R codes of exMCnebula2

Contents

1 File: aaa.R

```
# ============================================================================
# utilites
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases utilites
#'
#' @title utilites for programming
```

```r
#'
#' @description This is a combination of tools that are not always used.
#'
#' @name utilites
NULL
#> NULL

reCallMethod <-
  function(funName, args, ...){
    arg.order <- unname(getGeneric(funName)@signature)
    args.missing <- !arg.order %in% names(args)
    if (any(args.missing)) {
      args.missing <- arg.order[args.missing]
      args.missing <- sapply(args.missing, simplify = F,
        function(x) structure(0L, class = "missing"))
      args <- c(args, args.missing)
    }
    args <- lapply(arg.order, function(i) args[[i]])
    sig <- get_signature(args)
    method <- selectMethod(funName, sig)
    last_fun <- sys.function(sys.parent())
    n <- 0
    while (identical(last_fun, method@.Data, ignore.environment = T)) {
      if (n == 0) {
        mlist <- getMethodsForDispatch(getGeneric(funName))
      }
      n <- n + 1
      rm(list = paste0(method@defined, collapse = "#"), envir = mlist)
      method <- selectMethod(funName, sig, mlist = mlist)
    }
    expr <- paste0("method@.Data(",
      paste0(paste0(arg.order, " = args[[",
          1:length(arg.order), "]]"),
        collapse = ", "),
      ", ...)")
    eval(parse(text = expr))
  }

setMissing <-
  function(generic, ..., .SIG = "missing"){
    args <- list(...)
```

```
    sig <- getGeneric(generic)@signature
    res <- vapply(sig, FUN.VALUE = "character",
      function(name){
        if (is.null(args[[ name ]]))
          .SIG
        else
          args[[ name ]]
      })
    names(res) <- sig
    return(res)
  }

.fresh_param <-
  function(default, args){
    if (missing(args))
      args <- as.list(parent.frame())
    args <- args[ !vapply(args, is.name, T) ]
    sapply(unique(c(names(default), names(args))),
      simplify = F,
      function(name){
        if (any(name == names(args)))
          args[[ name ]]
        else
          default[[ name ]]
      })
  }

.fresh_param2 <-
  function(default, args){
    if (missing(args))
      return(default)
    if (length(args) == 0)
      return(default)
    .fresh_param(default, args)
  }

.fresh_param2f <-
  function(default, args, class = "gpar"){
    structure(.fresh_param2(default, args), class = class)
  }
```

```
.check_columns <-
  function(obj, lst, tip){
    if (!is.data.frame(obj))
      stop(paste0("'", tip, "' must be a 'data.frame'."))
    lapply(lst, function(col){
      if (is.null(obj[[ col ]]))
        stop(paste0("'", tip, "' must contains a column of '", col, "'."))
      })
  }

.message_info_viewport <-
  function(info = "info"){
    .message_info(info, "current.viewport:",
      paste0("\n\t", paste0(grid::current.viewport())))
  }

.message_info <-
  function(main, sub, arg = NULL, sig = "##"){
    message(sig, " ", main, ": ", sub, " ", arg)
  }

.suggest_bio_package <-
  function(pkg){
    if (!requireNamespace(pkg, quietly = T))
      stop("package '", pkg, "' not installed. use folloing to install:\n",
        '\nif (!require("BiocManager", quietly = TRUE))',
        '\n\tinstall.packages("BiocManager")',
        '\nBiocManager::install("', pkg, '")\n\n')
  }

#' @export tmp_pdf
#' @aliases tmp_pdf
#' @description \code{tmp_pdf}: ...
#' @rdname utilites
tmp_pdf <- function() {
  paste0(tempdir(), "/tmp_pdf.pdf")
}

#' @export op
#' @aliases op
#' @description \code{op}: ...
```

```r
#' @rdname utilites
op <- function(file) {
  system(paste0("xdg-open ", file))
}

#' @export .cairosvg_to_grob
#' @aliases .cairosvg_to_grob
#' @description \code{.cairosvg_to_grob}: Convert cairo svg to 'grob'.
#' @rdname utilites
.cairosvg_to_grob <-
  function(path){
    grImport2::grobify(grImport2::readPicture(path))
  }

#' @export .as_dic
#' @aliases .as_dic
#' @description \code{.as_dic}: ...
#' @rdname utilites
.as_dic <-
  function(vec, names, default,
    fill = T, as.list = T, na.rm = F){
    if (is.null(names(vec)))
      names(vec) <- names[1:length(vec)]
    if (fill) {
      if (any(!names %in% names(vec))) {
        ex.names <- names[!names %in% names(vec)]
        ex <- rep(default, length(ex.names))
        names(ex) <- ex.names
        vec <- c(vec, ex)
      }
    }
    if (as.list) {
      if (!is.list(vec))
        vec <- as.list(vec)
    }
    if (na.rm) {
      vec <- vec[!is.na(names(vec))]
    }
    vec
  }
```

```r
#' @export fill_list
#' @aliases fill_list
#' @description \code{fill_list}: ...
#' @rdname utilites
fill_list <- function(names, vec, default = vec[1]) {
  .as_dic(vec, names, default, fill = T, as.list = F, na.rm = F)
}

#' @export n
#' @aliases n
#' @description \code{n}: ...
#' @rdname utilites
n <- function(name, n){
  name <- as.character(substitute(name))
  paste0(name, 1:n)
}

#' @export namel
#' @aliases namel
#' @description \code{namel}: ...
#' @rdname utilites
namel <- function(...){
  call <- substitute(list(...))
  lst <- list(...)
  if (is.null(names(call))) {
    names <- lapply(2:length(call), function(n) as.character(call)[n])
  } else {
    names <- vapply(2:length(call), FUN.VALUE = character(1),
      function(n) {
        if (names(call)[n] == "")
          as.character(call)[n]
        else
          names(call)[n]
      })
  }
  names(lst) <- names
  lst
}

#' @export repSuffix
#' @aliases repSuffix
```

```r
#' @description \code{repSuffix}: ...
#' @rdname utilites
repSuffix <-
  function(chs, anno = ".rep."){
    gsub(paste0(anno, 1, "$"), "",
      vapply(1:length(chs), FUN.VALUE = character(1),
        function(n){
          paste0(chs[n], anno, length(chs[1:n][ chs[1:n] == chs[n] ]))
        }))
  }

#' @export %>%
#' @aliases %>%
#' @description \code{%>%}: ...
#' @rdname utilites
`%>%` <- magrittr::`%>%`

#' @export %<>%
#' @aliases %<>%
#' @description \code{%<>%}: ...
#' @rdname utilites
`%<>%` <- magrittr::`%<>%`

#' @export .expath
#' @aliases .expath
#' @description \code{.expath}: ...
#' @rdname utilites
.expath <- function() {
  .expath <- system.file("extdata", ".", package = gsub("^.*:", "", environmentName(topenv())))
  assign('.expath', .expath, envir = topenv())
}

.onLoad <- function(libname, pkgname) {
  .expath()
  .expathsvg()
  .check_external_svg()
}

#' @export agroup
#' @aliases agroup
#' @description \code{agroup}: ...
```

```r
#' @rdname utilites
agroup <- function(group, value, FUN.VALUE = character(1)) {
  ug <- unique(group)
  if (length(ug) > length(value))
    stop( "the length of 'value' not enough to assign" )
  dic <- .as_dic(value, ug, fill = F, na.rm = F)
  vapply(group, function(g) dic[[g]], FUN.VALUE)
}

#' @export write_tsv
#' @aliases write_tsv
#' @description \code{write_tsv}: ...
#' @rdname utilites
write_tsv <-
  function(x, filename, col.names = T, row.names = F){
    write.table(x, file = filename, sep = "\t",
      col.names = col.names, row.names = row.names, quote = F)
  }

#' @export read_tsv
#' @aliases read_tsv
#' @description \code{read_tsv}: ...
#' @rdname utilites
read_tsv <- function(path){
  file <- data.table::fread(input = path, sep = "\t",
    header = T, quote = "", check.names = F)
  return(file)
}

#' @export mapply_rename_col
#' @aliases mapply_rename_col
#' @description \code{mapply_rename_col}: ...
#' @rdname utilites
mapply_rename_col <-
  function(
    mutate_set,
    replace_set,
    names,
    fixed = F
    ){
    envir <- environment()
```

```
    mapply(mutate_set, replace_set,
      MoreArgs = list(envir = envir, fixed = fixed),
      FUN = function(mutate, replace, envir,
        fixed = F, names = get("names", envir = envir)){
        names <- gsub(mutate, replace, names, perl = ifelse(fixed, F, T), fixed = fixed)
        assign("names", names, envir = envir)
      })
    return(names)
  }

#' @export turn_vector
#' @aliases turn_vector
#' @description \code{turn_vector}: ...
#' @rdname utilites
turn_vector <- function(vec) {
  names <- names(vec)
  names(vec) <- unname(vec)
  vec[] <- names
  vec
}

#' @export group_switch
#' @aliases group_switch
#' @description \code{group_switch}: ...
#' @rdname utilites
group_switch <- function(data, meta.lst, by) {
  if (!is.character(data[[ by ]]))
    stop( "is.character(data[[ by ]]) == F" )
  meta <- unlist(meta.lst)
  names(meta) <- rep(names(meta.lst), lengths(meta.lst))
  meta <- as.list(turn_vector(meta))
  data <- data[data[[by]] %in% names(meta), ]
  group <- data.frame(order = 1:length(data[[ by ]]), col = data[[ by ]])
  group <- split(group, ~ col)
  group <- lapply(names(group),
    function(name){
      data <- group[[ name ]]
      data$col <- meta[[ name ]]
      return(data)
    })
  group <- data.table::rbindlist(group)
```

```
  group <- group[order(group$order), ]$col
  split(data, group)
}

#' @export .find_and_sort_strings
#' @aliases .find_and_sort_strings
#' @description \code{.find_and_sort_strings}: ...
#' @rdname utilites
.find_and_sort_strings <-
  function(strings, patterns){
    lapply(patterns,
      function(pattern){
        strings[grepl(pattern, strings, perl = T)]
      })
  }

#' @export maps
#' @aliases maps
#' @description \code{maps}: ...
#' @rdname utilites
maps <- function(data, value, from, to) {
  if (!is.list(value))
    value <- list(value)
  lapply(value,
    function(value) {
      data <- data[data[[from]] %in% value, ]
      vec <- data[[ to ]]
      names(vec) <- data[[ from ]]
      vec
    })
}

#' @export order_list
#' @aliases order_list
#' @description \code{order_list}: ...
#' @rdname utilites
order_list <-
  function(
    list
    ){
    lt <- list()
```

```
    length(lt) <- length(list)
    names(lt) <- sort(names(list))
    for(i in names(lt)){
      lt[[i]] <- list[[i]]
    }
    return(lt)
  }

#' @export molconvert_structure
#' @aliases molconvert_structure
#' @description \code{molconvert_structure}: ...
#' @rdname utilites
## use 'molconvert' ...
## https://chemaxon.com/marvin
molconvert_structure <-
  function(smile, path){
    system(paste0("molconvert mol \"", smile, "\" -o ", path))
    src <- paste(readLines(path), collapse = "\n")
    ChemmineOB::convertToImage("MOL", "SVG", source = src, toFile = path)
    rsvg::rsvg_svg(path, path)
  }

#' @export obj.size
#' @aliases obj.size
#' @description \code{obj.size}: ...
#' @rdname utilites
obj.size <- function(x, ...) {
  format(object.size(x), units = "MB", ...)
}
```

2 File: alignment_merge.R

```
# ============================================================================
# Merge data tables based on continuous variables.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases align_merge
#'
#' @title Alignment merge of 'features' via m/z and RT
#'
#' @description Merge two data.frame of 'features' according to m/z and RT.
```

```r
#' The formula is the same in MZmine2:
#' Score = (1 - rt.difference / rt.tolerance) * rt.weight +
#' (1 - mz.defference / mz.tolerance) * mz.weight
#'
#' @name align_merge
NULL
#> NULL

#' @export align_merge
#' @aliases align_merge
#' @description \code{align_merge}: ...
#' @rdname align_merge
align_merge <-
  function(main, sub, id = ".features_id",
    mz.main = "mz", mz.sub = "mz",
    rt.main = "rt.min", rt.sub = "rt.min",
    mz.tol = .05, rt.tol = .3, mz.weight = 75, rt.weight = 25,
    unique = T
    ) {
    lst <- list(main = main, sub = sub)
    check_col <- function(col.x, col.y, lst) {
      if (col.x == col.y) {
        lst <<- coln_suffix(lst, col.x)
        cols <- paste0(col.x, c(".main", ".sub"))
      } else {
        cols <- c(col.x, col.y)
      }
      return(cols)
    }
    mz.cols <- check_col(mz.main, mz.sub, lst)
    rt.cols <- check_col(rt.main, rt.sub, lst)
    data <- tol_merge(lst[[1]], lst[[2]], main_col = mz.cols[1],
      sub_col = mz.cols[2], tol = mz.tol)
    data <- dplyr::mutate(data, .rt_diff = abs(.data[[rt.cols[1]]] - .data[[rt.cols[2]]]),
      .mz_diff = abs(.data[[mz.cols[1]]] - .data[[mz.cols[2]]]))
    data <- dplyr::filter(data, .rt_diff < !!rt.tol)
    data <- dplyr::mutate(data, .score = (1 - .rt_diff / !!rt.tol) * rt.weight +
      (1 - .mz_diff / !!mz.tol) * mz.weight)
    data <- dplyr::arrange(data, .data[[id]], dplyr::desc(.score))
    if (unique)
      data <- dplyr::distinct(data, .data[[id]], .keep_all = T)
```

```
    data <- dplyr::select(data, -.rt_diff, -.mz_diff, -.score)
    tibble::as_tibble(data)
  }

#' @export tol_merge
#' @aliases tol_merge
#' @description \code{tol_merge}: ...
#' @rdname align_merge
tol_merge <-
  function(main,
    sub,
    main_col = "mz",
    sub_col = "mz",
    tol = 0.002,
    bin_size = 1
    ){
    if (main_col == sub_col) {
      new_name <- paste0(sub_col, ".sub")
      colnames(sub)[colnames(sub) == sub_col] <- new_name
      sub_col <- new_name
    }
    main$...seq <- 1:nrow(main)
    backup <- main
    ## to reduce computation, round numeric for limitation
    ## main
    main$...id <- round(main[[ main_col ]], bin_size)
    ## sub
    sub.x <- sub.y <- sub
    sub.x$...id <- round(sub.x[[ sub_col ]], bin_size)
    sub.y$...id <- sub.x$...id + ( 1 * 10^-bin_size )
    sub <- rbind(sub.x, sub.y)
    ## expand merge
    df <- merge(main, sub, by = "...id", all.x = T, allow.cartesian = T)
    df$...diff <- abs(df[[ main_col ]] - df[[ sub_col ]])
    df <- dplyr::filter(df, ...diff <= !!tol)
    ## get the non-merged
    backup <- backup[!backup$...seq %in% df$...seq, ]
    df <- dplyr::bind_rows(df, backup)
    ## remove the assist col
    dplyr::select(df, -...id, -...diff, -...seq)
  }
```

```r
`:=` <- rlang::`:=`

coln_suffix <-
  function(lst, col,
    suffix = c(".main", ".sub",
      ifelse(length(lst) <= 2, character(0),
        paste0(".other", 1:(length(lst) - 2))))
    ) {
    lapply(1:length(lst),
      function(n) {
        dplyr::rename(lst[[n]], !!paste0(col, suffix[n]) := col)
      })
  }
```

3 File: cross_select.R

```r
# ========================================================================
# filter data according to colunm within another data.frame
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases select_features
#'
#' @title Select 'features' for MCnebula2
#'
#' @description Select significant 'features' from MCnebula2 with
#' statistic results for downstream analysis of metabolomics.
#'
#' @name select_features
NULL
#> NULL

#' @export select_features
#' @aliases select_features
#' @description \code{select_features}: ...
#' @rdname select_features
select_features <- function(
  mcn, classes = unique(nebula_index(mcn)$class.name),
  q.value = .05, logfc = .3, coef = NULL, tani.score_cutoff = NULL,
  order_by_coef = NULL, togather = F)
{
```

```
  if (!requireNamespace("MCnebula2", quietly = T))
    stop("package 'MCnebula2' must be available.")
  .check_data(statistic_set(mcn), list(top_table = "binary_comparison"))
  .check_data(mcn, list(nebula_index = "create_nebula_index",
      features_annotation = "create_features_annotation"))
  stat <- top_table(statistic_set(mcn))
  if (!is.null(coef)) {
    stat <- stat[coef]
  }
  stat <- data.frame(data.table::rbindlist(stat))
  data.lst <- list(nebula_index(mcn), stat)
  filter.lst <- list(
    rlang::quos(class.name %in% dplyr::all_of(classes)),
    rlang::quos(adj.P.Val < q.value, abs(logFC) > logfc)
  )
  if (!is.null(tani.score_cutoff)) {
    data.lst[[3]] <- features_annotation(mcn)
    filter.lst[[3]] <- rlang::quos(tani.score >= tani.score_cutoff)
  }
  res <- cross_select(data.lst, filter.lst, ".features_id", "class.name")
  if (!is.null(order_by_coef)) {
    ranks <- top_table(statistic_set(mcn))[[ order_by_coef ]]$.features_id
    res <- lapply(res,
      function(ids) {
        ranks[ ranks %in% ids ]
      })
  }
  if (togather) {
    res <- unlist(res, use.names = F)
    res <- ranks[ ranks %in% res ]
  }
  return(res)
}

#' @export cross_select
#' @aliases cross_select
#' @description \code{cross_select}: ...
#' @rdname select_features
cross_select <- function(data.lst, filter.lst, target, split = NULL) {
  if (!is.list(data.lst) | !is.list(filter.lst))
    stop("`data.lst` and `filter.lst` must be 'list'.")
```

```
  if (length(data.lst) != length(filter.lst))
    stop("`data.lst` and `filter.lst` must be 'list' with the same length.")
  lst <- lapply(1:length(data.lst),
    function(n) {
      if (!is.null(filter.lst[[n]]))
        dplyr::filter(data.lst[[n]], !!!(filter.lst[[n]]))
      else
        data.lst[[n]]
    })
  fun <- function(res, lst) {
    for (i in 2:length(lst)) {
      res <- res[res %in% lst[[ i ]]]
    }
    return(res)
  }
  if (is.null(split)) {
    lst <- lapply(lst, function(data) data[[ target ]])
    res <- fun(lst[[1]], lst)
  } else {
    res <- lapply(split(lst[[1]], lst[[1]][[ split ]]),
      function(data) data[[ target ]])
    lst <- lapply(lst, function(data) data[[ target ]])
    res <- lapply(res, fun, lst = lst)
  }
  return(res)
}

#' @importFrom rlang as_label
.check_data <-
  function(object, lst, tip = "(...)"){
    target <- rlang::as_label(substitute(object))
    mapply(lst, names(lst), FUN = function(value, name){
      obj <- match.fun(name)(object)
      if (is.null(obj)) {
        stop(paste0("is.null(", name, "(", target, ")) == T. ",
            "use `", value, tip, "` previously."))
      }
      if (is.list(obj)) {
        if (length(obj) == 0) {
          stop(paste0("length(", name, "(", target, ")) == 0. ",
              "use `", value, tip, "` previously."))
```

```
      }
    }
    })
  }
```

4 File: dot_heatmap.R

```
# ========================================================================
# heat map with ggplot2
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases plot_heatmap
#'
#' @title Plot heat map with ggplot2
#'
#' @description According to list of 'ID' to draw mutiple heatmap...
#'
#' @name plot_heatmap
NULL
#> NULL

#' @export plot_heatmap
#' @aliases plot_heatmap
#' @description \code{plot_heatmap}: ...
#' @rdname plot_heatmap
plot_heatmap <- function(id.lst, data, metadata,
  pal_class = ggsci::pal_futurama()(12), pal_group,
  clust_row = T, clust_col = T, method = 'complete')
{
  if (is.null(names(id.lst))) {
    stop("is.null(names(id.lst)) == T. The names of `id.lst` should be chemical classes.")
  }
  if (is.null(names(pal_class))) {
    pal_class <- pal_class[1:length(id.lst)]
    names(pal_class) <- names(id.lst)
  }
  .check_columns(metadata, c("sample", "group"), "metadata")
  .check_columns(data, c(".features_id", "sample", "value"), "data")
  lst <- sapply(names(id.lst), simplify = F,
    function(class.name) {
      ## basic heatmap
```

```
      ids <- id.lst[[ class.name ]]
      data <- dplyr::filter(data, .data$.features_id %in% dplyr::all_of(ids))
      p <- tile_heatmap(data)
      ## chemical classes
      data.class <- data.frame(class = class.name, .features_id = ids)
      pal_class <- pal_class[names(pal_class) == class.name]
      p <- add_ygroup.tile.heatmap(data.class, p, pal_class)
      ## cluster tree
      if (clust_row | clust_col) {
        data.w <- tidyr::spread(data, .data$sample, .data$value)
        data.w <- data.frame(data.w)
        rownames(data.w) <- data.w$.features_id
        data.w <- dplyr::select(data.w, dplyr::all_of(metadata[[ "sample" ]]))
        p <- add_tree.heatmap(
          data.w, p, method = method,
          clust_row = clust_row, clust_col = clust_col
        )
      }
      ## sample metadata
      p <- add_xgroup.tile.heatmap(metadata, p, pal_group)
      return(p)
    })
  return(lst)
}

#' @export handling_na
#' @aliases handling_na
#' @description \code{handling_na}:
#' For each subset of data, the missing values will be filled with the average
#' value; if the set is all missing values, they will be filled with zero.
#' @rdname plot_heatmap
handling_na <- function(data, id.cols = c(".features_id"),
  metadata, sample.col = "sample", group.col = "group")
{
  metadata <- metadata[, c(sample.col, group.col)]
  metadata <- split(metadata, metadata[[ group.col ]])
  id.cols <- data[, id.cols]
  data <- lapply(names(metadata),
    function(group) {
      meta <- metadata[[ group ]]
      df <- data[, meta[[ sample.col ]]]
```

```
      lst <- apply(df, 1, simplify = F,
        function(vec) {
          if (all(is.na(vec))) {
            vec[] <- 0
          } else if (any(!is.na(vec))) {
            vec[is.na(vec)] <- mean(vec, na.rm = T)
          }
          dplyr::bind_rows(vec)
        })
      data.table::rbindlist(lst)
    })
  data <- do.call(dplyr::bind_cols, data)
  dplyr::bind_cols(id.cols, data)
}

#' @export log_trans
#' @aliases log_trans
#' @description \code{log_trans}:
#' Convert wide data to long data; log transform the values; if there is a
#' value 0, replace it with 1/10 of the minimum value of the value column.
#' @rdname plot_heatmap
log_trans <- function(data, id.cols = c(".features_id"),
  key = "sample", value = "value",
  set_min = T, factor = 10, fun = log2, center = T)
{
  data <- tidyr::gather(data, !!key, !!value, -dplyr::all_of(id.cols))
  if (set_min) {
    min <- min(dplyr::filter(data, .data[[ value ]] != 0)[[ value ]])
    data[[ value ]] <- ifelse(data[[ value ]] == 0, min / factor, data[[ value ]])
  }
  data[[ value ]] <- fun(data[[ value ]])
  if (center) {
    data[[ value ]] <- scale(data[[ value ]], scale = F)[, 1]
  }
  return(data)
}

dot_heatmap <- function(df){
  p <- ggplot(df, aes(x = sample, y = .features_id)) +
    geom_point(aes(size = abs(value), color = value), shape = 16) +
    theme_minimal() +
```

```
    guides(size = "none") +
    scale_color_gradient2(low = "#3182BDFF", high = "#A73030FF") +
    theme(text = element_text(family = .font),
      axis.text.x = element_text(angle = 90))
    return(p)
}

## long data
tile_heatmap <-
  function(df){
    p <- ggplot(df, aes(x = sample, y = .features_id)) +
      geom_tile(aes(fill = value),
        color = "white", height = 1, width = 1, size = 0.2) +
      theme_minimal() +
      scale_fill_gradient2(low = "#3182BDFF", high = "#A73030FF",
        limits = c(min(df$value), max(df$value))) +
      labs(x = "Sample", y = "Feature ID", fill = "log2 (Feature level)") +
      theme(text = element_text(family = .font, face = "bold"),
        axis.text = element_text(face = "plain"),
        axis.text.x = element_blank()
      )
      return(p)
  }

#' @export add_tree.heatmap
#' @aliases add_tree.heatmap
#' @description \code{add_tree.heatmap}: ...
#' @rdname plot_heatmap
add_tree.heatmap <-
  function(df, p, clust_row = T, clust_col = T, method = 'complete'){
    if (clust_row) {
      phr <- hclust(dist(df), method)
      phr <- ggtree::ggtree(phr, layout = "rectangular", branch.length = "branch.length") +
        theme(plot.margin = unit(c(0, 0, 0, 0), "cm"))
      p <- aplot::insert_left(p, phr, width = 0.3)
    }
    if (clust_col) {
      phc <- hclust(dist(t(df)), method)
      phc <- ggtree::ggtree(phc, layout = "rectangular", branch.length = "branch.length") +
        ggtree::layout_dendrogram() +
        theme(plot.margin = unit(c(0, 0, 0, 0), "cm"))
```

```
      p <- aplot::insert_top(p, phc, height = 0.3)
    }
    return(p)
  }

#' @export add_xgroup.heatmap
#' @aliases add_xgroup.heatmap
#' @description \code{add_xgroup.heatmap}: ...
#' @rdname plot_heatmap
add_xgroup.heatmap <-
  function(df, p){
    p.xgroup <- ggplot(df, aes(y = "Group", x = sample)) +
      geom_point(aes(color = group), size = 6) +
      ggsci::scale_color_simpsons() +
      labs(x = "", y = "", fill = "Group") +
      theme_minimal() +
      theme(
        axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        text = element_text(family = .font, face = "bold"),
        plot.margin = unit(c(0, 0, 0, 0), "cm")
      )
      com <- aplot::insert_bottom(p, p.xgroup, height = 0.05)
      return(com)
  }

#' @export add_xgroup.tile.heatmap
#' @aliases add_xgroup.tile.heatmap
#' @description \code{add_xgroup.tile.heatmap}: ...
#' @rdname plot_heatmap
add_xgroup.tile.heatmap <-
  function(df, p, pal = NA){
    expr.pal <- ifelse(is.na(pal),
      'ggsci::scale_fill_simpsons()',
      'scale_fill_manual(values = pal)')
    p.xgroup <- ggplot(df, aes(y = "Group", x = sample)) +
      geom_tile(aes(fill = group),
        color = "white", height = 1, width = 1, size = 0.2) +
      eval(parse(text = expr.pal)) +
      labs(x = "", y = "", fill = "Group") +
      theme_minimal() +
```

```
      theme(
        axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        text = element_text(family = .font, face = "bold"),
        plot.margin = unit(c(0, 0, 0, 0), "cm")
      )
      com <- aplot::insert_bottom(p, p.xgroup, height = 0.05)
      return(com)
  }

#' @export add_ygroup.tile.heatmap
#' @aliases add_ygroup.tile.heatmap
#' @description \code{add_ygroup.tile.heatmap}: ...
#' @rdname plot_heatmap
add_ygroup.tile.heatmap <-
  function(df, p, pal = NA){
    expr.pal <- ifelse(is.na(pal),
      'ggsci::scale_fill_npg()',
      'scale_fill_manual(values = pal)')
    p.ygroup <- ggplot(df, aes(x = "Class", y = .features_id)) +
      geom_tile(aes(fill = class),
        color = "white", height = 1, width = 1, size = 0.2) +
      labs(x = "", y = "", fill = "From") +
      eval(parse(text = expr.pal)) +
      theme_minimal() +
      theme(
        axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        text = element_text(family = .font, face = "bold"),
        plot.margin = unit(c(0, 0, 0, 0), "cm")
      )
      com <- aplot::insert_left(p, p.ygroup, width = 0.02)
      return(com)
  }
```

5 File: exReport.R

```
# ==========================================================================
# extra tool for report system
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```r
# gather_sections <- function(prefix = "s", envir = parent.frame(),
#   sort = T, get = T)
# {
#   objs <- ls(envir = envir)
#   sections <- objs[ grepl(paste0("^", prefix, "[0-9]"), objs) ]
#   if (sort) {
#     num <- stringr::str_extract(
#       sections, paste0("(?<=", prefix, ")[0-9.]{0,}[0-9]")
#     )
#     sections <- sections[order(as.numeric(num))]
#   }
#   if (get) {
#     sections <- sapply(sections, get, envir = envir, simplify = F)
#   }
#   sections
# }

#' @export .workflow_name
.workflow_name <-
  substitute(
    c("Abstract" = 1, "Introduction" = 1, "Set-up" = 1,
      "Integrate data and Create Nebulae" = 1,
      "Initialize analysis" = 2,
      "Filter candidates" = 2,
      "Filter chemical classes" = 2,
      "Create Nebulae" = 2,
      "Visualize Nebulae" = 2,
      "Nebulae for Downstream analysis" = 1,
      "Statistic analysis" = 2,
      "Set tracer in Child-Nebulae" = 2,
      "Quantification in Child-Nebulae" = 2,
      "Annotate Nebulae" = 2,
      "Query compounds" = 2,
      "Pathway enrichment" = 2,
      "Session infomation" = 1
      ))
```

6 File: grid_draw.R

```r
# ============================================================================
# class
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass graph
#'
#' @aliases graph
#'
#' @title A class built on 'grobs'
#'
#' @description ...
#'
#' @rdname graph-class
#'
#' @examples
#' \dontrun{
#' new('graph', ...)
#' }
graph <-
  setClass("graph",
    contains = character(),
    representation =
      representation(grob = "ANY",
        cvp = "ANY"
        ),
      prototype = NULL
  )

.grob_class <- c("grob", "frame", "gTree", "null",
  "text", "circle", "segments", "gtable",
  "curve", "polygon", "rastergrob")
setOldClass(.grob_class)
setOldClass("viewport")
setClassUnion("grob.obj", .grob_class)

.gg <- c("gg", "ggplot", "ggraph")
setOldClass(.gg)
setClassUnion("gg.obj", .gg)

.class_unit <- c("unit", "simpleUnit", "unit_v2")
setOldClass(.class_unit)
setClassUnion("units", .class_unit)
```

```r
# ============================================================================
# color
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @export .default_color
.default_color <- ggsci::pal_npg()(9)
```

```r
# ============================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases draw
#'
#' @title draw class of 'graph' and 'grobs'
#'
#' @description ...
#'
#' @name draw-methods
NULL
#> NULL

setGeneric("draw",
  function(x, content)
    standardGeneric("draw"))

#' @exportMethod draw
setMethod("draw",
  signature = c(x = "graph", content = "grob.obj"),
  function(x, content){
    grid.draw(x@grob)
    pushViewport(x@cvp)
    grid.draw(content)
    upViewport(1)
  })

#' @exportMethod draw
setMethod("draw",
  signature = setMissing("draw",
    x = "graph"),
  function(x){
    grid.draw(x@grob)
  })
```

```
#' @exportMethod draw
setMethod("draw",
  signature = c(x = "grob.obj"),
  function(x){
    grid.draw(x)
  })

#' @exportMethod into
#' @title place 'grobs' into 'graph'
#' @description ...
#' @rdname into-methods
setGeneric("into",
  function(x, content) standardGeneric("into"))

#' @exportMethod into
setMethod("into",
  signature = c(x = "graph", content = "grob.obj"),
  function(x, content){
    if (is.null(content$vp)) {
      content$vp <- x@cvp
    } else {
      content$vp <- vpStack(x@cvp, content$vp)
    }
    gTree(children = gList(x@grob, content))
  })

#' @exportMethod setvp
#' @title ...
#' @description ...
#' @rdname setvp-methods
setGeneric("setvp",
  function(x, ...) standardGeneric("setvp"))

#' @exportMethod setvp
setMethod("setvp",
  signature = c(x = "ANY"),
  function(x, ...){
    viewport(grobX(x, 90), grobY(x, 0),
      grobWidth(x), grobHeight(x), ...)
  })
```

```r
#' @exportMethod weight
setGeneric("weight",
  function(x, sub) standardGeneric("weight"))
setMethod("weight",
  signature = c(x = "ANY", sub = "character"),
  function(x, sub){
    if (isS4(x)) {
      weight <-
        sapply(sub, simplify = F, function(sub) {
          get_weight(slot(x, sub))
        })
    }
    as.list(sort(unlist(weight), decreasing = T))
  })

#' @exportMethod as_grob
#' @title convert 'ggplot' object to 'grobs'
#' @description ...
#' @rdname as_grob-methods
setGeneric("as_grob",
  function(x) standardGeneric("as_grob"))

#' @exportMethod as_grob
setMethod("as_grob",
  signature = c(x = "gg.obj"),
  function(x){
    ggplot2::ggplot_gtable(ggplot2::ggplot_build(x))
  })

#' @export get_weight
get_weight <- function(x){
  if (isS4(x)) {
    n <- length(slotNames(x))
    if (n == 1) {
      if (is.list(slot(x, slotNames(x)))) {
        n <- n * length(slot(x, slotNames(x)))
      }
    }
    return(n)
  } else if (is.list(x)) {
    length(x)
```

```
  } else {
    length(x)
  }
}

#' @aliases frame
#'
#' @title draw in grid frame
#'
#' @description ...
#'
#' @name frame
NULL
#> NULL

#' @export layout_row
#' @aliases layout_row
#' @description \code{layout_row}: ...
#' @rdname frame
layout_row <- function(weight){
  grid.layout(length(weight), 1, heights = weight)
}

#' @export frame_row
#' @aliases frame_row
#' @description \code{frame_row}: ...
#' @rdname frame
frame_row <- function(weight, data, if.ex){
  do.call(frame_place, .fresh_param(list(type = "row")))
}

#' @export layout_col
#' @aliases layout_col
#' @description \code{layout_col}: ...
#' @rdname frame
layout_col <- function(weight){
  grid.layout(1, length(weight), widths = weight)
}

#' @export frame_col
#' @aliases frame_col
```

```
#' @description \code{frame_col}: ...
#' @rdname frame
frame_col <- function(weight, data, if.ex){
  do.call(frame_place, .fresh_param(list(type = "col")))
}

#' @exportMethod frame_place
#' @aliases frame_place
#' @description \code{frame_place}: ...
#' @rdname frame
setGeneric("frame_place",
  function(weight, data, type, if.ex)
    standardGeneric("frame_place"))

#' @exportMethod frame_place
setMethod("frame_place",
  signature = setMissing("frame_place",
    weight = "vector",
    data = "list",
    type = "character"),
  function(weight, data, type){
    fun <- paste0("layout_", type)
    layout <- match.fun(fun)(weight)
    frame <- frameGrob(layout = layout)
    data <- sapply(names(data), simplify = F,
      function(name) {
        grob <- data[[ name ]]
        if (is(grob, "graph"))
          grob <- grob@grob
        gTree(children = gList(grob),
          vp = viewport(name = name))
      })
    names(weight) <- repSuffix(names(weight))
    for (i in 1:length(weight)) {
      i.name <- names(weight)[[ i ]]
      o.name <- gsub("\\.rep\\.[0-9]{1,}$", "", i.name)
      i.grob <- data[[ o.name ]]
      i.grob$vp$name <- i.name
      args <- list(frame, i.grob)
      args[[ type ]] <- i
      frame <- do.call(placeGrob, args)
```

```
    }
    frame
  })
setMethod("frame_place",
  signature = setMissing("frame_place",
    weight = "vector",
    data = "list",
    type = "character",
    if.ex = "logical"),
  function(weight, data, type, if.ex){
    main <- weight[!if.ex]
    sub <- weight[if.ex]
    funs <- list(frame_col, frame_row)
    ## which function
    which <- type == c("col", "row")
    ## ex
    ex <- paste0(names(sub), collapse = "__")
    data[[ ex ]] <- funs[!which][[1]](sub, data)
    ex.w <- list(sum(unlist(sub)))
    names(ex.w) <- ex
    weight <- c(main, ex.w)
    ## main
    funs[which][[1]](weight, data)
  })

#' @aliases setnull
#'
#' @title Set markers crossover viewports
#'
#' @description ...
#'
#' @name setnull
NULL
#> NULL

#' @export setnull
#' @aliases setnull
#' @description \code{setnull}: ...
#' @rdname setnull
setnull <- function(target, args, name = "null"){
  gPath <- grid.grep(gPath(target), vpPath = T, grep = T)
```

```
  vpPath <- attr(gPath, "vpPath")
  args <- .fresh_param2(list(name = name, vp = vpPath), args)
  do.call(nullGrob, args)
}

#' @export setnullvp
#' @aliases setnullvp
#' @description \code{setnullvp}: ...
#' @rdname setnull
setnullvp <- function(pattern, args, x, name = NULL, fix = T, perl = F){
  if (fix) pattern <- paste0("::", pattern, "$")
  vpPath <- gsub("ROOT::", "", grepPath(pattern, x = x, perl = perl)[1])
  vpPath <- vpPath(vpPath)
  args <- .fresh_param2(list(name = name, vp = vpPath), args)
  do.call(nullGrob, args)
}

#' @export ruler
ruler <- function(p1, p2){
  segmentsGrob(grobX(p1, 0), grobY(p1, 0),
    grobX(p2, 0), grobY(p2, 0))
}

#' @export grepPath
#' @aliases grepPath
#' @description \code{grepPath}: ...
#' @rdname setnull
grepPath <-
  function (pattern, x = NULL, grobs = T, viewports = T, perl = F) {
    args <- list(x = x, grobs = grobs, viewports = viewports, print = F)
    dl <- do.call(grid.ls, args)
    if (viewports) {
      keep <- dl$type == "vpListing" | dl$type == "grobListing" |
        dl$type == "gTreeListing"
    } else {
      keep <- dl$type == "grobListing" | dl$type == "gTreeListing"
    }
    vpPaths <- dl$vpPath[keep]
    vpPaths[grepl(pattern, vpPaths, perl = perl)]
  }
```

```
#' @export sort_vpPaths
#' @aliases sort_vpPaths
#' @description \code{sort_vpPaths}: ...
#' @rdname setnull
sort_vpPaths <- function(vpPaths){
  nums <- vapply(vpPaths, FUN.VALUE = 0,
    function(ch) {length(grepRaw("::", ch, all = T))})
  lapply(order(nums), function(n) vpPaths[[ n ]])
}

#' @export u
u <- function(n, unit){
  unit <- as.character(substitute(unit))
  unit(n, unit)
}

#' @export vptest
vptest <- function(r = .7, fill = "lightblue"){
  x11(width = 7, height = 7 * r,)
  pushViewport(viewport(, , .5, .5, gp = gpar(fill = fill)))
}

#' @export sym_chem
sym_chem <- function(smi){
  tmpsvg <- paste0(tempdir(), "/tempsvg.svg")
  ChemmineOB::convertToImage("SMI", "SVG", source = smi, toFile = tmpsvg)
  svgtxt <- readLines(tmpsvg)
  svgtxt <- gsub("stroke-width=\"2.0", "stroke-width=\"4.0", svgtxt)
  writeLines(svgtxt, tmpsvg)
  rsvg::rsvg_svg(tmpsvg, tmpsvg)
  .rm_background(.cairosvg_to_grob(tmpsvg))
}

#' @aliases ggather
#'
#' @title a mutate of grid::gTree
#'
#' @description ...
#'
#' @name ggather
NULL
```

```
#> NULL

#' @export ggather
#' @aliases ggather
#' @description \code{ggather}: ...
#' @rdname ggather
ggather <- function(..., vp = NULL, gp = NULL){
  objs <- list(...)
  objs <- lapply(objs, function(obj) {
    if (is(obj, "graph"))
      obj@grob
    else
      obj
    })
  glist <- do.call(gList, objs)
  gTree(children = glist, gp = gp, vp = vp)
}

#' @export zo
#' @aliases zo
#' @description \code{zo}: ...
#' @rdname ggather
zo <- function(x, w = .9, h = .9) {
  ggather(x, vp = viewport(, , w, h))
}
```

```
# ============================================================================
# arrow
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases arrow
#'
#' @title draw arrow
#'
#' @description ...
#'
#' @name arrow
NULL
#> NULL

#' @export .gpar_dashed_line
.gpar_dashed_line <- gpar(fill = "black", lty = "dashed", lwd = unit(2, "line"))
```

```r
#' @export .gpar_dotted_line
.gpar_dotted_line <- gpar(fill = "black", lty = "dashed", lwd = unit(2, "line"))

#' @export parrow
#' @aliases parrow
#' @description \code{parrow}: ...
#' @rdname arrow
parrow <- function(n = 5, col, type = "dashed", lwd = u(1, line)){
  y <- seq(0, 1, length.out = n)[c(-1, -n)]
  segs <- segmentsGrob(rep(.1, n - 2), y,
    rep(.9, n - 2), y,
    gp = gpar(lwd = lwd, fill = col, col = col, lty = type)
  )
  arrs <- segmentsGrob(rep(.8, n - 2), y,
    rep(.9, n - 2), y,
    arrow = arrow(angle = 15, length = unit(.7, "line"), type = "closed"),
    gp = gpar(lwd = lwd, fill = col, col = col)
  )
  ggather(segs, arrs)
}

setClassUnion("maybe_p1p2", c("null", "list"))
setClassUnion("maybe_function", c("NULL", "function"))
setClassUnion("maybe_list", c("NULL", "list"))

#' @exportMethod garrow
#' @aliases garrow
#' @description \code{garrow}: ...
#' @rdname arrow
setGeneric("garrow",
  function(p1, p2, args_line, args_arrow, fun_line, fun_arrow, city)
    standardGeneric("garrow"))

#' @exportMethod garrow
setMethod("garrow",
  signature = setMissing("garrow"),
  function(){
    args_line <- list(square = F, ncp = 10, curvature = .3,
      gp = gpar(fill = "black"))
    args_arrow <- list(angle = 15, length = unit(.7, "line"), type = "closed")
    list(args_line = args_line,
```

```
      args_arrow = args_arrow,
      fun_line = match.fun(curveGrob),
      fun_arrow = match.fun(arrow),
      city = NULL
    )
  })

#' @exportMethod garrow
setMethod("garrow",
  signature = c(p1 = "maybe_p1p2", p2 = "maybe_p1p2"),
  function(p1, p2, args_line, args_arrow,
    fun_line, fun_arrow, city){
    args <- as.list(environment())
    default <- garrow()
    args$args_line <- .fresh_param2(default$args_line, args_line)
    args$args_arrow <- .fresh_param2(default$args_arrow, args_arrow)
    args <- .fresh_param2(default, args)
    reCallMethod("garrow", args)
  })

#' @exportMethod garrow
setMethod("garrow",
  signature = c(p1 = "null", p2 = "null",
    args_line = "list",
    args_arrow = "list",
    fun_line = "maybe_function",
    fun_arrow = "maybe_function",
    city = "maybe_list"),
  function(p1, p2, args_line, args_arrow,
    fun_line, fun_arrow, city){
    args <- as.list(environment())
    args$p1 <- list(x1 = grobX(p1, 0), y1 = grobY(p1, 0))
    args$p2 <- list(x2 = grobX(p2, 0), y2 = grobY(p2, 0))
    reCallMethod("garrow", args)
  })

#' @exportMethod garrow
setMethod("garrow",
  signature = c(p1 = "list", p2 = "list",
    args_line = "list",
    args_arrow = "list",
```

```
  fun_line = "maybe_function",
  fun_arrow = "maybe_function",
  city = "maybe_list"),
function(p1, p2, args_line, args_arrow,
  fun_line, fun_arrow, city){
  if (!is.null(fun_arrow))
    args_line$arrow <- do.call(fun_arrow, args_arrow)
  if (!is.null(city)) {
    city <- .fresh_param2(garrow_city_args(), city)
    e <- segmentsGrob(p1$x1, p1$y1, p2$x2, p2$y2)
    if (city$axis == "x") {
      pm <- list(x = p1$x1 + city$shift, y = grobY(e, 0))
    } else if (city$axis == "y") {
      pm <- list(x = grobX(e, 90), y = p1$y1 + city$shift)
    } else {
      stop("city$axis != x & city$axis != y")
    }
    args_line <- .fresh_param2(args_line, city$args_line)
    args_line_mid <- args_line[names(args_line) != "arrow"]
    names(pm) <- c("x2", "y2")
    a1 <- do.call(fun_line, c(p1, pm, args_line_mid))
    names(pm) <- c("x1", "y1")
    if (city$rev) {
      args_line$curvature <- -(args_line$curvature)
    }
    a2 <- do.call(fun_line, c(pm, p2, args_line))
    ## as graph
    vp <- viewport(pm$x, pm$y, .1 * grobHeight(e), .1 * grobHeight(e))
    if (!is.null(city$grob_anno)) {
      if (!is.null(city$vp_anno)) {
        vp <- vpStack(vp, city$vp_anno)
      }
      anno <- ggather(city$grob_anno, vp = vp)
      return(ggather(a1, a2, anno))
    }
    return(graph(grob = ggather(a1, a2), cvp = vp))
  }
  args_line <- c(p1, p2, args_line)
  do.call(fun_line, args_line)
})
```

```r
#' @export garrow_city
#' @aliases garrow_city
#' @description \code{garrow_city}: ...
#' @rdname arrow
garrow_city <- function(p1, p2, up, left, shift, gp_line){
  shift <- abs(shift)
  curvature <- 1
  if (!up & left) {
    shift <- -shift
  } else if (!up & !left) {
    curvature <- -1
  } else if (up & left) {
    curvature <- -1
    shift <- -shift
  }
  args <- list(curvature = curvature,
    square = T,
    ncp = 1)
  garrow(p1, p2, list(gp = gp_line),
    city = list(args_line = args, shift = shift))
}

#' @export garrow_snake
#' @aliases garrow_snake
#' @description \code{garrow_snake}: ...
#' @rdname arrow
garrow_snake <- function(p1, p2, color, lwd = u(1, line), cur = 1){
  garrow(p1, p2, list(curvature = cur, square = T, ncp = 1, inflect = T,
      gp = gpar(lwd = lwd, col = color, fill = color)))
}

#' @export garrow_city_args
#' @aliases garrow_city_args
#' @description \code{garrow_city_args}: ...
#' @rdname arrow
garrow_city_args <-
  function(shift = u(2, line), axis = "x", mid = .5,
    args_line = list(ncp = 1, curvature = 1, square = T),
    grob_anno = NULL, vp_anno = NULL, rev = F
    ){
    as.list(environment())
```

```
  }

#' @export sagnage
#' @aliases sagnage
#' @description \code{sagnage}: ...
#' @rdname arrow
sagnage <- function(grob, left = T, l_gpar, borderF = 1.5, front_len = .1,
  vp_shift = u(1.5, line), ...){
  l_gpar <- .fresh_param2f(gpar(linejoin = "round", fill = "grey85",
      col = "transparent"), l_gpar)
  w <- grobWidth(grob) * borderF
  h <- grobHeight(grob) * borderF
  vp <- viewport(, , w * (1 + front_len), h)
  shift <- front_len / (front_len + 1)
  cw <- 1 / (front_len + 1)
  if (left) {
    poly <- polygonGrob(c(0, front_len, 1, 1, front_len), c(.5, 1, 1, 0, 0),
      vp = vp, gp = l_gpar)
    cvp <- viewport(.5 + shift, width = cw)
  } else {
    poly <- polygonGrob(c(0, 0, 1 - front_len, 1, 1 - front_len),
      c(0, 1, 1, .5, 0), vp = vp, gp = l_gpar)
    cvp <- viewport(.5 - shift, width = cw)
  }
  ggrob <- into(graph(grob = poly, cvp = vpStack(vp, cvp)), grob)
  if (left) {
    vp <- viewport(vp_shift, just = c("left", "centre"))
  } else {
    vp <- viewport(-vp_shift, just = c("left", "centre"))
  }
  c(list(grob_anno = ggrob, vp_anno = vp), list(...))
}

#' @export sagnage_shiny
#' @aliases sagnage_shiny
#' @description \code{sagnage_shiny}: ...
#' @rdname arrow
sagnage_shiny <- function(label, left, color){
  sagnage(gtext(label, gpar(col = "white", fontface = "plain")),
    left, gpar(fill = color))$grob_anno
}
```

```r
#' @export maparrow
#' @aliases maparrow
#' @description \code{maparrow}: ...
#' @rdname arrow
maparrow <-
  function(obj, data, pattern = list(), round_cur = .3,
    pos = list(r = list(x = 1), l = list(x = 0),
      t = list(y = 1), b = list(y = 0))
    ){
    .check_columns(data, c("from", "to", "group", "fun", "color", "left", "up",
        "shift"), "data")
    if (is.null(data$dup))
      data$dup <- duplicated(data$group)
    if (is.null(data$sag))
      data$sag <- paste0(substr(form(data$fun), 1, 3), ".")
    target <- unique(c(data$from, data$to))
    nulls <- sapply(target, simplify = F,
      function(name) {
        pattern <-
          if (!is.null(pattern[[ name ]]))
            pattern[[ name ]]
          else
            name
        sapply(names(pos), simplify = F,
          function(p){
            setnullvp(pattern, pos[[p]], obj)
          })
      })
    bafs <- sapply(target, simplify = F,
      function(name) {
        null <- nulls[[ name ]]
        null <- null[names(null) %in% c("l", "r")]
        lapply(null, function(null) {
          function(w = u(1, line), h = u(3, line)) {
            baf(grobX(null, 0), grobY(null, 0), w, h)
          }
          })
      })
    bafs <- unlist(bafs, F)
    keep <- lapply(c("from", "to"),
      function(ch) {
```

```
    pos <- ifelse(data[[ "left" ]], "l", "r")
    paste0(data[[ ch ]], ".", pos)
  })
bafs <- bafs[names(bafs) %in% unique(unlist(keep))]
arr_city <- apply(data, 1, simplify = F,
  function(v) {
    pos <- ifelse(as.logical(v[[ "left" ]]), "l", "r")
    garrow_city(nulls[[ v[["from"]] ]][[ pos ]],
      nulls[[ v[["to"]] ]][[ pos ]],
      as.logical(v[[ "up" ]]), as.logical(v[[ "left" ]]),
      u(as.numeric(v[["shift"]]), line),
      gpar(fill = v[["color"]], col = v[["color"]],
        lwd = u(1, line))
    )
  })
arr_round <- apply(data, 1, simplify = F,
  function(v) {
    pos <- ifelse(as.logical(v[[ "left" ]]), "l", "r")
    cur <- if (pos == "l") round_cur else -round_cur
    cur <- if (as.logical(v[[ "up" ]])) -cur else cur
    garrow(nulls[[ v[["from"]] ]][[ pos ]],
      nulls[[ v[["to"]] ]][[ pos ]],
      list(curvature = cur,
        gp = gpar(fill = v[["color"]], col = v[["color"]],
          lwd = u(1, line)))
    )
  })
sags <- lapply(1:length(arr_city),
  function(n) {
    if (data$dup[[n]]) return()
    left <- !data$left[[n]]
    up <- data$up[[n]]
    col <- data$color[[n]]
    gtext <- gtext(data$sag[[n]], gpar(col = "white", fontface = "plain"))
    sag <- sagnage(gtext, left, gpar(fill = col), 1.5)$grob_anno
    vp <- arr_city[[n]]@cvp
    x <- if (left) u(.5, npc) + u(2, line) else u(.5, npc) - u(2, line)
    y <- if (up) .5 + 5 else .5 -5
    vp <- vpStack(vp, viewport(x, y))
    ggather(sag, vp = vp)
  })
```

```
    sags <- sags[!vapply(sags, is.null, T)]
    namel(nulls, bafs, arr_city, arr_round, sags, data)
  }

#' @export baf
baf <- function(x, y, width = u(1, line), height = u(3, line)) {
  rect <- grectn(bgp_args = gpar(lty = "solid"))@grob
  clip <- clipGrob(, , .7)
  ggather(clip, rect, vp = viewport(x, y, width, height))
}
```

```
# ============================================================================
# text
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases text
#'
#' @title a mutate of grid::textGrob
#'
#' @description ...
#'
#' @name text
NULL
#> NULL

#' @export .font
.font <- "Times"

#' @export .title_gp
.title_gp <- gpar(col = "black", cex = 1, fontfamily = .font, fontface = "bold")

#' @export gtext
#' @aliases gtext
#' @description \code{gtext}: ...
#' @rdname text
gtext <- function(label, gp_arg, form = T, ...){
  args <- list(...)
  args <- .fresh_param2(list(x = 0.5, y = 0.5), args)
  args$label <- if (form) form(label) else label
  args$gp <- .fresh_param2f(.title_gp, gp_arg)
  do.call(textGrob, args)
}
```

```
#' @export gtextp
#' @aliases gtextp
#' @description \code{gtextp}: ...
#' @rdname text
gtextp <- function(label, gp_arg, form = T, ...){
  if (missing(gp_arg))
    gp_arg <- list()
  gp_arg$font <- c(plain = 1)
  gtext(label, gp_arg, form, ...)
}

#' @export gtext0
#' @aliases gtext0
#' @description \code{gtext0}: ...
#' @rdname text
gtext0 <- function(label, gp_arg, form = T, ...) {
  gtext(label, gp_arg, form, x = .1, hjust = 0, ...)
}

#' @export form
#' @aliases form
#' @description \code{form}: ...
#' @rdname text
form <- function(label){
  Hmisc::capitalize(gsub("_", " ",  label))
}

#' @export gltext
#' @aliases gltext
#' @description \code{gltext}: ...
#' @rdname text
gltext <- function(label, gp_arg = list(), args = list(),
  l_gp = .gpar_dotted_line, flip = F, borderF = 1.2){
  if (flip) args$rot <- 90
  title <- do.call(gtext, c(list(label, gp_arg), args))
  if (flip) {
    height <- grobHeight(title) * borderF
    seg <- list(.5, 0, .5, u(.5, npc) - height / 2)
    seg. <- list(.5, 1, .5, u(.5, npc) + height / 2)
  } else {
    width <- grobWidth(title) * borderF
```

```
    seg <- list(0, .5, u(.5, npc) - width / 2, .5)
    seg. <- list(1, .5, u(.5, npc) + width / 2, .5)
  }
  line <- do.call(segmentsGrob, c(seg, list(gp = l_gp)))
  line. <- do.call(segmentsGrob, c(seg., list(gp = l_gp)))
  ggather(title, line, line.)
}

# grid.draw(gtext("test", list(cex = 5)))
```

```
# ============================================================================
# rect
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases rect
#'
#' @title a mutate of grid::rectGrob
#'
#' @description ...
#'
#' @name rect
NULL
#> NULL

#' @export .rect_gp
.rect_gp <- gpar(fill = "transparent")
#' @export .rect.r
.rect.r <- unit(0.3, "lines")
#' @export .vp.sep
.vp.sep <- unit(0.25, "lines")

#' @export .grecti.vp.p
.grecti.vp.p <- list(width = unit(1, "npc") - .vp.sep,
  height = unit(1, "npc") - .vp.sep,
  clip = "on")
#' @export .grecti.vp
.grecti.vp <- do.call(viewport, .grecti.vp.p)
#' @export .grecto.vp
.grecto.vp <- do.call(viewport, .fresh_param2(.grecti.vp.p, list(clip = "inherit")))

#' @export grect
#' @aliases grect
```

```r
#' @description \code{grect}: ...
#' @rdname rect
grect <- function(name, tfill = "#E18727FF", bfill = "white",
  t_args, tgp_args, t = roundrectGrob,
  b_args, bgp_args, b = roundrectGrob,
  order = c(1, 2), vp = NULL){
  t <- match.fun(t)
  t_args <- .fresh_param2(list(x = 0.5, y = 0.5,
      width = 0.5, height = 0.5,
      r = .rect.r),
    t_args)
  t_args$gp <- .fresh_param2f(gpar(fill = tfill), tgp_args)
  trect <- do.call(t, t_args)
  ## b
  b <- match.fun(b)
  b_args <- .fresh_param2(list(x = 0.5, y = 0.5,
      r = .rect.r),
    b_args)
  b_args$gp <- .fresh_param2f(gpar(fill = bfill), bgp_args)
  brect <- do.call(b, b_args)
  gTree(children = gList(trect, brect)[order], name = name, vp = vp)
}

#' @export grecti
#' @aliases grecti
#' @description \code{grecti}: ...
#' @rdname rect
grecti <- function(label, cex = 1, x = 0.5, y = 1.005,
  borderF = 2, just = c("center", "top"),
  tfill = "#E18727FF", vp = .grecti.vp,
  order = c(2, 1), cvp_clip = "on",
  cvp_fix = T, ...){
  if (is.list(borderF)) {
    borderF <- borderF[[1]]
    xmax <- T
  } else {
    xmax <- F
  }
  if (is(label, "grob")) {
    gtext <- label
    label <- label$label
```

```
  } else {
    gtext <- gtext(label, x = x, y = y, gp_arg = list(cex = cex * borderF), just = just)
  }
  t_args <- list(x = grobX(gtext, 90), y = grobY(gtext, 0),
    width = grobWidth(gtext), height = grobHeight(gtext))
  if (xmax) {
    t_args$width <- u(1.2, npc)
  }
  lst <- list(t_args = t_args, tfill = tfill, order = order)
  args <- list(...)
  args <- .fresh_param2(lst, args)
  args$name <- label
  grect <- do.call(grect, args)
  gtext <- gtext(label, list(cex = cex, col = "white"), vp = setvp(gtext))
  grob <- gTree(children = gList(grect, gtext), name = label, vp = vp)
  g <- grect$children[[ order[2] ]]
  cvp_name <- paste0(form(label), "_content")
  if (cvp_fix) {
    cvp <- viewport(x = 0.5, y = 0,
      width = grobWidth(g),
      height = grobHeight(g) - t_args$height,
      just = c("center", "bottom"),
      clip = cvp_clip,
      name = cvp_name)
  } else {
    cvp <- setvp(g, clip = cvp_clip, name = cvp_name)
  }
  graph(grob = grob, cvp = cvp)
}

#' @export grecti2
#' @aliases grecti2
#' @description \code{grecti2}: ...
#' @rdname rect
grecti2 <- function(label, cex = 1, tfill = "#E18727FF",
  borderF = list(2), ...){
  tgp_args <- list(col = tfill)
  bgp_args <- list(col = tfill)
  args <- list(...)
  args <- c(namel(label, cex, borderF, tfill, tgp_args, bgp_args), args)
  do.call(grecti, args)
```

```
}

#' @export grecti3
#' @aliases grecti3
#' @description \code{grecti3}: ...
#' @rdname rect
grecti3 <- function(label, cex = 1, tfill = "#E18727FF",
  borderF = list(2), ...){
  tgp_args <- list(col = "black")
  bgp_args <- list(col = "transparent")
  args <- list(...)
  args <- c(namel(label, cex, borderF, tfill, tgp_args, bgp_args), args)
  do.call(grecti, args)
}

#' @export grecto
#' @aliases grecto
#' @description \code{grecto}: ...
#' @rdname rect
grecto <- function(label, cex = 1, x = 0.5, y = 0.995,
  borderF = 2, just = c("center", "bottom"),
  tfill = "#E18727FF", vp = .grecto.vp,
  order = c(1, 2), cvp_clip = "on", cvp_fix = F, ...){
  args <- c(as.list(environment()), list(...))
  do.call(grecti, args)
}

#' @export grectn
#' @aliases grectn
#' @description \code{grectn}: ...
#' @rdname rect
grectn <- function(bfill = "white", b_args, bgp_args, b = roundrectGrob,
  cvp_clip = "inherit") {
  b <- match.fun(b)
  b_args <- .fresh_param2(list(x = 0.5, y = 0.5, r = .rect.r), b_args)
  b_args$gp <- .fresh_param2f(gpar(fill = bfill, lty = "dotted"), bgp_args)
  brect <- do.call(b, b_args)
  graph(grob = brect, cvp = setvp(brect, clip = cvp_clip))
}

#' @export grectn_frame
```

```r
#' @aliases grectn_frame
#' @description \code{grectn_frame}: ...
#' @rdname rect
grectn_frame <- function(content, title, zo = T){
  if (zo) content <- zo(content)
  content <- frame_row(c(title = .2, content = 1), namel(title, content))
  rect <- grectn(, , list(lty = "solid"))
  into(rect, content)
}

#' @export lst_grecti
#' @aliases lst_grecti
#' @description \code{lst_grecti}: ...
#' @rdname rect
lst_grecti <- function(names, pal, tar = "slot", fun = grecti, ...){
  sapply(names, simplify = F,
    function(name){
      graph <- match.fun(fun)(form(name), , tfill = pal[[ tar ]], ...)
    })
}

#' @export grectN
#' @aliases grectN
#' @description \code{grectN}: ...
#' @rdname rect
grectN <- function(lab.1, lab.2, gp = gpar(fontface = "plain"),
  bfill = "white"){
  frame <- frame_row(list(lab.1 = 1, seg = .1, lab.2 = 1),
    list(lab.1 = gtext(lab.1, gp),
      seg = segmentsGrob(0, .5, 1, .5),
      lab.2 = gtext(lab.2, gp)))
  into(grectn(bfill, , list(lty = "solid")), frame)
}

#' @export grecta
#' @aliases grecta
#' @description \code{grecta}: ...
#' @rdname rect
grecta <- function(label, cex = 4) {
  grob <- gtext(
    label, list(cex = cex), form = F,
```

```
    x = 0, y = u(1, npc),
    just = c("left", "top")
  )
  cvp <- viewport(
    grobWidth(grob), 0, u(1, npc) - grobWidth(grob), .95,
    just = c("left", "bottom")
  )
  graph(grob = grob, cvp = cvp)
}

#' @export gshiny
#' @aliases gshiny
#' @description \code{gshiny}: ...
#' @rdname rect
gshiny <- function(xn = 4, yn = 3,
  xps = seq(0, 1, , xn), yps = seq(0, 1, , yn),
  size = c(.15, .02),
  color = .default_color,
  rect = rectGrob(),
  vp = viewport(clip = "off"),
  cvp = viewport(, , .9, .9)
  ){
  size <- seq(size[1], size[2], length.out = floor(max(c(length(xps), length(yps))) / 2) + 1)
  xsize <- sym_fill(xps, size)
  xpal <- sym_fill(xps, color)
  args <- list(c(xps, xps), c(rep(1, length(xps)), rep(0, length(xps))),
    c(xsize, xsize), c(xpal, xpal))
  fun <- function(n) {
    circleGrob(args[[1]][n], args[[2]][n], args[[3]][n],
      gp = gpar(fill = args[[4]][n], col = "transparent"))
  }
  xcir <- lapply(1:(2 * length(xps)), fun)
  rep <- xps[xps %in% c(0, 1)]
  rep <- yps[yps %in% rep]
  yps <- yps[!yps %in% rep]
  if (length(rep) > 0) {
    size <- size[-1]
    color <- color[-1]
  }
  ysize <- sym_fill(yps, size)
  ypal <- sym_fill(yps, color)
```

```
  args <- list(c(rep(0, length(yps)), rep(1, length(yps))), c(yps, yps),
    c(ysize, ysize), c(ypal, ypal))
  ycir <- lapply(1:(2 * length(yps)), fun)
  args <- c(list(rect), xcir, ycir, list(vp = vp))
  graph(grob = do.call(ggather, args), cvp = cvp)
}

#' @export sym_fill
sym_fill <- function(long, short){
  if (length(long) %% 2 == 0) {
    short <- short[1:(length(long) / 2)]
    res <- c(short, rev(short))
  } else {
    half <- (length(long) - 1) / 2
    res <- c(short[1:(half + 1)], rev(short[1:half]))
  }
  if (any(is.na(res)))
    stop( "any(is.na(res)) == T" )
  else
    return(res)
}
```

```
# ============================================================================
# get external grob
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @export .expathsvg
.expathsvg <- function() {
  .expathsvg <- system.file("extdata", "svg", package = gsub("^.*:", "",
      environmentName(topenv())))
  assign('.expathsvg', .expathsvg, envir = topenv())
}

prefix <- c()

#' @export .check_external_svg
.check_external_svg <- function(){
  files <- list.files(.expathsvg, "\\.svg$", full.names = T)
  log.path <- paste0(.expathsvg, "/log")
  if (file.exists(log.path)) {
    log <- readLines(log.path)
    log <- log[vapply(log, file.exists, T)]
```

```
  } else {
    log <- c()
  }
  if (length(files) > 0) {
    new.log <-
      lapply(files,
        function(file) {
          if (!file %in% log) {
            rsvg::rsvg_svg(file, file)
            file
          }
        })
    new.log <- unlist(new.log)
    log <- c(log, new.log)
  }
  if (!is.null(log))
    writeLines(log, log.path)
}

#' @export ex_grob
ex_grob <- function(name, fun = .cairosvg_to_grob){
  file <- paste0(.expathsvg, "/", name, ".svg")
  if (file.exists(file)) {
    fun(file)
  } else {
    stop("file.exsits(file) == F")
  }
}

#' @export ex_pic
ex_pic <- function(name){
  ex_grob(name, fun = grImport2::readPicture)
}
```

```
# ============================================================================
# layers
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @export glayer
#' @aliases glayer
#' @description \code{glayer}: ...
#' @rdname rect
```

```
glayer <-
  function(n = 5, to = .2, gp = gpar(fill = "white"), fun = rectGrob){
    grob <- fun(x = seq(0, to, length.out = n),
      y = seq(0, to, length.out = n),
      height = 1 - to,
      width = 1 - to,
      just = c("left", "bottom"),
      gp = gp
    )
    cvp <- viewport(to, to, 1 - to, 1 - to, just = c("left", "bottom"), clip = "on")
    graph(grob = grob, cvp = cvp)
  }
```

```
# ============================================================================
# network
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases network
#'
#' @title Quickly draw random network diagrams
#'
#' @description ...
#'
#' @name network
NULL
#> NULL

#' @export fast_layout
#' @aliases fast_layout
#' @description \code{fast_layout}: ...
#' @rdname network
fast_layout <- function(edges, layout = "fr", nodes = NULL){
  graph <- igraph::graph_from_data_frame(edges, directed = T, vertices = nodes)
  graph <- tidygraph::as_tbl_graph(graph)
  ggraph::create_layout(graph, layout)
}

#' @export random_graph
#' @aliases random_graph
#' @description \code{random_graph}: ...
#' @rdname network
random_graph <- function(ids, n = length(ids), e = 4, layout = "fr") {
```

```
  df <- data.frame(id = ids, size = rnorm(n, .5, .2))
  edges <- data.frame(id1 = sample(ids, e), id2 = sample(ids, e),
    width = rnorm(e, .5, .2))
  fast_layout(edges, layout, df)
}
```

```
# ============================================================================
# others
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @export sep_legend
sep_legend <- function(p, theme) {
  p.l <- MCnebula2:::.get_legend(p + theme)
  theme$legend.position <- "none"
  p.m <- p + theme
  p <- p + theme
  return(namel(p.l, p.m, p))
}

#' @aliases zoom_pdf
#'
#' @title Zoom in locally pdf to png
#'
#' @description ...
#'
#' @name zoom_pdf
NULL
#> NULL

#' @export zoom_pdf
#' @aliases zoom_pdf
#' @description \code{zoom_pdf}: ...
#' @rdname zoom_pdf
zoom_pdf <- function(file, position = c(.5, .5), size = c(.15, .1), page = 1, dpi = 2000,
  as.grob = T)
{
  position[2] <- 1 - position[2]
  png <- pdftools::pdf_render_page(file, page = page, dpi = dpi)
  png <- png::readPNG(png::writePNG(png))
  wxh <- dim(png)[2:1]
  sel <- lapply(1:2,
    function(n) {
```

```
      center <- wxh[n] * position[n]
      long <- wxh[n] * size[n]
      start <- center - long / 2
      end <- center + long / 2
      round(start):round(end)
    })
  res <- png[sel[[2]], sel[[1]], ]
  if (as.grob) {
    res <- rasterGrob(res)
  }
  return(res)
}
```

```
# ============================================================================
# shape
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

simulate_peaks <- function(all_range = list(1:30, 31:60, 61:100, 101:140),
  shift = rnorm(10, 2, 1))
{
  lst <- mapply(shift, 1:length(shift), SIMPLIFY = F, FUN = function(shift, id){
    peak <- mapply(all_range, SIMPLIFY = F,
      FUN = function(range){
        peak <- dnorm(range, median(range) + shift, rnorm(1, 5, 1.2)) *
          rnorm(1, 0.7, 0.15)
      })
    feature <- mapply(1:length(all_range), lengths(all_range),
      FUN = function(seq, rep){
        rep(paste0("peak", seq), rep)
      })
    tibble::tibble(x = unlist(all_range), y = unlist(peak),
      sample = paste0("sample", id),
      peak = unlist(feature)
    )
  })
  data.table::rbindlist(lst)
}
```

```
# ============================================================================
# get_ggsets
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```r
#' @export nebulae_as_grob
#' @title Convert Nebulae as 'grob' object
#' @aliases nebulae_as_grob
#' @description \code{nebulae_as_grob}: This will convert Child-Nebulae
#' as a 'grob' object. See package `grid` about 'grob' object.
#' @param x [mcnebula-class] object.
#' @rdname nebulae_as_grob
nebulae_as_grob <- function(x) {
  chAsGrob <- function(ch, x) {
    ggset <- modify_default_child(ch)
    as_grob(call_command(ggset))
  }
  sets <- lapply(ggset(child_nebulae(x)), chAsGrob, x = x)
  sets <- lapply(names(sets),
    function(name) {
      ggather(sets[[name]],
        vp = viewports(child_nebulae(x))[[name]])
    })
  sets_vp <- viewport(layout = grid_layout(child_nebulae(x)))
  sets <- do.call(ggather, c(sets, list(vp = sets_vp)))
  legendH <- MCnebula2:::.legend_hierarchy(child_nebulae(x), x)
  legendG <- MCnebula2:::.get_legend(
    call_command(modify_default_child(ggset(child_nebulae(x))[[1]], x))
  )
  ## integrate
  vis <- frame_row(list(sets = 5, legendH = .5), namel(sets, legendH))
  vis <- frame_col(list(vis = 4, legendG = 1), namel(vis, legendG))
  vis
}
```

7 File: output_identification.R

```r
# ============================================================================
# output compounds identification table
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases format_table
#'
#' @title Format table via dplyr::*
#'
#' @description Format the data.frame via: \code{dplyr::filter}, \code{dplyr::arrange},
```

```
#' \code{dplyr::distinct}, \code{dplyr::mutate}, \code{dplyr::select},
#' \code{dplyr::rename}.
#' @param data data.frame. From \code{features_annotation(mcn)}.
#'
#' @name format_table
NULL
#> NULL

#' @export rename_table
#' @aliases rename_table
#' @description \code{rename_table}: ...
#' @rdname format_table
rename_table <-
  function(data, export_name = .export_name) {
    format_table(data, NULL, NULL, NULL, NULL, NULL, export_name)
  }

#' @export format_table
#' @aliases format_table
#' @description \code{format_table}: ...
#' @rdname format_table
format_table <-
  function(data, filter = .filter_format, arrange = .arrange_format,
    distinct = .distinct_format, mutate = .mutate_format,
    select = .select_format, export_name = .export_name) {
    if (!is.null(filter))
      data <- dplyr::filter(data, !!!filter)
    if (!is.null(arrange)) {
      if (is.null(data.frame(data)$arrange.rank))
        data <- dplyr::mutate(data, arrange.rank = NA)
      data <- dplyr::arrange(data, !!!arrange)
    }
    if (!is.null(distinct))
      data <- dplyr::distinct(data, !!!distinct, .keep_all = T)
    if (!is.null(mutate))
      data <- dplyr::mutate(data, !!!mutate)
    if (!is.null(select)) {
      select <- select[select %in% colnames(data)]
      if (!is.null(select))
        data <- dplyr::select(data, dplyr::all_of(select))
    }
```

```
    if (!is.null(export_name)) {
      export_name <- export_name[names(export_name) %in% colnames(data)]
      export_name <- as.list(turn_vector(export_name))
      data <- dplyr::rename(data, !!!export_name)
    }
    tibble::as_tibble(data)
  }

#' @export .filter_format
#' @aliases .filter_format
#' @description \code{.filter_format}: ...
#' @rdname format_table
.filter_format <-
  list(quote(tani.score >= .5))

#' @export .arrange_format
#' @aliases .arrange_format
#' @description \code{.arrange_format}: ...
#' @rdname format_table
.arrange_format <-
  list(
    quote(arrange.rank),
    quote(inchikey2d),
    quote(desc(tani.score))
  )

#' @export .distinct_format
#' @aliases .distinct_format
#' @description \code{.distinct_format}: ...
#' @rdname format_table
.distinct_format <-
  list(quote(inchikey2d))

#' @export .mutate_format
#' @aliases .mutate_format
#' @description \code{.mutate_format}: ...
#' @rdname format_table
.mutate_format <-
  list(mz = quote(round(mz, 4)),
    error.mass = quote(floor(error.mass * 10) / 10),
    tani.score = quote(floor(tani.score * 100) / 100),
```

```
    rt.min = quote(round(rt.secound / 60, 1))
  )

#' @export .select_format
#' @aliases .select_format
#' @description \code{.select_format}: ...
#' @rdname format_table
.select_format <- c("No.", "synonym", ".features_id", "mz", "error.mass",
  "rt.min", "mol.formula", "adduct", "tani.score", "inchikey2d",
  "class", "logFC", "P.Value", "adj.P.Val"
)

#' @export .export_name
#' @aliases .export_name
#' @description \code{.export_name}: ...
#' @rdname format_table
.export_name <- c(mz = "Precursor m/z",
  rt.min = "RT (min)",
  similarity = "Spectral similarity",
  tani.score = "Tanimoto similarity",
  rel.index = "Relative index",
  rel.int. = "Relative intensity",
  group = "Group",
  .features_id = "ID",
  mol.formula = "Formula",
  inchikey2d = "InChIKey planar",
  error.mass = "Mass error (ppm)",
  synonym = "Synonym",
  adduct = "Adduct",
  class = "Class",
  logFC = "log2(FC)",
  P.Value = "P-value",
  adj.P.Val = "Q-value"
)
```

8 File: pathway_enrichment.R

```
# ============================================================================
# Combining multiple tools for pathway enrichment analysis.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```r
#' @aliases pathway_enrichment
#'
#' @title Perform pathway enrichment via package of 'FELLA'
#'
#' @description Pathway enrichment analysis was performed using KEGG ID
#' via package of 'FELLA'.
#' (Convert CID to KEGG ID using the 'MetaboAnalystR' package.
#' See <https://github.com/xia-lab/MetaboAnalystR> for installation.)
#'
#' @name pathway_enrichment
NULL
#> NULL

#' @export init_fella
#' @aliases init_fella
#' @description \code{init_fella}: ...
#' @seealso [FELLA::buildDataFromGraph()], [FELLA::buildGraphFromKEGGREST()]
#' @rdname pathway_enrichment
init_fella <-
  function(dir, org = c("hsa", "mmu", "rno"), seed = 1, rebuild = F) {
    if (!file.exists(dir))
      stop("file.exists(dir) == F")
    dir <- paste0(dir, "/fella_pathway")
    dir.create(dir, F)
    org <- match.arg(org)
    db.dir <- paste0(dir, "/", org, ".db.dir")
    if (file.exists(db.dir) & !rebuild) {
      return(db.dir)
    } else {
      graph.file <- paste0(dir, "/", org, ".graph.Rdata")
      unlink(db.dir, T)
      set.seed(seed)
      graph <- FELLA::buildGraphFromKEGGREST(organism = org)
      save(graph, file = graph.file)
      FELLA::buildDataFromGraph(
        keggdata.graph = graph,
        databaseDir = db.dir, internalDir = FALSE,
        matrices = c("hypergeom", "diffusion", "pagerank"),
        normality = c("diffusion", "pagerank"),
        dampingFactor = 0.85, niter = 100)
    }
```

```
    return(db.dir)
  }

#' @export load_fella
#' @aliases load_fella
#' @description \code{load_fella}: ...
#' @rdname pathway_enrichment
load_fella <- function(dir) {
  if(!file.exists(dir)){
    stop("file.exists(dir) == F")
  }
  FELLA::loadKEGGdata(
    databaseDir = dir, internalDir = FALSE,
    loadMatrix = c("hypergeom", "diffusion", "pagerank")
  )
}

#' @export enrich_fella
#' @aliases enrich_fella
#' @description \code{enrich_fella}: ...
#' @rdname pathway_enrichment
enrich_fella <- function(id.lst, data) {
  if (!is.list(id.lst)) {
    id.lst <- list(id.lst)
  }
  lapply(1:length(id.lst),
    function(n) {
      message("\n=========", "Enrichment:", n, "=========")
      id <- id.lst[[n]]
      res <- try(
        FELLA::enrich(
          id, data = data,
          method = FELLA::listMethods(),
          approx = "normality"
        )
      )
      if (inherits(res, "try-error"))
        return(NULL)
      else
        res
    })
```

```
}

#' @export graph_fella
#' @aliases graph_fella
#' @description \code{graph_fella}: ...
#' @rdname pathway_enrichment
graph_fella <- function( obj.lst, data, method = c("pagerank", "diffusion", "hypergeom"),
  threshold = .1)
{
  method <- match.arg(method)
  graph.lst <-
    lapply(obj.lst,
      function(obj) {
        if (is.null(obj))
          return()
        inmap <- FELLA::getInput(obj)
        graph <- FELLA::generateResultsGraph(
          object = obj,
          method = method,
          threshold = threshold,
          data = data
        )
        graph <- tidygraph::as_tbl_graph(graph)
        graph <- dplyr::select(graph, -entrez)
        graph <- dplyr::mutate(
          graph, NAME = vapply(NAME, function(c) c[1], ""),
          abbrev.name = stringr::str_trunc(NAME, 15),
          input = ifelse(input, "Input", "Others"),
          type = vapply(
            name, FUN.VALUE = "", USE.NAMES = F,
            function(str){
              str <- stringr::str_extract(str, "^[^0-9]]{1,3}|\\.")
              str <- ifelse(nchar(str) > 1, "pathway", str)
              switch(
                str, pathway = "Pathway",
                M = "Module",
                "." = "Enzyme",
                C = "Compound",
                R = "Reaction")
            }))
      })
```

```
}

#' @import ggraph
#' @export plotGraph_fella
#' @aliases plotGraph_fella
#' @description \code{plotGraph_fella}: Draw the graph via
#' package of 'ggplot2'.
#' @rdname pathway_enrichment
plotGraph_fella <- function(
  graph, layout = "graphopt", seed = 1,
  shape = c(Input = 15, Others = 16),
  color = c(
    Pathway = "#E64B35FF",
    Module = "#E377C2",
    Enzyme = "#EFC000",
    Reaction = "#4DBBD5FF",
    Compound = "#00A087FF"),
  size = c(
    Pathway = 7,
    Module = 5,
    Enzyme = 6,
    Reaction = 5,
    Compound = 10))
{
  set.seed(seed)
  layout <- ggraph::create_layout(graph, layout = layout)
  ggraph(layout) +
    geom_edge_fan(
      aes(edge_width = weight),
      color = "black",
      show.legend = F,
      end_cap = ggraph::circle(3, 'mm'),
      arrow = arrow(length = unit(2, 'mm'))) +
    geom_node_point(
      aes(color = type,
        shape = input,
        size = type),
      stroke = 0.1) +
    ggraph::geom_node_text(
      aes(label = stringr::str_wrap(
          abbrev.name, 15)),
```

```
      size = 3,
      family = .font,
      color = "black") +
    scale_shape_manual(values = shape) +
    scale_color_manual(values = color) +
    scale_size_manual(values = size) +
    scale_edge_width(range = c(0.1, 0.3)) +
    guides(
      size = "none",
      shape = guide_legend(override.aes = list(size = 4)),
      color = guide_legend(override.aes = list(size = 4))) +
    labs(color = "Category", shape = "Type") +
    theme_void() +
    theme(text = element_text(family = .font))
}

#' @export cid.to.kegg
#' @aliases cid.to.kegg
#' @description \code{cid.to.kegg}: ...
#' @rdname pathway_enrichment
cid.to.kegg <-
  function(cids){
    if (!requireNamespace("MetaboAnalystR", quietly = T)) {
      stop("package 'MetaboAnalystR' not available.",
        "See <https://github.com/xia-lab/MetaboAnalystR> for installation.")
    }
    obj <- MetaboAnalystR::InitDataObjects("conc", "msetora", F)
    obj <- MetaboAnalystR::Setup.MapData(obj, cids)
    obj <- MetaboAnalystR::CrossReferencing(obj, "pubchem")
    obj <- MetaboAnalystR::CreateMappingResultTable(obj)
    obj <- dplyr::as_tibble(obj$dataSet$map.table)
    dplyr::filter(obj, KEGG != "NA")
  }
```

9 File: pick_annotation.R

```
# ========================================================================
# Following a preset algorithm to get a unique value from the candidate items.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases pick_annotation
```

```
#'
#' @title Pick unique annotation for compounds
#'
#' @description Pick unique chemical class or synonyms for 'features'.
#' @family queries
#'
#' @name pick_annotation
NULL
#> NULL

#' @export pick_class
#' @aliases pick_class
#' @description \code{pick_class}: ...
#' @rdname pick_annotation
pick_class <-
  function(inchikey2d, class.rdata, filter = .filter_pick.class,
    fun = PickClass){
    class <- extract_rdata_list(class.rdata, inchikey2d)
    if (!is.null(filter)) {
      class <- data.frame(data.table::rbindlist(class, idcol = T))
      class <- dplyr::filter(class, !!!filter)
      class <- split(class, ~ .id)
    }
    class <- sapply(inchikey2d, simplify = F,
      function(key2d) {
        class[[ key2d ]]$Classification
      })
    if (!is.null(fun)) {
      class <- lapply(class, fun)
    }
    unlist(class)
  }

#' @export .filter_pick.class
#' @aliases .filter_pick.class
#' @description \code{.filter_pick.class}: ...
#' @rdname pick_annotation
.filter_pick.class <-
  list(quote(!Level %in% dplyr::all_of(c("kingdom", "level 7", "level 8", "level 9"))),
    quote(!grepl("[0-9]|Organ", Classification))
  )
```

```
#' @export PickClass
#' @aliases PickClass
#' @description \code{PickClass}: ...
#' @rdname pick_annotation
PickClass <-
  function(class){
    if (is.null(class)) NA
    else tail(class, n = 1)
  }

#' @export pick_synonym
#' @aliases pick_synonym
#' @description \code{pick_synonym}: ...
#' @rdname pick_annotation
pick_synonym <-
  function(inchikey2d = NULL, inchikey.rdata = NULL,
    synonym.rdata, iupac.rdata = NULL,
    filter = .filter_pick.general, fun = PickGeneral) {
    syno <- extract_rdata_list(synonym.rdata)
    syno <- data.frame(data.table::rbindlist(syno))
    if (!is.null(filter)) {
      syno <- dplyr::filter(syno, !!!filter)
    }
    if (!is.null(inchikey2d)) {
      inchikey <- extract_rdata_list(inchikey.rdata, inchikey2d)
      meta <- sapply(inchikey, simplify = F, function(x) as.character(x$CID))
      syno$cid <- as.character(syno$cid)
      syno <- group_switch(syno, meta, by = "cid")
      syno <- sapply(inchikey2d, simplify = F,
        function(key2d) {
          if (is.null(syno[[ key2d ]]))
            return()
          else
            syno[[ key2d ]]$syno
        })
    } else {
      syno <- lapply(split(syno, ~ cid), function(set) set$syno)
    }
    if (!is.null(iupac.rdata)) {
      iupac <- extract_rdata_list(iupac.rdata, inchikey2d)
      if (is.null(inchikey2d)) {
```

```
        iupac <- data.table::rbindlist(iupac)
        iupac <- lapply(split(iupac, ~ CID), function(set) set$IUPACName)
      } else {
        iupac <- lapply(iupac, function(set) set$IUPACName)
      }
      syno <- sapply(names(syno), simplify = F,
        function(name) {
          c(syno[[ name ]], iupac[[ name ]])
        })
    }
    if (!is.null(fun)) {
      syno <- lapply(syno, fun)
    }
    unlist(syno)
  }

#' @export .filter_pick.general
#' @aliases .filter_pick.general
#' @description \code{.filter_pick.general}: ...
#' @rdname pick_annotation
.filter_pick.general <-
  list(quote(!is.na(syno)),
    quote(!grepl('[0-9]{3}', syno)),
    quote(!grepl('^[A-Z-]{1,5}$', syno)),
    quote(!grepl('^[A-Z0-9]{1,}$', syno)),
    quote(!grepl('(?<=-)[A-Z0-9]{5,}$', syno, perl = T)),
    quote(!grepl('^[0-9-]*$', syno))
  )

#' @export PickGeneral
#' @aliases PickGeneral
#' @description \code{PickGeneral}: ...
#' @rdname pick_annotation
PickGeneral <- function(syno,
  ps = c("^[a-zA-Z]*$", "^[a-zA-Z-]*$",
    "^[a-zA-Z0-9-]*$", "^[^:]*$")
  ){
  if (is.null(syno)) return(NA)
  unlist(lapply(ps, function(p) syno[grepl(p, syno)]))[1]
}
```

10 File: plot_EIC_stack.R

```r
# ============================================================================
# plot extracted ions chromatograph (EIC) for features using `MSnbase`
# to extract mass data.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases plot_EIC_stack
#'
#' @title Draw extracted ions chromatography for 'features'
#'
#' @description Use quantification table (with peak start time and end time)
#' exported by MCmine to draw EIC plot.
#'
#' @name plot_EIC_stack
NULL
#> NULL

#' @export plot_EIC_stack
#' @aliases plot_EIC_stack
#' @description \code{plot_EIC_stack}: ...
#' @rdname plot_EIC_stack
plot_EIC_stack <-
  function(
    idset,
    metadata,
    quant.path,
    mzml.path,
    palette = ggsci::pal_npg()(10),
    mz.tol = 0.01,
    rt.tol = 0.1,
    cl = NULL,
    data = NULL)
  {
    if (is.null(data)) {
      .suggest_bio_package("MSnbase")
      ## metadata
      .check_columns(metadata, c("file", "sample", "group"), "metadata")
      metadata <- dplyr::arrange(metadata, sample)
      feature <- data.table::fread(quant.path)
      .check_columns(
        feature, c("row ID",    "row m/z", "row retention time"),
```

```
    "data.table::fread(quant.path)"
  )
  feature <- dplyr::select(
    feature, .features_id = 1, mz = 2, rt = 3,
    dplyr::contains(metadata$sample) & dplyr::contains("Peak RT")
  )
  if (ncol(feature) == 3) {
    stop("`feature` get by data.table::fread(quant.path) not contains 'Peak RT' information.")
  }
  feature <- dplyr::mutate(feature, .features_id = as.character(.features_id))
  feature <- dplyr::filter(feature, .features_id %in% idset)
  feature <- tidyr::gather(feature, type, time, -.features_id, -mz, -rt)
  feature <- feature.rt.during <- dplyr::mutate(
    feature, time = ifelse(time == 0, NA, time),
    sub.type = stringr::str_extract(type, "(?<=RT ).*?$"),
    sample = gsub("\\.mz.{-}ML Peak RT.*$", "", type)
  )
  feature <- dplyr::group_by(feature, .features_id, mz, rt, sub.type)
  feature <- dplyr::summarize(
    feature, sub.type.min = min(time, na.rm = T),
    sub.type.max = max(time, na.rm = T),
    .groups = "drop_last"
  )
  feature <- dplyr::mutate(
    feature, time = ifelse(sub.type == "start", sub.type.min, sub.type.max)
  )
  feature <- dplyr::select(feature, -contains("sub.type."))
  feature <- tidyr::spread(feature, sub.type, time)
  ## read data
  if (!is.null(cl))
    bioc.par(cl)
  data <- MSnbase::readMSData(
    paste0(mzml.path, "/", metadata$file),
    pdata = new("NAnnotatedDataFrame", metadata),
    mode = "onDisk"
  )
  ## extract EIC
  rt.tol.sec <- rt.tol * 60
  if (!is.null(cl))
    bioc.par(cl)
  eic.list <- pbapply::pbapply(
```

```
    feature, 1,
    function(vec){
      ## mz range for EIC
      mz <- as.numeric(vec[["mz"]])
      mz.range <- c(mz - mz.tol, mz + mz.tol)
      ## rt range for EIC
      rt.range <- c(vec[["start"]], vec[["end"]])
      rt.range <- as.numeric(rt.range) * 60
      rt.range <- c(rt.range[1] - rt.tol.sec, rt.range[2] + rt.tol.sec)
      ms1.vec <- MSnbase::chromatogram(data, msLevel = 1L, mz = mz.range,
        rt = rt.range, aggregationFun = "max")
      data.list <- lapply(unlist(ms1.vec),
        function(chr){
          int <- MSnbase::intensity(chr)
          rt <- MSnbase::rtime(chr)
          data.frame(real.time = rt, int = int)
        })
      names(data.list) <- metadata$sample
      df <- data.table::rbindlist(data.list, idcol = T)
      df <- dplyr::rename(df, sample = .id)
      dplyr::mutate(df, .features_id = vec[[".features_id"]])
    })
  ## define whether the peak belong to the feature
  eic.df <- data.table::rbindlist(eic.list)
  eic.df <- merge(
    eic.df, feature.rt.during, by = c(".features_id", "sample"), allow.cartesian = T
  )
  eic.df <- dplyr::select(eic.df, -type)
  eic.df <- tidyr::spread(eic.df, key = sub.type, value = time)
  eic.df <- merge(eic.df, metadata, by = "sample", all.x = T)
  eic.df <- dplyr::mutate(
    eic.df, real.time.min = real.time / 60,
    feature = ifelse(real.time.min >= start & real.time.min <= end,
      sample, "Non feature"),
    fill = ifelse(feature == "Non feature", feature, group),
    mz = round(mz, 4),
    anno.mz = paste("Precursor m/z:\n  ", mz - mz.tol, "~", mz + mz.tol),
    anno.rt = paste("RT (min):", round(rt, 1)),
    anno = paste0(anno.mz, "\n", anno.rt)
  )
  ## annotation (mz and rt)
```

```
      anno <- dplyr::select(eic.df, .features_id, int, real.time.min, contains("anno"))
      anno <- dplyr::group_by(anno, .features_id)
      anno <- dplyr::summarize(
        anno, anno.x = min(real.time.min, na.rm = T),
        anno.y = max(int, na.rm = T) * 3 / 4,
        anno = unique(anno)
      )
      data <- list(eic.df = eic.df, anno = anno)
    }
    if (!any(names(palette) == "Non feature")) {
      palette[[ "Non feature" ]] <- "grey95"
    }
    data$p <- ggplot(data[[ "eic.df" ]]) +
      geom_line(
        aes(x = real.time.min,
          y = int,
          group = sample,
          color = fill),
        lineend = "round") +
      labs(color = "Peak attribution", x = "RT (min)", y = "Intensity") +
      geom_text(data = data[[ "anno" ]],
        aes(x = anno.x, y = anno.y, label = anno),
        hjust = 0, fontface = "bold", family = .font) +
      scale_y_continuous(labels = scales::scientific) +
      facet_wrap( ~ paste("ID:", .features_id), scales = "free") +
      theme_minimal() +
      scale_color_manual(values = palette) +
      theme(text = element_text(family = .font),
        plot.background = element_rect(fill = "white", size = 0, color = "transparent"),
        strip.text = element_text(size = 12)) +
      geom_blank()
    return(data)
  }

#' @export bioc.par
#' @aliases bioc.par
#' @description \code{bioc.par}: ...
#' @rdname plot_EIC_stack
bioc.par <-
  function(cl = 4){
    BiocParallel::register(
```

```
      BiocParallel::bpstart(
        BiocParallel::MulticoreParam(cl)
      )
    )
  }
```

11 File: query_classification.R

```
# ============================================================================
# query classification for compounds using classyfire API
# run after query_inchikey
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases query_classification
#'
#' @title Query classification of compounds via packgage of 'classifyerR'
#'
#' @description The used function is:
#' \code{classyfireR::get_classification(inchikey)}
#' @family queries
#'
#' @name query_classification
NULL
#> NULL

#' @export query_classification
#' @aliases query_classification
#' @description \code{query_classification}: ...
#' @rdname query_classification
query_classification <-
  function(
    inchikey2d,
    dir,
    inchikey.rdata = paste0(dir, "/inchikey.rdata"),
    rdata.name = "classification.rdata",
    classyfire_cl = NULL,
    gather_as_rdata = T,
    ...
    ){
    rdata <- paste0(dir, "/", rdata.name)
    classes <- extract_rdata_list(rdata)
```

```
    if (!is.null(classes))
      inchikey2d <- inchikey2d[!inchikey2d %in% names(classes)]
    if(length(inchikey2d) == 0)
      return(paste0(dir, "/", rdata.name))
    inchikey_set <- extract_rdata_list(inchikey.rdata, inchikey2d)
    if (is.null(inchikey_set))
      stop("is.null(inchikey_set) == T. File `inchikey.rdata` may not exists.")
    sets <- lapply(inchikey_set, function(df){
      if("InChIKey" %in% colnames(df))
        return(df)
    })
    sets <- data.table::rbindlist(sets)
    sets <- dplyr::mutate(sets, inchikey2d = stringr::str_extract(InChIKey, "^[A-Z]{1,}"))
    l <- classyfire_get_classification(sets, dir, classyfire_cl = classyfire_cl, ...)
    if (is.logical(l))
      return(paste0(dir, "/", rdata.name))
    if (gather_as_rdata) {
      cat("## gather data\n")
      packing_as_rdata_list(dir, pattern = "^[A-Z]{14}$",
        rdata = rdata.name, extra = classes)
    }
    return(paste0(dir, "/", rdata.name))
  }

#' @export classyfire_get_classification
#' @aliases classyfire_get_classification
#' @description \code{classyfire_get_classification}: ...
#' @rdname query_classification
classyfire_get_classification <-
  function(
    sets,
    dir,
    classyfire_cl = NULL,
    log_file = paste0(dir, "/classyfire.log"),
    ...
    ){
    if (file.exists(log_file)){
      log_df <- data.table::fread(log_file)
      sets <- dplyr::filter(sets, !InChIKey %in% log_df$log)
      if(nrow(sets) == 0)
        return(F)
```

```
    }
    sets <- split(data.frame(sets), ~ inchikey2d)
    log <- pbapply::pblapply(names(sets), cl = classyfire_cl,
      function(inchikey2d) {
        set <- sets[[ inchikey2d ]]
        unlist(lapply(set[["InChIKey"]], .get_classification,
            file = paste0(dir, "/", inchikey2d)),
          use.names = F)
      })
    log <- unlist(log, use.names = F)
    log <- data.frame(log = log)
    if (exists("log_df"))
      log <- dplyr::bind_rows(log_df, log)
    write_tsv(log, file = log_file)
  }

.get_classification <-
  function(inchikey, file){
    if(!file.exists(file)){
      ch <- classyfireR::get_classification(inchikey)
    }else{
      return()
    }
    if(is.null(ch)){
      return(inchikey)
    }else{
      ch <- classyfireR::classification(ch)
      write_tsv(ch, file)
    }
  }
```

12 File: query_inchikey.R

```
# ==========================================================================
# query inchikey for compounds using pubchem API
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases query_inchikey
#'
#' @title Query InChIkey of compounds via 'InChIkey 2D'
#'
```

```
#' @description
#' The API:
#' url_start = paste0("https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/", type, "/")
#' url_end = paste0("/property/", paste(get, collapse = ","), "/CSV")
#' url = paste0(url_start, "/", inchikey2d, "/", url_end)
#'
#' @family queries
#'
#' @name query_inchikey
NULL
#> NULL

#' @export query_inchikey
#' @aliases query_inchikey
#' @description \code{query_inchikey}: ...
#' @rdname query_inchikey
query_inchikey <-
  function(
    inchikey2d,
    dir,
    rdata.name = "inchikey.rdata",
    curl_cl = NULL,
    gather_as_rdata = T,
    ...
    ){
    rdata <- paste0(dir, "/", rdata.name)
    inchikey_set <- extract_rdata_list(rdata)
    if (!is.null(inchikey_set))
      inchikey2d <- inchikey2d[!inchikey2d %in% names(inchikey_set)]
    if(length(inchikey2d) == 0)
      return(paste0(dir, "/", rdata.name))
    pbapply::pblapply(inchikey2d, pubchem_get_inchikey,
      dir = dir, cl = curl_cl, ...)
    if (gather_as_rdata) {
      cat("## gather data\n")
      packing_as_rdata_list(dir, pattern = "^[A-Z]{14}$",
        rdata = rdata.name, extra = inchikey_set)
    }
    return(paste0(dir, "/", rdata.name))
  }
```

```r
#' @export pubchem_get_inchikey
#' @aliases pubchem_get_inchikey
#' @description \code{pubchem_get_inchikey}: ...
#' @rdname query_inchikey
pubchem_get_inchikey <-
  function(
    inchikey2d,
    dir,
    type = "inchikey",
    get = "InChIkey",
    ...
    ){
    file <- paste0(dir, "/", inchikey2d)
    if(file.exists(file)){
      csv <- read_tsv(file)
      if("CID" %in% colnames(csv))
        return()
    }
    url_start = paste0("https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/", type, "/")
    url_end = paste0("/property/", paste(get, collapse = ","), "/CSV")
    url = paste0(url_start, "/", inchikey2d, "/", url_end)
    check <- 0
    while(check == 0 | inherits(check, "try-error")){
      check <- try(csv <- RCurl::getURL(url), silent = T)
    }
    if(grepl("Status: 404", csv)){
      write_tsv(csv, file = file)
      return()
    }
    while(grepl("Status:    503", csv)){
      csv <- RCurl::getURL(url)
    }
    csv <- data.table::fread(text = csv)
    write_tsv(csv, file = file)
  }
```

13 File: query_others.R

```r
# ============================================================================
# query other property for compounds using pubchem API
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```r
#' @aliases query_iupac
#'
#' @title Query IUPAC name of compounds via 'InChIkey 2D'
#'
#' @description Similar to [query_inchikey()], but get 'IUPACName'.
#' @family queries
#'
#' @name query_iupac
NULL
#> NULL

#' @export query_iupac
#' @aliases query_iupac
#' @description \code{query_iupac}: ...
#' @rdname query_iupac
query_iupac <-
  function(inchikey2d,
    dir,
    rdata.name = "iupac.rdata",
    curl_cl = NULL,
    gather_as_rdata = T,
    ...
    ) {
    query_inchikey(inchikey2d, dir, rdata.name, curl_cl, gather_as_rdata,
      get = "IUPACName")
  }
```

14 File: query_synonyms.R

```r
# ============================================================================
# query synonyms for compounds using pubchem API
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @aliases query_synonyms
#'
#' @title Query synonyms of compounds via CID
#'
#' @description Bulk search for compound synonyms via pubchem API.
#' (https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/.../synonyms/XML)
#' @family queries
```

```
#'
#' @name query_synonyms
NULL
#> NULL

#' @export query_synonyms
#' @aliases query_synonyms
#' @description \code{query_synonyms}: ...
#' @rdname query_synonyms
query_synonyms <-
  function(
    cid,
    dir,
    rdata.name = "synonyms.rdata",
    curl_cl = NULL,
    gather_as_rdata = T,
    group_number = 50,
    ...
    ){
    rdata <- paste0(dir, "/", rdata.name)
    cid_set <- extract_rdata_list(rdata)
    if (!is.null(cid_set)) {
      cid_set <- data.table::rbindlist(cid_set)
      if (nrow(cid_set) > 0)
        extra <- list(cid_set)
      else
        extra <- NULL
      if("cid" %in% colnames(cid_set)){
        cid_set <- dplyr::distinct(cid_set, cid, syno)
      }
      cid <- cid[!cid %in% cid_set$cid]
      if(length(cid) == 0)
        return(paste0(dir, "/", rdata.name))
    } else {
      extra <- NULL
    }
    group <- grouping_vec2list(cid, group_number = group_number)
    pbapply::pblapply(group, pubchem_get_synonyms,
      dir = dir, ..., cl = curl_cl)
    if (gather_as_rdata) {
      cat("## gather data\n")
```

```
      packing_as_rdata_list(dir, pattern = "^G[0-9]{1,}$",
        dedup = F,
        rdata = rdata.name,
        extra = extra)
    }
    return(paste0(dir, "/", rdata.name))
  }

#' @export pubchem_get_synonyms
#' @aliases pubchem_get_synonyms
#' @description \code{pubchem_get_synonyms}: ...
#' @rdname query_synonyms
pubchem_get_synonyms <-
  function(
    cid,
    dir,
    ...
    ){
    savename <- attr(cid, "name")
    file <- paste0(dir, "/", savename)
    cid <- paste(cid, collapse = ",")
    url_start <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/"
    url_end <- "/synonyms/XML"
    url <- paste0(url_start, cid, url_end)
    check <- 0
    while(check == 0 | inherits(check, "try-error")){
      check <- try(text <- RCurl::getURL(url), silent = T)
    }
    while(grepl("Status:    503", text)){
      text <- RCurl::getURL(url)
    }
    text <- XML::xmlToList(text)
    text <- text[names(text) == "Information"]
    text <-
      lapply(text,
        function(list){
          syno <- list[names(list) == "Synonym"]
          syno <-
            lapply(syno,
              function(char){
                if(is.null(char)){
```

```
                return(NA)
              }else{
                return(char)
              }
            })
        data.table::data.table(cid = list$CID, syno = unlist(syno))
      })
  text <- data.table::rbindlist(text, fill = T)
  write_tsv(text, filename = file)
}

#' @export grouping_vec2list
#' @aliases grouping_vec2list
#' @description \code{grouping_vec2list}: ...
#' @rdname query_synonyms
grouping_vec2list <-
  function(
    vector,
    group_number,
    byrow = F
    ){
    if(length(vector) < group_number){
      attr(vector, "name") <- "G1"
      return(list(vector))
    }
    rest <- length(vector) %% group_number
    group <- matrix(vector[1:(length(vector) - rest)],
      ncol = group_number,
      byrow = byrow)
    group <- apply(group, 1, c, simplify = F)
    group <- c(group, list(tail(vector, n = rest)))
    group <- lapply(1:length(group),
      function(n) {
        vec <- group[[n]]
        attr(vec, "name") <- paste0("G", n)
        vec
      })
    if(rest == 0)
      group <- group[1:(length(group) - 1)]
    return(group)
  }
```

```r
#' @export extract_rdata_list
#' @aliases extract_rdata_list
#' @description \code{extract_rdata_list}: extract results from .rdata
#' @rdname query_synonyms
extract_rdata_list <-
  function(
    rdata,
    names = NA
    ){
    if(!file.exists(rdata))
      return()
    load(rdata)
    if(!is.na(names[1])){
      list <- list[names(list) %in% names]
    }
    return(list)
  }

#' @export packing_as_rdata_list
#' @aliases packing_as_rdata_list
#' @description \code{packing_as_rdata_list}: gather table as .rdata
#' @rdname query_synonyms
packing_as_rdata_list <-
  function(
    path,
    pattern,
    rdata,
    extra = NULL,
    rm_files = T,
    dedup = T
    ){
    file_set <- list.files(path, pattern = pattern)
    if(length(file_set) == 0)
      return()
    list <- pbapply::pblapply(paste0(path, "/", file_set), read_tsv)
    names(list) <- file_set
    list <- c(extra, list)
    if(dedup){
      df <- data.table::data.table(name = names(list), n = 1:length(list))
      df <- dplyr::distinct(df, name, .keep_all = T)
      list <- list[df$n]
```

```
  }
  if(rm_files){
    lapply(paste0(path, "/", file_set), file.remove)
  }
  save(list, file = paste0(path, "/", rdata))
}
```