RNA-Seq baby-sitting Yifan

filter the counts.txt to counts_filter.txt, as we only need the read number for further analyze

```
counts = read.table("filter_star_counts.txt", header = TRUE,row.names = 1)

new_colnames <- c(
                  "N01", "N02", "N03", "N04",
                  "A2_R01", "A2_R02", "A2_R03", "A2_R04",
                  "A24_R01", "A24_R02", "A24_R03", "A24_R04")

colnames(counts) = new_colnames
# colnames(counts)
conditions <- factor(c(rep("Naive", 4), rep("Allo2h", 4), rep("Allo24h", 4)))
coldata <- data.frame(row.names=colnames(counts), condition=conditions)
# coldata

# check, suppose to get TRUE,indicates alignment between design and real data
all(rownames(coldata) %in% colnames(counts))
```

```
## [1] TRUE
```

```
rownames(coldata) == colnames(counts)
```

```
##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
# design DESeq matrix
dds = DESeqDataSetFromMatrix(countData = counts,
                             colData = coldata,
                             design = ~ condition)

dds = DESeq(dds)
```

```
## estimating size factors
```

```
## estimating dispersions
```

```
## gene-wise dispersion estimates
```

```
## mean-dispersion relationship
```

```
## final dispersion estimates
```

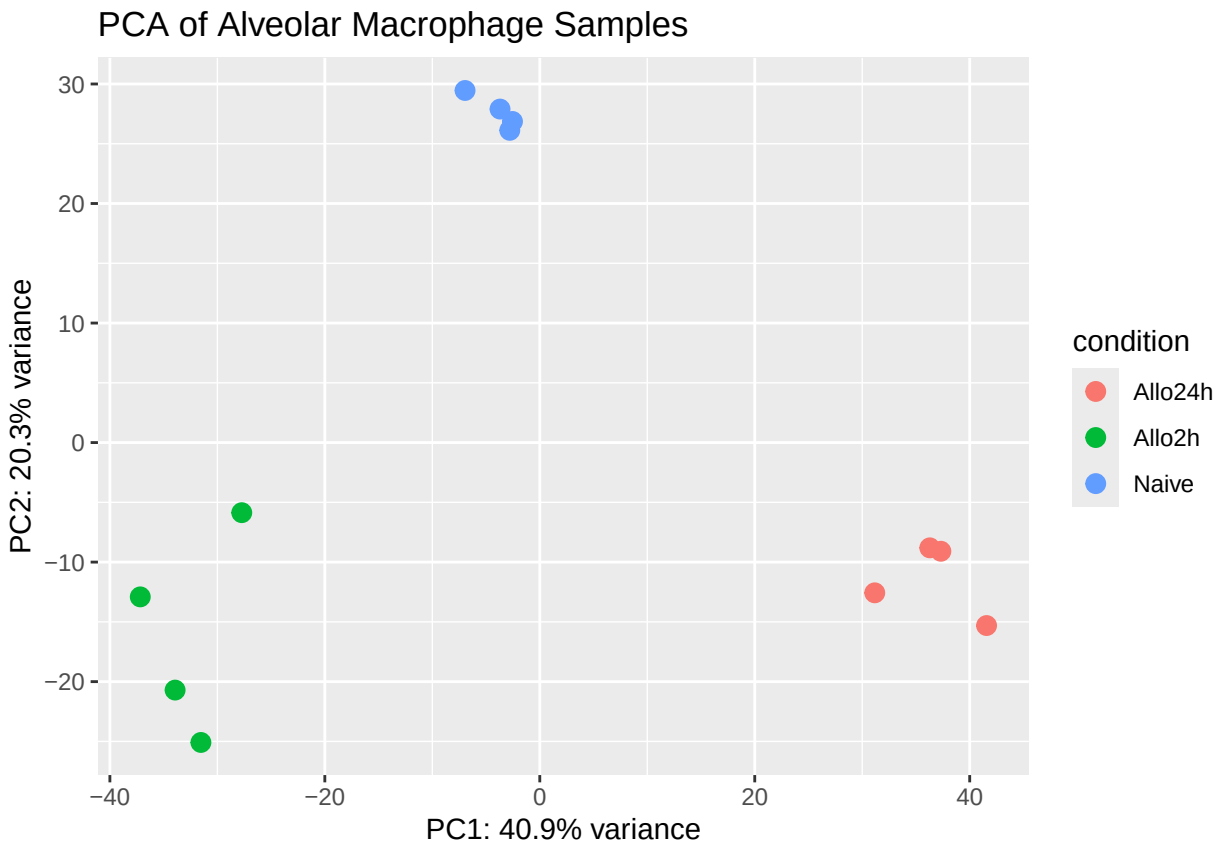
```
## fitting model and testing
```

```
length_table = read.table("S2_STAR2_practice_featurecounts.txt",  row.names = 1, header = TRUE, sep = "
# head(length_table)
mcols(dds)$basepairs = length_table[rownames(dds),"Length"]

normalized_counts <- counts(dds, normalized = TRUE)

cpm_values <- cpm(normalized_counts)

gene_lengths_kb <- mcols(dds)$basepairs / 1000

# Function to calculate RPKM from CPM and gene lengths
calculate_rpkm_from_cpm <- function(cpm_values, gene_lengths_kb) {
  rpkm <- sweep(cpm_values, 1, gene_lengths_kb, "/")
  return(rpkm)
```

```
}


rpkm <- calculate_rpkm_from_cpm(cpm_values, gene_lengths_kb)


log2_rpkm = log2(rpkm+1)
```

PCA and pearson

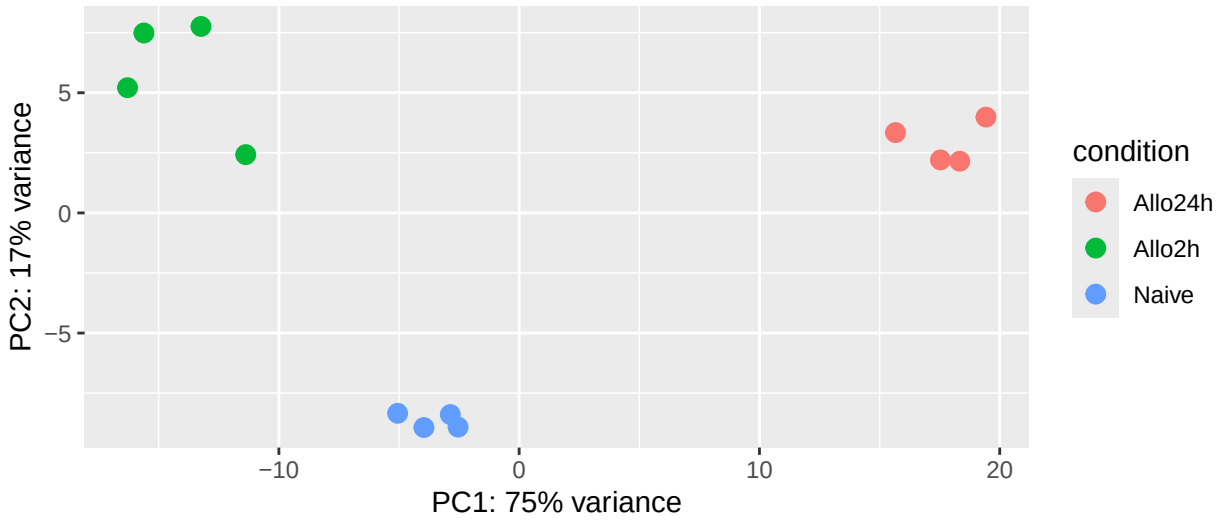
```
pca <- prcomp(t(log2_rpkm))

# PCA from rpkm normalized data

pca_data <- data.frame(PC1 = pca$x[,1], PC2 = pca$x[,2], condition = conditions)
ggplot(pca_data, aes(PC1, PC2, color = condition)) +
  geom_point(size = 3) +
  ggtitle("PCA of Alveolar Macrophage Samples") +
  xlab(paste0("PC1: ", round(summary(pca)$importance[2,1] * 100, 1), "% variance")) +
  ylab(paste0("PC2: ", round(summary(pca)$importance[2,2] * 100, 1), "% variance"))
```



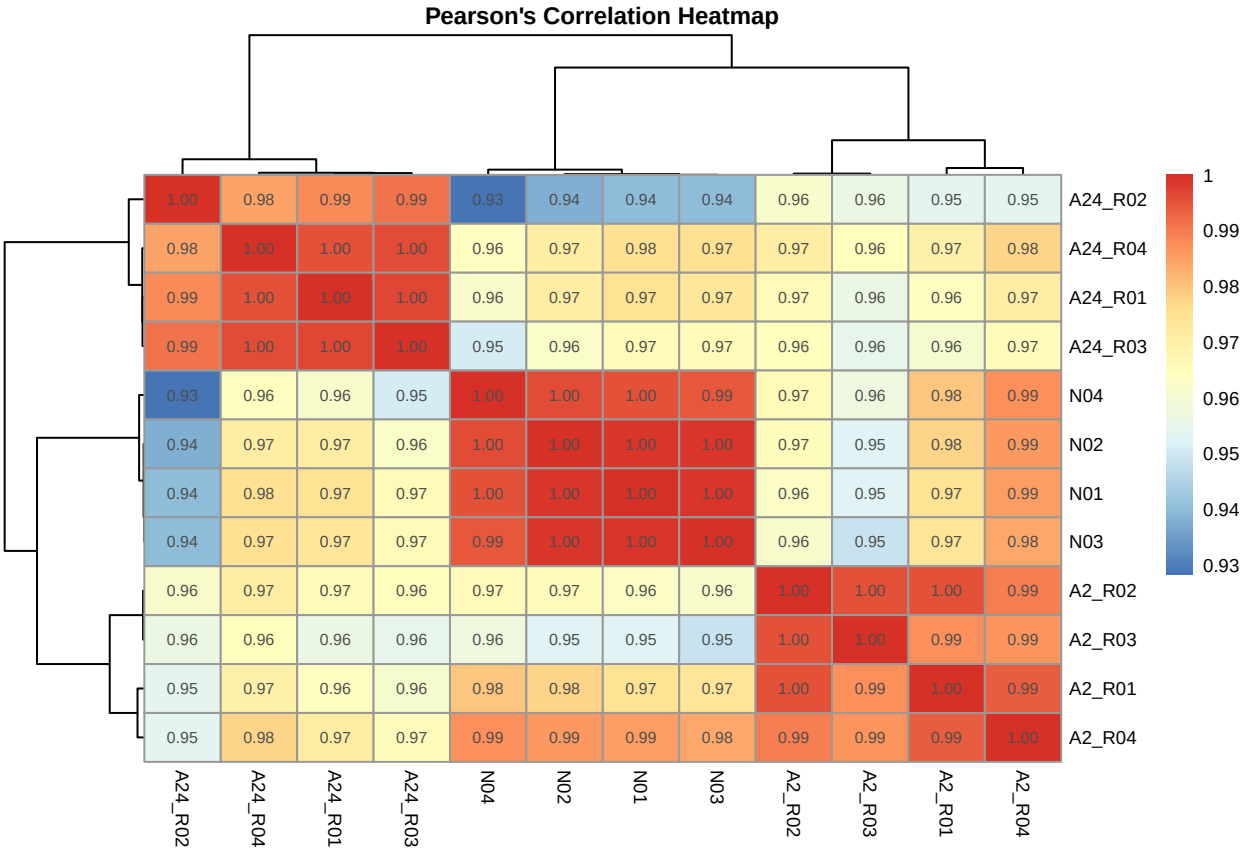

```
vsd = vst(dds,blind = FALSE)
## another PCA plot
pcaData <- plotPCA(vsd, intgroup="condition", returnData=TRUE)
```

```
## using ntop=500 top features by variance
```

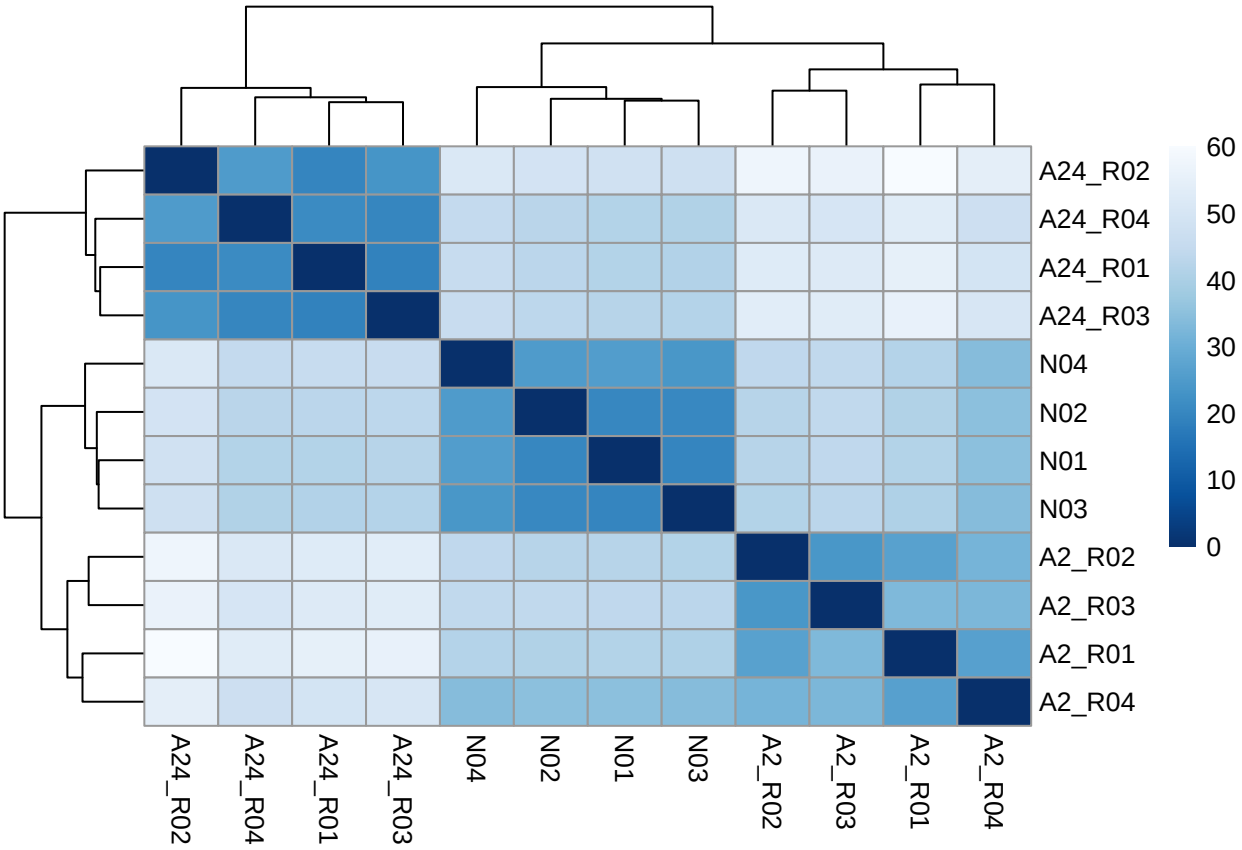
```
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(PC1, PC2, color=condition)) +
  geom_point(size=3) +
  xlab(paste0("PC1: ",percentVar[1],"% variance")) +
  ylab(paste0("PC2: ",percentVar[2],"% variance")) +
  coord_fixed()
```




```
# Compute Pearson correlation
cor_matrix <- cor(rpkm, method = "pearson")

# Create correlation pearson heatmap
pheatmap(cor_matrix,
         clustering_distance_rows = "correlation",
         clustering_distance_cols = "correlation",
         display_numbers = TRUE,
         fontsize = 7,
         main = "Pearson's Correlation Heatmap")
```



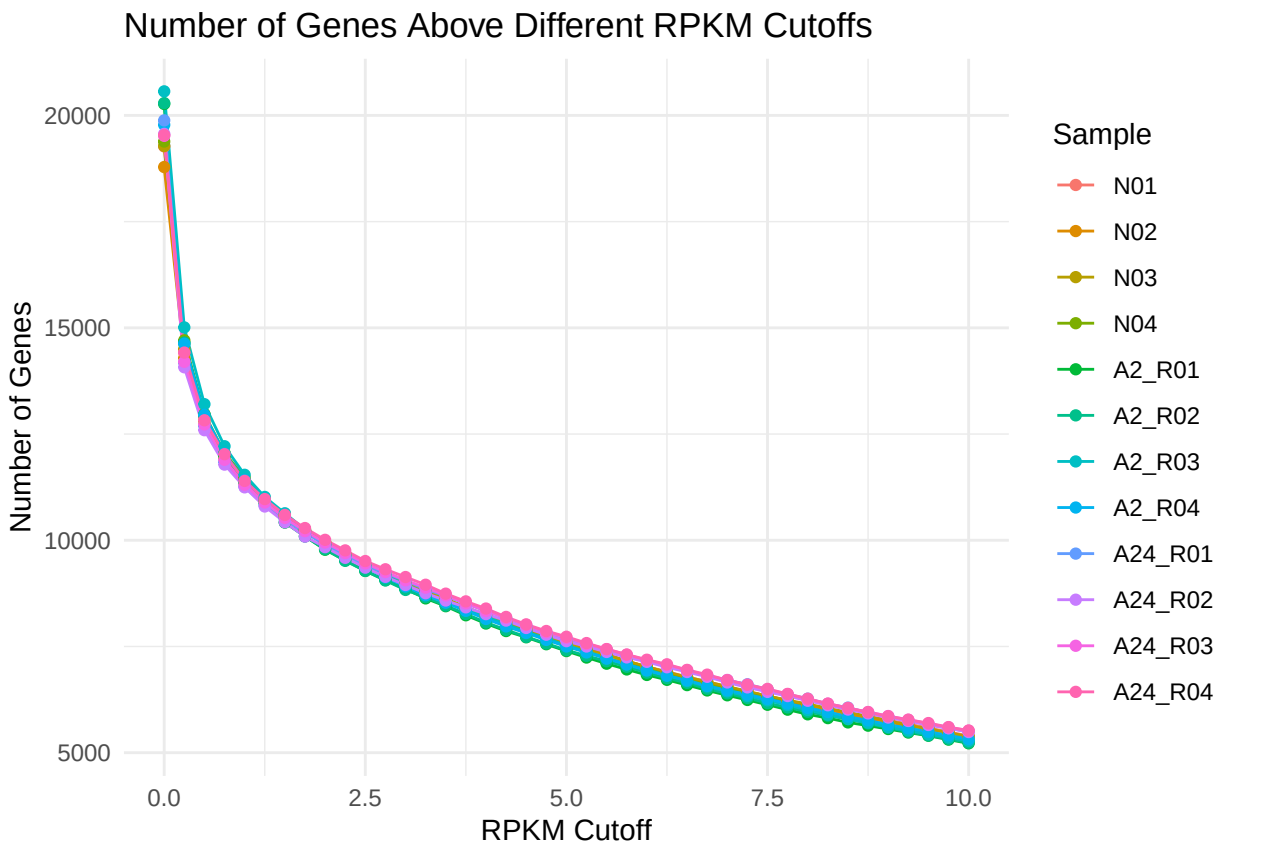

```
# compute sample to sample distance
sampleDists = dist(t(assay(vsd)))

# print(sampleDists)

library("RColorBrewer")
sampleDistMatrix <- as.matrix(sampleDists)

# sample to sampe distance plot
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
         clustering_distance_rows=sampleDists,
         clustering_distance_cols=sampleDists,
         col=colors)
```



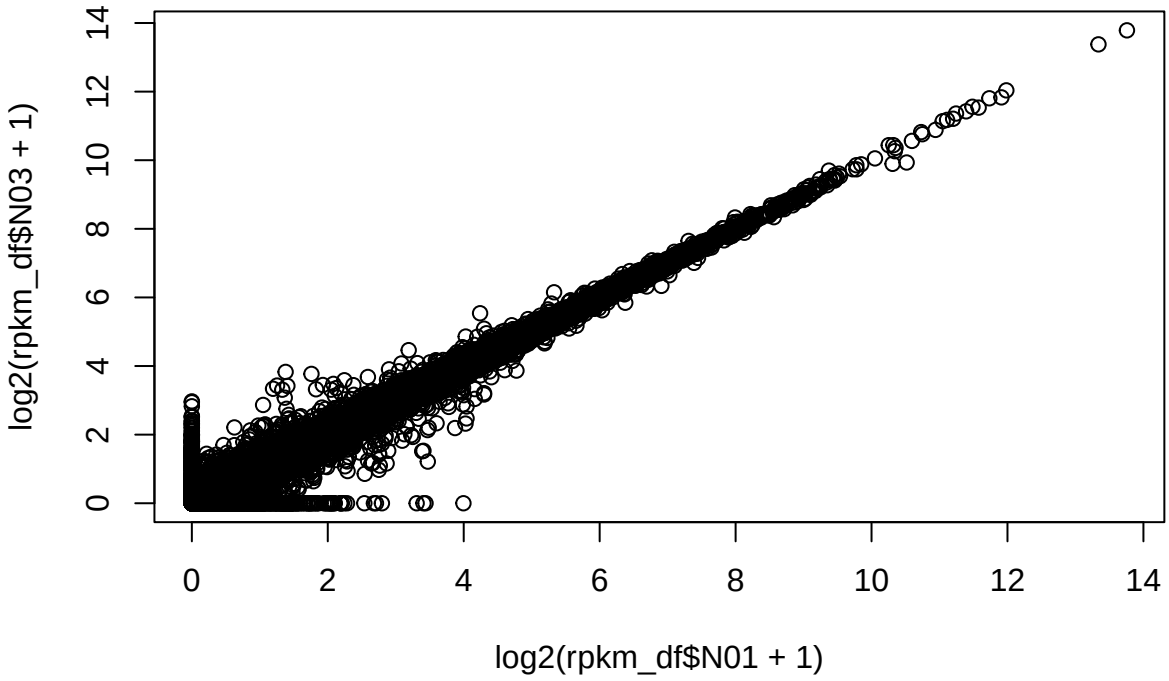
Filtering out Noise

```
rpkm_df = as.data.frame(rpkm)
cutoffs = seq(0, 10, by=0.25)
num_genes_above_cutoff = sapply(cutoffs, function(cutoff) {
  colSums(rpkm_df > cutoff)
})

# Convert the results to a data frame and transpose it
num_genes_above_cutoff_df = as.data.frame(t(num_genes_above_cutoff))

# Set the column names of the transposed data frame
colnames(num_genes_above_cutoff_df) = colnames(rpkm_df)


# Add the cutoff values as a new column
num_genes_above_cutoff_df$Cutoff = cutoffs

# Reshape the data frame to long format for plotting
df_num_melted = melt(num_genes_above_cutoff_df, id.vars = "Cutoff", variable.name = "Sample", value.nam

# Plot the number of genes above different cutoffs
ggplot(df_num_melted, aes(x = Cutoff, y = Number_of_Genes, color = Sample)) +
  geom_line() +
  geom_point() +
  theme_minimal() +
  xlab("RPKM Cutoff") +
  ylab("Number of Genes") +
```

```
  ggtitle("Number of Genes Above Different RPKM Cutoffs")
```




```
raw_counts = as.data.frame(normalized_counts)
plot(log2(rpkm_df$N01 + 1), log2(rpkm_df$N03+1),title("N1 vs N3"))
```

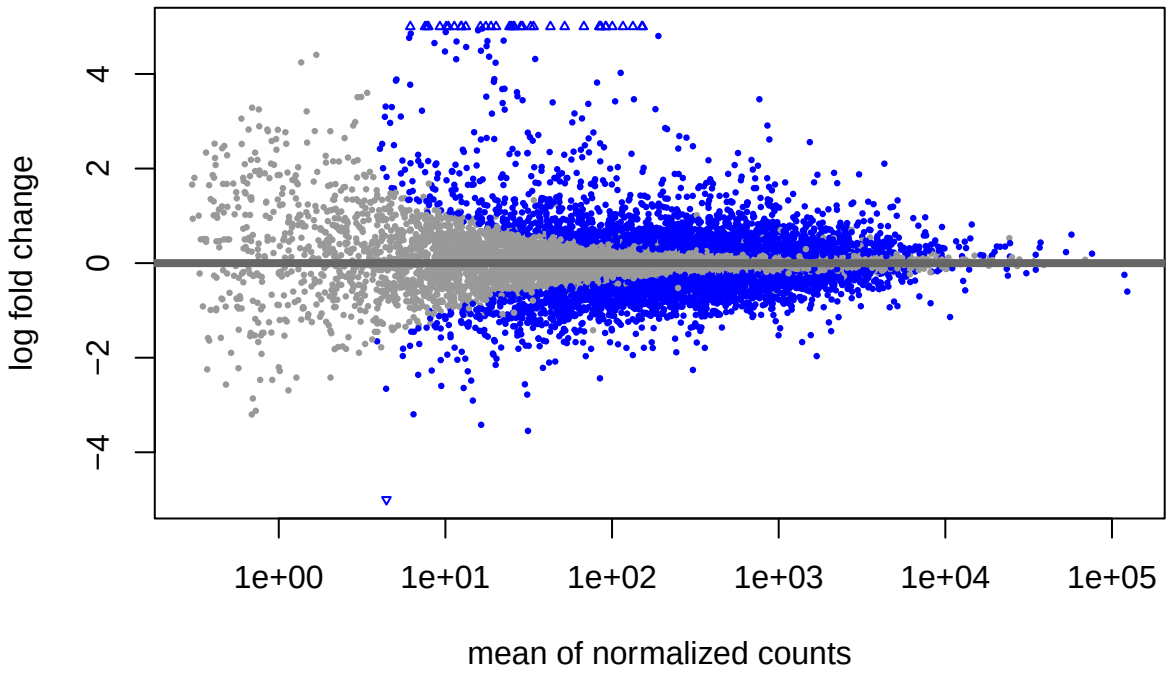



```
# set filter threshold as rpkm = 1

keep = (rowSums(rpkm) >= 6)
dds_filtered <- dds[keep,]
dds_filtered
```

```
## class: DESeqDataSet
## dim: 12908 12
## metadata(1): version
## assays(4): counts mu H cooks
## rownames(12908): TrnP TrnT ... Lypla1 Mrpl15
## rowData names(27): baseMean baseVar ... maxCooks basepairs
## colnames(12): N01 N02 ... A24_R03 A24_R04
## colData names(2): condition sizeFactor
```

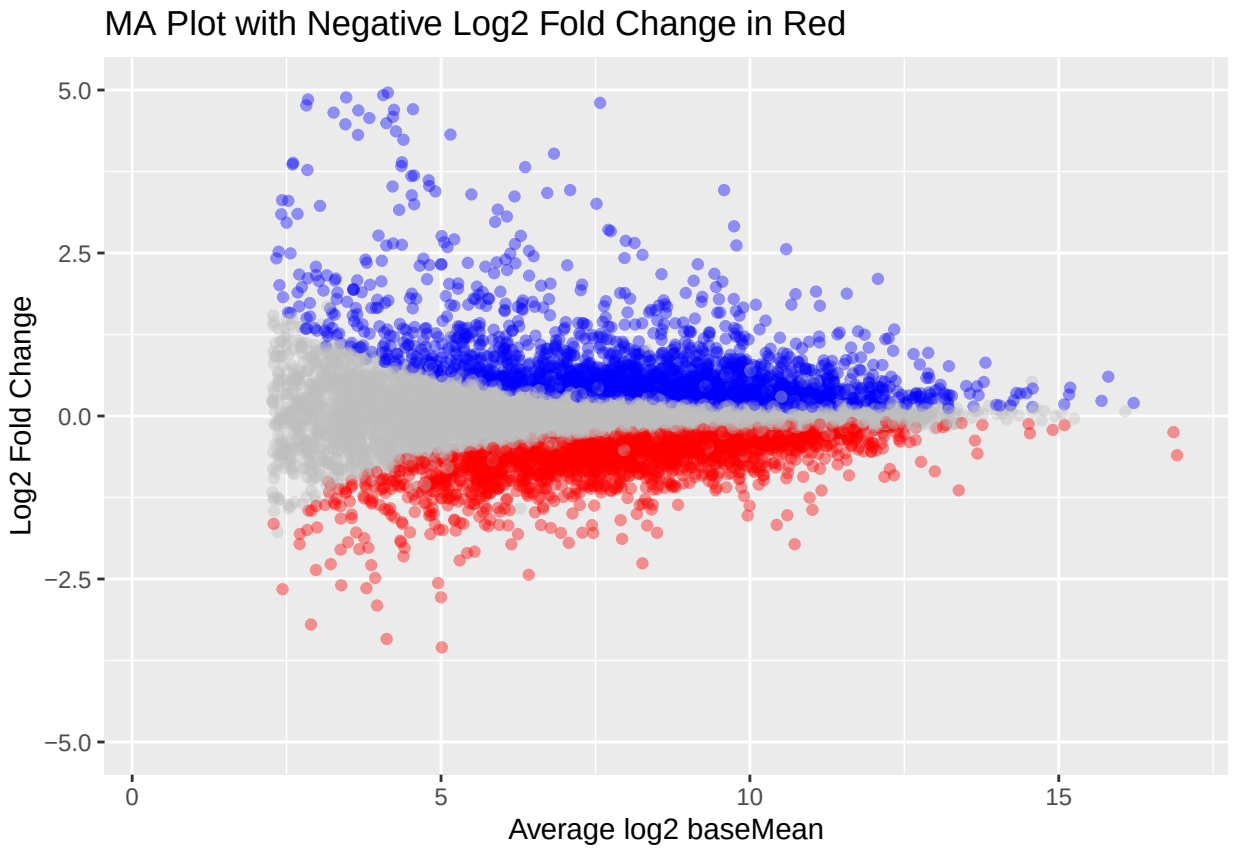
##MA plot

```
res_1 = results(dds_filtered, contrast =c ("condition","Allo2h","Naive"))

# use plotMA function
DESeq2::plotMA(res_1, ylim = c(-5, 5))
```



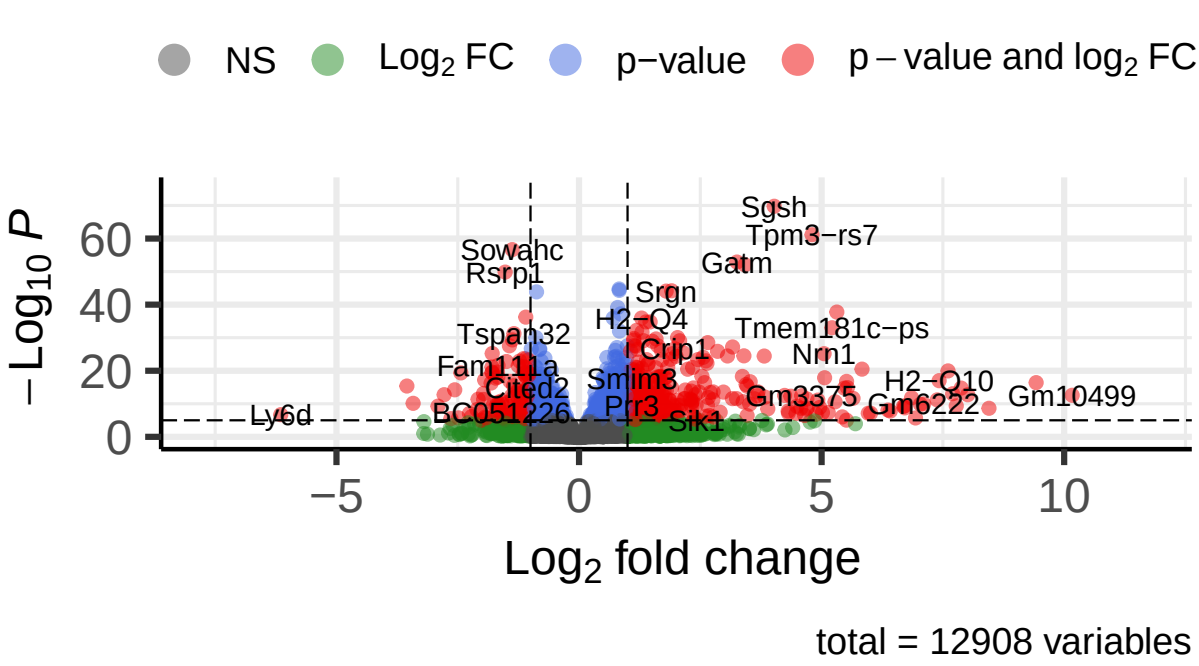

```
# use ggplot for beautiful plot
res_df <- as.data.frame(res_1)


# Add a column for color based on log2FoldChange and threshold of padj
res_df$color <- ifelse(res_df$padj < 0.1, ifelse(res_df$log2FoldChange < 0, "red", "blue"), "grey")
res_df$logBaseMean <- log2(res_df$baseMean + 1)
# Add a column for log-transformed baseMean



ggplot(res_df, aes(x =logBaseMean, y = log2FoldChange, color = color)) +
  geom_point(alpha = 0.4) +
  ylim(c(-5, 5)) +
  scale_color_identity() +
  xlab("Average log2 baseMean") +
  ylab("Log2 Fold Change") +
  ggtitle("MA Plot with Negative Log2 Fold Change in Red") +
  theme(legend.position = "none")
```

```
## Warning: Removed 538 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



Volcanol plot

```
res1 = results(dds,contrast = c("condition","Naive","Allo2h"))
p <- EnhancedVolcano(res_1,
                     lab = rownames(res_1),
                     title = "Naive vs.Allo2h",
                     x = 'log2FoldChange',
                     y = 'pvalue',
                     pointSize = 2.0,
                     labSize = 4.0)
print(p)
```



ANOVA

```
# dds_filtered
num_genes = length(dds_filtered)
num_genes
```

```
## [1] 12908
```

```
anova_table = as.data.frame(counts(dds_filtered))
anova_table$Gene = rownames(anova_table)
# head(anova_table)
Naive = as.data.frame(anova_table[,c("Gene","N01","N02","N03","N04")])
A2 = as.data.frame(anova_table[,c("Gene","A2_R01","A2_R02","A2_R03","A2_R04")])
A24 = as.data.frame(anova_table[,c("Gene","A24_R01","A24_R02","A24_R03","A24_R04")])

Naive_long = melt(Naive, varnames = c("Gene", "Sample"), value.name = "Expression")
```

```
## Using Gene as id variables
```

```
A2_long = melt(A2, varnames = c("Gene", "Sample"), value.name = "Expression")
```

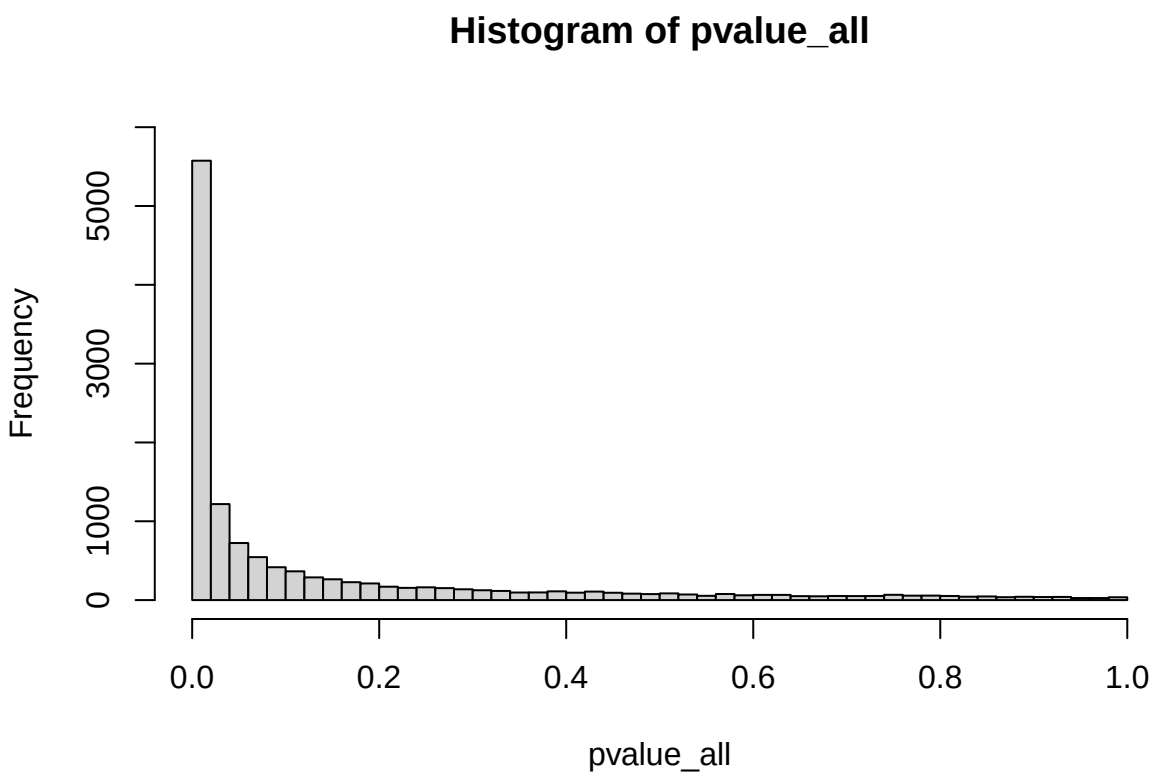
```
## Using Gene as id variables
```

```
A24_long = melt(A24, varnames = c("Gene", "Sample"), value.name = "Expression")
```

```
## Using Gene as id variables
```

```
Naive_long$Group = "Naive"

A24_long$Group = "A24"
```

```r
A2_long$Group = "A2"

all_long = bind_rows(Naive_long, A2_long,A24_long)


# Function to perform ANOVA for a single gene
perform_anova = function(df) {
  aov_result = aov(Expression ~ Group, data = df)
  p_value = summary(aov_result)[[1]]["Group", "Pr(>F)"]
  return(p_value)
}

# Initialize a data frame to store results
anova_results = data.frame(Gene = character(), p_value = numeric(), stringsAsFactors = FALSE)

# Loop through each gene and perform ANOVA

for (gene in unique(all_long$Gene)) {
  gene_data = all_long %>% filter(Gene == gene)
  p_value <- perform_anova(gene_data)
  anova_results <- rbind(anova_results, data.frame(Gene = gene, p_value = p_value))
}

# head(anova_results)

pvalue_all= anova_results$p_value
hist(pvalue_all, breaks = 50, ylim = c(0, 6000))
```



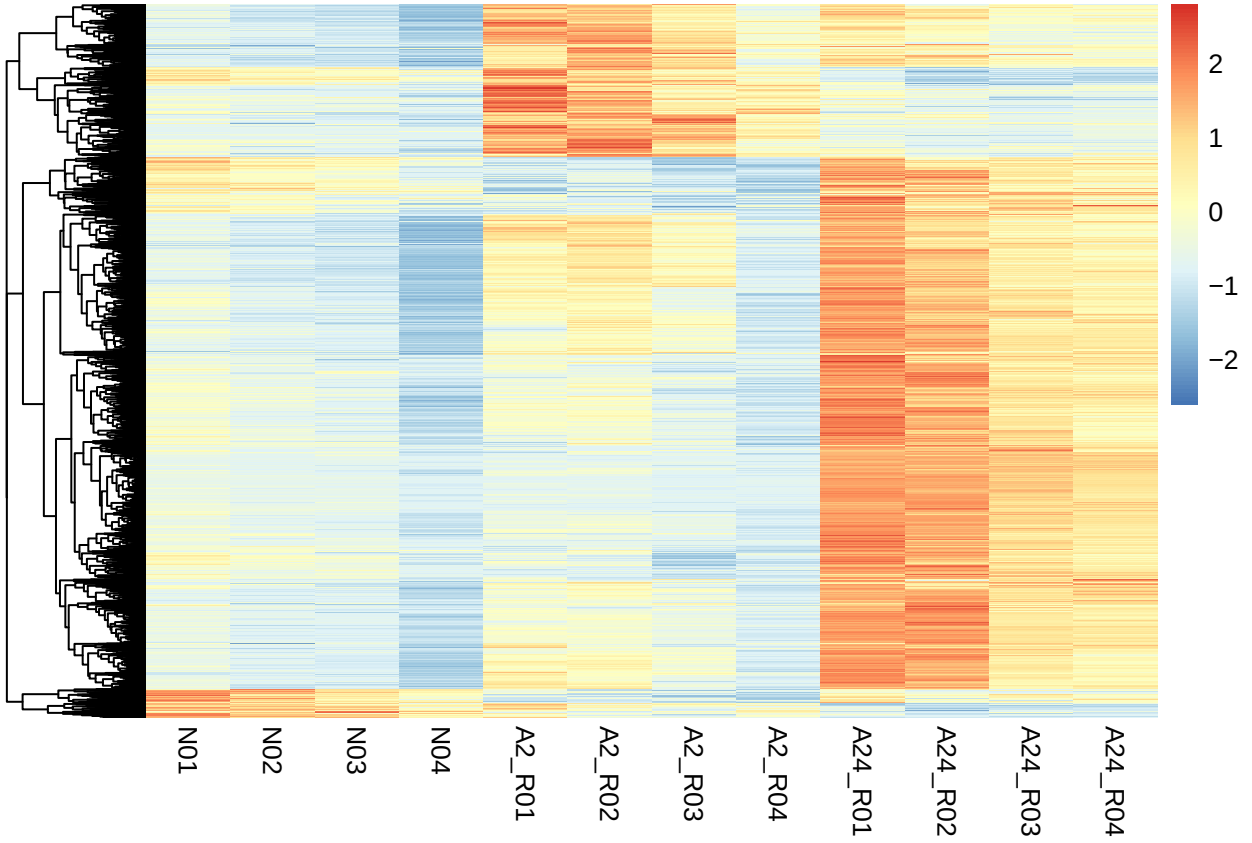
```r
anova_results <- anova_results %>%
  mutate(adj_p_value = p.adjust(p_value,method = "BH"))


# Filter significant results
significant_genes <- anova_results %>%
  filter(adj_p_value <= 0.05)

length(rownames(significant_genes))
```

```
## [1] 5770
```

Heatmap, clustering and GO analyse

```r
# Function to calculate z-scores
cal_zscore = function(df) {
  df$Expression = as.numeric(df$Expression)
  df$z_score = scale(df$Expression)
  return(df)  # Ensure the modified dataframe is returned
}

# Initialize a data frame to store results
results_all <- data.frame(Gene = character(), variable = numeric(), z_score = numeric(), stringsAsFacto

# Loop through each gene and calculate z-scores
for (gene in unique(significant_genes$Gene)) {
  gene_data <- all_long %>% filter(Gene == gene)
  gene_data <- cal_zscore(gene_data)
  # Store results
  results_all <- rbind(results_all, gene_data)
}

# results_all

# Spread the data to wide format
heatmap_data <- results_all %>%
  select(Gene, variable, z_score) %>%
  spread(key = variable, value = z_score)


# Set rownames to be the genes
rownames(heatmap_data) <- heatmap_data$Gene
heatmap_data <- heatmap_data[, -1]

# Perform k-means clustering
set.seed(123)  # For reproducibility
kmeans_result = kmeans(heatmap_data, centers = 6,
                       nstart = 2500,
                       iter.max = 2000)


kmeans_data = heatmap_data[order(kmeans_result$cluster),]
```

```
# Create a heatmap with the clustered data
pheatmap(kmeans_data,
         cluster_cols = FALSE,
         cluster_rows = TRUE,
         show_rownames = FALSE)
```




```
         # annotation_row = data.frame(Cluster = as.factor(kmeans_result$cluster)))
```

```
gene_list = dds_filtered[significant_genes$Gene,]
gene_list_df = results(gene_list)

GO_list = gene_list_df$log2FoldChange
names(GO_list) = rownames(gene_list_df)
GO_list = sort(GO_list, decreasing = TRUE)
anyDuplicated(names(GO_list))
```

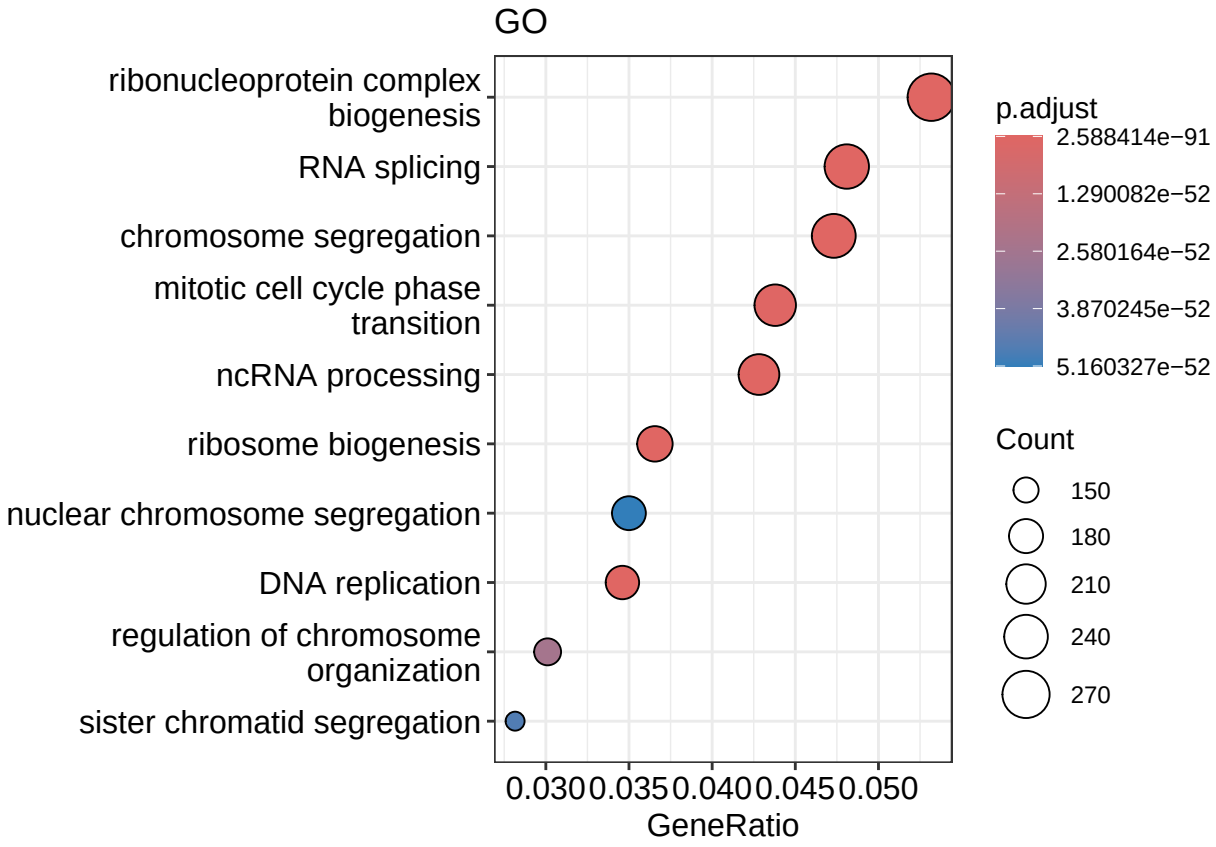
```
## [1] 0
```

```
# GO_list = gene_list[!duplicated(names(GO_list))]
# significant_genes
ego <- enrichGO(gene          = names(GO_list),
                OrgDb         = org.Mm.eg.db,
                keyType       = "SYMBOL",
                ont           = "ALL",
                pAdjustMethod = "BH",
                pvalueCutoff  = 0.01,
                qvalueCutoff  = 0.05,
```

```
                readable      = TRUE)

print(ego@result$ID[1:10],)
```

```
##  [1] "GO:0022613" "GO:0007059" "GO:0008380" "GO:0006260" "GO:0034470"
##  [6] "GO:0042254" "GO:0044772" "GO:0033044" "GO:0000819" "GO:0098813"
```

```
dotplot(ego, showCategory = 10, title = "GO")
```



GSEA

```
gsea_results <- gseGO(geneList = GO_list,
                      OrgDb = org.Mm.eg.db,
                      ont = "ALL",
                      keyType = "SYMBOL",
                      exponent = 1,
                      pvalueCutoff = 0.05,
                      verbose = TRUE,
                      seed = TRUE)
```

```
## using 'fgsea' for GSEA analysis, please cite Korotkevich et al (2019).

## preparing geneSet collections...

## GSEA analysis...

## Warning in fgseaMultilevel(pathways = pathways, stats = stats, minSize =
## minSize, : There were 58 pathways for which P-values were not calculated
## properly due to unbalanced (positive and negative) gene-level statistic values.
## For such pathways pval, padj, NES, log2err are set to NA. You can try to
## increase the value of the argument nPermSimple (for example set it nPermSimple
```

```
## = 10000)
```

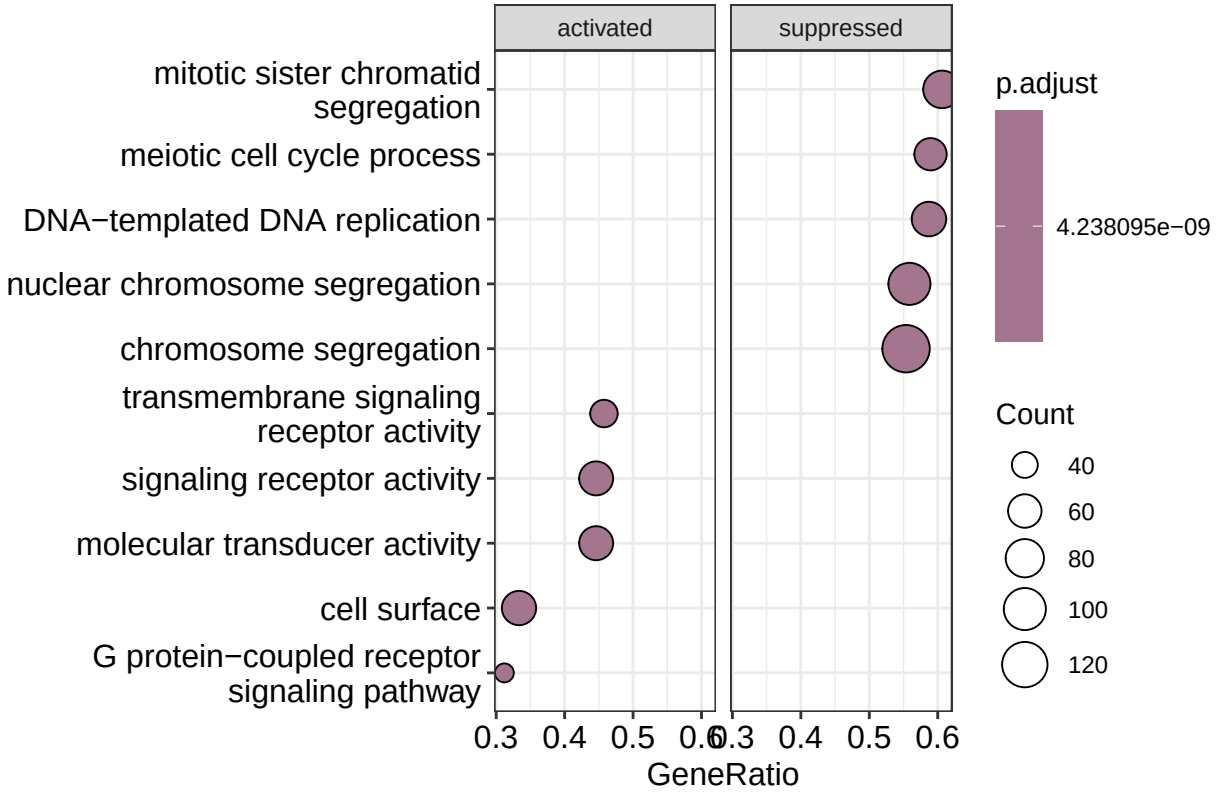
```
## Warning in fgseaMultilevel(pathways = pathways, stats = stats, minSize =
## minSize, : For some of the pathways the P-values were likely overestimated. For
## such pathways log2err is set to NA.
```

```
## Warning in fgseaMultilevel(pathways = pathways, stats = stats, minSize =
## minSize, : For some pathways, in reality P-values are less than 1e-10. You can
## set the `eps` argument to zero for better estimation.
```

```
## leading edge analysis...
```

```
## done...
```

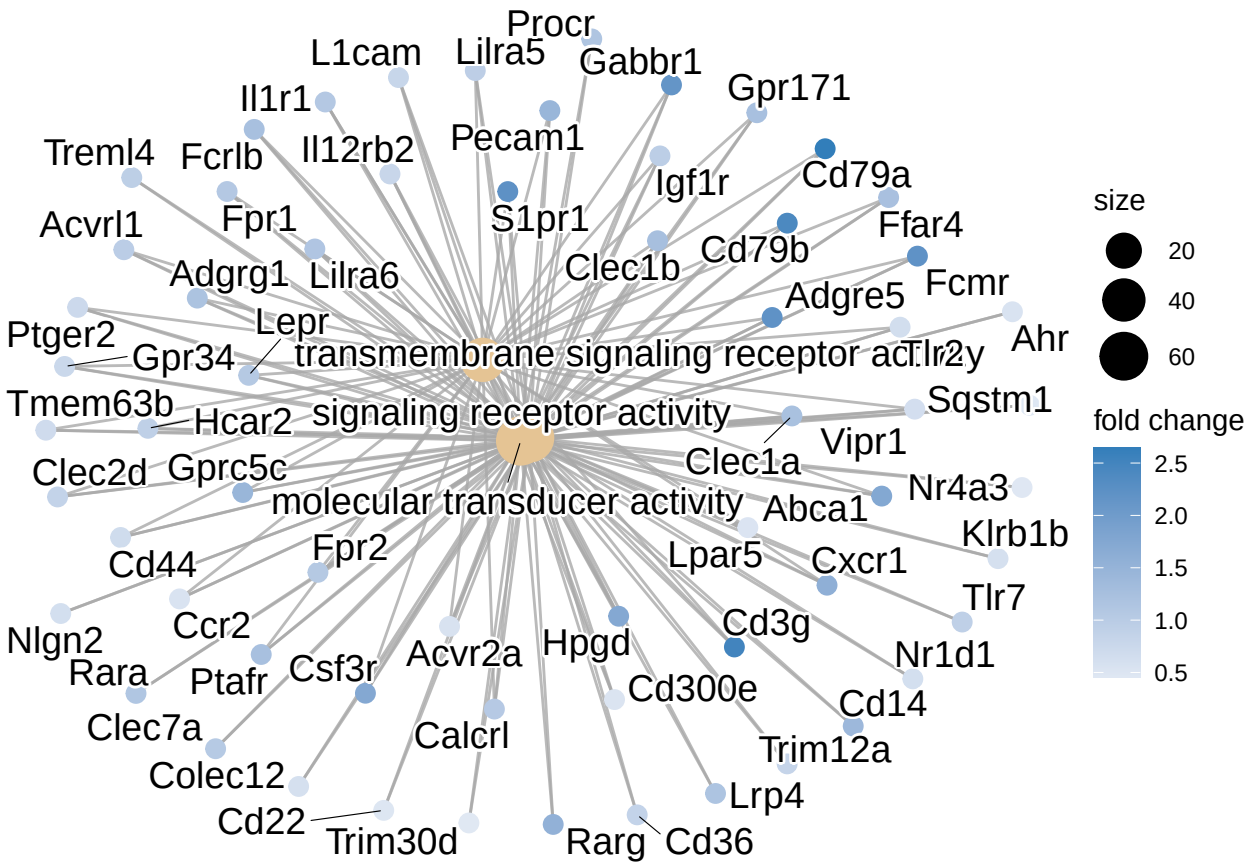
```
dotplot(gsea_results, showCategory=5, split=".sign") + facet_grid(.~.sign)
```




```
cnetplot(gsea_results, categorySize="pvalue", foldChange=GO_list, showCategory = 3)
```

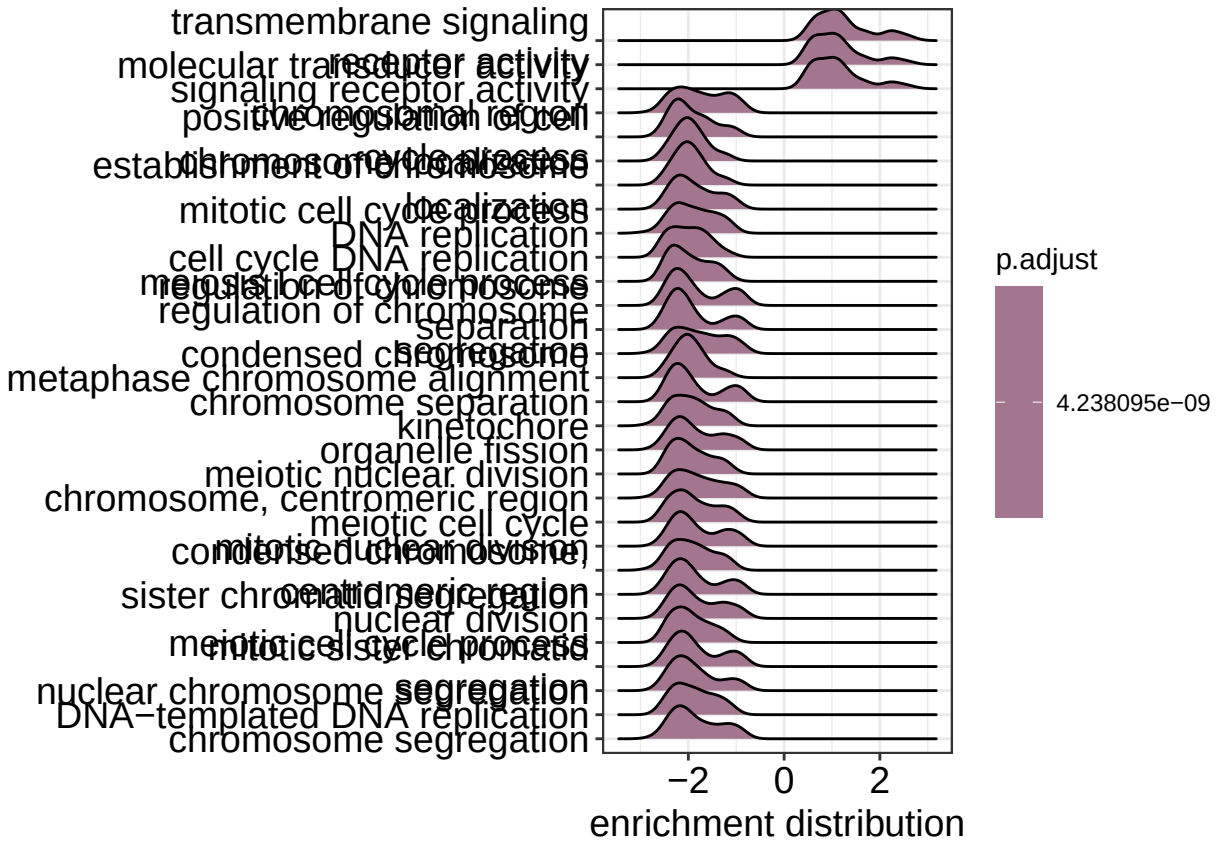
```
## Warning in cnetplot.enrichResult(x, ...): Use 'color.params = list(foldChange = your_value)' instead
##  The foldChange parameter will be removed in the next version.
```

```
## Scale for size is already present.
## Adding another scale for size, which will replace the existing scale.
```

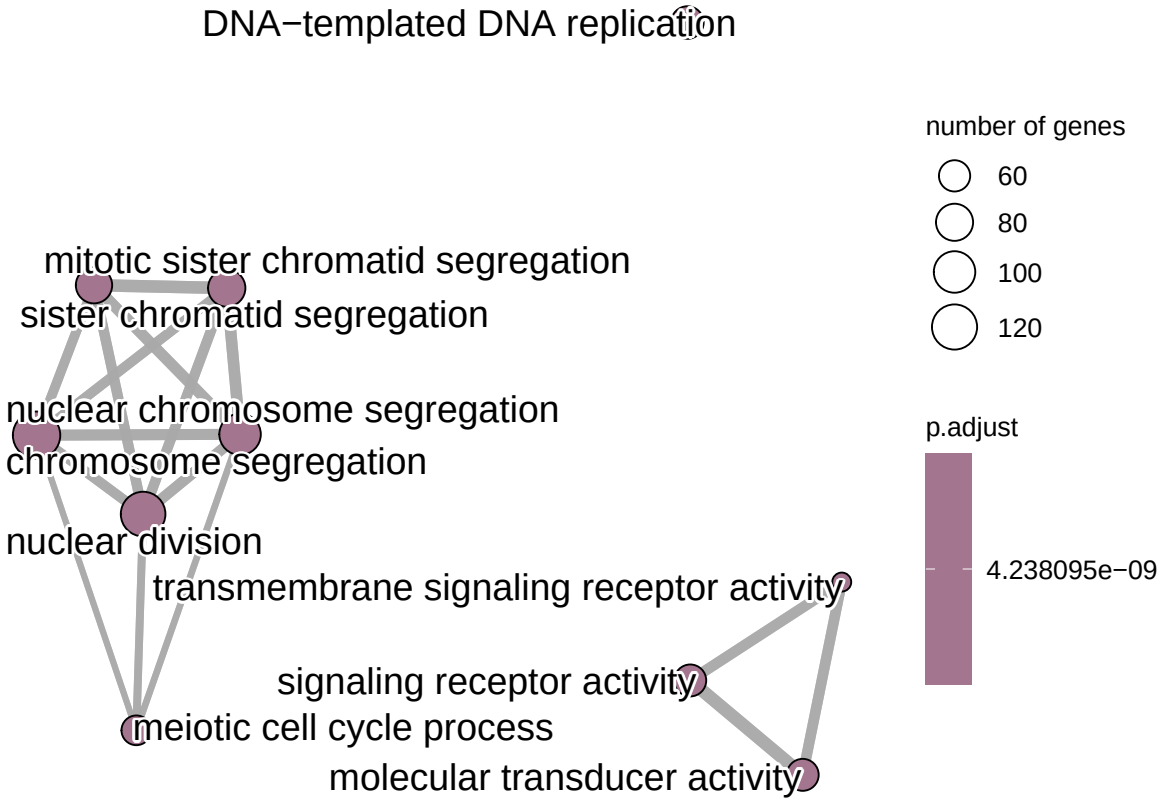



```
ridgeplot(gsea_results) + labs(x = "enrichment distribution")
```

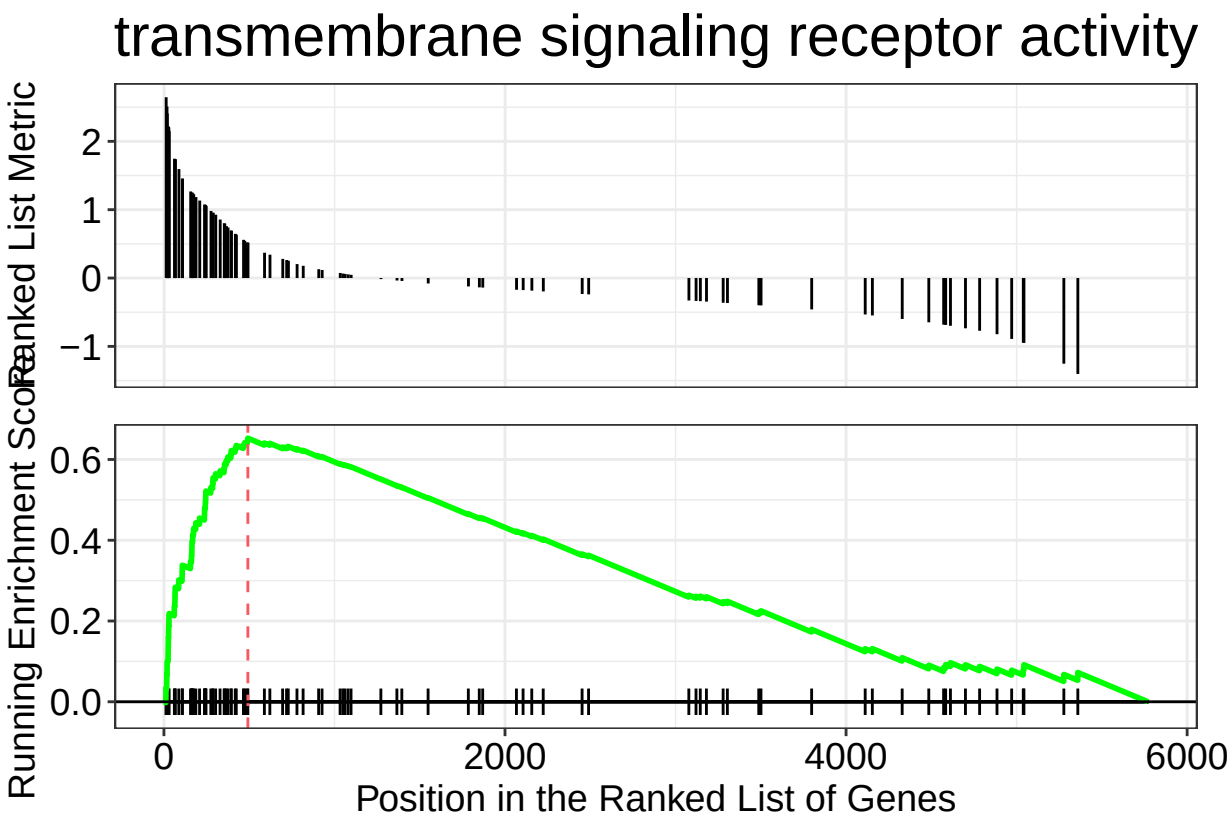
```
## Picking joint bandwidth of 0.177
```




```
gse <- pairwise_termsim(gsea_results)
emapplot(gse, showCategory = 10)
```




```
gseaplot(gsea_results, by = "all", title = gse$Description[1], geneSetID = 1)
```



GO based on cluster

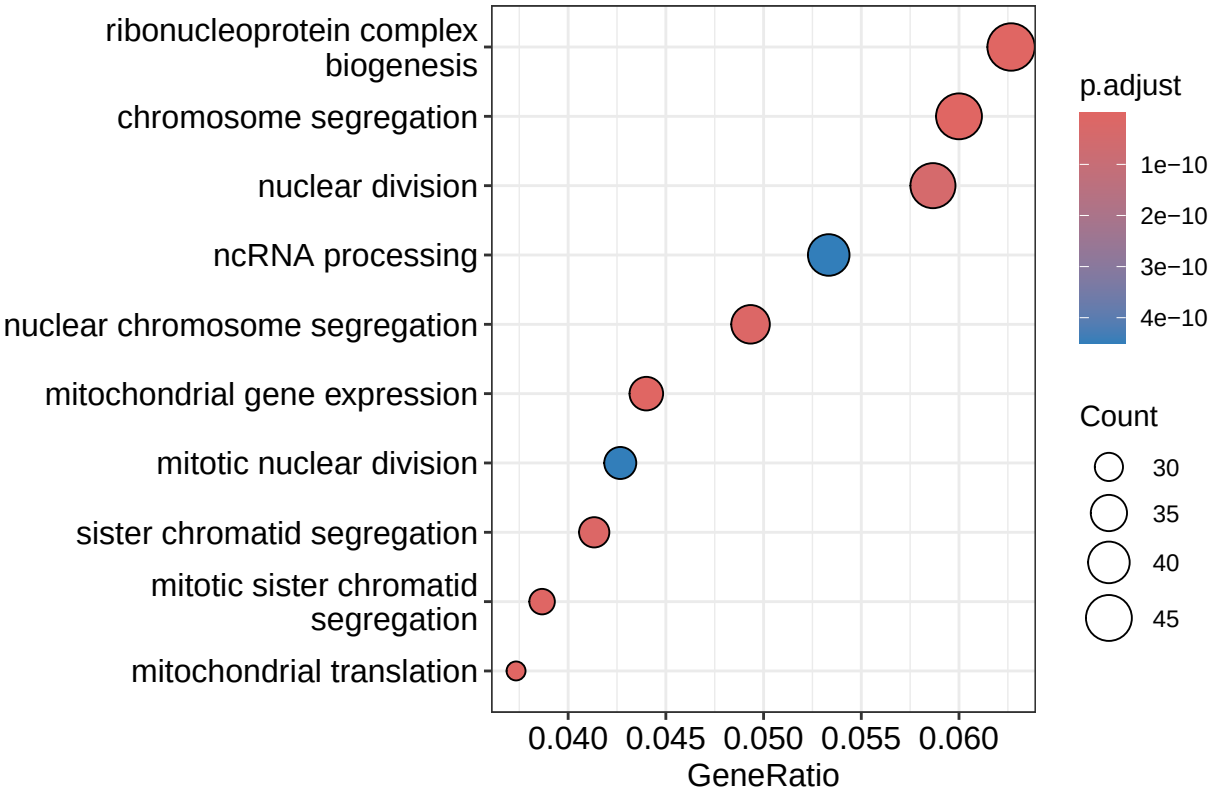
```
# Extract gene lists for each cluster
gene_clusters <- split(rownames(kmeans_data), kmeans_result$cluster)
enrichment_results <- list()
# Perform functional enrichment analysis on each gene list
for (i in 1:6) {
    cluster_genes <- gene_clusters[[i]]
    cluster_genes_entrez <- bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", OrgDb = org.M
    cluster_genes_entrez <- cluster_genes_entrez$ENTREZID

    ego <- enrichGO(gene         = cluster_genes_entrez,
                    OrgDb        = org.Mm.eg.db,
                    keyType      = "ENTREZID",
                    ont          = "BP",
                    pAdjustMethod = "BH",
                    pvalueCutoff  = 0.05,
                    qvalueCutoff  = 0.2)

    enrichment_results[[i]] <- ego
    p <- dotplot(ego, showCategory = 10)
    print(p)
}
```

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

```
## Warning in bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", :
## 2.03% of input gene IDs are fail to map...
```

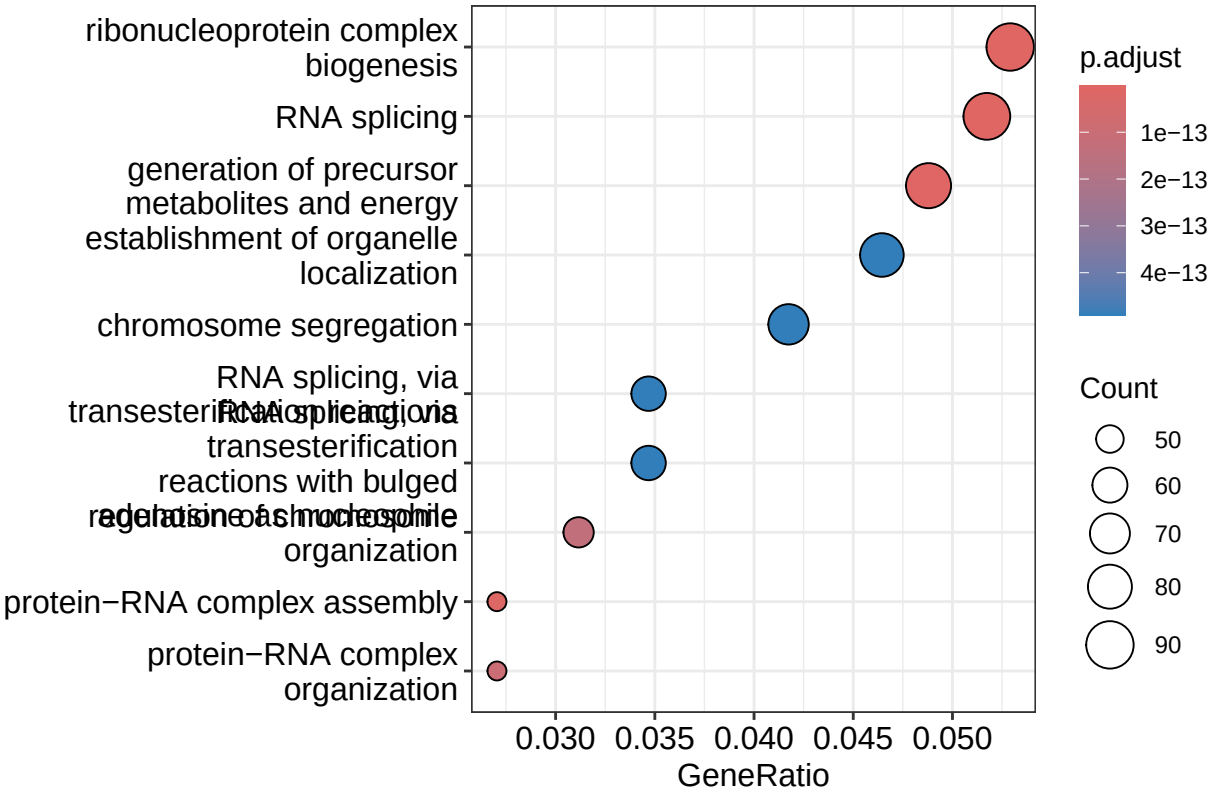
```
## 'select()' returned 1:1 mapping between keys and columns
```

```
## Warning in bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", :
## 2.57% of input gene IDs are fail to map...
```



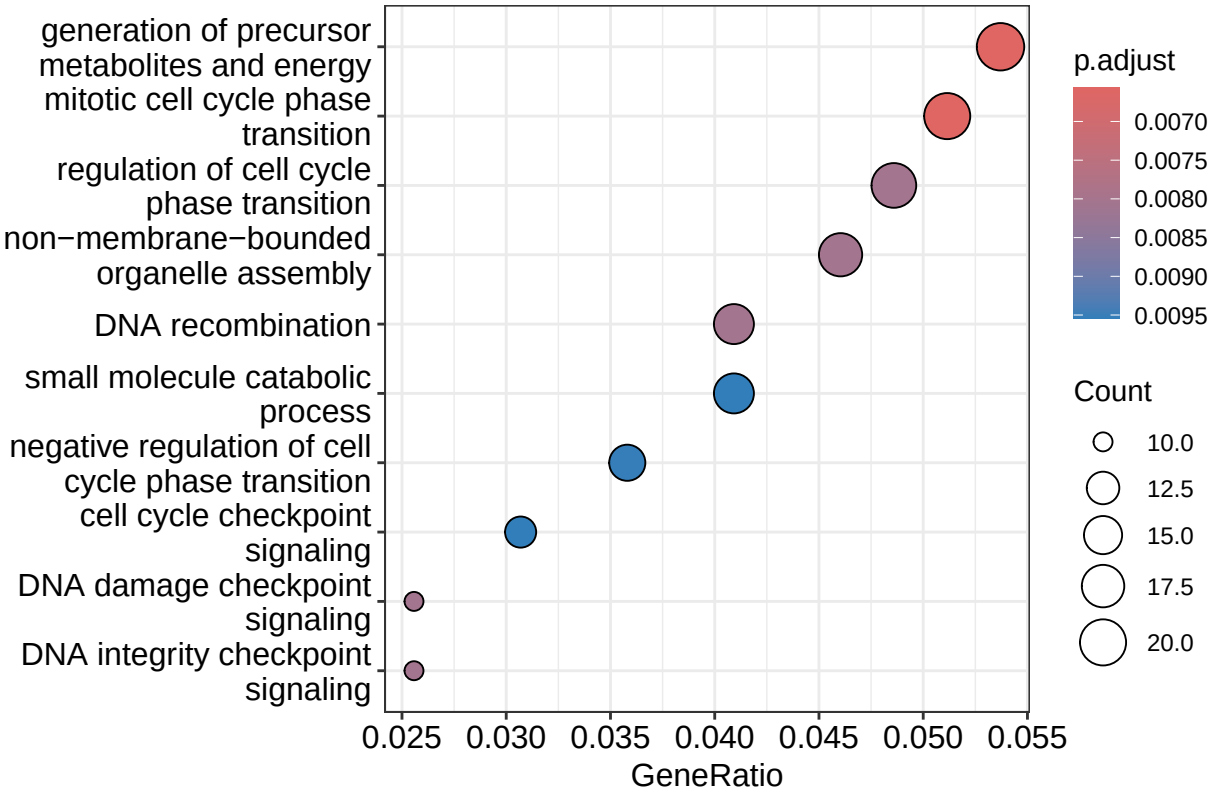

```
## 'select()' returned 1:1 mapping between keys and columns
```

```
## Warning in bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", :
## 2.07% of input gene IDs are fail to map...
```



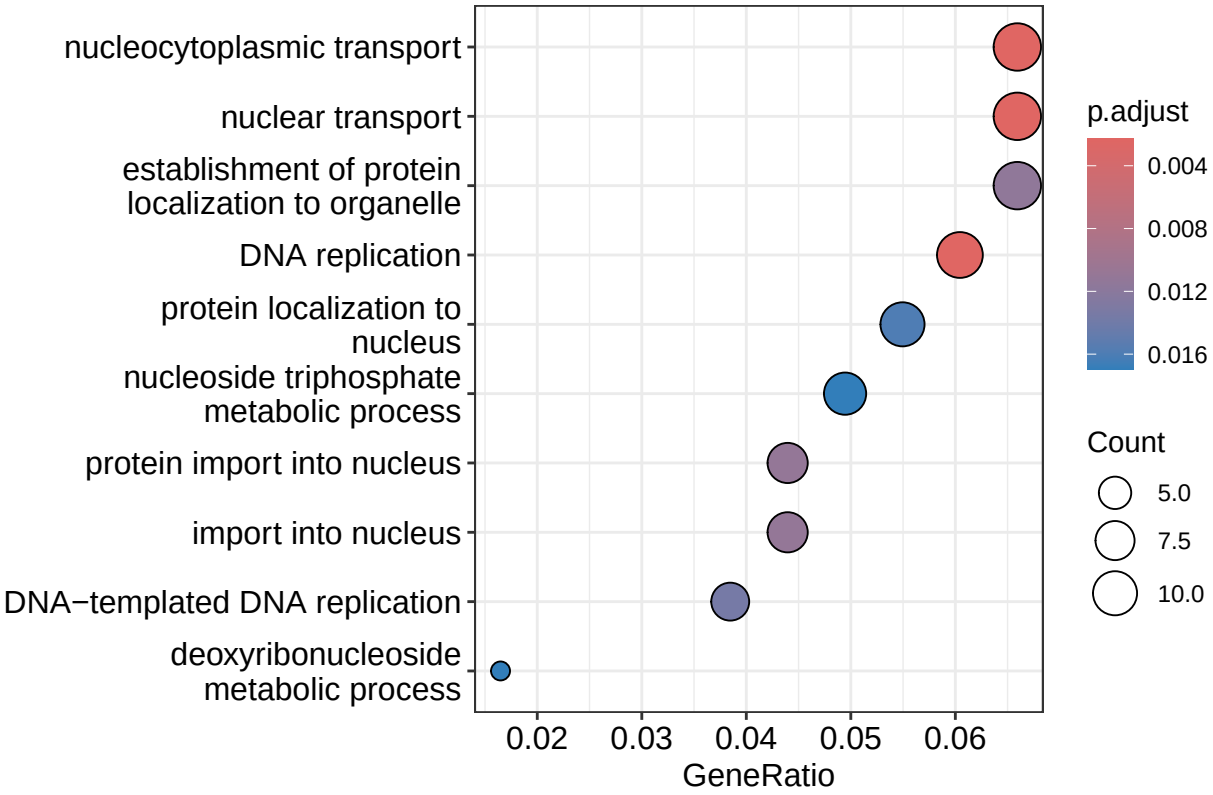
```
## 'select()' returned 1:1 mapping between keys and columns
```

```
## Warning in bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", :
## 0.99% of input gene IDs are fail to map...
```



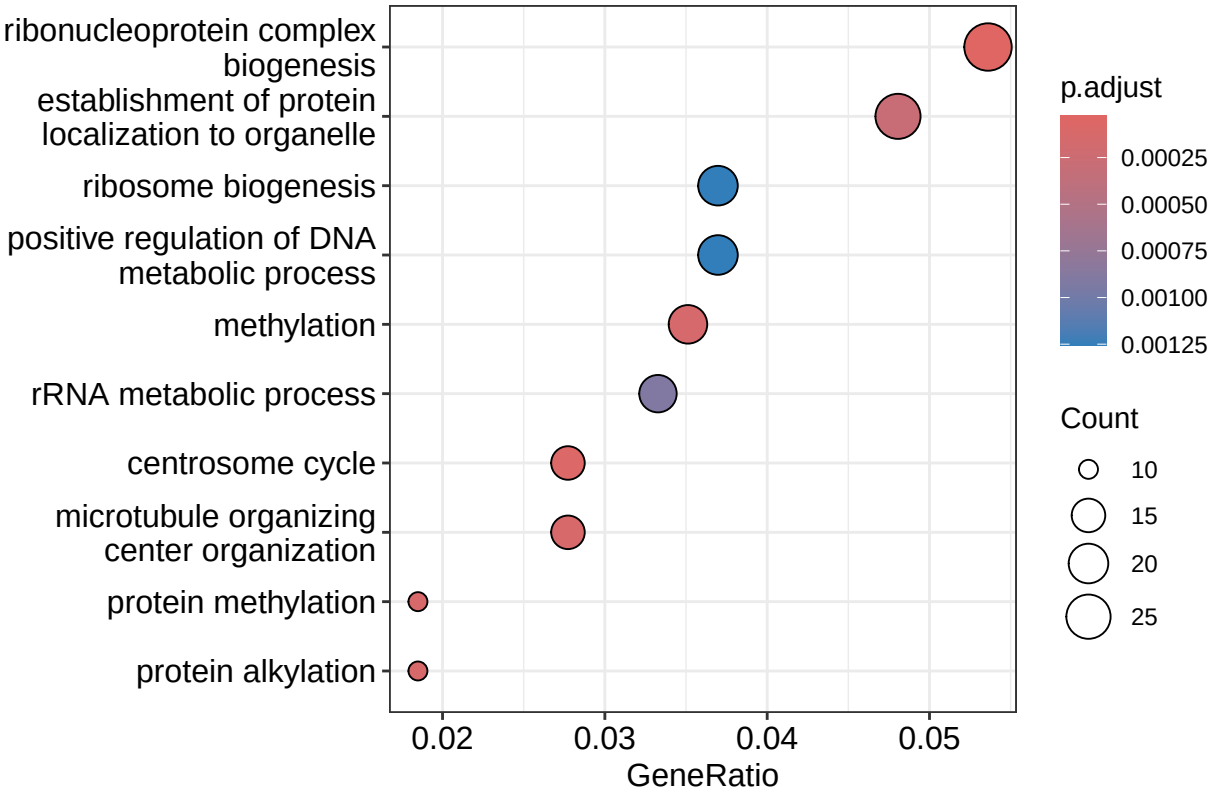

```
## 'select()' returned 1:1 mapping between keys and columns
```

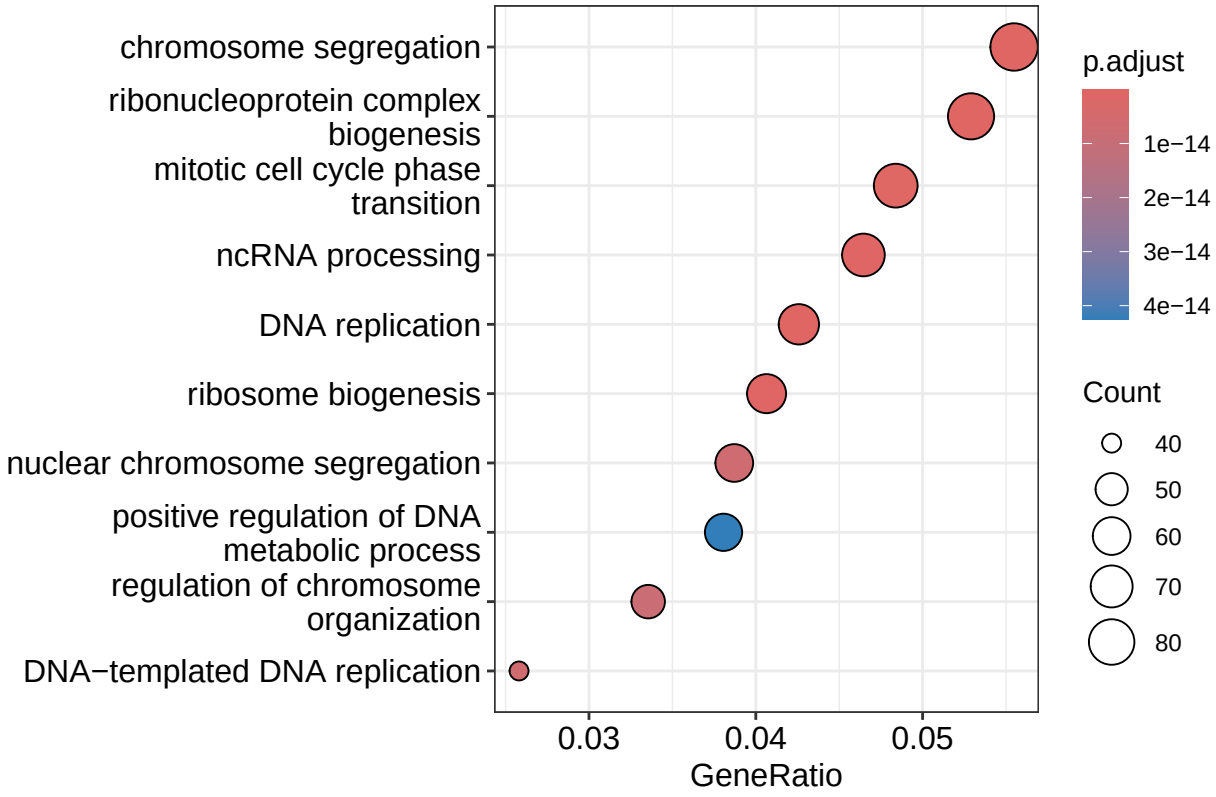
```
## Warning in bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", :
## 2.92% of input gene IDs are fail to map...
```




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

```
## Warning in bitr(cluster_genes, fromType = "SYMBOL", toType = "ENTREZID", :
## 2.87% of input gene IDs are fail to map...
```



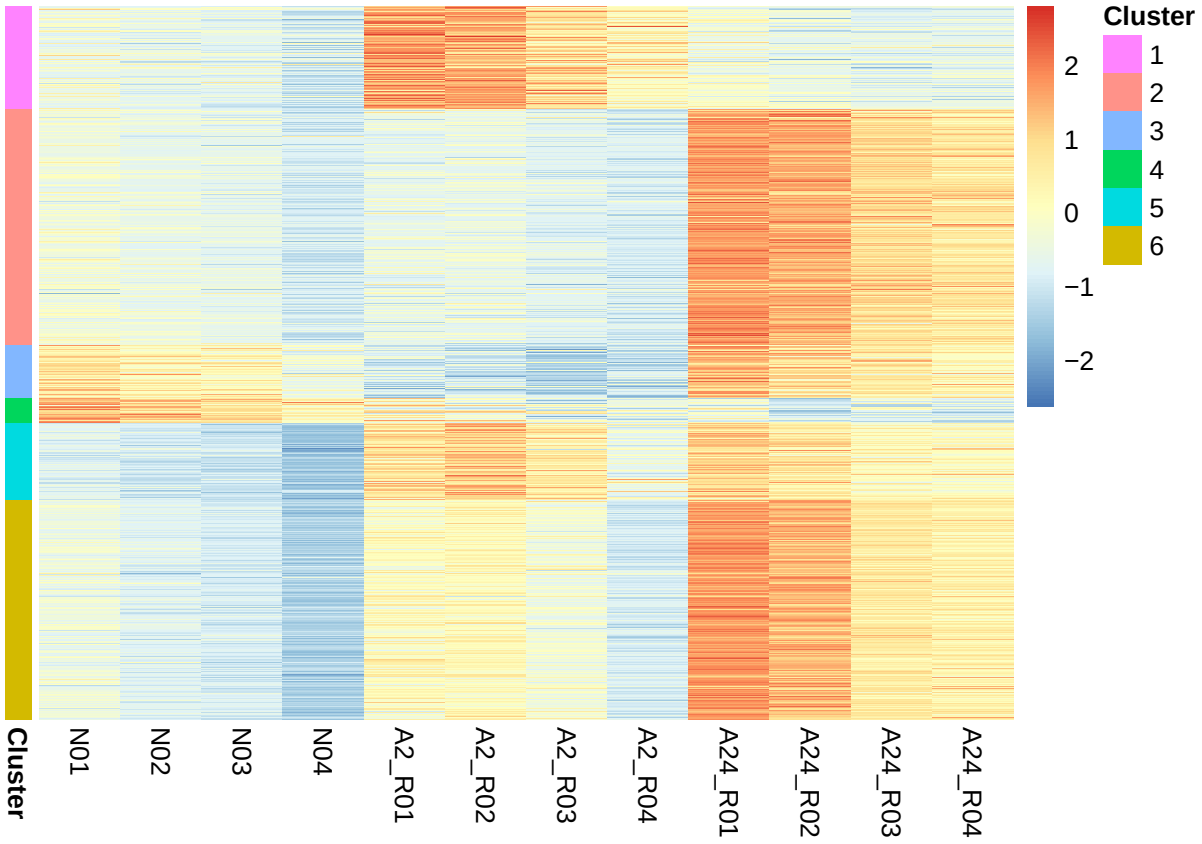



```
enrichment_results[[1]]
```

```
## #
## # over-representation test
## #
## #...@organism     Mus musculus
## #...@ontology     BP
## #...@keytype      ENTREZID
## #...@gene     chr [1:820] "73558" "320469" "11303" "18671" "66190" "23797" "11658" ...
## #...pvalues adjusted by 'BH' with cutoff <0.05
## #...428 enriched terms found
## 'data.frame':    428 obs. of  9 variables:
##  $ ID         : chr  "GO:0140053" "GO:0032543" "GO:0022613" "GO:0007059" ...
##  $ Description: chr  "mitochondrial gene expression" "mitochondrial translation" "ribonucleoprotein c
##  $ GeneRatio  : chr  "33/750" "28/750" "47/750" "45/750" ...
##  $ BgRatio    : chr  "167/28905" "130/28905" "442/28905" "419/28905" ...
##  $ pvalue     : num  7.60e-20 4.36e-18 3.16e-16 9.49e-16 4.57e-15 ...
##  $ p.adjust   : num  3.46e-16 9.93e-15 4.80e-13 1.08e-12 4.16e-12 ...
##  $ qvalue     : num  2.83e-16 8.12e-15 3.93e-13 8.84e-13 3.40e-12 ...
##  $ geneID     : chr  "211064/66971/229487/228019/68537/68463/68611/94065/67270/69163/18120/66493/662
##  $ Count      : int  33 28 47 45 29 31 37 44 40 32 ...
## #...Citation
##  T Wu, E Hu, S Xu, M Chen, P Guo, Z Dai, T Feng, L Zhou, W Tang, L Zhan, X Fu, S Liu, X Bo, and G Yu
##  clusterProfiler 4.0: A universal enrichment tool for interpreting omics data.
##  The Innovation. 2021, 2(3):100141
```

```
top_terms <- sapply(enrichment_results, function(x) {
  if (nrow(x) > 0) {
    return(x@result$Description[1])
```

```
  } else {
    return("No significant term")
  }
})


# Create a data frame for annotation
annotation_df <- data.frame(Cluster = factor(1:6), GO_Process = top_terms)
pheatmap(kmeans_data,
         cluster_cols = FALSE,
         cluster_rows = FALSE,
         show_rownames = FALSE,
         annotation_row = data.frame(Cluster = as.factor(kmeans_result$cluster)))
```




```
annotation_df
```

```
##   Cluster                          GO_Process
## 1       1        mitochondrial gene expression
## 2       2 ribonucleoprotein complex biogenesis
## 3       3  mitotic cell cycle phase transition
## 4       4                      DNA replication
## 5       5 ribonucleoprotein complex biogenesis
## 6       6               chromosome segregation
```

Visualization of Individual Genes

```
# Plot CDK2 RPKM histogram
cdk2_rpkm = rpkm["Cdk2",]
cdk2_df = data.frame(Group = conditions, RPKM = cdk2_rpkm)
summary_cdk2_df = cdk2_df %>%
  group_by(Group) %>%
  summarise(
    mean_RPKM = mean(RPKM),
    sd_RPKM = sd(RPKM)
  )

ggplot(summary_cdk2_df, aes(x = Group, y = mean_RPKM, fill = Group)) +
  geom_bar(stat = "identity", position = position_dodge(), width = 0.7) +
  geom_errorbar(aes(ymin = mean_RPKM - sd_RPKM, ymax = mean_RPKM + sd_RPKM),
                width = 0.2, position = position_dodge(0.7)) +
  theme_minimal() +
  labs(title = "RPKM Values by Group with SD Error Bars", x = "Group", y = "Mean RPKM") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```