Package ‘MutSeqR’

June 4, 2025

Title Analysis of error-corrected Next-Generation Sequencing Data for Mutation Detection

Version 0.0.0.9000

Description Standard methods for analysis of mutation data following ECS
for the purpose of mutagencity assessment. Functions include importing
the mutation lists provided by a variant caller, and a set of analytical
tools for statistical testing and visualization of mutation data;
comparison to COSMIC and/or germline signatures; etc.

License MIT + file LICENSE

Depends R (>= 3.5.0)

Imports BiocGenerics,
BiocManager,
Biostrings,
dplyr,
GenomicRanges,
ggplot2,
here,
IRanges,
magrittr,
plyranges,
rlang,
S4Vectors,
stringr,
SummarizedExperiment,
tidyr,
VariantAnnotation

Suggests binom,
BSgenome,
BSgenome.Hsapiens.UCSC.hg38,
BSgenome.Mmusculus.UCSC.mm10,
car,
colorspace,
doBy,
fmsb,
fs,

GenomeInfoDb,
GenVisR,
ggh4x,
ggrepel,
gtools,
httr,
knitr,
lme4,
openxlsx,
packcircles,
patchwork,
RColorBrewer,
reticulate,
rmarkdown,
scales,
shiny,
SigProfilerMatrixGeneratorR,
testthat (>= 3.0.0),
ToxicR,
trackViewer,
yaml,
xml2

Config/reticulate list(packages = list(list(package = ``SigProfilerAssignment"), list(package = ``SigProfilerExtractor"), list(package = ``SigProfilerMatrixGenerator")))

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Contents

`annotate_CpG_sites` *Annotate CpG sites*

Description

A simple method to test whether your trinucleotide context contains a CpG site. Vectorized version of Biostrings::vcountPattern is used.

Usage

```
annotate_CpG_sites(mut_data, motif = "CG", column_query = "context", ...)
```

Arguments

`mut_data`	A dataframe or GRanges object containing the genomic regions of interest in which to look for CpG sites.
`motif`	Default "CG", which returns CpG sites. You could in theory use an arbitrary string to look at different motifs. Use with caution. In this case the pattern being searched must be a column in the mutation data.
`column_query`	Default "context" but can be any column in the mutation data that you wish to look for a motif in.
`...`	Additional arguments to vcountPattern()

Value

A data frame with the same number of rows as there were ranges in the input, but with an additional metadata column indicating CpG sites in the target sequence of the mutation.

`bmd_proast` *BMD modeling using PROAST*

Description

Calculate the benchmark dose (BMD) of continuous, individual-level data with optional model averaging. This function is intended to model the dose-response of mutation frequency. This function is an extension of the PROAST software (copyright RIVM National Institute for Public Health and the Environment).

Usage

```
bmd_proast(
  mf_data,
  dose_col = "dose",
  response_col = "mf_min",
  covariate_col = NULL,
  bmr = 0.5,
  adjust_bmr_to_group_sd = FALSE,
  model_averaging = TRUE,
  num_bootstraps = 200,
  summary = TRUE,
  plot_results = FALSE,
  output_path = NULL
)
```

Arguments

- `mf_data` — A data frame containing the data to be analyzed. Data should be individual for each sample. Required columns are the column containing the dose `dose_col` the column(s) containing the mutation frequency `response_col`, and the column containing the covariate `covariate_col`, if applicable.
- `dose_col` — The column in `mf_data` containing the dose data. Values must be numeric. Default is "dose".
- `response_col` — The column(s) in `mf_data` containing the mutation frequency. Multiple `response_cols` can be provided. Default is "mf_min".
- `covariate_col` — The column in `mf_data` containing the covariate. If no covariate is present, set to `NULL` (default).
- `bmr` — The Benchmark Response value. The BMR is defined as a bmr-percent change in mean response relative to the controls. Default is 0.5 (50% change).
- `adjust_bmr_to_group_sd` — A logical value indicating whether the group standard deviation should be used as the BMR. If TRUE, the BMR will be bet set to one standard deviation above the control group mean. Default is FALSE.
- `model_averaging` — A logical value indicating whether confidence intervals should be calculated using model averaging. Default is TRUE (recommended).
- `num_bootstraps` — The number of bootstrap resamples to be used in the model averaging. Default is 200 (recommended).
- `summary` — A logical value indicating whether a summary of the results should be returned. If FALSE, raw results from the PROAST analysis are returned.
- `plot_results` — A logical value indicating whether to plot the BMD models and/or the Cleveland plots. Default is FALSE. If TRUE, the function will save plots to the `output_path`.
- `output_path` — The file path indicating where to save the plots. If NULL, the plots will be saved to the working directory. Default is NULL.

Details

This function is a modified version of the original interactive PROAST software (https://www.rivm.nl/en/proast that allows for batch processing of data. The function is designed to be used with the output of `calculate_mf` for the purpose of calculating the Benchmark Dose of mutation frequency data. As such, some functionality of the original PROAST software has been removed.

This function will accept continuous data, with an observation for each individual subject. It is assumed that data are lognormally distributed. The response data is log-transformed, then back-transformed after the statistical analysis. The function will fit model 3 or 5 from various families of models (Exponential, Hill, Inverse Exponential, LogNormal). It will then compare the fits of models 3 and 5 for each model family and select the model with the lowest AIC. The BMD 90% confidence intervals will be calculated based on the selected model (3 or 5) for each model family using the profile likelihood method. The BMD 90% confidence interval may also be calculated using the bootstrap method if `model_averaging = TRUE`. It is recommended to use 200 bootstraps for model averaging.

To replicate these results in the PROAST interactive software, select the following menu options:

1. f.proast(mf_data)
2. What type of response data do you want to consider? *1: continuous, individual data*
3. Do you want to fit a single model or fit various nested families of models? *3: select model 3 or 5 from various families of models*
4. Q1: Which variable do you want to consider as the independent variable? *# : dose_col*
5. Give number(s) of the response(s) you want to analyse. *# : response_col*
6. Give number of factor serving as potential covariate (e.g.sex) type 0 if none. *# : covariate_col*
7. Do you want to adjust CES to within group SD? *1: no, 2: yes | adjust_bmr_to_group_sd: FALSE/TRUE*
8. Give value for CES (always positive) type 0 to avoid calculation of CIs. *bmr*
9. Do you want to calculate the BMD confidence interval by model averaging? *1: no 2: yes | model_averaging: FALSE/TRUE*
10. give number of bootstrap runs for calculating BMD confidence interval based on MA (e.g. 200) *num_bootstraps*
11. Which models do you want to be fitted? *4 : previous option with lognormal DR model added*

Value

If summary is TRUE, a data frame of final results. If summary is FALSE, a list of the raw results from the PROAST analysis.

The summary will include the following for each response variable and covariate subgroup (if applicable):

- `Model`: The m3 or m5 model selected for each model family (Exponential, Hill, Inverse Exponential, LogNormal).
- `Response`: The response variable.
- `Covariate`: The covariate subgroup, if applicable.
- `bmr`: The specified Benchmark Response.
- `BMD`: The Benchmark Dose, in original dose units, estimated for the given model.
- `BMDL`: The lower bound of the 90% confidence interval for the BMD, calculated by the profile likelihood method.
- `BMDU`: The upper bound of the 90% confidence interval for the BMD, calculated by the profile likelihood method.
- `AIC`: The Akaike Information Criterion for the selected model. Lower values indicate a better fit. It is advised to choose the BMD value from the model with the lowest AIC.
- `weights`: The weight of the model in the model averaging process, if applicable.
- `Model averaging`: The BMDL and BMDU calculated by the bootstrap method if `model_averaging = TRUE`.

If there is no significant response in the data, the function will return an empty data frame.

If `plot_results = TRUE` the function will create the following plots for each response variable:

- Model Plots. The following plot will be created for each model family (Exponential, Hill, Inverse Exponential, LogNormal): The fitted curve of the selected (3 or 5) model. Data is log-transformed. Individual data points are plotted using small triangles. The geometric mean (median) at each dose is plotted as a large triangle. The BMD is indicated by the dotted line. If applicable, the covariate subgroup is indicated by color.
- ma.plot If `model_averaging = TRUE`, the bootstrap curves based on model averaging. The geometric mean (median) at each dose is plotted as a large triangle. Data is log-transformed.
- cleveland plot if `model_averaging = TRUE` The BMD estimate for each model is plotted with error bars representing the 90% confidence interval. The size of the point represents the model weight assigned during model averaging, based on the AIC.

Examples

```
# Calculate the BMD for a 50% increase in mutation frequency from control
# With Model averaging.
# For the purpose of this example, num_bootstraps is set to 5 to reduce
# run time. 200 bootstraps is recommended.
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
mf <- calculate_mf(example_data, retain_metadata_cols = "dose")
bmd <- bmd_proast(mf_data = mf,
                  dose_col = "dose",
                  response_col = c("mf_min", "mf_max"),
                  bmr = 0.5,
                  model_averaging = TRUE,
                  num_bootstraps = 5)
# Plot the Model Averaging 90% CI using plot_ci()
plot_df <- bmd %>%
 dplyr::filter(Model == "Model averaging") %>%
 dplyr::select(Response, BMD, BMDL, BMDU)
plot <- plot_ci(plot_df, order = "asc", log_scale = FALSE)
```

`bmd_toxicr` *BMD modeling using ToxicR*

Description

Calculate the benchmark dose (BMD) for continuous dose-response data with optional model averaging. This function is intended to model the dose-response of mutation frequency using the ToxicR software.

Usage

```
bmd_toxicr(
  mf_data,
  data_type = "individual",
  dose_col = "dose",
  response_col = c("mf_min", "mf_max"),
```

```
  sd_col = NULL,
  n_col = NULL,
  bmr_type = "rel",
  bmr = 0.5,
  model = "exp-aerts",
  alpha = 0.05,
  model_averaging = TRUE,
  plot_results = FALSE,
  ma_summary = FALSE,
  output_path = NULL,
  ...
)
```

Arguments

mf_data	A data frame containing the dose-response data. Data may be individual for each sample or averaged over dose groups. Required columns for individual data are the column containing the dose `dose_col` and the column(s) containing the mutation frequency data `response_col`(s). Summary data must include the `dose_col`, the `response_col`(s) containing the mean response for each dose group, the `sd_col` containing the standard deviation of the response data, and the `n_col` containing the sample size for each dose group.
data_type	A string specifying the type of response data. Data may be response per individual or summarised across dose groups. Options are ("individual", "summary"). Default is "individual".
dose_col	The column in `mf_data` containing the dose data. Values must be numeric. Default is "dose".
response_col	The column(s) in mf_data containing the mutation frequency data. For summarised data types, this should be the mean response for each dose group. Multiple `response_cols` can be provided.
sd_col	The column in mf_data containing the standard deviation of the summarised response data. This is only required for `data_type = "summary"`. If multiple response columns are provided, multiple `sd_cols` should be provided in the same order. Default is NULL.
n_col	The column in mf_data containing the sample size of each dose group. This is only required for `data_type = "summary"`. If multiple response columns are provided, multiple `n_cols` should be provided in the same order. Default is NULL.
bmr_type	The type of benchmark response. Options are: "rel", "sd", "hybrid", "abs". Default is "rel". See details for more information.
bmr	A numeric value specifying the benchmark response. The bmr is defined in relation to the calculation requested in bmr_type. Default is 0.5.
model	The model type to use. Options are "all" or a vector of model types. Default is "exp-aerts", the Exponential model. See details for available models. Note that model averaging will use a pre-defined model set. See details for more information.

`alpha` The specified nominal coverage rate for computation of the lower and upper confidence intervals for the benchmark dose (BMDL, BMDU). The confidence level is calculated as $100 \times (1 - 2\alpha)\%$. The default is 0.05 (90% CI).

`model_averaging` A logical value indicating whether to use model averaging. Default is TRUE (recommended).

`plot_results` A logical value indicating whether to plot the BMD models and/or the Cleveland plots. Default is FALSE. If TRUE, the function will save plots to the `output_path`.

`ma_summary` A logical value indicating whether to return the summary of the model averaging results. Default is FALSE.

`output_path` The file path indicating where to save the plots. If NULL, the plots will be saved to the working directory. Default is NULL.

`...` Additional arguments to be passed to the model fitting function. For more information, see single_continuous_fit or ma_continuous_fit if model averaging.

Details

Available model types for single model fitting are:

- `"exp-aerts"`: $f(x) = a(1 + (c - 1)(1 - \exp(-bx^d)))$
- `"invexp-aerts"`: $f(x) = a(1 + (c - 1)(\exp(-bx^{-d})))$
- `"gamma-aerts"`: $f(x) = a(1 + (c - 1)(Gamma(bx^d; xi)))$
- `"invgamma-aerts"`: $f(x) = a(1 + (c - 1)(1 - Gamma(bx^{-d}; xi)))$
- `"hill-aerts"`: $f(x) = a(1 + (c - 1)(1 - \frac{b^d}{b^d + x^d}))$
- `"lomax-aerts"`: $f(x) = a\left\{1 + (c - 1)(1 - \left(\frac{b}{b + x^d}\right)^{\xi})\right\}$
- `"invlomax-aerts"`: $f(x) = a\left\{1 + (c - 1)(\left(\frac{b}{b + x^{-d}}\right))^{\xi}\right\}$
- `"lognormal-aerts"`: $f(x) = a\left\{1 + (c - 1)\left(\Phi(\ln(b) + d \times \ln(x))\right)\right\}$
- `"logskew-aerts"`: $f(x) = a\left\{1 + (c - 1)\left(\Phi_{SN}(\ln(b) + d \times \ln(x); \xi)\right)\right\}$
- `"invlogskew-aerts"`: $f(x) = a\left\{1 + (c - 1)\left(1 - \Phi_{SN}(\ln(b) - d \times \ln(x); \xi)\right)\right\}$
- `"logistic-aerts"`: $f(x) = \frac{c}{1 + \exp(-a - b \times x^d)}$
- `"probit-aerts"`: $f(x) = c\left(\Phi(a + b \times x^d)\right)$
- `"LMS"`: $f(x) = a(1 + (c - 1)(1 - \exp(-bx - dx^2)))$
- `"gamma-efsa"`: $f(x) = a(1 + (c - 1)(Gamma(bx; d)))$

Here: $\Phi(\cdot)$ is the standard normal distribution and $\Phi_{SN}(\cdot;\cdot)$ is the skew-normal distribution. See single_continuous_fit for more details.

Model averaging is done over the the model set described in The European Food Safety Authority's (2022) Guidance on the use of the benchmark dose approach in risk assessment. These models are (normal then lognormal for each model): exp-aerts, invexp-aerts, hill-aerts, lognormal-aerts, gamma-efsa, LMS, probit-aerts, and logistic-aerts. See ma_continuous_fit for more details.

BMR types for continuous models:

- Relative deviation (default; `bmr_type = "rel"`). This defines the BMD as the dose that changes the control mean/median a certain percentage from the background dose. It is the dose that solves $\mid f(dose) - f(0) \mid = (1 \pm BMR)f(0)$
- Standard deviation (`bmr_type = "sd"`). This defines the BMD as the dose associated with the mean/median changing a specified number of standard deviations from the mean at the control dose. It is the dose that solves $\mid f(dose) - f(0) \mid = BMR \times \sigma$
- Absolute deviation (`bmr_type="abs"`). This defines the BMD as an absolute change from the control dose of zero by a specified amount. That is the BMD is the dose that solves the equation $\mid f(dose) - f(0) \mid = BMR$.
- Hybrid deviation (`bmr_type = "hybrid"`). This defines the BMD that changes the probability of an adverse event by a stated amount relative to no exposure (i.e 0). That is, it is the dose that solves $\frac{Pr(X>x|dose)-Pr(X>x|0)}{Pr(X<x|0)} = BMR$. For this definition, $Pr(X < x|0) = 1 - Pr(X > X|0) = \pi_0$, where $0 \leq \pi_0 < 1$ is defined by the user as "point_p," and it defaults to 0.01. Note: this discussion assumed increasing data. The fitter determines the direction of the data and inverts the probability statements for decreasing data.

Value

If `model_averaging = FALSE`, the function returns a data frame with the BMD values and the $100 \times (1 - 2\alpha)\%$ confidence intervals (BMDL, BMDU)for each response column and each model listed. The AIC value is calculated for each model to compare fits. The AIC is calculated as maximum likelihood + 2 * degrees of freedom. If `plot_results = TRUE`, the function will plot all fitted models to the data and save them to the `output_path`.

If `model_averaging = TRUE`, the function returns a data frame with the BMD values and the $100 \times (1-2\alpha)\%$ confidence intervals (BMDL, BMDU) for each response column calculated using model averaging. If `ma_summary = TRUE`, the function will return the posterior probabilities used in the model averaging. If `plot_results = TRUE`, the function will plot the model averaged model to the data and save it to the `output_path`. The function will also make a Cleveland plot, saved to the `output_path`. Here, the BMDs are plotted for each model in the set alongside the model averaged BMD. The BMD is represented by a red dot. The size of the dot is scaled on the model probability with the Model Average having a value of 100%. The BMDL and BMDU are expressed as interval bars.

Examples

```
# Calculate the BMD for a 50% increase in mutation frequency from control
# Individual data with Model averaging.
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)
mf <- calculate_mf(example_data, retain_metadata_cols = "dose")
bmd <- bmd_toxicr(mf_data = mf,
                  dose_col = "dose",
                  response_col = c("mf_min", "mf_max"))
# Plot the results using plot_ci()
plot <- plot_ci(bmd, order = "asc", log_scale = FALSE)

# Summary data with Model averaging.
```

```
mf_sum <- mf %>%
 dplyr::group_by(dose) %>%
 dplyr::summarise(mean_mf_min = mean(mf_min),
                  sd_min = sd(mf_min),
                  n_min = dplyr::n(),
                  mean_mf_max = mean(mf_max),
                  sd_max = sd(mf_max),
                  n_max = dplyr::n())
bmd <- bmd_toxicr(mf_data = mf_sum,
                  dose_col = "dose",
                  response_col = c("mean_mf_min", "mean_mf_max"),
                  sd_col = c("sd_min", "sd_max"),
                  n_col = c("n_min", "n_max"))
```

`calculate_mf` *Calculate mutation frequency*

Description

Calculates the mutation frequency for arbitrary groupings and creates a new dataframe with the results. Mutation frequency is # mutations / total bases, but this can be subset in different ways: e.g., by mutation context. In this case, it is necessary to change the denominator of total bases to reflect the sequencing depth at the proper reference bases under consideration. Additionally, by default, the operation is run by default using both the minimum and maximum independent methods for counting mutations.

Usage

```
calculate_mf(
  mutation_data,
  cols_to_group = "sample",
  subtype_resolution = "none",
 variant_types = c("snv", "deletion", "insertion", "complex", "mnv", "sv", "ambiguous",
    "uncategorized"),
  calculate_depth = TRUE,
  precalc_depth_data = NULL,
  d_sep = "\t",
  summary = TRUE,
  retain_metadata_cols = NULL
)
```

Arguments

`mutation_data` The data frame to be processed containing mutation data. Required columns are listed below. Synonymous names for these columns are accepted.

- `contig`: The reference sequence name.
- `start`: 1-based start position of the feature.
- `sample`: The sample name.

- `alt_depth`: The read depth supporting the alternate allele.
- `variation_type`: The category to which this variant is assigned.
- subtype_col: The column containing the mutation subtype. This column depends on the `subtype_resolution` parameter.
- reference context: The column containing the referene base(s) for the mutation. This column depends on the `subtype_resolution` parameter.
- cols to group: all columns across which you want to calculate the mutation frequency. Ex. `c("tissue", "dose")`. These columns should be listed in cols_to_group.

It is also required to include the total_depth column if you are calculating depth from the mutation data. If you are using precalculated depth data, the total_depth column is not required.

`cols_to_group` A vector of grouping variables: this should be the groups of interest that you want to calculate a frequency for. For instance, getting the frequency by `sample`. Other options might include `dose`, `locus`, or, `c("sample","locus")`. All listed variables must be a column in the mutation_data.

`subtype_resolution`

The resolution of the mutation subtypes. Options are

- "none" calculates mutation frequencies across all selected grouping columns.
- "type" calculates mutation frequencies across all selected grouping columns for each `variation_type` seperately; snv, mnv, deletion, insertion, complex, sv, ambiguous, uncategorized.
- "base_6" calculates mutation frequencies across all selected grouping columns for each variation_type with snv mutations separated by `normalized_subtype`; C>A, C>G, C>T, T>A, T>C, T>G. The reference context is `normalized_ref`.
- "base_12" calculates mutation frequencies across all selected grouping columns for each variation_type with snv mutations separated by `subtype`; A>C, A>G, A>T, C>A, C>G, C>T, G>A, G>C, G>T, T>A, T>C, T>G. The reference context is `short_ref`.
- "base_96" calculates mutation frequencies across all selected grouping columns for each variation_type with snv mutations separated by `normalized_context_with_mutation`, i.e. the 96-base trinucleotide context. Ex. A\[C>T\]A. The reference context is `normalized_context`.
- "base_192" calculates mutation frequencies across all selected grouping columns for each variation_type with snv mutations separated by `context_with_mutation`, i.e. the 192-base trinucleotide context. Ex A\[G>A\]A. The reference context is `context`.

`variant_types` Use this parameter to choose which variation types to include in the mutation counts. Provide a character vector of the variation types that you want to include. Alternatively, provide a character vector of the variation types that you want to exclude preceded by "-". Options are: "snv", "complex", "deletion", "insertion", "mnv", "sv", "ambiguous", "uncategorized". Ex. inclusion: "snv", exclusion: "-snv". Default includes all variants. For `calculate_depth` = `TRUE`: Regardless of whether or not a variant is included in the mutation counts, the total_depth for that position will be counted.

calculate_depth
: A logical variable, whether to calculate the per-group total_depth from the mutation data. If set to TRUE, the mutation data must contain a total_depth value for every sequenced base (including variants AND no-variant calls). If set to FALSE, pre-calculated per-group total_depth values may be supplied at the desired subtype_resolution using the precalc_depth_data parameter. Alternatively, if no per-group total_depth is available, per-group mutation counts will be calculated, but mutation frequency will not. In such cases, mutation subtype proportions will not be normalized to the total_depth.

precalc_depth_data
: A data frame or a file path to a text file containing pre-calculated per-group total_depth values. This data frame should contain the columns for the desired grouping variable(s) and the reference context at the desired subtype resolution (if applicable). The precalculated total_depth column(s) should be called one or both of `group_depth` and `subtype_depth`. `group_depth` is used for subtype resolutions of "none", "type", and all non-snv mutations in "base_6", "base_12", "base_96", and "base_192". `subtype_depth` is used for snv mutations in "base_6", "base_12", "base_96", and "base_192". You can access a list of context values for each subtype resolution using `MutSeqR::context_list$your_subtype_resolutio`

d_sep
: The delimiter used in the precalc_depth_data, if applicable. Default is tab-delimited.

summary
: A logical variable, whether to return a summary table (i.e., where only relevant columns for frequencies and groupings are returned). Setting this to false returns all columns in the original mutation_data, which might make plotting more difficult, but may provide additional flexibility to power users.

retain_metadata_cols
: a character vector that contains the names of the metadata columns that you would like to retain in the summary table. This may be useful for plotting your summary data. Ex. retain the "dose" column when summarising by "sample".

Value

A data frame with the mutation frequency calculated. If summary is set to TRUE, the data frame will be a summary table with the mutation frequency calculated for each group. If summary is set to FALSE, the mutation frequency will be appended to each row of the original mutation_data.

- `sum_min`: The sum of all mutations within the group, calculated using the "min" method for mutation counting. All identical mutations within a samples are assumed to be the result of clonal expansion and are thus only counted once.
- `sum_max`: The sum of all mutations within the group, calculated using the "max" method for mutaiton counting. All identical mutations within a sample are assumed to be idenpendant mutational evens and are included in the mutation frequency calculation.
- `mf_min`: The mutation frequency calculated using the "min" method for mutation counting. mf_min = sum_min / depth.
- `mf_max`: The mutation frequency calculated using the "max" method for mutation counting. mf_max = sum_max / depth.

- `proportion_min`: The proportion of each mutation subtype within the group, normalized to the depth. Calculated using the "min" method. This is only calculated if `subtype_resolution` is not "none". If no depth is calculated or provided, proportion is calculated without normalization to the depth.
- `proportion_max`: The proportion of each mutation subtype within the group, normalized to its read depth. Calculated using the "max" method. This is only calculated if `subtype_resolution` is not "none". If no depth is calculated or provided, proportion is calculated without normalization to the depth.

Examples

```
# Load example data
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)

# Example 1 Calculate mutation frequency by sample.
# Calculate depth from the mutation data
mf_example <- calculate_mf(mutation_data = example_data,
                           cols_to_group = "sample")
# Example 2: Calculate the trinucleotide mutation proportions for each dose
mf_96_example <- calculate_mf(mutation_data = example_data,
                              cols_to_group = "dose",
                              subtype_resolution = "base_96",
                              variant_types  = "snv")
# Example 3: Calculate the mean mutation frequency for each 6 base subtype
# per dose
# calculate_mf does not calculate mean mutation frequency for
# groups; this function only sums mutations across groups. Thus, if you are
# interested in calculating the mean of a group, this must be done
# separately.
# First, calculate 6 base MF per sample. Retain the dose column.
mf_6_example <- calculate_mf(mutation_data = example_data,
                             cols_to_group = "sample",
                             subtype_resolution = "base_6",
                             retain_metadata_cols = "dose")
# Note: NA values in retain_metadata_cols. When we create a summary table
# that includes mutation subtypes, there may occasionally be NA values in
# the metadata columns. This is because the mutation data does not contain
# any mutations (filtered or not) for that particular subtype within the
# given group. For example, our example_data does not contain any ambiguous
# or uncategorized mutations, so the dose column is NA for all those
# mutations in the summary table. This will not affect downstream analyses.

# Calculate the mean MF for each 6 base subtype per dose
mf_6_mean_example <- mf_6_example %>%
 dplyr::group_by(dose, normalized_subtype) %>%
 dplyr::summarise(mean_mf_min = mean(mf_min),
                  se_mf_min = sd(mf_min) / sqrt(dplyr::n()),
                  mean_mf_max = mean(mf_max),
                  se_mf_max = sd(mf_max) / sqrt(dplyr::n()))

# Example 4: Calculate MF using precalculated depth data
```

```
sample_depth_example <- data.frame(sample = c("dna00973.1", "dna00974.1",
                                              "dna00975.1", "dna00976.1",
                                              "dna00977.1", "dna00978.1",
                                              "dna00979.1", "dna00980.1",
                                              "dna00981.1", "dna00982.1",
                                              "dna00983.1", "dna00984.1",
                                              "dna00985.1", "dna00986.1",
                                              "dna00987.1", "dna00988.1",
                                              "dna00989.1", "dna00990.1",
                                              "dna00991.1", "dna00992.1",
                                              "dna00993.1", "dna00994.1",
                                              "dna00995.1", "dna00996.1"),
                                   group_depth = c(565395266, 755574283,
                                                   639909215, 675090988,
                                                   598104021, 611295330,
                                                   648531765, 713240735,
                                                   669734626, 684951248,
                                                   716913381, 692323218,
                                                   297661400, 172863681,
                                                   672259724, 740901132,
                                                   558051386, 733727643,
                                                   703349287, 884821671,
                                                   743311822, 799605045,
                                                   677693752, 701163532))
mf_example_precalc <- calculate_mf(mutation_data = example_data,
                                   cols_to_group = "sample",
                                   calculate_depth = FALSE,
                                   precalc_depth_data = sample_depth_example)

# Example 5: Calculate MF using precalculated depth data for 6 base
# mutation subtypes per sample.
# The base_6 resolution uses reference context 'normalized_ref'; C or T.
# Our precalc_depth_data needs group_depth (depth per sample) and the
# subtype_depth (depth per sample AND per normalized_ref)
# We will create the example precalc_depth data for the base_6 resolution
# from Example 3 results for simplicity.
sample_subtype_depth_example <- mf_6_example %>%
 dplyr::select(sample, normalized_ref, group_depth, subtype_depth) %>%
 unique() %>%
dplyr::filter(normalized_ref != "N")
mf_6_example_precalc <- calculate_mf(mutation_data = example_data,
                                     cols_to_group = "sample",
                                     subtype_resolution = "base_6",
                                     calculate_depth = FALSE,
                                     precalc_depth_data = sample_subtype_depth_example)
```

`check_required_columns` *Check that all required columns are present before proceeding with the function*

Description

A utility function that will check that all required columns are present.

Usage

```
check_required_columns(data, required_columns)
```

Arguments

`data`	mutation data
`required_columns`	a list of required column names.

Value

an error

`classify_variation` *classify_variation*

Description

Classify the variation type of a mutation based on its ref and alt values.

Usage

```
classify_variation(ref, alt)
```

Arguments

`ref`	The reference allele.
`alt`	The alternate allele.

Value

A character indicating the type of variation.

`cleveland_plot` *Cleveland Plot*

Description

Make a Cleveland plot for the PROAST results. Matches ToxicR.

Usage

```
cleveland_plot(results, covariate_col = NULL, output_path = NULL)
```

Arguments

`results`	PROAST results object.
`covariate_col`	Covariate column name.
`output_path`	Output path for the plot.

Value

ggplot object.

`cluster_spectra` *Hierarchical Clustering*

Description

perform hierarchical clustering of samples based on the mutation spectra.

Usage

```
cluster_spectra(
  mf_data = mf_data,
  group_col = "sample",
  response_col = "proportion_min",
  subtype_col = "normalized_subtype",
  dist = "cosine",
  cluster_method = "ward.D"
)
```

Arguments

- `mf_data` A data frame containing the mutation data. This data must include a column containing the mutation subtypes, a column containing the sample/cohort names, and a column containing the response variable.
- `group_col` The name of the column in data that contains the sample/cohort names.
- `response_col` The name of the column in data that contains the response variable. Typical response variables can be the subtype mf, proportion, or count.
- `subtype_col` The name of the column in data that contains the mutation subtypes.
- `dist` the distance measure to be used. This must be one of "cosine", "euclidean", "maximum", "manhattan","canberra", "binary" or "minkowski". See dist for details.
- `cluster_method` The agglomeration method to be used. See hclust for details.

Details

The cosine distance measure represents the inverted cosine similarity between samples:

$\text{Cosine Dissimilarity} = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$

This equation calculates the cosine dissimilarity between two vectors A and B.

Value

A dendrogram object representing the hierarchical clustering of the samples.

`context_list` *A list of reference contexts at different resolutions*

Description

A list of reference contexts at different resolutions

Usage

```
context_list
```

Format

A list with corresponding values

denominator_dict *Values used for denominators in frequency calculations*

Description

These values are used to cross reference base substitution types to their appropriate denominators for calculations. That is", "for example, the 6 base substitution frequency should be subsetted based on the normalized_ref column which would contain only T or C (i.e., the pyrimidine context for base substitutions).

Usage

```
denominator_dict
```

Format

A vector with corresponding values

f.proast *Run dose-response modeling using PROAST.*

Description

Run dose-response modeling using PROAST.

Usage

```
f.proast(
  odt = list(),
  ans.all = 0,
  er = FALSE,
  resize = FALSE,
  scale.ans = FALSE,
  const.var = FALSE,
  show.warnings = FALSE,
  interactive_mode = TRUE,
  datatype = NULL,
  model_choice = NULL,
  setting_choice = NULL,
  nested_model_choice = NULL,
  indep_var_choice = NULL,
  Vyans_input = NULL,
  covariates = NULL,
  custom_CES = 0.05,
  model_selection = NULL,
```

```
  lower_dd = NULL,
  upper_dd = NULL,
  selected_model = NULL,
  adjust_CES_to_group_SD = NULL,
  model_averaging = NULL,
  num_bootstraps = NULL,
  display_plots = TRUE,
  add_nonzero_val_to_dat = FALSE,
  nonzero_val = NULL,
  detection_limit = NULL
)
```

Arguments

`interactive_mode`
: A TRUE/FALSE value specifying whether you want to run interactively (i.e., TRUE, the default) or using command-line mode (i.e., FALSE, non-interactive). If FALSE, you must provide all other parameters.

`datatype`
: Non-interactive mode parameter. What type of response data do you want to consider? Options are 'continuous, individual data'.

`model_choice`
: Non-interactive mode parameter. Do you want to fit a single model or fit various nested families of models? Options are 'single model', 'select model 3 or 5 from various families of models', 'select model 3 from various nested families of models', 'select model 5 from various nested families of models', 'select model 15 in terms of RPF'. Recommended: 'select model 3 or 5 from various families of models'.

`setting_choice`
: Non-interactive mode parameter. Do you want to fit a set of models, or choose a single model? Options are 'single model', 'set of models'. Recommended: 'set of models'.

`nested_model_choice`
: ???

`indep_var_choice`
: Non-interactive mode parameter. The column name for the independent variable to use.

`Vyans_input`
: Non-interactive mode parameter. The column name(s) for the response variable(s) to use. If multiple, provide as a vector.

`covariates`
: Non-interactive mode parameter. The column name for the covariate to use. If none, enter 0.

`custom_CES`
: Non-interactive mode parameter. The critical effect size (BMR) to use, when adjust_CES_to_group_SD = 1 (FALSE).

`model_selection`
: Non-interactive mode parameter. The model selection to use. Options are "Exponential model only", "Exponential and Hill model", "previous option with inverse exponential model added" (run Expon, Hill, and Inv-Expon), "previous option with lognormal DR model added" (run Expon, Hill, Inv-Expon, and LN). Recommended: "previous option with lognormal DR model added".

lower_dd
: Non-interactive mode parameter. The lower constraint on d parameter. If NULL, existing defaults are used.

upper_dd
: Non-interactive mode parameter. The upper constraint on d parameter. If NULL, existing defaults are used.

selected_model
: Non-interactive mode parameter. Which model do you want to continue with? Options are "exponential", "Hill", "inverse exponential", "lognormal DR". The-function will output results for all models regardless of this choice. Really just to bypass the menu option. Recommended: "exponential".

adjust_CES_to_group_SD
: Non-interactive mode parameter. Set the BMR to the group standard deviation. Options are 1 (FALSE) or 2 (TRUE).

model_averaging
: Non-interactive mode parameter. Whether to perform model averaging to calculate 90% confidence intervals. TRUE/FALSE.

num_bootstraps
: Non-interactive mode parameter. The number of bootstraps to perform for model averaging. Recommended: 200.

display_plots
: Non-interactive mode parameter. Whether to display plots. TRUE/FALSE.

add_nonzero_val_to_dat
: Non-interactive mode parameter. When the response data contains 0s, whether to add a non-zero value to each observation. TRUE/FALSE. If TRUE, set the nonzero_val parameter with your desired (positive) number. If FALSE, a detection limit will used. Provide the detection limit in the detection_limit parameter. If no detection_limit is given, the function will use the minimum non-zero value in the data. Values below the detection limit will be plotted as half the detection limit.

nonzero_val
: Non-interactive mode parameter. The non-zero value to add to each observation when add_nonzero_val_to_dat = TRUE. Must be a positive number.

detection_limit
: Non-interactive mode parameter. The detection limit to use when add_nonzero_val_to_dat = FALSE. If NULL, the minimum non-zero value in the data will be used. This paramater accepts a numeric value, which will be applied to all response values, or a column name in the data, which will be used to apply different detection limits to different observations.

Value

Results from PROAST.

---

filter_mut *Filter your mutation data*

Description

This function creates a filter_mut column that will be read by the `calculate_mf` function. Variants with filter == TRUE will not be included in final mutation counts. This function may also remove records of given loci from the mutation data based on user specification. Running this function again on the same data will not overide the previous filters. To reset previous filters, set the filter_mut column values to FALSE.

Usage

```
filter_mut(
  mutation_data,
  correct_depth = FALSE,
  correct_depth_to_indel = TRUE,
  vaf_cutoff = 1,
  snv_in_germ_mnv = FALSE,
  rm_abnormal_vaf = FALSE,
  custom_filter_col = NULL,
  custom_filter_val = NULL,
  custom_filter_rm = FALSE,
  regions = NULL,
  regions_filter,
  allow_half_overlap = FALSE,
  rg_sep = "\t",
  is_0_based_rg = TRUE,
  rm_filtered_mut_from_depth = FALSE,
  return_filtered_rows = FALSE
)
```

Arguments

`mutation_data` Your mutation data.

`correct_depth` A logical value. If TRUE, the function will correct the `total_depth` column in `mutation_data` in order to prevent double-counting the `total_depth` values for the same genomic position. For rows with the same sample contig, and start values, the `total_depth` will be retained for only one row. All other rows in the group will have their `total_depth` set to 0. The default is FALSE

`correct_depth_to_indel` A logical value. If TRUE, during depth correction, should there be different `total_depth` values within a group of rows with the same sample, contig, and start values, the `total_depth` value for the row with the highest priority `variation_type` will be retained, while the other rows will have their `total_depth` set to 0. `variation_type` priority order is: deletion, complex, insertion, snv, mnv, sv, uncategorised, no_variant. If FALSE, the `total_depth` value for the first row in the group will be retained, while the other rows will have their `total_depth` set to 0. The default is TRUE.

`vaf_cutoff` Filter out ostensibly germline variants using a cutoff for variant allele fraction (VAF). Any variant with a `vaf` larger than the cutoff will be filtered. The default is 1 (no filtering). It is recommended to use a value of 0.01 (i.e. 1%) to retain only somatic variants.

`snv_in_germ_mnv`
: Filter out snv variants that overlap with germline mnv variants within the same samples. mnv variants will be considered germline if their vaf > vaf_cutoff. Default is FALSE.

`rm_abnormal_vaf`
: A logical value. If TRUE, rows in `mutation_data` with a variant allele fraction (VAF) between 0.05 and 0.45 or between 0.55 and 0.95 will be removed. We expect variants to have a VAF ~0. 0.5, or 1, reflecting rare somatic mutations, heterozygous germline mutations, and homozygous germline mutations, respectively. Default is FALSE.

`custom_filter_col`
: The name of the column in mutation_data to apply a custom filter to. This column will be checked for specific values, as defined by `custom_filter_val`. If any row in this column contains one of the specified values, that row will either be flagged in the `filter_mut` column or, if specified by `custom_filter_rm`, removed from mutation_data.

`custom_filter_val`
: A set of values used to filter rows in `mutation_data` based on `custom_filter_col`. If a row in `custom_filter_col` matches any value in `custom_filter_val`, it will either be set to TRUE in the `filter_mut` column or removed, depending on `custom_filter_rm`.

`custom_filter_rm`
: A logical value. If TRUE, rows in custom_filter_col that match any value in custom_filter_val will be removed from the mutation_data. If FALSE, filter_mut will be set to TRUE for those rows.

`regions`
: Remove rows that are within or outside of specified regions. Provide either a data frame or a file path of the specified intervals. Your regions must contain "contig", "start", and "end". Use the `regions_filter` parameter to specify whether rows within the regions should be kept versus removed. To use one of the TSpanels, set the regions parameter as follows: `regions = load_regions_file("TSpanel_mouse")`. Change the species as needed for human/rat.

`regions_filter`
: Specifies how the provided `regions` should be applied to `mutation_data`. Acceptable values are "remove_within" or "keep_within". If set to "remove_within", any rows that fall within the specified regions wil be removed from mutation_data. If set to "keep_within", only the rows within the specified regions will be kept in mutation_data, and all other rows will be removed.

`allow_half_overlap`
: A logical value. If TRUE, rows that start or end in your `regions`, but extend outside of them in either direction will be included in the filter. If FALSE, only rows that start and end within the `regions` will be included in the filter. Default is FALSE.

`rg_sep`
: The delimiter for importing the `regions` file, if applicable. Default is tab-delimited.

`is_0_based_rg`
: A logical value indicating whether the genomic intervals you provided in `regions` are 0_based. Set this to TRUE if you are using one of the TSpanels.

`rm_filtered_mut_from_depth`
: A logical value. If TRUE, the function will subtract the `alt_depth` of rows that were flagged by the `filter_mut` column from their `total_depth`. This will treat flagged variants as N-calls. This will not apply to variants flagged as germline by the `vaf_cutoff`. However, if the germline variant has additional filters applied, then the subtraction will still occur. If FALSE, the `alt_depth` will be retained in the `total_depth` for all variants. Default is FALSE.

`return_filtered_rows`
: A logical value. If TRUE, the function will return both the filtered mutation data and the rows that were removed/flagged in a seperate data frame. The two dataframes will be returned inside a list, with names `mutation_data` and `filtered_rows`. Default is FALSE.

Details

Depth correction is important for preventing double-counting of reads in mutation data when summing the total_depth across samples or other groups. Generally, when several mutations have been detected at the same genomic position, within a sample, the total_depth value will be the same for all of them. However, in some datasets, whenever a deletion is detected, the data may contain an additional row with the same genomic position calling a "no_variant". The total_depth will differ between the deletion and the no_variant. In these cases, correct_depth_to_indel == TRUE will ensure that the total_depth value for the deletion is retained, while the total_depth value for the no_variant is removed.

Examples

```
# Load example data
example_file <- system.file("extdata", "example_mutation_data.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
# Filter the data
## Basic Usage: correct the depth and filter out germline variants
filter_example_1 <- filter_mut(mutation_data = example_data,
                               correct_depth = TRUE,
                               vaf_cutoff = 0.01)
## Remove rows outside of specified regions (TwinStand Mouse Mutagenesis Panel)
filter_example_2 <- filter_mut(mutation_data = example_data,
                               correct_depth = TRUE,
                               vaf_cutoff = 0.01,
                               regions = load_regions_file("TSpanel_mouse"),
                               regions_filter = "keep_within")
## Apply a custom filter to flag rows with "EndRepairFillInArtifact" in the column 'filter'
filter_example_3 <- filter_mut(mutation_data = example_data,
                               correct_depth = TRUE,
                               vaf_cutoff = 0.01,
                               regions = load_regions_file("TSpanel_mouse"),
                               regions_filter = "keep_within",
                               custom_filter_col = "filter",
                               custom_filter_val = "EndRepairFillInArtifact",
                               custom_filter_rm = FALSE)
## Flag snv variants that overlap with germline mnv variants.
## Subtract the alt_depth of these variants from their total_depth (treat them as N-calls).
```

```
## Return all the flagged/removed rows in a seperate data frame
filter_example_4 <- filter_mut(mutation_data = example_data,
                               correct_depth = TRUE,
                               vaf_cutoff = 0.01,
                               regions = load_regions_file("TSpanel_mouse"),
                               regions_filter = "keep_within",
                               custom_filter_col = "filter",
                               custom_filter_val = "EndRepairFillInArtifact",
                               custom_filter_rm = FALSE,
                               snv_in_germ_mnv = TRUE,
                               rm_filtered_mut_from_depth = TRUE,
                               return_filtered_rows = TRUE)
# Flagging germline mutations...
# Found 612 germline mutations.
# Flagging SNVs overlapping with germline MNVs...
# Found 20 SNVs overlapping with germline MNVs.
# Applying custom filter...
# Flagged 2021 rows with values in <filter> column that matched EndRepairFillInArtifact
# Applying region filter...
# Removed 22 rows based on regions.
# Correcting depth...
# 909 rows had their total_depth corrected.
# Removing filtered mutations from the total_depth...
# Filtering complete.
# Returning a list: mutation_data and filtered_rows.
filtered_rows <- filter_example_4$filtered_rows
filtered_example_mutation_data <- filter_example_4$mutation_data
```

`get_binom_ci` *Add binomial confidence intervals to mutation frequencies.*

Description

Uses the binomial distribution to create confidence intervals for mutation frequencies calculated from a single point estimate. Calculating binomial confidence intervals for mutation frequencies is not part of MutSeqR's recommended workflow, but is provided here for users who wish to use it.

Usage

```
get_binom_ci(
  mf_data,
  sum_col = "sum_min",
  depth_col = "group_depth",
  conf_level = 0.95,
  method = "wilson"
)
```

Arguments

- `mf_data` — The data frame containing the mutation frequencies per sample. Obtained as an output from `calculate_mf`.
- `sum_col` — Column name that specifies the mutation count (e.g., sum_min)
- `depth_col` — Column name that specifies the sequencing depth (e.g., total_depth)
- `conf_level` — Confidence interval to calculate, default 95% (0.95)
- `method` — The method used by binom::binom.confint to calculate intervals. Default is "wilson" (recommended).

Value

A mf data frame with added columns indicating the confidence intervals.

Examples

```
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
mf <- calculate_mf(example_data)
confint <- get_binom_ci(mf_data = mf,
                        sum_col = "sum_min",
                        depth_col = "group_depth")
```

---

get_CpG_mutations   *Get mutations at CpG sites*

Description

Subset the mutation data provided and return only mutations that are found at CpG sites.

Usage

```
get_CpG_mutations(
  regions,
  mut_data,
  variant_types = c("snv", "insertion", "deletion", "mnv", "symbolic"),
  include_no_variants = TRUE,
  motif = "CG"
)
```

Arguments

- `regions` — A GRanges object containing the genomic regions of interest in which to look for CpG sites. Must have the metadata column "sequence" populated with the raw nucleotide sequence to search for CpGs. This object can be obtained using the get_seq.R function.
- `mut_data` — A dataframe or GRanges object containing the mutation data to be interrogated.

`variant_types` Include these variant types. A vector of one or more "snv", "complex", "deletion", "insertion", "mnv", "symbolic", "no_variant". Default includes all variants.

`include_no_variants`
TRUE or FALSE to indicate whether the table should include CpG sites with no variants. Useful if you want to know how many of the potential sites were mutated.

`motif` Default "CG", which returns CpG sites. You could in theory use an arbitrary string to look at different motifs. Use with caution.

Value

A GRanges object where each range is a mutation at a CpG site (a subset of mutations from the larger object provided to the function).

`get_CpG_regions` *Get the coordinates of the CpG sites within your genomic regions*

Description

Filters the ranges of your genomic regions to find a positions with a specific motif. The default is CpG sites, but can be customizable.

Usage

```
get_CpG_regions(regions, motif = "CG")
```

Arguments

`regions` A GRanges object containing the genomic regions of interest in which to look for CpG sites. Must have the metadata column "sequence" populated with the raw nucleotide sequence to search for CpGs. This object can be obtained using the get_seq.R function.

`motif` Default "CG", which returns CpG sites. You could in theory use an arbitrary string to look at different motifs. Use with caution.

Value

A GRanges object where each range is a CpG site (a subset of ranges from the larger object provided to the function).

`get_ref_of_mut` *A utility function that will return the reference context of a mutation*

Description

A utility function that will return the reference context of a mutation

Usage

```
get_ref_of_mut(mut_string)
```

Arguments

`mut_string`	the mutation. Ex. T>C, `A[G>T]C`

`get_seq` *Get sequence of genomic target regions*

Description

Create a GRanges object from the genomic target ranges and import raw nucleotide sequences.

Usage

```
get_seq(
  regions,
  custom_regions = NULL,
  rg_sep = "\t",
  is_0_based = TRUE,
  species = NULL,
  genome = NULL,
  masked = FALSE,
  padding = 0,
  ucsc = FALSE
)
```

Arguments

`regions`	The file containing the genomic regions. Regions TwinStrand Mutagenesis Panels are stored in the package files and can be accessed using the values `TSpanel_human`, `TSpanel_mouse`, and `TSpanel_rat`. If you have a custom range of genomic regions, set the value to `custom` and provide the regions file using the `custom_regions` argument.
`custom_regions`	If `regions = "custom"`, provide the genomic ranges. Can be a file path or a data frame. Required columns are `contig`, `start`, and `end`.

`rg_sep`	The delimiter for importing the custom_regions. The default is tab-delimited.
`is_0_based`	A logical variable. Indicates whether the position coordinates in the `custom_regions` are 0 based (TRUE) or 1 based (FALSE). If TRUE, positions will be converted to 1-based. Need not be supplied for TSpanels.
`species`	The species for which to retrieve the sequences. Species may be given as the scientific name or the common name. Ex. "Human", "Homo sapien". Used to choose the appropriate BS genome. Need not be supplied for TSpanels.
`genome`	The genome assembly version for which to retrieve the sequencies. Used to choose the appropriate genome (BS genome or UCSC). Ex. hg38, hg19, mm10, mm39, rn6, rn7. Need not be supplied for TSpanels.
`masked`	A logical value indicating whether to use the masked version of the BS genome when retrieving sequences. Default is FALSE.
`padding`	An integer value by which the function will extend the range of the target sequence on both sides. Start and end coordinates will be adjusted accordingly. Default is 0.
`ucsc`	A logical value. If TRUE, the function will retrieve the sequences from the UCSC genome browser using an API. If FALSE, the function will retrieve sequencies using the appropriate BSgenome package, which will be installed as needed.

Details

Consult `available.genomes(splitNameParts=FALSE, type=getOption("pkgType"))` for a full list of the available BS genomes and their associated species/genome/masked values. The BSgenome package will be installed if not already available. If using the UCSC API, the function will retrieve the sequences from the UCSC genome browser using the DAS API. See the UCSC website for available genomes: https://genome.ucsc.edu.

Value

a GRanges object with sequences of targeted regions. Region ranges coordinates will become 1-based.

Examples

```
# Example 1: Retrieve the sequences for TwinStrand Mouse Mutagenesis Panel
regions_seq <- get_seq(regions = "TSpanel_mouse")

# Example 2: Retrieve the sequences for custom regions
# We will load the TSpanel_human regions file as an example
human <- load_regions_file("TSpanel_human")
regions_seq <- get_seq(regions = "custom",
                       custom_regions = human,
                       is_0_based = TRUE,
                       species = "human",
                       genome = "hg38",
                       masked = FALSE,
                       padding = 0)
```

import_mut_data *Import tabular mutation data*

Description

Imports tabular mutation file into the local R environment.

Usage

```
import_mut_data(
  mut_file,
  mut_sep = "\t",
  is_0_based_mut = TRUE,
  sample_data = NULL,
  sd_sep = "\t",
  regions = "none",
  custom_regions = NULL,
  rg_sep = "\t",
  is_0_based_rg = TRUE,
  range_buffer = 0,
  genome = NULL,
  species = NULL,
  masked_BS_genome = FALSE,
  custom_column_names = NULL,
  output_granges = FALSE
)
```

Arguments

`mut_file` The mutation data file(s) to be imported. This can be either a data frame object or a filepath to a file or directory. If you specify a directory, the function will attempt to read all files in the directory and combine them into a single data frame. Mutation data should consist of a row for each variant. Required columns are listed below.

- `contig`: The name of the reference sequence.
- `start`: The start position of the feature.
- `end`: The half-open end position of the feature.
- `sample`: The sample name.
- `ref`: The reference allele at this position
- `alt`: The left-aligned, normalized, alternate allele at this position. Multiple alt alleles called for a single position should be represented as separate rows in the table.

The following columns are not required, but are recommended for full package functionality:

- `alt_depth`: The read depth supporting the alternate allele. If not included, the function will add this column, assuming a value of 1.

- `total_depth`: The total read depth at this position, excluding no-calls (N calls). If not present, the function will attempt to calculate the `total_depth` as `depth - no_calls`. If no_calls is not present, the function will use `depth` as the `total_depth`.
- `depth`: The total read depth at this position, including no-calls.
- `no_calls`: The number of no-calls (N-calls) at this position.

We recommend that files include a record for every sequenced position, regardless of whether a variant was called, along with the `total_depth` for each record. This enables site-specific depth calculations required for some downstream analyses.

`mut_sep` The delimiter for importing the mutation file. Default is tab-delimited.

`is_0_based_mut` A logical variable. Indicates whether the position coordinates in the mutation data are 0 based (TRUE) or 1 based (FALSE). If TRUE, positions will be converted to 1-based.

`sample_data` An optional file containing additional sample metadata (dose, timepoint, etc.). This can be a data frame or a file path. Metadata will be joined with the mutation data based on the sample column. Required columns are `sample` and any additional columns you wish to include.

`sd_sep` The delimiter for importing sample data. Default is tab-delimited.

`regions` An optional file containing metadata of genomic regions. Region metadata will be joined with mutation data and variants will be checked for overlap with the regions. Metadata for TwinStrand Mutagenesis Panels are stored in the package files and can be accessed using the values `TSpanel_human`, `TSpanel_mouse`, and `TSpanel_rat`. If you have a custom range of genomic regions, set the value to `custom` and provide the regions file using the `custom_regions` argument. If you do not wish to include region metadata, set value to `none`.

`custom_regions` If `regions` is set to "custom", provide the regions metadata. Can be a file path or a data frame. Required columns are `contig`, `start`, and `end`.

`rg_sep` The delimiter for importing the `custom_regions` file. Default is tab-delimited.

`is_0_based_rg` A logical variable. Indicates whether the position coordinates in the custom_regions are 0 based (TRUE) or 1 based (FALSE). If TRUE, positions will be converted to 1-based.

`range_buffer` An integer >= 0. Extend the range of your regions in both directions by the given amount. Ex. Structural variants and indels may start outside of the regions. Adjust the range_buffer to include these variants in your region's ranges.

`genome` The genome assembly version of the reference genome. This is required if your data does not include a context column. The function will install a BS genome for the given species/genome/masked arguments to populate the context column. Ex.Human GRCh38 = hg38 | Human GRCh37 = hg19 | Mouse GRCm38 = mm10 | Mouse GRCm39 = mm39 | Rat RGSC 6.0 = rn6 | Rat mRatBN7.2 = rn7

`species` The species. Required if your data does not include a context column. The function will install a BS genome for the given species/genome/masked to populate the context column. The species can be the common name of the species or the scientific name. Ex. "human" or "Homo sapiens".

masked_BS_genome
: A logical value. Required when using a BS genome to poulate the context column. Whether to use the masked version of the BS genome (TRUE) or not (FALSE). Default is FALSE.

custom_column_names
: A list of names to specify the meaning of column headers. Since column names can vary with data, this might be necessary to digest the mutation data properly. Typical defaults are set, but can be substituted in the form of `list(contig = "my_custom_contig_name", sample = "my_custom_sample_column_name")`. You can change one or more of these. Set column synonyms are defined in MutSeqR::op$column and will automatically be changed to their default value.

output_granges
: A logical variable; whether you want the mutation data to output as a GRanges object. Default output (FALSE) is as a dataframe.

Value

A table where each row is a mutation, and columns indicate the location, type, and other data. If `output_granges` is set to TRUE, the mutation data will be returned as a GRanges object, otherwise mutation data is returned as a dataframe.

Output Column Definitions:

- `short_ref`: The reference base at the start position.
- `normalized_ref`: The short_ref in C/T-base notation for this position (e.g. A -> T, G -> C).
- `context` The trinucleotide context at this position. Consists of the reference base and the two flanking bases (e.g. TAC).
- `normalized_context`: The trinucleotide context in C/T base notation for this position (e.g. TAG -> CTA).
- `variation_type` The type of variant (snv, mnv, insertion, deletion, complex, sv, no_variant, ambiguous, uncategorized).
- `subtype` The substitution type for the snv variant (12-base spectrum; e.g. A>C).
- `normalized_subtype` The C/T-based substitution type for the snv variant (6-base spectrum; e.g. A>C -> T>G).
- `context_with_mutation`: The substitution type for the snv variant including the two flanking nucleotides (192-trinucleotide spectrum; e.g. `T[A>C]G`)
- `normalized_context_with_mutation`: The C/T-based substitution type for the snv variant including the two flanking nucleotides (96-base spectrum e.g. `T[A>C]G` -> `C[T>G]A`).
- `nchar_ref`: The length (in bp) of the reference allele.
- `nchar_alt`: The length (in bp) of the alternate allele.
- `varlen`: The length (in bp) of the variant.
- `ref_depth`: The depth of the reference allele. Calculated as `total_depth - alt_depth`, if applicable.
- `vaf` : The variant allele fraction. Calculated as `alt_depth/total_depth`.
- `gc_content`: % GC of the trinucleotide context at this position.
- `is_known`: TRUE or FALSE. Flags known variants (ID != ".").
- `row_has_duplicate`: TRUE or FALSE. Flags rows whose position is the same as that of at least one other row for the same sample.

Examples

```
# Example: Import a single mutation file. This library was sequenced with
# Duplex Sequencing using the TwinStrand Mouse Mutagenesis Panel which
# consists of 20 2.4kb targets = 48kb of sequence.
example_file <- system.file("extdata", "example_import_mut_data.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
# We will create an example metadata table for this data.
sample_meta <- data.frame(sample = "dna00996.1",
                          dose = "50",
                          dose_group = "High")
# Import the data
imported_example_data <- import_mut_data(mut_file = example_data,
                                         sample_data = sample_meta,
                                         regions = "TSpanel_mouse",
                                         genome = "mm10",
                                         species = "mouse",
                                         masked_BS_genome = FALSE)
```

---

import_vcf_data *Import a VCF file*

Description

The function reads VCF file(s) and extracts the data into a dataframe.

Usage

```
import_vcf_data(
  vcf_file,
  sample_data = NULL,
  sd_sep = "\t",
  regions = "none",
  custom_regions = NULL,
  rg_sep = "\t",
  is_0_based_rg = FALSE,
  range_buffer = 0,
  genome = NULL,
  species = NULL,
  masked_BS_genome = FALSE,
  output_granges = FALSE
)
```

Arguments

`vcf_file` The path to the .vcf (.gvcf, gzip, bgzip) to be imported. If you specify a directory, the function will attempt to read all files in the directory and combine them into a single table. VCF files should follow the VCF specifications, version 4.5. Multisample VCF files are not supported; VCF files must contain one sample each. Required fields are listed below.

- FIXED FIELDS
- CHROM: The name of the reference sequence. Equivalent to contig.
- POS: The 1-based start position of the feature. Equivalent to start.
- REF: The reference allele at this position.
- ALT: The left-aligned, normalized, alternate allele at this position. Multiple alt alleles called for a single position should be represented as separate rows in the table.
- INFO FIELDS
- END: The half-open end position of the feature.
- sample: An identifying field for your samples; either in the INFO field or as the header to the FORMAT field.

The following FORMAT fields are not required, but are recommended for full package functionality:

- AD: The allelic depths for the reference and alternate allele in the order listed. The sum of AD is equivalent to the total_depth (read depth at this position excluding N-calls).
- DP: The read depth at this position (including N-calls). Equivalent to depth. Note that in many VCF files, the DP field is defined as total_depth. However, in most cases, the DP field includes N-calls.
- VD: The read depth supporting the alternate allele. If not included, the function will add this column, assuming a value of 1. Equivalent to alt_depth.

We recommend that files include a record for every sequenced position, regardless of whether a variant was called, along with the AD for each record. This enables site-specific depth calculations required for some downstream analyses. AD is used to calculate the total_depth (the read depth excluding No-calls). If AD is not available, the DP field will be used as the total_depth.

sample_data
: An optional file containing additional sample metadata (dose, timepoint, etc.). This can be a data frame or a file path. Metadata will be joined with the mutation data based on the sample column. Required columns are sample and any additional columns you wish to include.

sd_sep
: The delimiter for importing sample metadata tables. Default is tab-delimited.

regions
: An optional file containing metadata of genomic regions. Region metadata will be joined with mutation data and variants will be checked for overlap with the regions. Metadata for TwinStrand Mutagenesis Panels are stored in the package files and can be accessed using the values TSpanel_human, TSpanel_mouse, and TSpanel_rat. If you have a custom range of genomic regions, set the value to custom and provide the regions file using the custom_regions argument. If you do not wish to include region metadata, set value to none.

custom_regions
: If regions is set to "custom", provide the regions metadata. Can be a file path or a data frame. Required columns are contig, start, and end.

rg_sep
: The delimiter for importing the custom_regions. Default is tab-delimited.

is_0_based_rg
: A logical variable. Indicates whether the position coordinates in the custom_regions are 0 based (TRUE) or 1 based (FALSE). If TRUE, positions will be converted to 1-based.

range_buffer
: Extend the range of your regions in both directions by the given amount. Ex. Structural variants and indels may start outside of the regions. Adjust the range_buffer to include these variants in your region's ranges.

genome
: The genome assembly version of the reference genome. This is required if your data does not include a context column. The function will install a BS genome for the given species/genome/masked arguments to populate the context column. Ex.Human GRCh38 = hg38 | Human GRCh37 = hg19 | Mouse GRCm38 = mm10 | Mouse GRCm39 = mm39 | Rat RGSC 6.0 = rn6 | Rat mRatBN7.2 = rn7

species
: The species. Required if your data does not include a context column. The function will install a BS genome for the given species/genome/masked to populate the context column. The species can be the common name of the species or the scientific name. Ex. "human" or "Homo sapiens".

masked_BS_genome
: A logical value. Required when using a BS genome to poulate the context column. Whether to use the masked version of the BS genome (TRUE) or not (FALSE). Default is FALSE.

output_granges
: TRUE or FALSE; whether you want the mutation data to output as a GRanges object. Default output is as a dataframe.

Value

A table where each row is a mutation, and columns indicate the location, type, and other data. If output_granges is set to TRUE, the mutation data will be returned as a GRanges object, otherwise mutation data is returned as a dataframe.

Output Column Definitions:

- short_ref: The reference base at the start position.
- normalized_ref: The short_ref in C/T-base notation for this position (e.g. A -> T, G -> C).
- context The trinucleotide context at this position. Consists of the reference base and the two flanking bases (e.g. TAC).
- normalized_context: The trinucleotide context in C/T base notation for this position (e.g. TAG -> CTA).
- variation_type The type of variant (snv, mnv, insertion, deletion, complex, sv, no_variant, ambiguous, uncategorized).
- subtype The substitution type for the snv variant (12-base spectrum; e.g. A>C).
- normalized_subtype The C/T-based substitution type for the snv variant (6-base spectrum; e.g. A>C -> T>G).
- context_with_mutation: The substitution type for the snv variant including the two flanking nucleotides (192-trinucleotide spectrum; e.g. T[A>C]G)
- normalized_context_with_mutation: The C/T-based substitution type for the snv variant including the two flanking nucleotides (96-base spectrum e.g. T[A>C]G -> C[T>G]A).
- nchar_ref: The length (in bp) of the reference allele.
- nchar_alt: The length (in bp) of the alternate allele.
- varlen: The length (in bp) of the variant.

- ref_depth: The depth of the reference allele. Calculated as total_depth - alt_depth, if applicable.
- vaf : The variant allele fraction. Calculated as alt_depth/total_depth.
- gc_content: % GC of the trinucleotide context at this position.
- is_known: TRUE or FALSE. Flags known variants (ID != ".").
- row_has_duplicate: TRUE or FALSE. Flags rows whose position is the same as that of at least one other row for the same sample.

Examples

```
# Example: Import a single bg-zipped vcf file. This library was sequenced
# with Duplex Sequencing using the TwinStrand Mouse Mutagenesis Panel which
# consists of 20 2.4kb targets = 48kb of sequence.
example_file <- system.file("extdata",
                            "example_import_vcf_data_cleaned.vcf.bgz",
                            package = "MutSeqR")
# We will create an example metadata table for this data.
sample_meta <- data.frame(sample = "dna00996.1",
                          dose = "50",
                          dose_group = "High")
# Import the data
imported_example_data <- import_vcf_data(vcf_file = example_file,
                                         sample_data = sample_meta,
                                         regions = "TSpanel_mouse",
                                         genome = "mm10",
                                         species = "mouse",
                                         masked_BS_genome = FALSE)
```

---

install_ref_genome *Install the reference genome for the specified organism.*

Description

This function will use BSgenome to install the reference genome for a specified organism and assembly version.

Usage

```
install_ref_genome(organism, genome, masked = FALSE)
```

Arguments

organism	the name of the organism for which to install the reference genome. This can be the scientific name or a common name. For example Homo Sapiens, H. sapiens, or human
genome	The reference genome assembly version. Ex. hg18, mm10, rn6.
masked	Logical value. Whether to search for the 'masked' BSgenome. Default is FALSE.

Value

a BSgenome object

`load_regions_file` *Imports the regions file*

Description

A helper function to import the regions metadata file. It is used in import_mut_data and get_seq.

Usage

```
load_regions_file(regions, custom_regions = NULL, rg_sep = "\t")
```

Arguments

`regions`	"TSpanel_human", "TSpanel_mouse", 'TSpanel_rat', or "custom". The argument refers to the TS Mutagenesis panel of the specified species, or to a custom panel. If custom, provide file path in custom_regions.
`custom_regions`	If regions is set to 'custom', provide the regions metadata. Can be either a file path or a data frame. Required columns are "contig", "start", and "end".
`rg_sep`	The delimiter for importing the custom_regions. The default is tab-delimited

---

`lollipop_mutations` *Plot mutations in lollipop plot*

Description

TO DO: Create plt without trackViewer package. Uses the trackViewer package to plot mutations in a lollipop plot in specific regions as defined by the user input.

Usage

```
lollipop_mutations(species = "human", mutations, ...)
```

Arguments

`species`	One of "human" or "mouse"
`mutations`	A GRanges object with mutation data
`...`	Additional arguments to trackViewer::lolliplot (e.g., `ranges = GRanges("chr1", IRanges(104, 109))`) Suggests trackViewer lolliplot

make_CpG_summary_table

Summarize CpG sites

Description

Creates a summary table of CpG sites based on groupings of interest. This is basically a convenience function that wraps `calculate_mfs()` over CpG data (or any data). See the documentation for that function for parameters. It is up to the user to supply proper data to the function.

Usage

```
make_CpG_summary_table(cpg_muts, ...)
```

Arguments

`cpg_muts`	A data frame containing CpG mutations TO DO: cpg_muts = df "cpg_mutations" is created in the .Rmd file, but is not created by any other function.
`...`	Additional arguments to calculate_mfs()

model_mf

Perform linear modelling on mutation frequency for given fixed and random effects

Description

`model_mf` will fit a linear model to analyse the effect(s) of given factor(s) on mutation frequency and perform specified pairwise comparisons. This function will fit either a generalized linear model (glm) or, if supplied random effects, a generalized linear mixed-effects model (glmer) . Pairwise comparisons are conducted using the doBy library (esticon) and estimates are then back-transformed. The delta method is employed to approximate the back-transformed standard-errors. A Sidak correction is applied to adjust p-values for multiple comparisons.

Usage

```
model_mf(
  mf_data,
  fixed_effects,
  test_interaction = TRUE,
  random_effects = NULL,
  reference_level,
  muts = "sum_min",
  total_count = "group_depth",
  contrasts = NULL,
  cont_sep = "\t",
  ...
)
```

Arguments

- `mf_data` — The data frame containing the mutation frequency data. Mutation counts and total sequencing depth should be summarized per sample alongside columns for your fixed effects. This data can be obtained using `calculate_mf(summary=TRUE)`.
- `fixed_effects` — The name(s) of the column(s) that will act as the fixed_effects (factor/independent variable) for modelling mutation frequency.
- `test_interaction` — a logical value. Whether or not your model should include the interaction between the `fixed_effects`.
- `random_effects` — The name of the column(s) to be analysed as a random effect in the model. Providing this effect will cause the function to fit a generalized linear mixed-effects model.
- `reference_level` — Refers to one of the levels within each of your fixed_effects. The coefficient for the reference level will represent the baseline effect. The coefficients of the other levels will be interpreted in relation to the reference_level as deviations from the baseline effect.
- `muts` — The column containing the mutation count per sample.
- `total_count` — The column containing the sequencing depth per sample.
- `contrasts` — a data frame or a filepath to a file that will provide the information necessary to make pairwise comparisons between groups. The table must consist of two columns. The first column will be a group within your fixed_effects and the second column must be the group that it will be compared to. The values must correspond to entries in your mf_data column for each fixed effect. Put the group that you expect to have the higher mutation frequency in the 1st column and the group that you expect to have a lower mutation frequency in the second column. For multiple fixed effects, separate the levels of each `fixed_effect` of a group with a colon. Ensure that all `fixed_effects` are represented in each entry for the table. See `details` for examples.
- `cont_sep` — The delimiter for importing the contrast table file. Default is tab-delimited.
- `...` — Extra arguments for glm or glmer. The `glmer` function is used when a `random_effect` is supplied, otherwise, the model uses the `glm` function.

Details

`fixed_effects` are variables that have a direct and constant effect on the dependent variable (ie mutation frequency).They are typically the experimental factors or covariates of interest for their impact on the dependent variable. One or more fixed_effect may be provided. If you are providing more than one fixed effect, avoid using correlated variables; each fixed effect must independently predict the dependent variable. Ex. `fixed_effects = c("dose", "genomic_target", "tissue", "age", etc)`.

Interaction terms enable you to examine whether the relationship between the dependent and independent variable changes based on the value of another independent variable. In other words, if an interaction is significant, then the relationship between the fixed effects is not constant across all levels of each variable. Ex. Consider investigating the effect of dose group and tissue on mutation frequency. An interaction between dose and tissue would capture whether the dose response differs between tissues.

`random_effects` account for the unmeasured sources of statistical variance that affect certain groups in the data. They help account for unobserved heterogeneity or correlation within groups. Ex. If your model uses repeated measures within a sample, `random_effects = "sample"`.

Setting a `reference_level` for your fixed effects enhances the interpretability of the model. Ex. Consider a `fixed_effect` "dose" with levels 0, 25, 50, and 100 mg/kg. Intuitively, the reference_level would refer to the negative control dose, "0" since we are interested in testing how the treatment might change mutation frequency relative to the control.

Examples of `contrasts`:

If you have a `fixed_effect` "dose" with dose groups 0, 25, 50, 100, then the first column would contain the treated groups (25, 50, 100), while the second column would be 0, thus comparing each treated group to the control group.

25 0

50 0

100 0

Alternatively, if you would like to compare mutation frequency between treated dose groups, then the contrast table would look as follows, with the lower dose always in the second column, as we expect it to have a lower mutation frequency. Keeping this format aids in interpretability of the estimates for the pairwise comparisons. Should the columns be reversed, with the higher group in the second column, then the model will compute the fold-decrease instead of the fold-increase.

100 25

100 50

50 25

Ex. Consider the scenario where the `fixed_effects` are "dose" (0, 25, 50, 100) and "genomic_target" ("chr1", "chr2"). To compare the three treated dose groups to the control for each genomic target, the contrast table would look like:

25:chr1 0:chr1

50:chr1 0:chr1

100:chr1 0:chr1

25:chr2 0:chr2

50:chr2 0:chr2

100:chr2 0:chr2

Troubleshooting: If you are having issues with convergence for your generalized linear mixed- effects model, it may be advisable to increase the tolerance level for convergence checking during model fitting. This is done through the `control` argument for the `lme4::glmer` function. The default tolerance is tol = 0.002. Add this argument as an extra argument in the `model_mf` function. Ex. `control = lme4::glmerControl(check.conv.grad = lme4::.makeCC("warning", tol = 3e-3, relTol = NULL))`

Value

Model results are output as a list. Included are:

- model_data: the supplied mf_data with added column for the Pearson's residuals of the model.
- summary: the summary of the model.
- anova: the analysis of variance for models with two or more effects. Anova(model)
- residuals_histogram: the Pearson's residuals plotted as a histogram. This is used to check whether the variance is normally distributed. A symmetric bell-shaped histogram, evenly distributed around zero indicates that the normality assumption is likely to be true.
- residuals_qq_plot: the Pearson's residuals plotted in a quantile-quantile plot. For a normal distribution, we expect points to roughly follow the y=x line.
- point_estimates_matrix: the contrast matrix used to generate point-estimates for the fixed effects.
- point_estimates: the point estimates for the fixed effects.
- pairwise_comparisons_matrix: the contrast matrix used to conduct the pairwise comparisons specified in the `contrasts`.
- pairwise_comparisons: the results of pairwise comparisons specified in the `contrasts`.

Examples

```
# Example 1: Model MFmin by dose
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)
mf_example <- calculate_mf(mutation_data = example_data,
                           cols_to_group = "sample",
                          retain_metadata_cols = "dose")
# Create a contrasts table to define pairwise comparisons
# We will compare all treated groups to the control group
contrasts <- data.frame(col1 = c("12.5", "25", "50"),
                        col2 = c("0", "0", "0"))
# Fit the model
model1 <- model_mf(mf_data = mf_example,
                   fixed_effects = "dose",
                   reference_level = "0",
                   muts = "sum_min",
                   total_count = "group_depth",
                   contrasts = contrasts)
# The residuals histogram and QQ plot will help you assess the normality
# of the residuals.
model1$summary # Model Summary
model1$point_estimates # Point Estimates: Mean MFmin by dose
model1$pairwise_comparisons # Pairwise Comparisons
# All treated doses exhibited a significant increase in mutation frequency
# compared to the control.

# Plot the results using plot_model_mf()
```

```
plot <- plot_model_mf(model1,
                      plot_type = "bar",
                      x_effect = "dose",
                      plot_error_bars = TRUE,
                      plot_signif = TRUE,
                      x_order = c("0", "12.5", "25", "50"),
                      x_label = "Dose (mg/kg-bw/d)",
                      y_label = "Estimated Mean MF (mutations/bp)",
                      plot_title = "")

# Example 2: Model MFmin by dose and genomic target
# We will compare the treated groups to the control group for each genomic
# target

# Calculate MF
mf_example2 <- calculate_mf(mutation_data = example_data,
                            cols_to_group = c("sample", "label"),
                            retain_metadata_cols = "dose")
# Create a contrasts table to define pairwise comparisons
combinations <- expand.grid(dose = unique(mf_example2$dose),
                            label = unique(mf_example2$label))
combinations <- combinations[combinations$dose != 0, ]
combinations$col1 <- with(combinations, paste(dose, label, sep=":"))
combinations$col2 <- with(combinations, paste("0", label, sep=":"))
contrasts2 <- combinations[, c("col1", "col2")]
# Fit the model
# Fixed effects of dose and label
# Random effect of sample
# Control the optimizer for convergence issues
model2 <- model_mf(mf_data = mf_example2,
                   fixed_effects = c("dose", "label"),
                   random_effects = "sample",
                   reference_level = c("0", "chr1"),
                   muts = "sum_min",
                   total_count = "group_depth",
                   contrasts = contrasts2,
                   control = lme4::glmerControl(optimizer = "bobyqa",
                                                optCtrl = list(maxfun = 2e5)))
model2$summary # Fits a GLMM
model2$point_estimates
model2$pairwise_comparisons

# Plot the results using plot_model_mf()
# Define the order of the labels for the x-axis
label_order <- model2$point_estimates %>%
 dplyr::filter(dose == "50") %>%
 dplyr::arrange(Estimate) %>%
 dplyr::pull(label)
# Define the order of the doses for the fill
dose_order <- c("0", "12.5", "25", "50")
plot <- plot_model_mf(model = model2,
                      plot_type = "bar",
                      x_effect = "label",
```

```
                      plot_error_bars = TRUE,
                      plot_signif = TRUE,
                      ref_effect = "dose",
                      x_order = label_order,
                      fill_order = dose_order,
                      x_label = "Target",
                      y_label = "MF (mutations/bp)",
                      fill_label = "Dose",
                      plot_title = "",
                      custom_palette = c("#ef476f",
                                         "#ffd166",
                                         "#06d6a0",
                                         "#118ab2"))
# The output is a ggplot object and can be modified using ggplot2
# functions. For example, to rotate the x-axis labels by 90 degrees,
# use the following code:
p <- plot + ggplot2::theme(axis.text.x = ggplot2::element_text(angle = 90))
```

---

op   *Column names for mut tables*

Description

A list of column specifications

Usage

```
op
```

Format

A list with potential variable column names

---

plot_bubbles   *Generate Bubble Plots*

Description

Produces a ggplot object of bubble plots from given mutation data. Optionally, bubble plots can be facetted and coloured by a specified column.

Usage

```
plot_bubbles(
  mutation_data,
  size_by = "alt_depth",
  facet_col = NULL,
  color_by = "normalized_subtype",
  circle_spacing = 1,
  circle_outline = "none",
  circle_resolution = 50,
  custom_palette = NULL
)
```

Arguments

- `mutation_data` Data frame containing the mutation data.
- `size_by` The column name by which to size the circles. Recommended values are "alt_depth" or "vaf".
- `facet_col` The column name by which to facet . If NULL, no facetting will be done. Default is NULL.
- `color_by` The column name by which to colour the mutations. Default is "normalized_subtype".
- `circle_spacing` Numerical value to adjust the spacing between circles. Default is 1.
- `circle_outline` Colour for the circle outline. Default is "none", resulting in no outline colour. Other accepted values are colours in the R language.
- `circle_resolution` Number of points to use for the circle resolution. Default is 50.
- `custom_palette` A named vector of colors to be used for the mutation subtypes. The names of the vector should correspond to the levels in color_by. Alternatively, you can specify a color palette from the RColorBrewer package. See `brewer.pal` for palette options. You may visualize the palettes at the ColorBrewer website: https://colorbrewer2.org/. Default is `NULL`.

Details

The function will plot a circle for each mutation in `mutation_data`. Mutations flagged by the `filter_mut` column will be excluded from the plot. The size of the circle is determined by the `size_by` parameter. Sizing by the "alt_depth" or the "vaf" will give users the ability to visualize the the distribution of recurrent mutations within their data with large multiplets having a large circle.

Value

A ggplot object with the bubble plot, facetted if specified.

Examples

```
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
plot <- plot_bubbles(mutation_data = example_data,
                     facet_col = "dose_group")
```

plot_ci *plot_ci*

Description

Plot confidence intervals

Usage

```
plot_ci(
  data,
  order = "none",
  custom_order = NULL,
  nudge = 0.3,
  log_scale = FALSE,
  x_lab = NULL,
  y_lab = NULL,
  title = NULL
)
```

Arguments

data	A data frame with the results of the BMD analysis. Data must contain columns "Response", "BMD", "BMDL", and "BMDU". BMD values can be NA.
order	Indicates how the responses should be ordered. Options are "none" (default), "asc" for ascending BMD values, "desc" for descending BMD values, or a custom order.
custom_order	A character vector with the custom order of the Responses.
nudge	A numeric value to nudge the text labels away from points. Default is 0.3.
log_scale	A logical value indicating if the x-axis should be in log10 scale. Default is false.
x_lab	A character string with the x-axis label. Default is "BMD" or "log10(BMD)" if log_scale is TRUE.
y_lab	A character string with the y-axis label. Default is "Response".
title	A character string with the plot title. Default is "BMD with 90% Confidence Intervals".

Value

a ggplot object

Examples

```
# Plot results from PROAST and ToxicR
dat <- data.frame(Response = c("PROAST MF Min", "PROAST MF Max", "ToxicR MF Min", "ToxicR MF Max"),
                  BMD = c(NA, NA, 9.641894, 8.100164),
```

```
                 BMDL = c(7.38, 2.98, 8.032936,5.463013),
                 BMDU = c(10.9, 7.68, 10.97636, 10.04638))
plot <- plot_ci(dat)
```

---

plot_mean_mf *Plot the Mean Mutatation Frequency*

Description

This function calculates the mean mutation frequency across samples for given groups and plots the results.

Usage

```
plot_mean_mf(
  mf_data,
  group_col = "dose",
  fill_col = NULL,
  mf_type = "both",
  plot_type = "line",
  plot_error_bars = TRUE,
  plot_indiv_vals = TRUE,
  group_order = "none",
  group_order_input = NULL,
  add_labels = "mean_count",
  scale_y_axis = "linear",
  x_lab = NULL,
  y_lab = NULL,
  plot_title = NULL,
  custom_palette = NULL
)
```

Arguments

mf_data	A data frame containing the mutation frequency data. This is obtained from the calculate_mf function with SUMMARY = TRUE.
group_col	The column in mf_data by which to calculate the mean. Ex. "dose" or c("dose", "tissue").
fill_col	The column in mf_data by which to fill the color. Default is NULL. fill_col must be a variable of equal or higher level to group_col. fill_col may equal group_col.
mf_type	The type of mutation frequency to plot. Options are "min", "max", "both", or "stacked". If "both", the min and max mutation frequencies are plotted side by side. "stacked" can be chosen for bar plot_type only. If "stacked", the difference between the min and max MF is stacked on top of the min MF such that the total height of both bars represent the max MF. Default is "min".
plot_type	The type of plot to create. Options are "bar" or "line". Default is "bar".

plot_error_bars
: Whether to plot the error bars. Default is TRUE. Error bars are standard error of the mean.

plot_indiv_vals
: Whether to plot the individual values as data points. Default is FALSE.

group_order
: The order of the groups along the x-axis. ' Options include:

- `none`: No ordering is performed. Default.
- `smart`: Groups are ordered based on the sample names.
- `arranged`: Groups are ordered based on one or more factor column(s) in mf_data. Factor column names are passed to the function using the `group_order_input`.
- `custom`: Groups are ordered based on a custom vector of group names. The custom vector is passed to the function using the `group_order_input`.

group_order_input
: The order of the groups if group_order is "custom". The column name by which to arrange groups if group_order is "arranged".

add_labels
: The data labels to display on the plot. Either "indiv_count", "indiv_MF", "mean_count", "mean_MF", or "none". Count labels display the number of mutations, MF labels display the mutation frequency. Mean plots the mean value. Indiv plots the labels for individual data points (only if plot_indiv_vals = TRUE). Default is "none".

scale_y_axis
: The scale of the y axis. Either "linear" or "log". Default is "linear".

x_lab
: The x-axis label. Default is the value of group_col.

y_lab
: The y-axis label. Default is "Mutation Frequency (mutations/bp)".

plot_title
: The title of the plot. Default is "Mean Mutation Frequency".

custom_palette
: A custom color palette to use for the plot. Input a character vector of colours. Input a named character vector to specify olours to specific groups. Fill labels will be constructed by the following components

1. "Mean/Individual" if plot_indiv_vals = TRUE, fill labels will specify Mean/Individual values.
2. "min/max" if mf_type = "both" or "stacked", fill labels will specify min/max values.
3. fill_col value. Name colours to match the fill labels. Default is NULL. If no custom_palette, a rainbow palette is generated. Min/Max values and Mean/Individual values will be the same colour, different shades.

Value

a ggplot object

Examples

```
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)
```

```
example_data$dose_group <- factor(example_data$dose_group,
                                  levels = c("Control", "Low",
                                             "Medium", "High"))
mf <- calculate_mf(mutation_data = example_data,
                   cols_to_group = "sample",
                   subtype_resolution = "none",
                   retain_metadata_cols = "dose_group")
plot <- plot_mean_mf(mf_data = mf,
                     group_col = "dose_group",
                     mf_type = "min",
                     plot_type = "line",
                     fill_col = "dose_group",
                     plot_error_bars = TRUE,
                     plot_indiv_vals = TRUE,
                     add_labels = "none")
```

---

plot_mf *Plot the Mutation Frequency*

Description

This function creates a plot of the mutation frequency.

Usage

```
plot_mf(
  mf_data,
  group_col,
  plot_type = "bar",
  mf_type = "min",
  fill_col = NULL,
  custom_palette = NULL,
  group_order = "none",
  group_order_input = NULL,
  labels = "count",
  scale_y_axis = "linear",
  x_lab = NULL,
  y_lab = NULL,
  title = NULL
)
```

Arguments

mf_data	A data frame containing the mutation frequency data. This is obtained from the calculate_mf function with SUMMARY = TRUE.
group_col	The name of the column containing the sample/group names for the x-axis.
plot_type	The type of plot to create. Options are "bar" or "point".

mf_type — The type of mutation frequency to plot. Options are "min", "max", "both", or "stacked". If "both", the min and max mutation frequencies are plotted side by side. "stacked" can be chosen for bar plot_type only. If "stacked", the difference between the min and max MF is stacked on top of the min MF such that the total height of both bars represent the max MF.

fill_col — The name of the column containing the fill variable.

custom_palette — A character vector of colour codes to use for the plot. If NULL, a default palette is used

group_order — The order of the samples/groups along the x-axis. ' Options include:

- `none`: No ordering is performed. Default.
- `smart`: Samples are ordered based on the sample names.
- `arranged`: Samples are ordered based on one or more factor column(s) in mf_data. Factor column names are passed to the function using the `group_order_input`.
- `custom`: Samples are ordered based on a custom vector of sample names. The custom vector is passed to the function using the `group_order_input`.

group_order_input — The order of the samples/groups if group_order is "custom". The column name by which to arrange samples/groups if group_order is "arranged"

labels — The data labels to display on the plot. Either "count", "MF", or "none". Count labels display the number of mutations, MF labels display the mutation frequency.

scale_y_axis — The scale of the y axis. Either "linear" or "log".

x_lab — The label for the x axis.

y_lab — The label for the y axis.

title — The title of the plot.

Value

A ggplot object

Examples

```
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)
example_data$dose_group <- factor(example_data$dose_group,
                                  levels = c("Control", "Low",
                                             "Medium", "High"))
mf <- calculate_mf(mutation_data = example_data,
                   cols_to_group = "sample",
                   subtype_resolution = "none",
                   retain_metadata_cols = "dose_group")
plot <- plot_mf(mf_data = mf,
                group_col = "sample",
                plot_type = "bar",
                mf_type = "min",
```

```
fill_col = "dose_group",
group_order = "arranged",
group_order_input = "dose_group",
labels = "count",
title = "Mutation Frequency per Sample")
```

`plot_model_mf` *Plot your mf model*

Description

Provide a visualization of the point estimates derived using model_mf()

Usage

```
plot_model_mf(
  model,
  plot_type = "point",
  x_effect = NULL,
  plot_error_bars = TRUE,
  plot_signif = TRUE,
  ref_effect = NULL,
  x_order = NULL,
  fill_order = NULL,
  x_label = NULL,
  y_label = NULL,
  plot_title = NULL,
  fill_label = NULL,
  custom_palette = NULL
)
```

Arguments

`model` A model object created using model_mf()

`plot_type` The type of plot to create. Options are "bar" or "point".

`x_effect` If there are multiple fixed effects in the model, specify the fixed effect to plot on the x-axis. The other will be used in the fill aesthetic. Currently, only 2 fixed effects are supported.

`plot_error_bars`
Logical. If TRUE, the estimated standard error will be added to the plot.

`plot_signif` Logical. If TRUE, will add significance labels based on the pairwise_comparisons data frame in the model object. This is only valid if you supplied a contrasts table to model_mf(). Symbols will be applied to plotted values that are significantly different from the reference. Your contrasts table is structured as a data frame with two columns, each containing levels of the fixed effects to be contrasted. When adding significance labels, symbols will be added to the values defined in the first column, while the second column will represent the reference. A different symbol will be used for each unique reference level. If a single plotted value has been contrasted against multiple references, then it will gain multiple symbols for each significance difference.

`ref_effect`	The fixed effect to use as the reference level when adding significance labels. Only applicable if using two fixed effects.
`x_order`	A character vector indicating the order of the levels for the x_effect.
`fill_order`	A character vector indicating the order of the levels for the fill aesthetic, if applicable.
`x_label`	The label for the x-axis.
`y_label`	The label for the y-axis.
`plot_title`	The title of the plot.
`fill_label`	The label for the fill aesthetic, if applicable.
`custom_palette`	A vector of colors to use for the fill and color aesthetics. If not provided, a default palette will be used. When plotting a model that has a single fixed effect, you can specify colors for "fill" and "color" using a named vector. Likewise, when plotting a model with two fixed effects, you can specify colors for the levels within your fill variable.

Details

See model_mf() for examples.

Value

A ggplot object.

`plot_radar` *Create a radar plot*

Description

Create a radar plot

Usage

```
plot_radar(mf_data, response_col, label_col, facet_col, indiv_y = TRUE)
```

Arguments

`mf_data`	A data frame with the data to plot
`response_col`	The column with the response values
`label_col`	The column with the labels for the radar plot.
`facet_col`	The column with the group to facet the radar plots.
`indiv_y`	A logical indicating whether to use individual y-axis scales for each plot.

Value

A radar plot

Examples

```
# Plot the mean MFmin of each genomic target per dose group
# Order the genomic targets by their genic context.
#Load the example data and calculate MF
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)
mf <- calculate_mf(mutation_data = example_data,
                   cols_to_group = c("sample", "label"),
                   retain_metadata_cols = c("dose_group", "genic_context"))
# Define the order of the genomic targets
label_order <- mf %>% dplyr::arrange(genic_context) %>%
  pull(label) %>%
  unique()
# Calculate the mean MF per dose_group for each target.
mean <- mf %>%
  dplyr::group_by(dose_group, label) %>%
  dplyr::summarise(mean = mean(mf_min))
# Set the order of each column
mean$dose_group <- factor(mean$dose_group,
                          levels = c("Control",
                                     "Low",
                                     "Medium",
                                     "High"))
mean$label <- factor(mean$label,
                     levels = label_order)
# Plot
plot <- plot_radar(mf_data = mean,
                   response_col = "mean",
                   label_col = "label",
                   facet_col = "dose_group",
                   indiv_y = FALSE)
```

---

plot_spectra *Transition-transversion plot*

Description

Given mf data, construct a plot displaying the mutation subtypes observed in a cohort.

Usage

```
plot_spectra(
  mf_data,
```

```
  group_col = "sample",
  subtype_resolution = "base_6",
  response = "proportion",
  mf_type = "min",
  group_order = "none",
  group_order_input = NULL,
  dist = "cosine",
  cluster_method = "ward.D",
  custom_palette = NULL,
  x_lab = NULL,
  y_lab = NULL
)
```

Arguments

`mf_data` A data frame containing the mutation frequency data at the desired subtype resolution. This is obtained using the 'calculate_mf' function with subtype_resolution set to the desired resolution. Data must include a column containing the group_col, a column containing the mutation subtypes, a column containing the desired response variable (mf, proportion, sum) for the desired mf_type (min or max), and if applicable, a column containing the variable by which to order the samples/groups.

`group_col` The name of the column(s) in the mf data that contains the sample/group names. This will generally be the same values used for the cols_to_group argument in the calculate_mf function. However, you may also use groups that are at a higher level of the aggregation in mf_data.

`subtype_resolution` The subtype resolution of the mf data. Options are `base_6`, `base_12`, `base_96`, `base_192`, or `type`. Default is `base_6`.

`response` The desired response variable to be plotted. Options are mf, proportion, or sum. Default is `proportion`. Your mf_data must contain columns with the name of your desired response: `mf_min`, `mf_max`, `proportion_min`, `proportion_max`, `sum_min`, and `sum_max`.

`mf_type` The mutation counting method to use. Options are min or max. Default is `min`.

`group_order` The method for ordering the samples within the plot. Options include:

- `none`: No ordering is performed. Default.
- `smart`: Groups are automatically ordered based on the group names (alphabetical, numerical)
- `arranged`: Groups are ordered based on one or more factor column(s) in mf_data. Column names are passed to the function using the `group_order_input`.
- `custom`: Groups are ordered based on a custom vector of group names. The custom vector is passed to the function using the `group_order_input`.
- `clustered`: Groups are ordered based on hierarchical clustering. The dissimilarity matrix can be specified using the `dist` argument. The agglomeration method can be specified using the `cluster_method` argument.

group_order_input
: A character vector specifying details for the group order method. If group_order is arranged, group_order_input should contain the column name(s) to be used for ordering the samples. If group_order is custom, group_order_input should contain the custom vector of group names.

dist
: The dissimilarity matrix for hierarchical clustering. Options are cosine, euclidean, maximum, manhattan, canberra, binary or minkowski. The default is cosine. See dist for details.

cluster_method
: The agglomeration method for hierarchical clustering. Options are ward.D, ward.D2, single, complete, average (= UPGMA), mcquitty (= WPGMA), median (= WPGMC) or centroid (= UPGMC). The default is Ward.D. See hclust for details.

custom_palette
: A named vector of colors to be used for the mutation subtypes. The names of the vector should correspond to the mutation subtypes in the data. Alternatively, you can specify a color palette from the RColorBrewer package. See brewer.pal for palette options. You may visualize the palettes at the ColorBrewer website: https://colorbrewer2.org/. Default is NULL.

x_lab
: The label for the x-axis. Default is the value of group_col.

y_lab
: The label for the y-axis. Default is the value of response_col.

Examples

```
# Load example data
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)

# Example 1: plot the proportion of 6-based mutation subtypes
# for each sample, organized by dose group:

# Calculate the mutation frequency data at the 6-base resolution.
# Retain the dose_group column to use for ordering the samples.
mf_data <- calculate_mf(mutation_data = example_data,
                        cols_to_group = "sample",
                        subtype_resolution = "base_6",
                        retain_metadata_cols = "dose_group")
# Set the desired order for the dose_group levels.
mf_data$dose_group <- factor(mf_data$dose_group,
                             levels = c("Control", "Low", "Medium", "High"))
# Plot the mutation spectra
plot <- plot_spectra(mf_data = mf_data,
                     group_col = "sample",
                     subtype_resolution = "base_6",
                     response = "proportion",
                     group_order = "arranged",
                     group_order_input = "dose_group")

# Example 2: plot the proportion of 6-based mutation subtypes
# for each sample, ordered by hierarchical clustering:
```

```
plot <- plot_spectra(mf_data = mf_data,
                     group_col = "sample",
                     subtype_resolution = "base_6",
                     response = "proportion",
                     group_order = "clustered")
```

`plot_trinucleotide` *Plot the trinucleotide spectrum*

Description

Creates barplots of the trinucleotide spectrum for all levels of a given group based on the mutation data. All plots are exported.

Usage

```
plot_trinucleotide(
  mf_96,
  response = "proportion",
  mf_type = "min",
  group_col = "dose",
  indiv_y = FALSE,
  sum_totals = TRUE,
  output_path = NULL,
  output_type = "tiff"
)
```

Arguments

`mf_96`	A data frame containing the mutation frequency data at the 96-base resolution. This should be obtained using the 'calculate_mf' with subtype_resolution set to 'base_96'. Generally, cols_to_group should be the same as 'group_col'.
`response`	A character string specifying the type of response to plot. Must be one of 'frequency', 'proportion', or 'sum'.
`mf_type`	A character string specifying the mutation count method to plot. Must be one of 'min' or 'max'. Default is 'min'.
`group_col`	A character string specifying the column(s) in 'mf_96' to group the data by. Default is 'sample'. The sum, proportion, or frequency will be plotted for all unique levels of this group. You can specify more than one column to group by. Generally the same as the 'cols_to_group' parameter in 'calculate_mf' when generating mf_96_data.
`indiv_y`	A logical value specifying whether the the max response value for the y-axis should be scaled independently for each group (TRUE) or scaled the same for all groups (FALSE). Default is FALSE.
`sum_totals`	A logical value specifying whether to sum the total mutations.

output_path
: A character string specifying the path to save the output plot. Default is NULL. This will create an output directory in the current working directory.

output_type
: A character string specifying the type of output file. Options are 'jpeg', 'pdf', 'png', 'svg', or 'tiff'. Default is 'svg'.

Details

The function plots the trinucleotide spectrum for all levels of a given group from the provided mf_96 data; the output of calculate_mf with subtype_resolution = "base_96".

Examples

```
# Load example data
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)

# Use a temporary directory to save the example plots.
temp_output <- tempdir()

# Calculate the mutation frequency data at the 96-base resolution
mf_96 <- calculate_mf(mutation_data = example_data,
                      cols_to_group = "dose_group",
                      subtype_resolution = "base_96",
                      variant_types = "snv")
# Plot the trinucleotide proportions for each dose group
# Scale y-axis the same for all groups
plot_trinucleotide(mf_96 = mf_96,
                   response = "proportion",
                   mf_type = "min",
                   group_col = "dose_group",
                   indiv_y = FALSE,
                   output_path = temp_output)
list.files(temp_output)
# Note: The plots are saved as image files in the temporary directory.
# To view the plots, use the following code:
## if (!requireNamespace("tiff", quietly = TRUE)) install.packages("tiff")
## library(tiff)
## example_plot <- file.path(temp_output, "plot_Control_trinucleotide_proportion.tiff")
## image <- tiff::readTIFF(example_plot)
## plot(as.raster(image))
```

---

plot_trinucleotide_heatmap

Create a heatmap plot of mutation subtype proportions.

Description

This function creates a heatmap plot of subtype proportions for a given grouping variable. The groups may be facetted by a second variable. Mutation sums for each facet group and normalized subtype are calculated and displayed.

Usage

```
plot_trinucleotide_heatmap(
  mf_data,
  group_col = "sample",
  facet_col = "dose",
  mf_type = "min",
  mut_proportion_scale = "turbo",
  max = 0.2,
  rescale_data = FALSE,
  condensed = FALSE
)
```

Arguments

`mf_data` A data frame containing the mutation frequency data at the desired base resolution. This is obtained using the 'calculate_mf' with subtype_resolution set to the desired resolution. cols_to_group should be the same as 'group_col'.

`group_col` The variable to group by.

`facet_col` The variable to facet by.

`mf_type` The type of mutation frequency to plot. Options are "min" or "max". (Default: "min")

`mut_proportion_scale` The scale option for the mutation proportion. Options are passed to viridis::scale_fill_viridis_c. One of # inferno, magma, plasma, viridis, cividis, turbo, mako, or rocket. We highly reccomend the default for its ability to disciminate hard to see patterns. (Default: "turbo")

`max` Maximum value used for plotting the proportions. Proportions that are higher will have the maximum colour. (Default: 0.2)

`rescale_data` Logical value indicating whether to rescale the mutation proportions to increase the dynamic range of colors shown on the plot. (Default: TRUE)

`condensed` More condensed plotting format. Default = FALSE.

Value

A ggplot object representing the heatmap plot.

Examples

```
# Plot the trinucleotide proportions per sample, facetted by dose group.
example_file <- system.file("extdata",
                            "example_mutation_data_filtered.rds",
                            package = "MutSeqR")
example_data <- readRDS(example_file)
# define dose_group order
example_data$dose_group <- factor(example_data$dose_group,
                                  levels = c("Control", "Low",
                                             "Medium", "High"))
```

```
mf_96 <- calculate_mf(example_data,
                      cols_to_group = "sample",
                      variant_types = "snv",
                      subtype_resolution = "base_96",
                      retain_metadata_cols = "dose_group")
plot <- plot_trinucleotide_heatmap(mf_96,
                                   group_col = "sample",
                                   facet_col = "dose_group")
```

---

print_ascii_art — *This function prints ASCII art when the package is loaded*

Description

This function prints ASCII art when the package is loaded

Usage

```
print_ascii_art()
```

---

rename_columns — *Map column names of mutation data to default column names. A utility function that renames columns of mutation data to default columns names.*

Description

Map column names of mutation data to default column names. A utility function that renames columns of mutation data to default columns names.

Usage

```
rename_columns(data, column_map = op$column)
```

Arguments

`data`	mutation data
`column_map`	a list that maps synonymous column names to their default.

Value

the mutation data with column names changed to match default.

render_report — *Read configuration file and render R Markdown document*

Description

This function reads a configuration file in YAML format, extracts the parameters, and renders an R Markdown document using the specified parameters.

Usage

```
render_report(config_filepath, output_file)
```

Arguments

`config_filepath`	The path to the configuration file.
`output_file`	The path to the output file.

Value

None

reverseComplement — *Get the reverse complement of a DNA or RNA sequence.*

Description

Get the reverse complement of a DNA or RNA sequence.

Usage

```
reverseComplement(
  x,
  content = c("dna", "rna"),
  case = c("lower", "upper", "as is")
)
```

Arguments

`x`	A character vector of DNA or RNA sequences.
`content`	c("dna", "rna") The type of sequence to be reversed.
`case`	c("lower", "upper", "as is") The case of the output sequence.

Details

This file is part of the source code for SPGS: an R package for identifying statistical patterns in genomic sequences. Copyright (C) 2015 Universidad de Chile and INRIA-Chile A copy of Version 2 of the GNU Public License is available in the share/licenses/gpl-2 file in the R installation directory or from http://www.R-project.org/Licenses/GPL-2. reverseComplement.R

---

sidak *Correct p-values for multiple comparisons*

Description

Correct p-values for multiple comparisons

Usage

```
sidak(vecP)
```

Arguments

vecP	vector of p-values

Details

This function corrects a vector of probabilities for multiple testing using the Bonferroni (1935) and Sidak (1967) corrections. References: Bonferroni (1935), Sidak (1967), Wright (1992). Bonferroni, C. E. 1935. Il calcolo delle assicurazioni su gruppi di teste. Pp. 13-60 in: Studi in onore del Professore Salvatore Ortu Carboni. Roma. Sidak, Z. 1967. Rectangular confidence regions for the means of multivariate normal distributions. Journal of the American Statistical Association 62:626-633. Wright, S. P. 1992. Adjusted P-values for simultaneous inference. Biometrics 48: 1005-1013. Pierre Legendre, May 2007

Value

adjusted p-values

---

signature_fitting *Run COSMIC signatures comparison using SigProfilerAssignment*

Description

Run COSMIC signatures comparison using SigProfilerAssignment

Usage

```
signature_fitting(
  mutation_data,
  project_name = "Default",
  project_genome = "GRCh38",
  env_name = "MutSeqR",
  group = "sample",
  output_path = NULL,
  python_version
)
```

Arguments

`mutation_data`	A data frame containing mutation data.
`project_name`	The name of the project. This is used to format the data into required .txt format for SigProfiler tools.
`project_genome`	The reference genome to use. On first use, the function will install the genome using SigProfilerMatrixGeneratorR::install. e.x. GRCh37, GRCH38, mm10, mm9, rn6
`env_name`	The name of the virtual environment. This will be created on first use.
`group`	The column in the mutation data used to aggregate groups. Signature assignment will be performed on each group separately.
`output_path`	The filepath to the directory in which the output folder will be created to store results. Default is NULL. This will store results in the current working directory.
`python_version`	The version of python installed on the user's computer.

Details

Assign COSMIC SBS signatures to mutation data using SigProfilerAssignment. Data is cleaned and formatted for input into SigProfiler tools. This function will create a virtual environment using reticulate to run python, as this is a requirement for the SigProfiler suite of tools. Note that it will also install several python dependencies using a conda virtual environment on first use. Please be aware of the implications of this. For advanced use, it is suggested to use the SigProfiler python tools directly in python as described in their respective documentation. Users must have python installed on their computer to use this function.

Mutation data will be filtered to only include SNVs. Variants flagged by the filter_mut column will be excluded.

Value

Creates a subfolder "SigProfiler" in the output directory with SigProfiler tools results. For a complete breakdown of the results, see the Readme file for MutSeqR. Most relevant results are stored in SigProfiler > group > matrices > output > Assignment_Solution > Activities > SampleReconstruction > WebPNGs. These plots show a summary of the signature assignment results for each group. In each plot, the top left panel represents the base_96 mutation count for the group. The bottom left panel represents the reconstructed profile. Below the reconstruction are the solution statistics that indicate the goodness of fit of the reconstructed profile to the observed profile. (Recommended cosine similarity > 0.9). The panels on the right represent the SBS signatures that contribute to the reconstructed profile. The signature name and its contribution % are shown in the panel. A high contribution means a high association of the signature with the group's mutation spectra.

Examples

```
## Not run:
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
signature_fitting(mutation_data = example_data,
                  project_name = "Example",
                  project_genome = "mm10",
                  env_name = "MutSeqR",
                  group = "dose",
                  python_version = "3.11")

## End(Not run)
```

`spectra_comparison` *Compare the overall mutation spectra between groups*

Description

spectra_comparison compares the mutation spectra of groups using a modified contingency table approach.

Usage

```
spectra_comparison(
  mf_data,
  cols_to_group,
  mf_type = "min",
  contrasts,
  cont_sep = "\t"
)
```

Arguments

`mf_data`	A data frame containing the MF data. This is the output from calculate_mf(). MF data should be at the desired subtype resolution. Data should be calculated for the same columns as the `cols_to_group` argument. Required columns are the cols_to_group columns, the subtype column, and sum_min or sum_max.
`cols_to_group`	The column names of the variables across which the mutations were summed. by. Ex. c("dose", "tissue"). This function will sum the mutations across groups before running the comparison.
`mf_type`	The type of mutation frequency to use. Default is "min".

`contrasts` a filepath to a tab-delimited `.txt` file OR a dataframe that will specify the comparisons to be made between groups. The table must consist of two columns. The first column will be a level within your group column and the second column must be the group level that it will be compared to. All values must correspond to entries in your `cols_to_group` column. For more than one `group` variable, separate the levels of each group with a colon. Ensure that all `groups` listed in cols_to_group are represented in each entry for the table. See details for examples.

`cont_sep` The delimiter used to import the contrasts table. Default is tab.

Details

Examples of `contrasts`: If you have `group = "dose"` with dose groups 0, 25, 50, 100. The first column would contain the treated groups (25, 50, 100), while the second column would be 0, thus comparing each treated group to the control group.

25 0

50 0

100 0

Ex. Consider two grouping variables `group = c("dose", "tissue")`; with levels dose (0, 25, 50, 100) and tissue("bone_marrow", "liver"). To compare the mutation spectra between tissues for each dose group, the contrast table would look like:

0:bone_marrow 0:liver

25:bone_marrow 25:liver

50:bone_marrow 50:liver

100:bone_marrow 100:liver

Value

the log-likelihood statistic G2 for the specified comparisons with the p-value adjusted for multiple-comparisons.

Examples

```
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)

# Example: compare 6-base mutation spectra between dose groups

# Calculate the mutation frequency data at the 6-base resolution
mf_data <- calculate_mf(mutation_data = example_data,
                        cols_to_group = "dose_group",
                         subtype_resolution = "base_6")
# Create the contrasts table
contrasts <- data.frame(col1 = c("Low", "Medium", "High"),
                        col2 = rep("Control", 3))
# Run the comparison
spectra_comparison(mf_data = mf_data,
                   cols_to_group = "dose_group",
```

```
mf_type = "min",
contrasts = contrasts)
```

`subtype_dict` *Values accepted for mutation subtypes*

Description

These values are used to enable user input to translate to columns in a mut file

Usage

```
subtype_dict
```

Format

A vector with corresponding values

`subtype_list` *A list of mutation subtypes at different resolutions*

Description

A list of mutation subtypes at different resolutions

Usage

```
subtype_list
```

Format

A list with corresponding values

write_excel — *Write Excel tables*

Description

Writes data to an Excel file.

Usage

```
write_excel(data, output_path = NULL, workbook_name, model_results = FALSE)
```

Arguments

data	A data frame or a list of data frames. If a data frame, it will be written to a single sheet in the Excel workbook. If a list, each data frame will be written to a separate sheet in the Excel workbook. Data may also be the output to model_mf, in which case set `model_results = TRUE`.
output_path	The directory where the Excel file should be written. Default is NULL, which will write the file to the current working directory.
workbook_name	The file name for the Excel file.
model_results	A logical value indicating whether the data is the output of model_mf. Default is FALSE. If TRUE, the function will grab the model_data, point_estimates, and pairwise_comparisons data frames from the model_mf output and write them to separate sheets in the Excel workbook.

Value

A saved Excel workbook.

Examples

```
## Not run:
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
mf1 <- calculate_mf(example_data,
                    cols_to_group = "sample",
                    subtype_resolution = "none",
                    retain_metadata_cols = "dose")
mf2 <- calculate_mf(example_data,
                    cols_to_group = c("sample", "label"),
                    subtype_resolution = "none")
mf3 <- calculate_mf(example_data,
                    cols_to_group = "dose",
                    subtype_resolution = "base_6",
                    variant_types = c("-ambiguous", "-uncategorized"))
list <- list(mf1, mf2, mf3)
names(list) <- c("mf1", "mf2", "mf3")
```

```
# save a single data frame to an Excel file
write_excel(mf1, output_path, workbook_name = "test_single")
#save a list of data frames to an Excel file
write_excel(list, output_path, workbook_name = "test_list")

# save model results to an Excel file
model  <- model_mf(mf1,
                   fixed_effects = "dose",
                   reference_level = 0,
                   contrasts = data.frame(col1 = c(12.5, 25, 50),
                                          col2 = rep(0,3)))
write_excel(model,
            workbook_name = "test_model",
            model_results = TRUE)

## End(Not run)
```

`write_mutational_matrix` *Write a Mutational Matrix to input into the sigprofiler web application*

Description

Creates a .txt file from mutation data that can be used for mutational signatures analysis using the SigProfiler web application. Can handle group analyses (ex dose, tissue, etc). Currently only supports SBS matrices i.e. snvs.

Usage

```
write_mutational_matrix(
  mutation_data,
  group = "dose",
  subtype_resolution = "base_96",
  mf_type = "min",
  output_path = NULL
)
```

Arguments

`mutation_data`	The object containing the mutation data. The output of import_mut_data() or import_vcf_data().
`group`	The column in the mutation data used to aggregate groups (e.g., sample, tissue, dose).
`subtype_resolution`	The resolution of the mutation subtypes. Options are "base_6" or "base_96". Default is "base_96".
`mf_type`	The mutation counting method to use. Options are "min" or "max". Default is "min".

output_path — The path to save the output file. If not provided, the file will be saved in the current working directory. Default is NULL.

Details

Mutations will be be filtered for SNVs. Mutations flagged in `filter_mut` will be excluded from the output. Mutations will be summed across the groups specified in the `group` argument.

Value

a .txt file that can be uploaded to the SigProfiler Assignment web application (https://cancer.sanger.ac.uk/signatures/assignmer as a "Mutational Matrix".

Examples

```
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
temp_output <- tempdir()
write_mutational_matrix(mutation_data = example_data,
                        group = "dose_group",
                        subtype_resolution = "base_96",
                        mf_type = "min",
                        output_path = temp_output)
list.files(temp_output)
# The file is saved in the temporary directory
# To view the file, use the following code:
## output_file <- file.path(temp_output, "dose_group_base_96_mutational_matrix.txt")
## file.show(output_file)
```

---

write_mutation_calling_file

Write the mutation calling file to input into the SigProfiler Assignment web application.

Description

Creates a .txt file from mutation data that can be used for mutational signatures analysis using the SigProfiler Assignment web application. Currently only supports SBS analysis i.e. snvs.

Usage

```
write_mutation_calling_file(
  mutation_data,
  project_name = "Example",
  project_genome = "GRCm38",
  output_path = NULL
)
```

Arguments

`mutation_data`	The object containing the mutation data. The output of import_mut_data() or import_vcf_data().
`project_name`	The name of the project. Default is "Example".
`project_genome`	The reference genome to use. (e.g., Human: GRCh38, Mouse mm10: GRCm38)
`output_path`	The path to save the output file. If NULL, files will be saved in the current working directory. Default is NULL.

Details

Mutations will be be filtered for SNVs. Mutations flagged in `filter_mut` will be excluded from the output.

Value

a .txt file that can be uploaded to the SigProfiler Assignment web application (https://cancer.sanger.ac.uk/signatures/assignmer as a "Mutational calling file".

Examples

```
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
temp_output <- tempdir()
write_mutation_calling_file(mutation_data = example_data,
                            project_name = "Example",
                            project_genome = "GRCm38",
                            output_path = temp_output)
list.files(temp_output)
# The file is saved in the temporary directory
# To view the file, use the following code:
## output_file <- file.path(temp_output, "mutation_calling_file.txt")
## file.show(output_file
```

`write_reference_fasta` *Write FASTA file of reference sequences.*

Description

Write FASTA file of reference sequences.

Usage

```
write_reference_fasta(regions_gr, output_path = NULL)
```

Arguments

regions_gr
: A GRanges object including the sequences of the reference regions included for the data. This can be generated from the `get_seq` function.

output_path
: The directory where the FASTA file should be written. Default is NULL, which will write the file to the current working directory.

Details

Generate an arbitrary multi-sequence FASTA file from GRanges including the reference sequences.

Value

Writes a FASTA reference file "reference_output.fasta". If multiple ranges are included in the GRanges object, the sequences will be written to a single FASTA file. Sequences names will be the seqnames (contig) of the range.

Examples

```
## Not run:
# Write FASTA files for the 20 genomic target sequences
# of TwinStrand's Mouse Mutagenesis Panel.
rg <- get_seq("TSpanel_mouse")
write_reference_fasta(rg, output_path = NULL)

## End(Not run)
```

`write_vcf_from_mut` *Write mutation_data to a VCF file*

Description

Export your mutation_data to a VCF file for downstream applications.

Usage

```
write_vcf_from_mut(mutation_data, output_path = NULL)
```

Arguments

mutation_data
: A data frame of a GRanges object containing your mutation data. This can be the output of `import_mut_data`, `import_vcf_data`, or `filter_mut`. Coordinates must be 1-based. Required columns are "contig", "start", "end", "ref", "alt", "sample", "alt_depth", "total_depth", and "ref_depth". Additional columns are allowed.

output_path
: The directory where the VCF file should be written. Default is NULL, which will write the file to the current working directory.

Value

Writes a VCF file of mutations "mutation_output.vcf".

Examples

```
## Not run:
example_file <- system.file("extdata", "example_mutation_data_filtered.rds", package = "MutSeqR")
example_data <- readRDS(example_file)
write_vcf_from_mut(example_data)

## End(Not run)
```

Index