

# Package: notameStats (via r-universe)

May 31, 2026

**Type** Package

**Title** Workflow for non-targeted LC-MS metabolic profiling

**Version** 1.3.0

**Description** Provides univariate and multivariate statistics for feature prioritization in untargeted LC-MS metabolomics research.

**License** MIT + file LICENSE

**Encoding** UTF-8

**biocViews** BiomedicalInformatics, Metabolomics, DataImport, MassSpectrometry, BatchEffect, MultipleComparison, Normalization, QualityControl, Visualization, Preprocessing

**Depends** R (>= 4.5.0), SummarizedExperiment,

**Imports** BiocGenerics, BiocParallel, broom, dplyr, methods, notame, stats, tibble, tidyr, utils

**Suggests** BiocStyle, car, knitr, lmerTest, missForest, mixOmics, MuMIn, MUVR2, notameViz, PERMANOVA, PK, randomForest, rmcrr, testthat

**URL** <https://github.com/hanhineva-lab/notameStats>

**BugReports** <https://github.com/hanhineva-lab/notameStats/issues>

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Config/testthat/parallel** true

**Config/pak/sysreqs** libicu-dev zlib1g-dev

**Repository** <https://bioc.r-universe.dev>

**Date/Publication** 2026-04-28 13:05:39 UTC

**RemoteUrl** <https://github.com/bioc/notameStats>

**RemoteRef** HEAD

**RemoteSha** 66400ee40bd842bac05fdf272dc440f09f6ff200

## Contents

cohens_d . . . . .	2
fit_rf . . . . .	3
fold_change . . . . .	4
importance_rf . . . . .	5
muvr_analysis . . . . .	5
perform_auc . . . . .	8
perform_correlation_tests . . . . .	9
perform_homoscedasticity_tests . . . . .	10
perform_kruskal_wallis . . . . .	11
perform_lm . . . . .	12
perform_lm_anova . . . . .	13
perform_lmer . . . . .	14
perform_logistic . . . . .	16
perform_non_parametric . . . . .	17
perform_oneway_anova . . . . .	18
perform_permanova . . . . .	19
perform_t_test . . . . .	20
pls . . . . .	21
pls_da . . . . .	23
summarize_results . . . . .	26
summary_statistics . . . . .	27
<b>Index</b>	<b>28</b>

---

cohens_d	<i>Cohen's D</i>
----------	------------------

---

### Description

Computes Cohen's D for each feature. If time and ID are supplied, change between two time points is computed for each subject, and Cohen's d is computed from the changes.

### Usage

```
cohens_d(object, group, id = NULL, time = NULL, assay.type = NULL)
```

### Arguments

object	a <a href="#">SummarizedExperiment</a> object
group	character, name of the group column
id	character, name of the subject ID column
time	character, name of the time column
assay.type	character, assay to be used in case of multiple assays

**Value**

A data frame with Cohen's d for each feature.

**Examples**

```
data(toy_notame_set, package = "notame")
d_results <- cohens_d(notame::drop_qcs(toy_notame_set), group = "Group")
d_results_time <- cohens_d(notame::drop_qcs(toy_notame_set),
  group = "Group", time = "Time", id = "Subject_ID"
)
```

---

fit\_rf

*Fit Random Forest*

---

**Description**

Fits a random forest, where given response column in pheno data is predicted using the features. Can be used both for classification and regression. For more information, see the documentation of [randomForest](#). After fitting the random forest, use [importance\\_rf](#) as a shortcut for getting the feature importance in random forest prediction.

**Usage**

```
fit_rf(
  object,
  y,
  all_features = FALSE,
  covariates = NULL,
  importance = TRUE,
  assay.type = NULL,
  ...
)
```

**Arguments**

object	a <a href="#">SummarizedExperiment</a> object
y	character, column name of pheno data giving the dependent variable of the model
all_features	logical, should all features be included in the model? if FALSE, flagged features are left out
covariates	character, column names of pheno data to use as covariates in the model, in addition to molecular features
importance	Should importance of features be assessed?
assay.type	character, assay to be used in case of multiple assays
...	other parameters passed to <a href="#">randomForest</a>

**Value**

An object of class randomForest.

**See Also**

[randomForest](#), [importance\\_rf](#)

**Examples**

```
data(toy_notame_set, package = "notame")
rf <- fit_rf(toy_notame_set, y = "Group")
rf
importance_rf(rf)
```

---

fold\_change

*Fold change*

---

**Description**

Computes fold change between each group for each feature.

**Usage**

```
fold_change(object, group, assay.type = NULL)
```

**Arguments**

object	a <a href="#">SummarizedExperiment</a> object
group	character, name of the group column
assay.type	character, assay to be used in case of multiple assays

**Value**

A data frame with fold changes for each feature.

**Examples**

```
data(toy_notame_set, package = "notame")
# Between groups
fc <- fold_change(toy_notame_set, group = "Group")
# Between time points
fc <- fold_change(toy_notame_set, group = "Time")
```

---

importance_rf	<i>Feature importance in random forest</i>
---------------	--

---

**Description**

Extracts feature importance in random forest in a nice format.

**Usage**

```
importance_rf(rf)
```

**Arguments**

rf                    An object of class randomForest

**Value**

A data frame of feature importance.

**See Also**

[randomForest](#), [fit\\_rf](#)

**Examples**

```
data(toy_notame_set, package = "notame")
rf <- fit_rf(toy_notame_set, y = "Group")
rf
importance_rf(rf)
```

---

muvr_analysis	<i>Multivariate modelling with minimally biased variable selection (MUVR)</i>
---------------	---

---

**Description**

A wrapper around [MUVR2](#) (random forest, PLS(-DA)) and [MUVR2\\_EN](#) (elastic net) functions from the MUVR2 package.

**Usage**

```

muvr_analysis(
  object,
  y = NULL,
  id = NULL,
  multi_level = FALSE,
  multi_level_var = NULL,
  covariates = NULL,
  static_covariates = NULL,
  all_features = FALSE,
  nRep = 50,
  nOuter = 6,
  nInner = nOuter - 1,
  varRatio = 0.75,
  method = c("PLS", "RF"),
  assay.type = NULL,
  ...
)

```

**Arguments**

object	a <a href="#">SummarizedExperiment</a> object
y	character, column name in pheno data of the target variable
id	character, column name in pheno data of the subject ID variable in case of repeated measurements
multi_level	logical, whether multi-level modeling should be applied, see <a href="#">Details</a>
multi_level_var	character, column name in pheno data of the variable for splitting the data in multi-level modeling
covariates, static_covariates	character, column names of pheno data to use as covariates in the model, in addition to molecular features. <code>static_covariates</code> are ignored for non-multi-level models. For multi-level models, the change in <code>covariates</code> is computed, while <code>static_covariates</code> are taken from the first time point.
all_features	logical, should all features be included in the model? if FALSE, flagged features are left out
nRep	Number of repetitions of double CV, parameter of MUVR
nOuter	Number of outer CV loop segments, parameter of MUVR
nInner	Number of inner CV loop segments, parameter of MUVR
varRatio	Ratio of variables to include in subsequent inner loop iteration, parameter of MUVR
method	Multivariate method. Supports 'PLS', 'RF' and 'EN'
assay.type	character, assay to be used in case of multiple assays
...	other parameters to <a href="#">MUVR2</a> or <a href="#">MUVR2_EN</a> and <a href="#">getVar</a> (when <code>method == "EN"</code> )

## Details

This function is now using the MUVR2 package, characterized as an upgrade extending the original MUVR package by the inclusion of elastic net regression (EN) and some functionality not covered by this wrapper. Elastic net regression supports covariate adjustment by suppressing regularization of specified features from the regularization procedure. Note that this is different from simply including covariates such as sex. EN also differs from PLS and RF in that no recursive variable elimination is performed, so an additional scheme is used to obtain the 'min', 'mid' and 'max' models using [getVar](#).

Sex would be entered as a static covariate, since the change in sex is zero for all individuals, so computing the change and using that as a covariate does not make sense.

Note that there are several more plots available in MUVR2 for inspecting the results, notably [plotMV](#), [plotStability](#) and [plotVIRank](#). Many of these return different plots depending on the model specification.

## Value

A MUVR object.

## See Also

[MUVR2](#) [MUVR2\\_EN](#) [getVar](#) [plotMV](#) [plotStability](#) [plotVIRank](#) [plotVAL](#)

## Examples

```
data(toy_notame_set, package = "notame")
ex_set <- notame::drop_qcs(toy_notame_set)[1:10, ]
ex_set$Injection_order <- as.numeric(ex_set$Injection_order)
# Simple PLS regression model
pls_model <- muvr_analysis(ex_set,
  y = "Injection_order", nRep = 2, method = "PLS")

# RF classification with covariate and repeated measures (not longitudinal)
rf_model <- muvr_analysis(ex_set, y = "Group", id = "Subject_ID",
  nRep = 2, method = "RF", covariates = "Injection_order")

# RF classification on multilevel variable comparing levels of y
rf_model_ <- muvr_analysis(ex_set,
  y = "Group", multi_level = TRUE, id = "Subject_ID",
  multi_level_var = "Time", method = "RF", nRep = 2)

# EN regression on multilevel variable with covariate and static covariate
ex_set$Group <- as.numeric(ex_set$Group)
en_model <- muvr_analysis(ex_set, id = "Subject_ID",
  multi_level = TRUE, multi_level_var = "Time",
  covariates = "Injection_order", static_covariates = "Group",
  method = "EN", nRep = 2)
```

---

perform_auc	<i>Area under curve</i>
-------------	-------------------------

---

### Description

Compute area under curve (AUC) for each subject and feature. Creates a pseudo SummarizedExperiment object, where the "samples" are subjects (or subject/group combinations in case the same subjects are submitted to different treatments) and the "abundances" are AUCs. This object can then be used to compute results of e.g. t-tests of AUCs between groups.

### Usage

```
perform_auc(object, time, subject, group, assay.type = NULL)
```

### Arguments

object            a [SummarizedExperiment](#) object  
time, subject, group        column names of pheno data holding time, subject and group labels  
assay.type        character, assay to be used in case of multiple assays

### Value

A pseudo SummarizedExperiment object with the AUCs.

### See Also

[auc](#)

### Examples

```
data(toy_notame_set, package = "notame")  
# Drop QC samples before computing AUCs  
aucs <- perform_auc(notame::drop_qcs(toy_notame_set), time = "Time",  
                    subject = "Subject_ID", group = "Group")  
# t-test with the AUCs  
t_test_results <- perform_t_test(aucs, formula_char = "Feature ~ Group")
```

---

`perform_correlation_tests`*Correlation test*

---

### Description

Performs a correlation test between two sets of variables. All the variables must be either feature names or column names of pheno data (sample information). There are two ways to use this function: either provide a set of variables as `x`, and all correlations between those variables are computed. Or provide two distinct sets of variables `x`, `y` and correlations between each `x` variable and each `y` variable are computed.

### Usage

```
perform_correlation_tests(  
  object,  
  x,  
  y = x,  
  id = NULL,  
  object2 = NULL,  
  fdr = TRUE,  
  all_pairs = TRUE,  
  duplicates = FALSE,  
  assay.type1 = NULL,  
  assay.type2 = NULL,  
  ...  
)
```

### Arguments

<code>object</code>	a <a href="#">SummarizedExperiment</a> object
<code>x</code>	character vector, names of variables to be correlated
<code>y</code>	character vector, either identical to <code>x</code> (the default) or a distinct set of variables to be correlated against <code>x</code>
<code>id</code>	character, column name for subject IDs. If provided, the correlation will be computed using the <code>rmcorr</code> package
<code>object2</code>	optional second object. If provided, <code>x</code> variables will be taken from <code>object</code> and <code>y</code> variables will be taken from <code>object2</code> . Both objects should have the same number of samples.
<code>fdr</code>	logical, whether p-values from the correlation test should be adjusted with FDR correction
<code>all_pairs</code>	logical, whether all pairs between <code>x</code> and <code>y</code> should be tested. If <code>FALSE</code> , <code>x</code> and <code>y</code> give the exact pairs of variables to test, and should have the same length.

<code>duplicates</code>	logical, whether correlations should be duplicated. If TRUE, each correlation will be included in the results twice, where the order of the variables '(which is x and which is y) is changed. Can be useful for e.g. plotting a heatmap of the results, see examples of <a href="#">plot_effect_heatmap</a> .
<code>assay.type1</code>	character, assay of object(1) to be used in case of multiple assays
<code>assay.type2</code>	character, assay of object2 to be used in case of multiple assays
<code>...</code>	other parameters passed to <a href="#">cor.test</a> , such as method

**Value**

A data frame with the results of correlation tests: the pair of variables, correlation coefficient and p-value.

**See Also**

[cor.test](#), [rmcorr](#)

**Examples**

```
data(toy_notame_set, package = "notame")
# Correlations between all features
correlations <- perform_correlation_tests(toy_notame_set,
  x = rownames(toy_notame_set), id = "Subject_ID")

# Spearman Correlations between features and sample information variables
# Drop QCs and convert time to numeric
no_qc <- notame::drop_qcs(toy_notame_set)
no_qc$Time <- as.numeric(no_qc$Time)
correlations <- perform_correlation_tests(no_qc,
  x = rownames(toy_notame_set),
  y = c("Time", "Injection_order"), method = "spearman"
)

# Correlations between variables from two distinct objects
cross_object_cor <- perform_correlation_tests(toy_notame_set,
  x = rownames(toy_notame_set),
  object2 = toy_notame_set,
  y = rownames(toy_notame_set),
  all_pairs = FALSE
)
```

---

`perform_homoscedasticity_tests`

*Test homoscedasticity*

---

**Description**

Performs Bartlett's, Levene's and Fligner-Killeen tests for equality of variances.

### Usage

```
perform_homoscedasticity_tests(  
  object,  
  formula_char,  
  all_features = FALSE,  
  assay.type = NULL  
)
```

### Arguments

object	a <a href="#">SummarizedExperiment</a> object
formula_char	character, the formula to be used in the linear model (see Details)
all_features	should all features be included in FDR correction?
assay.type	character, assay to be used in case of multiple assays

### Details

The model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting. For example, if testing for equality of variances in study groups, use "Feature ~ Group".

### Value

A data frame with the results.

### See Also

[bartlett.test](#), [leveneTest](#), [fligner.test](#)

### Examples

```
data(toy_notame_set, package = "notame")  
perform_homoscedasticity_tests(toy_notame_set,  
  formula_char = "Feature ~ Group")
```

---

perform\_kruskal\_wallis

*Kruskal-Wallis rank-sum test*

---

### Description

Performs Kruskal-Wallis rank-sum test for equality.

**Usage**

```
perform_kruskal_wallis(  
  object,  
  formula_char,  
  all_features = FALSE,  
  assay.type = NULL  
)
```

**Arguments**

object	a <a href="#">SummarizedExperiment</a> object
formula_char	character, the formula to be used in the linear model (see <a href="#">Details</a> )
all_features	should all features be included in FDR correction?
assay.type	character, assay to be used in case of multiple assays

**Details**

The model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting. For example, if testing for equality of means in study groups, use "Feature ~ Group".

**Value**

A data frame with the results.

**See Also**

[kruskal.test](#)

**Examples**

```
data(toy_notame_set, package = "notame")  
perform_kruskal_wallis(toy_notame_set, formula_char = "Feature ~ Group")
```

---

perform\_lm

*Linear models*

---

**Description**

Fits a linear model separately for each feature. Returns all relevant statistics.

**Usage**

```
perform_lm(object, formula_char, all_features = FALSE, assay.type = NULL, ...)
```

**Arguments**

object a [SummarizedExperiment](#) object  
 formula\_char character, the formula to be used in the linear model (see [Details](#))  
 all\_features should all features be included in FDR correction?  
 assay.type character, assay to be used in case of multiple assays  
 ... additional parameters passed to [lm](#)

**Details**

The linear model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting, see the example.

**Value**

A data frame with one row per feature, with all the relevant statistics of the linear model as columns.

**See Also**

[lm](#)

**Examples**

```

data(toy_notame_set, package = "notame")
# A simple example without QC samples
# Features predicted by Group and Time
lm_results <- perform_lm(notame::drop_qcs(toy_notame_set),
  formula_char = "Feature ~ Group + Time")

```

---

perform\_lm\_anova      *Linear models ANOVA table*

---

**Description**

Fits a linear model separately for each feature and compute an ANOVA table. Returns all relevant statistics.

**Usage**

```

perform_lm_anova(
  object,
  formula_char,
  all_features = FALSE,
  lm_args = NULL,
  anova_args = NULL,
  assay.type = NULL
)

```

## Arguments

object	a <a href="#">SummarizedExperiment</a> object
formula_char	character, the formula to be used in the linear model (see Details)
all_features	should all features be included in FDR correction?
lm_args	list of arguments to lm, list names should be parameter names
anova_args	list of arguments to anova, list names should be parameter names
assay.type	character, assay to be used in case of multiple assays

## Details

The linear model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting, see the example.

## Value

A data frame with one row per feature, with all the relevant statistics of the linear model as columns.

## See Also

[lm](#)

## Examples

```
data(toy_notame_set, package = "notame")
# A simple example without QC samples
# Features predicted by Group and Time
lm_anova_results <- perform_lm_anova(notame::drop_qcs(toy_notame_set),
  formula_char = "Feature ~ Group + Time")
```

---

perform\_lmer

*Linear mixed models*

---

## Description

Fits a linear mixed model separately for each feature. Returns all relevant statistics.

## Usage

```
perform_lmer(  
  object,  
  formula_char,  
  all_features = FALSE,  
  ci_method = c("Wald", "profile", "boot"),  
  test_random = FALSE,  
  assay.type = NULL,  
  ...  
)
```

## Arguments

object	a <a href="#">SummarizedExperiment</a> object
formula_char	character, the formula to be used in the linear model (see <a href="#">Details</a> )
all_features	should all features be included in FDR correction?
ci_method	The method for calculating the confidence intervals as in <a href="#">confint</a>
test_random	logical, whether tests for the significance of the random effects should be performed
assay.type	character, assay to be used in case of multiple assays
...	additional parameters passed to <a href="#">lmer</a>

## Details

The model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting, see the example. With bootstrap ("boot") confidence intervals, the results are reproducible if `RNGseed` is set for the `BiocParallel` backend.

## Value

A data frame with one row per feature, with all the relevant statistics of the linear mixed model as columns.

## See Also

[lmer](#) for model specification

## Examples

```
data(toy_notame_set, package = "notame")  
# A simple example without QC samples  
# Features predicted by Group and Time as fixed effects with Subject ID as a  
# random effect  
lmer_results <- perform_lmer(notame::drop_qcs(toy_notame_set),  
  formula_char = "Feature ~ Group + Time + (1 | Subject_ID)",  
  ci_method = "Wald"  
)
```

---

perform\_logistic      *Logistic regression*

---

## Description

Fits a logistic regression model separately for each feature. Returns all relevant statistics.

## Usage

```
perform_logistic(  
  object,  
  formula_char,  
  all_features = FALSE,  
  assay.type = NULL,  
  ...  
)
```

## Arguments

object	a <a href="#">SummarizedExperiment</a> object
formula_char	character, the formula to be used in the linear model (see Details)
all_features	should all features be included in FDR correction?
assay.type	character, assay to be used in case of multiple assays
...	additional parameters passed to <a href="#">glm</a>

## Details

The logistic regression model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting, see the example.

## Value

A data frame with one row per feature, with all the relevant statistics of the linear model as columns.

## See Also

[glm](#)

## Examples

```
data(toy_notame_set, package = "notame")  
# A simple example without QC samples  
# Time predicted by features  
logistic_results <- perform_logistic(notame::drop_qcs(toy_notame_set),  
  formula_char = "Time ~ Feature + Group")
```

```
)
```

---

```
perform_non_parametric
```

*Pairwise and paired non-parametric tests*

---

## Description

Performs pairwise and paired non-parametric tests. The default is Mann-Whitney U test, use `is_paired` for Wilcoxon signed rank tests.

## Usage

```
perform_non_parametric(
  object,
  formula_char,
  is_paired = FALSE,
  id = NULL,
  all_features = FALSE,
  assay.type = NULL,
  ...
)
```

## Arguments

<code>object</code>	a <a href="#">SummarizedExperiment</a> object
<code>formula_char</code>	character, the formula to be used in the tests
<code>is_paired</code>	logical, use paired test
<code>id</code>	character, name of the subject identification column for paired version
<code>all_features</code>	should all features be included in FDR correction?
<code>assay.type</code>	character, assay to be used in case of multiple assays
<code>...</code>	other parameters passed to test <a href="#">wilcox.test</a>

## Details

P-values of each comparison are corrected separately from each other. The model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual features during model fitting. For example, if testing for equality of means in study groups, use "Feature ~ Group".

## Value

A data frame with the results.

**See Also**[wilcox.test](#)**Examples**

```

data(toy_notame_set, package = "notame")
# Including QCs as a study group for example for pairwise tests
mann_whitney_results <- perform_non_parametric(toy_notame_set,
  formula_char = "Feature ~ Group")
# Using paired mode (pairs with QC are skipped as there are no common IDs in
# 'toy_notame_set')
wilcoxon_signed_results <- perform_non_parametric(toy_notame_set,
  formula_char = "Feature ~ Time",
  is_paired = TRUE,
  id = "Subject_ID")
# Only two groups
mw_results <- perform_non_parametric(notame::drop_qcs(toy_notame_set),
  formula_char = "Feature ~ Group")

```

---

perform\_oneway\_anova    *Welch's ANOVA and classic ANOVA*

---

**Description**

Performs ANOVA with Welch's correction as default, to deal with heterogeneity of variances. Can also perform classic ANOVA with assumption of equal variances. Uses base R function `oneway.test`.

**Usage**

```

perform_oneway_anova(
  object,
  formula_char,
  all_features = FALSE,
  assay.type = NULL,
  ...
)

```

**Arguments**

<code>object</code>	a <a href="#">SummarizedExperiment</a> object
<code>formula_char</code>	character, the formula to be used in the linear model (see <a href="#">Details</a> ).
<code>all_features</code>	should all features be included in FDR correction?
<code>assay.type</code>	character, assay to be used in case of multiple assays
<code>...</code>	other parameters to <a href="#">oneway.test</a>

**Details**

The model is fit on `combined_data(object)`. Thus, column names in pheno data can be specified. To make the formulas flexible, the word "Feature" must be used to signal the role of the features in the formula. "Feature" will be replaced by the actual Feature IDs during model fitting. For example, if testing for equality of means in study groups, use "Feature ~ Group".

**Value**

A data frame with the results.

**See Also**

[oneway.test](#)

**Examples**

```
data(toy_notame_set, package = "notame")
perform_oneway_anova(toy_notame_set, formula_char = "Feature ~ Group")
```

---

perform_permanova	<i>PERMANOVA</i>
-------------------	------------------

---

**Description**

Performs permutational multivariate analysis of variance. Uses package called PERMANOVA.

**Usage**

```
perform_permanova(
  object,
  group,
  all_features = FALSE,
  transform = "Standardize columns",
  coef = "Pythagorean",
  assay.type = NULL,
  ...
)
```

**Arguments**

object	a <a href="#">SummarizedExperiment</a> object
group	character, name of the column to compare
all_features	should all features be included?
transform	Transformation to use in <a href="#">IniTransform</a> . By default uses "Standardize columns".
coef	Coefficient to calculate continuous distances in <a href="#">IniTransform</a> . By default uses Pythagorean distances.

assay.type      character, assay to be used in case of multiple assays  
 ...              other parameters to [PERMANOVA](#)

### Value

A PERMANOVA object.

### Examples

```
data(toy_notame_set, package = "notame")
permanova_res <- perform_permanova(
  notame::drop_qcs(toy_notame_set),
  group = "Group")
```

---

perform\_t\_test      *Pairwise and paired t-tests*

---

### Description

Performs pairwise and paired t-tests. The R default is Welch's t-test (unequal variances), use `var.equal = TRUE` for Student's t-test. Use `is_paired` for paired t-tests.

### Usage

```
perform_t_test(
  object,
  formula_char,
  is_paired = FALSE,
  id = NULL,
  all_features = FALSE,
  assay.type = NULL,
  ...
)
```

### Arguments

object              a [SummarizedExperiment](#) object  
 formula\_char      character, the formula to be used in the linear model (see [Details](#))  
 is\_paired          logical, use paired t-test  
 id                  character, name of the subject identification column for paired version  
 all\_features      should all features be included in FDR correction?  
 assay.type        character, assay to be used in case of multiple assays  
 ...                other parameters passed to [t.test](#)

**Details**

P-values of each comparison are corrected separately from each other.

**Value**

A data frame with the results.

**See Also**

[t.test](#)

**Examples**

```
data(toy_notame_set, package = "notame")
# Including QCs as a study group for example
t_test_results <- perform_t_test(toy_notame_set,
  formula_char = "Feature ~ Group")
# Using paired mode (pairs with QC are skipped as there are no common IDs in
# 'toy_notame_set')
t_test_results <- perform_t_test(toy_notame_set,
  formula_char = "Feature ~ Time", is_paired = TRUE, id = "Subject_ID")
# Only two groups
t_test_results <- perform_t_test(notame::drop_qcs(toy_notame_set),
  formula_char = "Feature ~ Group")
```

---

pls

*PLS*

---

**Description**

Simple wrappers for fitting a PLS model using mixOmics package. The result can then be passed to many of the mixOmics functions for prediction, performance evaluation etc.

**Usage**

```
mixomics_pls(
  object,
  y,
  ncomp,
  all_features = FALSE,
  covariates = NULL,
  assay.type = NULL,
  ...
)

mixomics_pls_optimize(
  object,
```

```

y,
ncomp,
plot_perf = FALSE,
folds = 5,
nrepeat = 50,
all_features = FALSE,
covariates = NULL,
assay.type = NULL,
...
)

mixomics_spls_optimize(
  object,
  y,
  ncomp,
  plot_perf = FALSE,
  n_features = c(seq_len(10), seq(20, 300, 10)),
  folds = 5,
  nrepeat = 50,
  all_features = FALSE,
  covariates = NULL,
  assay.type = NULL,
  ...
)

```

### Arguments

object	a <a href="#">SummarizedExperiment</a> object
y	character vector, column names of the grouping variable to predict
ncomp	number of X components
all_features	logical, should all features be included in the model? if FALSE, flagged features are left out
covariates	character, column names of pheno data to use as covariates in the model, in addition to molecular features
assay.type	character, assay to be used in case of multiple assays
...	any parameters passed to <a href="#">pls</a> or <a href="#">spls</a>
plot_perf	plot performance of models in cross-validation
folds	the number of folds to use in k-fold cross validation
nrepeat	the number of times to repeat the cross validation. Lower this for faster testing.
n_features	the number of features to try for each component

### Details

- `mixomics_pls` A simple PLS model with set number of components and all features
- `mixomics_pls_optimize` Test different numbers of components
- `mixomics_spls_optimize` sPLS model: Test different numbers of components and features

**Value**

An object of class "mixo\_pls" or "mixo\_spls". For the optimized and sparse models, a list with object of class "mixo\_plsda" and a performance plot.

**See Also**

[pls](#), [perf](#), [spls](#), [tune.spls](#)

**Examples**

```
data(toy_notame_set, package = "notame")
pls_res <- mixomics_pls(toy_notame_set, y = "Injection_order", ncomp = 3)
# Cross-validation repeated only 5 times for quick run time
pls_opt <- mixomics_pls_optimize(toy_notame_set,
  y = "Injection_order", ncomp = 3, nrepeat = 5)
spls_opt <- mixomics_spls_optimize(toy_notame_set,
  y = "Injection_order", ncomp = 3,
  n_features = c(1:10, 12, 15, 20), nrepeat = 5
)
# Plot score plot of any final model
mixOmics::plotIndiv(pls_res,
  comp = seq_len(2), group = toy_notame_set$Group,
  ind.names = FALSE, title = "PLS scores plot", legend = TRUE)

# Proportion of variance explained
pls_res$prop_expl_var$X[seq_len(2)] |> round(digits = 3) * 100
```

---

pls\_da

*PLS-DA*

---

**Description**

A simple wrapper for fitting a PLS-DA model using mixOmics package. The object can then be passed to many of the mixOmics functions for prediction, performance evaluation etc.

- `mixomics_plsda` A simple PLS-DA model with set number of components and all features
- `mixomics_plsda_optimize` Test different numbers of components, choose the one with minimal balanced error rate
- `mixomics_splsda_optimize` Test different numbers of components and features, choose the one with minimal balanced error rate

**Usage**

```
mixomics_plsda(
  object,
  y,
  ncomp,
```

```

    all_features = FALSE,
    covariates = NULL,
    assay.type = NULL,
    ...
)

mixomics_plsda_optimize(
  object,
  y,
  ncomp,
  plot_perf = FALSE,
  folds = 5,
  nrepeat = 50,
  all_features = FALSE,
  covariates = NULL,
  assay.type = NULL,
  ...
)

mixomics_splsda_optimize(
  object,
  y,
  ncomp,
  dist,
  plot_perf = FALSE,
  n_features = c(seq_len(10), seq(20, 300, 10)),
  folds = 5,
  nrepeat = 50,
  all_features = FALSE,
  covariates = NULL,
  assay.type = NULL,
  ...
)

```

### Arguments

object	a SummarizedExperiment object
y	character, column name of the grouping variable to predict
ncomp	the number of X components
all_features	logical, should all features be included in the model? if FALSE, flagged features are left out
covariates	character, column names of pheno data to use as covariates in the model, in addition to molecular features
assay.type	character, assay to be used in case of multiple assays
...	any parameters passed to <a href="#">plsda</a>
plot_perf	plot performance of models in cross-validation

fold	the number of folds to use in k-fold cross validation
nrepeat	the number of times to repeat the cross validation. Lower this for faster testing.
dist	the distance metric to use, one of "max.dist", "mahalanobis.dist", "centroids.dist". use <a href="#">mixomics_plsda_optimize</a> to find the best distance metric
n_features	the number of features to try for each component

### Value

An object of class "mixo\_plsda" or for the optimized and sparse models, a list with object of class "mixo\_plsda" and a performance plot.

### See Also

[plsda](#), [perf](#), [splsda](#), [tune.splsda](#)

### Examples

```
data(toy_notame_set, package = "notame")
noqc <- notame::drop_qcs(toy_notame_set)
plsda_res <- mixomics_plsda(noqc, y = "Group", ncomp = 2)
# Cross-validation repeated only 5 times for quick run time
set.seed(38)
plsda_opt <- mixomics_plsda_optimize(noqc,
  y = "Group", ncomp = 3, nrepeat = 5
)
set.seed(38)
splsda_opt <- mixomics_splsda_optimize(noqc,
  y = "Group", dist = "max.dist", ncomp = 2,
  n_features = c(1:10, 12, 15, 20), nrepeat = 5
)
# Plot PLS-DA scores
mixOmics::plotIndiv(plsda_res,
  comp = seq_len(2), group = notame::drop_qcs(toy_notame_set)$Group,
  ind.names = FALSE, title = "PLS-DA scores plot", legend = TRUE,
  ellipse = TRUE)

# Plot prediction areas
background <- mixOmics::background.predict(plsda_res,
  comp.predicted = 2, dist = "max.dist")
mixOmics::plotIndiv(plsda_res,
  comp = seq_len(2), group = notame::drop_qcs(toy_notame_set)$Group,
  ind.names = FALSE,
  title = "prediction areas", legend = TRUE, background = background)
```

---

summarize\_results      *Statistics cleaning*

---

### Description

Uses `regex` to remove unnecessary columns from statistics results data frame. Can also rename columns effectively.

### Usage

```
summarize_results(  
  df,  
  remove = c("Intercept", "CI95", "Std_error", "t_value", "z_value", "R2"),  
  rename = NULL,  
  summary = TRUE,  
  p_limit = 0.05,  
  fdr = TRUE  
)
```

### Arguments

<code>df</code>	data frame, statistics results
<code>remove</code>	list, should contain strings that are matching to unwanted columns
<code>rename</code>	named list, names should contain matches that are replaced with values
<code>summary</code>	logical, should summary columns be added
<code>p_limit</code>	numeric, limit for p-values to be counted
<code>fdr</code>	logical, should summary be done with fdr-fixed values

### Value

A data frame with removed and/or renamed columns.

### Examples

```
data(toy_notame_set, package = "notame")  
# Simple manipulation to linear model results  
lm_results <- perform_lm(notame::drop_qcs(toy_notame_set),  
  formula_char = "Feature ~ Group + Time")  
lm_results <- summarize_results(lm_results,  
  rename = c("GroupB" = "GroupB_vs_A", "Time2" = "Time2_vs_1")  
)
```

---

summary_statistics	<i>Summary statistics</i>
--------------------	---------------------------

---

### Description

Computes summary statistics for each feature, possibly grouped by a factor. The statistics include mean, standard deviation (sd), median, median absolute deviation (mad), minimum (min), maximum (max) as well as 25

### Usage

```
summary_statistics(object, grouping_cols = NULL, assay.type = NULL)
```

### Arguments

object	a <a href="#">SummarizedExperiment</a> object
grouping_cols	character vector, the columns by which grouping should be done. Use NA to compute statistics without grouping.
assay.type	character, assay to be used in case of multiple assays

### Value

A data frame with the summary statistics.

### Examples

```
data(toy_notame_set, package = "notame")
# Group by "Group"
sum_stats <- summary_statistics(toy_notame_set, grouping_cols = "Group")
# Group by Group and Time
sum_stats <- summary_statistics(toy_notame_set,
  grouping_cols = c("Group", "Time"))
# No Grouping
sum_stats <- summary_statistics(toy_notame_set)
```

# Index

auc, 8

bartlett.test, 11

cohens\_d, 2  
confint, 15  
cor.test, 10

fit\_rf, 3, 5  
fligner.test, 11  
fold\_change, 4

getVar, 6, 7  
glm, 16

importance\_rf, 3, 4, 5  
IniTransform, 19

kruskal.test, 12

leveneTest, 11  
lm, 13, 14  
lmer, 15

mixomics\_pls(pls), 21  
mixomics\_pls\_optimize(pls), 21  
mixomics\_plsda(pls\_da), 23  
mixomics\_plsda\_optimize, 25  
mixomics\_plsda\_optimize(pls\_da), 23  
mixomics\_spls\_optimize(pls), 21  
mixomics\_splsda\_optimize(pls\_da), 23  
MUVR2, 5–7  
MUVR2\_EN, 5–7  
muvr\_analysis, 5

oneway.test, 18, 19

perf, 23, 25  
perform\_auc, 8  
perform\_correlation\_tests, 9  
perform\_homoscedasticity\_tests, 10  
perform\_kruskal\_wallis, 11  
perform\_lm, 12  
perform\_lm\_anova, 13  
perform\_lmer, 14  
perform\_logistic, 16  
perform\_non\_parametric, 17  
perform\_oneway\_anova, 18  
perform\_permanova, 19  
perform\_t\_test, 20  
PERMANOVA, 20  
plot\_effect\_heatmap, 10  
plotMV, 7  
plotStability, 7  
plotVAL, 7  
plotVIRank, 7  
pls, 21, 22, 23  
pls\_da, 23  
plsda, 24, 25

randomForest, 3–5  
rmcorr, 10

spls, 22, 23  
splsda, 25  
summarize\_results, 26  
SummarizedExperiment, 2–4, 6, 8, 9, 11–20, 22, 27  
summary\_statistics, 27

t.test, 20, 21  
tune.spls, 23  
tune.splsda, 25

wilcox.test, 17, 18