BGGN239 Class 2*

RNA-Seq mini project

Barry Grant

2023-03-25

Table of contents

*http://thegrantlab.org/teaching/

1. Background

The data for this hands-on session comes from a recently published RNA-seq experiment profiling multiple immune cell types from healthy (control) and **lupus** (disease) patient blood (Panwar et al. 2021).

Lupus (more formally known as **s**ystemic **l**upus **e**rythematosus, or **SLE** for short) is a chronic *autoimmune disease* in which the immune system attacks the body's own tissues and organs (Figure 1).

The exact cause of lupus is unknown, but is believed to be a combination of genetic, environmental, and hormonal factors leading to a heterogeneous aberrant autoimmune response (Kaul et al. 2016).

As the authors of the current study state *"A major obstacle in finding targeted therapies for SLE is its remarkable heterogeneity in clinical manifestations as well as the involvement of distinct cell types"* (Panwar et al. 2021). Their work aims to discover molecular signatures of SLE to help inform future therapies.

Outline

In this class session we will:

- Open a new *RStudio Project* and *Quarto document* for today's class;
- Explore the gene expression data available for this study on GEO;
- Use base R, GEOquery, dplyr and ggplot2 packages to answer specific questions about the experiments;

- Perform a detailed differential gene expression analysis with the DESeq2 package.
- Render a reproducible PDF report of your work with answers to all questions below.

2. Working with GEO

The Gene Expression Omnibus (GEO) from NCBI serves as the main public repository for a wide range of high-throughput gene expression data.

> More about GEO (click to expand)
>
> The **G**ene **E**xpression **O**mnibus (or GEO for short) is NCBIs main repository for high-throughput gene expression data, such as micro array and next-generation sequencing (NGS) data. The data in GEO is organized into three types of records: Sample, Series, and Platform records.

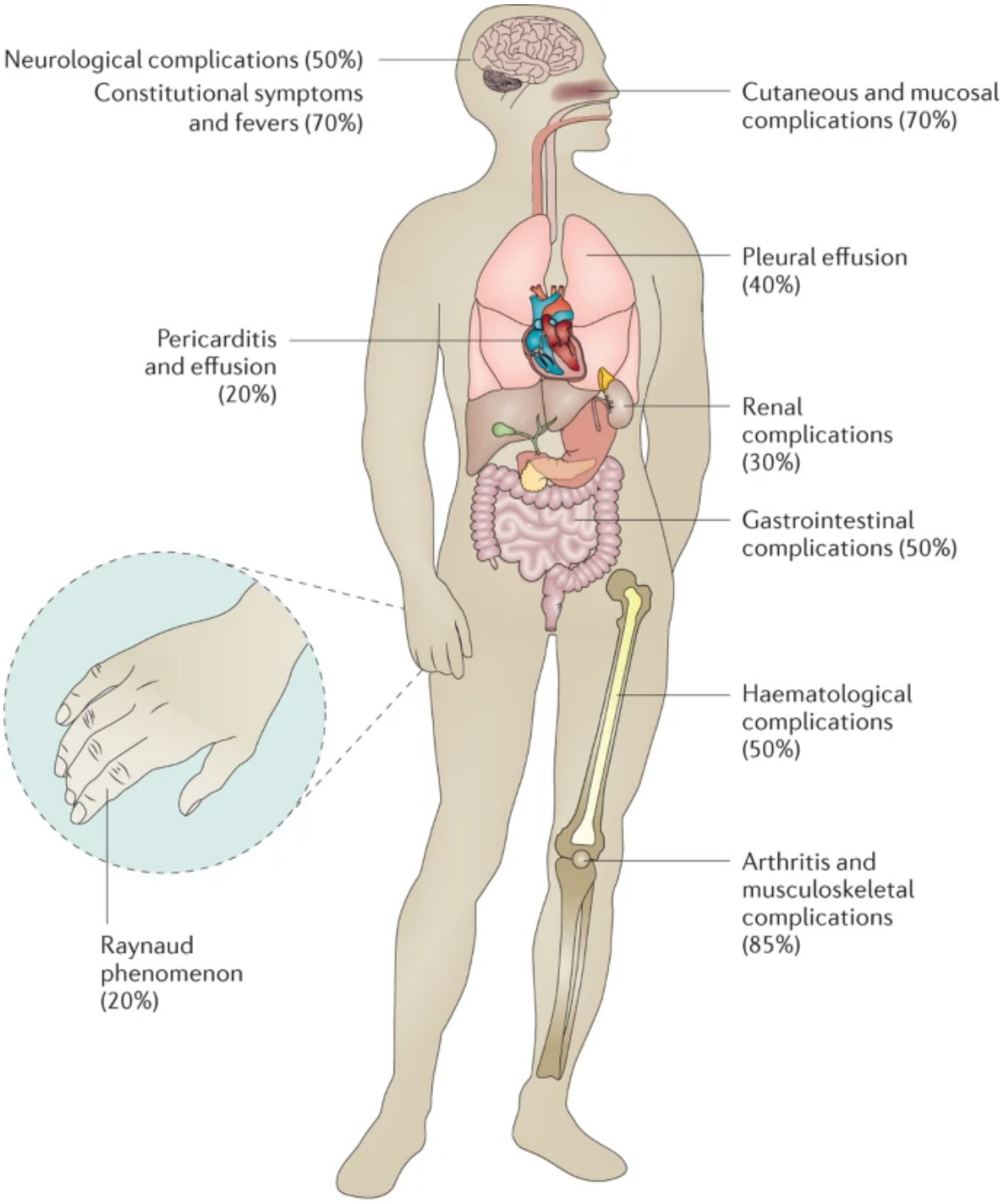


Figure 1: Clinical heterogeneity of SLE. Taken from Kaul *et al.* 2016.

- **Sample records** contain information about a single biological sample, including the experimental conditions, the type of platform used to generate the data, and the raw data files. Note their **GSM** accession codes in GEO.

- **Series records** group together related samples that were generated using similar experimental conditions or belong to the same study. Series records also contain additional metadata about the study, such as the experimental design, the hypothesis being tested, and the publications related to the study. Note their **GSE** accession codes in GEO.

- **Platform records** describe the platform or technology used to generate the data, including information about the probes, arrays, or sequencing methods used. Note their **GPL** accession codes in GEO.

All records in GEO are assigned a *unique accession number* that can be used to access and download the data. The data in GEO can be searched and accessed through the GEO website, or programmatically using the NCBI **Entrez** system or R packages such as **GEOquery** that we will use further below.

From publication to count data

Examining the PubMed entry (or full text) of the Panwar et al. (2021) paper leads us to the **GEO DataSets** with *SeriesAccession:* **GSE149050** under the Related information section.

Click through from PubMed to visit this GEO entry: GSE149050 and scroll down to find the **RawCounts** data.

Download the **GSE149050_Bulk_Human_RawCounts.txt.gz** file and move it to your RStudio project directory.

While this is downloading determine how many samples (GSM entries) are associates with this GEO series entry (GSE). Click on one of these entries to see how it describe the actual experimental data we will examine below.

- **Q1.** How many samples (GSM entries) are associated with this GEO series entry (GSE)?

Hint (click to expand)

You can use a Find in Page option of your web browser for the word "Samples". If the terminology is confusing for you here then you are not alone and you should visit the *More about GEO* expandable section above.

While our count data downloads we can use the GEOquery package form BioConductor to help organize and make sense of all these samples.

Using the GEOquery package

If necessary use the `BiocManager::install("GEOquery")` function call to install the package. We will also use **dplyr** and **ggplot2** packages to help explore this dataset further.

```
# Load the required packages
library(GEOquery)
library(dplyr)
library(ggplot2)
```

The main `getGEO()` function from GEOquery will return available metadata from this study and parse it into an R list object.

```
# Complete the code with the GEO ID
gse <- getGEO(___)
gse
```

```
$GSE149050_series_matrix.txt.gz
ExpressionSet (storageMode: lockedEnvironment)
assayData: 0 features, 288 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: GSM4489145 GSM4489146 ... GSM4489432 (288 total)
  varLabels: title geo_accession ... visit number:ch1 (64 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
  pubMedIds: 33674349
Annotation: GPL16791
```

A note about slow connections (click to expand)

If you are on a particularly slow WiFi connection and the `getGEO()` function gives a connection timeout error then we can download the matrix file from NCBI and read it directly:

```
# Download "Series Matrix File" from the web browser or in the Terminal
#wget https://ftp.ncbi.nlm.nih.gov/geo/series/GSE149nnn/GSE149050/matrix/GSE149050_series_m

# then in R
gse <- getGEO(filename="GSE149050_series_matrix.txt.gz")
```

As essentially all information in the GEO record is reflected in the resulting list data structure we will need to do a little work to focus on just the *experimental phenotypes* data we want.

We will extract information on experimental phenotypes with the `phenoData()` function and use `pData()` function to return this as a more conventional `data.frame` with samples as rows and variables as columns.

```
# Our gse list has only one entry
metadata <- pData(phenoData(gse[[1]]))
dim(metadata)
```

```
[1] 288  64
```

See how many "characteristics" are in this dataset

```
grep("characteristics", colnames(metadata), value=TRUE)
```

```
 [1] "characteristics_ch1"    "characteristics_ch1.1"  "characteristics_ch1.2"
 [4] "characteristics_ch1.3"  "characteristics_ch1.4"  "characteristics_ch1.5"
 [7] "characteristics_ch1.6"  "characteristics_ch1.7"  "characteristics_ch1.8"
[10] "characteristics_ch1.9"  "characteristics_ch1.10"
```

Let's look at a few of these. First note that the underscore "ch1" and "ch1.2" tell us the disease state (healthy or lupus) and cell type analyzed. For example, to find out how many "healthy control" samples are in the dataset we can use:

```
table(metadata$characteristics_ch1)[1]
```

```
disease state: healthy control
                            85
```

Or to get an "ethnicity" by "gender" breakdown:

```
table(metadata$characteristics_ch1.10, metadata$characteristics_ch1.7)
```

```
                                gender: Female gender: Male
  ethnicity: Hispanic                       24            0
  ethnicity: Hispanic/Latino                53            6
  ethnicity: Non Hispanic                   37            0
  ethnicity: Non-Hispanic/Latino           137            1
  ethnicity: Not available                  23            0
  ethnicity: not listed                      1            0
  ethnicity: Pacific Islander                6            0
```

Point for discussion: Why do you think this study features more Female than Male samples? What do you know about the prevalence of lupus? *ChatGPT can help you here* ;-)

Use a similar approach of examining the different "characteristics" columns to answer the following questions:

- **Q2.** How many Lupus (SLE) samples are there?
- **Q3.** True or False? There are more Female patient samples in the dataset than Male? TRUE FALSE
- **Q4.** How many different cell types were analyzed?
- **Q5.** Which cell type has the most samples?
- **Q6.** What does `ifn stats` refer to and why might this be important to consider for lupus and related diseases?

Hint (click to expand)

Try `table(metadata$characteristics_ch1)` etc. to help answer these questions. The base R `table()` function is super useful and we will need it several times in this lab.
The interferon response in lupus is sometimes referred to as the "interferon signature" or "interferon pathway activation," and it is commonly observed in the blood of patients with lupus. Researchers are still working to fully understand the role of the interferon response in the development and progression of lupus, and how it can be targeted for therapeutic benefit. Some current treatments for lupus, such as hydroxychloroquine and belimumab, are thought to work by inhibiting the interferon response.

3. Select your favorate immune cell type

You have by now found that this study targeted multiple major immune cell types. You should now **select YOUR favorite immune cell type** and use it for future sections of this lab.

For demonstration purposes I will select my favorite **classical monocytes (cMo)**. They are the most abundant sub-type of monocytes in the blood and changes in their numbers and/or function has been observed in various inflammatory and autoimmune diseases, including lupus, rheumatoid arthritis, and atherosclerosis. Therefore, the study of classical monocytes and their role in immune regulation is an important area of research.

Here I will select all the `"cell type: cMo"` samples for `""visit number: 1"` with the help of **dplyr** `filter()` function:

```
metadata.cmo <- filter(metadata, characteristics_ch1.2 == "cell type: cMo" &
                       characteristics_ch1.6 == "visit number: 1")
head(metadata.cmo[,1:3])
```

```
                          title geo_accession                status
GSM4489314 170_S3004a_IFNpos_cMo    GSM4489314 Public on Feb 01 2021
GSM4489318 174_S1055a_IFNpos_cMo    GSM4489318 Public on Feb 01 2021
GSM4489321 177_S1082a_IFNpos_cMo    GSM4489321 Public on Feb 01 2021
GSM4489322 178_S3008a_IFNpos_cMo    GSM4489322 Public on Feb 01 2021
GSM4489323 179_S3011a_IFNpos_cMo    GSM4489323 Public on Feb 01 2021
GSM4489325 181_S1065a_IFNpos_cMo    GSM4489325 Public on Feb 01 2021
```

Use a similar approach of examining the different "characteristics" columns to answer the following questions:

- **Q7.** How many lupus and healthy control samples are there for your cell type?
- **Q8.** How were these samples processed (alignment/mapping software version and genome build used)?

Hint (click to expand)

For cMo we can use `table()` function as before:

```
table(metadata.cmo$characteristics_ch1)
```

```
                  disease state: healthy control
                                              24
disease state: systemic lupus erythematosus (SLE)
                                              64
```

For the next question we can examine the `$data_processing` columns, For example:

```
metadata.cmo$data_processing[1]
```

```
[1] "Bulk RNA-seq data (FASTQ files) were mapped against the hg38 genome (GRCh38.p7) refe
```

Book-keeping: Focus on columns of interest

Now that we have made some initial explorations of the dataset we have some book-keeping and renaming to get things better organized for further analysis.

We will use the **dplyr** `select()` and `mutate()` functions extensively here:

Select the columns of interest.

```
# TO DO: Build up this example by piping to head() after each line
metadata.subset <- metadata.cmo %>%
  select(title,
         disease_state = characteristics_ch1,
         ifn_status = characteristics_ch1.3,
         patient_id = characteristics_ch1.4) %>%
  mutate(disease_state = gsub("disease state: ","", disease_state)) %>%
```

```
  mutate(ifn_status = gsub("ifn status: ","", ifn_status)) %>%
  mutate(patient_id = gsub("patientuid: ","", patient_id)) %>%
  mutate(state = recode(disease_state,
                        "healthy control" = "control",
                        "systemic lupus erythematosus (SLE)" = "lupus") )
```

```
table(metadata.subset$state)
```

```
control   lupus
     24      64
```

4. Read count data

Finally, we are ready to read in the **count data** for all samples that we downloaded from GEO in Section 2 above. Note that the file is compressed with `gzip` and you may need to "unzip" it if your browser did not already do this. You can use the bash/UNIX command `gunzip` or `gzip -d` if you have a mac or PC with **git bash** setup. Ask Barry if you are not sure about this.

```
counts.all <- read.delim("GSE149050_Bulk_Human_RawCounts.txt",
                         check.names=FALSE, row.names = 1)
head(counts.all[,1:3])
```

```
          001_L0038_HC_T 002_L0088fresh_HC_T 003_L0140_HC_T
5S_rRNA                2                    0              2
5_8S_rRNA              0                    0              0
7SK                    1                    1              2
A1BG                  53                   56            105
A1BG-AS1               7                   24             46
A1CF                   0                    2              2
```

Note that we are setting the `row.names` as Gene IDs here and making sure the column names are not prepended with an `X` via `check.names`. Try running without these arguments to see the difference in output format.

Select only the samples we want

Select only the columns (samples) that correspond to the cell type we want to examine. We have these in `metadata.subset$title`

```
counts.subset <- counts.all %>% select(metadata.subset$title)
```

You should check here that your `counts.subset` columns and `metadata.subset` rows match.

- **Q9.** How many columns does your `counts.subset` object contain (this should be the same as the number of rows in `metadata.subset`)?

Hint (click to expand)

You can use `nrow()`, `ncol()` or simply `dim()` here:

```
dim(counts.subset)
dim(metadata.subset)
```

5. Setting up for DESeq

For DESeq we need to do some book-keeping and get the data in the correct format. First double check on column correspondences between counts and metadata

```
all(colnames(counts.subset) == metadata.subset$title)
```

```
[1] TRUE
```

Finally, DESeq needs our sample names as rownames of the metadata.

```
# Remove the title column then use it as rownames
colData <- metadata.subset[,-1]
rownames(colData) <- metadata.subset[,1]
head(colData)
```

```
                                                 disease_state ifn_status patient_id
170_S3004a_IFNpos_cMo systemic lupus erythematosus (SLE)     IFNpos      S3004
174_S1055a_IFNpos_cMo systemic lupus erythematosus (SLE)     IFNpos      S1055
177_S1082a_IFNpos_cMo systemic lupus erythematosus (SLE)     IFNpos      S1082
178_S3008a_IFNpos_cMo systemic lupus erythematosus (SLE)     IFNpos      S3008
179_S3011a_IFNpos_cMo systemic lupus erythematosus (SLE)     IFNpos      S3011
181_S1065a_IFNpos_cMo systemic lupus erythematosus (SLE)     IFNpos      S1065
                      state
170_S3004a_IFNpos_cMo lupus
174_S1055a_IFNpos_cMo lupus
177_S1082a_IFNpos_cMo lupus
178_S3008a_IFNpos_cMo lupus
179_S3011a_IFNpos_cMo lupus
181_S1065a_IFNpos_cMo lupus
```

Now we can setup our DESeq object with the long but appropriately named `DESeqDataSetFromMatrix()` function:

```
library(DESeq2)

dds <- DESeqDataSetFromMatrix(countData = counts.subset,
                              colData = colData,
                              design = ~state)
```

Before we go any further we should remove any genes that have very low read counts. Here we will follow general practice of keeping rows that have at least 10 reads total. This will speed up our future calculations and avoid diluting our statistical power by doing needless tests on genes that we can not say much about anyway.

```
keep.inds <- rowSums(counts(dds)) >= 10
dds <- dds[keep.inds,]
```

- **Q10.** How many genes did we remove here?

6. Principal Component Analysis (PCA)

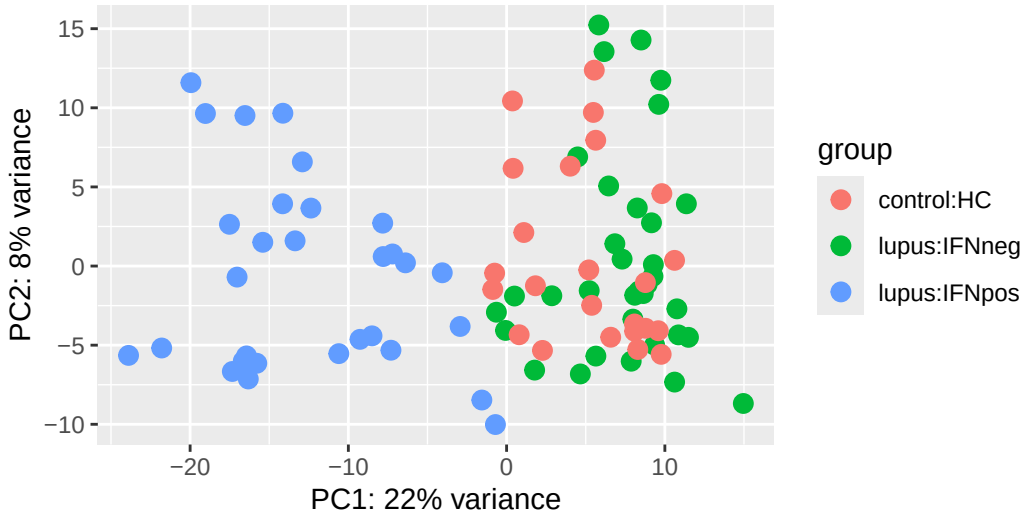
Before running DESeq analysis we can look how the count data samples are related to one another via our old friend Principal Component Analysis (PCA). We will follow the DESeq recommended procedure and associated functions for PCA. First calling `vst()` to apply a

variance stabilizing transformation (read more about this in the expandable section below) and then `plotPCA()` to calculate our PCs and plot the results.

> Variance stabilizing transformation (click to expand)
>
> The purpose of the variance stabilizing transformation in DESeq2 is to normalize count data and stabilize the variance across the mean expression level. This is important because count data often have different variances at different levels of expression, and this can lead to false positive or false negative results in downstream analysis. The variance stabilizing transformation is a type of power transformation that aims to stabilize the variance across the mean expression level of genes.
> The `vst()` function in DESeq2 transforms raw count data into a log2-counts per million (logCPM) space, where the variance is approximately independent of the mean. This transformation improves the performance of statistical tests and reduces the impact of outliers on the analysis. The variance-stabilized data can then be used for exploratory data analysis, visualization, and downstream statistical analysis, such as differential expression analysis, clustering, and dimension reduction. For motivated readers this vignette has additional details.

```
vsd <- vst(dds, blind = FALSE)
plotPCA(vsd, intgroup = c("state", "ifn_status"))
```

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



The `plotPCA()` function comes with DESeq2 and the two terms specified by `intgroup` are the interesting groups for labeling the samples; they tell the function to use them to choose colors.

We can also build the PCA plot from scratch using the **ggplot2 package**. This is done by asking the plotPCA function to return the data used for plotting rather than building the plot.

```
pcaData <- plotPCA(vsd, intgroup=c("state", "ifn_status"), returnData=TRUE)
```

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

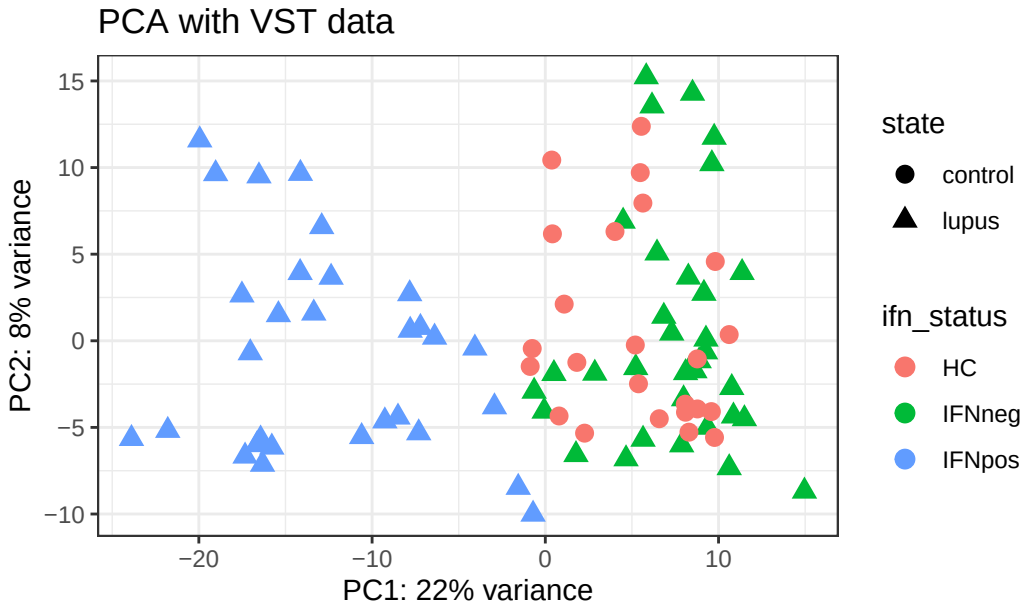
```
head(pcaData)
```

```
                            PC1        PC2        group state ifn_status
170_S3004a_IFNpos_cMo  -7.799237  0.6012251 lupus:IFNpos lupus     IFNpos
174_S1055a_IFNpos_cMo -15.788003 -6.1432683 lupus:IFNpos lupus     IFNpos
177_S1082a_IFNpos_cMo -23.889066 -5.6467291 lupus:IFNpos lupus     IFNpos
178_S3008a_IFNpos_cMo -15.407654  1.5003440 lupus:IFNpos lupus     IFNpos
179_S3011a_IFNpos_cMo  -9.263411 -4.6312344 lupus:IFNpos lupus     IFNpos
181_S1065a_IFNpos_cMo -21.794772 -5.1795433 lupus:IFNpos lupus     IFNpos
                                      name
```

```
170_S3004a_IFNpos_cMo 170_S3004a_IFNpos_cMo
174_S1055a_IFNpos_cMo 174_S1055a_IFNpos_cMo
177_S1082a_IFNpos_cMo 177_S1082a_IFNpos_cMo
178_S3008a_IFNpos_cMo 178_S3008a_IFNpos_cMo
179_S3011a_IFNpos_cMo 179_S3011a_IFNpos_cMo
181_S1065a_IFNpos_cMo 181_S1065a_IFNpos_cMo
```

```
# Calculate percent variance per PC for the plot axis labels
percentVar <- round(100 * attr(pcaData, "percentVar"))
```

```
ggplot(pcaData) +
  #aes(x = PC1, y = PC2, color = state, shape = ifn_status) +
  aes(x = PC1, y = PC2, color = ifn_status, shape=state) +
  geom_point(size =3) +
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance")) +
  coord_fixed() +
  ggtitle("PCA with VST data") +
  theme_bw()
```



TO DO: - **Q10:** Question about PCA result interpenetration. Comment on the low variance captured. Lack of lupus vs control split. But there is this interesting IFN split to examine. More on this next day! We should run DESeq of lupus:IFNpos vs healthy etc…

7. Running DESeq analysis

Now let's run the DESeq analysis pipeline on the dataset, and reassign the resulting object back to the same `dds` object:

```
dds <- DESeq(dds)
```

Here, we're running the DESeq pipeline on the `dds` object and it can take some time. The `DESeq()` function calls a number of other functions within the package to essentially run the entire pipeline (normalizing by library size by estimating the "size factors," estimating dispersion for the negative binomial model, and fitting models and getting statistics for each gene for the design specified when we imported the data).

Getting results

Since we've got a fairly simple design (single factor, two groups, lupus vs control), we can get results out of the object simply by calling the `results()` function on the DESeqDataSet that has been run through the pipeline.

The help page for `?results` and the vignette both have extensive documentation about how to pull out the results for more complicated models (multi-factor experiments, specific contrasts, interaction terms, time courses, etc.).

```
res <- results(dds)
head(res)
```

```
log2 fold change (MLE): state lupus vs control
Wald test p-value: state lupus vs control
DataFrame with 6 rows and 6 columns
          baseMean log2FoldChange     lfcSE      stat    pvalue      padj
         <numeric>      <numeric> <numeric> <numeric> <numeric> <numeric>
5S_rRNA   8.668615      0.0951538  0.170068  0.559505 0.5758170  0.935789
7SK       0.733043     -0.5730441  0.617795 -0.927564 0.3536339        NA
A1BG     91.745817     -0.2840853  0.150951 -1.881973 0.0598397  0.570403
A1BG-AS1 25.942398      0.3101825  0.230798  1.343956 0.1789626  0.758740
A1CF      1.017252     -0.3106790  0.410694 -0.756474 0.4493650        NA
A2M       8.605330     -0.1413624  0.629530 -0.224552 0.8223276  0.979400
```

We can summarize some basic tallies using the `summary()` function.

```
summary(res)
```

```
out of 32472 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 173, 0.53%
LFC < 0 (down)     : 37, 0.11%
outliers [1]       : 0, 0%
low counts [2]     : 13381, 41%
(mean count < 4)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

And with a different `alpha` (adjusted p-value) threshold of 0.05:

```
res_p05 <- results(dds, alpha=0.05)
summary(res_p05)
```

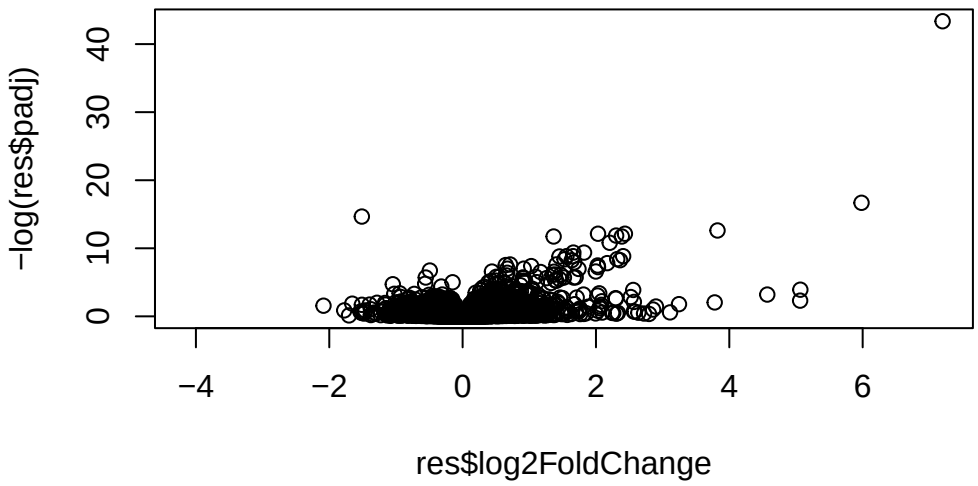
```
out of 32472 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)       : 121, 0.37%
LFC < 0 (down)     : 14, 0.043%
outliers [1]       : 0, 0%
low counts [2]     : 16528, 51%
(mean count < 10)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

The low numbers here are consistent with the (Panwar et al. 2021) paper. This set of "differentially expressed genes" will be interrogated further in *Class 3* and *Class 4*. We will finish up this class session by visualizing and saving our results.

8. Visualizating and saving results

First volcano plot with no color or annotation.

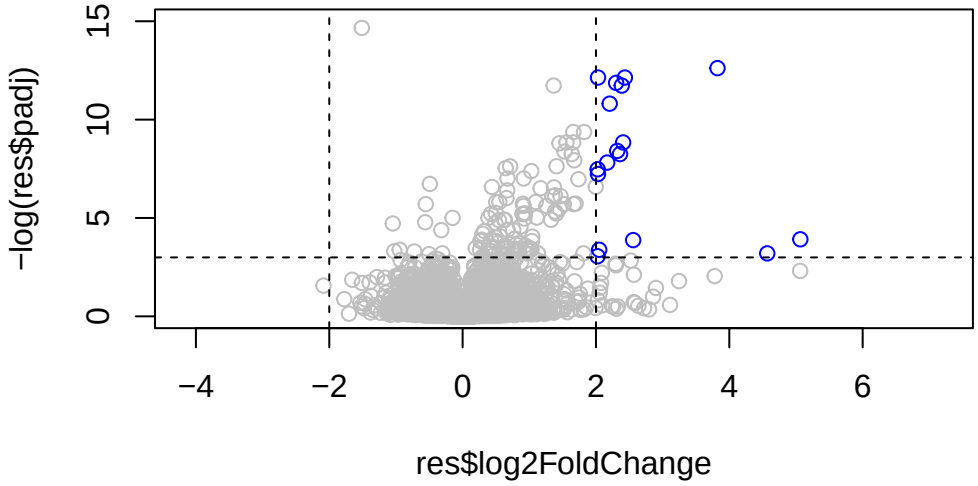
```
plot(res$log2FoldChange, -log(res$padj))
```



Let's add some color with a color vector, `mycols`, based on p-value and fold change:

```
mycols <- rep("gray", nrow(res))
mycols[ abs(res$log2FoldChange) > 2] = "blue"
mycols[ res$padj > 0.05 ] = "gray"

plot(res$log2FoldChange, -log(res$padj), ylim=c(0,15), col=mycols)
abline(v=c(-2,2), lty=2)
abline(h=-log(0.05), lty=2)
```



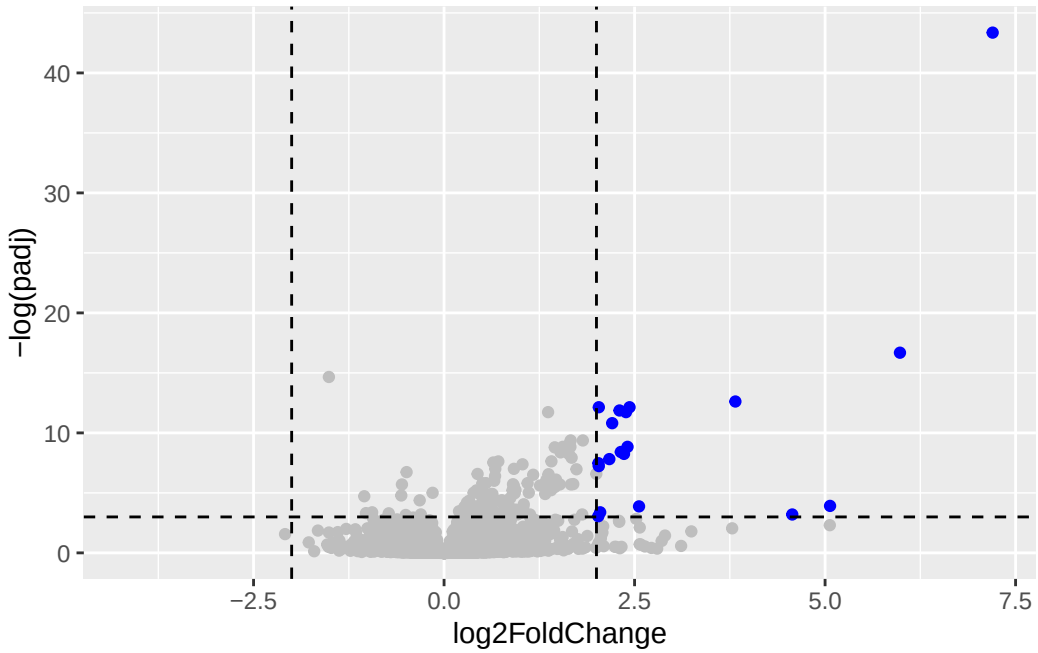
and make a ggplot version

```
results <- as.data.frame(res)

library(ggplot2)

ggplot(results) +
  aes(log2FoldChange, -log(padj)) +
  geom_point(col=mycols) +
  geom_vline(xintercept = c(-2,+2), linetype=2) +
  geom_hline(yintercept = -log(0.05), linetype=2)
```

```
Warning: Removed 13381 rows containing missing values or values outside the scale range
(`geom_point()`).
```



Let's finally extract our top genes of interest for the NEXT CLASS!

```
top.genes <- results %>% filter(padj <= 0.05 & abs(log2FoldChange) >= 2)
head(top.genes)
```

	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
ADAMTS2	80.75075	3.823089	0.6197866	6.168395	6.898659e-10	3.320842e-06
CCL2	170.72446	2.319374	0.4458390	5.202268	1.968713e-07	2.229857e-04
CCL8	29.79196	2.558495	0.6535411	3.914819	9.047210e-05	2.073858e-02
ETV7	159.00642	2.025962	0.4098111	4.943648	7.667391e-07	5.678293e-04
HESX1	23.23881	2.023995	0.5600267	3.614104	3.013877e-04	4.718064e-02
IFI27	1743.22556	7.199210	0.7150802	10.067695	7.675604e-24	1.477938e-19

We can ask **ChatGPT** and **Google Bard** about these genes and or do some pathway analysis. We will also save full DESeq results as a CSV file:

```
save(top.genes, file = "top_genes.RData")
write.csv(res, file="deseq_results.csv")
```

9. Optional Extension

As an optional extension let's can compare IFNneg vs. IFNpos (lupus:IFNpos vs healthy control (or control and lupus:IFNneg)).

Setup a new metadata/colData object for IFNneg vs. IFNpos

```
table(colData$ifn_status)
```

```
    HC IFNneg IFNpos
    24     34     30
```

```
ifn_colData <- filter(colData, ifn_status %in% c("IFNneg", "IFNpos"))
#ifn_colData <- filter(colData, ifn_status %in% c("HC", "IFNpos"))

table(ifn_colData$ifn_status)
```

```
IFNneg IFNpos
    34     30
```

```
# Select only IFNneg and IFNpos count columns
ifn_countData <- select(counts.subset, rownames(ifn_colData))
dim(ifn_countData)
```

```
[1] 56269    64
```

```
dim(ifn_colData)
```

```
[1] 64  4
```

Now we can setup and run DESeq as before:

```
ifn_dds <- DESeqDataSetFromMatrix(countData = ifn_countData,
                                  colData = ifn_colData,
                                  design = ~ifn_status)

ifn_dds <- DESeq(ifn_dds)
ifn_res <- results(ifn_dds)
```

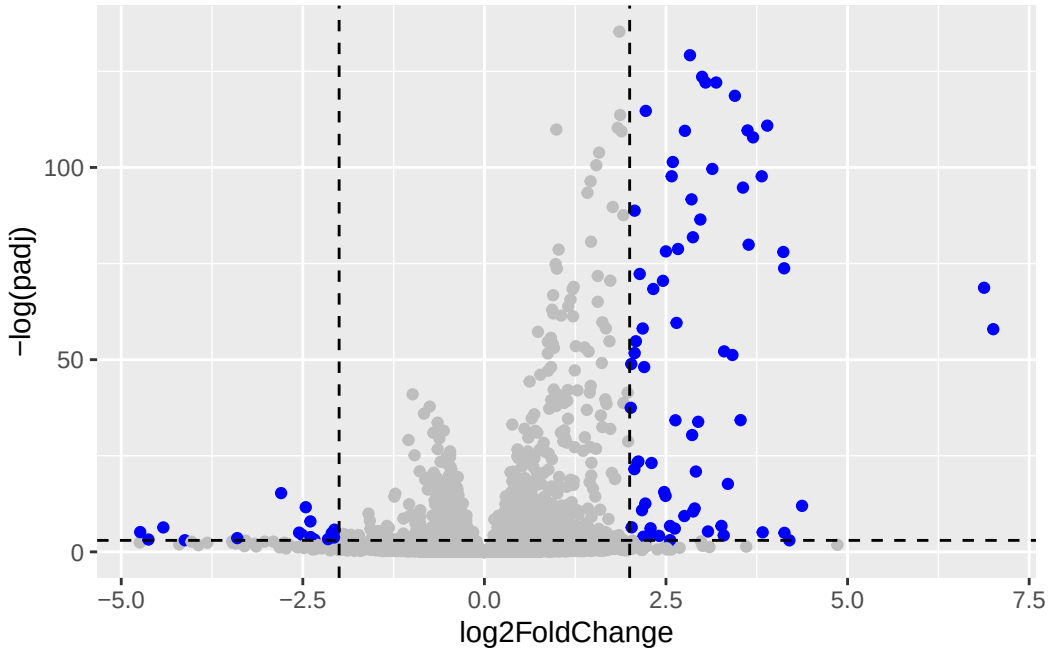
and a plot

```
ifn_results <- as.data.frame(ifn_res)

ifn_cols <- rep("gray", nrow(ifn_res))
ifn_cols[ abs(ifn_res$log2FoldChange) > 2] = "blue"
ifn_cols[ ifn_res$padj > 0.05 ] = "gray"


ggplot(ifn_results) +
  aes(log2FoldChange, -log(padj)) +
  geom_point(col=ifn_cols) +
  geom_vline(xintercept = c(-2,+2), linetype=2) +
  geom_hline(yintercept = -log(0.05), linetype=2)
```

```
Warning: Removed 34189 rows containing missing values or values outside the scale range
(`geom_point()`).
```



Let's finally extract our top genes of interest for the NEXT CLASS!

```
ifn_genes <- ifn_results %>% filter(padj <= 0.05 & abs(log2FoldChange) >= 2)
head(ifn_genes)
```

```
              baseMean log2FoldChange     lfcSE      stat       pvalue
ABCA6         11.359801      -2.508709 0.7199788 -3.484420 4.932048e-04
ADAMTS5       63.814100      -2.459209 0.4663791 -5.272982 1.342244e-07
AGRN          14.421869       2.475145 0.4115442  6.014288 1.806792e-09
ANKRD22      203.431431       2.301576 0.3202911  7.185887 6.677221e-13
AP001610.5     3.764462       2.288457 0.5708389  4.008936 6.099291e-05
APOBEC3A    5954.424665       2.068756 0.1975059 10.474398 1.132563e-25
                   padj
ABCA6      1.196396e-02
ADAMTS5    9.008314e-06
AGRN       1.727011e-07
ANKRD22    9.100805e-11
AP001610.5 2.148619e-03
APOBEC3A   3.473194e-23
```

```
dim(ifn_genes)
```

```
[1] 88  6
```

Look up TNFSF13B and IL1RN for the original lupus vs control run:

```
i1 <- grep("IL1RN", rownames(rowData(dds)))
i2 <- grep("TNFSF13B", rownames(rowData(dds)))
results[c(i1,i2),]
```

```
          baseMean log2FoldChange      lfcSE     stat      pvalue       padj
IL1RN     1091.299      0.5494599 0.14987570 3.666104 0.000246274 0.04067994
TNFSF13B 4733.300      0.2611536 0.09395405 2.779588 0.005442786 0.22156625
```

And for IFN-pos vs IFN-neg:

```
i1 <- grep("IL1RN", rownames(rowData(ifn_dds)))
i2 <- grep("TNFSF13B", rownames(rowData(ifn_dds)))
ifn_results[c(i1,i2),]
```

```
          baseMean log2FoldChange      lfcSE     stat       pvalue         padj
IL1RN     1189.982      0.9300144 0.12696031 7.325237 2.384762e-13 3.397132e-11
TNFSF13B 4938.118      0.6524722 0.07520505 8.675910 4.102581e-18 8.465886e-16
```

Great job, we will stop here for today and take these results further in our next **Class 3**.

10. Session Information

The `sessionInfo()` prints version information about R and any attached packages. It's a good practice to always run this command at the end of your R session and record it for the sake of reproducibility in the future.

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: aarch64-apple-darwin20
Running under: macOS Sequoia 15.3.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  l

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] DESeq2_1.46.0               SummarizedExperiment_1.36.0
 [3] MatrixGenerics_1.18.1       matrixStats_1.5.0
 [5] GenomicRanges_1.58.0        GenomeInfoDb_1.42.3
 [7] IRanges_2.40.1              S4Vectors_0.44.0
 [9] ggplot2_3.5.1               dplyr_1.1.4
[11] GEOquery_2.74.0             Biobase_2.66.0
[13] BiocGenerics_0.52.0         labsheet_0.1.2

loaded via a namespace (and not attached):
 [1] gtable_0.3.6            httr2_1.1.1             xfun_0.51
 [4] lattice_0.22-6          tzdb_0.5.0              vctrs_0.6.5
 [7] tools_4.4.2             generics_0.1.3          curl_6.2.2
[10] parallel_4.4.2          tibble_3.2.1            R.oo_1.27.0
[13] pkgconfig_2.0.3         Matrix_1.7-3            data.table_1.17.0
[16] rentrez_1.2.3           lifecycle_1.0.4         GenomeInfoDbData_1.2.13
```

```
[19] farver_2.1.2           compiler_4.4.2         statmod_1.5.0
[22] munsell_0.5.1          codetools_0.2-20       htmltools_0.5.8.1
[25] yaml_2.3.10            pillar_1.10.1          crayon_1.5.3
[28] tidyr_1.3.1            R.utils_2.13.0         BiocParallel_1.40.0
[31] DelayedArray_0.32.0    limma_3.62.2           abind_1.4-8
[34] locfit_1.5-9.12        tidyselect_1.2.1       digest_0.6.37
[37] purrr_1.0.4            labeling_0.4.3         fastmap_1.2.0
[40] grid_4.4.2             colorspace_2.1-1       cli_3.6.4
[43] SparseArray_1.6.2      magrittr_2.0.3         S4Arrays_1.6.0
[46] XML_3.99-0.18          readr_2.1.5            withr_3.0.2
[49] rappdirs_0.3.3         UCSC.utils_1.2.0       scales_1.3.0
[52] rmarkdown_2.29         XVector_0.46.0         httr_1.4.7
[55] R.methodsS3_1.8.2      hms_1.1.3              evaluate_1.0.3
[58] knitr_1.50             rlang_1.1.5            Rcpp_1.0.14
[61] glue_1.8.0             formatR_1.14           xml2_1.3.8
[64] rstudioapi_0.17.1      jsonlite_1.9.1         R6_2.6.1
[67] zlibbioc_1.52.0
```