Glioblastoma-Stage2

2024-09-14

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Load the data

Load the Glioblastoma data Directly from this URL : https://raw.githubusercontent.com/HackBio-Internship/public_datasets/main/Cancer2024/glioblastoma.csv

```
# Load and manipulate the data
gene_data <- read_csv("/home/admin/gene_data.csv")
```

```
## Rows: 457 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): symbols
## dbl (10): TCGA-19-4065-02A-11R-2005-01, TCGA-19-0957-02A-11R-2005-01, TCGA-0...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
gene_data <- as.data.frame(gene_data)
rownames(gene_data) <- gene_data$symbols
gene_data <- gene_data[,-11]
```

Create sample info dataframe for DESEq2 running step

```
sample_info <- data.frame(
  SampleID = colnames(gene_data),
  tissue_type = sapply(colnames(gene_data), function(barcode) {
    sample_type <- substr(barcode, 14, 15)
    switch(sample_type,
           "01" = "Solid Tissue Normal",
           "02" = "Primary Tumor",
           "06" = "Metastatic",
           "11" = "Solid Tissue Normal",
```

```
            "12" = "Primary Blood Derived Cancer",
            "14" = "Primary Blood Derived Cancer",
            "Unknown")
  })
)
```

The first 6 rows from sample_info data frame

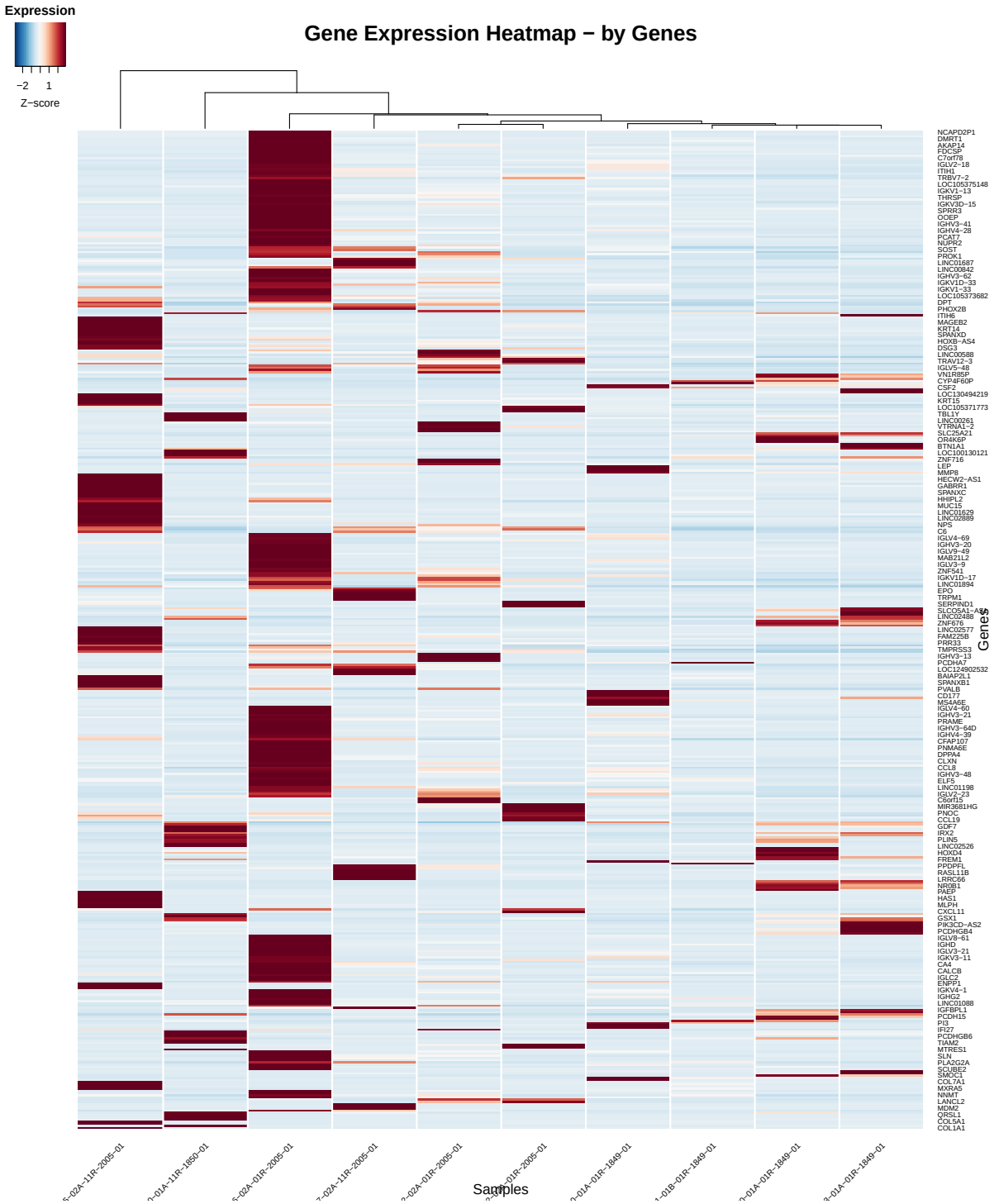
```
print(head(sample_info))
```

```
##                                                  SampleID         tissue_type
## TCGA-19-4065-02A-11R-2005-01 TCGA-19-4065-02A-11R-2005-01       Primary Tumor
## TCGA-19-0957-02A-11R-2005-01 TCGA-19-0957-02A-11R-2005-01       Primary Tumor
## TCGA-06-0152-02A-01R-2005-01 TCGA-06-0152-02A-01R-2005-01       Primary Tumor
## TCGA-14-1402-02A-01R-2005-01 TCGA-14-1402-02A-01R-2005-01       Primary Tumor
## TCGA-14-0736-02A-01R-2005-01 TCGA-14-0736-02A-01R-2005-01       Primary Tumor
## TCGA-06-5410-01A-01R-1849-01 TCGA-06-5410-01A-01R-1849-01 Solid Tissue Normal
```

This code creates a gene expression heatmap using the heatmap.2 function, visualizing all samples across genes. It applies col-wise scaling, uses a custom color palette, and includes various customizations for improved readability, such as rotated column labels, adjusted margins, and a color key representing Z-scores.

```
heatmap.2(as.matrix(gene_data), # or use gene_data_subset if subsetting
          scale = "row",
          col = colorRampPalette(c("#053061", "#2166AC", "#4393C3", "#92C5DE", "#D1E5F0",
                                   "#F7F7F7", "#FDDBC7", "#F4A582", "#D6604D", "#B2182B", "#67001F"))(1
          trace = "none",
          dendrogram = "col",
          main = "Gene Expression Heatmap - by Genes",
          xlab = "Samples",
          ylab = "Genes",
          margins = c(5, 5),
          cexRow = 0.6,
          cexCol = 0.7,
          srtCol = 45,
          adjCol = c(1,1),
          keysize = 1,
          key.title = "Expression",
          key.xlab = "Z-score",
          density.info = "none",
          colsep = 1:ncol(gene_data),
          sepcolor = "white",
          sepwidth = c(0.01, 0.01),
```

```
        lhei = c(1, 8),
        lwid = c(1, 8))
```



This code creates a gene expression heatmap using the heatmap.2 function, visualizing all genes across samples. It applies row-wise scaling, uses a custom color palette, and includes various customizations for improved readability, such as rotated column labels, adjusted margins, and a color key representing Z-scores.

```r
# Optional: Subset data if needed
# top_genes <- head(order(rowVars(as.matrix(gene_data)), decreasing=TRUE), 1000)
# gene_data_subset <- gene_data[top_genes,]

heatmap.2(as.matrix(gene_data), # or use gene_data_subset if subsetting
          scale = "row",
          col = colorRampPalette(c("#053061", "#2166AC", "#4393C3", "#92C5DE", "#D1E5F0",
                                   "#F7F7F7", "#FDDBC7", "#F4A582", "#D6604D", "#B2182B", "#67001F"))(1
          trace = "none",
          dendrogram = "row",
          main = "Gene Expression Heatmap - by Genes",
          xlab = "Samples",
          ylab = "Genes",
          margins = c(5, 5),
          cexRow = 0.6,
          cexCol = 0.7,
          srtCol = 45,
          adjCol = c(1,1),
          keysize = 1,
          key.title = "Expression",
          key.xlab = "Z-score",
          density.info = "none",
          colsep = 1:ncol(gene_data),
          sepcolor = "white",
          sepwidth = c(0.01, 0.01),
          lhei = c(1, 8),
          lwid = c(1, 8))
```

This code creates a gene expression heatmap using the heatmap.2 function, visualizing all genes across samples. It applies both-wise scaling, uses a custom color palette, and includes various customizations for improved readability, such as rotated column labels, adjusted margins, and a color key representing Z-scores.

```
# Optional: Subset data if needed
# top_genes <- head(order(rowVars(as.matrix(gene_data)), decreasing=TRUE), 1000)
# gene_data_subset <- gene_data[top_genes,]
```

```r
heatmap.2(as.matrix(gene_data), # or use gene_data_subset if subsetting
          scale = "row",
          col = colorRampPalette(c("#053061", "#2166AC", "#4393C3", "#92C5DE", "#D1E5F0",
                                   "#F7F7F7", "#FDDBC7", "#F4A582", "#D6604D", "#B2182B", "#67001F"))(1
          trace = "none",
          dendrogram = "both",
          main = "Gene Expression Heatmap - Both",
          xlab = "Samples",
          ylab = "Genes",
          margins = c(5, 5),
          cexRow = 0.6,
          cexCol = 0.7,
          srtCol = 45,
          adjCol = c(1,1),
          keysize = 1,
          key.title = "Expression",
          key.xlab = "Z-score",
          density.info = "none",
          colsep = 1:ncol(gene_data),
          sepcolor = "white",
          sepwidth = c(0.01, 0.01),
          lhei = c(1, 8),
          lwid = c(1, 8))
```

This code performs differential expression analysis using DESeq2, including data preparation, pre-filtering, and running the DESeq algorithm. The results are then filtered to identify statistically significant (padj < 0.05) and biologically relevant ($|\log2\text{FoldChange}| > 2$) differentially expressed genes, prioritizing those with the largest expression changes.

```
## DESeq2 Analysis

# Create DESeqDataSet object
```

```
dds <- DESeqDataSetFromMatrix(countData = round(gene_data),
                              colData = sample_info,
                              design = ~ tissue_type)
```

```
## converting counts to integer mode

## Warning in DESeqDataSet(se, design = design, ignoreRank): some variables in
## design formula are characters, converting to factors

##   Note: levels of factors in the design contain characters other than
##   letters, numbers, '_' and '.'. It is recommended (but not required) to use
##   only letters, numbers, and delimiters '_' or '.', as these are safe characters
##   for column names in R. [This is a message, not a warning or an error]
```

```
# Pre-filtering
dds <- dds[rowSums(counts(dds)) >= 10, ]

# Set factor levels
dds$tissue_type <- relevel(dds$tissue_type, ref = "Solid Tissue Normal")

# Run DESeq
dds <- DESeq(dds)
```

```
## estimating size factors

##   Note: levels of factors in the design contain characters other than
##   letters, numbers, '_' and '.'. It is recommended (but not required) to use
##   only letters, numbers, and delimiters '_' or '.', as these are safe characters
##   for column names in R. [This is a message, not a warning or an error]

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

##   Note: levels of factors in the design contain characters other than
##   letters, numbers, '_' and '.'. It is recommended (but not required) to use
##   only letters, numbers, and delimiters '_' or '.', as these are safe characters
##   for column names in R. [This is a message, not a warning or an error]

## final dispersion estimates

## fitting model and testing
```

```
res <- results(dds)

# Filter results
res_filtered <- res %>%
```

```r
  as.data.frame() %>%
  filter(!is.na(padj), !is.na(log2FoldChange), padj < 0.05, abs(log2FoldChange) > 2) %>%
  arrange(desc(abs(log2FoldChange)))
significant_genes <- res_filtered
gene_list <- rownames(significant_genes)

# Convert gene symbols to Entrez IDs
gene_symbols <- rownames(significant_genes)
gene_list <- bitr(gene_symbols, fromType = "SYMBOL", toType = "ENTREZID", OrgDb = org.Hs.eg.db)
```

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

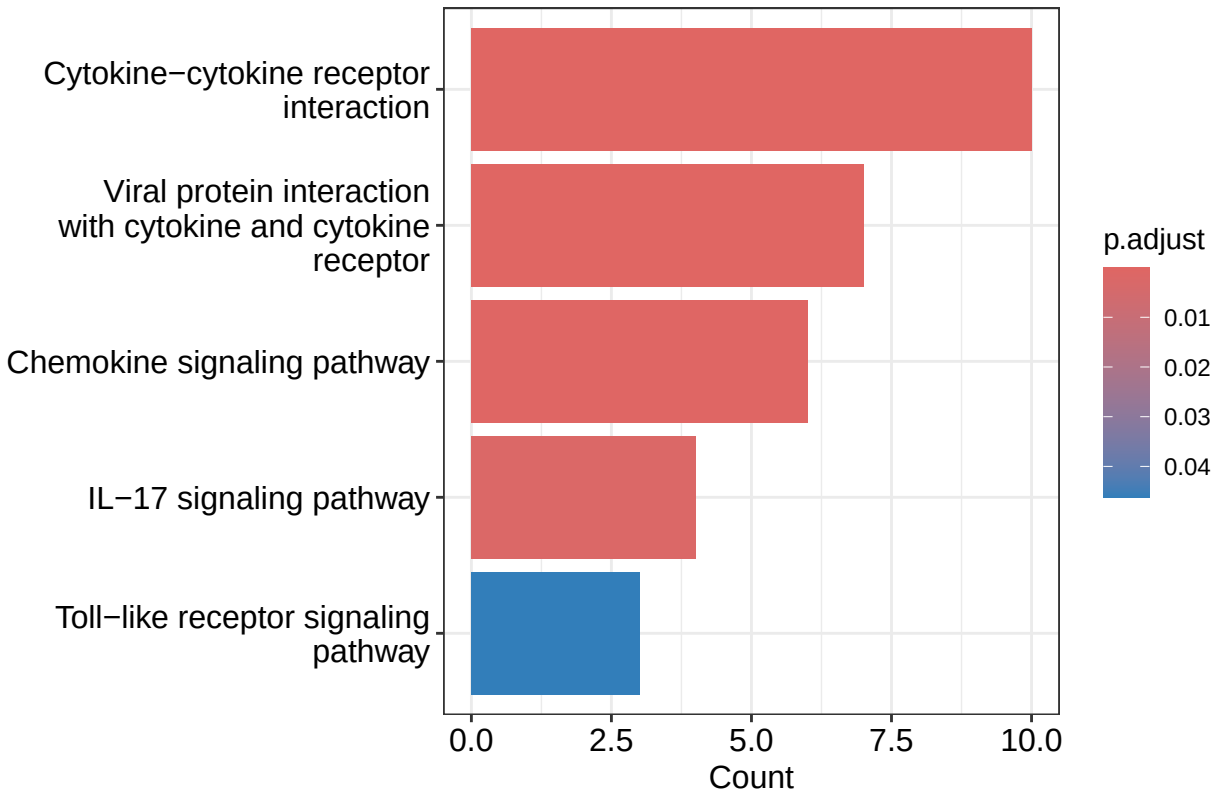
```r
# Filter out any NA values that may result from conversion
gene_list <- gene_list[!is.na(gene_list$ENTREZID), "ENTREZID"]

ekegg <- enrichKEGG(gene         = gene_list,
                    organism     = 'hsa', # Human
                    pvalueCutoff = 0.05)
```

```
## Reading KEGG annotation online: "https://rest.kegg.jp/link/hsa/pathway"...

## Reading KEGG annotation online: "https://rest.kegg.jp/list/pathway/hsa"...
```

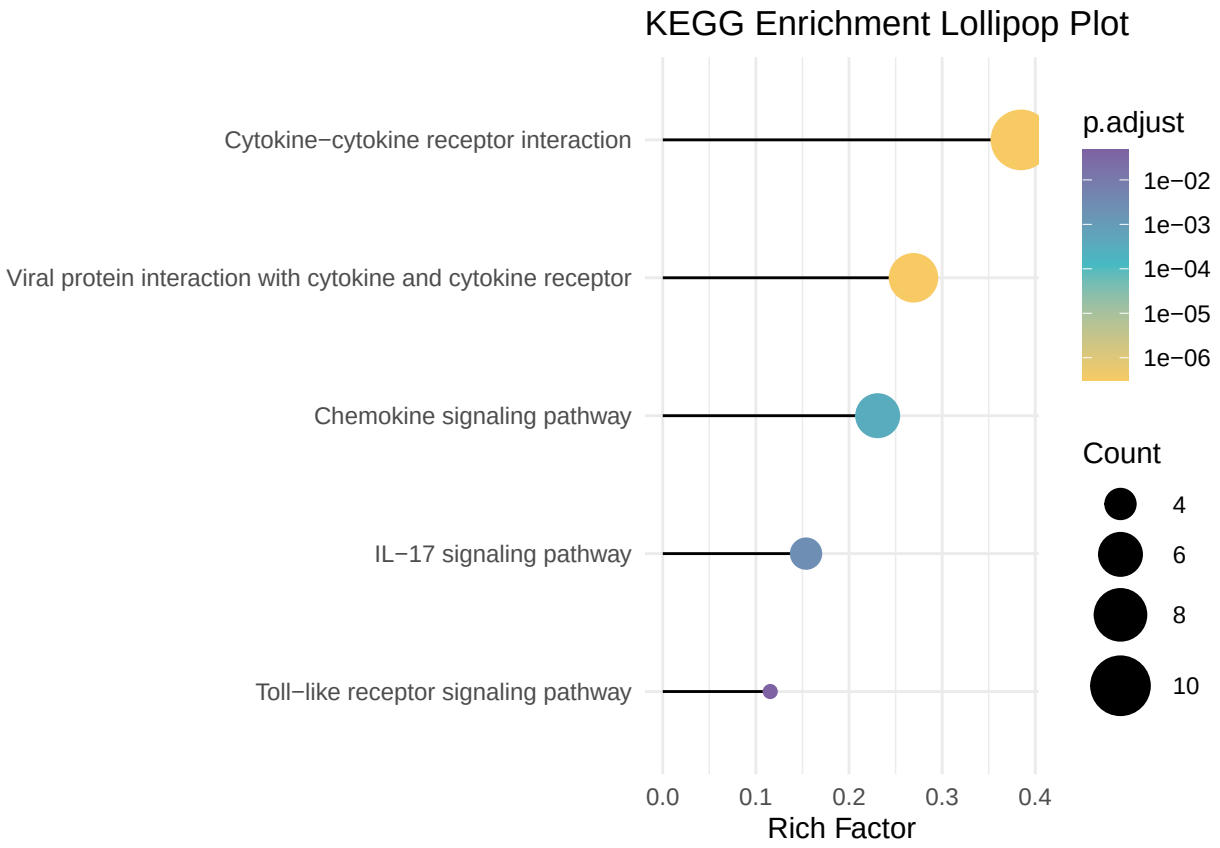
```r
barplot(ekegg, showCategory = 5)
```



```r
ekegg_df <- as.data.frame(ekegg@result)

# Assuming 'Count' and 'GeneRatio' are columns in your data frame
ekegg_df$RichFactor <- ekegg_df$Count / as.numeric(sapply(strsplit(ekegg_df$GeneRatio, "/"), `[`, 2))

# Create a lollipop plot using the calculated RichFactor
ggplot(ekegg_df[1:5, ], aes(x = RichFactor, y = fct_reorder(Description, RichFactor))) +
  geom_segment(aes(xend = 0, yend = Description)) +
  geom_point(aes(color = p.adjust, size = Count)) +
  scale_color_gradientn(colours = c("#f7ca64", "#46bac2", "#7e62a3"), trans = "log10") +
  scale_size_continuous(range = c(2, 10)) +
  theme_minimal() +
  xlab("Rich Factor") +
  ylab(NULL) +
  ggtitle("KEGG Enrichment Lollipop Plot")
```



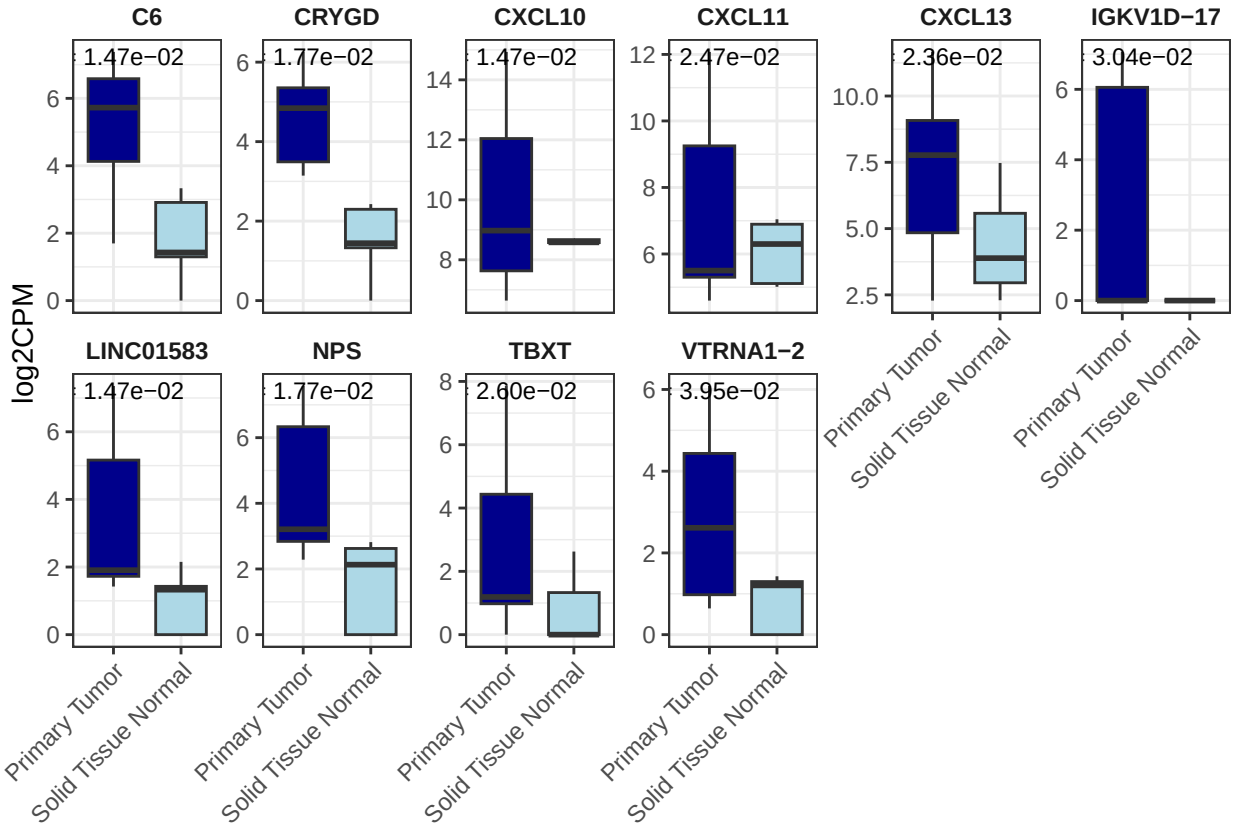

```r
# subset the upregulated genes from res_filtered object
# Order by log2FoldChange
res_upregulated <- res_filtered[order(res_filtered$log2FoldChange, decreasing = TRUE),]

# Select top 18 upregulated genes
top_upregulated <- head(res_upregulated, 10)

# Extract normalized counts for these genes
normalized_counts <- counts(dds, normalized=TRUE)[rownames(top_upregulated),]
```

```r
# Prepare data for plotting
plot_data <- normalized_counts %>%
  as.data.frame() %>%
  rownames_to_column("gene") %>%
  pivot_longer(cols = -gene, names_to = "sample", values_to = "count") %>%
  left_join(sample_info, by = c("sample" = "SampleID"))

# Add log2FoldChange and padj values
plot_data <- plot_data %>%
  left_join(as.data.frame(res_filtered) %>%
              rownames_to_column("gene") %>%
              dplyr::select(gene, log2FoldChange, padj),
            by = "gene")
# Create the plot without points
ggplot(plot_data, aes(x = tissue_type, y = log2(count + 1), fill = tissue_type)) +
  geom_boxplot(outlier.shape = NA) +
  facet_wrap(~ gene, scales = "free_y", ncol = 6) +
  scale_fill_manual(values = c("Solid Tissue Normal" = "lightblue", "Primary Tumor" = "darkblue")) +
  labs(y = "log2CPM", x = NULL) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.background = element_blank(),
        strip.text = element_text(face = "bold"),
        legend.position = "none") +
  geom_text(data = plot_data %>% group_by(gene) %>% slice(1),
            aes(x = tissue_type, y = Inf, label = sprintf("q = %.2e", padj)),
            vjust = 1.5, size = 3, inherit.aes = FALSE)
```




```r
# subset the upregulated genes from res_filtered object
# Order by log2FoldChange
res_downregulated <- res_filtered[order(res_filtered$log2FoldChange, decreasing = FALSE),]

# Select top 18 upregulated genes
top_downregulated <- head(res_downregulated, 10)

# Extract normalized counts for these genes
normalized_counts_down <- counts(dds, normalized=TRUE)[rownames(top_downregulated),]

# Prepare data for plotting
plot_data <- normalized_counts_down %>%
  as.data.frame() %>%
  rownames_to_column("gene") %>%
  pivot_longer(cols = -gene, names_to = "sample", values_to = "count") %>%
  left_join(sample_info, by = c("sample" = "SampleID"))

# Add log2FoldChange and padj values
plot_data <- plot_data %>%
  left_join(as.data.frame(res_filtered) %>%
              rownames_to_column("gene") %>%
              dplyr::select(gene, log2FoldChange, padj),
            by = "gene")
# Create the plot without points
ggplot(plot_data, aes(x = tissue_type, y = log2(count + 1), fill = tissue_type)) +
  geom_boxplot(outlier.shape = NA) +
```

```
  facet_wrap(~ gene, scales = "free_y", ncol = 6) +
  scale_fill_manual(values = c("Solid Tissue Normal" = "lightblue", "Primary Tumor" = "darkblue")) +
  labs(y = "log2CPM", x = NULL) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        strip.background = element_blank(),
        strip.text = element_text(face = "bold"),
        legend.position = "none") +
  geom_text(data = plot_data %>% group_by(gene) %>% slice(1),
            aes(x = tissue_type, y = Inf, label = sprintf("q = %.2e", padj)),
            vjust = 1.5, size = 3, inherit.aes = FALSE)
```