R codes of extra

Contents

1 File: as_pca_df.R

```r
as_pca_df <-
  function(
           df
           ){
    rownames(df) <- df[[1]]
    df <- df %>%
      t()
    df <- df[-1,]
    rownames(df) <- rownames(df) %>%
      gsub(" Peak area", "", .)
    return(df)
  }
adjust_extremity <-
  function(
           df
           ){
```

```
    df <- dplyr::summarise_all(df, add_1)
    return(df)
  }
add_1 <-
  function(
           x
           ){
    x <- x + 1
    return(x)
  }
```

2 File: auto_classy.R

```
gather_classyfire <-
  function(
           path = "classyfire",
           class = "Indoles and derivatives",
           inchi_df = NA
           ){
    file_set <- list.files(path, pattern = "^[0-9]{1,100}$", full.names = T)
    list <- file_set %>%
      lapply(read_tsv)
    names(list) <- file_set %>%
      stringr::str_extract("(?<=/)[0-9]{1,100}$")
    ## ------------------------------------
    if(is.na(class) == F){
      list <- list %>%
        data.table::rbindlist(idcol = T) %>%
        dplyr::filter(Classification == dplyr::all_of(class)) %>%
        dplyr::select(.id, Classification) %>%
        dplyr::rename(classification = Classification)
    }
    ## ------------------------------------
    if(is.data.frame(inchi_df)){
      list <- inchi_df %>%
        dplyr::rename(inchikey = InChIKey) %>%
        dplyr::mutate(inchi2d = stringr::str_extract(inchikey, "^[A-Z]{1,1000}")) %>%
        merge(list, by = ".id", all.y = T) %>%
        dplyr::distinct(inchi2d, .keep_all = T) %>%
        dplyr::select(inchi2d, classification)
    }
    return(list)
  }
mutate_auto_classy <-
  function(
           df,
           path = "classyfire",
           ...
           ){
    if(file.exists(path) == F)
      dir.create(path)
    origin = getwd()
    setwd(path)
    auto_classy(df, ...)
    setwd(origin)
  }
auto_classy <-
  function(
           df,
           ...
           ){
    ## classyfireR
    list <- by_group_as_list(df, ".id")
    pbapply::pblapply(list, base_auto_classy,
                      ...)
  }
base_auto_classy <-
  function(
           df
           ){
    .id <- df[1,][[".id"]]
    lapply(df[["InChIKey"]], base2_classy,
                      .id = .id)
  }
base2_classy <-
  function(
           inchi,
           .id
           ){
    ch <- try(read_tsv(paste0(.id)), silent = T)
    if(class(ch)[1] == "try-error"){
      ch <- classyfireR::get_classification(inchi)
    }else{
      return()
    }
    if(is.null(ch)){
      return()
    }else{
      ch <- classyfireR::classification(ch)
      write_tsv(ch, paste0(.id))
    }
  }
```

3 File: by_group_as_list.R

```
by_group_as_list <-
  function(
           df,
           colnames
           ){
    vector <- unique(df[[colnames]])
    list <- lapply(vector, by_group_as_list_select,
                   df = df,
                   colNames = colnames)
    names(list) <- vector
    return(list)
  }
by_group_as_list_select <-
  function(
           KEY,
           df,
           colNames
           ){
    df <- df[which(df[[colNames]] == KEY), ]
    return(df)
  }
```

4 File: collate_as_noise_pool.R

```
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")
    origin_list <- pbapply::pblapply(origin_list, is.data.frame) %>%
      unlist(use.names = F) %>%
      origin_list[.]
    ## ----------------------------------------------------------------------
    ## 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)
    ## ----------------------------------------------------------------------
    ## list merge (use mapply)
    cat("## Merge to get noise list\n")
    noise_list <- pbapply::pbmapply(numeric_round_merge, origin_list, valid_list,
      main_col = "mass", sub_col = "mz",
      mz.tol = 0.002, noise = T, SIMPLIFY = F)
    ## -------------------------------------
    noise_df <- data.table::rbindlist(noise_list, fill = T)
    ## -------------------------------------
    noise_df <- mutate(noise_df, mass = as.numeric(mass), inte = as.numeric(inte))
    ## -------------------------------------
    return(noise_df)
    ## ----------------------------------------------------------------------
  }
load_all_valid_spectra <-
  function(
    formula_adduct = .MCn.formula_set,
    path = .MCn.sirius
    ){
    cat("Collate as metadata\n")
    metadata <- list.files(path = path, pattern = "^[0-9]{1,}_(.*)_(.*)[0-9]{1,}$") %>%
      data.table::data.table(dir = .) %>%
      dplyr::mutate(.id = stringr::str_extract(dir, "(?<=_)[^_]{1,}$")) %>%
      merge(formula_adduct, by = ".id", all.x = T) %>%
      dplyr::select(.id, dir, precursorFormula, adduct) %>%
      dplyr::mutate(adduct = gsub(" ", "", adduct),
        file = paste0(path, "/", dir, "/spectra/", precursorFormula, "_", adduct, ".tsv"),
        exists = unlist(pbapply::pblapply(file, file.exists), use.names = F)) %>%
      dplyr::filter(exists == T)
    ## ------------------------------------
    cat("Read file and collate\n")
    list <- pbapply::pblapply(metadata$file, read_tsv) %>%
      pbapply::pblapply(dplyr::select, mz, rel.intensity)
    ## ------------------------------------
    names(list) <- metadata$.id
    return(list)
  }
```

5 File: collate_comment.R

```
collate_comment <-
  function(
           vector,
           names
           ){
    list <- pbapply::pblapply(vector, base_collate_comment)
    names(list) <- names
    return(list)
  }
base_collate_comment <-
  function(
           string
           ){
    ch <- strsplit(string, split = "\" \"")
    ch <- unlist(ch)
    ch <- gsub("\"", "", ch)
    ch <- sub("=", "###", ch)
    ch <- strsplit(ch, split = "###")
    ch <- lapply(ch, function(str){
```

```r
            df <- data.table::data.table(comment = str[1], record = str[2])
            return(df)
        })
    ch <- data.table::rbindlist(ch)
    return(ch)
  }
```

6 File: collate_structure_table.R

```r
collate_structure_table <-
  function(
           file = "method_pick_formula_excellent.structure.tsv",
           class = "../canopus_summary.tsv",
           cut_tanimoto = 0.4,
           delete_null = T,
           ...
           ){
    order = c(".id", "name", "molecularFormula", "adduct",
              "most specific class", "inchikey2D", "tanimotoSimilarity")
    df <- read_tsv(file)
    df <- dplyr::filter(df, tanimotoSimilarity >= cut_tanimoto) %>%
      dplyr::select(.id, name, tanimotoSimilarity, molecularFormula, inchikey2D) %>%
      dplyr::mutate(.id = as.character(.id))
    class <- read_tsv(class) %>%
      dplyr::select(name, `most specific class`, adduct) %>%
      dplyr::mutate(.id = stringr::str_extract(name, "(?<=_)[0-9]{1,4}$")) %>%
      dplyr::select(.id, `most specific class`, adduct)
    df <- merge(df, class, by = ".id", all.x = T, sort = T) %>%
      dplyr::select(all_of(order)) %>%
      dplyr::distinct(name, .keep_all = T)
    if(delete_null == T)
      df <- dplyr::filter(df, name != "null")
    write_tsv(df, "table_of_pdf.tsv")
    return(df)
  }
```

7 File: collate_top_score_structure.R

```r
collate_top_score_structure <-
  function(
           path = "."
           ){
    file_set <- list.files(path) %>%
      stringr::str_extract(pattern = "(?<=_)[a-z]{1,50}[0-9]{1,50}$") %>%
      sort()
    id_set <- pbapply::pblapply(file_set, grep_id) %>%
      unlist()
    structure_set <- pbapply::pblapply(id_set, base_collate_top)
    names(structure_set) = id_set
    structure_set <- data.table::rbindlist(structure_set, idcol = T, fill = T)
    return(structure_set)
  }
base_collate_top <-
  function(
           key_id
           ){
    check <- try(df <- get_structure(key_id, return_row = 1), silent = T)
    if(class(check)[1] == "try-error"){
      df <- data.frame()
    }
    return(df)
  }
```

8 File: compound_align.R

```r
compound_align <-
  function(
           main,
           sub,
           main_col = "mz",
           mz_tol = 0.002,
           rt_tol = 0.1,
           mz_wight = 75,
           rt_wight = 25
           ){
    list <- lapply(main[[main_col]], mz_align,
                   df = sub, mz_tol = mz_tol)
    if(is.na(rt_tol))
      return()
  }
rt_align <-
```

```
  function(
           the_rt,
           df,
           rt_tol = 0.1
           ){
    df <- dplyr::filter(df, rt <= the_rt + rt_tol &
                        rt >= the_rt - rt_tol)
    return(df)
  }
mz_align <-
  function(
           the_mz,
           df,
           mz_tol = 0.002
           ){
    df <- dplyr::filter(df, mz <= the_mz + mz_tol &
                        mz >= the_mz - mz_tol)
    return(df)
  }
```

9 File: deal_with_msp_record.R

```
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")
    )
```

```r
    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 <- get.isotopes.pattern(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])
  }
```

10 File: filter_via_rownames.R

```
filter_via_rownames <-
  function(
           df,
           row
           ){
    df <- df[rownames(df) %in% row, ]
    return(df)
  }
```

11 File: flow_collate_skeleton.R

```r
flow_collate_skeleton <-
  function(
           classes,
           ...
           ){
    lapply(classes, base_flow, ...)
  }
base_flow <-
  function(
           class,
           savepath = "~/extra/data"
           ){
    cat("[INFO] the class is >>>", paste0(class), "\n")
    cat("Enter the skeleton smiles:\n")
    smiles <- scan(n = 1, what = "character")
    if(length(smiles) == 1){
      write.table(smiles, file = paste0(savepath, "/", class),
                  col.names = F, row.names = F, quote = F)
    }
  }
```

12 File: generic_horizon_bar.R

```r
generic_horizon_bar <-
  function(
           df,
           scale_fill_expression = "scale_fill_gradient(high = '#E6550DFF', low = '#FDD0A2FF')",
           x = colnames(df)[1],
           y = colnames(df)[2],
           ylab = "stat",
           xlab = "type",
           position = "identity",
           save = "tmp.svg"
           ){
    df[[x]] <- stringr::str_wrap(df[[x]], width = 35)
    df[[x]] <- factor(df[[x]], levels = df[[x]][order(df[[y]], decreasing = F)])
    p <- ggplot(data = df,
                aes(x = eval(parse(text = x)),
                    y = eval(parse(text = y)),
                    fill = eval(parse(text = y)))) +
      geom_col(width = 0.7, position = position) +
      eval(parse(text = scale_fill_expression)) +
      labs(y = Hmisc::capitalize(ylab),
           x = Hmisc::capitalize(xlab)) +
      coord_flip() +
      theme(legend.position = "none",
            text = element_text(size = 16, family = "Times", face = "bold"),
            axis.text = element_text(size = 10),
            plot.title = element_text(hjust = 0.3))
    ggsave(p, file = save, width = 6, height = 12)
  }
```

13 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)
  }
```

14 File: get_reference_class_density.R

```r
get_reference_class_density <-
  function(
           path = "classyfire"
           ){
    distribution <- show_distribution(path = path)
    ## metadata show all classification
    meta_distribution <- distribution %>%
      dplyr::distinct(Level, Classification) %>%
      dplyr::rename(level = Level, classification = Classification)
    ## classes distribution density table
    table_distribution <- table(distribution$Classification) %>%
      data.table(classification = names(.), sum = unname(.)) %>%
      dplyr::select(classification, sum.N)
    ## merge meta and density stat
    reference_density <- merge(meta_distribution, table_distribution,
                               by = "classification", all.x = T) %>%
      dplyr::filter(is.na(level) == F)
    return(reference_density)
  }
get_reference_class_parent <-
  function(
           df,
           min_possess = 50
           ){
    test <- df %>%
      filter(sum.N >= min_possess, !level %in% c("superclass", "kingdom"))
    p_test <- mutate_get_parent_class(test$class, this_class = T) %>%
      lapply(end_of_vector) %>%
      unlist() %>%
      unname() %>%
      unique()
    return(p_test)
  }
```

15 File: horizon_bar_accuracy.R

```r
horizon_bar_accuracy <-
  function(
           df,
           title,
           savename,
           palette = ggsci::pal_npg()(9),
           ylab = "stat ratio",
           xlab = "classification",
           fill_lab = "type",
           extra_sides_df = NA,
           return_p = T
           ){
    df <- reshape2::melt(df, id.vars = "classification",
                         variable.name = "type",
                         value.name = "value")
    df <- dplyr::mutate(df, classification = stringr::str_wrap(classification, width = 25),
                        type = as.character(type),
                        type = Hmisc::capitalize(type))
    ## -----------------------------------------------------------------------
    p <- ggplot(data = df,
                aes(x = classification,
                    y = value,
                    fill = type)) +
      geom_col(width = 0.7,
               position = "stack") +
      scale_fill_manual(values = palette) +
      labs(title = Hmisc::capitalize(title),
           y = Hmisc::capitalize(ylab),
           x = Hmisc::capitalize(xlab),
           fill = Hmisc::capitalize(fill_lab)) +
      coord_flip() +
      theme(legend.position = "bottom",
            text = element_text(family = "Times", size = 20, face = "bold"),
            plot.title = element_text(hjust = 0.3))
    ## -----------------------------------------------------------------------
    if(!is.na(extra_sides_df)){
      max = 500
      ps <- ggplot(data = extra_sides_df) +
        geom_col(width = 0.7, aes(x = classification, y = ifelse(sum >= max, max, sum))) +
        coord_flip() +
        ylim(0, max) +
        theme(axis.text.y = element_blank(),
              text = element_text(family = "Times", size = 20, face = "bold"))
      ## -------------------------------------
      svg(savename, width = 14, height = 15)
      grid.newpage()
      pushViewport( viewport(layout = grid.layout(100, 20) ))
      ## ------------------
      print( p, vp=viewport(layout.pos.row=1:100, layout.pos.col=1:12))
      print( ps, vp=viewport(layout.pos.row=4:95, layout.pos.col=13:19))
      ## ------------------
      dev.off()
      ## -------------------------------------
      return()
    }
    ## ----------------------------------------------------------------------
    if(return_p == T)
      return(p)
    ggsave(p, file = savename, width = 9, height = 15)
  }
```

16 File: list_merge_df.R

```
list_merge_df <-
  function(
           list,
           df,
           ...
           ){
    assign("envir_meta", environment(), parent.env(environment()))
    list <- lapply(list, merge,
                   y = get("df", envir = get("envir_meta")),
                   ...)
    return(list)
  }
```

17 File: load_extra.R

```
load_extra <-
  function(
           path = "~/extra/R"
           ){
    load_all(path)
  }
```

18 File: load_mgf.R

```
filter_mgf <-
  function(
           filter_id = prapare_inst_data(.MCn.structure_set)$.id,
           file = "~/Downloads/msp/msms_pos_gnps.msp.mgf"
           ){
    mgf <- read_msp(file)
    start <- which(mgf$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)
  }
```

19 File: load_svg.R

```
read_svg <-
  function(
           file,
           as_cairo = T,
           grobify = F,
           arrange = F
           ){
    if(as_cairo)
      rsvg::rsvg_svg(file, file)
    ## ------------------------------------
    svg <- grImport2::readPicture(file)
    ## ------------------------------------
    if(grobify)
      svg <- grImport2::grobify(svg)
    ## ------------------------------------
    if(arrange)
      svg <- gridExtra::arrangeGrob(svg)
    return(svg)
  }
grid_draw_svg.legend <-
  function(
           main,
           legend,
           savename,
           position.main = 0.55,
           position.legend = 0.1,
           main_size = 1,
           legend_size = 0.8,
           width = 13,
           height = 12
           ){
    svg(savename, width = width, height = height)
    ## ------------------
    grImport2::grid.picture(main, width = main_size, height = main_size, x = position.main)
    grImport2::grid.picture(legend, width = legend_size, height = legend_size, x = position.legend)
    ## ------------------
    dev.off()
  }
```

20 File: mass_shift.R

```
# |mass      |inte |
# |:---------|:----|
# |117.06971 |20   |
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)
```

```r
    ## 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))), ]
    ## reshape intensity
    noise <- dplyr::mutate(noise, inte = max(df$inte) * re.inte)
    ## 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)
  }
```

21 File: merge_horizon_accuracy.R

```r
merge_horizon_accuracy <-
  function(
           list,
           title,
           savename,
           palette = ggsci::pal_simpsons()(9),
           ylab = "stat ratio",
           xlab = "classification",
           fill_lab = "type",
           return_p = t
           ){
    list.name <- names(list)
    list <- lapply(list, reshape2::melt,
                   id.vars = "classification",
                   variable.name = "type",
                   value.name = "value")
    list <- lapply(list, dplyr::mutate,
                   classification = stringr::str_wrap(classification, width = 25),
                   type = as.character(type),
                   type = hmisc::capitalize(type))
    df <- data.table::rbindlist(list, idcol = t) %>%
      dplyr::filter(type == "true")
    line_df <- reshape2::dcast(df, classification + type ~ .id)
    ## ----------------------------------------------------------------------
    p <- ggplot(data = df,
                aes(x = classification,
                    y = value,
                    color = .id)) +
      geom_segment(data = line_df,
                   aes(x = classification,
                       xend = classification,
                       y = eval(parse(text = paste0("`", list.name[1], "`"))),
                       yend = eval(parse(text = paste0("`", list.name[2], "`")))),
                   color = "black") +
      geom_point(size = 5,
                 position = "identity") +
      scale_color_manual(values = palette) +
      labs(title = hmisc::capitalize(title),
           y = hmisc::capitalize(ylab),
           x = hmisc::capitalize(xlab),
           color = hmisc::capitalize(fill_lab)) +
      coord_flip() +
      theme(legend.position = "bottom",
            text = element_text(family = "times", size = 20, face = "bold"),
            plot.title = element_text(hjust = 0.3))
      ## ----------------------------------------------------------------------
      if(return_p == t)
        return(p)
      ggsave(p, file = savename, width = 9, height = 15)
  }
mutate_merge_horizon_accuracy <-
  function(
           list,
        title,
        savename,
        palette = ggsci::pal_simpsons()(9),
        ylab = "stat ratio",
        xlab = "classification",
        fill_lab = "type",
        return_p = T
        ){
list <- lapply(list, reshape2::melt,
               id.vars = "classification",
               variable.name = "type",
               value.name = "value")
list <- lapply(list, dplyr::mutate,
               classification = stringr::str_wrap(classification, width = 25),
               type = as.character(type),
               type = Hmisc::capitalize(type))
df <- data.table::rbindlist(list, idcol = T) %>%
  dplyr::filter(type == "True")
line_df <- reshape2::dcast(df, classification + type ~ .id)
## ----------------------------------------------------------------------
p <- ggplot(data = df,
            aes(x = classification,
                y = value,
                color = .id)) +
  geom_segment(data = line_df, aes(x = classification, xend = classification, y = origin, yend = re
               color = "black") +
  geom_point(size = 5,
             position = "identity") +
  scale_color_manual(values = palette) +
  labs(title = Hmisc::capitalize(title),
       y = Hmisc::capitalize(ylab),
       x = Hmisc::capitalize(xlab),
       fill = Hmisc::capitalize(fill_lab)) +
  coord_flip() +
  theme(legend.position = "bottom",
        text = element_text(family = "Times", size = 20, face = "bold"),
        plot.title = element_text(hjust = 0.3))
  ## ----------------------------------------------------------------------
  if(return_p == T)
    return(p)
  ggsave(p, file = savename, width = 9, height = 15)
  }
```

22 File: meta_compound_filter.R

```
meta_compound_filter <-
  function(
           list,
           vip,
           dose = "high",
           l_abs_log_fc = 1, ## or 0
           l_q_value = 0.05, ## or 1
           l_vip = 1, ## or 0
           id_fix = T,
           round = T
           ){
    if(id_fix == T)
      vip <- dplyr::mutate(vip, id = gsub("X", "", id))
    ## ----------------------------------------------------------------------
    set <- data.table::rbindlist(list)
    ## -------------------------------------
    if(round == T){
      set <- dplyr::mutate(set, log2.fc = round(log2.fc, 2))
    }
    ## ----------------------------------------------------------------------
    ## control and model
    part1 <- dplyr::filter(set, facet_row == "extra") %>%
      merge(vip, by = "id", all.x = T, sort = F) %>%
      dplyr::filter(vip > l_vip,
                    abs(log2.fc) > l_abs_log_fc,
                    q_value < l_q_value) %>%
      dplyr::as_tibble()
    ## -------------------------------------
    if(round == T){
      part1 <- dplyr::mutate(part1, vip = round(vip, 2))
    }
    ## ----------------------------------------------------------------------
    ## drug dispose
    part2 <- dplyr::filter(set, facet_row == "high") %>%
      by_group_as_list("facet_col") %>%
      ## ------------------ rename the col
      lapply(., meta_rename_prefix,
             col = c("p_value", "q_value", "log2.fc"),
             prefix_from_col = "facet_col", internal = "#") %>%
      ## ------------------ select the needed col
      lapply(., dplyr::select,
             id, contains("#")) %>%
      ## ------------------ merge into a data.frame
      meta_merge_list(col = "id") %>%
      dplyr::as_tibble()
    ## ----------------------------------------------------------------------
    ## gather data
    df <- part1 %>%
      dplyr::select(id, vip) %>%
      merge(., part2, by = "id", all.x = T, sort = F) %>%
      dplyr::filter(abs(`raw_model#log2.fc`) > 1 | abs(`pro_model#log2.fc`) > 1) %>%
      dplyr::as_tibble()
    return(df)
  }
meta_rename_prefix <-
  function(
           df,
           col,
           prefix = NA,
           prefix_from_col = NA,
           internal = "."
           ){
    if(is.na(prefix_from_col) == F){
      prefix <- paste0(df[1, ][[prefix_from_col]], internal)
    }
    colnames(df) <- base_meta_rename_prefix(colnames(df),
                                            mutate = col,
                                            PREFIX = prefix)
    return(df)
  }
base_meta_rename_prefix <-
  function(
           names,
           mutate,
           PREFIX
           ){
    df <- data.table::data.table(origin = names) %>%
      dplyr::mutate(change = ifelse(origin %in% mutate,
```

```r
                                    paste0(PREFIX, origin),
                                    origin))
    return(df$change)
  }
meta_merge_list <-
  function(
           list,
           col = "id"
           ){
    df <- list[[1]]
    for(i in 2:length(list)){
      df <- merge(df, list[[i]], by = col, all.x = T, sort = F)
    }
    return(df)
  }
```

23 File: meta_do_list.R

```r
meta_do_list <-
  function(
           metadata
           ){
    list <- by_group_as_list(metadata, "group")
    list <- lapply(list, select, sample) %>%
      lapply(unlist) %>%
      lapply(unname)
    return(list)
  }
```

24 File: meta_gather_pub_classyfire_sirius.R

```r
meta_gather_pub_classyfire_sirius <-
  function(
           ## contain sp.id
           pub,
           class,
           ## contain sp.id
           sirius
           ){
    class_anno <- gather_classyfire(class = class, inchi_df = pub)
    ## --------------------------------------
    ## annotate sirius results with classification
    simp_candi <- sirius %>%
      merge(class_anno, by.x = "inchikey2D", by.y = "inchi2d", all.x = T) %>%
      dplyr::filter(classification == class) %>%
      dplyr::select(.id, inchikey2D, name, classification, tanimotoSimilarity) %>%
      dplyr::mutate(sp.id = rownames(.)) %>%
      dplyr::as_tibble()
    return(simp_candi)
  }
```

25 File: meta_get_couple.R

```
meta_get_couple <-
  function(
           group
           ){
    compare <- group %>%
      .[which(!. %in% c("blank", "positive"))] %>%
      unique() %>%
      combn(2) %>%
      t() %>%
      data.table::data.table() %>%
      ## exclude group compare in different dosage
      dplyr::filter(stringr::str_extract(V1, "_.*$") == stringr::str_extract(V2, "_.*$") |
                    ## get the control or model group
                    V1 %in% c("control", "model") |
                    V2 %in% c("control", "model"),
                  ## with control, only compare with model
                  !(V1 == "control" & V2 != "model"),
                  !(V2 == "control" & V1 != "model"))
      return(compare)
  }
meta_get_extra_couple <-
  function(
           compare,
           ...
           ){
    extra_compare_1 <- compare %>%
      dplyr::filter(!((V1 == "control" & V2 =="model") | (V1 == "model" & V2 == "control"))) %>%
      dplyr::mutate(V3 = "control", V4 = "model")
    extra_compare_2 <- extra_compare_1 %>%
      apply(., 1, .meta_muti_add) %>%
      lapply(unlist) %>%
      unique()
    extra_compare_3 <- compare %>%
      apply(., 1, .meta_muti_add) %>%
      lapply(unlist) %>%
      unique() %>%
      .[which(lengths(.) != 2)]
    list <- list(extra_compare_1, extra_compare_2, extra_compare_3)
    names(list) <- c("c1", "c2", "c3")
    return(list)
  }
.meta_muti_add <-
  function(
           vector,
           dose = c("high", "medium", "low")
           ){
    group <- stringr::str_extract(vector, "^.*(?=_)") %>%
      sort() %>%
      expand.grid(dose) %>%
      dplyr::mutate(combine = paste0(Var1, "_", Var2))
    combine <- c(group$combine, vector) %>%
      unique()
    return(combine)
  }
vector_delete_var <-
  function(
           vector,
           delete
           ){
    vector <- vector[!vector %in% delete]
    return(vector)
  }
```

26 File: meta_get_metadata.R

```
meta_get_metadata <-
  function(
           mzmine_col_peak_area
           ){
```

```
    metadata <- mzmine_col_peak_area %>%
      gsub(" Peak area", "", .) %>%
      data.table::data.table(sample = .) %>%
      dplyr::mutate(name = stringr::str_extract(sample, ".*(?=\\.)"),
                    prefix = stringr::str_extract(name, "[^[0-9]]*(?=[0-9])"),
                    identifier = stringr::str_extract(name, "[0-9]{1,100}"),
                    group = unlist(lapply(prefix, sub_data, meta = GROUP)),
                    identifier = paste0(group, "_", identifier),
                    super_group = gsub("_.*$", "", group)) %>%
      dplyr::select(sample, identifier, group, super_group) %>%
      dplyr::arrange(identifier)
    return(metadata)
  }
```

27 File: meta__metabo__pathway.R

```
meta_metabo_pathway <-
  function(
           export = NA,
           mz_rt = NA,
           p_col = NA,
           extra_entity = NA,
           only_return = F,
           ## from name involves the character rename into the columns
           key = c("mz", "q_value", "log2.fc", "rt"),
           ## note that both key and as_col must be ordered
           as_col = c("m.z", "p.value", "t.score", "r.t"),
           ion_mode = "negative",
           ppm = 10,
           p_cutoff = 0.05,
           db_pathway = "hsa_mfn"
           ){
    ## ---------------------------------------------------------------------
    if(is.data.frame(extra_entity)){
      df_mz_rt <- extra_entity
    }else{
      df_mz_rt <- mz_rt %>%
        dplyr::filter(id %in% export$id) %>%
        merge(export[, c("id", p_col)], all.x = T, by = "id") %>%
        dplyr::as_tibble()
    }
    ## ----------------------------------------------------------------------
    ## note that this step is setting up to auto find and rename
    colnames(df_mz_rt) <- colnames(df_mz_rt) %>%
      ## find and sort as order of `key`
      .meta_find_and_sort(., key) %>%
      ## rename columns of df_mz_rt
      mapply_rename_col(., as_col, colnames(df_mz_rt))
    ## -------------------------------------
    ## then, select and relocate (sort) the columns of df
    df <- dplyr::select(df_mz_rt, all_of(as_col)) %>%
      ## convert rt from min to secounds
      dplyr::mutate(r.t = r.t * 60)
    ## ----------------------------------------------------------------------
    ## save to file
    write_tsv(df, file = "tmp.txt")
    ## -------------------------------------
    ## get the submit file
    if(only_return == T)
      return(list(id = df_mz_rt, submit = df))
    ## -------------------------------------
    cat("## submit to MetaboAnalyst\n")
    print(dplyr::as_tibble(df))
    ## submit to MetaboAnalyst
    mSet <- InitDataObjects("mass_all", "mummichog", FALSE)
    mSet <- SetPeakFormat(mSet, "mprt")
    mSet <- UpdateInstrumentParameters(mSet, ppm, ion_mode, "yes", 0.02);
    mSet <- Read.PeakListData(mSet, "tmp.txt");
    mSet <- SetRTincluded(mSet, "seconds")
    mSet <- SanityCheckMummichogData(mSet)
    mSet <- SetPeakEnrichMethod(mSet, "mum", "v2")
    mSet <- SetMummichogPval(mSet, p_cutoff)
    mSet <- PerformPSEA(mSet, db_pathway, "current", 3 , 100)
    return(mSet)
  }
.meta_find_and_sort <-
  function(
           name_set,
           pattern_set
           ){
    name_set <- lapply(pattern_set, .meta_mutate_grep_get,
                       string_set = name_set) %>%
      unlist()
    return(name_set)
  }
.meta_mutate_grep_get <-
  function(
           pattern,
           string_set
           ){
    string <- string_set %>%
      .[grepl(pattern, .)]
    return(string)
  }
```

28 File: meta_oplsda.R

```
meta_oplsda <-
  function(
           df,
           metadata,
           GROUP
           ){
    metadata <- metadata %>%
      dplyr::filter(group %in% all_of(GROUP)) %>%
      dplyr::select(sample, group)
    ## ----------------------------------------------------------------------
    ## collate data
    df <- filter_via_rownames(df, metadata$sample) %>%
      data.frame() %>%
      dplyr::mutate(sample = rownames(.)) %>%
      ## in order to sort sample with group
      merge(metadata, by = "sample", all.x = T)
    ## select the col of var
    gr <- grepl("sample|group", colnames(df))
    ## scale the data
    matrix <- meta_scale(df[, !gr])
    ## ----------------------------------------------------------------------
    ## opls-da
    cat("## calculate OPLS-DA...\n")
    oplsda <- ropls::opls(x = matrix, y = unlist(df[, "group"]),
                          predI = 1, orthoI = NA)
    ## ----------------------------------------------------------------------
    ## gather opls-da results
    ## T score
    tscore <- oplsda@modelDF[1, "R2X"] * 100
    ## O score
    oscore <- oplsda@modelDF[2, "R2X"] * 100
    ## ------------------------------------
    ## collate as df
    part1 <- data.table(h1 = oplsda@scoreMN[, 1], o1 = oplsda@orthoScoreMN[, 1]) %>%
      dplyr::bind_cols(df[, gr]) %>%
      dplyr::mutate(x_lab = paste0("T score[1] (", tscore, "%)"),
                    y_lab = paste0("Orthogonal T score[1] (", oscore, "%)"))
      dplyr::as_tibble()
    ## ----------------------------------------------------------------------
    ## gather VIP value
    part2 <- data.frame(oplsda@vipVn) %>%
      ## get id
      dplyr::mutate(id = rownames(.)) %>%
      ## rename
      dplyr::rename(vip = oplsda.vipVn) %>%
      dplyr::as_tibble()
    ## ----------------------------------------------------------------------
    list <- list(oplsda_coord = part1, vip = part2)
    ## ------------------------------------
    return(list)
  }
```

29 File: meta_re_collate_iupac_via_inchi.R

```
meta_re_collate_iupac_via_inchi <-
  function(
           simp_candi,
           name_df,
           export
           ){
    export <- simp_candi %>%
      merge(name_df, by = "sp.id", all.x = T) %>%
      dplyr::mutate(name = ifelse(name == "null", IUPACName, name)) %>%
      dplyr::select(.id, name, classification, tanimotoSimilarity) %>%
      dplyr::rename(id = .id, info = classification) %>%
      merge(export, by = "id", all.y = T) %>%
      dplyr::filter(is.na(name) == F) %>%
      dplyr::arrange(desc(tanimotoSimilarity)) %>%
      dplyr::as_tibble()
    return(export)
  }
```

30 File: meta_summarise_via_group.R

```
# |V1         |V2         |
# |:----------|:----------|
# |control    |model      |
# |model      |pro_high   |
# |model      |pro_low    |
# |model      |pro_medium |
# |model      |raw_high   |
# |model      |raw_low    |
# |model      |raw_medium |
# |pro_high   |raw_high   |
# |pro_low    |raw_low    |
# |pro_medium |raw_medium |
## ------------------------------------
## load into a metadata
meta_summarise_via_group <-
  function(
           df,
           compare
           ){
    list <- apply(compare, 1, base_meta_summarise_via_group,
                  df = df, simplify = F)
    return(list)
  }
## ----------------------------------------------------------------------
# $fc
# [1] "test"

# $p_value
# [1] "test"

# $q_value
# [1] "test"
## ------------------------------------
base_meta_summarise_via_group <-
```

```
  function(
           group,
           df
           ){
    GROUP <- mutate_meta_sort(group)
    cat("## computaton of", paste(GROUP, collapse = "/"), "\n")
    ## -------------------
    df <- dplyr::filter(df, group %in% all_of(GROUP))
    ## -------------------
    x_row <- grep(GROUP[1], df$group)
    ## x versus y
    y_row <- grep(GROUP[2], df$group)
    ## ----------------------------------------
    cat("## ------------------- log2(FC)\n")
    fc_set <- meta_calculate_couple(df, x_row, y_row, group,
                                    "base_meta_calculate_fc") %>%
      dplyr::rename(log2.fc = expr)
    ## ----------------------------------------
    cat("## ------------------- t.test\n")
    p_set <- meta_calculate_couple(df, x_row, y_row, group,
                                    "base_meta_calculate_p")
    ## ----------------------------------------
    cat("## ------------------- FDR\n")
    q_set <- p_set %>%
      dplyr::filter(is.na(expr) == F) %>%
      dplyr::mutate(q_value = fdrtool::fdrtool(expr, statistic = 'pvalue', plot = F)$qval) %>%
      dplyr::rename(p_value = expr)
    ## ----------------------------------------
    ## gather all data
    df <- q_set %>%
      dplyr::select(id, p_value, q_value) %>%
      merge(fc_set, by = "id", all.x = T, sort = F) %>%
      dplyr::as_tibble()
    ## ----------------------------------------
    return(df)
  }
## ---------------------------------------------------------------------------
meta_calculate_couple <-
  function(
           df,
           x_row,
           y_row,
           group,
           fun
           ){
    fun <- match.fun(fun)
    gr <- grepl("sample|group", colnames(df))
    df <- df[, !gr]
    ## use in multiple function
    data_set <- pbapply::pbapply(df, 2, fun, x_row, y_row)
    ## reformat and then return
    data_set <- data.table::data.table(id = names(data_set), expr = unname(data_set),
                                       facet_col = meta_get_facet_col(group),
                                       facet_row = meta_get_facet_row(group))
    return(data_set)
  }
## ------------------------------------
base_meta_calculate_fc <-
  function(
           vector,
           x_row,
           y_row
           ){
    vector <- vector + 1
    fc <- log2(mean(vector[x_row]) / mean(vector[y_row]))
    return(fc)
  }
## ------------------------------------
base_meta_calculate_p <-
  function(
           vector,
           x_row,
           y_row
           ){
    vector <- vector + 1
    x <- vector[x_row]
    y <- vector[y_row]
    check <- try(stat <- t.test(x, y, var.equal = T, paired = F), silent = T)
    if(class(check) == "try-error"){
      return(NA)
    }
    stat <- stat$p.value
    return(stat)
  }
## ---------------------------------------------------------------------
## trans df format
meta_array_to_df <-
  function(
           compute_df,
           metadata
           ){
    df <- compute_df %>%
      data.frame(check.names = F) %>%
      dplyr::mutate(sample = rownames(.)) %>%
      merge(metadata[, c("sample", "group"), with = F], by = "sample", all.x = T) %>%
      dplyr::select(sample, group, colnames(.)) %>%
      dplyr::as_tibble()
    return(df)
  }
mutate_meta_sort <-
  function(
           vector,
           levels = c("pro", "raw", "model", "control")
           ){
    df <- data.table::data.table(origin = vector)
    df <- dplyr::mutate(df, mutate = gsub("_.*$", "", origin),
                        mutate = factor(mutate, levels = levels))
    df <- dplyr::arrange(df, mutate)
    return(df$origin)
  }
```

31 File: metabo_get_id_via_mz_rt.R

```
metabo_get_id_via_mz_rt <-
  function(
           metabo_results,
           mutate_mz_rt
           ){
    metabos <- metabo_results %>%
      lapply(dplyr::rename, mz = Query.Mass, rt = Retention.Time) %>%
      lapply(merge, mutate_mz_rt, by = c("mz", "rt")) %>%
      lapply(dplyr::mutate,
             info = paste0(pathway,  " ---- Gamma: ", Gamma, " ---- Hits.sig: ", Hits.sig)) %>%
      lapply(dplyr::select, id, name, mz, info) %>%
      lapply(dplyr::as_tibble)
  }
```

32 File: mgf_add_anno.gnps.R

```
mgf_add_anno.gnps <-
  function(
           df
           ){
    slice_line <- list("1:3", "4:6", "7:nrow(df)")
    list <- lapply(slice_line, function(lines){
                   list <- slice(df, eval(parse(text = lines)))
                   return(list)
           })
    ## ------------------------------------
    ## scans
    scans <- str_extract(list[[1]][2, 1], "[0-9]{1,}$")
    scans <- c(V1 = paste0("SCANS=", scans))
    ## ------------------------------------
    ## merge
    merge <- c(V1 = "MERGED_STATS=1 / 1 (0 removed due to low quality, 0 removed due to low cosine)")
    ## ------------------------------------
    df <- bind_rows(list[[1]], scans, list[[2]], merge, list[[3]])
    return(df)
  }
mgf_add_anno.mistree <-
  function(
           df
           ){
    mass_level <- df$V1[grepl("MSLEVEL", df$V1)]
    id_line <- df$V1[grepl("FEATURE_ID", df$V1)]
    ## ---------------------------------------------------------------------
    ## process level 1
    if(mass_level == "MSLEVEL=1"){
      slice_line <- list("1:4", "5", "6:nrow(df)")
      list <- lapply(slice_line, function(lines){
                     list <- slice(df, eval(parse(text = lines)))
                     return(list)
           })
      ## ------------------------------------
```

```r
      ## rt
      rt <- c(V1 = "RTINSECONDS=1000")
      ## spectype
      sp <- c(V1 = "SPECTYPE=CORRELATED MS")
      ## -------------------------------------
      ## filename
      filename <- c(V1 = "FILENAME=sample.mzML")
      ## scans
      scans <- c(V1 = paste0("SCANS=", str_extract(id_line, "[0-9]{1,}$")))
      ## -------------------------------------
      ## bind rows
      df <- bind_rows(list[[1]], rt, sp, list[[2]], filename, scans, list[[3]])
      return(df)
    }else{
      ## ---------------------------------------------------------------------
      ## process level 2
      slice_line <- list("1:6", "7:nrow(df)")
      list <- lapply(slice_line, function(lines){
                       list <- slice(df, eval(parse(text = lines)))
                       return(list)
           })
      ## -------------------------------------
      filename <- c(V1 = "FILENAME=sample.mzML")
      df <- bind_rows(list[[1]], filename, list[[2]])
      return(df)
    }
  }
```

33 File: msp_to_mgf.R

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

34 File: multi_formula_adduct_align.R

```
multi_formula_adduct_align <-
  function(
           list,
           db,
           ...
           ){
    list <- pbapply::pblapply(list, mz_df_align, db = db, ...)
    return(list)
  }
mz_df_align <-
  function(
           df,
           db,
           col = "mass",
           ...
           ){
    list <- lapply(df[[col]], mz_align, df = db, ...)
    adduct <- by_group_as_list(df, col)
    list <- mapply(mutate_bind_cols, adduct, list, SIMPLIFY = F)
    df <- data.table::rbindlist(list, fill = T)
    return(df)
  }
mutate_bind_cols <-
  function(
           row,
           df
           ){
    df <- dplyr::bind_cols(row[rep(1, nrow(df)),], df)
    return(df)
  }
```

35 File: mutate_head.R

```
mutate_head <-
  function(
           df,
           row = 10,
           col = 10
           ){
```

```r
    df <- df[1:row, 1:col]
    return(df)
  }
```

36 File: mutate_horizon_bar_accuracy.R

```r
mutate_horizon_bar_accuracy <-
  function(
           df,
           title,
           savename,
           palette = ggsci::pal_npg()(9),
           extra_palette = ggsci::pal_rickandmorty()(12),
           ylab = "stat ratio",
           xlab = "classification",
           fill_lab = "type",
           extra_sides_df = NULL,
           return_p = T,
           width = 16,
           height = 15,
           l_ratio = 63,
           m_ratio = 138,
           extra_col_max = 500
           ){
    ## ------------------------------------
    ## get parent class
    parent_class <- mutate_get_parent_class(df$classification) %>%
      lapply(., end_of_vector) %>%
      unlist(use.names = F)
    df <- dplyr::mutate(df, parent_class = ifelse(is.na(parent_class), classification, parent_class))
    ## ----------------------------------------------------------------------
    annotation <- df %>%
      dplyr::mutate(combine = paste0(classification, " ---- ", parent_class))
    df <- reshape2::melt(df, id.vars = c("classification", "parent_class"),
                         variable.name = "type",
                         value.name = "value")
    df <- dplyr::mutate(df,
                        classification = stringr::str_wrap(classification, width = 25),
                        parent_class = stringr::str_wrap(parent_class, width = 25),
                        type = as.character(type),
                        type = Hmisc::capitalize(type))
```

```r
## ----------------------------------------------------------------------
p <- ggplot(data = df,
            aes(x = classification,
                y = value,
                fill = type)) +
  geom_col(width = 0.7,
           position = "stack") +
  scale_fill_manual(values = palette) +
  labs(title = Hmisc::capitalize(title),
       y = Hmisc::capitalize(ylab),
       x = Hmisc::capitalize(xlab),
       fill = Hmisc::capitalize(fill_lab)) +
  coord_flip() +
  theme(legend.position = "bottom",
        axis.text.y = element_blank(),
        text = element_text(family = "Times", size = 20, face = "bold"),
        plot.title = element_text(hjust = 0.3))
## ----------------------------------------------------------------------
if(is.null(extra_sides_df) == F){
  max = extra_col_max
  ps <- ggplot(data = extra_sides_df) +
    geom_col(width = 0.7,
             fill = "#709AE1FF",
             alpha = 0.7,
             aes(x = classification, y = ifelse(sum >= max, max, sum))) +
    coord_flip() +
    ylim(0, max) +
    labs(x = "", y = "Compounds number") +
    theme(axis.text.y = element_blank(),
          axis.ticks = element_blank(),
          text = element_text(family = "Times", size = 20, face = "bold"))
  ## -------------------------------------
  pa1 <- ggplot(annotation) +
    geom_tile(aes(x = "classification", y = stringr::str_wrap(classification, width = 25),
                  fill = stringr::str_wrap(parent_class, width = 25)),
              width = 1, height = 1, alpha = 0.5, size = 1, color = "black") +
    labs(fill = "", x = "", y = "") +
    theme_minimal() +
    scale_fill_manual(values = colorRampPalette(extra_palette)(length(unique(annotation$parent_class
    theme(text = element_text(size = 14, face = "bold", family = "Times"),
          axis.text.x = element_blank(),
              legend.key.height = unit(1.5, "cm"),
              legend.position = "left",
              panel.grid = element_blank())
      ## -------------------------------------
      svg(savename, width = width, height = height)
      grid.newpage()
      pushViewport( viewport(layout = grid.layout(100, 200) ))
      ## ------------------
      ## classification
      print( pa1, vp = viewport(layout.pos.row = 5:94, layout.pos.col = 1:l_ratio))
      ## cluster accuracy
      print( p, vp = viewport(layout.pos.row = 2:100, layout.pos.col = (l_ratio + 2):m_ratio))
      ## compounds number
      print( ps, vp = viewport(layout.pos.row = 5:96, layout.pos.col = (m_ratio + 4):195))
      ## ------------------
      dev.off()
      ## -------------------------------------
      return()
    }
    ## ----------------------------------------------------------------------
    if(return_p)
      return(p)
    ggsave(p, file = savename, width = 9, height = 15)
  }
end_of_vector <-
  function(
           vector
           ){
    if(length(vector) == 0){
      return(NA)
    }
    var <- vector[length(vector)]
    return(var)
  }
```

37 File: mutate2_horizon_bar_accuracy.R

```
mutate2_horizon_bar_accuracy <-
  function(
           df_list,
           extra_list,
           ## -------------------------------------
           title,
           savename,
           ylab = "stat ratio",
           xlab = "classification",
           fill_lab = "Type",
           ## -------------------------------------
           palette = ggsci::pal_npg()(9),
           mutate_palette = c("true" = palette[3],
                              "latent" = palette[2],
                              "false" = palette[1],
                              "noise" = "#FED439FF",
                              "high_noise" = "#8A4198FF"),
           extra_palette = c("sum" = "#95CC5EFF",
                             "noise" = "#FED439FF",
                             "high_noise" = "#8A4198FF"),
           group_palette = ggsci::pal_rickandmorty()(12),
           ## -------------------------------------
           width = 18,
           height = 15,
           l_ratio = 57,
           m_ratio = 130,
           y_cut_left = c(50, 500),
           y_cut_right = c(900, 1300),
           y_cut_left_breaks = c(50, seq(100, 500, by = 100)),
           y_cut_right_breaks = c(1000, 1200)
           # extra_col_max = NA
           ){
    ## ----------------------------------------------------------------------
    ## ----------------------------------------------------------------------
    ## ----------------------------------------------------------------------
    ## get parent class
    df_list <- lapply(df_list, function(df){
                        parent_class <- mutate_get_parent_class(df$classification) %>%
                          lapply(., end_of_vector) %>%
                          unlist(use.names = F)
                        df <- dplyr::mutate(df, parent_class = ifelse(is.na(parent_class),
                                                                      classification,
                                                                      parent_class),
                                            st.true = 0, en.true = true,
                                            st.latent = en.true, en.latent = st.latent + latent,
                                            st.false = en.latent, en.false = st.false + false)
           })
    ## ----------------------------------------------------------------------
    ## ----------------------------------------------------------------------
    ## ----------------------------------------------------------------------
    ## group draw
    annotation <- df_list[["origin"]]
    pa1 <- ggplot(annotation) +
      geom_tile(aes(x = "classification", y = stringr::str_wrap(classification, width = 25),
                    fill = stringr::str_wrap(parent_class, width = 25)),
                width = 1, height = 1, alpha = 0.5, size = 1, color = "black") +
      labs(fill = "", x = "", y = "") +
      theme_minimal() +
      scale_fill_manual(values = colorRampPalette(group_palette)(length(unique(annotation$parent_class)
      theme(text = element_text(size = 14, face = "bold", family = "Times"),
            axis.text.x = element_blank(),
            legend.key.height = unit(1.5, "cm"),
            legend.position = "left",
            panel.grid = element_blank())
    ## ----------------------------------------------------------------------
    ## ----------------------------------------------------------------------
    ## ----------------------------------------------------------------------
    ## initial stat
    mutate_origin <- df_list[["origin"]] %>%
      reshape2::melt(., id.vars = colnames(.)[!colnames(.) %in% c("true", "false", "latent")],
                     variable.name = "type",
                     value.name = "value") %>%
    dplyr::mutate(., y = as.numeric(apply(., 1, function(v){
                                            v[[paste0("st.", v[["type"]])]]
                                                                      })),
                  yend = as.numeric(apply(., 1, function(v){
                                 v[[paste0("en.", v[["type"]])]]
                                            })))
    ## ----------------------------------------------------------------------
    ## noise dirft
    noise_df <- mutate2.horizon.tmp_merge("origin", "noise", df_list) %>%
      dplyr::filter(y != yend, exclude == F,
                    classification %in% mutate_origin$classification)
    ## high noise drift
    h_noise_df <- mutate2.horizon.tmp_merge("noise", "h_noise", df_list) %>%
      dplyr::filter(y != yend, exclude == F,
```

```r
                  classification %in% mutate_origin$classification)
## ------------------------------------------------------------------
p <- ggplot() +
  ## origin
  geom_segment(data = mutate_origin,
               aes(x = classification, xend = classification,
                   y = y, yend = yend,
                   color = type),
               size = 7) +
  ## noise drift
  geom_segment(data = noise_df,
             aes(x = classification, xend = classification,
                 y = y, yend = yend,
                 color = "noise"),
               size = 7,
               inherit.aes = F) +
  ## high noise drift
  geom_segment(data = h_noise_df,
             aes(x = classification, xend = classification,
                 y = y, yend = yend,
                 color = "high_noise"),
               size = 7,
               inherit.aes = F) +
  ## the point indicate the start of noise drift
  geom_segment(data = noise_df,
             aes(x = classification, xend = classification,
                 y = ifelse(yend > y, y - 0.001, y + 0.001), yend = y,
                 color = "noise"),
               arrow = arrow(length = unit(10, "pt")),
               size = 0.5, lineend = "round") +
  ## the point indicate the start of high noise drift
  geom_segment(data = h_noise_df,
             aes(x = classification, xend = classification,
                 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", noise = "middle noise", high_noise = "high noise")) +
  labs(title = Hmisc::capitalize(title),
       y = Hmisc::capitalize(ylab),
        x = Hmisc::capitalize(xlab),
        color = Hmisc::capitalize(fill_lab)) +
  coord_flip() +
  theme(legend.position = "bottom",
        axis.text.y = element_blank(),
        text = element_text(family = "Times", size = 20, face = "bold"),
        plot.title = element_text(hjust = 0.3))
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
extra.noise_df <- mutate2.extra.horizon.tmp_merge(extra_list = extra_list)
## -------------------------------------
ps <- ggplot() +
    ## origin sum
    geom_segment(data = extra_list[["origin"]],
                 aes(x = classification,
                     xend = classification,
                     y = 0,
                     yend = sum,
                     color = "sum"),
                 size = 7
                 ) +
    ## noise drift
    geom_segment(data = dplyr::mutate(extra.noise_df,
                                      sum.x = ifelse(is.na(sum.x), 0, sum.x)),
                 aes(x = classification,
                     xend = classification,
                     y = sum,
                     yend = sum.x,
                     color = "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 = classification,
                     xend = classification,
                     y = sum.x,
                     yend = sum.y,
                     color = "high_noise"),
                  size = 7
                  ) +
    scale_color_manual(values = extra_palette,
                       labels = c(sum = "sum", noise = "middle noise", high_noise = "high noise")) +
    labs(x = NULL, y = NULL, color = "Type") +
    theme(axis.text.y = element_blank(),
          axis.ticks = element_blank(),
          text = element_text(family = "Times", size = 20, face = "bold"))
## -------------------------------------
## do coord. axis cut off
ps1 <- ps +
  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)
ps2 <- ps +
  coord_flip(ylim = y_cut_right) +
  scale_y_continuous(breaks = y_cut_right_breaks)
ps <- ggpubr::ggarrange(ps1, ps2, ncol = 2, nrow = 1,
                        widths = c(2/3, 1/3),
                        common.legend = TRUE, legend = "right", align = "h")
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
svg(savename, width = width, height = height)
grid.newpage()
pushViewport( viewport(layout = grid.layout(1000, 200) ))
## ------------------
## classification
## while 2 line of bottom legend, set to 923
adjust <- 940
print( pa1, vp = viewport(layout.pos.row = 30:adjust, layout.pos.col = 1:l_ratio))
## cluster accuracy
print( p, vp = viewport(layout.pos.row = 3:1000, layout.pos.col = (l_ratio + 2):m_ratio))
## compounds number
print( ps, vp = viewport(layout.pos.row = 30:adjust, layout.pos.col = (m_ratio + 4):195))
## ------------------
dev.off()
return()
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
```

```r
    ## ---------------------------------------------------------------------
  }
## difine a function
mutate2.extra.horizon.tmp_merge <-
  function(
           v1 = "noise",
           v2 = "h_noise",
           extra_list
           ){
    df <- merge(extra_list[[v1]], extra_list[[v2]],
                               by = "classification", all.x = T) %>%
      merge(extra_list[["origin"]], by = "classification", all.y = T) %>%
    return(df)
  }
mutate2.horizon.tmp_merge <-
  function(
           v1,
           v2,
           df_list
           ){
  df <- merge(df_list[[v1]], df_list[[v2]],
                    by = "classification", all.x = T) %>%
  dplyr::mutate(flow1 = "true", flow2 = "latent") %>%
  ## -------------------------------------
  ## both true and false is changing, so duplicated the col
  reshape2::melt(., id.vars = colnames(.)[!colnames(.) %in% c("flow1", "flow2")],
                 variable.name = "type",
                 value.name = "value") %>%
  ## -------------------------------------
  ## calculate segment from y to yend
  dplyr::mutate(., y = as.numeric(apply(., 1, function(v){
                                          v[[paste0("en.", v[["value"]], ".x")]]
                                            })),
                yend = as.numeric(apply(., 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))
  return(df)
}
```

38 File: pathway_horizon.R

```r
pathway_horizon <-
  function(
           df,
           title,
           ylab = "-log10(Gamma p)",
           xlab = "pathway",
           fill_lab = "",
           save = "tmp.svg"
           ){
    df <- dplyr::mutate(df, Gamma = -log10(Gamma),
                       pathway = stringr::str_wrap(pathway, width = 30)) %>%
      dplyr::arrange(desc(Gamma)) %>%
      dplyr::slice(1:20)
    p <- ggplot(data = df) +
      geom_point(position = "identity",
                 aes(x = reorder(pathway, Gamma),
                     y = Gamma,
                     size = Hits.sig,
                     color = Hits.sig)) +
      # geom_segment(aes(x = pathway, xend = pathway, y = 0, yend = Gamma)) +
      scale_color_gradient2(low = "white", mid = "#FED439FF", high = "#BB0021FF") +
      # geom_hline(yintercept = -log10(0.05), linetype = 3) +
      guides(size = "none") +
      labs(title = Hmisc::capitalize(title),
           y = Hmisc::capitalize(ylab),
           x = Hmisc::capitalize(xlab),
           fill = Hmisc::capitalize(fill_lab)) +
      coord_flip() +
      theme(legend.position = "right",
            plot.margin = unit(c(1.5, 1.5, 1.5, 1.5), "cm"),
            text = element_text(family = "Times", face = "bold")
            # axis.text = element_text(size = 6),
            # plot.title = element_text(hjust = 0.3)
      )
    ggsave(p, file = save, width = 7, height = 8)
  }
```

39 File: pca_via_group.R

```
pca_via_group <-
  function(
           df,
           compare,
           extra_compare,
           db_get_sample,
           ...
           ){
    ## -------------------------------------
    if(is.data.frame(compare)){
      cat("## compute dominant compare group\n")
      part1 <- pbapply::pbapply(compare, MARGIN = 1, base_pca_via_group,
                                df = df, meta = db_get_sample,
                                ...)
      part1 <- data.table::rbindlist(part1)
    }
    ## -------------------------------------
    if(is.list(extra_compare)){
      cat("## compute extra compare group\n")
      part2 <- pbapply::pblapply(extra_compare, internal_base_pca_via_group,
                                 df = df, meta = db_get_sample,
                                 ...)
      part2 <- data.table::rbindlist(part2)
    }
    ## -------------------------------------
    df <- dplyr::bind_rows(part1, part2) %>%
      dplyr::as_tibble()
    return(df)
  }
internal_base_pca_via_group <-
  function(
           extra_compare_part,
           df,
           meta,
           ...
           ){
    if(is.data.frame(extra_compare_part)){
      part <- pbapply::pbapply(extra_compare_part, MARGIN = 1, base_pca_via_group,
                               df = df, meta = meta, ...)
      part <- data.table::rbindlist(part)
      return(part)
    }
    if(is.list(extra_compare_part)){
      part <- lapply(extra_compare_part, base_pca_via_group,
                                  df = df, meta = meta, ...)
      part <- data.table::rbindlist(part)
      return(part)
    }
  }
base_pca_via_group <-
  function(
           group,
           df,
           meta,
           dose = c("high", "medium", "low"),
           ...
           ){
    ## according to group name to get sample file name
    group <- unique(group)
    list <- meta[names(meta) %in% group]
    sample <- unlist(list)
    ## ----------------------------------------------------------------------
    ## facet col (different dispose)
    facet_col <- meta_get_facet_col(group, ...)
    ## facet row (different dosage)
    facet_row <- meta_get_facet_row(group, dose)
    ## ----------------------------------------------------------------------
    df <- filter_via_rownames(df, sample)
    ## scale the data
    df <- meta_scale(df)
    ## ------------------
    pca <- prcomp(df, scale. = F)
    ## -------------------------------------
    ## PC annotation
    summary <- summary(pca)
    ## following, return a vector project
    #     PC1    PC2    PC3
    #  0.2595 0.2190 0.1774
    summary = c(round(summary$importance[2,],4))[1:3]
    ## ----------------------------------------------------------------------
    pca_coord <- dplyr::as_tibble(pca$x) %>%
      dplyr::select(PC1, PC2, PC3) %>%
      dplyr::mutate(sample = rownames(df),
                    ## facet annotation
                    facet_col = facet_col, facet_row = facet_row,
                    ## annotation of PC importance
                    im_PC1 = summary[["PC1"]], im_PC2 = summary[["PC2"]],
                    ## importance lengend in figure
                    legend_PC1 = paste0("PC1 (", im_PC1 * 100, "%)"),
                    ## the same as PC1
                    legend_PC2 = paste0("PC2 (", im_PC2 * 100, "%)"),
                    ## calculate the PC importance annotation coord in figure
                    ## x coord
                    anno_x = min(PC1) * (20 / 20),
                    ## y coord
                    anno_y = max(PC2) * (22 / 20),
                    ## pc3 is not considered ruster into figure
                    im_PC3 = summary[["PC3"]])
    ## ----------------------------------------------------------------------
    return(pca_coord)
  }
meta_sort <-
  function(
           vector,
           levels = c("pro", "raw", "model", "control"),
           ...
           ){
    vector <- gsub("_.*$", "", vector) %>%
      vector_delete_var("control") %>%
      unique()
    if(length(vector) == 1)
      vector <- c("control", "model")
    levels = c(levels, vector) %>%
      unique()
    vector <- sort(factor(vector, levels = levels))
    return(vector)
  }
meta_scale <-
  function(
           df
           ){
    df <- scale(df, center = T, scale = T)
```

```r
    exclude <- df[1,] %>%
      is.nan()
    df <- df[, !exclude]
    return(df)
  }
multi_extract <-
  function(
           vector,
           pattern_set
           ){
    vector <- lapply(pattern_set, base_muti_extract,
                     vector = vector)
    vector <- unlist(vector)
    return(vector)
  }
base_muti_extract <-
  function(
           pattern,
           vector
           ){
    character <- stringr::str_extract(vector, pattern)
    return(character)
  }
meta_get_facet_col <-
  function(
           group,
           ...
           ){
    facet_col <- meta_sort(group, ...) %>%
      paste(collapse = "_")
    ## if multiple group
    if(length(group) >= 3){
      check <- c("control", "model") %in% group
      if(F %in% check == F)
        facet_col <- paste0("multiple_", facet_col)
    }
    return(facet_col)
  }
meta_get_facet_row <-
  function(
           group,
           dose = c("high", "medium", "low")
           ){
    facet_row <- multi_extract(group, dose) %>%
      sort() %>%
      unique()
    ## if multiple dose
    if(length(facet_row) > 1){
      facet_row <- "multiple"
    }
    ## only control and model
    if(length(facet_row) == 0)
      facet_row <- "extra"
    return(facet_row)
  }
```

40 File: plot.facet_compare.R

```
plot.facet_compare <-
  function(
           list1,
           list2,
           ## -------------------------------------
           title,
           savename,
           ylim_min = 50,
           group_levels = c("origin", "noise", "high noise"),
           from = c("MCnebula", "MolnetEnhancer"),
           by_col = "classification",
           ylab = "In cluster numbers",
           xlab = "Classification",
           fill_lab = "Type",
           ## -------------------------------------
           palette = ggsci::pal_npg()(9),
           width = 18,
           height = 12
           ){
    ## select in common classification
    common.class <- merge(list1[[1]], list2[[1]], by = by_col) %>%
      dplyr::select(1)
    ## filter classification
    list <- list(list1, list2) %>%
      lapply(function(list){
               list <- lapply(list, merge, y = common.class, by = by_col, all.y = T) %>%
                 ## reset col name as sum
                 lapply(dplyr::rename, sum = 2) %>%
                 lapply(dplyr::mutate, sum = ifelse(is.na(sum), 0, sum))
           }) %>%
      lapply(data.table::rbindlist, idcol = T) %>%
      lapply(dplyr::rename, group = .id) %>%
      lapply(function(df){
               dplyr::mutate(df, group = mapply_rename_col("h_noise", "high noise", group))
           }) %>%
      mapply(function(df, VALUE){
               dplyr::mutate(df, from = VALUE)
           }, ., from, SIMPLIFY = F)
    ## for segment
    df <- merge(list[[1]], list[[2]], by = c("group", by_col))
    ## for point
    df2 <- dplyr::bind_rows(list[[1]], list[[2]])
    ## ----------------------------------------------------------------------
    ## plot figure
    p <- ggplot() +
      geom_segment(data = df,
                   aes(x = classification,
                       xend = classification,
                       y = sum.x,
                       yend = sum.y),
                   color = "black") +
      geom_point(data = df2,
                 aes(x = classification, y = sum, color = from),
                 size = 5,
                 position = "identity") +
      scale_color_manual(values = palette) +
      scale_y_continuous(breaks = c(ylim_min, 300, 600, 900, 1200)) +
      labs(title = Hmisc::capitalize(title),
           y = Hmisc::capitalize(ylab),
           x = Hmisc::capitalize(xlab),
           fill = Hmisc::capitalize(fill_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 = "Times", size = 20, face = "bold"),
            axis.text.x = element_text(size = 10),
            strip.text = element_text(size = 20),
            plot.title = element_text(hjust = 0.3))
    ggsave(p, file = savename, width = width, height = height)
  }
```

41 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()
  }
get_path <-
  function(
           path_str
           ){
    path <- stringr::str_extract(path_str, ".*(?=/)")
    return(path)
  }
get_filename <-
  function(
           path_str
           ){
    filename <- stringr::str_extract(path_str, "(?<=/)[^/]*$")
    return(filename)
  }
## ----------------------------------------------------------------------
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()
  }
```

42 File: png_auto_zoom.R

```
png_auto_zoom <-
  function(
           file,
           zoom.center.x = 0.5,
           zoom.center.y = 0.5,
           global.h = 5000,
           zoom.width = 1.5,
           internal.width = 0.1,
           zoom.window = 0.2
           ){
    ## readpng
    png <- EBImage::readImage(file)
    ## size ratio
    ratio.png <- ncol(png) / nrow(png)
    ## dev ratio
    ratio.dev <- zoom.width + ratio.png + internal.width
    ## create dev
    png(filename = paste0(file, ".zoom.png"),
        width = global.h * ratio.dev,
        height = global.h)
    ## use grid
    grid.newpage()
    ## ----------------------------------------------------------------------
    ## origin plot
    ## position and size
    ori.x <- (internal.width + zoom.width) / ratio.dev
```

```r
    ori.y <- 0
    ori.tmp.width <- ratio.png / ratio.dev
    ori.tmp.height <- 1
    ## create windows
    view.port <- fast.view(ori.x, ori.y, ori.tmp.width, ori.tmp.height)
    ## push
    grid.raster(png, vp = view.port)
    ## ----------------------------------------------------------------------
    ## zoom plot
    ## clip the plot
    ## ------------------
    ## row clip
    row.start <- nrow(png) * (1 - zoom.center.y - zoom.window / 2)
    row.end <- nrow(png) * (1 - zoom.center.y + zoom.window / 2)
    ## relative width of zoom size
    zoom.relative.width <- (1 / ratio.dev) * zoom.window * zoom.width
    ## col clip
    col.start <- ncol(png) * (zoom.center.x - zoom.relative.width / 2)
    col.end <- ncol(png) * (zoom.center.x + zoom.relative.width / 2)
    ## ------------------
    focus.png <- png[row.start:row.end, col.start:col.end, ]
    ## -------------------------------------
    ## position
    focus.x <- 0
    focus.y <- 0
    focus.tmp.width <- zoom.width / ratio.dev
    focus.tmp.height <- 1
    ## viewport
    view.port <- fast.view(focus.x, focus.y, focus.tmp.width, focus.tmp.height)
    ## push focus.png
    grid.raster(focus.png, vp = view.port)
    ## zoom rect
    grid.rect(gp = gpar(fill = NA, lwd = 100, col = "black"),
              vp = view.port)
    ## ----------------------------------------------------------------------
    dev.off()
  }
fast.view <-
  function(
           x, y, tmp.width, tmp.height,
           just = c("left", "bottom")
```

```
          ){
  view.port <- viewport(x = unit(x, "npc"),
                       y = unit(y, "npc"),
                       width = unit(tmp.width, "npc"),
                       height = unit(tmp.height, "npc"),
                       just = just)
  return(view.port)
}
```

43 File: prapare_inst_data.R

```
prapare_inst_data <-
  function(
           df,
           raw_path = ".",
           to_path = "~/MCnebula/inst/extdata"
           ){
    df <- df %>%
      dplyr::filter(tanimotoSimilarity >= 0.8) %>%
      dplyr::select(.id, rank, score, tanimotoSimilarity, file_name) %>%
      dplyr::slice(1:5)
    return(df)
  }
```

44 File: pubmed_query.R

```
pubmed_query <-
  function(
           key,
           mindate = 2000,
           maxdate = data.table::year(Sys.Date()),
           retmax = 1000
           ){
    cat("[INFO] step1: RISmed::EUtilsSummary\n")
    res <- RISmed::EUtilsSummary(query = key,
                                 mindate = mindate, maxdate = maxdate, retmax = retmax,
                                 type = "esearch", db = "pubmed", datetype = "ppdt")
    ## ---------------------------------------------------------------------
    cat("[INFO] step2: RISmed::EUtilsGet\n")
    res.get <- RISmed::EUtilsGet(res)
    return(res.get)
  }
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
## ----------------------------------------------------------------------
pubmed_query.meta <-
  function(
           res.get,
           affi = F,
           author = F,
           mesh = F,
           citation = F,
           metadata = T,
           IF.data = NULL
           ){
    ## cite times
    if(citation){
      cites = RISmed::Citations(res.get) %>%
        base_pubmed_query.meta("ref")
      return(cites)
    }
    if(affi){
      ## Affiliation, a vector
      extra.affi <- RISmed::Affiliation(res.get) %>%
        base_pubmed_query.meta("affiliation")
      return(extra.affi)
    }
    if(author){
      ## authors
      extra.author <- RISmed::Author(res.get) %>%
        base_pubmed_query.meta("name")
      return(extra.author)
    }
    ## meshs, a df
    if(mesh){
      extra.mesh <- RISmed::Mesh(res.get) %>%
        lapply(function(df){
                 if(is.data.frame(df))
                   return(df)
           }) %>%
      data.table::rbindlist(idcol = T) %>%
      dplyr::rename(id = .id)
      return(extra.mesh)
    }
    ## ----------------------------------------------------------------------
    metadata <- data.table::data.table(titles = RISmed::ArticleTitle(res.get),
                                       ## journal name
                                       journal = RISmed::ISOAbbreviation(res.get),
                                       ## institutions
                                       year_pubmed = RISmed::YearPubmed(res.get),
                                       id = RISmed::ArticleId(res.get))
    if(!is.null(IF.data)){
      IF.data <- readxl::read_xlsx(IF.data, skip = 1) %>%
        dplyr::select(`Full Journal Title`, `Impact Factor`)
      metadata <- merge(metadata, IF.data, by.x = "journal", by.y = "Full Journal Title",
                        all.x = T, sort = F)
    }
    metadata <- dplyr::as_tibble(metadata)
    ## ----------------------------------------------------------------------
    return(metadata)
  }
base_pubmed_query.meta <-
  function(
           list,
           index = "name"
           ){
    df <- list %>%
      lapply(unlist, use.names = F) %>%
      lapply(function(vec){
               data.table::data.table(name = vec)
           }) %>%
      data.table::rbindlist(idcol = T, fill = T) %>%
      dplyr::rename(id = .id) %>%
      dplyr::as_tibble()
    ## set colnames
    colnames(df)[2] <- index
    return(df)
  }
```

45 File: read_tsv.R

```
read_tsv <- function(path, ...){
  file <- data.table::fread(input=path, sep="\t", header=T, quote="\"", check.names=F, ...)
  return(file)
}
write_tsv <-
  function(x, filename){
    write.table(x, file = filename, sep = "\t", col.names = T, row.names = F, quote = F)
  }
```

46 File: roman_convert.R

```
roman_convert <-
  function(
           file,
           from = "Nimbus Roman",
           to = "Times New Roman"
           ){
    txt <- read_txt(file)
    txt <- dplyr::mutate(txt, V1 = gsub(from, to, V1))
    write.table(txt, file = file, sep = "", col.names = F, row.names = F, quote = F)
  }
```

47 File: select_app.R

```
select_app <-
  function(
           df,
           col
           ){
    df <- dplyr::select(df, all_of(col))
    return(df)
  }
```

48 File: set_export.no.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)
  }
```

49 File: set_instance_MCnebula.R

```
set_instance_MCnebula <-
  function(
          path = "~/MCnebula"
          ){
    inst_structure_set <- dplyr::filter(.MCn.structure_set, tanimotoSimilarity >= 0.6) %>%
      dplyr::select(.id, file_name, score, smiles, tanimotoSimilarity, structure_rank) %>%
      dplyr::slice(., sample(1:nrow(.), 700))
    ## ---------------------------------------------------------------------
    inst_formula_set <- dplyr::filter(.MCn.formula_set, .id %in% inst_structure_set$.id)
    ## ---------------------------------------------------------------------
    inst_ppcp_dataset <- .MCn.ppcp_dataset %>%
      .[names(.) %in% inst_structure_set$.id] %>%
      lapply(function(df){
               df <- dplyr::mutate(df, V1 = round(V1, 2))
               return(df)
          })
    dir <- getwd()
    setwd(path)
    usethis::use_data(inst_formula_set, overwrite = TRUE)
    usethis::use_data(inst_structure_set, overwrite = TRUE)
    usethis::use_data(inst_ppcp_dataset, overwrite = TRUE)
    setwd(dir)
    cat("Set instance files done\n")
  }
```

50 File: show_chem.R

```r
## ---------------------------------------------------------------------
## a function to fast show df
show_meta <-
  function(
           x,
           df = meta
           ){
    prefix <- stringr::str_extract(df[1,]$".id", "^[a-z]{1,100}(?=[0-9])")
    df <- dplyr::filter(df, .id == paste0(prefix, x)) %>%
      data.table::data.table()
    return(df)
  }
## ---------------------------------------------------------------------
getk <-
  function(
           x,
           col = "INCHIKEY"
           ){
    df <- show_meta(x)
    return(df[[col]])
  }
## ---------------------------------------------------------------------
show_stru <-
  function(
           df,
           key = c("smiles", "SMILES")
           ){
    key <- key[key %in% colnames(df)][1]
    smiles <- df[[key]]
    molconvert_structure(smiles)
  }
## ---------------------------------------------------------------------
auto <-
  function(
           id
           ){
    id <- as.character(substitute(id))
    show_meta(id) %>%
      show_stru()
  }
```

51 File: show_distribution.R

```r
show_distribution <-
  function(
           path = ".",
           level = NA
           ){
    set <- list.files(path, pattern = "(.*)[0-9]{1-5}$", full.names = T)
    list <- pbapply::pblapply(set, read_tsv)
    names(list) <- set
    Level = "Level"
    df <- data.table::rbindlist(list, idcol = T, fill = T)
    if(is.na(level) == F){
      df <- dplyr::filter(df, Level %in% all_of(level))
    }
    df <- dplyr::mutate(df, .id = stringr::str_extract(.id, "(?<=/)[a-z]{1,10}[0-9]{1,5}$"))
    df <- dplyr::as_tibble(df)
    return(df)
  }
```

52 File: show_palette.R

```r
show_palette <-
  function(
           palette,
           width = 2,
           height = 10,
           font_size = 5,
           ylab = "Re-ID",
           xlab = "Color",
           title = "",
           re_order = T
           ){
    df <- data.table::data.table(name = names(palette), color = unname(palette)) %>%
      dplyr::mutate(name = stringr::str_wrap(name, width = 25))
    if(!re_order){
      df <- dplyr::mutate(df, name = factor(name, levels = name))
    }
    ## ---------------------------------------------------------------------
    p <- ggplot(df) +
      geom_tile(aes(x = "color", y = name,
                    fill = name),
                width = 1, height = 1, size = 1, color = "black") +
      labs(x = xlab, y = ylab) +
      ggtitle(title) +
      guides(fill = "none") +
      scale_fill_manual(values = palette) +
      theme_minimal() +
      theme(text = element_text(size = font_size, face = "bold", family = "Times"),
            title = element_text(hjust = -2),
            axis.text.x = element_blank(),
            panel.grid = element_blank())
    ggsave(p, file = "tmp.svg", width = width, height = height)
    return()
  }
mutate_show_palette <-
  function(
           palette,
           width = 5,
           height = 10,
           font_size = 5,
           ylab = "Re-ID",
           xlab = "Color",
           legend.position = "right",
           legend.key.height = unit(5, "cm"),
           legend.key.width = unit(0.5, "cm"),
           fill_lab = "",
           title = "",
           re_order = T
           ){
    df <- data.table::data.table(name = names(palette), color = unname(palette)) %>%
      dplyr::mutate(name = stringr::str_wrap(name, width = 25))
    if(!re_order){
      df <- dplyr::mutate(df, name = factor(name, levels = name))
    }
    ## ----------------------------------------------------------------------
    p <- ggplot(df) +
      geom_tile(aes(x = "color", y = name,
                    fill = name),
                width = 1, height = 1, size = 1, color = "black") +
      labs(x = xlab, y = ylab, fill = fill_lab) +
      ggtitle(title) +
      scale_fill_manual(values = palette) +
      theme_minimal() +
      theme(text = element_text(size = font_size, face = "bold", family = "Times"),
            title = element_text(hjust = -2),
            axis.text.x = element_blank(),
            legend.position = legend.position,
            legend.key.height = legend.key.height,
            legend.key.width = legend.key.width,
            panel.grid = element_blank())
    p <- ggpubr::get_legend(p)
    p <- ggpubr::as_ggplot(p)
    ggsave(p, file = "tmp.svg", width = width, height = height)
    return()
  }
```

53 File: simulate_gnps_quant.R

```
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) <- colnames(meta) %>%
      .meta_find_and_sort(., c(id, mz, "rt", "sample")) %>%
      mapply_rename_col(., c(simu_id, simu_mz, simu_rt, simu_quant),
                        colnames(meta))
    if(return_df)
```

```r
    return(meta)
  write.table(meta, file = file, sep = ",", row.names = F, col.names = T, quote = F)
}
```

54 File: spectrum_add_noise.R

```r
## for .mgf data
spectrum_add_noise <-
  function(
           list,
           cl = 8,
           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){
      empty <- !vapply(list, is.data.frame, logical(1), USE.NAMES = F)
      empty <- unique(names(list[empty]))
      list <- list[!names(list) %in% empty]
    }
    return(list)
  }
```

```r
mass_process_level_1 <-
  function(df, ...){}
mass_process_level_2 <-
  function(
    df,
    mass_shift = T,
    ...
    ){
    list <- separate_peak_info(df, ...)
    if(mass_shift == F)
      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 <- slice(df, peak_row)
    peak_info <- separate(peak_info, col = "V1", into = c("mass", "inte"), sep = sep)
    if(only_peak_info == T)
      return(peak_info)
    list <- list(slice(df, 1:(min(peak_row) - 1)),
      peak_info,
      slice(df, (max(peak_row) + 1):nrow(df))
    )
    return(list)
  }
```

55 File: stat_accuracy.R

```r
stat_accuracy <-
  function(
           ## a list contain .id
           dominant_list,
           ## the col are .id, inchikey2D
           structure,
           ## the col are .id, standard
           meta,
           return_id_stat = F
           ){
    id_stat <- lapply(dominant_list, merge, y = structure,
                      by = ".id",
                      all.x = T)
    id_stat <- lapply(id_stat, merge, y = meta,
                      by = ".id", all.x = T)
    id_stat <- lapply(id_stat, dplyr::mutate,
                      evaluate = ifelse(inchikey2D == standard, "true", "false"))
    if(return_id_stat == T)
      return(id_stat)
    table <- lapply(id_stat, table_app) %>%
      data.table::rbindlist(idcol = T, fill = T) %>%
      dplyr::rename(classification = .id)
    return(table)
  }
stat_topn_candidates_accuracy <-
  function(
           nebula_name,
           path = "mcnebula_results/candidates",
           meta,
           return_id_stat = F
           ){
    file_set <- lapply(nebula_name, mutate_list_files, path = path) %>%
      unlist()
    id_stat <- lapply(file_set, read_tsv) %>%
      lapply(dplyr::select, .id, inchikey2D, structure_rank) %>%
      lapply(merge, meta, by = ".id", all.x = T) %>%
      lapply(dplyr::mutate, evaluate = ifelse(inchikey2D == standard, "true", "false")) %>%
      lapply(dplyr::arrange, .id, desc(evaluate)) %>%
      lapply(dplyr::distinct, .id, .keep_all = T)
    names(id_stat) <- nebula_name
    if(return_id_stat == T)
      return(id_stat)
    table <- lapply(id_stat, table_app) %>%
      data.table::rbindlist(idcol = T) %>%
      dplyr::rename(classification = .id)
    return(table)
  }
mutate_list_files <-
  function(
           pattern,
           path
           ){
    files <- list.files(path, pattern, full.names = T)
    return(files)
  }
```

56 File: stat_results_class.R

```
stat_results_class <-
  function(
           ## only .id is needed
           df,
           standard,
           path = ".",
           class_cutoff = 4
           ){
    ## -------------------------------------
    if(is.data.frame(df) == F)
      return()
    db <- dplyr::filter(.MCn.class_tree_data, hierarchy >= class_cutoff)
    ## -------------------------------------
    ## for name to get id
    db_id <- lapply(db$id, c)
    names(db_id) <- db$name
    ## for id to get parentId
    db_parent <- lapply(db$parentId, c)
    names(db_parent) <- db$id
    ## for id to get name
    db_name <- lapply(db$name, c)
    names(db_name) <- db$id
    ## -------------------------------------
    set <- get_parent_class(standard,
                            db_id,
                            db_parent,
                            db_name)
    ## ------------------------------------
    cat("stat:", standard, "\n")
    id_set <- df[[".id"]]
    list <- pbapply::pblapply(id_set, base_stat_results_class,
                      standard = standard,
                      set = set,
                      path = path)
    df <- data.table::rbindlist(list)
    return(df)
  }
base_stat_results_class <-
  function(
           id,
           set,
           path = ".",
           standard
           ){
    check <- try(class <- read_tsv(paste0(path, "/", id)), silent = T)
    if(class(check)[1] == "try-error"){
      stat <- data.table::data.table(id = id, evaluate = NA)
      return(stat)
    }
    if(standard %in% class[["Classification"]]){
      stat <- data.table::data.table(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.table::data.table(id = id, evaluate = evaluate)
    }
    return(stat)
  }
```

57 File: struc_match_in_df.R

```
struc_match_in_df <-
  function(
           df,
           pattern,
           id_col = "id",
           smiles_col = "SMILES"
           ){
    cat("\n## match:", pattern, "\n")
    list <- pbapply::pblapply(df[[smiles_col]], base_pattern_chem,
                              pattern = pattern)
    df <- data.table::data.table(.id = df[[id_col]],
                                 evaluate = c(unlist(list)))
    return(df)
  }
base_pattern_chem <-
  function(
           smiles,
           pattern
           ){
    mol <- rcdk::parse.smiles(smiles)
    check <- try(match <- rcdk::matches(pattern, mol, return.matches = F), silent = T)
    if(class(check) == "try-error")
      return(c(X = "is.na"))
    return(match)
  }
```

58 File: table_app.R

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

59 File: test_rerank_method.R

```r
test_rerank_method <-
  function(
           ## nebula_name
           name,
           ## involve col of .id, standard
           meta,
           top_n = 10,
           ...
           ){
    # rerank method
    test_structure <- nebula_re_rank(nebula_name = name, top_n = top_n, ...)
    ## gather results with reference data
    stat <- merge(test_structure[, c(".id", "inchikey2D")], meta,
                          by = ".id", all.x = T) %>%
      dplyr::mutate(evaluate = ifelse(inchikey2D == standard, "true", "false")) %>%
      table_app()
    cat("## nebula_name:", name, "\n")
    print(stat)
    return(stat)
  }
```

60 File: visualize_facet_pca.R

```r
visualize_facet_pca <-
  function(
           df,
           palette,
           metadata,
           savename = "pca_facet.svg",
           anno_adjust = 3/4
           ){
    ## ---------------------------------------------------------------------
    ## get PC importance df
    df <- df %>%
      dplyr::filter(facet_row != "extra") %>%
      merge(metadata, by = "sample", all.x = T) %>%
      dplyr::as_tibble()
    ## --------------------------------------
    annotation <- df %>%
      dplyr::distinct(im_PC1, im_PC2, legend_PC1, legend_PC2,
                      anno_x, anno_y,
                      facet_col, facet_row) %>%
      dplyr::mutate(anno_x = min(anno_x) * anno_adjust,
                    anno_y = max(anno_y) * anno_adjust)
    ## ----------------------------------------------------------------------
    ## drawing part
    p <- ggplot(df, aes(x = PC1, y = PC2, fill = group)) +
      geom_point(alpha = 0.8, size = 3, shape = 21, stroke = 0.1) +
      ## draw confidence ellipse
      stat_ellipse(aes(color = group), level = 0.95) +
      scale_color_manual(values = palette) +
      scale_fill_manual(values = palette) +
      ## exlude repeat legend
      guides(color =  "none") +
      ## for PC1
      geom_text(data = annotation, aes(x = anno_x * 1.4, y = anno_y * 1.4, label = legend_PC1),
                hjust = 0, color = "black",
                fontface = "bold", alpha = 0.6,
                size = 2, inherit.aes = FALSE,
                family = "Times") +
      ## for PC2
      geom_text(data = annotation, aes(x = anno_x * 1.4, y = anno_y * (18/22) * 1.4, label = legend_PC2
                hjust = 0, color = "black", fontface = "bold",
                alpha = 0.6, size = 2, inherit.aes = FALSE,
                family = "Times") +
      labs(y = "PC2", x = "PC1", fill = "Group") +
      ## facet into multiple panel
      facet_grid(Hmisc::capitalize(facet_row) ~ Hmisc::capitalize(facet_col)) +
      theme(legend.position = "right",
            text = element_text(family = "Times"))
    ggsave(p, file = savename, width = 11, height = 6.5)
    return(p)
  }
```