R codes of utils.tool

Contents

1 File: aaa.R

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

#' @aliases utilites
#'
#' @title utilites for programming
#'
#' @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), "]]"),
```

```r
                          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}: ...
```

```r
#' @rdname utilites
tmp_pdf <- function() {
  paste0(tempdir(), "/tmp_pdf.pdf")
}

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

#' @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){
  if (n == 0) {
    return(NULL)
  }
  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] == "")
```

```r
                        as.character(call)[n]
                      else
                        names(call)[n]
                    })
  }
  names(lst) <- names
  lst
}

#' @export repSuffix
#' @aliases repSuffix
#' @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())
```

```r
}

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

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

```r
}

#' @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", ...)
}
```

```r
clearMatch <- function(strs)
{
  strs <- unlist(strs)
  strs <- strs[ !vapply(strs, is.na, logical(1)) ]
  strs
}
```

2 File: alignment_merge.R

```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.
#' 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)
  }

`:=` <- 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: build_docu_from_script.R

```r
build_docu_from_script <-
  function(
           path = "~/outline",
           pattern = "\\.R$",
           output = ifelse(file.exists("mcnebula_results"), "mcnebula_results", "."),
           ref.docu = "mcnebula.workflow.Rmd",
           ref.path = system.file("extdata", ref.docu, package = "utils.tool")
           ){
    ## ----------------------------------------------------------------------
    ref.cluster <- msource(script = ref.path, block = "^@.*", source = F)
    ref <- readLines(ref.path)
    ## metadata of inset
    df <- data.table::data.table(tag = stringr::str_extract(ref, "^@.*")) %>%
      dplyr::mutate(nrow = 1:length(tag)) %>%
      dplyr::filter(!is.na(tag)) %>%
      dplyr::mutate(tag.name = stringr::str_extract(tag, "(?<=@).{1,}"))
    ## ------------------------------------
    ## build empty list
    merge <- list()
    length(merge) <- length(ref.cluster) * 2 - 1
    for(i in 1:length(merge)){
      if(i %% 2 == 1){
        merge[[i]] <- ref.cluster[[(i + 1) / 2]]
      }else{
        names(merge)[i] <- df$tag.name[i / 2]
      }
    }
    merge <- lapply(merge, function(vec){
                      if(class(vec) == "numeric"){
                        ref[vec]
                      }
           })
    ## ----------------------------------------------------------------------
    script <- find_latest_script(path, pattern)$file[1]
    cat("[INFO]: Latest script is:", script, "\n")
    ## catch tag in script
    code <- readLines(script)
    tag.script <- data.table::data.table(
      start = grep("## \\^[a-z]{1,}_{1,}", code),
      end = grep("## \\$[a-z]{1,}_{1,}", code)
    ) %>%
      dplyr::mutate(tag = stringr::str_extract(code[start], "(?<=## \\^)[a-z]{1,}(?=_)"),
                    start = start + 1,
                    end = end -1)
    ## ------------------------------------
    insert.line <-
        apply(tag.script, 1, simplify = F,
               function(vec){
                 start <- as.numeric(vec[["start"]])
                 end <- as.numeric(vec[["end"]])
                 code <- code[start:end]
                 cluster <- msource(CODE = code, source = F) %>%
                   lapply(
                          function(num){
                            code <- code[num] %>%
                              .[!grepl("^## -{1,}", .)]
                            code.intr <- code[grepl("^## \\[introduction\\] ", code)] %>%
                              gsub("^## \\[introduction\\] ", "", .)
                            ## ---------------------------------------------------------------------
                            code.run <- code[!grepl("^## \\[[a-z]{1,}\\] ", code)]
                            ## show figure of table
                            if(T %in% grepl("^## @[a-z]{1,}", code)){
                              code.run <- gsub("^\\s{0,}# |^## @[a-z]{1,}", "", code.run)
                              return(c(code.intr, code.run))
                            }
                            if(length(code.run) != 0){
                              code.run <- c(
                                "",
                                "```{r, eval = F}",
                                code.run,
                                "```",
                                ""
                              )
                            }
                            code.eval <- code[grepl("^## \\[echo\\]", code)]
                            if(length(code.eval) != 0){
                              code.eval <- c(
                                "",
                                "```{r, echo = T}",
                                code.eval,
                                "```",
                                ""
                              )
                            }
                            gather <- c(code.intr, code.run, code.eval)
                          })
               })
    names(insert.line) <- tag.script$tag
```

```
    for(i in names(insert.line)){
      merge[[i]] <- insert.line[[i]]
    }
    md <- unlist(merge, use.names = F, recursive = T)
    ## ----------------------------------------------------------------------
    savename <- paste0(output, "/reports.Rmd")
    writeLines(md, savename)
    ## output pdf
    rmarkdown::render(savename)
    cat("Done\n")
  }
```

4 File: cat_to_clipboard.R

```
gett <-
  function(
           obj
           ){
    system(paste("echo", obj, "| xsel -b -i"))
  }
```

5 File: classify.gnps.R

```
classify.gnps <-
  function(
           inchikey,
           ...
           ){
    ## url path
    url_start <- "https://gnps-classyfire.ucsd.edu/entities/"
    url_end <- ".json"
    ## gather
    url <- paste0(url_start, inchikey, url_end)
    ## -------------------------------------
    check <- 0
    while(check == 0 | class(check)[1] == "try-error"){
      check <- try(json <- RCurl::getURL(url), silent = T)
    }
    if(grepl("Key not found", json)){
      return(NA)
    }
    ## read json
    list <- rjson::fromJSON(json_str = json, ...)
    return(list)
  }
```

6 File: collate_codes_as_report.R

```
# ==========================================================================
# for codes as report output
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

rough_records <- function(dir, rdname, title, files= NULL)
{
  codes <- get_codes.dir(dir, files = files)
  chunks <- as_chunk.codes(codes)
  lines <- c("---", "---", "", chunks)
  report <- as_report.rough(lines)
  # write_thesisDocx(report, "codes_of_mcnebula2Docx.Rmd", "R codes of MCnebula2")
  write_articlePdf(report, rdname, title)
}

get_codes.dir <- function(dir, files = NULL, pattern = "\\.R$") {
  if (is.null(files)) {
    files <- list.files(dir, full.names = T)
  }
  files <- files[grepl(pattern, files)]
  codes <- sapply(files, readLines, simplify = F)
  names(codes) <- vapply(names(codes), get_filename, character(1))
  codes
}

as_chunk.codes <- function(lst, as_lines = T) {
  name <- names(lst)
  lines <- lst
  fun <- function(lines) {
    lst <- sep_list(lines, "^# =*\\s*$", before = T)
    lst <- lapply(lst,
      function(ch) {
        c('```{r eval = F, echo = T}', ch, '```', "")
      })
```

```
    unlist(lst)
  }
  chunks <- lapply(1:length(name),
    function(n) {
      c(paste0("# File: ", name[n]), "", fun(lines[[ n ]]))
    })
  if (as_lines) {
    chunks <- unlist(chunks)
  }
  chunks
}
```

7 File: compress.R

```
# ==========================================================================
# use about zip, gunzip, gzip, tar ... in R
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

update_tgz <- function(tar.gz, file) { }
```

8 File: create_xlsx.R

```
# ==========================================================================
# create .xlsx via openxlsx2
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

xl_dim <- function(rows, cols) {
  if (max(cols) > 26) {
    stop( "max(cols) > 26" )
  }
  paste0(LETTERS[min(cols)], min(rows),
    ":", LETTERS[max(cols)], max(rows))
}

xl_table <- function(
  data, title = "Table",
  group_by = "Group",
  font = "Times New Roman",
  wb = openxlsx2::wb_workbook())
{
  wb$add_worksheet()
  ## title
  wb$merge_cells(rows = 1, cols = seq_along(data))
  wb$add_data(x = title)
  title_dim <- xl_dim(1, seq_along(data))
  wb$add_font(, title_dim, font, bold = "double")
  ## data
  wb$add_data_table(x = data, startRow = 2, withFilter = F, na.strings = "")
  data_dim <- xl_dim(2:(nrow(data) + 2), 1:ncol(data))
  wb$add_font(, data_dim, font)
  wb$add_cell_style(, data_dim, horizontal = "left", vertical = "top")
  ## group
  group_col <- grep(group_by, colnames(data))
  if (length(group_col) != 0) {
    group <- split(1:nrow(data) + 1, data[[ group_by ]])
    for (i in group) {
      wb$merge_cells(rows = i + 1, cols = group_col)
    }
  }
  ## width
  nchar <- rbind(nchar(colnames(data)), apply(data, 2, nchar))
  nchar.max <- apply(nchar, 2, function(x) max(x, na.rm = T))
  nchar.max <- vapply(nchar.max, function(x) if (x > 30) 30 else x, numeric(1))
  for (i in 1:ncol(data)) {
    wb$set_col_widths(, cols = i, width = nchar.max[i] * 1 + 3)
  }
  ## border
  header_dim <- xl_dim(2, seq_along(data))
  wb$add_border( , header_dim,
    left_border = NULL, right_border = NULL,
    top_border = "double", bottom_border = "double"
  )
  end_dim <- xl_dim(nrow(data) + 2, seq_along(data))
  wb$add_border(, end_dim,
    left_border = NULL, right_border = NULL,
    top_border = NULL, bottom_border = "double"
  )
  return(wb)
}

xl_save <- function(wb, path) {
```

```r
  openxlsx2::wb_save(wb, path)
}
```

9 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."))
               }
             }
           })
  }
```

10 File: DISCARD.R

```r
## to deal with the bug in `grid.grep`
# valide_vp <- function(ch, chs, sort = F){
  # check <- grid.grep(ch, viewports = T, strict = T, no.match = F)
  # if (is(check, "logical")) {
  #   return(F)
  # }
  # if (sort) {
  #   chs <- sort_vpPaths(chs)
  # }
  # nums <- vapply(chs, FUN.VALUE = 0,
  #                 function(ch) {length(grepRaw("::", ch, all = T))})
  # parent.n <- min(nums)
  # match <- grepRaw("::", ch, all = T)
  # end <- match[ parent.n + 1 ] - 1
  # if (!is.na(end)) {
  #   ch.parent <- stringr::str_sub(ch, 1L, end)
  # } else {
  #   ch.parent <- ch
  # }
  # chs <- chs[which( nums > parent.n )]
  # if (length(chs) == 0)
  #   return(F)
  # if (any(!grepl(ch.parent, chs))) {
  #   return(F)
  # }
  # return(ch)
# }

# files <- c("mcn_serum6501.rdata", "serum.tar.gz")
#   urls <- paste0("https://raw.githubusercontent.com/Cao-lab-zcmu/utils_tool/master/",
#                  "inst/extdata/", files)
#   lapply(1:length(files),
#          function(n) {
#            url <- urls[n]
#            data <- RCurl::getURLContent(url)
#            file <- paste0(tmp, "/", files[n])
#            target <- file(file, "wb")
#            writeBin(data, target)
#            close(target)
#          })

# check_pkg <-
#   function(
#            packages = c("data.table", "dplyr", "pbapply", "RCurl", "XML")
#            ){
#     lapply(packages,
#          function(pkg){
#            if (!requireNamespace(pkg, quietly = T))
#              install.packages(pkg)
#          })
#     message("Job Done")
# }

## ----------------------------------------------------------------------
# add_spanner <-
#   function(
#            t,
#            names,
#            group,
#            col_spanner
#            ){
#     envir <- environment()
#     mapply(base_add_spanner, group, col_spanner,
#            MoreArgs = list(envir = envir))
#     return(t)
#   }
# base_add_spanner <-
#   function(
#            the_group,
#            spanner,
#            envir,
#            names = get("names", envir = envir),
#            t = get("t", envir = envir)
#            ){
#     columns <- names %>%
#       .[grepl(the_group, .)]
#     t <- t %>%
#       tab_spanner(label = spanner,
#                   columns = columns)
#     assign("t", t, envir = envir)
#   }
## ----------------------------------------------------------------------
```

11 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)
        })
```

```r
      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 <-
```

```r
  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
```

```r
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)
  }
```

12 File: exReport.R

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

# 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]")
#     )
```

```r
#     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
      ))
```

13 File: exReport2.R

```r
# ==========================================================================
# for conversion of 'report'
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -




as_report.rough <- function(lines) {
  yaml.pos <- grep("^---", lines)
  yaml.pos <- 1:(yaml.pos[2])
  yaml <- lines[ yaml.pos ]
  lines <- lines[-yaml.pos]
  new_report(lines, yaml = yaml)
}

write_biocStyle <- function(
  report, savename, title, change_include_fun = "inclu.fig",
  bioyml = readLines(paste0(.expath, "/", "biocstyle.yml")),
  origin_include_fun = "knitr::include_graphics",
  render = rmarkdown::render,
  bib = NULL)
{
  require("MCnebula2")
  if (is.character(report)) {
    if (file.exists(report)) {
      report <- as_report.rough(readLines(report))
    } else {
      stop( "file.exists(report) == F" )
    }
  }
  if (length(x <- grep("^title:", bioyml)) > 0) {
    bioyml <- bioyml[-x]
  }
  if (!is.null(title)) {
    if (!grepl("^title: ", title)) {
      title <- paste0("title: ", title)
    }
    bioyml <- c(title, bioyml)
  }
  if (!is.null(bib)) {
    bioyml <- gsub("(bibliography:).*", paste0("\\1 ", bib), bioyml)
  }
  yaml(report) <- bioyml
  lines <- call_command(report)
  if (!is.null(change_include_fun)) {
    lines <- gsub(origin_include_fun, change_include_fun, lines)
  }
  writeLines(lines, savename)
  if (is.function(render)) {
    render(savename)
  }
}
```

```r
write_thesisDocx <- function(report, savename, title,
  change_include_fun = "inclu.fig",
  yml = readLines(paste0(.expath, "/", "ch_thesis.yml")),
  origin_include_fun = "knitr::include_graphics", ...)
{
  write_biocStyle(report, savename, title, change_include_fun, yml, origin_include_fun, ...)
}

write_thesisDocxEn <- function(report, savename, title,
  change_include_fun = "inclu.fig",
  yml = readLines(paste0(.expath, "/", "en_thesis.yml")),
  origin_include_fun = "knitr::include_graphics", ...)
{
  write_biocStyle(report, savename, title, change_include_fun, yml, origin_include_fun, ...)
}

write_articlePdf <- function(report, savename, title,
  change_include_fun = NULL,
  yml = readLines(paste0(.expath, "/", "articleWithCode.yml")),
  origin_include_fun = "knitr::include_graphics", ...)
{
  write_biocStyle(report, savename, title, change_include_fun, yml, origin_include_fun, ...)
}

kable_less <- function(x, ...) {
  if (nrow(x) > 50) {
    x <- head(x, n = 25)
  }
  knitr::kable(x, ...)
}
```

14 File: find_latest_script.R

```r
find_latest_script <-
  function(
           path = "~/outline",
           pattern = ".*R$"
           ){
    set <- list.files(path = path,
                      pattern = pattern,
                      full.names = T, recursive = T)
    df <- file.info(set) %>%
      dplyr::mutate(file = rownames(.)) %>%
      dplyr::relocate(file) %>%
      dplyr::arrange(desc(mtime)) %>%
      dplyr::as_tibble()
    return(df)
  }
ssource <-
  function(
           path = "~/outline",
           pattern = ".*R$"
           ){
    script <- find_latest_script(path, pattern)$file[1]
    cat("[INFO]: Latest script is:", script, "\n")
    source(script)
  }
rrender <-
  function(
           path = "~/outline",
           pattern = "md$",
           format = "pdf_document"
           ){
    script <- find_latest_script(path, pattern)$file[1]
    cat("[INFO]: Latest script is:", script, "\n")
    rmarkdown::render(script, format)
  }
msource <-
  function(
           path = "~/outline",
           pattern = ".*R$",
           block = "^## -{1,}",
           symbol = "^## ========== Run block ==========",
           extra = NA,
           script = NA,
           CODE = NA,
           source = T
           ){
    if(is.na(script))
      script <- find_latest_script(path, pattern)$file[1]
    ## ---------------------------------------
    if(length(CODE) == 1 & is.na(CODE[1])){
```

```r
    CODE <- data.table::fread(file = script, header = F, sep = NULL)
  }
  if(!is.data.frame(CODE)){
    CODE <- data.table::data.table(CODE)
  }
  df <- CODE %>%
    dplyr::rename(code = 1) %>%
    dplyr::mutate(valid = ifelse(!grepl(block, code), T, F),
                  valid = ifelse(grepl(symbol, code), T, valid),
                  row = rownames(.),
                  line = ifelse(valid, row, ","))
  ## ------------------------------------
  cluster <- paste(df$line, collapse = ",") %>%
    gsub(",{2,}", "-", .) %>%
    strsplit(split = "-") %>%
    unlist() %>%
    strsplit(split = ",") %>%
    lapply(as.numeric)
  if(!source)
    return(cluster)
  ## ------------------------------------
  run.block <- grep(symbol, df$code)
  if(length(class) == 0){
    run.block <- length(df$code)
  }
  if(!is.na(extra)){
    run.block <- c(run.block, extra)
  }
  ## ------------------------------------
  script <- lapply(cluster,
                   function(vec){
                     if(T %in% (run.block %in% vec))
                       return(vec)
                   }) %>%
    unlist() %>%
    df$code[.] %>%
    paste(collapse = "\n") %>%
    parse(text = .)
  ## ------------------------------------
  source(exprs = script)
  ## ------------------------------------
```

```
    # source(exprs = parse(text = script))
  }
```

15 File: flowChart.R

```
# ==========================================================================
# help with drawing flow chart
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

sortSugi <- function(values, dec = T) {
  num <- values
  i <- 1L
  order <- order(values, decreasing = dec)
  num[order] <- vapply(num[order],
    FUN.VALUE = integer(1),
    function(value) {
      x <- i
      i <<- i + 1L
      return(x)
    })
  num
}

as_network <- function(lst, layout = 'sugiyama', seed = 100)
{
  lst <- vapply(lst, function(ch) gsub(" ", "", ch), character(1))
  if (any(grepl("::", lst))) {
    lst <- unlist(lapply(lst,
        function(ch) {
          if (grepl("::", ch)) {
            ch <- strsplit(ch, "::")[[ 1 ]]
            ch <- c(paste0(ch, collapse = ":"), paste0(rev(ch), collapse = ":"))
          } else ch
        }))
  }
  lst <- strsplit(lst, ":")
  lst <- lapply(lst,
    function(ch) {
      ch <- strsplit(ch, ",")
      names(ch) <- c("from", "to", n(attr, length(ch) - 2))
      do.call(data.frame, ch)
```

```r
    })
  data <- data.table::rbindlist(lst)
  if (!is.null(seed)) {
    data <- lapply(seed,
      function(s) {
        set.seed(s)
        fast_layout(data, layout)
      })
    if (length(seed) == 1)
      data <- data[[1]]
  }
  data
}

#' @import ggplot2
flowChart <- function(graph, scale.x = 1.2, scale.y = 1.2, node.size = 4,
  sc = 8, ec = 8, arr.len = 2, edge.color = 'lightblue', edge.width = 1)
{
  if (is(graph, "layout_tbl_graph")) {
    graphs <- list(graph)
    num <- 1L
  } else {
    graphs <- graph
    num <- 2L
  }
  p.lst <- lapply(graphs,
    function(graph) {
      p <- ggraph(graph) +
        geom_edge_fan(aes(x = x, y = y),
          start_cap = circle(sc, 'mm'),
          end_cap = circle(ec, 'mm'),
          arrow = arrow(length = unit(arr.len, 'mm')),
          color = edge.color, width = edge.width) +
        geom_node_label(aes(label = paste0(sortSugi(y), ". ", name)),
          size = node.size, label.padding = u(.5, lines)) +
        scale_x_continuous(limits = zoRange(graph$x, scale.x)) +
        scale_y_continuous(limits = zoRange(graph$y, scale.y)) +
        theme_void()
      p
    })
  if (num == 1L) {
    return(p.lst[[1]])
  } else {
    preview.gl(p.lst)
  }
  p.lst
}

preview.gl <- function(p.lst) {
  lst <- lapply(p.lst, as_grob)
  names(lst) <- n(p, length(lst))
  panel <- frame_col(fill_list(names(lst), 1), lst)
  legend <- sapply(names(lst), simplify = F, gtext, gp_arg = list(cex = 2))
  legend <- frame_col(fill_list(names(legend), 1), legend)
  frame <- frame_row(c(legend = 1, panel = 5), namel(panel, legend))
  dev.new(width = 20)
  draw(frame)
}

zoRange <- function (x, factor)
{
  x <- range(x)
  ex <- abs(x[2] - x[1]) * (factor - 1)
  x[1] <- x[1] - ex
  x[2] <- x[2] + ex
  return(x)
}

ll <- function(str, sep = ":", link = ":") {
  strs <- strsplit(str, split = sep)[[ 1 ]]
  lst <- list()
  length(lst) <- length(strs) - 1
  for (i in 1:(length(strs) - 1)) {
    lst[[ i ]] <- paste0(strs[i], ":", strs[i + 1])
  }
  return(lst)
}
```

16 File: generate_slidy.R

```
# ==========================================================================
# slidy
```

```r
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

generate_slidy <- function(name, path = "/mnt/data/wizard/Documents/zcmu_reports")
{
  file <- paste0(path, "/", name, ".Rmd")
  meta <- generate_slidy_meta()
  cat(meta, file = file)
}

make_slidy <- function(name, path = "/mnt/data/wizard/Documents/zcmu_reports")
{
  file <- paste0(path, "/", name, ".Rmd")
  rmarkdown::render(file)
}

preprocess_bib <- function(file = paste0(.expath, "/library.bib")) {
  lst <- read_bib(file)
  which <- grepl("^@[0-9]*$", names(lst))
  lst[which] <- lapply(lst[which],
    function(l) {
      l[1] <- gsub("\\{", "{FIXN", l[1])
      return(l)
    })
  writeLines(unlist(lst), "library.bib")
}

reload_bib <- function(file = "library.bib",
  exclude = c("doi", "urldate", "issn", "address", "isbn"))
{
  bib <- bibtex::read.bib(file)
  bib <- fix_bib(bib, exclude)
  assign(".bib", bib, envir = topenv())
}

fix_bib <- function(bib, exclude = c("doi", "urldate", "issn", "address", "isbn"))
{
  bib <- lapply(bib,
    function(b) {
      expr <- paste0(paste0("b$", exclude, " <- NULL"), collapse = "\n")
      eval(parse(text = expr))
      return(b)
    })
  bib <- do.call(c, bib)
  return(bib)
}

citethis <- function(..., trunc = T, trunc.author = 1, trunc.title = F,
  prefix = "\\tiny ", sep = "\\vspace{0.5em} \\newline\n",
  pkgs = NULL, pkgs.fix = fix_bib, exbibentry = NULL, postFun = NULL)
{
  keys <- list(...)
  if (length(keys) > 0) {
    keys <- vapply(keys,
      function(ch) {
        if (is.numeric(ch) | grepl("^[0-9]*$", ch)) {
          return(paste0("FIXN", ch))
        } else {
          ch
        }
      }, character(1))
    keys <- paste0(keys, ".", keys)
    if (!exists(".bib", where = topenv())) {
      reload_bib()
    }
    all <- names(.bib)
    bib <- .bib[ match(keys, all) ]
  } else {
    bib <- bibentry()
  }
  if (!is.null(pkgs)) {
    bib.pkg <- lapply(pkgs,
      function(pkg) {
        c(citation(pkg))
      })
    bib.pkg <- do.call(c, bib.pkg)
    if (!is.null(pkgs.fix)) {
      bib.pkg <- pkgs.fix(bib.pkg)
    }
  } else {
    bib.pkg <- bibentry()
  }
  bib <- c(bib, bib.pkg)
```

```r
  if (!is.null(exbibentry)) {
    bib <- c(bib, exbibentry)
  }
  if (trunc) {
    bib <- lapply(bib,
      function(b) {
        if (length(b$author) > trunc.author) {
          b$author <- c(b$author[1:trunc.author], person("et al."))
        }
        if (trunc.title) {
          b$title <- stringr::str_trunc(gsub("\\{|\\}", "", b$title), 20)
        } else {
          b$title <- "#";
        }
        b$number <- NULL
        b$volume <- NULL;
        b$pages <- NULL
        return(b)
      })
    bib <- do.call(c, bib)
  }
  if (!is.null(bib))
    writeLines(prefix)
  text <- format(bib, "text")
  if (!trunc.title) {
    text <- gsub("[\" “” #]\\.*", "", text)
  }
  if (!is.null(postFun)) {
    text <- vapply(text, postFun, character(1))
  }
  writeLines(text, sep = sep)
}

testSection <- function(file, pattern, level = 2, render = rmarkdown::render) {
  lines <- readLines(file)
  yml <- getyml(lines)
  getSecPos <- function(lines, level) {
    pattern <- paste0("^", paste0("#{", level, "}[^#]"))
    grep(pattern, lines)
  }
  sec.pos <- getSecPos(lines, level)
  tar.pos <- grep(pattern, lines)
  if (length(tar.pos) == 0) {
    stop("length(tar.pos) == 0")
  }
  if (length(tar.pos) > 1) {
    warning("length(tar.pos) > 1")
    tar.pos <- tar.pos[1]
  }
  if (tar.pos < (yend <- attr(yml, "pos")$end)) {
    stop("tar.pos < attr(yml, 'pos')$end")
  }
  if (any(sec.pos == tar.pos)) {
    start.pos <- tar.pos
    if (level > 1) sec.pos <- getSecPos(lines, paste0("1,", level))
    end.pos <- sec.pos[head(which(sec.pos > tar.pos), n = 1)]
  } else {
    start.pos <- sec.pos[tail(which(sec.pos < tar.pos), n = 1)]
    if (level > 1) sec.pos <- getSecPos(lines, paste0("1,", level))
    end.pos <- sec.pos[head(which(sec.pos > tar.pos), n = 1)]
  }
  end.pos <- end.pos - 1
  if (length(start.pos) == 0) {
    start.pos <- yend + 1
  }
  if (start.pos > length(lines)) {
    stop("start.pos > length(lines)")
  }
  if (length(end.pos) == 0) {
    end.pos <- length(lines)
  }
  if (end.pos < start.pos) {
    stop("end.pos < start.pos")
  }
  content <- lines[ start.pos:end.pos ]
  lines <- c(yml, "", content)
  writeLines(lines, nfile <- paste0("_temptest_", get_filename(file)))
  if (!is.null(render)) {
    output <- render(nfile)
    op(output)
  }
}
```

```r
getyml <- function(lines) {
  pos <- grep("^---", lines)
  if (length(pos) != 2) {
    stop("length(pos) != 2")
  }
  yml <- lines[pos[1]:pos[2]]
  attr(yml, "pos") <- list(start = pos[1], end = pos[2])
  yml
}

arrange_figsPath <- function(file, pattern = "^!\\[.*\\]", to = NULL, overwrite = F)
{
  path <- get_path(file)
  filename <- get_filename(file)
  if (!is.null(to)) {
    if (!dir.exists(to)) {
      stop("dir.exists(to) == F")
    } else {
      dir <- to
    }
  } else {
    dir <- paste0(path, "/", gsub("\\.[a-zA-Z]*$", "", filename))
  }
  lines <- readLines(file)
  pos <- grep(pattern, lines)
  fun <- function(n) {
    line <- lines[n]
    file <- gsub("\"", "", stringr::str_extract(line, "(?<=\\]\\().*(?=\\))"))
    filename <- get_filename(file)
    nfile <- paste0(dir, "/", filename)
    line <- gsub("\\]\\(.*\\)", paste0("](", nfile, ")"), line)
    file.copy(file, dir, overwrite)
    line
  }
  for (i in pos) {
    lines[i] <- fun(i)
  }
  writeLines(lines, file)
}

latexfig <- function(file, caption = NULL, scale.width = 0.7, scale.height = 0.45)
```

```r
{
  if (grepl("\\.pdf$", file)) {
    info <- pdftools::pdf_pagesize(file)
  } else {
    info <- bitmap_info(file)
  }
  ratio <- info$width / info$height
  width <- normSize(ratio, list(width = scale.width, height = scale.height))$width
  md <- c(paste0("::: {.col data-latex=\"{", round(width, 2), "\\textwidth}\"}"),
    paste0("![", caption, "](", file, ")"),
    ":::")
  writeLines(md)
}

hlName <- function(text, name = "Huang L") {
  gsub(paste0("(", name, ")"), "\\\\underline{\\\\textbf{\\1}}", text)
}

normSize <- function(ratio, scale)
{
  if ((scale$width / scale$height) >= ratio) {
    ## height as reference
    height <- scale$height
    width <- height * ratio
  } else {
    ## width as reference
    width <- scale$width
    height <- width / ratio
  }
  list(height = height, width = width)
}

bitmap_info <- function(file) {
  img <- magick::image_read(file)
  info <- magick::image_info(img)
  magick::image_destroy(img)
  info
}

get_title <- function(file) {
  lines <- readLines(file)
  pos <- grepl("^#", lines)
  lines[pos]
}
```

17 File: get_hierarchy.in_df.R

```r
get_hierarchy.in_df <-
  function(
           df,
           col = "Classification"
           ){
    deep <- mutate_get_parent_class(df[[col]], class_cutof = 1) %>%
      lapply(function(vec){length(vec) + 1}) %>%
      unlist()
    df <- dplyr::mutate(df, hierarchy = unlist(lapply(eval(parse(text = col)),
                                                      function(class){
                                                        deep[[class]]
                                                      })))
    return(df)
  }
```

18 File: get_metadata_from_names.R

```r
get_metadata_from_names <-
  function(
           names,
           meta.group
           ){
    metadata <- meta.group %>%
      lapply(function(vec){
               str <- .meta_find_and_sort(names, vec)
           })
    metadata <- mapply(metadata, names(metadata), SIMPLIFY = F,
                       FUN = function(vec, name){
                         df <- data.table::data.table(group = name, sample = vec)
                         return(df)
                       })
    metadata <- data.table::rbindlist(metadata)
    return(metadata)
  }
```

19 File: grid_draw.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)
```

```r
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), ...)
  })

#' @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
```

```r
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 ...
```

```r
#'
#' @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))
}
```

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

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

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

#' @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)
```

```r
    do.call(fun_line, args_line)
  })

#' @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){
```

```r
  sagnage(gtext(label, gpar(col = "white", fontface = "plain")),
    left, gpar(fill = color))$grob_anno
}

#' @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)
```

```r
        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))
}
```

```r
# ==========================================================================
# 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)))
```

```r
# ==========================================================================
# 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)
```

```r
#' @export .grecto.vp
.grecto.vp <- do.call(viewport, .fresh_param2(.grecti.vp.p, list(clip = "inherit")))

#' @export grect
#' @aliases grect
#' @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), ...){
```

```r
  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
#' @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
```

```r
#' @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(){
```

```r
  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)
}
```

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

```r
#' @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)
  }
```

```r
# ==========================================================================
# 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)
}
```

```r
#' @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)
}
```

```r
# ==========================================================================
# 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)
}
```

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

#' @export nebulae_as_grob
#' @aliases nebulae_as_grob
#' @title Convert Nebulae as 'grob' object
#' @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
}
```

20 File: hugo_doks.R

```r
set_hugoDir <- function(path) {
  assign("hugoDir", path, topenv())
}

hugoDir <- "~/siteBlog/"
```

```r
new_notes <- function(scene, weight, parent = "notes") {
  names(scene) <- rep(parent, length(scene))
  weight <- rep(weight, length(scene))
  ex_weight <- weight + 1:length(scene)
  new_scene(scene, weight, ex_weight)
}

new_scene <- function(scene, weight = rep(100, length(scene)), ex_weight = weight,
    path = hugoDir, suffix = "/content/en", tar = "docs", index_Draft = T){
    if (!is.vector(scene)) {
      stop("is.vector(scene) == F")
    }
    if (!is.character(names(scene))){
      stop( "is.character(names(scene)) == F" )
    }
    path <- paste0(path, suffix)
    path <- target_dir(path, tar)
    lapply(1:length(scene),
      function(n){
        name <- names(scene)[n]
        dir <- paste0(path, "/", name)
        if (!dir.exists(dir)) {
          dir.create(dir)
          if (index_Draft) {
            index <- paste0(dir, "/_index.Rmd")
            writeLines(index_Draft(name, weight[[n]]), index)
          }
        }
        file <- paste0(dir, "/", scene[[n]], ".Rmd")
        if (!file.exists(file)) {
          writeLines(Draft(scene[[n]], ex_weight[[n]], tar, name), file)
        }
      })
    message("Done")
  }

target_dir <-
  function(path, tar){
    lst <- list.dirs(path, recursive = T)
    lst <- lst[grepl(paste0("(?<=/)", tar, "$"), lst, perl = T)][1]
    lst
  }

record_time <- function(){
  format(Sys.time(), "%Y %b %e %H:%M:%S | %a")
}

Draft <- function(title, weight = 100, tar, name){
  title <- gsub("_", " ", title)
  c("---",
    "contributors:\n- LiChuang Huang",
    paste0("title: ", "\"", Hmisc::capitalize(title), "\""),
    paste0("date: ", "\"", record_time(), "\""),
    paste0("lastmod: ", "\"", record_time(), "\""),
    "draft: false",
    "images: []",
    "menu:",
    strwrap(paste0(tar, ":"), indent = 2),
    paste0(strwrap("parent:", indent = 4), " \"", name, "\""),
    paste0("weight: ", weight),
    "toc: true",
    "---"
  )
}

index_Draft <- function(title, weight = 100){
  title <- gsub("_", " ", title)
  c("---",
    paste0("title: ", "\"", Hmisc::capitalize(title), "\""),
    paste0("date: ", "\"", format(Sys.time(), record_time()), "\""),
    paste0("lastmod: ", "\"", format(Sys.time(), record_time()), "\""),
    "draft: false",
    "images: []",
    paste0("weight: ", weight),
    "toc: true",
    "---"
  )
}

target_file <-
  function(path, tar){
    lst <- list.files(path, recursive = T, full.names = T)
```

```r
    lst <- lst[grepl(paste0("(?<=/)", tar, "$"), lst, perl = T)][1]
    lst
  }

setGeneric("set_home",
  function(x, ...) standardGeneric("set_home"))
setMethod("set_home",
  signature = setMissing("set_home"),
  function(){
    function(path = paste0(hugoDir, "/config"), tar = "params.toml") {
      target_file(path, tar)
    }
  })

setMethod("set_home",
  signature = setMissing("set_home",
    x = "vector"),
  function(x, ...){
    path <- set_home()(...)
    lines <- readLines(path)
    for (i in names(x)) {
      lines <- repl_huto(i, x[[i]], lines)
    }
    writeLines(lines, path)
  })

repl_huto <-
  function(key, content, lines,
    left = "\"", right = "\"", link = " = "){
    pattern <- paste0("^\\s{0,}", key, "\\s{0,}(?![a-z|A-Z|0-9|_|.])")
    n <- grep(pattern, lines, perl = T)
    pattern <- paste0("(?<=", link, left, ")", "[^\"]{1,}", "(?=", right, ")")
    lines[n] <- gsub(pattern, content, lines[n], perl = T)
    lines
  }
repl_yaml <- function(key, content, lines){
  repl_huto(key, content, lines, link = ": |: \"",
    left = "", right = "\"|$"
  )
}
```

```
setGeneric("set_index",
  function(x, tar, ...) standardGeneric("set_index"))
setMethod("set_index",
  signature = setMissing("set_index"),
  function(){
    function(tar = "en/_index.Rmd", path = paste0(hugoDir, "/content/en")) {
      target_file(path, tar)
    }
  })

setMethod("set_index",
  signature = setMissing("set_index",
    x = "vector",
    tar = "character"),
  function(x, tar, ...){
    path <- set_index()(tar, ...)
    lines <- readLines(path)
    time <- grep("^lastmod|^date", lines)
    for (i in time) {
      if (!grepl(": \"", lines[i]))
        lines[i] <- sub(": ", ": \"", lines[i])
      if (!grepl("\"$", lines[i]))
        lines[i] <- sub("$", "\"", lines[i])
    }
    for (i in names(x)) {
      lines <- repl_yaml(i, x[[i]], lines)
    }
    writeLines(lines, path)
  })

inht2 <- inclu.fig.ht2 <- function(src, caption = "...",
  file = get_filename(src), parent = "/docs/notes/",
  parent.ex = "figs", width = "100%", height = NULL,
  rel.path = paste0(hugoDir, "/content/en"))
{
  inclu.fig.ht(src, to = paste0(parent, "/", parent.ex, "/", file), caption,
    width, height, rel.path)
}

smallsvg <- function(p, file, width = 4, height = 3, mkdir = "figs") {
  if (!file.exists(mkdir))
```

```
    dir.create(mkdir)
  ggsave(paste0(mkdir, "/", file), p, width = width, height = height)
}

inclu.fig.ht <- function(src, to, caption = "This is a figure",
  width = "100%", height = NULL,
  rel.path = paste0(hugoDir, "/content/en"))
{
  if (!file.exists(paste0(rel.path, to))) {
    if (!file.exists(src)) {
      stop("file.exists(", src, ") == F")
    }
    if (grepl("\\.pdf$", src)) {
      system(paste0("pdf2svg ", src, " ", rel.path, to, " 1"))
    } else {
      file.copy(src, paste0(rel.path, to))
    }
  }
  draft <- c("<figure>", "", "", "</figure>")
  img <- paste0("<center>", "<img src=\"", to, "\"", ">", "</center>")
  cap <- paste0("<center>", "<figcaption>", caption, "</figcaption>", "</center>")
  draft[2:3] <- c(img, cap)
  writeLines(draft)
}
```

21 File: lite_research.R

```
# ==========================================================================
# plot or ...
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

bibnetwork <- function(db, save, topic = "keywords", n = 30,
  labelsize = 1.2, seed = 20, cluster = "none", width = 12, height = 7,
  title = "", ...)
{
  NetMatrix <- bibliometrix::biblioNetwork(
    db, analysis = "co-occurrences", network = topic, sep = ";"
  )
  pdf(save, width, height)
  set.seed(seed)
  net <- bibliometrix::networkPlot(
    NetMatrix, normalize = "association",
    weighted = T, n = n, Title = title,
    type = "fruchterman", size = T, edgesize = 5, labelsize = labelsize,
    cluster = cluster, ...
  )
  dev.off()
  return(net)
}
```

22 File: lite.citation.R

```
# ==========================================================================
# manualy set figure number in artical
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
pandoc.docx <- function(
    file.md,
    format = ".docx",
    output = sub("\\.[a-z]*$", format, file.md),
    deal_with = T,
    script = "pandoc.sh",
    script_path = system.file("extdata", script, package = "utils.tool"),
    template = "template.tex",
    hasLink = T
  )
  {
    ## read md
    md <- readLines(file.md)
    if (deal_with) {
      citation <- stringr::str_extract(md, "^\\[citation\\]: .*$")
      ## get all citation
      citation <- citation[!is.na(citation)]
      citation.key <- stringr::str_extract(citation, "(?<=@).{1,20}(?=:)")
      ## unique
      citation.key <- unique(citation.key)
      ## paste as pattern.set
      pattern.set <- paste0(
        "\\{@", citation.key, ":[^{]{1,50}\\}",
        if (hasLink) "\\{nolink=True\\}" else NULL
      )
      record <-  lapply(pattern.set,
        function(pattern){
          cite <- stringr::str_extract(citation, pattern)
          cite <- cite[!is.na(cite)]
          ## as table
          df <- data.table::data.table(cite = cite, seq = 1:length(cite))
          ## replace
          md <- mapply_rename_col(df$cite, df$seq, md)
          ## push to parent envir
          assign("md", md, envir = parent.frame(2))
          return(df)
        })
    }
    cat(md, sep = "\n", file = ".TMP.md")
    tmp_script <- paste0(".TMP.", script)
    file.copy(script_path, tmp_script)
    file.copy(paste0(.expath, "/custom-reference.docx"), "custom-reference.docx")
    system(paste0("bash ", tmp_script, " .TMP.md ", output, " ", template))
    file.remove(tmp_script, "custom-reference.docx")
    if (deal_with)
      return(record)
  }

read_captions <- function(file.md, sig = "Fig. \\{@.*\\|")
{
  lst <- sep_list(readLines(file.md))
  pos <- vapply(lst, function(ch) grepl(sig, ch[1]), logical(1))
  lst <- lapply(lst[pos],
    function(ch) {
      ch <- paste0(ch, collapse = " ")
      gsub("^.* \\| ", "**", ch)
    })
  lst
}

read_bib <- function(bib) {
  lst <- sep_list(readLines(bib))
  getKey.biblst(lst)
}

shunt_bib <- function(bib, keys, export = "library.bib"){
  lst <- read_bib(bib)
  lst <- lst[names(lst) %in% keys]
  lst <- unlist(lst)
  writeLines(lst, export)
}

asTex.rmd <- function(file.md, yaml = paste0(.expath, "/", "article.yml"),
  export = paste0("tex_", sub("\\.[a-z]*$", "", get_filename(file.md))),
  bib = paste0(.expath, "/library.bib"), style = paste0(.expath, "/style.csl"))
{
  dir.create(export)
  file.copy(file.md, nfile.md <- paste0(export, "/",
      sub("\\.[a-z]*$", ".Rmd", get_filename(file.md))), T)
  md <- readLines(nfile.md)
  fig.pos <- grep("^!\\[.*\\]\\(.*\\)", md)
  fig.num <- 1
  md[fig.pos] <- vapply(md[fig.pos], FUN.VALUE = character(1),
    function(ch) {
      file <- stringr::str_extract(ch, "(?<=\\]\\().*(?=\\))")
      if (!grepl("^!\\[\\]", ch)) {
        prefix <- paste0("fig", fig.num, ".")
        fig.num <<- fig.num + 1
      } else {
        prefix <- character(1)
      }
      nfile <- gsub("fig[0-9]\\.", prefix, get_filename(file))
      whether <- file.copy(file, paste0(export, "/", nfile), T)
      if (!whether) stop("The file of figure not found")
      gsub(file, nfile, ch, fixed = T)
    })
  yml <- c("---", readLines(yaml), "---\n")
  writeLines(c(yml, md), nfile.md)
  shunt_bib(bib, extract_ref(nfile.md), paste0(export, "/", "library.bib"))
  file.copy(style, paste0(export, "/", get_filename(style)), T)
  writeLines("Done")
}

sep_list <- function(lines, sep = "^\\s*$", before = F)
{
  seps <- grep(sep, lines) + if (before) 0 else 1
  group <- 1
  groups <- vapply(1:length(lines), FUN.VALUE = double(1),
    function(n) {
      if (any(n == seps))
        group <<- group + 1
      group
    })
  split(lines, groups)
}

getKey.biblst <- function(lst){
  names(lst) <- vapply(lst, FUN.VALUE = character(1),
    function(lines) {
      paste0("@", stringr::str_extract(lines[1], "(?<=\\{).*(?=,$)"))
    })
  lst
}

getFilePath.biblst <- function(lst){
  lapply(lst,
    function(lines){
      n.file <- grep("^\\s*file", lines)
      if (length(n.file) == 1) {
        stringr::str_extract(lines[n.file], "(?<=\\{).*(?=\\})")
      } else {
        NULL
      }
    })
}

fix_ch2en <- function(md){
  md <- gsub(" (", " (", md)
  md <- gsub(") ", ") ", md)
  md <- gsub(": ", ": ", md)
  md <- gsub("[\u2002]", " ", md)
  md
}

cite_extract <- function(data) {
  data <- dplyr::mutate(
    data, cite = stringr::str_extract(cite, "(?<=:)[a-zA-Z0-9_.]{1,}(?=\\})")
  )
  .as_dic(data$seq, data$cite)
}
```

```r
fts_copy <- function(ft, seq.lst, path, prefix = "fig", sub_target = "fts") {
  lapply(ft, function(x){
    name <- gsub("\\..*$", "", x[2])
    file.copy(paste0(path, "/", x[1]),
      paste0(path, "/", sub_target, "/", prefix, seq.lst[[ name ]], ".", x[2]), T)
  })
}

insert_tocg <- function(md, fig) {
  pos <- grep("^## Abstract", md)
  blanks <- grep("^\\s*$", md)
  pos <- blanks[blanks > pos + 1][1]
  md[pos] <- paste0(md[pos], "\n", "\\includegraphics", "{", fig, "}", "\n")
  md
}

insert_figs <- function(md, ids, figs, captions,
  patterns = paste0("Fig\\. ", 1:length(ids)),
  at_ref = paste0("Fig. {@fig:", ids, "}"),
  figs_command = paste0("![", gsub("\n", "", captions), "]",
    "(", figs, ")", "{#fig:", ids, "}"))
{
  fb <- function(n, blanks) {
    blanks[blanks > n][1]
  }
  insert <- function(i, ch) {
    paste0(ch, "\n", figs_command[i], "\n")
  }
  rp <- function(ch, i){
    gsub(patterns[i], at_ref[i], ch)
  }
  blanks <- grep("^\\s*$", md)
  for (i in 1:length(ids)) {
    ns <- grep(patterns[i], md)
    md[ns] <- unlist(vapply(md[ns], rp, character(1), i = i))
    insert.pos <- fb(ns[1], blanks)
    md[insert.pos] <- insert(i, md[insert.pos])
  }
  return(md)
}
```

```r
# ==========================================================================
# post modification (reference version dependent)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## Science bulletin

extract_ref_pan2md.sci_bull <- function(md){
  ref <- unlist(stringr::str_extract_all(md, "\\\\\\[\\^\\[.*?\\)\\\\\\]"))
  generate_ref_key(ref)
}

## Vancouver (superscript); ac

revise_pan2md.vanco <- function(file, output = "tmp.md"){
  md <- readLines(file)
  md <- revise_symbol_pan2md(md)
  ## ref
  md <- gsub("log2", "log~2~", md)
  md <- gsub("log10", "log~10~", md)
  md <- gsub("\\(#ref-[0-9a-zA-Z_.]*\\)", "(\\\\1)", md)
  md <- gsub("\\[\\^", "^[", md)
  md <- gsub("\\^\\]\\(\\\\1\\)", "](\\\\1)^", md)
  md <- gsub("\\^\\]", "]^", md)
  md <- gsub("\\(\\\\1\\)(?!,\\[|\\-\\-|\\^)", "(\\\\1)^", md, perl = T)
  md <- gsub("(?<!\\^|1\\),|\\-\\-)\\[(?=[0-9])", "^[", md, perl = T)
  return(md)
}

extract_ref_pan2md.vanco <- function(md) {
  ref <- unlist(stringr::str_extract_all(md, "\\^\\[.*?\\)\\^|\\^\\[[0-9\\-]*\\]\\^"))
  generate_ref_key(ref)
}
```

```r
# ==========================================================================
# utilites for revise
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

revise_pan2md <- function(file, output = "tmp.md") {
  md <- readLines(file)
  md <- revise_symbol_pan2md(md)
}
```

```r
revise_symbol_pan2md <- function(md) {
  md <- gsub("\\\\<", "&lt;", md)
  md <- gsub("\\\\>", "&gt;", md)
  md
}

## the sep is ",|--"
generate_ref_key <- function(ref, pattern = "(?<=\\[)[0-9]{1,}(?=\\])") {
  ref_num <- stringr::str_extract_all(ref, pattern)
  ref_sep <- stringr::str_extract_all(ref, ",|--")
  key <- lapply(1:length(ref_num),
    function(n) {
      ch <- rep("", length(ref_num[[n]]) * 2 - 1)
      for (i in 1:length(ch)) {
        if (i %% 2 == 1)
          ch[i] <- ref_num[[n]][(i + 1) / 2]
        else
          ch[i] <- ref_sep[[n]][i / 2]
      }
      set <- integer(0)
      i <- 1
      if (length(ch) > 1) {
        while(i < length(ch)) {
          if (ch[ i + 1 ] == ",") {
            set <- c(set, as.integer(ch[i]))
          } else if (ch[ i + 1] == "--") {
            set <- c(set, as.integer(ch[i]):as.integer(ch[i + 2]))
          }
          i <- i + 2
        }
      } else {
        set <- as.integer(ch)
      }
      set
    })
  names(key) <- ref
  key
}

extract_ref <- function(file) {
  md <- readLines(file)
  key <- stringr::str_extract_all(md, "(?<=\\[|; |^)@[0-9|a-z|A-Z|_\\-]*(?=;|\\])")
  key <- unique(unlist(key))
  key[!grepl("@fig|@tab|@.*fig$|@.*tab$", key)]
}

comb_ref <- function(n, ref) {
  com <- paste0(ref[n], collapse = "; ")
  paste0(" [", com, "]")
}

get_path <- function(path_str){
  if (!grepl("/", path_str))
    return(".")
  stringr::str_extract(path_str, ".*(?=/)")
}

get_filename <- function(path_str){
  if (!grepl("/", path_str))
    return(path_str)
  stringr::str_extract(path_str, "(?<=/)[^/]*$")
}
```

```r
# ==========================================================================
# modify figure citation
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

rm.instFig <- function(md, getID = T) {
  pos <- grepl("^!\\[", md)
  target <- md[pos]
  md[pos] <- ""
  if (getID) {
    ids <- stringr::str_extract(
      target, "(?<=\\{#).*(?=\\})"
    )
    notes <- paste0('[citation]: {@', ids, '}\n')
    md <- c(md, notes)
  }
  return(md)
}
```

23 File: make_temp.R

```r
# ==========================================================================
# Backing up temporary files
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

backup.tmp <- function(tmp, backup_path = "~/tmp_backup/R") {
  if (!file.exists(backup_path))
    dir.create(backup_path, recursive = T)
  if (!file.exists(tmp))
    stop("file.exists(tmp) == F")
  file.copy(tmp, backup_path, T, T)
}
```

24 File: metabo_collate.R

```r
metabo_collate <-
  function(
           path = "~/Desktop"
           ){
    ## ---------------------------------------------------------------------
    ## read file
    compound <- list.files(path, pattern = "compound_all.{0,5}.csv$", full.names = T) %>%
      data.table::fread()
    pathway <- list.files(path, pattern = "pathway_enrichment.{0,5}.csv$", full.names = T) %>%
      data.table::fread()
    ## -------------------------------------
    pathway <- metabo_collate_pathway(pathway)
    ## -------------------------------------
    compound <- metabo_collate_compound(compound)
    ## ---------------------------------------------------------------------
    ## gather pathway and compound
    list <- lapply(pathway, merge, y = compound,
                   by.x = "compound", by.y = "Empirical.Compound", all.x = T) %>%
      lapply(dplyr::as_tibble)
    return(list)
  }
metabo_collate_pathway <-
  function(
           pathway
           ){
```

```r
    db <- dplyr::rename(pathway, pathway = V1) %>%
      by_group_as_list("pathway") %>%
      lapply(add_row_via_separate_col, only_split = F)
    return(db)
  }
add_row_via_separate_col <-
  function(
           df_row,
           col = "EC.Hits",
           only_split = F
           ){
    vector <- df_row[[col]] %>%
      unlist(use.names = F) %>%
      strsplit(split = ";") %>%
      unlist(use.names = F)
    ## ------------------
    if(only_split == T)
      return(vector)
    ## ------------------
    df <- vector %>%
      data.table::data.table(compound = .)
    ## ------------------
    df_row <- df_row %>%
      .[, which(colnames(.) != col)] %>%
      .[rep(1, nrow(df)), ] %>%
      dplyr::bind_cols(df) %>%
      dplyr::select(pathway, Hits.sig, Gamma, compound) %>%
      dplyr::as_tibble()
    return(df_row)
  }
## ----------------------------------------------------------------------
metabo_collate_compound <-
  function(
           compound
           ){
    ## BiGG database download
    if(file.exists("bigg_compound.tsv") == F){
      system("curl http://bigg.ucsd.edu/static/namespace/bigg_models_metabolites.txt > bigg_compound.ts
    }
    bigg <- read_tsv("bigg_compound.tsv", fill = T) %>%
      dplyr::select(universal_bigg_id, name)
    ## ---------------------------------------------------------------------
    part_bigg <- compound[["Matched.Compound"]] %>%
      data.table::data.table(bigg = .) %>%
      merge(bigg, by.x = "bigg", by.y = "universal_bigg_id") %>%
      distinct(bigg, .keep_all = T)
    ## ---------------------------------------------------------------------
    ## main header: universal_bigg_id name
    ## API: KEGGREST::keggGet(vector) ## return a list
    part_kegg <- compound[["Matched.Compound"]] %>%
      .[grepl("^C[0-9]{1,100}$", .)] %>%
      unique() %>%
      data.table::data.table(kegg = .) %>%
      dplyr::mutate(name = batch_kegg_get(kegg))
    ## ---------------------------------------------------------------------
    compound <- compound %>%
      ## merge bigg compound name
      merge(part_bigg, by.x = "Matched.Compound", by.y = "bigg", all.x = T, sort = F) %>%
      ## merge kegg compound name
      merge(part_kegg, by.x = "Matched.Compound", by.y = "kegg", all.x = T, sort = F) %>%
      dplyr::distinct(Query.Mass, Retention.Time, Matched.Compound, .keep_all = T) %>%
      dplyr::mutate(name = ifelse(is.na(name.y), name.x, name.y)) %>%
      dplyr::select(-name.x, -name.y) %>%
      dplyr::as_tibble()
    ## ---------------------------------------------------------------------
    return(compound)
  }
## -------------------------------------
batch_kegg_get <-
  function(
           kegg
           ){
    cat("## kegg compound query\n")
    db <- data.table::data.table(kegg = kegg, seq = 1:length(kegg)) %>%
      dplyr::mutate(index = (seq - seq %% 10) / 10) %>%
      by_group_as_list("index") %>%
      lapply(select, kegg) %>%
      lapply(unlist) %>%
      lapply(unname) %>%
      pbapply::pblapply(.meta_kegg_check_kegg_get) %>%
      unlist() %>%
      unname()
  }
.meta_kegg_check_kegg_get <-
  function(
           id_set
           ){
    db <- KEGGREST::keggGet(id_set) %>%
      lapply(.meta_kegg_check_get_name) %>%
      unlist() %>%
      data.table::data.table(kegg = names(.), compound = unname(.))
    df <- data.table::data.table(kegg = id_set) %>%
      merge(db, by = "kegg", all.x = T, sort = F)
    return(df$compound)
  }
.meta_kegg_check_get_name <-
  function(
           list,
           id = "ENTRY",
           name = "NAME"
           ){
    id <- list[[id]][1]
    compound <- list[[name]][1]
    if(is.null(compound))
      compound <- NA
    names(compound) <- id
    return(compound)
  }
## ---------------------------------------
.meta_sort_uniq_df <-
  function(
           df,
           col,
           pattern_set
           ){
    levels <- df[[col]] %>%
      .meta_find_and_sort(., pattern_set) %>%
      unique()
    df[[col]] <- factor(df[[col]], levels = levels)
    df <- df[order(df[[col]]), ] %>%
      .[!duplicated(.[[col]]), ]
    return(df)
  }
```

25 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"
)
```

26 File: pathway_enrichment.R

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

#' @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)
```

```r
      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")
  }
```

27 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)
```

```r
  }

#' @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
```

```r
#' @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]
}
```

28 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)
      )
    )
  }
```

29 File: png_add_margin.R

```
png_gather_two <-
  function(
           png_file1,
           png_file2 = NA,
           width = 5000,
           height = 3500,
           internal = 0.06
           ){
    ## read png1
    png1 <- png::readPNG(png_file1)
    ratio.1 <- ncol(png1) / nrow(png1)
    ## read png2
    if(!is.na(png_file2)){
      png2 <- png::readPNG(png_file2)
      ratio.2 <- ncol(png2) / nrow(png2)
      ## filename fix
      fix <- "gather_"
      ## png postion shift
      position_shift <- "0"
    }else{
      ratio.2 <- NULL
      ## filename fix
      fix <- "ps_"
      ## png postion shift
      position_shift <- "(unit.width / 2 + internal / 2) / width.adjust"
    }
    ## ------------------
    max <- max(c(ratio.1, ratio.2))
    ## according to height, calculate needed width
    expect.width <- height * max / (1 - internal) * 2
    ## if width not enough
    if(width < expect.width){
      width <- expect.width
      width.adjust <- 1
    }else{
      width.adjust <- expect.width / width
    }
    ## save path and savename
    path <- get_path(png_file1)
    name <- get_filename(png_file1)
    png(paste0(path, "/", fix, name), width = width, height = height)
    plot.new()
    ## x, y, xend, yend
    unit.width <- (0.5 - internal / 2) * width.adjust
    rasterImage(png1,
                xleft = unit.width * (1 - ratio.1 / max) +
                  eval(parse(text = position_shift)),
                ybottom = 0,
                xright = unit.width +
                  eval(parse(text = position_shift)),
                ytop = 1)
    if(!is.na(png_file2)){
      ## x, y, xend, yend
      rasterImage(png2,
                  xleft = unit.width + internal,
                  ybottom = 0,
                  xright = unit.width * (1 + ratio.2 / max) + internal,
                  ytop = 1)
    }
    dev.off()
  }
```

```r
png_to_pdf <-
  function(
           file
           ){
    file.pdf <- gsub("\\.png", "\\.pdf", file)
    pdf(file = file.pdf)
    png <- EBImage::readImage(file)
    EBImage::display(png, method = "raster", all = T)
    dev.off()
  }
```

30 File: pretty_table.R

```r
# ==========================================================================
# format table as .tex .html ...
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @import gt
pretty_table <-
  function(
    df, title = "compounds summary", subtitle = "LC-MS",
    footnote = "...", spanner = F, default = F,
    filename = "tmp.html", path = tempdir(),
    font = "Times New Roman", widths = NULL, caption = NULL)
  {
    if (!is.null(title)) {
      title = paste0("**", Hmisc::capitalize(title), "**")
      subtitle = paste0("**", Hmisc::capitalize(subtitle), "**")
    }
    colnames(df) <- Hmisc::capitalize(colnames(df))
    if(!default){
      t <- gt_solid_line(df, title = title, subtitle = subtitle,
        footnote = footnote, font = font, widths = widths, caption = caption)
    }
    if(default){
      t <- opt_table_font(gt(df), font=list(font))
      t <- tab_header(t, title = md(title), subtitle = md(subtitle))
      t <- opt_align_table_header(t, align = "left")
      t <- tab_footnote(t, footnote = footnote,
        locations = cells_title(groups = c("title")))
    }
    if(spanner){
      columns <- colnames(df) %>%
        .[grepl("#", .)]
      t <- tab_spanner_delim(t, columns = columns,
        delim = "#")
    }
    if (!is.null(filename)) {
      gtsave(t, filename, path)
    }
    return(t)
  }

footnote <- function(gt, text, columns){
  tab_footnote(gt, footnote = text,
    locations = cells_column_labels(columns = !!columns))
}

gt_solid_line <-
  function(df, title = "Table", subtitle = "Table", footnote = "...",
    font = "Times New Roman", widths = NULL, caption = NULL)
  {
    t <- opt_table_font(gt(df, caption = caption), font = list(font))
    if (!is.null(title)) {
      t <- tab_header(t, title = md(title),
        subtitle = if (!is.null(subtitle)) md(subtitle) else NULL)
    }
    if (!is.null(footnote)) {
      t <- tab_footnote(t, footnote = footnote,
        locations = cells_title(groups = c("title")))
    }
    t <- opt_align_table_header(t, align = "left")
    t <- cols_align(t, align = "left",
      columns = everything())
    t <- tab_style(t, style = cell_text(v_align = "top"),
      locations = cells_column_labels(columns = everything()))
    t <- opt_table_lines(t, extent = c("none"))
    t <- tab_style(t, style = cell_borders(sides = c("top", "bottom"),
        color = "black",
        weight = px(1.5),
        style = "solid"),
      locations = cells_column_labels())
    t <- tab_style(t, style = cell_borders(sides = c("bottom"),
        color = "black",
        weight = px(1.5),
        style = "solid"),
      locations = cells_body(columns = everything(),
        rows = eval(parse(text = nrow(df)))))
    t <- tab_style(t, style = cell_text(align = "center",
        weight = "bold"),
      locations = cells_row_groups(groups = everything()))
    t <- tab_style(t, style = cell_borders(sides = c("top", "bottom"),
        color = "grey",
        weight = px(1),
        style = "solid"),
      locations = cells_row_groups(groups = everything()))
    if (!is.null(widths)) {
      t <- cols_width(t, .list = widths)
    }
    return(t)
  }
```

```
# ==========================================================================
# for kable usage
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

kroup_row <- function(data, group = "Group", order = unique(data[[ group ]]))
{
  lst <- split(data.frame(data), data[[ group ]])
  lst <- lapply(lst,
    function(data) {
      data[[ group ]] <- c(data[[ group ]][1], rep("", nrow(data) - 1))
      data
    })
  lst <- lapply(order, function(name) lst[[ name ]])
  tibble::as_tibble(data.table::rbindlist(lst))
}
```

31 File: qi_get_format.R

```
qi_get_format <-
  function(
           file,
           metadata = F
           ){
    df <- data.table::fread(file) %>%
      dplyr::as_tibble()
    ## ------------------------------------
    meta.ori <- dplyr::slice(df, 1:3)
    ## group
    meta.group <- meta.ori[2, ] %>%
      unlist() %>%
      .[which(. != "")] %>%
      .[1:(length(.) / 2 + 1)] %>%
      data.table::data.table(group = ., col = names(.)) %>%
      ## from col to col_end is the sample of group
      dplyr::mutate(col_end = c(col[2:length(col)], NA),
                    col = stringr::str_extract(col, "[0-9]{1,}"),
                    col_end = stringr::str_extract(col_end, "[0-9]{1,}"),
                    col = as.numeric(col),
                    col_end = as.numeric(col_end) - 1) %>%
      dplyr::slice(1:(nrow(.) - 1))
    ## ------------------------------------
    if(metadata){
      meta.sample <- mapply(function(from, to){
                                name <- unlist(meta.ori[3, ], use.names = F)
                                data.table::data.table(sample = name[from:to])
                    }, meta.group$col, meta.group$col_end,
                    SIMPLIFY = F)
      names(meta.sample) <- meta.group$group
      meta.sample <- data.table::rbindlist(meta.sample, idcol = T) %>%
        dplyr::rename(group = .id)
      return(dplyr::as_tibble(meta.sample))
    }
    ## ----------------------------------------------------------------------
    df <- fread(file, skip = 2) %>%
      dplyr::select(grep("^Compound$|m/z|Retention time", colnames(.)),
                    meta.group$col[1]:meta.group$col_end[nrow(meta.group)])
    return(dplyr::as_tibble(df))
  }
## ------------------------------------
qi_as_metabo_inte.table <-
  function(
           df,
           metadata,
           select
           ){
    select.sam <- dplyr::filter(metadata, group %in% all_of(select))
    ## ---------------------------------------------------------------------
    anno <- metadata$group
    names(anno) <- metadata$sample
    ## -------------------------------------
    mz_rt <- dplyr::select(df, 2:3) %>%
      dplyr::rename(mz = 1, rt = 2) %>%
      dplyr::mutate(Sample = paste0(mz, "__", rt)) %>%
      dplyr::select(Sample)
    ## -------------------------------------
    df.data <- df[, 4:ncol(df)] %>%
      dplyr::summarise_all(as.character)
    ## ---------------------------------------------------------------------
    ## bind id
    qi_format.id <- dplyr::bind_rows(c(Sample = "Lable"), mz_rt)
    ## bind data
    qi_format.df <- dplyr::bind_rows(anno, df.data) %>%
      dplyr::select(colnames(.)[colnames(.) %in% select.sam$sample])
    ## -------------------------------------
    qi_format <- dplyr::bind_cols(qi_format.id, qi_format.df)
    return(qi_format)
  }
```

32 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
```

```r
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)
    }
  }
```

33 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))
  }

#' @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)
  }
```

34 File: query_others.R

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

#' @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")
}
```

35 File: query__synonyms.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)) {
```

```r
      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){
```

```r
      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)
  }

#' @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))
  }
```

36 File: set_export.no.R

```r
set_export.no <-
  function(
           df,
           col = "name"
           ){
    tmp <- data.table::data.table(name = unique(df[[col]])) %>%
      dplyr::mutate(., No = 1:nrow(.)) %>%
      dplyr::select(No, name)
    df <- merge(df, tmp, by.x = col, by.y = "name", all.x = T, sort = F) %>%
      dplyr::relocate(No) %>%
      dplyr::as_tibble()
    return(df)
  }
```

37 File: show_png.R

```
show_png <-
  function(
           file,
           size = "500x",
           path = "."
           ){
    if(grepl("\\.svg$", file)){
      tofile <- sub("\\.svg$", ".png", file)
      if(!file.exists(tofile)){
        path <- normalizePath(path)
        system(
               paste0("cairosvg -i ", path, "/", file, " -d 300", " -o ", path, "/", tofile)
        )
      }
      file <- tofile
    }
    png <- magick::image_read(file)
    png <- magick::image_resize(png, size)
    grid::grid.raster(png)
  }
```

38 File: shunt_package.R

```
# ==========================================================================
# Used to shift a series of functions (from files) from one package to another.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

new_package.fromFiles <- function(pkg.path, files, path = NULL,
  depends = c("ggplot2", "grid"),
  exclude = c("MSnbase"),
  ...) {
  imports <- parent_packs(files, path)
  imports <- imports[ !imports %in% exclude ]
  new_package(pkg.path, imports, depends, ...)
  if (!is.null(path))
    files <- paste0(path, "/", files)
```

```r
  file.copy(files, paste0(pkg.path, "/R"), T)
}

new_package <- function(path, imports = NULL, depends = NULL, extdata = T,
  fields = .new_package_fields(),
  gitignore_templ = .gitignore_templ()
  ) {
  pre.wd <- getwd()
  if (!file.exists(path)) {
    usethis::create_package(path, fields)
    usethis::use_mit_license()
    dir.create(paste0(path, "/inst/extdata"), recursive = T)
    file.copy(gitignore_templ, ".")
    writeLines(c(paste0("# ", stringr::str_extract(path, "[^/]*$")),
        "", "Under preparation...", ""), "README.md")
  } else {
    setwd(path)
  }
  cat("Now:", getwd(), "\n")
  if (!is.null(depends)) {
    lapply(depends, usethis::use_package, type = "depends")
  }
  if (!is.null(imports)) {
    lapply(imports, usethis::use_package)
  }
  setwd(pre.wd)
}

parent_packs <- function(files, path = NULL){
  if (!is.null(path))
    files <- paste0(path, "/", files)
  names <- lapply(files,
    function(file){
      file <- readLines(file)
      names <- grep_operater(file)
      names
    })
  unique(unlist(names))
}

grep_operater <- function(txt){
  packs <- stringr::str_extract(txt, "[a-zA-Z][a-zA-Z0-9._]{0,}(?=::)")
  packs <- packs[!vapply(packs, is.na, logical(1))]
  packs <- packs[vapply(packs, requireNamespace, logical(1), quietly = T)]
}

match_function <- function(txt){
  funs <- stringr::str_extract(txt, "[.a-zA-Z][a-zA-Z0-9._]{0,}(?=\\()")
  funs <- funs[!vapply(funs, is.na, logical(1))]
  funs <- unique(funs)
  parent <- lapply(funs, findFunction)
  parent <- parent[vapply(parent, function(p) if (length(p) > 0) T else F, logical(1))]
  parent <- lapply(parent, function(envs) {
    name <- lapply(envs,
      function(env) {
        attr(env, "name")
      })
    gsub("^.*:", "", unlist(name))
    })
  unique(unlist(parent))
}

.new_package_fields <- function() {
  author <- c(person(given = "LiChuang", family = "Huang",
      email = "shaman.yellow@foxmail.com",
      role = c("aut"),
      comment = c(ORCID = "0000-0002-5445-1988")),
    person(given = "Gang", family = "Cao",
      role = c("cre")))
  list(`Authors@R` = author, Author = author,
    Maintainer = "LiChuang Huang <shaman.yellow@foxmail.com>"
  )
}

.gitignore_templ <- function() {
  "~/MCnebula2/.gitignore"
}
```

39 File: simulate_and_evaluate.R

```r
# ==========================================================================
# simulate .mgf from .msp for evaluation
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
## main function
msp_to_mgf <-
  function(
    name,
    id_prefix,
    path = "~/Downloads/msp/MoNA/",
    write_meta_data = paste0(path, "/", name, ".meta.tsv"),
    fun = "mutate_deal_with_msp_record",
    pre_modify = T,
    ...
    ){
    if(pre_modify == T){
      system(paste0("sed -i 's/\r//g' ", path, "/", name))
    }
    msp <- read_msp(paste0(path, "/", name))
    cache <- new.env()
    store <- new.env()
    assign("id", 0, envir = cache)
    mgf <- paste0(path, "/", name, ".mgf")
    assign("envir_meta", environment(), envir = parent.env(environment()))
    cat("", file = mgf)
    ms_fun <- match.fun(fun)
    pbapply::pblapply(msp[[1]], ms_fun,
      id_prefix = id_prefix,
      cache = cache,
      store = store,
      ...)
    set <- ls(envir = store)
    meta_data <- lapply(set, get_envir_df,
      envir = store)
    meta_data <- data.table::rbindlist(meta_data, fill = T)
    if(is.null(write_meta_data) == F){
      write_tsv(meta_data, write_meta_data)
    }
    return(meta_data)
  }
read_msp <-
  function(
    filepath
    ){
```

```r
    msp <- data.table::fread(filepath, sep = NULL, header = F)
  }
get_envir_df <-
  function(
    var,
    envir
    ){
    df <- get(var, envir = envir)
    return(df)
  }

## deal with each line
mutate_deal_with_msp_record <-
  function(
    ...
    ){
    args <- list(...,
      mass_sep = " ",
      input = c(
        name = "Name",
        mass = "PrecursorMZ",
        adduct = "Precursor_type",
        formula = "Formula",
        rt = "NA"),
      other = c(
        "Name", "Synon", "DB#", "InChIKey",
        "Precursor_type", "Spectrum_type", "PrecursorMZ",
        "Instrument_type", "Instrument", "Ion_mode",
        "Collision_energy", "Formula",
        "MW", "ExactMass", "Comments")
    )
    do.call(deal_with_msp_record, args)
  }
deal_with_msp_record <-
  function(
    string,
    id_prefix,
    cache,
    store,
    mass_level = 2,
    set_rt = NA,
    mass_sep = "\t",
    id = get("id", envir = cache),
    input = c(name = "NAME",
      mass = "PRECURSORMZ",
      adduct = "PRECURSORTYPE",
      formula = "FORMULA",
      rt = "RETENTIONTIME"),
    other = c("NAME", "PRECURSORMZ", "PRECURSORTYPE",
      "FORMULA", "Ontology", "INCHIKEY", "SMILES",
      "RETENTIONTIME", "CCS", "IONMODE",
      "INSTRUMENTTYPE","INSTRUMENT",
      "COLLISIONENERGY", "Comment", "Num Peaks"),
    output = c(begin = "BEGIN IONS",
      id = "FEATURE_ID=",
      mass = "PEPMASS=",
      charge = "CHARGE=",
      rt = "RTINSECONDS=",
      level = "MSLEVEL=",
      end = "END IONS"),
    add_scans = F
    ){
    ## get name and value
    name = get_name(string)
    name = ifelse(is.na(name) == T, "", name)
    if(grepl("^[A-Z]", name) == T){
      value = get_value(string)
    }
    cat = 0
    if(name == input[["name"]]){
      catapp(output[["begin"]], "\n")
      ## id update
      id = id + 1
      assign("id", id, envir = cache)
      assign("ion", 1, envir = cache)
      ## output
      cat = 1
      p = output[["id"]]
      s = paste0(id_prefix, id)
      ## new var in envir: store
      info <- data.table::data.table(.id = s, name = value)
      assign(paste0(id), info, envir = store)
    }else if(name == input[["mass"]]){
      cat = 1
      p = output[["mass"]]
      s = value
    }else if(name == input[["adduct"]]){
      cat = 1
      p = output[["charge"]]
      s = ifelse(grepl("]-|]+", value) == F, "0",
        ifelse(grepl("]-", value), "1-", "1+"))
      id <- get("id", envir = cache)
      info = get(paste0(id), envir = store)
      info[["charge"]] = s
      assign(paste0(id), info, envir = store)
      assign("adduct", value, envir = cache)
    }else if(name == input[["formula"]]){
      assign("formula", value, envir = cache)
    }else if(name == input[["rt"]]){
      cat = 1
      p = output[["rt"]]
      if(is.na(set_rt)){
        s = value
      }else{
        s = set_rt
      }
    }else if(name == "Num Peaks"){
      cat = 0
      id <- get("id", envir = cache)
      info = get(paste0(id), envir = store)
      if(mass_level == "all"){
        catapp(output[["level"]], "1\n")
        catapp(info[["PRECURSORMZ"]], "100\n", sep = " ")
        ## here, use rcdk to simulate calculate the isotope pattern
        adduct <- get("adduct", envir = cache)
        if(grepl("FA|ACN", adduct)){
          adduct <- gsub("FA", "CO2H2", adduct)
          adduct <- gsub("ACN", "C2H3N", adduct)
        }
        if(adduct != "[M+H-99]+"){
          formula <- get("formula", envir = cache)
          ## according to adduct to revise formula
          formula <- formula_reshape_with_adduct(formula, adduct)
          ## rcdk function
          array <- rcdk::get.isotopes.pattern(rcdk::get.formula(formula))
          apply(array, 1, cat_isotope)
        }
        catapp(output[["end"]], "\n")
        catapp("\n")
        ## begin mass level 2
        catapp(output[["begin"]], "\n")
        catapp(output[["id"]], info[[".id"]], "\n")
        catapp(output[["mass"]], info[["PRECURSORMZ"]], "\n")
        catapp(output[["charge"]], info[["charge"]], "\n")
      }
      if(!is.na(set_rt))
        info[["RETENTIONTIME"]] = set_rt
      catapp(output[["rt"]], info[["RETENTIONTIME"]], "\n")
      catapp(output[["level"]], "2\n")
      catapp("MERGED_STATS=1 / 1 (0 removed due to low quality, 0 removed due to low cosine)\n")
    }else if(grepl("^[0-9]", string)){
      cat = 2
      p = get_name(string, sep = mass_sep)
      s = get_value(string, sep = mass_sep)
    }else if(string == ""){
      ion <- get("ion", envir = cache)
      if(ion == 0){
        return()
      }
      assign("ion", 0, envir = cache)
      cat = 1
      p = output[["end"]]
      s = "\n"
    }
    if(cat == 1){
      catapp(p, s, "\n")
      if(add_scans == T){
        if(p == output[["mass"]]){
          id <- get("id", envir = cache)
          catapp("SCANS=", id, "\n")
        }
      }
    }else if(cat == 2){
      catapp(p, s, "\n", sep = " ")
    }
    ## data store
    if(name %in% other == T){
      id <- get("id", envir = cache)
      info = get(paste0(id), envir = store)
      info[[name]] = value
      assign(paste0(id), info, envir = store)
    }
    return()
    ## output
  }
catapp <-
  function(
    ...,
    sep = "",
    mgf = get("mgf", envir = get("envir_meta"))
    ){
    cat(paste(..., sep = sep), file = mgf, append = T)
  }
cat_isotope <-
  function(
    vector
    ){
    catapp(vector[1], vector[2] * 100, "\n", sep = " ")
  }
get_value <-
  function(
    string,
    sep = ": "
    ){
    string <- unlist(strsplit(string, split = sep))
    return(string[2])
  }
get_name <-
  function(
    string,
    sep = ": "
    ){
    string <- unlist(strsplit(string, split = sep))
    return(string[1])
  }
```

```r
# ==========================================================================
# Calculate ion mass of formula with adduct
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

formula_adduct_mass <-
  function(
    formula = NA,
    compound_weight = NA,
    get_formula_weight = F,
    iontype = "neg",
    db_adduct = "[M+H]+,[M+K]+,[M+Na]+,[M+H-H2O]+,[M+H-H4O2]+,[M+NH4]+,[M-H]-,[M+Cl]-,[M-H2O-H]-,
    [M+Br]-,[M+FA-H]-,[M+ACN-H]-"
    ){
    welement <- c(H = 1.007825,
      C = 12.0,
      N = 14.003074,
      O = 15.994915,
      F = 18.998403,
      P = 30.973762,
      S = 31.972071,
      Cl = 34.968853,
      Br = 78.918336,
      Na = 22.989770,
      K = 38.963708)
    db_adduct <- unlist(strsplit(db_adduct, split = ","))
    if(iontype == "neg"){
      db_adduct <- db_adduct[grepl("(?<=\\])-$", db_adduct, perl = T)]
    }else{
      db_adduct <- db_adduct[grepl("(?<=\\])\\+$", db_adduct, perl = T)]
    }
    db_adduct <- gsub("FA", "CO2H2", db_adduct)
    db_adduct <- gsub("ACN", "C2H3N", db_adduct)
    cat(paste0("[INFO] use adduct: ", paste(db_adduct, collapse = " | "), " \n\n"))
    ## calculate adduct mass
    adduct_mass <- unlist(lapply(db_adduct, get_adduct_mass))
    if(is.na(compound_weight)){
      compound_weight <- lapply(formula, element_extract)
      compound_weight <- unlist(lapply(compound_weight, element_calculate, welement = welement))
    }
    if(get_formula_weight){
      return(compound_weight)
    }
    list <- lapply(compound_weight, function(x, plus){x + plus}, adduct_mass)
    list <- lapply(list,
      function(mass, adduct){
        data.table::data.table(adduct = adduct, mass = mass)
      }, adduct = db_adduct)
    names(list) <- formula
    return(list)
  }
get_adduct_mass <-
  function(
    adduct
    ){
    welement <- c(H = 1.007825,
      C = 12.0,
      N = 14.003074,
      O = 15.994915,
      F = 18.998403,
      P = 30.973762,
      S = 31.972071,
      Cl = 34.968853,
      Br = 78.918336,
      K = 38.963708,
      Na = 22.989770)
    com <- stringr::str_extract_all(adduct, "(?<=[\\-\\+])[A-Z&a-z&0-9]{1,}(?=[\\-\\+\\]])")
    com <- unlist(com)
    ufunc <- stringr::str_extract_all(adduct, "(?<=[0-9|a-z|A-Z])\\+|-(?=[0-9|a-z|A-Z])")
    ufunc <- unlist(ufunc)
    mass <- lapply(com, element_extract)
    mass <- lapply(mass, element_calculate, welement = welement)
    mass <- unlist(mass)
    df <- data.table::data.table(ufunc = ufunc, mass = mass)
    df <- dplyr::mutate(df, mass = ifelse(ufunc == "+", mass, mass * (-1)))
    sum <- sum(df$mass)
    return(sum)
  }
element_calculate <-
  function(
    df,
    welement
    ){
```

```
    weight <- mapply(function(element, number, welement){
      welement[[element]] * number},
      df$element, df$number,
      MoreArgs = list(welement = welement))
    weight <- sum(weight)
    return(weight)
  }
element_extract <-
  function(
    formula
    ){
    element <- unlist(stringr::str_extract_all(formula, "[A-Z]{1}[a-z]{0,1}"))
    number <- unlist(lapply(paste0("(?<=", element, ")[0-9]{0,}[0-9]{0,}[0-9]{0,}"),
        mutate_extract, db = formula))
    df <- data.table::data.table(element = element, number = number)
    ## if is NA, set as 1
    df <- dplyr::mutate(df, number = ifelse(number == "", 1, as.numeric(number)))
    if(T %in% duplicated(df$element))
      error
    return(df)
  }
mutate_extract <-
  function(
    pattern,
    db
    ){
    ch <- stringr::str_extract(db, pattern)
    return(ch)
  }
get_adduct_df <-
  function(
    adduct
    ){
    com <- stringr::str_extract_all(adduct, "(?<=[\\-\\+])[A-Z&a-z&0-9]{1,}(?=[\\-\\+\\]])")
    com <- unlist(com)
    com <- lapply(com, element_extract)
    ufunc <- stringr::str_extract_all(adduct, "(?<=[0-9|a-z|A-Z])\\+|-(?=[0-9|a-z|A-Z])")
    ufunc <- unlist(ufunc)
    names(com) <- ufunc
    com <- data.table::rbindlist(com, idcol = T)
    return(com)
  }
formula_reshape_with_adduct <-
  function(
    formula,
    adduct,
    order = F
    ){
    adduct <- get_adduct_df(adduct)
    if(nrow(adduct) == 0)
      return(formula)
    df <- element_extract(formula)
    meta <- environment()
    df <- merge(df, adduct, by = "element", all = T)
    df <- dplyr::mutate(df, number = ifelse(is.na(.id), number.x,
        ifelse(.id == "-", number.x - number.y,
          ifelse(is.na(number.x), number.y, number.x + number.y))))
    df <- df[, c("element", "number")]
    df <- dplyr::summarise_at(dplyr::group_by(df, element), "number", sum)
    if(order){
      levels <- c("C", "H", "Cl", "F", "I", "K", "N", "Na", "O", "P", "S")
      levels <- levels[levels %in% df$element]
      df <- dplyr::arrange(df, factor(element, levels = levels))
    }
    df$number <- as.character(df$number)
    ch <- apply(df, 1, paste0)
    ch <- paste(ch, collapse = "")
    return(ch)
    # apply(adduct, 1, base_formula_reshape_with_adduct, envir = meta)
    # return(df)
  }
```

```
# ==========================================================================
# add noise to mgf
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

collate_as_noise_pool <-
  function(
    origin_list,
    valid_list
    ){
    ## filter origin_list
    args <- list(list = origin_list, discard_level1 = T, only_peak_info = T, mass_shift = F)
    ## get mz and intensity
    cat("## Catch main peak information\n")
    origin_list <- do.call(spectrum_add_noise, args)
    ## discard the NULL data
    cat("## Discard empty dataset\n")
    order_list <- origin_list[vapply(origin_list, is.data.frame, logical(1), USE.NAMES = F)]
    ## order the origin_list and valid_list according to .id
    ## first, filter the origin_list, only the .id in valid_list is reserved.
    origin_list <- origin_list[names(origin_list) %in% names(valid_list)]
    ## keep identical
    valid_list <- valid_list[names(valid_list) %in% names(origin_list)]
    ## order
    cat("## Order the lists...\n")
    origin_list <- order_list(origin_list)
    valid_list <- order_list(valid_list)
    origin_list <- lapply(origin_list,
      function(df) {
        df[[ "mass" ]] <- as.double(df[[ "mass" ]])
        df[[ "inte" ]] <- as.double(df[[ "inte" ]])
        df
      })
    cat("## Merge to get noise list\n")
    noise_list <- pbapply::pblapply(names(origin_list),
      function(name) {
        df <- tol_mergeEx(
          origin_list[[ name ]], valid_list[[ name ]],
          main_col = "mass", sub_col = "mz"
        )
        df$rel.int. <- df$inte / max(origin_list[[ name ]]$inte)
        df
      })
    noise_df <- data.table::rbindlist(noise_list, fill = T)
    dplyr::mutate(noise_df, mass = as.numeric(mass), inte = as.numeric(inte))
  }

filter_mgf <-
  function(
           filter_id = prapare_inst_data(.MCn.structure_set)$.id,
           file
           ){
    mgf <- read_msp(file)
    start <- which(mgf$V1 == "BEGIN IONS")
    end <- which(mgf$V1 == "")
    id <- mgf[grepl("FEATURE_ID", mgf$V1), ]
    id <- stringr::str_extract(id, "(?<==).*$")
    list <- pbapply::pbmapply(
      function(start, end, mgf) {
        dplyr::slice(mgf, start:end)
      }, start, end, MoreArgs = list(mgf = mgf), SIMPLIFY = F
    )
    names(list) <- id
    if(is.null(filter_id) == F){
      list <- list[names(list) %in% filter_id]
    }
    return(list)
  }

## mutate function of tol_merge, get the non-merged data
tol_mergeEx <-
  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
```

```r
    df <- merge(main, sub, by = "...id", all.x = T, allow.cartesian = T)
    df$...diff <- abs(df[[ main_col ]] - df[[ sub_col ]])
    df <- dplyr::filter(df, ...diff <= !!tol)
    ## get the non-merged
    df <- backup[!backup$...seq %in% df$...seq, ]
    df$...seq <- NULL
    df
  }

mass_shift <-
  function(
           df,
           merge = T,
           sep = " ",
           int.sigma = 1,
           re.ppm = 1e-6,
           global.sigma = 10/3 * re.ppm,
           indivi.sigma = 10/3 * re.ppm,
           sub.factor = 0.03,
           .noise_pool = noise_pool,
           alpha = 0.2,
           ...
           ){
    df <- dplyr::mutate(df, mass = as.numeric(mass), inte = as.numeric(inte))
    ## intensity variation
    var <- rnorm(nrow(df), 1, int.sigma)
    df <- dplyr::mutate(df, inte = inte * var)
    ## subtract according to max intensity
    df <- dplyr::mutate(df, inte = round(inte - max(inte) * sub.factor, 0))
    ## if intensity less than 0, discard
    df <- dplyr::filter(df, inte > 0)
    ## almost one peak, discard the data
    if(nrow(df) <= 1)
      return()
    ## global shift
    var <- rnorm(1, 0, global.sigma)
    df <- dplyr::mutate(df, mass = mass + mass * var)
    ## individual shift
    var <- rnorm(nrow(df), 0, indivi.sigma)
    df <- dplyr::mutate(df, mass = round(mass + mass * var, 4))
    ## add noise peak
    ## random drawn noise peak from noise pool
    noise <- .noise_pool[sample(1:nrow(.noise_pool), round(alpha * nrow(df))), ]
    ## re-size intensity
    noise <- dplyr::mutate(noise, inte = max(df$inte) * rel.int.)
    ## bind into df
    df <- bind_rows(df, dplyr::select(noise, mass, inte))
    if(merge){
      df <- dplyr::mutate(df, V1 = paste0(mass, sep, inte))
      df <- dplyr::select(df, V1)
    }
    return(df)
  }
```

```
# ==========================================================================
# spectra add noise
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

## for .mgf data
spectrum_add_noise <-
  function(list, cl = NULL, filter_empty = T, ...)
  {
    list <- pbapply::pblapply(list,
      function(df, discard_level1 = F, mass_process_level_1 = F,
        mass_process_level_2 = T, ...)
      {
        mass_level <- df$V1[grepl("MSLEVEL", df$V1)]
        ## process level 1
        if(mass_level == "MSLEVEL=1"){
          if(discard_level1)
            return()
          if(mass_process_level_1)
            df <- mass_process_level_1(df, ...)
          return(df)
          ## process level 2
        }else{
          if(mass_process_level_2)
            df <- mass_process_level_2(df, ...)
          return(df)
        }
      }, cl = cl, ...)
    ## filter the empty spectrum
    if(filter_empty){
```

```r
      list <- list[!vapply(list, is.null, logical(1), USE.NAMES = F)]
    }
    return(list)
  }

discard_level1 <-
  function(list) {
    spectrum_add_noise(
      list = list, mass_process_level_2 = F, discard_level1 = T
    )
  }

mass_process_level_1 <-
  function(df, ...){}

mass_process_level_2 <-
  function(df, mass_shift = T, ...){
    list <- separate_peak_info(df, ...)
    if(!mass_shift)
      return(list)
    list[[2]] <- mass_shift(list[[2]], merge = T, ...)
    if(length(list) == 2)
      return()
    df <- rbindlist(list)
    return(df)
  }

separate_peak_info <-
  function(df, sep = " ", only_peak_info = F, ...)
  {
    peak_row <- grep("^[0-9]", df$V1)
    peak_info <- dplyr::slice(df, peak_row)
    peak_info <- tidyr::separate(peak_info, col = "V1", into = c("mass", "inte"), sep = sep)
    if(only_peak_info == T)
      return(peak_info)
    list <- list(dplyr::slice(df, 1:(min(peak_row) - 1)),
      peak_info,
      dplyr::slice(df, (max(peak_row) + 1):nrow(df))
    )
    return(list)
  }
```

```r
# ==========================================================================
# other
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

mgf_add_anno.gnps <-
  function(df){
    slice_line <- list("1:3", "4:6", "7:nrow(df)")
    list <- lapply(slice_line, function(lines){
      list <- dplyr::slice(df, eval(parse(text = lines)))
      return(list)
    })
    ## scans
    scans <- stringr::str_extract(list[[1]][2, 1], "[0-9]{1,}$")
    scans <- c(V1 = paste0("SCANS=", scans))
    ## merge
    merge <- c(
      V1 = paste0("MERGED_STATS=1 / 1 ",
        "(0 removed due to low quality, 0 removed due to low cosine)"
      )
    )
    dplyr::bind_rows(list[[1]], scans, list[[2]], merge, list[[3]])
  }

simulate_gnps_quant <-
  function(
           meta,
           save_path,
           file = paste0(save_path, "/", "quant.csv"),
           rt = 1000,
           area = 10000,
           id = ".id",
           mz = "PRECURSORMZ",
           simu_id = "row ID",
           simu_mz = "row m/z",
           simu_rt = "row retention time",
           simu_quant = "sample.mzML Peak area",
           return_df = F
           ){
    meta <- dplyr::select(meta, all_of(c(id, mz)))
    meta <- dplyr::mutate(meta, rt = rt, sample = area)
    colnames(meta) <-
```

```r
      mapply_rename_col(
        .find_and_sort_strings(colnames(meta), c(id, mz, "rt", "sample")),
        c(simu_id, simu_mz, simu_rt, simu_quant),
        colnames(meta)
      )
    if(return_df)
      return(meta)
    write.table(meta, file = file, sep = ",", row.names = F, col.names = T, quote = F)
  }
```

```r
# ==========================================================================
# evaluate
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

stat_classify <-
  function(id_set, resClass, id2Key, mcn, ref)
  {
    set <- get_parent_classes(resClass, mcn)
    cat("stat:", resClass, "\n")
    list <- pbapply::pblapply(
      id_set,
      function(id) {
        class <- ref[[ id2Key[[ id ]] ]]
        if (is.null(class)) {
          stat <- data.frame(id = id, evaluate = NA)
          return(stat)
        }
        if (resClass %in% class[["Classification"]]){
          stat <- data.frame(id = id, evaluate = "true")
        } else {
          if (class[3,]$Classification %in% set){
            ## at least the cluster is T in "class" level
            evaluate <- "latent"
          } else {
            evaluate <- "false"
          }
          stat <- data.frame(id = id, evaluate = evaluate)
        }
        return(stat)
      }
    )
    df <- data.table::rbindlist(list)
    return(df)
  }

table_app <-
  function(df, col = "evaluate", prop = T){
    if (!is.data.frame(df))
      return()
    stat <- table(df[[col]])
    if (prop) {
      sum <- sum(stat)
      stat <- prop.table(stat)
    }
    stat <- dplyr::bind_rows(stat)
    stat$sum <- sum
    return(stat)
  }

stat_identification <-
  function(id_lst, id_key2d, ref)
  {
    id_stat <- lapply(id_lst, merge, y = id_key2d, by = ".features_id", all.x = T)
    id_stat <- lapply(id_stat, merge, y = ref, by = ".features_id", all.x = T)
    id_stat <- lapply(id_stat, dplyr::mutate,
      evaluate = ifelse(inchikey2d.x == inchikey2d.y, "true", "false"))
    lst <- lapply(id_stat, table_app)
    df <- data.table::rbindlist(lst, idcol = T, fill = T)
    dplyr::rename(df, class.name = .id)
  }
```

```
# ==========================================================================
# visualize the results of evaluation
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

visualize_stat <-
  function(df, mcn,
           palette = ggsci::pal_npg()(9),
           group_palette = ggsci::pal_rickandmorty()(12),
           ylab = "ratio",
           xlab = "classes",
           fill_lab = "type",
           col_max = 500,
           weight = c(pl = 1, pm = 1.2, pr = 1)
```

```r
          )
{
  parent_class <- unlist(
    lapply(get_parent_classes(df$class.name, mcn),
      function(vec) if (length(vec) == 0) NA else tail(vec, n = 1)),
    use.names = F
  )
  df.back <- df
  df <- dplyr::mutate(
    df, parent_class = ifelse(is.na(parent_class), class.name, !!parent_class)
  )
  annotation <- df
  df <- tidyr::gather(df, "type", "value", -class.name, -parent_class, -sum)
  df <- dplyr::mutate(
    df, class.name = stringr::str_wrap(class.name, width = 25),
    parent_class = stringr::str_wrap(parent_class, width = 25),
    type = as.character(type),
    type = Hmisc::capitalize(type)
  )
  pm <- ggplot(data = df, aes(x = class.name, y = value, fill = type)) +
    geom_col(width = 0.7, position = "stack") +
    scale_fill_manual(values = palette) +
    labs(y = Hmisc::capitalize(ylab),
      x = Hmisc::capitalize(xlab),
      fill = Hmisc::capitalize(fill_lab)
      ) +
    coord_flip()
  pm.theme <- pm.theme2 <- theme(
    axis.text.y = element_blank(),
    text = element_text(family = .font, face = "bold"),
    plot.title = element_text(hjust = 0.3)
  )
  pm.theme$legend.position <- "none"
  pm.theme2$legend.position <- "bottom"
  pm.legend <- MCnebula2:::.get_legend(pm + pm.theme2)
  pm <- pm + pm.theme
  pr <- ggplot(df.back) +
    geom_col(aes(x = class.name,
        y = ifelse(sum >= col_max, col_max, sum)),
      width = 0.7, fill = "#709AE1FF", alpha = 0.7) +
    ylim(0, col_max) +
      labs(x = "", y = "Classified number") +
      coord_flip() +
      theme(axis.text.y = element_blank(),
        axis.ticks = element_blank(),
        text = element_text(family = .font, face = "bold")) +
      geom_blank()
    pl <- ggplot(annotation) +
      geom_col(aes(x = stringr::str_wrap(class.name, width = 25), y = 1,
          fill = stringr::str_wrap(parent_class, width = 25)),
        width = .7) +
      labs(fill = "Super Classes", x = "", y = "Classes") +
      theme_minimal() +
      scale_fill_manual(
        values = colorRampPalette(group_palette)(length(unique(annotation$parent_class)))
        ) +
      theme(
        text = element_text(face = "bold", family = .font),
        axis.text.x = element_text(color = "transparent"),
        axis.ticks.x = element_blank(),
        legend.key.height = unit(1.5, "cm"),
        legend.position = "left",
        panel.grid = element_blank()) +
      coord_flip()
    lst <- lapply(namel(pl, pm, pr), as_grob)
    lst <- frame_col(weight, lst)
    frame_row(c(lst = 10, pm.legend = .5), namel(lst, pm.legend))
  }

visualize_statComplex <-
  function(
    df_list,
    mcn,
    weight = c(pl = 1, pm = 1, pr = 1),
    ylab = "ratio",
    xlab = "classes",
    fill_lab = "type",
    palette = ggsci::pal_npg()(9),
    mutate_palette = c("true" = palette[3], "latent" = palette[2], "false" = palette[1],
      "medium_noise" = "#FED439FF", "high_noise" = "#8A4198FF"),
    extra_palette = c("sum" = "#95CC5EFF",
      "medium_noise" = "#FED439FF", "high_noise" = "#8A4198FF"),
```

```r
  group_palette = ggsci::pal_rickandmorty()(12),
  y_cut_left = c(50, 450),
  y_cut_right = NULL,
  y_cut_left_breaks = c(50, seq(100, 450, by = 100)),
  y_cut_right_breaks = NULL
)
{
  ## get parent class
  df_list.back <- df_list
  df_list <- lapply(df_list,
    function(df){
      parent_class <- unlist(
        lapply(get_parent_classes(df$class.name, mcn),
          function(vec) if (length(vec) == 0) NA else tail(vec, n = 1)),
        use.names = F
      )
      df <- dplyr::mutate(
        df, parent_class = ifelse(is.na(parent_class), class.name, !!parent_class),
        st.true = 0, en.true = true,
        st.latent = en.true, en.latent = st.latent + latent,
        st.false = en.latent, en.false = st.false + false
      )
    })
  ## group draw
  annotation <- df_list[["origin"]]
  pl <- ggplot(annotation) +
    geom_col(aes(x = stringr::str_wrap(class.name, width = 25), y = 1,
        fill = stringr::str_wrap(parent_class, width = 25)),
      width = .7) +
    labs(fill = "Super classes", x = "", y = "Classes") +
    theme_minimal() +
    scale_fill_manual(
      values = colorRampPalette(group_palette)(length(unique(annotation$parent_class)))
      ) +
    theme(
      text = element_text(face = "bold", family = .font),
      axis.text.x = element_text(color = "transparent"),
      axis.ticks.x = element_blank(),
      legend.key.height = unit(1.5, "cm"),
      legend.position = "left",
      panel.grid = element_blank()) +
    coord_flip()
  ## initial stat
  mutate_origin <- tidyr::gather(
    df_list[[ "origin" ]], "type", "value", true, false, latent
  )
  mutate_origin <- dplyr::mutate(
    mutate_origin,
    y = as.numeric(apply(mutate_origin, 1, function(v) v[[paste0("st.", v[["type"]])]] )),
    yend = as.numeric(apply(mutate_origin, 1, function(v) v[[paste0("en.", v[["type"]])]] ))
  )
  ## medium noise dirft
  mergeMutate <-
    function(v1, v2, df_list){
      df <- merge(df_list[[v1]], df_list[[v2]], by = "class.name", all.x = T)
      df <- dplyr::mutate(df, flow1 = "true", flow2 = "latent")
      df <- tidyr::gather(df, "type", "value", flow1, flow2)
      ## calculate segment from y to yend
      dplyr::mutate(
        df, y = as.numeric(apply(df, 1, function(v) v[[paste0("en.", v[["value"]], ".x")]] )),
        yend = as.numeric(apply(df, 1, function(v) v[[paste0("en.", v[["value"]], ".y")]] )),
        exclude = ifelse(is.na(yend), T, F), y = ifelse(is.na(yend), 0, y),
        yend = ifelse(is.na(yend), 1, yend)
      )
    }
  medium_noise_df <- mergeMutate("origin", "medium_noise", df_list)
  medium_noise_df <- dplyr::filter(
    medium_noise_df, y != yend, !exclude, class.name %in% mutate_origin$class.name
  )
  ## high noise drift
  high_noise_df <- mergeMutate("medium_noise", "high_noise", df_list)
  high_noise_df <- dplyr::filter(
    high_noise_df, y != yend, !exclude, class.name %in% mutate_origin$class.name
  )
  pm <- ggplot() +
    ## origin
    geom_segment(
      data = mutate_origin,
      aes(x = class.name, xend = class.name, y = y, yend = yend, color = type),
      size = 7) +
    ## medium noise drift
    geom_segment(
      data = medium_noise_df,
      aes(x = class.name, xend = class.name, y = y, yend = yend, color = "medium_noise"),
      size = 7) +
    ## high noise drift
    geom_segment(
      data = high_noise_df,
      aes(x = class.name, xend = class.name, y = y, yend = yend, color = "high_noise"),
      size = 7) +
    ## the point indicate the start of noise drift
    geom_segment(
      data = medium_noise_df,
      aes(x = class.name, xend = class.name, y = ifelse(yend > y, y - 0.001, y + 0.001),
        yend = y, color = "medium_noise"),
      arrow = arrow(length = unit(10, "pt")), size = 0.5, lineend = "round") +
    ## the point indicate the start of high noise drift
    geom_segment(
      data = high_noise_df,
      aes(x = class.name, xend = class.name, y = ifelse(yend > y, y - 0.001, y + 0.001),
        yend = y, color = "high_noise"),
      arrow = arrow(length = unit(10, "pt")), size = 0.5, lineend = "round") +
    scale_color_manual(
      values = mutate_palette,
      labels = c(sum = "Sum",
        true = "True",
        false = "False",
        latent = "Latent",
        medium_noise = "Medium noise",
        high_noise = "High noise")) +
    labs(
      y = Hmisc::capitalize(ylab),
      x = Hmisc::capitalize(xlab),
      color = Hmisc::capitalize(fill_lab)) +
    coord_flip()
  pm.theme <- theme(
    legend.position = "bottom",
    axis.text.y = element_blank(),
    text = element_text(family = .font, face = "bold"),
    plot.title = element_text(hjust = 0.3)
  )
  pm.legend <- MCnebula2:::.get_legend(pm + pm.theme)
  pm.theme$legend.position <- "none"
```

```r
pm <- pm + pm.theme
extra_list <- lapply(df_list.back, dplyr::select, class.name, sum)
extra.noise_df <- merge(
  extra_list[[ "medium_noise" ]], extra_list[[ "high_noise" ]],
  by = "class.name", all.x = T
)
extra.noise_df <- merge(
  extra.noise_df, extra_list[[ "origin" ]], by = "class.name", all.y = T
)
pr <- ggplot() +
  ## origin sum
  geom_segment(
    data = extra_list[["origin"]],
    aes(x = class.name, xend = class.name, y = 0, yend = sum, color = "sum"),
    size = 7) +
  ## medium_noise drift
  geom_segment(
    data = dplyr::mutate(extra.noise_df, sum.x = ifelse(is.na(sum.x), 0, sum.x)),
    aes(x = class.name, xend = class.name, y = sum, yend = sum.x, color = "medium_noise"),
    size = 7) +
  ## high_noise drift
  geom_segment(
    data = dplyr::mutate(
      dplyr::filter(extra.noise_df, is.na(sum.x) == F),
      sum.x = ifelse(is.na(sum.y), 0, sum.x),
      sum.y = ifelse(is.na(sum.y), sum.x, sum.y)),
    aes(x = class.name, xend = class.name, y = sum.x, yend = sum.y, color = "high_noise"),
    size = 7) +
  scale_color_manual(
    values = extra_palette,
    labels = c(sum = "Sum", medium_noise = "Medium noise", high_noise = "High noise")) +
  geom_blank()
pr.lab <- pr.lab1 <- pr.lab2 <- labs(x = NULL, y = NULL, color = "Type")
pr.lab1$y <- "Classified number"
pr.lab2$y <- "..."
pr.theme <- theme(
  axis.text.y = element_blank(),
  axis.ticks = element_blank(),
  text = element_text(family = .font, face = "bold")
)
pr.legend <- MCnebula2:::.get_legend(pr + pr.theme + pr.lab)
```

```
    pr.theme$legend.position <- "none"
    pr <- pr + pr.theme
    pr1 <- pr + pr.lab1 +
      coord_flip(ylim = y_cut_left) +
      geom_hline(yintercept = c(50), linetype = "dashed", size = 0.7,
        color = "grey") +
      scale_y_continuous(breaks = y_cut_left_breaks)
    pr1 <- as_grob(pr1)
    if (is.null(y_cut_right)) {
      pr2 <- nullGrob()
      pr.w <- c(pr1 = 2, pr.legend = 1)
    } else {
      pr2 <- pr + pr.lab2 +
        coord_flip(ylim = y_cut_right) +
        scale_y_continuous(breaks = y_cut_right_breaks)
      pr2 <- as_grob(pr2)
      pr.w <- c(pr1 = 2, pr2 = 1, pr.legend = 1)
    }
    pr <- frame_col(pr.w, namel(pr1, pr2, pr.legend))
    ## gather all
    pl <- as_grob(pl)
    pm <- as_grob(pm)
    lst <- frame_col(weight, namel(pl, pm, pr))
    frame_row(c(lst = 10, pm.legend = .5), namel(lst, pm.legend))
  }

visualize_comparison <-
  function(
    list1,
    list2,
    ylim_min = 50,
    from = c("MCnebula2", "GNPS"),
    ylab = "Classified number (TP + FP)",
    xlab = "Classes",
    color_lab = "Methods",
    palette = ggsci::pal_npg()(9)
    )
  {
    ## select in common classification
    common.class <- dplyr::select(merge(list1[[1]], list2[[1]], by = "class.name"), 1)
    ## filter classification
    list <- list(list1, list2)
    list <- lapply(list, function(lst){
      lst <- lapply(lst, dplyr::select, class.name, sum)
      lst <- lapply(lst, merge, y = common.class, by = "class.name", all.y = T)
      lst <- lapply(lst, dplyr::mutate, sum = ifelse(is.na(sum), 0, sum))
      df <- data.table::rbindlist(lst, idcol = T)
      dplyr::rename(df, group = .id)
    })
    list <- mapply(list, from, SIMPLIFY = F,
      FUN = function(df, VALUE) {
        dplyr::mutate(df, from = !!VALUE)
      })
    ## for segment
    df <- merge(list[[1]], list[[2]], by = c("group", "class.name"))
    group_levels = c(origin = "Origin", medium_noise = "Medium noise",
      high_noise = "High noise")
    df$group <- vapply(df$group, function(x) group_levels[[ x ]], character(1))
    ## for point
    df2 <- dplyr::bind_rows(list[[1]], list[[2]])
    df2$group <- vapply(df2$group, function(x) group_levels[[ x ]], character(1))
    ## plot figure
    p <- ggplot() +
      geom_segment(
        data = df,
        aes(x = class.name, xend = class.name, y = sum.x, yend = sum.y),
        color = "black") +
      geom_point(
        data = df2,
        aes(x = class.name, y = sum, color = from),
        size = 3, position = "identity") +
      scale_color_manual(values = palette) +
      scale_y_continuous(breaks = c(ylim_min, 300, 600, 900, 1200)) +
      labs(
        y = Hmisc::capitalize(ylab),
        x = Hmisc::capitalize(xlab),
        color = Hmisc::capitalize(color_lab)) +
      coord_flip(ylim = c(ylim_min, max(df2$sum) * 1.1)) +
      facet_wrap(~ factor(group, levels = group_levels)) +
      theme(
        legend.position = "bottom",
        text = element_text(family = .font, face = "bold"),
        plot.title = element_text(hjust = 0.3)) +
      geom_blank()
    grob <- as_grob(p)
    attr(grob, "data") <- list
    return(grob)
  }

visualize_summary <-
  function(summary){
    data <- data.frame(do.call(rbind, summary))
    data <- dplyr::mutate_if(data, is.list, unlist)
    data <- dplyr::mutate(
      data, type = form(rownames(data)),
      type = factor(type, levels = type)
    )
    ## draw plot of classified number
    theme1 <- theme2 <- theme(text = element_text(family = .font))
    theme2$legend.position <- "none"
    p.num <- ggplot(dplyr::mutate(data, type = factor(type, levels = rev(type)))) +
      geom_col(aes(x = type, y = sum, fill = type), width = .5, fill = "#709AE1FF") +
      geom_text(aes(x = type, y = sum, label = round(sum)),
        family = .font, hjust = -1) +
      labs(x = "", y = "Sum") +
      coord_flip(ylim = c(0, 250)) +
      theme_classic() + theme2
    theme3 <- theme1
    theme3$legend.position <- "bottom"
    theme3$panel.spacing <- u(0, line)
    data.ratioFal <- dplyr::mutate(data, `non-false` = 100 - false)
    data.ratioFal <- tidyr::gather(data.ratioFal, evaluate, value, -sum, -type)
    data.ratioSt <- dplyr::mutate(
      data.ratioFal, change = (max(sum) - sum ) / max(sum) * 100,
      evaluate = ifelse(evaluate == "false", "lost", "non-lost"),
      change = ifelse(evaluate == "lost", change, 100 - change),
      change = round(change, 1)
    )
    ## draw plot of stability
    data.ratioSt <- dplyr::filter(data.ratioSt, type != "Origin")
    p.ratioSt <- ggplot(data.ratioSt) +
      geom_col(aes(x = 0, y = change, fill = form(evaluate))) +
      geom_text(data = dplyr::filter(data.ratioSt, evaluate == "non-lost"),
```

```r
      aes(x = -2, y = 0, label = paste0(change, "%")),
      hjust = .5, family = .font) +
    coord_polar(theta = "y", direction = -1) +
    xlim(-2, .5) +
    ylim(0, 100) +
    labs(fill = "") +
    scale_fill_manual(values = c("grey95", "#91D1C2")) +
    facet_wrap(~ type, nrow = 1) +
    theme_void()
  p.ratioSt <- sep_legend(p.ratioSt, theme3)
  ## draw precision (1 - relative false rate)
  lostRate <- mean(dplyr::filter(data.ratioSt, evaluate == "lost")$change) / 100
  data.ratioRelFal <- dplyr::mutate(
    data.ratioFal, rel.value = ifelse(evaluate == "false",
      100 - (100 - value) * (1 - lostRate),
      value * (1 - lostRate)
      ),
    prec.value = 100 - rel.value
  )
  p.precision <- ggplot(data.ratioRelFal) +
    geom_col(aes(x = 0, y = prec.value, fill = form(evaluate))) +
    geom_text(data = dplyr::filter(data.ratioRelFal, evaluate == "false"),
      aes(x = -2, y = 0, label = paste0(round(prec.value, 1), "%")),
      hjust = .5, family = .font) +
    coord_polar(theta = "y", direction = 1) +
    xlim(-2, .5) +
    ylim(0, 100) +
    labs(fill = "") +
    scale_fill_manual(values = c("#FED439FF", "grey95")) +
    facet_wrap(~ type, nrow = 1) +
    theme_void()
  p.precision <- sep_legend(p.precision, theme3)
  ## precision = TP. = 1 - rel.value; FP. = rel.value; FN. = lostRate
  ## recall = TP / (TP + FN)
  data.recall <- dplyr::mutate(
    dplyr::filter(data.ratioRelFal, evaluate == "non-false"),
    recall.value = rel.value * 100 / (rel.value + lostRate * 100)
  )
  fun <- function(data) {
    data2 <- data
    data2$evaluate <- 'false'
```

```
      data2$recall.value <- 100 - data2$recall.value
      rbind(data, data2)
    }
    tt <<- data.recall <- fun(data.recall)
    p.recall <- ggplot(data.recall) +
      geom_col(aes(x = 0, y = recall.value, fill = form(evaluate))) +
      geom_text(data = dplyr::filter(data.recall, evaluate == "non-false"),
        aes(x = -2, y = 0, label = paste0(round(recall.value, 1), "%")),
        hjust = .5, family = .font) +
      coord_polar(theta = "y", direction = -1) +
      xlim(-2, .5) +
      ylim(0, 100) +
      labs(fill = "") +
      scale_fill_manual(values = c("grey95", "#D5E4A2FF")) +
      facet_wrap(~ type, nrow = 1) +
      theme_void()
    p.recall <- sep_legend(p.recall, theme3)
    namel(p.num, p.ratioSt, p.precision, p.recall)
  }

visualize_idRes <-
  function(
    list,
    palette = ggsci::pal_simpsons()(9),
    ylab = "Ratio",
    xlab = "Classes",
    color_lab = "type"
  )
  {
    list.name <- names(list)
    list <- lapply(list, tidyr::gather,
      "type", "value", -class.name
    )
    list <- lapply(list, dplyr::mutate,
      class.name = stringr::str_wrap(class.name, width = 25),
      type = as.character(type),
      type = Hmisc::capitalize(type))
    df <- data.table::rbindlist(list, idcol = T)
    df <- dplyr::filter(df, type == "True")
    line_df <- tidyr::spread(df, .id, value)
    p <- ggplot(data = df, aes(x = class.name, y = value, color = .id)) +
      geom_segment(data = line_df,
        aes(x = class.name, xend = class.name,
          y = eval(parse(text = paste0("`", list.name[1], "`"))),
          yend = eval(parse(text = paste0("`", list.name[2], "`")))),
        color = "black") +
      geom_point(size = 3, position = "identity") +
      scale_color_manual(values = palette) +
      labs(y = Hmisc::capitalize(ylab),
        x = Hmisc::capitalize(xlab),
        color = Hmisc::capitalize(color_lab)) +
      coord_flip() +
      theme(legend.position = "bottom",
        text = element_text(family = .font, face = "bold"),
        plot.title = element_text(hjust = 0.3)) +
      geom_blank()
    as_grob(p)
  }

gtext90 <- function(label, fill, rot = 90) {
  rect_mcnebula <- roundrectGrob(gp = gpar(col = "transparent", fill = fill))
  gtext <- gtext(label, list(cex = 1.2, col = "white"), rot = rot)
  ggather(rect_mcnebula, gtext)
}
```

40 File: stack_ms2.R

```
stack_ms2 <-
  function(
           idset,
           raw.color = "black",
           sig.color = "#E6550DFF",
           width = 13 * 1.5,
           height = 10 * 1.5,
           struc.size.factor = 0.5,
           struc.x = 0.7,
           struc.y = 0.3,
           filename = ifelse(file.exists("mcnebula_results"),
                             paste0("mcnebula_results", "/", "mirror.ms2.svg"), "mirror.ms2.svg"),
           merge_via_int = T
           ){
    if(!require("MCnebula", quietly = T)){
```

```r
  cat("MCnebula not loaded\n")
  return()
}
## ---------------------------------------------------------------------
meta_dir <- method_formula_based_spec_compare(
    target_ids = idset,
    filter_only_max = Inf,
    get_meta_dir = T) %>%
  dplyr::mutate(ms.spec = gsub("spectra/[^/]{1,}$", "spectrum.ms", full.name))
## ---------------------------------------------------------------------
spec.list <- pbapply::pbapply(
   meta_dir, 1, function(vec) {
     ## read sig spectra
     sig.spec <- read_tsv(vec[["full.name"]])
     ## read raw spectra
     raw.spec <- readLines(vec[["ms.spec"]])
     ## mz
     line.mz <- grep("^>parentmass", raw.spec)
     mz <- stringr::str_extract(raw.spec[line.mz], "(?<=\\s)[0-9|\\.]{1,}[ ]{0,}$")
     mz <- round(as.numeric(mz), 4)
     ## rt
     line.rt <- grep("^>rt", raw.spec)
     rt <- stringr::str_extract(raw.spec[line.rt], "(?<=\\s)[0-9|\\.]{1,}(?=s)")
     rt <- round(as.numeric(rt) / 60, 1)
     ## ms2 db
     line.ms2 <- grep("^>ms2peaks", raw.spec)
     ms2 <- paste(raw.spec[(line.ms2 + 1):(length(raw.spec))], collapse = "\n")
     ms2 <- data.table::fread(ms2)
     ms2 <- dplyr::rename(ms2, mass = 1, int = 2)
     ms2 <- dplyr::mutate(ms2, rel.int = int / max(int) * 100)
     ## merge raw with assigned ms2
     ms2.merge <- numeric_round_merge(ms2, sig.spec, main_col = "mass", sub_col = "mz")
     if(merge_via_int)
       ms2.merge <- dplyr::filter(ms2.merge, abs(rel.int - rel.intensity) < 1)
     ms2.merge <- dplyr::bind_rows(ms2.merge, ms2)
     ms2.merge <- dplyr::distinct(ms2.merge, mass, int, .keep_all = T)
     return(list(mz = mz, rt = rt, ms2 = ms2.merge))
   })
## ---------------------------------------------------------------------
ms2.set <- lapply(spec.list, `[[`, 3)
names(ms2.set) <- meta_dir$.id
```

```
  ms2.set <- data.table::rbindlist(ms2.set, idcol = T)
  sig.ms2.set <- dplyr::filter(ms2.set, !is.na(mz))
  ## -------------------------------------
  anno <- lapply(spec.list,
                 function(lst){
                   data.table::data.table(precur.mz = lst[[1]], rt = lst[[2]])
                 }) %>%
  data.table::rbindlist() %>%
  dplyr::mutate(.id = meta_dir$.id,
                anno.mz = paste("Precursor m/z:", precur.mz),
                anno.rt = paste("RT (min):", rt),
                anno = paste0(anno.mz, "\n", anno.rt),
                anno.x = 0, anno.y = 65)
  ## -------------------------------------
  ## add tanimotoSimilarity
  anno <- merge(anno, .MCn.structure_set[, c(".id", "tanimotoSimilarity")],
                by = ".id", all.x = T) %>%
    dplyr::mutate(ts = round(tanimotoSimilarity, 2),
                  anno.ts = paste("TS:", ifelse(is.na(ts), "-", ts)),
                  anno = paste0(anno, "\n", anno.ts))
  ## ----------------------------------------------------------------------
  p <- ggplot() +
## raw mass2 peak
    geom_segment(data = ms2.set,
                 aes(x = mass,
                     xend = mass,
                     y = 0,
                     yend = rel.int),
                 color = raw.color,
                 size = 0.8,
                 alpha = 0.8) +
    ## sig mass2 peak
    geom_segment(data = sig.ms2.set,
                 aes(x = mz,
                     xend = mz,
                     y = 0,
                     yend = -rel.intensity),
                 color = sig.color,
                 size = 0.8) +
    ## match in raw
    geom_point(data = sig.ms2.set,
               aes(x = mass,
                   y = rel.int),
               size = 0.8,
               color = raw.color,
               alpha = 0.8) +
    ## match in sig
    geom_point(data = sig.ms2.set,
               aes(x = mz,
                   y = -rel.intensity),
               size = 0.8,
               color = sig.color) +
    geom_text(data = anno,
              aes(x = anno.x,
                  y = anno.y,
                  label = anno),
              hjust = 0, fontface = "bold", family = "Times") +
    ## theme
    scale_y_continuous(limits = c(-100, 100)) +
    theme_minimal() +
    theme(text = element_text(family = "Times"),
          strip.text = element_text(size = 12),
          panel.grid = element_line(color = "grey85"),
          plot.background = element_rect(fill = "white", size = 0)
          ) +
    facet_wrap(~ paste("ID:", .id), scales = "free")
  ## ---------------------------------------------------------------------
  ## visualize structure
  struc.dir <- "mcnebula_results/tmp/structure"
  if(!file.exists(struc.dir)){
    struc.vis <- T
  }else{
    check <- sapply(paste0(struc.dir, "/", meta_dir$.id, ".svg"), file.exists)
    if(T %in% check){
      struc.vis <- F
    }else{
      struc.vis <- T
    }
  }
  struc.set <- dplyr::filter(.MCn.structure_set, .id %in% meta_dir$.id)
  if(struc.vis){
    cat("## Draw structure via Molconvert and Openbabal\n")
```

```
      vis_via_molconvert(struc.set$smiles, struc.set$.id)
    }
    ## ----------------------------------------------------------------------
    ## draw page
    svg(filename, width = width, height = height)
    cat("[INFO] BEGIN: current.viewport:", paste0(current.viewport()), "\n")
    ## the main picture
    print(p)
    ## structure mapping
    df.vp <- get_facet.wrap.vp(meta_dir$.id)
    apply(df.vp, 1, function(vec){
            struc.path <- paste0(struc.dir, "/", vec[["strip"]], ".svg")
            if(file.exists(struc.path)){
              ## read structure
              svg <- grImport2::readPicture(struc.path)
              ## estimate size of chem.
              lwd <- svg[[1]][[1]]@gp$lwd
              ## to according viewport
              downViewport(paste0(vec[["vp"]]))
              grImport2::grid.picture(svg, width = 0.5 / lwd * struc.size.factor,
                                      height = width, x = struc.x, y = struc.y)
              ## return to ROOT
              upViewport(2)
            }
            })
    cat("[INFO] END: current.viewport:", paste0(current.viewport()), "\n")
    dev.off()
  }
## ----------------------------------------------------------------------
get_facet.wrap.vp <-
  function(
           strip,
           grid.force = T
           ){
    if(grid.force){
      grid::grid.force()
    }
    ## grep vp of panel
    panel <- grid::grid.grep("panel", grep = T, global= T, viewports = T, grobs = F)
    ## vp name
    panel <- sapply(panel, paste)
```

```r
    ## the specific seq number of vp
    panel.seq <- stringr::str_extract(panel, "(?<=panel-)[0-9]{1,}(?=-)")
    panel.seq <- max(as.integer(panel.seq))
    ## number stat
    len <- length(strip)
    len.p <- length(panel)
    ## the number of blank panel
    na <- len.p - (len %% len.p)
    if(na == len.p)
      na <- 0
    ## as matrix
    mat <- matrix(c(sort(strip), rep(NA, na)), ncol = panel.seq, byrow = T)
    vec <- as.vector(mat)
    ## as data.frame
    df <- data.table::data.table(vp = panel, strip = vec)
    ## filter out the NA
    df <- dplyr::filter(df, !is.na(strip))
    return(df)
  }
```

41 File: tmp.ahr.R

```r
set.sig.wd <-
  function(gse)
  {
    path.work <- "~/operation/geo_db/ahr_sig"
    path.de.1 <- "/ftp.ncbi.nlm.nih.gov/geo/series/"
    gse.dir <- gsub("[0-9]{3}$", "nnn", gse)
    wd <- paste0(path.work, path.de.1, gse.dir, "/", gse, "/suppl")
    setwd(wd)
    cat("## The work directory at:", getwd(), "\n")
    print(list.files(all.files = T))
    cat("## Done\n")
  }

set.initial.wd <-
  function(
    wd = "~/operation/geo_db/ahr_sig"
    )
  {
    setwd(wd)
  }
decomp_tar2txt <-
  function(){
    list.files(pattern = "_RAW\\.tar$") %>%
      utils::untar(exdir = ".")
    list.files(pattern = "\\.gz$") %>%
      lapply(R.utils::gunzip)
    file <- list.files(pattern = "^GSM.*txt$")
    df <- data.table::data.table(file = file)
    return(df)
  }

anno.gene.biomart <-
  function(
    db = "hsapiens_gene_ensembl",
    host = NULL,
    attr = NA,
    ex.attr = NA
    )
  {
    if(is.null(host)){
      ensembl <- biomaRt::useEnsembl(biomart = "ensembl", dataset = db)
    }else{
      ensembl <- biomaRt::useEnsembl(biomart = "ensembl", dataset = db, host = host)
    }
    if(!is.character(attr)){
      attr <- c("ensembl_gene_id",
        "hgnc_symbol",
        "chromosome_name",
        "start_position",
        "end_position",
        "description"
      )
    }
    if(is.character(ex.attr))
      attr <- c(attr, ex.attr)
    anno <- biomaRt::getBM(attr, mart = ensembl)
    anno <- dplyr::as_tibble(anno)
    return(anno)
  }
```

```r
anno.gene.edb <-
  function(keys,
    package = "EnsDb.Hsapiens.v86",
    columns = c("GENEID", "SYMBOL", "EXONID",
      "EXONSEQSTART", "EXONSEQEND"),
    keytype = "GENEID",
    ex.attr = NA
    )
  {
    if(is.character(ex.attr))
      columns <- c(columns, ex.attr)
    obj <- paste0(package, "::", package)
    obj <- eval(parse(text = obj))
    genes <- AnnotationDbi::select(obj,
      keytype = keytype,
      keys = keys,
      columns = columns)
    genes <- dplyr::distinct(
      genes, dplyr::across(dplyr::all_of(keytype)), .keep_all = T
    )
  }

anno.into.list <-
  function(dge = dge.list, anno = gene.anno, col = "ensembl_gene_id")
  {
    genes <- data.frame(col = rownames(dge))
    colnames(genes) <- col
    genes <- merge(genes, anno, all.x = T, by = col, sort = F)
    genes <- dplyr::distinct(genes, dplyr::across(dplyr::all_of(col)), .keep_all = T)
    cat("## is.na:\n")
    check.col <- colnames(genes)[2]
    check.na <- dplyr::filter(
      genes, is.na(!!!rlang::syms(check.col))) %>%
      dplyr::as_tibble()
    print(check.na)
    dge$genes <- genes
    return(dge)
  }

re.sample.group <-
  function(dge = dge.list, meta = meta.df)
  {
    colnames(dge) <- meta$sample
    dge$samples$group <- meta$group
    return(dge)
  }
# fpkm2count <-
# function(
#          dge = dge.list,
#          eff.len = "eff.len",
#          lim.min = 0.0001
#          ){
#   num <- nrow(dge$counts)
#   eff.len <- dge$genes[[eff.len]]
#   ## counts...
#   cset <- apply(dge$counts, 2,
#                 function(vec){
#                   vec <- log(vec + lim.min, base = exp(1))
#                   vec <- vec - log(1e9) + log(eff.len)
#                   sum.counts.delog <- sum(vec) / (1 - num)
#                   vec <- vec + sum.counts.delog
#                   round(10 ^ vec)
#                 })
#   dge$counts <- cset
#   return(dge)
# }

fpkm_log2tpm <-
  function(dge = dge.list)
  {
    cset <- apply(dge$counts, 2,
      function(vec){
        log2(
          # vec / sum(vec) * 1e6 + 1
          exp(log(vec) - log(sum(vec, na.rm = T)) + log(1e6)) + 1
        )
      })
    dge$counts <- cset
    return(dge)
  }

get_gsm.data <-
```

```r
  function(info)
  {
    lapply(info, function(obj){
      gsm <- Biobase::phenoData(obj) %>%
        Biobase::sampleNames() %>%
        unlist() %>%
        pbapply::pblapply(function(gsm){
          gsm.dir <- gsub("[0-9]{3}$", "nnn", gsm)
          ftp <- paste0("ftp://ftp.ncbi.nlm.nih.gov/geo/samples/",
            gsm.dir, "/", gsm, "/suppl/")
          system(paste("wget -np -m", ftp))
        })
    })
  }

limma_downstream <-
  function(dge.list, group., design, contr.matrix,
    min.count = 10, voom = T, cut.q = 0.05, cut.fc = 0.3,
    get_ebayes = F, get_normed.exprs = F, block = NULL)
  { # if(NA %in% dge.list$samples[["lib.size"]]){
    # dge.list$samples[["lib.size"]] <- apply(dge.list$counts, 2, sum, na.rm = T)
    # }
    keep.exprs <- edgeR::filterByExpr(dge.list, group = group., min.count = min.count)
    dge.list <- edgeR::`[.DGEList`(dge.list, keep.exprs, , keep.lib.sizes = F)
    if(voom){
      dge.list <- edgeR::calcNormFactors(dge.list, method = "TMM")
      dge.list <- limma::voom(dge.list, design)
    }else{
      genes <- dge.list$genes
      targets <- dge.list$samples
      dge.list <- scale(dge.list$counts, scale = F, center = T)
    }
    if(get_normed.exprs)
      return(dge.list)
    if(!is.null(block)){
      dupcor <- limma::duplicateCorrelation(dge.list, design, block = block)
      cor <- dupcor$consensus.correlation
      cat("## Within-donor correlation:", cor, "\n")
    }else{
      cor <- NULL
    }
    fit <- limma::lmFit(dge.list, design, block = block, correlation = cor)
    if(!voom){
      fit$genes <- genes
      fit$targets <- targets
    }
    fit.cont <- limma::contrasts.fit(fit, contrasts = contr.matrix)
    ebayes <- limma::eBayes(fit.cont)
    if(get_ebayes)
      return(ebayes)
    res <- lapply(1:ncol(contr.matrix), function(coef){
      results <- limma::topTable(ebayes, coef = coef, number = Inf) %>%
        dplyr::filter(adj.P.Val < cut.q, abs(logFC) > cut.fc) %>%
        dplyr::as_tibble()
      return(results)
    })
    names(res) <- colnames(contr.matrix)
    return(res)
  }

limma_downstream.eset <-
  function(
    eset,
    design,
    contr.matrix,
    cut.q = 0.05,
    cut.fc = 0.3
    ){
    fit <- limma::lmFit(eset, design)
    fit.cont <- limma::contrasts.fit(fit, contrasts = contr.matrix)
    ebayes <- limma::eBayes(fit.cont)
    res <- lapply(1:ncol(contr.matrix), function(coef){
      results <- limma::topTable(ebayes, coef = coef, number = Inf) %>%
        dplyr::filter(adj.P.Val < cut.q, abs(logFC) > cut.fc) %>%
        dplyr::as_tibble()
      return(results)
    })
    names(res) <- colnames(contr.matrix)
    return(res)
  }

list.attr.biomart <-
```

```
  function(biomart = "ensembl", dataset = "hsapiens_gene_ensembl")
  {
    attr <- biomaRt::useEnsembl(biomart = biomart,
      dataset = dataset) %>%
    biomaRt::listAttributes(mart = .) %>%
    dplyr::as_tibble()
  return(attr)
  }

show_boxplot <-
  function(df)
  {
    df <- dplyr::as_tibble(df) %>%
      dplyr::mutate(id = rownames(df)) %>%
      reshape2::melt(id.vars = "id", variable.name = "class", value.name = "value")
    p <- ggplot(df) +
      geom_boxplot(aes(x = class, y = value))
    p
  }
```

42 File: try_do.R

```
try_do <-
  function(
           text,
           envir
           ){
    check <- 0
    n <- 0
    while(check == 0){
      n <- n + 1
      check <- try(res <- eval(parse(text = text), envir = envir), silent = T)
      if(class(check)[1] == "try-error"){
        check <- 0
      }else{
        check <- 1
      }
      cat("##", "Try...", n, "\n")
    }
    return(res)
  }
```

43 File: write_thesis.R

```
# ==========================================================================
# tools for fastly write formatting thesis
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

inclu.fig <- function(image, land = F, saveDir = "thesis_fig", dpi = 300,
  scale = if (land) {
    list(width = 10, height = 6.2)
  } else {
    list(width = 6.2, height = 10)
  })
{
  if (!file.exists(saveDir))
    dir.create(saveDir)
  upper <- get_path(image)
  if (is.na(upper))
    upper <- "."
  file <- get_filename(image)
  ## backup for figure
  if (grepl("\\.pdf$", file)) {
    savename <- paste0(saveDir, "/", sub("\\.pdf$", ".png", file))
    if (!file.exists(savename)) {
      pdf_convert(image, filenames = savename, dpi = dpi, pages = 1)
      need_trim <- T
    } else need_trim <- F
  } else {
    savename <- paste0(saveDir, "/", file)
    if (!file.exists(savename)) {
      file.copy(image, savename)
      need_trim <- T
    } else need_trim <- F
  }
  ## trim the border
  if (!need_trim) {
    gc()
    pic_trim(savename)
  }
  ## record image info
  if (is.null(image_info[[ savename ]])) {
    img <- magick::image_read(savename)
    info <- magick::image_info(img)
    magick::image_destroy(img)
    image_info[[ savename ]] <- list(width = info$width, height = info$height)
    assign("image_info", image_info, envir = .env)
  } else {
    info <- image_info[[ savename ]]
  }
  ## set figure display
  ratio <- info$width / info$height
  if ((scale$width / scale$height) >= ratio) {
    ## height as reference
    height <- scale$height
    width <- height * ratio
  } else {
    ## width as reference
    width <- scale$width
    height <- width / ratio
  }
  knitr::opts_current$set(fig.height = height, fig.width = width)
  knitr::include_graphics(savename)
}

prepare.fig <- function(image, saveDir = "thesis_fig", dpi = 300)
{
  if (!file.exists(saveDir))
    dir.create(saveDir)
  upper <- get_path(image)
  if (is.na(upper))
    upper <- "."
  file <- get_filename(image)
  ## backup for figure
  if (grepl("\\.pdf$", file)) {
    savename <- paste0(saveDir, "/", sub("\\.pdf$", ".png", file))
    if (!file.exists(savename)) {
      pdf_convert(image, filenames = savename, dpi = dpi, pages = 1)
      need_trim <- T
    } else need_trim <- F
  } else {
    savename <- paste0(saveDir, "/", file)
    if (!file.exists(savename)) {
      file.copy(image, savename)
      need_trim <- T
```

```r
    } else need_trim <- F
  }
  ## trim the border
  if (need_trim) {
    gc()
    pic_trim(savename)
  }
}

inclu.capt <- function(img, saveDir = "thesis_fig") {
  if (!file.exists(saveDir)) {
    dir.create(saveDir)
  }
  filename <- get_filename(img)
  savename <- paste0(saveDir, "/", filename)
  if (!file.exists(savename)) {
    png(savename, 1000, 1000, res = 150)
    grid::grid.raster(png::readPNG(img))
    dev.off()
  }
  inclu.fig(savename)
}

.env <- topenv(environment())

image_info <- list()

pic_trim <- function(file){
  img <- magick::image_read(file)
  img <- magick::image_trim(img)
  magick::image_write(img, file)
  magick::image_destroy(img)
}

pretty_flex2 <- function(data,
  caption = "This is table", footer = NULL,  bold_header = F,
  weight = sc(colnames(data), rep(1, length(data))), width = 6,
  family = "Times New Roman", e.family = "SimSun", font.size = 10.5,
  caption_style = "Table Caption", form_body = F, form_header = T)
{
  args <- as.list(environment())
  do.call(pretty_flex, args)
}

pretty_flex <- function(data,
  caption = "This is table", footer = "This is footer",  bold_header = F,
  weight = sc(colnames(data), rep(1, length(data))), width = 6,
  family = "Times New Roman", e.family = "SimSun", font.size = 10.5,
  caption_style = "Table Caption", form_body = T, form_header = T)
{
  if (form_body) {
    data <- dplyr::mutate_if(
      data, function(x) ifelse(is.character(x) | is.factor(x), T, F), form
    )
  }
  if (form_header) {
    colnames(data) <- form(colnames(data))
    if (!is.null(names(weight)))
      names(weight) <- form(names(weight))
  }
  weight <- as.list(weight)
  weight <- vapply(colnames(data),
    function(name) {
      if (is.null(weight[[ name ]])) 1 else weight[[ name ]]
    }, double(1), USE.NAMES = F)
  w.width <- weight * (width / sum(weight))
  data <- flextable::flextable(data)
  data <- flextable::valign(data, valign = "top", part = "body")
  if (bold_header) {
    data <- flextable::bold(data, part = "header", bold = TRUE)
  }
  if (!is.null(footer)) {
    if (is.character(footer)) {
      footer <- paste0(footer, collapse = "\n")
    }
    data <- flextable::add_footer_lines(data, footer)
  }
  data <- flextable::width(data, width = w.width)
  data <- flextable::font(
    data, fontname = family, eastasia.family = e.family, part = 'all'
  )
  data <- flextable::fontsize(data, size = font.size, part = 'all')
  data <- flextable::set_caption(data, caption, word_stylename = caption_style)
  data
}

flex_footer <- function(x, props = fp_text(font.size = 10.5, font.family = "SimSun")) {
  flextable::as_chunk(x, props = props)
}

match_fun2 <- function(txt, pattern = "^[0-9_.A-Za-z]*(?= <-)"){
  res <- stringr::str_extract(txt, pattern)
  res <- res[ !vapply(res, is.na, logical(1)) ]
}

match_method2 <- function(txt){
  match_fun2(txt, "(?<=setMethod\\(\")[^\"]*(?=\")")
}

match_class <- function(txt){
  class <- match_fun2(txt, "(?<=setClass\\(\").*(?=\")")
  sapply(class,
    function(x) {
      if (!isVirtualClass(x))
        slotNames(new(x))
    }, simplify = F)
}

as_code_list <- function(names, value = rep("", length(names)), prefix = "c"){
  writeLines(paste0(prefix, "("))
  lapply(1:length(names),
    function(n) {
      end <- if (n == length(names)) "\"" else "\","
      writeLines(paste0("  ", names[[ n ]], " = \"", value[[ n ]], end))
    })
  writeLines(")")
}

as_code_list2 <- function(lst){
  lapply(1:length(lst),
    function(n) {
      writeLines(paste0("## ", names(lst[n])))
      as_code_list(rep(names(lst[n]), length(lst[[ n ]])),
        lst[[ n ]])
      writeLines("")
    })
}

as_pander.flex <- function(obj, page.width = 12) {
  require(pander)
  data <- obj$body$dataset
  caption <- obj$caption$value
  pandoc.table(data, caption)
}

as_kable.flex <- function(obj, page.width = 14, landscape.width = 19) {
  require(kableExtra)
  data <- obj$body$dataset
  if (length(n <- grep("InChIKey", colnames(data))) > 0) {
    data[[ n ]] <- gsub("([A-Z]{7})([A-Z]{1,})", "\\1- \\2", data[[ n ]])
  }
  caption <- obj$caption$value
  kable <- kable(
    data, "latex", booktabs = TRUE,
    longtable = TRUE, caption = caption
  )
  if (ncol(data) > 8) {
    page.width <- landscape.width
  }
  widths <- obj$body$colwidths
  widths <- as.list(
    paste0(widths / sum(widths) * page.width, "cm")
  )
  for (i in 1:length(widths)) {
    kable <- column_spec(kable, i, width = widths[[ i ]])
  }
  kable <- kable_styling(
    kable, latex_options = c("hold_position", "repeat_header"),
    font_size = 10.5
  )
  if (length(footnotes <- obj$footer$dataset[[ 1 ]]) > 0 ) {
    footnotes <- strsplit(footnotes, split = "\n")[[1]]
    kable <- kableExtra::footnote(
      kable, number = footnotes, fixed_small_size = T
    )
  }
  kable
}

as_gt.flex <- function(obj, page.width = 650) {
  require(gt)
  data <- obj$body$dataset
  colnames <- colnames(data)
  widths <- obj$body$colwidths
  widths <- as.list(widths / sum(widths) * page.width)
  widths <- lapply(1:length(widths),
    function(n) {
      eval(parse(text = paste0("`", colnames[n], "`", " ~ ",
            "px(", widths[n], ")")))
    })
  caption <- obj$caption$value
  gt <- pretty_table(
    data, title = NULL, footnote = NULL,
    filename = NULL, caption = md(paste0("**", caption, "**")), widths = widths
  )
  if (length(footnotes <- obj$footer$dataset[[ 1 ]]) > 0 ) {
    if (grepl(":|: ", footnotes)) {
      footnotes <- strsplit(footnotes, split = "\n")[[1]]
      locates <- stringr::str_extract(footnotes, "^.*?(?=:|: )")
      footnotes <- gsub("^.*?[:: ]", "", footnotes)
      for (i in 1:length(footnotes)) {
        gt <- tab_footnote(gt, footnote = footnotes[i],
          locations = cells_column_labels(columns = dplyr::starts_with(locates[i])))
      }
    } else {
      gt <- tab_footnote(gt, footnote = footnotes)
    }
  }
  gt <- tab_options(gt, table.font.size = px(10.5), footnotes.font.size = px(8))
  if (knitr::is_latex_output()) {
    chunk_label <- knitr::opts_current$get("tab.id")
    gt <- as_latex_with_caption(gt, chunk_label, caption)
  }
  gt
}
```

```r
as_latex_with_caption <- function(gt, chunk_label, caption) {
  gt <- gt::as_latex(gt)
  caption <- paste0("\\caption{\\label{tab:", chunk_label, "}", caption, "}\\\\")
  latex <- strsplit(gt[1], split = "\n")[[1]]
  pos <- grep("\\\\begin\\{longtable\\}", latex)
  # pos.end <- grep("\\\\end\\{longtable\\}", latex)
  if (!length(pos) == 1)
    stop("length(pos) != 1")
  # if (!length(pos.end) == 1)
    # stop("length(pos.end) != 1")
  latex[pos] <- paste0(latex[pos], "\n", caption, "\n")
  # latex[pos] <- paste0("\\resizebox{\\linewidth}{!}{\n", latex[pos], "\n", caption, "\n")
  # latex[pos.end] <- paste0(latex[pos.end], "\n", "}")
  latex <- paste(latex, collapse = "\n")
  gt[1] <- latex
  return(gt)
}

thesis_as_latex.word <- function(md, flex_as_kable = "(^flex_.*)") {
  if (length(md) > 1) {
    message("Guess `md` has been read by readLines.")
  } else {
    md <- readLines(md)
    md <- gsub("<!---BLOCK_LANDSCAPE_START--->", "\\\\landstart", md)
    md <- gsub("<!---BLOCK_LANDSCAPE_STOP--->", "\\\\landend", md)
    md <- gsub("<!---BLOCK_TOC--->", "\\\\tableofcontents", md)
    md <- gsub("<!---BLOCK_TOC\\{seq_id: 'fig'\\}--->", "\\\\listoffigures", md)
    md <- gsub("<!---BLOCK_TOC\\{seq_id: 'tab'\\}--->", "\\\\listoftables", md)
    md <- gsub("(\\*\\*.*?[:: ]\\*\\*)", "\\\\noindent\n\\1", md)
    if (!is.null(flex_as_kable)) {
      md <- gsub(flex_as_kable, "as_kable.flex(\\1)", md)
    }
    md <- gsub("^(```\\{r.*)tab.id", "\\1label", md)
    # md <- gsub("tab.id\\s*=\\s*\"(.*?)\"", "\\1", md)
    md <- gsub("&emsp;&emsp;\\s*", "", md)
  }
}

insert_pdf.tex <- function(lines, pos, pdf, set_pagenumber = T) {
  if (set_pagenumber) {
    command <- "\\includepdfset{pagecommand={\\thispagestyle{fancy}}}"
```

```r
    lines[pos] <- paste0(lines[pos], "\n", command)
  }
  pages <- paste0("1-", pdftools::pdf_info(pdf)$pages)
  command <- paste0("\\includepdfmerge{", pdf, ",", pages, "}\n")
  lines[pos] <- paste0(lines[pos], "\n", command)
  return(lines)
}

insert_tocCont.tex <- function(lines, pos, name, style = "section") {
  command <- paste0("\n\\newpage\n", "\\addcontentsline{toc}{", style, "}{", name, "}")
  lines[pos] <- paste0(lines[pos], "\n", command)
  return(lines)
}
```

```r
# ==========================================================================
# insert text
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ftext <- officer::ftext
fp_text <- officer::fp_text
fpar <- officer::fpar
fp_par <- officer::fp_par

st.index <- fp_text(font.size = 14, font.family = "SimHei")
```

```r
# ==========================================================================
# custom render
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

custom_render <- function(file, theme = 'thesis', fix = fix_spell,
  fun_render = rmarkdown::render, heading_mark = "#") {
  ## initialize
  global <- environment()
  clear <- function(ns) {
    lapply(ns, function(n) assign(paste0("n", n), 0, envir = global))
  }
  clear(1:9)
  attr(n1, 'part') <- 0
  ## numbering heading
  md <- fix(readLines(file))
  h.pos <- grep(paste0("^", heading_mark), md)
  chunk.pos <- get_chunkPos(md)
```

```r
h.pos <- h.pos[ !h.pos %in% chunk.pos ]
toc <- lapply(md[h.pos],
  function(ch) {
    level <- length(stringr::str_extract_all(ch, heading_mark)[[ 1 ]])
    fun <- match.fun(paste0("rheader", ifelse(level > 4, 4, level), ".", theme))
    nch <- fun(ch, level = level, global = global, clear = clear)
    if (grepl("@@part", ch)) level <- 0
    list(heading = nch, level = level)
  })
md[h.pos] <- vapply(toc, function(x) x$heading, character(1))
## cross-reference
ids <- stringr::str_extract(md, "(?<=\\{@@id:)[a-zA-Z._0-9]*(?=\\})")
ids.pos <- (1:length(md))[ !is.na(ids) ]
if (length(ids.pos) > 0) {
  ids.relToc <- vapply(ids.pos, FUN.VALUE = character(1),
    function(pos) {
      as_ref.ch(search_toc(pos, toc = toc, toc.pos = h.pos))
    })
  ids.db <- as.list(sc(ids[ !is.na(ids) ], ids.relToc))
  md[ ids.pos ] <- gsub("\\{@@id:[a-zA-Z._0-9]*\\}", "", md[ ids.pos ])
  refs <- stringr::str_extract_all(md, "(?<=\\{@@ref:)[a-zA-Z._0-9]*(?=\\})")
  refs <- unique(unlist(refs))
  refs <- refs[ !is.na(refs) ]
  if (length(refs) > 0) {
    refs.content <- vapply(refs, FUN.VALUE = character(1),
      function(ref) {
        content <- ids.db[[ ref ]]
        if (is.null(content)) {
          stop("The reference id: ", ref, " not found.")
        }
        content
      })
    for (i in 1:length(refs)) {
      md <- gsub(paste0("{@@ref:", refs[i], "}"), refs.content[i], md, fixed = T)
    }
  }
}
## output
filename <- get_filename(file)
filepath <- get_path(file)
writeLines(md, nfile <- paste0(filepath, "/", "_temp_", filename))
```

```r
  if (!is.null(fun_render))
    fun_render(nfile)
}

get_chunkPos <- function(md) {
  chunk.pos <- grep("^```", md)
  unlist(lapply(seq(1, length(chunk.pos), by = 2),
      function(n) {
        chunk.pos[n]:chunk.pos[ n + 1 ]
      }))
}

## chinese \u4e00-\u9fa5
## double [^\x00-\xff]

pch <- function() "[\u4e00-\u9fa5]"

fix_spell <- function(md) {
  chunk.pos <- get_chunkPos(md)
  yaml.pos <- grep("^---", md)
  yaml.pos <- 1:(yaml.pos[2])
  pos <- (1:length(md))
  pos <- pos[ !pos %in% c(chunk.pos, yaml.pos) ]
  md[ pos ] <- gsub("\"",  "\'", md[ pos ])
  md[ pos ] <- fix_quote(md[ pos ])
  md
}

fix_quote <- function(lines, left = " ‘", right = "’ ", extra = ":、,。+ () ") {
  pattern <- paste0("[\u4e00-\u9fa5", extra, "]")
  lines <- gsub(paste0("(?<=", pattern, ")\\s*\'\\s*(?=[a-zA-Z])"), left, lines, perl = T)
  lines <- gsub(paste0("(?<=[a-zA-Z0-9])\\s*\'\\s*(?=", pattern, ")"), right, lines, perl = T)
  lines
}

fix_quote.latex <- function(lines) {
  lines <- gsub(' ‘', '`', lines)
  lines <- gsub('’ ', '\'', lines)
  lines <- gsub("\'([a-zA-Z_.\\(\\)0-9]{1,})\'", "`\\1\'", lines)
  lines <- gsub("`([a-zA-Z_.\\(\\)0-9]{1,})`", "`\\1\'", lines)
  lines
```

```r
}

as_ref.ch <- function(rel.toc, sep = " &gt; ") {
  if (is.list(rel.toc)) {
    rel.toc <- vapply(rel.toc, function(x) x$heading, character(1))
  }
  heading <- gsub("^#*\\s*|\\s*\\{-\\}", "", rel.toc)
  paste0(heading, collapse = sep)
}

search_toc <- function(pos, toc, toc.pos)
{
  previous <- toc[toc.pos <= pos]
  pre.levels <- vapply(previous, function(x) x$level, double(1))
  top.level <- tail(pre.levels, n = 1) + 1
  is.rel <- rev(vapply(rev(pre.levels), FUN.VALUE = logical(1),
    function(level) {
      if (level < top.level) {
        top.level <<- level
        return(T)
      } else F
    }))
  previous[ is.rel ]
}

rheader1.thesis <- function(x, global, clear, ...,
  fun.part = function() {
    x <- sub("# @@part ", paste0("# ", " 第", chn(attr(n, 'part')), " 部分 "), x)
    paste0(x, " {-}")
  },
  fun.normal = function() {
    paste0(sub("# ", paste0("# ", chn(n), "、"), x), "{-}")
  })
{
  if (grepl('\\{-\\}', x)) {
    return(x)
  } else if (grepl('@@part ', x)) {
    n <- 0
    n1 <- get("n1", envir = global)
    attr(n, 'part') <- attr(n1, 'part') + 1
    clear(1:9)
    assign("n1", n, envir = global)
    fun.part()
  } else {
    n <- get("n1", envir = global) + 1
    clear(1:9)
    assign("n1", n, envir = global)
    fun.normal()
  }
}

rheader2.thesis <- function(x, global, clear, ...,
  fun = function() {
    sub("## ", paste0("## ", " (", chn(n), ") "), sub("$", " {-}", x))
  })
{
  n <- get("n2", envir = global) + 1
  clear(2:9)
  assign("n2", n, envir = global)
  fun()
}

rheader3.thesis <- function(x, global, clear, ...,
  fun = function() {
    sub("### ", paste0("### ", n, ". "), sub("$", " {-}", x))
  })
{
  n <- get("n3", envir = global) + 1
  clear(3:9)
  assign("n3", n, envir = global)
  fun()
}

rheader4.thesis <- function(x, global, clear, level = 4, ...) {
  n <- get(paste0("n", level), envir = global) + 1
  clear(level:9)
  assign(paste0("n", level), n, envir = global)
  num <- vapply(3:level, function(n) get(paste0("n", n), envir = global), double(1))
  num <- paste0(num, collapse = ".")
  sub("#* ",
    paste0(paste0(rep("#", level), collapse = ""), " ", num, " "),
    sub("$", " {-}", x))
```

```r
}

chn <- function(n) {
  switch(n,
    "1" = " 一", "2" = " 二", "3" = " 三", "4" = " 四", "5" = " 五",
    "6" = " 六", "7" = " 七", "8" = " 八", "9" = " 九", "10" = " 十"
  )
}
```

```r
# ==========================================================================
# custom docx tools
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

custom_docx_document <- function(...){
  args <- list(...)
  expr <- args$reference_docx
  if (grepl("^`r ", expr)) {
    path <- eval(parse(text = gsub("^`r |`$", "", expr)))
    args$reference_docx <- path
  }
  args <- list(
    reference_docx = args$reference_docx,
    df_print = "tibble",
    tables = list(
      style = "Table", width = 1, topcaption = TRUE, tab.lp = "tab:",
      caption = list(
        style = "Table Caption", pre = " 表", sep = " ", tnd = 0, tns = "-",
        fp_text = structure(list(
          font.size = NA, bold = TRUE, italic = NA,
          underlined = NA, color = NA, font.family = NA, bold.cs = FALSE,
          font.size.cs = NA, vertical.align = 'baseline', shading.color = NA,
          hansi.family = NA, eastasia.family = NA, cs.family = NA,
          lang.val = NA, lang.eastasia = NA, lang.bidi = NA
        ), structure = "fp_text")
      )),
    plots = list(
      style = "Normal", align = "center", fig.lp = "fig:", topcaption = FALSE,
      caption = list(
        style = "Image Caption", pre = " 图", sep = " ", tnd = 0, tns = "-",
        fp_text = structure(list(
          font.size = NA, bold = FALSE, italic = NA,
          underlined = NA, color = NA, font.family = NA, bold.cs = FALSE,
          font.size.cs = NA, vertical.align = 'baseline', shading.color = NA,
          hansi.family = NA, eastasia.family = NA, cs.family = NA,
          lang.val = NA, lang.eastasia = NA, lang.bidi = NA
        ), structure = "fp_text")
        )),
    page_margins = list(
      bottom = 1, top = 1, right = 1.25, left = 1.25, header = 0.5,
      footer = 0.5, gutter = 0.5)
  )
  do.call(officedown::rdocx_document, c(list('bookdown::word_document2'), args))
}

custom_docx_document2 <- function(...){
  args <- list(...)
  expr <- args$reference_docx
  if (grepl("^`r ", expr)) {
    path <- eval(parse(text = gsub("^`r |`$", "", expr)))
    args$reference_docx <- path
  }
  args <- list(
    reference_docx = args$reference_docx,
    df_print = "tibble",
    tables = list(
      style = "Table", width = 1, topcaption = TRUE, tab.lp = "tab:",
      caption = list(
        style = "Table Caption", pre = "Tab. ", sep = " ", tnd = 0, tns = "-",
        fp_text = structure(list(
          font.size = NA, bold = TRUE, italic = NA,
          underlined = NA, color = NA, font.family = NA, bold.cs = FALSE,
          font.size.cs = NA, vertical.align = 'baseline', shading.color = NA,
          hansi.family = NA, eastasia.family = NA, cs.family = NA,
          lang.val = NA, lang.eastasia = NA, lang.bidi = NA
        ), structure = "fp_text")
      )),
    plots = list(
      style = "Normal", align = "center", fig.lp = "fig:", topcaption = FALSE,
      caption = list(
        style = "Image Caption", pre = "Fig. ", sep = " ", tnd = 0, tns = "-",
        fp_text = structure(list(
          font.size = NA, bold = TRUE, italic = NA,
          underlined = NA, color = NA, font.family = NA, bold.cs = FALSE,
```

```r
          font.size.cs = NA, vertical.align = 'baseline', shading.color = NA,
          hansi.family = NA, eastasia.family = NA, cs.family = NA,
          lang.val = NA, lang.eastasia = NA, lang.bidi = NA
        ), structure = "fp_text")
        )),
    page_margins = list(
      bottom = 1, top = 1, right = 1.25, left = 1.25, header = 0.5,
      footer = 0.5, gutter = 0.5)
  )
  do.call(officedown::rdocx_document, c(list('bookdown::word_document2'), args))
}
```

```r
# =========================================================================
# xml tools
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

touch_docSt <- function(file = "templ_zcmu_thesis.docx",
  path = ".", to = paste0(path, "/", ".templ"))
{
  utils::unzip(paste0(path, "/", file), exdir = to)
  style <- paste0(to, "/word/styles.xml")
  attr(style, "dir") <- to
  style
}

update_docSt <- function(st, xml){
  XML::saveXML(xml, st, prefix = '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>\n')
  file <- readLines(st)
  content <- c(file[1], "")
  xml <- paste0(gsub("^\\s*|\\s*$", "", file[-1]), collapse = "")
  pat <- unique(unlist(stringr::str_extract_all(xml, "(?<=\\s)[a-zA-Z]*=")))
  for (i in pat) {
    xml <- gsub(paste0("\\s", i), paste0(" ", "w:", i), xml)
  }
  content[2] <- xml
  writeLines(content, st)
}

pack_docSt <- function(st, file = "temple.docx"){
  if (!file.exists(file)) {
    file.create(file)
  }
  to_file <- normalizePath(file)
  file.remove(file)
  wd <- getwd()
  setwd(attr(st, "dir"))
  utils::zip(to_file, "./")
  setwd(wd)
}

readxml <- function(file){
  xml <- XML::xmlRoot(XML::xmlTreeParse(file))
  writeLines(paste0("XML length:", length(xml)))
  xml
}

ft <- function(xml, target){
  names <- XML::xmlSApply(xml, xmlName)
  unname(which(names == target))
}

sc <- function(name, value) {
  names(value) <- name
  return(value)
}

allnames <- function(xml2) {
  vapply(1:length(xml2), function(n) XML::xmlName(xml2[[ n ]]), character(1))
}

cp_xml.style <- function(xml, target = "Table", src = "Plain Table 1"){
  src.n <- st.search(xml, src)
  target.n <- st.search(xml, target)
  xml[[ target.n ]] <- XML::removeChildren(
    xml[[ target.n ]], kids = allnames(xml[[ target.n ]])[ -1 ]
  )
  xml[[ target.n ]] <- do.call(
    XML::addChildren,
    c(list(xml[[ target.n ]]), XML::xmlChildren(xml[[ src.n ]])[ -1 ])
  )
  return(xml)
}
```

```
st.search <- function(xml, name) {
  unlist(lapply(1:length(xml),
      function(n) {
        if (st.name(xml[[ n ]]) == name) n
      }))
}

st.name<- function(xml2){
  nametag <- xml2[[ "name" ]]
  if (!is.null(nametag)) {
    return(XML::xmlAttrs(nametag)[[ "val" ]])
  }
  return("_NULL_")
}

repl_xml.fontShd <- function(xml,
  param = "fill", as = "ededed", name = show_allTok(xml),
  target = "rPr", ttag = "shd")
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

insert_xml.tabs <-
  function(xml,
    param = c("val", "pos"), as = c("left", 240), name = paste0("heading ", 1:9),
    target = "pPr", ttag = "tabs", namespace = "w",
    insert_ttag = XML::xmlNode("tabs",
      XML::xmlNode("tab", attrs = sc(param, as), namespace = namespace),
      namespace = namespace))
  {
    args <- as.list(environment())
    args$match.arg <- F
    do.call(repl_xml.font, args)
  }

insert_xml.praNum <-
  function(xml,
    param = "val", as = 20, name = paste0("heading ", 1:9),
    target = "pPr", ttag = "numPr", namespace = "w",
    insert_ttag = XML::xmlNode("numPr",
      XML::xmlNode("numId", attrs = sc(param, as), namespace = namespace),
      namespace = namespace))
  {
    args <- as.list(environment())
    args$match.arg <- F
    do.call(repl_xml.font, args)
  }

repl_xml.fontSpace <- function(xml,
  param = "after", as = "0", name = "caption",
  main = "style", target = "pPr", ttag = "spacing", namespace = "w",
  insert_ttag = XML::xmlNode(ttag, attrs = sc(param, as), namespace = namespace),
  force_target = F, insert_param = T)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

repl_xml.fontOutl <- function(xml,
  param = "val", as = 0, name = c(paste0("heading ", 1:9), paste0("toc ", 1:9)),
  main = "style", target = "pPr", ttag = "outlineLvl", namespace = "w",
  insert_ttag = XML::xmlNode(ttag, attrs = sc(param, as), namespace = namespace),
  force_target = F)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

repl_xml.fontAlign <- function(xml,
  param = "val", as = "center", name = "caption",
  main = "style", target = "pPr", ttag = "jc", namespace = "w",
  insert_ttag = XML::xmlNode(ttag, attrs = sc(param, as), namespace = namespace),
  force_target = F)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}
```

```r
repl_xml.fontSz <- function(xml,
  param = "val", as = "18", name = "heading 1",
  main = "style", target = "rPr", ttag = "sz", namespace = "w",
  insert_ttag = XML::xmlNode(ttag, attrs = sc(param, as), namespace = namespace),
  force_target = F)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

repl_xml.fontSzCs <- function(xml,
  param = "val", as = "24", name = c(paste0("heading ", 1:9), paste0("toc ", 1:9)),
  main = "style", target = "rPr", ttag = "szCs", namespace = "w",
  insert_ttag = XML::xmlNode(ttag, attrs = sc(param, as), namespace = namespace),
  force_target = F)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

dele_xml.lang <- function(xml,
  name = "Normal", target = "rPr", ttag = "lang", delete_ttag = T)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

dele_xml.ital <- function(xml,
  name = "caption", target = "rPr", ttag = "i", delete_ttag = T)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

dele_xml.szCs <- function(xml,
  name = c(paste0("heading ", 1:9), paste0("toc ", 1:9)),
  target = "rPr", ttag = "szCs", delete_ttag = T)
```

```
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

dele_xml.qform <- function(xml,
  name = c(paste0("heading ", 1:9), paste0("toc ", 1:9)),
  target = "qFormat", delete_target = T)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

dele_xml.unhide <- function(xml,
  name = c(paste0("heading ", 1:9), paste0("toc ", 1:9)),
  target = "unhideWhenUsed", delete_target = T)
{
  args <- as.list(environment())
  args$match.arg <- F
  do.call(repl_xml.font, args)
}

reset_xml.fontCol <- function(xml,
  param = "val", as = "000000",
  ttag = "color", name = "TOC Heading",
  remove_ttagAttr = T, ...)
{
  args <- as.list(environment())
  args$match.arg <- F
  args <- c(args, list(...))
  do.call(repl_xml.font, args)
}

repl_xml.font <- function(xml,
  param = c("eastAsia", "ascii", "hAnsi", "cs"),
  as = c("SimHei", "SimSun", "Times New Roman"),
  name = show_allName(xml),
  main = "style", target = "rPr", ttag = "rFonts", namespace = "w",
  match.arg = T, delete_target = F,
```

```r
  insert_ttag = NULL, force_target = F, delete_ttag = F, insert_param = F,
  remove_ttagAttr = F)
{
  if (match.arg) {
    param <- match.arg(param)
    as <- match.arg(as)
  }
  xml[1:length(xml)] <- XML::xmlSApply(xml,
    function(xml2) {
      if (XML::xmlName(xml2) == main) {
        xml2.name <- XML::xmlAttrs(xml2[[ "name" ]])[[ "val" ]]
        if (any(xml2.name == name)) {
          if (delete_target) {
            if (!is.null(xml2[[ target ]]))
              xml2 <- XML::removeChildren(xml2, target)
            return(xml2)
          }
          tag <- xml2[[ target ]][[ ttag ]]
          if (!is.null(tag)) {
            if (delete_ttag) {
              if (!is.null(xml2[[ target ]])) {
                xml2[[ target ]] <- XML::removeChildren(xml2[[ target ]], ttag)
              }
              return(xml2)
            } else if (remove_ttagAttr) {
              xml2[[ target ]][[ ttag ]] <- XML::removeAttributes(xml2[[ target ]][[ ttag ]])
            }
            tag.attr <- XML::xmlAttrs(tag)
            if (any(param == names(tag.attr)) | insert_param) {
              XML::xmlAttrs(xml2[[ target ]][[ ttag ]])[[ param ]] <- as
            }
          } else if (!is.null(insert_ttag)) {
            if (is.null(xml2[[ target ]]) & force_target) {
              node <- XML::xmlNode(target, namespace = namespace)
              xml2 <- XML::append.xmlNode(xml2, node)
            }
            if (!is.null(xml2[[ target ]])) {
              xml2[[ target ]] <- XML::append.xmlNode(xml2[[ target ]], insert_ttag)
            }
          }
        }
      }
      return(xml2)
    })
  return(xml)
}

rm_xml.lsdAttr <- function(xml2,
  name = c(paste0("heading ", 1:9), paste0("toc ", 1:9)), attrs = "qFormat")
{
  xml2[1:length(xml2)] <- XML::xmlSApply(xml2,
    function(xml3) {
      if (any(XML::xmlAttrs(xml3)[[ "name" ]] == name)) {
        vals <- XML::xmlAttrs(xml3)
        vals <- vals[ names(vals) != attrs ]
        xml3 <- XML::removeAttributes(xml3)
        XML::xmlAttrs(xml3) <- vals
      }
      return(xml3)
    })
  return(xml2)
}

show_allName <- function(xml){
  obj <- XML::xmlSApply(xml,
    function(x) {
      x.name <- x[[ "name" ]]
      if (!is.null(x.name)) {
        if (!is.null(x[[ "rPr" ]]) | !is.null(x[[ "pPr" ]]))
          XML::xmlAttrs(x.name)[[ "val" ]]
      }
    })
  unlist(obj)
}

list_allName <- function(xml){
  lapply(1:length(xml),
    function(n) {
      xml[[n]][[ "name"]]
    })
}
```

```r
show_allTok <- function(xml){
  all <- show_allName(xml)
  c(all[ grep("Tok$", all) ])
}

.xmlFont <- XML::xmlNode("rFonts",
  attrs = c(ascii = "Times New Roman",
    hAnsi = "Times New Roman",
    eastAsia = "SimSun",
    cs = "Nimbus Roman"), namespace = "w")

all_tableStyle <- function(xml){
  lst <- lapply(1:length(xml),
    function(n) {
      attr <- XML::xmlAttrs(xml[[n]])
      if (!is.null(attr)) {
        if (any(names(attr) == "type")) {
          if (attr[[ "type" ]] == "table")
            XML::xmlAttrs(xml[[ n ]][[ "name" ]])[[ "val" ]]
        }
      }
    })
  unlist(lst)
}
```

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

mutate_get_fun <- function (x) {
  if (is.function(x)) {
    return(x)
  }
  if (grepl("::", x, fixed = TRUE)) {
    coupounds <- strsplit(x, split = "::", x, fixed = TRUE)[[1]]
    z <- getFromNamespace(coupounds[2], ns = coupounds[1])
  } else {
    z <- getAnywhere(x)
    if (length(z$objs) < 1) {
      stop("could not find any function named ", shQuote(z$name),
        " in loaded namespaces or in the search path. If the package is installed, specify name w
        ith `packagename::function_name`.")
    }
  }
  z
}

pdf_convert <- function (pdf, format = "png", pages = NULL, filenames = NULL,
    dpi = 72, antialias = TRUE, opw = "", upw = "", verbose = TRUE)
{
  config <- pdftools::poppler_config()
  if (!config$can_render || !length(config$supported_image_formats))
    stop("You version of libppoppler does not support rendering")
  format <- match.arg(format, config$supported_image_formats)
  if (is.null(pages))
    pages <- seq_len(pdftools::pdf_info(pdf, opw = opw, upw = upw)$pages)
  if (!is.numeric(pages) || !length(pages))
    stop("Argument 'pages' must be a one-indexed vector of page numbers")
  if (length(filenames) < 2) {
    input <- ifelse(is.raw(pdf), "output", sub(".pdf", "",
        basename(pdf), fixed = TRUE))
  }
  if (is.null(filenames)) {
    filenames <- sprintf("%s_%d.%s", input, pages, format)
  }
  if (length(filenames) != length(pages))
    stop("Length of 'filenames' must be one or equal to 'pages'")
  antialiasing <- isTRUE(antialias) || isTRUE(antialias ==
    "draw")
  text_antialiasing <- isTRUE(antialias) || isTRUE(antialias ==
    "text")
  pdftools:::poppler_convert(pdftools:::loadfile(pdf), format, pages, filenames,
    dpi, opw, upw, antialiasing, text_antialiasing, verbose)
}
```

```r
# ==========================================================================
# other tools
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

as_df.lst <- function(lst, col.name = 'type', col.value = 'name') {
  data <- data.frame(
    x = rep(names(lst), lengths(lst), each = T),
    y = unlist(lst, use.names = F)
  )
  colnames(data) <- c(col.name, col.value)
  data
}

cutWords <- function(ch, len = 15, len.post = 3, max = len + len.post + 1) {
  while( any(grepl(paste0("[^ ^\\-]{", max, "}"), ch)) ) {
    ch <- gsub(paste0("\\b([^ ^\\-]{", len, "})([^ ^\\-]{", len.post, ",})\\b"), "\\1-\\2", ch)
  }
  ch
}

formatBib <- function(bib = paste0(.expath, "/library.bib"), savename = "tmp.bib",
  journalAbb_file = paste0(.expath, "/endlib.txt"))
{
  endlib <- tibble::as_tibble(data.table::fread(journalAbb_file, header = F))
  dics <- .as_dic(endlib[[2]], endlib[[1]])
  names(dics) <- tolower(names(dics))
  lst <- read_bib(bib)
  lst <- lapply(lst,
    function(ch) {
      pos <- grepl("^\\s*journal =", ch)
      if (any(pos)) {
        journal <- stringr::str_extract(ch[pos], "(?<=\\{).*(?=\\})")
        journal <- tolower(journal)
        if (journal %in% names(dics)) {
          abb <- dics[[ journal ]]
          ch[pos] <- sub("\\{.*\\}", paste0("{", abb, "}"), ch[pos])
        } else {
          message("No journal abbrev: ", journal)
        }
      }
      pos <- grepl("^\\s*doi =", ch)
      ch[ !pos ]
    })
  writeLines(unlist(lst), savename)
  return(savename)
}
```