Package 'specmine'

September 8, 2026

Type Package

Title Metabolomics and Spectral Data Analysis and Mining

Version 4.0.0

Depends R (>= 4.0.0)

Imports baseline, caret, compare, ellipse, genefilter, GGally, ggplot2, grDevices, graphics, impute, imputeTS, MASS, Matrix, methods, Metrics, narray, pcaPP, plotly, pls, RColorBrewer, readJDX, stats, utils

Suggests clusterCrit, curl, dbscan, fastICA, ggdendro, KEGGgraph, KEGGREST, knitr, MAIT, mclust, pins, qdap, qpdf, RCurl, cyjShiny, reticulate, rgl, Rtsne, rmarkdown, scatterplot3d, specmine.datasets, uwot, randomForest, xcms

VignetteBuilder knitr

Description Provides methods for metabolomics and spectral data analysis, including data import, preprocessing, visualization, univariate and multivariate analysis, machine learning, feature selection, and pathway analysis. The package supports analytical workflows for different data types used in metabolomics and spectroscopy. Some optional functionality uses the suggested packages 'cyjShiny' and 'specmine.datasets'. The package 'specmine.datasets' is maintained separately at <https://github.com/PedroFontao/specmine.datasets>.

License GPL (>= 2)

URL https://github.com/PedroFontao/specmine

BugReports https://github.com/PedroFontao/specmine/issues

Encoding UTF-8

LazyData true

SystemRequirements Python (>= 3.5.2), Python module 'nmrglue'

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Christopher Costa [aut],
Marcelo Maraschin [aut],
Miguel Rocha [aut],
Sara Cardoso [aut],
Telma Afonso [aut],
Bruno Pereira [aut],
Pedro Fontão [aut, cre],
C. Beleites [cph],
Jie Hao [cph]

Maintainer Pedro Fontão <pedrofontao812004@gmail.com>

Repository CRAN

Date/Publication 2026-09-08 12:30:02 UTC

Contents

`aggregate_samples` *Aggregate samples*

Description

Aggregate samples according to an aggregate function like mean, median, etc. This can be used to merge replicates.

Usage

```
aggregate_samples(dataset, indexes, aggreg.fn = "mean", meta.to.remove = c())
```

Arguments

dataset	List representing the dataset from a metabolomics experiment.
indexes	Index vector with the samples that are going to be aggregated (e.g. c(1,1,2,2), this index vector will aggregate the first two samples and the last two samples).
aggreg.fn	Aggregation function (e.g. "mean", "median", etc).
meta.to.remove	Metadata variables to be removed.

Value

A dataset object in which samples have been aggregated according to ‘indexes‘ and ‘aggreg.fn‘, with updated ‘data‘, ‘metadata‘, and preserved dataset-level fields such as labels, type, and description.

Examples

```
data <- matrix(
  c(1, 2, 10, 12,
    3, 4, 14, 16),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3", "s4"))
)
metadata <- data.frame(
  class = factor(c("A", "A", "B", "B")),
  batch = c(1, 1, 2, 2),
  row.names = c("s1", "s2", "s3", "s4")
)
dataset <- list(
  data = data,
  metadata = metadata,
  labels = list(),
  type = "example",
  description = "toy dataset"
)
aggregate_samples(dataset, c(1, 1, 2, 2), "mean")
```

---

airPLS_fast_dataset *Baseline correction of Raman and SERS spectra using airPLS*

Description

Applies adaptive iteratively reweighted penalized least squares (airPLS) baseline correction to Raman or surface-enhanced Raman spectroscopy (SERS) spectra.

Usage

```
airPLS_fast_dataset(dataset, lambda = 1e+05, max_iter = 50)
```

Arguments

`dataset`	A dataset list containing a numeric matrix in `dataset$data`, with wavenumbers in rows and samples in columns.
`lambda`	Positive numeric smoothing parameter. Larger values produce a smoother estimated baseline.
`max_iter`	Positive integer giving the maximum number of airPLS iterations.

Details

The dataset is expected to contain spectral variables or wavenumbers in rows and samples in columns. Each sample is baseline-corrected independently. Corrected spectra are shifted so that their minimum intensity is zero.

Value

A dataset equivalent to `dataset`, with baseline-corrected values stored in `dataset$data`. Details of the correction are stored in `dataset$baseline`.

References

Zhang, Z.-M., Chen, S., and Liang, Y.-Z. (2010). Baseline correction using adaptive iteratively reweighted penalized least squares. *Analyst*, 135(5), 1138-1146. doi:10.1039/B922045C.

Examples

```
set.seed(123)

dataset = list(
  data = matrix(
    rnorm(100, mean = 10, sd = 2),
    nrow = 10,
    dimnames = list(
      paste0("wavenumber", 1:10),
      paste0("sample", 1:10)
    )
  )
)

corrected = airPLS_fast_dataset(
  dataset,
  lambda = 1e4,
  max_iter = 5
)

range(corrected$data)
```

aov_all_vars *Analysis of variance*

Description

Performs one-way ANOVA for all variables in a dataset.

Usage

```
aov_all_vars(
  dataset,
  column.class,
  doTukey = TRUE,
  write.file = FALSE,
  file.out = NULL
)
```

Arguments

dataset	Dataset to analyze.
column.class	Metadata column used to define groups.
doTukey	Logical. If TRUE, also performs TukeyHSD post-hoc test.
write.file	Logical. If TRUE, writes results to file.
file.out	Output file name.

Value

A data frame with ANOVA results for all variables.

Examples

```
datamatrix <- matrix(
  c(10, 11, 20, 21,
    5,  6,  5,  6),
  nrow = 2,
  byrow = TRUE,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3", "s4"))
)
metadata <- data.frame(
  group = factor(c("A", "A", "B", "B")),
  row.names = c("s1", "s2", "s3", "s4")
)
dataset <- list(data = datamatrix, metadata = metadata)
aov_all_vars(dataset, "group", doTukey = FALSE)
```

aov_one_var *Analysis of variance for one variable*

Description

Performs one-way ANOVA for a single variable in a dataset.

Usage

```
aov_one_var(dataset, x.val, groups, doTukey = TRUE)
```

Arguments

dataset	Dataset to analyze.
x.val	Variable name or x value to test.
groups	Grouping factor.
doTukey	Logical. If TRUE, also performs TukeyHSD post-hoc test.

Value

A list with p-value, -log10(p), FDR, and optional Tukey results.

Examples

```
datamatrix <- matrix(
  c(10, 11, 20, 21,
    5,  6,  5,  6),
  nrow = 2,
  byrow = TRUE,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3", "s4"))
)
metadata <- data.frame(
  group = factor(c("A", "A", "B", "B")),
  row.names = c("s1", "s2", "s3", "s4")
)
dataset <- list(data = datamatrix, metadata = metadata)
aov_one_var(dataset, "x1", metadata$group, doTukey = FALSE)
```

apply_by_group *Apply by group*

Description

Applies a function to the variables of samples belonging to a given group.

Usage

```
apply_by_group(dataset, fn.to.apply, metadata.var, var.value)
```

Arguments

dataset	Dataset to analyze.
fn.to.apply	Function to apply.
metadata.var	Metadata variable used to define the group.
var.value	Value or values of the metadata variable to select.

Value

A named vector or matrix, depending on the output of ‘fn.to.apply‘, containing the values obtained by applying the function to each variable across the samples belonging to the selected group.

Examples

```
data <- matrix(
  c(1, 2, 3, 4, 5, 6),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  group = c("control", "control", "case"),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
apply_by_group(dataset, mean, "group", "control")
```

apply_by_groups *Apply by groups*

Description

Applies a function to groups defined by a metadata variable.

Usage

```
apply_by_groups(
  dataset,
  metadata.var,
  fn.to.apply = "mean",
  variables = NULL,
  variable.bounds = NULL
)
```

Arguments

dataset	Dataset to analyze.
metadata.var	Metadata variable used to define groups.
fn.to.apply	Function to apply.
variables	Variables to include.
variable.bounds	Optional numeric bounds for variables.

Value

A matrix-like object with one row per selected variable and one column per group defined by ‘metadata.var‘. Each cell contains the result of applying ‘fn.to.apply‘ to the values of that variable within the corresponding group.

Examples

```
data <- matrix(
  c(1, 2, 3, 4, 5, 6),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  group = c("control", "control", "case"),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
apply_by_groups(dataset, "group", mean)
```

convert_hmdb_to_kegg *Get kegg codes from hmdb codes:*

Description

Get kegg codes from hmdb codes:

Usage

```
convert_hmdb_to_kegg(hmdb_codes)
```

Arguments

hmdb_codes	Character vector of HMDB identifiers.

Value

A named `character` vector with the KEGG compound identifiers corresponding to the input HMDB codes. The vector values are KEGG compound codes prefixed with `"cpd:"`, and the element names are the matched compound names.

Examples

```
## Not run:
convert_hmdb_to_kegg(c("HMDB0000122"))

## End(Not run)
```

convert_keggpathway_2_reactiongraph *Convert KEGGPathway object to graph object*

Description

Convert KEGGPathway object to graph object

Usage

```
convert_keggpathway_2_reactiongraph(pathObj)
```

Arguments

pathObj	TODO.

Value

A graph object representing the reactions in the input KEGGPathway object. The returned object is suitable for graph-based inspection or for use in downstream visualization functions.

Examples

```
## Not run:
path <- get_MetabolitePath("hsa00010")
graph <- convert_keggpathway_2_reactiongraph(path)
class(graph)

## End(Not run)
```

---

convert_multiple_spcmnm_to_kegg

Get kegg codes from spcmnm codes:

Description

Get kegg codes from spcmnm codes:

Usage

```
convert_multiple_spcmnm_to_kegg(spcmnm_codes)
```

Arguments

spcmnm_codes	Character vector of SPCMNM identifiers.

Value

A named character vector with the KEGG compound identifiers corresponding to the input SPCMNM codes. The vector values are KEGG compound codes prefixed with "cpd:", and the element names are the matched compound names.

Examples

```
## Not run:
convert_multiple_spcmnm_to_kegg(c("SPCM00001"))

## End(Not run)
```

count_missing_values *Count missing values*

Description

Returns the total number of missing values in the dataset.

Usage

```
count_missing_values(dataset)
```

Arguments

dataset	Dataset to inspect.

Value

A single numeric value giving the total count of 'NA' entries present in 'dataset$data'.

Examples

```
data <- matrix(
  c(1, NA, 3, 4),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2"))
)
metadata <- data.frame(class = c("A", "B"), row.names = c("s1", "s2"))
dataset <- list(data = data, metadata = metadata)
count_missing_values(dataset)
```

count_missing_values_per_sample *Count missing values per sample*

Description

Returns the number of missing values per sample.

Usage

```
count_missing_values_per_sample(dataset, remove.zero = TRUE)
```

Arguments

dataset	Dataset to inspect.
remove.zero	If TRUE, removes zero counts.

Value

A named numeric vector giving the number of missing values for each sample in ‘dataset$data‘. If ‘remove.zero‘ is ‘TRUE‘, only samples with at least one missing value are returned.

Examples

```
data <- matrix(
  c(1, NA, 3,
    4,  5, NA),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
count_missing_values_per_sample(dataset)
```

---

count_missing_values_per_variable

Count missing values per variable

Description

Returns the number of missing values per variable.

Usage

```
count_missing_values_per_variable(dataset, remove.zero = TRUE)
```

Arguments

dataset	Dataset to inspect.
remove.zero	If TRUE, removes zero counts.

Value

A named numeric vector giving the number of missing values for each variable in ‘dataset$data‘. If ‘remove.zero‘ is ‘TRUE‘, only variables with at least one missing value are returned.

Examples

```
data <- matrix(
  c(1, 2, NA,
    4, 5, NA),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
```

```
dataset <- list(data = data, metadata = metadata)
count_missing_values_per_variable(dataset)
```

`create_dataset` *Create dataset*

Description

Creates a dataset from a numeric matrix.

Usage

```
create_dataset(
  datamatrix,
  type = "undefined",
  metadata = NULL,
  description = "",
  sample.names = NULL,
  x.axis.values = NULL,
  label.x = NULL,
  label.values = NULL,
  xSet = NULL
)
```

Arguments

`datamatrix`	Matrix with numerical data; rows are variables and columns are samples.
`type`	Type of data, such as "nmr-spectra", "nmr-peaks", "ir-spectra", "uvv-spectra", "concentrations", or "undefined".
`metadata`	Optional metadata as a data frame or matrix.
`description`	Dataset description.
`sample.names`	Optional sample names.
`x.axis.values`	Optional x-axis values.
`label.x`	Optional x-axis label.
`label.values`	Optional value label.
`xSet`	Optional xSet object.

Value

A list representing a specmine dataset. The returned object contains at least the elements `data`, a numeric matrix with variables in rows and samples in columns; `type`, a character string identifying the dataset type; `description`, a character string describing the dataset; `metadata`, a data frame with one row per sample when available; `labels`, a list with axis and value labels when provided; and `xSet`, an optional object associated with LC-MS processing. This object is the standard input structure used by downstream specmine analysis functions.

Examples

```
datamatrix <- matrix(
  c(1.1, 2.2, 3.3, 4.4, 5.5, 6.6),
  nrow = 2,
  dimnames = list(NULL, NULL)
)
metadata <- data.frame(
  class = c("A", "B", "A"),
  row.names = c("s1", "s2", "s3")
)
create_dataset(
  datamatrix,
  type = "concentrations",
  metadata = metadata,
  sample.names = c("s1", "s2", "s3"),
  x.axis.values = c("v1", "v2"),
  label.x = "variable",
  label.values = "intensity"
)
```

---

create_pathway_with_reactions

Creates the pathway, with reactions included in the nodes

Description

Creates the pathway, with reactions included in the nodes

Usage

```
create_pathway_with_reactions(
  path,
  path.name,
  identified_cpds,
  nodeNames = "kegg",
  nodeTooltip = FALSE,
  map.zoom = FALSE,
  map.layout = "preset",
  map.width = NULL,
  map.height = NULL
)
```

Arguments

path	TODO.
path.name	TODO.

identified_cpds
: TODO.

nodeNames
: TODO.

nodeTooltip
: TODO.

map.zoom
: TODO.

map.layout
: TODO.

map.width
: TODO.

map.height
: TODO.

Value

A cyjShiny widget representing the pathway with reactions included in the nodes. The widget contains the pathway graph ready for interactive visualization, with identified compounds highlighted when they are present in the pathway.

Examples

```
## Not run:
path <- get_MetabolitePath("hsa00010")
create_pathway_with_reactions(
  path = path,
  path.name = "hsa00010",
  identified_cpds = c("cpd:C00031", "cpd:C00022")
)

## End(Not run)
```

---

dataset_from_peaks *Create a dataset from peak lists*

Description

Merges equivalent peaks across samples, extracts intensities, and creates a dataset object in standard package format.

Usage

```
dataset_from_peaks(
  sample.list,
  metadata = NULL,
  description = "",
  type = "nmr-peaks"
)
```

Arguments

sample.list	A named list of peak tables, one per sample.
metadata	Optional sample metadata.
description	Character string with a dataset description.
type	Character string describing the dataset type.

Value

A dataset object created from the input peak lists.

Examples

```
sample.list <- list(
  s1 = data.frame(ppm = c(1.0, 2.0), int = c(10, 20)),
  s2 = data.frame(ppm = c(1.0, 3.0), int = c(15, 30))
)
metadata <- data.frame(class = c("A", "B"), row.names = c("s1", "s2"))
dataset <- dataset_from_peaks(sample.list, metadata = metadata)
class(dataset)
```

---

filter_feature_selection

Feature Selection Using Univariate Filters

Description

Performs feature selection using univariate filters.

Usage

```
filter_feature_selection(
  datamat,
  samples.class,
  functions = caret::rfSBF,
  method = "cv",
  repeats = 5
)
```

Arguments

datamat	Data matrix with features in rows and samples in columns.
samples.class	Sample class labels.
functions	Caret SBF functions list.
method	Resampling method passed to `caret::sbfControl()`.
repeats	Number of repeats for resampling.

Value

An sbf object.

Examples

```
datamat <- matrix(
  rnorm(20),
  nrow = 4,
  dimnames = list(paste0("x", 1:4), paste0("s", 1:5))
)
classes <- factor(c("A", "A", "B", "B", "A"))
filter_feature_selection(datamat, classes, caret::rfSBF)
```

---

flat_pattern_filter *Flat pattern filter*

Description

Performs a flat pattern filter over the dataset.

Usage

```
flat_pattern_filter(
  dataset,
  filter.function = "iqr",
  by.percent = TRUE,
  by.threshold = FALSE,
  red.value = 0
)
```

Arguments

dataset	A dataset object to filter.
filter.function	Filtering function to use. One of ‘"iqr"‘, ‘"rsd"‘, ‘"rnsd"‘, ‘"sd"‘, ‘"mad"‘, ‘"mean"‘, or ‘"median"‘.
by.percent	Logical. If ‘TRUE‘, the number of variables to filter will be defined as a percentage of the number of variables in the dataset; percentage is given by ‘red.value‘.
by.threshold	Logical. If ‘TRUE‘, filtering selects variables where the filtering function is above or equal to a threshold.
red.value	Reduction value. If ‘by.percent = TRUE‘, this is the percentage of variables to remove, or ‘"auto"‘ for automatic calculation. If ‘by.threshold = TRUE‘, this is the minimum value needed to keep the variable.

Value

A dataset object with the same overall structure as the input, where ‘dataset$data‘ has been filtered according to the selected flat-pattern criterion. Variables failing the selection rule are removed, the remaining dataset components are preserved, and ‘dataset$description‘ is updated to record the filtering step.

Examples

```
if (requireNamespace("specmine.datasets", quietly = TRUE)) {
  data(propolis, package = "specmine.datasets")
  propolis_proc = missingvalues_imputation(propolis)
  propolis_proc = flat_pattern_filter(propolis_proc, "iqr",
    by.percent = TRUE, red.value = 75)
}
```

---

get_cpd_names *Get compound names from KEGG codes*

Description

Returns compound names associated with KEGG compound identifiers.

Usage

```
get_cpd_names(kegg_codes)
```

Arguments

kegg_codes	Character vector of KEGG compound identifiers.

Value

A named character vector in which the values are KEGG compound identifiers and the names are the corresponding compound names. Each element of the returned vector represents one KEGG compound code matched to its primary compound name.

Examples

```
## Not run:
get_cpd_names(c("C00031", "C00022"))

## End(Not run)
```

get_metabolights_study

Download a complete MetaboLights study

Description

Downloads files and metadata for every assay in a public MetaboLights study.

Usage

```
get_metabolights_study(studyID, directory, verbose = TRUE)
```

Arguments

studyID	Character string with the MetaboLights study identifier.
directory	Output directory where study files and metadata will be written.
verbose	Logical indicating whether progress messages should be shown.

Value

Invisibly returns a `list` with one entry per assay containing downloaded files, metadata, and sample-file mappings.

Examples

```
## Not run:
tmpdir <- tempdir()
get_metabolights_study("MTBLS1", directory = tmpdir, verbose = FALSE)

## End(Not run)
```

get_metabolights_study_files_assay

Download files for one MetaboLights assay

Description

Downloads the data files associated with one assay of a MetaboLights study.

Usage

```
get_metabolights_study_files_assay(studyID, assay, directory, verbose = TRUE)
```

Arguments

studyID	Character string with the MetaboLights study identifier.
assay	Numeric or character index selecting the assay in the study.
directory	Output directory where assay files will be downloaded.
verbose	Logical indicating whether progress messages should be shown.

Value

Invisibly returns a character vector with downloaded file paths.

Examples

```
## Not run:
tmpdir <- tempdir()
get_metabolights_study_files_assay("MTBLS1", assay = 1, directory = tmpdir, verbose = FALSE)

## End(Not run)
```

---

get_metabolights_study_metadata_assay

Get metadata for one MetaboLights assay

Description

Returns the metadata associated with the samples in one assay of a MetaboLights study and can optionally save it as a CSV file.

Usage

```
get_metabolights_study_metadata_assay(studyID, assay, directory = NULL)
```

Arguments

studyID	Character string with the MetaboLights study identifier.
assay	Numeric or character index selecting the assay in the study.
directory	Optional output directory where the metadata CSV will be written.

Value

Invisibly returns a data.frame with sample metadata for the selected assay.

Examples

```
## Not run:
md <- get_metabolights_study_metadata_assay("MTBLS1", assay = 1)
head(md)

## End(Not run)
```

`get_metabolights_study_samples_files` *Get sample-file mapping for one MetaboLights assay*

Description

Returns the sample-to-file mapping for one assay in a MetaboLights study and can optionally save it as a CSV file.

Usage

```
get_metabolights_study_samples_files(studyID, assay, directory = NULL)
```

Arguments

`studyID`	Character string with the MetaboLights study identifier.
`assay`	Numeric or character index selecting the assay in the study.
`directory`	Optional output directory where `samples_files.csv` will be written.

Value

Invisibly returns a two-column object with samples and corresponding files.

Examples

```
## Not run:
get_metabolights_study_samples_files("MTBLS1", assay = 1)

## End(Not run)
```

`get_MetabolitePath` *Returns an object of KEGGPathway of the pathway especified in pathcode*

Description

Returns an object of KEGGPathway of the pathway especified in pathcode

Usage

```
get_MetabolitePath(pathcode)
```

Arguments

`pathcode`	TODO.

Value

A `KEGGPathway` object corresponding to the pathway specified by `pathcode`. This object contains the parsed KEGG pathway structure and can be used as input to downstream graph conversion and visualization functions.

Examples

```
## Not run:
path <- get_MetabolitePath("hsa00010")
class(path)

## End(Not run)
```

`get_metabPaths_org` *Get vector with paths numbers that occur in the given organism, named with the full path name:*

Description

Get vector with paths numbers that occur in the given organism, named with the full path name:

Usage

```
get_metabPaths_org(org_code)
```

Arguments

`org_code`	TODO.

Value

A named character vector with the pathway identifiers available for the specified organism, where each value is an organism-specific pathway code and each name is the corresponding KEGG pathway name.

Examples

```
## Not run:
head(get_metabPaths_org("hsa"))

## End(Not run)
```

`get_OrganismsCodes` *Get code, t number, full name and phylogeny of all organisms in KEGG:*

Description

Get code, t number, full name and phylogeny of all organisms in KEGG:

Usage

```
get_OrganismsCodes()
```

Value

A `data.frame` with the KEGG T number, organism code, species names, and phylogeny for all organisms available in KEGG. Each row represents one organism and the columns are `Tnumber`, `organismCode`, `speciesNames`, and `phylogeny`.

Examples

```
## Not run:
head(get_OrganismsCodes())

## End(Not run)
```

get_paths_with_cpds_org *Get only the paths of the organism that contain given compounds:*

Description

Get only the paths of the organism that contain given compounds:

Usage

```
get_paths_with_cpds_org(organism_code, compounds, full.result = TRUE)
```

Arguments

organism_code	TODO.
compounds	TODO.
full.result	TODO.

Value

A data.frame with the pathways containing the input compounds. Each row represents one matched pathway. When full.result = TRUE, the returned data frame includes the columns pathways, ratio, compounds, and compounds_names; otherwise it contains only pathways and ratio. Row names correspond to pathway names.

Examples

```
## Not run:
cpds <- c(glucose = "cpd:C00031", pyruvate = "cpd:C00022")
head(get_paths_with_cpds_org("hsa", cpds, full.result = FALSE))

## End(Not run)
```

get_x_label *Get x label*

Description

Returns the x-axis label associated with the dataset.

Usage

```
get_x_label(dataset)
```

Arguments

dataset	Dataset object.

Value

A character string giving the label of the x axis stored in ‘dataset$labels$x‘. If no x-axis label has been defined, the function returns an empty string.

Examples

```
datamatrix <- matrix(
  c(1, 2, 3, 4),
  nrow = 2,
  dimnames = list(c("10.5", "11.0"), c("s1", "s2"))
)
metadata <- data.frame(class = c("A", "B"), row.names = c("s1", "s2"))
dataset <- create_dataset(
  datamatrix,
  type = "concentrations",
  metadata = metadata,
  label.x = "ppm",
  label.values = "intensity"
)
get_x_label(dataset)
```

---

get_x_values_as_text *Get x values as text*

Description

Returns the x values of a dataset as text.

Usage

```
get_x_values_as_text(dataset)
```

Arguments

dataset	Dataset object.

Value

A character vector containing the variable identifiers stored in ‘rownames(dataset$data)‘. Each element corresponds to one row of the data matrix and represents the x-axis value or variable label associated with that feature.

Examples

```
datamatrix <- matrix(
  c(1, 2, 3, 4),
  nrow = 2,
  dimnames = list(c("10.5", "11.0"), c("s1", "s2"))
)
metadata <- data.frame(class = c("A", "B"), row.names = c("s1", "s2"))
dataset <- create_dataset(
  datamatrix,
  type = "concentrations",
  metadata = metadata
)
get_x_values_as_text(dataset)
```

---

ica_analysis_dataset *ICA analysis*

Description

Performs Independent Component Analysis (ICA) on a specmine dataset using the FastICA algorithm.

Usage

```
ica_analysis_dataset(
  dataset,
  n_components = 2,
  alg.typ = "parallel",
  fun = "logcosh",
  maxit = 200,
  tol = 1e-04,
  scale = FALSE,
  seed = 42,
  write.file = FALSE,
  file.out = NULL,
  ...
)
```

Arguments

dataset	Dataset to analyse.
n_components	Number of independent components to extract.
alg.typ	Algorithm used by FastICA. Supported values include "parallel" and "deflation".
fun	Contrast function used by FastICA. Supported values include "logcosh" and "exp".
maxit	Maximum number of iterations.

tol	Convergence tolerance.
scale	Logical indicating whether the data should be scaled before ICA.
seed	Random seed used for reproducibility.
write.file	Logical indicating whether the independent components should be written to a CSV file.
file.out	Output file prefix used when write.file = TRUE.
...	Additional arguments passed to fastICA::fastICA().

Value

A list containing:

- embedding: the independent component scores;
- S: the estimated source matrix;
- A: the estimated mixing matrix;
- K: the whitening matrix;
- W: the estimated unmixing matrix;
- loadings: the component loadings;
- params: the analysis parameters.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(240),
    nrow = 8,
    dimnames = list(
      paste0("feature", 1:8),
      paste0("sample", 1:30)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 2,
    maxit = 100,
    seed = 42
  )

  dim(ica.result$embedding)
}
```

`ica_kmeans_plot2D` *ICA 2D k-means plot*

Description

Creates a two-dimensional ICA plot coloured by k-means clusters.

Usage

```
ica_kmeans_plot2D(
  dataset,
  ica.result,
  num.clusters = 3,
  dims = c(1, 2),
  kmeans.result = NULL,
  use.embedding = TRUE,
  labels = FALSE,
  bw = FALSE,
  ellipses = FALSE,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

`dataset`	Dataset to cluster.
`ica.result`	Result returned by `ica_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`dims`	Two independent components to plot.
`kmeans.result`	Optional k-means result containing a `cluster` vector.
`use.embedding`	Logical indicating whether k-means should be applied to the ICA embedding when `kmeans.result` is not supplied.
`labels`	Logical indicating whether sample labels should be displayed.
`bw`	Logical indicating whether a black-and-white plot should be used.
`ellipses`	Logical indicating whether group ellipses should be displayed.
`leg.pos`	Legend position.
`xlim`	Optional x-axis limits.
`ylim`	Optional y-axis limits.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE)) {
  datamat <- matrix(rnorm(240), nrow = 8)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 2,
    maxit = 100,
    seed = 42
  )

  kmeans.result <- list(
    cluster = rep(1:2, each = 15)
  )

  ica_kmeans_plot2D(
    dataset,
    ica.result,
    num.clusters = 2,
    kmeans.result = kmeans.result
  )
}
```

`ica_kmeans_plot3D` *ICA 3D k-means plot*

Description

Creates an interactive three-dimensional ICA plot coloured by k-means clusters.

Usage

```
ica_kmeans_plot3D(
  dataset,
  ica.result,
  num.clusters = 3,
  dims = c(1, 2, 3),
  kmeans.result = NULL,
  use.embedding = TRUE,
  title = "ICA 3D K-means Plot"
)
```

Arguments

dataset	Dataset to cluster.
ica.result	Result returned by ica_analysis_dataset().
num.clusters	Number of k-means clusters.
dims	Three independent components to plot.
kmeans.result	Optional k-means result containing a cluster vector.
use.embedding	Logical indicating whether k-means should be applied to the ICA embedding when kmeans.result is not supplied.
title	Plot title.

Value

A plotly htmlwidget.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE) &&
    requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(rnorm(240), nrow = 8)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 3,
    maxit = 100,
    seed = 42
  )

  kmeans.result <- list(
    cluster = rep(1:2, each = 15)
  )

  ica_kmeans_plot3D(
    dataset,
    ica.result,
    num.clusters = 2,
    kmeans.result = kmeans.result
  )
}
```

`ica_loadingsplot` *ICA loadings plot*

Description

Creates a two-dimensional plot of the loadings associated with two independent components.

Usage

```
ica_loadingsplot(ica.result, dims = c(1, 2), top.n = NULL, labels = FALSE)
```

Arguments

`ica.result`	Result returned by `ica_analysis_dataset()`.
`dims`	Two independent components whose loadings should be plotted.
`top.n`	Optional number of variables with the largest loading magnitude to display.
`labels`	Logical indicating whether variable labels should be displayed.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(240),
    nrow = 8,
    dimnames = list(
      paste0("feature", 1:8),
      paste0("sample", 1:30)
    )
  )

  dataset <- list(data = datamat)

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 2,
    maxit = 100,
    seed = 42
  )

  ica_loadingsplot(
    ica.result,
    dims = c(1, 2),
    labels = TRUE
  )
```

```
}
```

`ica_pairs_kmeans_plot` *ICA pairs k-means plot*

Description

Creates a pairs plot for independent components coloured by k-means clusters.

Usage

```
ica_pairs_kmeans_plot(
  dataset,
  ica.result,
  num.clusters = 3,
  kmeans.result = NULL,
  use.embedding = TRUE,
  dims = 1:min(5, ncol(ica.result$embedding))
)
```

Arguments

`dataset`	Dataset to cluster.
`ica.result`	Result returned by `ica_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`kmeans.result`	Optional k-means result containing a `cluster` vector.
`use.embedding`	Logical indicating whether k-means should be applied to the ICA embedding when `kmeans.result` is not supplied.
`dims`	Independent components to include in the pairs plot.

Value

A `GGally` pairs plot object.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE) &&
    requireNamespace("GGally", quietly = TRUE)) {
  datamat <- matrix(rnorm(240), nrow = 8)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )
```

```
  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 3,
    maxit = 100,
    seed = 42
  )

  kmeans.result <- list(
    cluster = rep(1:2, each = 15)
  )

  ica_pairs_kmeans_plot(
    dataset,
    ica.result,
    num.clusters = 2,
    kmeans.result = kmeans.result,
    dims = 1:3
  )
}
```

`ica_pairs_plot` *ICA pairs plot*

Description

Creates a pairs plot for selected independent components.

Usage

```
ica_pairs_plot(
  dataset,
  ica.result,
  column.class = NULL,
  dims = 1:min(5, ncol(ica.result$embedding)),
  ...
)
```

Arguments

`dataset`	Dataset used to calculate the independent components.
`ica.result`	Result returned by `ica_analysis_dataset()`.
`column.class`	Optional metadata column used to colour samples.
`dims`	Independent components to include in the pairs plot.
`...`	Additional arguments passed to `GGally::ggpairs()`.

Value

A `GGally` pairs plot object.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE) &&
    requireNamespace("GGally", quietly = TRUE)) {
  datamat <- matrix(rnorm(240), nrow = 8)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 3,
    maxit = 100,
    seed = 42
  )

  ica_pairs_plot(
    dataset,
    ica.result,
    column.class = "class",
    dims = 1:3
  )
}
```

---

ica_scoresplot2D *ICA 2D scores plot*

Description

Creates a two-dimensional scores plot from an ICA result.

Usage

```
ica_scoresplot2D(
  dataset,
  ica.result,
  column.class = NULL,
  dims = c(1, 2),
  labels = FALSE,
  ellipses = FALSE,
  bw = FALSE,
  palette = 2,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

dataset	Dataset used to calculate the independent components.
ica.result	Result returned by ica_analysis_dataset().
column.class	Optional metadata column used to colour or group samples.
dims	Two independent components to plot.
labels	Logical indicating whether sample labels should be displayed.
ellipses	Logical indicating whether group ellipses should be displayed.
bw	Logical indicating whether a black-and-white plot should be used.
palette	RColorBrewer palette number.
leg.pos	Legend position.
xlim	Optional x-axis limits.
ylim	Optional y-axis limits.

Value

A ggplot object.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE)) {
  datamat <- matrix(rnorm(240), nrow = 8)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 2,
    maxit = 100,
    seed = 42
  )

  ica_scoresplot2D(
    dataset,
    ica.result,
    column.class = "class"
  )
}
```

`ica_scoresplot3D` *ICA 3D scores plot*

Description

Creates an interactive three-dimensional scores plot from an ICA result.

Usage

```
ica_scoresplot3D(
  dataset,
  ica.result,
  column.class = NULL,
  dims = c(1, 2, 3),
  title = "ICA 3D Scores Plot"
)
```

Arguments

`dataset`	Dataset used to calculate the independent components.
`ica.result`	Result returned by `ica_analysis_dataset()`.
`column.class`	Optional metadata column used to colour samples.
`dims`	Three independent components to plot.
`title`	Plot title.

Value

A `plotly` htmlwidget.

Examples

```
if (requireNamespace("fastICA", quietly = TRUE) &&
    requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(rnorm(240), nrow = 8)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 15))
    )
  )

  ica.result <- ica_analysis_dataset(
    dataset,
    n_components = 3,
    maxit = 100,
    seed = 42
  )
```

```
  ica_scoresplot3D(
    dataset,
    ica.result,
    column.class = "class"
  )
}
```

`impute_nas_knn` *Impute missing values with kNN*

Description

Replace missing values using k-nearest neighbors imputation.

Usage

```
impute_nas_knn(dataset, k = 10, ...)
```

Arguments

`dataset`	A dataset object to modify.
`k`	Number of neighbors to use in the imputation procedure.
`...`	Additional arguments passed to ‘impute::impute.knn()‘.

Value

A dataset object with the same structure as the input, where missing values in ‘dataset$data‘ have been imputed using the k-nearest neighbors method implemented in ‘impute::impute.knn()‘. The returned object preserves the remaining dataset components unchanged.

Examples

```
data <- matrix(
  c(1,  2, NA, 4,
    2,  3,  4, 5,
    3, NA,  5, 6,
    4,  5,  6, 7),
  nrow = 4,
  byrow = TRUE,
  dimnames = list(c("x1", "x2", "x3", "x4"), c("s1", "s2", "s3", "s4"))
)
dataset <- list(data = data)
impute_nas_knn(dataset, k = 2)
```

impute_nas_mean *Impute missing values with mean*

Description

Replace missing values in each variable by the mean of the observed values for that variable.

Usage

```
impute_nas_mean(dataset)
```

Arguments

dataset	A dataset object to modify.

Value

A dataset object with the same structure as the input, where missing values in ‘dataset$data‘ have been replaced by the mean of the corresponding variable calculated with ‘na.rm = TRUE‘.

Examples

```
data <- matrix(
  c(1, NA, 3, 4),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2"))
)
dataset <- list(data = data)
impute_nas_mean(dataset)
```

impute_nas_median *Impute missing values with median*

Description

Replace missing values in each variable by the median of the observed values for that variable.

Usage

```
impute_nas_median(dataset)
```

Arguments

dataset	A dataset object to modify.

Value

A dataset object with the same structure as the input, where missing values in ‘dataset$data‘ have been replaced by the median of the corresponding variable calculated with ‘na.rm = TRUE‘.

Examples

```
data <- matrix(
  c(1, NA, 5, 4),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2"))
)
dataset <- list(data = data)
impute_nas_median(dataset)
```

`impute_nas_value` *Impute missing values with a constant*

Description

Replace all ‘NA‘ values in ‘dataset$data‘ by a user-defined constant.

Usage

```
impute_nas_value(dataset, value)
```

Arguments

`dataset`	A dataset object to modify.
`value`	A numeric or character value used to replace missing entries.

Value

A dataset object with the same structure as the input, where all missing values in ‘dataset$data‘ have been replaced by ‘value‘. Other components of the dataset are preserved unchanged.

Examples

```
data <- matrix(
  c(1, NA, 3, 4),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2"))
)
dataset <- list(data = data)
impute_nas_value(dataset, 0)
```

`merge_data_metadata` *Merge data and metadata*

Description

Merges data and metadata into a data frame.

Usage

```
merge_data_metadata(
  dataset,
  samples = NULL,
  metadata.vars = NULL,
  x.values = NULL,
  by.index = FALSE
)
```

Arguments

`dataset`	Dataset to merge.
`samples`	Samples to keep.
`metadata.vars`	Metadata variables to keep.
`x.values`	X values to keep.
`by.index`	Logical. If TRUE, x.values are indexes.

Value

A ‘data.frame‘ containing the selected data matrix, transposed so that samples are rows, combined with the selected metadata columns.

Examples

```
data <- matrix(
  1:6,
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = c("A", "B", "A"),
  batch = c(1, 1, 2),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
merge_data_metadata(dataset, metadata.vars = "class")
```

metabolights_studies_list

List public MetaboLights studies

Description

Returns the identifiers of public studies available in MetaboLights.

Usage

```
metabolights_studies_list()
```

Value

A `character` vector with public MetaboLights study identifiers.

Examples

```
## Not run:
head(metabolights_studies_list())

## End(Not run)
```

missingvalues_imputation

Missing values imputation

Description

Impute missing values in a dataset using different methods.

Usage

```
missingvalues_imputation(dataset, method = "value", value = 5e-04, k = 5)
```

Arguments

`dataset`	A dataset object to process.
`method`	Imputation method: ‘"value"‘, ‘"mean"‘, ‘"median"‘, ‘"knn"‘, or ‘"linapprox"‘.
`value`	If ‘method = "value"‘, the value used to replace missing entries.
`k`	If ‘method = "knn"‘, the number of neighbors used for imputation.

Value

A dataset object with the same overall structure as the input, in which missing values in ‘dataset$data‘ have been imputed according to the selected method. The returned object preserves the dataset components and updates the description to record the imputation step.

Examples

```
data <- matrix(
  c(1, NA, 3, 4, 5, NA),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
dataset <- list(data = data, description = "toy dataset")
missingvalues_imputation(dataset, method = "value", value = 0)
```

---

multiClassSummary *Multi-class summary metrics*

Description

Compute overall and class-averaged performance metrics for multiclass classification models, including ROC AUC and log-loss when class probabilities are available.

Usage

```
multiClassSummary(data, lev = NULL, model = NULL)
```

Arguments

data	A data frame containing at least the columns ‘pred‘ and ‘obs‘, plus one probability column per class when ROC or log-loss are needed.
lev	An optional character vector with the class levels.
model	An optional fitted model object passed by ‘caret‘.

Value

A named numeric vector with overall and class-averaged classification statistics.

Examples

```
data <- data.frame(
  pred = factor(c("A", "B", "A", "B"), levels = c("A", "B")),
  obs = factor(c("A", "B", "B", "B"), levels = c("A", "B")),
  A = c(0.8, 0.2, 0.7, 0.3),
  B = c(0.2, 0.8, 0.3, 0.7)
)
multiClassSummary(data)
```

pathway_analysis *Creates the pathway wanted. If any of the given compounds is present in the pathway, it is coloured differently.*

Description

Creates the pathway wanted. If any of the given compounds is present in the pathway, it is coloured differently.

Usage

```
pathway_analysis(
  compounds,
  pathway,
  nodeNames = "kegg",
  nodeTooltip = FALSE,
  map.zoom = FALSE,
  map.layout = "preset",
  map.width = NULL,
  map.height = NULL,
  verbose = TRUE
)
```

Arguments

compounds	TODO.
pathway	TODO.
nodeNames	TODO.
nodeTooltip	TODO.
map.zoom	TODO.
map.layout	TODO.
map.width	TODO.
map.height	TODO.
verbose	Logical indicating whether progress messages should be shown.

Value

A cyjShiny widget representing the selected pathway with the input compounds highlighted when present. The returned widget can be printed or embedded in an interactive R session to inspect the pathway graph.

Examples

```
## Not run:
pathway_analysis(
  compounds = c(glucose = "cpd:C00031", pyruvate = "cpd:C00022"),
  pathway = "hsa00010",
  verbose = FALSE
)

## End(Not run)
```

`pca_analysis_dataset` *Classical PCA analysis*

Description

Performs classical Principal Component Analysis (PCA) on a specmine dataset using `stats::prcomp()`.

Usage

```
pca_analysis_dataset(
  dataset,
  scale = TRUE,
  center = TRUE,
  write.file = FALSE,
  file.out = NULL,
  ...
)
```

Arguments

`dataset`	Dataset to analyse. The numeric data matrix must be stored in `dataset$data`, with variables in rows and samples in columns.
`scale`	Logical indicating whether variables should be scaled before PCA. The default is `TRUE`.
`center`	Logical indicating whether variables should be centred before PCA. The default is `TRUE`.
`write.file`	Logical indicating whether the PCA scores and loadings should be written to CSV files.
`file.out`	Output file prefix used when `write.file = TRUE`. Two files are created: `<file.out>_scores.csv` and `<file.out>_loadings.csv`.
`...`	Additional arguments passed to `stats::prcomp()`.

Details

The dataset is expected to contain variables in rows and samples in columns. The data matrix is transposed internally so that samples are treated as observations and variables as features.

Value

An object of class `prcomp`. The returned object contains:

- `x`: a matrix of PCA scores, with one row per sample;
- `rotation`: a matrix of PCA loadings, with one row per variable;
- `sdev`: the standard deviations of the principal components;
- `center`: the centring values used by the PCA;
- `scale`: the scaling values used by the PCA when `scale = TRUE`;
- `call`: the matched function call.

The row names of `x` correspond to sample names when the columns of `dataset$data` are named. The row names of `rotation` correspond to variable names when the rows of `dataset$data` are named.

Examples

```
datamat <- matrix(
  rnorm(40),
  nrow = 4,
  ncol = 10,
  dimnames = list(
    paste0("feature", 1:4),
    paste0("sample", 1:10)
  )
)

dataset <- list(data = datamat)

pca.result <- pca_analysis_dataset(
  dataset,
  scale = TRUE,
  center = TRUE
)

dim(pca.result$x)
dim(pca.result$rotation)


file.prefix <- file.path(tempdir(), "pca_example")

pca_analysis_dataset(
  dataset,
  write.file = TRUE,
  file.out = file.prefix
)

file.exists(paste0(file.prefix, "_scores.csv"))
file.exists(paste0(file.prefix, "_loadings.csv"))
```

`pca_biplot` *PCA biplot*

Description

Draws a PCA biplot for a `prcomp` or `princomp` result.

Usage

```
pca_biplot(
  dataset,
  pca.result,
  cex = 0.8,
  legend.cex = 0.8,
  x.colors = 1,
  inset = c(0, 0),
  legend.place = "topright",
  ...
)
```

Arguments

`dataset`	Dataset used in the PCA.
`pca.result`	PCA result object.
`cex`	Text expansion factor.
`legend.cex`	Legend text size.
`x.colors`	Colour vector or metadata column name.
`inset`	Legend inset.
`legend.place`	Legend position.
`...`	Additional arguments passed to the plotting method.

Value

No return value, called for its side effect of plotting.

Examples

```
datamat <- matrix(
  rnorm(20),
  nrow = 4,
  dimnames = list(paste0("x", 1:4), paste0("s", 1:5))
)
metadata <- data.frame(class = factor(c("A", "A", "B", "B", "A")))
dataset <- list(data = datamat, metadata = metadata)
pca_res <- pca_analysis_dataset(dataset)
pca_biplot(dataset, pca_res)
```

pca_biplot3D *PCA 3D biplot*

Description

Creates an interactive 3D PCA biplot using `rgl`.

Usage

```
pca_biplot3D(dataset, pca.result, column.class = NULL, pcas = c(1, 2, 3))
```

Arguments

`dataset`	Dataset used in the PCA.
`pca.result`	PCA result object.
`column.class`	Optional metadata column used for colouring groups.
`pcas`	Principal components to plot.

Value

No return value, called for its side effect of plotting.

Examples

```
datamat <- matrix(
  rnorm(20),
  nrow = 4,
  dimnames = list(paste0("x", 1:4), paste0("s", 1:5))
)
metadata <- data.frame(class = factor(c("A", "A", "B", "B", "A")))
dataset <- list(data = datamat, metadata = metadata)
pca_res <- pca_analysis_dataset(dataset)
pca_biplot3D(dataset, pca_res, column.class = "class")
```

pca_importance *PCA component importance*

Description

Calculates the standard deviation, proportion of variance and cumulative proportion of variance explained by selected principal components.

Usage

```
pca_importance(
  pca.res,
  pcs = 1:length(pca.res$sdev),
  sd = TRUE,
  prop = TRUE,
  cumul = TRUE,
  min.cum = NULL
)
```

Arguments

pca.res	An object inheriting from class prcomp or princomp.
pcs	Integer vector specifying the principal components to return. By default, all available components are returned.
sd	Logical indicating whether the standard deviation row should be returned.
prop	Logical indicating whether the proportion of variance row should be returned.
cumul	Logical indicating whether the cumulative proportion row should be returned.
min.cum	Optional numeric value between 0 and 1. When supplied, components are selected up to the first component whose cumulative proportion is greater than or equal to this value.

Details

The function accepts PCA results generated by pca_analysis_dataset(), stats::prcomp() or stats::princomp().

Value

A numeric matrix with one row for each requested measure: Standard deviation, Proportion of Variance and/or Cumulative Proportion. Columns correspond to the selected principal components.

For a prcomp object, the values are calculated from its sdev component. For a princomp object, the same calculation is applied to its sdev component.

Examples

```
datamat <- matrix(
  rnorm(40),
  nrow = 4,
  ncol = 10,
  dimnames = list(
    paste0("feature", 1:4),
    paste0("sample", 1:10)
  )
)

dataset <- list(data = datamat)
```

```
pca.result <- pca_analysis_dataset(dataset)

pca_importance(
  pca.result,
  pcs = 1:3,
  sd = TRUE,
  prop = TRUE,
  cumul = TRUE
)

pca_importance(
  pca.result,
  min.cum = 0.90,
  sd = FALSE,
  prop = TRUE,
  cumul = TRUE
)
```

`pca_kmeans_plot2D` *PCA 2D k-means plot*

Description

Plots PCA scores in 2D coloured by k-means cluster assignment.

Usage

```
pca_kmeans_plot2D(
  dataset,
  pca.result,
  num.clusters = 3,
  pcas = c(1, 2),
  kmeans.result = NULL,
  labels = FALSE,
  bw = FALSE,
  ellipses = FALSE,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

`dataset`	Dataset used in the PCA.
`pca.result`	PCA result object.
`num.clusters`	Number of clusters.
`pcas`	Principal components to plot.

kmeans.result	Optional precomputed k-means result.
labels	Logical indicating whether sample labels should be shown.
bw	Logical indicating whether a black-and-white style should be used.
ellipses	Logical indicating whether cluster ellipses should be drawn.
leg.pos	Legend position.
xlim	Optional x-axis limits.
ylim	Optional y-axis limits.

Value

A ggplot object.

Examples

```
datamat <- matrix(
  rnorm(20),
  nrow = 4,
  dimnames = list(paste0("x", 1:4), paste0("s", 1:5))
)
metadata <- data.frame(class = factor(c("A", "A", "B", "B", "A")))
dataset <- list(data = datamat, metadata = metadata)
pca_res <- pca_analysis_dataset(dataset)
pca_kmeans_plot2D(dataset, pca_res, num.clusters = 2)
```

---

pca_kmeans_plot3D *PCA 3D k-means plot*

Description

Plots PCA scores in 3D coloured by k-means cluster assignment.

Usage

```
pca_kmeans_plot3D(
  dataset,
  pca.result,
  num.clusters = 3,
  pcas = c(1, 2, 3),
  kmeans.result = NULL,
  labels = FALSE,
  size = 1,
  ...
)
```

Arguments

dataset	Dataset used in the PCA.
pca.result	PCA result object.
num.clusters	Number of clusters.
pcas	Principal components to plot.
kmeans.result	Optional precomputed k-means result.
labels	Logical indicating whether sample labels should be shown.
size	Point size.
...	Additional arguments passed to rgl::plot3d().

Value

No return value, called for its side effect of plotting.

Examples

```
datamat <- matrix(
  rnorm(20),
  nrow = 4,
  dimnames = list(paste0("x", 1:4), paste0("s", 1:5))
)
metadata <- data.frame(class = factor(c("A", "A", "B", "B", "A")))
dataset <- list(data = datamat, metadata = metadata)
pca_res <- pca_analysis_dataset(dataset)
pca_kmeans_plot3D(dataset, pca_res, num.clusters = 2)
```

---

pca_pairs_kmeans_plot *PCA pairs plot with k-means clusters*

Description

Creates a pairs plot of PCA scores coloured by k-means cluster assignment.

Usage

```
pca_pairs_kmeans_plot(
  dataset,
  pca.result,
  num.clusters = 3,
  kmeans.result = NULL,
  pcas = c(1, 2, 3, 4, 5)
)
```

Arguments

dataset	Dataset used in the PCA.
pca.result	PCA result object.
num.clusters	Number of clusters.
kmeans.result	Optional precomputed k-means result.
pcas	Principal components to include.

Value

A ggmatrix object.

Examples

```
datamat <- matrix(
  rnorm(30),
  nrow = 6,
  dimnames = list(paste0("x", 1:6), paste0("s", 1:5))
)
metadata <- data.frame(class = factor(c("A", "A", "B", "B", "A")))
dataset <- list(data = datamat, metadata = metadata)
pca_res <- pca_analysis_dataset(dataset)
pca_pairs_kmeans_plot(dataset, pca_res, num.clusters = 2, pcas = 1:3)
```

---

pca_pairs_plot *PCA pairs plot*

Description

Creates a pairs plot of PCA scores.

Usage

```
pca_pairs_plot(
  dataset,
  pca.result,
  column.class = NULL,
  pcas = c(1, 2, 3, 4, 5),
  ...
)
```

Arguments

dataset	Dataset used in the PCA.
pca.result	PCA result object.
column.class	Optional metadata column used for colouring groups.
pcas	Principal components to include.
...	Additional arguments passed to GGally::ggpairs().

Value

A ggmatrix object.

Examples

```
datamat <- matrix(
  rnorm(30),
  nrow = 6,
  dimnames = list(paste0("x", 1:6), paste0("s", 1:5))
)
metadata <- data.frame(class = factor(c("A", "A", "B", "B", "A")))
dataset <- list(data = datamat, metadata = metadata)
pca_res <- pca_analysis_dataset(dataset)
pca_pairs_plot(dataset, pca_res, column.class = "class", pcas = 1:3)
```

---

pca_robust *Robust PCA analysis*

Description

Performs robust Principal Component Analysis using pcaPP::PCAgrid().

Usage

```
pca_robust(
  dataset,
  center = "median",
  scale = "mad",
  k = 10,
  write.file = FALSE,
  file.out = NULL,
  ...
)
```

Arguments

dataset	Dataset to analyse. The numeric data matrix must be stored in dataset$data, with variables in rows and samples in columns.
center	Method used to centre the data. Common choices are "mean", "median" or NULL, according to the options supported by pcaPP::PCAgrid().
scale	Method used to scale the data. Common choices are "sd", "mad" or NULL, according to the options supported by pcaPP::PCAgrid().
k	Number of robust principal components to compute.
write.file	Logical indicating whether the robust PCA scores and loadings should be written to CSV files.
file.out	Output file prefix used when write.file = TRUE. The function creates <file.out>_scores.csv and <file.out>_loadings.csv.
...	Additional arguments passed to pcaPP::PCAgrid().

Details

The dataset is expected to contain variables in rows and samples in columns. The data matrix is transposed internally so that samples are treated as observations and variables as features.

Value

The object returned by pcaPP::PCAgrid(). Depending on the version of pcaPP, the result contains robust PCA scores, loadings, component deviations and the centring and scaling information used in the analysis. When scores are available, they can be accessed with pca.result$scores; loadings can be accessed with pca.result$loadings.

Examples

```
if (requireNamespace("pcaPP", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 4,
    ncol = 10,
    dimnames = list(
      paste0("feature", 1:4),
      paste0("sample", 1:10)
    )
  )

  dataset <- list(data = datamat)

  robust.result <- pca_robust(
    dataset,
    center = "median",
    scale = "mad",
    k = 2
  )
```

```
  is.list(robust.result)
}


if (requireNamespace("pcaPP", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 4,
    ncol = 10,
    dimnames = list(
      paste0("feature", 1:4),
      paste0("sample", 1:10)
    )
  )

  dataset <- list(data = datamat)
  file.prefix <- file.path(tempdir(), "robust_pca_example")

  pca_robust(
    dataset,
    k = 2,
    write.file = TRUE,
    file.out = file.prefix
  )

  file.exists(paste0(file.prefix, "_scores.csv"))
  file.exists(paste0(file.prefix, "_loadings.csv"))
}
```

`pca_scoresplot2D` *PCA 2D scores plot*

Description

Creates a two-dimensional PCA scores plot.

Usage

```
pca_scoresplot2D(
  dataset,
  pca.result,
  column.class = NULL,
  pcas = c(1, 2),
  labels = FALSE,
  ellipses = FALSE,
  bw = FALSE,
  pallette = 2,
```

```
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

dataset	Dataset used in the PCA. Its data matrix must contain variables in rows and samples in columns.
pca.result	PCA result object of class prcomp or princomp.
column.class	Optional metadata column used for colouring or grouping samples.
pcas	Integer vector of length two specifying the principal components to plot.
labels	Logical indicating whether sample labels should be shown.
ellipses	Logical indicating whether group ellipses should be drawn.
bw	Logical indicating whether a black-and-white style should be used.
pallette	RColorBrewer palette identifier. The argument name pallette is retained for compatibility with previous versions.
leg.pos	Legend position.
xlim	Optional numeric vector of length two defining the x-axis limits.
ylim	Optional numeric vector of length two defining the y-axis limits.

Details

The function accepts results from pca_analysis_dataset(), stats::prcomp() or stats::princomp().

Value

A ggplot object containing the PCA scores plot. The plotted points represent samples, and their coordinates are the selected PCA scores. When column.class is supplied, point colours or shapes represent the corresponding metadata groups.

Examples

```
datamat <- matrix(
  rnorm(40),
  nrow = 4,
  ncol = 10,
  dimnames = list(
    paste0("feature", 1:4),
    paste0("sample", 1:10)
  )
)

dataset <- list(
  data = datamat,
  metadata = data.frame(
    class = factor(rep(c("A", "B"), each = 5))
```

```
  )
)

pca.result <- pca_analysis_dataset(dataset)

pca_scoresplot2D(
  dataset,
  pca.result,
  column.class = "class",
  pcas = c(1, 2),
  labels = TRUE
)
```

`pca_scoresplot3D` *PCA 3D scores plot*

Description

Creates an interactive three-dimensional scores plot from a PCA result.

Usage

```
pca_scoresplot3D(
  dataset,
  pca.result,
  column.class = NULL,
  pcas = c(1, 2, 3),
  title = "PCA 3D Scores Plot"
)
```

Arguments

`dataset`	Dataset used to calculate the PCA.
`pca.result`	Result returned by `pca_analysis_dataset()`.
`column.class`	Optional metadata column used to colour samples.
`pcas`	Integer vector of length three specifying the principal components to plot.
`title`	Plot title.

Value

A `plotly` htmlwidget.

Examples

```
if (requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    ncol = 8
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  result <- pca_analysis_dataset(dataset)

  pca_scoresplot3D(
    dataset,
    result,
    column.class = "class",
    pcas = c(1, 2, 3)
  )
}
```

---

pca_scoresplot3D_rgl *PCA 3D scores plot using rgl*

Description

Creates an interactive three-dimensional PCA scores plot using `rgl`.

Usage

```
pca_scoresplot3D_rgl(
  dataset,
  pca.result,
  column.class = NULL,
  pcas = c(1, 2, 3),
  size = 1,
  labels = FALSE
)
```

Arguments

`dataset`	Dataset used in the PCA. Its data matrix must contain variables in rows and samples in columns.

pca.result	PCA result object of class prcomp or princomp.
column.class	Optional metadata column used for colouring sample groups.
pcas	Integer vector of length three specifying the principal components to plot.
size	Numeric point size passed to rgl::plot3d().
labels	Logical indicating whether sample labels should be shown.

Details

Samples are represented as points in the selected principal component space. When column.class is supplied, point colours represent the corresponding metadata groups.

Value

Invisibly returns the object produced by rgl::plot3d(); the three-dimensional plot is also drawn as a side effect.

Examples

```
if (requireNamespace("rgl", quietly = TRUE)) {
  options(rgl.useNULL = TRUE)

  datamat <- matrix(
    rnorm(40),
    nrow = 4,
    ncol = 10,
    dimnames = list(
      paste0("feature", 1:4),
      paste0("sample", 1:10)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), each = 5))
    )
  )

  pca.result <- pca_analysis_dataset(dataset)

  pca_scoresplot3D_rgl(
    dataset,
    pca.result,
    column.class = "class",
    pcas = c(1, 2, 3),
    labels = TRUE
  )
}
```

`pca_screeplot` *PCA scree plot*

Description

Draws a scree plot showing the individual and cumulative percentage of variance explained by the principal components.

Usage

```
pca_screeplot(
  pca.result,
  num.pcs = NULL,
  cex.leg = 0.8,
  leg.pos = "right",
  lab.text = c("individual percent", "cumulative percent"),
  fill.col = c("blue", "red"),
  ylab = "Percentage",
  xlab = "Principal components",
  ...
)
```

Arguments

`pca.result`	PCA result object inheriting from class `prcomp` or `princomp`.
`num.pcs`	Optional number of principal components to display. If `NULL`, all available components are displayed.
`cex.leg`	Numeric value controlling the legend text size.
`leg.pos`	Legend position accepted by `graphics::legend()`.
`lab.text`	Character vector of length two containing the labels for the individual and cumulative variance curves.
`fill.col`	Character or numeric vector of length two containing the colours for the individual and cumulative variance curves.
`ylab`	Y-axis label.
`xlab`	X-axis label.
`...`	Additional graphical parameters passed to `graphics::matplot()`.

Value

No return value; the scree plot is drawn as a side effect.

Examples

```
datamat <- matrix(
  rnorm(40),
  nrow = 4,
  ncol = 10,
  dimnames = list(
    paste0("feature", 1:4),
    paste0("sample", 1:10)
  )
)

dataset <- list(data = datamat)
pca.result <- pca_analysis_dataset(dataset)

pca_screeplot(
  pca.result,
  num.pcs = 4
)
```

---

peak_detection2d *Detect peaks in 2D NMR spectra*

Description

Detects local maxima in each spectrum of a 2D NMR dataset and converts the detected peaks into a 1D specmine dataset suitable for downstream analysis.

Usage

```
peak_detection2d(
  specmine_2d_dataset,
  baseline_thresh = NULL,
  noiseFilt = 0,
  negatives = FALSE,
  verbose = TRUE
)
```

Arguments

specmine_2d_dataset	A specmine 2D dataset containing one numeric matrix per sample.
baseline_thresh	Optional numeric threshold used as the minimum intensity for peak detection. If NULL, a threshold is estimated from the positive values of each spectrum.
noiseFilt	Integer noise filter level: 0 for no filter, 1 for a mild filter, and 2 for a strong filter.
negatives	Logical indicating whether variables with negative ppm values should be kept.
verbose	Logical indicating whether progress messages should be shown.

Value

A specmine dataset object as a list. The `data` element is a numeric matrix where rows correspond to detected 2D peak coordinates encoded as combined F1/F2 positions and columns correspond to samples. Matrix entries contain peak intensities, with missing values indicating that a given peak was not detected in a sample. The returned object also includes the original sample metadata, the dataset description, the dataset type set to `"nmr-peaks"`, and labels describing the x-axis and data values. This output represents a reduced 1D peak table derived from the input 2D NMR spectra.

Examples

```
m <- matrix(c(0, 5, 0, 0), nrow = 2)
rownames(m) <- c("1.0", "1.1")
colnames(m) <- c("2.0", "2.1")
x <- list(data = list(s1 = m), metadata = data.frame(), description = "Toy")
try(peak_detection2d(x, baseline_thresh = 1, negatives = TRUE, verbose = FALSE))
```

---

raman_align_peaks *Align detected peaks across spectra into wavenumber bins*

Description

Groups peaks detected in multiple spectra into wavenumber bins defined by a tolerance. The resulting feature matrix can be used in downstream analyses such as PCA, clustering or classification.

Usage

```
raman_align_peaks(
  peak_results,
  tolerance = 4,
  reference_wavenumbers = NULL,
  feature_method = "max_height"
)
```

Arguments

`peak_results`	A non-empty list of peak-detection results, one per sample. Each element must contain `sample_id`, `wavenumbers`, `peaks` and `properties`.
`tolerance`	Positive numeric value defining the half-width, in wavenumbers, around each reference position.
`reference_wavenumbers`	Optional non-empty numeric vector of reference wavenumbers. If `NULL`, reference positions are derived from detected peaks and clustered according to `tolerance`.
`feature_method`	Character string defining the feature summarised within each bin. One of `"max_height"`, `"max_prominence"`, `"presence"` or `"mean_height"`.

Details

Missing peaks are represented by zero in the returned feature matrix.

Value

Data frame with one row per reference wavenumber and one column per sample. The first column, `wavenumber`, contains bin reference positions.

Examples

```
wn = seq(400, 1800, by = 1)

x1 = dnorm(wn, mean = 1000, sd = 20) +
  0.01 * rnorm(length(wn))

x2 = dnorm(wn, mean = 1002, sd = 20) +
  0.01 * rnorm(length(wn))

p1 = raman_find_peaks(x1, height = 0.01, distance = 5)
p2 = raman_find_peaks(x2, height = 0.01, distance = 5)

peak_results = list(
  list(
    sample_id = "sample1",
    wavenumbers = wn,
    peaks = p1$peaks,
    properties = p1$properties
  ),
  list(
    sample_id = "sample2",
    wavenumbers = wn,
    peaks = p2$peaks,
    properties = p2$properties
  )
)

features = raman_align_peaks(
  peak_results,
  tolerance = 4,
  feature_method = "max_height"
)

head(features)
```

`raman_crop_spectra` *Crop Raman or SERS spectra to a wavenumber interval*

Description

Restricts a Raman or surface-enhanced Raman spectroscopy (SERS) dataset to an inclusive wavenumber interval.

Usage

```
raman_crop_spectra(dataset, min_wn, max_wn)
```

Arguments

dataset	A dataset list containing a numeric matrix in dataset$data, with wavenumbers in rows and samples in columns. Optionally, it may contain a numeric vector in dataset$wavenumbers.
min_wn	Single finite numeric value specifying the lower inclusive wavenumber limit.
max_wn	Single finite numeric value specifying the upper inclusive wavenumber limit.

Details

The dataset must contain wavenumbers in rows and samples in columns. Wavenumbers are obtained from dataset$wavenumbers when available. Otherwise, numeric row names in dataset$data are used.

Value

A dataset equivalent to dataset, with dataset$data restricted to wavenumbers between min_wn and max_wn. The dataset$wavenumbers element is updated or created to match the cropped data.

Examples

```
dataset = list(
  data = matrix(
    seq_len(30),
    nrow = 6,
    dimnames = list(
      as.character(seq(400, 900, by = 100)),
      paste0("sample", 1:5)
    )
  ),
  wavenumbers = seq(400, 900, by = 100)
)

cropped = raman_crop_spectra(
  dataset,
  min_wn = 500,
  max_wn = 800
)

cropped$wavenumbers
```

raman_despike *Remove spikes from Raman or SERS spectra*

Description

Detects and replaces intensity spikes in Raman or surface-enhanced Raman spectroscopy (SERS) spectra using modified Z-scores based on the median absolute deviation (MAD).

Usage

```
raman_despike(dataset, ma = 10, threshold = 7)
```

Arguments

dataset	A dataset list containing a numeric matrix in dataset$data, with wavenumbers in rows and samples in columns.
ma	Non-negative integer specifying the number of neighbouring points considered on each side of a detected spike.
threshold	Positive finite numeric value specifying the modified Z-score threshold used to identify spikes.

Details

The dataset must contain wavenumbers in rows and samples in columns. Each sample spectrum is processed independently. Detected spikes are replaced by the mean intensity of neighbouring non-spike points.

Value

A dataset equivalent to dataset, with despiked spectra stored in dataset$data. Information about the operation is stored in dataset$despike.

Examples

```
dataset = list(
  data = matrix(
    c(
      1, 1.1, 1.2, 20, 1.1, 1.2,
      2, 2.1, 2.2, 30, 2.1, 2.2
    ),
    nrow = 6,
    ncol = 2,
    dimnames = list(
      paste0("wavenumber", 1:6),
      paste0("sample", 1:2)
    )
  )
)
```

```
despiked = raman_despike(
  dataset,
  ma = 1,
  threshold = 3
)

despiked$despike$n_spikes
```

`raman_find_peaks` *Find peaks in Raman or SERS spectra*

Description

Finds local maxima in one-dimensional spectra according to peak properties such as height, threshold, distance, prominence, width and plateau size.

Usage

```
raman_find_peaks(
  x,
  height = NULL,
  threshold = NULL,
  distance = NULL,
  prominence = NULL,
  width = NULL,
  plateau_size = NULL,
  wlen = NULL,
  rel_height = 0.5
)
```

Arguments

`x`	Numeric vector containing the signal intensity values.
`height`	Numeric value or length-two numeric vector specifying the required height of peaks. A single value is interpreted as a minimum height. A vector `c(min, max)` defines a closed interval.
`threshold`	Numeric value or length-two numeric vector specifying the required vertical distance to neighbouring samples.
`distance`	Positive integer specifying the minimum horizontal distance, in samples, between neighbouring peaks.
`prominence`	Numeric value or length-two numeric vector specifying the required prominence of peaks.
`width`	Numeric value or length-two numeric vector specifying the required width of peaks, in samples.

plateau_size	Numeric value or length-two numeric vector specifying the required size of the flat top of peaks, in samples.
wlen	Optional positive integer. It defines the maximum number of samples inspected on each side of a peak when calculating prominence.
rel_height	Numeric value between zero and one used for peak-width calculations.

Details

The interface and peak-property terminology are inspired by `scipy.signal.find_peaks()`. Prominence and width calculations use an R implementation tailored to Raman/SERS spectra and are not intended to reproduce SciPy results exactly.

Value

A list with two elements:

`peaks` Integer vector of one-based indices of peaks in `x` satisfying the requested conditions.

`properties` Data frame containing properties of the returned peaks, including `peak_index` and `peak_height`. Additional columns are included when the respective properties are calculated.

Examples

```
x = c(0, 1, 0, 2, 1, 0, 3, 2, 0, 4, 1)

res = raman_find_peaks(
  x,
  height = 1,
  distance = 2
)

res$peaks
res$properties
```

`raman_normalize` *Normalize Raman or SERS spectra*

Description

Normalizes Raman or surface-enhanced Raman spectroscopy (SERS) spectra using min-max normalization, L2 vector normalization, or normalization by the maximum intensity near a target silicon peak.

Usage

```
raman_normalize(
  dataset,
  method = "min-max",
  target_peak = 520,
  search_window = 20
)
```

Arguments

dataset	A dataset list containing a numeric matrix in dataset$data, with wavenumbers in rows and samples in columns. For method = "silicon", the dataset must also contain a numeric vector in dataset$wavenumbers, with one value per row of dataset$data.
method	Character string specifying the normalization method. One of "min-max", "L2", or "silicon".
target_peak	Single finite numeric value specifying the target wavenumber of the silicon peak when method = "silicon".
search_window	Positive finite numeric value specifying the half-width of the wavenumber search interval around target_peak when method = "silicon".

Details

The dataset must contain wavenumbers in rows and samples in columns. Each sample spectrum is normalized independently.

Value

A dataset equivalent to dataset, with normalized spectra stored in dataset$data. Information about the selected normalization method and its parameters is stored in dataset$normalization.

Examples

```
dataset = list(
  data = matrix(
    c(
      1, 2, 3, 4, 5,
      2, 4, 6, 8, 10
    ),
    nrow = 5,
    ncol = 2,
    dimnames = list(
      as.character(seq(500, 900, by = 100)),
      paste0("sample", 1:2)
    )
  ),
  wavenumbers = seq(500, 900, by = 100)
)
```

```
normalized = raman_normalize(
  dataset,
  method = "min-max"
)

range(normalized$data[, 1])
```

`raman_normalize_peak_features` *Normalize peak feature matrix*

Description

Normalizes a peak-feature matrix returned by `raman_align_peaks()`.

Usage

```
raman_normalize_peak_features(features, method = "min-max")
```

Arguments

`features`	Data frame with a numeric `wavenumber` column and one or more numeric sample-feature columns.
`method`	Normalization method: `"min-max"` or `"L2"`.

Details

With `method = "min-max"`, normalization is carried out row-wise across samples. Rows containing identical values for every sample are set to zero. With `method = "L2"`, normalization is performed column-wise for each sample. Columns with zero L2 norm are retained as zero.

Value

Data frame with the same structure as `features`, containing normalized sample-feature values.

Examples

```
features = data.frame(
  wavenumber = c(1000, 1200, 1600),
  sample1 = c(1, 2, 3),
  sample2 = c(2, 4, 6)
)

raman_normalize_peak_features(features, method = "L2")
```

raman_sgolay_derivative *Compute Savitzky-Golay second derivative of Raman or SERS spectra*

Description

Applies a Savitzky-Golay smoothing filter and computes the second derivative of Raman or surface-enhanced Raman spectroscopy (SERS) spectra.

Usage

```
raman_sgolay_derivative(dataset, window_size = 7, polynomial_order = 2)
```

Arguments

dataset	A dataset list containing a numeric matrix in dataset$data, with wavenumbers in rows and samples in columns. The dataset must also contain a numeric vector in dataset$wavenumbers, with one value per row of dataset$data.
window_size	Odd positive integer specifying the size of the moving window (number of points). Must be less than or equal to the number of wavenumbers.
polynomial_order	Positive integer specifying the order of the polynomial used for local fitting. Must be less than window_size.

Details

This function is a wrapper around savitzky_golay() specialized for Raman/SERS datasets. It fixes the differentiation order to 2 and provides argument names and validation tailored to spectroscopic workflows.

The dataset must contain wavenumbers in rows and samples in columns. Each sample spectrum is processed independently. The second derivative is computed using polynomial fitting within a moving window.

Value

A dataset equivalent to dataset, with second derivative spectra stored in dataset$data. The original wavenumbers are stored in dataset$original_wavenumbers, when available. Information about the transform is stored in dataset$derivative.

Examples

```
wavenumbers = seq(400, 1000, length.out = 21)

dataset = list(
  data = cbind(
    sample1 = sin(seq(0, 2 * pi, length.out = 21)),
    sample2 = cos(seq(0, 2 * pi, length.out = 21))
```

```
  ),
  wavenumbers = wavenumbers
)

deriv2 = raman_sgolay_derivative(
  dataset,
  window_size = 7,
  polynomial_order = 2
)

dim(deriv2$data)
```

`raman_transform_fourier` *Extract Fourier power features from Raman or SERS spectra*

Description

Computes one-sided Fourier power spectra for each Raman or surface-enhanced Raman spectroscopy (SERS) sample.

Usage

```
raman_transform_fourier(dataset, remove_dc = FALSE)
```

Arguments

`dataset`	A dataset list containing a numeric matrix in `dataset$data`, with wavenumbers in rows and samples in columns.
`remove_dc`	Logical value indicating whether the zero-frequency direct-current component should be excluded from the stored output.

Details

This function follows the `specmine` Raman workflow by preserving the original spectrum in `dataset$data`. Fourier features are stored in `dataset$fourier$data`; their normalized frequencies are stored in `dataset$fourier$frequencies`.

The input data matrix must have wavenumbers in rows and samples in columns. Each sample is transformed independently.

Value

The input `dataset`, with a `fourier` element containing:

`data` Matrix of one-sided Fourier power features, with frequencies in rows and samples in columns.

`frequencies` Numeric vector of normalized frequencies.

`remove_dc` Logical value indicating whether the DC component was removed.

`n_wavenumbers` Number of original spectral variables.

Examples

```
wavenumbers = seq(400, 1000, length.out = 16)

dataset = list(
  data = cbind(
    sample1 = sin(seq(0, 2 * pi, length.out = 16)),
    sample2 = cos(seq(0, 2 * pi, length.out = 16))
  ),
  wavenumbers = wavenumbers
)

transformed = raman_transform_fourier(dataset)

dim(transformed$fourier$data)
```

---

raman_transform_wavelet

Extract Haar wavelet features from Raman or SERS spectra

Description

Computes a multilevel discrete Haar wavelet transform for each Raman or surface-enhanced Raman spectroscopy (SERS) sample.

Usage

```
raman_transform_wavelet(dataset, level = 1)
```

Arguments

`dataset`	A dataset list containing a numeric matrix in `dataset$data`, with wavenumbers in rows and samples in columns.
`level`	Positive integer specifying the number of Haar decomposition levels.

Details

This function preserves the original data in `dataset$data`, following the `specmine` pipeline. Wavelet coefficients are stored in `dataset$wavelet$data`.

The input data matrix must have wavenumbers in rows and samples in columns. If needed, each spectrum is extended by repeating its final intensity value until its length is a power of two.

Value

The input `dataset`, with a `wavelet` element containing:

`data` Matrix of Haar wavelet coefficients, with coefficients in rows and samples in columns.

`filter` The character string "haar".

`level` Number of decomposition levels.

`original_length` Number of original wavenumbers.

`padded_length` Length used after optional padding.

Examples

```
wavenumbers = seq(400, 1100, length.out = 8)

dataset = list(
  data = cbind(
    sample1 = sin(seq(0, 2 * pi, length.out = 8)),
    sample2 = cos(seq(0, 2 * pi, length.out = 8))
  ),
  wavenumbers = wavenumbers
)

transformed = raman_transform_wavelet(
  dataset,
  level = 2
)

dim(transformed$wavelet$data)
```

---

read_csvs_folder *Read all CSV peak files in a folder*

Description

Lists CSV files in a folder and reads them into a named list of data frames.

Usage

```
read_csvs_folder(foldername, verbose = TRUE, ...)
```

Arguments

`foldername`	Path to the folder containing CSV files.
`verbose`	Logical; whether progress messages should be shown.
`...`	Additional arguments passed to `read.csv()`.

Value

A named list of data frames, one per CSV file found in the folder.

Examples

```
folder <- file.path(tempdir(), "peak_csvs")
dir.create(folder, showWarnings = FALSE)
utils::write.csv(data.frame(ppm = c(1, 2), int = c(10, 20)),
                 file.path(folder, "sample1.csv"), row.names = FALSE)
utils::write.csv(data.frame(ppm = c(1, 3), int = c(15, 30)),
                 file.path(folder, "sample2.csv"), row.names = FALSE)
read_csvs_folder(folder, verbose = FALSE)
```

---

read_dataset_csv *Reads a dataset from CSV files*

Description

Reads a dataset from a CSV file containing the data matrix and, optionally, a second CSV file containing metadata.

Usage

```
read_dataset_csv(
  filename.data,
  filename.meta = NULL,
  type = "undefined",
  description = "",
  label.x = NULL,
  label.values = NULL,
  sample.names = NULL,
  format = "row",
  header.col = TRUE,
  header.row = TRUE,
  sep = ",",
  header.col.meta = TRUE,
  header.row.meta = TRUE,
  sep.meta = ","
)
```

Arguments

filename.data	Path to the CSV file containing the data matrix.
filename.meta	Optional path to the CSV file containing metadata.
type	Character string describing the dataset type.
description	Character string with a dataset description.
label.x	Optional x-axis label.
label.values	Optional value labels.

sample.names	Optional sample names.
format	Data layout format, either "row" or "col".
header.col	Logical; whether the data file has a header row.
header.row	Logical; whether the data file has a row names column.
sep	Field separator used in the data file.
header.col.meta	Logical; whether the metadata file has a header row.
header.row.meta	Logical; whether the metadata file has a row names column.
sep.meta	Field separator used in the metadata file.

Value

An object of class dataset created from the input files. This object contains the imported data matrix and, if provided, the sample metadata, together with dataset-level information such as type, description, labels, and sample names. The returned object is intended to be used as input to downstream processing and analysis functions in the package.

Examples

```
data_file <- tempfile(fileext = ".csv")
meta_file <- tempfile(fileext = ".csv")

data_in <- data.frame(
  x1 = c(1.1, 2.2, 3.3),
  x2 = c(4.4, 5.5, 6.6),
  row.names = c("s1", "s2", "s3")
)
metadata_in <- data.frame(
  class = c("A", "B", "A"),
  row.names = c("s1", "s2", "s3")
)

utils::write.csv(data_in, data_file)
utils::write.csv(metadata_in, meta_file)

dataset <- read_dataset_csv(
  filename.data = data_file,
  filename.meta = meta_file,
  format = "row",
  header.row = TRUE,
  header.row.meta = TRUE
)
class(dataset)
```

`read_dataset_dx` *Read a dataset from JDX files*

Description

Reads all JDX spectra files in a folder, combines their intensity values into a data matrix, and optionally attaches sample metadata.

Usage

```
read_dataset_dx(
  folder.data,
  filename.meta = NULL,
  type = "undefined",
  description = "",
  label.x = NULL,
  label.values = NULL,
  header.col.meta = TRUE,
  header.row.meta = TRUE,
  sep.meta = ",",
  verbose = TRUE
)
```

Arguments

`folder.data`	Path to the folder containing JDX files.
`filename.meta`	Optional metadata CSV file.
`type`	Character string describing the dataset type.
`description`	Character string with a dataset description.
`label.x`	Optional x-axis label.
`label.values`	Optional value labels.
`header.col.meta`	Logical; whether the metadata file has a header row.
`header.row.meta`	Logical; whether the metadata file has a row names column.
`sep.meta`	Field separator used in the metadata file.
`verbose`	Logical; whether progress messages should be shown.

Value

A dataset object created from the spectra found in `folder.data` and the optional metadata file.

Examples

```
folder <- tempdir()
meta_file <- tempfile(fileext = ".csv")
metadata_in <- data.frame(sample = c("sample1", "sample2"), class = c("A", "B"))
utils::write.csv(metadata_in, meta_file, row.names = FALSE)
## Not run:
read_dataset_dx(folder, filename.meta = meta_file)

## End(Not run)
```

`read_data_dx` *Read JDX spectra files from a folder*

Description

Reads all `.dx` files in a folder using `readJDX::readJDX()`.

Usage

```
read_data_dx(foldername, debug = 0, verbose = TRUE)
```

Arguments

`foldername`	Path to the folder containing JDX files.
`debug`	Debug level passed to `readJDX::readJDX()`.
`verbose`	Logical; whether progress messages should be shown.

Value

A named list with one imported JDX object per file.

Examples

```
folder <- tempdir()
## Not run:
read_data_dx(folder)

## End(Not run)
```

read_metadata *Reads metadata from a CSV file*

Description

Reads metadata from a CSV file and returns it as a data frame.

Usage

```
read_metadata(filename, header.col = TRUE, header.row = TRUE, sep = ",")
```

Arguments

filename	Path to the metadata CSV file.
header.col	Logical; whether the metadata file has a header row.
header.row	Logical; whether the metadata file has a row names column.
sep	Field separator used in the metadata file.

Value

A data.frame containing the imported metadata. Rows usually correspond to samples and columns correspond to metadata variables.

Examples

```
meta_file <- tempfile(fileext = ".csv")
metadata_in <- data.frame(
  class = c("A", "B", "A"),
  batch = c("b1", "b1", "b2"),
  row.names = c("s1", "s2", "s3")
)
utils::write.csv(metadata_in, meta_file)
read_metadata(meta_file, header.row = TRUE)
```

read_multiple_csvs *Read multiple CSV peak files*

Description

Reads multiple CSV files, one per sample, and returns them as a named list.

Usage

```
read_multiple_csvs(filenames, ext = ".csv", verbose = TRUE, ...)
```

Arguments

filenames	Character vector with file names or file paths.
ext	File extension appended to each file name.
verbose	Logical; whether progress messages should be shown.
...	Additional arguments passed to read.csv().

Value

A named list of data frames, one per input file.

Examples

```
f1 <- tempfile(fileext = ".csv")
f2 <- tempfile(fileext = ".csv")
utils::write.csv(data.frame(ppm = c(1, 2), int = c(10, 20)), f1, row.names = FALSE)
utils::write.csv(data.frame(ppm = c(1, 3), int = c(15, 30)), f2, row.names = FALSE)
read_multiple_csvs(c(f1, f2), ext = "", verbose = FALSE)
```

---

read_spc_nosubhdr *Import for Thermo Galactic's spc file format These functions allow to import .spc files. A detailed description of the .spc file format is available at*

Description

Import for Thermo Galactic's spc file format These functions allow to import .spc files. A detailed description of the .spc file format is available at

Usage

```
read_spc_nosubhdr(
  filename,
  keys.hdr2data = c("fexper", "fres", "fsource"),
  keys.hdr2log = c("fdate", "fpeakpt"),
  keys.log2data = FALSE,
  keys.log2log = TRUE,
  log.txt = TRUE,
  log.bin = FALSE,
  log.disk = FALSE,
  hdr = list(),
  nosubhdr = TRUE,
  no.object = FALSE
)
```

Arguments

filename	The complete file name of the .spc file.
keys.hdr2data, keys.hdr2log, keys.log2data, keys.log2log	character vectors with the names of parameters in the .spc file's log block (log2xxx) or header (hdr2xxx) that should go into the extra data (yyy2data) or into the long.description field of the returned object's log (yyy2log).
log.txt	Should the text part of the .spc file's log block be read?
log.bin, log.disk	Should the normal and on-disk binary parts of the .spc file's log block be read?
hdr	A list with fileheader fields that overwrite the settings of actual file's header.
nosubhdr	Boolean value to decide if the header should be read or not.
no.object	If TRUE, a list with wavelengths, spectra, labels, log and data are returned instead of a hyperSpec object.

Value

A list with imported spectra information.

Author(s)

C. Beleites

References

Reference information for the SPC file format.

Examples

```
filename <- system.file("extdata", "toy.spc", package = "specmine")
if (nzchar(filename)) {
  read_spc_nosubhdr(filename)
}
```

---

recursive_feature_elimination

Recursive Feature Elimination

Description

Performs recursive feature elimination on a data matrix.

Usage

```
recursive_feature_elimination(
  datamat,
  samples.class,
  functions = caret::rfFuncs,
  method = "cv",
  repeats = 5,
  number = 10,
  subsets = 2^(2:4)
)
```

Arguments

datamat	Data matrix with features in rows and samples in columns.
samples.class	Sample class labels.
functions	Caret RFE functions list.
method	Resampling method passed to caret::rfeControl().
repeats	Number of repeats for resampling.
number	Number of resampling folds or iterations.
subsets	Subset sizes to evaluate.

Value

An rfe object.

Examples

```
if (requireNamespace("randomForest", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(20),
    nrow = 4,
    dimnames = list(paste0("x", 1:4), paste0("s", 1:5))
  )
  classes <- factor(c("A", "A", "B", "B", "A"))
  recursive_feature_elimination(datamat, classes, caret::rfFuncs)
}
```

---

remove_data *Remove data*

Description

Removes selected samples, data variables, or metadata variables from a dataset.

Usage

```
remove_data(
  dataset,
  data.to.remove,
  type = "sample",
  by.index = FALSE,
  rebuild.factors = TRUE
)
```

Arguments

dataset	Dataset to modify.
data.to.remove	Data to remove.
type	One of "sample", "data", or "metadata".
by.index	Logical. If TRUE, data.to.remove has indexes.
rebuild.factors	Logical. If TRUE, rebuild factors in metadata.

Value

A dataset object with the same overall structure as the input, where the requested samples, data variables, or metadata variables have been removed according to ‘type‘.

Examples

```
data <- matrix(
  1:9,
  nrow = 3,
  dimnames = list(c("x1", "x2", "x3"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
remove_data(dataset, "s2", type = "sample")
```

---

remove_data_variables *Remove data variables*

Description

Removes x variables from a dataset.

Usage

```
remove_data_variables(dataset, variables.to.remove, by.index = FALSE)
```

Arguments

dataset	Dataset to modify.
variables.to.remove	Variables to remove.
by.index	Logical. If TRUE, variables.to.remove are indexes.

Value

A dataset object with the same structure as the input, where the selected rows of 'dataset$data' have been removed. If no matching variables are found, the input dataset is returned unchanged and a warning is issued.

Examples

```
data <- matrix(
  1:9,
  nrow = 3,
  dimnames = list(c("10", "20", "30"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
remove_data_variables(dataset, "20")
```

---

remove_metadata_variables

Remove metadata variables

Description

Removes metadata variables from a dataset.

Usage

```
remove_metadata_variables(dataset, variables.to.remove)
```

Arguments

dataset	Dataset to modify.
variables.to.remove	Metadata variables to remove.

Value

A dataset object with the same structure as the input, where the selected columns have been removed from 'dataset$metadata'. If no matching metadata fields are found, the dataset is returned unchanged and a warning is issued.

Examples

```
data <- matrix(
  1:6,
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = c("A", "B", "A"),
  batch = c(1, 1, 2),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
remove_metadata_variables(dataset, "batch")
```

`remove_samples` *Remove samples*

Description

Removes samples from a dataset.

Usage

```
remove_samples(dataset, samples.to.remove, rebuild.factors = TRUE)
```

Arguments

`dataset`	Dataset to modify.
`samples.to.remove`	Samples to remove.
`rebuild.factors`	Logical. If TRUE, rebuild factors in metadata.

Value

A dataset object with the same structure as the input, with the selected samples removed from both ‘dataset$data‘ and ‘dataset$metadata‘. If ‘rebuild.factors‘ is ‘TRUE‘, unused factor levels in metadata are dropped.

Examples

```
data <- matrix(
  1:9,
  nrow = 3,
  dimnames = list(c("x1", "x2", "x3"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
```

```
  class = factor(c("A", "B", "A")),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
remove_samples(dataset, "s2")
```

`remove_samples_by_nas` *Remove samples by NAs*

Description

Removes samples with too many missing values.

Usage

```
remove_samples_by_nas(dataset, max.nas = 0, by.percent = FALSE)
```

Arguments

`dataset`	Dataset to modify.
`max.nas`	Maximum number of missing values allowed.
`by.percent`	Logical. If TRUE, max.nas is treated as a percentage.

Value

A dataset object with the same structure as the input, where samples whose number of missing values exceeds ‘max.nas‘ have been removed from both ‘dataset$data‘ and ‘dataset$metadata‘.

Examples

```
data <- matrix(
  c(1, NA, 3,
    4,  5, NA,
    7,  8, 9),
  nrow = 3,
  dimnames = list(c("x1", "x2", "x3"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
remove_samples_by_nas(dataset, max.nas = 0)
```

`remove_samples_by_na_metadata` *Remove samples by NA metadata*

Description

Removes samples with NA in a given metadata variable.

Usage

```
remove_samples_by_na_metadata(dataset, metadata.var)
```

Arguments

`dataset`	Dataset to modify.
`metadata.var`	Metadata variable name.

Value

A dataset object with the same structure as the input, where samples with missing values in the selected metadata variable have been removed.

Examples

```
data <- matrix(
  1:9,
  nrow = 3,
  dimnames = list(c("x1", "x2", "x3"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = c("A", NA, "B"),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
remove_samples_by_na_metadata(dataset, "class")
```

`remove_variables_by_nas` *Remove variables by NAs*

Description

Removes variables with too many missing values.

Usage

```
remove_variables_by_nas(dataset, max.nas = 0, by.percent = FALSE)
```

Arguments

dataset	Dataset to modify.
max.nas	Maximum number of missing values allowed.
by.percent	Logical. If TRUE, max.nas is treated as a percentage.

Value

A dataset object with the same structure as the input, where variables whose number of missing values exceeds ‘max.nas‘ have been removed from ‘dataset$data‘.

Examples

```
data <- matrix(
  c(1,  2, 3,
    NA, 5, NA,
    7,  8, 9),
  nrow = 3,
  dimnames = list(c("x1", "x2", "x3"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
remove_variables_by_nas(dataset, max.nas = 1)
```

---

remove_x_values_by_interval

Remove x values by interval

Description

Removes x values inside an interval.

Usage

```
remove_x_values_by_interval(dataset, min.value, max.value)
```

Arguments

dataset	Dataset to modify.
min.value	Minimum x value.
max.value	Maximum x value.

Value

A dataset object with the same structure as the input, where all variables with x values between ‘min.value‘ and ‘max.value‘, inclusive, have been removed from ‘dataset$data‘.

Examples

```
data <- matrix(
  1:12,
  nrow = 4,
  dimnames = list(c("10", "20", "30", "40"), c("s1", "s2", "s3"))
)
metadata <- data.frame(class = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
remove_x_values_by_interval(dataset, 15, 35)
```

---

spectra_options *Spectra processing options*

Description

Default options used in spectral data processing workflows.

Usage

```
data(spectra_options)
```

Format

A list containing spectral processing options.

Value

A named list of default options used by the package for spectral data processing. Each element stores a processing parameter or lookup table that controls analysis steps such as preprocessing, identification, or scoring.

Examples

```
data(spectra_options)
names(spectra_options)
```

`subset_by_samples_and_xvalues` *Subset by samples and x values*

Description

Selects samples and x values simultaneously.

Usage

```
subset_by_samples_and_xvalues(
  dataset,
  samples,
  variables = NULL,
  by.index = FALSE,
  variable.bounds = NULL,
  rebuild.factors = TRUE
)
```

Arguments

`dataset`	Dataset to subset.
`samples`	Sample indexes.
`variables`	Variables to keep.
`by.index`	Logical. If TRUE, variables are interpreted as indexes.
`variable.bounds`	Optional numeric bounds for x values.
`rebuild.factors`	If TRUE, rebuild factors in metadata.

Value

A dataset object with the same overall structure as the input, containing only the selected samples and selected x values. The returned object includes the corresponding subset of ‘dataset$data‘ and matching rows of ‘dataset$metadata‘; factor levels are rebuilt when ‘rebuild.factors = TRUE‘.

Examples

```
data <- matrix(
  1:12,
  nrow = 4,
  dimnames = list(c("10", "20", "30", "40"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = factor(c("A", "B", "A")),
  row.names = c("s1", "s2", "s3")
)
```

```
dataset <- list(data = data, metadata = metadata)
subset_by_samples_and_xvalues(dataset, samples = c(1, 3), variables = c("10", "30"))
```

`subset_metadata` *Subset metadata*

Description

Selects metadata variables to keep.

Usage

```
subset_metadata(dataset, variables)
```

Arguments

`dataset`	Dataset to subset.
`variables`	Metadata variables to keep.

Value

A dataset object with the same structure as the input, where ‘dataset$metadata‘ contains only the selected metadata variables.

Examples

```
data <- matrix(
  1:6,
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = c("A", "B", "A"),
  batch = c(1, 1, 2),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
subset_metadata(dataset, "class")
```

subset_random_samples *Subset random samples*

Description

Selects a random subset of samples from the dataset.

Usage

```
subset_random_samples(dataset, nsamples)
```

Arguments

dataset	Dataset to subset.
nsamples	Number of samples to select.

Value

A dataset object with the same structure as the input, containing a random subset of ‘nsamples‘ samples.

Examples

```
set.seed(123)
data <- matrix(
  1:12,
  nrow = 3,
  dimnames = list(c("x1", "x2", "x3"), c("s1", "s2", "s3", "s4"))
)
metadata <- data.frame(
  class = factor(c("A", "A", "B", "B")),
  row.names = c("s1", "s2", "s3", "s4")
)
dataset <- list(data = data, metadata = metadata)
subset_random_samples(dataset, 2)
```

subset_samples *Subset samples*

Description

Returns a dataset with a selected set of samples.

Usage

```
subset_samples(dataset, samples, rebuild.factors = TRUE)
```

Arguments

dataset	Dataset to subset.
samples	Vector with indexes or names of the samples to select.
rebuild.factors	If TRUE, rebuild factors in metadata.

Value

A dataset object with the same overall structure as the input, containing only the selected samples in both ‘dataset$data‘ and ‘dataset$metadata‘. If ‘rebuild.factors‘ is ‘TRUE‘, unused factor levels in metadata are dropped.

Examples

```
data <- matrix(
  c(1, 2, 3, 4, 5, 6),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = factor(c("A", "B", "A")),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
subset_samples(dataset, c("s1", "s3"))
```

---

subset_samples_by_metadata_values

Subset samples by metadata values

Description

Selects a set of samples by the value of a metadata variable.

Usage

```
subset_samples_by_metadata_values(dataset, metadata.varname, values)
```

Arguments

dataset	Dataset to subset.
metadata.varname	Metadata variable name.
values	Values to keep.

Value

A dataset object with the same structure as the input, containing only samples whose ‘metadata.varname‘ value matches one of the requested ‘values‘.

Examples

```
data <- matrix(
  c(1, 2, 3, 4, 5, 6),
  nrow = 2,
  dimnames = list(c("x1", "x2"), c("s1", "s2", "s3"))
)
metadata <- data.frame(
  class = factor(c("A", "B", "A")),
  row.names = c("s1", "s2", "s3")
)
dataset <- list(data = data, metadata = metadata)
subset_samples_by_metadata_values(dataset, "class", "A")
```

---

subset_x_values *Subset x values*

Description

Selects rows of the dataset by x values or by row index.

Usage

```
subset_x_values(dataset, variables, by.index = FALSE)
```

Arguments

dataset	Dataset to subset.
variables	Variables to keep.
by.index	Logical. If TRUE, variables are interpreted as row indexes.

Value

A dataset object with the same structure as the input, where ‘dataset$data‘ contains only the selected variables or row indexes.

Examples

```
data <- matrix(
  1:9,
  nrow = 3,
  dimnames = list(c("10", "20", "30"), c("s1", "s2", "s3"))
)
```

```
metadata <- data.frame(group = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
subset_x_values(dataset, c("10", "30"))
```

`subset_x_values_by_interval` *Subset x values by interval*

Description

Selects x values within a numeric interval.

Usage

```
subset_x_values_by_interval(dataset, min.value, max.value)
```

Arguments

`dataset`	Dataset to subset.
`min.value`	Minimum x value.
`max.value`	Maximum x value.

Value

A dataset object with the same structure as the input, where 'dataset$data' contains only variables whose x values fall between 'min.value' and 'max.value', inclusive.

Examples

```
data <- matrix(
  1:12,
  nrow = 4,
  dimnames = list(c("10", "20", "30", "40"), c("s1", "s2", "s3"))
)
metadata <- data.frame(group = c("A", "B", "A"), row.names = c("s1", "s2", "s3"))
dataset <- list(data = data, metadata = metadata)
subset_x_values_by_interval(dataset, 15, 35)
```

summary_var_importance *Summarise variable importance tables*

Description

Keep the top rows of each variable-importance table produced during model comparison.

Usage

```
summary_var_importance(performances, number.rows)
```

Arguments

performances	A list returned by train_models_performance().
number.rows	Number of rows to keep from each variable-importance table.

Value

A list of truncated variable-importance tables, one per model.

Examples

```
performances <- list(
  vips = list(
    rf = data.frame(Mean = c(0.9, 0.7, 0.4), row.names = c("v1", "v2", "v3")),
    svm = data.frame(Mean = c(0.8, 0.6, 0.5), row.names = c("v1", "v2", "v3"))
  )
)
summary_var_importance(performances, 2)
```

train_and_predict *Train a classifier and predict new samples*

Description

Train a classification model from a dataset and use the fitted model to predict the class labels of new samples.

Usage

```
train_and_predict(
  dataset,
  new.samples,
  column.class,
  model,
  validation,
  num.folds = 10,
  num.repeats = 10,
  tunelength = 10,
  tunegrid = NULL,
  metric = NULL,
  summary.function = caret::defaultSummary
)
```

Arguments

dataset	A dataset object containing data and metadata.
new.samples	A data frame or matrix with new samples to classify.
column.class	The metadata column containing the class labels.
model	A model name accepted by caret::train().
validation	Validation method, such as "boot", "boot632", "cv", "repeatedcv", "LOOCV", "LGOCV", or "oob" where supported.
num.folds	Number of folds used in resampling.
num.repeats	Number of repeats used in repeated resampling.
tunelength	Number of tuning levels evaluated by caret.
tunegrid	Optional data frame of tuning parameter combinations.
metric	Optional performance metric used for model selection.
summary.function	Summary function passed to caret::trainControl().

Value

A list with two elements: train.result, the fitted training object, and predictions.result, a data frame with predicted classes for new.samples.

Examples

```
## Not run:
datamat <- matrix(
  rnorm(24),
  nrow = 4,
  dimnames = list(paste0("v", 1:4), paste0("s", 1:6))
)
metadata <- data.frame(class = factor(c("A", "A", "A", "B", "B", "B")))
dataset <- list(data = datamat, metadata = metadata)
```

```
new.samples <- datamat[, 1:2, drop = FALSE]
train_and_predict(dataset, new.samples, "class", model = "rpart", validation = "cv")

## End(Not run)
```

`train_classifier` *Train a classifier*

Description

Train a classifier from a dataset object using metadata or data-derived class labels and a resampling strategy supported by `caret`.

Usage

```
train_classifier(
  dataset,
  column.class,
  model,
  validation,
  num.folds = 10,
  num.repeats = 10,
  tunelength = 10,
  tunegrid = NULL,
  metric = NULL,
  summary.function = caret::defaultSummary,
  class.in.metadata = TRUE
)
```

Arguments

`dataset`	A dataset object.
`column.class`	The metadata column containing the class labels.
`model`	A model name accepted by `caret::train()`.
`validation`	Validation method used in training.
`num.folds`	Number of folds used in resampling.
`num.repeats`	Number of repeats used in repeated resampling.
`tunelength`	Number of tuning levels evaluated by `caret`.
`tunegrid`	Optional data frame of tuning parameter combinations.
`metric`	Optional performance metric used for model selection.
`summary.function`	Summary function passed to `caret::trainControl()`.
`class.in.metadata`	Logical; if `TRUE`, class labels are taken from `dataset$metadata`, otherwise from `dataset$data`.

Value

A caret training object returned by caret::train().

Examples

```
## Not run:
datamat <- matrix(
  rnorm(24),
  nrow = 4,
  dimnames = list(paste0("v", 1:4), paste0("s", 1:6))
)
metadata <- data.frame(class = factor(c("A", "A", "A", "B", "B", "B")))
dataset <- list(data = datamat, metadata = metadata)
train_classifier(dataset, "class", model = "rpart", validation = "cv")

## End(Not run)
```

---

train_models_performance

Train multiple models and compare their performance

Description

Train a set of models, collect their resampling performance, optionally compute variable importance, and store fitted models and tuning summaries.

Usage

```
train_models_performance(
  dataset,
  models,
  column.class,
  validation,
  num.folds = 10,
  num.repeats = 10,
  tunelength = 10,
  tunegrid = NULL,
  metric = NULL,
  summary.function = "default",
  class.in.metadata = TRUE,
  compute.varimp = TRUE
)
```

Arguments

dataset	A dataset object.
models	A character vector with model names accepted by caret::train().

column.class	The metadata column containing the class labels.
validation	Validation method used in training.
num.folds	Number of folds used in resampling.
num.repeats	Number of repeats used in repeated resampling.
tunelength	Number of tuning levels evaluated by caret.
tunegrid	Optional list of tuning grids, one per model.
metric	Optional performance metric used for model selection.
summary.function	Summary function, or "default" to select the package default.
class.in.metadata	Logical; if TRUE, class labels are taken from metadata.
compute.varimp	Logical; if TRUE, variable importance is computed.

Value

A list containing model performance, variable importance, full tuning results, best tuning settings, optional confusion matrices, and final fitted models.

Examples

```
## Not run:
datamat <- matrix(
  rnorm(24),
  nrow = 4,
  dimnames = list(paste0("v", 1:4), paste0("s", 1:6))
)
metadata <- data.frame(class = factor(c("A", "A", "A", "B", "B", "B")))
dataset <- list(data = datamat, metadata = metadata)
train_models_performance(
  dataset,
  models = c("rpart"),
  column.class = "class",
  validation = "cv",
  compute.varimp = FALSE
)

## End(Not run)
```

---

tsne_analysis_dataset *t-SNE analysis*

Description

Performs t-distributed Stochastic Neighbor Embedding (t-SNE) dimensionality reduction on a specmine dataset.

Usage

```
tsne_analysis_dataset(
  dataset,
  n_components = 2,
  perplexity = 30,
  max_iter = 1000,
  theta = 0.5,
  eta = "auto",
  scale = FALSE,
  seed = 42,
  write.file = FALSE,
  file.out = NULL,
  ...
)
```

Arguments

dataset	Dataset to analyse.
n_components	Number of dimensions in the embedding.
perplexity	Perplexity parameter controlling the balance between local and global structure. It must be sufficiently small relative to the number of samples.
max_iter	Maximum number of optimization iterations.
theta	Speed-accuracy trade-off parameter. A value of zero performs exact t-SNE, whereas larger values are faster but approximate.
eta	Learning rate. The default value "auto" is converted to a numeric value based on the number of samples.
scale	Logical indicating whether the data should be scaled before t-SNE.
seed	Random seed used for reproducibility.
write.file	Logical indicating whether the embedding should be written to a CSV file.
file.out	Output file prefix used when write.file = TRUE.
...	Additional arguments passed to Rtsne::Rtsne().

Value

A list containing the t-SNE embedding and the analysis parameters.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
```

```
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  tsne.result <- tsne_analysis_dataset(
    dataset,
    n_components = 2,
    perplexity = 2,
    max_iter = 250,
    seed = 42
  )
}
```

`tsne_kmeans_plot2D` *t-SNE 2D k-means plot*

Description

Creates a two-dimensional t-SNE plot coloured by k-means clusters.

Usage

```
tsne_kmeans_plot2D(
  dataset,
  tsne.result,
  num.clusters = 3,
  dims = c(1, 2),
  kmeans.result = NULL,
  labels = FALSE,
  bw = FALSE,
  ellipses = FALSE,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

`dataset`	Dataset to cluster.
`tsne.result`	Result returned by `tsne_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`dims`	Two embedding dimensions to plot.

kmeans.result	Optional result returned by clustering().
labels	Logical indicating whether sample labels should be displayed.
bw	Logical indicating whether a black-and-white plot should be used.
ellipses	Logical indicating whether group ellipses should be displayed.
leg.pos	Legend position.
xlim	Optional x-axis limits.
ylim	Optional y-axis limits.

Value

A ggplot object.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  tsne.result <- tsne_analysis_dataset(
    dataset,
    perplexity = 2,
    max_iter = 250,
    seed = 42
  )

  kmeans.result <- list(
    cluster = rep(1:2, each = 4)
  )

  tsne_kmeans_plot2D(
    dataset,
    tsne.result,
    num.clusters = 2,
    kmeans.result = kmeans.result
  )
}
```

`tsne_kmeans_plot3D` *t-SNE 3D k-means plot*

Description

Creates an interactive three-dimensional t-SNE plot coloured by k-means clusters.

Usage

```
tsne_kmeans_plot3D(
  dataset,
  tsne.result,
  num.clusters = 3,
  dims = c(1, 2, 3),
  kmeans.result = NULL,
  title = "t-SNE 3D K-means Plot"
)
```

Arguments

`dataset`	Dataset to cluster.
`tsne.result`	Result returned by `tsne_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`dims`	Three embedding dimensions to plot.
`kmeans.result`	Optional result returned by `clustering()`.
`title`	Plot title.

Value

A `plotly` htmlwidget.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE) &&
    requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
```

```
    class = factor(rep(c("A", "B"), length.out = 8))
  )
)

tsne.result <- tsne_analysis_dataset(
  dataset,
  n_components = 3,
  perplexity = 2,
  max_iter = 250,
  seed = 42
)

kmeans.result <- list(
  cluster = rep(1:2, each = 4)
)

tsne_kmeans_plot3D(
  dataset,
  tsne.result,
  num.clusters = 2,
  kmeans.result = kmeans.result
)
}
```

`tsne_pairs_kmeans_plot` *t-SNE pairs k-means plot*

Description

Creates a pairs plot for t-SNE dimensions coloured by k-means clusters.

Usage

```
tsne_pairs_kmeans_plot(
  dataset,
  tsne.result,
  num.clusters = 3,
  kmeans.result = NULL,
  dims = 1:min(5, ncol(tsne.result$embedding))
)
```

Arguments

`dataset`	Dataset to cluster.
`tsne.result`	Result returned by `tsne_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`kmeans.result`	Optional result returned by `clustering()`.
`dims`	Embedding dimensions to include in the pairs plot.

Value

A GGally pairs plot object.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE) &&
    requireNamespace("GGally", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  tsne.result <- tsne_analysis_dataset(
    dataset,
    n_components = 3,
    perplexity = 2,
    max_iter = 250,
    seed = 42
  )

  kmeans.result <- list(
    cluster = rep(1:2, each = 4)
  )

  tsne_pairs_kmeans_plot(
    dataset,
    tsne.result,
    num.clusters = 2,
    kmeans.result = kmeans.result,
    dims = 1:3
  )
}
```

---

tsne_pairs_plot *t-SNE pairs plot*

Description

Creates a pairs plot for selected t-SNE dimensions.

Usage

```
tsne_pairs_plot(
  dataset,
  tsne.result,
  column.class = NULL,
  dims = 1:min(5, ncol(tsne.result$embedding)),
  ...
)
```

Arguments

dataset	Dataset used to calculate the embedding.
tsne.result	Result returned by tsne_analysis_dataset().
column.class	Optional metadata column used to colour samples.
dims	Embedding dimensions to include in the pairs plot.
...	Additional arguments passed to GGally::ggpairs().

Value

A GGally pairs plot object.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE) &&
    requireNamespace("GGally", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  tsne.result <- tsne_analysis_dataset(
    dataset,
    n_components = 3,
    perplexity = 2,
    max_iter = 250,
    seed = 42
  )

  tsne_pairs_plot(
```

```
    dataset,
    tsne.result,
    column.class = "class",
    dims = 1:3
  )
}
```

`tsne_scoresplot2D` *t-SNE 2D scores plot*

Description

Creates a two-dimensional scores plot from a t-SNE result.

Usage

```
tsne_scoresplot2D(
  dataset,
  tsne.result,
  column.class = NULL,
  dims = c(1, 2),
  labels = FALSE,
  ellipses = FALSE,
  bw = FALSE,
  pallette = 2,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

`dataset`	Dataset used to calculate the embedding.
`tsne.result`	Result returned by `tsne_analysis_dataset()`.
`column.class`	Optional metadata column used to colour or group samples.
`dims`	Two embedding dimensions to plot.
`labels`	Logical indicating whether sample labels should be displayed.
`ellipses`	Logical indicating whether group ellipses should be displayed.
`bw`	Logical indicating whether a black-and-white plot should be used.
`pallette`	RColorBrewer palette number.
`leg.pos`	Legend position.
`xlim`	Optional x-axis limits.
`ylim`	Optional y-axis limits.

Value

A ggplot object.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  tsne.result <- tsne_analysis_dataset(
    dataset,
    perplexity = 2,
    max_iter = 250,
    seed = 42
  )

  tsne_scoresplot2D(
    dataset,
    tsne.result,
    column.class = "class"
  )
}
```

---

tsne_scoresplot3D *t-SNE 3D scores plot*

Description

Creates an interactive three-dimensional scores plot from a t-SNE result.

Usage

```
tsne_scoresplot3D(
  dataset,
  tsne.result,
```

```
  column.class = NULL,
  dims = c(1, 2, 3),
  title = "t-SNE 3D Scores Plot"
)
```

Arguments

dataset	Dataset used to calculate the embedding.
tsne.result	Result returned by tsne_analysis_dataset().
column.class	Optional metadata column used to colour samples.
dims	Three embedding dimensions to plot.
title	Plot title.

Value

A plotly htmlwidget.

Examples

```
if (requireNamespace("Rtsne", quietly = TRUE) &&
    requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  tsne.result <- tsne_analysis_dataset(
    dataset,
    n_components = 3,
    perplexity = 2,
    max_iter = 250,
    seed = 42
  )

  tsne_scoresplot3D(
    dataset,
    tsne.result,
    column.class = "class"
  )
}
```

umap_analysis_dataset *UMAP analysis*

Description

Performs Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction on a specmine dataset.

Usage

```
umap_analysis_dataset(
  dataset,
  n_components = 2,
  n_neighbors = 15,
  min_dist = 0.1,
  metric = "euclidean",
  scale = FALSE,
  ret_model = FALSE,
  seed = 42,
  write.file = FALSE,
  file.out = NULL,
  ...
)
```

Arguments

dataset	Dataset to analyse.
n_components	Number of dimensions in the embedding.
n_neighbors	Number of nearest neighbours.
min_dist	Minimum distance between points in the embedding.
metric	Distance metric passed to uwot::umap().
scale	Logical indicating whether the data should be scaled.
ret_model	Logical indicating whether the UMAP model should be returned.
seed	Random seed used for reproducibility.
write.file	Logical indicating whether the embedding should be written to a CSV file.
file.out	Output file prefix used when write.file = TRUE.
...	Additional arguments passed to uwot::umap().

Value

A list containing the UMAP embedding and the analysis parameters. When ret_model = TRUE, the fitted UMAP model is also returned.

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(paste0("x", 1:5), paste0("s", 1:8))
  )
  dataset <- list(
    data = datamat,
    metadata = data.frame(class = factor(rep(c("A", "B"), length.out = 8)))
  )
  result <- umap_analysis_dataset(
    dataset,
    n_components = 2,
    n_neighbors = 3,
    seed = 42
  )
}
```

---

umap_kmeans_plot2D *UMAP 2D k-means plot*

Description

Creates a two-dimensional UMAP plot coloured by k-means clusters.

Usage

```
umap_kmeans_plot2D(
  dataset,
  umap.result,
  num.clusters = 3,
  dims = c(1, 2),
  kmeans.result = NULL,
  labels = FALSE,
  bw = FALSE,
  ellipses = FALSE,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

dataset	Dataset to cluster.
umap.result	Result returned by umap_analysis_dataset().

num.clusters	Number of k-means clusters.
dims	Two embedding dimensions to plot.
kmeans.result	Optional result returned by clustering().
labels	Logical indicating whether sample labels should be displayed.
bw	Logical indicating whether a black-and-white plot should be used.
ellipses	Logical indicating whether group ellipses should be displayed.
leg.pos	Legend position.
xlim	Optional x-axis limits.
ylim	Optional y-axis limits.

Value

A ggplot object.

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  umap.result <- umap_analysis_dataset(
    dataset,
    n_components = 2,
    n_neighbors = 3,
    seed = 42
  )

  umap_kmeans_plot2D(
    dataset,
    umap.result,
    num.clusters = 2,
    dims = c(1, 2)
  )
}
```

`umap_kmeans_plot3D` *UMAP 3D k-means plot*

Description

Creates an interactive three-dimensional UMAP plot coloured by k-means clusters.

Usage

```
umap_kmeans_plot3D(
  dataset,
  umap.result,
  num.clusters = 3,
  dims = c(1, 2, 3),
  kmeans.result = NULL,
  title = "UMAP 3D K-means Plot"
)
```

Arguments

`dataset`	Dataset to cluster.
`umap.result`	Result returned by `umap_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`dims`	Three embedding dimensions to plot.
`kmeans.result`	Optional result returned by `clustering()`.
`title`	Plot title.

Value

A `plotly` htmlwidget.

Examples

```
if (requireNamespace("uwot", quietly = TRUE) &&
    requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
```

```
    class = factor(rep(c("A", "B"), length.out = 8))
  )
)

umap.result <- umap_analysis_dataset(
  dataset,
  n_components = 3,
  n_neighbors = 3,
  seed = 42
)

umap_kmeans_plot3D(
  dataset,
  umap.result,
  num.clusters = 2,
  dims = c(1, 2, 3)
)
}
```

`umap_pairs_kmeans_plot` *UMAP pairs k-means plot*

Description

Creates a pairs plot for UMAP dimensions coloured by k-means clusters.

Usage

```
umap_pairs_kmeans_plot(
  dataset,
  umap.result,
  num.clusters = 3,
  kmeans.result = NULL,
  dims = 1:min(5, ncol(umap.result$embedding))
)
```

Arguments

`dataset`	Dataset to cluster.
`umap.result`	Result returned by `umap_analysis_dataset()`.
`num.clusters`	Number of k-means clusters.
`kmeans.result`	Optional result returned by `clustering()`.
`dims`	Embedding dimensions to include in the pairs plot.

Value

A GGally pairs plot object.

Examples

```
if (requireNamespace("uwot", quietly = TRUE) &&
    requireNamespace("GGally", quietly = TRUE)) {
  datamat <- matrix(
    rnorm(40),
    nrow = 5,
    dimnames = list(
      paste0("x", 1:5),
      paste0("s", 1:8)
    )
  )

  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )

  umap.result <- umap_analysis_dataset(
    dataset,
    n_components = 3,
    n_neighbors = 3,
    seed = 42
  )

  umap_pairs_kmeans_plot(
    dataset,
    umap.result,
    num.clusters = 2,
    dims = 1:3
  )
}
```

---

umap_pairs_plot *UMAP pairs plot*

Description

Creates a pairs plot for selected UMAP dimensions.

Usage

```
umap_pairs_plot(
  dataset,
  umap.result,
  column.class = NULL,
  dims = 1:min(5, ncol(umap.result$embedding)),
  ...
)
```

Arguments

dataset	Dataset used to calculate the embedding.
umap.result	Result returned by umap_analysis_dataset().
column.class	Optional metadata column used to colour samples.
dims	Embedding dimensions to include in the pairs plot.
...	Additional arguments passed to GGally::ggpairs().

Value

A GGally pairs plot object.

Examples

```
if (requireNamespace("uwot", quietly = TRUE) &&
    requireNamespace("GGally", quietly = TRUE)) {
  datamat <- matrix(rnorm(40), nrow = 5)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )
  result <- umap_analysis_dataset(
    dataset,
    n_components = 3,
    n_neighbors = 3
  )
  umap_pairs_plot(
    dataset,
    result,
    column.class = "class",
    dims = 1:3
  )
}
```

umap_scoresplot2D *UMAP 2D scores plot*

Description

Creates a two-dimensional scores plot from a UMAP result.

Usage

```
umap_scoresplot2D(
  dataset,
  umap.result,
  column.class = NULL,
  dims = c(1, 2),
  labels = FALSE,
  ellipses = FALSE,
  bw = FALSE,
  pallette = 2,
  leg.pos = "right",
  xlim = NULL,
  ylim = NULL
)
```

Arguments

dataset	Dataset used to calculate the embedding.
umap.result	Result returned by umap_analysis_dataset().
column.class	Optional metadata column used to colour or group samples.
dims	Two embedding dimensions to plot.
labels	Logical indicating whether sample labels should be displayed.
ellipses	Logical indicating whether group ellipses should be displayed.
bw	Logical indicating whether a black-and-white plot should be used.
pallette	RColorBrewer palette number.
leg.pos	Legend position.
xlim	Optional x-axis limits.
ylim	Optional y-axis limits.

Value

A ggplot object.

Examples

```
if (requireNamespace("uwot", quietly = TRUE)) {
  datamat <- matrix(rnorm(40), nrow = 5)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )
  result <- umap_analysis_dataset(dataset, n_neighbors = 3)
  umap_scoresplot2D(dataset, result, column.class = "class")
}
```

---

umap_scoresplot3D *UMAP 3D scores plot*

Description

Creates an interactive three-dimensional scores plot from a UMAP result.

Usage

```
umap_scoresplot3D(
  dataset,
  umap.result,
  column.class = NULL,
  dims = c(1, 2, 3),
  title = "UMAP 3D Scores Plot"
)
```

Arguments

dataset	Dataset used to calculate the embedding.
umap.result	Result returned by umap_analysis_dataset().
column.class	Optional metadata column used to colour samples.
dims	Three embedding dimensions to plot.
title	Plot title.

Value

A plotly htmlwidget.

Examples

```
if (requireNamespace("uwot", quietly = TRUE) &&
    requireNamespace("plotly", quietly = TRUE)) {
  datamat <- matrix(rnorm(40), nrow = 5)
  dataset <- list(
    data = datamat,
    metadata = data.frame(
      class = factor(rep(c("A", "B"), length.out = 8))
    )
  )
  result <- umap_analysis_dataset(
    dataset,
    n_components = 3,
    n_neighbors = 3
  )
  umap_scoresplot3D(dataset, result, column.class = "class")
}
```

Index