Proteomics Workflow and Visualizations in R

William Skelly

```
# Required Packages

# General data handling
library(readxl)
library(tidyverse)

# Core analysis and data structures
library(Biobase)
library(limma)
library(imputeLCMD)

# Functional enrichment analysis
library(ReactomePA)
library(reactome.db)
library(AnnotationDbi)

# Visualization
library(ggplot2)
library(ComplexHeatmap)
library(circlize)
library(patchwork)
library(ggpubr)


# Report formatting
library(knitr)
```

Introduction:

Mass-spectrometry based proteomics workflows are important for examining the large-scale quantification of protein expression levels. Data visualization is an important part of this

workflow, as it helps interpret results in meaningful ways. This project focuses on the preparation, and visualization of proteomics data across four experimental conditions (KO, KO-IV, KO-NEB, WT). The goal is to apply data visualization techniques to explore differential protein expression patterns across condition groups.

A series of visualization methods, including multidimensional scaling (MDS) plots, volcano plots, heat-maps and gene-concept network plots, were applied to visualize relationships between condition groups. The methods section describes data preprocessing workflows and statistical modeling steps, while the results section presents the figures. Both sections include reproducible R code. The overall structure of the data processing workflow, including the sequence of visualizations and major code design elements, was adapted from Sanz and Sánchez-Pla (2019) (Sanz and Sánchez-Pla 2019). Additionally, some of the visualizations were inspired by and adapted from Schessner (2022) (Schessner, Voytik, and Bludau 2022).

This project was completed with the help of my P.I., Dr. Leonardo Ferreira and my statistics advisor, Dr. Edwin Iversen.

2. Methods:

Data preprocessing, modeling, and visualization were conducted using R and Bioconductor packages (Huber et al. 2015). Specific packages used included `limma` (Ritchie et al. 2015), `Biobase` (Gentleman et al. 2004), ReactomePA (Yu and He 2016), and others, as described below.

The proteomics dataset was processed and visualized following a structured computational workflow. Data preprocessing steps included $\log_2$ transformation, missing value imputation, and filtering based on data completeness. Differential expression analysis was conducted through the `limma` package , which included linear modeling and empirical Bayes moderation. Enriched pathway analysis was conducted through the `ReactomePA` package . Visualization techniques, including volcano plots, multidimensional scaling (MDS) plots, heatmaps, venn diagrams, and gene-concept network plots, were applied to explore expression patterns and pathway enrichments.

2.1 Data Acquisition and Pre-processing

The proteomics dataset was provided by Dr. Leonardo Ferreira as a excel spreadsheet. The Accession Number and Entrez Gene ID identify each protein found through the MS process. Columns 50:61 contain the normalized protein abundances for each of the biological replicants. Each of the four condition groups (KO, KO-NEB, KO-IV, WT) have three samples each. Before I received the normalized abundance data, it was processed using the 100% presence filter. This means that for each condition group, if a protein was not detected in all three samples, then the data for the three samples were not included.

```r
proteomics_df <- read_excel("../Data/33677_HEART_LFQ_PD.xlsx")

exprs_df <- proteomics_df[ , c(50:61)]

rownames(exprs_df) <- proteomics_df$Accession

clean_names <- gsub("Abundances \\(Normalized\\): (.*): Sample, (.*)",
                    "\\1: \\2", colnames(exprs_df))

colnames(exprs_df) <- clean_names

annotation_df <- proteomics_df[ , c(22:28)]

rownames(annotation_df) <- proteomics_df$Accession

annotation_mat <- as.matrix(annotation_df)
```

The normalized protein abundances were first $\log_2$ transformed. Then, expression matrices and associated sample metadata were organized using the ExpressionSet object structure from the Biobase R package (Gentleman et al. 2004).

```r
log_exprs_df <- log2(exprs_df)

exprs_df <- sapply(log_exprs_df, as.numeric)

exprs_mat <- as.matrix(exprs_df)
```

```r
build_eset <- function(exprs_mat, annotation_mat,
                       exprs_type = c("unfiltered", "filtered", "imputed")) {

  #' Build an ExpressionSet from proteomics data
  #' with optional filtering or imputation
  #'
  #' @param exprs_mat A numeric matrix of protein abundance
  #' values (rows = proteins, cols = samples)
  #' @param annotation_mat A matrix or dataframe of protein-level
  #' annotations (same rownames as exprs_mat)
  #' @param exprs_type Character. One of "unfiltered", "filtered",
  #' or "imputed", determining preprocessing steps
  #'
  #' @return An ExpressionSet object containing assay data, feature
```

```
#' data, and sample metadata
#' @export

# exprs_mat and annotation df must be the same length
stopifnot(nrow(exprs_mat) == nrow(annotation_mat))

# exprs_mat and annotation_mat must be matrices
stopifnot(is.matrix(exprs_mat) || is.matrix(annotation_mat))

# exprs_mat and annotation_mat must have the same row names.
stopifnot(rownames(annotation_mat) == rownames(exprs_mat))

exprs_type <- match.arg(exprs_type)

if (exprs_type == "unfiltered") {

  exprs_mat_use <- exprs_mat
  annotation_df_use <- annotation_df

} else if (exprs_type == "filtered") {

  # Keep rows where there are no NA values.
  na_percent <- rowMeans(is.na(exprs_mat))

  exprs_mat_filtered <- exprs_mat[na_percent <= 0, ]
  annotation_df_filtered <- annotation_df[rownames(exprs_mat_filtered), ]

  exprs_mat_use <- exprs_mat_filtered
  annotation_df_use <- annotation_df_filtered

} else if (exprs_type == "imputed") {

  na_percent <- rowMeans(is.na(exprs_mat))

  exprs_mat_filtered <- exprs_mat[na_percent <= 0.25 , ]

  exprs_mat_imputed <- impute.MinProb(as.matrix(exprs_mat_filtered),
                                      q = 0.01)

  annotation_df_imputed <- annotation_df[rownames(exprs_mat_imputed), ]

  exprs_mat_use <- exprs_mat_imputed
```

```
    annotation_df_use <- annotation_df_imputed

  }

  # Reconstruct sample metadata (doesn't change between types)
  sample_names <- colnames(exprs_mat_use)

  condition <- sapply(strsplit(sample_names, ": "),
                      function(x) x[2])

  condition <- factor(condition)

  sample_data <- data.frame(
    sampleID = sample_names,
    Condition = condition
  )

  # Create the ExpressionSet
  eset_out <- ExpressionSet(assayData = exprs_mat_use)
  fData(eset_out) <- annotation_df_use
  pData(eset_out) <- sample_data

  return(eset_out)
}


eset_unfiltered <- build_eset(exprs_mat, annotation_mat,
                              exprs_type = "unfiltered")
eset_filtered <- build_eset(exprs_mat, annotation_mat,
                            exprs_type = "filtered")
eset_imputed <- build_eset(exprs_mat, annotation_mat,
                           exprs_type = "imputed")
```

```
[1] 0.4542286
```

Missing values were imputed using the `MinProb` method from the `imputeLCMD` R package (Lazar et al. 2016). More information about the imputation method can be found in Lazar et al. 2016).

For all subsequent analysis, `eset_unfiltered` was used as the expression set for differential expression analysis and enriched pathway analysis. The choice to use the unfiltered expression set stems from the fact that the data was already filtered using the 100% presence filter, and

so imputing values for a full set of samples results in condition comparisons that may be completely dependent on imputed data.

For a comprehensive discussion on imputation techniques for proteomics data with missing values, refer to Wang (2017). (Wang et al. 2017).

```
eset <- eset_unfiltered

#exprs(eset)      # expression values
#fData(eset)      # annotations
#pData(eset)      # sample metadata
```

2.2 Visualization Workflows

In this section, we discuss the methods used to create each visualization. Code associated with each graph is displayed.

2.2.1 Sample-Level MDS Plots and Dimension Variance

I defined a plot_mds function that takes an expression matrix, and the plotting dimensions as input, and uses the `plotMDS` function in the `limma` package (Ritchie et al. 2015) to generate sample-level MDS plots across different dimensions. The `top` argument specifies how many of the most variable proteins to use for computing pairwise distances between samples. Because `gene.selection` is set to "pairwise", a different set of `top` proteins are selected for each pair of samples. According to the `limma` documentation, the "distances on the plot can be interpreted as *leading log2-fold-change*, meaning the typical (root-mean-square) log2-fold-change between the samples for the genes that distinguish those samples" (Bioconductor n.d.).

```
plot_mds <- function(exprs_matrix, x_dim, y_dim, group, group_color,
                     plot = TRUE, clean = FALSE) {
  #' Plot MDS (Multidimensional Scaling) results from
  #' an expression matrix
  #'
  #' @param exprs_matrix A numeric expression matrix (e.g., from
  #'  ExpressionSet) with protein abundances (features x samples)
  #' @param x_dim Integer. The MDS dimension to use for the x-axis
  #' @param y_dim Integer. The MDS dimension to use for the y-axis
  #' @param title_prefix Character. Optional prefix for the plot
  #'  title
  #' @param plot Logical. If TRUE, generates the MDS plot;
  #' if FALSE, only returns explained variance
```

```r
  #' @param clean Logical. Indicates whether the data is
  #' cleaned/imputed (affects plot title)
  #'
  #' @return A numeric vector of the percent variance
  #' explained by each MDS dimension
  #' @export

   # Run MDS but do not plot immediately
  mds_result <- plotMDS(exprs_matrix, gene.selection = "pairwise",
                        top = 500, dim.plot = c(x_dim, y_dim),
                        plot= FALSE)

  # Extract and format variance explained
  var_expl <- round(mds_result$var.explained, 4) * 100
  # convert to percentage

  if (plot) {
    # Adjust title based on whether data is imputed
    title <- ifelse(clean, "Imputed", "Filtered")

    # Plot the MDS dimensions
    par(mar = c(5, 4, 4, 8), xpd = TRUE)
    plot(mds_result$x, mds_result$y,
         col = group_colors[group],
         pch = 16,
         cex = 1.5,
         xlab = paste0("Dim ", x_dim, " (", var_expl[x_dim], "%)"),
         ylab = paste0("Dim ", y_dim, " (", var_expl[y_dim], "%)"),
         main = paste0(title))


    # Add legend outside the plot area
    legend("topright", inset = c(-0.25, 0),
           legend = levels(group),
           col = group_colors,
           pch = 16,
           pt.cex = 1.5,
           bty = "n")
  }

  invisible(var_expl)
}
```

2.2.2 Volcano Plots for Differential Expression Analysis

Differential expression analysis was conducted using the `eBayes` function in the `limma` package (Phipson et al. 2016). The functions used, `model.matrix`, `makeContrasts`, `contrast.fit`, `topTable` and `eBayes`, all come from the `limma` package. For a comprehensive discussion about why we used `eBayes` for differential expression analysis of proteomics data, refer to (Kammers et al. 2015).

First, I built a design matrix using `model.matrix` from condition group labels (WT, KO-IV, KO-NEB, KO). Then, I fit a linear model to each protein (row) in the expression set separately using the `lmFit` function.

```
# Each row in the design matrix is a sample, 1 - 12, and each column
# is a condition group (KO, KO-IV, KO-NEB, WT). Each cell is either
# 1 if that sample belongs to that condition group, or 0 if not.

designMat<- model.matrix(~0+Condition, pData(eset))

colnames(designMat) <- c("KO", "KO.IV", "KO.NEB", "WT")

print(designMat)
```

```
   KO KO.IV KO.NEB WT
1   1     0      0  0
2   1     0      0  0
3   1     0      0  0
4   0     1      0  0
5   0     1      0  0
6   0     1      0  0
7   0     0      1  0
8   0     0      1  0
9   0     0      1  0
10  0     0      0  1
11  0     0      0  1
12  0     0      0  1
attr(,"assign")
[1] 1 1 1 1
attr(,"contrasts")
attr(,"contrasts")$Condition
[1] "contr.treatment"
```

```
fit <- lmFit(eset, designMat)
```

Then, I used the `makeContrasts` function to define the comparisons between condition groups (WT:KO, KO-IV:KO, KO-NEB:KO) and used `contrast.fit` to apply the contrasts to the linear model.

```
# Define contrasts (other groups compared to KO)
contrastMat <- makeContrasts(
  WTvsKO     = WT - KO,
  KO.IVvsKO  = KO.IV - KO,
  KO.NEBvsKO = KO.NEB - KO,
  levels = designMat
)

print(contrastMat)
```

```
        Contrasts
Levels   WTvsKO KO.IVvsKO KO.NEBvsKO
  KO          -1        -1         -1
  KO.IV        0         1          0
  KO.NEB       0         0          1
  WT           1         0          0
```

```
fit.main <- contrasts.fit(fit, contrastMat)
```

Then, I used the `eBayes` function to shrink the variances of each protein. This improves the stability of tests when the sample size is small, in our case, 3 samples per condition.

```
fit.main <- eBayes(fit.main)
```

I used the `topTable` function to summarize the linear model fit object produced by `eBayes`. This orders the top ranked proteins by differential expression for each condition group, where differentially expressed proteins are defined as $|\log_2$ Fold Change $ 1$ and Benjamini-Hochberg adjusted p-values < 0.05.

```
topTab_WTvsKO <- topTable(fit.main, number = nrow(fit.main),
                          coef = "WTvsKO", adjust = "BH")


topTab_KO.NEBvsKO <- topTable(fit.main, number = nrow(fit.main),
```

```
                            coef = "KO.NEBvsKO", adjust = "BH")


topTab_KO.IVvsKO <- topTable(fit.main, number = nrow(fit.main),
                             coef = "KO.IVvsKO", adjust = "BH")
```

I added a significance flag to the `topTable` to document whether a protein was significant (differentially expressed).

```
# Add significance flag
SigP <- 0.05
SigLFC <- 1

topTab_WTvsKO$Significant <- with(topTab_WTvsKO,
  ifelse(adj.P.Val < SigP & abs(logFC) > SigLFC,
         "Significant", "Not Significant"))

topTab_KO.IVvsKO$Significant <- with(topTab_KO.IVvsKO,
  ifelse(adj.P.Val < SigP & abs(logFC) > SigLFC,
         "Significant", "Not Significant"))

topTab_KO.NEBvsKO$Significant <- with(topTab_KO.NEBvsKO,
  ifelse(adj.P.Val < SigP & abs(logFC) > SigLFC,
         "Significant", "Not Significant"))
```

2.2.3 Overlap of Differentially Expressed Proteins Across Contrasts

The `DecideTests` function from the `limma` package was used to compare the number of differentially expressed proteins in each contrast group.

```
# Shows the number of upregulated, downregulated, and non
# significant proteins from each contrast group.

res <- decideTests(fit.main, method = "separate",
                   adjust.method = "BH", p.value = SigP,
                   lfc = SigLFC)

sum.res.rows <-apply(abs(res),1,sum)

res.selected <-res[sum.res.rows!=0,]
```

```
print(summary(res))
```

```
       WTvsKO KO.IVvsKO KO.NEBvsKO
Down      132        65        129
NotSig   1733      1918       1825
Up         41        54         53
```

2.2.4 Heatmap of Differentially Expressed Proteins

Differentially expressed proteins from the `decideTests` function were selected for the heatmap. Z-scores represent scaled expression levels relative to the mean across samples. The `HeatmapAnnotation` and `Heatmap` functions from the `ComplexHeatmap` package were used to create the heatmap. (Gu, Eils, and Schlesner 2016).

```
# Get only selected DE proteins
proteinsInHeatmap <- rownames(res.selected)
HMdata <- exprs(eset)[rownames(exprs(eset)) %in% proteinsInHeatmap, ]

# Z-score scale each row (protein)
HMdata_scaled <- t(scale(t(HMdata)))

# Create annotation from column names
annotation_col <- data.frame(
  Condition = factor(sapply(strsplit(colnames(HMdata_scaled), ": "),
                            function(x) x[2]))
)
rownames(annotation_col) <- colnames(HMdata_scaled)

# Define condition colors
ann_colors <- list(
  Condition = c(
    "KO" = "red",
    "KO-IV" = "blue",
    "KO-NEB" = "green",
    "WT" = "yellow"
  )
)

# Format annotation for ComplexHeatmap
ha <- HeatmapAnnotation(
```

```
  Condition = annotation_col$Condition,
  col = ann_colors
)

# Define color scale for z-score
col_fun <- colorRamp2(c(-2, 0, 2), c("blue", "white", "red"))

# Draw the heatmap
heatmap_plot <- Heatmap(HMdata_scaled,
        name = "z-score",
        col = col_fun,
        top_annotation = ha,
        show_row_names = FALSE,
        show_column_names = TRUE,
        column_names_gp = gpar(fontsize = 10),
        cluster_rows = TRUE,
        cluster_columns = TRUE,
        clustering_method_rows = "complete",
        clustering_method_columns = "complete",
        column_title = "")
```

2.2.5 Enrichment Analysis

Reactome pathway enrichment analysis was performed using a custom wrapper function, `runReactomeEnrichment`, which iteratively applies the `enrichPathway` function from the ReactomePA package (Yu and He 2016) to multiple sets of differentially expressed proteins. The background universe was defined as the intersection between proteins in the full dataset and those annotated in the Reactome database using the `reactome.db` package (Ligtenberg 2019). Gene-to-pathway mapping and annotation extraction were handled using functions from the `AnnotationDbi` package (Pagès et al. 2023). For each contrast group, differentially expressed proteins were tested for pathway over-representation. The function returns the `enrichResult` as a dataframe and the object itself for downstream analysis.

```
listOfTables <- list(KO.NEBvsKO = topTab_KO.NEBvsKO,
                     KO.IVvsKO = topTab_KO.IVvsKO,
                     WTvsKO = topTab_WTvsKO)
```

```
runReactomeEnrichment <- function(listOfTables, eset, organism = "mouse", pval_cutoff = 0.05)
  #' Run Reactome pathway enrichment for a list of top tables
  #'
  #' @param listOfTables A named list of top tables
```

```
#' (each with 'adj.P.Val' and 'Entrez.Gene.ID')
#' @param eset An ExpressionSet object containing
#' feature metadata with Entrez Gene IDs
#' @param organism Character. Organism name (default = \"mouse\")
#' @param pval_cutoff Numeric. Adjusted p-value cutoff for
#' selecting significant genes
#'
#' @return A list with two elements:
#'   - enrichmentResults: list of data.frames of enrichment results
#'   - enrichObjects: list of enrichPathway objects (for further plotting)
#' @export

# Create list of selected significant Entrez IDs
listOfSelected <- list()

for (i in seq_along(listOfTables)) {
  topTab <- listOfTables[[i]]
  SigGenes <- topTab$adj.P.Val < pval_cutoff

  selectedIDs <- topTab$Entrez.Gene.ID[SigGenes]
  selectedIDs <- selectedIDs[!is.na(selectedIDs)]

  listOfSelected[[i]] <- selectedIDs
  names(listOfSelected)[i] <- names(listOfTables)[i]
}

# Map Entrez IDs to Reactome genes
reactome_map <- AnnotationDbi::select(
  reactome.db,
  keys = keys(reactome.db, keytype = "ENTREZID"),
  columns = c("ENTREZID", "REACTOMEID"),
  keytype = "ENTREZID"
)
reactome_genes <- unique(reactome_map$ENTREZID)

# Define universe genes
entrez_raw <- fData(eset)$`Entrez Gene ID`
entrez_clean <- unlist(strsplit(as.character(entrez_raw), ";"))
universe <- unique(trimws(na.omit(entrez_clean)))
universe <- intersect(universe, reactome_genes)

# Run enrichment
```

```
  enrichmentResults <- list()
  enrichObjects <- list()

  comparisonNames <- names(listOfSelected)

  for (i in seq_along(listOfSelected)) {
    genesIn_raw <- listOfSelected[[i]]
    genesIn <- unlist(strsplit(genesIn_raw, ";"))
    genesIn <- as.character(unique(na.omit(trimws(genesIn))))

    comparison <- comparisonNames[i]

    overlap <- length(intersect(genesIn, universe))
    cat("Overlap with Universe genes for", comparison, ":", overlap, "\n")

    enrich.result <- enrichPathway(
      gene = genesIn,
      pvalueCutoff = 0.05,
      organism = organism,
      universe = universe,
      pAdjustMethod = "BH",
      minGSSize = 5,
      maxGSSize = 500
    )

    enrichObjects[[i]] <- enrich.result
    names(enrichObjects)[i] <- comparison

    enrichmentDf <- as.data.frame(enrich.result)
    enrichmentResults[[i]] <- enrichmentDf
    names(enrichmentResults)[i] <- comparison
  }

  return(list(
    enrichmentResults = enrichmentResults,
    enrichObjects = enrichObjects
  ))
}

# Run the enrichment process
enrichment_output <- runReactomeEnrichment(listOfTables, eset)
```

```
Overlap with Universe genes for KO.NEBvsKO : 152
Overlap with Universe genes for KO.IVvsKO : 83
Overlap with Universe genes for WTvsKO : 133
```

2.2.6 Protein-Level MDS Plots Across Contrast Groups

To visualize protein-level separation between sample groups for each contrast, I developed a custom function, `plotMDSContrast`, which applies the `plotMDS` function from the `limma` package to each condition group. The function selects proteins with no missing values and that are common to both the expression matrix and the corresponding `topTable` of differential expression results.

```r
plotMDSContrast <- function(eset, topTab, samples, title) {
  #' Plot MDS (Multidimensional Scaling) results for a selected contrast
  #'
  #' @param eset An ExpressionSet object containing expression data
  #' (features x samples)
  #' @param topTab A data frame with feature significance
  #' information (must include a 'Significant' column)
  #' @param samples A character vector indicating which sample
  #' conditions to include (matches pData(eset)$Condition)
  #' @param title A character string for the plot title
  #' @return A ggplot object of MDS results with points colored by significance
  #' @export


  # Identify columns (samples) matching the requested conditions
  samples_contrast <- which(pData(eset)$Condition %in% samples)

  # Subset expression matrix to these samples
  exprs_sub <- exprs(eset)[, samples_contrast]

  # Remove rows (features) with any missing values
  exprs_sub <- exprs_sub[rowSums(is.na(exprs_mat)) == 0, ]

  # Ensure only proteins in both topTable and subset matrix
  common_proteins <- intersect(rownames(exprs_sub), rownames(topTab))
  exprs_sub <- exprs_sub[common_proteins, ]
  topTab_sub <- topTab[common_proteins, ]

  # Perform MDS analysis but do not plot immediately
  mds <- plotMDS(t(exprs_sub), dim.plot = c(2,3), plot = FALSE)
```

```r
  mds_df <- data.frame(
    Dim1 = mds$x,
    Dim2 = mds$y,
    Significant = factor(topTab_sub$Significant,
                         levels = c("Significant", "Not Significant"))
  )

  # Define a color map
  color_map <- c(
    "Significant" = rgb(178, 34, 34, alpha = 180, maxColorValue = 255),
    "Not Significant" = rgb(169, 169, 169, alpha = 80, maxColorValue = 255)
  )

  p <- ggplot(mds_df, aes(x = Dim1, y = Dim2, color = Significant)) +
    geom_point(size = 1.5, alpha = 0.8) +
    scale_color_manual(values = color_map) +
    labs(
      title = title,
      x = paste0("Dim 1 (", round(mds$var.explained[1] * 100, 1), "%)"),
      y = paste0("Dim 2 (", round(mds$var.explained[2] * 100, 1), "%)")
    ) +
    theme_minimal() +
    theme(
      plot.title = element_text(hjust = 0.5),
      legend.title = element_blank()
    )

  return(p)
}
```

3. Results

This section presents the results of a structured data visualization workflow applied to a proteomics dataset. Visualizations include volcano plots, multidimensional scaling (MDS) plots, heatmaps, Venn diagrams, and gene-concept network plots. These figures, each at a different stage in the analysis pipeline, demonstrate the progression from statistical modeling, to differential expression analysis, to pathway enrichment analysis.

3.1 Sample-Level MDS Plots

Samples are plotted in two dimensions, and are colored by their condition group. Sample-level MDS plots were created for filtered and imputed data to compare how well samples within the same condition group cluster.

```
# For unfiltered:
sample_names_filtered <- colnames(exprs(eset_filtered))
group_filtered <- factor(sapply(strsplit(sample_names_filtered, ": "),
                                function(x) x[2]))

# For imputed:
sample_names_imputed <- colnames(exprs(eset_imputed))
group_imputed <- factor(sapply(strsplit(sample_names_imputed, ": "),
                               function(x) x[2]))


group_colors <- c("KO" = "blue", "KO-IV" = "red", "KO-NEB" = "forestgreen", "WT" = "orange")
```

```
par(mfrow = c(2, 1))

plot_mds(exprs(eset_filtered), 1, 2, group_filtered, group_colors)
plot_mds(exprs(eset_imputed), 1, 2, group_imputed, group_colors, clean = TRUE)
```

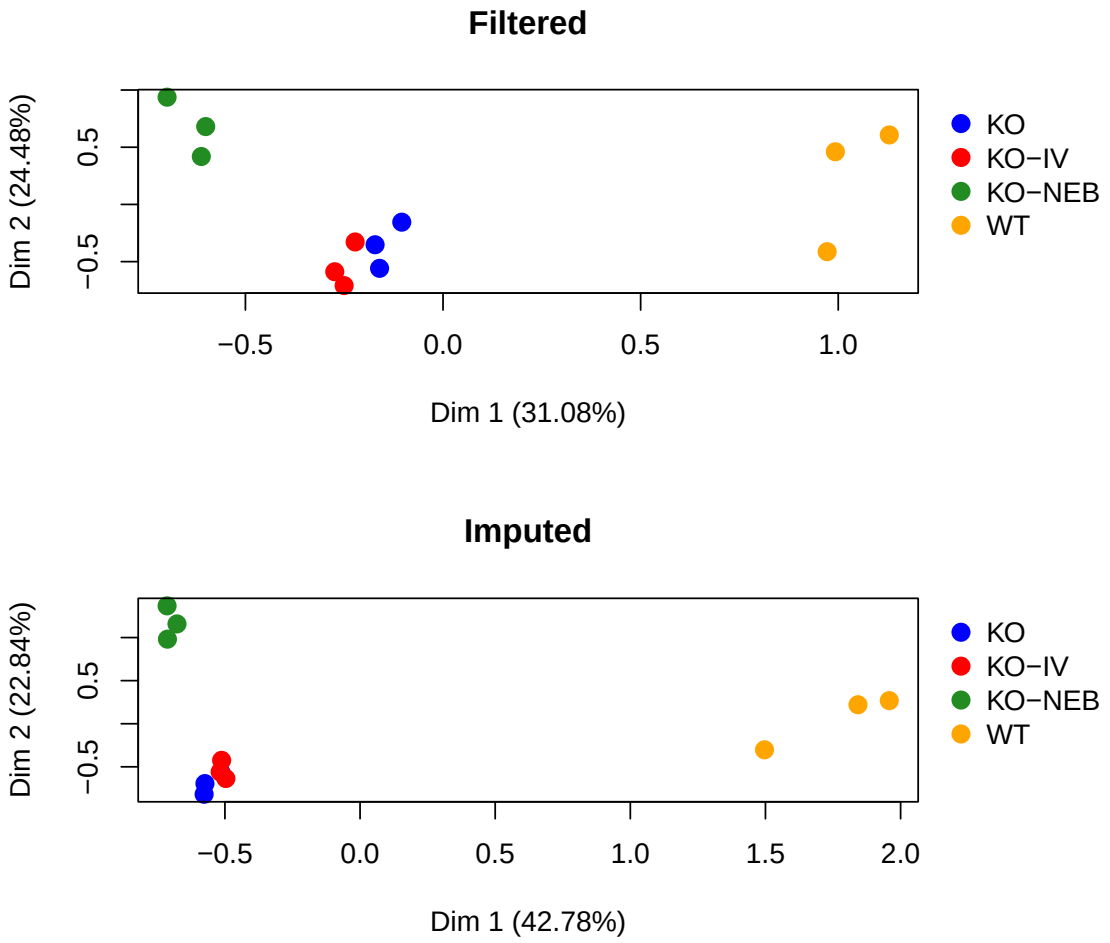

Figure 1A. MDS plots by sample, across dimension 1 and 2, for imputed and filtered data. Each sample is colored by condition group.

```
par(mfrow = c(2, 1))
plot_mds(eset_filtered, 1, 3, group_filtered, group_colors)
plot_mds(eset_imputed, 1, 3, group_imputed, group_colors, clean = TRUE)
```

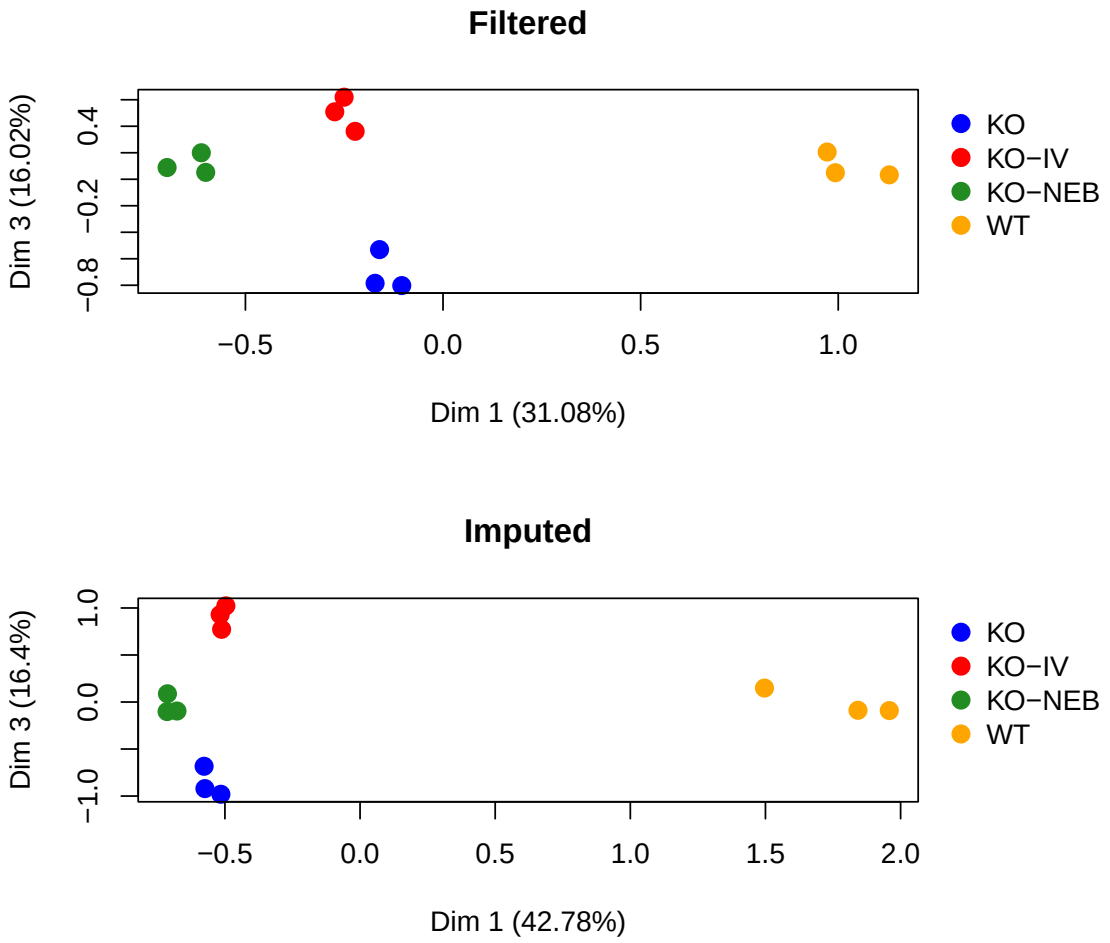

Figure 1B. MDS plots by sample, across dimension 1 and 3, for imputed and filtered data. Each sample is colored by condition group.

```
par(mfrow = c(2, 1))
plot_mds(eset_filtered, 2, 3, group_filtered, group_colors)
plot_mds(eset_imputed, 2, 3, group_imputed, group_colors, clean = TRUE)
```

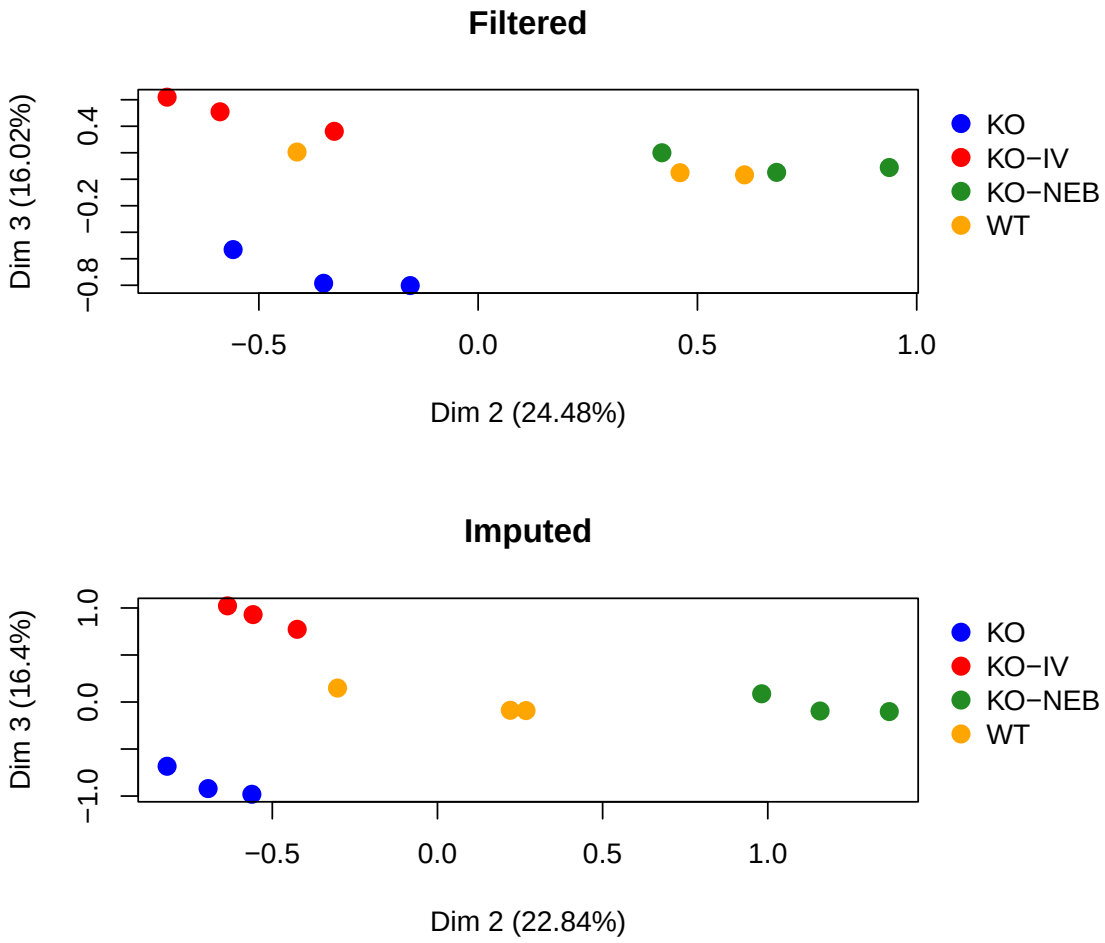

Figure 1C. MDS plots by sample, across dimension 2 and 3. Each sample is colored by condition group.

3.2 Variability Across MDS Plot Dimensions

These graphs show the variability accounted for by each dimension from MDS analysis, for filtered data and imputed data. The interpretability of each of the dimensions is not clear.

```
var_expl <- plot_mds(eset_filtered, 1, 2, plot = FALSE)

var_df <- data.frame(
  PC = factor(1:6),
  Variability = var_expl[1:6] / 100  # Convert back to proportion for y-axis
```

```
)


p1 <- ggplot(var_df, aes(x = PC, y = Variability)) +
  geom_bar(stat = "identity", fill = "#1f77b4") +
  labs(title = "Filtered Data",
       x = "Dimension",
       y = "Variability") +
  theme_minimal(base_size = 14)

var_expl1 <- plot_mds(eset_imputed, 1, 2, plot = FALSE)

var_df <- data.frame(
  PC = factor(1:6),
  Variability = var_expl1[1:6] / 100  # Convert back to proportion for y-axis
)

p2 <- ggplot(var_df, aes(x = PC, y = Variability)) +
  geom_bar(stat = "identity", fill = "#1f77b4") +
  labs(title = "Imputed Data",
       x = "Dimension",
       y = "Variability") +
  theme_minimal(base_size = 14)

p1 + p2
```

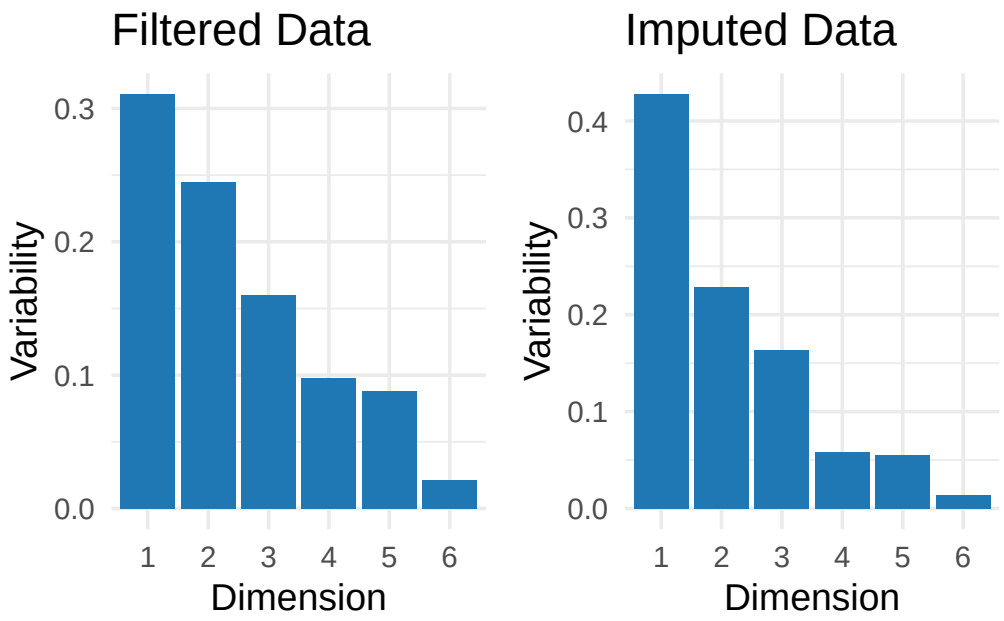

Figure 2. Bar plot of variability explained by each dimension from MDS analysis, for filtered and imputed data.

3.3 Volcano Plots

Volcano plots visualize the distribution of protein-level differential expression. The x-axis is the $\log_2 FC$, and the y-axis is the $-\log_{10}$ adjusted P-value. The horizontal and vertical dotted line display the significance thresholds (FDR < 0.05, $|\log_2 FC| \geq 1$ respectively). Proteins outside and above the dotted lines are classified as differentially expressed, and colored in red.

```
volcano1 <- ggplot(topTab_WTvsKO, aes(x = logFC, y = -log10(adj.P.Val),
                                      color = Significant)) +
  geom_point(alpha = 0.7) +
  scale_color_manual(values = c("gray", "red")) +
  geom_vline(xintercept = c(-SigLFC, SigLFC), linetype = "dashed") +
  geom_hline(yintercept = -log10(SigP), linetype = "dashed") +
  labs(title = "WT vs KO",
       x = "log2 Fold Change",
       y = "-log10 Adjusted P-Value",
       color = "significance") +
  theme_minimal()
```

```
volcano2 <- ggplot(topTab_KO.IVvsKO, aes(x = logFC, y = -log10(adj.P.Val),
                                         color = Significant)) +
  geom_point(alpha = 0.7) +
  scale_color_manual(values = c("gray", "red")) +
  geom_vline(xintercept = c(-SigLFC, SigLFC), linetype = "dashed") +
  geom_hline(yintercept = -log10(SigP), linetype = "dashed") +
  labs(title = "KO-IV vs KO",
       x = "log2 Fold Change",
       y = "-log10 Adjusted P-Value",
       color = "Significance"
       ) +
  theme_minimal()
```

```
volcano3 <- ggplot(topTab_KO.NEBvsKO, aes(x = logFC, y = -log10(adj.P.Val),
                                          color = Significant)) +
  geom_point(alpha = 0.7) +
  scale_color_manual(values = c("gray", "red")) +
  geom_vline(xintercept = c(-SigLFC, SigLFC), linetype = "dashed") +
  geom_hline(yintercept = -log10(SigP), linetype = "dashed") +
  labs(title = "KO-NEB vs KO",
       x = "log2 Fold Change",
       y = "-log10 Adjusted P-Value",
       color = "Significance") +
  theme_minimal()
```

```
volcano1_clean <- volcano1 + theme(axis.title = element_blank())
volcano2_clean <- volcano2 + theme(axis.title = element_blank())
volcano3_clean <- volcano3 + theme(axis.title = element_blank())

combined_volcano <- ggarrange(volcano1_clean, volcano2_clean, volcano3_clean,
                      ncol = 3,
                      common.legend = TRUE,
                      legend = "right")

final_volcano_plot <- annotate_figure(combined_volcano,
  left = text_grob("-log10 Adjusted P-Value", rot = 90, vjust = 1, size = 12),
  bottom = text_grob("log2 Fold Change", size = 12)
)

# Display the result
print(final_volcano_plot)
```

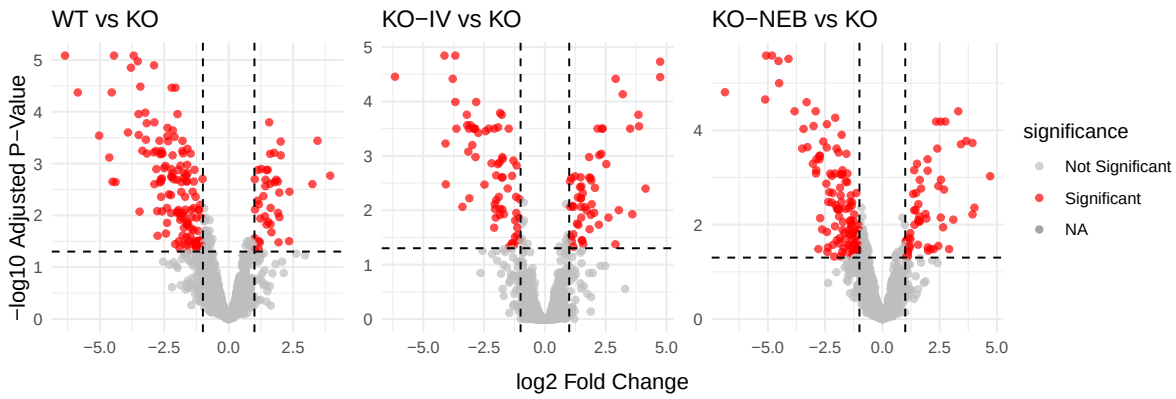

Figure 3. Volcano Plots across comparisons. Red points represent Differentially Expressed proteins (FDR < 0.05, $|\log_2 FC| \geq 1$), and grey points represent insignificant proteins.

3.4 Venn Diagram of Differentially Expressed Protein Overlap

The `vennDiagram` function from the `limma` package was used to create a Venn diagram displaying the number of differentially expressed proteins that overlap between condition groups.

```
vennDiagram(res.selected[,1:3], cex=0.7)
```

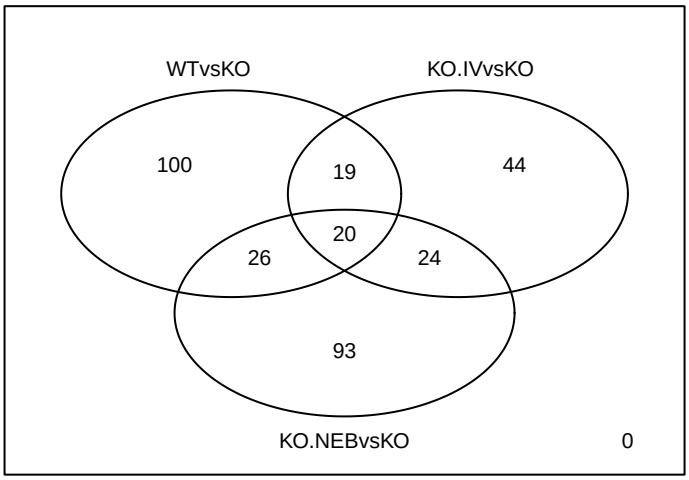

Figure 4. Venn Diagram of the number of significant proteins shared in each contrast group.

3.5 Heatmap

The Heatmap displays the similarities between proteins and condition groups, and hierarchically clusters them. Z-scores are colored from blue to white to red, where blue represents relatively low expression, white represents average expression, and red represents relatively high expression across differentially expressed proteins.

```
heatmap_plot
```

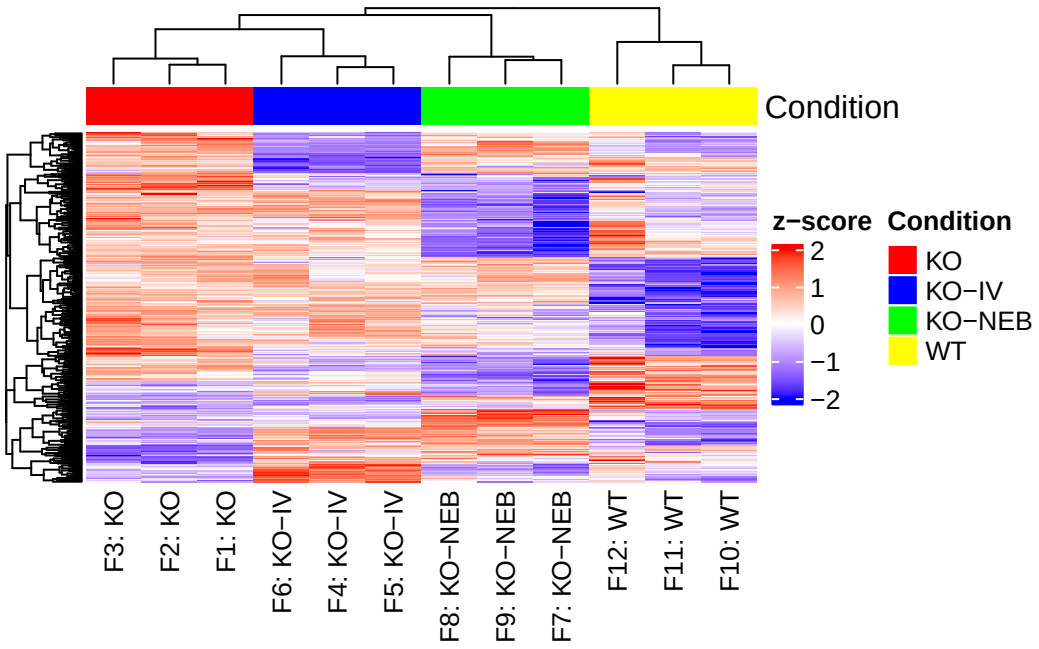

Figure 5. Heatmap of differentially expressed proteins (FDR < 0.05, $|\log_2 FC| \geq 1$) across conditions. Hierarchical clustering was performed across proteins (rows) and samples (columns) using dendrograms to group similar expression levels. Colors represent Z-scores indicating relative expression levels normalized to the mean.

3.6 Gene-Concept Network Plots

Gene-concept network plots were created used the `cnetplot` function from the `clusterProfiler` package (Yu et al. 2012). Significant proteins are labeled and represented by the small grey points, and pathways are represented by the yellow dots. The size of the yellow dot corresponds to the number of significant proteins found in that pathway, and edges between proteins and pathways indicate that that protein was found in the pathway. Edges are colored by pathway as well.

```
enrichObjects <- enrichment_output$enrichObjects

for (name in names(enrichObjects)) {
  cat("Plotting for:", name, "\n")

  enrichObj <-enrichObjects[[name]]

  if (nrow(as.data.frame(enrichObj)) > 0) {
    p <- cnetplot(enrichObj, categorySize = "geneNum", colorEdge = TRUE,
                  node_label = "gene")
    p <- p + ggtitle(name)

    print(p)

  }
}
```

```
Plotting for: KO.NEBvsKO
Plotting for: KO.IVvsKO
Plotting for: WTvsKO
```

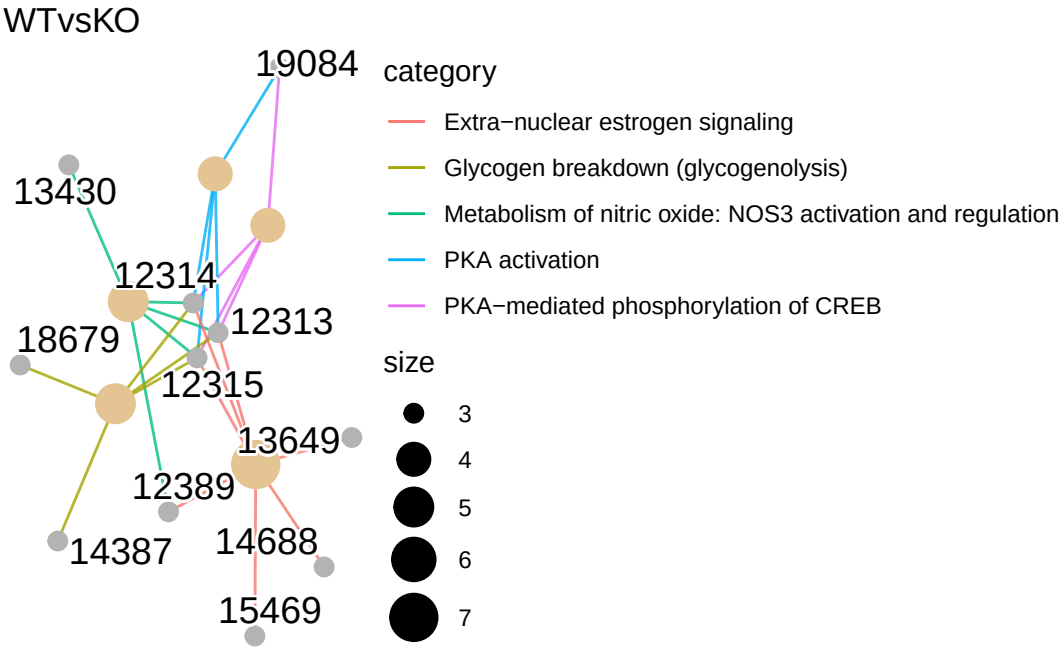

Figure 6. Reactome pathway network showing group membership of significant proteins in enriched pathways, for WT vs KO comparison. Edges are colored by pathway, and node sizes correspond to group size.

3.7 Enrichment Analysis Plots

A figure similar to Schessner et al. (2022, Fig. 4G) (Schessner, Voytik, and Bludau 2022) was recreated in R for this analysis. This figure displays the enriched pathways for each condition group, ordered by $-\log_{10}$ adjusted p-value. The odds ratio was computed by taking the number of significant proteins found in a pathway out of the total number of significant proteins, and the number of universe proteins found in a pathway out of the total number of universe proteins, and compute their ratio. The size of the points detail the number of significant proteins found in each pathway. In our case, only the WT:KO condition group had enriched pathways, so only one graph was displayed.

```
enrichmentResults <- enrichment_output$enrichmentResults

# For each contrast group
for (i in seq_along(enrichmentResults)){
  if (nrow(enrichmentResults[[i]]) > 0){

    enrichmentDf <- enrichmentResults[[i]]

    # create dataframe order by adjusted p-value
    plotDf <- enrichmentDf[order(enrichmentDf$p.adjust), ]

    # Calculate odds ratio

    geneHits <- as.numeric(sub("/.*", "", enrichmentDf$GeneRatio))
    # x in x/y
    geneTotal <- as.numeric(sub(".*/", "", enrichmentDf$GeneRatio))
    # y in x/y

    bgHits <- as.numeric(sub("/.*", "", enrichmentDf$BgRatio))
    # x in x/y
    bgTotal <- as.numeric(sub(".*/", "", enrichmentDf$BgRatio))
    # y in x/y

    geneHits <- geneHits[order(enrichmentDf$p.adjust)]
    geneTotal <- geneTotal[order(enrichmentDf$p.adjust)]
    bgHits <- bgHits[order(enrichmentDf$p.adjust)]
    bgTotal <- bgTotal[order(enrichmentDf$p.adjust)]
```

```
    plotDf$oddsRatio <- (geneHits / geneTotal )/ (bgHits / bgTotal)

    print(ggplot(plotDf, aes(x = -log10(p.adjust),
                             y = reorder(Description, -p.adjust),
                             size = Count)) +
      geom_point(aes(color = oddsRatio)) +
      scale_color_viridis_c() +
      labs(title = names(enrichmentResults)[i],
           x = expression(-log[10]("adjusted p-value")),
           y = NULL,
           size = "n significant",
           color = "odds ratio") +
      theme_minimal(base_size = 12) +
      theme(
        axis.text.y = element_text(size = 10),
        axis.text.x = element_text(angle = 45, hjust = 1),
        plot.title = element_text(hjust = 0.5, face = "bold"),
        legend.position = "right"
      )
    )
  }
}
```

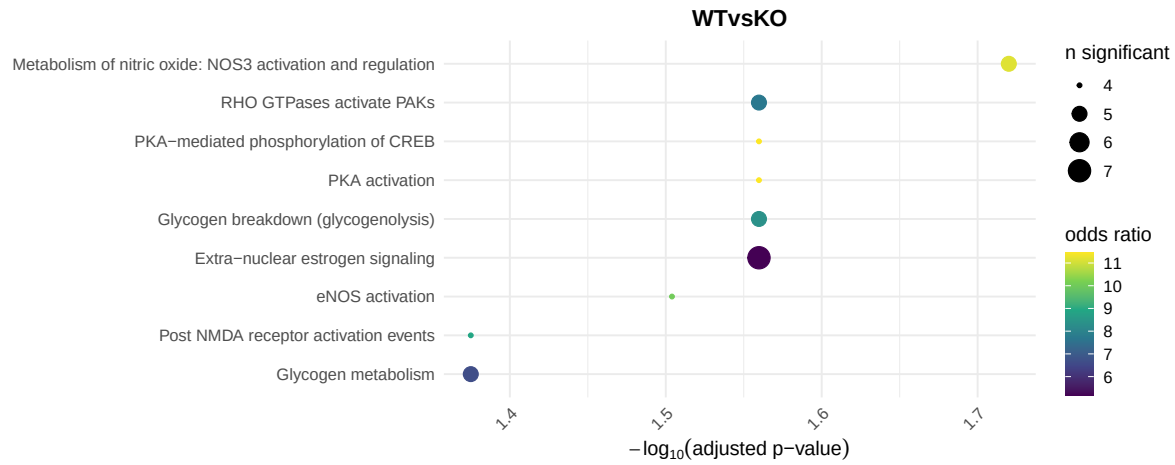

Figure 7. Enriched pathways for the WT:KO comparison, ordered by $-\log_{10}$ adjusted p-value. Point size indicates the number of significant proteins per pathway, and odds ratios were computed relative to the background protein set.

3.8 Protein Level MDS Plots

Proteins are plotted in two dimensions, with point colors reflecting statistical significance status. Significant proteins are highlighted in red, while non-significant proteins are shown in gray.

```
p1 <- plotMDSContrast(eset_unfiltered, topTab_WTvsKO, c("WT", "KO"),
                      "WT vs KO")
```

```
p2 <- plotMDSContrast(eset_unfiltered, topTab_KO.IVvsKO, c("KO-IV", "KO"),
                      "KO-IV vs KO")
```

```
p3 <- plotMDSContrast(eset_unfiltered, topTab_KO.NEBvsKO, c("KO-NEB", "KO"),
                      "KO-NEB vs KO")
```

```
combined_MDS <- ggarrange(p1, p2, p3,
                          nrow = 3,
                          common.legend = TRUE,
                          legend = "right")

# Display the result
print(combined_MDS)
```

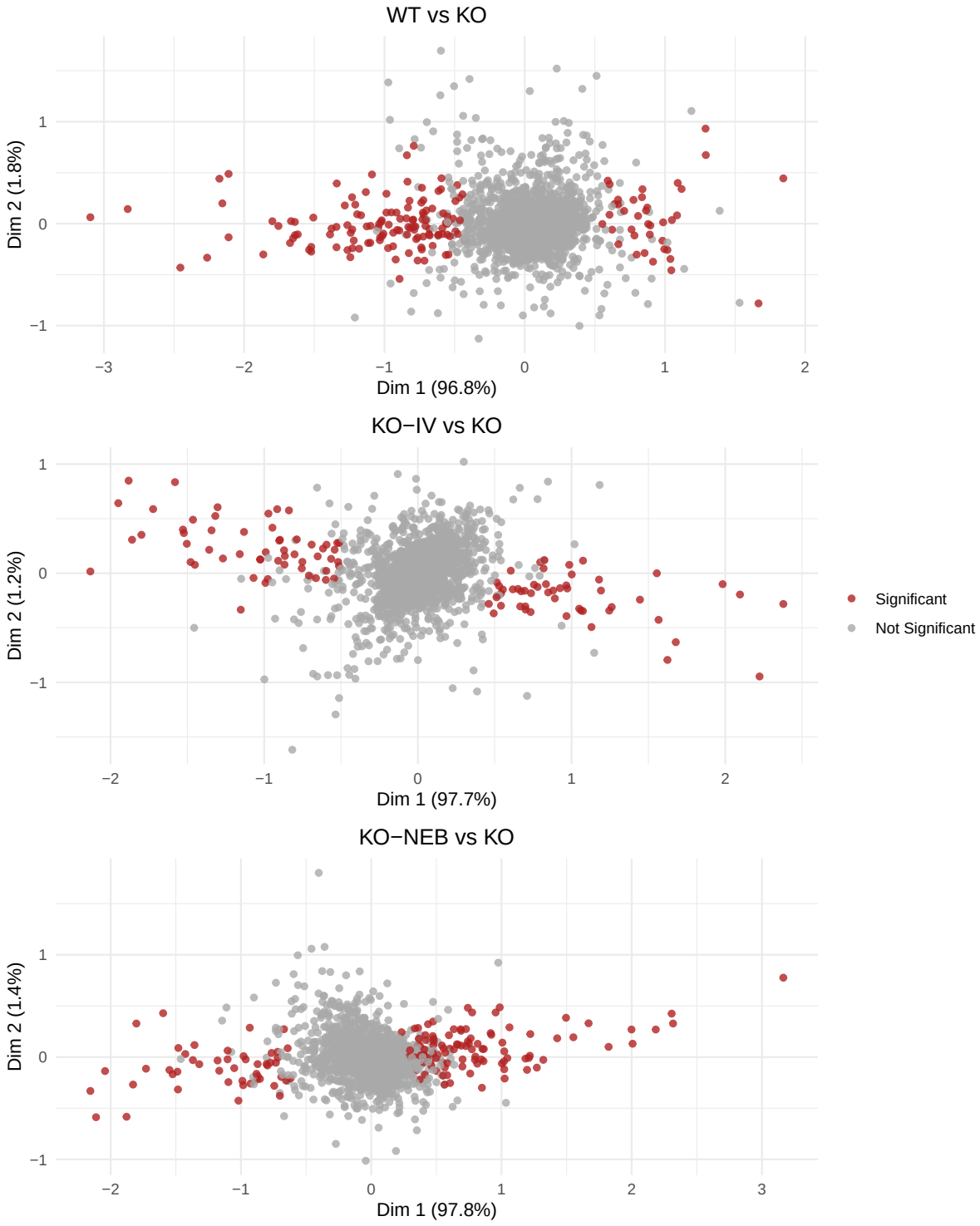

Figure 8. Protein-Level MDS plots for each contrast group, with points representing individual

proteins colored by significance. For each contrast, MDS was performed on the subset of proteins without missing values, using leading log -fold-change distances between expression profiles across samples.

4. Conclusion

Through this project, I learned a proteomics workflow in R. This included data pre-processing and imputation, differential expression analysis and enriched pathway analysis. I built many visualizations, including volcano plots, MDS plots, venn diagrams, gene-concept network plots and enrichment analysis visualizations.

Some things I would do next time is outline the figure of my project before starting, and making changes as necessary. I would also keep track of my citations and add documentation as I go rather than doing it all at the end, because it is harder to remember my process after having already done it. I would also like to explore further imputation methods for proteomics data with missing values. A project I could research further would be to develop an R function that implements the empirical bayesian random censoring threshold model described in Koopmans et al. (2014) (Koopmans et al. 2014). I could also develop R functions for imputation methods described in Wang et al. (2017) (Wang et al. 2017), and access how they perform against each other for differential expression analysis through `limma`.

5. References

Bioconductor. n.d. "plotMDS Function (Limma Package, Version 3.28.14)." https://www.rdocumentation.org/packages/limma/versions/3.28.14/topics/plotMDS.

Gentleman, Robert, Vincent Carey, Douglas Bates, Ben Bolstad, Mark Dettling, Sandrine Dudoit, Byron Ellis, et al. 2004. "Bioconductor: Open Software Development for Computational Biology and Bioinformatics." *Genome Biology* 5 (10): R80. https://doi.org/10.1186/gb-2004-5-10-r80.

Gu, Zuguang, Roland Eils, and Matthias Schlesner. 2016. "Complex Heatmaps Reveal Patterns and Correlations in Multidimensional Genomic Data." *Bioinformatics* 32 (18): 2847–49. https://doi.org/10.1093/bioinformatics/btw313.

Huber, Wolfgang, Vincent J. Carey, Robert Gentleman, Simon Anders, Marc Carlson, Benilton S. Carvalho, Hector Corrada Bravo, et al. 2015. "Orchestrating High-Throughput Genomic Analysis with Bioconductor." *Nature Methods* 12: 115–21. https://doi.org/10.1038/nmeth.3252.

Kammers, Kamil, Robert N. Cole, Calvin Tiengwe, and Ingo Ruczinski. 2015. "Detecting Significant Changes in Protein Abundance." *EuPA Open Proteomics* 7: 11–19. https://doi.org/10.1016/j.euprot.2015.02.002.

Koopmans, Freek, Lennart N Cornelisse, Tom Heskes, and Ton M Dijkstra. 2014. "Empirical Bayesian Random Censoring Threshold Model Improves Detection of Differentially Abundant Proteins." *Journal of Proteome Research* 13 (9): 3871–80. https://doi.org/10.1021/pr500171u.

Lazar, Cristian, Laurent Gatto, Michaël Ferro, Christophe Bruley, and Thomas Burger. 2016. "Accounting for the Multiple Natures of Missing Values in Label-Free Quantitative Proteomics Data Sets to Compare Imputation Strategies." *Journal of Proteome Research* 15 (4): 1116–25. https://doi.org/10.1021/acs.jproteome.5b00981.

Ligtenberg, Willem. 2019. *Reactome.db: A Set of Annotation Maps for Reactome*. https://bioconductor.org/packages/reactome.db/.

Pagès, Hervé, Marc Carlson, Seth Falcon, and Nianhua Li. 2023. *AnnotationDbi: Manipulation of SQLite-Based Annotations in Bioconductor*. https://bioconductor.org/packages/AnnotationDbi/.

Phipson, Belinda, Si Lee, Ian J Majewski, Warren S Alexander, and Gordon K Smyth. 2016. "Robust Hyperparameter Estimation Protects Against Hypervariable Genes and Improves Power to Detect Differential Expression." *Annals of Applied Statistics* 10 (2): 946–63. https://doi.org/10.1214/16-AOAS920.

Ritchie, Matthew E., Belinda Phipson, Di Wu, Yifang Hu, Charity W. Law, Wei Shi, and Gordon K. Smyth. 2015. "Limma Powers Differential Expression Analyses for RNA-Sequencing and Microarray Studies." *Nucleic Acids Research* 43 (7): e47. https://doi.org/10.1093/nar/gkv007.

Sanz, Ricardo Gonzalo, and Alex Sánchez-Pla. 2019. "Statistical Analysis of Microarray Data." Edited by Verónica Bolón-Canedo and Amparo Alonso-Betanzos, Methods in molecular biology, 1986: 87–121. https://doi.org/10.1007/978-1-4939-9442-7_5.

Schessner, Julia Patricia, Eugenia Voytik, and Isabell Bludau. 2022. "A Practical Guide to Interpreting and Generating Bottom-up Proteomics Data Visualizations." *Proteomics* 22 (8): e2100103. https://doi.org/10.1002/pmic.202100103.

Wang, Jianqiang, Lin Li, Ting Chen, et al. 2017. "In-Depth Method Assessments of Differentially Expressed Protein Detection for Shotgun Proteomics Data with Missing Values." *Scientific Reports* 7: 3367. https://doi.org/10.1038/s41598-017-03650-8.

Yu, Guangchuang, and Qing-Yu He. 2016. "ReactomePA: An r/Bioconductor Package for Reactome Pathway Analysis and Visualization." *Molecular BioSystems* 12 (2): 477–79. https://doi.org/10.1039/C5MB00663E.

Yu, Guangchuang, Li-Gen Wang, Yanyan Han, and Qing-Yu He. 2012. "clusterProfiler: An r Package for Comparing Biological Themes Among Gene Clusters." *OMICS: A Journal of Integrative Biology* 16 (5): 284–87. https://doi.org/10.1089/omi.2011.0118.