An Introduction to RNA-sequencing

GSND 5340Q, BMDA

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

2024-05-29

Installing R Packages:

Install the following tools: `Rsubread`, `Rsamtools`, `edgeR`, `DESeq2`, `sva` `SummarizedExperiment`, `ComplexHeatmap`, `umap`, and the `TBSignatureProfiler`. We will also need help from the `tidyverse`.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(c("Rsubread","Rsamtools","tidyverse",
  "SummarizedExperiment","edgeR","DESeq2","sva",
  "ComplexHeatmap", "TBSignatureProfiler","umap"))
```

Installing and using the SCTK

```
install.packages("devtools")
devtools::install_github("wevanjohnson/singleCellTK")
library(singleCellTK)
singleCellTK()

### Example: open downstream_analysis/
### features_combined.txt and meta_data.txt
```

Load Packages for RNA-Seq

We will be using the following packages for our RNA-seq lecture:

```
library(tidyverse) ## tools for data wranging
library(Rsubread) ## alignment and feature counts
library(Rsamtools) ## managing .sam and .bam files
library(SummarizedExperiment) ## managing counts data
library(edgeR) ## differential expression
library(DESeq2) ## differential expression
library(ComplexHeatmap) ## Heatmap visualization
library(TBSignatureProfiler) ## TB signature analysis
library(umap) ## dimenstion reduction and plotting data
```

Objective

- Disclaimer: non-comprehensive introduction to RNA-sequencing
- Introduce preprocessing steps
- Visualization
- Analytical methods
- Common software tools

Steps to an RNA-seq Analysis (Literacy)

1. Preprocessing and QC:
   - Fasta and Fastq files
   - FastQC: good vs. bad examples
   - Visualization
2. Alignment
   - Obtaining genome sequence and annotation
   - Software: Bowtie, TopHat, STAR, Subread/Rsubread
3. Expression Quantification
   - Count reads hitting genes, etc
   - Approaches/software: HT-Seq, STAR, Cufflinks, RPKM FPKM or CPM, RSEM, edgeR, findOverlaps (GenomicRanges). featureCounts (Rsubread)

Steps to an RNA-seq Analysis (Literacy)

4. More visualization
   - Heatmaps, boxplots, PCA, t-SNE, UMAP
5. Differential Expression
   - Batch correction
   - Overdispersion
   - General Workflow
   - Available tools: edgeR, DESeq, Limma/voom
   - Even more visualization!!

Illumina Sequencing Workflow

1 Library Preparation

Fragment DNA
Repair ends
Add A overhang
Ligate adapters
Purify

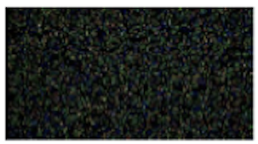
2 Cluster Generation

Hybridize to flow cell
Extend hybridized template
Perform bridge amplification
Prepare flow cell for sequencing

3 Sequencing

Perform sequencing
Generate base calls

4 Data Analysis



Images
Intensities
Reads
Alignments

Sequencing Data Formats

Genome sequcencing data is often stored in one of two formats, FASTA and FASTQ text files. For example a FASTA files looks like the following:

```
>chrX
ttgaactcctgacctcaggtgatccgcccgccttgacctcccaaagcgct
cctgcctcagcctcccgagtagctgggactacaggtgcctgccaccatgc
....
caggctaatttttgtatttttagtagagacggggtttcaccatgttagc
caggatggtctcaatctcctgacctcatgatccgcctgcctcggcctccc
>chrY
tgtacactttaaatgggtgaattttatggaatgtgaattataCGTGTTGG
CTTGTAAAAAAAAATGATGGAGATGGAGACGTGACTCTAGCGTGAAGGGG
...
GTGGGGAGAGTAGATCTAGAGTGGAGACACCACTTTTAGGAGGTATGATC
cctgccaccatgcctggctaattttgtattttagtagagacagggttt
```

FASTQ Files

We can also store confidence or quality scores using a FASTQ format:

```
@HWI-EAS240_0001:2:1:1142:17571#0/1
CTCTCTTTTCTCCCCANGTCTCCTCATGACCATATCCNTGTTGTCCATTGTGTANGGNNNCTTGCGCTGCAACATC
+HWI-EAS240_0001:2:1:1142:17571#0/1
bababb`abbbbabbYB^[^^`_`_bbbbbbbbbb`_B`_``^`^bbbba^`^^BOOBBBNOOOObY^^bbbbbbb
@HWI-EAS240_0001:2:1:1142:6453#0/1
CAGGACGTGCACTATGNCATCCACGATGCAAGTCTTANCATTATTCAGGATACANATANNGGCAAACGTGTAATTA
+HWI-EAS240_0001:2:1:1142:6453#0/1
_U][aa_aa^^```^[B][][]]]S```Y``a`aaZ]B]]^^^[^```[OVY]XBNNNBB]]]]]L^^^^```[V`
@HWI-EAS240_0001:2:1:1142:19443#0/1
TCAGAAAACAGAAAGGTCATTTTCTTACTTCTTTGCANGATGCCACCCTCCAGANCAGNNAATTGCTACTTTGCAC
+HWI-EAS240_0001:2:1:1142:19443#0/1
bbbabb^^bbb^bbabb]bb]^]`]]]bbbbbbbbYYBYXOOOOOb`bbVY^HBOOOBBOJOLO_bbbb]b_b`a
```

FASTQ Encoding

In order to translate FASTQ quality scores:

```
 SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS.....................................................
 ..........................XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.....................
 ............................IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII.....................
 ..............................JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ.....................
 LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL.....................................................
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
  |                          |    |        |                              |                     |
33                        59   64       73                            104                   126

S - Sanger        Phred+33,  raw reads typically (0, 40)
X - Solexa        Solexa+64, raw reads typically (-5, 40)
I - Illumina 1.3+ Phred+64,  raw reads typically (0, 40)
J - Illumina 1.5+ Phred+64,  raw reads typically (3, 40)
   with 0=unused, 1=unused, 2=Read Segment Quality Control Indicator (bold)
   (Note: See discussion above).
L - Illumina 1.8+ Phred+33,  raw reads typically (0, 41)
```

FASTQ Probability

And now converting to confidence probabilities:

> Sanger

	ASCII	Quality
!	33	0.0000
"	34	0.2057
#	35	0.3690
$	36	0.4988
%	37	0.6019
&	38	0.6838
'	39	0.7488
(	40	0.8005
)	41	0.8415
*	42	0.8741
+	43	0.9000
,	44	0.9206
-	45	0.9369
.	46	0.9499
/	47	0.9602
0	48	0.9684

> Solexa

	ASCII	Quality
;	59	0.2403
<	60	0.2847
=	61	0.3339
>	62	0.3869
?	63	0.4427
@	64	0.5000
A	65	0.5573
B	66	0.6131
C	67	0.6661
D	68	0.7153
E	69	0.7597
F	70	0.7992
G	71	0.8337
H	72	0.8632
I	73	0.8882
J	74	0.9091
K	75	0.9264
L	76	0.9406

> Illumina1.3

	ASCII	Quality
@	64	0.5000
A	65	0.5573
B	66	0.6131
C	67	0.6661
D	68	0.7153
E	69	0.7597
F	70	0.7992
G	71	0.8337
H	72	0.8632
I	73	0.8882
J	74	0.9091
K	75	0.9264
L	76	0.9406
M	77	0.9523
N	78	0.9617
O	79	0.9693
P	80	0.9755
Q	81	0.9804

> Illumina1.5

	ASCII	Quality
B	66	0.3690
C	67	0.4988
D	68	0.6019
E	69	0.6838
F	70	0.7488
G	71	0.8005
H	72	0.8415
I	73	0.8741
J	74	0.9000
K	75	0.9206
L	76	0.9369
M	77	0.9499
N	78	0.9602
O	79	0.9684
P	80	0.9749
Q	81	0.9800
R	82	0.9842
S	83	0.9874

Preprocessing and QC using FASTQC

FastQC provides a simple way to do QC checks on raw sequence data:

- Import of data from BAM, SAM or FastQ files
- Quick overview and summary graphs and tables to quickly assess your data
- Export of results to an HTML based permanent report
- Offline operation to allow automated generation of reports without running the interactive application

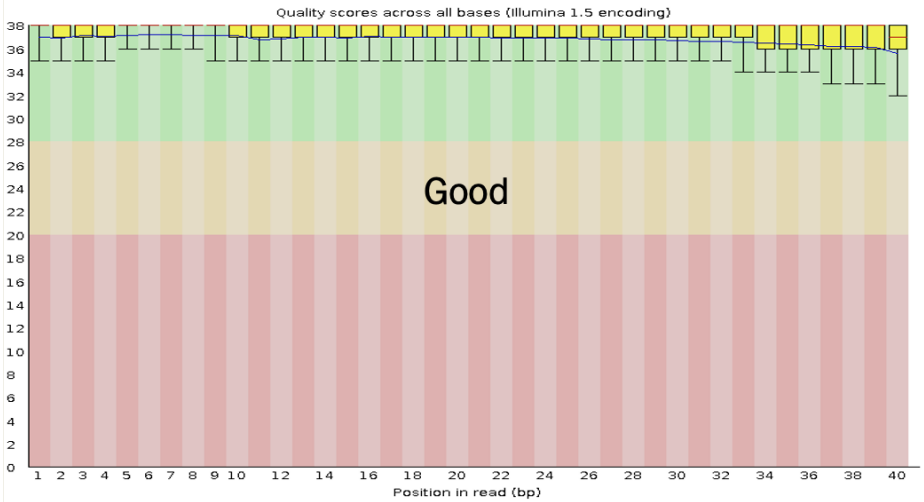
Preprocessing and QC using FASTQC

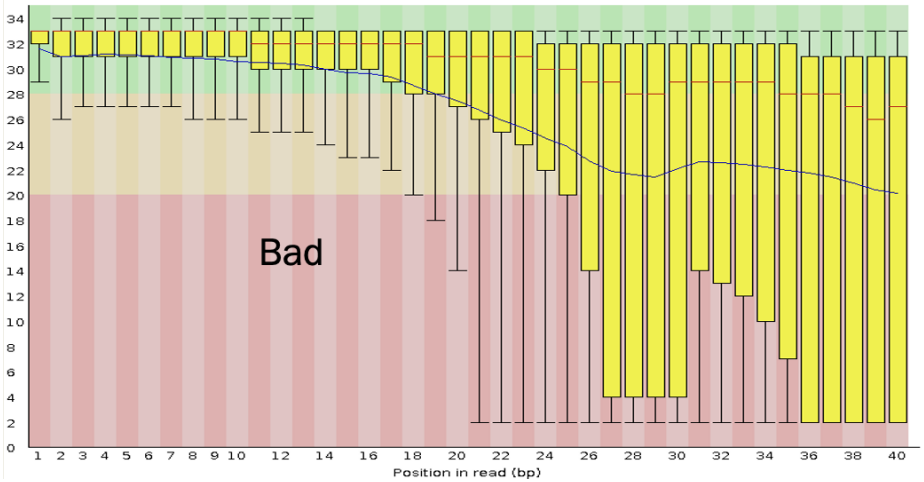
To run FastQC you can launch the GUI app, or run form the command line:

```
rna_seq/FastQC/./fastqc \
  rna_seq/reads/R01_10_short500K.fq.gz
```

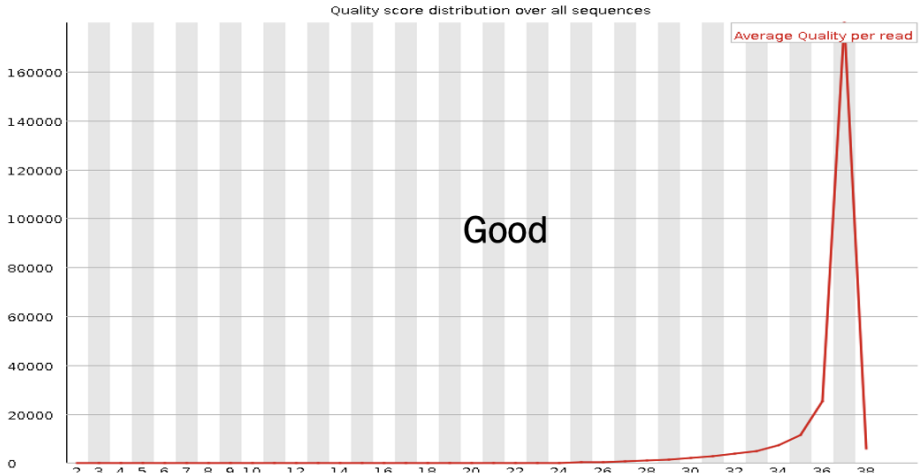
FastQC Score Distribution

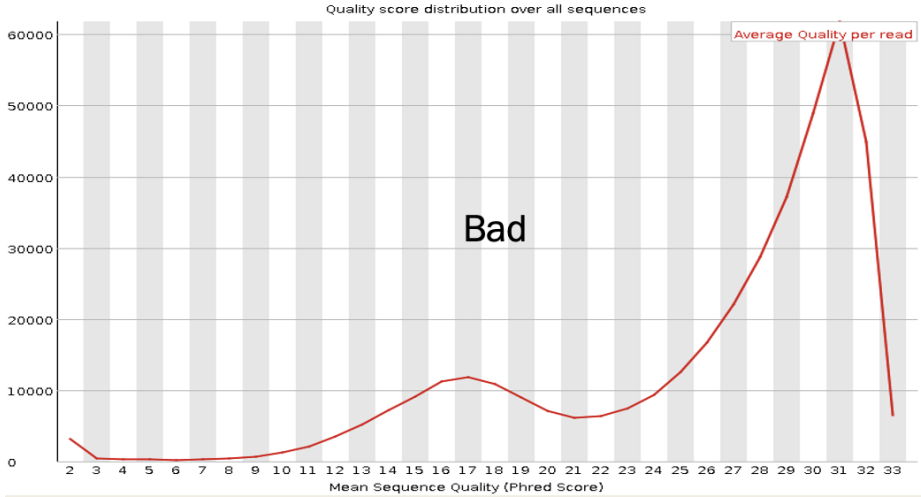
Quality scores



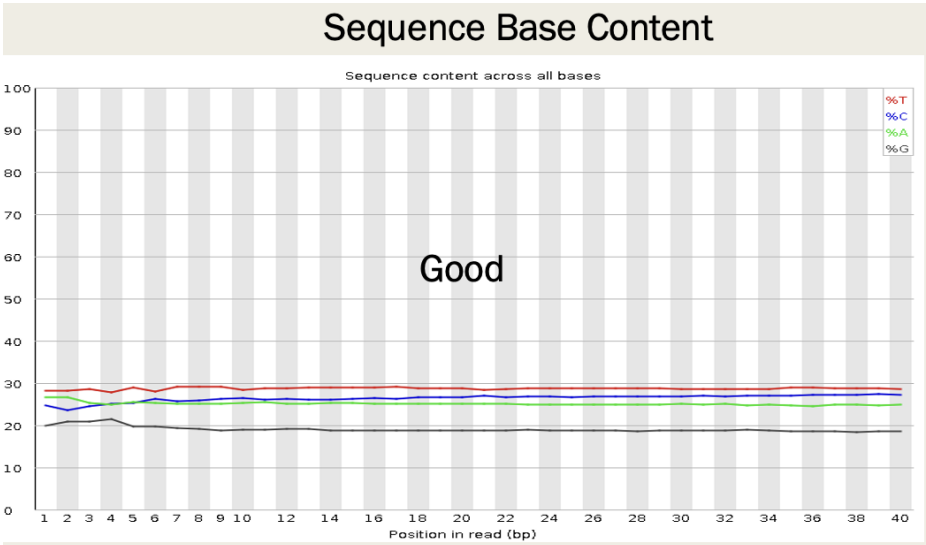


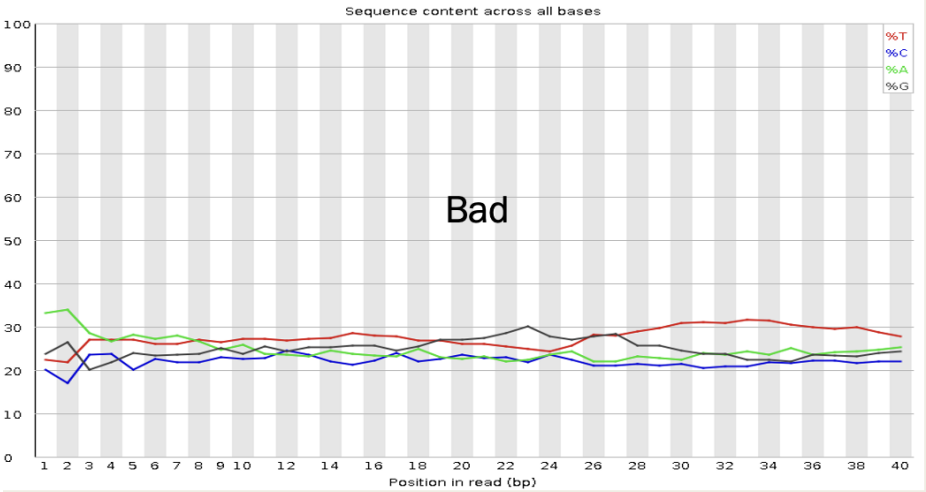
Quality score distribution

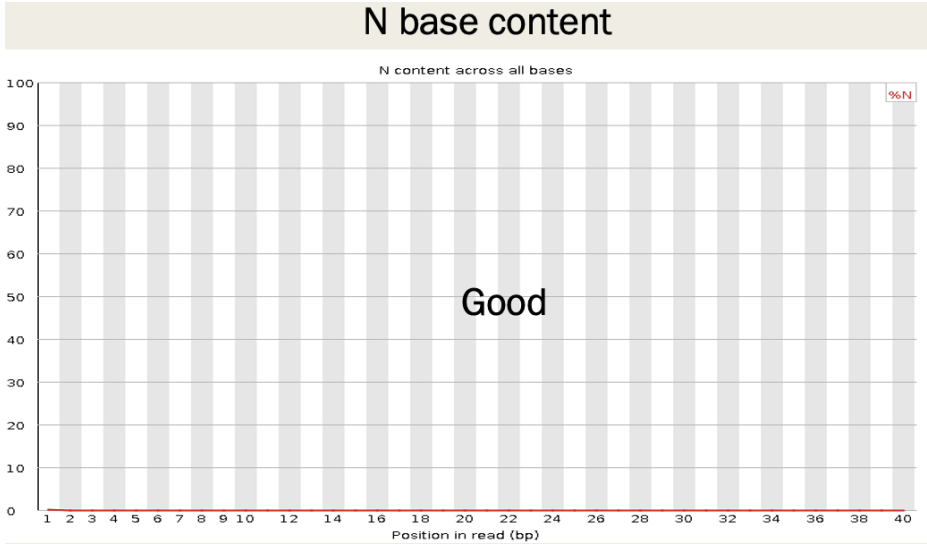


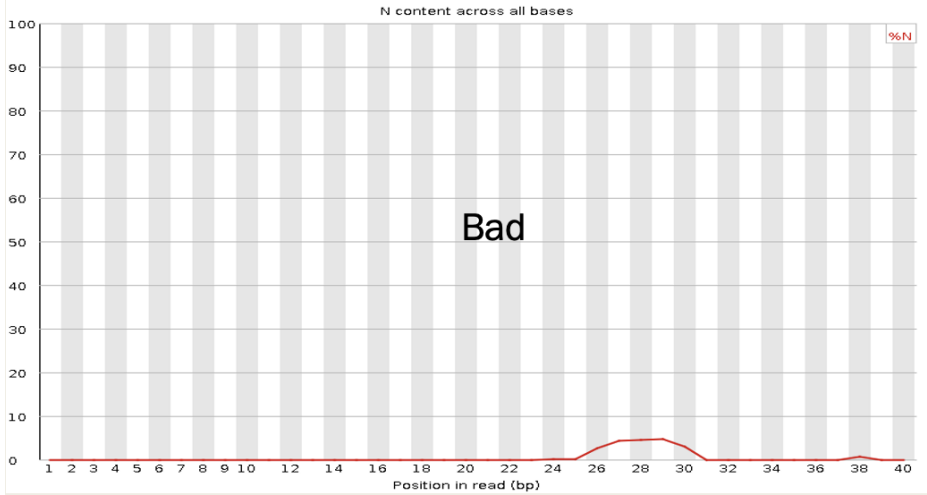


FastQC Base and N Distribution



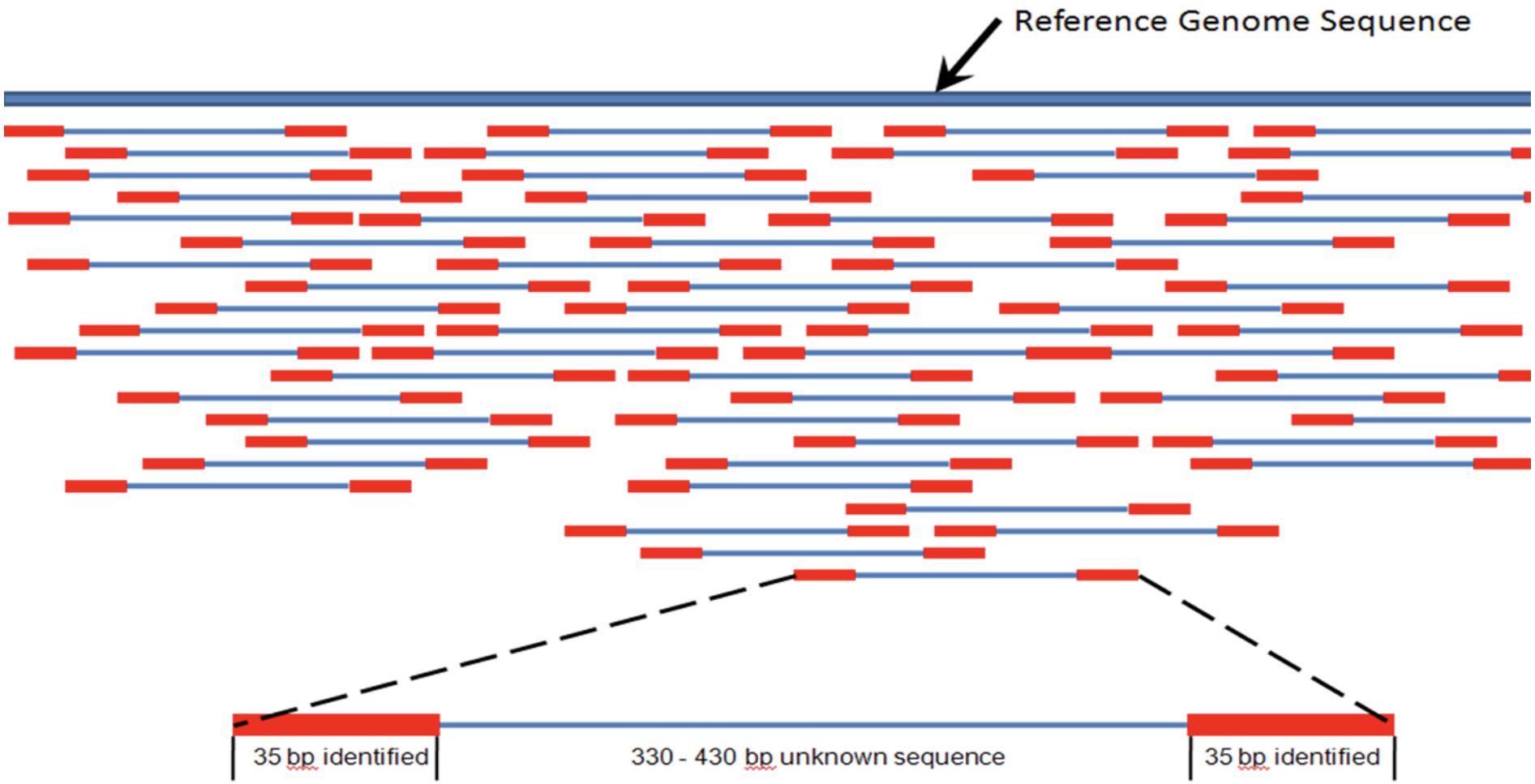






Alignment to the Reference Genome

Goal: Find the genomic Location of origin for the sequencing read.
Software: `Bowtie2`, `TopHat`, `STAR`, `Subread/Rsubread`, many others!



Indexing your genome

Abraham Lincoln: "Give me six hours to chop down a tree and I will spend the first four sharpening the axe."

(4 minutes indexing the genome, 2 minutes aligning the reads)

Indexing your genome

Note that you will rarely do this for human alignment. You will usually download an existing index given to you by others who have already done this work. You will do this often if you are aligning microbial reads, e.g. MTB or some other organism for which others have not already made your index for you.

```
buildindex(basename="rna_seq/genome/ucsc.hg19.chr1_120-150M",
 reference="rna_seq/genome/ucsc.hg19.chr1_120-150M.fasta.gz")
```

Took me ~0.2 minutes!

Aligning your reads:

Note that this outputs results in a .bam file and not a .sam file

```
align(index="rna_seq/genome/ucsc.hg19.chr1_120-150M",
      readfile1="rna_seq/reads/R01_10_short500K.fq.gz",
      output_file="rna_seq/alignments/R01_10_short.bam",
      nthreads=4)
```

My laptop is an Apple M2, which has 8 cores (used 4 cores), 24GB RAM:

- Took 15.7 minutes to align ~60M reads to the 30M bases
- Took 0.7 minutes to align ~6.5M reads to the 30M bases
- Took 0.3 minutes to align ~500K reads to the 30M bases

Aligned Sequencing Data Formats (SAM and BAM)

Note that Rsubread outputs a .bam file (bam = binary alignment map) and not a .sam file (sam = sequence alignment map). Here is some information about a .sam file: https://en.wikipedia.org/wiki/SAM_(file_format)

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,255}`	Query template NAME
2	FLAG	Int	`[0,2^16-1]`	bitwise FLAG
3	RNAME	String	`\*|[!-()+-<>-~][!-~]*`	Reference sequence NAME
4	POS	Int	`[0,2^29-1]`	1-based leftmost mapping POSition
5	MAPQ	Int	`[0,2^8-1]`	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[!-()+-<>-~][!-~]*`	Ref. name of the mate/next segment
8	PNEXT	Int	`[0,2^29-1]`	Position of the mate/next segment
9	TLEN	Int	`[-2^29+1,2^29-1]`	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

Aligned Sequencing Data Formats (SAM and BAM)

To convert .sam to .bam or vice versa, a package called Rsamtools. Using Rsamtools, you can convert bam to sam as follows:

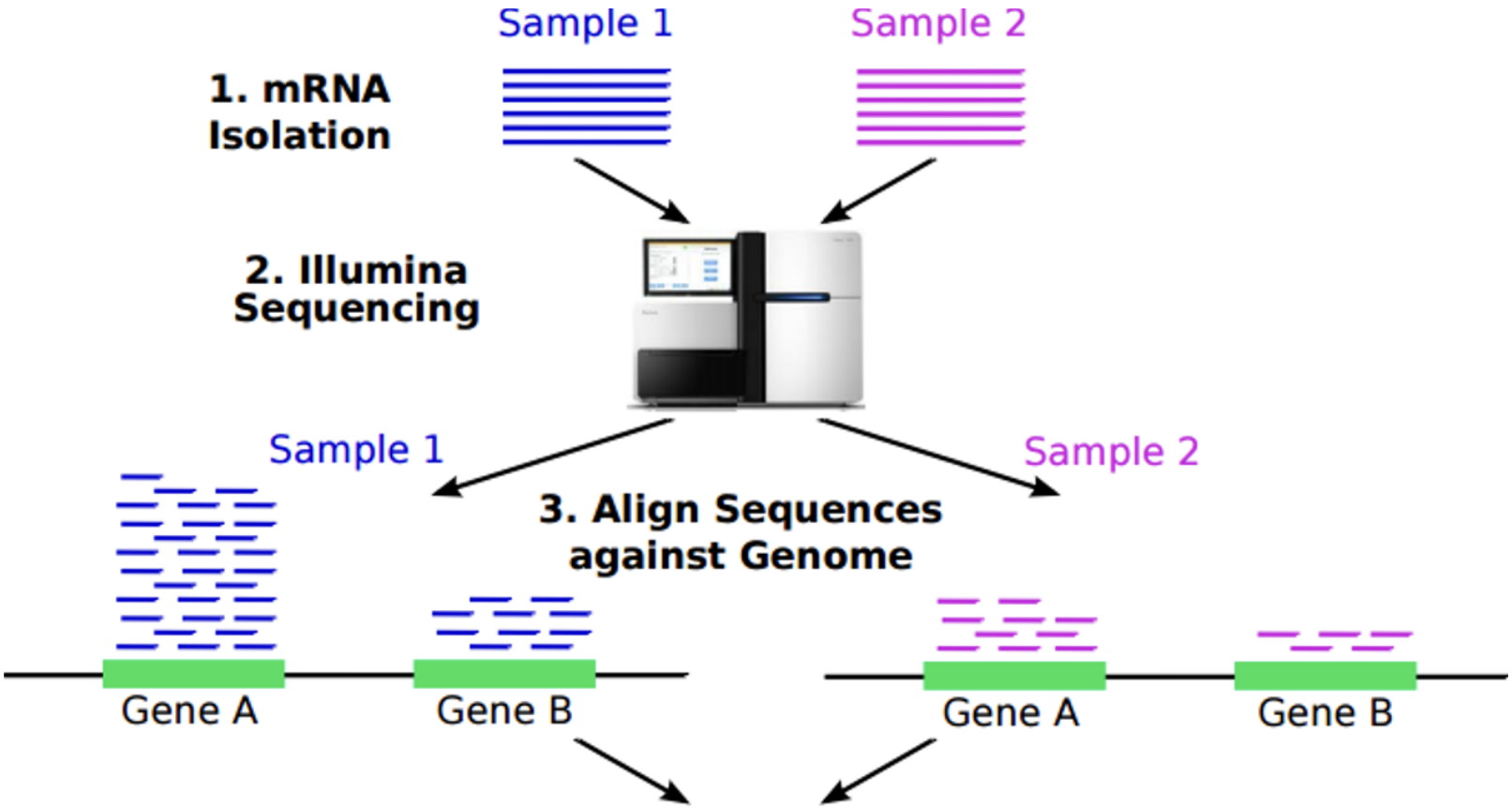
```
asSam("rna_seq/alignments/R01_10_short.bam",
      overwrite=T)

# To convert to bam:
#asBam("rna_seq/alignments/R01_10_short.bam")
```

Feature counts

Now we can count reads hitting genes.
Approaches/software:

- HT-Seq
- STAR
- Cufflinks
- RPKM FPKM or CPM
- RSEM
- edgeR
- findOverlaps (GenomicRanges)
- featureCounts (Rsubread)

Feature counts



Feature counts

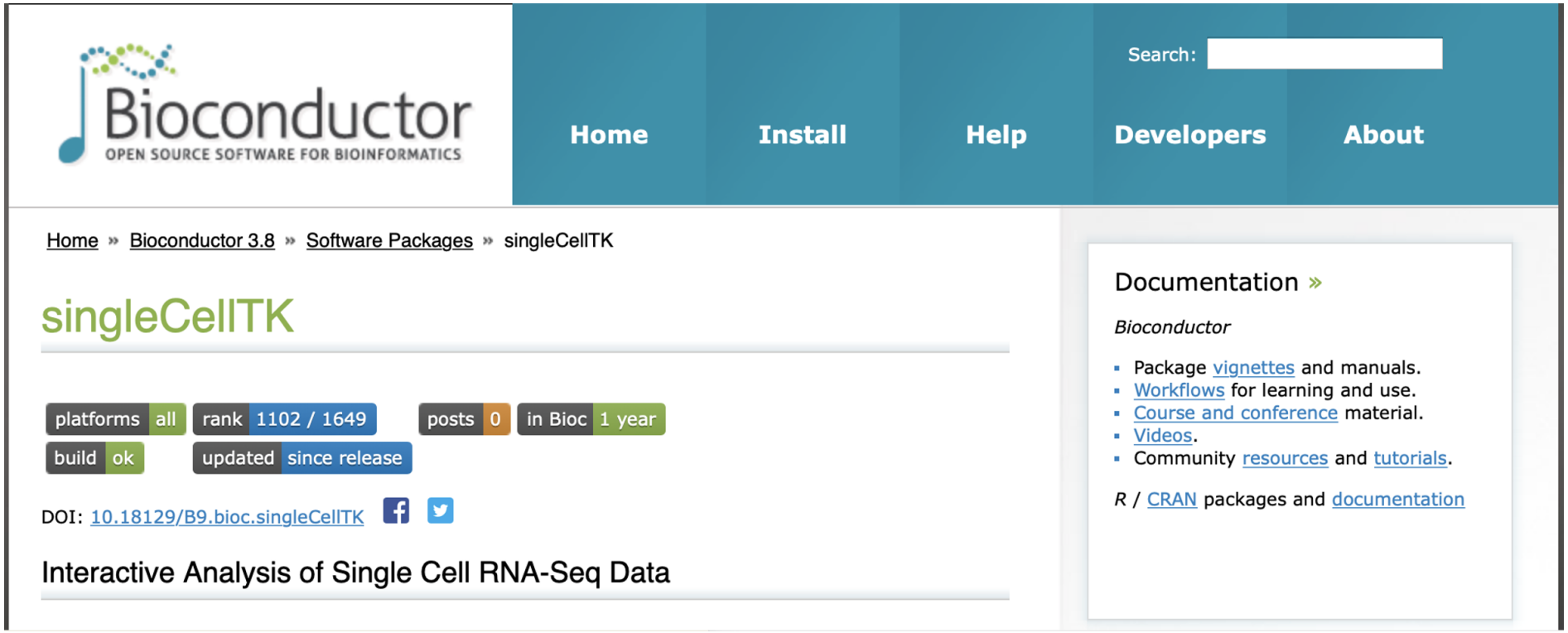
```
fCountsList = featureCounts(
  "rna_seq/alignments/R01_10_short.bam",
  annot.ext="rna_seq_files/genome/genes.chr1_120-150M.gtf",
  isGTFAnnotationFile=TRUE)

featureCounts = cbind(fCountsList$annotation[,1],
                      fCountsList$counts)

write.table(featureCounts,
    "rna_seq/alignments/R01_10_short.features.txt",
    sep="\t", col.names=FALSE, row.names=FALSE, quote=FALSE)
```

SCTK

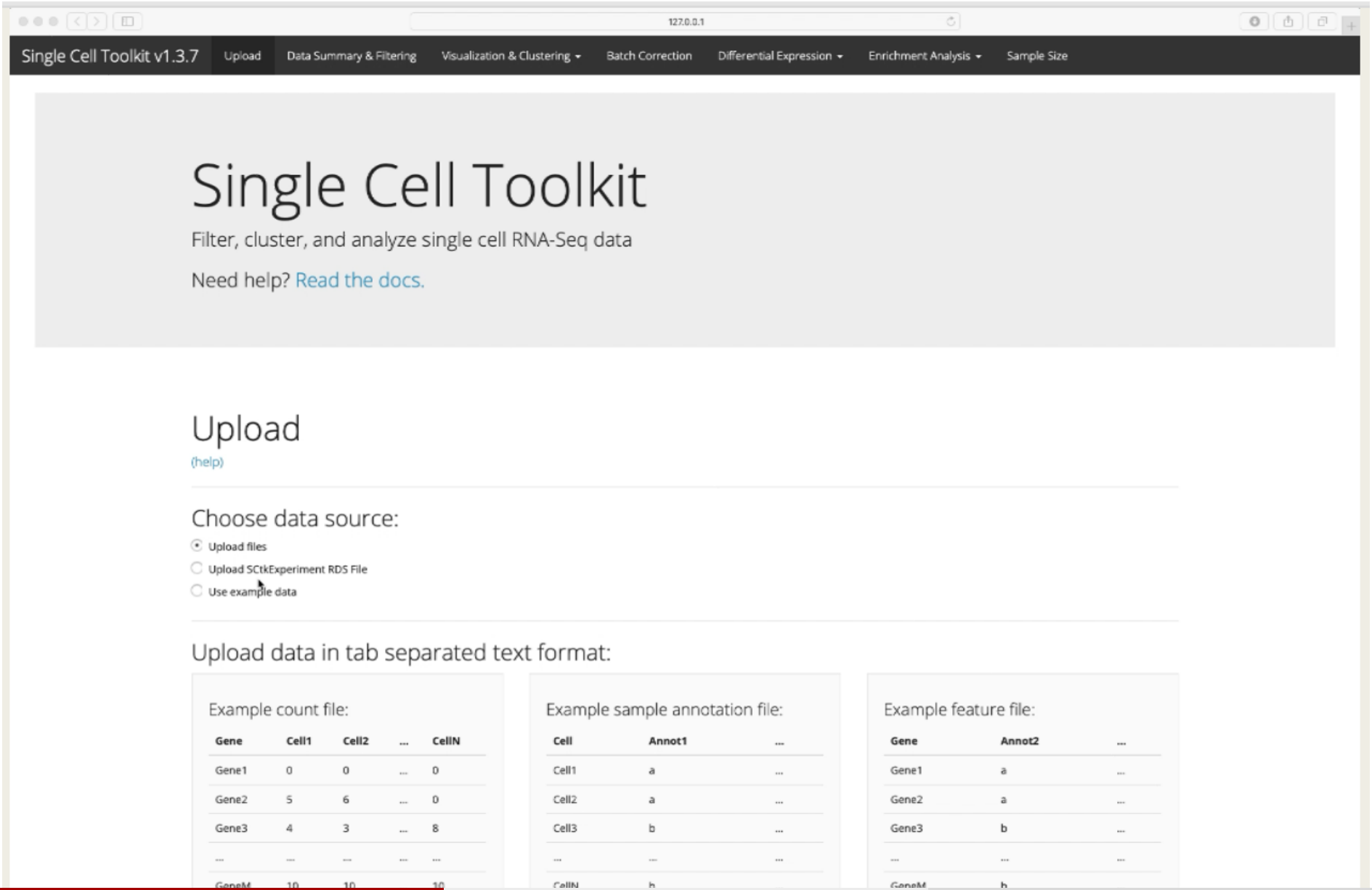
Use the Single Cell Toolkit (SCTK) to analyze your RNA-seq data!

- Inputs: RNA-seq, Nanostring, Proteomic, immunological assay data
- Interactive analyses and visualization of data
- Save results, figures, etc
- Sophisticated data structures
- R/Bioconductor package

SCTK


Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS
Search:
Home
Install
Help
Developers
About
Home » Bioconductor 3.8 » Software Packages » singleCellTK
singleCellTK
platforms all
rank 1102 / 1649
posts 0
in Bioc 1 year
build ok
updated since release
DOI: 10.18129/B9.bioc.singleCellTK
Interactive Analysis of Single Cell RNA-Seq Data
Documentation »
Bioconductor
Package vignettes and manuals.
Workflows for learning and use.
Course and conference material.
Videos.
Community resources and tutorials.
R / CRAN packages and documentation

SCTK


127.0.0.1
Single Cell Toolkit v1.3.7
Upload
Data Summary & Filtering
Visualization & Clustering
Batch Correction
Differential Expression
Enrichment Analysis
Sample Size
Single Cell Toolkit
Filter, cluster, and analyze single cell RNA-Seq data
Need help? Read the docs.
Upload
(help)
Choose data source:
Upload files
Upload SCtkExperiment RDS File
Use example data
Upload data in tab separated text format:
Example count file:
Gene Cell1 Cell2 ... CellN
Gene1 0 0 ... 0
Gene2 5 6 ... 0
Gene3 4 3 ... 8
Example sample annotation file:
Cell Annot1 ...
Cell1 a ...
Cell2 a ...
Cell3 b ...
Example feature file:
Gene Annot2 ...
Gene1 a ...
Gene2 a ...
Gene3 b ...

Installing and using the SCTK

```
install.packages("devtools")
devtools::install_github("wevanjohnson/singleCellTK")
library(singleCellTK)
singleCellTK()

### Example: open downstream_analysis/
### features_combined.txt and meta_data.txt
```

Data Structures

A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks.

Data Structures

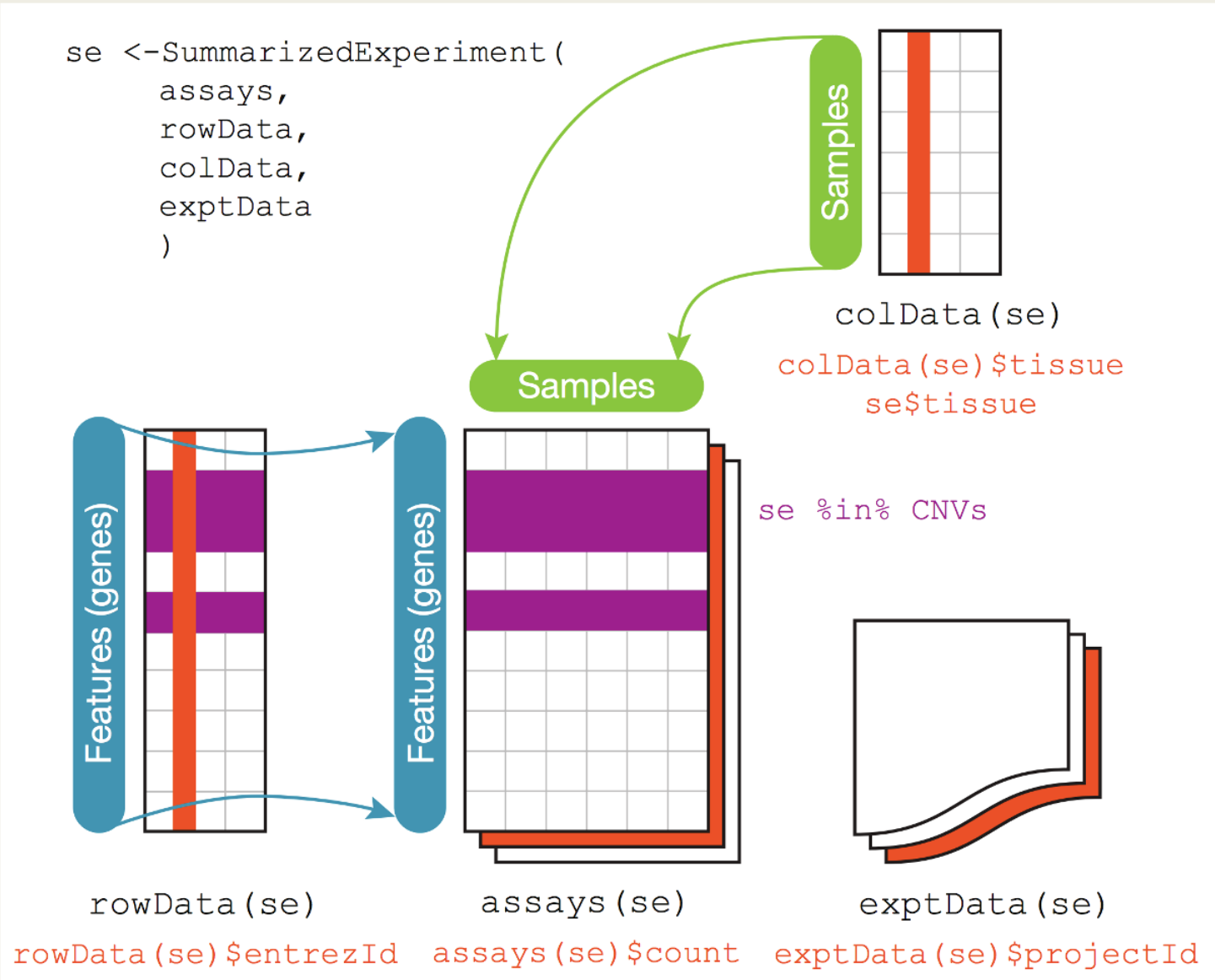
Data structures in R programming are tools for holding multiple values, variables, and sometimes functions

Please think very carefully about the way you manage and store your data! This can make your life much easier and make your code and data cleaner and more portable!

Data Structures

There are advanced R data structures, **S3** and **S4** class objects, that can facilitate object orientated programming. One useful example of an S4 class data structure is the **SummarizedExperiment** object.

Data Structures

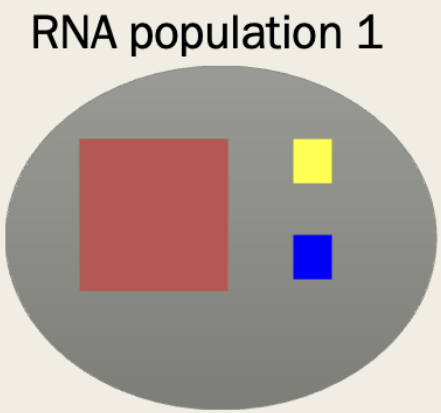


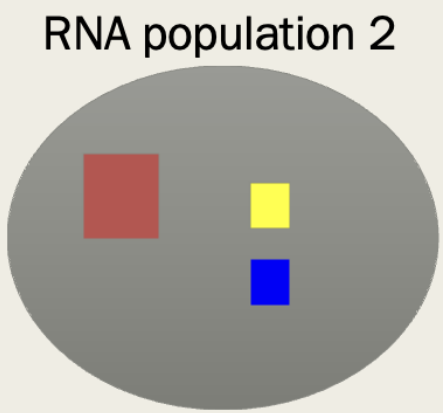
Normalization

Need to normalize data because of:

- Sequencing depth difference in each RNA sample
- RNA composition differences
- Highly expressed genes can consume a substantial proportion of RNA-Seq reads, causing other genes to be under-sampled
- Different methods
  - Log counts
  - Counts per million (CPM and logCPM; RPKM, FPKM)
  - Trimmed mean of M-values (edgeR/limma)
  - Median of Ratios method (DESeq)

Normalization





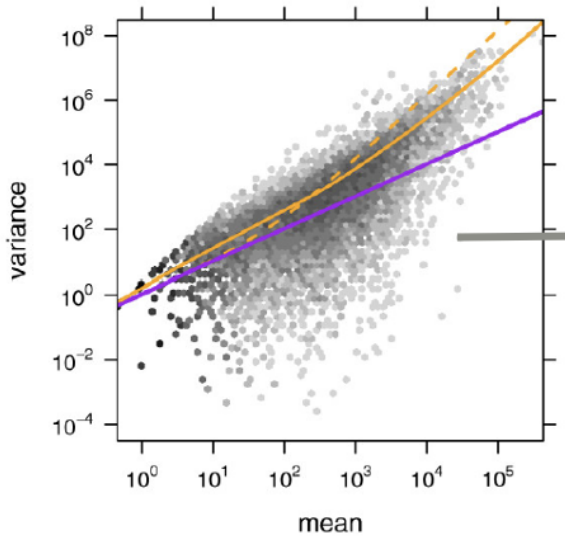
Assuming equal sequencing depth, yellow and blue will get lower RPKM in RNA population 1 although they have equal expression levels

Problem of overdispersion:

Alignment and feature counting result in discrete count data (i.e. the number of reads to each gene). A first thought might be to use a Poisson distribution to model the counts. However, the Poisson makes a strict mean-variance assumption (i.e. they are the same). Studies have demonstrated that a negative binomial fits data better.

Problem of overdispersion:

Many studies have shown that the variance grows faster than the mean in RNAseq data. This is known as **overdispersion**.



Software use:
Negative binomial distribution
Non-parametric methods
Voom transformation

- Mean count vs variance of RNA seq data. Orange line: the fitted observed curve. Purple: the variance implied by the Poisson distribution.

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

Batch effects

Batch Effect: Non-biological variation due to differences in batches of data that confound the relationships between covariates of interest.

Batch effects are caused by differences in:

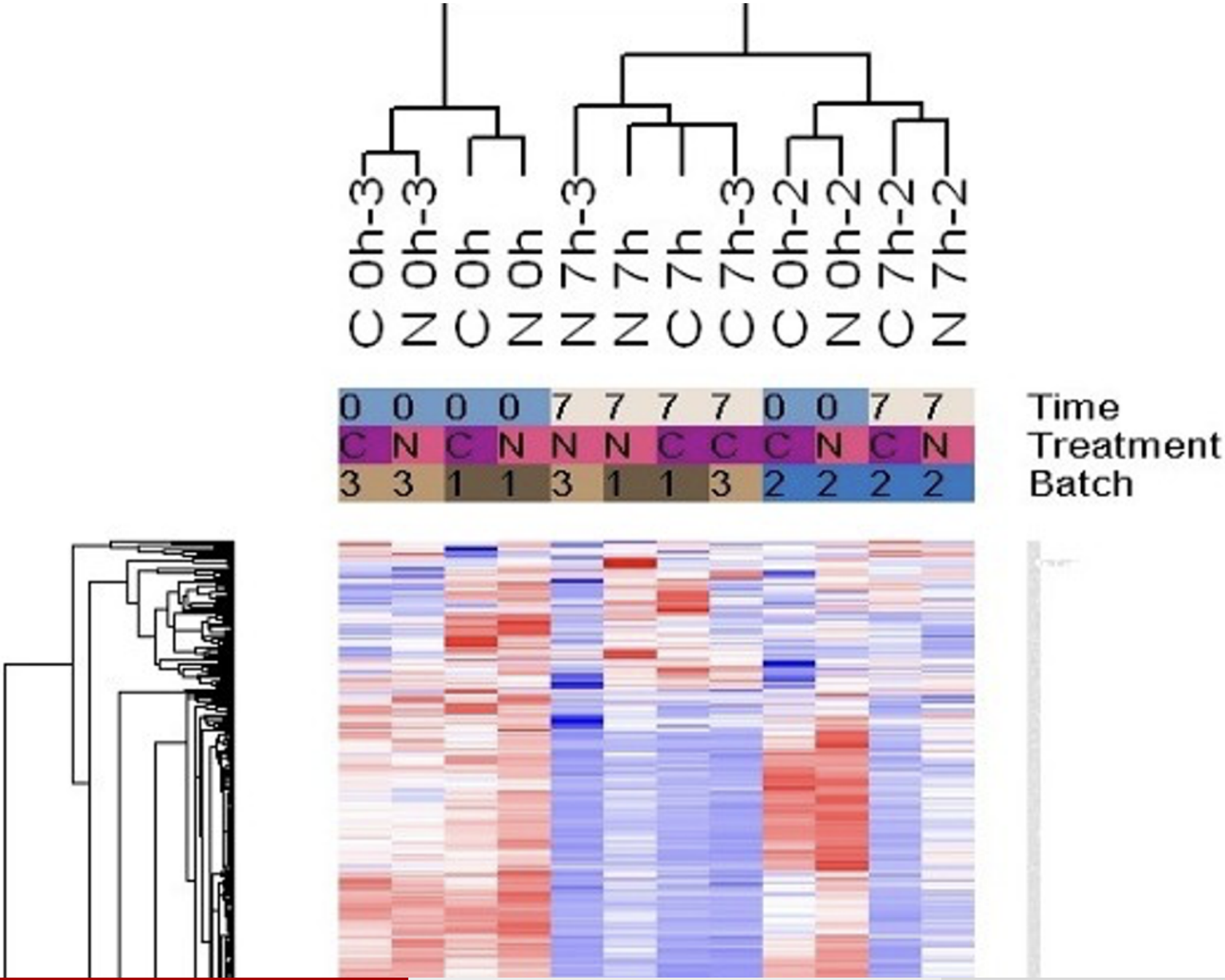
- Gene expression profiling platform
- Lab protocol or experimenter
- Time of day or processing
- Atmospheric ozone level (Rhodes et al. 2004)

Batch Effect Example #1: Nirtic Oxide

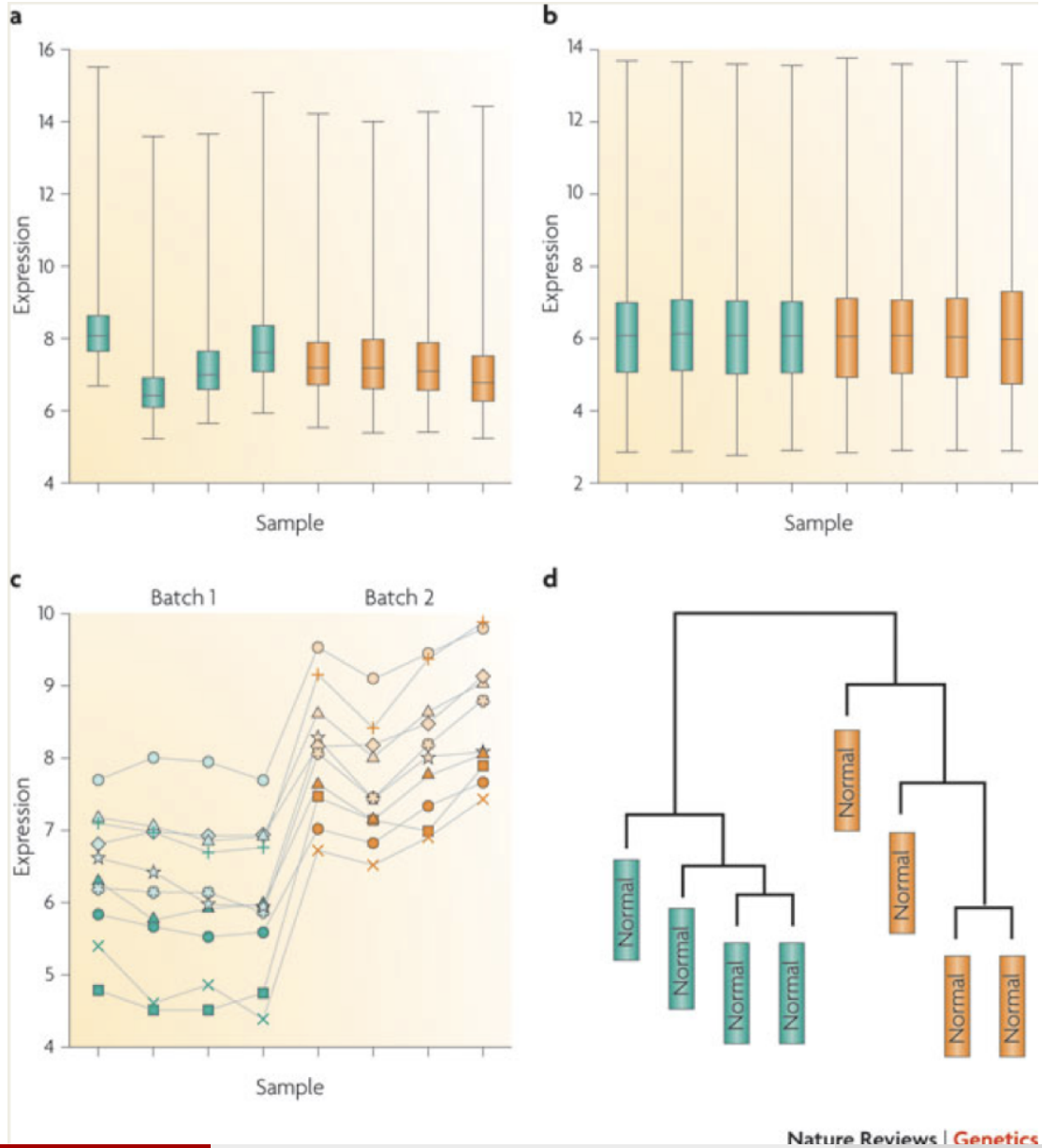
This Example is an oligonucleotide microarray (Affymetrix HG-U133A) experiment on human lung fibroblast cells (IMR90) designed to reveal whether exposing mammalian cells to nitric oxide (NO) stabilizes mRNAs.

Microarray data were collected at baseline (0 h, just before transcription inhibition) and at the end of the experiment (after 7.5 h) for both the control and the NO-treated group.

Batch Effect Example #1: Nirtic Oxide



Batch Effect Example #2: Control Gene Expression



Batch Effect Example #3: Proteomic markers

Proteomic markers to predict endometriosis (39 total):

Single peptide predictors of disease (AUC): 0.82, 0.76, 0.74, 0.74, 0.70 (+12 more >0.6)

Single peptide predictors of batch (AUC): 0.99, 0.94, 0.91, 0.86, 0.86, 0.84, 0.84, 0.84, 0.83, 0.82 (+7 more >0.6)

Predict batch better than disease!

ComBat Batch Adjustment

Consider the following model:

$$Y_{ijg} = \alpha_g + X\beta_g + \gamma_{ig} + \delta_{ig}\epsilon_{ijg}$$

where:

- α_g is the overall gene expression
- X is a design matrix
- β_g contains the regression coefficients
- The error terms $\epsilon_{ijg} \sim N(0, \sigma_g^2)$
- γ_{ig} and δ_{ig} are additive and multiplicative batch effects

ComBat Batch Adjustment

Adjust for batch effects:

$$Y^*_{ijg} = \frac{Y_{ig} - \hat{\alpha}_g - X\hat{\beta}_g - \hat{\gamma}_{ig}}{\hat{\delta}_{ig}} + \hat{\alpha}_g + X\hat{\beta}_g$$

Answer: Empirical Bayes!

BatchQC Example

We can use BatchQC to evaluate and correct for batch effects:

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

library(BatchQC)
BatchQC()
BiocManager::install("BatchQC")
```

Visualization and Dimension reduction

Using an example dataset from: Verma, et al., 2018

```
## read in data
counts <- read.table(
  "rna_seq/downstream_analysis/features_combined.txt",
  sep="\t", header=T, row.names=1)
meta_data <- read.table(
  "rna_seq/downstream_analysis/meta_data.txt",
  sep="\t", header=T, row.names=1)
group <- meta_data$Disease
```

Visualization and dimension reduction

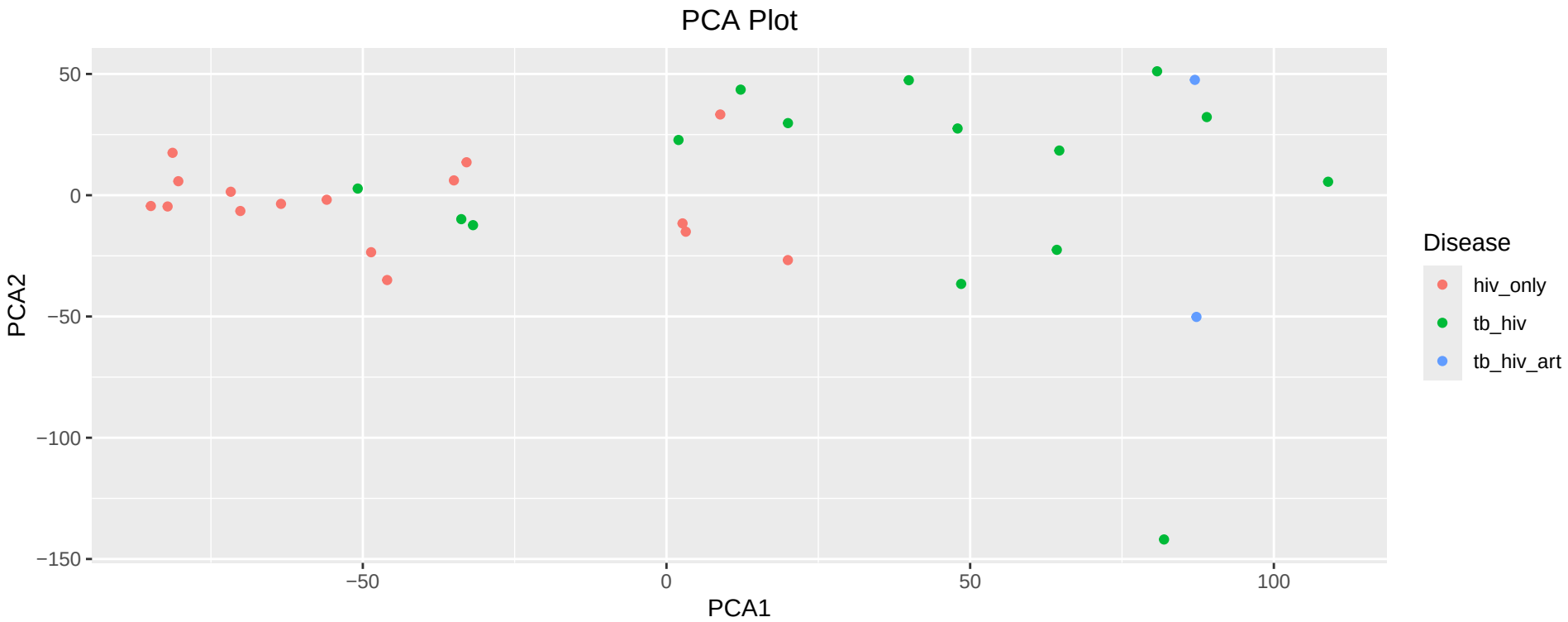
```
## Make SummarizedExperiemnt
sce_hivtb <- SummarizedExperiment(assays=list(counts=counts),
                        colData = meta_data)

## Make log counts, counts per million (cpm), logcpm
sce_hivtb <- mkAssay(sce_hivtb, log = TRUE,
                        counts_to_CPM = TRUE)
assays(sce_hivtb)
```

```
## List of length 4
## names(4): counts log_counts counts_cpm log_counts_cpm
```

Principal Components Analysis (PCA)

```
set.seed(1)
pca_out <- prcomp(t(assay(sce_hivtb,"log_counts_cpm")))

pca_plot <- as.data.frame(pca_out$x)
pca_plot$Disease <- as.factor(sce_hivtb$Disease)

g <- pca_plot %>% ggplot(aes(x=PC1, y=PC2, color=Disease)) +
  geom_point(size=1.5) + xlab("PCA1") + ylab("PCA2") +
  theme(plot.title = element_text(hjust = 0.5)) +
  ggtitle("PCA Plot")

plot(g)
```

Principal Components Analysis (PCA)



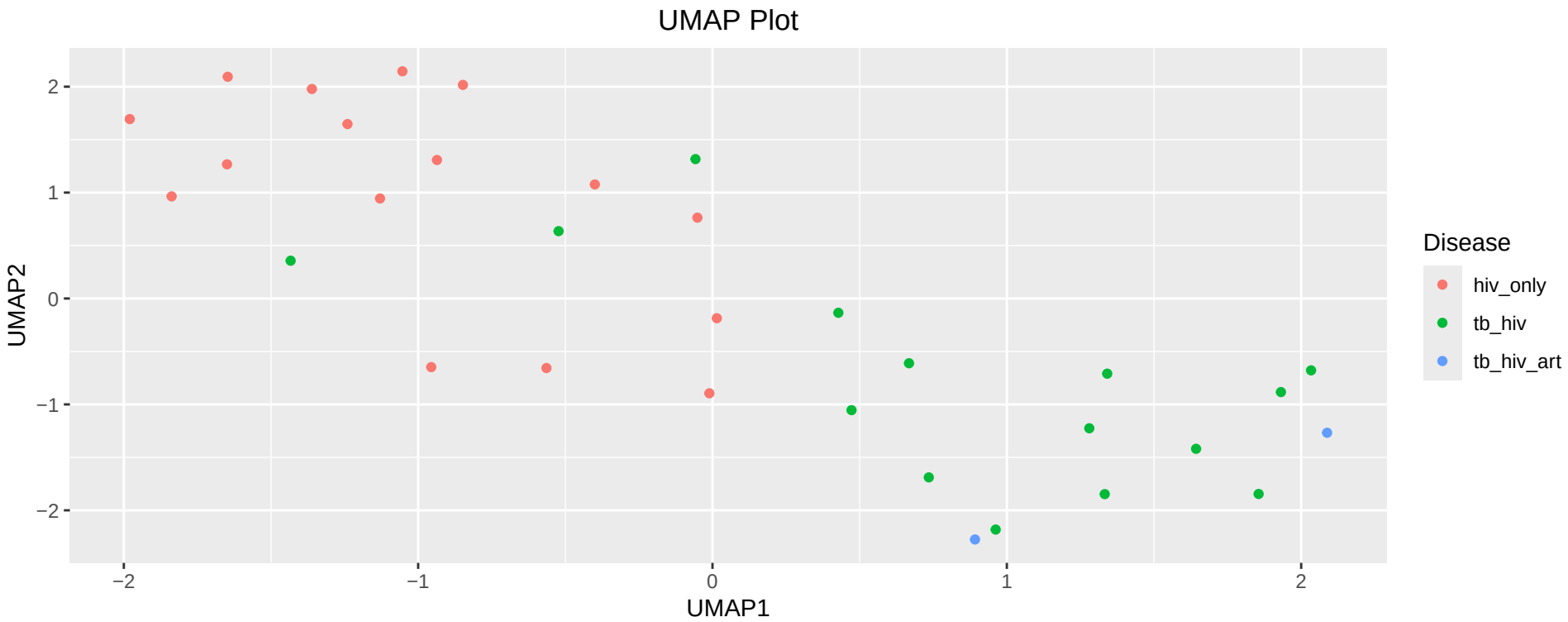
Uniform Manifold Approximation and Projection (UMAP)

For more on UMAP, please visit the following excellent tutorial: https://pair-code.github.io/understanding-umap/

Uniform Manifold Approximation and Projection (UMAP)

```
set.seed(1)
umap_out <- umap(t(assay(sce_hivtb,"log_counts_cpm")))

umap_plot <- as.data.frame(umap_out$layout)
umap_plot$Disease <- as.factor(sce_hivtb$Disease)

g <- umap_plot %>% ggplot(aes(x=V1, y=V2, color=Disease)) +
  geom_point(size=1.5) + xlab("UMAP1") + ylab("UMAP2") +
  theme(plot.title = element_text(hjust = 0.5)) +
  ggtitle("UMAP Plot")

plot(g)
```

Uniform Manifold Approximation and Projection (UMAP)



Differential Expression

Table I: Software packages for detecting differential expression

Method	Version	Reference	Normalization[a]	Read count distribution assumption	Differential expression test
edgeR	3.0.8	[4]	TMM/Upper quartile/RLE (DESeq-like)/None (all scaling factors are set to be one)	Negative binomial distribution	Exact test
DESeq	1.10.1	[5]	DESeq sizeFactors	Negative binomial distribution	Exact test
baySeq	1.12.0	[6]	Scaling factors (quantile/TMM/total)	Negative binomial distribution	Assesses the posterior probabilities of models for differentially and non-differentially expressed genes via empirical Bayesian methods and then compares these posterior likelihoods
NOIseq	1.1.4	[7]	RPKM/TMM/Upper quartile	Nonparametric method	Contrasts fold changes and absolute differences within a condition to determine the null distribution and then compares the observed differences to this null
SAMseq (samr)	2.0	[8]	SAMseq specialized method based on the mean read count over the null features of the data set	Nonparametric method	Wilcoxon rank statistic and a resampling strategy
Limma	3.14.4	[9]	TMM	voom transformation of counts	Empirical Bayes method
Cuffdiff 2 (Cufflinks)	2.0.2-beta	[10]	Geometric (DESeq-like)/quartile/classic-fpkm	Beta negative binomial distribution	*t*-test
EBSeq	1.1.7	[11]	DESeq median normalization	Negative binomial distribution	Evaluates the posterior probability of differentially and non-differentially expressed entities (genes or isoforms) via empirical Bayesian methods

[a]In case of availability of several normalization methods, the default one is underlined.

EdgeR Example

Implements statistical methods for DE analysis based on the negative binomial model:

```
#Gene Filtering
counts<-counts[which(rowSums(cpm(counts))>1),]
#Computes library size
dge <- DGEList(counts=counts, group=group)
#TMM normalization
dge <- calcNormFactors(dge)
# Design matrix
design<-model.matrix(~group)
#Estimates common, trended and tagwise dispersion
dge<-estimateDisp(counts,design)
```

EdgeR Example

In negative binomial models, each gene is given a dispersion parameter. Dispersions control the variances of the gene counts and underestimation will lead to false discovery and overestimation may lead to a lower rate of true discovery.

EdgeR Example

```
# Neg Bin GLM with the dispersion estimates
fit<-glmFit(counts,design,
            dispersion=dge$tagwise.dispersion)
# Performs likelihood ratio test
# Compares full versus reduced model
lrt<-glmLRT(fit, coef=2)
```

EdgeR Example

```
# Prints the top results
topTags(lrt)
```

```
## Coefficient:  grouptb_hiv
##               logFC   logCPM        LR       PValue          FDR
## IL1R2      4.334196 8.207931 100.01344 1.513665e-23 2.933634e-19
## AP3B2      5.758193 2.952770  71.58586 2.654527e-17 2.572370e-13
## FCGR1C     2.818498 4.536149  65.07230 7.220003e-16 4.664362e-12
## VNN1       3.150158 8.071776  64.39089 1.020287e-15 4.943545e-12
## CYP1B1     3.135471 6.873000  63.00698 2.059751e-15 7.984006e-12
## IL18R1     2.726131 6.487474  60.75863 6.451968e-15 2.084093e-11
## SLC29A1 -4.135274 3.969739  59.69166 1.109455e-14 3.071764e-11
## CACNG8  -4.134373 3.189823  58.97316 1.598376e-14 3.872266e-11
## SOCS3      2.665230 6.475922  57.71505 3.029759e-14 6.524418e-11
## ZAK        2.601746 6.126750  56.08435 6.942749e-14 1.345574e-10
```

EdgeR Example

```
# Perform quasi-likelihood F-tests
## Replace the chisquare approximation to the likelihood
## ratio statistic with a quasi-likelihood F-test,
## more control of error rate
fit<-glmQLFit(counts, design,
        dispersion=dge$tagwise.dispersion)
## use for small dataset, uncertainty in estimating
## control when number of replicates is small
## dispersion for each gene, more robust and reliable
qlf<-glmQLFTest(fit, coef=2)
```

EdgeR Example

```
# Prints the top results
topTags(qlf)
```

```
## Coefficient:  grouptb_hiv
##              logFC   logCPM         F       PValue          FDR
## IL1R2     4.334269 8.207931 104.44957 1.620179e-24 3.140069e-20
## AP3B2     5.765316 2.952770  74.47435 6.158177e-18 5.967581e-14
## VNN1      3.150194 8.071776  67.12438 2.554151e-16 1.527305e-12
## FCGR1C    2.818735 4.536149  66.70962 3.152170e-16 1.527305e-12
## CYP1B1    3.135368 6.873000  64.83799 8.146782e-16 3.157856e-12
## IL18R1    2.726108 6.487474  62.35723 2.869726e-15 9.269692e-12
## SLC29A1 -4.134814 3.969739  61.45697 4.532840e-15 1.239518e-11
## CACNG8  -4.134063 3.189823  61.21850 5.116427e-15 1.239518e-11
## SOCS3     2.665271 6.475922  59.23241 1.403255e-14 3.021831e-11
## ZAK       2.601744 6.126750  57.45967 3.454883e-14 6.695908e-11
```

EdgeR Example

```
#For visualization, heatmaps/PCA
Logcpm<-cpm(counts,log=TRUE)
```

DESeq2 Example

```
#colData is a data frame of demographic/phenotypic data
dds <- DESeqDataSetFromMatrix(countData = counts,
                              colData=meta_data,
                              design=~Disease)
#Gene Filtering
dds<-dds[rowSums(counts(dds))>1,]
```

DESeq2 Example

```
#Performs estimation of size factors,
#dispersion, and negative binomial GLM fitting
dds<-DESeq(dds)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

## -- replacing outliers and refitting for 180 genes
## -- DESeq argument 'minReplicatesForReplace' = 7
## -- original counts are preserved in counts(dds)

## estimating dispersions

## fitting model and testing
```

DESeq2 Example

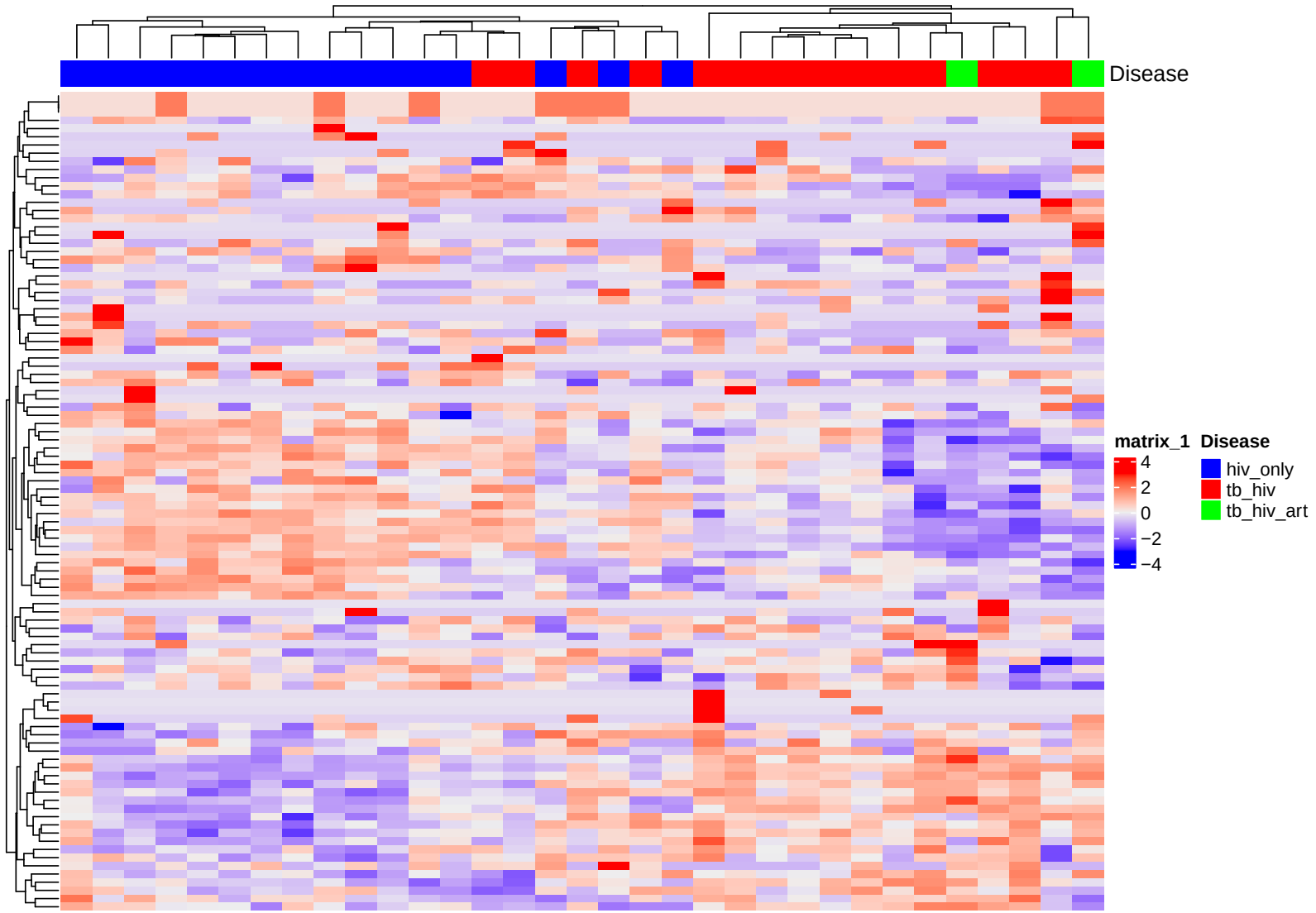
```
res <- results(dds)[order(results(dds)[,6]),]
res[1:10,]
```

```
## log2 fold change (MLE): Disease tb hiv art vs hiv only
## Wald test p-value: Disease tb hiv art vs hiv only
## DataFrame with 10 rows and 6 columns
##          baseMean log2FoldChange     lfcSE      stat      pvalue        padj
##         <numeric>      <numeric> <numeric> <numeric>   <numeric>   <numeric>
## ZEB2    1702.8164        1.75543  0.217662   8.06493 7.32802e-16 1.19996e-11
## DCUN1D3   48.7029        2.52495  0.361657   6.98163 2.91765e-12 2.38883e-08
## TIPARP   724.9429        2.22978  0.334020   6.67558 2.46254e-11 1.34414e-07
## ITPKC    196.8723        3.04144  0.465638   6.53177 6.49981e-11 2.66086e-07
## LAIR1   1528.6852        3.01392  0.472158   6.38328 1.73337e-10 4.40512e-07
## COPS4    559.7527        2.67612  0.419911   6.37306 1.85294e-10 4.40512e-07
## IGF2BP3  360.1066        3.29761  0.517782   6.36873 1.90604e-10 4.40512e-07
## GSAP     964.2702        1.26506  0.199220   6.35007 2.15212e-10 4.40512e-07
## RBMS1   3297.7791        2.51612  0.400420   6.28370 3.30604e-10 6.01516e-07
## ZDHHC20  554.9810        2.51989  0.403490   6.24523 4.23175e-10 6.92949e-07
```

Heatmap of DEGs

```
# Make a Heatmap of DEGs
mat = as.matrix(assay(sce_hivtb,"log_counts_cpm")
                )[order(results(dds)[,6])[1:100],]
                # Using first 1000 genes to simplify
mat = t(scale(t(mat)))
df=data.frame(Disease=colData(sce_hivtb)$Disease)
ha = HeatmapAnnotation(df = df,
                       col = list(Disease=c(
                         "tb_hiv"="Red",
                         "hiv_only"="Blue",
                         "tb_hiv_art"="Green")))
Heatmap(mat,show_row_names=F, show_column_names = F,
        top_annotation = ha)
```

Heatmap of DEGs



Limma Example

- Most similar to microarray data flow
- Reads counts are converted to log2 counts per million (logCPM) and the mean-variance relationship is modeled with precision weights (voom transform)

Limma Example

```
#From edgeR, Computes library size
dge <- DGEList(counts=counts, group=group)
#Gene Filtering
counts<-counts[which(rowSums(cpm(counts))>1),]
dge <- DGEList(counts=counts, group=group)
dge <- calcNormFactors(dge) #TMM normalization
```

Limma Example

```
design<-model.matrix(~group)
#voom transform to calculate weights to
#eliminate mean-variance relationship
v<-voom(dge, design)
#use usual limma pipelines
fit<-lmFit(v,design)
fit<-eBayes(fit)
```

Limma Example

```
topTable(fit, coef=ncol(design))
```

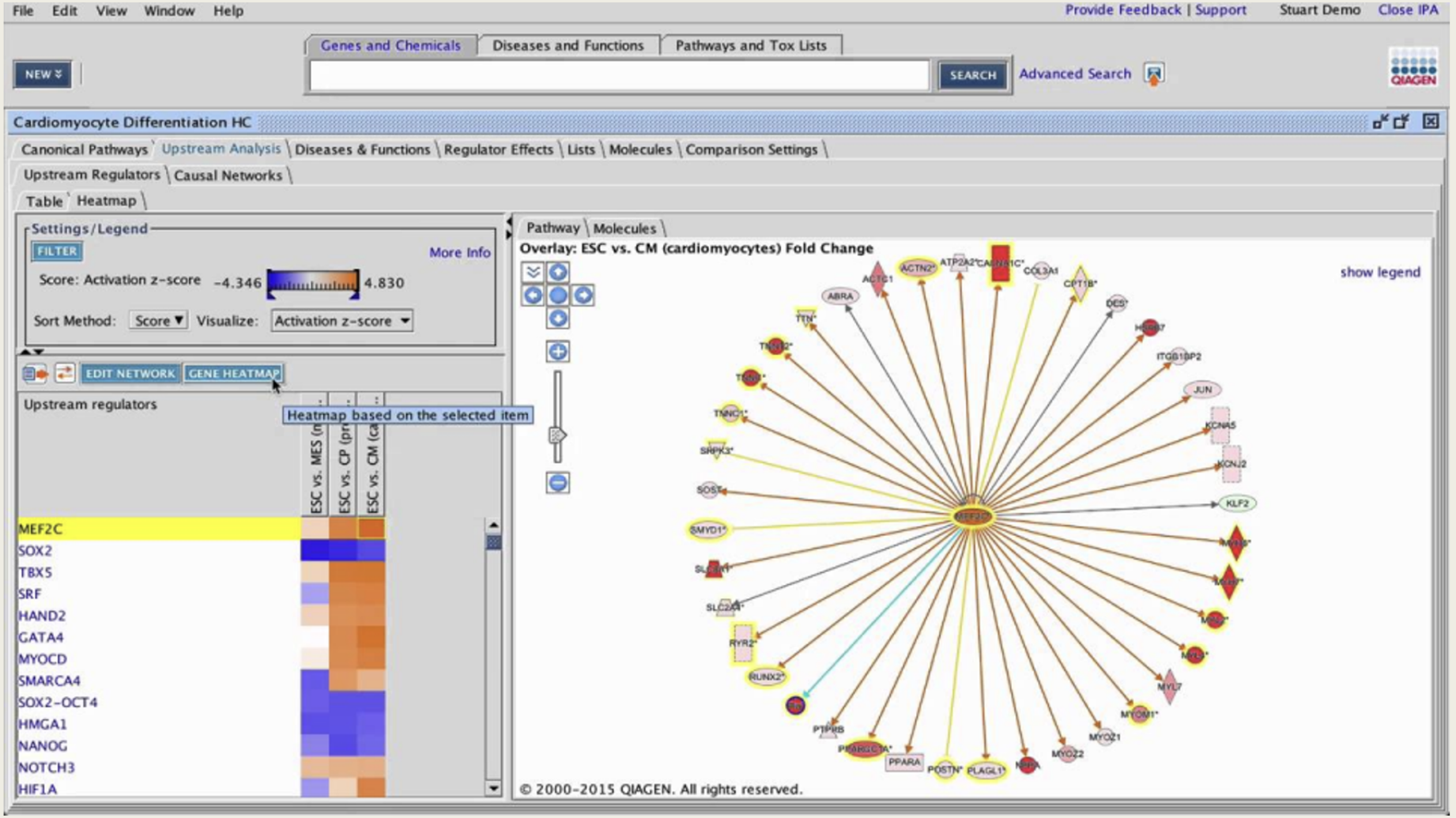
```
##                logFC    AveExpr        t      P.Value    adj.P.Val        B
## DCUN1D3     2.415048  1.1213257 7.597067 5.330234e-09 5.174623e-05 10.335712
## ZEB2        1.681109  6.4208329 7.439069 8.537472e-09 5.174623e-05 10.090462
## LINC01093   5.474634  0.8710200 7.414057 9.200542e-09 5.174623e-05  9.932749
## FAM151B     3.709561  1.2894208 7.364262 1.067978e-08 5.174623e-05  9.825077
## C7orf61     2.599925  2.7347797 7.206753 1.714091e-08 5.536801e-05  9.443534
## CYP19A1     6.863347 -0.9439843 7.234226 1.578062e-08 5.536801e-05  9.241797
## TIPARP      2.118756  5.0816090 6.625677 9.994987e-08 1.885771e-04  7.773090
## IGF2BP3     3.336976  3.6439397 6.618279 1.022347e-07 1.885771e-04  7.751390
## COL4A2-AS1 5.463005 -3.3142037 6.975753 3.444177e-08 9.535941e-05  7.614380
## LAYN        3.342889 -0.9931241 6.639498 9.581743e-08 1.885771e-04  7.409620
```

Pathway analysis

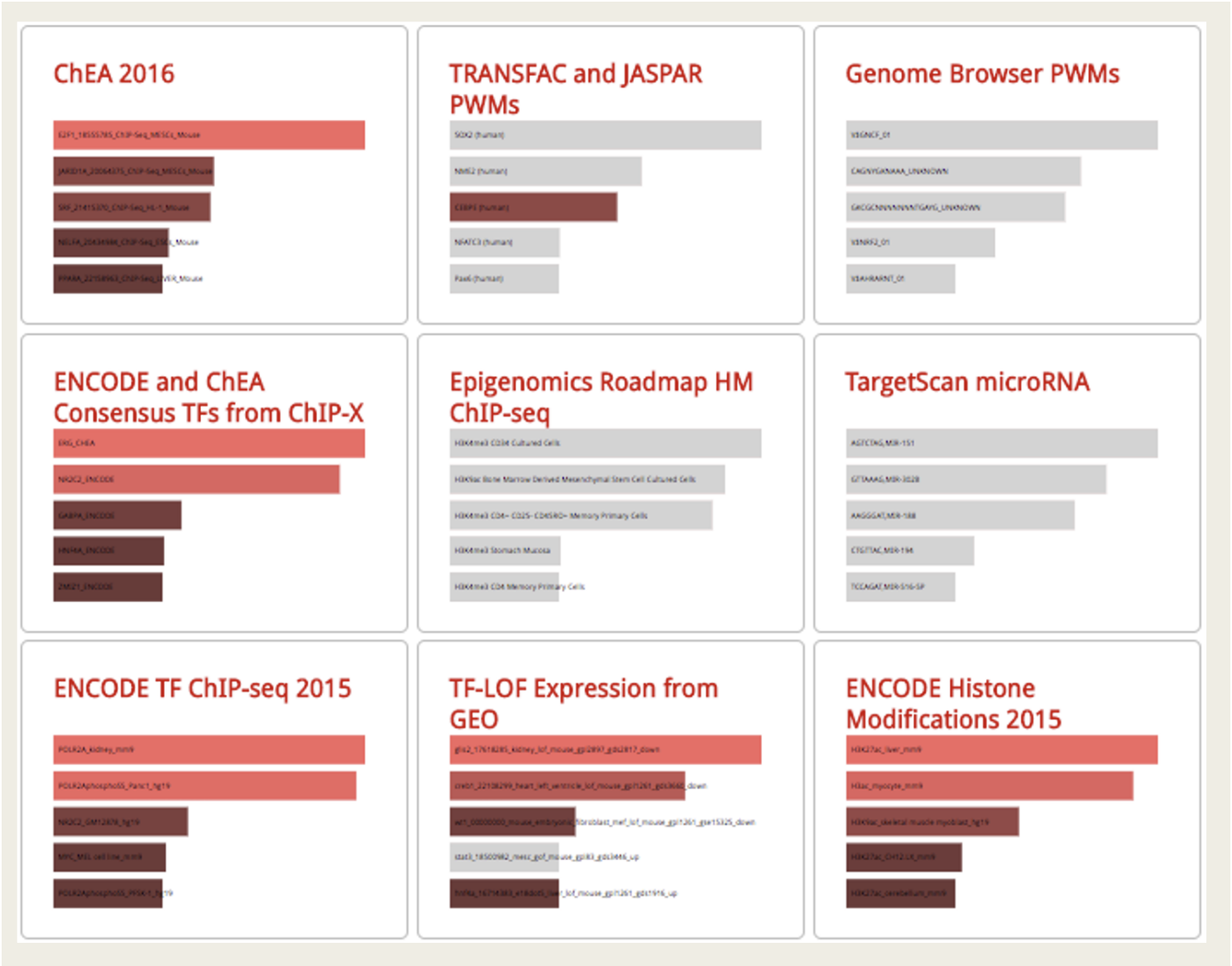
After finding DEGs, look for correlated genes/networks and enriched pathway sets in the gene set using:

- Weighted gene coexpression network analysis (WGCNA)
- GSEA, GSVA, EnrichR, many more!!
- Qiagen Ingenuity Pathway Analysis (IPA)

Pathway analysis



Pathway analysis



TBSignatureProfiler Analysis

The TBSignatureProfiler was developed in the Johnson Lab in 2021 to profile new and existing TB gene expression signatures:

https://bmcinfectdis.biomedcentral.com/articles/10.1186/s12879-020-05598-z

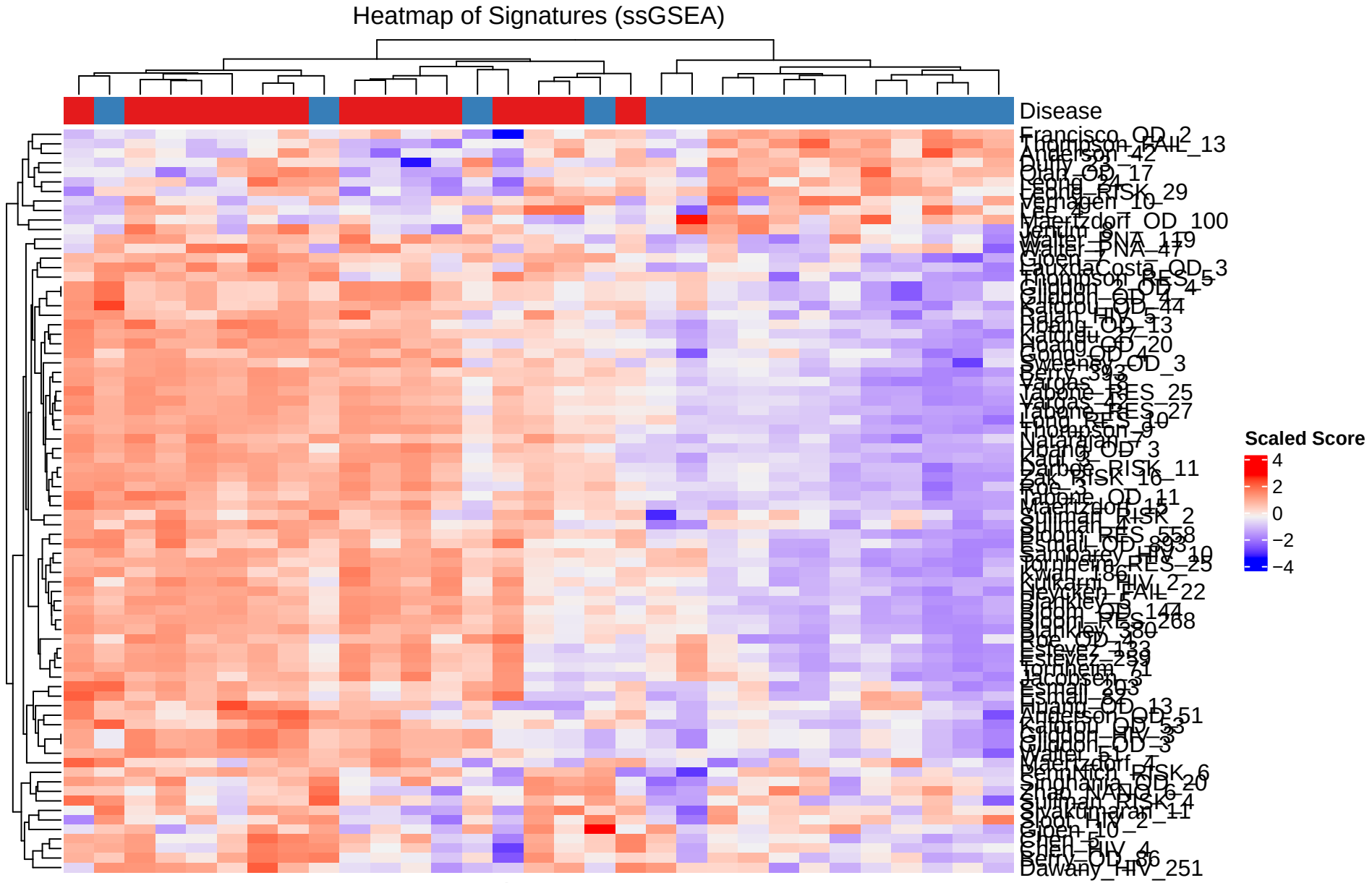
TBSignatureProfiler Analysis

```
sce_hivtb_2 <- sce_hivtb[,
      colData(sce_hivtb)$Disease != "tb_hiv_art"]
TBsigs <- TBsignatures[-12]
ssgsea_res <- runTBsigProfiler(sce_hivtb_2,
                    useAssay = "log_counts_cpm",
                    signatures = TBsigs,
                    algorithm = "ssGSEA",
                    combineSigAndAlgorithm = TRUE,
                    parallel.sz = 1)
```

Signature Heatmap:

```
# Colors for gradient
signatureHeatmap(ssgsea_res,
        name = "Heatmap of Signatures (ssGSEA)",
        signatureColNames = names(TBsigs),
        annotationColNames = c("Disease"),
        scale = TRUE,
        split_heatmap = "none",
        showColumnNames = FALSE)
```

Signature Heatmap:



Signature Boxplots

```
signatureBoxplot(ssgsea_res, name="ssGSEA",
                 signatureColNames = names(TBsigs)[1:4],
                 annotationColName = c("Disease"))
```