Analysis of RNA-seq Data

Michael J. Guertin

April 1, 2020

This vignette provides step-by-step procedure for analysis of RNA-seq data.

Contents

List of Figures

1 RNA isolation and library preparation

1.1 Before beginning analysis

When designing an RNA-seq experiment, the first step is to determine what type of molecular biology workflow to perform for the sample preparation and the library preparation. Likewise, one needs to understand the molecular biology that generated the data of publicly available data.

1.2 RNA extraction

There are many different methods to generate RNA-seq data. RNA extraction is routine and kits are often used to extract total RNA. Although a DNase treatment is optional for many kits, I recommend this step.

1.3 Poly-A selection and ribosomal RNA depletion

Over 80% of the RNA in a given cell is ribosomal RNA (rRNA). Most researchers view rRNA as uninformative for the purposes of RNA-seq and there are two main methods to enrich for mRNA: 1) enrichment of poly-adenylated (poly-A) RNAs–typically referred to a mRNA library prep; and 2) rRNA depletion–typically referred to as total RNA library prep. Both methods involve DNA/RNA hybridization to enrich or deplete the RNA species of interest. For mRNA preps, poly-A RNA is hybrized to oligo-dT beads and non-hybridized RNA is washed away. The positively selected poly-A RNA is then eluted and used as the input. For total RNA preparations, the ribosomal RNAs are hybridized to ssDNA that is complementary to the rRNA sequence. This negatively selects rRNA and the supernant is the input for the library preparation. Alternatively this DNA/RNA hybrid is digested with RNaseH. The H refers to the RNase specificity–only digesting RNA in a DNA **H**ybrid.

1.4 Molecular biology of library preps

The molecular biology of library preps diverge depending upon whether the experimenter wishes to preserve the RNA strand information in the sequencing data. Unstranded libraries were common over a decade ago, but I see no reason why contemporary research would employ strand-ambiguous RNA-seq. Therefore, I will briefly discuss the workflow for the library prepartion that I prefer and how clever molecular biologists developed methods to retain the strand specificity of the RNA throughout the library prep.
1) Fragment the RNA with heat.
2) Anneal random hexamer DNA primers to the RNA.
3) Perform reverse transcription of the RNA using a reverse transcriptase that is engineered to operate at high temperatures. RT at higher temperatures relieves secondary structure of the RNA and leads to less biased libraries. Including actinomycin D at this step prevents second strand synthesis during the RT by inhibiting DNA-dependent DNA synthesis.
4) Perform second strand synthesis using the dUTP nucleotide as opposed to dTTP. dUTP incorporation is important for conferring strand specificity.
5) The next step is to blunt the ends of the DNA and generate a 5′ A overhang. T4 Polymerase harbors 3′ to 5′ exonuclease activity and 5′ to 3′ polymerase activity, which will blunt both overhangs. T4 Polymerase also phosphorylates 5′ DNA ends.
6) Taq polymerase is used to add an A to the 3′ end of the dsDNA. The high throughput sequencing adapter is usually a forked Y adapter that contains a T overhang on the 5′ end. This pairs with the A overhang, traditionally referred to as TA cloning. Alternatively, a hairpin adapter is used that incorporates a dUTP that effectively generates a Y adapter when the dUTP is digested.
7) Digest dUTPs incorporated during the second strand synthesis with uracil-DNA glycosylase (UGDase). This confers sequence specificity because now you have a single strand DNA molecule with a distinct adapter on each end. The ssDNA represents the template strand.
8) The last step is to PCR amplify with a variety of multiplexed adapters that permit pooling of samples.

The dsDNA molecules can be sequenced using a specific primer for each end of the molecule. Refer to step 7 to determine which sequencing primer would result in the coding strand and which would be the reverse complement thereof.

1.5 Small (17-35 nucleotides) RNA preps

Even small RNA-seq libraries are generated from total RNA preparations, but a poly-A enrichment or rRNA depletion step is not necessary, as the libraries are size selected. For small RNA-seq libraries, one typically ligates the 3′ adapter and the 5′ adapter directly to RNAs isolated from cells. RNA isolation preserves the 3′ Hydroxyl and 5′ Phosphorylation modifications, so the input does not require enzymatic pre-treatment for ligation competency. The 3′ adapter and the 5′ adapter have distinct sequences, so strand specificity is preserved throughout the work flow.

1.6 Additional resources

The following is a good reference for the molecular biology behind different RNA-seq protocols: `https://www.neb.com/~/media/nebus/files/brochures/nebnext_poster.pdf`

1.7 Designing experiments

The second step, particularly when designing RNA-seq experiments, is to be aware of the features and limitations of your experimental design. A single experiment will provide the relative expression levels of RNAs in a particular cell type. This may be useful if you are looking to classify an unknown cell type by comparing to a panel of known transcription profiles of various tissues or cell types. This may be useful if your purpose is to annotate the transcriptome of a poorly characterized tissue/organism. Shotgun RNA-seq could also be used to characterize the genome of a novel RNA virus in order to develop a dignostic test. However, it would be a challenge to use these data to develop new hypotheses to test. Another common implmentation of RNA-seq is to compare expression between two conditions. This type of design will let you know which genes and isoforms are differentially expressed. These differentially expressed genes can be input to software that performs gene set enrichment analysis (GSEA) (Subramanian *et al.*, 2005) to identify biological pathways that are changing or to identify which transcription factor is likely regulating the differentially expressed genes (Wang *et al.*, 2018). One can begin to develop testable hypotheses from these data.

One can also use RNA-seq to test hypotheses, as outlined by this hypothetical example: One treats human cells with the plant hormone auxin and performs RNA-seq to determine which genes change expression. GSEA (Subramanian *et al.*, 2005) or Gene Ontology (Mi *et al.*, 2019) analysis indicates that canonical aryl hydrocarbon responsive genes are activated. BART (Wang *et al.*, 2018) identifies the Aryl Hydrocarbon Receptor (AHR) as the likely factor that regulates the differentially expressed genes. One can hypothesize that auxin activates AHR to regulate a set of genes. Test this hypothesis by depleting AHR, treating with auxin, and performing RNA-seq under the four conditions: -/+ AHR depletion and -/+ auxin treatment.

I use this as an example because we were using the auxin-inducible degron (AID) system and performed this control. We developed the hypothesis that auxin is activating the AHR and that this is a previously unappreciated deficiency of the AID system (Sathyan *et al.*, 2019). This brings us an exciting feature of RNA-seq: **we can develop hypotheses that were inconceivable prior to analyzing the data!**

2 Retrieving and processing raw RNA-seq reads

2.1 Installing `XCode`, `wget`, `homebrew`, and `SRA toolkit`

2.1.1 Installing `XCode` toolkit

Some of you will have personal Mac computers that require additional software installation. These code chunks will be contained within light blue boxes.

All Mac users should first install Xcode from the Mac App Store.

2.1.2 Installing `homebrew`

`Homebrew` is a convenient package manager and installer for macOS. Common Linux packages and some bioinformatics tools can be easily installed with `homebrew`. The website provides this installation code: `https://docs.brew.sh/Installation`. I do not recommend copying and pasting code from a PDF, because formatting and characters can be erroneously interpreted dependign on your PDF reader/viewer. For instance, I cannot copy most code from Preview, but Adobe Acrobat works reasonable well. Note that these code chunks should be run successively, because often the output of one chunk is the input for a subsequent chunk.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

2.1.3 Installing `wget`

The `wget` software package is used to retrieve files from the internet.

```
brew install wget
```

2.1.4 Installing `SRA toolkit`

`fasterq-dump` is an SRA tool (Kodama *et al.*, 2011) that extracts data directly from the Sequence Read Archive, where most high throughput sequencing data is deposited.

Software installation:

`sra-tools`: `https://github.com/ncbi/sra-tools`
Downloading the pre-compiled binaries that are compatible with your computer architecture is easiest (`https://github.com/ncbi/sra-tools/wiki/01.-Downloading-SRA-Toolkit`). Below we download the binaries for a Mac, move them to a directory that is present on most Mac computers, then update the computer's $PATH so that Terminal will look in the proper place for the software.

```
wget http://ftp-trace.ncbi.nlm.nih.gov/sra/sdk/2.9.6-1/sratoolkit.2.9.6-1-mac64.tar.gz
tar -xzf sratoolkit.2.9.6-1-mac64.tar.gz
mv sratoolkit.2.9.6-1-mac64 /usr/local/bin/
export PATH="$PATH:/usr/local/bin/sratoolkit.2.9.6-1-mac64/bin"
```

2.1.5 Updating your $PATH to find software

I add the PATH directly to my `.bash_profile` so that every time I open a new login shell the $PATH variable updates. Alternatively, you could modify your `.bashrc` file, which runs on every interactive shell launch.

```
emacs ~/.bash_profile
```

Add the following line to the .bash_profile file:

```
export PATH="$PATH:/usr/local/bin/sratoolkit.2.9.6-1-mac64/bin"
```

Save and close `emacs` using the following keystokes in sequence while keeping the previous keys engaged with each subsequent key (note that the "+" signs do not represent a key that should be pressed):
control + x + s
control + x + c

2.2 Finding relevant data and experiment details using ENCODE

The ENCODE consortium generated thousands of molecular genomics data sets that are publicly available (`https://www.encodedcc.org`). The data below are from A549, which is a adenocarcinomic human alveolar basal epithelial cell line. The cells were treated with dexamethasone for various time points. All the available experimental information can be found on the ENCODE website: `https://www.encodeproject.org/treatment-time-series/ENCSR897XFT/`. Below, I download the *mRNA-seq on A549 cell line treated with 100 nM dexamethasone for 0 hours AND 0.5 hours* for comparison. I encourage you to follow my workflow for the 0 minutes vs. 30 minutes comparison. For you adventurous types, I encourage you to choose an additional time point to analyze and compare against time zero.

2.3 Acquiring data from GEO

Sequencing files will come off the machine in FASTQ format, but they are stored as SRA (short read archive) files in data bases such as GEO (Gene Expression Omnibus) (`http://www.ncbi.nlm.nih.gov/geo/`).

All the files should be named with the cell type, conditions, description, the replicate number. No spaces. This makes it so that downstream analyses can be automated.

2.3.1 Processing SRA files

Retrieve and convert the binary SRA files to FASTQ files with the `sra-tools` kit; `fasterq-dump` automatically detects whether the data is single end or paired end. Paired end data will result in two output files and these data are paired end.

Back in Section 2.1.2 I recommended that you do not copy and paste from a PDF, but you may want the raw working shell and `R` scripts, so I uploaded them to `GitHub` and I will include the links. The following two code chanks are included in a single `bash` script: `https://raw.githubusercontent.com/stefbekir/bioc8145/master/Guertin_week4/section2.3.1.sh`. This `bash` script can be accessed directly with `wget` and all URLs that begine with *raw.githubusercontent.com* are the raw code.

```
#no dex rep 1 (as defined by GEO description)
fasterq-dump SRR5093037
#no dex rep2
fasterq-dump SRR5093036
#no dex rep3
fasterq-dump SRR5093038
#no dex rep4
fasterq-dump SRR5093039

#30min 100nM dex rep 1
fasterq-dump SRR5093268
#30min 100nM dex rep 2
fasterq-dump SRR5093265
#30min 100nM dex rep 3
```

```
fasterq-dump SRR5093267
#30min 100nM dex rep 4
fasterq-dump SRR5093266
```

Use the `mv` command to name the files something descriptive: cell line, treatment, replicate number, and paired end status.

```
mv SRR5093037_1.fastq A549_no_treatment_rep1_PE1.fastq
mv SRR5093036_1.fastq A549_no_treatment_rep2_PE1.fastq
mv SRR5093038_1.fastq A549_no_treatment_rep3_PE1.fastq
mv SRR5093039_1.fastq A549_no_treatment_rep4_PE1.fastq

mv SRR5093268_1.fastq A549_30min100nMdex_rep1_PE1.fastq
mv SRR5093265_1.fastq A549_30min100nMdex_rep2_PE1.fastq
mv SRR5093267_1.fastq A549_30min100nMdex_rep3_PE1.fastq
mv SRR5093266_1.fastq A549_30min100nMdex_rep4_PE1.fastq

mv SRR5093037_2.fastq A549_no_treatment_rep1_PE2.fastq
mv SRR5093036_2.fastq A549_no_treatment_rep2_PE2.fastq
mv SRR5093038_2.fastq A549_no_treatment_rep3_PE2.fastq
mv SRR5093039_2.fastq A549_no_treatment_rep4_PE2.fastq

mv SRR5093268_2.fastq A549_30min100nMdex_rep1_PE2.fastq
mv SRR5093265_2.fastq A549_30min100nMdex_rep2_PE2.fastq
mv SRR5093267_2.fastq A549_30min100nMdex_rep3_PE2.fastq
mv SRR5093266_2.fastq A549_30min100nMdex_rep4_PE2.fastq


gzip *fastq
```

The `gzcat` command uncompresses the gzipped file and writes the contents to stdout, which is piped to either `head` or `wc -l` in order to view the first ten lines and count the number of lines.

Note that each sequencing read has four lines associated with it:
Line 1 begins with a '@' and is followed by a unique identifier
Line 2 is the raw nucleotide sequence.
Line 3 begins with a '+' and is optionally followed by the same sequence identifier.
Line 4 encodes the quality scores for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence. This line is a measure of how confident the base is called correctly.

```
gzcat A549_no_treatment_rep1_PE1.fastq.gz | head
gzcat A549_no_treatment_rep1_PE1.fastq.gz | wc -l
```

3 Aligning with HISAT2

3.1 Install HISAT2, samtools, and bedtools

Commonly used software has extensive documentation and installation instructions, so if you run into challenges installing software, please refer to these resources.

3.1.1 Install hisat2

HISAT2 is commonly used alignment software for RNA-seq data (Kim *et al.*, 2015).

Software installation:

hisat2: https://daehwankimlab.github.io/hisat2/

```
wget https://cloud.biohpc.swmed.edu/index.php/s/hisat2-220-OSX_x86_64/download
mv download hisat2-2.2.0.zip
unzip hisat2-2.2.0.zip
mv hisat2-2.2.0 /usr/local/bin

export PATH="$PATH:/usr/local/bin/hisat2-2.2.0"
```

Refer to Section 2.1.5 to add this directory to your ∼/.bash_profile.

3.1.2 Install samtools

Samtools is an extensive suite of tools that allow you to interface with high-troughput sequencing data files in many ways.

Software installation:

samtools: http://www.htslib.org/download/

```
wget https://github.com/samtools/samtools/releases/download/1.10/samtools-1.10.tar.bz2
tar -xf samtools-1.10.tar.bz2
cd samtools-1.10
mkdir /usr/local/bin/samtools-1.10
./configure --prefix=/usr/local/bin/samtools-1.10
make
make install

export PATH=/usr/local/bin/samtools-1.10/bin:$PATH
```

Refer to Section 2.1.5 to add this directory to your ∼/.bash_profile.

3.1.3 Install bedtools

Bedtools utilities provide efficient and well-documented methods to perform *arithmetic* of genomic coordinate files. Originally developed at UVA by Aaron Quinlan and Ira Hall, this is the go to set of tools that most genomicists use to operate on genomic intervals.

Software installations:

bedtools: https://bedtools.readthedocs.io/en/latest/

```
brew install bedtools
```

3.2 Acquiring and processing the genome and annotation files

3.2.1 Retrieve the *hg38.fa* genome file

The first task is to retrieve the relevant genome from UCSC (Karolchik *et al.*, 2014): (`http://hgdownload.soe.ucsc.edu/downloads.html`). We want the *hg38.fa.gz* file. This is a zipped fasta file of the entire human genome.

```
wget http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz

gunzip hg38.fa.gz
```

Look at the format of a FASTA (pronounced with a hard A) file. Most of the sequence is either A,C,T, or G–but toward the telomeres there are many N's, due to the fact that assembling repetitive DNA is non-trivial.

```
head hg38.fa
```

3.2.2 Build a set of genome index files with HISAT2

Next we build the genome index with `hisat2` (Kim *et al.*, 2015). This task is performed once per genome.

```
hisat2-build hg38.fa hg38
```

3.3 Aquiring and building gene annotations from GENCODE

We built the indexed genome for `hisat2`, but now we will need to acquire and index the gene annotation GTF file. We will use GENCODE (`https://www.gencodegenes.org/human/`).

```
wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_33/gencode.v33.annotation.gtf.gz
gunzip gencode.v33.annotation.gtf.gz
python3 /usr/local/bin/hisat2-2.2.0/hisat2_extract_splice_sites.py gencode.v33.annotation.gtf > gencode.v33.splice.txt
```

3.4 Aligning with `hisat2`

Aligning using multiple processors in parallel speeds up the alignment. How many processors does your Mac have? I have 4 processors and I will leave one processor free by specifying the `-p` option as 3.

```
sysctl hw.ncpu | awk '{print $2}'


## 4
```

To align all the files in the same manner we can use shell scripting to loop over each file. The files are names descriptively, so we can extract the descriptive text from the input file (the `$name` variable below) and use this to label the output files. The input files include the indexed hg38 genome (`-x`), the splice sites files, and each paired end FASTQ file. These files are used in the alignment and piped (`|`) to `samtools` to generate a BAM file (`hisat2` output is SAM format), and lastly this file is piped to a sorting function. SAM is a human-readable alignment file and BAM files are the binary counterpart, which stores data more efficiently, but is not human-readable text. Note that the `\` after `gencode.v33.splice.txt` (and in other contexts throughout these shell chunks) is used to indicate that the line of code is extended onto the next line in the chunk.

```
for i in *PE1.fastq.gz
do
    name=$(echo $i | awk -F"_PE." '{print $1}')
    echo $name
    hisat2 -p 3 -x hg38 --rna-strandness FR --known-splicesite-infile gencode.v33.splice.txt \
        -1 $i -2 ${name}_PE2.fastq.gz | samtools view -bS - | samtools sort -n - -o $name.sorted.bam
done
```

This loop hard codes the `hg38` prefix for the indexed files, the splice site file, and the number of processors. However, these will be changed depending upon the experiment, so these can be initialized and passed to the loop for more general usage. In the chunk below you can change the *idx*, *spliceFile*, and *processors* names in the header and these are passed to the loop. However, the loop still asssumes that the data is paired end and named with the suffixes *_PE1.fastq.gz* and *_PE2.fastq.gz*. This chunk is uploaded as a `bash` script: `https://raw.githubusercontent.com/stefbekir/bioc8145/master/Guertin_week4/section3.4.sh`.

```
idx=hg38
spliceFile=gencode.v33.splice.txt
processors=3

for i in *PE1.fastq.gz
do
    name=$(echo $i | awk -F"_PE." '{print $1}')
    echo $name
    hisat2 -p ${processors} -x ${idx} --rna-strandness FR --known-splicesite-infile ${spliceFile} \
        -1 ${i} -2 ${name}_PE2.fastq.gz | samtools view -bS - | samtools sort -n - -o $name.sorted.bam
done
```

3.5 Removing PCR duplicates

We can identify reads that start and end at the exact same genomic coordinates as presumptive PCR duplicates. This employs `samtools` once again and you will need to refer to `fixmate`, `sort`, and `markdup` within the `samtools` user manual to investigate the function of the pipe below. How does this `sort` command functionally differ than Section 3.4?

This code chunk is provided at: `https://github.com/stefbekir/bioc8145/blob/master/Guertin_week4/section3.5.sh`. Note that this is not the *raw.githubusercontent.com* file, but you should be able to compare with the previous links to find the raw file.

```
for i in *.sorted.bam
do
      name=$(echo $i | awk -F".sorted.bam" '{print $1}')
      echo $name
      samtools fixmate -m $i - | samtools sort - | samtools markdup -rs - $name.rmPCRdup.bam
done
```

3.6 Converting files to upload into UCSC

You can also upload RNA-seq data to view in UCSC. You will use a postion-sorted BAM file. Then use the `bedtools` function `genomeCoverageBed`. The zipped *bedGraph* can be uploaded to UCSC.
The *BAM* file is sorted by position so the input to `genomeCoverageBed` is properly formatted. I am fastidious about how I want my data visualized in a browser, so the code chunk is a bit complicated. This code uses both paired end reads to visualize the data and generates a plus and minus track that are colored red and blue. I did leave out some code that normalizes each track, because it will further complicate the loop. But note that these are not scaled by sequencing depth. I typically use the `size.factors` calculated by `DESeq2` (not yet covered) and the `-scale` option in `genomeCoverageBed` to scale these browser tracks.
Code: `https://github.com/stefbekir/bioc8145/blob/master/Guertin_week4/section3.6.sh`

```
for i in *.rmPCRdup.bam
do
    name=$(echo $i |  awk -F".rmPCRdup.bam" '{print $1}')
    echo $name
```

```
    samtools view -b -f 0x40 $i | samtools sort - -o ${name}.PE1.sorted
    samtools view -b -f 0x80 $i | samtools sort - -o ${name}.PE2.sorted
    genomeCoverageBed -bg -strand - -split -ibam ${name}.PE1.sorted | sortBed -i - > $name.pe1.minus.bedGraph
    genomeCoverageBed -bg -strand + -split -ibam ${name}.PE1.sorted | sortBed -i - > $name.pe1.plus.bedGraph
    genomeCoverageBed -bg -strand + -split -ibam ${name}.PE2.sorted | sortBed -i - > $name.pe2.minus.bedGraph
    genomeCoverageBed -bg -strand - -split -ibam ${name}.PE2.sorted | sortBed -i - > $name.pe2.plus.bedGraph
    cat $name.pe1.plus.bedGraph $name.pe2.plus.bedGraph | sortBed -i - > $name.merged.plus.bedGraph
    touch temp.txt
    echo "track type=bedGraph name=$name.plus color=255,0,0 visibility=2" >> temp.txt
    cat temp.txt $name.merged.plus.bedGraph > $name.plus.bedGraph
    rm temp.txt
    cat $name.pe1.minus.bedGraph $name.pe2.minus.bedGraph | sortBed -i - > $name.merged.minus.bedGraph
    touch temp.txt
    echo "track type=bedGraph name=$name.minus color=0,0,255 visibility=2" >> temp.txt
    cat temp.txt $name.merged.minus.bedGraph > $name.minus.bedGraph
    rm temp.txt
    rm *.merged.*us.bedGraph
    rm *.pe*.*us.bedGraph
    gzip $name.minus.bedGraph
    gzip $name.plus.bedGraph
done
```

3.7 Visualize peaks and bedGraph files with the UCSC browser

I prefer the UCSC browser to visual my data and for exporting publication-quality figures (`https://genome.ucsc.edu`) (Karolchik *et al.*, 2014). Make sure you have the correct assembly, we are using the latest: *hg38*: `https://genome.ucsc.edu/cgi-bin/hgTracks?db=hg38`.

After clicking *manage custom tracks* (Figure 1), use *add custom tracks* to upload the **.bedGraph.gz* files. Upload each file and customize the settings. You probably want to register and save sessions if you have your own data. Then you can save sessions and you will not need to upload the data more than once. I previously uploaded the data and saved a session: `https://genome.ucsc.edu/s/Mike%20Guertin/hg38_class_example`

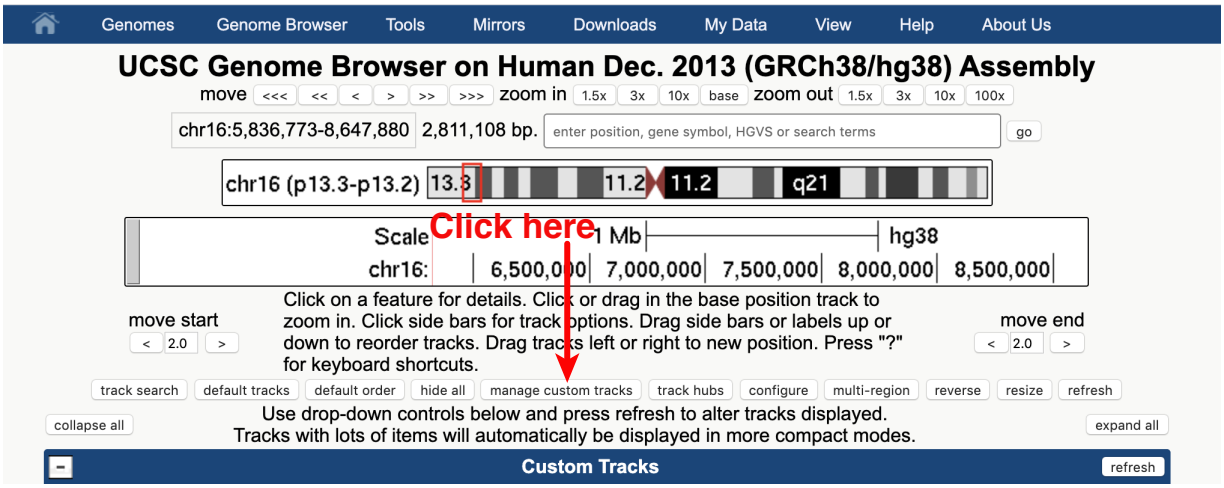

Figure 1: **UCSC genome browser.** There are many types of browsers one can use to visualize data, I like UCSC because it support track hubs, sharing, and it does not run on the local computer.

Looking at your data in a browser can save you hours, days, or longer in troubleshooting.

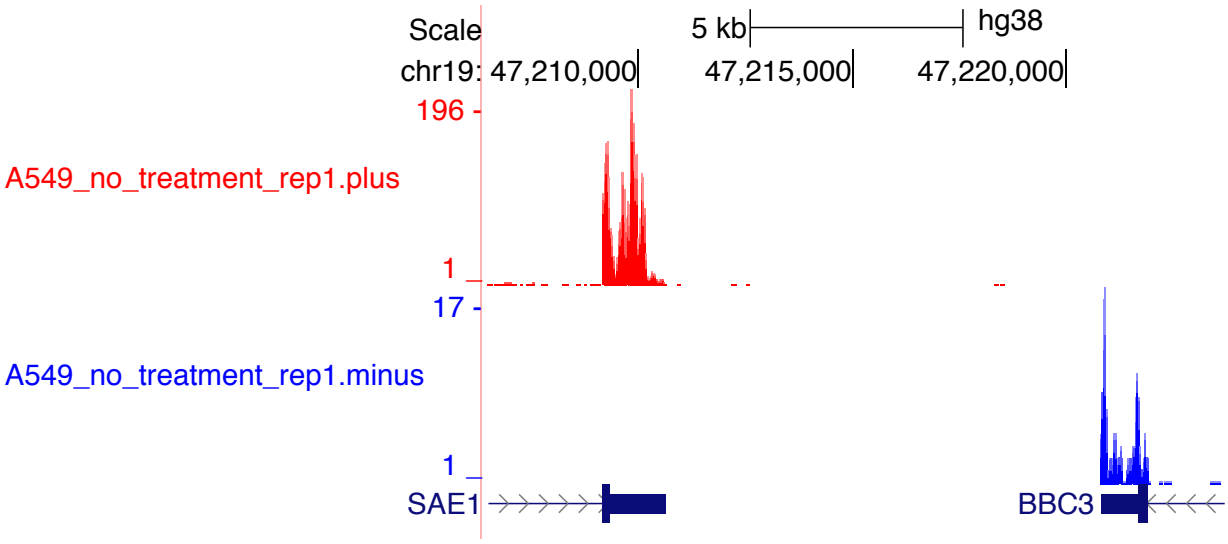

Figure 2: **Visualize your data in a browser.** Exploring your data in a browser can be the best way to troubleshoot and confirm or refute assumptions. For instance, looking at these tracks we can clearly see that the plus and minus tracks fall within the correct gene annotations. If we observed comparable densities on both the plus and minus exons, then it would be clear that the data was not strand-specific. If we saw that stranded tracks were inversed (i.e. the minus track overlaps the plus gene annotation), then we would have to change the `-strand` option in `genomeCoverageBed` in section 3.6. In fact, I could not find the method that ENCODE used to generate these libraries, so I had to empirically determine whether the reverse complement template or coding strand was sequenced.

4 Counting reads in genes

4.1 Installing HTSeq

```
pip3 install HTseq
```

4.2 Count reads read in gene annotations

We use `htseq-count` to bin reads into genomic features (i.e. genes). The options below include `-r` which tells the program the bam file is sorted by position, `-f` which tells the program the input is a BAM file, and `--stranded` tells the program the strand information of the file. Notice the .gtf file is included. The output of this command is a .txt file that contains the gene counts for that sample. We will get one .txt file per sample. We confirmed that the data is stranded, but even if I know everything about the experiment, is not intuitive for me to know whether to specify `yes` or `reverse` for the `--stranded` argument.

To ensure that I use the correct option, I run both options and observe the results. Whatever option has more reads in the vast majority of the genes is correct. I appreciate that this is not clean, but these empirical checks are often necessary to ensure the data is properly processed. The biggest mistakes that I have observed in genomics could have been avoided if the bioinformatician looked at the output of analyses and confirmed that the result is expected or, at minimum, reasonable.

The output counts file has a footer (see the `tail` output) that we want to ignore in the counting. The `grep -v` command prints all the lines without `__` to *stdout*, and we use `awk` to count the second column. I have been implementing `awk` code for ∼15 years and I use *the Google* if I am not copying and pasting previously *Googled* commands. It's no *Ask Jeeves*, *Yahoo!*, or *Bing*, but I cannot overemphasize how useful *the Google* can be.

```
name=A549_30min100nMdex_rep1
htseq-count -r pos -f bam --stranded=yes $i gencode.v33.annotation.gtf > $name.gene.counts.yes.txt
htseq-count -r pos -f bam --stranded=reverse $i gencode.v33.annotation.gtf > $name.gene.counts.reverse.txt

tail $name.gene.counts.yes.txt
tail $name.gene.counts.reverse.txt

echo yes
grep -v __ $name.gene.counts.yes.txt | awk '{sum+=$2} END{print "sum=",sum}'
echo reverse
grep -v __ $name.gene.counts.reverse.txt | awk '{sum+=$2} END{print "sum=",sum}'

for i in *rmPCRdup.bam
do
    name=$(echo $i | awk -F".rmPCRdup." '{print $1}')
    echo $name
    htseq-count -r pos -f bam --stranded=yes $i gencode.v33.annotation.gtf > $name.gene.counts.txt
done
```

Use `bash` commands to take a look at the **.gene.counts.txt* files. How many lines does each file contain? Are the first columns identical between files? Note the previously mentioned footer. What do the *ENSG000** names represent and where are they found in the *gtf* annotation file?

4.3 Importing counts data into R

We will use `R` to further analyze and plot the data. You must preinstall several libraries including *lattice*, *DESeq2*, *dplyr*, *fgsea*, *ggplot2* and *gage* (optional). I am not going into details for installing packages in `R`, but I am happy to try and help if you attempt to install these on your computer and fail. The first few lines of the chunk load in the libraries, and then I have you load a set of functions from my GitHub repository. These are not all used in the next code chunk, but they will be used in subsequent chunks. After defining the directory of interest, which contains my **.gene.counts.txt* files, I wrote a

simple loop that will run through each **.gene.counts.txt* file and generate a data frame that has all the gene identifiers as row names, the exprerimental conditions as column names, and the corresponding counts at each column/row intersect. You can deconstruct the code to determine what is happening and I included comments. This *for loop* is something that you may want to call many times if you start analyzing a lot of RNA-seq data, so you can make this routine into a function.

```
library(lattice)
library(DESeq2)
library(dplyr)
library(fgsea)
library(ggplot2)
library(gage)
library(limma)

source('https://raw.githubusercontent.com/mjg54/znf143_pro_seq_analysis/master/docs/ZNF143_functions.R')


path.dir='~/Desktop/course/'
file.suffix='.gene.counts.txt'

#keep track of the experiment names in this list so I can go back and label the columns
vec.names = c()
#we only want generate a data frame label the rows for the first file we loop through
count = 0
#here we will loop through our suffix-defined files in the defined directory
for (txt in Sys.glob(file.path(path.dir, paste0("*", file.suffix)))) {
      count = count + 1
#this complicated code simply splits strings to extract the experiment name from the file
      experiment.name = strsplit(strsplit(txt,
            "/")[[1]][length(strsplit(txt, "/")[[1]])], file.suffix)[[1]][1]
      print(experiment.name)
#only do this for the first file, since count is greater than 1 with every other file
      if (count == 1) {
#generate a data frame with the gene row names and no columns
          all.counts = data.frame(row.names = read.table(txt)[,1])
          }
#add the experiment name to a growing list of column names
      vec.names = c(vec.names, experiment.name)
#for each file (including the first file) add a column with the counts information
      all.counts = cbind(all.counts, data.frame(read.table(txt)[,2]))
}

#the last 5 lines are not gene counts, so delete them
all.counts = all.counts[1:(nrow(all.counts) - 5),]
#name the columns
colnames(all.counts) = vec.names

head(all.counts)

dim(all.counts)
```

4.3.1 Defininging a function in R

```
#remove the objects path.dir and file.suffix
rm(path.dir, file.suffix)

#these objects no longer exist
print(path.dir)
print(file.suffix)
```

```
#define a function
htseq.to.counts.df <- function(path.dir, file.suffix = '.gene.counts.txt') {
    vec.names = c()
    count = 0
    for (txt in Sys.glob(file.path(path.dir, paste0("*", file.suffix)))) {
        count = count + 1
        experiment.name = strsplit(strsplit(txt,
                "/")[[1]][length(strsplit(txt, "/")[[1]])], file.suffix)[[1]][1]
        print(experiment.name)
        if (count == 1) {
            all.counts = data.frame(row.names = read.table(txt)[,1])
        }
        vec.names = c(vec.names, experiment.name)
        all.counts = cbind(all.counts, data.frame(read.table(txt)[,2]))
    }
    all.counts = all.counts[1:(nrow(all.counts) - 5),]
    colnames(all.counts) = vec.names
    return(all.counts)
}

all.counts.from.func = htseq.to.counts.df(path.dir = '~/Desktop/course/',
                                file.suffix = '.gene.counts.txt')

#are the results identical?
identical(all.counts.from.func, all.counts)
```

You can change the inputs to a function depending upon how you labeled the suffix of the counts files and where they are located on your computer.

4.4 Convert the Ensembl Gene (ENSG) name into a gene symbol

I prefer to have the gene names be more interpretable, so I convert the Ensembl Gene (ENSG) ID to the gene symbol.

```
gencode.all = read.table('gencode.v33.annotation.gtf', sep='\t', header =F)

gencode.gene.names = data.frame(sapply(strsplit(sapply(strsplit(
    as.character(gencode.all[,9]),'gene_name '), "[", 2), ";"), "[", 1))

gencode.id.names = data.frame(sapply(strsplit(sapply(strsplit(
    as.character(gencode.all[,9]),'gene_id '), "[", 2), ";"), "[", 1))

gencode.key = cbind(gencode.gene.names, gencode.id.names)

gencode.key = gencode.key[!duplicated(gencode.key[,2]),]

rownames(gencode.key) = gencode.key[,2]

colnames(gencode.key) = c('gene', 'id')

#optional to save intermediate files
#save(gencode.key, file = 'gencode.key.Rdata')

#why do I check the dimensions?
dim(all.counts)
dim(gencode.key)

merged.counts.pre = merge(all.counts, gencode.key, by="row.names", all.x=F)
```

```
merged.counts = merged.counts.pre[, c(2:(ncol(merged.counts.pre) - 2))]

rownames(merged.counts) = make.names(merged.counts.pre$gene, unique=TRUE)

#Homework:
#Make this code chunk into a function.
```

4.5 Differential expression with DESeq2

DESeq (Anders and Huber, 2010; Love *et al.*, 2014) is an R library with the tools for differential gene expression analysis based on the negative binomial distribution (http://www.bioconductor.org/packages/release/bioc/html/DESeq2.html).

Refer to the DESeq2 vignette (http://bioconductor.org/packages/release/bioc/vignettes/DESeq2/inst/doc/DESeq2.pdf) and publication (Love *et al.*, 2014) for a detailed explanation of the processes and statistics.

The workflow below follows the DESeq2 vignette.

```
#set the no treatment as the reference level (i.e. what happens upon treatment)
sample.conditions = factor(sapply(strsplit(as.character(colnames(merged.counts)),
              '_rep'), '[', 1), levels=c("A549_no_treatment","A549_30min100nMdex"))

#Note:
# Although not covered herein,
# I typically use these size factors to normalize bedGraph files.
# After I normalize each file (-scale in genomeCoverageBed),
# I convert to bigWig using bedGraphToBigWig
# and use bigWigMerge to make a signal representative track per condition.
# estimateSizeFactorsForMatrix(merged.counts)

deseq.counts.table = DESeqDataSetFromMatrix(merged.counts,
                as.data.frame(sample.conditions), ~ sample.conditions)

dds = DESeq(deseq.counts.table)

normalized.counts.rna = counts(dds, normalized=TRUE)

rld = rlog(dds, blind=TRUE)

pca.plot = plotPCA(rld, intgroup="sample.conditions", returnData=TRUE)
plotPCAlattice(pca.plot, file = 'PCA_A549_dex_lattice.pdf')
```

Principal components analysis (PCA) is a statistical method to reduce highly complex data into a limited number of dimensions that we can easily visualize. *A priori*, we expect that the sample separate based on conditions. The samples do not separate in this reasonable and expected manner. It is your responsibility to figure out the biological or technical feature that is driving the principal components. Here it is clear that replicate 4 that is dominating PC1. We can hypothesize that something unique to replicate 4 is underlying PC1, which accounts for over half of the varaince.

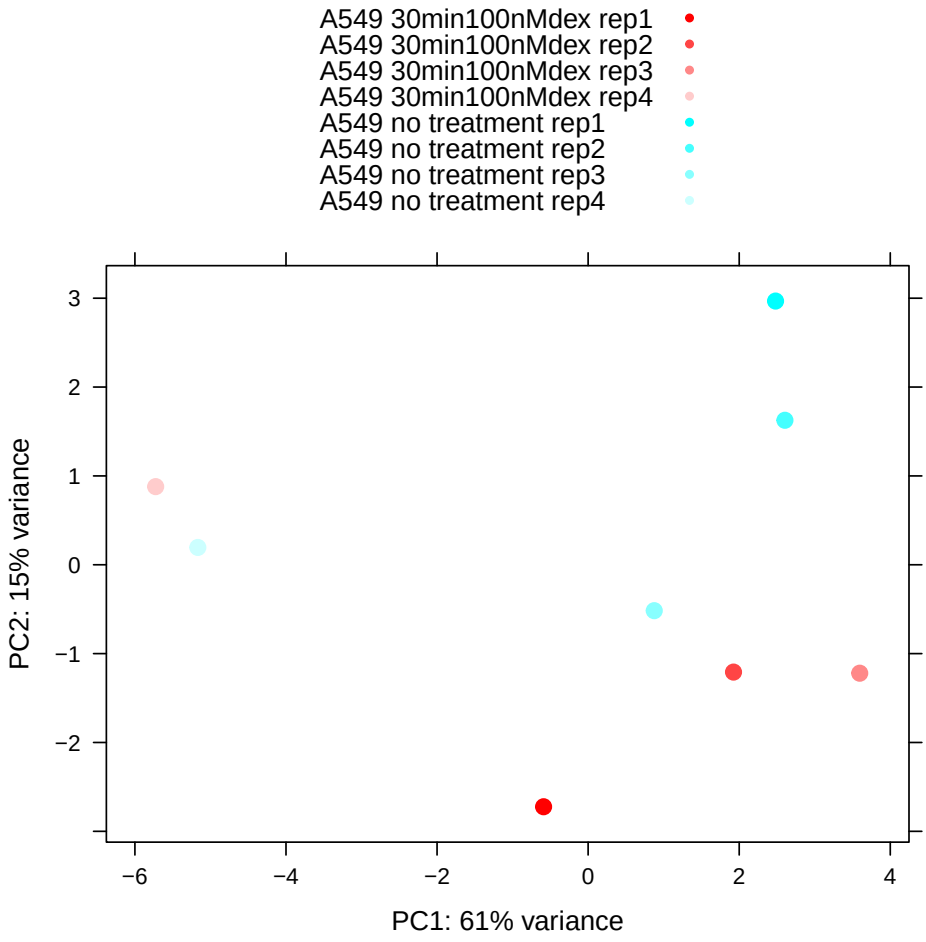

Figure 3: **PCA of RNA-seq analysis.** This shows that replicate 4 is dominating principal component 1, which is unexpected because the biology of interest is the dexamethasone treatment.

Visualize the DESeq2 dataframe with *results()*. I added a step to include a categorical column for different gene classes. The categories are defined by the multiple testing corrected p-value and and a $\log_2$ fold change. How many genes are considered activated at the specified FDR (0.1)?

```
DE.results.all.4reps = results(dds)

head(DE.results.all.4reps)

DE.results.all.4reps.lattice =
    categorize.deseq.df(DE.results.all.4reps,
                        fdr = 0.1, log2fold = 0.0, treat = 'Dexamethasone')

head(DE.results.all.4reps.lattice)

activated.all = DE.results.all.4reps.lattice[DE.results.all.4reps.lattice$response ==
                                             'Dexamethasone Activated',]

print(activated.all)
```

Take a look at the *DE.results.all.4reps.lattice* data frame and you will notice that no genes are differentially expressed. This can occur if the variance between replicates exceeds the variance between conditions. If we hypothesize that replicate 4 is driving the variation between replicates, then we would still expect that comparing the $\log_2$ fold change and maybe the p-value to be meaningful. I would expect them to reflect real biology because the change when one adds dexamethasone may be consistent, even though the starting expression levels are clearly different for replicate 4. These differences in the basal expression will account for a lot of the variance and the differences will be common to both the untreated and Dexamethasone treated replicate 4, which would account for a high degree of variance within each experimental condition. The corrected p-values may not be significant, but the genes that change upon dexamethasone treatment should have a lower p-value.

4.6 Gene Set Enrichment Analysis

So we can use gene set enrichment analysis of a ranked gene set to determine what genes are enriched in our ranked set (Subramanian *et al.*, 2005). Recall that no genes are significantly activated or repressed, so we cannot use an overlap statistical analysis (like Fisher's Exact). While categorically thresholding activated and repressed genes is useful, the threshold are biologically artbitrary and the quantitative nature of fold change is lost. GSEA can employs rank information without the need for thresholding. Search around the Broad's GSEA site to learn more about the process. The following is a list of defined gene sets: `https://data.broadinstitute.org/gsea-msigdb/msigdb/release/`. Note the different collections to which you can compare in Figure 5 (`https://www.gsea-msigdb.org/gsea/msigdb/index.jsp`).

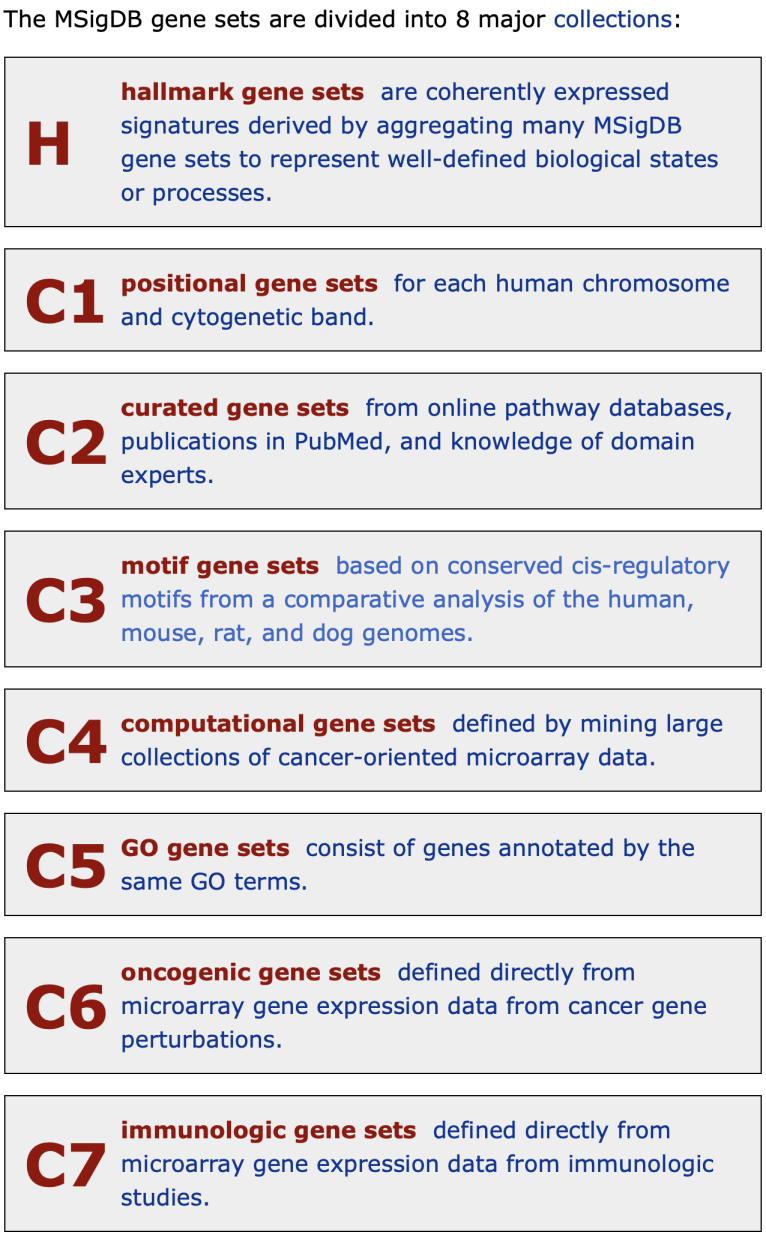

Figure 4: **Gene Set Enrichment Gene Collections.** These gene set enrichment collections are lifted directly from: `https://www.gsea-msigdb.org/gsea/msigdb/index.jsp`.

The first step is to order a gene list. One can justify ordering by p-value of fold change. Using either, I kept getting a warning: *There are ties in the preranked stats (XX% of the list). The order of those tied genes will be arbitrary, which may produce unexpected results.* Therefore, I multiplied the fold change by the -$\log_{10}$ p-value and used this to rank order the genes. You are encouraged to use other ranking methods to determine if any conclusions are robust to arbitrary ranking criteria. We will use `fgsea` in `R` for GSEA: `https://bioconductor.org/packages/release/bioc/html/fgsea.html`.

Analytical positive (and negative) controls can be as important as experimental controls. I use these as coherence check, especially when I am implementing some black box analysis tools that I do not fully understand. I manually typed table 2 from a previous Dexamethasone-treated RNA-seq publication (the first authorfrom this publication is the PI who deposited these data: Tim Reddy) (Reddy *et al.*, 2009).

```
#you need to order the gene list:
gene.list= 2^DE.results.all.4reps.lattice$log2FoldChange *
                 -log(DE.results.all.4reps.lattice$pvalue, base = 10)

names(gene.list) = row.names(DE.results.all.4reps.lattice)

#rank order
gene.list = sort(gene.list, decreasing = TRUE)

#remove NA entries
gene.list = gene.list[!is.na(gene.list)]

#remove duplicated genes
gene.list = gene.list[!duplicated(names(gene.list))]


head(gene.list)


# what file do you want to use?
#the more categories you have, the higher the multiple testing corrected test statistic

msigdb.url = 'https://data.broadinstitute.org/gsea-msigdb/msigdb/release/7.0/'

#GMT file options
#GO_file= paste0(msigdb.url, "h.all.v7.0.symbols.gmt")
#GO_file= paste0(msigdb.url, 'msigdb.v7.0.symbols.gmt')
#GO_file= paste0(msigdb.url, 'c5.all.v7.0.symbols.gmt')
#GO_file= paste0(msigdb.url, 'c2.all.v7.0.symbols.gmt')



#positive control
#check the content (second column) of the file to see where I manually copied it from
system('wget https://data.broadinstitute.org/gsea-msigdb/msigdb/release/7.0/h.all.v7.0.symbols.gmt')
system('wget https://raw.githubusercontent.com/stefbekir/bioc8145/master/Guertin_week4/GO_positive_ctrl.gmt')
system('cat h.all.v7.0.symbols.gmt GO_positive_ctrl.gmt > h.all.v7.0.symbols.plusGRresp.gmt')

GO_file = 'h.all.v7.0.symbols.plusGRresp.gmt'
myGO = gmtPathways(GO_file)


fgRes = as.data.frame(fgsea(pathways = myGO,
                            stats = gene.list,
                            minSize=5,
                            maxSize=600,
                            nperm=10000))

fgRes[fgRes$padj < 0.1,][order(fgRes$NES[fgRes$padj < 0.1], decreasing =TRUE),]
```

```
plotEnrichment(myGO[["GO_DEX_ACTIVATED_GENES_WITH_GR_IN_A549"]],
               gene.list) + labs(title="GR-bound Dex. Activated Genes in A549 (Reddy, et. al., 2009)")
```

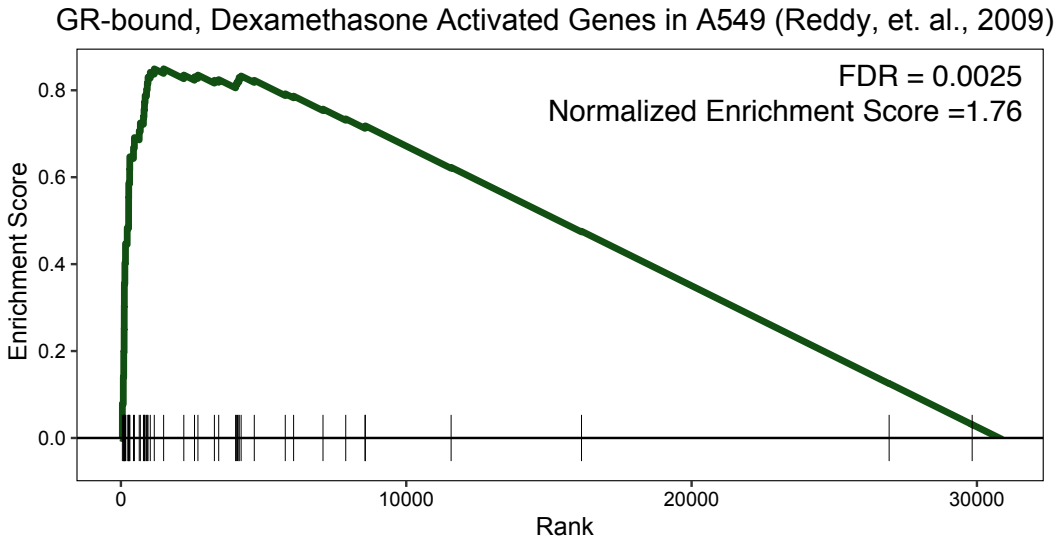

Figure 5: **Gene Set Enrichment plot.** The enrichment score is plotted agains the genes ranked by the -$\log_{10}$ p-value multiplied by the fold change. Each gene in the class (in this case the 50 positive control genes) is represented by a black tick mark along the x-axis.

5 Unexpected results

I chose these data blindly, my criteria were stranded, paired-end, multiple replicates, publicly available, and two conditions separated by a short time interval. I was interested in these data because my group typically uses PRO-seq to measure nascent RNA and immediate changes in transcription. I was skeptical that we would see many genes changing expression and that the only genes that we would be able to detect activation would be short genes and those that were relatively lowly expressed during no-dex conditions. I hypothesized this because of the 30 minute treatment and because RNA-seq measures stable RNA. The two main barriers to look at changing expression are 1) the basal levels of RNA and 2) the gene length. Gene length is important because it take a full minute for RNA polymerase to travel between 2-3 kilobases, so no gene that is 90 kilobases or more should be detectably changed at 30 minutes.

Now I realize that by not cherry-picking a polished and clean data set with straightforward results, I have the opportunity to teach you about the most challenging, time consuming, and rewarding component of bioinformatics: exploratory analyses to try to understand the unexpected. I believe that molecular biologists, or those with a deep understanding of biology, are at a distinct advantage in this realm.

I will take you step-by-step exactly what I did to try to better understand these data and determine whether I believe dropping replicate 4 is justified.

5.1 Modeling batch effect of the replicates

Refer to the DESeq2 vignette for modeling batch effect: `http://bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html`. Below we model the batch effect of each replicate, then we model two batches, operating under the assumption that an unknown variable in replicate 4 should be modeled. What *positive control* criteria can I use to evaluate whether the batch effect modeling is working? No treatment and Dexamethasone should separate by PCA and we should observe a few canonical dexamethasone activated genes as activated by GSEA.

```
# I am going to make the workflow below into a function, because I will be carrying
# out the same analyses with slight variations each time.

deseq.gsea.workflow <- function(counts.matrix, batch = FALSE, sample.conditions,
                          GO.file = "h.all.v7.0.symbols.gmt",
                          name.analysis = 'test', treatment = 'Dexamethasone',
                          fdr = 0.1, log2fold = 0.0) {
#not the best if statement because it gives me a warning...
    if (batch == FALSE) {
        deseq.counts.table = DESeqDataSetFromMatrix(counts.matrix,
                    as.data.frame(sample.conditions), ~ sample.conditions)
        dds = DESeq(deseq.counts.table)
        rld = rlog(dds, blind=TRUE)
        pca.plot = plotPCA(rld, intgroup="sample.conditions", returnData=TRUE)
        plotPCAlattice(pca.plot, file = paste0("PCA", name.analysis,'.pdf'))
        DE.results = results(dds)
        DE.results.lattice =
            categorize.deseq.df(DE.results,
                        fdr = fdr, log2fold = log2fold, treat = treatment)
        }
#This is the code to incorporate the batch effect into the model:
    else {
        deseq.counts.table = DESeqDataSetFromMatrix(counts.matrix,
                cbind(as.data.frame(batch), as.data.frame(sample.conditions)),
                ~ batch + sample.conditions)
        dds = DESeq(deseq.counts.table)
        vsd = vst(dds)
#remove batch effect from PCA
        assay(vsd) = removeBatchEffect(assay(vsd), vsd$batch)
        pca.plot = plotPCA(vsd, intgroup="sample.conditions", returnData=TRUE)
```

```
        plotPCAlattice(pca.plot, file =paste0('PCA_', name.analysis, '.pdf'))
        DE.results = results(dds)
        DE.results.lattice =
            categorize.deseq.df(DE.results,
                        fdr = fdr, log2fold = log2fold, treat = treatment)
    }
    gene.list= 2^DE.results.lattice$log2FoldChange *
                    -log(DE.results.lattice$pvalue, base = 10)
    names(gene.list) = row.names(DE.results.lattice)
#rank order
    gene.list = sort(gene.list, decreasing = TRUE)
#remove NA entries
    gene.list = gene.list[!is.na(gene.list)]
#remove duplicated genes
    gene.list = gene.list[!duplicated(names(gene.list))]
    myGO = gmtPathways(GO.file)
    fgRes = as.data.frame(fgsea(pathways = myGO,
                            stats = gene.list,
                            minSize=5,
                            maxSize=600,
                            nperm=10000))
    print(head(fgRes[order(fgRes$NES, decreasing =TRUE),]))
#plot most significant two
    pdf(paste("GSEA_first_", name.analysis, ".pdf", sep=''),
      useDingbats = FALSE, width=6.83, height=3.83);
    print(
        plotEnrichment(myGO[[fgRes[order(fgRes$NES, decreasing =TRUE),][[1]][1]]],
                        gene.list) + l
        abs(title=fgRes[order(fgRes$NES, decreasing =TRUE),][[1]][1])
    )
    dev.off()
    pdf(paste("GSEA_second_", name.analysis, ".pdf", sep=''),
      useDingbats = FALSE, width=6.83, height=3.83);
    print(
        plotEnrichment(myGO[[fgRes[order(fgRes$NES, decreasing =TRUE),][[1]][2]]],
                        gene.list) +
        labs(title=fgRes[order(fgRes$NES, decreasing =TRUE),][[1]][2])
    )
    dev.off()
    print('Significantly activated genes')
    print(nrow(DE.results.lattice[DE.results.lattice$response ==
                                  paste0(treatment, " Activated"),]))
    print('Significantly repressed genes')
    print(nrow(DE.results.lattice[DE.results.lattice$response ==
                                  paste0(treatment, " Repressed"),]))
    return(list(DE.results.lattice, fgRes[order(fgRes$NES, decreasing =TRUE),]))
}
```

Running the function multiple times:

```
#This should be identical to the workflow that we perfomed outside the function:

sample.conditions = factor(sapply(strsplit(as.character(colnames(merged.counts)),
              '_rep'), '[', 1), levels=c("A549_no_treatment","A549_30min100nMdex"))

batch = factor(sapply(strsplit(as.character(colnames(merged.counts)),
              '_rep'), '[', 2))

gmt.file = 'h.all.v7.0.symbols.plusGRresp.gmt'

#recall merged.counts is the counts table

no.batch.all.four = deseq.gsea.workflow(merged.counts,
```

```
                  batch = FALSE, sample.conditions,
                  GO.file = gmt.file, name.analysis = 'No_batch_all_four_reps',
                  treatment = 'Dexamethasone',
                          fdr = 0.1, log2fold = 0.0)

head(no.batch.all.four[[1]])
head(no.batch.all.four[[2]])

#Apply a batch effect to each replicate

batch.all = factor(sapply(strsplit(as.character(colnames(merged.counts)),
              '_rep'), '[', 2))

batch.all.four = deseq.gsea.workflow(merged.counts,
                        batch = batch.all, sample.conditions,
                        GO.file = gmt.file, name.analysis = 'Four_batches_all_four_reps',
                        treatment = 'Dexamethasone',
                           fdr = 0.1, log2fold = 0.0)


head(batch.all.four[[1]])
head(batch.all.four[[2]])

#is it reasonable to think that replicate 4 is a single batch and reps 1-3 are another?

batch.test = factor(c('batch1', 'batch1', 'batch1', 'batch2',
                'batch1', 'batch1', 'batch1', 'batch2'))


batch.all.two = deseq.gsea.workflow(merged.counts, batch = batch.test, sample.conditions,
                           GO.file = gmt.file, name.analysis = 'Two_batches_all_four_reps',
                           treatment = 'Dexamethasone', fdr = 0.1, log2fold = 0.0)


head(batch.all.two[[1]])
head(batch.all.two[[2]])
```

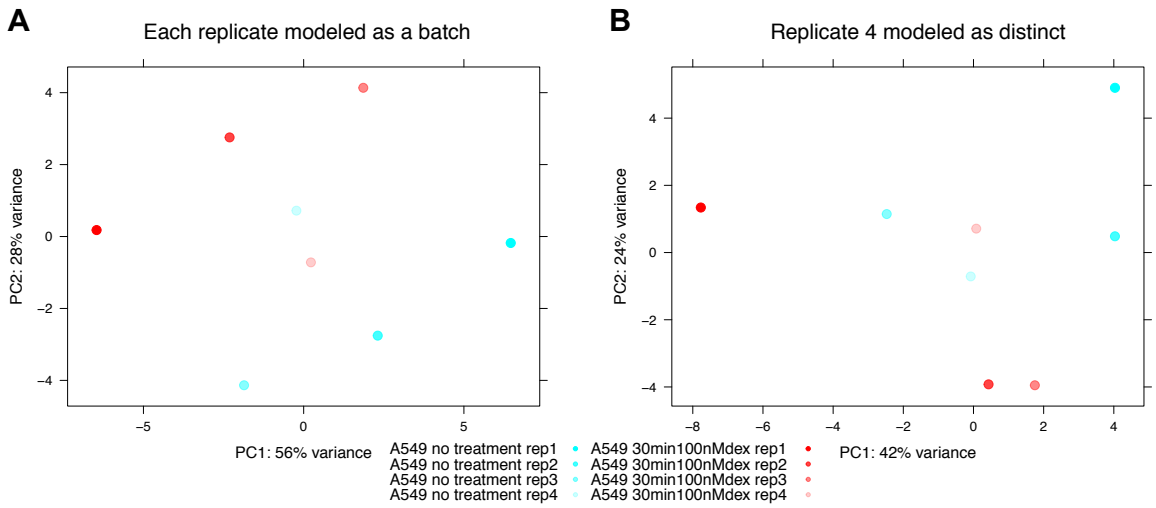

Figure 6: **PCA after batch effect modeling.** Incorporating batch effect residuals into the linear models employed by DESeq2 does not cleanly separate by experiemntal condition. Batch effect was modeled for each replicate (A) and as replicate 4 being a distinct batch compared to replicates 1-3 (B).

5.2 Exploratory analyses

In an attempt to get a feel for the data, I decided to perform a *leave one replicate out* as the comparison group analysis. I hypothesized that replicate 4 is distinct, so I started with comparing the other replicates to replicate 4.

```
#we can change the sample conditinos/comparison groups
sample.conditions.test = factor(c(rep("Replicates1-3", 3), "Replicate4",
                            rep("Replicates1-3", 3), "Replicate4"),
                            levels=c("Replicates1-3","Replicate4"))

deseq.counts.table = DESeqDataSetFromMatrix(merged.counts,
                as.data.frame(sample.conditions.test), ~ sample.conditions.test)

dds = DESeq(deseq.counts.table)

normalized.counts.rna = counts(dds, normalized=TRUE)

DE.results.rep4 = results(dds)

head(DE.results.rep4)

DE.results.rep4.lattice =
    categorize.deseq.df(DE.results.rep4,
                        fdr = 0.1, log2fold = 0.0,
                        treat = 'Replicate 4')

head(DE.results.rep4.lattice)

activated.rep4 = DE.results.rep4.lattice[DE.results.rep4.lattice$response ==
                                'Replicate 4 Activated',]
dim(activated.rep4)
dim(DE.results.rep4.lattice[DE.results.rep4.lattice$response ==
                                'Replicate 4 Repressed',])

# Plot these results function
# I am in the minority in that I prefer lattice to ggplot

ma.plot.lattice <- function(ma.df, filename = 'file.name',
         title.main = "Differential RNA-seq Expression",
         col = c("grey90",  "grey60", "red" , "blue"))
  {
  pdf(paste("MA_plot_", filename, ".pdf", sep=''),
      useDingbats = FALSE, width=3.83, height=3.83);
  print(xyplot(ma.df$log2FoldChange ~ log(ma.df$baseMean, base=10),
               groups=ma.df$response,
               col= col,
                main=title.main, scales="free", aspect=1, pch=20, cex=0.5,
               ylab=expression("log"[2]~"RNA-seq change"),
               xlab=expression("log"[10]~"Mean of Normalized Counts"),
               par.settings=list(par.xlab.text=list(cex=1.1,font=2),
                                 par.ylab.text=list(cex=1.1,font=2))));
  dev.off()
  }

ma.plot.lattice(DE.results.rep4.lattice, filename = 'A549_Replicate_4',
        title.main = "Differential Expression")
```

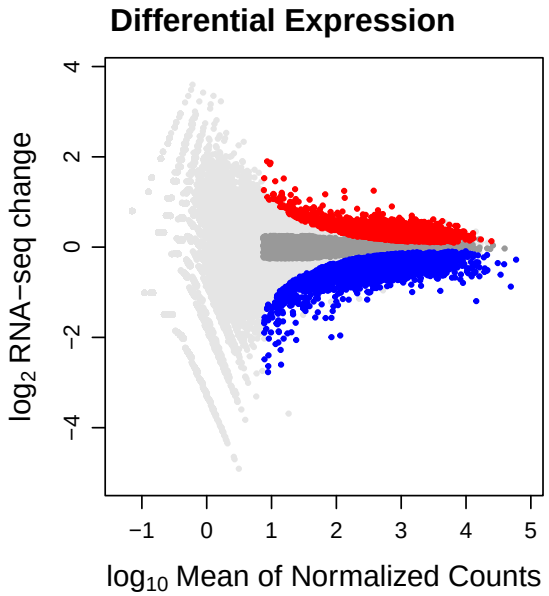

Figure 7: **Differential Expression of Replicate 4.** Each gene is a point that is colored by it's category. The dark grey points are confidently *unchanged*. The blue are *repressed* and red are *activated*. All other points are light grey. The red and blue genes are defined by the *fdr* and *log2fold* options with the *categorize.deseq.df* function. Comparing differential espression of Replicate 4 compared to the other three replicates indicates that about 3000 genes are both activated and repressed.

Comparing replicate 1-3 to replicate 4 resulted in 6019 genes differentially expressed (DE) at an FDR of 0.1, so I was curious if these number would be similar in other comparisons. 3 genes were DE when we used replicate 3 as the out group;14 genes were DE when replicate 2 was the out group; 2 genes were DE when replicate 1 was the out group.

Taking a look at the gene set enrichment analysis, the categories *ribosome, translation,* and *serum response* are found multiple times among the top pathways. Ribosome biogenesis is linked to nutrient availability (Thomas, 2000) and cells in culture react strongly to serum (Mahat and Lis, 2017). I hypothesize that the experimental conditions of these cells were distinct compared to the other replicates in the time since they were split or the lot number (and thus nutrient content) of the FBS that was used.

I do not know of a way to test this hypothesis, but it highlights the importance of keeping a well annotated lab notebook that you can refer back to. Regardless, I think there is sufficient justification for excluding replicate 4 from the analyses. However, this is subjective and I would urge transparency in methods if you have to make these calls in a publication. One can use `GitHub` to track the step-by-step analyses that go into a manuscript and provide excruciating detail for these type of exploratory analyses that inform on good faith judgement calls that the researcher has to take.

```
gmt.file = paste0(msigdb.url, 'c2.all.v7.0.symbols.gmt')

sample.conditions.4out = factor(c(rep("Replicates1-3", 3), "Replicate4",
                            rep("Replicates1-3", 3), "Replicate4"),
                            levels=c("Replicates1-3","Replicate4"))

#same as prior
no.batch.4out = deseq.gsea.workflow(merged.counts,
                  batch = FALSE, sample.conditions.4out,
                  GO.file = gmt.file, name.analysis = 'Leave Rep 4 out',
                  treatment = 'Leave Rep 4 Out',
                          fdr = 0.1, log2fold = 0.0)

sample.conditions.3out = factor(c(rep("Replicates124", 2), "Replicate3",
                            rep("Replicates124", 3) , "Replicate3","Replicates124"),
                            levels=c("Replicates124","Replicate3"))

sample.conditions.2out = factor(c("Replicates134", "Replicate2",
                             rep("Replicates134", 3),"Replicate2",
                             rep("Replicates134",2)))

sample.conditions.1out = factor(c("Replicate1", rep("Replicates2-4", 3),
                             "Replicate1", rep("Replicates2-4", 3)),
                            levels=c("Replicates2-4","Replicate1"))


no.batch.3out = deseq.gsea.workflow(merged.counts,
                  batch = FALSE, sample.conditions.3out,
                  GO.file = gmt.file, name.analysis = 'Leave Rep 3 out',
                  treatment = 'Leave Rep 4 Out',
                          fdr = 0.1, log2fold = 0.0)

no.batch.2out = deseq.gsea.workflow(merged.counts,
                  batch = FALSE, sample.conditions.2out,
                  GO.file = gmt.file, name.analysis = 'Leave Rep 2 out',
                  treatment = 'Leave Rep 4 Out',
                          fdr = 0.1, log2fold = 0.0)

no.batch.1out = deseq.gsea.workflow(merged.counts,
                  batch = FALSE, sample.conditions.1out,
                  GO.file = gmt.file, name.analysis = 'Leave Rep 1 out',
                  treatment = 'Leave Rep 4 Out',
                          fdr = 0.1, log2fold = 0.0)
```

5.3 Final analyses

Comparing dexamethasone treatment in replicates 1-3 indicates that canonical NFκB signaling genes are changing expression and there are 31 activated and 11 repressed genes at an FDR of 0.1. Much is already known about the antagonism of glucocorticoid signaling and NFκB signaling, so this result is consistent with known biology.

```
sample.conditions.1to3 = factor(sapply(strsplit(as.character(colnames(merged.counts)),
              '_rep'), '[', 1)[c(1,2,3,5,6,7)],
              levels=c("A549_no_treatment","A549_30min100nMdex"))

merged.counts.1to3 = merged.counts[,c(1,2,3,5,6,7)]


deseq.gsea.1to3 = deseq.gsea.workflow(merged.counts.1to3,
                  batch = FALSE, sample.conditions.1to3,
                  GO.file = 'h.all.v7.0.symbols.plusGRresp.gmt',
                  name.analysis = 'Dexamethasone',
                  treatment = 'Dexamethasone',
                          fdr = 0.1, log2fold = 0.0)

ma.plot.lattice(deseq.gsea.1to3[[1]], filename = 'A549_RNA_dex',
        title.main = "Differential Expression")
```

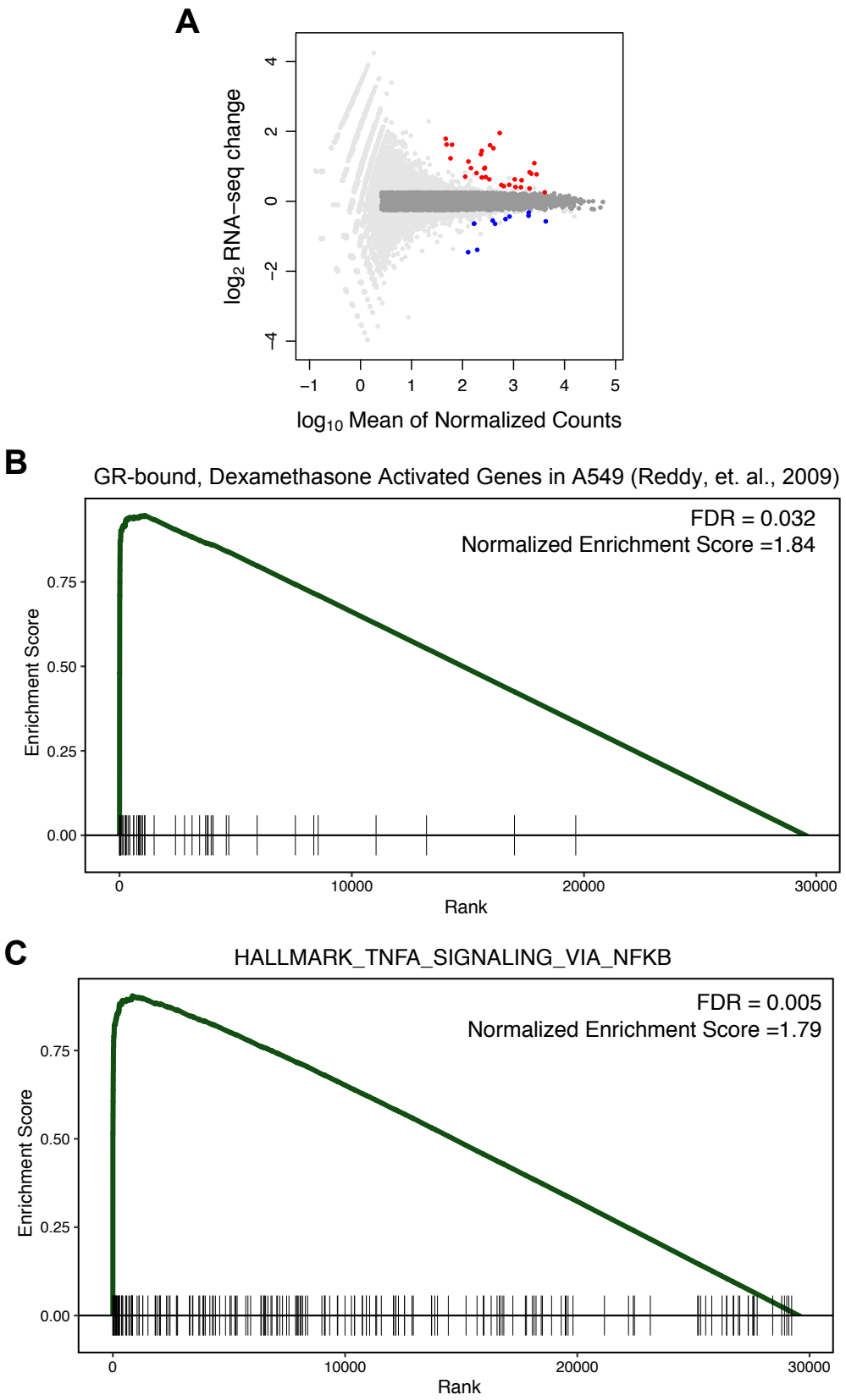

Figure 8: **MA plot and GSEA of final analysis.** A) An MA plot comparing no treatment and dexamethasone treated cells with replicate 4 excluded. B) The positive control gene set contains the highes normalized enrichment score. C) NFκB signaling is the most enriched gene set outside the positive control.

References

Anders S, Huber W (2010). "Differential expression analysis for sequence count data." *Genome biol*, **11**(10), R106.

Karolchik D, Barber GP, Casper J, Clawson H, Cline MS, Diekhans M, Dreszer TR, Fujita PA, Guruvadoo L, Haeussler M, *et al.* (2014). "The UCSC genome browser database: 2014 update." *Nucleic acids research*, **42**(D1), D764–D770.

Kim D, Langmead B, Salzberg SL (2015). "HISAT: a fast spliced aligner with low memory requirements." *Nature methods*, **12**(4), 357–360.

Kodama Y, Shumway M, Leinonen R (2011). "The Sequence Read Archive: explosive growth of sequencing data." *Nucleic acids research*, **40**(D1), D54–D56.

Love MI, Huber W, Anders S (2014). "Moderated estimation of fold change and dispersion for RNA-Seq data with DESeq2." *bioRxiv*.

Mahat DB, Lis JT (2017). "Use of conditioned media is critical for studies of regulation in response to rapid heat shock." *Cell Stress and Chaperones*, **22**(1), 155–162.

Mi H, Muruganujan A, Ebert D, Huang X, Thomas PD (2019). "PANTHER version 14: more genomes, a new PANTHER GO-slim and improvements in enrichment analysis tools." *Nucleic acids research*, **47**(D1), D419–D426.

Reddy TE, Pauli F, Sprouse RO, Neff NF, Newberry KM, Garabedian MJ, Myers RM (2009). "Genomic determination of the glucocorticoid response reveals unexpected mechanisms of gene regulation." *Genome research*.

Sathyan KM, McKenna BD, Anderson WD, Duarte FM, Core L, Guertin MJ (2019). "An improved auxin-inducible degron system preserves native protein levels and enables rapid and specific protein depletion." *Genes & development*, **33**(19-20), 1441–1455.

Subramanian A, Tamayo P, Mootha VK, Mukherjee S, Ebert BL, Gillette MA, Paulovich A, Pomeroy SL, Golub TR, Lander ES, *et al.* (2005). "Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles." *Proceedings of the National Academy of Sciences*, **102**(43), 15545–15550.

Thomas G (2000). "An encore for ribosome biogenesis in the control of cell proliferation." *Nature cell biology*, **2**(5), E71–E72.

Wang Z, Civelek M, Miller C, Sheffield N, Guertin M, Zang C (2018). "BART: a transcription factor prediction tool with query gene sets or epigenomic profiles." *Bioinformatics (Oxford, England)*, **34**(16), 2867.