details

A layer is a list with column (default "label") and values, or filter (named list of column to values, all must match); colour (one colour, or a named list of value to colour for the layer's column); size; optional edge_colour (default white), edge_width (default 0.25 points for sizes above 4, else 0) and z (drawing order). Cells in no layer are drawn with rest (set rest = NULL to omit them).

Value

A 'grid' grob.

Examples

```
set.seed(1)
d <- data.frame(x = runif(300, 0, 100), y = runif(300, 0, 100),
                label = sample(c("t", "a", "b"), 300, TRUE))
m <- CellMapData(d, centre = c(50, 50), size = 60,
                 interface = list(labels = "t", radius = 5, step = 2))
g <- PlotCellMap(m, layers = list(list(values = "t", colour = "#A6A6A6", size = 2.5),
                                  list(values = "b", colour = "#7B2FBE", size = 16)),
                 scale_bar = list(length = 10, label = "10 um"))
grid::grid.newpage(); grid::grid.draw(g)
```

PlotDataBindRows *Bind Plot Data Tables*

Description

PlotDataBindRows() stacks tables by column name, filling columns missing from some tables with NA. It can record the source of each row (names_to, after the table columns) and add constant key columns per table (keys, before the table columns). PlotDataBindColumns() places one-row (or equally long) tables side by side.

Usage

```
PlotDataBindRows(
  tables,
  names_to = NULL,
  labels = NULL,
  keys = NULL,
  columns = NULL,
  rename = NULL,
```

```
  element = NULL
)

PlotDataBindColumns(tables)
```

Arguments

tables	A list of data frames (named when names_to without labels, keys or rename are used), or a single data frame.
names_to	Optional column recording the source of each row: the list name, the matching entry of labels, or the list position.
labels	Optional source labels, one per table.
keys	Optional named list: table name to a named list of scalar values added as leading columns. For a single data frame, keys is the named list of values itself.
columns	Optional columns kept from every table.
rename	Optional named list of renamings per table: table name to a named vector (new name = old name).
element	Optional element name when the list items are lists that hold the tables.

Value

A data frame.

See Also

Other plot-data builders: PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataBindRows(list(a = data.frame(x = 1), b = data.frame(x = 2, y = 3)), names_to = "src")
PlotDataBindRows(list(a = data.frame(v = 1), b = data.frame(v = 2:3)),
                 keys = list(a = list(batch = "one"), b = list(batch = "two")))
PlotDataBindColumns(list(data.frame(n = 9), data.frame(text = "p1, 1")))
```

PlotDataConstant *Add a Constant Column*

Description

Add a Constant Column

Usage

```
PlotDataConstant(data, column, value)
```

Arguments

data	A data frame.
column	Name of the new column.
value	Value written on every row.

Value

data with the column.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataConstant(data.frame(x = 1:2), "series", "a")
```

PlotDataContrastSummary

Summary of a Family of Two-Group Tests

Description

One row describing a set of exact two-group tests: the number of tests with a finite P value, of nominal (p < alpha) and of adjusted (q < alpha) results, the group sizes and ExactPermutationBound().

Usage

```
PlotDataContrastSummary(
  data,
  p = "p",
  q = "q",
  n_a = "n_a",
  n_b = "n_b",
  alpha = 0.05,
  label = NULL
)
```

Arguments

data	Test table, one row per endpoint.
p, q	P value and adjusted P value columns.
n_a, n_b	Group-size columns (the most frequent sizes are used).
alpha	Significance level.
label	Optional column whose distinct values are counted as endpoints.

Value

A one-row data frame with tests, nominal, adjusted, n_a, n_b, p_bound and endpoints.

Examples

```
t <- data.frame(p = c(0.01, 0.2, NA), q = c(0.1, 0.4, NA), n_a = 4, n_b = 2)
PlotDataContrastSummary(t)
```

PlotDataCount *Counts and Shares by Group*

Description

Counts rows per group (or overall), optionally the number of distinct values of a column and the rows per distinct value, the share of all counted rows, and counts of selected values of further columns.

Usage

```
PlotDataCount(
  data,
  by = NULL,
  filter = NULL,
  distinct = NULL,
  tally = NULL,
  scale = 100
)
```

Arguments

data	A data frame.
by	Grouping columns (none for one overall row).
filter	Optional named list of columns and values selecting rows.
distinct	Optional column whose distinct values are counted.
tally	Optional named list of extra counts: each a list of column and values.
scale	Multiplier of the share (100 for percentages).

Value

A data frame with the groups, `n`, `share`, and if requested `n_distinct`, `n_per_distinct` (integer division) and the tallies.

See Also

Other plot-data builders: `PlotDataBindRows()`, `PlotDataConstant()`, `PlotDataCrossShares()`, `PlotDataDirection()`, `PlotDataDominant()`, `PlotDataFromArrays()`, `PlotDataJoin()`, `PlotDataKeyColours()`, `PlotDataLabels()`, `PlotDataLonger()`, `PlotDataMembership()`, `PlotDataPairRatio()`, `PlotDataProduct()`, `PlotDataQuantiles()`, `PlotDataRankRows()`, `PlotDataRatio()`, `PlotDataRecode()`, `PlotDataRelative()`, `PlotDataSelect()`, `PlotDataShares()`, `PlotDataSummarise()`, `PlotDataUnite()`

Examples

```
d <- data.frame(g = c("a", "a", "b"), p = c("x", "y", "x"), t = c("T", "U", "T"))
PlotDataCount(d, "g", distinct = "p", tally = list(n_t = list(column = "t", values = "T")))
```

PlotDataCrossShares *Shares of One Category within Another*

Description

Cross-tabulates two columns and returns every combination with its count, the total of its row category and the share of the row total (for example the cell-type composition of each cluster).

Usage

```
PlotDataCrossShares(
  data,
  row,
  column,
  filter = NULL,
  rows = NULL,
  columns = NULL,
  scale = 100
)
```

Arguments

`data`	A data frame.
`row`, `column`	Column names.
`filter`	Optional named list of columns and values selecting rows.
`rows`, `columns`	Optional level sets (and order); default the sorted values present.
`scale`	Multiplier of the share (100 for percentages).

Value

A data frame with `row`, `column` (named as the inputs), `n`, `total` and `share`.

See Also

Other plot-data builders: `PlotDataBindRows()`, `PlotDataConstant()`, `PlotDataCount()`, `PlotDataDirection()`, `PlotDataDominant()`, `PlotDataFromArrays()`, `PlotDataJoin()`, `PlotDataKeyColours()`, `PlotDataLabels()`, `PlotDataLonger()`, `PlotDataMembership()`, `PlotDataPairRatio()`, `PlotDataProduct()`, `PlotDataQuantiles()`, `PlotDataRankRows()`, `PlotDataRatio()`, `PlotDataRecode()`, `PlotDataRelative()`, `PlotDataSelect()`, `PlotDataShares()`, `PlotDataSummarise()`, `PlotDataUnite()`

Examples

```
d <- data.frame(cl = c(1, 1, 2, 2, 2), type = c("a", "b", "a", "a", "c"))
PlotDataCrossShares(d, "cl", "type")
```

PlotDataDirection *Count of Units in the Direction of an Estimate*

Description

Adds `k`, the number of units (patients) whose difference has the sign of the summary estimate: `up` when the estimate is positive or zero and `n - up` otherwise, as reported in "k/n patients".

Usage

```
PlotDataDirection(data, n, up, estimate, into = "k")
```

Arguments

`data`	A data frame.
`n`	Column with the number of units.
`up`	Column with the number of positive differences.
`estimate`	Column with the summary estimate.
`into`	Name of the new column.

Value

`data` with the count column.

See Also

Other plot-data builders: `PlotDataBindRows()`, `PlotDataConstant()`, `PlotDataCount()`, `PlotDataCrossShares()`, `PlotDataDominant()`, `PlotDataFromArrays()`, `PlotDataJoin()`, `PlotDataKeyColours()`, `PlotDataLabels()`, `PlotDataLonger()`, `PlotDataMembership()`, `PlotDataPairRatio()`, `PlotDataProduct()`, `PlotDataQuantiles()`, `PlotDataRankRows()`, `PlotDataRatio()`, `PlotDataRecode()`, `PlotDataRelative()`, `PlotDataSelect()`, `PlotDataShares()`, `PlotDataSummarise()`, `PlotDataUnite()`

Examples

```
PlotDataDirection(data.frame(n = c(9, 9), up = c(9, 1), est = c(1.1, -0.4)), "n", "up", "est")
```

`PlotDataDominant` *Order Groups by Their Dominant Category*

Description

For each row group of a share table (from `PlotDataCrossShares()`), finds the category with the largest share (ties go to the earlier category in `columns`) and orders the groups by the position of that category in `priority`, then by the group value. Categories missing from `priority` take position zero, before all listed ones. A group without any share among `columns` has a missing `dominant` category and also takes position zero.

Usage

```
PlotDataDominant(
  data,
  row,
  column,
  value = "share",
  columns = NULL,
  priority = NULL
)
```

Arguments

`data`	Long share table.
`row`, `column`, `value`	Column names.
`columns`	Category order used to break ties.
`priority`	Category order defining the group order.

Value

A data frame with the group, `dominant`, `dominant_share` and `rank`, sorted by rank.

See Also

Other plot-data builders: `PlotDataBindRows()`, `PlotDataConstant()`, `PlotDataCount()`, `PlotDataCrossShares()`, `PlotDataDirection()`, `PlotDataFromArrays()`, `PlotDataJoin()`, `PlotDataKeyColours()`, `PlotDataLabels()`, `PlotDataLonger()`, `PlotDataMembership()`, `PlotDataPairRatio()`, `PlotDataProduct()`, `PlotDataQuantiles()`, `PlotDataRankRows()`, `PlotDataRatio()`, `PlotDataRecode()`, `PlotDataRelative()`, `PlotDataSelect()`, `PlotDataShares()`, `PlotDataSummarise()`, `PlotDataUnite()`

Examples

```
s <- PlotDataCrossShares(data.frame(cl = c(1, 1, 2, 3), type = c("b", "b", "a", "b")), "cl", "type")
PlotDataDominant(s, "cl", "type", "share", columns = c("a", "b"), priority = c("a", "b"))
```

PlotDataFromArrays *Table from a List of Arrays*

Description

Turns a named list of equally long arrays, for example the arrays of a NumPy `.npz` archive, into a data frame. Given a file path instead of a list, only the arrays needed are read with `cellspecR::cs_read_npz()` (pure R). One-dimensional arrays become columns, the columns of matrices become further columns, arrays defined only for some rows (`masked`) are expanded with `NA`, and rows can be restricted to a logical array.

Usage

```
PlotDataFromArrays(
  arrays,
  columns = NULL,
  matrices = NULL,
  masked = NULL,
  keep = NULL,
  row_id = NULL,
  row_base = 0L,
  row_first = FALSE
)
```

Arguments

`arrays`	Named list of vectors and matrices with one row per record, or the path of an `.npz` archive.
`columns`	Names of one-dimensional arrays to keep; a named vector or list renames them (new name = array name). Default all vectors of the record length.
`matrices`	Named list: for each matrix, either the new column names or `list(names_from = "<array>", prefix = "")` taking the names from a character array.
`masked`	Named list: output column (as named in `columns`) to the name of the logical array marking the rows its values belong to.
`keep`	Optional name of a logical array; only rows where it is `TRUE` are returned.
`row_id`	Optional name of a column holding the position of each row in the arrays (before `keep`).
`row_base`	Position of the first row (0 or 1).
`row_first`	Put the `row_id` column first instead of last.

Value

A data frame.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
a <- list(xy = matrix(1:6, 3), label = c("a", "b", "a"), names = c("u", "v"),
          feat = matrix(c(0.1, 0.2, 0.3, 1, 2, 3), 3), kept = c(TRUE, FALSE, TRUE),
          cls = c(7L, 8L), has_cls = c(TRUE, FALSE, TRUE))
PlotDataFromArrays(a, columns = "label",
                   matrices = list(xy = c("x", "y"), feat = list(names_from = "names")))
PlotDataFromArrays(a, c(type = "label", cls = "cls"), masked = list(cls = "has_cls"),
                   keep = "kept", row_id = "row")
```

PlotDataJoin *Join Columns of Another Table*

Description

Left join that keeps the rows and row order of `data` and adds selected columns of `other`, matched on key columns. Keys of `other` must be unique.

Usage

```
PlotDataJoin(data, other, by, by_other = by, columns = NULL, rename = NULL)
```

Arguments

`data`	A data frame.
`other`	A data frame.
`by`	Key columns of `data`.
`by_other`	Key columns of `other`; default `by`.
`columns`	Columns of `other` to add; default all non-key columns.
`rename`	Optional new names for `columns`.

Value

`data` with the added columns (missing matches are `NA`).

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
a <- data.frame(id = c("p2", "p1", "p3"), v = 1:3)
b <- data.frame(key = c("p1", "p2"), group = c("A", "B"))
PlotDataJoin(a, b, by = "id", by_other = "key", columns = "group")
```

PlotDataKeyColours *Key Colours from a Categorical Palette*

Description

Assigns colours to integer keys (for example cluster numbers) by cycling through one or more categorical palettes: key `k` takes colour `((k - first) mod n) + 1` of the joined palettes.

Usage

```
PlotDataKeyColours(
  keys,
  column = "key",
  palettes = c("tab20", "tab20b"),
  first = 1
)
```

Arguments

`keys`	Keys: a vector, or a data frame whose `column` holds them.
`column`	Key column when `keys` is a data frame.
`palettes`	Palette names: `"tab20"`, `"tab20b"`, `"tab20c"` or `"clusters"` (the style's cluster palette), joined in order.
`first`	Key that takes the first colour.

Value

A data frame with `key` and `colour`, sorted by key.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataKeyColours(1:25)
```

`PlotDataLabels` *Text Labels from Table Rows*

Description

`PlotDataLabels()` writes one text per row with `FormatTemplate()` (for example titles such as `"{cells:%.0f}k cells; {area:%.0f} mm2"`), after optional per-column scaling. `PlotDataCollapse()` writes every row and joins the results into one text, for example to list patients with their cell counts in a figure note.

Usage

```
PlotDataLabels(data, template, id = NULL, scales = NULL, into = "text")

PlotDataCollapse(
  data,
  template,
  sep = "; ",
  order_by = NULL,
  decreasing = FALSE,
  into = "text",
  n_name = "rows"
)
```

Arguments

`data`	A data frame.
`template`	Template (see `FormatTemplate()`).
`id`	Optional columns of `data` kept in the output of `PlotDataLabels()`.
`scales`	Named list: column to a multiplier applied before formatting.
`into`	Name of the text column.
`sep`	Separator of the collapsed rows.
`order_by`	Optional column to sort by before collapsing (stable, increasing).
`decreasing`	Sort decreasing.
`n_name`	Name of a column with the number of rows collapsed.

Value

`PlotDataLabels()`: a data frame with the `id` columns and `into`; `PlotDataCollapse()`: a one-row data frame.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
d <- data.frame(p = c("p3", "p1"), n = c(27081, 1450), g = c("A", "B"))
PlotDataLabels(d, "{n:%.0f}k cells, {g}", id = "p", scales = list(n = 0.001))
PlotDataCollapse(d, "{p}, {n|comma}", order_by = "n")
```

PlotDataLonger *Reshape Wide Columns into Long Plot Data*

Description

Stacks value columns of a summary table (one row per patient, say) into one row per identifier and column, the form used by the plot builders. The columns are given by name (`values`), by a regular expression (`pattern`), or default to all columns other than `id` and `exclude`.

Usage

```
PlotDataLonger(
  data,
  id,
  values = NULL,
  conditions = NULL,
  pattern = NULL,
  exclude = NULL,
  names_to = "condition",
  values_to = "value",
  drop_na = FALSE
)
```

Arguments

`data`	A data frame.
`id`	Identifier columns kept on every row.
`values`	Value columns to stack, in condition order.
`conditions`	Condition names for the stacked columns; default the column names (or the pattern groups).
`pattern`	Regular expression selecting the value columns when `values` is `NULL`.
`exclude`	Columns never stacked when `values` is `NULL`.
`names_to, values_to`	Names of the new condition and value columns.
`drop_na`	Drop rows whose value is missing.

Details

With `pattern`, the first capture group of the pattern becomes the condition name (`"^log2\\|(.*)$"` turns `log2|p1` into `p1`); a pattern without a group keeps the column names.

Value

A data frame with `id` columns, `names_to` (character, in column order) and `values_to` (numeric).

See Also

Other plot-data builders: `PlotDataBindRows()`, `PlotDataConstant()`, `PlotDataCount()`, `PlotDataCrossShares()`, `PlotDataDirection()`, `PlotDataDominant()`, `PlotDataFromArrays()`, `PlotDataJoin()`, `PlotDataKeyColours()`, `PlotDataLabels()`, `PlotDataMembership()`, `PlotDataPairRatio()`, `PlotDataProduct()`, `PlotDataQuantiles()`, `PlotDataRankRows()`, `PlotDataRatio()`, `PlotDataRecode()`, `PlotDataRelative()`, `PlotDataSelect()`, `PlotDataShares()`, `PlotDataSummarise()`, `PlotDataUnite()`

Examples

```
d <- data.frame(patient = c("p1", "p2"), a = 1:2, b = 3:4)
PlotDataLonger(d, "patient", c("a", "b"), conditions = c("A", "B"))
w <- data.frame(type = c("a", "b"), `v|p1` = 1:2, `v|p2` = c(3, NA), check.names = FALSE)
PlotDataLonger(w, "type", pattern = "^v\\|(.*)$", names_to = "patient", drop_na = TRUE)
```

PlotDataMembership *Flag Rows Whose Key Occurs in Another Table*

Description

Flag Rows Whose Key Occurs in Another Table

Usage

```
PlotDataMembership(data, reference, by, into = "member", negate = FALSE)
```

Arguments

`data`	A data frame.
`reference`	A data frame.
`by`	Key columns (same names in both tables).
`into`	Name of the new logical column.
`negate`	If `TRUE`, flag rows whose key does not occur.

Value

`data` with the logical column.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataMembership(data.frame(id = c("a", "b")), data.frame(id = "a"), "id",
                   into = "excluded", negate = TRUE)
```

PlotDataPairRatio *Ratios Between Two Levels of a Key Column*

Description

For tables with one row per identifier and level (for example per patient and cell set), divides the values of one level by those of a reference level of the same identifier, for one or several value columns, and returns one row per identifier and value column.

Usage

```
PlotDataPairRatio(
  data,
  id,
  key,
  numerator,
  denominator,
  values,
  names = values,
  pseudocount = 0,
  log_base = NULL,
  names_to = "variable",
  values_to = "ratio"
)
```

Arguments

data	A data frame.
id	Identifier columns that pair rows.
key	Column holding the levels.
numerator, denominator	Levels of key.
values	Value columns.
names	Names for values in the output; default values.

pseudocount	Added to both values.
log_base	Optional logarithm base.
names_to, values_to	Names of the output columns.

Value

A data frame with id, names_to, values_to and the two input values (numerator_value, denominator_value). Identifiers lacking a level give NA.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
d <- data.frame(id = rep(c("p1", "p2"), each = 2), set = c("x", "ref", "x", "ref"),
                v = c(4, 2, 1, 2))
PlotDataPairRatio(d, "id", "set", "x", "ref", "v", log_base = 2)
```

PlotDataProduct *Product of Columns*

Description

Multiplies columns row by row, divides by a constant and optionally rounds (half to even), for example a count from a total and a percentage.

Usage

```
PlotDataProduct(data, columns, divide = 1, round = FALSE, into = "value")
```

Arguments

data	A data frame.
columns	Columns to multiply, in order.
divide	Divisor applied after the product.
round	Round to whole numbers.
into	Name of the new column.

Value

data with the new column.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataProduct(data.frame(n = c(44, 2), pct = c(54.5454545, 50)), c("n", "pct"),
                divide = 100, round = TRUE, into = "k")
```

PlotDataQuantiles *Quantile Summaries of Plot Data*

Description

Summarises a value column by groups (for example a distance curve across patients) into lower, middle and upper quantiles and the number of non-missing values. Quantiles use R's default (type 7) interpolation, which equals the linear rule of common numerical libraries.

Usage

```
PlotDataQuantiles(
  data,
  by,
  value,
  probs = c(0.25, 0.5, 0.75),
  names = c("lower", "median", "upper"),
  n_name = "n"
)
```

Arguments

data	A data frame.
by	Grouping columns; groups keep their order of first appearance.
value	Value column.
probs	Three probabilities.
names	Names of the three quantile columns.
n_name	Name of the count column.

Value

A data frame with the by columns, the quantile columns and the count column.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
d <- data.frame(type = rep(c("a", "b"), each = 6), x = rep(1:2, 6), v = 1:12)
PlotDataQuantiles(d, by = c("type", "x"), value = "v")
```

PlotDataRankRows *Rank Rows by the Sum of Selected Columns*

Description

Sums the columns named in `columns` or matching `pattern` in each row and ranks the rows (rank 1 = largest sum by default; ties keep the row order).

Usage

```
PlotDataRankRows(
  data,
  id,
  columns = NULL,
  pattern = NULL,
  decreasing = TRUE,
  ids = NULL
)
```

Arguments

`data`	Wide data frame with one row per item.
`id`	Identifier column.
`columns`	Columns to add up.
`pattern`	Regular expression selecting further columns.
`decreasing`	Rank the largest sum first.
`ids`	Optional identifiers to keep (a vector, or a data frame whose first column holds them); other rows are dropped.

Value

A data frame with `id`, `value` and `rank`, sorted by rank.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
d <- data.frame(p = c("a", "b", "c"), "CN1 T" = c(10, 40, 20), check.names = FALSE)
PlotDataRankRows(d, "p", pattern = "^CN1 ")
```

PlotDataRatio *Ratios Between Columns*

Description

Adds `scale * (numerator + pseudocount) / (denominator + pseudocount)`, optionally on a logarithmic scale. Zero numerators give `-Inf` on a log scale; builders such as PlotBandProfiles() draw them at a floor.

Usage

```
PlotDataRatio(
  data,
  numerator,
  denominator,
  pseudocount = 0,
  log_base = NULL,
  scale = 1,
  into = "ratio"
)
```

Arguments

`data`	A data frame.
`numerator, denominator`	Numeric columns.
`pseudocount`	Added to both columns.
`log_base`	Optional logarithm base.
`scale`	Multiplier applied before the logarithm.
`into`	Name of the new column.

Value

`data` with the ratio column.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataRatio(data.frame(a = c(1, 0, 4), b = 2), "a", "b", log_base = 2)
```

PlotDataRecode *Recode the Values of a Column*

Description

Maps the values of a column to new values. Each value is looked up, in this order, in an exact `map`, in a list of regular expressions (`patterns`, the first match wins) and by position in `index` (integer codes counted from `index_base`); an optional `extract` expression first picks the part of each value that is looked up. Unmatched values are kept, or replaced by `default`. Finally `substitute` applies regular-expression replacements to every value, in order (for example to turn analysis names into display labels).

Usage

```
PlotDataRecode(
  data,
  column,
  map = NULL,
  patterns = NULL,
  index = NULL,
  index_base = 0L,
  extract = NULL,
  default = NULL,
  keep_unmatched = is.null(default),
  substitute = NULL,
  into = column
)
```

Arguments

`data`	A data frame.
`column`	Column to recode.
`map`	Optional named vector or list: old value to new value.
`patterns`	Optional named vector or list: regular expression to new value, tried in order for values not in `map`.

index	Optional names for integer codes: a vector, or a data frame whose first column is used.
index_base	Code of the first name of index.
extract	Optional regular expression with one capture group; the group is looked up instead of the whole value.
default	Value for unmatched entries; NULL keeps them.
keep_unmatched	Keep unmatched values; with FALSE they become default (missing when default is NULL).
substitute	Optional list of c(pattern, replacement) pairs (Perl regular expressions) applied to every value at the end, in order.
into	Name of the result column; default column.

Details

The result is numeric when every replacement value is numeric and unmatched values are not kept (for example ordinal codes for reported categories); otherwise it is text.

Value

data with the recoded column.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
d <- data.frame(e = c("pct|B cells", "density|B cells"), g = c("A-x", "B"), cn = c(0, 2))
PlotDataRecode(d, "e", substitute = list(c("^pct\\|(.*)$", "\\1 (% of cells)"),
                                         c("^density\\|(.*)$", "\\1 (per mm2)")))
PlotDataRecode(d, "g", patterns = c("^A" = "first"), default = "other", into = "label")
PlotDataRecode(d, "cn", index = c("zero", "one", "two"), into = "name")
PlotDataRecode(data.frame(tps = c("0", "<1", "8", "n.a.")), "tps",
           map = list("0" = 0, "<1" = 0.5, "8" = 8), keep_unmatched = FALSE, into = "score")
```

PlotDataRelative *Scale a Column to a Reference Row*

Description

Divides a column by its value in a reference row (the first row by default, or the row where at_column equals at), for curves shown relative to their first point.

Usage

```
PlotDataRelative(data, column, at_column = NULL, at = NULL, into = column)
```

Arguments

`data`	A data frame.
`column`	Column to scale.
`at_column`, `at`	Optional reference row selector.
`into`	Name of the result column; default `column`.

Value

`data` with the scaled column.

See Also

Other plot-data builders: `PlotDataBindRows()`, `PlotDataConstant()`, `PlotDataCount()`, `PlotDataCrossShares()`, `PlotDataDirection()`, `PlotDataDominant()`, `PlotDataFromArrays()`, `PlotDataJoin()`, `PlotDataKeyColours()`, `PlotDataLabels()`, `PlotDataLonger()`, `PlotDataMembership()`, `PlotDataPairRatio()`, `PlotDataProduct()`, `PlotDataQuantiles()`, `PlotDataRankRows()`, `PlotDataRatio()`, `PlotDataRecode()`, `PlotDataSelect()`, `PlotDataShares()`, `PlotDataSummarise()`, `PlotDataUnite()`

Examples

```
PlotDataRelative(data.frame(k = 4:6, inertia = c(50, 40, 30)), "inertia", into = "relative")
```

PlotDataSelect *Select Rows and Columns of Plot Data*

Description

Keeps rows whose columns take given values (`filter`) and that meet comparison conditions (`where`), optionally orders rows by a given order of one column, and keeps or renames columns.

Usage

```
PlotDataSelect(
  data,
  filter = list(),
  where = list(),
  columns = NULL,
  rename = NULL,
  order_by = NULL,
  order = NULL
)
```

Arguments

data	A data frame.
filter	Named list: column name to the values to keep. Logical values are compared as logical, all others as text.
where	Conditions, each a list with column, op and (except for is_na, not_na) value. Operators: <, <=, >, >=, ==, !=, in, not_in, is_na, not_na, matches, not_matches (regular expression). Conditions are combined with AND; comparisons with missing values are false.
columns	Columns to keep (all if NULL).
rename	Named character vector or list, new name to old name.
order_by	Optional column whose values follow order.
order	Value order for order_by; rows with other values are dropped.

Value

A data frame.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataShares(), PlotDataSummarise(), PlotDataUnite()

Examples

```
d <- data.frame(type = c("x", "y", "x"), v = 1:3, n = c(1, 12, NA))
PlotDataSelect(d, filter = list(type = "x"), rename = c(value = "v"))
PlotDataSelect(d, where = list(list(column = "n", op = ">=", value = 10)))
```

PlotDataShares *Shares of a Total in Several Columns*

Description

Divides selected count columns by a total column row by row, for example excluded objects of several kinds as percentages of all objects.

Usage

```
PlotDataShares(data, columns, total, scale = 100, suffix = "")
```

Arguments

data	A data frame.
columns	Count columns.
total	Column holding the total.
scale	Multiplier (100 for percentages).
suffix	Suffix of the new columns (empty replaces the columns).

Value

data with the share columns.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataSummarise(), PlotDataUnite()

Examples

```
PlotDataShares(data.frame(a = 1:2, b = 3:4, n = c(10, 20)), c("a", "b"), "n")
```

PlotDataSummarise *Summarise Columns by Group*

Description

One summary per group (or one in total) of each selected column, in wide form (one column per summarised column) or long form (one row per group and column).

Usage

```
PlotDataSummarise(
  data,
  columns = NULL,
  fun = "sum",
  by = NULL,
  into = NULL,
  filter = NULL,
  long = FALSE,
  names_to = "name",
  values_to = "value",
  sort = FALSE,
  na_rm = TRUE
)
```

Arguments

data	A data frame.
columns	Columns to summarise (for fun = "count" only the first is used, default the first column of data).
fun	One of "sum", "mean", "median", "min", "max", "count" (rows) or "n_distinct".
by	Optional grouping columns.
into	Names of the result columns in wide form; default columns (or "n" for counts).
filter	Optional named list of columns and values selecting rows first.
long	Return long form with names_to (column name) and values_to (summary).
names_to, values_to	Names of the long-form columns.
sort	Order groups by their values; by default groups keep the order of first appearance.
na_rm	Ignore missing values.

Value

A data frame.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataUnite()

Examples

```
d <- data.frame(g = c("b", "b", "a"), x = 1:3, y = c(2, 4, 6))
PlotDataSummarise(d, c("x", "y"), "sum")
PlotDataSummarise(d, fun = "count", by = "g")
PlotDataSummarise(d, c("x", "y"), "mean", by = "g", long = TRUE, sort = TRUE)
```

PlotDataTableCells *Text-Table Cells from a Per-Unit Table*

Description

Transposes a table with one row per unit (patient, analysis) into the cells of a small text table for PlotTextBlock(): a header row with the unit labels and one row per value column, each with a row label in the first column. Values are written as numbers (whole numbers without decimals, missing as "NE"), as text for text columns, or from one template per table row (see FormatTemplate()). Cells whose value lies below highlight_below are flagged.

Usage

```
PlotDataTableCells(
  data,
  id,
  values = NULL,
  row_labels = values,
  header = "",
  key = NULL,
  highlight_below = NULL,
  zero_text = NULL,
  templates = NULL
)
```

Arguments

data	A data frame, one row per unit, in display order.
id	Column with the unit labels shown in the header.
values	Value columns, one table row each (optional with templates).
row_labels	Labels of the value rows.
header	Label of the header row.
key	Optional column copied to the header cells as key (for example a group used to colour header text).
highlight_below	Values below this are flagged (highlight = TRUE).
zero_text	Optional text written instead of zero values.
templates	Optional templates, one per table row, filled from each unit's row (for example "p = {p|p}").

Value

A data frame of cells with row, column, label, header, highlight and key.

Examples

```
d <- data.frame(id = c("1", "2"), a = c(3, 0), b = c(12, 40), g = c("x", "y"))
PlotDataTableCells(d, "id", c("a", "b"), c("First", "Second"), key = "g",
                   highlight_below = 10, zero_text = "0 (NE)")
PlotDataTableCells(data.frame(id = c("A", "B"), p = c(0.018, 0.058)), "id",
                   row_labels = "Permutation test", templates = "p = {p|p}")
```

PlotDataUnite *Paste Columns into a Key Column*

Description

Paste Columns into a Key Column

Usage

```
PlotDataUnite(data, columns, into, sep = " ")
```

Arguments

data	A data frame.
columns	Columns to paste, in order.
into	Name of the new column.
sep	Separator.

Value

data with the key column. Whole numbers are written without decimals.

See Also

Other plot-data builders: PlotDataBindRows(), PlotDataConstant(), PlotDataCount(), PlotDataCrossShares(), PlotDataDirection(), PlotDataDominant(), PlotDataFromArrays(), PlotDataJoin(), PlotDataKeyColours(), PlotDataLabels(), PlotDataLonger(), PlotDataMembership(), PlotDataPairRatio(), PlotDataProduct(), PlotDataQuantiles(), PlotDataRankRows(), PlotDataRatio(), PlotDataRecode(), PlotDataRelative(), PlotDataSelect(), PlotDataShares(), PlotDataSummarise()

Examples

```
PlotDataUnite(data.frame(type = "a", r = 20), c("type", "r"), "row")
```

PlotDensityOutlines *Density Outlines on an Embedding*

Description

Draws, for each group (for example each patient), the region of the embedding inside a density level as a lightly filled area with an outline, as computed by DensityLevelContour(). Each group takes the colour of its slot; a compact key of line segments with the group labels is set centred below the panel.

Usage

```
PlotDensityOutlines(
  data,
  group = "group",
  limits = NULL,
  subset = NULL,
  colours = NULL,
  fill_alpha = 0.14,
  line_width = 0.5,
  key = TRUE,
  key_size = 4.2,
  key_ncol = 6,
  texts = NULL,
  values = NULL,
  style = ManuscriptStyle()
)
```

Arguments

data	Contour data from DensityLevelContour(): the group column, slot, label, kind ("band" or "line"), piece, x, y.
group	Name of the group column.
limits	Display limits as in PlotEmbedding().
subset	Optional named list of columns and values to keep.
colours	Colours by slot (vector) or named by group.
fill_alpha	Opacity of the filled region.
line_width	Outline width in points.
key	Draw the key below the panel.
key_size	Key text size in points.
key_ncol	Key columns.
texts, values	Free texts and template values.
style	A ManuscriptStyle() object.

Value

A 'ggplot2' object.

Examples

```
set.seed(2)
pts <- data.frame(x = c(rnorm(300), rnorm(300, 2)), y = c(rnorm(300), rnorm(300, 1)),
                  g = rep(c("p1", "p2"), each = 300))
lim <- c(-4, 6, -4, 5)
cont <- DensityLevelContour(pts, group = "g", limits = lim, bins = 40)
PlotDensityOutlines(cont, group = "g", limits = lim, colours = c("#e34948", "#eda100"))
```

PlotDotMatrix *Dot Matrix*

Description

One dot per row and column, its area growing with one measure (for example the percentage of positive cells, as a figure scatter area `area[1] + area[2] * size` in points squared) and its colour showing a second (for example a z-scored signal on a diverging scale). Rows without any value are dropped. Thin grid lines run through the dots and a light frame surrounds the matrix; row annotation boxes, a colour bar and a size key can be added outside it.

Usage

```
PlotDotMatrix(
  data,
  row = "row",
  column = "column",
  size = "size",
  signal = "signal",
  rows = NULL,
  columns = NULL,
  row_labels = NULL,
  column_labels = NULL,
  area = c(2, 0.55),
  scale = "rdbu_r",
  limits = c(-3, 3),
  edge_colour = "#777777",
  edge_width = 0.25,
  grid_colour = "#F0F0F0",
  grid_width = 0.5,
  frame_colour = "#BBBBBB",
  frame_width = 0.4,
  pad = 0.1,
  row_colours = NULL,
  row_box = c(-1.05, 0.35, 0.76),
  column_angle = 55,
  column_size = 5.1,
  column_pad = 3.5,
  row_size = 5.3,
  row_pad = 9,
  colourbar = NULL,
  size_key = NULL,
  texts = NULL,
  values = NULL,
  style = ManuscriptStyle()
)
```

Arguments

data	Long data: one row per matrix cell.
row, column	Columns with the row and column keys.
size, signal	Columns with the size and colour measures.
rows, columns	Row order (top to bottom) and column order (left to right): character vectors or data frames (ordered by a rank column if present); default first appearance.
row_labels, column_labels	Display labels: named mapping or vector.
area	Scatter area c(intercept, slope) in squared points.
scale	Colour scale name or vector of anchor colours.
limits	Colour limits; values are clipped to them.
edge_colour, edge_width	Dot outline colour and width (points).
grid_colour, grid_width	Grid lines.
frame_colour, frame_width	Frame around the matrix.
pad	Extra space beyond the outer dots, in row and column units.
row_colours	Named colours of row annotation boxes.
row_box	Box geometry c(left, width, height) in column units, left measured from the centre of the first column.
column_angle, column_size, column_pad	Column label rotation (degrees), size and distance from the matrix (points).
row_size, row_pad	Row label size (points) and distance from the matrix (points).
colourbar	Optional list: box (c(left, bottom, width, height) in panel fractions), title, breaks, labels, tick_size, title_size.
size_key	Optional list: values (sizes shown), at (panel fractions of the first key dot), step (panel fraction between dots), fill, label_dx (panel fraction from dot to label), title, title_at, size (text points), suffix (default "%").
texts	Optional texts at panel fractions (list of label, xy, size, colour, bold, hjust, vjust, angle); labels may use FormatTemplate() placeholders filled from values.
values	Optional data frame for text templates.
style	A ManuscriptStyle() object.

Value

A 'ggplot2' object.

Examples

```
d <- expand.grid(row = c("a", "b", "c"), column = c("m1", "m2"), stringsAsFactors = FALSE)
d$size <- c(90, 10, 40, 5, 70, 20)
d$signal <- c(2, -1, 0.5, -2, 1.5, 0)
PlotDotMatrix(d)
```

PlotEmbedding *Embedding Scatter*

Description

Draws cells on two embedding coordinates (for example UMAP) as small points without frame, coloured by a key (cell type, cluster, neighbourhood). Points are drawn in layers in the order of the data rows, as given by `EmbeddingLayers()`, and rasterised at the style resolution so that vector output stays small. Point sizes follow figure scatter areas (squared points); individual keys can be drawn as diamonds with an outline. Optional cluster labels with a white halo, arrow axes or plain axis titles, and texts filled from plot data complete the panel.

Usage

```
PlotEmbedding(
  data,
  x = "x",
  y = "y",
  key = "key",
  limits = NULL,
  subset = NULL,
  colours = NULL,
  default_colour = "#BBBBBB",
  sizes = 0.3,
  shapes = NULL,
  edge_colours = NULL,
  edge_widths = NULL,
  labels = NULL,
  label_size = 4.2,
  label_colour = "ink",
  halo = 0.6,
  axis_style = c("none", "arrows", "titles"),
  axis_titles = c("UMAP 1", "UMAP 2"),
  arrow_fraction = 0.16,
  arrow_text_size = 4,
  title_size = 4.6,
  title_pad = 1.5,
  raster = TRUE,
  legend = NULL,
  texts = NULL,
  values = NULL,
  style = ManuscriptStyle()
)
```

Arguments

data	Point data with coordinate and key columns.

x, y, key	Column names.
limits	Display limits c(xmin, xmax, ymin, ymax) or a one-row data frame with these columns (see EmbeddingLimits()).
subset	Optional named list: keep rows whose columns take these values (for example one clinical group per panel).
colours	Colours per key: named vector or list, or a data frame of key and colour.
default_colour	Colour of keys without a colour.
sizes	Scatter areas in squared points, one value or per key.
shapes	Optional marker per key: "o" (circle) or "D" (diamond).
edge_colours, edge_widths	Optional outline colour and width (points) per key.
labels	Optional data frame of text labels (label, x, y), for example from ClusterLabelPositions(), drawn bold with a white halo.
label_size, label_colour, halo	Label size, colour and halo width.
axis_style	"none", "arrows" (two arrows in the lower left corner) or "titles" (axis titles beside the panel).
axis_titles	Titles of the two axes.
arrow_fraction	Arrow length as a fraction of the x range.
arrow_text_size, title_size, title_pad	Text sizes and the distance of axis titles from the panel (points).
raster	Rasterise the points (default) or draw vectors.
legend	Optional legend: list of keys, labels, at (upper left corner in panel fractions), ncol, size, marker_size (points), label_spacing, column_spacing, handle_text_pad, handle_length (in font sizes), title, title_size, title_align, and optional colours and shapes overriding those of the points. Columns are filled first.
texts, values	Free texts and template values, as in PlotAnnotatedHeatmap(); templates are filled from values restricted to subset, or from the subset data.
style	A ManuscriptStyle() object.

Value

A 'ggplot2' object.

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(500), y = rnorm(500), key = sample(c("a", "b"), 500, TRUE))
PlotEmbedding(d, limits = c(-4, 4, -4, 4), colours = c(a = "#e34948", b = "#2a78d6"),
              axis_style = "arrows", axis_titles = c("Dim 1", "Dim 2"))
```

PlotFluorescenceField *Multichannel Fluorescence Field*

Description

Composes an additive multichannel image from channel windows (a field as returned by `segmantR::sg_read_field()`), display levels and channel colours and weights (see `CompositeFields()` for the layer and level rules), and draws it at its physical extent with optional scale bar, corner text, arrows pointing at cells, crosses on partner cells and boxes.

Usage

```
PlotFluorescenceField(
  field,
  layers,
  slide = NULL,
  levels = NULL,
  marks = NULL,
  index = NULL,
  marks_index = NULL,
  gamma = 0.8,
  min_span = 1,
  interpolate = TRUE,
  arrow = NULL,
  cross = NULL,
  corner_text = NULL,
  corner_size = 5,
  corner_colour = "white",
  corner_at = c(0.04, 0.96),
  label = NULL,
  label_size = 5.5,
  scale_bar = list(length = 50, label = "50 um", colour = "white"),
  boxes = NULL,
  box_colour = "white",
  box_width = 0.6,
  box_size = NULL,
  frame = FALSE,
  style = ManuscriptStyle()
)
```

Arguments

`field`	A field (`channels`, `centre`, `size`) or a list of fields with `index` selecting one.
`layers`	List of channel layers (see `CompositeFields()`).
`slide, levels`	Optional range tables (`channel`, `lo`, `hi`).
`marks`	Optional data frame of cells to annotate (`x`, `y`, and for crosses `partner_x`, `partner_y`); `marks_index` selects rows.

index, marks_index
: Optional selections.

gamma, min_span
: Display gamma and minimum level span.

interpolate
: Smooth the raster when scaling (bilinear).

arrow
: NULL or a list: tip, tail (offsets from the cell along both axes, physical units), colour, linewidth (points), head (arrow-head scale in points) and shrink (points left free at both ends).

cross
: NULL or a list: colour, size (marker area in square points), linewidth; drawn at the partner positions.

corner_text
: NULL, a fixed text, or a list with column and format (sprintf) taking the value from the first mark.

corner_size, corner_colour
: Corner text size and colour.

corner_at
: Corner text position (fractions of the field, top-left anchored).

label
: Optional field label in the top-left corner (as corner_text, larger).

label_size
: Label size.

scale_bar, boxes, box_colour, box_width, box_size, frame
: As in PlotCellMap().

style
: A ManuscriptStyle() object.

Value

A 'grid' grob.

Examples

```
set.seed(3)
field <- list(channels = list(cd3 = matrix(runif(400, 0, 100), 20),
                              foxp3 = matrix(runif(400, 0, 100), 20)),
              centre = c(10, 10), size = 10)
g <- PlotFluorescenceField(field,
  layers = list(list(channel = "cd3", colour = "#00C8FF", lo = "q0.01", hi = "q0.99"),
                list(channel = "foxp3", colour = "#FF00C8", lo = "q0.5", hi = "q1",
                     overlay = TRUE)),
  scale_bar = list(length = 5, label = "5 um"))
grid::grid.newpage(); grid::grid.draw(g)
```

PlotGroupDifferences *Differences Between Groups as Segments*

Description

One row per contrast or endpoint: a segment from zero to the estimate (not an interval) ending in a point, and a P-value text at the right. With marker = "colour" points, segments and text take significant_colour when q is below alpha and other_colour otherwise; with marker = "open" significant rows have filled points in significant_colour and the others open points with an open_colour edge, segments are drawn in segment_colour and the text is ink or muted. The x axis is symmetric around zero unless x_limits are given.

Usage

```
PlotGroupDifferences(
  data,
  row = "contrast",
  estimate = "diff",
  q = "q_BH",
  row_order = NULL,
  row_labels = NULL,
  alpha = 0.05,
  significant_colour = "significant",
  other_colour = "muted",
  label = "{diff|sf1}; p = {p|p}; q = {q_BH|p}",
  label_size = 4.6,
  label_x = 1.04,
  label_title = NULL,
  label_title_size = 4.8,
  span = 1.25,
  linewidth = 1.2,
  point_size = 11,
  marker = c("colour", "open"),
  segment_colour = "#C9C9C9",
  open_colour = "open_edge",
  open_size = 13,
  open_stroke = 0.7,
  x_limits = NULL,
  y_limits = NULL,
  x_breaks = NULL,
  x_nbins = 6,
  x_label = NULL,
  grid = c("none", "x"),
  tick_size_y = 5,
  title_size = 5,
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Test table, one row per contrast.
`row`, `estimate`, `q`	Columns of the contrast, estimate and the (adjusted or nominal) P value compared with `alpha`.
`row_order`	Rows from top to bottom.
`row_labels`	Named tick labels.
`alpha`	Threshold for `q`.
`significant_colour`, `other_colour`	Colours.
`label`	Template written at the right (see `FormatTemplate()`).

`label_size`, `label_x`
: Label size and position (panel fraction).

`label_title`
: Optional column title written above the labels.

`label_title_size`
: Its size.

`span`
: Half-width of the axis relative to the largest absolute estimate.

`linewidth`, `point_size`
: Segment width (points) and point area.

`marker`
: `"colour"` or `"open"` (see Details).

`segment_colour`
: Segment colour with `marker = "open"`.

`open_colour`, `open_size`, `open_stroke`
: Open points: edge colour, area and edge width.

`x_limits`
: Optional x limits.

`y_limits`
: Optional y limits (rows are at 1 to n from the bottom).

`x_breaks`, `x_nbins`
: x ticks.

`x_label`
: Axis title.

`grid`
: `"none"` or `"x"` (vertical grid lines).

`tick_size_y`
: Row label size.

`title_size`
: Axis title size.

`box_mm`
: Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).

`style`
: A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
t <- data.frame(contrast = c("A vs B", "C vs B"), diff = c(3.2, -1.1), p = c(0.01, 0.4),
                q_BH = c(0.03, 0.5))
PlotGroupDifferences(t)
PlotGroupDifferences(t, q = "p", marker = "open", label = "{p|p}", label_title = "P",
                     x_limits = c(-4, 4), grid = "x")
```

PlotGroupStrip *Unit Values by Group*

Description

Jittered points of one value per unit (patient) for each group, with a bar at the group median. With `category`, groups are drawn side by side within each category (for example one neighbourhood per category) with faint separators between categories. On a logarithmic axis, values at or below zero are not drawn and counted in a note at the bottom.

Usage

```
PlotGroupStrip(
  data,
  value = "value",
  group = "group",
  groups = NULL,
  group_labels = NULL,
  group_colours = NULL,
  colour_by = NULL,
  point_colours = NULL,
  category = NULL,
  category_order = NULL,
  category_labels = NULL,
  dodge = 0.27,
  jitter = NULL,
  seed = 3,
  point_size = 10,
  median_width = NULL,
  median_linewidth = 1,
  log_y = FALSE,
  zero_text = "{n} at zero",
  y_limits = NULL,
  y_breaks = NULL,
  y_nbins = 9,
  y_label = NULL,
  x_label = NULL,
  tick_size = NULL,
  title_size = 5.6,
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Tidy data: one row per unit (and category).
`value, group`	Columns.

groups	Group order.
group_labels	Named tick labels per group (for example short names).
group_colours	Named colours per group.
colour_by	Optional column colouring the points instead of the group (for example the clinical group of each unit), with point_colours.
point_colours	Named colours per value of colour_by.
category	Optional category column; category_order and category_labels order and label it.
category_order, category_labels	Category order and tick labels.
dodge	Offset between groups within a category.
jitter	Half-width of the horizontal jitter.
seed	Seed of the jitter (presentation only).
point_size	Point area (pt^2).
median_width	Half-width of the median bar (x units).
median_linewidth	Its width in points.
log_y	Logarithmic y axis.
zero_text	Template for the count of values at or below zero (field {n}).
y_limits, y_breaks, y_nbins	y axis.
y_label, x_label	Axis titles.
tick_size	Tick label size.
title_size	Axis title size.
box_mm	Size of the data area c(width, height) in millimetres, passed by RenderFigure(); used by "auto" tick intervals (see NiceBreaks()).
style	A ManuscriptStyle() object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(group = rep(c("A", "B", "C"), c(5, 2, 4)), value = c(1:5, 2.5, 4, 3:6))
PlotGroupStrip(d, group_colours = c(A = "#009E73", B = "#D55E00", C = "#0072B2"))
```

PlotImage *Image, Image Gallery or Schematic*

Description

Draws one raster image, a gallery of images, or a schematic of shapes.

Usage

```
PlotImage(
  data,
  row = NULL,
  row_levels = NULL,
  tile_mm = 10,
  view_gap_mm = 0.36,
  field_gap_mm = 1.8,
  row_pitch_mm = 11,
  column_labels = NULL,
  header_size = 4.6,
  header_colour = "muted",
  header_gap_mm = 0.5,
  row_labels = NULL,
  cross = list(colour = "#FFD54F", size = 3.5, linewidth = 0.5),
  corner = NULL,
  scale_bar = NULL,
  title = NULL,
  total = NULL,
  title_size = 5.6,
  title_gap_mm = 4.5,
  interpolate = TRUE,
  style = ManuscriptStyle()
)
```

Arguments

data	An array/matrix, a CompositeFields() result or a SchematicShapes() result.
row	Column of the gallery info defining gallery rows; NULL gives one row per field.
row_levels	Optional order of the row values.
tile_mm, view_gap_mm, field_gap_mm, row_pitch_mm	Gallery geometry.
column_labels	Optional header texts per view (repeated for every field of the widest row) with header_size and header_colour.
header_size, header_colour, header_gap_mm	Header text settings.

row_labels	Optional list of row-label lines, each a list with column (info column) or map (named list from the row value), dy (fraction of the tile height from the bottom), size, colour, bold, replace (named list of text substitutions) and x_mm (position relative to the panel's left edge; negative is left of it); hjust aligns the text.
cross	Draw a centre cross on every tile (list with colour, size points, linewidth points) or NULL.
corner	Optional list with column, view, size, colour: text in the top-left corner of tiles of that view.
scale_bar	Optional list: length (physical units), pixel_size, view, linewidth, colour, which ("first" tile only or "row": first tile of every row).
title	Optional gallery title placed above the column headers; {n} is replaced by the number of fields shown and {total} by the number of rows of total.
total	Optional data frame counted for {total}.
title_size, title_gap_mm	Title size (points) and distance above the tiles (mm).
interpolate	Smooth raster scaling.
style	A ManuscriptStyle() object.

Details

- A single image (RGB array or grey matrix with values in 0-1) fills the panel box with optional centre cross, scale bar and corner label.
- A CompositeFields() result is laid out as a gallery: one gallery row per value of row (or per field), each field's views side by side, in absolute millimetres from the top-left of the panel box: tiles of tile_mm, view_gap_mm between the views of a field, field_gap_mm between fields and row_pitch_mm from row to row. Column headers, row labels (several lines, from columns of the fields' rows or a label map), centre crosses, corner labels and a scale bar are optional.
- A SchematicShapes() result is drawn in its own coordinates with equal aspect.

Value

A 'grid' grob.

Examples

```
shapes <- SchematicShapes(shapes = list(
  list(type = "roundrect", x = 0.5, y = 0.5, w = 0.8, h = 0.5, fill = "#EEEEEE",
       colour = "#999999", r = 1),
  list(type = "text", x = 0.5, y = 0.5, label = "Box"),
  list(type = "arrow", x = 0.1, y = 0.1, xend = 0.9, yend = 0.1, colour = "#000000")))
grid::grid.newpage(); grid::grid.draw(PlotImage(shapes))
set.seed(4)
grid::grid.newpage(); grid::grid.draw(PlotImage(array(runif(300), c(10, 10, 3))))
```

`PlotIntervalForest` *Interval Forest with Unit Values*

Description

One row per analysis or cell type: an interval (for example a bootstrap confidence interval of the median), a marked estimate, optional jittered unit (patient) values behind it, a reference line and a statistic text to the right of the panel written from the data. Units flagged in the unit table (for example patients with low support) can be drawn as open symbols in their group colour with a short label.

Usage

```
PlotIntervalForest(
  data,
  points = NULL,
  row = "row",
  estimate = "estimate",
  lower = "lower",
  upper = "upper",
  value = "value",
  row_order = NULL,
  row_labels = NULL,
  colours = "ink",
  interval_colours = NULL,
  estimate_size = 18,
  estimate_edge = 0.4,
  interval_linewidth = 1.2,
  point_colour = "#BDBDBD",
  point_size = 5,
  jitter = 0.18,
  seed = 1,
  flag = NULL,
  flag_group = NULL,
  group_colours = NULL,
  flag_size = 15,
  flag_edge = 0.8,
  flag_label = NULL,
  flag_label_size = 4.8,
  flag_label_vjust = 0.5,
  flag_spread = 0.17,
  label = NULL,
  label_size = 4.8,
  label_colour = "muted",
  label_x = 1.03,
  reference_x = 0,
  reference_linewidth = 0.6,
```

```
  x_limits = NULL,
  y_limits = NULL,
  x_breaks = NULL,
  x_nbins = 6,
  x_label = NULL,
  y_label = NULL,
  tick_size_x = NULL,
  tick_size_y = NULL,
  title_size = NULL,
  grid = c("x", "none"),
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Summary table, one row per row of the forest.
`points`	Optional unit table with `row` and `value` columns.
`row`, `estimate`, `lower`, `upper`	Columns of `data`.
`value`	Value column of `points`.
`row_order`	Rows from top to bottom (default: order in `data`).
`row_labels`	Named tick labels per row.
`colours`	Estimate colours: one colour or named per row.
`interval_colours`	Interval colours (default: the estimate colours).
`estimate_size`	Estimate marker area (pt^2), drawn with a white edge of `estimate_edge` points.
`estimate_edge`	Edge width.
`interval_linewidth`	Interval width in points.
`point_colour`, `point_size`	Unit points (area in pt^2).
`jitter`	Half-width of the vertical jitter of unit points (rows are one unit apart).
`seed`	Seed of the jitter (presentation only).
`flag`	Optional logical column of `points` marking units drawn as open symbols in their group colour instead of grey points; these are spread over `flag_spread` around the row in their order of appearance.
`flag_group`	Group column of `points` for flagged units.
`group_colours`	Named group colours.
`flag_size`, `flag_edge`	Open symbol area and edge width.
`flag_label`	Optional column of `points` written left of flagged symbols.
`flag_label_size`, `flag_label_vjust`	Its size and vertical justification.

`flag_spread` Half-range of the vertical spread of flagged symbols.

`label` Optional template (see `FormatTemplate()`) written for each row at `label_x` (panel fraction) to the right.

`label_size`, `label_colour`, `label_x`
Label style and position.

`reference_x` Position of the vertical reference line (`NULL` for none).

`reference_linewidth`
Its width in points.

`x_limits`, `y_limits`
Limits; rows sit at 1 (bottom) to n (top). Automatic limits add 5% of the data range.

`x_breaks`, `x_nbins`
x ticks.

`x_label`, `y_label`
Axis titles.

`tick_size_x`, `tick_size_y`
Tick label sizes.

`title_size` Axis title size.

`grid` `"x"` (default) or `"none"`.

`box_mm` Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).

`style` A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
s <- data.frame(row = c("a", "b"), estimate = c(1, -0.5), lower = c(0.2, -1.2),
                upper = c(1.5, 0.1), k = c(9, 7), n = 9, p = c(0.004, 0.1))
pts <- data.frame(row = rep(c("a", "b"), each = 5), value = c(rnorm(5, 1), rnorm(5, -0.5)))
PlotIntervalForest(s, pts, colours = c(a = "#D9730D", b = "#C2378E"),
                   label = "{k|d}/{n|d} patients; p = {p|p}")
```

PlotPairedPatients *Paired Patient Lines*

Description

One line per patient across ordered conditions, coloured by clinical group, with a short bold bar at the median of each condition. Patients flagged in `excluded` are drawn dashed with open symbols and are left out of the medians, as patients not included in a test.

Usage

```
PlotPairedPatients(
  data,
  patient = "patient",
  group = "group",
  condition = "condition",
  value = "value",
  conditions = NULL,
  labels = NULL,
  excluded = NULL,
  group_colours = NULL,
  log_y = FALSE,
  y_label = NULL,
  x_label = NULL,
  y_limits = NULL,
  y_nbins = "auto",
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Tidy data: one row per patient and condition.
`patient`, `group`, `condition`, `value`	Column names.
`conditions`	Condition order; defaults to first appearance.
`labels`	Tick labels for the conditions; default `conditions`.
`excluded`	Optional logical column marking patients outside the test.
`group_colours`	Named colours per group level.
`log_y`	Logarithmic y axis.
`y_label`, `x_label`	Axis titles.
`y_limits`	Optional y limits.
`y_nbins`	Maximum number of y intervals of the tick locator (see `NiceBreaks()`); `"auto"` follows the axis length.
`box_mm`	Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`.
`style`	A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(patient = rep(c("p1", "p2", "p3"), each = 2),
                group = rep(c("A", "B", "A"), each = 2),
                condition = rep(c("x", "y"), 3), value = c(3, 5, 2, 6, 4, 4.5))
PlotPairedPatients(d, group_colours = c(A = "#009E73", B = "#0072B2"))
```

`PlotQuantileCurves` *Summary Curves with Quantile Bands*

Description

Draws one curve per series over a continuous axis (for example the median across patients of a cumulative distance distribution) with an optional band between lower and upper quantiles, an optional dashed reference line, direct series labels and a corner note written from the data. With `facet`, draws a grid of small multiples with identical axes and a coloured title above each panel.

Usage

```
PlotQuantileCurves(
  data,
  x = "x",
  y = "median",
  lower = NULL,
  upper = NULL,
  series = "series",
  series_order = NULL,
  colours = NULL,
  linewidth = 1.4,
  band_alpha = 0.15,
  labels = NULL,
  label_at = NULL,
  label_dx = 3,
  label_size = 5.3,
  reference_x = NULL,
  reference_linewidth = 0.5,
  x_limits = NULL,
  y_limits = NULL,
  x_breaks = NULL,
  y_breaks = NULL,
  x_nbins = 6,
  y_nbins = 9,
  x_label = NULL,
  y_label = NULL,
  tick_size = NULL,
  note = NULL,
  note_xy = c(0.97, 0.04),
  note_hjust = 1,
```

```
  note_vjust = 0,
  note_size = 5,
  note_colour = "muted",
  facet = NULL,
  facet_order = NULL,
  facet_ncol = 4,
  facet_gap_mm = c(5, 5),
  facet_data = NULL,
  facet_title = NULL,
  facet_note = NULL,
  facet_group = NULL,
  group_colours = NULL,
  facet_title_size = 5.6,
  facet_note_size = 4.6,
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Tidy data: one row per series (and facet) and x value.
`x`, `y`	Columns of the x value and the curve value.
`lower`, `upper`	Optional band columns.
`series`	Series column.
`series_order`	Drawing order of the series (later on top).
`colours`	Named colours per series.
`linewidth`	Curve width in points.
`band_alpha`	Band opacity.
`labels`	Optional named direct labels per series.
`label_at`	x value at which direct labels are placed; the label sits `label_dx` x units to the right of the curve value there.
`label_dx`, `label_size`	Label offset (x units) and size (points).
`reference_x`	Optional x of a dashed vertical reference line.
`reference_linewidth`	Its width in points.
`x_limits`, `y_limits`	Axis limits (automatic with 5% margins when `NULL`).
`x_breaks`, `y_breaks`	Optional tick positions; by default chosen as by the automatic locator of common figure code with at most `x_nbins` and `y_nbins` intervals.
`x_nbins`, `y_nbins`	Maximum numbers of tick intervals.
`x_label`, `y_label`	Axis titles.

`tick_size`	Tick label size in points.
`note`	Optional template (see `FormatTemplate()`) filled from the first row of `data`, drawn at `note_xy` (panel fractions).
`note_xy`, `note_hjust`, `note_vjust`, `note_size`, `note_colour`	Note placement and style.
`facet`	Optional column defining small multiples.
`facet_order`	Order of the facets (row by row).
`facet_ncol`	Number of columns.
`facet_gap_mm`	Gaps between neighbouring panels in millimetres (horizontal, vertical), measured between the data areas.
`facet_data`	Optional data frame with one row per facet (column `facet`) used for titles.
`facet_title`, `facet_note`	Templates written above each panel: a bold title coloured by `facet_group` and a smaller muted note to its right.
`facet_group`	Column of `facet_data` whose value selects the title colour from `group_colours`.
`group_colours`	Named colours per group.
`facet_title_size`, `facet_note_size`	Sizes in points.
`box_mm`	Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).
`style`	A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(series = rep(c("a", "b"), each = 4), x = rep(c(0, 10, 20, 30), 2),
           median = c(0, 40, 70, 90, 0, 20, 45, 70), lower = c(0, 30, 60, 85, 0, 10, 35, 60),
                upper = c(0, 50, 80, 95, 0, 30, 55, 80), n = 5)
PlotQuantileCurves(d, colours = c(a = "#7B2FBE", b = "#3B7DD8"), reference_x = 20,
                  labels = c(a = "Series a", b = "Series b"), label_at = 10,
                  note = "Median and IQR\nn = {n|d} units")
```

PlotScatterGroups *Scatter or Dot Rows Coloured by Group*

Description

Points coloured by the group of each unit, with optional open symbols for flagged units, labels with offsets in points, lines connecting the points of a unit across conditions (paired values), median bars and categorical axes. A categorical y axis gives one row per unit (dot rows), with tick labels coloured by group, separators between groups and a "not estimable" text where the value is missing.

Usage

```
PlotScatterGroups(
  data,
  x = "x",
  y = "y",
  group = "group",
  group_colours = NULL,
  id = NULL,
  x_order = NULL,
  x_labels = NULL,
  y_order = NULL,
  y_labels = NULL,
  colour_ticks = TRUE,
  open = NULL,
  size = 20,
  stroke = 0,
  open_stroke = 0.7,
  connect = FALSE,
  linewidth = 0.65,
  line_alpha = 0.85,
  open_linewidth = NULL,
  open_linetype = "32",
  open_shift = NULL,
  open_shift_default = 0,
  summary = FALSE,
  summary_half = 5.9,
  summary_linewidth = 1.4,
  label = NULL,
  label_only = NULL,
  label_first = FALSE,
  label_offset = c(5, 0),
  label_offsets = NULL,
  label_size = 5.3,
  label_vjust = 0.5,
  label_colour = "group",
  missing_text = "NE",
  missing_size = 5.3,
  separators = FALSE,
  x_limits = NULL,
  y_limits = NULL,
  x_breaks = NULL,
  y_breaks = NULL,
  x_nbins = 6,
  y_nbins = 9,
  log_y = FALSE,
  x_label = NULL,
  y_label = NULL,
  grid = c("y", "x", "both", "none"),
```

```
  tick_size = NULL,
  title_size = NULL,
  stats = NULL,
  note = NULL,
  note_xy = c(0.97, 0.96),
  note_hjust = 1,
  note_vjust = 1,
  note_size = 4.8,
  note_colour = "muted",
  reference = NULL,
  reference_x = "x",
  reference_group = NULL,
  reference_linewidth = 0.5,
  reference_linetype = "32",
  ties = FALSE,
  tie_dodge = 0.16,
  tie_label_dx = c(0.12, 0.08),
  tie_label_size = 4.3,
  tie_label_colour = "muted",
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Tidy data.
`x`, `y`	Columns. Character columns are categorical.
`group`	Group column.
`group_colours`	Named colours per group.
`id`	Unit column (for connections, labels and offsets).
`x_order`, `x_labels`	Order and tick labels of a categorical x.
`y_order`, `y_labels`	Order (top to bottom) and tick labels of a categorical y; `y_labels = FALSE` hides them.
`colour_ticks`	Colour categorical y tick labels by group.
`open`	Optional logical column: open symbols (and dashed connections) when `TRUE`.
`size`	Point area (pt^2).
`stroke`	Edge width of filled points (pt); open symbols use `open_stroke`.
`open_stroke`	Edge of open symbols.
`connect`	Connect the points of each `id` across x.
`linewidth`, `line_alpha`	Connection style (points).
`open_linewidth`, `open_linetype`	Width and dash pattern of connections of open units (default: `linewidth`, dashed).

`open_shift` Horizontal shift of open units (x units); a named list gives shifts per `id` overriding `open_shift_default`.

`open_shift_default` Shift for open units without an entry.

`summary` Draw a median bar per x over units that are not open.

`summary_half`, `summary_linewidth` Half-width (points) and width of median bars.

`label` Optional label column; labels of points outside the axes are not drawn.

`label_only` Optional logical column: label only these rows.

`label_first` Label only the first point (lowest x) of each unit.

`label_offset` Default label offset in points (`c(dx, dy)`).

`label_offsets` Named list of offsets per `id`.

`label_size` Label size; `label_colour "group"` uses group colours.

`label_vjust` Vertical justification of labels at their offset (0.2 is close to baseline alignment).

`label_colour` Label colour.

`missing_text` Text for missing x values on a categorical y axis.

`missing_size` Its size.

`separators` Draw lines between rows of different groups (categorical y).

`x_limits`, `y_limits`, `x_breaks`, `y_breaks`, `x_nbins`, `y_nbins` Axes.

`log_y` Logarithmic y axis (values at or below zero are not drawn).

`x_label`, `y_label` Axis titles.

`grid` `"y"`, `"x"`, `"both"` or `"none"`.

`tick_size` Tick label size, or `c(x, y)`.

`title_size` Axis title size, or `c(x, y)`.

`stats` Optional one-row data frame for `note` (default: first row of `data`).

`note` Optional template (see `FormatTemplate()`) written at `note_xy` (panel fractions), for example a correlation coefficient.

`note_xy`, `note_hjust`, `note_vjust`, `note_size`, `note_colour` Note placement and style.

`reference` Optional data frame of vertical reference lines (for example the value chosen in each analysis), one per row.

`reference_x` Column of `reference` with the x positions.

`reference_group` Optional column of `reference` whose values take their colour from `group_colours`.

`reference_linewidth`, `reference_linetype` Reference line width (points) and dash pattern.

`ties` Draw units at the same position side by side (`tie_dodge` x units apart) and write their `labels` once, joined by commas, right of the group at `tie_label_dx[1] + tie_label_dx[2] * n` x units.

`tie_dodge`, `tie_label_dx`, `tie_label_size`, `tie_label_colour`
: Layout and style of tied units and their labels.

`box_mm`
: Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).

`style`
: A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(id = paste0("u", 1:5), group = c("A", "A", "B", "B", "B"),
                x = c(0.1, 0.4, 0.2, 0.8, 0.5), y = c(1, 3, 2, 5, 4))
PlotScatterGroups(d, id = "id", label = "id", group_colours = c(A = "#009E73", B = "#0072B2"))
```

PlotStackedBars *Stacked Bars*

Description

One bar per category (for example per patient), stacked by a second variable (cell types, neighbourhoods, clinical groups) in a fixed order, with thin white separators between the segments. Bars can run upwards or to the right. Category labels are drawn as rotated figure tick labels and can be coloured (for example by clinical group); a short text above each bar (for example the tissue area) and a horizontal reference line can be added. A single stack level gives plain bars.

Usage

```
PlotStackedBars(
  data,
  x = "x",
  stack = "stack",
  value = "value",
  x_order = NULL,
  x_order_by = NULL,
  stacks = NULL,
  colours = NULL,
  fill_column = NULL,
  scale = 1,
  width = 0.82,
  edge_colour = "white",
  edge_width = 0.6,
  horizontal = FALSE,
  limits = NULL,
  breaks = NULL,
  break_labels = NULL,
```

```
  category_limits = NULL,
  value_label = NULL,
  value_label_size = 5.8,
  value_label_angle = NULL,
  tick_size = 5.5,
  tick_length = 2,
  category_label = NULL,
  category_label_size = 5.5,
  show_categories = TRUE,
  x_labels = NULL,
  x_label_colours = NULL,
  category_angle = 50,
  category_size = 5.1,
  category_ticks = TRUE,
  axes = TRUE,
  top_text = NULL,
  top_format = "%.0f",
  top_at = 101.5,
  top_size = 4.4,
  top_colour = "muted",
  top_offset = NULL,
  nbins = NULL,
  value_expand = 1.05,
  top_title = NULL,
  top_title_at = -0.6,
  top_title_hjust = 1,
  reference = NULL,
  legend = NULL,
  texts = NULL,
  values = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Long data: one row per category and stack level.
`x`, `stack`, `value`	Column names: category, stack level and segment size.
`x_order`	Category order: character vector or data frame (ordered by a `rank` column if present); default first appearance.
`x_order_by`	Optional numeric column giving the category order.
`stacks`	Stack order from the baseline outwards; default first appearance.
`colours`	Named colours per stack level.
`fill_column`	Optional column with explicit segment colours.
`scale`	Factor applied to `value` (for example 100 for fractions).
`width`	Bar width in category units.
`edge_colour`, `edge_width`	Segment outline (points; 0 for none).

`horizontal` Bars running to the right, first category on top.

`limits`, `breaks`, `break_labels`
: Value axis limits, breaks and labels; default limits from zero to 5% beyond the tallest bar.

`category_limits`
: Category axis limits; default half a category beyond the outer bars (5% of the bar extent for horizontal bars).

`value_label`, `value_label_size`, `value_label_angle`
: Value axis title.

`tick_size`, `tick_length`
: Value axis tick label size and tick length.

`category_label`, `category_label_size`
: Category axis title.

`show_categories`, `x_labels`, `x_label_colours`
: Category labels: shown or not, display mapping and named text colours.

`category_angle`, `category_size`
: Category label rotation and size.

`category_ticks` Draw tick marks at the categories.

`axes` Draw the value and category axis lines.

`top_text` Optional column with one value per category, printed above the bar (right of it for horizontal bars) with `top_format` at `top_at` (value units).

`top_format`, `top_at`, `top_size`, `top_colour`
: Format and placement.

`top_offset` Optional offset (value units) of the top values above each bar instead of the fixed position `top_at`.

`nbins` Optional maximum number of value-axis intervals; default breaks then follow `NiceBreaks()` instead of `pretty()`.

`value_expand` Default upper limit as a multiple of the tallest bar.

`top_title` Optional text left (or right) of the top values, at `top_title_at` category units from the first bar centre (above or below them for horizontal bars).

`top_title_at`, `top_title_hjust`
: Placement of `top_title`.

`reference` Optional reference line: list of `value`, `colour`, `width` (points) and `linetype`.

`legend` Optional legend of the stack colours, as the `legend` of `PlotEmbedding()` (use `shapes "s"` for squares).

`texts`, `values` Free texts and template values, as in `PlotAnnotatedHeatmap()`.

`style` A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
d <- data.frame(x = rep(c("p1", "p2"), each = 3), stack = rep(c("a", "b", "c"), 2),
                value = c(0.2, 0.5, 0.3, 0.6, 0.1, 0.3), area = rep(c(6.2, 118), each = 3))
PlotStackedBars(d, scale = 100, limits = c(0, 100),
                colours = c(a = "#e34948", b = "#2a78d6", c = "#D9D9D9"),
                top_text = "area", top_title = "Area (mm2)", value_label = "Cells (%)")
```

PlotTerritories *Cell Territory Map*

Description

Draws Voronoi territories from `TerritoryTiles()` filled by class, with thin white borders, a frame and a scale bar; boxes mark sub-fields.

Usage

```
PlotTerritories(
  data,
  colours,
  border_colour = "white",
  border_width = 0.12,
 scale_bar = list(length = 100, label = "100 um", y = 0.07, linewidth = 1.4, size = 4.8),
  frame = TRUE,
  frame_colour = "frame",
  frame_width = NULL,
  boxes = NULL,
  box_colour = "ink",
  box_width = 0.9,
  box_size = NULL,
  title = NULL,
  title_size = 5.6,
  style = ManuscriptStyle()
)
```

Arguments

`data`	A `TerritoryTiles()` result.
`colours`	Named colours by class.
`border_colour`, `border_width`	Tile border colour and width (points).
`scale_bar`, `frame`, `frame_colour`, `frame_width`, `boxes`, `box_colour`, `box_width`, `box_size`	As in `PlotCellMap()`; `boxes` may carry a `colour` column.
`title`	Optional title centred above the map.
`title_size`	Title size (points).
`style`	A `ManuscriptStyle()` object.

Value

A 'grid' grob.

Examples

```
if (requireNamespace("deldir", quietly = TRUE)) {
  set.seed(2)
  d <- data.frame(x = runif(300, 0, 100), y = runif(300, 0, 100),
                  label = sample(c("t", "a"), 300, TRUE))
  tt <- TerritoryTiles(d, c(50, 50), 40, cap = 5, classes = list(tumour = "t"))
  g <- PlotTerritories(tt, colours = list(tumour = "#F4B6C2", other = "#DDDDDD"))
  grid::grid.newpage(); grid::grid.draw(g)
}
```

PlotTextBlock *Text Block or Text Table*

Description

A panel of text placed in its figure box: lines written from templates and data (statistic lines, headings that contain numbers, notes) or, with `layout = "table"`, a small table of text cells as produced by `PlotDataTableCells()`.

Usage

```
PlotTextBlock(
  data = NULL,
  template = NULL,
  text = NULL,
  size = NULL,
  colour = "muted",
  bold = FALSE,
  hjust = 0,
  vjust = 0,
  x = NULL,
  y = NULL,
  lineheight = 1.1,
  angle = 0,
  layout = c("lines", "table"),
  widths = NULL,
  header_fill = "#F3F3F3",
  highlight_fill = "#FFF2D6",
  border_colour = "#D9D9D9",
  border_linewidth = 0.4,
  key_colours = NULL,
  table_size = 5.5,
  first_pad = 0.02,
```

```
  borders = c("cells", "rules"),
  align = c("centre", "left"),
  style = ManuscriptStyle()
)
```

Arguments

`data`	Optional data frame.
`template`	Optional templates.
`text`	Optional fixed lines (placed before the template lines).
`size`	Font size(s) in points; default the style's statistic size.
`colour`	Colour(s); style names allowed.
`bold`	Bold face per line.
`hjust`, `vjust`	Justification of the block.
`x`, `y`	Anchor in fractions of the box (default `hjust`, `vjust`).
`lineheight`	Line height multiplier.
`angle`	Rotation in degrees.
`layout`	`"lines"` or `"table"`.
`widths`	Relative column widths of a table (default: first column 0.23, others equal).
`header_fill`, `highlight_fill`, `border_colour`	Table colours.
`border_linewidth`	Cell border width in points.
`key_colours`	Named colours for header text by `key`.
`table_size`	Table font size.
`first_pad`	Left padding of first-column text (fraction of width).
`borders`	`"cells"` (a frame around every cell, with fills) or `"rules"` (only horizontal rules between rows, no fills).
`align`	Alignment of the columns after the first: `"centre"` or `"left"` (at `first_pad` from the column edge).
`style`	A `ManuscriptStyle()` object.

Details

Lines: `text` gives fixed lines; `template` is filled from `data` with `FormatTemplate()` (one template per data row, or one template for every row). `size`, `colour` and `bold` may be given per line. The block is anchored at `x`, `y` (fractions of the box) with `hjust`, `vjust`.

Table: cells with `row`, `column`, `label`, `header`, `highlight` and `key` columns; the first column is left aligned, the others centred; header cells are bold on `header_fill` and coloured by `key` through `key_colours`; highlighted cells use `highlight_fill`.

Value

A 'grid' grob.

Examples

```
g <- PlotTextBlock(data.frame(k = 11, n = 12, p = 0.0015),
                   template = "{k|d}/{n|d} patients; p = {p|p}")
grid::grid.newpage(); grid::grid.draw(g)
```

`PlotVolcano` *Volcano Plot of Exact Group Tests*

Description

Effect size against minus log10 of the exact P value for one family of tests: open grey symbols for P at or above `alpha`, filled symbols below it, labels for the `top` smallest nominal P values, a grey line at P = `alpha` and a dashed line at the lower bound of the exact P value (from `stats`, see `PlotDataContrastSummary()`).

Usage

```
PlotVolcano(
  data,
  stats = NULL,
  effect = "cliffs_delta",
  p = "p",
  label = NULL,
  alpha = 0.05,
  top = 3,
  bound = "p_bound",
  bound_label = "Minimum attainable p = {p_bound|p}",
  bound_colour = "minimum_p",
  open_size = 8,
  open_edge = 0.6,
  open_colour = "open_edge",
  filled_size = 11,
  filled_colour = "ink",
  label_size = 4.3,
  bound_label_size = 4.4,
  x_limits = c(-1.08, 1.08),
  y_limits = c(0, 3.2),
  x_nbins = 6,
  y_nbins = 9,
  x_label = NULL,
  y_label = NULL,
  show_y_label = TRUE,
  box_mm = NULL,
  style = ManuscriptStyle()
)
```

Arguments

`data`	Test table, one row per endpoint.
`stats`	Optional one-row table with the bound column.
`effect`, `p`	Columns of the effect size and the P value.
`label`	Optional column with display labels of the endpoints.
`alpha`	Nominal threshold.
`top`	Number of labelled endpoints.
`bound`	Column of `stats` with the lower bound of P.
`bound_label`	Template written at the bound line.
`bound_colour`	Colour of the bound line and text.
`open_size`, `open_edge`, `open_colour`	Open symbols (area, edge, colour).
`filled_size`, `filled_colour`	Filled symbols (area; drawn with a 1 pt edge of the same colour as in common figure code).
`label_size`, `bound_label_size`	Text sizes.
`x_limits`, `y_limits`	Axis limits.
`x_nbins`, `y_nbins`	Tick intervals.
`x_label`	Axis title; by default "Cliff's δ" (drawn with the Greek letter). An explicit `NULL` draws no title.
`y_label`	Axis title; `NULL` gives a typeset "-log10 exact P".
`show_y_label`	Draw the y axis title.
`box_mm`	Size of the data area `c(width, height)` in millimetres, passed by `RenderFigure()`; used by `"auto"` tick intervals (see `NiceBreaks()`).
`style`	A `ManuscriptStyle()` object.

Value

A 'ggplot2' object.

Examples

```
t <- data.frame(d = c(-0.9, 0.2, 0.7, -0.1), p = c(0.02, 0.6, 0.04, 0.9),
                name = c("e1", "e2", "e3", "e4"))
p <- PlotVolcano(t, stats = data.frame(p_bound = 0.0048), effect = "d", label = "name")
```

ProjectSpatialTable *Project Spatial Tables Without Changing Column Metadata*

Description

Selects explicit rows and columns from a plain data frame or data.table. Row order and repetitions follow `rows`; column order follows `columns`. Selection uses a copied data.table inside the package namespace, preserving column storage types, classes and attributes according to data.table subset semantics. The result is a plain data frame with reset row names. No eligibility, missing-value filtering, sorting or numerical transformation is inferred. The supplied input is not modified.

Usage

```
ProjectSpatialTable(data, rows, columns)
```

Arguments

`data`	A plain data.frame or data.table with distinct, nonempty column names. Other data-frame subclasses are not accepted.
`rows`	A dimensionless, unclassed integer vector of positive row indices no greater than the number of input rows. Missing indices are not allowed. `integer()` selects zero rows; repeated and shuffled indices are retained.
`columns`	A dimensionless, unclassed character vector of distinct, nonempty existing column names, in the requested order.

Value

A plain data.frame with the requested rows and columns and reset row names. A one-column selection remains a data frame; an empty row selection retains typed columns and their metadata. Invalid inputs are rejected rather than coerced, recycled or partially matched.

Examples

```
d <- data.frame(id = letters[1:3], weight = c(0.5, 1.5, 2.5))
ProjectSpatialTable(d, c(3L, 1L, 3L), c("weight", "id"))
ProjectSpatialTable(d, integer(), "weight")
```

PtLinewidth *Convert Points to a 'ggplot2' Line Width*

Description

'ggplot2' draws a line width of 1 as 0.75 mm (R's line width unit is 1/96 inch); this converts a width given in typographic points.

Usage

```
PtLinewidth(pt)
```

Arguments

pt	Numeric line width in points.

Value

Numeric 'ggplot2' linewidth.

Examples

```
PtLinewidth(0.5)
```

---

qc_filter *Quality Control Filter*

Description

Removes cells that fail quality control criteria based on marker intensity ranges, cell area, or custom filters.

Usage

```
qc_filter(
  dt,
  min_area = NULL,
  max_area = NULL,
  min_intensity = 0,
  max_intensity = NULL
)
```

Arguments

dt	A data.table as returned by read_spatial().
min_area	Numeric or NULL. Minimum cell area. Cells below this threshold, or with a missing area, are removed.
max_area	Numeric or NULL. Maximum cell area. Cells above this threshold are removed.
min_intensity	Numeric. Minimum total marker intensity. Cells with total intensity below this value are removed. Default 0.
max_intensity	Numeric or NULL. Maximum total marker intensity.

Value

A filtered data.table.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:10,
  x = runif(10), y = runif(10),
  cell_area = c(5, seq(50, 200, length.out = 8), 5000),
  CD3 = rnorm(10, 300, 50)
)
filtered <- qc_filter(dt, min_area = 10, max_area = 1000)
nrow(filtered)
```

QCFilter *Quality Control Filter (SpatialCellData)*

Description

Removes cells that fail quality control criteria based on cell area and/or total marker intensity.

Usage

```
QCFilter(
  object,
  min_area = NULL,
  max_area = NULL,
  min_intensity = 0,
  max_intensity = NULL
)
```

Arguments

`object`	An `SpatialCellData-class` object.
`min_area`	Numeric or `NULL`. Minimum cell area; cells with a missing area fail area thresholds.
`max_area`	Numeric or `NULL`. Maximum cell area.
`min_intensity`	Numeric. Minimum total marker intensity. Default `0`.
`max_intensity`	Numeric or `NULL`. Maximum total marker intensity.

Value

A filtered `SpatialCellData-class` object.

Examples

```
counts <- matrix(rnorm(60, 5), nrow = 20,
                 dimnames = list(NULL, c("CD3", "CD8", "CD20")))
coords <- data.frame(x = runif(20), y = runif(20))
meta <- data.frame(cell_id = 1:20, sample_id = "s1",
                   cell_area = c(5, seq(50, 200, length.out = 18), 5000))
obj <- CreateSpatialObject(counts, coords, meta)
filtered <- QCFilter(obj, min_area = 10, max_area = 1000)
NCells(filtered)
```

QCPlot *QC Scatter Plot*

Description

Scatter plot of two QC metrics (e.g. cell area vs total intensity).

Usage

```
QCPlot(object, x, y, colour_by = NULL, pt_size = 0.5)
```

Arguments

object	An SpatialCellData object.
x	Character. Metadata column for x-axis.
y	Character. Metadata column for y-axis.
colour_by	Character or NULL. Metadata column for colour.
pt_size	Numeric. Default 0.5.

Value

A ggplot object.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50), y = runif(50))
meta <- data.frame(cell_id = 1:50, sample_id = "s1",
                   cell_area = runif(50, 10, 500))
obj <- CreateSpatialObject(counts, coords, meta)
QCPlot(obj, x = "cell_area", y = "cell_id")
```

QuadratAnalysis *Quadrat Analysis*

Description

Divides the tissue area into a grid and counts cells per quadrat. Tests for Complete Spatial Randomness using a chi-squared test.

Usage

```
QuadratAnalysis(object, nx = 5L, ny = 5L, target = NULL, by_sample = FALSE)
```

Arguments

object	An SpatialCellData-class object.
nx	Integer. Number of columns in the grid. Default 5.
ny	Integer. Number of rows in the grid. Default 5.
target	Character or NULL. Restrict to a phenotype.
by_sample	Logical. If TRUE and the object holds several samples, the statistic is computed per sample and returned as a named list. Default FALSE; otherwise a single sample is required.

Value

A list with counts, chi_sq, p_value, and VMR (variance-to-mean ratio).

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 100), y = runif(50, 0, 100))
obj <- CreateSpatialObject(counts, coords)
QuadratAnalysis(obj, nx = 3, ny = 3)
```

read_spatial *Read Single-Cell Spatial Biology Data*

Description

Reads cell segmentation CSV files from spatial biology imaging platforms or image analysis software. The function auto-detects the column naming convention, delimiter, and BOM encoding.

Usage

```
read_spatial(
  path,
  sample_id = NULL,
  markers = NULL,
  compartment = "Cell",
  statistic = "mean",
  sep = "auto"
)
```

Arguments

path	Character string. Path to a CSV file or a directory containing CSV files. When a directory is given, all .csv files are read and combined.
sample_id	Character string or NULL. An identifier appended to each row. When path is a directory and sample_id is NULL, the file name (without extension) is used. For QuPath full-format files with an Image column, the scan identity is parsed automatically. This argument fills absent or blank identities; it does not overwrite a nonblank QuPath Image-derived identity. The original Image label is retained as source_image. Distinct Image labels that collapse to the same parsed identity are refused, including across files. Reserved cell/sample/Image identifier columns are read as text, retaining leading zeros; measurement and coordinate columns retain numeric inference.
markers	Character vector or NULL. If provided, only these marker columns are retained. Matching is case-insensitive.
compartment	Character. For QuPath full-format data, which compartment to extract intensities from. One of "Cell" (default), "Nucleus", or "Cytoplasm".
statistic	Character. For QuPath full-format data, which summary statistic to extract. One of "mean" (default), "sum", "std dev", "max", "min", or "range".
sep	Character or "auto". Column delimiter. Default "auto" detects comma vs. semicolon from the first line.

Details

Three input formats are supported:

QuPath Full Contains Object ID, Classification, multi-compartment marker intensities (Compartment: Marker statistic).

QuPath Minimal Contains only spatial coordinates and Marker: Cell: Mean columns.

Flat Format Contains Cell ID, Cell X Position, Cell Y Position, and flat marker intensity columns.

Value

A data.table with standardised column names:

sample_id Sample identifier.

cell_id Unique cell identifier (per sample).

x Cell centroid x-coordinate.

y Cell centroid y-coordinate.

cell_area Cell area (if available).

classification Cell classification label (if available).

source_image Original QuPath Image label, when supplied.

Additional columns contain marker intensities (one per marker).

Examples

```
tmp <- tempfile(fileext = ".csv")
write.csv(data.frame(
  `Cell ID` = 1:5,
  `Cell X Position` = runif(5, 0, 1000),
  `Cell Y Position` = runif(5, 0, 1000),
  `Cell Area (px)` = runif(5, 50, 200),
  DAPI = rnorm(5, 500, 100),
  CD3 = rnorm(5, 300, 80),
  check.names = FALSE
), tmp, row.names = FALSE)
dat <- read_spatial(tmp)
head(dat)
unlink(tmp)
```

`ReadCosMx` *Read NanoString CosMx Cell Output*

Description

Convenience wrapper for CosMx: the `*_exprMat_file.csv` (cell-by-gene, with `fov`/`cell_ID` columns) and `*_metadata_file.csv` (with `CenterX_global_px`/`CenterY_global_px`). CosMx numbers cells within each FOV, so cells are matched on `fov` and `cell_ID`; the `cell_ID` becomes `cell_id` and `fov` is kept.

Usage

```
ReadCosMx(expr_file, meta_file, sample_id = "sample1", project = "CosMx")
```

Arguments

`expr_file`	Path/data frame of the expression matrix.
`meta_file`	Path/data frame of the cell metadata.
`sample_id, project`	Passed to `ReadMatrixCoords()`.

Value

A SpatialCellData-class object.

Examples

```
expr_file <- tempfile(fileext = ".csv")
meta_file <- tempfile(fileext = ".csv")
write.csv(data.frame(fov = c(1, 1, 2, 2), cell_ID = c(1, 2, 1, 2),
                     CD3E = c(3, 0, 1, 5), KRT8 = c(0, 9, 2, 0)),
          expr_file, row.names = FALSE)
write.csv(data.frame(fov = c(1, 1, 2, 2), cell_ID = c(1, 2, 1, 2),
                     CenterX_global_px = c(100, 140, 2100, 2160),
                     CenterY_global_px = c(50, 80, 60, 95)),
          meta_file, row.names = FALSE)
obj <- ReadCosMx(expr_file, meta_file)
obj
unlink(c(expr_file, meta_file))
```

ReadFigureSpec *Read a Figure Specification*

Description

Reads one or more JSON specification files (or a directory of them) and merges them as described in RenderFigure().

Usage

```
ReadFigureSpec(paths)
```

Arguments

paths	JSON file paths or one directory.

Value

The merged specification list.

Examples

```
if (requireNamespace("jsonlite", quietly = TRUE)) {
  dir <- tempfile("spec")
  dir.create(dir)
 writeLines('{"height_mm": 50, "panels": [{"box": [15, 10, 45, 30], "grammar": "group_strip",
    "data": "values"}]}', file.path(dir, "1-page.json"))
 writeLines('{"texts": [{"label": "Note", "at": [5, 48]}]}', file.path(dir, "2-texts.json"))
  str(ReadFigureSpec(dir), max.level = 1)
  unlink(dir, recursive = TRUE)
}
```

`ReadMatrixCoords` *Read an Expression Matrix and a Coordinate/Metadata Table*

Description

General-purpose reader for the common spatial layout of one cell-by-feature expression table plus one cell-metadata table holding coordinates. Each argument may be a file path (read with `data.table::fread`), a matrix, or a data frame.

Usage

```
ReadMatrixCoords(
  expression,
  metadata,
  x_col = "x",
  y_col = "y",
  cell_id_col = NULL,
  id_cols = character(0),
  sample_id = "sample1",
  transpose = FALSE,
  project = "SpatialProject"
)
```

Arguments

`expression`	Cells-by-markers expression (matrix, data frame, or CSV path). Non-numeric and `id_cols` columns are dropped.
`metadata`	Cell metadata containing the coordinate columns (data frame or CSV path).
`x_col, y_col`	Character. Coordinate column names in `metadata`; values must be real numeric, finite vectors. Factor codes are never used as coordinate values.
`cell_id_col`	Character or `NULL`. Shared cell-id column; if present in both tables, rows are matched on it, otherwise row order is assumed. Further columns (for example `"fov"`) can follow: those present in both tables are matched as well, so ids need only be unique within them. The first column becomes `cell_id`. Matched ids must be unique. Declared ID columns and reserved cell/sample identifiers in CSV files are read as text before type inference, so leading-zero identities remain distinct.
`id_cols`	Character vector. Non-marker id columns to drop from `expression` (e.g. `"fov"`, `"cell_ID"`).
`sample_id`	Character. Sample identifier. Default `"sample1"`.
`transpose`	Logical. Set `TRUE` if `expression` is markers-by-cells. Default `FALSE`.
`project`	Character. Project name.

Value

A `SpatialCellData-class` object.

Examples

```
expr <- matrix(rpois(40, 5), nrow = 10,
               dimnames = list(NULL, c("CD3", "CD8", "CD20", "PanCK")))
meta <- data.frame(x = runif(10, 0, 100), y = runif(10, 0, 100))
obj <- ReadMatrixCoords(expr, meta)
obj
```

ReadMERSCOPE *Read Vizgen MERSCOPE Cell Output*

Description

Convenience wrapper for MERSCOPE: `cell_by_gene.csv` (cell-by-gene, first column the cell id) and `cell_metadata.csv` (with `center_x`/`center_y`).

Usage

```
ReadMERSCOPE(
  matrix_file,
  metadata_file,
  sample_id = "sample1",
  project = "MERSCOPE"
)
```

Arguments

`matrix_file`	Path/data frame of the cell-by-gene matrix.
`metadata_file`	Path/data frame of the cell metadata.
`sample_id, project`	Passed to `ReadMatrixCoords()`.

Value

A `SpatialCellData-class` object.

Examples

```
matrix_file <- tempfile(fileext = ".csv")
metadata_file <- tempfile(fileext = ".csv")
write.csv(data.frame(cell = c(101, 102, 103), CD3E = c(2, 0, 6), PTPRC = c(5, 1, 8)),
          matrix_file, row.names = FALSE)
write.csv(data.frame(cell = c(103, 101, 102), center_x = c(15, 210, 88),
                     center_y = c(40, 12, 160)),
          metadata_file, row.names = FALSE)
obj <- ReadMERSCOPE(matrix_file, metadata_file)
obj
unlink(c(matrix_file, metadata_file))
```

ReadPlotTable *Read a Plot Data Table*

Description

Reads a tab-separated (optionally gzipped), comma-separated or RDS table as a plain data frame with the column names unchanged, using one thread. Empty strings and NA are missing values.

Usage

```
ReadPlotTable(path)
```

Arguments

path	File path.

Value

A data frame (or the RDS object).

Examples

```
f <- tempfile(fileext = ".tsv")
writeLines(c("patient\tvalue", "p1\t2.5"), f)
ReadPlotTable(f)
```

ReadSpatial *Read Cell Segmentation Data into a SpatialCellData Object*

Description

Reads cell segmentation CSV files and returns a SpatialCellData-class object. This is the recommended high-level entry point.

Usage

```
ReadSpatial(
  path,
  sample_id = NULL,
  markers = NULL,
  filter = "DAPI|Blank|Empty",
  compartment = "Cell",
  statistic = "mean",
  sep = "auto",
  project = "SpatialProject"
)
```

Arguments

path	Character. Path to a CSV file or a directory containing CSV files.
sample_id	Character or NULL. Fallback sample identifier; nonblank QuPath Image identities are retained as in read_spatial().
markers	Character vector or NULL. If provided, only these marker columns are retained.
filter	Character or NA. Regex pattern for marker names to exclude. Default "DAPI|Blank|Empty".
compartment	Character. Compartment for intensity extraction (QuPath full format). Default "Cell".
statistic	Character. Summary statistic to extract (QuPath full format). Default "mean".
sep	Character or "auto". Column delimiter. Default "auto".
project	Character. Project name. Default "SpatialProject".

Value

A SpatialCellData-class object.

Examples

```
tmp <- tempfile(fileext = ".csv")
write.csv(data.frame(
  `Cell ID` = 1:5,
  `Cell X Position` = runif(5, 0, 1000),
  `Cell Y Position` = runif(5, 0, 1000),
  CD3 = rnorm(5, 300, 80),
  CD8 = rnorm(5, 200, 60),
  check.names = FALSE
), tmp, row.names = FALSE)
obj <- ReadSpatial(tmp, filter = NA)
obj
unlink(tmp)
```

ReadXenium *Read 10x Xenium Cell Output*

Description

Convenience wrapper for Xenium: a cells table with x_centroid/y_centroid coordinates plus a cell-by-gene expression matrix (load the feature matrix yourself, e.g. via **Matrix**, and pass it here).

Usage

```
ReadXenium(expression, cells, sample_id = "sample1", project = "Xenium")
```

Arguments

expression	Cells-by-genes matrix / data frame / CSV path.
cells	The Xenium cells.csv(.gz) metadata (path or data frame).
sample_id, project	Passed to ReadMatrixCoords().

Value

A SpatialCellData-class object.

Examples

```
cells <- tempfile(fileext = ".csv")
write.csv(data.frame(cell_id = c("aaa-1", "aab-1", "aac-1"),
                     x_centroid = c(10.5, 42, 77.25), y_centroid = c(5, 18.5, 33)),
          cells, row.names = FALSE)
counts <- data.frame(cell_id = c("aac-1", "aaa-1", "aab-1"),
                     CD3E = c(0, 4, 1), MS4A1 = c(7, 0, 2))
obj <- ReadXenium(expression = counts, cells = cells)
obj
unlink(cells)
```

Reductions *List Available Reductions*

Description

List Available Reductions

Usage

```
Reductions(object)

## S4 method for signature 'SpatialCellData'
Reductions(object)
```

Arguments

object	A SpatialCellData-class object.

Value

Character vector of reduction names.

Examples

```
data(phenoscapR_example)
Reductions(RunPCA(phenoscapR_example, n_pcs = 5))
```

RenderFigure *Render a Figure from a Specification*

Description

Builds every panel of a figure from a declarative specification and named plot data, then composes the page. A specification is a list (or a JSON file) with `height_mm`, optional `width_mm`, `panels`, `texts` and `legends`. Each panel has a `box` (`[left, top, width, height]` in mm), optional `letter`, `letter_at`, `heading` and `note`, a `grammar` from `FigureGrammars()`, the name(s) of its `data` and builder `args`.

Usage

```
RenderFigure(spec, data = list(), style = ManuscriptStyle())
```

Arguments

`spec`	A specification list, a JSON path, several JSON paths or a directory of JSON files.
`data`	Named list of data frames (or other inputs) referred to by the panels.
`style`	A `ManuscriptStyle()` object; `spec$style` entries are applied on top.

Details

Positions can also be given as fractions of the page. The ordered fields are `left`, `bottom`, `width` and `height`. A panel `axes` field replaces `box`, `letter_xy` replaces `letter_at`, and a text or legend `xy` replaces `at` (fractions measured from the bottom-left corner; letters and texts hang from the given point).

A specification may be split over several files: give a character vector of JSON paths (or a directory of `*.json` files). Page fields are taken from the first file that has them; `panels`, `texts` and `legends` are concatenated in file order.

Colours in `args`, `texts` and `legends` may be hex values, style colour names (`"ink"`, `"muted"`, ...) or palette entries written `"palette/name"` (for example `"groups/group1"`); character vectors and named lists of such values are resolved element by element. Every builder argument whose name ends in `colour`, `color`, `fill` or `palette` (or their plurals) is resolved this way, so arguments that name data columns must not end like that. The value `"group"` is kept as it is, for builders that colour by group (`label_colour = "group"`).

Builders with a `box_mm` argument receive the size of their data area in millimetres (used for `"auto"` tick intervals, see `NiceBreaks()`).

Typesetting (`TypesetLabel()`) can be switched off with `"typeset": false` for the whole specification, for one panel (its heading, note and plot), for one text or for one legend; `"typeset": {"off": ["treg"]}` switches off single rules only (see `TypesetLabel()`).

Legends take `labels`, `colours`, `shape`, `ncol`, `size` and the layout options of `LegendKeys()` (`key_mm`, `marker_size`, `row_mm`, `text_gap_mm`, `column_gap_mm`, `byrow`). Texts take `label`, `xy` or `at`, `size`, `colour`, `bold`, `hjust`, `vjust` (a number or `"baseline"`) and `angle`.

Value

A ComposeFigure() result.

Examples

```
d <- data.frame(group = rep(c("A", "B"), each = 3), value = c(1, 2, 3, 2, 3, 5))
spec <- list(height_mm = 50, width_mm = 70, panels = list(
  list(box = c(15, 10, 45, 30), letter = "a", heading = "Values", grammar = "group_strip",
       data = "values",
       args = list(group_colours = list(A = "groups/group1", B = "groups/group2")))))
fig <- RenderFigure(spec, list(values = d))
grid::grid.newpage(); print(fig)
```

RidgePlot *Ridge Plot of Marker Expression*

Description

Ridge (joy) plot showing the distribution of marker expression for each group. Implemented with overlapping density plots.

Usage

```
RidgePlot(
  object,
  features,
  group_by = "phenotype",
  slot = "data",
  colours = NULL,
  ncol = NULL
)
```

Arguments

object	An SpatialCellData object.
features	Character vector. Marker names to plot.
group_by	Character. Metadata column to group by. Default "phenotype".
slot	Character. "data" or "counts".
colours	Named character vector or NULL.
ncol	Integer or NULL.

Value

A ggplot object.

Examples

```
counts <- matrix(c(rnorm(30, 5), rnorm(30, 1)), ncol = 2,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(30), y = runif(30))
meta <- data.frame(cell_id = 1:30, sample_id = "s1",
                   phenotype = rep(c("T cell", "Other"), each = 15))
obj <- CreateSpatialObject(counts, coords, meta)
RidgePlot(obj, features = "CD3")
```

RipleysK *Ripley's K Function*

Description

Computes Ripley's K function to assess spatial clustering or dispersion at multiple scales.

Usage

```
RipleysK(
  object,
  r_seq = NULL,
  target = NULL,
  correction = c("none", "border", "translation"),
  by_sample = FALSE
)
```

Arguments

object	An SpatialCellData-class object.
r_seq	Numeric vector of radii, or NULL for automatic.
target	Character or NULL. Restrict to a phenotype.
correction	Character. Edge correction: "none" (default), "border" (reduced-sample), or "translation" (each pair weighted by the inverse window/translate overlap; rigorous for rectangular windows).
by_sample	Logical. If TRUE and the object holds several samples, the statistic is computed per sample and returned as a named list (one entry per sample). Default FALSE; otherwise a single sample is required.

Value

A data frame with columns r, K, L, and expected.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 200), y = runif(50, 0, 200))
obj <- CreateSpatialObject(counts, coords)
RipleysK(obj)
```

`run_app` *Launch the phenoscapR Shiny Application*

Description

Opens an interactive, Hugo Coder-themed Shiny interface to the package's analysis functions using the bundled synthetic teaching data: run quality control, spatial statistics, cellular neighbourhoods, spatial domains, dimensionality reductions, and differential abundance, and explore the results with live plots and tables. Simulated phenotypes are supplied with the example. This app does not import user objects or export results; use the package functions for those workflows. Each example section is one sample, so differential-abundance p-values are not suitable for replicated group inference.

Usage

```
run_app(...)
```

Arguments

`...`	Additional arguments passed to `runApp` (e.g. `launch.browser`, `port`, `host`).

Details

The app ships in `inst/app`. It depends on optional packages (Shiny, bslib, sass, thematic, reactable, bsicons); `run_app()` checks for them and reports any that are missing. Branding requires bslib >= 0.9.0. The optional UMAP view requires uwot; PCA remains available without it. If QC removes every cell, widen or reset the cell-area filter.

Value

Called for its side effect of launching the app; returns the value of `runApp` invisibly.

Examples

```
if (interactive()) {
  run_app()
}
```

RunPCA *Principal Component Analysis*

Description

Computes a PCA embedding of the marker-expression matrix and stores it in the `reductions` slot under the name `"pca"`. PCA uses only base R and is always available; it is also the default input space for `RunUMAP`, `RunTSNE`, and `RunSONG`.

Usage

```
RunPCA(object, n_pcs = 30L, slot = "data", markers = NULL)
```

Arguments

`object`	A `SpatialCellData-class` object.
`n_pcs`	Integer. Number of principal components to keep. Capped at the rank of the data. Default `30`.
`slot`	Character. `"data"` (default) or `"counts"`.
`markers`	Character vector or `NULL`. Markers to use; `NULL` uses all.

Value

The object with a `"pca"` entry in its `reductions` slot.

Examples

```
counts <- matrix(rnorm(500), nrow = 50,
                 dimnames = list(NULL, paste0("M", 1:10)))
coords <- data.frame(x = runif(50), y = runif(50))
obj <- CreateSpatialObject(counts, coords)
obj <- RunPCA(obj, n_pcs = 5)
Embeddings(obj, "pca")[1:3, ]
```

RunSONG *SONG Embedding*

Description

Computes a 2-D SONG (Self-Organising Nebulous Growths) embedding via the **songR** package and stores it under `"song"`. SONG is incremental, noise-robust, and preserves global structure. By default the embedding is computed on the top principal components (see `use_pca`).

Usage

```
RunSONG(
  object,
  dims = 30L,
  use_pca = TRUE,
  slot = "data",
  markers = NULL,
  seed = NULL,
  ...
)
```

Arguments

object	A SpatialCellData-class object.
dims	Integer. Number of PCs (or markers) to embed. Default 30.
use_pca	Logical. Embed on PCA space (default TRUE) or directly on the marker matrix.
slot	Character. "data" (default) or "counts".
markers	Character vector or NULL.
seed	Integer or NULL. Random seed for reproducibility. The random number stream of the caller is restored afterwards.
...	Passed to songR::song().

Details

The optional **songR** package (https://github.com/CTTIR/songR) is available from the CTTIR R-universe; install it with `install.packages("songR", repos = c("https://cttir.r-universe.dev", getOption("repos")))`. This function reports a missing dependency without installing software or changing the input object.

Value

The object with a "song" entry in its reductions slot.

Examples

```
## Not run:
if (requireNamespace("songR", quietly = TRUE)) {
  counts <- matrix(rnorm(500), nrow = 50,
                   dimnames = list(NULL, paste0("M", 1:10)))
  coords <- data.frame(x = runif(50), y = runif(50))
  obj <- CreateSpatialObject(counts, coords)
  obj <- RunSONG(obj, seed = 1, verbose = FALSE)
}

## End(Not run)
```

RunTSNE *t-SNE Embedding*

Description

Computes a 2-D t-SNE embedding via the **Rtsne** package and stores it under `"tsne"`. By default the embedding is computed on the top principal components (see `use_pca`).

Usage

```
RunTSNE(
  object,
  dims = 30L,
  use_pca = TRUE,
  perplexity = 30,
  slot = "data",
  markers = NULL,
  seed = NULL,
  ...
)
```

Arguments

`object`	A `SpatialCellData-class` object.
`dims`	Integer. Number of PCs (or markers) to embed. Default `30`.
`use_pca`	Logical. Embed on PCA space (default `TRUE`) or directly on the marker matrix.
`perplexity`	Numeric. t-SNE perplexity. Automatically capped at `(n - 1) / 3`. Default `30`.
`slot`	Character. `"data"` (default) or `"counts"`.
`markers`	Character vector or `NULL`.
`seed`	Integer or `NULL`. Random seed for reproducibility. The random number stream of the caller is restored afterwards.
`...`	Passed to `Rtsne::Rtsne()`.

Value

The object with a `"tsne"` entry in its `reductions` slot.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE)) {
  counts <- matrix(rnorm(500), nrow = 50,
                   dimnames = list(NULL, paste0("M", 1:10)))
  coords <- data.frame(x = runif(50), y = runif(50))
  obj <- CreateSpatialObject(counts, coords)
  obj <- RunTSNE(obj, seed = 1)
}
```

RunUMAP *UMAP Embedding*

Description

Computes a 2-D UMAP embedding via the **uwot** package and stores it under `"umap"`. By default the embedding is computed on the top principal components (see `use_pca`).

Usage

```
RunUMAP(
  object,
  dims = 30L,
  use_pca = TRUE,
  n_neighbors = 15L,
  densmap = FALSE,
  slot = "data",
  markers = NULL,
  seed = NULL,
  n_threads = 1L,
  pca_method = "svd",
  ...
)
```

Arguments

`object`	A `SpatialCellData-class` object.
`dims`	Integer. Number of PCs (or markers) to embed. Default `30`.
`use_pca`	Logical. Embed on PCA space (default `TRUE`) or directly on the marker matrix.
`n_neighbors`	Integer. UMAP neighbourhood size. Default `15`.
`densmap`	Logical. Preserve local densities with the approximation of densMAP in **uwot** (`dens_scale = 1` in `uwot::umap()`) instead of plain UMAP. Default `FALSE`. For partial density preservation, pass `dens_scale` through `...` instead.
`slot`	Character. `"data"` (default) or `"counts"`.
`markers`	Character vector or `NULL`.
`seed`	Integer or `NULL`. Random seed for reproducibility. The random number stream of the caller is restored afterwards.
`n_threads`	Integer. Number of worker threads for UMAP neighbour search. Defaults to `1`; increase explicitly for larger analyses.
`pca_method`	Character. PCA solver used by **uwot** for initialisation or optional dimensionality reduction. Defaults to `"svd"`, using base R for the small marker/PC panels typical of spatial data. Pass `"auto"` or `"irlba"` to use an optional truncated solver for larger panels.
`...`	Passed to `uwot::umap()`.

Value

The object with a "umap" entry in its reductions slot.

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  counts <- matrix(rnorm(500), nrow = 50,
                   dimnames = list(NULL, paste0("M", 1:10)))
  coords <- data.frame(x = runif(50), y = runif(50))
  obj <- CreateSpatialObject(counts, coords)
  obj <- RunUMAP(obj, seed = 1)
}
```

SaveFigure *Save a Composed Figure as PDF and PNG*

Description

Writes the page at its exact size: PDF through the Cairo device with embedded TrueType fonts, PNG through 'ragg' at the style's resolution (falling back to the 'png' Cairo device when 'ragg' is absent).

Usage

```
SaveFigure(figure, pdf = NULL, png = NULL, dpi = NULL)
```

Arguments

figure	A ComposeFigure() result.
pdf	Output PDF path, or NULL.
png	Output PNG path, or NULL.
dpi	Positive finite raster resolution; defaults to the style's.

Details

With 'ragg', the page is drawn on a new capture device for every save and the captured pixels are written by ragg::agg_png(). 'gridtext' keeps rich text metrics for the session under the device name, not its resolution, so a fresh device name makes the PNG independent of what was drawn before in the session. Without 'ragg', PNGs saved at different resolutions in one session can place rich text a pixel apart.

Both outputs are rendered to temporary files beside their destinations before replacing either destination. A rendering failure leaves existing files unchanged. If replacement fails, previous files are restored where the file system permits it; any backup that cannot be restored is retained and its path is reported. This is recovery from reported errors, not a transaction across process termination or a concurrent writer. Destinations must be distinct files in existing directories.

Value

The output paths, invisibly.

Examples

```
p <- ggplot2::ggplot(data.frame(x = 1:3, y = c(2, 1, 3)), ggplot2::aes(x, y)) +
  ggplot2::geom_point()
fig <- ComposeFigure(list(FigurePanel(p, c(10, 8, 50, 30), letter = "a")),
                     height_mm = 45, width_mm = 70)
if (capabilities("cairo")) {
  out <- SaveFigure(fig, pdf = tempfile(fileext = ".pdf"),
                    png = tempfile(fileext = ".png"), dpi = 72)
  file.exists(out)
}
```

SchematicShapes *Schematic Shapes from a Specification*

Description

Builds the shape table drawn by `PlotImage()` for a schematic: fixed shapes (rectangles, rounded boxes, ellipses, circles, segments, arrows and text) and, optionally, clouds of dots placed at random inside ellipses (`seed` makes them reproducible).

Usage

```
SchematicShapes(
  shapes = list(),
  clouds = list(),
  flow = NULL,
  values = NULL,
  fill = list(),
  xlim = c(0, 1),
  ylim = c(0, 1),
  seed = 1,
  equal = TRUE
)
```

Arguments

`shapes`	List of shapes; each a list with `type` (`"rect"`, `"roundrect"`, `"ellipse"`, `"circle"`, `"segment"`, `"arrow"`, `"text"`), position fields (`x`, `y`, and `w`, `h`, `r`, `xend`, `yend`, `angle` as needed), `fill`, `colour`, `linewidth` (points), `label`, `size` (points), `hjust`, `vjust`, `z` (drawing order).
`clouds`	List of dot clouds: `n`, `x`, `y` (ellipse centre), `rx`, `ry` (ellipse radii), `r` (dot radius), `fill`, optional `colour` and `linewidth` (outline), `z`.

flow	Optional list describing a row of boxes joined by arrows: labels, x0, width, gap, y, height, fill, colour, text_size, arrow_colour.
values	Optional data frame used to fill {name} placeholders in labels; see fill.
fill	Named list: placeholder to a list with column, fun ("sum", "mean", "median", "n"), scale, format (sprintf) and an optional filter (named list of column to values).
xlim, ylim	Drawing limits.
seed	Random seed for the clouds.
equal	Keep equal x and y scales (centred in the panel box); with FALSE the limits fill the box.

Value

A list of class phenoscapR_shapes with shapes, xlim, ylim and equal. Rounded-box radii (r of "roundrect") are millimetres.

Examples

```
s <- SchematicShapes(
  shapes = list(list(type = "roundrect", x = 0.5, y = 0.6, w = 0.6, h = 0.3,
                     fill = "#EEEEEE", colour = "#999999", r = 1),
                list(type = "text", x = 0.5, y = 0.6, label = "{n} cells")),
 clouds = list(list(n = 20, x = 0.5, y = 0.2, rx = 0.3, ry = 0.1, r = 0.01, fill = "#7B2FBE")),
  values = data.frame(count = c(120, 80)),
  fill = list(n = list(column = "count", fun = "sum", format = "%.0f"))
)
head(s$shapes)
```

ScreePlot *Scree Plot of PCA Variance*

Description

Bar-and-line plot of the percent variance explained by each principal component – the standard tool for choosing how many PCs to retain.

Usage

```
ScreePlot(object, n_pcs = NULL)
```

Arguments

object	A SpatialCellData-class object.
n_pcs	Integer or NULL. Number of PCs to show. Default all.

Value

A ggplot object.

Examples

```
counts <- matrix(rnorm(500), nrow = 50,
                 dimnames = list(NULL, paste0("M", 1:10)))
coords <- data.frame(x = runif(50), y = runif(50))
obj <- RunPCA(CreateSpatialObject(counts, coords), n_pcs = 8)
ScreePlot(obj)
```

SetPalette *Set or Get the Default Colour Palette*

Description

Sets a global colour palette used by all plotting functions in phenoscapR. The palette can be a viridis option, a custom gradient defined by 2-3 anchor colours, or a named vector of discrete colours.

Usage

```
SetPalette(palette)

GetPalette()
```

Arguments

palette	Character or named character vector. Either: • A viridis-family palette name: "viridis" (default), "inferno", "plasma", "cividis", "rocket", or "mako". • A character vector of 2-3 hex colours for a custom gradient (e.g. c("#440154", "#21918c", "#fde725")). • NULL to return the current palette without changing it. An unrecognised name falls back to "viridis" rather than erroring.

Value

Invisibly returns the previous palette setting.

Examples

```
SetPalette("inferno")
GetPalette()

# Custom 3-colour gradient
SetPalette(c("navy", "white", "firebrick"))
GetPalette()

# Reset to default
SetPalette("viridis")
```

---

show,SpatialCellData-method *Show a SpatialCellData Object*

Description

Show a SpatialCellData Object

Usage

```
## S4 method for signature 'SpatialCellData'
show(object)
```

Arguments

object	A SpatialCellData object.

Value

Invisibly NULL; called for printing a summary of the object.

---

spatial_clusters *Spatial Cell Clustering*

Description

Clusters cells based on their spatial coordinates using k-means or hierarchical clustering.

Usage

```
spatial_clusters(dt, k, method = c("kmeans", "hierarchical"))
```

Arguments

dt	A data.table with columns x and y.
k	Integer. Number of clusters.
method	Character string. Clustering method: "kmeans" (default) or "hierarchical".

Value

The input data.table with an added cluster column.

Examples

```
dt <- data.table::data.table(
  sample_id = "s1", cell_id = 1:50,
  x = c(rnorm(25, 0, 5), rnorm(25, 50, 5)),
  y = c(rnorm(25, 0, 5), rnorm(25, 50, 5))
)
result <- spatial_clusters(dt, k = 2)
table(result$cluster)
```

SpatialCellData-class *The SpatialCellData Class*

Description

Central S4 class for storing and analysing single-cell spatial biology data. Inspired by the Seurat object design, it holds raw and normalised marker intensities, spatial coordinates, cell-level metadata, and results from spatial analyses in a single container.

Details

Raw and normalized matrices must have identical dimensions and row/marker names. They contain real numeric values; NA and NaN are allowed, infinity is not. Coordinates are finite real numeric vectors, and coordinate/metadata row counts must agree even for empty objects.

Slots

counts Numeric matrix of raw marker intensities (cells x markers).

data Numeric matrix of normalised marker intensities (cells x markers). Identical to counts until NormaliseData is called.

coords Data frame with columns x and y holding spatial coordinates for each cell.

meta_data Data frame of per-cell metadata. Always contains cell_id and sample_id. Additional columns such as cell_area, phenotype, or cluster are added by analysis functions.

project Character string. Project or experiment name.

spatial List for storing spatial analysis results (nearest-neighbour distances, density values, interaction matrices).

`reductions` Named list of dimensionality-reduction embeddings. Each element is a numeric matrix with one row per cell (e.g. `"pca"`, `"umap"`, `"tsne"`, `"song"`), as produced by `RunPCA`, `RunUMAP`, `RunTSNE`, or `RunSONG`.

`SpatialClusters` *Spatial Clusters (SpatialCellData)*

Description

Clusters cells based on spatial coordinates.

Usage

```
SpatialClusters(object, k, method = c("kmeans", "hierarchical"))
```

Arguments

`object`	An `SpatialCellData-class` object.
`k`	Integer. Number of clusters.
`method`	Character. `"kmeans"` (default) or `"hierarchical"`.

Value

An `SpatialCellData-class` with a `cluster` column in `meta_data`.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(
  x = c(rnorm(25, 0, 5), rnorm(25, 50, 5)),
  y = c(rnorm(25, 0, 5), rnorm(25, 50, 5)))
obj <- CreateSpatialObject(counts, coords)
obj <- SpatialClusters(obj, k = 2)
table(Meta(obj)$cluster)
```

SpatialDomains *Spatial Domains*

Description

Partitions the tissue into spatial domains (regions of coherent marker expression) by clustering cells on their spatially smoothed expression: each cell's profile is averaged with its k nearest neighbours before clustering, so neighbouring cells tend to share a domain.

Usage

```
SpatialDomains(
  object,
  n_domains = 6L,
  k = 20L,
  slot = "data",
  markers = NULL,
  seed = NULL
)
```

Arguments

object	A SpatialCellData-class object.
n_domains	Integer. Number of domains. Default 6.
k	Integer. Number of neighbours to smooth over. Default 20.
slot	Character. "data" (default) or "counts".
markers	Character vector or NULL. Markers to use; NULL uses all.
seed	Integer or NULL. Random seed for k-means.

Value

The object with a domain column in meta_data.

Examples

```
data(phenoscapR_example)
obj <- NormaliseData(phenoscapR_example, "zscore")
obj <- SpatialDomains(obj, n_domains = 4, k = 15, seed = 1)
table(obj$domain)
```

SpatialNetworkPlot *Plot Spatial Network*

Description

Plots cells as circles connected by edges from Delaunay triangulation or nearest-neighbour graph, coloured by phenotype. Matches the network visualisation style with coloured nodes on dark background.

Usage

```
SpatialNetworkPlot(
  object,
  edges = NULL,
  colour_by = "phenotype",
  colours = NULL,
  pt_size = 2,
  edge_alpha = 0.15,
  edge_colour = "grey60",
  dark_theme = TRUE
)
```

Arguments

object	An SpatialCellData object.
edges	Data frame with columns from and to (row indices), as returned by DelaunayNetwork or FindNeighbours. If NULL and the spatial slot contains delaunay_edges, those are used.
colour_by	Character. Metadata column for node colour. Default "phenotype".
colours	Named character vector or NULL.
pt_size	Numeric. Node size. Default 2.
edge_alpha	Numeric. Edge transparency. Default 0.15.
edge_colour	Character. Edge colour. Default "grey60".
dark_theme	Logical. Default TRUE.

Value

A ggplot object.

Examples

```
counts <- matrix(rnorm(100), nrow = 50,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(50, 0, 500), y = runif(50, 0, 500))
meta <- data.frame(cell_id = 1:50, sample_id = "s1",
                   phenotype = sample(c("A", "B"), 50, TRUE))
```

```
obj <- CreateSpatialObject(counts, coords, meta)
obj <- DelaunayNetwork(obj)
SpatialNetworkPlot(obj)
```

`summarise_phenotypes` *Summarise Phenotype Proportions*

Description

Computes phenotype frequencies and proportions per sample.

Usage

```
summarise_phenotypes(dt)
```

Arguments

`dt`	A `data.table` with a `phenotype` column, as returned by `phenotype_cells()`.

Value

A `data.table` with columns `sample_id`, `phenotype`, `count`, and `proportion`.

Examples

```
dt <- data.table::data.table(
  sample_id = rep("s1", 100),
  phenotype = sample(c("CD3+", "CD8+", "Negative"), 100, replace = TRUE)
)
summarise_phenotypes(dt)
```

`SummarizeCellComposition` *Cell-Type Composition, Densities and Declared State Fractions of a Section*

Description

Counts cells per registered type, expresses them as fractions of all cells and as densities per unit area, and adds declared subset fractions such as "marker-positive cells among cells of some types". Fractions are computed from exact counts with `SummarizeGroupedCountFractions()`.

Usage

```
SummarizeCellComposition(
  cells,
  type,
  levels,
  area = NULL,
  area_divisor = 1,
  fractions = NULL,
  percent = TRUE,
  prefix = c(fraction = "fraction|", density = "density|"),
  totals = c(cells = "cells", area = "area")
)
```

Arguments

cells	Cell table (one section).
type	Cell-type column; every value must be in levels.
levels	Character vector: the complete type registry (zero counts are reported).
area	Optional column summed over all cells into the section area; adds density rows.
area_divisor	Divisor applied to the summed area.
fractions	Optional named list of subset fractions. Each element is a list with numerator (types), denominator (types) and optional flag (name of a logical or 0/1 column that numerator cells must have set).
percent	Express fractions in percent.
prefix	Named character vector with prefixes of the fraction and density endpoint names (followed by the type).
totals	Named character vector: endpoint names of the cells and area totals.

Value

A data frame with endpoint, measure ("fraction", "density", "subset_fraction" or "total"), type (for per-type rows), numerator, denominator and value.

Examples

```
cells <- data.frame(type = c("t", "t", "a", "b", "t"), pos = c(1, 0, 0, 1, 1),
                    area = c(10, 12, 9, 11, 8))
SummarizeCellComposition(cells, "type", c("t", "a", "b", "c"), area = "area",
  fractions = list("pos % t" = list(numerator = "t", denominator = "t",
                                    flag = "pos")))
```

`SummarizeComparisons` *Summarise Comparison Results into Pass/Fail Checks*

Description

Condenses outputs of `CompareTables()` or `CompareArrays()` (one or a list of them) into one row per check: rows compared, rows whose values all match, values compared and matching, the largest difference and tolerance, and `PASS` when every value matches and the declared number of rows (if any) was compared.

Usage

```
SummarizeComparisons(
  comparisons,
  unit = "row_key",
  expected_rows = NULL,
  by = "check"
)
```

Arguments

`comparisons`	A comparison data frame or a list of them. When an element lacks the first by column, its list name is used.
`unit`	Columns that identify one compared row (default `"row_key"`).
`expected_rows`	Optional named list or vector: value of the first by column (the check label) to the number of rows that must be compared in each summary row of that check.
`by`	Columns defining one summary row (default `"check"`), for example `c("check", "branch")`. Missing columns are treated as `NA`.

Value

A data frame with the by columns, `rows`, `matching_rows`, `values`, `matching_values`, `max_difference`, `tolerance`, `expected_rows` and `status`.

Examples

```
a <- data.frame(id = c("x", "y"), p = c(0.1, 0.2))
cmp <- CompareTables(a, a, "id", "p", check = "self")
SummarizeComparisons(cmp, expected_rows = c(self = 2))
```

`SummarizeFixedGrid` *Summarize Cell Annotations Within Fixed Grids*

Description

Joins annotations by sample and cell key, never by row position or coordinates. Counts only supplied memberships, retaining each sample separately. Unobserved grid bins and unobserved categories are not manufactured as zeros. Additional annotation rows outside the membership are allowed and ignored.

Usage

```
SummarizeFixedGrid(
  membership,
  annotations,
  settings,
  categories = character(),
  indicators = character()
)
```

Arguments

`membership`	Data frame returned by `AssignFixedGrid()`. Each cell occurs at most once per setting and sample. Bin geometry and sample frames must be internally consistent. No coordinate transformation is performed.
`annotations`	Data frame with unique character `sample_id`, `cell_id` keys and requested category/indicator columns. Every membership must match.
`settings`	The explicit grid settings supplied to `AssignFixedGrid()`. Size and all bounds must exactly match these settings and grid indices.
`categories`	Character vector of unique category column names. Values must be character vectors; NA is permitted, but empty values are rejected.
`indicators`	Character vector of unique logical indicator column names. NA is permitted. Names must be disjoint from categories and key columns.

Details

Logical indicators retain counts of TRUE, FALSE and unavailable (NA). Their fraction is returned only for complete bins: any unavailable value makes the fraction NA. Category NA is an explicit unavailable bucket, whose fraction is the fraction of memberships with unknown category, not biological absence. These are cell fractions, not tissue densities or patient-level replicates. Supplied annotations do not establish biological identity or review authority.

Value

List of data frames: `bins` with geometry and `n_cells`, `categories` with bin keys, category column, value, count, fraction and known/unavailable status, and `indicators` with bin keys, indicator column, TRUE/FALSE/NA counts, fraction and COMPLETE/PARTIAL/UNAVAILABLE status. Results have deterministic ordering. Empty inputs return empty typed tables.

Examples

```
cells <- data.frame(cell_id = c("a", "b"), sample_id = "s",
  frame_id = "physical", x = c(1, 2), y = 0)
settings <- data.frame(setting_id = "g", size_um = 10,
  offset_x_um = 0, offset_y_um = 0)
frames <- data.frame(sample_id = "s", frame_id = "physical", unit = "um")
membership <- AssignFixedGrid(cells, settings, frames)
annotations <- data.frame(sample_id = "s", cell_id = c("a", "b"),
  type = c("A", "B"), signal = c(TRUE, NA))
SummarizeFixedGrid(membership, annotations, settings, "type", "signal")
```

SummarizeGroupedCountFractions

Summarize Grouped Subset Counts as Fractions

Description

Computes sum(numerator) / sum(denominator) from supplied subset counts, without averaging observation fractions or manufacturing absent groups. Eligibility and the independent sampling unit must be chosen upstream. IDs must be unique within the complete group key.

Usage

```
SummarizeGroupedCountFractions(data, groups, id, numerator, denominator)
```

Arguments

`data`	Data frame of observation keys and subset counts.
`groups`	Nonempty character vector of distinct group column names.
`id`	Name of the observation ID column. Group and ID keys must be dimensionless nonblank character vectors without NA or ASCII 28 separators.
`numerator`	Name of the subset-count column, allowing NA.
`denominator`	Name of the total-count column, allowing NA.

Details

Counts must be plain integer/double vectors (names are permitted), with no classes or dimensions. Known counts must be finite nonnegative whole numbers at most 2^53 - 1. Numerators cannot exceed their corresponding denominators where both are known. NaN is rejected; NA is explicitly unavailable. Both known-count totals are checked incrementally against 2^53 - 1 before each addition, even when missingness will suppress the reported totals.

Missingness is always propagated: any missing count suppresses both totals and the fraction. n_available counts rows with both counts known, including observed zero counts. A zero denominator total leaves the fraction unavailable; a zero numerator and positive denominator gives an observed zero fraction.

Value

Data frame sorted by group keys with integer n_supplied, n_available and n_unavailable; double numerator_total, denominator_total and fraction; character status and missing_policy. Status precedence is UNAVAILABLE (no complete count pairs), PARTIAL (some missing counts), ZERO_DENOMINATOR, COMPLETE. missing_policy is always "propagate". Empty input returns a typed empty table. Count totals outside the exact integer range fail. Fractions use ordinary double division after exact count accumulation.

Examples

```
d <- data.frame(sample = c("one", "one"), bin = c("a", "b"),
                positive = c(1, 2), total = c(3, 7))
SummarizeGroupedCountFractions(d, "sample", "bin", "positive", "total")
```

SummarizeGroupedValues

Summarize Continuous Observations in Explicit Groups

Description

Computes arithmetic means, medians and optional threshold fractions or weighted means over supplied observations. No unobserved groups or zeros are manufactured. Observation IDs must be unique within the complete group key. Include the independent sampling unit in `groups` when required; this function does not establish independence, eligibility, review authority or tissue area.

Usage

```
SummarizeGroupedValues(
  data,
  groups,
  id,
  value,
  weight = NULL,
  threshold = NULL,
  min_available = 1L,
  missing = c("propagate", "available")
)
```

Arguments

`data`	Data frame containing group/observation keys and numeric values.
`groups`	Nonempty character vector of distinct group columns. Their values must be dimensionless, nonblank character vectors without NA.
`id`	Character name of the observation ID column. IDs must be dimensionless, nonblank character values without NA, unique within each complete group key. The reserved ASCII 28 separator is rejected in keys.

value	Character column containing finite numeric values or NA.
weight	Optional character column with finite nonnegative weights or NA.
threshold	Optional finite numeric scalar. Reports the fraction of available values greater than or equal to this fixed threshold.
min_available	Positive integer minimum number of available observations.
missing	Missingness policy: "propagate" or explicit "available".

Details

Missing observations remain counted. The default suppresses summaries when any required value is unavailable. missing = "available" explicitly selects available-case summaries, still labelled PARTIAL. Weighted summaries require both a value and its nonnegative weight, and positive total weight. A weight of zero is an observed zero contribution, not unavailable. min_available applies separately to unweighted and weighted available observations.

Value

Data frame sorted by group keys with supplied/available counts, mean, median, threshold fraction, weighted mean, total available weight, and separate availability statuses and the missingness policy. Status is COMPLETE, PARTIAL, INSUFFICIENT or UNAVAILABLE; optional weighted status also uses NOT_REQUESTED and NO_POSITIVE_WEIGHT. Empty input returns a typed empty table. Numerical overflow or product underflow fails rather than producing a misleading summary. weight_sum counts only pairs with both value and weight available; it is zero for observed zero weights, and NA when policy suppresses a weighted summary or weights were not requested.

Examples

```
d <- data.frame(sample = c("one", "one"), cell = c("a", "b"),
  score = c(1, 3), area = c(1, 3))
SummarizeGroupedValues(d, "sample", "cell", "score", "area", threshold = 2)
```

SummarizeNearestDistances

Nearest-Target Distances Summarised per Cell Set

Description

For each declared source set of cell types, finds the Euclidean distance from every source cell to its nearest target cell (via CrossNNDistance()) and reports the number of source cells, the median distance and the percentage (or fraction) within each radius (inclusive). Optionally adds the summed area of all cells (for example Voronoi territories) as tissue area.

Usage

```
SummarizeNearestDistances(
  cells,
  from,
  to,
  x = "x",
  y = "y",
  type = "type",
  radius = NULL,
  percent = TRUE,
  area = NULL,
  area_divisor = 1
)
```

Arguments

cells	Cell table with coordinate and type columns.
from	Named list (or named character vector) of source sets: output label to cell types. An unnamed character vector uses each type as its own set. Sets must be disjoint from each other and from to.
to	Character vector of target cell types.
x, y, type	Column names in cells.
radius	Optional numeric radii; one output row per set and radius.
percent	Report percentages (TRUE) or fractions.
area	Optional column summed over all cells into area.
area_divisor	Divisor applied to the summed area (e.g. 1e6 for square micrometres to square millimetres).

Value

A data frame with from, n, n_target, median_distance, radius, within and optionally area.

Examples

```
set.seed(1)
cells <- data.frame(x = runif(200, 0, 100), y = runif(200, 0, 100),
                    type = sample(c("t", "a", "b"), 200, replace = TRUE))
SummarizeNearestDistances(cells, from = list(A = "a", B = "b"), to = "t",
                          radius = 10)
```

`TerritoryTiles` *Voronoi Territory Tiles of the Cells in a Field*

Description

Computes the Voronoi tessellation of the cells around a square field and caps every territory at `cap` from its cell by moving the polygon vertices that lie further away onto that radius; cells on the outer hull (unbounded territories) get a regular 12-gon of radius `0.7 * cap`. Each tile carries a class from `classes` (first matching entry), and tiles are ordered for drawing by `order` (classes listed first are drawn first, all others afterwards; ties keep cell order). Requires 'deldir'.

Usage

```
TerritoryTiles(
  cells,
  centre = NULL,
  size,
  cap = 11,
  margin = 30,
  classes = list(),
  default = "other",
  order = c("other", "tumour"),
  centre_cell = NULL,
  id = "row",
  offset = NULL,
  x = "x",
  y = "y",
  label = "label"
)
```

Arguments

`cells`	Data frame of cells.
`centre`	`c(x, y)`, a data frame with `x`, `y`, or `NULL` with `centre_cell`.
`size`	Field width.
`cap`	Maximum territory radius.
`margin`	Neighbourhood kept around the field for the tessellation.
`classes`	Named list: class name to labels; unmatched cells get `default`.
`default`	Class of unmatched cells.
`order`	Classes drawn first, in this order.
`centre_cell`, `id`, `offset`	As in `CellMapData()`.
`x`, `y`, `label`	Column names.

Value

A list of class `phenoscapR_territories` with `tiles` (data frame with `tile`, `x`, `y`, `class`), `centre`, `size` and `extent`.

Examples

```
if (requireNamespace("deldir", quietly = TRUE)) {
  set.seed(2)
  d <- data.frame(x = runif(300, 0, 100), y = runif(300, 0, 100),
                  label = sample(c("t", "a"), 300, TRUE))
  tt <- TerritoryTiles(d, c(50, 50), 40, cap = 5, classes = list(tumour = "t"))
  head(tt$tiles)
}
```

ThemeManuscript *Manuscript 'ggplot2' Theme*

Description

A compact theme for multi-panel figures: no panel title (headings are placed by `ComposeFigure()`), thin grey left and bottom axis lines, short ticks, light horizontal grid lines when `grid = "y"`, no legend background and zero outer margins. When the optional packages 'ggtext' and 'gridtext' are installed, axis titles and tick labels accept the rich text produced by `TypesetLabel()`.

Usage

```
ThemeManuscript(
  style = ManuscriptStyle(),
  grid = c("y", "x", "both", "none"),
  axes = c("xy", "x", "y", "none")
)
```

Arguments

`style`	A `ManuscriptStyle()` object.
`grid`	One of `"y"`, `"x"`, `"both"` or `"none"`.
`axes`	One of `"xy"` (left and bottom lines), `"x"`, `"y"` or `"none"`.

Value

A 'ggplot2' theme.

Examples

```
library(ggplot2)
ggplot(data.frame(x = 1:3, y = c(2, 4, 3)), aes(x, y)) +
  geom_point() +
  ThemeManuscript()
```

TypesetLabel *Typeset Figure Text*

Description

Applies manuscript typesetting to plain labels and returns rich text for 'gridtext'/'ggtext': italic *P*, *q* and *n* before = or <; powers of ten for `4.9e-04`; superscript plus signs on markers (`CD8+`, `FOXP3+`, `CD4 T`, `CD8 T`); `Treg`/`Tregs` with subscript *reg*; `log2`; `mm2`; micrometres; italic gene symbols; and line breaks. Straight quotes stay straight (no typographic apostrophes).

Usage

```
TypesetLabel(
  x,
  genes = c("TP53", "RB1"),
  markers = c("CD\\d+[a-z]?", "FOXP3", "PD-L1", "PD-1", "Ki67", "MPO"),
  rich = .fig_has_rich_text()
)
```

Arguments

`x`	Character vector of labels.
`genes`	Gene symbols to set in italics.
`markers`	Marker names that take a superscript + when followed by +.
`rich`	If `FALSE`, return plain text with Unicode substitutes (superscript plus, subscript two, minus sign) instead of markup. The default uses markup when 'gridtext' and 'ggtext' are installed.

Details

Typesetting can be switched off with `options(phenoscapR.typeset = FALSE)`; labels are then only escaped for rich text. Single rules are switched off by naming them instead, for example `options(phenoscapR.typeset = "treg")`: `"exponent"`, `"p"`, `"q"`, `"n"`, `"markers"`, `"treg"`, `"log2"`, `"mm2"`, `"um"` and `"genes"`. Figure specifications do this per figure, panel, text or legend (`"typeset"`, see `RenderFigure()`).

Value

Character vector of the same length.

Examples

```
TypesetLabel(c("p = 4.9e-04", "Tregs near CD8+ cells", "log2 ratio (mm2)"),
             rich = FALSE)
```

VarianceExplained *Variance Explained by Principal Components*

Description

Returns the percentage of total variance captured by each principal component from a previously computed PCA reduction.

Usage

```
VarianceExplained(object)
```

Arguments

object	A SpatialCellData-class object.

Value

A named numeric vector of percent-variance-explained per PC.

Examples

```
counts <- matrix(rnorm(500), nrow = 50,
                 dimnames = list(NULL, paste0("M", 1:10)))
coords <- data.frame(x = runif(50), y = runif(50))
obj <- RunPCA(CreateSpatialObject(counts, coords), n_pcs = 5)
VarianceExplained(obj)
```

ViolinPlot *Violin Plot of Marker Expression*

Description

Violin plot showing the distribution of marker expression across phenotypes or other grouping variables.

Usage

```
ViolinPlot(
  object,
  features,
  group_by = "phenotype",
  slot = "data",
  colours = NULL,
  ncol = NULL,
  pt_size = 0
)
```

Arguments

object	An SpatialCellData object.
features	Character vector. Marker names to plot.
group_by	Character. Metadata column to group by. Default "phenotype".
slot	Character. "data" or "counts".
colours	Named character vector or NULL.
ncol	Integer or NULL. Number of columns for facets.
pt_size	Numeric. Jitter point size. 0 to hide. Default 0.

Value

A ggplot object.

Examples

```
counts <- matrix(c(rnorm(30, 5), rnorm(30, 1)), ncol = 2,
                 dimnames = list(NULL, c("CD3", "CD8")))
coords <- data.frame(x = runif(30), y = runif(30))
meta <- data.frame(cell_id = 1:30, sample_id = "s1",
                   phenotype = rep(c("T cell", "Other"), each = 15))
obj <- CreateSpatialObject(counts, coords, meta)
ViolinPlot(obj, features = c("CD3", "CD8"))
```

WriteFigure *Write a Figure from a Specification*

Description

File-producing form of RenderFigure() for workflow graphs: reads the specification and the panel data, renders the figure and writes PDF and PNG files.

Usage

```
WriteFigure(spec, data, pdf, png, style = ManuscriptStyle())
```

Arguments

spec	Specification list or JSON path.
data	Named list of data frames, or of paths to .rds, .tsv, .tsv.gz or .csv files.
pdf, png	Output paths.
style	A ManuscriptStyle() object.

Value

Output paths, invisibly.

Examples

```
values <- tempfile(fileext = ".tsv")
write.table(data.frame(group = rep(c("A", "B"), each = 3), value = c(1, 2, 3, 2, 3, 5)),
            values, sep = "\t", row.names = FALSE, quote = FALSE)
spec <- list(height_mm = 50, width_mm = 70, panels = list(
  list(box = c(15, 10, 45, 30), letter = "a", grammar = "group_strip", data = "values")))
if (capabilities("cairo")) {
  WriteFigure(spec, list(values = values), pdf = tempfile(fileext = ".pdf"),
              png = tempfile(fileext = ".png"))
}
```

Index

∗ **datasets**
phenoscapR_example, 102
∗ **figure style**
NiceBreaks, 94
∗ **figure text**
FormatTemplate, 73
∗ **plot-data builders**
PlotDataBindRows, 121
PlotDataConstant, 122
PlotDataCount, 124
PlotDataCrossShares, 125
PlotDataDirection, 126
PlotDataDominant, 127
PlotDataFromArrays, 128
PlotDataJoin, 129
PlotDataKeyColours, 130
PlotDataLabels, 131
PlotDataLonger, 132
PlotDataMembership, 133
PlotDataPairRatio, 134
PlotDataProduct, 135
PlotDataQuantiles, 136
PlotDataRankRows, 137
PlotDataRatio, 138
PlotDataRecode, 139
PlotDataRelative, 140
PlotDataSelect, 141
PlotDataShares, 142
PlotDataSummarise, 143
PlotDataUnite, 146
[,SpatialCellData-method, 6
[[,SpatialCellData-method, 7
$,SpatialCellData-method, 8

AdjustPatientFamilies, 8
AdjustPatientFamilies(), *32*, *41*, *60, 61*
AlterationMatrixData, 9
AnnotateClusters, 10
AnnotationTracks, 11
ApplyGroupedThresholds, 12
as_Seurat, 13
as_SpatialCellData, 14
as_SpatialExperiment, 14
as_SpatialExperiment(), *13*
AssemblePairedSummaries, 15
AssignFixedGrid, 17
AssignFixedGrid(), *211*

BestSubWindow, 18
BootstrapMedianInterval, 19
BoxPlot, 20
BuildExactDelaunayEdges, 21
BuildExactDelaunayEdges(), *93*, *105, 106*

cell_density, 24
cell_density(), *107*
CellDensity, 25, *53*
CellMap, 25
CellMapData, 26
CellMapData(), *66*, *120*, *216*
CellPositionsFor, 28
CellularNeighbourhoods, 29
ClusterLabelPositions, 30
ClusterLabelPositions(), *151*
CompareArrays, 30
CompareArrays(), *210*
CompareTables, 31
CompareTables(), *210*
CompleteRegisteredValues, 32
ComposeFigure, 34
ComposeFigure(), *69*, *84, 85*, *192*, *199*, *217*
CompositeFields, 35
CompositeFields(), *152*, *158, 159*
CompositionPlot, 36
ContrastGroupedOverlap, 37
ContrastGroupedRanks, 39
ContrastPatientEndpoints, 40
Coords, 42
Coords,SpatialCellData-method *(*Coords*)*, 42

CorrelateGroupedRanks, 42
CountDisjointEdges, 43
CountRadiusSets, 45
CreateSpatialObject, 48
CrossNNDistance, 49
CrossNNDistance(), *214*
CustomGradient, 49

dbscan::frNN(), *46*
DelaunayNetwork, 50, *207*
DensestGridWindow, 51
DensityLevelContour, 51
DensityLevelContour(), *146, 147*
DensityPlot, 53
DifferentialAbundance, 54
dim,SpatialCellData-method, 55
DimPlot *(*EmbeddingPlot*)*, 58
DotPlot, 55

EmbeddingLayers, 56
EmbeddingLayers(), *150*
EmbeddingLimits, 57
EmbeddingLimits(), *52*, *151*
EmbeddingPlot, 58
Embeddings, 59
Embeddings,SpatialCellData-method *(*Embeddings*)*, 59
exact graph builder, *94*
ExactPairedTests, 60
ExactPairedTests(), *99*
ExactPatientTests, 61
ExactPatientTests(), *40, 41*
ExactPermutationBound, 63
ExactPermutationBound(), *123*
ExpressionClusters, 63

FeaturePlot, 64
FieldCellsRanked, 65
FieldCentre, 66
FigureGrammars, 67
FigureGrammars(), *191*
FigureGrob, 68
FigureGrob(), *34*, *81*
FigurePanel, 68
FigurePanel(), *34*
FigureText, 69
FigureText(), *34*
FindNeighbours, 70, *207*
FormatP, 71
FormatP(), *73*
FormatStatistic, 72
FormatTemplate, 73
FormatTemplate(), *111*, *113*, *131*, *144*, *149*, *154*, *162*, *166*, *169*, *175*
FromCellspec, 74

GetData, 75
GetData,SpatialCellData-method *(*GetData*)*, 75
GetPalette *(*SetPalette*)*, 202

HistogramPlot, 75

Idents, 76
Idents,SpatialCellData-method *(*Idents*)*, 76
interaction_matrix, 77
interaction_matrix(), *109*
InteractionMatrix, 78, *79*
InteractionPlot, 79

LeaveOnePatientOut, 79
LegendKeys, 81
LegendKeys(), *68*, *191*
LookupNestedRows, 82

make.unique(), *13*
ManuscriptPalettes, 83
ManuscriptPalettes(), *85*
ManuscriptStyle, 84
ManuscriptStyle(), *11*, *34, 35*, *82, 83*, *111*, *113*, *115*, *118*, *121*, *147*, *149*, *151*, *153*, *155*, *157*, *159*, *162, 163*, *166*, *170*, *172, 173*, *175*, *177*, *191*, *217*, *220*
MarkerHeatmap, 85
Markers, 86
Markers,SpatialCellData-method *(*Markers*)*, 86
Meta, 87
Meta,SpatialCellData-method *(*Meta*)*, 87
Mod(), *94*
MoransI, 87

NCells, 88
NCells,SpatialCellData-method *(*NCells*)*, 88
nearest_neighbours, 89
NearestLabelDistance, 90

NearestPartnerPairs, 91
NeighbourhoodEnrichment, 92
NeighbourTypeCounts, 93
NiceBreaks, 94
NiceBreaks(), *115*, *118*, *155*, *157*, *162*, *163*, *166*, *170*, *172*, *177*, *191*
NMarkers, 95
NMarkers,SpatialCellData-method *(*NMarkers*)*, 95
normalise_markers, 96
normalise_markers(), *103*
NormaliseData, 97, *204*

PairCorrelation, 98
PairedDifferences, 99
PairedDifferences(), *61*
PaletteContinuous, 99
PaletteDiscrete, 100
PermutationCorrelation, 101
phenoscapR_example, 102
phenotype_cells, 103
phenotype_cells(), *208*
PhenotypeCells, 103
PhenotypeSummary, 104
PlanExactDelaunayWorkspace, 105
PlanExactDelaunayWorkspace(), *22*
plot_cell_map, 106
plot_density, 107
plot_heatmap, 108
plot_interactions, 108
PlotAlterationMatrix, 109
PlotAnnotatedHeatmap, 111
PlotAnnotatedHeatmap(), *151*, *172*
PlotBandProfiles, 114
PlotBandProfiles(), *138*
PlotBars, 116
PlotCellMap, 119
PlotCellMap(), *26*, *153*, *173*
PlotDataBindColumns *(*PlotDataBindRows*)*, 121
PlotDataBindRows, 121
PlotDataBindRows(), *123*, *125–127*, *129*, *130*, *132–144*, *146*
PlotDataCollapse *(*PlotDataLabels*)*, 131
PlotDataConstant, 122
PlotDataConstant(), *122*, *125–127*, *129*, *130*, *132–144*, *146*
PlotDataContrastSummary, 123
PlotDataContrastSummary(), *176*
PlotDataCount, 124
PlotDataCount(), *122*, *123*, *126*, *127*, *129*, *130*, *132–144*, *146*
PlotDataCrossShares, 125
PlotDataCrossShares(), *122*, *123*, *125–127*, *129*, *130*, *132–144*, *146*
PlotDataDirection, 126
PlotDataDirection(), *122*, *123*, *125–127*, *129*, *130*, *132–144*, *146*
PlotDataDominant, 127
PlotDataDominant(), *122*, *123*, *125*, *126*, *129*, *130*, *132–144*, *146*
PlotDataFromArrays, 128
PlotDataFromArrays(), *122*, *123*, *125–127*, *130*, *132–144*, *146*
PlotDataJoin, 129
PlotDataJoin(), *122*, *123*, *125–127*, *129*, *130*, *132–144*, *146*
PlotDataKeyColours, 130
PlotDataKeyColours(), *122*, *123*, *125–127*, *129*, *130*, *132–144*, *146*
PlotDataLabels, 131
PlotDataLabels(), *122*, *123*, *125–127*, *129*, *130*, *133–144*, *146*
PlotDataLonger, 132
PlotDataLonger(), *122*, *123*, *125–127*, *129*, *130*, *132*, *134–144*, *146*
PlotDataMembership, 133
PlotDataMembership(), *122*, *123*, *125–127*, *129*, *130*, *132*, *133*, *135–144*, *146*
PlotDataPairRatio, 134
PlotDataPairRatio(), *122*, *123*, *125–127*, *129*, *130*, *132–134*, *136–144*, *146*
PlotDataProduct, 135
PlotDataProduct(), *122*, *123*, *125–127*, *129*, *130*, *132–135*, *137–144*, *146*
PlotDataQuantiles, 136
PlotDataQuantiles(), *122*, *123*, *125–127*, *129*, *130*, *132–136*, *138–144*, *146*
PlotDataRankRows, 137
PlotDataRankRows(), *122*, *123*, *125–127*, *129*, *130*, *132–137*, *139–144*, *146*
PlotDataRatio, 138
PlotDataRatio(), *122*, *123*, *125–127*, *129*, *130*, *132–138*, *140–144*, *146*
PlotDataRecode, 139
PlotDataRecode(), *122*, *123*, *125–127*, *129*, *130*, *132–139*, *141–144*, *146*

PlotDataRelative, 140
PlotDataRelative(), *122, 123*, *125–127*, *129, 130*, *132–140*, *142–144*, *146*
PlotDataSelect, 141
PlotDataSelect(), *122, 123*, *125–127*, *129, 130*, *132–141*, *143, 144*, *146*
PlotDataShares, 142
PlotDataShares(), *122, 123*, *125–127*, *129, 130*, *132–142*, *144*, *146*
PlotDataSummarise, 143
PlotDataSummarise(), *122, 123*, *125–127*, *129, 130*, *132–143*, *146*
PlotDataTableCells, 144
PlotDataTableCells(), *174*
PlotDataUnite, 146
PlotDataUnite(), *122, 123*, *125–127*, *129, 130*, *132–144*
PlotDensityOutlines, 146
PlotDotMatrix, 148
PlotEmbedding, 150
PlotEmbedding(), *111*, *147*, *172*
PlotFluorescenceField, 152
PlotGroupDifferences, 153
PlotGroupStrip, 156
PlotImage, 158
PlotImage(), *200*
PlotIntervalForest, 160
PlotPairedPatients, 162
PlotQuantileCurves, 164
PlotQuantileCurves(), *115*
PlotScatterGroups, 166
PlotStackedBars, 170
PlotTerritories, 173
PlotTextBlock, 174
PlotTextBlock(), *144*
PlotVolcano, 176
pretty(), *172*
ProjectSpatialTable, 178
PtLinewidth, 178
PtLinewidth(), *85*

qc_filter, 179
QCFilter, 180
QCPlot, 181
QuadratAnalysis, 182

read_spatial, 182
read_spatial(), *96*, *103*, *179*, *189*
ReadCosMx, 184
ReadFigureSpec, 185
ReadMatrixCoords, 186
ReadMatrixCoords(), *184*, *187*, *190*
ReadMERSCOPE, 187
ReadPlotTable, 188
ReadSpatial, 188
ReadXenium, 189
Reductions, 190
Reductions,SpatialCellData-method *(*Reductions*)*, 190
RenderFigure, 191
RenderFigure(), *67*, *115*, *118*, *155*, *157*, *162, 163*, *166*, *170*, *177*, *185*, *218*, *220*
RidgePlot, 192
RipleysK, 193
run_app, 194
runApp, *194*
RunPCA, 195, *205*
RunSONG, 195, *195*, *205*
RunTSNE, *195*, 197, *205*
RunUMAP, *195*, 198, *205*

SaveFigure, 199
SaveFigure(), *34*
SchematicShapes, 200
SchematicShapes(), *158, 159*
ScreePlot, 201
SetPalette, *100*, 202
show,SpatialCellData-method, 203
spatial_clusters, 203
SpatialCellData, *7, 8*, *20*, *26*, *36*, *42*, *53*, *55, 56*, *65*, *75–77*, *85–87*, *89*, *95*, *102*, *181*, *192*, *203*, *207*, *220*
SpatialCellData-class, 204
SpatialClusters, 205
SpatialDomains, 206
SpatialNetworkPlot, 207
summarise_phenotypes, 208
SummarizeCellComposition, 208
SummarizeComparisons, 210
SummarizeComparisons(), *32*
SummarizeFixedGrid, 211
SummarizeGroupedCountFractions, 212
SummarizeGroupedCountFractions(), *208*
SummarizeGroupedValues, 213
SummarizeNearestDistances, 214

TerritoryTiles, 216
TerritoryTiles(), *173*

ThemeManuscript, 217
ThemeManuscript(), *85*
TypesetLabel, 218
TypesetLabel(), *69–71*, *81*, *191*, *217*

VarianceExplained, 219
ViolinPlot, 219

workspace planner, *22*
WriteFigure, 220