Multi-cytokine HRECs Markdown

Fergus McLellan

2024-10-23

Introduction

This R Markdown outlines the steps undertaken to perform differential expression and pathway analysis for the investigation of cytokine and DMFU-incuded signalling in HRECs. # Setup Work Space

To begin, I load tidyverse and run some code to clear my environment and set my desired information printing parameters

```
#rm(list = ls(all.names = TRUE)) unhash to clear environment
#gc() unhash to clear cache
options(max.print = .Machine$integer.max, scipen = 999, stringsAsFactors = F, dplyr.summarise.inform = F

library(tidyverse) #contains multiple packages for general tidiness
library(magrittr) #package adding useful functions for pipe operator
library(dtplyr) #package adding useful functions for pipe operator
library(textshaping)

set.seed(12345)
```

next, I set my and input/output paths

```
in_path <- "raw_data/" #set input path from working directory
out_path = "significant_data/" # set output path from working directory
```

Differential Expression Analysis

The following section will describe the steps undertaken for initial differential expression analysis, with the goal of producing: 1. Normalized counts 2. Principal component analysis 3. Significantly differentially expressed gene lists

This markdown will use analysis in Part 2 as an example, however the same techniques were applied for Part 1 with names swapped to match experimental groups used (Control DMFu VEGF VEGFDMFu).

This data was then exported as .CSV files and used in **morpheus to generate count heatmaps** (*Figure 1B*) and **biovenn for venn diagram generation**.

DESeq2

Differential expression analysis using DESEQ2

There are 3 objects required for DESeq to run properly:

1. A *Counts* matrix, with one 1 set of labels (Ensembl ID in this case) as the row name
2. A *Design* factor with 1 level per treatment, and repetitions = n
3. A *colData* data frame containing a column reflecting the design

Data and meta data

```
Counts <-read.delim(paste0(in_path,"raw_counts.csv"), header = TRUE, row.names = 1, sep = ",")

Counts <- as.matrix(Counts[,1:48])

head(Counts)
```

Counts matrix

```
##                 Control_1 Control_2 Control_3 Control_4 Control_5 Control_6
## ENSG00000223972         0         0         0         0         0         0
## ENSG00000227232       100        90       110        99       134       112
## ENSG00000278267        18        12        16        22        24        10
## ENSG00000243485         0         0         0         0         0         0
## ENSG00000237613         0         0         0         0         0         0
## ENSG00000268020         0         0         0         0         0         0
##                 TGFb1_1 TGFb1_2 TGFb1_3 TGFb1_4 TGFb1_5 TGFb1_6 TGFb2_1 TGFb2_2
## ENSG00000223972       0       0       0       0       0       0       0       0
## ENSG00000227232      76     110     137      92     136      92      88     108
## ENSG00000278267      26      19      27      30      12      22      16      18
## ENSG00000243485       0       0       0       0       0       0       0       0
## ENSG00000237613       0       0       0       0       0       0       0       0
## ENSG00000268020       0       0       0       0       0       0       0       0
##                 TGFb2_3 TGFb2_4 TGFb2_5 TGFb2_6 TNFa_1 TNFa_2 TNFa_3 TNFa_4
## ENSG00000223972       0       0       0       0      0      0      0      0
## ENSG00000227232      92     126     128     102     97     85    135    103
## ENSG00000278267      14      21      35      13     14     14     22     13
## ENSG00000243485       0       0       0       0      0      0      0      0
## ENSG00000237613       0       0       0       0      0      0      0      0
## ENSG00000268020       0       0       0       0      0      0      0      0
##                 TNFa_5 TNFa_6 Thrombin_1 Thrombin_2 Thrombin_3 Thrombin_4
## ENSG00000223972      0      0          0          0          0          0
## ENSG00000227232    106    115        106        116         89         60
## ENSG00000278267     13     11         12         18         24         19
## ENSG00000243485      0      0          0          0          0          0
## ENSG00000237613      0      0          0          0          0          0
## ENSG00000268020      0      0          0          0          0          0
##                 Thrombin_5 Thrombin_6 IL.6_1 IL.6_2 IL.6_3 IL.6_4 IL.6_5 IL.6_6
## ENSG00000223972          0          1      0      0      0      0      0      0
## ENSG00000227232         93         93     68    127    105     72    107     80
## ENSG00000278267         14         14     16     24      9     22     17     21
## ENSG00000243485          0          0      0      0      0      0      0      0
## ENSG00000237613          0          0      0      0      0      0      0      0
## ENSG00000268020          0          0      0      0      0      0      0      0
##                 VEGF_1 VEGF_2 VEGF_3 VEGF_4 VEGF_5 VEGF_6 All_1 All_2 All_3
```

```
## ENSG00000223972      0      0      0      0      1      0      0      0      0
## ENSG00000227232    103     70     75     89    100    108     98     80    113
## ENSG00000278267      5     15     21     22     15     11     21     23     11
## ENSG00000243485      0      0      0      0      0      0      0      0      0
## ENSG00000237613      0      0      0      0      0      0      0      0      0
## ENSG00000268020      0      0      0      0      0      0      0      0      0
##                 All_4 All_5 All_6
## ENSG00000223972     0     0     0
## ENSG00000227232   112    72    91
## ENSG00000278267    24    16    23
## ENSG00000243485     0     0     0
## ENSG00000237613     0     0     0
## ENSG00000268020     0     0     0
```

Meta Data So as to keep the global environment neat, a custom function was written to generate meta data based on the investigation being performed.

```
#Possible comparison types: all_groups, fibro, inflam, angio, single *IF SINGLE MUST ALSO IDENTIFY TREA
metadatapicker <- function(comparisontype, treatment = NULL){



  if(comparisontype == "important"){
    condition <- factor(c(rep("Control", 6), rep("TGFb2", 6), rep("TNFa", 6), rep("Thrombin", 6), rep("
    coldata <- data.frame(condition)
    coldata$condition <- relevel(coldata$condition, ref = "Control")

    # All Condition ------------------------------------------------------

  }else if (comparisontype == "all"){

    condition <- factor(c(rep("Control", 6), rep("TGFb1", 6), rep("TGFb2", 6), rep("TNFa", 6), rep("Thr
    coldata <- data.frame(condition)
    coldata$condition <- relevel(coldata$condition, ref = "Control")

    # Fibro Condition ------------------------------------------------------

  }else if (comparisontype == "fibro"){

    condition <- factor(c(rep("Control", 6), rep("TGFb1", 6), rep("TGFb2", 6), rep("TNFa", 6), rep("Thr
    coldata <- data.frame(condition)
    coldata$condition <- relevel(coldata$condition, ref = "Control")

    # Inflam Condition ------------------------------------------------------

  }else if (comparisontype == "inflam"){

    condition <- factor(c(rep("Control", 6), rep("TGFb1", 6), rep("TNFa", 6), rep("Thrombin", 6) , rep(
    coldata <- data.frame(condition)
    coldata$condition <- relevel(coldata$condition, ref = "Control")

    # Angio Condition ------------------------------------------------------
```

```
  }else if (comparisontype == "angio"){

    condition <- factor(c(rep("Control", 6), rep("TGFb2", 6), rep("Thrombin", 6) ,rep("VEGF", 6)))
    coldata <- data.frame(condition)
    coldata$condition <- relevel(coldata$condition, ref = "Control")

    # Individuals only (no cocktail) ------------------------------------------------------------

  }else if (comparisontype == "individuals"){
    condition <- factor(c(rep("Control", 6), rep("TGFb1", 6), rep("TGFb2", 6), rep("TNFa", 6), rep("Thr
    coldata <- data.frame(condition)
    coldata$condition <- relevel(coldata$condition, ref = "Control")


    # Single Comparison Condition ---------------------------------------------

  }else if (comparisontype == "single"){

    if(!is.null(treatment)){
      condition <- factor(c(rep("Control", 6), rep(treatment, 6)))
      coldata <- data.frame(condition)
      coldata$condition <- relevel(coldata$condition, ref = "Control")

    }else{
      print("Please select a treatment for your single comparison. Returning only Control group in data
    }

  }else{
    print("Invalid selection. Please choose from the list of possible comparisons.")
    return(NULL)
  }

  return(list("condition" = condition, "coldata" = coldata,"name"= paste0(comparisontype,"_",treatment))

}
```

Use function to set meta data. Allows: - all - individuals - important - fibro - inflam - angio - single*

*single comparisons requires selection of the treatment of interest

Assign a list for the meta data objects and split this list to give a condition factor and coldata data frame object

```
metadata <- metadatapicker(comparisontype = "all",
                           treatment =
                            ) %>%
               list2env(.GlobalEnv)
```

```
levels(condition)
```

Condition Factor

```
## [1] "ALL"      "Control"  "IL6"      "TGFb1"    "TGFb2"    "Thrombin" "TNFa"
## [8] "VEGF"
```

```
unique(coldata)
```

colData Data frame

```
##    condition
## 1    Control
## 7      TGFb1
## 13     TGFb2
## 19      TNFa
## 25  Thrombin
## 31       IL6
## 37      VEGF
## 43       ALL
```

Data Cleaning

Next, it is good practice to clean the data, removing any NA values, or unusable counts. For this I have used the WGCNA package goodSamplesGenes function.

```
library(WGCNA, quietly = TRUE) #Weighted Gene Co-expression Analysis package containing GoodSamplesGene

ddsCounts <- Counts #create duplicate of Counts before manipulating counts

gsg <- goodSamplesGenes(t(ddsCounts))
```

```
##  Flagging genes and samples with too many missing values...
##   ..step 1
##   ..step 2
```

```
summary(gsg)
```

```
##             Length Class  Mode
## goodGenes   57500  -none- logical
## goodSamples    48  -none- logical
## allOK           1  -none- logical
```

```
gsg$allOK #false in all ok indicates NA Values/iterative 0 values
```

```
## [1] FALSE
```

```
table(gsg$goodGenes) # individual gene check (iterative check for repeated 0 values)?
```

```
##
## FALSE  TRUE
## 18955 38545
```

```
table(gsg$goodSamples) #sample check
```

```
##
## TRUE
##   48
```

```
ddsCounts <- ddsCounts[gsg$goodGenes == TRUE, gsg$goodSamples == TRUE] #omit bad genes
```

Run DESeq2

DESeq2 was utilised to perform repeated wald tests to identify significantly differentially expressed genes. Several output objects were generated for future use: - DESeq object - Normalised counts data frame - Averaged normalised counts data frame - Variance stabilised counts data frame

```
dds <- DESeqDataSetFromMatrix(countData = ddsCounts, colData = coldata, design= ~condition)

dds <- DESeq(dds)
res <- results(dds) #results object example
```

```
## [1] "38545 genes examined, example results object:"
```

```
## log2 fold change (MLE): condition VEGF vs Control
## Wald test p-value: condition VEGF vs Control
## DataFrame with 6 rows and 6 columns
##                    baseMean log2FoldChange     lfcSE       stat     pvalue
##                   <numeric>      <numeric> <numeric>  <numeric>  <numeric>
## ENSG00000223972  0.0444196       0.370065  5.889216  0.0628377 0.94989571
## ENSG00000227232 98.8416405      -0.355984  0.122943 -2.8955075 0.00378546
## ENSG00000278267 17.8343074      -0.282354  0.294598 -0.9584400 0.33784093
## ENSG00000238009  0.1769574      -1.072555  3.354243 -0.3197608 0.74914968
## ENSG00000233750  0.5498715      -0.651585  1.742873 -0.3738567 0.70851090
## ENSG00000268903  4.8581692      -0.266914  0.498427 -0.5355127 0.59229540
##                       padj
##                  <numeric>
## ENSG00000223972        NA
## ENSG00000227232 0.0283983
## ENSG00000278267 0.6009277
## ENSG00000238009        NA
## ENSG00000233750        NA
## ENSG00000268903        NA
```

Normalised counts can be exported as .csv for use in web tools, or as an RDS using:

```
#saveRDS(dds, file= paste0(in_path,name,"dds.RDS"))  #unhash to save RDS
```

Generating an average, normalised counts data frame and variance stabilised transformation data frame

```
avg_ncounts <- sapply(seq(1, ncol(normcounts), 6), function(j) rowMeans(normcounts[,j+(0:(5))])) %>% #s
  as.data.frame()%>% #represents the result as a data frame
  set_colnames(levels(condition)) #sets the column names to match the levels of condition


vsdata <- vst(dds, blind= FALSE) #variance stablising transformation for PCA
vscounts <- assay(vsdata) #variance stabilised counts
```

Principal Component Analysis

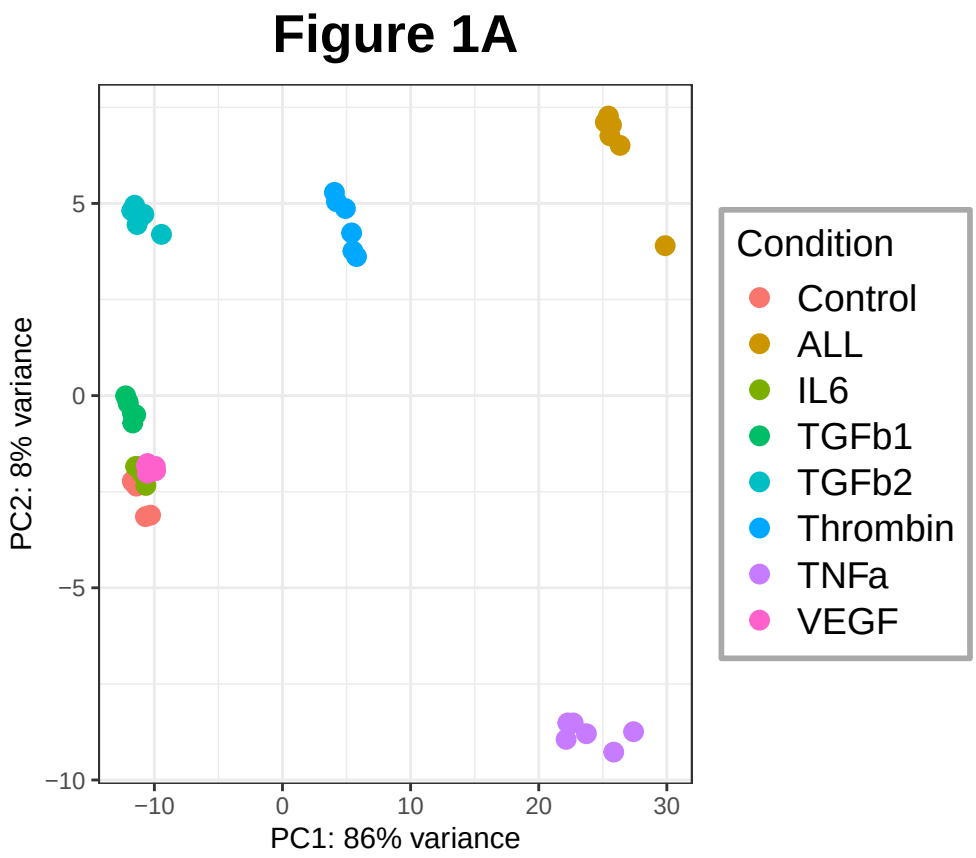
Load Necessary Packages

```
library(RColorBrewer) #color palettes
library(ggplot2) #plotting package
library(ggrepel) #text and point repel
library(svglite) #save SVGs
```

Plot PCA in ggplot

Generate PCA using ggplot2 **plotPCA**

```
PCA <- plotPCA(vsdata, intgroup = "condition", ntop = 1000)+
  theme_bw()+ #base theme
  coord_equal(3)+ #set a plot size
  labs(title = "Figure 1A", color = "Condition")+ #labels
  theme(plot.title = element_text(hjust=0.5,vjust = 1.5, size = 20, face = "bold", color = "black"),
        legend.title = element_text(size=14), legend.text = element_text(size=14),
        legend.margin = margin(0.2, 0.2, 0.2, 0.2, "cm"),
        legend.spacing = unit(2, "cm"),
        legend.key.size = unit(1.2, "lines"),
        legend.background = element_rect(fill = ,linewidth = 1, linetype ="solid", colour = "darkgrey"))
```

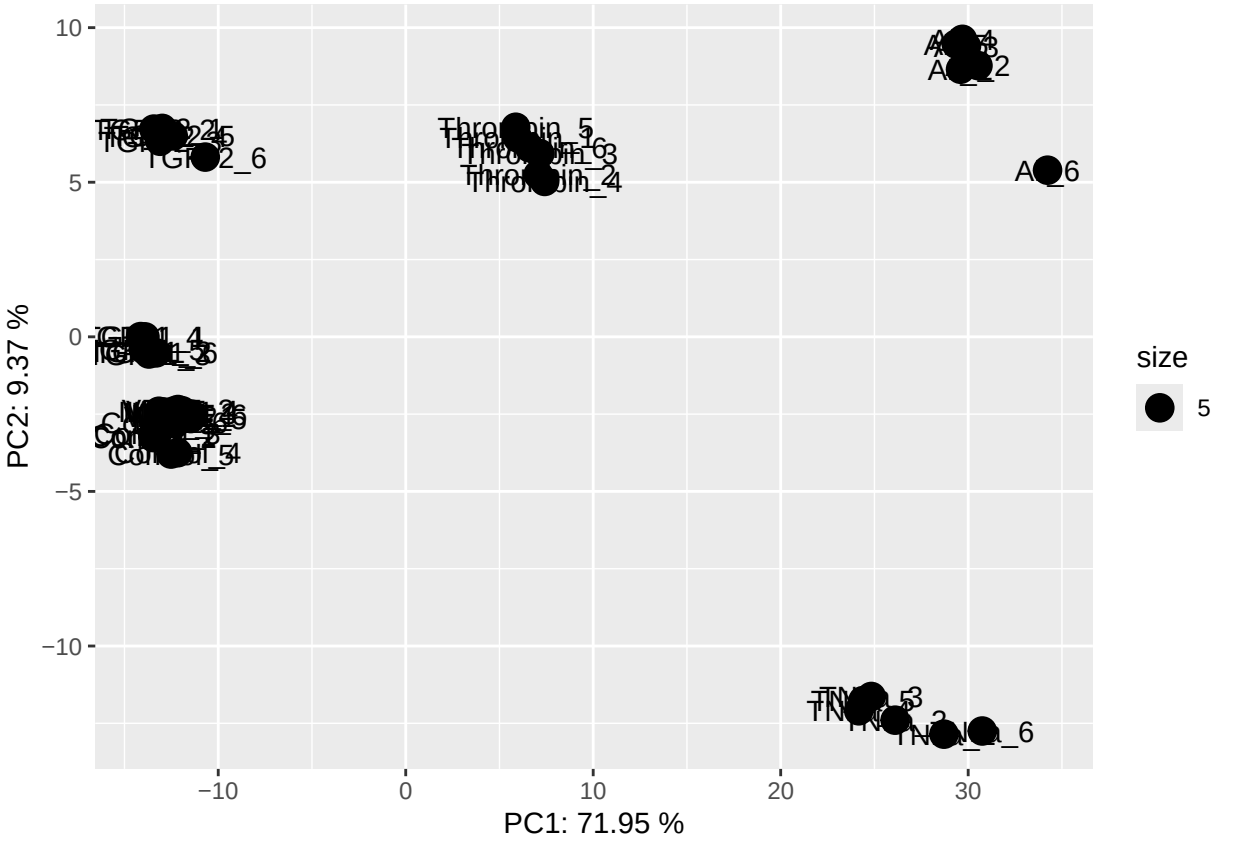


Further style changes were made in an external vector editor to increase legibility for **Figure 1A**, these include: - Adding of ellipses bounding points - Adjustments to color scheme - Removal of cocktail group

Numerical PCA

Kruskal test is only really exploratory, and is best applied when performing a single comparison to assess significance of difference between groups.

```
pca <- prcomp(t(vscounts))
pca.dat <- pca$x

pca.var <- pca$sdev^2 #variance as standard deviation squared

pca.var.percent <- round(pca.var/sum(pca.var)*100, digits = 2) #percentage of explained variance

pca.dat <- as.data.frame(pca.dat) #convert to dataframe so its usable

ggplot(pca.dat, aes(PC1, PC2)) + #check that the PCA being assessed is similar to PCAplot
  geom_point(aes(size = 5)) +
  geom_text(label = rownames(pca.dat)) +
  labs(x = paste0('PC1: ', pca.var.percent[1], ' %'),
       y = paste0('PC2: ', pca.var.percent[2], ' %'),
  )
```




```
P1 <- kruskal.test(pca.dat$PC1 ~ condition)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  pca.dat$PC1 by condition
## Kruskal-Wallis chi-squared = 43.111, df = 7, p-value = 0.0000003176
```

```
P2 <- kruskal.test(pca.dat$PC2 ~ condition)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  pca.dat$PC2 by condition
## Kruskal-Wallis chi-squared = 44.451, df = 7, p-value = 0.0000001748
```

Significance Check

Custom, loopable function to export a .csv of DESeq2 $\log_2$ Fold Change, with additional introduced columns: - HGNC gene symbols - ENTREZ gene IDs - Differential expression status (UP, Down or NO) referencing adjustable limits

```
library(org.Hs.eg.db) #read in annotation DBI human database


significance <- function(condition, treatment, Padj = 0.05, L2FC = 1, version){

  library(DESeq2)

  res <- results(dds, contrast = c(condition,treatment, "Control"), )

  sigdeg_df<- as.data.frame(res) #generate results data frame

  sigdeg_df$symbol<- mapIds(org.Hs.eg.db,
                            keys = rownames(sigdeg_df),
                            keytype = "ENSEMBL", column = "SYMBOL") #gene symbols
  sigdeg_df$ENTREZID<- mapIds(org.Hs.eg.db,
                              keys = rownames(sigdeg_df),
                              keytype = "ENSEMBL", column = "ENTREZID") #ENTREZ IDs

  sigdeg_df <- sigdeg_df %>%
    mutate(diffexpressed = case_when(
      log2FoldChange > +(L2FC) & padj < Padj ~ 'UP', #check if up regulated
      log2FoldChange < -(L2FC) & padj < Padj ~ 'DOWN', #check if down regulated
      padj > Padj ~ 'NO', #check if not significant
      TRUE ~ 'NO' #Label anything else as this (good for troubleshooting)
    ))


  write.csv(sigdeg_df, file = paste0(out_path,treatment,"_deg_table_",version,".csv"))

  print(paste0(treatment," DEG Table with L2FC cut-off = ",L2FC," and adjusted p=value cut-off = ",Padj
}
```

When running the function, ensure a condition has been established using *metadatapicker*

```
significance(condition = "condition",
             treatment = "ALL",
             L2FC = 1,
             Padj = 0.05,
             version = "1L2FC_cutoff_003")
```

```
## 'select()' returned 1:many mapping between keys and columns
## 'select()' returned 1:many mapping between keys and columns
```

```
## [1] "ALL DEG Table with L2FC cut-off = 1 and adjusted p=value cut-off = 0.05 generated in significan
```

```
#cytokines <- c("TGFb1","TGFb2","TNFa","Thrombin","IL6","VEGF","ALL")

#for (cytokine in cytokines) {
  #significance(condition = "all_groups_condition",
               #cytokine = cytokine,
               #version = "nobaseMean_cutoff_003")
  #}
```

Volcano Plots

Here, I utilise *enhancedVolcano* to generate a volcano plot, with the top 10 highest $\log_2$ Fold Change DEGs labelled.

This code will function irrespective of if above **Significance Check** has been run.

First, read in results object for comparison to be plotted

unhash function lines (397 + 487) to

```
library(EnhancedVolcano)
cytokine = "TGFb2"
version = "001"

#volcanomaker <- function(cytokine,version){
  set.seed(12345)

  res <- results(dds, contrast = c("condition", cytokine, "Control"))
  sigdeg <- na.omit(res)

  sigdeg_df<- as.data.frame(sigdeg)
```

Next, label with HGNC symbols, and assign Up, Down and NS labels with colors associated based on volcano plot parameters. This is not required in enhancedVolcano as the package will identify and label based on *Pcutoff* and *Fcutoff* values, however allows for the custom colour filtering used later.

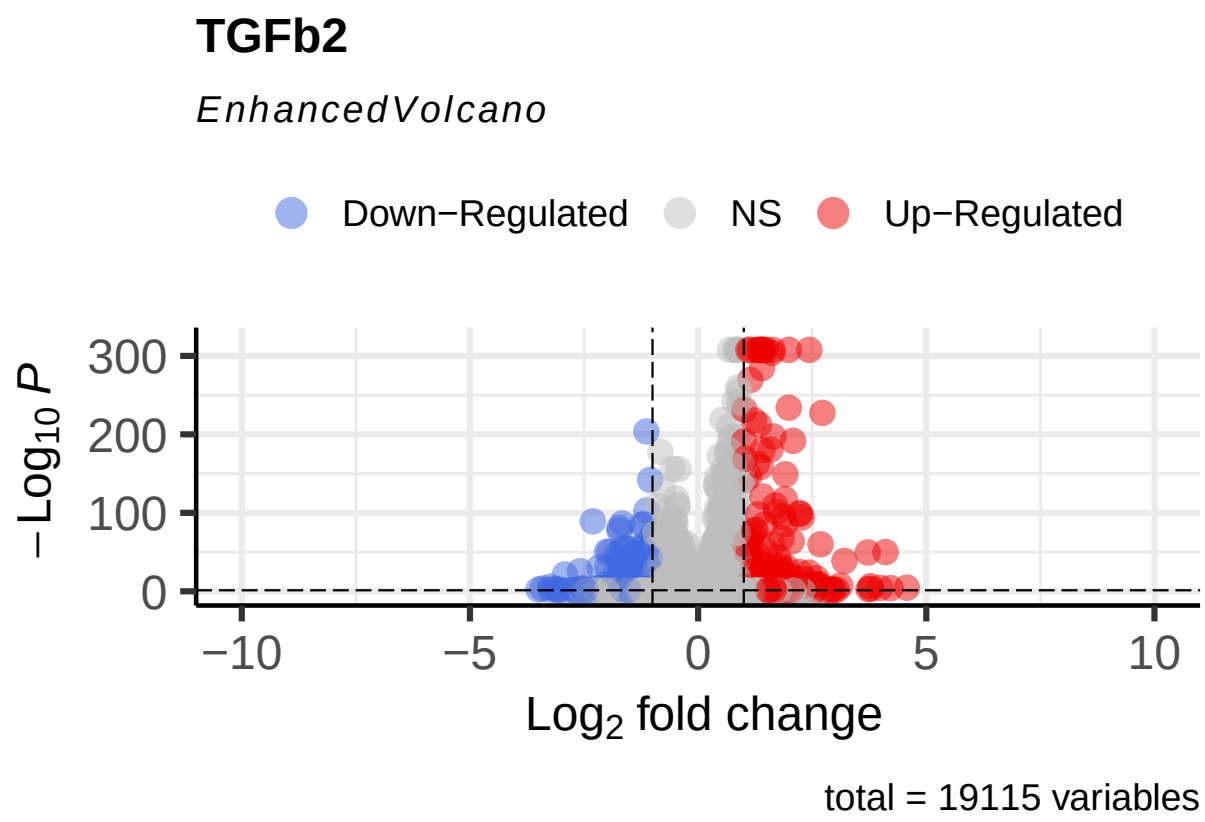
```
#Labelling system
  sigdeg_df$symbol<- mapIds(org.Hs.eg.db,
                            keys = rownames(sigdeg_df),
                            keytype = "ENSEMBL",
                            column = "SYMBOL")

  sigdeg_df <- sigdeg_df %>%
    mutate(diffexpressed = case_when(
      log2FoldChange > 1 & padj < 0.05 ~ "UP",
      log2FoldChange < -1 & padj < 0.05 ~ "DOWN",
      between(sigdeg_df$log2FoldChange, -1, 1) ~ "NO",
      padj > 0.05 ~ 'NO',
  ))

  sigdeg_df$padj <- ifelse(sigdeg_df$padj == 0, .Machine$double.xmin, sigdeg_df$padj) #ifelse to replac

keyvals <- ifelse(sigdeg_df$diffexpressed == "DOWN", 'royalblue',
                  ifelse(sigdeg_df$diffexpressed == "UP", 'red2','grey'))

names(keyvals)[keyvals == 'royalblue'] <- 'Down-Regulated'
names(keyvals)[keyvals == 'red2'] <- 'Up-Regulated'
names(keyvals)[keyvals == "grey"] <- 'NS'
```

Select a certain number of DEGs to be labelled, this section of code can be lengthened to slice separately for up and down-regulated genes by pulling separate slices based on *sigdeg_df$diffexpressed* value. Here, the top 10 overall were selected.

```
selected <- sigdeg_df %>%
  arrange(desc(abs(log2FoldChange))) %>%
  filter(diffexpressed != 'NO')%>%
  filter(!is.na(symbol))%>%
  dplyr::slice(1:10)
```

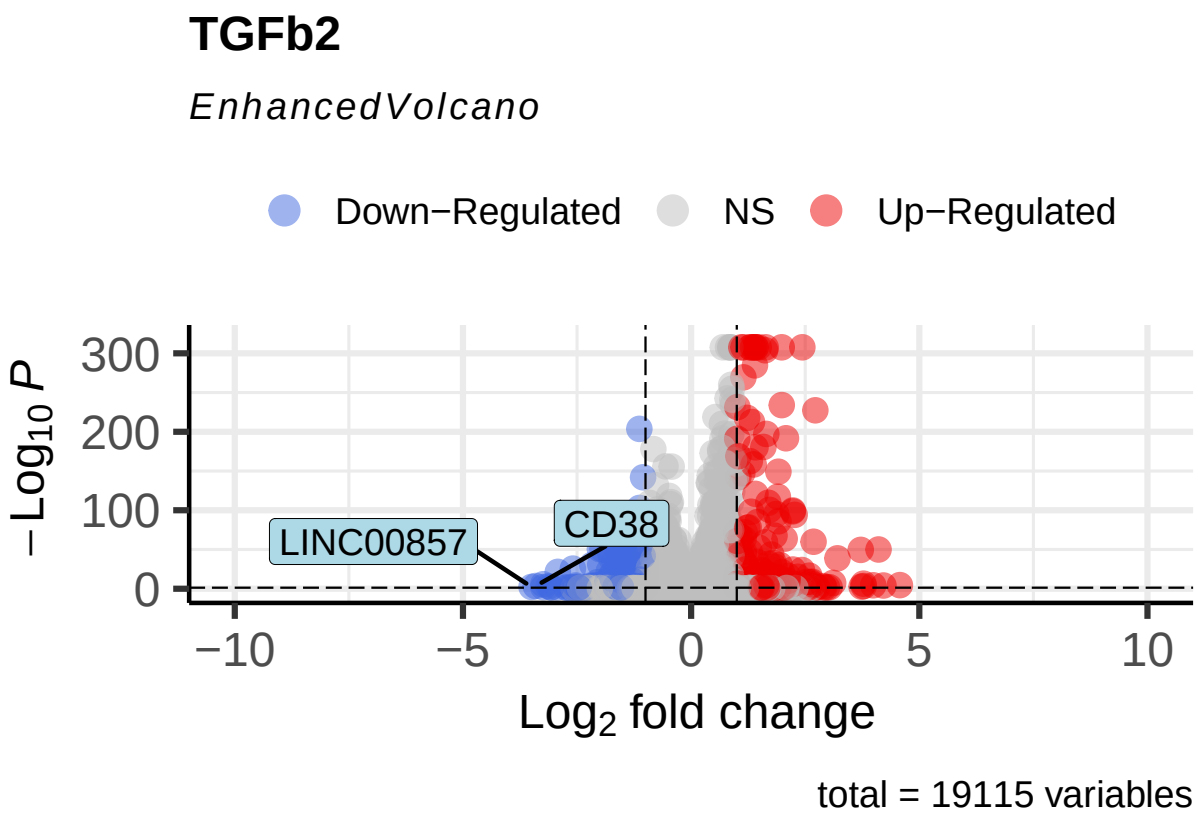
Generate volcano plot with no labels

```
volc <- EnhancedVolcano(sigdeg_df,
                        x = "log2FoldChange",
                        y = "padj",
                        lab = NA,  # Remove labels for now
                        title = cytokine, #title to match cytokine
                        colCustom = keyvals, #set colours to match keyed values
                        pCutoff = 0.05, # p-value cutoff abline
                        FCcutoff = 1, # Fold change cutoff abline
                        ylim = c(-2, 320), # Y-axis limits (set to machine limit)
                        xlim = c(-10, 10),  # Widen x-axis limits
                        pointSize = 4.0, #size of points
                        parseLabels = TRUE,
                        max.overlaps = 25)  # Allow more overlaps

volc
```



Append down-regulated labels

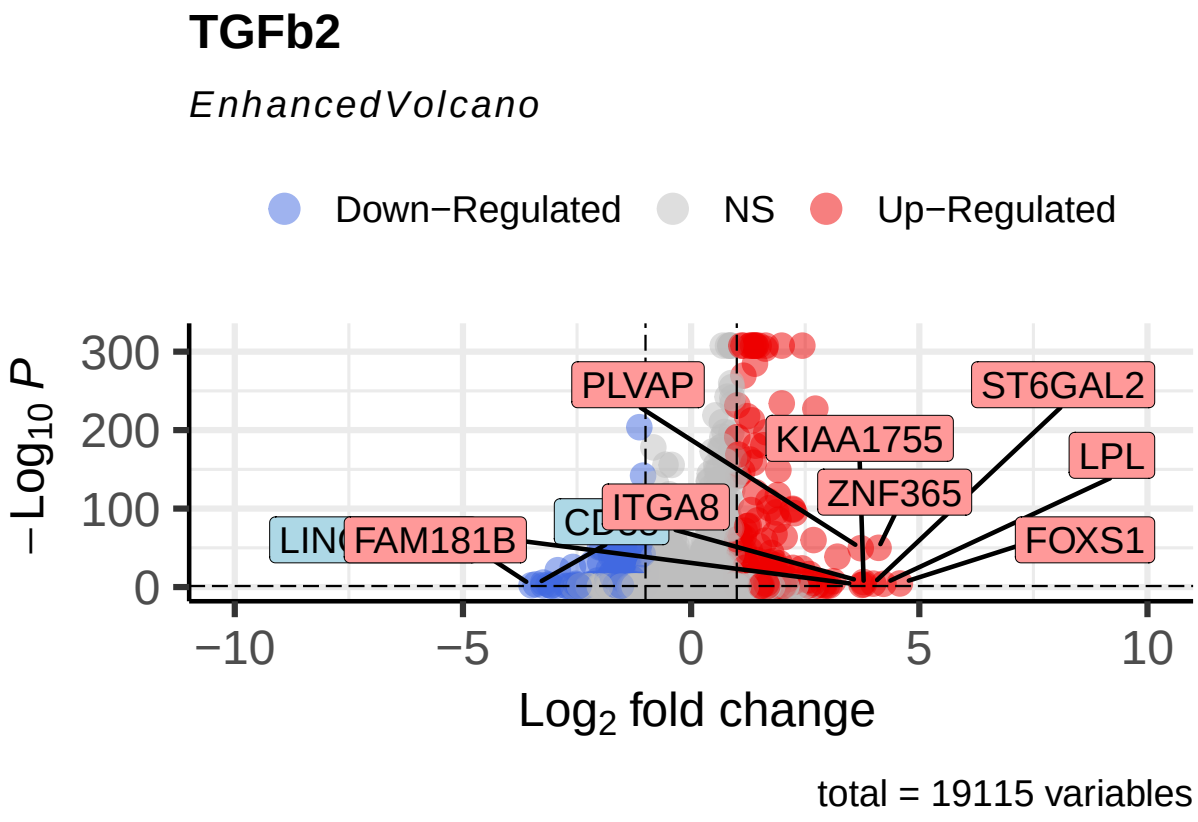
```
# Add labels using geom_text_repel
volc2 <- volc + geom_label_repel(data = sigdeg_df[sigdeg_df$log2FoldChange < -1 & sigdeg_df$padj <0.05 &
                                 aes(x = log2FoldChange, y = -log10(padj), label = symbol),
                                 size = 5,
                                 label.size = 0.1,
                                 box.padding = 1.0,  # Increase box padding
                                 point.padding = 0.5,  # Increase point padding
                                 segment.color = 'black',
                                 segment.size = 0.8,
                                 nudge_x = -2,
                                 max.time = 5,
                                 fill = "lightblue",
                                 nudge_y = 0.5,# Nudge labels to the left
                                 max.overlaps = 30)  # Ensure all labels are shown
```



Append up-regulated labels

```
# Add labels for points with positive log2FoldChange

volc3 <- volc2 + geom_label_repel(data = sigdeg_df[sigdeg_df$log2FoldChange > 1 & sigdeg_df$padj <0.05 &
                                  aes(x = log2FoldChange, y = -log10(padj), label = symbol),
                                  size = 5,
                                  label.size = 0.1,
                                  box.padding = 1,  # Increase box padding
                                  point.padding = 0.5,  # Increase point padding
                                  segment.color = 'black',
```

```
                                fill = "#FF9999",
                                segment.size = 0.8,
                                nudge_x = 1,
                                max.time = 5, # Nudge labels to the right
                                max.overlaps = 30)  # Ensure all labels are shown
```



Save and print text to show code has run through (good for if using a function)

```
ggsave(file =paste0(out_path,cytokine,"_volcano_001.svg"),
       plot=volc3, #plot
       width=13,   #width
       height=15)  #height

print(paste0("Volcano plot generated for ",cytokine,"."))
```

```
## [1] "Volcano plot generated for TGFb2."
```

```
#}
```

```
#volcanomaker(cytokine = "TNFa", version = "001")
```

Functional Analyses

The following sections of code outline packages used and steps undertaken to identify biological themes and pathways enriched by each treatment. This includes:

1. Over Representation Analysis
2. Gene Set Enrichment Analysis
3. Phenotype and pathway of interest filtering

Over-Representaiton Analysis

Pathway enrichment was performed in **clusterProfiler**, both by direct referencing to downloaded GMT files as well as integrated referencing through **clusterProfiler**.

Custom .GMT ORA

First, a pathway was assigned to reference lists folder.

```
ref_path <- "reference_data/"
```

Next, a background gene list was generated, using all HGNC/NCBI named genes in the raw data file.

```
library('org.Hs.eg.db')
library('clusterProfiler')

bg <- read.csv(paste0(in_path,"raw_counts.csv"), header = TRUE)
rownames(bg) <- bg$ID
bg <- bg[,c(1,48)]
bg <- bg %>%
  mutate("symbol" = mapIds(org.Hs.eg.db,
                           keys = rownames(bg),
                           keytype = "ENSEMBL",
                           column = "SYMBOL"))%>%
  mutate("ENTREZ" = mapIds(org.Hs.eg.db,
                           keys = rownames(bg),
                           keytype = "ENSEMBL",
                           column = "ENTREZID"))

bg <- na.omit(bg) %>%
  dplyr::select(ID, ENTREZ, symbol)

bgnames <- bg$symbol
```

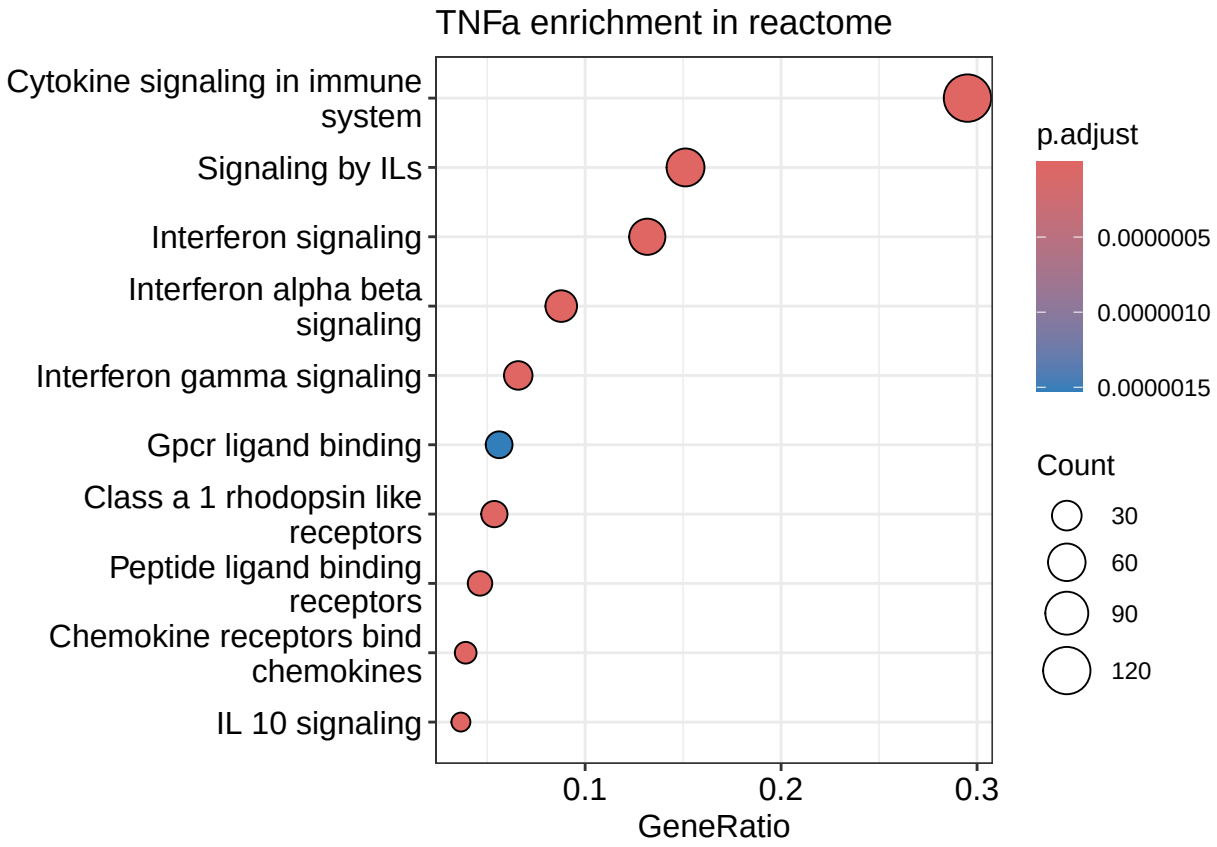
GMT files were downloaded from www.gsea-msigdb.org, and a loop was used to identify and create an RDS of overlapping genes between GMT and background list for easier reference.

```
    #subset genes in background from genes in gmt files,
          #these ones are from www.gsea-msigdb.org below uses a loop to identify and then make a new fi


     gmt_files <- list.files(path = ref_path, pattern = '.gmt', full.names = TRUE)
```

```
  for (i in gmt_files){
    file <- gmt_files[i] #set a variable for position in list
    pwl <- read.gmt(file)
    pwl <- pwf[pwf$gene %in% bgnames,]
    filename <- paste(gsub('c2.cp\\.', '',
                           gsub('.v2023.2.*$', '', file)), '.RDS', sep = '') #assumes you are using 2
    saveRDS(pwf, filename)
  }

    #NOTE: only need to do this once, have to manually rename RDS files to source.RDS as I can't figu
    #NOTE: genes may have multiple functions in the same GSEA set, therefore pwl may have more genes
    #NOTE: have to delete RDS files if you want to run it twice.
```

A custom function was written to then read in a chosen RDS, run over-representation analysis for up and down regulated significant DEGs at specified limits for a specified treatment, clean pathway names, and output a .CSV and dot plot of findings.

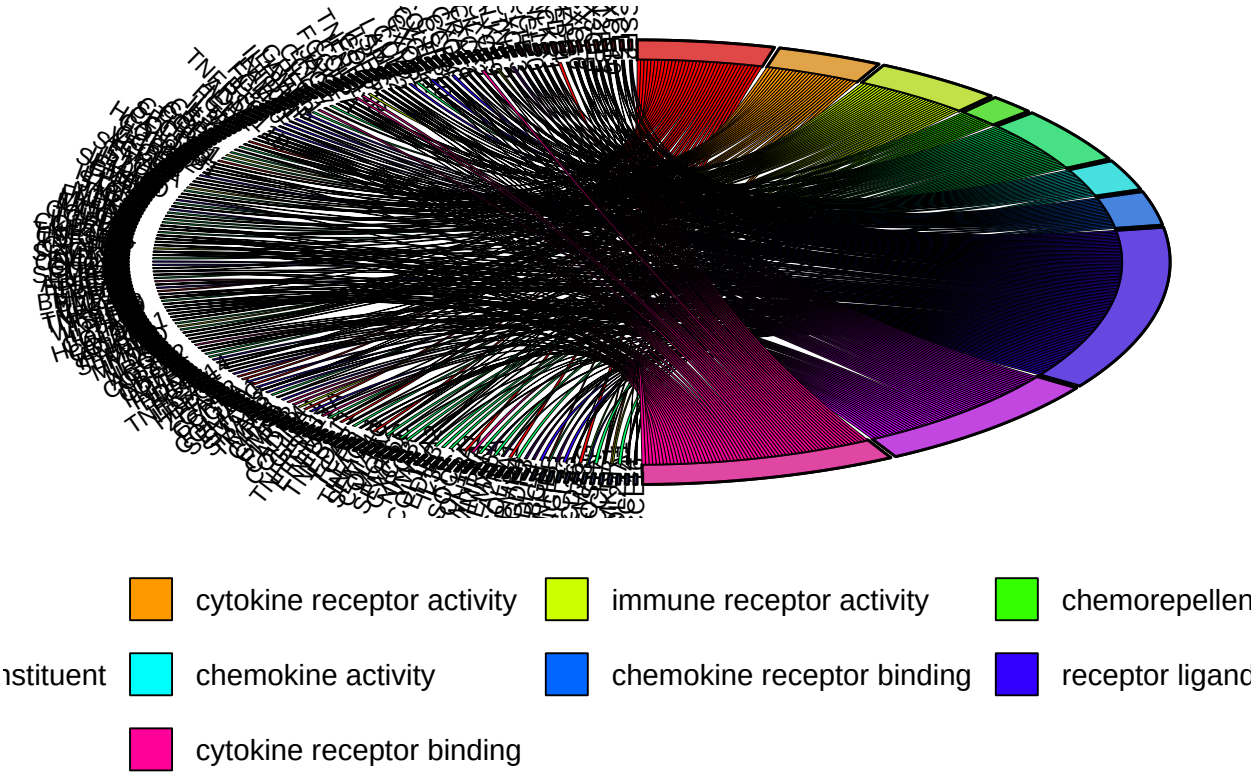
```
## [1] "reactome selected"
## [1] "Saving reactome clusterprofiler results for TNFa."
```



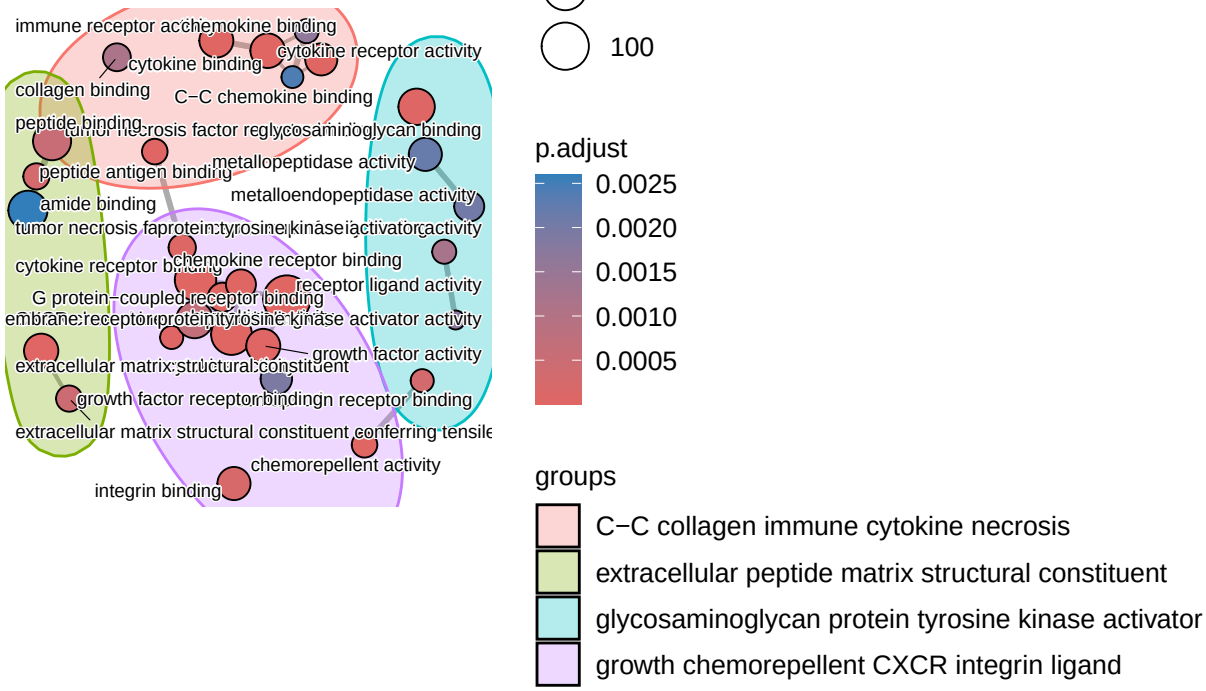
Although useful in allowing a wide array of pathway databases to be investigated simultaneously, this method does not integrate effectively for visualisation, and was thus only run to assess if a treatment was eligible for ORA enrichment and assess the pathway categories reflected by major databases.

###GOora and GOChord

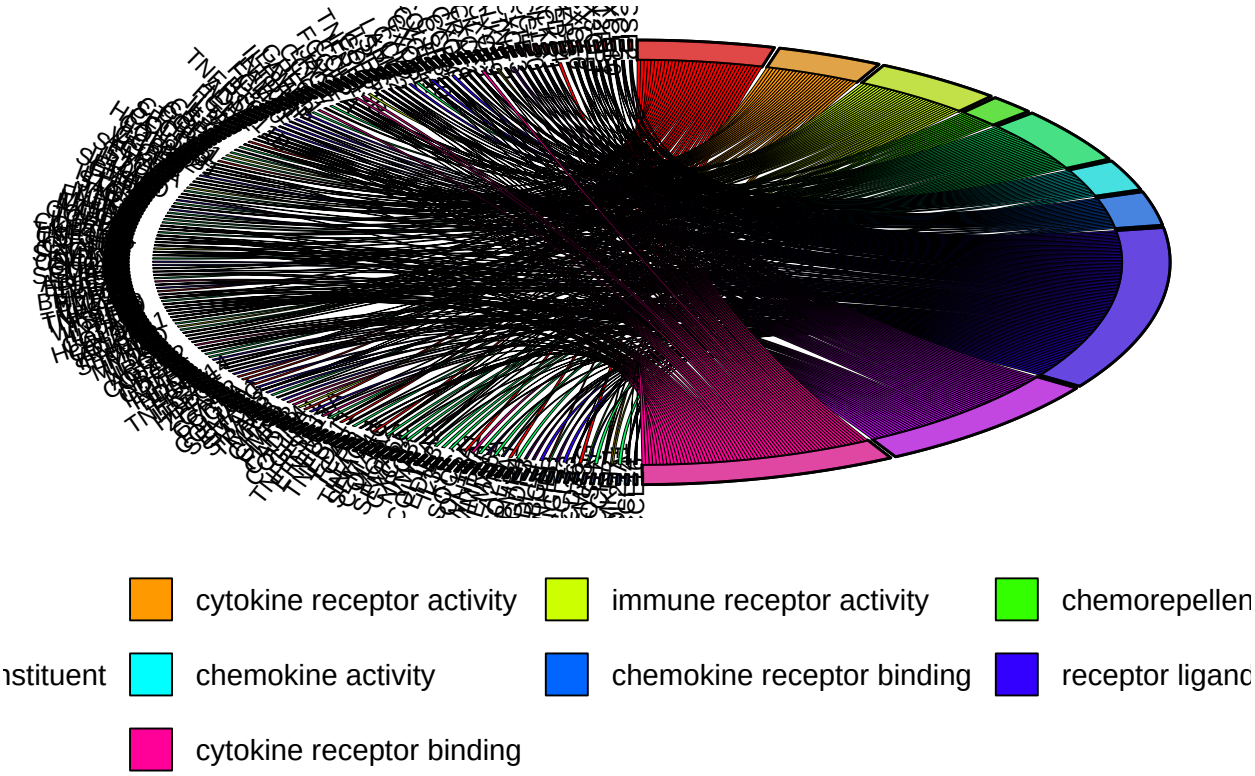
```
## Loading required package: ggdendro

## Loading required package: gridExtra

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:Biobase':
##
##     combine

## The following object is masked from 'package:BiocGenerics':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine
```



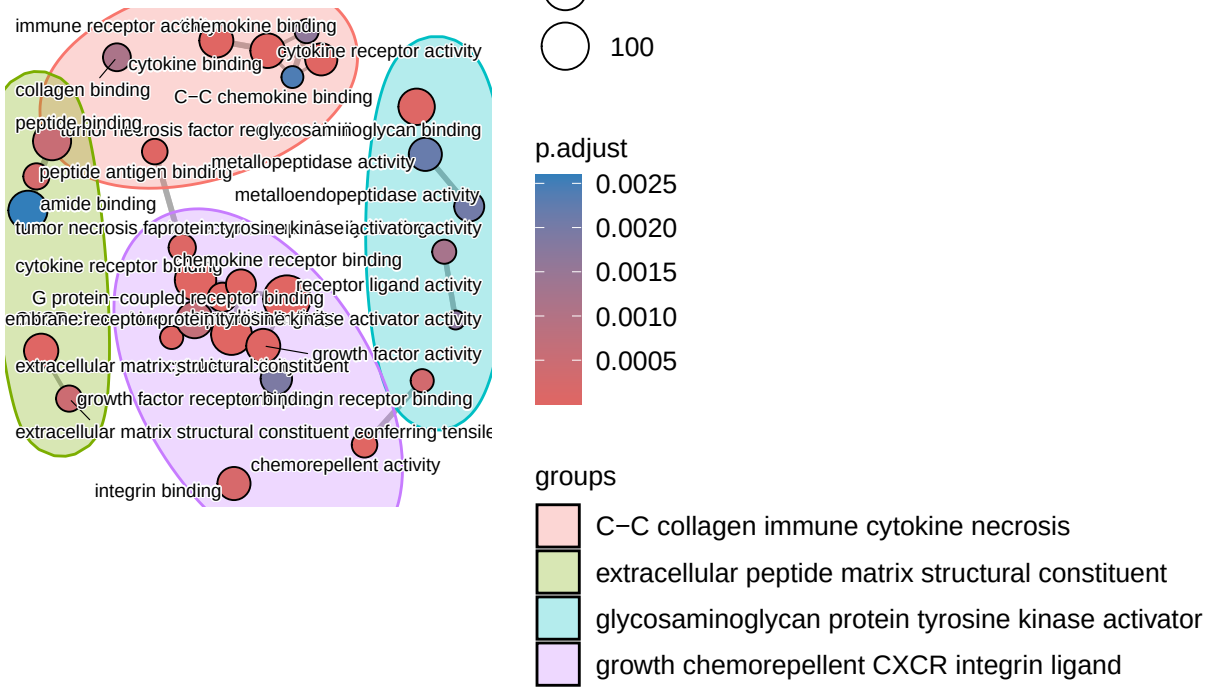

```
## [1] "Chord 1 Generated for TNFa."
```




```
## [1] "eplot 1 Generated for TNFa."
```




```
## [1] "Chord 2 Generated for TNFa."
```




```
## [1] "eplot 2 Generated for TNFa."
```