MCnebula2

March 31, 2023

Title MCnebula algorithm integrated in S4 system of R.

Version 0.0.9000

Description MCnebula2 was used for metabonomics data analysis.
It is written in the S4 system of object-oriented programming. Its workflow, i.e.,
MCnebula workflow, is a complete metabolomics data analysis process,
including initial data preprocessing (data format conversion, feature detection),
compound identification based on MS/MS, statistical analysis,
compound structure and chemical class focusing, multi-level data visualization,
output report, etc.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.0

Depends ggplot2,
R (>= 2.10)

Imports BiocStyle,
bookdown,
ChemmineOB,
crayon,
data.table,
dplyr,
ggimage,
ggraph,
ggsci,
ggtext,
grid,
gridExtra,
grImport2,
igraph,
knitr,
methods,
pbapply,
rlang,
rmarkdown,

rsvg,
stringr,
styler,
svglite,
tibble,
tidyr

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Collate utils.R project.sirius.v4.R project.sirius.v5.R tools-colors.R tools-methods.R
tools-default_visualize.R tools-modify_ggset.R tools-export.R
tools-MSnbase-MODIFIED_compareSpectra.R tools-yaml.R tools-report.R
base-generic.R extra-generic.R main-generic.R
class-VIRTUAL_slots.R class-melody.R
class-project_conformation.R class-project_metadata.R
class-project_api.R class-project_dataset.R class-project.R
class-statistic_set.R class-mcn_dataset.R
class-msframe.R class-command.R class-ggset.R class-report.R
class-section.R class-nebula.R class-mcnebula.R
methods-initialize_mcnebula.R
extraMethods-collate_data.R methods-filter_structure.R
methods-filter_formula.R methods-filter_ppcp.R
methods-create_hierarchy.R methods-create_reference.R
methods-create_features_annotation.R
methods-create_stardust_classes.R methods-cross_filter_stardust.R
methods-backtrack_stardust.R methods-create_nebula_index.R
methods-compute_spectral_similarity.R
methods-create_parent_nebula.R methods-create_child_nebulae.R
methods-create_parent_layout.R methods-create_child_layouts.R
methods-activate_nebulae.R methods-visualize.R
extraMethods-draw_structures.R extraMethods-draw_nodes.R
methods-annotate_nebula.R extraMethods-binary_comparison.R
extraMethods-report.R extraMethods-workflow.R
data.R functions-plot_msms_mirrors.R functions-clear.R
functions-report.R
zzz.R

LazyData true

Author LiChuang Huang <shaman.yellow@foxmail.com> [aut] (<https://orcid.org/0000-0002-5445-1988>)

Maintainer LiChuang Huang <shaman.yellow@foxmail.com>

R topics documented:

`.font` *Set font for visualization of MCnebula2*

Description

Note that your R harbours the font you set.

Usage

```
.font
```

Arguments

`font`	character(1). Such as 'Times'. If you output the visualization for pdf, use `grDevices::pdfFonts()` to checkout the available fonts; else, you might need help with package `extrafont`.

Format

An object of class `character` of length 1.

`ABSTRACT-MCnebula2` *Overview of MCnebula2*

Description

MCnebula2 was used for metabonomics data analysis. It is written in the S4 system of object-oriented programming, and starts with a "class", namely "mcnebula". The whole process takes the "mcnebula" as the operating object to obtain visual results or operating objects.

Most methods of MCnebula2 are S4 methods and have the characteristics of parameterized polymorphism, that is, different functions will be used for processing according to different parameters passed to the same method.

MCnebula workflow is a complete metabolomics data analysis process, including initial data preprocessing (data format conversion, feature detection), compound identification based on MS/MS, statistical analysis, compound structure and chemical class focusing, multi-level data visualization, output report, etc.

It should be noted that the MCnebula2 R package currently cannot realize the entire analysis process of MCnebula workflow. If users want to complete the entire workflow, other software beyond the R console (for example, the MSconvert tool of proteowizard is used for data format conversion, which is a tool widely applicable to metabonomics and proteomics) should be used. This is a pity, but we will gradually integrate all parts of the workflow into this R package in the future to achieve one-stop analysis.

The analysis process in R is integrated into the following methods:

- `initialize_mcnebula()`
- `filter_structure()`
- `create_reference()`
- `filter_formula()`
- `create_stardust_classes()`
- `create_features_annotation()`
- `cross_filter_stardust()`
- `create_nebula_index()`
- `compute_spectral_similarity()`
- `create_parent_nebula()`
- `create_child_nebulae()`
- `create_parent_layout()`
- `create_child_layouts()`
- `activate_nebulae()`
- `visualize()`
- `binary_comparison()`
- ...

Details

Overall. We know that the analysis of untargeted LC-MS/MS dataset generally begin with feature detection. It detects 'peaks' as features in MS1 (MASS level 1) data. Each feature may represents a compound, and assigned with MS2 (MASS level 2) spectra. The MS2 spectra was used to find out

the compound identity. The difficulty lies in annotating these features to discover their compound identity, mining out meaningful information, so as to serve further biological research. In addition, the un-targeted LC-MS/MS dataset is general a huge dataset, which leads to time-consuming analysis of the whole process. Herein, a classified visualization method, called MCnebula, was used for addressing these difficulty.

MCnebula utilizes the state-of-the-art computer prediction technology, SIRIUS workflow (SIRIUS, ZODIAC, CSI:fingerID, CANOPUS), for compound formula prediction, structure retrieve and classification prediction (https://bio.informatik.uni-jena.de/software/sirius/). MCnebula integrates an abundance-based classes (ABC) selection algorithm into features annotation: depending on the user, MCnebula focuses chemical classes with more or less features in the dataset (the abundance of classes), visualizes them, and displays the features they involved; these classes can be dominant structural classes or sub-structural classes. With MCnebula, we can switch from untargeted to targeted analysis, focusing precisely on the compound or chemical class of interest to the researcher.

MCnebula2. The MCnebula2 package itself does not contain any part of molecular formula prediction, structure prediction and chemical prediction of compounds, so the accuracy of these parts is not involved. MCnebula2 performs downstream analysis by extracting the prediction data from SIRIUS project. The core of MCnebula2 is its chemical filtering algorithm, called ABC selection algorithm.

Chemical structure and formula. To explain the ABC selection algorithm in detail, we need to start with MS/MS spectral analysis and identification of compounds: The analysis of MS/MS spectrum is a process of inference and prediction. For example, we speculate the composition of elements based on the molecular weight of MS1; combined with the possible fragmentation pattern of MS2 spectrum, we speculate the potential molecular formula of a compound; finally, we look for the exact compound from the compound structure database. Sometimes, this process is full of uncertainty, because there are too many factors that affect the reliability of MS/MS data and the correctness of inference. It can be assumed that there are complex candidates for the potential chemical molecular formula, chemical structure and chemical class behind MS/MS spectrum. Suppose we have these data of candidates now, MCnebula2 extracted these candidates and obtained the unique molecular formula and chemical structure for each MS/MS spectrum based on the highest score of chemical structure prediction; in this process, as most algorithms do, we make a choice based on the score, and only select the result of highest score.

The chemical formula and structure candidates can obtain by methods:

- `filter_formula()`
- `filter_structure()`

In order to obtain the best (maybe), corresponding and unique chemical formula and structure from complex candidates, an important intermediate link:

- `create_reference()`

Above, we talked about chemical molecular formula, chemical structural formula and chemical classes. We obtained the unique chemical molecular formula and chemical structure formula for reference by scoring and ranking. But for chemical classes, we can't adopt such a simple way to get things done.

Chemical classification. Chemical classification is a complex system. Here, we only discuss the structure based chemotaxonomy system, because the MS/MS spectrum is more indicative of the structure of compounds than biological activity and other information.

According to the division of the overall structure and local structure of compounds, we can call the structural characteristics as the dominant structure and substructure. (https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0174-y). Correspondingly, in the chemical classification system, we can not only classify according to the dominant structure, but also classify according to the substructure. The chemical classification based on the dominant structure of compounds is easy to understand, because we generally define it in this way. For example, we will classify Taxifolin as "flavones", not "phenols", although its local structure has a substructure of "phenol".

We hope to classify a compound by its dominant structure rather than substructure, because such classify is more concise and contains more information. However, in the process of MS/MS spectral analysis, we sometimes can only make chemical classification based on the substructure of compounds, which may be due to: uncertainty in the process of structural analysis; it may be an unknown compound; MS/MS spectral fragment information is insufficient. In this case, it is necessary for us to classify the compounds with the aid of substructure information, otherwise we have no knowledge of the compounds for which we cannot obtain dominant structure information.

Above, we discussed the complex chemical classification for the substructure and dominant structure of compounds. We must also be clear about the complexity of another aspect of chemotaxonomy, i.e., the hierarchy of classification. This is easy to understand. For example, "Flavones" belongs to its superior, "Flavonoids"; its next higher level, "Phynylpropanoids and polyketides"; the further upward classification is "organic compounds".

ABC selection. The above section discusses the inferential prediction of individual MS/MS spectrum. In the un-targeted LC-MS/MS dataset, each feature has a corresponding MS/MS spectrum, and there are thousands of features in total. The ABC selection algorithm regards all features as a whole, examines the number and abundance of features of each chemical classification (classification at different levels, classification of substructure and dominant structure), and then selects representative classes (mainly screening the classes according to the number or abundance range of features) to serve the subsequent analysis. The core methods for ABC selection algorithm are:

- `create_stardust_classes()`
- `cross_filter_stardust()`
- `create_nebula_index()`

Whether it is all filtered by the algorithm provided by MCnebula2's function or custom filtered for some chemical classes, we now have a data called 'nebula_index'. This data records a number of chemical classes and the 'features' attributed to them. The subsequent analysis process or visualization will be based on it. Each chemical class is considered as a 'nebula' and its classified 'features' are the components of these 'nebulae'. In the visualization, these 'nebulae' will be visualized as networks. Formally, we call these 'nebulae' formed on the basis of 'nebula_index' data as Child-Nebulae. In comparison, when we put all the 'features' together to form a large network, then this 'nebula' is called Parent-Nebulae.

`activate_nebulae-methods`

activate Nebulae for visualization

Description

Based on layouts create by `create_parent_layout()` or `create_child_layouts()`, use functions to activate Nebulae as ggset object for `visualize()` methods to draw them.

`activate_nebulae()`: get the default parameters for the method `activate_nebulae`.

`activate_nebulae(x, ...)`: use the default parameters whatever 'missing' while performing the method `activate_nebulae`.

`ggset_activate_parent_nebula`: create ggset object of Parent-Nebula.

`ggset_activate_child_nebulae`: create lists of ggset object of Child-Nebulae.

Usage

```
## S4 method for signature 'missing,missing,missing'
activate_nebulae()

## S4 method for signature 'mcnebula,ANY,ANY'
activate_nebulae(x, fun_default_parent, fun_default_child)

## S4 method for signature 'mcnebula,`function`,`function`'
activate_nebulae(x, fun_default_parent, fun_default_child)

ggset_activate_parent_nebula(x)

ggset_activate_child_nebulae(x)
```

Arguments

`x`	mcnebula object.
`fun_default_parent`	function. Passed to create ggset object for Parent-Nebula. Default is `ggset_activate_parent_nebula`. Normally not used.
`fun_default_child`	function. Passed to create ggset object for Child-Nebulae. Default is `ggset_activate_child_nebulae`. Normally not used.

See Also

ggset, `create_parent_layout()`, `create_child_layouts()`...

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
```

```
test1 <- create_features_annotation(test1)
test1 <- cross_filter_stardust(test1, 2, 1)
test1 <- create_nebula_index(test1)
test1 <- compute_spectral_similarity(test1)
test1 <- create_parent_nebula(test1, 0.01)
test1 <- create_child_nebulae(test1, 0.01)
test1 <- create_parent_layout(test1)
test1 <- create_child_layouts(test1)

## default parameters
activate_nebulae()

test1 <- activate_nebulae(test1)
## see results
ggset(parent_nebula(test1))
head(ggset(child_nebulae(test1)))

## visualize
call_command(ggset(parent_nebula(test1)))
## or
visualize(test1, "parent")
## child nebula
call_command(ggset(child_nebulae(test1))[[1]])
## or
visualize(test1, 1)

## End(Not run)
```

---

annotate_nebula-methods

Add multiple annotation data for visualization of Child-Nebula.

Description

Use methods draw_nodes() and draw_structures() to standby visualization of Child-Nebula with mutiple annotation: chemical classification, 'features' quantification, chemical structure... Run after activate_nebulae().

Usage

```
## S4 method for signature 'ANY,character'
annotate_nebula(x, nebula_name)
```

Arguments

x	mcnebula object.
nebula_name	character(1). Chemical classes in 'nebula_index' data.

Details

Primarily, remove the ggraph::geom_node_point() layer in ggset object of Child-Nebula. The 'nodes' would be replaced with 'grob' object create by draw_nodes(). The function of ggimage::geom_subview() is used to add 'grob' object into 'ggplot' object.

See Also

activate_nebulae(), draw_nodes(), draw_structures(), set_ppcp_data(), set_ration_data()...

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)

  ## set features quantification data
  ids <- features_annotation(test1)$.features_id
  quant. <- data.frame(
    .features_id = ids,
    sample_1 = rnorm(length(ids), 1000, 200),
    sample_2 = rnorm(length(ids), 2000, 500)
  )
  metadata <- data.frame(
    sample = paste0("sample_", 1:2),
    group = c("control", "model")
  )
  features_quantification(test1) <- quant.
  sample_metadata(test1) <- metadata

  ## optional 'nebula_name'
  visualize(test1)
  ## a class for example
  class <- visualize(test1)$class.name[1]
  tmp <- export_path(test1)
  test1 <- annotate_nebula(test1, class)

  ## The following can be run before "annotate_nebula()"
  ## to customize the visualization of nodes.
  # test1 <- draw_structures(test1, "Fatty Acyls")
  ## set parameters for visualization of nodes
```

```
  # test1 <- draw_nodes(
  #   test1, "Fatty Acyls",
  #   add_id_text = T,
  #   add_structure = T,
  #   add_ration = T,
  #   add_ppcp = T
  # )
  # test1 <- annotate_nebula(test1, class)

  ## see results
  ggset <- ggset_annotate(child_nebulae(test1))
  ggset[[class]]
  ## visualize 'ggset'
  call_command(ggset[[class]])

## End(Not run)
```

---

backtrack-class *Share slots and methods for classes inherite from VIRTUAL_backtrack*

Description

This VIRTUAL class provides a slot for storing discarded data.

backtrack, backtrack<-: getter and setter for the backtrack slot of the object.

Usage

```
## S4 method for signature 'ANY'
backtrack(x)

## S4 replacement method for signature 'ANY'
backtrack(x) <- value
```

Arguments

x	object inherit class backtrack.
value	The value for the slot.

Slots

backtrack list with names.

backtrack_stardust-methods *Recover filtered chemical classses for 'stardust_classes'*

Description

These methods used for custom modify chemical classes in 'stardust_classes' data. Users can use the method to recover classes which filtered out by `cross_filter_stardust()` into 'stardust_classes' data. In addition, users can use the method to delete chemical classes in 'stardust_classes' data.

`backtrack_stardust(object)`: get the filtered chemical classes after using `cross_filter_stardust()`.

Run after `cross_filter_stardust()`.

Usage

```
## S4 method for signature 'mcnebula,missing,missing,missing'
backtrack_stardust(x)

## S4 method for signature 'mcnebula,character,missing,ANY'
backtrack_stardust(x, class.name, remove)

## S4 method for signature 'mcnebula,missing,numeric,ANY'
backtrack_stardust(x, rel.index, remove)
```

Arguments

`x`	mcnebula object.
`class.name`	character. The chemical classes name.
`remove`	logical. If `TRUE`, remove the specified chemical classes in 'stardust_classes' data. If `FALSE`, recover the data of specified chemical classes into 'stardust_classes'; the classes must in slot `backtrack(mcn_dataset(object))`.
`rel.index`	numeric. The index number of chemical classes. See columns of 'rel.index' in 'nebula_index' or 'stardust_classes'.

See Also

`cross_filter_stardust()`

`binary_comparison-methods` *Binary comparison for 'features' quantification data*

Description

Use the functions in the 'limma' package for simple binary statistical analysis.

`binary_comparison()`: get the default parameters for the method `binary_comparison`.

`binary_comparison(x, ...)`: use the default parameters whatever 'missing' while performing the method `binary_comparison`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing,missing'
binary_comparison()

## S4 method for signature 'ANY,ANY,ANY,ANY,ANY'
binary_comparison(x, ..., formula, fun_norm, top_coef, contrasts)

## S4 method for signature 'ANY,formula,`function`,ANY,character'
binary_comparison(x, formula, fun_norm, top_coef, contrasts)
```

Arguments

`x`	mcnebula object.
`...`	expressions, or character strings which can be parsed to expressions, specifying contrasts. See parameter of `...` in `limma::makeContrasts()`.
`formula`	formula. Passed to `model.matrix()`.
`fun_norm`	function. For normalization of 'features' quantification data.
`top_coef`	list, NULL or character(1). Specified the parameter of `coef` in `limma::topTable()`. If `"all"`, all coefficient in contrast matrix would be used one by one.
`contrasts`	character vector specifying contrasts. See parameter `contrasts` in `limma::makeContrasts()`.

See Also

`stats::model.matrix()`, `limma::makeContrasts()`, `limma::lmFit()`, `limma::eBayes()`, `limma::contrasts.fit()`, `limma::topTable()`...

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
```

```
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_features_annotation(test1)

  ## set up a simulated quantification data.
  test1 <- .simulate_quant_set(test1)
  ## the simulated data
  features_quantification(test1)
  sample_metadata(test1)

  test1 <- binary_comparison(
    test1, control - model,
    model - control, 2 * model - control
  )
  ## see results
  top_table(statistic_set(test1))

  ## the default parameters
  binary_comparison()

## End(Not run)
```

`clear` *Clean up data in the mcnebula object that is no longer in use*

Description

`clear_dataset`: The data (chemical formula, chemical structure, chemical classes) in `project_dataset(x)`, and data in `backtrack(mcn_dataset(mcn)` would be clean up to reduced memory usage. This may be best used after running the `create_nebula_index()`, if your machine doesn't have much Random Access Memory (RAM).

`clear_nodes`: Clear data ('grobs' and 'ggset') of 'nodes' in slot `child_nebulae`.

Usage

```
clear_dataset(x)

clear_nodes(x)
```

Arguments

x	mcnebula object.

Note

If this function has conducted, the PPCP dataset would not be available for downstream methods, such as `set_ppcp_data()`, `annotate_nebula()`...

code_block-class *Sequestrate code and setting run parameters.*

Description

Mainly desiged for R code block. The job of this class object is to record the codes and the running parameters of its source language or program; These information can then be output as formatted code block text (use `call_command()`).

`code_block_table`: class inherit from `code_block`, with default values for slot `command_args` facilitate showing table in document.

`code_block_figure`: class inherit from `code_block`, with default values for slot `command_args` facilitate showing figure in document.

`code_block`, `code_block<-`: getter and setter for the `code_block` slot of the object.

`codes`, `codes<-`: getter and setter for the `codes` slot of the object.

`new_code_block`: create a code_block object.

`new_code_block()`: get the default parameters for the method `new_code_block`.

`new_code_block(x, ...)`: use the default parameters whatever 'missing' while performing the method `new_code_block`.

`new_code_block_figure`: create code_block_figure object. This methods simplified parameter settings for displaying figures in documents.

`new_code_block_table`: create code_block_table object. This methods simplified parameter settings for displaying table in documents.

`call_command`: Format 'code_block' object as character.

`get_ref`: get the string for cross-reference.

Usage

```
## S4 method for signature 'code_block'
show(object)

## S4 method for signature 'code_block_table'
show(object)

## S4 method for signature 'code_block_figure'
show(object)

## S4 method for signature 'heading'
show(object)

## S4 method for signature 'section'
show(object)

## S4 method for signature 'ANY'
```

```
code_block(x)

## S4 replacement method for signature 'ANY'
code_block(x) <- value

## S4 method for signature 'code_block'
codes(x)

## S4 replacement method for signature 'code_block'
codes(x) <- value

## S4 method for signature 'character,character,list,logical,`function`'
new_code_block(language, codes, args, prettey, fun_prettey)

## S4 method for signature 'missing,missing,missing,missing,missing'
new_code_block()

## S4 method for signature 'ANY,ANY,ANY,ANY,ANY'
new_code_block(language, codes, args, prettey, fun_prettey)

## S4 method for signature 'character'
new_code_block_figure(name, caption, ...)

## S4 method for signature 'character'
new_code_block_table(name, ...)

## S4 method for signature 'code_block'
call_command(x)

get_ref(object, type = c("fig", "tab"))
```

Arguments

object	code_block_figure or code_block_table object.
value	The value for the slot.
language	character(1). For slot command_name.
codes	character. For slot codes.
args	list. For slot command_args.
prettey	logical. If ture, use styler::style_text() to pretty the codes.
fun_prettey	function. Default is styler::style_text.
name	character(1). For cross-reference in document. See https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref.
caption	character(1). Caption of figure display in document.
...	Other parameters passed to new_code_block().
type	character. "fig" or "tab".

Slots

`codes` character. Codes.

`command_name` character(1). Program or language. e.g., "r".

`command_function` function. Used for gather the codes and args as code block.

`command_args` list. Args passed to program.

See Also

command-class. https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref. https://bookdown.org/yihui/rmarkdown/compile.html.

Other call_commands: command-class, ggset-class, report-class, section-class

Examples

```
## Not run:
  ## general
  codes <- "df <- data.frame(x = 1:10)
    df<-dplyr::mutate(df,y=x*1.5)%>%
    dplyr::filter(x >= 5)
    p <- ggplot(df)+
    geom_point(aes(x=x,y=y))
    p"
  block <- new_code_block("r", codes, list(eval = T, echo = T, message = T))
  ## see results
  block
  call_command(block)
  writeLines(call_command(block))

  ## figure
  fig_block <- new_code_block_figure(
    "plot1",
    "this is a caption",
    codes = codes
  )
  ## see results
  fig_block
  writeLines(call_command(fig_block))
  command_args(fig_block)
  cat(get_ref(fig_block), "\n")

  ## table
  codes <- "df <- data.frame(x = 1:10) %>%
    dplyr::mutate(y = x, z = x * y)
    knitr::kable(df, format = 'markdown', caption = 'this is a caption') "
  tab_block <- new_code_block_table("table1", codes = codes)
  ## see results
  tab_block
  cat(get_ref(tab_block), "\n")

  ## default parameters
```

```
  new_code_block()


## End(Not run)
## Not run:
  ## general
  codes <- "df <- data.frame(x = 1:10)
    df<-dplyr::mutate(df,y=x*1.5)%>%
    dplyr::filter(x >= 5)
    p <- ggplot(df)+
    geom_point(aes(x=x,y=y))
    p"
  block <- new_code_block("r", codes, list(eval = T, echo = T, message = T))
  ## see results
  block
  call_command(block)
  writeLines(call_command(block))

  ## figure
  fig_block <- new_code_block_figure(
    "plot1",
    "this is a caption",
    codes = codes
  )
  ## see results
  fig_block
  writeLines(call_command(fig_block))
  command_args(fig_block)
  cat(get_ref(fig_block), "\n")

  ## table
  codes <- "df <- data.frame(x = 1:10) %>%
    dplyr::mutate(y = x, z = x * y)
    knitr::kable(df, format = 'markdown', caption = 'this is a caption') "
  tab_block <- new_code_block_table("table1", codes = codes)
  ## see results
  tab_block
  cat(get_ref(tab_block), "\n")

  ## default parameters
  new_code_block()


## End(Not run)
```

---

collate_data-methods *Extract and format data from raw project directory*

Description

The primary method used to extract data from the raw project directory. By specifying subscript, this method reads all corresponding files, followed by gathering and formating the data, then stores these data in the slot (`dataset(project_dataset(object))`).

`collate_data()`: get the default parameters for the method `collate_data`.

`collate_data(x, ...)`: use the default parameters whatever 'missing' while performing the method `collate_data`.

`collate_used`: Use `filter_structure()` and `create_reference()` to build 'specific_candidate' data, then collate all used data of MCnebula workflow from Project directory, for subsequent data processing.

Usage

```
## S4 method for signature 'missing,missing,missing'
collate_data()

## S4 method for signature 'ANY,ANY,ANY'
collate_data(x, subscript, fun_collate, ...)

## S4 method for signature 'ANY,character,`function`'
collate_data(x, subscript, fun_collate, ...)

collate_used(x)
```

Arguments

`x`	project object or other class object inheriting it.
`subscript`	character(1). See subscript.
`fun_collate`	function. Used to extract and format the data from raw project directory. The default is `MCnebula2:::.collate_data.msframe`.
`...`	Other parameters passed to the fun_collate.

Details

This methods requires the name and path of the file in the raw project directory, as well as the reading function; These are recorded in project.

Note

Normally, users do not need to use this method for MCnebula2 analysis. `filter_formula()`, `filter_structure()`, `filter_ppcp()` provide more understandable usage.

Examples

```
## Not run:
  ## The raw data used for the example
  tmp <- paste0(tempdir(), "/temp_data")
```

```
  dir.create(tmp)
  eg.path <- system.file("extdata", "raw_instance.tar.gz",
                         package = "MCnebula2")

  utils::untar(eg.path, exdir = tmp)

  ## initialize 'mcnebula' object
  test <- mcnebula()
  test <- initialize_mcnebula(test, "sirius.v4", tmp)

  ## extract candidates data in SIRIUS project directory
  ## chemical structure
  test <- collate_data(test, ".f3_fingerid")
  latest(project_dataset(test))

  ## chemical formula
  test <- collate_data(test, ".f2_formula")
  latest(project_dataset(test))

  ## chemical classes
  test <- collate_data(test, ".f3_canopus")
  latest(project_dataset(test))

  ## mz and rt
  test <- collate_data(test, ".f2_info")
  latest(project_dataset(test))

  ## classification description
  test <- collate_data(test, ".canopus")

  ## the extracted data in 'mcnebula'
  dataset(project_dataset(test))
  entity(dataset(project_dataset(test))$.f3_fingerid)

  unlink(tmp, T, T)

## End(Not run)
```

`command-class` *Preparation of an instruction to be executed*

Description

Packing the funciton and the args inside this class object, so that it can be performed easily at any time.

`command_name`, `command_name<-`: getter and setter for the `command_name` slot of the object.

`command_function`, `command_function<-`: getter and setter for the `command_function` slot of the object.

`command_args`, `command_args<-`: getter and setter for the `command_args` slot of the object.

new_command: create an object of command.

call_command: Execute the function (slot command_function) with the parameters (slot command_args).

Usage

```
## S4 method for signature 'command'
show(object)

## S4 method for signature 'command'
command_name(x)

## S4 replacement method for signature 'command'
command_name(x) <- value

## S4 method for signature 'command'
command_function(x)

## S4 replacement method for signature 'command'
command_function(x) <- value

## S4 method for signature 'command'
command_args(x)

## S4 replacement method for signature 'command'
command_args(x) <- value

## S4 method for signature '`function`,character'
new_command(fun, ..., name)

## S4 method for signature '`function`,missing'
new_command(fun, ..., name)

## S4 method for signature 'command'
call_command(x)
```

Arguments

value	The value for the slot.
fun	function.
...	parameters (with names or without names) passed to the function.
name	character(1). Name to slot command_name.

Slots

command_name character(1). Describe the command name.

command_function function.

command_args the parameters passed to the function.

See Also

Other call_commands: code_block-class, ggset-class, report-class, section-class

Other call_commands: code_block-class, ggset-class, report-class, section-class

Examples

```
## Not run:
  ## example 1
  com <- new_command(plot, x = 1:10)
  com
  call_command(com)

  ## example 2
  com <- new_command(data.frame, x = 1:10, y = 1:10, z = 1:10)
  call_command(com)

  ## example 3
  data <- data.frame(x = 1:10, y = 1:10)
  com1 <- new_command(ggplot, data)
  com2 <- new_command(geom_point, aes(x = x, y = y))
  call_command(com1) + call_command(com2)

  ## slots
  command_name(com)
  command_args(com)
  command_function(com)

## End(Not run)
```

---

compute_spectral_similarity-methods

Compute MS2 spectral similarity

Description

These are methods stripped from MSnbase::compareSpectra. The unnecessary parts were removed, compute_spectral_similarity() only calculate the 'dotproduct' for two spectra and get the similarity value. It allows faster results for batch comparisons.

compute_spectral_similarity(): get the default parameters for the method compute_spectral_similarity.

compute_spectral_similarity(x, ...): use the default parameters whatever 'missing' while performing the method compute_spectral_similarity.

Usage

```
## S4 method for signature 'missing,missing,missing,missing,missing'
compute_spectral_similarity()
```

```
## S4 method for signature 'mcnebula,ANY,ANY,ANY,ANY'
compute_spectral_similarity(x, within_nebula, recompute, sp1, sp2)

## S4 method for signature
## 'missing,missing,missing,lightSpectrum,lightSpectrum'
compute_spectral_similarity(sp1, sp2)

## S4 method for signature 'missing,missing,missing,data.frame,data.frame'
compute_spectral_similarity(sp1, sp2)

## S4 method for signature 'mcnebula,logical,logical,missing,missing'
compute_spectral_similarity(x, within_nebula, recompute)
```

Arguments

x	mcnebula object.
within_nebula	logical. If TRUE, only 'features' that exist in a Child-Nebula are compared for spectral similarity. Data of 'nebula_index' (nebula_index(object)) would be used for assigning and combining the 'features' of comparison.
recompute	logical. If TRUE, discard the existing data in the object, and recompute the spectral similarity.
sp1	data.frame. An additional channel for comparing two spectrum. Contains 'mz' and 'intensity' for spectral comparison.
sp2	data.frame. An additional channel for comparing two spectrum. Contains 'mz' and 'intensity' for spectral comparison.

See Also

MSnbase::compareSpectra().

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)

  test1 <- compute_spectral_similarity(test1)
  ## see results
  spectral_similarity(test1)
  ## or
  reference(test1)$spectral_similarity
```

```
## or
reference(mcn_dataset(test1))$spectral_similarity

## compare the two spectra individually
spectra <- latest(test1, "project_dataset", ".f3_spectra")
data1 <- dplyr::select(
  dplyr::filter(spectra, .features_id == "gnps1537"),
  mz, int.
)
data2 <- dplyr::select(
  dplyr::filter(spectra, .features_id == "gnps1539"),
  mz, int.
)
e1 <- compute_spectral_similarity(sp1 = data1, sp2 = data2)
e1
# [1] 0.7670297

## MSnbase
if (requireNamespace("MSnbase")) {
  MSnbase::compareSpectra
  spec1 <- new("Spectrum2", mz = data1$mz, intensity = data1$int.)
  spec2 <- new("Spectrum2", mz = data2$mz, intensity = data2$int.)
  e2 <- MSnbase::compareSpectra(spec1, spec2, fun = "dotproduct")
  identical(e1, e2)
}

## End(Not run)
```

`create_child_layouts-methods`

Create layouts for visualization of Child-Nebulae

Description

Create visual style of Child-Nebulae. The 'style' means a variety of layouts for drawing the networks (i.e. all Child-Nebulae). See details.

`create_child_layouts()`: get the function for generating default parameters for the method `create_child_layouts`.

`create_child_layouts(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_child_layouts`.

Usage

```
## S4 method for signature
## 'missing,missing,missing,missing,missing,missing,missing'
create_child_layouts()

## S4 method for signature 'mcnebula,ANY,ANY,ANY,ANY,ANY,ANY'
```

```
create_child_layouts(
  x,
  ggraph_layouts,
  seeds,
  grid_layout,
  viewports,
  panel_viewport,
  legend_viewport
)
```

Arguments

x	mcnebula object.
ggraph_layouts	character with names or not. If with names, the names should be chemical classes in 'nebula_index' data. The names used to specify layout for all or partial Child-Nebulae. The value, see `ggraph::create_layout()`.
seeds	numeric with names or not. The names, see parameter `ggraph_layouts`. The values would passed to `set.seed()`
grid_layout	'layout' object. Create by `grid::grid.layout()`.
viewports	list with names or not. Each element is a 'viewport' object create by `grid::viewport()`
panel_viewport	'viewport' object. Describe the size and position for drawing overall Child-Nebulae (panel).
legend_viewport	'viewport' object. Describe the size and position for drawing legend of Child-Nebulae.

Details

This method provides a flexible way to draw Child-Nebulae. Users can create visual style based on default parameters. For experienced users of 'grid' package, the related functions such as `grid::grid.layout()`, `grid::viewport()` can be used to create customized visualizations. The layouts for visualization of Child-Nebulae include:

- nodes position: `layout_ggraph`
- size and position of grid panel: `grid_layout`
- size and position of each Child-Nebula (inside the panel): `viewports`
- size and position of overall Child-Nebulae: `panel_viewport`
- size and position of overall legend: `legend_viewport`

See Also

`grid::viewport()`, `grid::grid.layout()`, `ggraph::create_layout()`...

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_child_nebulae(test1, 0.01)

  ## function to generate default parameters
  create_child_layouts()
  ## default parameters
  create_child_layouts()(test1)

  test1 <- create_child_layouts(test1)
  ## see results (a object for 'ggraph' package to visualization)
  lapply(
    layout_ggraph(child_nebulae(test1)),
    tibble::as_tibble
  )

## End(Not run)
```

---

create_child_nebulae-methods

Gather data to create Child-Nebulae

Description

Similar to `create_parent_nebula()`, gather 'spectral_similarity' data and and 'features_annotation' data; but additionally, use 'nebula_index' data to group 'features' by chemical classes. Each chemical classes in 'nebula_index' data would lead to a 'igraph' object.

`create_child_nebulae()`: get the default parameters for the method `create_child_nebulae`.

`create_child_nebulae(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_child_nebulae`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing'
create_child_nebulae()

## S4 method for signature 'mcnebula,ANY,ANY,ANY'
```

```
create_child_nebulae(x, edge_cutoff, max_edge_number, use_tracer)

## S4 method for signature 'mcnebula,numeric,numeric,logical'
create_child_nebulae(x, edge_cutoff, max_edge_number, use_tracer)
```

Arguments

x	mcnebula object.
edge_cutoff	numeric(1). Value in (0,1). Set a threshold to create edges upon similarity value of 'spectral_similarity' data.
max_edge_number	numeric(1). For nodes (features) in each Child-Nebulae (i.e. network), the maximum number of edges link with. If the number exceeds the limitation, only edges representing higher spectral similarity would be retained.
use_tracer	logical. If TRUE, 'tracer' in 'nebula_index' data would be used to filter out Child-Nebulae: a Child-Nebula without any 'feature' being marked as 'tracer', this Child-Nebula would be filtered out. See create_nebula_index().

See Also

compute_spectral_similarity(), create_features_annotation(), create_nebula_index(), igraph::graph_from_data_frame().

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)

  ## default parameters
  create_child_nebulae()

  test1 <- create_child_nebulae(test1, 0.01)
  ## see results
  igraph(child_nebulae(test1))
  ## write output for 'Cytoscape' or other network software
  tmp <- paste0(tempdir(), "/child_nebulae/")
  dir.create(tmp)
  res <- igraph(child_nebulae(test1))
  lapply(
    names(res),
    function(name) {
```

```
      igraph::write_graph(
        res[[name]],
        file = paste0(tmp, name, ".graphml"),
        format = "graphml"
      )
    }
  )
  list.files(tmp)

  unlink(tmp, T, T)

## End(Not run)
```

`create_features_annotation-methods`
merge annotation for 'features'

Description

According to `specific_candidate(object)` data, merge the latest filtered chemical formulae annotation, structural annotation. The ion mass and retention time for each 'feature' would also be gathered. User can also pass custom annotation for each 'feature', as long as the 'data.frame' with column of '.features_id'.

Usage

```
## S4 method for signature 'mcnebula,data.frame,numeric'
create_features_annotation(x, extra_data, column)

## S4 method for signature 'mcnebula,data.frame,missing'
create_features_annotation(x, extra_data)

## S4 method for signature 'mcnebula,missing,missing'
create_features_annotation(x)
```

Arguments

`x`	mcnebula object.
`extra_data`	data.frame.
`column`	numeric(1). If name of columns not contain ".features_id", used to specify ID column for 'features'.

Details

The 'features_annotation' data created from:

- The 'specific_candidate' data: `specific_candidate(object)`
- The filtered chemical formula data: `latest(object, subscript = ".f2_formula")`

- The filtered structural data: `latest(object, subscript = ".f3_fingerid")`
- The ion mass and retention time (m/z and RT): latest(object, "project_dataset", ".f2_info")

The last would be collated via: `collate_data(object, subscript = ".f2_info")`

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference=T)
  test1 <- create_stardust_classes(test1)

  test1 <- create_features_annotation(test1)
  ## see results
  features_annotation(test1)
  ## or
  reference(test1)$features_annotation
  ## or
  reference(mcn_dataset(test1))$features_annotation

  ## merge additional data
  ids <- features_annotation(test1)$.features_id
  data <- data.frame(.features_id = ids, quant. = rnorm(length(ids), 1000, 200))
  test1 <- create_features_annotation(test1, extra_data = data)

## End(Not run)
```

`create_hierarchy-methods`

Create hierarchy data of chemical classification

Description

Methods used to create hierarchy data of chemical classification. Annotate all chemical classes with hierarchy number.

`create_hierarchy()`: get the default parameters for the method `create_hierarchy`.

`create_hierarchy(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_hierarchy`.

`get_parent_classes`: For chemical classes to get its parent chemical classes.

Usage

```
## S4 method for signature 'missing,missing'
create_hierarchy()

## S4 method for signature 'mcnebula,ANY'
create_hierarchy(x, fun_organize)

## S4 method for signature 'mcnebula,`function`'
create_hierarchy(x, fun_organize)

get_parent_classes(classes, x, hierarchy_cutoff = 3, re_class_no_parent = F)
```

Arguments

x	mcnebula object.
fun_organize	function. Normally not used. Default is `MCnebula2:::.build_hierarchy`.
classes	character. Names of chemical classes.
re_class_no_parent	logical(1). If `TRUE`, once a chemical class find with no parent, the chemical class itself would be returned.
hierarchy_cutoff.	numeric(1). The highest hierarchy of parent chemical classes that needs to be searched.

`create_nebula_index-methods`

Set down the chemical classes for visualization

Description

Arrange the filtered 'stardust_classes' data as 'nebula_index' data. The chemical classes in 'nebula_index' data would be visualized as Child-Nebulae. Run after `cross_filter_stardust()`.

`create_nebula_index()`: get the default parameters for the method `create_nebula_index`.

`create_nebula_index(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_nebula_index`.

Usage

```
## S4 method for signature 'missing,missing'
create_nebula_index()

## S4 method for signature 'mcnebula,ANY'
create_nebula_index(x, force)

## S4 method for signature 'mcnebula,logical'
create_nebula_index(x, force)
```

Arguments

x	mcnebula object.
force	logical. The number of chemical classes in 'stardust_classes' data would be checked. The maximum is 120. If there were too many classes, return with error. Set to FALSE, escape from maximum check.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)

  test1 <- create_nebula_index(test1)
  ## see results
  nebula_index(test1)
  ## or
  reference(test1)$nebula_index
  ## or
  reference(mcn_dataset(test1))$nebula_index

## End(Not run)
```

---

create_parent_layout-methods

Create layout for visualization of Parent-Nebula

Description

These methods use functions of `tidygraph::as_tbl_graph()` and `ggraph::create_layout()` to create 'layout_ggraph' (data.frame) object to standby visualization of Parent-Nebula. Run after `create_parent_nebula()`.

`create_parent_layout()`: get the default parameters for the method `create_parent_layout`.

`create_parent_layout(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_parent_layout`.

Usage

```
## S4 method for signature 'missing,missing,missing'
create_parent_layout()
```

```
## S4 method for signature 'mcnebula,ANY,ANY'
create_parent_layout(x, ggraph_layout, seed)

## S4 method for signature 'mcnebula,character,numeric'
create_parent_layout(x, ggraph_layout, seed)
```

Arguments

x	mcnebula object.
ggraph_layout	character(1). Layout name. See ggraph::create_layout().
seed	numeric(1). Passed to set.seed().

See Also

tidygraph::as_tbl_graph(), ggraph::create_layout().

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_parent_nebula(test1, 0.01)

  ## default parameters
  create_parent_layout()

  test1 <- create_parent_layout(test1)
  ## see results (a object for 'ggraph' package to visualization)
  tibble::as_tibble(layout_ggraph(parent_nebula(test1)))

## End(Not run)
```

---

create_parent_nebula-methods

Gather data to create Parent-Nebula

Description

Gather 'spectral_similarity' data and 'features_annotation' data to create 'igraph' object use function of `igraph::graph_from_data_frame()`.

`create_parent_nebula()`: get the default parameters for the method `create_parent_nebula`.

`create_parent_nebula(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_parent_nebula`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing'
create_parent_nebula()

## S4 method for signature 'mcnebula,ANY,ANY,ANY'
create_parent_nebula(x, edge_cutoff, max_edge_number, remove_isolate)

## S4 method for signature 'mcnebula,numeric,numeric,logical'
create_parent_nebula(x, edge_cutoff, max_edge_number, remove_isolate)
```

Arguments

`x`	mcnebula object.
`edge_cutoff`	numeric(1). Value in (0,1). Set a threshold to create edges upon similarity value of 'spectral_similarity' data.
`max_edge_number`	numeric(1). For nodes (features) in each Parent-Nebulae (i.e. network), the maximum number of edges link with. If the number exceeds the limitation, only edges representing higher spectral similarity would be retained.
`remove_isolate`	logical. If `TRUE`, remove the isolate 'features' (in network, i.e. the nodes without edge)

See Also

`compute_spectral_similarity()`, `create_features_annotation()`, `igraph::graph_from_data_frame()`.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
```

```
  ## default parameters
  create_parent_nebula()

  test1 <- create_parent_nebula(test1, 0.01)
  ## see results
  igraph(parent_nebula(test1))
  ## write output for 'Cytoscape' or other network software
  tmp <- tempdir()
  igraph::write_graph(
    igraph(parent_nebula(test1)),
    file = paste0(tmp, "/parent_nebula.graphml",
      format = "graphml"
    )
  )

  unlink(tmp, T, T)

## End(Not run)
```

`create_reference-methods`

Establish 'specific candidate' for each 'feature'

Description

According to the filtered data, whether obtained by `filter_formula()`, `filter_structure()` or `filter_ppcp()`, establishing specific candidate of each 'feature' for subsequent data filtering. This step is an important intermediate link for the three part of data filtering, makes the final filtered results of chemical formula, structure and classification consistent.

`create_reference()`: get the default parameters for the method `create_reference`.

Usage

```
## S4 method for signature 'mcnebula,ANY,ANY,ANY,ANY,logical,ANY'
create_reference(x, from, subscript, data, columns, fill, MoreArgs)

## S4 method for signature
## 'missing,missing,missing,missing,missing,missing,missing'
create_reference()

## S4 method for signature
## 'mcnebula,missing,missing,missing,missing,missing,missing'
create_reference(x)

## S4 method for signature
## 'mcnebula,character,missing,missing,missing,missing,missing'
create_reference(x, from)
```

```
## S4 method for signature
## 'mcnebula,missing,character,missing,missing,missing,missing'
create_reference(x, subscript)

## S4 method for signature
## 'mcnebula,missing,missing,data.frame,character,missing,missing'
create_reference(x, data, columns)

## S4 method for signature
## 'mcnebula,missing,missing,data.frame,integer,missing,missing'
create_reference(x, data, columns)

## S4 method for signature
## 'mcnebula,missing,missing,data.frame,missing,missing,missing'
create_reference(x, data)
```

Arguments

x	mcnebula object.
from	character(1). "structure", "formula" or "ppcp".
subscript	character(1). ".f3_fingerid", ".f2_formula" or ".f3_canopus". See subscript.
data	data.frame. An external channel for user to specify candidate customarily. Normally not used.
columns	character(2) or numeric(2). Specify the key columns in the parameter of data. Normally not used.
fill	logical. If TRUE, run post modification. Run filter_formula(object), and use its results to fill the data specific_candidate for 'features' without specified top candidate. Only useful when the data specific_candidate were based on scores of chemical structure or classes, as for some 'features' there may be no chemical structural or classified candidates but candidates for chemical formula.
MoreArgs	list. Used only fill = T. Parameters passed to filter_formula().

Details

Establish reference upon top candidate Suppose we predicted a potential compound represented by LC-MS/MS spectrum, and obtained the candidates of chemical molecular formula, structure and chemical class. These candidates include both positive and negative results: for chemical molecular formula and chemical structure, the positive prediction was unique; for chemical class, multiple positive predictions that belong to various classification were involved. We did not know the exact negative and positive. Normally, we ranked and filtered these according to the scores. There were numerious scores, for isotopes, for mass error, for structural similarity, for chemical classes... Which score selected to rank candidates depends on the purpose of research. Such as:

- To find out the chemical structure mostly be positive, ranking the candidates by structural score.
- To determine whether the potential compound may be of a certain chemical classes, ranking the candidates by the classified score.

Ether by filter_formula(), filter_structure() or filter_ppcp(), the candidate with top score can be obtained. However, for the three module (formula, structure, classes), sometimes thier top score candidates were not in line with each other. That is, thier top score towards different chemical molecular formulas. To find out the corresponding data in other modules, create_reference should be performed to establish the 'specific_candidate' for subsequent filtering.

Examples

```
## Not run:
  test <- mcn_5features

  ## set specific candidate
  ## -------------------------------------
  ## from chemical structure
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  ## see results
  specific_candidate(test1)
  ## or
  reference(test1)$specific_candidate
  ## or
  reference(mcn_dataset(test1))$specific_candidate
  ## 'create_reference(test1)' equals to
  test1 <- create_reference(test1, from = "structure", fill = T)
  e1 <- specific_candidate(test1)

  ## the above equals to following:
  data <- latest(filter_structure(test1))
  test1 <- create_reference(test1, data = data, fill = T)
  e2 <- specific_candidate(test1)
  identical(e1, e2)

  ## the 'specific_candidate' data used for filtering
  test1 <- filter_formula(test1, by_reference = T)

  ## -------------------------------------
  ## from chemical formula
  test1 <- filter_formula(test1)
  test1 <- create_reference(test1, from = "formula")

  ## -------------------------------------
  ## from chemical classes
  ## A complex example:
  ## suppose there were some classes we were interested in
  all_classes <- latest(test1, "project_dataset", ".canopus")$class.name
  set.seed(1)
  classes <- sample(all_classes, 50)
  classes
  test1 <- filter_ppcp(test1,
    dplyr::filter,
    class.name %in% classes,
    pp.value > 0.5,
```

```
  by_reference = F
)
data <- latest(test1)
data
## 'feature' have a plural number of candidates.
ids <- data$.features_id
id <- unique(ids[duplicated(ids)])
## get the candidate of top chemical structural score.
`%>%` <- magrittr::`%>%`
candidates <- filter_structure(test1, dplyr::filter, .features_id %in% id) %>%
  latest() %>%
  dplyr::filter(.candidates_id %in% data$.candidates_id) %>%
  dplyr::arrange(.features_id, dplyr::desc(csi.score)) %>%
  dplyr::distinct(.features_id, .keep_all = T)
## for refecrence
data <- data %>%
  dplyr::filter(
    .features_id != candidates$.features_id |
      (.features_id == candidates$.features_id &
        .candidates_id == candidates$.candidates_id)
  )
test1 <- create_reference(test1, data = data, fill = T)
specific_candidate(test1)


## End(Not run)
```

`create_stardust_classes-methods`
'Inner' filter for PPCP data

Description

Perform 'inner' filter for PPCP (posterior probability of classification prediction) data of each 'feature', then gathered as 'stardust_classes' data. Run after `create_reference()`. Standby for next step `cross_filter_stardust()`.

`create_stardust_classes()`: get the default parameters for the method `create_stardust_classes`.

`create_stardust_classes(x, ...)`: use the default parameters whatever 'missing' while performing the method `create_stardust_classes`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing,missing'
create_stardust_classes()

## S4 method for signature 'mcnebula,ANY,ANY,ANY,ANY'
create_stardust_classes(
  x,
```

```
  pp.threshold,
  hierarchy_priority,
  position_isomerism,
  inherit_dataset
)

## S4 method for signature 'mcnebula,numeric,numeric,logical,logical'
create_stardust_classes(
  x,
  pp.threshold,
  hierarchy_priority,
  position_isomerism,
  inherit_dataset
)
```

Arguments

x	mcnebula object.
pp.threshold	numeric(1) Threshold for PPCP. pp.threshold = 0.5 may work well.
hierarchy_priority	numeric. The specified hierarchy of classes to retain. The other hierarchy would be filtered out. The hierarchy: • n: ... • 5: Classes of Level 5. • 4: Classes of Subclass. • 3: Classes of Class. • 2: Classes of Super Class. • ...
position_isomerism	logical. If TRUE, use pattern match to filter out all classes names contains Arabic numerals. Generally, these classes describe about the position of chemical functional group, which were too subtle for machine to predict from LC-MS/MS spectrum.
inherit_dataset	logical. If TRUE, use latest PPCP data formed by filter_ppcp(). i.e., data of: • latest(x, subscript = ".f3_canopus") Else, run filter_ppcp().

Details

The PPCP data for each 'feature' contains the prediction of thousands of classes for the potential compound (even if the chemical structure was unknown). See http://www.nature.com/articles/s41587-020-0740-8 for details about the prediction. The data contains attributes of:

- class.name: name of classes.
- pp.value: value of posterior probability.

- hierarchy: hierarchy of classes in the taxonomy. See https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0174-y for details about hierarchy and taxonomy of chemical classification.
- ...

The method create_stardust_classes() use these inner attributes to filter classes candidates for each 'feature'.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)

  test1 <- create_stardust_classes(test1)
  ## see results
  stardust_classes(test1)
  ## or
  reference(test1)$stardust_classes
  ## or
  reference(mcn_dataset(test1))$stardust_classes

  ## the default parameters
  create_stardust_classes()

## End(Not run)
```

---

cross_filter_stardust-methods

'Cross' filter for 'stardust_classes' data

Description

'Cross' filter for 'stardust_classes' data. Use 'features_annotation' data and 'stardust_classes' data for chemical classes filtering. Run after create_stardust_classes(). Methods cross_filter_stardust are integration of the following three method:

- cross_filter_quantity
- cross_filter_score
- cross_filter_identical

cross_filter_stardust(): get the default parameters for the method cross_filter_stardust.

cross_filter_stardust(x, ...): use the default parameters whatever 'missing' while performing the method cross_filter_stardust.

cross_filter_quantity: Filter chemical classes in 'stardust_classes' data according to:

- Absolute quantity: the classified 'features' of the class.
- Relative proportion: the classified 'features' of the class comparing with all features of all classes.

`cross_filter_score` Filter chemical classes in 'stardust_classes' data according to the attributes in 'features_annotation' data. Set cut-off value of attributes for all 'features', then inspect overall satisfaction of the classified 'features' of the class.

`cross_filter_identical` Filter chemical classes in 'stardust_classes' data by comparing the classified 'features'.

Usage

```
## S4 method for signature 'missing'
cross_filter_stardust()

## S4 method for signature 'mcnebula'
cross_filter_stardust(
  x,
  min_number,
  max_ratio,
  types,
  cutoff,
  tolerance,
  hierarchy_range,
  identical_factor
)

## S4 method for signature 'mcnebula,numeric,numeric'
cross_filter_quantity(x, min_number, max_ratio)

## S4 method for signature 'mcnebula,character,numeric,numeric'
cross_filter_score(x, types, cutoff, tolerance)

## S4 method for signature 'mcnebula,numeric,numeric'
cross_filter_identical(x, hierarchy_range, identical_factor)
```

Arguments

`x`	mcnebula object.
`min_number`	numeric(1). Value in (1,). For classified 'features' of chemical classes, minimum quantity.
`max_ratio`	numeric(1). Value in (0, 1]. For classified 'features' of chemical classes, maximum proportion: the 'features' quantity versus all 'features' (unique) quantity of all classes.
`types`	character. The target attributes for Goodness assessment. See details. There can be plural ones.

`cutoff` numeric. For Goodness assessment of target attributes. The size of the value depends on the target attribute. Note, the `cutoff` must be 'vector' the same length as `types`.

`tolerance` numeric. Value in (0, 1). For Goodness assessment of target attributes. The thresholds of Goodness. Note, the `tolerance` must be 'vector' the same length as `types`.

`hierarchy_range`
numeric(2). The hierarchy range of chemical classification passed for similarity assessment of chemical classes. The hierarchy:

- 10: ...
- n: ...
- 5: Classes of Level 5.
- 4: Classes of Subclass.
- 3: Classes of Class.
- 2: Classes of Super Class.
- 1: ...
- 0: ...

`identical_factor`
numeric(1). Value in (0, 1). Threshold value for classes similarity assessment.

Details

Compared to the chemical class filtering within PPCP data by `create_stardust_classes()`, the filtering within 'stardust_classes' data by `cross_filter_stardust()` is fundamentally different.

- For `create_stardust_classes()`, the PPCP data belongs to each 'feature'. When performing the filtering, only simple threshold conditions or absolute conditions are set to filter the chemical classes; there is no crossover between the different attributes and no crossover between the 'features'. Therefore, we consider this as 'inner' filtering.
- For `cross_filter_stardust()`, the data of the chemical classes and their classified 'features', i.e. 'stardust_classes' data, were combined and then grouped upon the chemical classes. After grouping, each chemical class has a certain quantity of "features". When filtering, statistics may be performed on 'features' data within a group; statistics may be performed on these data in conjunction with 'features_annotation' data; and statistics may be performed to compare groups with each other. As its crossover, we consider this as 'cross' filtering.

Cross_filter_quantity Set 'features' quantity limitation for each group. The groups with too many 'features' or too few 'features' would be filtered out. This means the chemical class would be filtered out. These thresholds are about:

- Minimum quantity: the 'features'.
- Maximum proportion: the 'features' quantity versus all 'features' (unique) quantity of all groups.

The purpose of this step is to filter out chemical classes that have too large or too subtle a conceptual scope. For example, 'Organic compounds', which covers almost all compounds that can be detected in metabolomics data, is too large in scope to be of any help to our biological research. The setting of

parameters is not absolute, and there is no optimal solution. Users can draw up thresholds according to the necessity of the study.

Cross_filter_score This step associate 'stardust_classes' data with 'features_annotation' data. For each group, the Goodness assessment is performed for each target attribute (continuous attribute, generally be a scoring attribute of compound identification, such as 'tani.score'). If the group met all the expected Goodness, the chemical class would be retained; otherwise, the chemical class would be filtered out. The Goodness (G) related with the 'features' within the group:

- n: the quantity of 'features' of which target attributes satisfied with the cut-off.
- N: the quantity of all 'features'.

The Goodness: G = n / N.

The assessment of Goodness is related to the parameters of `cutoff` and `tolerance`:

- Expected Goodness, i.e. value of `tolerance`.
- Actual Goodness, related to parameter `cutoff`. G = n / N.

Goodness assessment can be given to plural target attributes. Note that the chemical class would retained only if it passed the Goodness assessment of all target attributes.

The main purpose of this step is to filter out those chemical classes with too many 'features' of low structural identification.

Cross_filter_identical A similarity assessment of chemical classes. Set a hierarchical range for chemical classification and let groups (each group, i.e. a chemical class with its classified 'features') within this range be compared for similarity to each other. For two groups, if the classified 'features' almost identical to each other, the chemical class represented by one of the groups would be discarded. The assessment of identical degree of two groups (A and B):

- x: ratio of the classified 'features' of A belonging to B
- y: ratio of the classified 'features' of B belonging to A
- i: value of parameter `identical_factor`

If x > i and y > i, the two groups would be considered as identical. Then the group with fewer 'features' would be discarded.

The purpose of this step is to filter out classes that may incorporate each other and are similar in scope. The in silico prediection approach may not be able to distinguish which class the potential compound belongs to from the LC-MS/MS spectra.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
```

```
## the default parameters
cross_filter_stardust()
# This is a simulated dataset with only 5 'features',
# so the default parameters are meaningless for it.

# Note that real datasets often contain thousands of "features"
# and the following 'min_number' and 'max_ratio' parameter values are not suitable.
test1 <- cross_filter_stardust(
  test1,
  min_number = 2,
  max_ratio = 1
)
## see results
stardust_classes(test1)
## or
reference(test1)$stardust_classes
## or
reference(mcn_dataset(test1))$stardust_classes

e1 <- stardust_classes(test1)

## see the filtered classes
backtrack_stardust(test1)

## reset the 'stardust_classes'
test1 <- create_stardust_classes(test1)

## customized filtering
# Note that real datasets often contain thousands of "features"
# and the following 'min_number' and 'max_ratio' parameter values are not suitable.
test1 <- cross_filter_quantity(test1, min_number = 2, max_ratio = 1)
test1 <- cross_filter_score(test1,
  types = "tani.score",
  cutoff = 0.3,
  tolerance = 0.6
)
test1 <- cross_filter_identical(
  test1,
  hierarchy_range = c(3, 11),
  identical_factor = 0.7
)
e2 <- stardust_classes(test1)

identical(e1, e2)

## reset
test1 <- create_stardust_classes(test1)
## targeted plural attributes
test1 <- cross_filter_stardust(
  test1,
  min_number = 2,
  max_ratio = 1,
  types = c("tani.score", "csi.score"),
```

```
    cutoff = c(0.3, -150),
    tolerance = c(0.6, 0.3)
  )

## End(Not run)
```

`dataset-class` *Share slots and methods for classes inherite from VIRTUAL_dataset*

Description

This VIRTUAL class provides a slot for storing data and methods for accessing data in slot.

`dataset`, `dataset<-`: getter and setter for the `dataset` slot of the object.

Usage

```
## S4 method for signature 'ANY'
dataset(x)

## S4 replacement method for signature 'ANY'
dataset(x) <- value
```

Arguments

`value`	The value for the slot.

Slots

`dataset` list with names (subscript, imply file names).

See Also

Other datasets: `mcn_dataset-class`, `project_dataset-class`

`draw_nodes-methods` *Draw and visualize chemcial structures for Child-Nebulae*

Description

Methods used for drawing and visualizing nodes of 'features' in Child-Nebulae (networks). The methods used to visualize 'features' with annotations of:

- chemical structures
- chemical classification
- quantification data (peak area)

- ID of 'feature' (.features_id)

`draw_nodes()`: get the function for generating default parameters for the method `draw_nodes`.

`draw_nodes(x, ...)`: use the default parameters whatever 'missing' while performing the method `draw_nodes`.

`show_node()`: get the default parameters for the method `show_node`.

Visualize the node of 'feature' which has been drawn by methods `draw_nodes()` (or drawn by methods `annotate_nebula()`).

`show_node(x, ...)`: use the default parameters whatever 'missing' while performing the method `show_node`.

`ggset_activate_nodes`: create the ggset object of node of specified 'feature'.

Usage

```
## S4 method for signature
## 'missing,missing,missing,missing,missing,missing,missing'
draw_nodes()

## S4 method for signature 'mcnebula,character,ANY,ANY,ANY,ANY,ANY'
draw_nodes(
  x,
  nebula_name,
  nodes_color,
  add_id_text,
  add_structure,
  add_ppcp,
  add_ration
)

## S4 method for signature
## 'mcnebula,character,character,logical,logical,logical,logical'
draw_nodes(
  x,
  nebula_name,
  nodes_color,
  add_id_text,
  add_structure,
  add_ppcp,
  add_ration
)

## S4 method for signature 'missing,missing,missing,missing'
show_node()

## S4 method for signature 'ANY,character,ANY,ANY'
show_node(x, .features_id, panel_viewport, legend_viewport)

ggset_activate_nodes(
```

```
  x,
  .features_id,
  nodes_color = "#FFF9F2",
  add_ppcp = T,
  add_ration = T
)
```

Arguments

x	mcnebula object.
nebula_name	character(1). Chemical classes in 'nebula_index' data. Specified to draw nodes (of network) of all the 'features' of that.
nodes_color	character with names or not. The Value is Hex color. Specified colors for 'features' to draw nodes. If the number of the colors were not enough, the rest 'features' would be fill with default color. If `set_tracer()` has been run, the colors specified in 'nebula_index' would be used preferentially.
add_id_text	logical. If TRUE, add ID (.features_id) for 'features' inside the nodes.
add_structure	logical. If TRUE, draw chemical structures inside the nodes. See `draw_structures()`.
add_ppcp	logical. If TRUE, draw radical bar plot inside the nodes for annotation of PPCP data. See `set_ppcp_data()` for custom modify the annotated PPCP data. Hex colors in `palette_col(object)` would be used for fill the bar plot (Used by `ggplot2::scale_fill_manual()`).
add_ration	logical. If TRUE, draw ring plot inside the nodes for annotation of features quantification data. See `set_ration_data()` for custom modify the annotated quantification data. Hex colors in `palette_stat(object)` would be used for fill be ring plot.
.features_id	character(1). ID of 'feature' to show node.
panel_viewport	'viewport' object. Create by `grid::viewport()`.
legend_viewport	'viewport' object.

Details

Those annotated visualizations are drawn in steps and then are put together. In order to render the text as a graphical path (otherwise, the graphics would not be compatible with too small fonts and would result in misplaced text), the 'ggplot' object or 'grob' object is first exported as an SVG file, which is subsequently read by `grImport2::readPicture()`, followed by `grImport2::grobify()` as 'grob' object, and then combined into the final 'grob'. In general, this process is time consuming, especially when there are a lot of 'features' for visualization.

See Also

`grid::grid.draw()`, `grid::grob()`, `grImport2::readPicture()`, `grImport2::grobify()`...

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)

  ## set features quantification data
  ids <- features_annotation(test1)$.features_id
  quant. <- data.frame(
    .features_id = ids,
    sample_1 = rnorm(length(ids), 1000, 200),
    sample_2 = rnorm(length(ids), 2000, 500)
  )
  metadata <- data.frame(
    sample = paste0("sample_", 1:2),
    group = c("control", "model")
  )
  features_quantification(test1) <- quant.
  sample_metadata(test1) <- metadata

  ## optional 'nebula_name'
  visualize(test1)
  ## a class for example
  class <- visualize(test1)$class.name[1]
  tmp <- export_path(test1)
  test1 <- draw_structures(test1, class)
  test1 <- draw_nodes(test1, class)

  ## see results
  grobs <- nodes_grob(child_nebulae(test1))
  grobs
  grid::grid.draw(grobs[[1]])
  ## visualize with ID of 'feature' (.features_id)
  ## with legend
  ids <- names(grobs)
  x11(width = 9, height = 5)
  show_node(test1, ids[1])

  ## default parameters
  draw_nodes()
```

```
  unlink(tmp, T, T)

## End(Not run)
```

---

draw_structures-methods

Draw and visualize chemcial structure

Description

Methods used for drawing and visualizing chemical structures of 'features' in Child-Nebulae. ChemmineOB::convertToImag is the core function used for drawing chemical structures.

show_structure: visualize the chemical structure of 'feature' which has been drawn.

Usage

```
## S4 method for signature 'mcnebula,character,missing,missing'
draw_structures(x, nebula_name, .features_id, data, ...)

## S4 method for signature 'mcnebula,missing,character,missing'
draw_structures(x, nebula_name, .features_id, data, ...)

## S4 method for signature 'mcnebula,missing,missing,data.frame'
draw_structures(x, nebula_name, .features_id, data, ...)

## S4 method for signature 'ANY,character'
show_structure(x, .features_id)
```

Arguments

x	mcnebula object.
nebula_name	character(1). Chemical classes in 'nebula_index' data. Specified to draw chemical structures of all the 'features' of that.
.features_id	character(1). The ID of 'features'.
data	data.frame. A 'data.frame' contains columns of '.features_id' and 'smiles'.
...	...

See Also

ChemmineOB::convertToImage().

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)

  ## optional 'nebula_name'
  visualize(test1)
  ## a class for example
  class <- visualize(test1)$class.name[1]
  tmp <- export_path(test1)
  test1 <- draw_structures(test1, class)

  ## see results
  grobs <- structures_grob(child_nebulae(test1))
  grobs
  grid::grid.draw(grobs[[1]])
  ## visualize with ID of 'feature' (.features_id)
  ids <- names(grobs)
  show_structure(test1, ids[1])

  unlink(tmp, T, T)

## End(Not run)
```

`export-class` *Share slots and methods for classes inherite from VIRTUAL_export*

Description

This VIRTUAL class provides slots for recording export path and export name of attributes.

`export_name`, `export_name<-`: getter and setter for the `export_name` slot of the object.

`export_path`, `export_path<-`: getter and setter for the `export_path` slot of the object.

Usage

```
## S4 method for signature 'ANY'
export_name(x)
```

```
## S4 replacement method for signature 'ANY'
export_name(x) <- value

## S4 method for signature 'ANY'
export_path(x)

## S4 replacement method for signature 'ANY'
export_path(x) <- value
```

Arguments

x	object inherit class export.
value	The value for the slot.

Slots

export_path character(1). The export directory path.

export_name character with names. While export, the attribute name will be converted to the value.

---

filter_formula-methods

Collate and filter candidates of chemical formula for each 'feature'

Description

This methods provide an approach to collate and filter chemical formula candidates data in baches for each 'feature'.

filter_formula(): get the default parameters for the method filter_formula.

filter_formula(x, ...): use the default parameters whatever 'missing' while performing the method filter_formula.

Usage

```
## S4 method for signature 'missing,missing,missing'
filter_formula()

## S4 method for signature 'mcnebula,ANY,ANY'
filter_formula(x, fun_filter, ..., by_reference)

## S4 method for signature 'mcnebula,`function`,logical'
filter_formula(x, fun_filter, ..., by_reference)
```

Arguments

x	mcnebula object.
fun_filter	function. Used to filter data.frame. The function would run for candidates data (data.frame) for each 'features'. Such as: • `lapply(split(all_data, ~.features_id), fun_filter, ...)`. This parameter provides an elegant and flexible way to filter data. Users can pass function `dplyr::filter()` to specify any attributes condition to filter the data.
...	Other parameters passed to the function `fun_filter`.
by_reference	logical. Use `specific_candidate(object)` data to filter candidates data. See `create_reference()`.

Details

In SIRIUS project directory, if the computation job has done, each 'feature' has multiple prediction candidates whether for chemical formula, structure, or classification. This method provides an approach to collate and filter these data in baches. See MCnebula2 for details of chemical formula, structure and classification.

Examples

```
## Not run:
  test <- mcn_5features

  ## filter chemical formula candidates
  ## use default parameters
  test1 <- filter_formula(test)
  latest(test1)

  ## the default parameters:
  filter_formula()

  ## customized filtering
  ## according to score
  test1 <- filter_formula(test1, dplyr::filter, zodiac.score > 0.5)
  latest(test1)

  ## get top rank
  test1 <- filter_formula(test1, dplyr::filter, rank.formula <= 3)
  latest(test1)

  ## complex filtering
  test1 <- filter_formula(
    test1, dplyr::filter,
    ## molecular formula
    !grepl("N", mol.formula),
    ## mass error
    abs(error.mass) < 0.001
  )
```

```
  latest(test1)

  ## select columns
  test1 <- filter_formula(test1, dplyr::select, 1:5)
  latest(test1)

## End(Not run)
```

`filter_ppcp-methods` *Collate and filter candidates of chemical classification for each 'feature'*

Description

This methods provide an approach to collate and filter chemical classification candidates data in baches for each 'feature'.

`filter_ppcp()`: get the default parameters for the method `filter_ppcp`.

`filter_ppcp(x, ...)`: use the default parameters whatever 'missing' while performing the method `filter_ppcp`.

Usage

```
## S4 method for signature 'missing,missing,missing'
filter_ppcp()

## S4 method for signature 'mcnebula,ANY,ANY'
filter_ppcp(x, fun_filter, ..., by_reference)

## S4 method for signature 'mcnebula,`function`,logical'
filter_ppcp(x, fun_filter, ..., by_reference)
```

Arguments

`x`	mcnebula object.
`fun_filter`	function. Used to filter data.frame. The function would run for candidates data (data.frame) for each 'features'. Such as: • `lapply(split(all_data, ~.features_id), fun_filter, ...)`. This parameter provides an elegant and flexible way to filter data. Users can pass function `dplyr::filter()` to specify any attributes condition to filter the data.
`...`	Other parameters passed to the function `fun_filter`.
`by_reference`	logical. Use `specific_candidate(object)` data to filter candidates data. See `create_reference()`.

Details

Filter for PPCP (posterior probability of classification prediction) data. See details about classification prediction for compounds: http://www.nature.com/articles/s41587-020-0740-8. See other details in filter_formula().

Examples

```
## Not run:
  test <- mcn_5features

  ## filter chemical class candidates
  ## the default parameters:
  filter_ppcp()

  ## if 'by_reference' set with TRUE, 'create_reference' should be
  ## run previously.
  test1 <- filter_ppcp(test, by_reference = F)
  latest(test1)

  ## customized filtering
  ## according to score
  test1 <- filter_ppcp(test1, dplyr::filter, pp.value > 0.5,
                       by_reference = F)
  latest(test1)

  ## complex filtering
  test1 <- filter_ppcp(
    test1, dplyr::filter,
    ## PPCP value
    pp.value > 0.5,
    ## speicifid class
    class.name %in% c("Azoles"),
    by_reference = F
  )
  latest(test1)

  ## select columns
  test1 <- filter_ppcp(test1, dplyr::select, 1:5,
                       by_reference = F)
  latest(test1)

## End(Not run)
```

---

filter_structure-methods

Collate and filter candidates of chemical structure for each 'feature'

Description

This methods provide an approach to collate and filter chemical structure candidates data in baches for each 'feature'.

`filter_structure()`: get the default parameters for the method `filter_structure`.

`filter_structure(x, ...)`: use the default parameters whatever 'missing' while performing the method `filter_structure`.

Usage

```
## S4 method for signature 'missing,missing,missing'
filter_structure()

## S4 method for signature 'mcnebula,ANY,ANY'
filter_structure(x, fun_filter, ..., by_reference)

## S4 method for signature 'mcnebula,`function`,logical'
filter_structure(x, fun_filter, ..., by_reference)
```

Arguments

x	mcnebula object.
fun_filter	function. Used to filter data.frame. The function would run for candidates data (data.frame) for each 'features'. Such as: • `lapply(split(all_data, ~.features_id), fun_filter, ...)`. This parameter provides an elegant and flexible way to filter data. Users can pass function `dplyr::filter()` to specify any attributes condition to filter the data.
...	Other parameters passed to the function `fun_filter`.
by_reference	logical. Use `specific_candidate(object)` data to filter candidates data. See `create_reference()`.

Details

See details in `filter_formula()`.

Examples

```
## Not run:
  test <- mcn_5features

  ## filter chemical structure candidates
  ## use default parameters
  test1 <- filter_structure(test)
  latest(test1)

  ## the default parameters:
  filter_structure()
```

```
  ## customized filtering
  ## according to score
  test1 <- filter_structure(test1, dplyr::filter, tani.score > 0.4)
  latest(test1)

  ## get top rank
  test1 <- filter_structure(test1, dplyr::filter, rank.structure <= 3)
  latest(test1)

  ## complex filtering
  test1 <- filter_structure(
    test1, dplyr::filter,
    ## molecular formula
    !grepl("N", mol.formula),
    ## Tanimoto similarity
    tani.score > 0.4
  )
  latest(test1)

  ## select columns
  test1 <- filter_structure(test1, dplyr::select, 1:5)
  latest(test1)

## End(Not run)
```

`fun_modify` *Modify 'ggset' object*

Description

These are multiple functions used for post modification of ggset object. These functions provide a convenient, fast, and repeatable way to make improvements to ggset object.

`modify_default_child`: Used for `visualize_all()`. `modify_rm_legend` + `modify_set_labs` + `modify_unify_scale_limits`. In addition, if the 'use_tracer' is TRUE (see `set_nodes_color()`), `modify_tracer_node` and `modify_color_edge` would be performed.

`modify_stat_child`: Repalce `scale_fill_gradientn()` with `scale_fill_gradient2()` in 'layers'; unify the "aes" scale except for "fill"; perfrom `modify_set_labs()`; only keep the legend for 'fill', and adjust its width; move the position of the legend to the bottom; remove the title of the legend.

`modify_set_labs_and_unify_scale_limits`: `modify_set_labs` + `modify_unify_scale_limits`

`modify_annotate_child`: `modify_set_labs` + ... (for parameters of `panel.grid` and `panel.background` in `ggplot2::theme()`).

`modify_rm_legend`: remove the legend. For parameter of `legend.position` in `ggplot2::theme()`.

`modify_tracer_node`: Set the stroke for nodes in Nebulae (network) as 0, and the color as 'transparent'; Override the node color (border color) in legend.

`modify_color_edge`: Set color for edge.

`modify_set_margin`: reduce margin. For parameter of `plot.margin` in `ggplot2::theme()`.

`modify_unify_scale_limits`: Uniform mapping 'scale' for all Child-Nebulae. Related to `ggplot2::scale_*` function. Use `MCnebula2:::.LEGEND_mapping()` to get the possibly mapping.

`modify_set_labs_xy`: According to names in slot `export_name` of mcnebula object to rename the labs of x and y axis.

`modify_set_labs`: According to names in slot `export_name` of mcnebula object to rename the labs of legends.

Usage

```
modify_default_child(ggset, x)

modify_stat_child(ggset, x)

modify_set_labs_and_unify_scale_limits(ggset, x)

modify_annotate_child(ggset, x)

modify_rm_legend(ggset)

modify_tracer_node(ggset)

modify_color_edge(ggset, color)

modify_set_margin(ggset, margin = grid::unit(rep(-8, 4), "lines"))

modify_unify_scale_limits(ggset, x, aes_name = NA)

modify_set_labs_xy(ggset, x)

modify_set_labs(ggset, x)
```

Arguments

`ggset`	ggset object.
`x`	mcnebula object.
`color`	character(1).
`aes_name`	character. Specify which 'aes' to unify scale, e.g., c("fill", "size", "edge_width").

See Also

ggset

gather_sections *Quickly gather all the 'sections' in environment*

Description

gather_sections: Gathers all eligible variable names in an environment by means of Regex matches. These variables must: have a uniform character prefix, and the first character that follows must be a number. e.g., "s1", "s2", "s12.2", "s15.5.figure"...

Usage

```
gather_sections(prefix = "s", envir = parent.frame(), sort = T, get = T)
```

Arguments

prefix	character(1). The character prefix of the variable name.
envir	environment. The environment to get the variables.
sort	logical(1). If TRUE, sort the variable names according to the first number string that accompanies the prefix.
get	logical(1). If TRUE, return with a list of the value of the variables. If FALSE, return with the variable names.

ggset-class *Management for 'ggplot' visualzation*

Description

Let each packed "ggplot2" function (packed as command object) into layers in sequence, allowing post modifications programmatically and visualizing as "ggplot2" plot at any time.

show_layers: show functions and parameters in layers with a pretty and readable form.

new_ggset: Simplified creation of ggset object.

mutate_layer: Pass new parameters or modify pre-existing parameters to the packed function.

add_layers: add extra command objects into slot layers.

call_command: plot as 'ggplot' object.

Usage

```
## S4 method for signature 'ggset'
show_layers(x)

## S4 method for signature 'ANY'
new_ggset(...)

## S4 method for signature 'ggset,numeric'
mutate_layer(x, layer, ...)

## S4 method for signature 'ggset,character'
mutate_layer(x, layer, ...)

## S4 method for signature 'ggset'
add_layers(x, ...)

## S4 method for signature 'ggset'
call_command(x)
```

Arguments

x	object contains slot layers.
...	extra command objects.
layer	numeric(1) or character(1). If "character", the name must be unique in slot layers.

Slots

layers list with names. Each element of list must be a command object packed 'ggplot2' function and its args.

See Also

Other layerSets: layerSet-class, report-class

Other call_commands: code_block-class, command-class, report-class, section-class

Examples

```
## Not run:
  data <- data.frame(x = 1:10, y = 1:10)
  layer1 <- new_command(ggplot, data)
  layer2 <- new_command(geom_point, aes(x = x, y = y))
  layer3 <- new_command(labs, x = "x label", y = "y label")
  layer4 <- new_command(theme, text = element_text(family = "Times"))

  ## gather
  ggset <- new_ggset(layer1, layer2, layer3, layer4)
  ggset
  ## visualize
```

```
p <- call_command(ggset)
p

## add layers
layer5 <- new_command(
  geom_text,
  aes(x = x, y = y, label = paste0("label_", x))
)
layer6 <- new_command(ggtitle, "this is title")
ggset <- add_layers(ggset, layer5, layer6)
call_command(ggset)

## delete layers
ggset <- delete_layers(ggset, 5:6)
call_command(ggset)

## mutate layer
ggset <- mutate_layer(ggset, "theme",
  legend.position = "none",
  plot.background = element_rect(fill = "red")
)
ggset <- mutate_layer(ggset, "geom_point",
  mapping = aes(x = x, y = y, color = x)
)
call_command(ggset)

## End(Not run)
```

`history_rblock-methods`

Create 'code_block' object from history codes

Description

Get codes from R history, then formatted as code_block object.

`history_rblock()`: get the default parameters for the method `history_rblock`.

`history_rblock(x, ...)`: use the default parameters whatever 'missing' while performing the method `history_rblock`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing'
history_rblock()

## S4 method for signature 'numeric,ANY,ANY,ANY'
history_rblock(nrow, pattern_start, pattern_end, exclude)

## S4 method for signature 'numeric,missing,missing,numeric'
```

```
history_rblock(nrow, exclude)

## S4 method for signature 'missing,character,character,ANY'
history_rblock(pattern_start, pattern_end, exclude)
```

Arguments

nrow	numeric(1). The number of lines of code to fetch.
pattern_start	character(1). The pattern string used to match the starting line of codes in R history.
pattern_end	character(1). The pattern string used to match the ending line of codes in R history.
exclude	numeric(1). Used to exclude the last lines of code.

See Also

code_block, history()...

Examples

```
## Not run:
  test1 <- 1
  test2 <- 2
  test3 <- 3

  block <- history_rblock(, "^test1", "^test3")
  block

## End(Not run)
```

---

include_figure-methods

Easily embed figure into document

Description

Creates a pre-defined code_block_figure object containing the codes of knitr::include_graphics() for formatting display the figure in document.

Usage

```
## S4 method for signature 'character,character,character'
include_figure(file, name, caption)
```

Arguments

file	character(1). The path of file. See knitr::include_graphics() for the supported image formats.
name	character(1). For cross-reference in document. See https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref.
caption	character(1). Caption of figure display in document.

See Also

code_block_figure, report, knitr::include_graphics()...

Examples

```
## Not run:
  tmp <- paste0(tempdir(), "/test.pdf")
  pdf(tmp)
  plot(1:10)
  dev.off()

  fig_block <- include_figure(
    tmp, "plot", "This is caption"
  )
  fig_block

## End(Not run)
```

---

include_table-methods *Easily embed table into document*

Description

Creates a pre-defined code_block_table object containing the codes of knitr::kable() for formatting display the table in document.

Usage

```
## S4 method for signature 'data.frame,character,character'
include_table(data, name, caption)
```

Arguments

data	'data.frame' object. The data of table to display in document.
name	character(1). For cross-reference in document. See https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref.
caption	character(1). Caption of figure display in document.

Examples

```
## Not run:
  data <- data.frame(x = 1:10, y = 1:10)
  tab_block <- include_table(
    data, "table1",
    "This is caption"
  )
  tab_block

## End(Not run)
```

`initialize_mcnebula-methods` *Initialize mcnebula object*

Description

Set SIRIUS project path and its version to initialize mcnebula object. In addition, the methods can be used for some related object to given default value.

Usage

```
## S4 method for signature 'mcnebula,ANY'
initialize_mcnebula(x, sirius_version, sirius_project, output_directory)

## S4 method for signature 'melody,ANY'
initialize_mcnebula(x)

## S4 method for signature 'project_conformation,character'
initialize_mcnebula(x, sirius_version)

## S4 method for signature 'project_api,character'
initialize_mcnebula(x, sirius_version)
```

Arguments

`x`	mcnebula object, melody object, project_conformation or project_api object.
`sirius_version`	character. e.g., "sirius.v4", "sirius.v5"
`sirius_project`	character. The path of SIRIUS project space.
`output_directory`	character. The path for output.

See Also

`ggsci::pal_simpsons()`, `ggsci::pal_igv()`, `ggsci::pal_ucscgb()`, `ggsci::pal_d3()`...

Examples

```
## Not run:
  ## The raw data used for the example
  tmp <- paste0(tempdir(), "/temp_data")
  dir.create(tmp)
  eg.path <- system.file("extdata", "raw_instance.tar.gz",
                         package = "MCnebula2")

  utils::untar(eg.path, exdir = tmp)

  ## initialize 'mcnebula' object
  test <- mcnebula()
  test <- initialize_mcnebula(test, "sirius.v4", tmp)
  ## check the setting
  export_path(test)
  palette_set(test)
  ion_mode(test)
  project_version(test)

  ## initialize 'melody' object
  test <- new("melody")
  test <- initialize_mcnebula(test)
  ## check...
  palette_stat(test)

  ## initialize 'project_conformation' object
  test <- new("project_conformation")
  test <- initialize_mcnebula(test, "sirius.v4")
  ## check
  file_name(test)

  ## initialize 'project_api' object
  test <- new("project_api")
  test <- initialize_mcnebula(test, "sirius.v4")
  ## check
  methods_format(test)

  unlink(tmp, T, T)

## End(Not run)
```

`layerSet-class` *Share slots and methods for classes inherite from VIRTUAL_layerSet*

Description

This VIRTUAL class provides: slot `layers` for storing hierarchical data; and methods for modify slot `layers`.

`layers`, `layers<-`: getter and setter for the `layers` slot of the object.

add_layers: add extra "layer" into slot layers.

delete_layers: delete "layer" in slot layers.

move_layers: change the order of "layer" in slot layers.

insert_layers: Insert "layers" into the specified position (sequence) of slot layers.

Usage

```
## S4 method for signature 'layerSet'
layers(x)

## S4 replacement method for signature 'layerSet'
layers(x) <- value

## S4 method for signature 'layerSet'
show(object)

## S4 method for signature 'layerSet'
add_layers(x, ...)

## S4 method for signature 'layerSet,numeric'
delete_layers(x, layers)

## S4 method for signature 'layerSet,numeric,numeric'
move_layers(x, from, to)

## S4 method for signature 'layerSet,numeric'
insert_layers(x, to, ...)
```

Arguments

x	object contains slot layers.
value	The value for the slot.
...	extra "layer".
layers	numeric. The specified "layer" in slot layers.
from	sequence (sequence in list) of "layer" move from.
to	sequence (sequence in list) of "layer" move to.

Slots

layers list with names.

See Also

Other layerSets: ggset-class, report-class

mcnebula-class *Overall object class of MCnebula2*

Description

For analysis of MCnebula2, all data stored in this class object, all main methods performed with this object.

`latest(x, slot, subscript)`: get the data in slot (`mcn_dataset(object)` or `prject_dataset(object)`) and format as 'tbl'.

`latest()`: get the default parameters for the method `latest`.

`latest(x, ...)`: use the default parameters whatever 'missing' while performing the method `latest`.

`creation_time`, `creation_time<-`: getter and setter for the `creation_time` slot of the object.

`ion_mode`, `ion_mode<-`: getter and setter for the `ion_mode` slot of the object.

`palette_set`, `palette_gradient`, `palette_stat`, `palette_col`: fast channel to obtain the downstream slot. For `palette_set`, e.g., getter for the `palette_set` slot in sub-object of `melody` slot of the object. Equals:

- `palette_set(melody(object))`
- `palette_set(object)`.

`reference`: fast channel to obtain the downstream slot, getter for the `reference` slot in sub-object of `mcn_dataset` slot of the object. Equals:

- `reference(mcn_dataset(object))`
- `reference(object)`

`specific_candidate`, `hierarchy`, `stardust_classes`, `nebula_index`, `spectral_similarity`, `features_annotation`, `features_quantification`, `sample_metadata`: fast channel to obtain data (mostly 'tbl' or 'data.frame') inside the downstream slot ('list'). e.g., getter for the data named `specific_candidate` in `reference` slot (a 'list') in sub-object of `mcn_dataset` slot of the object. Equals:

- `reference(mcn_dataset(object))$specific_candidate`
- `specific_candidate(object)`.

`spectral_similarity<-`, `features_quantification<-`, `sample_metadata<-`: fast channel to replace data (mostly 'tbl' or 'data.frame') inside the downstream slot ('list'). e.g., setter for the data named `spectral_similarity` in `reference` slot (a 'list') in sub-object of `mcn_dataset` slot of the object. Similar:

- `reference(mcn_dataset(object))$spectral_similarity<-`
- `spectral_similarity(object)<-`.

But the latter not only replace and also validate.

`classification`: fast channel to obtain data deeply inside the downstream slot ('list'), getter for the data named `".canopus"` in `dataset` slot (a 'list') in sub-object of `project_dataset` slot of the object. Equals:

- `tibble::as_tibble(entity(dataset(project_dataset(object))$.canopus))`
- `classification(object)`.

Usage

```
## S4 method for signature 'mcnebula'
show(object)

## S4 method for signature 'mcnebula,character,ANY'
latest(x, slot, subscript)

## S4 method for signature 'missing,missing,missing'
latest()

## S4 method for signature 'mcnebula,ANY,ANY'
latest(x, slot, subscript)

## S4 method for signature 'mcnebula'
creation_time(x)

## S4 replacement method for signature 'mcnebula'
creation_time(x) <- value

## S4 method for signature 'mcnebula'
ion_mode(x)

## S4 replacement method for signature 'mcnebula'
ion_mode(x) <- value

## S4 method for signature 'mcnebula'
palette_set(x)

## S4 method for signature 'mcnebula'
palette_gradient(x)

## S4 method for signature 'mcnebula'
palette_stat(x)

## S4 method for signature 'mcnebula'
palette_col(x)

## S4 method for signature 'mcnebula'
palette_label(x)
```

```
## S4 method for signature 'mcnebula'
reference(x)

## S4 method for signature 'mcnebula'
specific_candidate(x)

## S4 method for signature 'mcnebula'
hierarchy(x)

## S4 method for signature 'mcnebula'
stardust_classes(x)

## S4 method for signature 'mcnebula'
nebula_index(x)

## S4 method for signature 'mcnebula'
spectral_similarity(x)

## S4 replacement method for signature 'mcnebula'
spectral_similarity(x) <- value

## S4 method for signature 'mcnebula'
features_annotation(x)

## S4 method for signature 'mcnebula'
features_quantification(x)

## S4 replacement method for signature 'mcnebula'
features_quantification(x) <- value

## S4 method for signature 'mcnebula'
sample_metadata(x)

## S4 replacement method for signature 'mcnebula'
sample_metadata(x) <- value

## S4 method for signature 'mcnebula'
classification(x)
```

Arguments

x	mcnebula object
slot	Character. Slot name.
subscript	numeric or character. The sequence or name for dataset in the 'list'.
value	The value for the slot.

Slots

`creation_time` character(1).

`ion_mode` character(1).

`melody` melody object.

`mcn_dataset` mcn_dataset object.

`statistic_set` statistic_set object.

`...` Slots inherit from project, nebula, export.

See Also

`tibble::as_tibble()`

Other nebulae: `nebula-class`

Other latests: `mcn_dataset-class`, `msframe-class`, `project_dataset-class`, `project_metadata-class`

Other subscripts: `msframe-class`, `project_conformation-class`, `subscript-class`

Examples

```
## Not run:
  test <- mcnebula()
  class(test)

  test <- mcn_5features
  ## slots
  ion_mode(test)
  project_version(test)
  melody(test)
  export_name(test)
  ## ...

  ## 'fast channel'
  palette_label(test)
  palette_stat(test)
  sample_metadata(test)
  ## ...

## End(Not run)
```

`mcn_5features` *Example object containing only five 'features'.*

Description

This is a pre-extracted data from the SIRIUS project of example data using MCnebula2, containing chemical formulae, chemical structure, chemical classification candidates, etc. for five 'features' (It is assumed to be a pre-processed metabolomic dataset). In order to reduce the memory footprint, some of its data columns have been removed, for example, the 'links' data column has been converted to character(1) for the chemical structure data.

Usage

```
mcn_5features
```

Format

`mcn_5features`:

mcnebula object.

Details

Data extracted via MCnebula2 package from path:

- `system.file("extdata", "raw_instance.tar.gz", package = "MCnebula2")`.

The MS/MS spectra were source from MoNA (MassBank of North America). The 5 MS/MS spectra were randomly extracted from GNPS spectral library of that. The candidates data were predicted via SIRIUS version 4 ...

Source

The related website:

- https://mona.fiehnlab.ucdavis.edu/downloads.
- https://bio.informatik.uni-jena.de/software/sirius/

---

mcn_dataset-class *Store processed data*

Description

This is a class object used to store filtered data and formated data. These data would be used for further analysis or visualization.

`mcn_dataset`, `mcn_dataset<-`: getter and setter for the `mcn_dataset` slot of the object.

`latest`: get the first data in `dataset` slot and format as "tbl". Equals:

- `latest(object)`
- `tibble::as_tibble(entity(dataset(x)[[1]]))`.

`extract_mcnset`: For fast extract data in object which containing `mcn_dataset` slot. Normally not used.

Usage

```
## S4 method for signature 'ANY'
mcn_dataset(x)

## S4 replacement method for signature 'ANY'
mcn_dataset(x) <- value

## S4 method for signature 'mcn_dataset,ANY,ANY'
latest(x)

## S4 method for signature 'ANY,character'
extract_mcnset(x, subscript)
```

Arguments

value	The value for the slot.
subscript	See subscript

Slots

`dataset` list with names of subscript. Store preliminary filtered data.

`reference` list with names of standard names. Store formated data, which is useful reference for further analysis or visualization.

`backtrack` list with names. Recovery stations halfway through data processing.

See Also

dataset

subscript

Other datasets: `dataset-class`, `project_dataset-class`

Other latests: `mcnebula-class`, `msframe-class`, `project_dataset-class`, `project_metadata-class`

`melody-class` *Mutiple color palette in hexadecimal code*

Description

This is a class object store Hex color used for visualization. In default (use `initialize_mcnebula()` to initialize the object), these these Hex color in each palette were get from package `ggsci`. Most of these palette in this package would passed to ggplot2::scale_fill_manual for filling color. So, let these Hex color with names may work well to specify target.

`melody`, `melody<-`: getter and setter for the `melody` slot of the object.

`palette_set`, `palette_set<-`: getter and setter for the `palette_set` slot of the object.

palette_gradient, palette_gradient<-: getter and setter for the palette_gradient slot of the object.

palette_stat, palette_stat<-: getter and setter for the palette_stat slot of the object.

palette_col, palette_col<-: getter and setter for the palette_col slot of the object.

palette_label, palette_label<-: getter and setter for the palette_label slot of the object.

Usage

```
## S4 method for signature 'melody'
show(object)

## S4 method for signature 'ANY'
melody(x)

## S4 replacement method for signature 'ANY'
melody(x) <- value

## S4 method for signature 'melody'
palette_set(x)

## S4 replacement method for signature 'melody'
palette_set(x) <- value

## S4 method for signature 'melody'
palette_gradient(x)

## S4 replacement method for signature 'melody'
palette_gradient(x) <- value

## S4 method for signature 'melody'
palette_stat(x)

## S4 replacement method for signature 'melody'
palette_stat(x) <- value

## S4 method for signature 'melody'
palette_col(x)

## S4 replacement method for signature 'melody'
palette_col(x) <- value

## S4 method for signature 'melody'
palette_label(x)

## S4 replacement method for signature 'melody'
palette_label(x) <- value
```

Arguments

value	The value for the slot.

Slots

palette_set character with names or not. Hex color.

palette_gradient character with names or not. Hex color.

palette_stat character with names or not. Hex color.

palette_col character with names or not. Hex color.

palette_label character with names or not. Hex color.

See Also

ggsci::pal_simpsons(), ggsci::pal_igv(), ggsci::pal_ucscgb(), ggsci::pal_d3()...

---

msframe-class *format and filter table data*

Description

Class for table data manipulation inside this package.

msframe, msframe<-: getter and setter for the msframe slot of the object.

latest: get data inside entity(object) and format as 'tbl'.

entity, entity<-: getter and setter for the entity slot of the object.

format_msframe:

filter_msframe: filter data in slot entity (data.frame).

Usage

```
## S4 method for signature 'msframe'
show(object)

## S4 method for signature 'ANY'
msframe(x)

## S4 replacement method for signature 'ANY'
msframe(x) <- value

## S4 method for signature 'msframe,ANY,ANY'
latest(x)

## S4 method for signature 'msframe'
entity(x)
```

```
## S4 replacement method for signature 'msframe'
entity(x) <- value

## S4 method for signature 'msframe,missing,missing,missing,missing,`function`'
format_msframe(x, fun_format)

## S4 method for signature
## 'data.frame,missing,missing,missing,missing,`function`'
format_msframe(x, fun_format)

## S4 method for signature
## 'msframe,character,missing,character,missing,missing'
format_msframe(x, names, types)

## S4 method for signature 'msframe,missing,missing,missing,missing,missing'
format_msframe(x)

## S4 method for signature
## 'msframe,missing,`function`,missing,`function`,missing'
format_msframe(x, fun_names, fun_types)

## S4 method for signature 'msframe,`function`,missing'
filter_msframe(x, fun_filter, f, ...)

## S4 method for signature 'msframe,`function`,formula'
filter_msframe(x, fun_filter, f, ...)
```

Arguments

x	msframe object.
value	The value for the slot.
fun_format	function to format slot entity. e.g., MCnebula2:::.format_msframe()
names	character with names. e.g., c(tani.score = "tanimotoSimilarity", mol.formula = "molecularFormula").
types	character with names. e.g., c(tani.score = "numeric", mol.formula = "character").
fun_names	function to get names. e.g., MCnebula2:::.get_attribute_name_sirius.v4()
fun_types	function to get types. e.g., MCnebula2:::.get_attribute_type_sirius.v4()
fun_filter	function used to filter the slot entity (data.frame). e.g., dplyr::filter(), head().
f	formula passed to split().
...	extra parameter passed to fun_filter.

Slots

entity data.frame.

subscript character(1). See subscript.

Note

The class is not for normal use of the package.

See Also

`tibble::as_tibble()`

Other subscripts: `mcnebula-class`, `project_conformation-class`, `subscript-class`

Other latests: `mcn_dataset-class`, `mcnebula-class`, `project_dataset-class`, `project_metadata-class`

`nebula-class` *Visualization component of chemical Nebulae/Nebula*

Description

This class store multiple components for visualization.

`parent_nebula`: Store data for visualization of Parent-Nebula.

`child_nebulae`: store data for visualization of Child-Nebulae.

`parent_nebula`, `parent_nebula<-`: getter and setter for the `parent_nebula` slot of the object.

`child_nebulae`, `child_nebulae<-`: getter and setter for the `child_nebulae` slot of the object.

`igraph`, `igraph<-`: getter and setter for the `igraph` slot of the object.

`tbl_graph`, `tbl_graph<-`: getter and setter for the `tbl_graph` slot of the object.

`layout_ggraph`, `layout_ggraph<-`: getter and setter for the `layout_ggraph` slot of the object.

`grid_layout`, `grid_layout<-`: getter and setter for the `grid_layout` slot of the object.

`viewports`, `viewports<-`: getter and setter for the `viewports` slot of the object.

`ggset`, `ggset<-`: getter and setter for the `ggset` slot of the object.

`panel_viewport`, `panel_viewport<-`: getter and setter for the `panel_viewport` slot of the object.

`legend_viewport`, `legend_viewport<-`: getter and setter for the `legend_viewport` slot of the object.

`structures_grob`, `structures_grob<-`: getter and setter for the `structures_grob` slot of the object.

`nodes_ggset`, `nodes_ggset<-`: getter and setter for the `nodes_ggset` slot of the object.

`nodes_grob`, `nodes_grob<-`: getter and setter for the `nodes_grob` slot of the object.

`ppcp_data`, `ppcp_data<-`: getter and setter for the `ppcp_data` slot of the object.

`ration_data`, `ration_data<-`: getter and setter for the `ration_data` slot of the object.

`ggset_annotate`, `ggset_annotate<-`: getter and setter for the `ggset_annotate` slot of the object.

Usage

```
## S4 method for signature 'parent_nebula'
show(object)

## S4 method for signature 'child_nebulae'
show(object)

## S4 method for signature 'ANY'
parent_nebula(x)

## S4 replacement method for signature 'ANY'
parent_nebula(x) <- value

## S4 method for signature 'ANY'
child_nebulae(x)

## S4 replacement method for signature 'ANY'
child_nebulae(x) <- value

## S4 method for signature 'ANY'
igraph(x)

## S4 replacement method for signature 'ANY'
igraph(x) <- value

## S4 method for signature 'ANY'
tbl_graph(x)

## S4 replacement method for signature 'ANY'
tbl_graph(x) <- value

## S4 method for signature 'ANY'
layout_ggraph(x)

## S4 replacement method for signature 'ANY'
layout_ggraph(x) <- value

## S4 method for signature 'ANY'
grid_layout(x)

## S4 replacement method for signature 'ANY'
grid_layout(x) <- value

## S4 method for signature 'ANY'
viewports(x)

## S4 replacement method for signature 'ANY'
viewports(x) <- value
```

```
## S4 method for signature 'ANY'
ggset(x)

## S4 replacement method for signature 'ANY'
ggset(x) <- value

## S4 method for signature 'ANY'
panel_viewport(x)

## S4 replacement method for signature 'ANY'
panel_viewport(x) <- value

## S4 method for signature 'ANY'
legend_viewport(x)

## S4 replacement method for signature 'ANY'
legend_viewport(x) <- value

## S4 method for signature 'ANY'
structures_grob(x)

## S4 replacement method for signature 'ANY'
structures_grob(x) <- value

## S4 method for signature 'ANY'
nodes_ggset(x)

## S4 replacement method for signature 'ANY'
nodes_ggset(x) <- value

## S4 method for signature 'ANY'
nodes_grob(x)

## S4 replacement method for signature 'ANY'
nodes_grob(x) <- value

## S4 method for signature 'ANY'
ppcp_data(x)

## S4 replacement method for signature 'ANY'
ppcp_data(x) <- value

## S4 method for signature 'ANY'
ration_data(x)

## S4 replacement method for signature 'ANY'
ration_data(x) <- value
```

```
## S4 method for signature 'ANY'
ggset_annotate(x)

## S4 replacement method for signature 'ANY'
ggset_annotate(x) <- value
```

Arguments

value	The value for the slot.

Slots

`parent_nebula` parent_nebula object.

`child_nebulae` child_nebulae object.

`igraph` "igraph" object or its list. See `igraph::graph_from_data_frame()`. The slot contains edges and nodes data of Child-Nebulae or Parent-Nebula. The "igraph" object can be output use `igraph::write_graph()` as ".graphml" file, which belong to a network data format that can be operated by other software such as Cytoscape (https://cytoscape.org/).

`tbl_graph` "tbl_graph" object or its list. See `tidygraph::as_tbl_graph()`. Converted from slot `igraph`.

`layout_ggraph` "layout_ggraph" object or its list. See `ggraph::create_layout()`. Create from slot `tbl_graph`, passed to `ggraph::ggraph()` for visualization.

`grid_layout` "layout" object. See `grid::grid.layout()`. Grid layout for position of each Child-Nebula to visualize.

`viewports` list with names. Each element must be "viewport" object. See `grid::viewport()`. Position for each Child-Nebula to visualize.

`panel_viewport` "viewport" object. See `grid::viewport()`. For visualization, the position to place overall Child-Nebulae.

`legend_viewport` "viewport" object. See `grid::viewport()`. For visualization, the position to place legend.

`ggset` ggset object or its list with names. Each ggset object can be visualized directly use `call_command()`.

`structures_grob` list with names. Each element is a "grob" object. See `grid::grob()`. Use `grid::grid.draw()` to visualize the chemical structure.

`nodes_ggset` list of ggset object. For drawing each node of 'features' ('features' means the detected peaks while processing LC-MS data) with annotation. Use `call_command()` to visualize the ggset.

`nodes_grob` list of "grob" object. Converted from slot `nodes_ggset` with slot `structures_grob`. Use `grid::grid.draw()` to visualize the "grob".

`ppcp_data` list with names. Each element is a data.frame. This is an annotation data of 'features' which would be visualize in nodes border as a radial bar plot. `ppcp_data`, i.e., posterior probability of classification prediction. See `filter_ppcp()`.

`ration_data` list with names. Each element is a data.frame. This is an annotation data of 'features' which would be visualize in nodes nucleus as ring plot. Generally, `ration_data` is the statistic data for samples.

`ggset_annotate` a list of ggset object. The annotated Child-Nebulae gathered from slot `ggset` and slot `nodes_grob`. Use `call_command()` to visualize the ggset. Be care, the object sometimes is too large that need lot of time to loading for visualization.

See Also

Other nebulae: `mcnebula-class`

`plot_msms_mirrors` *Draw MS/MS mirror bar plots*

Description

Draw MS/MS spectra as mirror bar plots using 'ggplot2' with `facet_wrap()`. The sub-panel of each 'features' would be found by `grid::grid.grep()`... Then chemical structures would be drawn into sub-panel.

Usage

```
plot_msms_mirrors(
  x,
  .features_id,
  fun_modify = modify_set_labs_xy,
  structure_vp = grid::viewport(0.7, 0.3, 0.3, 0.3)
)
```

Arguments

`x`	mcnebula object.
`.features_id`	character. The ID of 'features'.
`fun_modify`	function. Used to post modify the ggset object before visualization. See fun_modify.
`structure_vp`	'viewport' object. Created by `grid::viewport()`. The 'viewport' to draw chemical structures in sub-panel. Can be `NULL`, this would return a 'ggplot' object without chemical structure visualization.

See Also

`facet_wrap()`, grid::grid.force, `grid::grid.grep()`...

project-class *Collection of Interface for extracting data from raw directory*

Description

This is a class object designed to extract files in the project directory. Its responsibility is to describe the name, path and reading method of the file under the project directory; Use these information to extract and store data.

`project_version`, `project_version<-`: getter and setter for the `project_version` slot of the object.

`project_path`, `project_path<-`: getter and setter for the `project_path` slot of the object.

`file_name`, `file_api`, `attribute_name`: fast channel to obtain the downstream slot. e.g., getter for the `file_name` slot in sub-object of `project_conformation` slot of the object. Equals:

- `file_name(project_conformation(object))`
- `file_name(object)`.

`metadata`: fast channel to obtain the downstream slot, getter for the `metadata` slot in sub-object of `project_metadata` slot of the object. Equals:

- `metadata(project_metadata(object))`
- `metadata(object)`.

`methods_read`, `methods_format`, `methods_match`: fast channel to obtain the downstream slot. e.g., getter for the `methods_read` slot in sub-object of `project_api` slot of the object. Equals:

- `methods_read(project_api(object))`
- `methods_read(object)`.

`match.candidates_id`, `match.features_id`: fast channel to obtain data (mostly 'tbl' or 'data.frame') inside the downstream slot ('list'), getter for the data named `match.candidates_id` in `methods_match` slot (a 'list') in sub-object of `project_api` slot of the object. Equals:

- `methods_match(project_api(object))$match.candidates_id`
- `match.candidates_id(object)`.

`get_upper_dir_subscript`: Get the "subscript" name of the folder.

Usage

```
## S4 method for signature 'ANY'
project_version(x)

## S4 replacement method for signature 'ANY'
project_version(x) <- value

## S4 method for signature 'ANY'
```

```
project_path(x)

## S4 replacement method for signature 'ANY'
project_path(x) <- value

## S4 method for signature 'ANY'
file_name(x)

## S4 method for signature 'ANY'
file_api(x)

## S4 method for signature 'ANY'
attribute_name(x)

## S4 method for signature 'ANY'
metadata(x)

## S4 method for signature 'ANY'
methods_read(x)

## S4 method for signature 'ANY'
methods_format(x)

## S4 method for signature 'ANY'
methods_match(x)

## S4 method for signature 'ANY'
match.candidates_id(x)

## S4 method for signature 'ANY'
match.features_id(x)

## S4 method for signature 'ANY,character,missing'
get_upper_dir_subscript(x, subscript)
```

Arguments

x	Maybe object of class inherit project.
value	The value for the slot.
subscript	the "subscript" name of file. See subscript.

Details

It is a collection of classes whose names start with "project_":

- project_conformation: The name, path and attribute name of the file are described.
- project_api: Functions for reading and formatting data are provided.
- project_metadata: Metadata, which records the files stored in the project directory.

- project_dataset: The extracted data is stored here.

The above class objects are coordinated into a whole through the "subscript" name (see subscript). For example, when a command (`collate_data(x, ".f3_fingerid")`) requests to extract the files of subscript of ".f3_fingerid", the data extraction module:

- from slot of `project_conformation`, get the file name (pattern string) and path of subscript of ".f3_fingerid";
- match the files under the path with the pattern string (i.e., get the metadata of the files), then stored the metadata into slot of `project_metadata`;
- from slot of `project_api`, get the functions of subscript of ".f3_fingerid";
- use these functions to read and format the data in batches;
- store the extracted data into slot of `project_dataset`.

This class is mainly designed for extracting files under the SIRIUS project directory. These files are: mainly "tables" that can be read through functions such as `read.table`; numerous and have multiple directories; need to be processed in batches. SIRIUS project may alter the name and path of internal files during version changes, which is in fact deadly for MCnebula2. To make the data extraction module of MCnebula2 free from version issues, this class object is designed to flexibly handle the extraction of internal files. Most contents need to be considered by MCnebula2 developers. The only thing users need to know: slot of project_dataset object stores the extracted data.

Slots

`project_version` character(1). The target project version. e.g., "sirius.v4".

`project_path` character(1). The target project path.

`project_conformation` project_conformation object.

`project_metadata` project_metadata object.

`project_api` project_api object.

`project_dataset` project_dataset object.

See Also

Other projects: `project_api-class`, `project_conformation-class`, `project_dataset-class`, `project_metadata-class`

`project_api-class` *Function set for extracting data*

Description

This is a class object used to store various functions for extracting and formatting data. See project for joint application with other related classes.

`project_api`, `project_api<-`: getter and setter for the `project_api` slot of the object.

`methods_read`, `methods_read<-`: getter and setter for the `methods_read` slot of the object.

`methods_format`, `methods_format<-`: getter and setter for the `methods_format` slot of the object.

`methods_match`, `methods_match<-`: getter and setter for the `methods_match` slot of the object.

Usage

```
## S4 method for signature 'project_api'
show(object)

## S4 method for signature 'ANY'
project_api(x)

## S4 replacement method for signature 'ANY'
project_api(x) <- value

## S4 method for signature 'project_api'
methods_read(x)

## S4 replacement method for signature 'project_api'
methods_read(x) <- value

## S4 method for signature 'project_api'
methods_format(x)

## S4 replacement method for signature 'project_api'
methods_format(x) <- value

## S4 method for signature 'project_api'
methods_match(x)

## S4 replacement method for signature 'project_api'
methods_match(x) <- value
```

Arguments

`value`	The value for the slot.

Slots

`methods_read` list. Store a list of functions for reading data. The list with the names: "read" + "subscript". e.g., "read.f3_fingerid".

`methods_format` function. The function is used to format the data (e.g., rename the column names; convert the columns of character type into numeric).

methods_match list. Store a list of functions for matching and extracting string.

Note

The class is not for normal use of the package.

See Also

Other projects: project-class, project_conformation-class, project_dataset-class, project_metadata-class

---

project_conformation-class

Clarify the name, path and attribute name of files in the project (directory)

Description

This is a class object used to record the name, path and attribute name of the file. These records can be retrieved by "subscript" (see subscript). See project for joint application with other related classes.

project_conformation, project_conformation<-: getter and setter for the project_conformation slot of the object.

file_name, file_name<-: getter and setter for the file_name slot of the object.

file_api, file_api<-: getter and setter for the file_api slot of the object.

attribute_name, attribute_name<-: getter and setter for the attribute_name slot of the object.

Usage

```
## S4 method for signature 'project_conformation'
show(object)

## S4 method for signature 'ANY'
project_conformation(x)

## S4 replacement method for signature 'ANY'
project_conformation(x) <- value

## S4 method for signature 'project_conformation'
file_name(x)

## S4 replacement method for signature 'project_conformation'
file_name(x) <- value

## S4 method for signature 'project_conformation'
file_api(x)
```

```
## S4 replacement method for signature 'project_conformation'
file_api(x) <- value

## S4 method for signature 'project_conformation'
attribute_name(x)

## S4 replacement method for signature 'project_conformation'
attribute_name(x) <- value
```

Arguments

value	The value for the slot.

Slots

`file_name` character with names. Record the filenames or pattern string or function name (begin with "FUN_") for each "subscript" (imply file names).

`file_api` character with names. Record the file path for each "subscript" (imply file names). The path is descriped by "subscript" with "/".

`attribute_name` character with names. Record the attribute name for each "subscript" (imply column names).

Note

The class is not for normal use of the package.

See Also

Other projects: `project-class`, `project_api-class`, `project_dataset-class`, `project_metadata-class`

Other subscripts: `mcnebula-class`, `msframe-class`, `subscript-class`

`project_dataset-class` *Store extracted data*

Description

This is a class object used to store extracted data (raw data). See project for joint application with other related classes.

`project_dataset`, `project_dataset<-`: getter and setter for the `project_dataset` slot of the object.

`latest`: get the first data in `dataset` slot ('list') and format as 'tbl'. Equals:

- `latest(object)`
- `tibble::as_tibble(entity(dataset(object)[[1]]))`

`extract_rawset`: For fast extract data in object which containing `project_dataset` slot. Normally not used.

Usage

```
## S4 method for signature 'ANY'
project_dataset(x)

## S4 replacement method for signature 'ANY'
project_dataset(x) <- value

## S4 method for signature 'project_dataset,ANY,ANY'
latest(x)

## S4 method for signature 'ANY,character,ANY'
extract_rawset(x, subscript)

## S4 method for signature 'ANY,character,`function`'
extract_rawset(x, subscript, fun_collate, ...)
```

Arguments

x	an object contain project_dataset slot.
value	The value for the slot.
subscript	character. Specified the data in dataset slot in project_dataset slot. See VIRTUAL_subscript.
fun_collate	function. If the specified data not exists in dataset slot, it will be used to collate data. This parameter is not for normal use.
...	parameters passed to 'fun_collate'.

Slots

dataset list. See dataset.

See Also

tibble::as_tibble()

Other projects: project-class, project_api-class, project_conformation-class, project_metadata-class

Other datasets: dataset-class, mcn_dataset-class

Other latests: mcn_dataset-class, mcnebula-class, msframe-class, project_metadata-class

---

project_metadata-class

Metadata of files

Description

This is a class object used to store metadata of files. See project for joint application with other related classes.

`project_metadata`, `project_metadata<-`: getter and setter for the `project_metadata` slot of the object.

`latest`: get the first data in `metadata` slot and format as "tbl".

`metadata`, `metadata<-`: getter and setter for the `metadata` slot of the object.

`add_dataset`: add the list into slot `metadata`.

`extract_metadata`: use "subscript" to extract metadata from an object with slot `project_metadata`, and then return it as a new `project_metadata`.

`get_metadata`: for an object with slot of `project_metadata`, get the metadata of files of specified "subscript", then return the object.

Usage

```
## S4 method for signature 'project_metadata'
show(object)

## S4 method for signature 'ANY'
project_metadata(x)

## S4 replacement method for signature 'ANY'
project_metadata(x) <- value

## S4 method for signature 'project_metadata,ANY,ANY'
latest(x)

## S4 method for signature 'project_metadata'
metadata(x)

## S4 replacement method for signature 'project_metadata'
metadata(x) <- value

## S4 method for signature 'project_metadata,list'
add_dataset(x, list)

## S4 method for signature 'ANY,character'
extract_metadata(x, subscript)

## S4 method for signature 'ANY,character,ANY,ANY,ANY,ANY'
get_metadata(x, subscript)

## S4 method for signature
## 'missing,
##   character,
##   project_metadata,
```

```
##   project_conformation,
##   character,
##   character'
get_metadata(
  subscript,
  project_metadata,
  project_conformation,
  project_version,
  path
)
```

Arguments

value	The value for the slot.
list	a list (with names) of metadata (data.frame) with names.
subscript	see subscript.
project_metadata	project_metadata object. Used by get_metadata(). If 'missing', the slot project_metadata inside the object will be used.
project_conformation	project_conformation object. Used by get_metadata(). If 'missing', the slot project_conformation inside the object will be used.
path	character. The path of the project directory (generally, SIRIUS project). If 'missing', the slot project_path inside the object will be used.

Slots

metadata a list with names of subscript. Each element of the list is a data.frame.

Note

The class is not for normal use of the package.

See Also

Other projects: project-class, project_api-class, project_conformation-class, project_dataset-class

Other latests: mcn_dataset-class, mcnebula-class, msframe-class, project_dataset-class

---

rblock *Eval the code as well create 'code_block' object*

Description

rblock: Run or not run the code with formatting as code_block object.

Usage

```
rblock(code, eval = T, envir = parent.frame(), ...)
```

Arguments

code	The code to run or document. Braces (‘’) must be used.
eval	logical. Whether to eval the code.
envir	environment. The ‘environment’ in which the code is to be evaluated.
...	Other parameters passed to new_code_block().

Examples

```
## Not run:
  rblock({
    test1 <- 1
    test2 <- 2
    test3 <- 3
  })

  rblock({
    test <- mcn_5features
    ## this annotation line would be ignored
    test1 <- filter_structure(test)
    test1 <- create_reference(test1)
    test1 <- filter_formula(test1, by_reference=T)
    test1 <- create_stardust_classes(test1)
  })

## End(Not run)
```

---

reference-class *Share slots and methods for classes inherite from VIRTUAL_reference*

Description

This VIRTUAL class provides a slot for storing processed data.

reference, reference<-: getter and setter for the reference slot of the object.

Usage

```
## S4 method for signature 'ANY'
reference(x)

## S4 replacement method for signature 'ANY'
reference(x) <- value
```

Arguments

x	object inherit class reference.
value	The value for the slot.

Slots

reference list with names (formal name).

---

render_report *Convert 'report' as .Rmd file*

Description

render_report: Write down report onject as .Rmd file, then rmarkdown::render() can be used to output any available formation.

Usage

```
render_report(x, file, set_all_eval = F)
```

Arguments

x	report object.
file	character(1). File name to save as.
set_all_eval	logical(1). If TRUE, all code_block object or section object related with "r" would be set with eval = T.

---

report-class *Creating a formatted report*

Description

The report module can create output report quickly for and not just for the mcnebula2 workflow. The report system is primarily a class object that manages text and code blocks. Heading or paragraphs or code blocks were treated as individual report units and deposited sequentially in "layers". The report system provides methods to exhibit, modify these layers. Reports can be exported as ".Rmd" text files, and subsequently, users can call rmarkdown::render() for output formatted documents.

show_layers: show layers slots in a pretty and readable style.

yaml, yaml<-: getter and setter for the yaml slot of the object.

new_report: Create a report object.

new_report(): get the default parameters for the method new_report.

new_report(x, ...): use the default parameters whatever 'missing' while performing the method new_report.

call_command: Format 'report' object as character, which can be output by writeLines() function as '.Rmd' file and than use rmarkdown::render output as pdf, html, or other format files.

search_heading: Regex match for heading object

Usage

```
## S4 method for signature 'report'
show_layers(x)

## S4 method for signature 'ANY'
yaml(x)

## S4 replacement method for signature 'ANY'
yaml(x) <- value

## S4 method for signature 'character'
new_report(..., yaml)

## S4 method for signature 'missing'
new_report(..., yaml)

## S4 method for signature 'report'
call_command(x)

search_heading(x, pattern, level = NULL)
```

Arguments

x	report object.
value	The value for the slot.
...	An arbitrary number of heading, section or code_block in sequence. Specially, NULL can be passed herein, but would be ignored.
yaml	character. Passed to .Rmd for setting format of documentation.
pattern	character(1). For Regex match. Allowed Perl expression.
level	numeric. in slot layers.

Slots

yaml character. Metadata passed to .Rmd for setting format of documentation. See https://bookdown.org/yihui/rmarkdown/compile.html for details.

layers list. Element in list must be section, heading or code_block.

See Also

writeLines(), rmarkdown::render()...

Other layerSets: ggset-class, layerSet-class

Other call_commands: code_block-class, command-class, ggset-class, section-class

Examples

```
## Not run:
  s1 <- new_heading("Title 1", 1)

  s1.5 <- new_section2(
    c("..."), NULL
  )

  s2 <- new_section2(
    c("This is sentence 1.",
      "This is sentence 2.",
      "This is sentence 3."),
    rblock({
      df <- data.frame(x = 1:10, y = 1:10)
      df <- dplyr::mutate(df,
        group = rep(paste0("p", 1:3), c(3, 3, 4))
      )
    })
  )

  s3 <- new_heading("Title 2", 1)

  s4 <- new_section2(
    c("This is a paragraph..."),
    rblock({
      p <- ggplot(df) +
        geom_point(aes(x = x, y = y, fill = group))
      ggsave(f <- paste0(tempdir(), "/test.pdf"), p)
    })
  )

  s5 <- include_figure(f, "name", "Caption: ...")

  s6 <- rblock({
    print(1:100)
  }, args = list(echo = F, eval = F))

  s7 <- c("... paragraph ...")

  sections <- gather_sections()
  report <- do.call(new_report, sections)
  render_report(report, paste0(tempdir(), "/report.Rmd"), T)

  ####################
  ##### Another example
  ####################

  h1 <- new_heading("heading, level 1", 1)

  sec1 <- new_section(
    "sub-heading", 2,
    "This is a description.",
```

```
  new_code_block(codes = "seq <- lapply(1:10, cat)")
)

h2 <- new_heading("heading 2, level 1", 1)

fig_block <- new_code_block_figure("plot", "this is a caption",
  codes = "df <- data.frame(x = 1:10, y = 1:10)
    p <- ggplot(df) +
      geom_point(aes(x = x, y = y))
    p"
)
sec2 <- new_section(
  "sub-heading2", 2,
  paste0(
    "This is a description. ",
    "See Figure ", get_ref(fig_block), "."
  ),
  fig_block
)

a_data <- dplyr::storms[1:15, 1:10]
table_block <- include_table(a_data, "table1", "This is a caption")

sec3 <- new_section(
  NULL, ,
  paste0("See Table ", get_ref(table_block, "tab"), "."),
  NULL
)

tmp_p <- paste0(tempdir(), "/test.pdf")
pdf(tmp_p)
plot(1:10)
dev.off()
fig_block_2 <- include_figure(tmp_p, "plot2", "this is a caption")
sec4 <- history_rblock(, "^tmp_p <- ", "^fig_block_2")
sec4

## gather
yaml <- "title: 'title'\noutput:\n  bookdown::pdf_document2"
report <- new_report(
  h1, sec1, h2, sec2,
  table_block, sec3,
  fig_block_2, sec4,
  yaml = yaml
)
report

## output
tmp <- paste0(tempdir(), "/tmp_output.Rmd")
render_report(report, tmp)
rmarkdown::render(tmp)
file.exists(sub("Rmd$", "pdf", tmp))
```

```
## End(Not run)
```

---

reportDoc *Example text for report description.*

Description

Lazy data used to supplement the presentation of the report. It doesn't make the description of the report outstanding, but it at least makes it decent (maybe).

Usage

```
reportDoc
```

Format

`reportDoc`**:**
A list object.

---

section-class *Basic cells in the report*

Description

A class object consist of heading, paragraph (character), and code_block. These section belong to basic cells of report.

This is a class object used to clarify the heading and its hierarchy.

`heading`, `heading<-`: getter and setter for the `heading` slot of the object.

`level`, `level<-`: getter and setter for the `level` slot of the object.

`new_heading`: create heading object.

`call_command`: Format 'heading' object as character.

`paragraph`, `paragraph<-`: getter and setter for the `paragraph` slot of the object.

`new_section()`: get the default parameters for the method `new_section`.

`new_section(x, ...)`: use the default parameters whatever 'missing' while performing the method `new_section`.

`new_section`: create section object.

`new_section2`: Identical to `new_section(NULL, , ...)`

`call_command`: Format 'section' object as character.

Usage

```
## S4 method for signature 'ANY'
heading(x)

## S4 replacement method for signature 'ANY'
heading(x) <- value

## S4 method for signature 'heading'
level(x)

## S4 replacement method for signature 'heading'
level(x) <- value

## S4 method for signature 'character,numeric'
new_heading(heading, level)

## S4 method for signature 'heading'
call_command(x)

## S4 method for signature 'section'
paragraph(x)

## S4 replacement method for signature 'section'
paragraph(x) <- value

## S4 method for signature 'missing,missing,missing,missing'
new_section(heading, level, paragraph, code_block)

## S4 method for signature 'ANY,ANY,ANY,ANY'
new_section(heading, level, paragraph, code_block)

## S4 method for signature 'character,numeric,character,maybe_code_block'
new_section(heading, level, paragraph, code_block)

## S4 method for signature '`NULL`,numeric,character,maybe_code_block'
new_section(heading, level, paragraph, code_block)

new_section2(paragraph, code_block)

## S4 method for signature 'section'
call_command(x)

## S4 method for signature '`NULL`'
call_command(x)
```

Arguments

value	The value for the slot.

heading	character(1). For slot .Data.
level	numeric(1). For slot level.
paragraph	character. Text for description.
code_block	code_block object.

Slots

heading heading object.

paragraph character. Text for description.

code_block code_block object.

.Data character(1). Text of heading.

level numeric. Level of heading.

See Also

Other call_commands: code_block-class, command-class, ggset-class, report-class

Examples

```
## Not run:
  ## -------------------------------------
  ## heading
  new_heading("this is a heading", 2)

  ## -------------------------------------
  ## section
  ## example 1
  para <- "This is a paragraph stating"
  section <- new_section("this is a heading", 2, para)
  ## see results
  section
  call_command(section)
  writeLines(call_command(section))

  ## example 2
  para <- "This is a paragraph stating"
  section <- new_section(NULL, , para, NULL)

  ## example 3
  para <- "This is a paragraph stating"
  block <- new_code_block(codes = "df <- data.frame(x = 1:10)")
  section <- new_section("heading", 2, para, block)
  section

  ## example 4
  codes <- "df <- data.frame(x = 1:10, y = 1:10)
    p <- ggplot(df) +
      geom_point(aes(x = x, y = y))
    p"
  fig_block <- new_code_block_figure("plot", "this is caption", codes = codes)
```

```
  para <- paste0("This is a paragraph describing the picture. ",
                 "See Figure ", get_ref(fig_block), ".")
  section <- new_section("heading", 2, para, fig_block)
  section
  ## output
  tmp <- paste0(tempdir(), "/tmp_output.Rmd")
  writeLines(call_command(section), tmp)
  rmarkdown::render(tmp, output_format = "bookdown::pdf_document2")
  file.exists(sub("Rmd$", "pdf", tmp))
  ## see [report-class] object:
  ## A complete output report, including multiple 'section'.

  ## defalt parameters
  new_section()

## End(Not run)
```

`set_nodes_color-methods`

Custom defined nodes color in Nebulae (network)

Description

Custom defined the nodes color for visualizing. Run after `activate_nebulae()`.

Usage

```
## S4 method for signature 'mcnebula,character,data.frame,missing'
set_nodes_color(x, attribute, extra_data)

## S4 method for signature 'mcnebula,character,missing,missing'
set_nodes_color(x, attribute)

## S4 method for signature 'mcnebula,missing,missing,logical'
set_nodes_color(x, use_tracer)
```

Arguments

`x`	mcnebula object.
`attribute`	character. The attribute specified to colorful the nodes. Can be continues attribute or discrete attribute, exist in 'layout_ggraph' object or data of `extra_data`. Related with `ggplot2::scale_fill_gradientn()` and `ggplot2::scale_fill_manual()`. If the attribute is continues, colors in `palette_gradient(object)` would be used. If the attribute is discrete, use colors in `palette_set(object)`.
`extra_data`	data.frame. Extra data used for setting nodes color. The data.frame must contains column of '.features_id'.
`use_tracer`	logical. If `TRUE`, hightlight the 'top' 'features' marked in 'nebula_index' data. See `set_tracer()`.

See Also

activate_nebulae(), set_tracer()...

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_parent_nebula(test1, 0.01)
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_parent_layout(test1)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)

  ids <- features_annotation(test1)$.features_id
  extra_data <- data.frame(
    .features_id = ids,
    attr_1 = rnorm(length(ids), 100, 50),
    attr_2 = sample(c("special", "normal"), 5, replace = T)
  )

  test1 <- set_nodes_color(test1, "attr_1", extra_data)
  visualize(test1, 1)
  visualize_all(test1)
  ## set labal of the legend
  export_name(test1) <- c(
    export_name(test1),
    attr_1 = "Continuous attribute",
    attr_2 = "Discrete attribute"
  )
  visualize_all(test1)

  test1 <- set_nodes_color(test1, "attr_2", extra_data)
  visualize(test1, 1)
  visualize_all(test1)

  ## set colors for 'tracer'
  test1 <- set_tracer(test1, ids[1:2])
  ## re-build Child-Nebulae
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)
  ## set color
```

```
  test1 <- set_nodes_color(test1, use_tracer = T)
  visualize_all(test1)

## End(Not run)
```

`set_ppcp_data-methods` *Custom specify PPCP data for visualization in nodes*

Description

Run before `annotate_nebula()` or `draw_nodes()`. Custom specify PPCP data for visualization in nodes. All chemical classes exists in PPCP data could be specified.

`set_ppcp_data()`: get the function for generating default parameters for the method `set_ppcp_data`.

`set_ppcp_data(x, ...)`: use the default parameters whatever 'missing' while performing the method `set_ppcp_data`.

Usage

```
## S4 method for signature 'missing,missing'
set_ppcp_data()

## S4 method for signature 'mcnebula,ANY'
set_ppcp_data(x, classes)

## S4 method for signature 'mcnebula,character'
set_ppcp_data(x, classes)
```

Arguments

`x`	mcnebula object.
`classes`	character. The names of chemical classes. Use `classification(object)` to get optional candidates.

See Also

`annotate_nebula()`, `draw_nodes()`.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
```

```
test1 <- cross_filter_stardust(test1, 2, 1)
test1 <- create_nebula_index(test1)
test1 <- compute_spectral_similarity(test1)
test1 <- create_child_nebulae(test1, 0.01)
test1 <- create_child_layouts(test1)
test1 <- activate_nebulae(test1)

## optional 'nebula_name'
visualize(test1)
## a class for example
class <- visualize(test1)$class.name[1]
tmp <- export_path(test1)
## customize the chemical classes displayed
## in the radial bar plot in node.
classes <- classification(test1)
## get some random classes
set.seed(10)
classes <- sample(classes$class.name, 50)
classes
test1 <- set_ppcp_data(test1, classes)
test1 <- draw_nodes(test1, class,
  add_structure = F,
  add_ration = F
)

## visualize with ID of 'feature' (.features_id)
## with legend
ids <- names(nodes_grob(child_nebulae(test1)))
x11(width = 15, height = 5)
show_node(test1, ids[1])

## get a function to generate default parameters
set_ppcp_data()
## the default parameters
set_ppcp_data()(test1)

unlink(tmp, T, T)

## End(Not run)
```

`set_ration_data-methods` *Custom specify the quantification data for visualization in nodes*

Description

Run before `annotate_nebula()` or `draw_nodes()`. Set whether to use the group average value to annotate the 'features' quantification in nodes. Before this methods, user should use `features_quantification<-` and `sample_metadata<-` to set quantification data and metadata in mcnebula object.

`set_ration_data()`: get the default parameters for the method `set_ration_data`.

`set_ration_data(x, ...)`: use the default parameters whatever 'missing' while performing the method `set_ration_data`.

Usage

```
## S4 method for signature 'missing,missing'
set_ration_data()

## S4 method for signature 'mcnebula,ANY'
set_ration_data(x, mean)

## S4 method for signature 'mcnebula,logical'
set_ration_data(x, mean)
```

Arguments

x	mcnebula object.
mean	logical. If TRUE, calculate mean value for all group of the samples.

See Also

`annotate_nebula()`, `draw_nodes()`.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)

  ## set features quantification data
  ids <- features_annotation(test1)$.features_id
  quant. <- data.frame(
    .features_id = ids,
    sample_1 = rnorm(length(ids), 1000, 200),
    sample_2 = rnorm(length(ids), 2000, 500)
  )
  quant. <- dplyr::mutate(quant.,
    sample_3 = sample_1 * 1.5,
    sample_4 = sample_2 * 5
```

```
  )
  metadata <- data.frame(
    sample = paste0("sample_", 1:4),
    group = rep(c("control", "model"), c(2, 2))
  )
  features_quantification(test1) <- quant.
  sample_metadata(test1) <- metadata

  ## a more convenient way to obtain simulation data
  # test1 <- MCnebula2:::.simulate_quant_set(test1)

  ## optional 'nebula_name'
  visualize(test1)
  ## a class for example
  class <- visualize(test1)$class.name[1]
  tmp <- export_path(test1)

  test1 <- set_ration_data(test1, mean = F)
  test1 <- draw_nodes(test1, class,
    add_structure = F,
    add_ppcp = F
  )

  ## visualize with ID of 'feature' (.features_id)
  ## with legend
  ids <- names(nodes_grob(child_nebulae(test1)))
  x11(width = 15, height = 5)
  show_node(test1, ids[1])

  ## the default parameters
  set_ration_data()

  unlink(tmp, T, T)

## End(Not run)
```

`set_tracer-methods` *Mark top 'features' in 'nebula_index' data*

Description

Custom defined the specific 'features' in 'nebula_index' data. Mark these 'features' for subsequent visualization with eye-catching highlighting (`set_nodes_color()`). Run after `create_nebula_index()`.

`set_tracer()`: get the function for generating default parameters for the method `set_tracer`.

`set_tracer(x, ...)`: use the default parameters whatever 'missing' while performing the method `set_tracer`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing'
set_tracer()

## S4 method for signature 'mcnebula,ANY,ANY,ANY'
set_tracer(x, .features_id, colors, rest)

## S4 method for signature 'mcnebula,character,character,character'
set_tracer(x, .features_id, colors, rest)
```

Arguments

x	mcnebula object.
.features_id	character. The ID of 'features' to mark.
colors	character. Hex color.
rest	character(1). Hex color.

See Also

create_nebula_index(), set_nodes_color().

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)

  ids <- features_annotation(test1)$.features_id
  test1 <- set_tracer(test1, ids[1:2])
  ## see results
  nebula_index(test1)

  ## see examples in 'set_nodes_color()'

## End(Not run)
```

`statistic_set-class` *Data used for statistic analysis*

Description

A class object for statistic analysis, associate with package of "limma" for binary comparison.

`statistic_set`, `statistic_set<-`: getter and setter for the `statistic_set` slot of the object.

`design_matrix`, `design_matrix<-`: getter and setter for the `design_matrix` slot of the object.

`contrast_matrix`, `contrast_matrix<-`: getter and setter for the `contrast_matrix` slot of the object.

`top_table`, `top_table<-`: getter and setter for the `top_table` slot of the object.

Usage

```
## S4 method for signature 'ANY'
statistic_set(x)

## S4 replacement method for signature 'ANY'
statistic_set(x) <- value

## S4 method for signature 'ANY'
design_matrix(x)

## S4 replacement method for signature 'ANY'
design_matrix(x) <- value

## S4 method for signature 'ANY'
contrast_matrix(x)

## S4 replacement method for signature 'ANY'
contrast_matrix(x) <- value

## S4 method for signature 'ANY'
top_table(x)

## S4 replacement method for signature 'ANY'
top_table(x) <- value
```

Arguments

`value`	The value for the slot.

Slots

`design_matrix` matrix. Create by `stats::model.matrix()`.

`contrast_matrix` matrix. Create by `limma::makeContrasts()`.

`dataset` ANY. Dataset used for `limma::lmFit()`, `limma::eBayes()` and other functions.

`top_table` list with names. Each element of list should be "data.frame" or "tbl".

`subscript-class` *Share slots and methods for classes inherite from VIRTUAL_subscript*

Description

This VIRTUAL class provides a slot for signing the data. The "subscript" like the signature for data, used to distinguish different data or file and retrieve it accurately. The "subscript" is mostly used for project (as well as its related classes):

- imply file names. e.g., for "sirius.v4", ".f3_fingerid" indicate all files in directory of "fingerid" for each features.
- imply attribute names. e.g., for "sirius.v4", "tani.score" indicate attribute name of "tanimotoSimilarity".

In essence, "subscript" is the alias of a file or data or attribute. In this package, using the "subscript" system means that all external data names are given an alias. In fact, this makes things more complicated. Why did we do this? Because the naming system of external data is not constant, these names may change with the version of the data source. In order to enable this R package to accurately extract and call these data, it is necessary to establish a set of aliases within the package. "Subscript" names are used internally by this package. They correspond to external data and are equivalent to providing an interface to interface with external data.

`subscript`, `subscript<-`: getter and setter for the `subscript` slot of the object.

Usage

```
## S4 method for signature 'ANY'
subscript(x)

## S4 replacement method for signature 'ANY'
subscript(x) <- value
```

Arguments

`x`	object inherit class `subscript`.
`value`	The value for the slot.

Slots

`subscript` character(1).

See Also

Other subscripts: `mcnebula-class`, `msframe-class`, `project_conformation-class`

visualize-methods *Visualize Nebulae in R graphic device*

Description

Methods used for visualization. Show chemical Nebulae (either Parent-Nebula or Child-Nebulae) in R graphic device. Run after `activate_nebulae()`

`visualize(x)`: get a 'tbl' about Child-Nebulae candidates for `visualize` methods to visualize.

`visualize()`: get the default parameters for the method `visualize`.

`visualize(x, ...)`: use the default parameters whatever 'missing' while performing the method `visualize`.

`get_ggset`: similar to `visualize(...)`, but get ggset object.

`visualize_all()`: get the default parameters for the method `visualize_all`.

`visualize_all(x, ...)`: use the default parameters whatever 'missing' while performing the method `visualize_all`.

`visualize_all`: visualize overall Child-Nebulae into R graphic device.

`visualize_ids`: Plot a label map about the location of the 'features'.

Usage

```
## S4 method for signature 'mcnebula,missing,ANY,missing'
visualize(x, fun_modify)

## S4 method for signature 'missing,missing,missing,missing'
visualize()

## S4 method for signature 'mcnebula,ANY,ANY,ANY'
visualize(x, item, fun_modify, annotate)

## S4 method for signature 'mcnebula,character,`function`,missing'
visualize(x, item, fun_modify)

## S4 method for signature 'mcnebula,numeric,`function`,missing'
visualize(x, item, fun_modify)

## S4 method for signature 'mcnebula,numeric_or_character,`function`,logical'
visualize(x, item, fun_modify, annotate)

get_ggset(x, item, fun_modify, annotate = F)

## S4 method for signature 'missing,missing,missing,missing'
visualize_all()

## S4 method for signature 'mcnebula,ANY,ANY,ANY'
```

```
visualize_all(x, newpage, fun_modify, legend_hierarchy)

## S4 method for signature 'mcnebula,logical,`function`,logical'
visualize_all(x, newpage, fun_modify, legend_hierarchy)

visualize_ids(x, item)
```

Arguments

x	mcnebula object.
fun_modify	function. Used to post modify the ggset object before visualization. See fun_modify.
item	character(1) or numeric(1). If `character`, the value should be a name of chemical class in 'nebula_index' data. Its Nebulae has been activated via `activate_nebulae()`. If `numeric`, the value should be the sequence of Nebulae... Use `visualize(object)` to get the optional value.
annotate	logical. If `TRUE`, visualize the Nebula with the annotation. Only available `annotate_nebula()` has been run for the Nebula.
newpage	logical. If `TRUE`, use `grid::grid.newpage()` before visualization.
legend_hierarchy	logical. If `TRUE`, visualize the legend of chemical hierarchy.

Examples

```
## Not run:
  test <- mcn_5features

  ## the previous steps
  test1 <- filter_structure(test)
  test1 <- create_reference(test1)
  test1 <- filter_formula(test1, by_reference = T)
  test1 <- create_stardust_classes(test1)
  test1 <- create_features_annotation(test1)
  test1 <- cross_filter_stardust(test1, 2, 1)
  test1 <- create_nebula_index(test1)
  test1 <- compute_spectral_similarity(test1)
  test1 <- create_parent_nebula(test1, 0.01)
  test1 <- create_child_nebulae(test1, 0.01)
  test1 <- create_parent_layout(test1)
  test1 <- create_child_layouts(test1)
  test1 <- activate_nebulae(test1)

  ## optional Child-Nebulae
  visualize(test1)

  visualize(test1, "parent")
  visualize(test1, 1)
  visualize_all(test1)
  ## ...

  ## use 'fun_modify'
```

```
  visualize(test1, 1, modify_default_child)
  visualize(test1, 1, modify_unify_scale_limits)
  visualize(test1, 1, modify_set_labs)
  ## ...

## End(Not run)
```

`workflow-methods` *Quickly create analysis templates.*

Description

Quickly create analysis templates and quickly construct a report. This template contains a number of pre-defined sections, each with a fixed description and code. The flexible approach is to use this method to obtain the codes that form these sections and then modify these codes, which is more likely to result in analysis results and reports that meet the requirements.

`workflow()`: get the available section names and its heading level.

`workflow(...)`: use the default parameters whatever 'missing' while performing the method `workflow`.

Usage

```
## S4 method for signature 'missing,missing,missing,missing,missing,missing'
workflow()

## S4 method for signature 'ANY,character,ANY,ANY,ANY,ANY'
workflow(sections, mode, envir, sirius_version, sirius_project, ion_mode, ...)

## S4 method for signature
## 'numeric,character,environment,character,character,character'
workflow(sections, mode, envir, sirius_version, sirius_project, ion_mode, ...)
```

Arguments

`sections`	numeric with names. Use `workflow()` to show the available sections.
`mode`	character(1). "print" or "run". If "print", print the template of the select workflow; If "run", the codes would be eval, and return with a report object.
`envir`	The environment to eval the codes. Default is `parent.frame()`.
`sirius_version`	character(1). Passed to `initialize_mcnebula()`.
`sirius_project`	character(1). Passed to `initialize_mcnebula()`.
`ion_mode`	character(1). Set this using `ion_mode<-`.
`...`	...

Index