Supporting Code for Analysis and Visualization

Tolga Tabanli

2025-02-15

Contents

Readme

This document is an extensive summary of the main code snippets that were used for the processing of RNAseq count data and the following plot generation procedure. Many code sections were omitted that were used for exploration, trial-and-errors, accession mapping, and of course the Python code. Python code can be accessed in GitHub Page. Plots used in the IDP Selection were also omitted for conciseness, and can again be accessed in the GitHub page.

IDP Decision

AIUPred

The disorder scores were calculated using the AIUPred's Python package. Basic plots to explore the score distributions were hence also done by matplotlib's pyplot. So instead, I will continue with decision/filtering.

Accession ID Conversion

The database used for input to AIUPred was UniProt. However, the counts data for the DESeq analysis uses proteins with Ensembl IDs. This is why I used biomaRt library to do this conversion.

```r
library(tidyverse)
library(biomaRt)

# for recreating biomart
ensembl_up_mapping <- useEnsembl(biomart = "genes",
                                 dataset = "scerevisiae_gene_ensembl") %>%
  getBM(attributes = c("ensembl_gene_id", "uniprotswissprot"),
        mart = .)

# use prepared mapping from biomart
ensembl_up_mapping <- readRDS("IDP decisions/ensembl_up_mapping.Rds")
```

Finally, the disorder modes of the proteins were read from the TSV file that was generated through python. Proteins were then filtered around the cut-off of 0.6.

```r
aiupred_all_proteins_mode <- read_tsv(here::here("aiupred/aiupred_modes_of_proteins.tsv"))

aiupred_all_proteins_mean %>%
  inner_join(ensembl_up_mapping, join_by(uniprot == uniprotswissprot)) %>%
  dplyr::select(ensembl_gene_id, mean) %>%
  dplyr::rename("mean_disorder" = "mean") %>%
  write_tsv("IDP decisions/all_proteins_from_aiupred_with_mean.tsv")
```

AlphaFold

In AlphaFold's .cif files, the local pLDDTs are given for each residue. These were extracted using R's extensive regex capabilities. To optimize the process, certain flags/patterns were used to exactly locate the required segment.

```r
dir <- here::here("alphafold cifs")
files <- list.files(dir)

local_plddts <- tibble(uniprot = character(), scores = list())
start <- "_ma_qa_metric_local.ordinal_id"
end <- "#"
score_pattern <- "\\b\\d+\\.\\d+\\b"  # regex for floating point numbers
```

The file is first scanned until a start flag is encountered (found_start) and then the pattern is searched until an end flag (end).

```r
i <- 1
for (file in files) {
  print(i) # for progress

  accession <- str_split_1(file, "-")[2] # get accession from the file name
  file_connection <- file(paste0(dir, "/", file), open = "r")
  found_start <- FALSE
  scores <- c()
  while(TRUE) {
    line <- readLines(file_connection, n = 1)
    if(!found_start) {
      if (line == start) found_start <- TRUE
    } else {
      if (line == end) {
        break
      }
      # extract the decimal number and concatenate to the scores
      scores <- c(scores, as.numeric(regmatches(line, regexpr(score_pattern, line))))
    }
  }
  local_plddts <- local_plddts %>% add_row(uniprot = accession, scores = list(scores))
  i = i + 1
}
```

Next, the object was stored in an Rds for later access.

```r
saveRDS(local_plddts, "IDP decisions/local_plddts.Rds") # Store

local_plddts <- readRDS("IDP decisions/local_plddts.Rds") # Load
```

Accession ID Conversion

Next, the accessions of the proteins are converted to Ensembl Gene IDs for compatibility with count data by using a mapping that has been created by "biomaRt".

```r
ensembl_up_mapping <- readRDS("IDP decisions/ensembl_up_mapping.Rds") # through biomaRt

local_plddts %>%
  group_by(uniprot) %>%
  mutate(mean_plddt = mean(unlist(scores))) %>%
  ungroup() %>%
  inner_join(ensembl_up_mapping, join_by(uniprot == uniprotswissprot)) %>%
  dplyr::select(ensembl_gene_id, mean_plddt) %>%
  write_tsv("IDP decisions/all_proteins_from_alphafold_with_mean.tsv")
```

Distributions

For distribution of all local pLDDTs, the following code was used. The data frame of proteins residue scores are first reduced to a "pool" of all scores, which is then stored in csv file. By reading this csv file, the data is also automatically enframed, which can be given to ggplot as data. Saving steps are commented out to avoid side effects. The extensive design parameters had to be used for obtaining similarity to Python's Matplotlib design.

```r
all_scores <- local_plddts %>% select(scores) %>% pull() %>% reduce(c)
# write.csv(all_scores, "alphafold/local_plddts.csv")
all_scores <- read.csv("alphafold/local_plddts.csv") %>%
  `colnames<-`(c("row_names", "locals"))
# saveRDS(all_scores, "IDP decisions/all_local_plddt_scores.Rds")

# Hist of all local plddt scores
ggplot(all_scores, aes(locals)) +
  geom_histogram(bins = 100, fill = "#1f77b4") +
  xlab("Local pLDDT") +
  ylab("") +
  labs(title = "Distribution of all local pLDDTs from AlphaFold") +
  theme(axis.line = element_line(colour = "black"),
        axis.text = element_text(size=14),
        axis.title = element_text(size=14),
        title = element_text(size=12),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        plot.margin = margin(t = 20),
        aspect.ratio = 0.75) +
  scale_x_continuous(breaks = c(0,20,40,60,80,100)) +
  scale_y_continuous(expand = c(0,0,0.1,0)) +
  theme(plot.title = element_text(hjust = 0.5))
```

It was a harder challenge to find the modes of the proteins. I first needed to define a function to create "intrinsic histograms" for each protein. I had used a similar method in python with scipy's stats. (Hence the name)

```
find_mode_python <- function(scores, bins) {
  bin_edges <- seq(min(scores), max(scores), length.out = bins + 1)
  bin_counts <- hist(scores, breaks = bin_edges, plot = FALSE)$counts
  bin_centers <- (bin_edges[-1] + bin_edges[-length(bin_edges)]) / 2
  mode_value <- bin_centers[which.max(bin_counts)]
  return(mode_value)
}

# Calculate and save the modes
with_modes <- local_plddts %>% mutate(mode_plddt =
                          sapply(scores, find_mode_python, bins = 20)) %>%
  dplyr::select(uniprot, mode_plddt)
```

And the plot was created using the same design idea:

```
ggplot(with_modes, aes(x = mode_plddt)) +
  geom_histogram(bins = 50, fill = "#1f77b4") +
  xlab("Modes of proteins (pLDDT)") +
  ylab("") +
  labs(title = "Distribution of mode pLDDTs") +
  theme(axis.line = element_line(colour = "black"),
        axis.text = element_text(size=14),
        axis.title = element_text(size=14),
        title = element_text(size=12),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        panel.border = element_rect(color = "black", fill = NA, linewidth = 1),
        plot.margin = margin(t = 30),
        aspect.ratio = 0.75) +
  scale_x_continuous(breaks = c(0,20,40,60,80,100), limits = c(0,100)) +
  scale_y_continuous(expand = c(0,0,0.1,0)) +
  theme(plot.title = element_text(hjust = 0.5))
```

Finally, the IDPs are decided based on a cut-off of 60.

```
# first convert accessions:
ensembl_up_mapping %>%
  inner_join(with_modes, join_by(uniprotswissprot == uniprot)) %>%
  dplyr::select(ensembl_gene_id, mode_plddt) %>%
```

```r
  write_tsv("IDP decisions/all_proteins_from_alphafold_with_mode.tsv")


# Filter by threshold 60
read_tsv("IDP decisions/all_proteins_from_alphafold_with_mode.tsv") %>%
  filter(mode_plddt < 60) %>%
  write_tsv("IDP decisions/idps_from_alphafold_mode.tsv")
```

DESeq

DESeq Call with IDPs

This chapter summarizes how the experiment samples were mapped and the DESeq was called accordingly. It also includes the creation of volcano plots and the highlighting of the inferred intrinsically disordered proteins.

Libraries and Sample Preparations

```r
library(DESeq2)
library(tidyverse)
library(ggpubr)
library(grid)

colData <- read_tsv("sample_mapping.tsv", col_names = T) %>%
  mutate(across(everything(), as.factor)) %>%
  column_to_rownames(var = "sample")

heatshock_counts <- read_tsv(here::here("complex_yeast_heatshock.tsv"), col_names = T) %>%
  column_to_rownames(var = "gene_id") %>%
  relocate(rownames(colData))

knockouts <- colData %>%
  select(knockout) %>%
  distinct() %>%
  pull() %>%
  relevel("Wildtype")

idps <- readRDS("IDP decisions/commons_modes.Rds")
deseq_results <- list()
```

Function Definitions for Multiple DESeq and Plotting

Firstly, the function that calls DESeq with given parameters is defined. This function takes the genotype, temperature and time information and carries out DESeq with reference to temperature = 37 of the respective genotype. The reference-25 version of the function was defined very similarly with the difference of the reference being 25 and time = 0 for the corresponding time. As the report presents and discusses results from reference-37, only the code for this is shown here.

Importantly, this function uses a **global assignment** to store DESeq results in a list outside of the function's scope. This way was preferred, because the function's return value is used to create the grid plot.

```r
multiple_deseq_37 <- function(countData, colData, knock, temp, t) {
  subColData <- colData %>%
    filter(knockout == knock & time == t) %>%
    filter(temperature == 37 | temperature == temp) # REFERENCE AS 37

  subcounts <- heatshock_counts %>% select(rownames(subColData))

  dds <- DESeqDataSetFromMatrix(countData = subcounts,
                                colData = subColData,
                                design = ~ temperature)
  deseq <- DESeq(dds, quiet = TRUE)
  res <- results(deseq)

  deseq_results[[paste0(knock,"_",temp,"_",t)]] <<- res

  return(plot_res(res, knock, temp, t))
}
```

The following function was used for creating an aesthetic grid plot that brings together 8 different volcano plots and adding significance thresholds.

```
plot_res <- function(res, knock, temp, t) {
  res <- res %>%
    as.data.frame() %>%
    drop_na()
  significant_idps <- res$padj < 0.05 &
    abs(res$log2FoldChange) > 0.6 &
    rownames(res) %in% idps

  grob <- grobTree(textGrob(
    paste("Significant IDP ratio:\n",
          round(sum(significant_idps) / length(idps), digits = 2)),
    x=0.5,  y=0.6))

  p <- ggplot(data = res,
              aes(x = log2FoldChange, y = -log10(pvalue))) +
    scale_color_manual(values = c("#0400ff", "#747880"),
                       labels = c("Significant IDPs", "Others"),
                       breaks = c(TRUE, FALSE)) +
    geom_point(data = res %>% filter(!significant_idps), aes(colour = FALSE)) +
    geom_point(data = res %>% filter(significant_idps), aes(colour = TRUE)) +
    geom_vline(xintercept = 0.6, linetype="dashed") +
    geom_vline(xintercept = -0.6, linetype="dashed") +
    geom_hline(yintercept = -log10(0.05), linetype="dashed") +
    annotation_custom(grob) +
    labs(title = paste(knock, temp, "C, t =", t, "min"),
         color = "Significance")
  return(p)
}
```

Multiple-DESeq Call

To analyze data, first the parameter space was created for the comparisons.

```
times <- c(10, 30)
temps <- 42
knockouts <- colData %>%
  dplyr::select(knockout) %>%
  unique() %>%
  pull()

combs <- expand.grid(time = times, temperature = temps, knockout = knockouts)
plots <- list()
```

Next, we generate the DESeq results. While the actual results are being stored in a global list, the individual plots are stored in "plots". These plots were then plotted using the plot_res function for labeling and highlighting IDP proteins.

```
for (row in 1:nrow(combs)) {
  plots[[row]] <- multiple_deseq_37(heatshock_counts,
                                   colData,
                                   as.character(combs[row,"genotype"]),
                                   combs[row,"temperature"],
                                   combs[row, "time"])
}

for (ind in seq_along(deseq_results)) {
  plots[[ind]] <- plot_res(deseq_results[[ind]],
                           combs[ind, "genotype"],
                           combs[ind, "temperature"],
                           combs[ind, "time"])
}
```

Then, the plot was saved using ggarrange for fusion:

```
ggarrange(plotlist = plots, ncol = 2, nrow = 4, common.legend = T)
ggsave("significant plots/volcanoes_reference37.png", device = "png", height =15, width = 10)
saveRDS(deseq_results, "deseq_reference37.Rds")
```

Comparison of Differential Expressions

This chapter deals with how the results of DESeq analysis was visualized and used to graphically compare the differences between IDP expression and the overall expression patterns. I have used bar plots to distinguish the number of up and down-regulated genes, heatmaps for comparison of expression patterns and clusters in different genotypes and lastly gene enrichment analysis to find out the functions of the IDPs implicated in heat shock and possible broken down in knock-down and knock-out.

Libraries and Preparation

The used libraries give an overview of this chapter. IDPs were read from the RDS data. The relevant DESeq results from DESeq pipeline were read in and the label "ref" was used for saves and plots. The results were lastly filtered for significance.

```
library(tidyverse)
library(UpSetR)
library(gprofiler2)
library(ggpubr)
library(janitor)

idps <- readRDS("IDP decisions/commons_modes.Rds")

deseq_results <- readRDS("deseq_reference37.Rds") %>%
  map(as.data.frame)
```

```r
ref <- 37

deseq_sig <- deseq_results %>%
  map(drop_na) %>%
  map(arrange, padj) %>%
  map(~ dplyr::select(.x, log2FoldChange, padj)) %>%
  map(~ filter(.x, padj < 0.05)) %>%
  map(~ filter(.x, abs(log2FoldChange) > 0.6)) # for lfc filtering
```

Up or Down?

To be able follow what the ratio of up- and down-regulated genes under different conditions was, the filtered DESeq results were labeled according to direction of regulation:

```r
expr_direction <- deseq_sig %>%
  map(~ mutate(.x, direction = case_when(
    log2FoldChange < 0 ~ "Down",
    log2FoldChange > 0 ~ "Up"
  ))) %>%
  map(rownames_to_column) %>%
  map(~ mutate(.x, idp = case_when(
    rowname %in% idps ~ TRUE,
    !(rowname %in% idps) ~ FALSE
  )))
```

To compare IDPs' expression behaviour with the background (whole genome), I needed to use a function to convert the up/down labels to counts both for whole genome and for IDPs, a kind of tabularization. In short, what this does is 1) summarise counts for Up/Downregulated genes, 2) repeat for IDPs, and 3) return a new data frame consisting of these four groups.

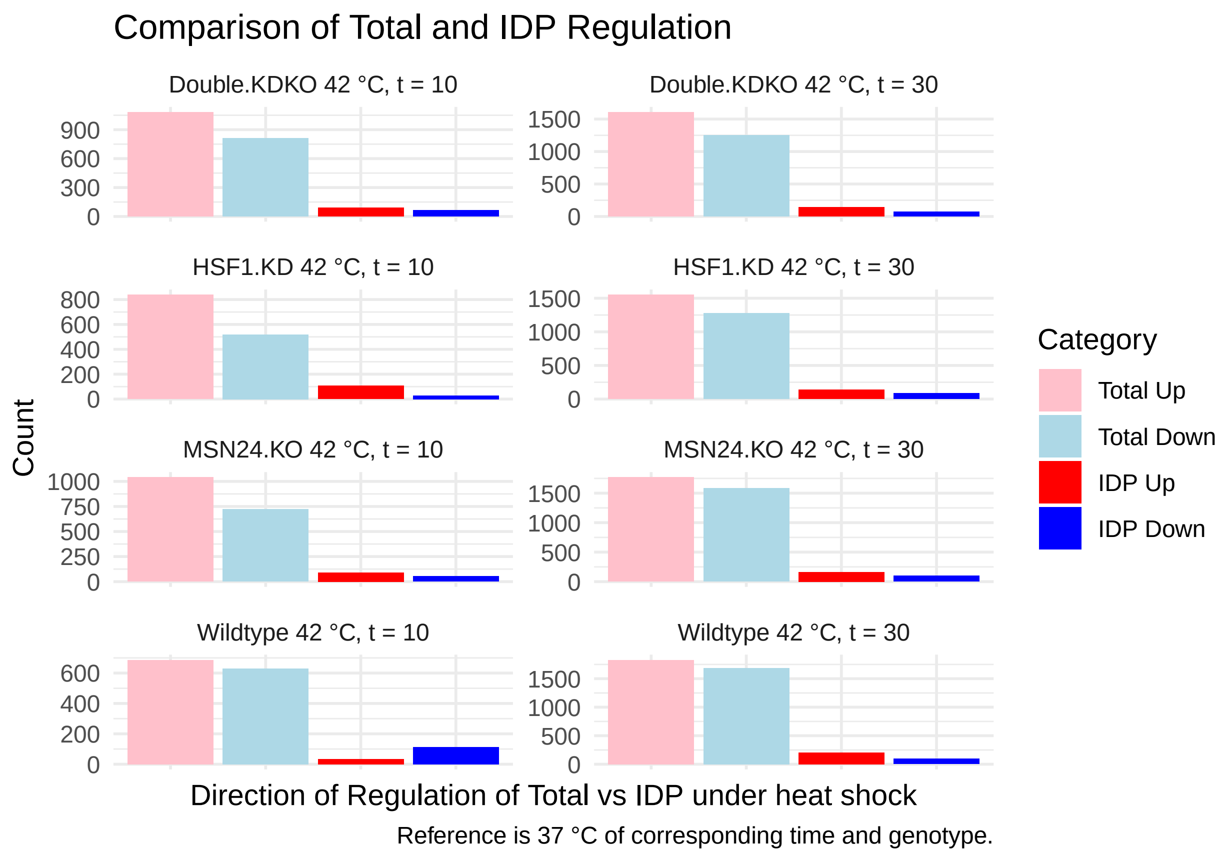
```r
convert_to_counts <- function(df) {
  n_total <- df %>%
    group_by(direction) %>%
    summarise(count = n(), .groups = "drop") %>%
    mutate(group = paste0("Total ", direction))

  n_idp <- df %>%
    filter(idp == TRUE) %>%
    group_by(direction) %>%
    summarise(count = n(), .groups = "drop") %>%
    mutate(group = paste0("IDP ", direction))

  bind_rows(n_total, n_idp)
}
```

Then, the data with information on regulation direction was passed through tabularization function described above and then plotted on a bar plot.

```r
expr_direction %>%
  map_dfr(convert_to_counts, .id = "dataset") %>%
  # filter(str_starts(dataset, "Wildtype")) %>% # for creating Wild-type only
  mutate(group = factor(group, levels = c("Total Up", "Total Down",
                                          "IDP Up", "IDP Down"))) %>%
  ggplot(aes(x = group, y = count, fill = group)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ dataset, ncol = 2, scales = "free",
            labeller = labeller(dataset = function(x) {
              gsub("(\\w+?)_(\\d+?)_(\\d+?)", "\\1 \\2 °C, t = \\3", x)
            })) +
  labs(
    title = "Comparison of Total and IDP Regulation",
    x = "Direction of Regulation of Total vs IDP under heat shock",
    caption = paste("Reference is", ref, "°C of corresponding time and genotype."),
    y = "Count",
    fill = "Category"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_blank()) +
  scale_fill_manual(values = c(
    "IDP Up" = "red",
    "Total Up" = "pink",
    "IDP Down" = "blue",
    "Total Down" = "lightblue"
  ))
```

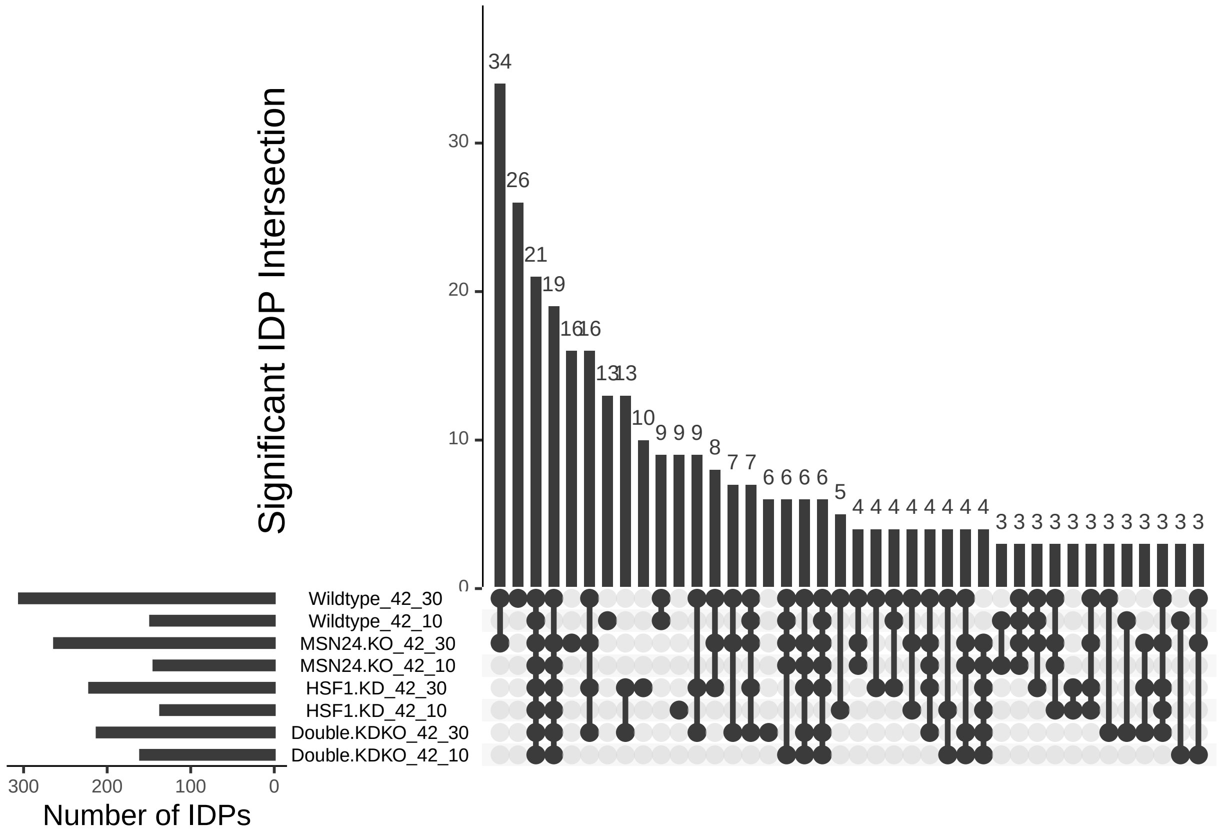


UpSet

To see if there is an overlap between the significant IDPs from the results of different conditions, I used UpSet plot. The row names are the gene names. Text scale parameters indicate the font sizes used in the plot.

```
deseq_sig_idps <- deseq_sig %>%
  map(row.names) %>%
  map(intersect, idps)

upset(fromList(deseq_sig_idps), sets = names(deseq_sig_idps),
      keep.order = T,
      nsets = length(deseq_sig_idps),
      point.size = 3, line.size = 1,
      mainbar.y.label = "Significant IDP Intersection",
      sets.x.label = "Number of IDPs",
      order.by = c("freq"),
      text.scale = c(1.7, 1, 1.3, 1, 1, 1.3))
```



A similar UpSet plot was created for the intersection on only wild-type. Here is given only the set for the input to upset():

```
updown_wt <- expr_direction %>%
  keep(str_starts(names(.), "Wildtype")) %>%
  map(filter, idp) %>%
  map(~ mutate(.x, direction = factor(.x$direction, levels = c("Up", "Down")))) %>%
  map(~ split(.x$rowname, .x$direction)) %>%
  flatten() %>%
  set_names(c("Wildtype_42_10 Up", "Wildtype_42_10 Down",
              "Wildtype_42_30 Up", "Wildtype_42_30 Down"))
```

Heatmaps

Heatmaps were used the compare a possible pattern of expression that was hypothesized to change between all genes and IDPs. The data preparation step included:

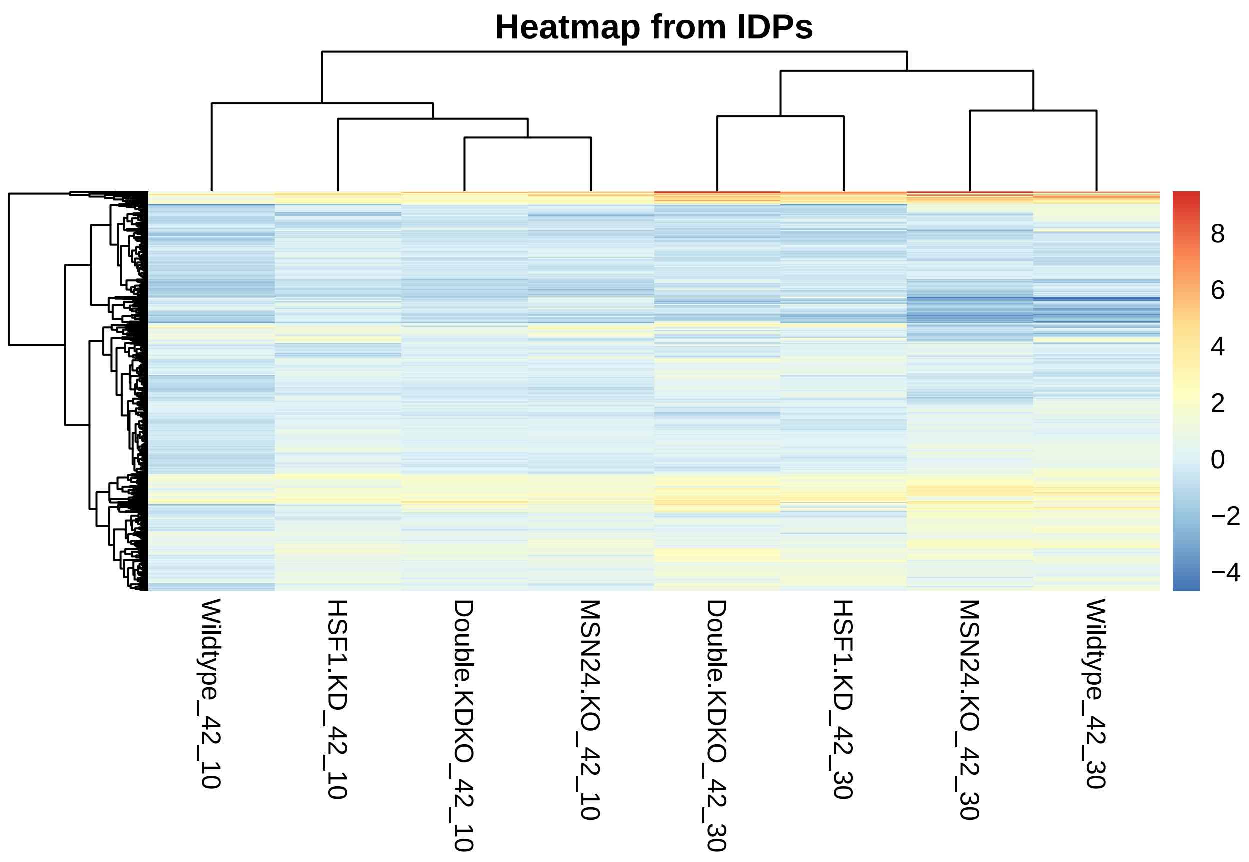
```r
library(pheatmap)
library(tidyverse)

# PREPARE
idps <- readRDS("IDP decisions/commons_modes.Rds")
heat_matrix <- data.frame(rowname = idps) # For IDP step
deseq <- readRDS("deseq_reference37.Rds") %>%
  map(data.frame) %>%
  map(rownames_to_column) %>%
  map(select, rowname, log2FoldChange)
```

Next, matrices of log2 fold changes were created. For IDP case, an empty matrix with row names as IDPs were used as join partner. As heatmap cannot accept NA values in the matrix, NA-rows in both matrices were dropped. (The NA values are naturally generated in DESeq in cases such as zero or low counts, extreme variability or other statistical test mechanisms.)

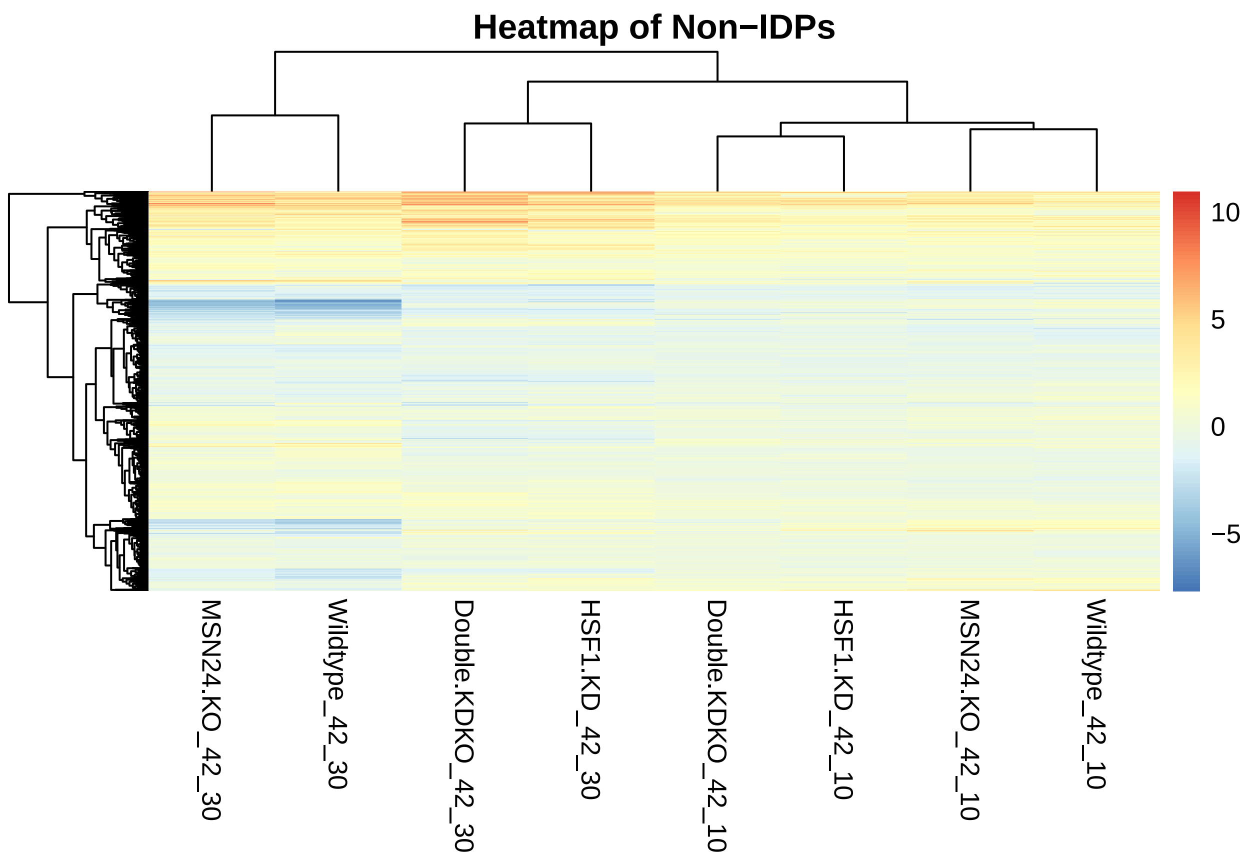
```r
# Heatmap of IDPs
joined_matrix <- deseq %>%
  imap(~ setNames(.x, gsub("log2FoldChange", .y, names(.x)))) %>%
  map(right_join, heat_matrix) %>% # joins here with IDPs
  purrr::reduce(full_join) %>%
  drop_na() %>%
  column_to_rownames() %>%
  as.matrix()

pheatmap(joined_matrix, show_rownames = F,
         main = "Heatmap from IDPs")
```



```r
# Heatmap of Non-IDPs
joined_matrix <- deseq %>%
  map(filter, !(rowname %in% idps)) %>%
  imap(~ setNames(.x, gsub("log2FoldChange", .y, names(.x)))) %>%
  purrr::reduce(full_join) %>%
  drop_na() %>%
  column_to_rownames() %>%
  as.matrix()

pheatmap(joined_matrix, show_rownames = F,
         main = "Heatmap of Non-IDPs")
```



Additionally, I wanted to compare the "real" IDPs from DisProt, which included 212 proteins. The reason why these proteins dod not directly form the basis of the exploration is given in the Discussion of the Report.

```r
# Heatmap of DisProt Proteins
## accession mapping
biomart_mapping <- read_tsv("IDP decisions/biomart_mapping.tsv")

## read the disprot proteins
disprot <- read_tsv("DisProt/DisProt release_2024_12.tsv") %>%
  select(acc) %>%
  distinct() %>%
  inner_join(biomart_mapping, by = join_by(acc == uniprotswissprot)) %>%
  pull(ensembl_gene_id)

heat_matrix_disprot <- data.frame(rowname = disprot)

joined_matrix <- deseq %>%
  imap(~ setNames(.x, gsub("log2FoldChange", .y, names(.x)))) %>%
  map(right_join, heat_matrix_disprot) %>%
  purrr::reduce(full_join) %>%
  drop_na() %>%
  column_to_rownames() %>%
  as.matrix()
```

It should be noted that the heatmaps did not take *significance* into account, meaning that they were not filtered according to p-value or LFC and the color labels mirror all the LFC values of genes that had a non-NA value in each condition.

Functional Enrichment Analysis

Functional enrichment analysis was done with help of g:Profiler interface gprofiler2. Dotplot was implemented using ggplot to facilitate query management, as a direct conversion of gost() results to other packages such enrich was not successful (multiquery requests would cause problems, another option would be to vertically add results of single queries).

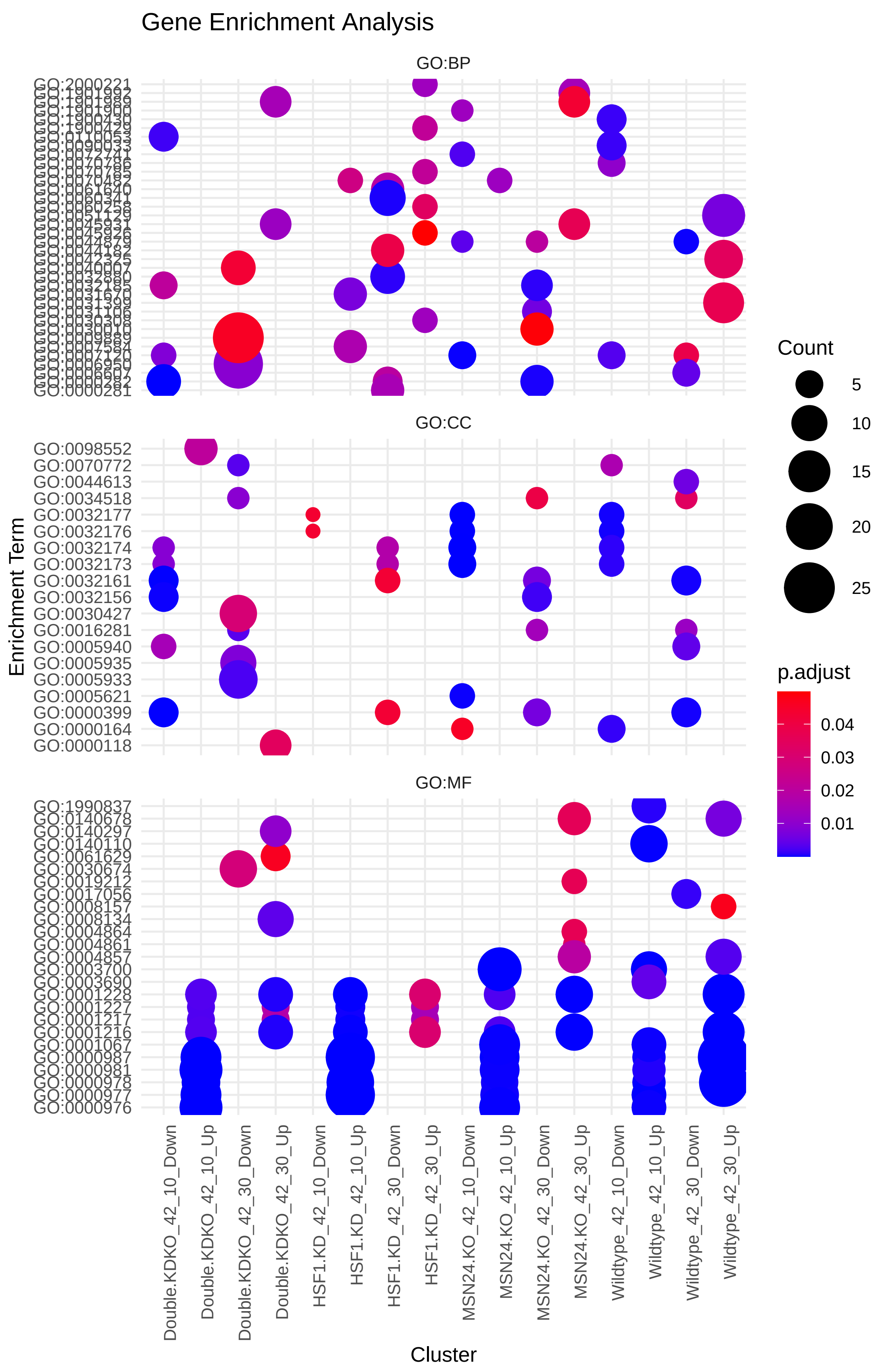
First, a function was defined the underlying dotplot data which are the enrichment data for up and down regulation (reg) in each condition (subdeseq), resulting in 16 sets.

```r
dotplot_updown <- function(sub_deseq, reg) {
  xdirection <- sub_deseq %>%
    filter(direction == reg & idp) %>%
    pull(rowname)
  multi_gp = gost(query = xdirection,
                  organism = "scerevisiae",
                  multi_query = FALSE, evcodes = TRUE)
  gp_mod = multi_gp$result[,c("query", "source", "term_id",
                              "term_name", "p_value", "query_size",
                              "intersection_size", "term_size",
                              "effective_domain_size", "intersection")]
  gp_mod$GeneRatio = gp_mod$intersection_size / gp_mod$query_size
  gp_mod$BgRatio = paste0(gp_mod$term_size, "/", gp_mod$effective_domain_size)
  names(gp_mod) = c("Cluster", "Category", "ID", "Description", "p.adjust",
                    "query_size", "Count", "term_size", "effective_domain_size",
                    "geneID", "GeneRatio", "BgRatio")
  gp_mod$geneID = gsub(",", "/", gp_mod$geneID)
  return(gp_mod)
}
```

Finally, the function was called for each condition in a for loop, with plots saved and then arranged in a grid. Importantly, I sort the gost() result according to increasing term_size and decreasing Count, to get more specific terms that are shared by as most of the genes as possible. Here, only the GO-IDs are plotted due to size issues. While concentrating on Wildtypes, the filter that was commented out was also carried out.

```r
dot_data <- list()
expr_names <- names(expr_direction)
for (cond in seq_along(expr_direction)*2) {
  label <- expr_names[[cond/2]]
  dot_data[[cond - 1]] <- dotplot_updown(expr_direction[[cond/2]], "Up") %>%
    mutate(Cluster = paste0(label, "_", "Up"))
  dot_data[[cond]] <- dotplot_updown(expr_direction[[cond/2]], "Down") %>%
    mutate(Cluster = paste0(label, "_", "Down"))
}

dot_data %>%
  map(filter, Category %in% c("GO:MF", "GO:BP", "GO:CC")) %>%
  map(arrange, term_size, desc(Count)) %>%
  map(slice_head, n = 10) %>%
  bind_rows() %>%
  #filter(str_starts(Cluster, "Wildtype")) %>%
  ggplot(aes(x = Cluster, y = ID, size = Count, color = p.adjust)) +
  geom_point() +
  scale_size_continuous(range = c(3, 12)) +
  scale_color_gradient(low = "blue", high = "red") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle=90, hjust=1, vjust=1)) +
  labs(x = "Cluster", y = "Enrichment Term", title = "Gene Enrichment Analysis") +
  facet_wrap(Category ~ ., scales = "free_y", ncol = 1)
```


Gene Enrichment Analysis
GO:BP
GO:2000221
GO:0000281
GO:CC
GO:0098552
GO:0070772
GO:0044613
GO:0034518
GO:0032177
GO:0032176
GO:0032174
GO:0032173
GO:0032161
GO:0032156
GO:0030427
GO:0016281
GO:0005940
GO:0005935
GO:0005933
GO:0005621
GO:0000399
GO:0000164
GO:0000118
GO:MF
GO:1990837
GO:0140678
GO:0140297
GO:0140110
GO:0061629
GO:0030674
GO:0019212
GO:0017056
GO:0008157
GO:0008134
GO:0004864
GO:0004861
GO:0004857
GO:0003700
GO:0003690
GO:0001228
GO:0001227
GO:0001217
GO:0001216
GO:0001067
GO:0000987
GO:0000981
GO:0000978
GO:0000977
GO:0000976
Enrichment Term
Double.KDKO_42_10_Down
Double.KDKO_42_10_Up
Double.KDKO_42_30_Down
Double.KDKO_42_30_Up
HSF1.KD_42_10_Down
HSF1.KD_42_10_Up
HSF1.KD_42_30_Down
HSF1.KD_42_30_Up
MSN24.KO_42_10_Down
MSN24.KO_42_10_Up
MSN24.KO_42_30_Down
MSN24.KO_42_30_Up
Wildtype_42_10_Down
Wildtype_42_10_Up
Wildtype_42_30_Down
Wildtype_42_30_Up
Cluster
Count
5
10
15
20
25
p.adjust
0.04
0.03
0.02
0.01

Session Info

```r
sessionInfo()
```

```
## R version 4.4.2 (2024-10-31)
## Platform: x86_64-pc-linux-gnu
## Running under: TUXEDO OS
## 
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0 
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0
## 
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C              
##  [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=en_GB.UTF-8    
##  [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=en_GB.UTF-8   
##  [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C                 
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C            
## [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C       
## 
## time zone: Europe/Berlin
## tzcode source: system (glibc)
## 
## attached base packages:
## [1] grid      stats4    stats     graphics  grDevices utils     datasets 
## [8] methods   base     
## 
## other attached packages:
##  [1] pheatmap_1.0.12             janitor_2.2.1              
##  [3] gprofiler2_0.2.3            UpSetR_1.4.0               
##  [5] ggpubr_0.6.0                DESeq2_1.46.0              
##  [7] SummarizedExperiment_1.36.0 Biobase_2.66.0             
##  [9] MatrixGenerics_1.18.1       matrixStats_1.5.0          
## [11] GenomicRanges_1.58.0        GenomeInfoDb_1.42.3        
## [13] IRanges_2.40.1              S4Vectors_0.44.0           
## [15] BiocGenerics_0.52.0         lubridate_1.9.4            
## [17] forcats_1.0.0               stringr_1.5.1              
## [19] dplyr_1.1.4                 purrr_1.0.2                
## [21] readr_2.1.5                 tidyr_1.3.1                
## [23] tibble_3.2.1                ggplot2_3.5.1              
## [25] tidyverse_2.0.0            
## 
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-9            gridExtra_2.3           rlang_1.1.5            
##  [4] magrittr_2.0.3          snakecase_0.11.1        compiler_4.4.2         
##  [7] vctrs_0.6.5             pkgconfig_2.0.3         crayon_1.5.3           
## [10] fastmap_1.2.0           backports_1.5.0         XVector_0.46.0         
## [13] labeling_0.4.3          rmarkdown_2.29          tzdb_0.4.0             
## [16] UCSC.utils_1.2.0        bit_4.5.0.1             xfun_0.50              
## [19] zlibbioc_1.52.0         jsonlite_1.8.9          DelayedArray_0.32.0    
## [22] BiocParallel_1.40.0     broom_1.0.7             parallel_4.4.2         
## [25] R6_2.5.1                stringi_1.8.4           RColorBrewer_1.1-3     
## [28] car_3.1-3               Rcpp_1.0.14             knitr_1.49             
## [31] Matrix_1.7-1            timechange_0.3.0        tidyselect_1.2.1       
## [34] rstudioapi_0.15.0       abind_1.4-8             yaml_2.3.10
## [37] codetools_0.2-20        lattice_0.22-5          plyr_1.8.9
## [40] withr_3.0.2             evaluate_1.0.3          pillar_1.10.1
## [43] carData_3.0-5           plotly_4.10.4           generics_0.1.3
## [46] vroom_1.6.5             rprojroot_2.0.4         RCurl_1.98-1.16
## [49] hms_1.1.3               munsell_0.5.1           scales_1.3.0
## [52] glue_1.8.0              lazyeval_0.2.2          tools_4.4.2
## [55] data.table_1.16.4       locfit_1.5-9.10         ggsignif_0.6.4
## [58] colorspace_2.1-1        GenomeInfoDbData_1.2.13 Formula_1.2-5
## [61] cli_3.6.3               S4Arrays_1.6.0          viridisLite_0.4.2
## [64] gtable_0.3.6            rstatix_0.7.2           digest_0.6.37
## [67] SparseArray_1.6.1       htmlwidgets_1.6.4       farver_2.1.2
## [70] htmltools_0.5.8.1       lifecycle_1.0.4         httr_1.4.7
## [73] here_1.0.1              bit64_4.6.0-1
```