Package ‘PepMapViz’

January 20, 2025

Title A Versatile Toolkit for Peptide Mapping, Visualization, and
Comparative Exploration

Version 1.0.0

Description A versatile R visualization package that empowers
researchers with comprehensive visualization tools for seamlessly mapping peptides
to protein sequences, identifying distinct domains and regions of interest,
accentuating mutations, and highlighting post-translational modifications,
all while enabling comparisons across diverse experimental conditions.
Potential applications of 'PepMapViz' include the visualization of cross-software
mass spectrometry results at the peptide level for specific protein and domain
details in a linearized format and post-translational modification coverage
across different experimental conditions; unraveling insights into disease
mechanisms. It also enables visualization of major histocompatibility complex-
presented peptides in
different antibody regions predicting immunogenicity in antibody drug development.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.1

Imports ggplot2, stringr, ggforce, ggh4x, ggnewscale, data.table,
rlang

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

biocViews Immunogenicity, MassSpectrometry, Proteomics, Peptidomics,
Software, Visualization

NeedsCompilation no

Maintainer Zhenru Zhou `<zhou.zhenru@gene.com>`

Note The following words are correctly spelled domain-specific terms:
MHC, 'PepMapViz', immunogenicity, linearized, spectrometry,
translational. The package includes large datasets in the
'extdata' directory, which are essential for demonstrating the
functionality and performance of the tools provided. These

datasets are necessary for reproducibility and comprehensive
testing.

Author Zhenru Zhou [aut, cre],
Qui Phung [ctb],
Corey Bakalarski [aut],
Genentech, Inc. [cph]

Repository CRAN

Date/Publication 2024-11-13 14:00:13 UTC

Contents

`calculate_all_Area` *Calculate Area/Intensity for the whole input sequence dataframe*

Description

Calculate Area/Intensity for the whole input sequence dataframe

Usage

```
calculate_all_Area(
  whole_seq,
  matching_result,
  matching_columns,
  distinct_columns,
  area_column,
  with_PTM = FALSE,
  reps = FALSE
)
```

Arguments

`whole_seq`	A dataframe holding whole sequence information. 'Region_Sequence' column is required for the sequence information. Change the column name if it is different than 'Region_Sequence'.
`matching_result`	The dataframe that contains the matched results and PTM information.
`matching_columns`	Vector of column names that should match between each row of 'whole_seq' and the 'matching_result' dataframe.
`distinct_columns`	Vector of column names that should be used to calculate Area separately for each unique combination of these columns.
`area_column`	The name of the column in 'matching_result' that contains the area/intensity information.
`with_PTM`	A boolean parameter indicating whether PTM should be considered during calculation of Area. Default is `FALSE`.
`reps`	A boolean parameter indicating whether the area/intensity should be divided by the number of replicates. Default is `FALSE`.

Value

Returns `data_with_area`, a dataframe contains calculated Area for each record in 'whole_seq'.

Examples

```
whole_seq <- data.frame(
  Region_Sequence = c(
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD",
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD"
  ),
  Condition_1 = c(
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2"
  ),
  Condition_2 = c(
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2"
  ),
  Region_1 = c(
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL"
  ),
  Region_2 = c(
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2"
  )
)
matching_result <- data.frame(
  Sequence = c("AAA", "DDD", "DDD"),
  Condition_1 = c("Drug1", "Drug2", "Drug2"),
  Condition_2 = c("Donor1", "Donor2", "Donor2"),
  Region_1 = c("VH", "VL", "VL"),
  Region_2 = c("Arm_1", "Arm_2", "Arm_2"),
  Start_Position = c(4, 4, 4),
  End_Position = c(6, 6, 6),
  PTM_position = c(NA, 2, 0),
  PTM_type = c(NA, "O", "C"),
  Area = c(100, 200, 200),
  reps = c(1, 2, 2)
)
matching_columns <- c("Condition_1", "Region_2")
area_column <- "Area"
data_with_area <- calculate_all_Area(
  whole_seq,
  matching_result,
  matching_columns,
  distinct_columns = c("Condition_2", "Region_1"),
  area_column,
  with_PTM = TRUE,
  reps = TRUE
)
```

calculate_all_PSM *Calculate Spectra Count (PSM) for the whole input sequence dataframe*

Description

Calculate Spectra Count (PSM) for the whole input sequence dataframe

Usage

```
calculate_all_PSM(
  whole_seq,
  matching_result,
  matching_columns,
  distinct_columns,
  with_PTM = FALSE,
  reps = FALSE
)
```

Arguments

`whole_seq`	A dataframe holding whole sequence information. 'Region_Sequence' column is required for the sequence information. Change the column name if it is different than 'Region_Sequence'.
`matching_result`	The dataframe that contains the matched results and PTM information.
`matching_columns`	Vector of column names that should match between each row of 'whole_seq' and the 'matching_result' dataframe.
`distinct_columns`	Vector of column names that should be used to calculate PSM separately for each unique combination of these columns.
`with_PTM`	A boolean parameter indicating whether PTM should be considered during calculation of PSM. Default is `FALSE`.
`reps`	A boolean parameter indicating whether the area/intensity should be divided by the number of replicates. Default is `FALSE`.

Value

Returns `data_with_psm`, a dataframe contains calculated PSM for each record in 'whole_seq'.

Examples

```
whole_seq <- data.frame(
  Region_Sequence = c(
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD",
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD"
  ),
  Condition_1 = c(
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2"
  ),
  Condition_2 = c(
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2"
  ),
  Region_1 = c(
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL"
  ),
  Region_2 = c(
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2"
  )
)
matching_result <- data.frame(
  Sequence = c("AAA", "DDD", "DDD"),
  Condition_1 = c("Drug1", "Drug2", "Drug2"),
  Condition_2 = c("Donor1", "Donor2", "Donor2"),
  Region_1 = c("VH", "VL", "VL"),
  Region_2 = c("Arm_1", "Arm_2", "Arm_2"),
  Start_Position = c(4, 4, 4),
  End_Position = c(6, 6, 6),
  PTM_position = c(NA, 2, 0),
  PTM_type = c(NA, "O", "C"),
  Area = c(100, 200, 200),
  reps = c(1, 2, 2)
)
matching_columns <- c("Condition_1", "Region_2")
data_with_psm <- calculate_all_PSM(
  whole_seq,
  matching_result,
  matching_columns,
  distinct_columns = c("Condition_2", "Region_1"),
  with_PTM = TRUE,
  reps = TRUE
)
```

---

calculate_Area *Calculate Area/Intensity for one row of the input sequence dataframe*

Description

Calculate Area/Intensity for one row of the input sequence dataframe

Usage

```
calculate_Area(
  row,
  matching_result,
  matching_columns,
  distinct_columns = NULL,
  area_column,
  with_PTM = FALSE,
  reps = FALSE
)
```

Arguments

`row`	A row of dataframe containing the sequence for the 'Character' column in region_data.
`matching_result`	The dataframe that contains the matched results and PTM information.
`matching_columns`	Vector of column names that should match between the 'row' and 'matching_result' dataframes.
`distinct_columns`	Vector of column names that should be used to calculate Area separately for each unique combination of these columns.
`area_column`	The name of the column in 'matching_result' that contains the area/intensity information.
`with_PTM`	A boolean parameter indicating whether PTM should be considered. If `with_PTM = TRUE`, the function will also add 'PTM' and 'PTM_type' to the result 'region_data' dataframe. Default is `FALSE`.
`reps`	A boolean parameter indicating whether the area/intensity should be divided by the number of replicates. Default is `FALSE`.

Value

This function returns the modified `region_data` dataframe that includes the "Area" column, and optionally "PTM" and "PTM_type" columns. If the 'filter_conditions' do not match, an empty dataframe will be returned early. An AttributeError is raised if 'PTM_position' and 'PTM_type' columns do not exist in the 'result' dataframe when 'with_PTM' is `TRUE`.

Examples

```
row <- data.frame(
 Region_Sequence = c("XYZAAA"),
 Condition_1 = c("Drug1"),
 Condition_2 = c("Donor1"),
 Region_1 = c("VH"),
 Region_2 = c("Arm_1")
)
matching_result <- data.frame(
  Sequence = c("AAA", "DDD", "DDD"),
  Condition_1 = c("Drug1", "Drug2", "Drug2"),
  Condition_2 = c("Donor1", "Donor2", "Donor2"),
  Region_1 = c("VH", "VL", "VL"),
  Region_2 = c("Arm_1", "Arm_2", "Arm_2"),
  Start_Position = c(4, 4, 4),
  End_Position = c(6, 6, 6),
  PTM_position = c(NA, 2, 0),
  PTM_type = c(NA,"O","C"),
  Area = c(100, 200, 200),
  reps = c(1, 2, 2)
)
matching_columns <- c("Condition_1", "Region_2")
area_column <- "Area"
data_with_area <- calculate_Area(
  row,
  matching_result,
  matching_columns,
  distinct_columns = c("Condition_2", "Region_1"),
  area_column,
  with_PTM = TRUE,
  reps = TRUE
)
```

`calculate_PSM` *Calculate Spectra Count (PSM) for one row of the input sequence dataframe*

Description

Calculate Spectra Count (PSM) for one row of the input sequence dataframe

Usage

```
calculate_PSM(
  row,
  matching_result,
  matching_columns,
  distinct_columns,
  with_PTM = FALSE,
  reps = FALSE
)
```

Arguments

`row`	A row of dataframe containing the sequence for the 'Character' column in region_data.
`matching_result`	The dataframe that contains the matched results and PTM information.
`matching_columns`	Vector of column names that should match between the 'row' and 'matching_result' dataframes.
`distinct_columns`	Vector of column names that should be used to calculate PSM separately for each unique combination of these columns.
`with_PTM`	A boolean parameter indicating whether PTM should be considered. If `with_PTM = TRUE`, the function will also add 'PTM' and 'PTM_type' to the result 'region_data' dataframe. Default is `FALSE`.
`reps`	A boolean parameter indicating whether the area/intensity should be divided by the number of replicates. Default is `FALSE`.

Value

This function returns the modified `region_data` dataframe that includes the "PSM" column, and optionally "PTM" and "PTM_type" columns. If the 'filter_conditions' do not match, an empty dataframe will be returned early. An AttributeError is raised if 'PTM_position' and 'PTM_type' columns do not exist in the 'result' dataframe when 'with_PTM' is `TRUE`.

Examples

```
row <- data.frame(
 Region_Sequence = c("XYZDDD"),
 Condition_1 = c("Drug2"),
 Region_1 = c("VL"),
 Region_2 = c("Arm_2")
)
matching_result <- data.frame(
  Sequence = c("AAA", "DDD", "DDD"),
  Condition_1 = c("Drug1", "Drug2", "Drug2"),
  Condition_2 = c("Donor1", "Donor2", "Donor2"),
  Region_1 = c("VH", "VL", "VL"),
  Region_2 = c("Arm_1", "Arm_2", "Arm_2"),
  Start_Position = c(4, 4, 4),
  End_Position = c(6, 6, 6),
  PTM_position = c(NA, 2, 0),
  PTM_type = c(NA,"O","C"),
  Area = c(100, 200, 200),
  reps = c(1, 2, 2)
)
matching_columns <- c("Condition_1", "Region_2")
result <- calculate_PSM(
  row,
  matching_result,
  matching_columns,
  distinct_columns = c("Condition_2", "Region_1"),
  with_PTM = TRUE,
  reps = TRUE
)
```

`combine_files_from_folder`
Combine CSV and TXT Files from a Folder

Description

This function reads all CSV and TXT files from a specified folder and combines them into a single data.table.

Usage

```
combine_files_from_folder(folder_path)
```

Arguments

`folder_path`	The path to the folder containing the CSV or TSV files.

Value

A data.table containing the combined data from all files.

Examples

```
folder_path <- ""
combined_df <- combine_files_from_folder(folder_path)
print(combined_df)
```

`convert_to_regex_pattern` *Convert Peptide Sequence to Regex Pattern*

Description

This function converts a peptide sequence into a regular expression pattern that accounts for ambiguous amino acids. Each amino acid is replaced by a character class that includes itself, 'X', and any specific ambiguities.

Usage

```
convert_to_regex_pattern(peptide)
```

Arguments

`peptide`	A character string representing the peptide sequence.

Value

A character string containing the regex pattern for matching.

Examples

```
# Convert a peptide sequence to a regex pattern
peptide <- "NDEQIL"
regex_pattern <- convert_to_regex_pattern(peptide)
print(regex_pattern) # Output: "[NBX][DBX][EZX][QZX][ILX][ILX]"
```

`create_peptide_plot` *Create a peptide Plot*

Description

This function generates a peptide plot using the provided data and allows for customization of the plot layout.

Usage

```
create_peptide_plot(
  data,
  y_axis_vars,
  x_axis_vars,
  y_expand = c(0.1, 0.15),
  x_expand = c(0.6, 0.6),
  theme_options = NULL,
  labs_options = NULL,
  color_fill_column,
  fill_gradient_options = list(),
  label_size = 3,
  add_domain = TRUE,
  domain = NULL,
  domain_start_column = "domain_start",
  domain_end_column = "domain_end",
  domain_type_column = "domain_type",
  domain_color = NULL,
  PTM = FALSE,
  PTM_type_column = "PTM_type",
  PTM_color = NULL,
  add_label = TRUE,
  label_column = "Character",
  label_value = NULL,
  column_order = NULL
)
```

Arguments

`data`	A dataframe containing the PSM data or Area data got from peptide_cluster_quantification.
`y_axis_vars`	A list of variables for the donor and type facets.
`x_axis_vars`	A list of variables for the region facets.
`y_expand`	A numeric vector of length 2 specifying the expansion for the y-axis. Default is `c(0.1, 0.15)`.
`x_expand`	A numeric vector of length 2 specifying the expansion for the x-axis. Default is `c(0.6, 0.6)`.
`theme_options`	A list of additional theme options to customize the plot. Default is an empty list.
`labs_options`	A list of additional labs options to customize the plot labels. Default is an empty list.
`color_fill_column`	The name of the column in `data_with_psm` to be used for the fill aesthetic. Default is 'PSM'.
`fill_gradient_options`	A list of options for `scale_fill_gradient`. Default is an empty list.
`label_size`	The size of the labels in the plot. Default is 3.
`add_domain`	A logical value indicating whether to add domain like CDR (Complementarity-Determining Region) to the plot. Default is TRUE.
`domain`	A dataframe containing the domain data with columns including 'domain_start', 'domain_end', and 'domain_type'.
`domain_start_column`	The name of the column in `domain` containing the start position of the domain Default is 'domain_start'.

`domain_end_column`	The name of the column in `domain` containing the end position of the domain Default is 'domain_end'.
`domain_type_column`	The name of the column in `domain` containing the type of the domain Default is 'domain_type'.
`domain_color`	A list of colors for the domain types. Default is NULL.
`PTM`	A logical value indicating whether to include PTM (Post-Translational Modification) data in the plot. Default is FALSE.
`PTM_type_column`	The name of the column in `data_with_psm` containing the type of the PTM. Default is 'PTM_type'.
`PTM_color`	A list of colors for the PTM types. Default is NULL.
`add_label`	A logical value indicating whether to add labels to the plot. Default is TRUE.
`label_column`	The name of the column in `data_with_psm` containing the labels to be added to the plot. Default is 'Character'.
`label_value`	A list of column names and their values to filter the data for the labels. Default is NULL.
`column_order`	A list of column names and their order for the plot. Default is NULL.

Value

This function returns a ggplot object representing the PSM plot.

Examples

```
data <- data.frame(
  Character = c("X", "Y", "Z", "A", "A", "A"),
  Position = 1:6,
  Condition_1 = rep("Drug1", 6),
  Region_2 = rep("Arm_1", 6),
  Area = c(0.000000, 0.000000, 0.000000, 6.643856, 6.643856, 6.643856),
  Condition_2 = rep("Donor1", 6),
  Region_1 = rep("VH", 6),
  PTM = c(FALSE, TRUE, FALSE, FALSE, FALSE, FALSE),
  PTM_type = c(NA, "O", NA, NA, NA, NA)
)
domain <- data.frame(
  domain_type = c("CDR H1", "CDR H2", "CDR H3"),
  Region_1 = c("VH", "VH", "VH"),
  Region_2 = c("Arm_1", "Arm_1", "Arm_1"),
  Condition_1 = c("Drug1", "Drug1", "Drug1"),
  domain_start = c(1, 3, 5),
  domain_end = c(2, 4, 6)
)
x_axis_vars <- c("Region_2", "Region_1")
y_axis_vars <- c("Condition_2")
domain_color <- c(
"CDR H1" = "#F8766D",
"CDR H2" = "#B79F00",
"CDR H3" = "#00BA38",
"CDR L1" = "#00BFC4",
"CDR L2" = "#619CFF",
"CDR L3" = "#F564E3"
)
PTM_color <- c(
  "Ox" = "red",
  "Deamid" = "cyan",
  "Cam" = "blue",
  "Acetyl" = "magenta"
)
p <- create_peptide_plot(
  data,
  y_axis_vars,
  x_axis_vars,
  y_expand = c(0.2, 0.2),
  x_expand = c(0.5, 0.5),
  theme_options = list(),
  labs_options = list(title = "PSM Plot", x = "Position", fill = "PSM"),
  color_fill_column = 'Area',
  fill_gradient_options = list(),
  label_size = 5,
  add_domain = TRUE,
  domain = domain,
  domain_start_column = "domain_start",
  domain_end_column = "domain_end",
  domain_type_column = "domain_type",
  domain_color = domain_color,
  PTM = FALSE,
  PTM_type_column = "PTM_type",
  PTM_color = PTM_color,
  add_label = TRUE,
  label_column = "Character",
  label_value = NULL,
  column_order = list(Region_1 = 'VH,VL')
)
print(p)
```

match_and_calculate_positions

Match peptide sequence with provided sequence and calculate positions

Description

This function matches peptide sequences from the 'peptide_data' data frame to corresponding provided sequences in the 'whole_seq' data frame. It calculates the start and end positions of the matched sequences and returns a data frame with information about the matching positions.

Usage

```
match_and_calculate_positions(
  peptide_data,
  column,
  whole_seq,
  match_columns,
  sequence_length = NULL,
  column_keep = NULL
)
```

Arguments

`peptide_data`	A data frame containing peptide sequence information to match.
`column`	The name of the column in peptide_data containing the peptide sequences to be matched.
`whole_seq`	A data frame containing details about antibody sequence information including the domain and region information. 'Region_Sequence' column is required for the sequence information. Change the column name if it is different than 'Region_Sequence'.
`match_columns`	A character vector of column names to match on while matching peptide sequence.
`sequence_length`	(Optional) The sequence length range of peptide that we want to keep in the result. (e.g. c(1, 5) will include peptide sequence length from 1 to 5.)
`column_keep`	(Optional) The name of the columns in peptide_data to keep in result data frame.

Value

A data frame with columns from 'peptide_data' and 'whole_seq' indicating the matched positions and related information.

Examples

```
peptide_data <- data.frame(
  Sequence = c("AILNK", "BXLMR", "JJNXX", "DDEEF"),
  Condition_1 = c("Drug1", "Drug1", "Drug2", "Drug2"),
  Condition_2 = c("Donor1", "Donor2", "Donor1", "Donor2"),
  Region_1 = c("VH", "VL", "VH", "VL"),
  Region_2 = c("Arm_1", "Arm_2", "Arm_1", "Arm_2"),
  Area = c(100, 2, 4, NA)
)
whole_seq <- data.frame(
  Region_Sequence = c(
    "XYZAILNKPQR",
    "ABCBXLMRDEF",
    "GHIJJNXXKLM",
    "NOPDDEEFQRS",
    "AILXKPQR",
    "BNJLMRDEF",
    "ILNXXKLM",
    "DDEEXQRS",
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD"
  ),
  Condition_1 = c(
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2"
  ),
  Condition_2 = c(
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2"
  ),
  Region_1 = c(
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL"
  ),
  Region_2 = c(
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2"
  )
)
match_columns <- c("Condition_1", "Condition_2", "Region_1")
column_keep <- c("Region_2")
sequence_length <- c(1, 5)
column <- "Sequence"
matching_result <- match_and_calculate_positions(peptide_data,
                                                 column,
                                                 whole_seq,
                                                 match_columns,
                                                 sequence_length,
                                                 column_keep)
```

obtain_mod

Obtain post translational modification(PTM) information from Peptide data based on the specified data type

Description

This function takes outputs from multiple platform, a data frame with column containing modified peptide sequence with the detailed post translational modification(PTM) information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information. Due to the flexibility of outputs from multiple platform, the PTM mass to type table needs to be provided if convertion to PTM_type is needed. The result includes ’Peptide’, ’PTM_position’, ’PTM_type’ and ’PTM_mass’ columns.The function chooses the appropriate converting method based on the specified data type (’PEAKS’, ’Spectronaut’, ’MSFragger’, ’Comet’, ’DIANN’, ’Skyline’ or ’Maxquant’), allowing you to convert the data into a consistent format for further analysis.

Usage

```
obtain_mod(
  data,
  column,
  type,
  strip_seq_col = NULL,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A data frame with the peptide sequences.
`column`	The name of the column containing the modified peptide sequences.
`type`	A character string specifying the data type (e.g. ’Skyline’ or ’Maxquant’).
`strip_seq_col`	(Optional) The name of the column containing the stripped peptide sequences.
`PTM_table`	A data frame with columns ’PTM_mass’ and ’PTM_type’ containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information.

Value

A data.table with ’PTM_position’, ’PTM_type’, ’PTM_mass’, ’reps’, and other columns.

Examples

```
library(data.table)
data_skyline <- data.table(
  'Peptide Modified Sequence' = c(
    "AGLC[+57]QTFVYGGC[+57]R",
    "AAAASAAEAGIATTGTEDSDDALLK",
    "IVGGWEC[+57]EK"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(
  PTM_mass = c(57.02, -0.98, 15.9949),
  PTM_type = c("Cam", "Amid", "Ox")
)
converted_data_skyline <- obtain_mod(
  data_skyline,
  'Peptide Modified Sequence',
  'Skyline',
  strip_seq_col = NULL,
  PTM_table,
  PTM_annotation = TRUE,
  PTM_mass_column = "PTM_mass"
)

data_maxquant <- data.table(
  'Modified sequence' = c(
    "_(ac)AAAAELRLLEK_",
    "_EAAENSLVAYK_",
    "_AADTIGYPVM(ox)IRSAYALGGLGSGICPNK_"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(
  PTM_mass = c('Phospho (STY)', 'Oxidation (M)'),
  PTM_type = c("Phos", "Ox")
)
converted_data_maxquant <- obtain_mod(
  data_maxquant,
  'Modified sequence',
  'Maxquant',
  strip_seq_col = NULL,
  PTM_table,
  PTM_annotation = TRUE,
  PTM_mass_column = "PTM_mass"
)
```

`obtain_mod_Comet` *Obtain modification information from Peptide data generated by Comet*

Description

This function takes Comet output containing a column with modified peptide sequences including PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_Comet(
  data,
  column,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A data.table with a column containing PTM information.
`column`	The name of the column containing the modified peptide sequences.
`PTM_table`	A data.table with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information

Value

A data.table with 'PTM_position', 'PTM_type', 'reps', and other columns.

Examples

```
library(data.table)
data <- data.table(
  modified_peptide = c(
    "AAM[15.9949]Q[-0.98]RGSLYQCDYSTGSC[57.02]EPIR",
    "K.AAQQTGKLVHANFGT.K",
    "K.[-0.98]AATVTGKLVHANFGT.K"
  ),
  plain_peptide = c(
    "AAMQRGSLYQCDYSTGSCEPIR",
    "AAQQTGKLVHANFGT",
    "AATVTGKLVHANFGT"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(
  PTM_mass = c(57.02, -0.98, 15.9949),
  PTM_type = c("Cam", "Amid", "Ox")
)
column <- 'modified_peptide'
PTM_mass_column <- "PTM_mass"
converted_data <- obtain_mod_Comet(data, column, PTM_table, PTM_annotation = TRUE, PTM_mass_column)
```

obtain_mod_DIANN — *Obtain modification information from Peptide data generated by DIA-NN*

Description

This function takes DIA-NN output containing a column with modified peptide sequences including PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_DIANN(
  data,
  column,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A dataframe with 'Stripped.Sequence' column and 'Modified.Sequence' column containing modified peptide sequences.
`column`	The name of the column containing the modified peptide sequences.
`PTM_table`	A dataframe with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information

Value

A dataframe with 'Peptide', 'PTM_position', and 'PTM_type' columns.

Examples

```
library(data.table)
data <- data.table(
  Modified.Sequence = c(
    "AAAAGPGAALS(UniMod:21)PRPC(UniMod:4)DSDPATPGAQSPK",
    "AAAASAAEAGIATTGTEDSDDALLK",
    "AAAAALSGSPPQTEKPT(UniMod:21)HYR"
  ),
  Stripped.Sequence = c(
    "AAAAGPGAALSPRPCDSDPATPGAQSPK",
    "AAAASAAEAGIATTGTEDSDDALLK",
    "AAAAALSGSPPQTEKPTHYR"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(PTM_mass = c('UniMod:21', 'UniMod:4'),
                        PTM_type = c("Phos", "Cam"))
converted_data <- obtain_mod_DIANN(
  data,
  'Modified.Sequence',
  PTM_table,
  PTM_annotation = TRUE,
  PTM_mass_column = "PTM_mass"
)
```

obtain_mod_Maxquant   *Obtain modification information from Peptide data generated by Maxquant*

Description

This function takes Maxquant output containing a column with modified peptide sequences including PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_Maxquant(
  data,
  column,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A data.table with a column containing modified peptide sequences.
`column`	The name of the column containing the modified peptide sequences.
`PTM_table`	A data.table with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information

Value

A data.table with 'PTM_position', 'PTM_type', 'reps', and other columns.

Examples

```
library(data.table)
data <- data.table(
  'Modified sequence' = c(
    "_GLGPSPAGDGPS(Phospho (STY))GSGK_",
    "_HSSYPAGTEDDEGM(Oxidation (M))GEEPSPFR_",
    "_HSSYPAGTEDDEGM(Oxidation (M))GEEPS(Phospho (STY))PFR_"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(
  PTM_mass = c('Phospho (STY)', 'Oxidation (M)'),
  PTM_type = c("Phos", "Ox")
)
converted_data <- obtain_mod_Maxquant(
  data,
  'Modified sequence',
  PTM_table,
  PTM_annotation = TRUE,
  PTM_mass_column = "PTM_mass"
)
```

`obtain_mod_MSFragger` *Obtain modification information from Peptide data generated by MS-Fragger*

Description

This function takes MSFragger output containing a 'Assigned Modifications' column with PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_MSFragger(
  data,
  column,
  strip_seq_col,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A data.table with a column containing stripped sequence and a column containing PTM information.
`column`	The name of the column containing the modified peptide sequences.
`strip_seq_col`	The name of the column containing the stripped peptide sequences.
`PTM_table`	A data.table with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information

Value

A data.table with 'PTM_position', 'PTM_type', 'reps', and other columns.

Examples

```
library(data.table)
data <- data.table(
  Peptide = c("DDREDMLVYQAK", "EAAENSLVAYK", "IEAELQDICNDVLELLDK"),
  `Assigned Modifications` = c("C-term(15.9949), 6M(-0.98)", "", "N-term(42.0106)"),
  Condition1 = c("A", "B", "B"),
  Condition2 = c("C", "C", "D")
)
PTM_table <- data.table(
  PTM_mass = c(42.0106, -0.98, 15.9949),
  PTM_type = c("Acet", "Amid", "Ox")
)
column <- "Assigned Modifications"
strip_seq_col <- "Peptide"
converted_data <- obtain_mod_MSFragger(
  data,
  column,
  strip_seq_col,
  PTM_table,
  PTM_annotation = TRUE,
  PTM_mass_column = "PTM_mass"
)
```

`obtain_mod_PEAKS` *Obtain modification information from Peptide data generated by PEAKS*

Description

This function takes PEAKS output containing a column with modified peptide sequences including PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_PEAKS(
  data,
  column,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A dataframe with a column containing modified peptide sequences.
`column`	The name of the column containing the modified peptide sequences.
`PTM_table`	A dataframe with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information

Value

A data.table with 'PTM_position', 'PTM_type', 'PTM_mass', 'reps', and other columns.

Examples

```
library(data.table)
data <- data.table(
  Peptide = c(
    "AAN(+42)Q(-0.98)RGSLYQCDYSTGSC(+57.02)EPIR",
    "K.AAQQTGKLVHANFGT.K",
    "K.(-0.98)AATVTGKLVHANFGT.K"
  ),
  Sequence = c(
    "AANQRGSLYQCDYSTGSCEPIR",
    "AAQQTGKLVHANFGT",
    "AATVTGKLVHANFGT"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(PTM_mass = c(42, -0.98, 57.02),
                        PTM_type = c("Acet", "Amid", "Cam"))
column <- "Peptide"
```

```
PTM_mass_column <- "PTM_mass"
converted_data <- obtain_mod_PEAKS(data, column, PTM_table, PTM_annotation = TRUE, PTM_mass_column)
```

obtain_mod_Skyline *Obtain modification information from Peptide data generated by Skyline*

Description

This function takes Skyline output containing a column with modified peptide sequences including PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_Skyline(
  data,
  column,
  PTM_table,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

data	A data.table with a column containing PTM information.
column	The name of the column containing the modified peptide sequences.
PTM_table	A data.table with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
PTM_annotation	A logical value indicating whether to include PTM annotation information in the result.
PTM_mass_column	The name of the column containing the PTM mass information

Value

A data.table with 'PTM_position', 'PTM_type', 'reps', and other columns.

Examples

```
library(data.table)
data <- data.table(
  'Peptide Modified Sequence' = c(
    "AAM[15.9949]Q[-0.98]RGSLYQCDYSTGSC[57.02]EPIR",
    "AAQQTGKLVHANFGT",
    "[-0.98]AATVTGKLVHANFGT"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(
  PTM_mass = c(57.02, -0.98, 15.9949),
  PTM_type = c("Cam", "Amid", "Ox")
)
converted_data <- obtain_mod_Skyline(
  data,
  'Peptide Modified Sequence',
  PTM_table,
  PTM_annotation = TRUE,
  PTM_mass_column = "PTM_mass"
)
```

`obtain_mod_Spectronaut` *Obtain modification information from Peptide data generated by Spectronaut*

Description

This function takes Spectronaut output containing a column with modified peptide sequences including PTM information and converts it into a new dataframe with the desired format of peptide sequences and associated PTM information.

Usage

```
obtain_mod_Spectronaut(
  data,
  column,
  PTM_table = NULL,
  PTM_annotation = FALSE,
  PTM_mass_column
)
```

Arguments

`data`	A data.table with a column containing modified peptide sequences.
`column`	The name of the column containing the modified peptide sequences.
`PTM_table`	A data.table with columns 'PTM_mass' and 'PTM_type' containing PTM annotation information.
`PTM_annotation`	A logical value indicating whether to include PTM annotation information in the result.
`PTM_mass_column`	The name of the column containing the PTM mass information

Value

A data.table with 'PTM_position', 'PTM_type', 'reps', and other columns.

Examples

```
library(data.table)
data <- data.table(
  EG.ModifiedPeptide = c(
    "_[Acetyl (Protein N-term)]M[Oxidation (M)]DDREDLVYQAK_",
    "_EAAENSLVAYK_",
    "_IEAELQDIC[Carbamidomethyl (C)]NDVLELLDK_"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(
  PTM_mass = c(
    'Acetyl (Protein N-term)',
    'Oxidation (M)',
    'Carbamidomethyl (C)'
  ),
  PTM_type = c("Acet", "Ox", "Cam")
)
converted_data <- obtain_mod_Spectronaut(data, 'EG.ModifiedPeptide',
                                         PTM_table, PTM_annotation = TRUE,
                                         PTM_mass_column = "PTM_mass")
data <- data.table(
  EG.IntPIMID = c(
    "_[+42]M[-0.98]DDREDLVYQAK_",
    "_EAAENSLVAYK_",
    "_IEAELQDIC[+57]NDVLELLDK_"
  ),
  Condition = c("A", "B", "B")
)
PTM_table <- data.table(PTM_mass = c(42, -0.98, 57),
                        PTM_type = c("Acet", "Amid", "Cam"))
PTM_mass_column <- "PTM_mass"
converted_data <- obtain_mod_Spectronaut(data,
                                         'EG.IntPIMID',
                                         PTM_table,
                                         PTM_annotation = TRUE,
                                         PTM_mass_column)
```

---

peptide_quantification *Peptide Quantification*

Description

Peptide Quantification

Usage

```
peptide_quantification(
  whole_seq,
  matching_result,
  matching_columns,
  distinct_columns,
  quantify_method,
  area_column = NULL,
  with_PTM = FALSE,
  reps = FALSE
)
```

Arguments

`whole_seq`	A dataframe holding whole sequence information. 'Region_Sequence' column is required for the sequence information. Change the column name if it is different than 'Region_Sequence'.
`matching_result`	The dataframe that contains the matched results and PTM information.
`matching_columns`	Vector of column names that should match between each row of 'whole_seq' and the 'matching_result' dataframe.
`distinct_columns`	Vector of column names that should be used to calculate PSM or Area separately for each unique combination of these columns.
`quantify_method`	A string indicating the quantification method. It can be either "PSM" or "Area".
`area_column`	The name of the column in 'matching_result' that contains the area/intensity information. Required if quantify_method is "Area".
`with_PTM`	A boolean parameter indicating whether PTM should be considered during calculation. Default is `FALSE`.
`reps`	A boolean parameter indicating whether the area/intensity should be divided by the number of replicates. Default is `FALSE`.

Value

Returns a dataframe containing the calculated PSM or Area for each record in 'whole_seq'.

Examples

```
whole_seq <- data.frame(
  Region_Sequence = c(
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD",
    "XYZAAA",
    "XYZCCC",
    "XYZBBB",
    "XYZDDD",
    "XYZAAB",
    "XYZCCD",
    "XYZBBB",
    "XYZDDD"
  ),
  Condition_1 = c(
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2",
    "Drug1",
    "Drug1",
    "Drug2",
    "Drug2"
  ),
  Condition_2 = c(
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor1",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2",
    "Donor2"
  ),
  Region_1 = c(
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL",
    "VH",
    "VL"
  ),
  Region_2 = c(
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_1",
    "Arm_2",
    "Arm_2",
    "Arm_2",
    "Arm_2"
  )
)
matching_result <- data.frame(
  Sequence = c("AAA", "DDD", "DDD"),
  Condition_1 = c("Drug1", "Drug2", "Drug2"),
  Condition_2 = c("Donor1", "Donor2", "Donor2"),
  Region_1 = c("VH", "VL", "VL"),
  Region_2 = c("Arm_1", "Arm_2", "Arm_2"),
  Start_Position = c(4, 4, 4),
  End_Position = c(6, 6, 6),
  PTM_position = c(NA, 2, 0),
  PTM_type = c(NA, "O", "C"),
  Area = c(100, 200, 200),
  reps = c(1, 2, 2)
)
matching_columns <- c("Condition_1", "Region_2")
area_column <- "Area"
data_with_quantification <- peptide_quantification(
  whole_seq,
  matching_result,
  matching_columns,
  distinct_columns = c("Condition_2", "Region_1"),
  quantify_method = "Area",
  area_column = area_column,
  with_PTM = TRUE,
  reps = TRUE
)
```

strip_sequence *Strip peptide sequences based on the specified data type*

Description

This function takes outputs from multiple platform, a data frame with a column containing peptide sequences to be stripped, and a column where the stripped sequences will be stored. The function chooses the appropriate stripping method based on the specified data type ('PEAKS', 'Spectronaut', 'MSFragger', 'Comet', 'DIANN', 'Skyline' or 'Maxquant').

Usage

```
strip_sequence(data, column, convert_column, type)
```

Arguments

data	A data frame with the peptide sequences.
column	The name of the column containing the peptide sequences to be stripped.
convert_column	The name of the column where the stripped sequences will be stored.
type	A character string specifying the data type (e.g. 'Skyline' or 'Maxquant').

Value

A data frame with the specified column containing stripped sequences.

Examples

```
library(data.table)
data_skyline <- data.table(
  'Peptide Modified Sequence' = c(
    "AGLC[+57]QTFVYGGC[+57]R",
    "AAAASAAEAGIATTGTEDSDDALLK",
    "IVGGWEC[+57]EK"
  ),
  Condition = c("A", "B", "B")
)
data_maxquant <- data.table(
  'Modified sequence' = c(
    "_(ac)AAAAELRLLEK_",
    "_EAAENSLVAYK_",
    "_AADTIGYPVM(ox)IRSAYALGGLGSGICPNK_"
  ),
  Condition = c("A", "B", "B")
)
```

```
converted_data_skyline <- strip_sequence(data_skyline,
                                         'Peptide Modified Sequence',
                                         'Sequence',
                                         "Skyline")
converted_data_maxquant <- strip_sequence(data_maxquant, 'Modified sequence',
                                          'Sequence', "Maxquant")
```

---

strip_sequence_Comet *Strip sequence from Comet outputs*

Description

This function takes Comet output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_Comet(data, column, convert_column)
```

Arguments

data	A dataframe with a column containing peptide sequences to be stripped
column	The name of the column containing the peptide sequences to be stripped.
convert_column	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  modified_peptide = c(
    "AAM[15.9949]Q[-0.98]RGSLYQCDYSTGSC[57.02]EPIR",
    "K.AAQQTGKLVHANFGT.K",
    "K.[0.98]AATVTGKLVHANFGT.K"
  ),
  Condition = c("A", "B", "B")
)
column <- 'modified_peptide'
convert_column <- 'Sequence'
converted_data <- strip_sequence_Comet(data, column, convert_column)
```

`strip_sequence_DIANN` *Strip sequence from DIANN outputs*

Description

This function takes DIANN output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_DIANN(data, column, convert_column)
```

Arguments

`data`	A dataframe with a column containing peptide sequences to be stripped
`column`	The name of the column containing the peptide sequences to be stripped.
`convert_column`	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  Modified.Sequence = c(
    "AAAAGPGAALS(UniMod:21)PRPC(UniMod:4)DSDPATPGAQSPK",
    "AAAASAAEAGIATTGTEDSDDALLK",
    "AAAAALSGSPPQTEKPT(UniMod:21)HYR"
  ),
  Condition = c("A", "B", "B")
)
column <- 'Modified.Sequence'
convert_column <- 'Sequence'
converted_data <- strip_sequence_DIANN(data, column, convert_column)
```

`strip_sequence_Maxquant` *Strip sequence from Maxquant outputs*

Description

This function takes Maxquant output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_Maxquant(data, column, convert_column)
```

Arguments

`data`	A dataframe with a column containing peptide sequences to be stripped
`column`	The name of the column containing the peptide sequences to be stripped.
`convert_column`	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  'Modified sequence' = c(
    "_(ac)AA(ox)AAELRLLEK_",
    "_EAAENSLVAYK_",
    "_AADTIGYPVM(ox)IRSAYALGGLGSGICPNK_"
  ),
  Condition = c("A", "B", "B")
)
column <- 'Modified sequence'
convert_column <- 'Sequence'
converted_data <- strip_sequence_Maxquant(data, column, convert_column)
```

`strip_sequence_MSFragger` *Strip sequence from MSFragger outputs*

Description

This function takes MSFragger output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_MSFragger(data, column, convert_column)
```

Arguments

`data`	A dataframe with a column containing peptide sequences to be stripped
`column`	The name of the column containing the peptide sequences to be stripped.
`convert_column`	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  'Modified Peptide' = c(
    "AAM[15.9949]Q[-0.98]RGSLYQCDYSTGSC[57.02]EPIR",
    "K.AAQQTGKLVHANFGT.K",
    "K.[0.98]AATVTGKLVHANFGT.K"
  ),
  Condition = c("A", "B", "B")
)
column <- 'Modified Peptide'
convert_column <- 'Sequence'
converted_data <- strip_sequence_MSFragger(data, 'Modified Peptide', 'Sequence')
```

---

strip_sequence_PEAKS *Strip sequence from PEAKS outputs*

Description

This function takes PEAKS output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_PEAKS(data, column, convert_column)
```

Arguments

data	A dataframe with a column containing peptide sequences to be stripped
column	The name of the column containing the peptide sequences to be stripped.
convert_column	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  Peptide = c(
    "AAN(+0.98)Q(-0.98)RGSLYQCDYSTGSC(+57.02)EPIR",
    "K.AAQQTGKLVHANFGT.K",
    "K.(+0.98)AATVTGKLVHANFGT.K"
  ),
  Condition = c("A", "B", "B")
)
column <- "Peptide"
convert_column <- "Sequence"
converted_data <- strip_sequence_PEAKS(data, column, convert_column)
```

---

strip_sequence_Skyline

Strip sequence from Skyline outputs

Description

This function takes Skyline output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_Skyline(data, column, convert_column)
```

Arguments

`data`	A dataframe with a column containing peptide sequences to be stripped
`column`	The name of the column containing the peptide sequences to be stripped.
`convert_column`	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  'Peptide Modified Sequence' = c(
    "AGLC[+57]QTFVYGGC[+57]R",
    "AAAASAAEAGIATTGTEDSDDALLK",
    "IVGGWEC[+57]EK"
  ),
  Condition = c("A", "B", "B")
)
column <- 'Peptide Modified Sequence'
convert_column <- 'Sequence'
converted_data <- strip_sequence_Skyline(data, column, convert_column)
```

`strip_sequence_Spectronaut` *Strip sequence from Spectronaut outputs*

Description

This function takes Spectronaut output containing a column with peptide sequences to be stripped and converts it into a new dataframe with the stripped sequence

Usage

```
strip_sequence_Spectronaut(data, column, convert_column)
```

Arguments

`data`	A dataframe with a column containing peptide sequences to be stripped
`column`	The name of the column containing the peptide sequences to be stripped.
`convert_column`	The name of the column where the stripped sequences will be stored.

Value

A dataframe with a column containing stripped sequence

Examples

```
library(data.table)
data <- data.table(
  EG.IntPIMID = c(
    "_[+42]M[-16]DDREDLVYQAK_",
    "_EAAENSLVAYK_",
    "_IEAELQDIC[+57]NDVLELLDK_"
  ),
  Condition = c("A", "B", "B")
)
converted_data <- strip_sequence_Spectronaut(data, 'EG.IntPIMID', 'Sequence')
```

Index