Analysis of single cell RNA-seq data

Vladimir Kiselev, Tallulah Andrews, Davis McCarthy and Martin Hemberg

2016-12-14

Contents

Chapter 1

About the course

Recent technological advances have made it possible to obtain genome-wide transcriptome data from single cells using high-throughput sequencing (scRNA-seq). The main advantage of scRNA-seq is that the cellular resolution and the genome wide scope makes it possible to address issues that are intractable using other methods, e.g. bulk RNA-seq or single-cell RT-qPCR. However, to analyze scRNA-seq data, novel methods are required and some of the underlying assumptions for the methods developed for bulk RNA-seq experiments are no longer valid.

In this course we will be surveying the existing problems as well as the available computational and statistical frameworks available for the analysis of scRNA-seq. The course is taught through the University of Cambridge Bioinformatics training unit, but the material found on these pages is meant to be used for anyone interested in learning about computational analysis of scRNA-seq data.

1.1 Video

1.2 Registration

Please follow this link and register for the **"Analysis of single cell RNA-seq data"** course: http://training.csx.cam.ac.uk/bioinformatics/search

1.3 GitHub

https://github.com/hemberg-lab/scRNA.seq.course

1.4 License

1.5 Prerequisites

The course is intended for those who have basic familiarity with Unix and the R scripting language.

We will also assume that you are familiar with mapping and analysing bulk RNA-seq data as well as with the commonly available computational tools.

We recommend attending the Introduction to RNA-seq and ChIP-seq data analysis or the Analysis of high-throughput sequencing data with Bioconductor before attending this course.

1.6 Contact

If you have any **comments**, **questions** or **suggestions** about the material, please contact Vladimir Kiselev.

Chapter 2

Technical requirements

This course is based on the popular programming language R. However, one of the methods that we describe (SNN-Cliq) is only partly R-based. It makes a simple *python* call from R and requires a user to have write permissions to the working directory.

To be able to run all code chunks of the course one needs to clone or download the course GitHub repository and start an R session in the cloned folder.

One also needs to install the following R packages (ordered by purposes):

2.1 General

devtools for installing packages from GitHub:

```
install.packages("devtools")
```

BiocInstaller for installing packages from BioConductor:

```
source('https://bioconductor.org/biocLite.R')
biocLite('BiocInstaller')
```

scRNA.seq.funcs - R package containing some special functions used in this course:

```
devtools::install_github("hemberg-lab/scRNA.seq.funcs")
```

2.2 Plotting

ggplot2 for plotting general plots:

```
install.packages("ggplot2")
```

pheatmap for plotting heatmaps:

```
install.packages("pheatmap")
```

limma for plotting Venn diagrams:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("limma")
```

2.3 QC and normalisation

scater is a single-cell analysis toolkit for expression:

```
source('https://bioconductor.org/biocLite.R')
biocLite('scater')
```

mvoutlier - for an automatic outlier detection used by the scater package.

```
install.packages("mvoutlier")
```

statmod - a dependency for mvoutlier.

```
install.packages("statmod")
```

Rtsne for Rtsne data embedding:

```
install.packages("Rtsne")
```

scran for a new single cell normalisation method (LSF):

```
source('https://bioconductor.org/biocLite.R')
biocLite('scran')
```

RUVSeq for normalization using ERCC controls:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("RUVSeq")
```

2.4 Clustering

pcaReduce for unsupervised clustering of scRNA-seq data:

```
devtools::install_github("JustinaZ/pcaReduce")
```

pcaMethods is a pcaReduce dependency:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("pcaMethods")
```

SC3 for unsupervised clustering of scRNA-seq data:

```
source("https://bioconductor.org/biocLite.R")
biocLite("SC3")
```

SEURAT for density clustering of scRNA-seq data (at the moment we are using an old version of this package - v. 1.3):

```
devtools::install_github('satijalab/seurat', ref = 'da6cd08')
```

2.5 Dropouts

M3Drop for identification of important and DE genes:

```
devtools::install_github("tallulandrews/M3D")
```

2.6 Pseudotime

TSCAN for pseudotime analysis:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("TSCAN")
```

monocle for pseudotime analysis:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("monocle")
```

destiny for pseudotime analysis:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("destiny")
```

2.7 Differential Expression

ROCR for performance estimations:

```
install.packages("ROCR")
```

edgeR for identification of differentially expressed genes:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("edgeR")
```

DESeq2 for identification of differentially expressed genes:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("DESeq2")
```

scde for identification of differentially expressed genes:

```
devtools::install_github("hms-dbmi/scde", build_vignettes = FALSE)
```

- Installation on Mac OS X may require this additional gfortran library:

  ```
  curl -O http://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2
  sudo tar fvxz gfortran-4.8.2-darwin13.tar.bz2 -C /
  ```

- See the help page for additional support.

2.8 Extra tools

MultiAssayExperiment for working with conquer datasets:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("MultiAssayExperiment")
```

SummarizedExperiment for working with conquer datasets:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("SummarizedExperiment")
```

Chapter 3

Introduction to single-cell RNA-seq

3.1 Bulk RNA-seq

- A major breakthrough (replaced microarrays) in the late 00's and has been widely used since
- Measures the **average expression level** for each gene across a large population of input cells
- Useful for comparative transcriptomics, e.g. samples of the same tissue from different species
- Useful for quantifying expression signatures from ensembles, e.g. in disease studies
- **Insufficient** for studying heterogeneous systems, e.g. early development studies, complex tissues (brain)
- Does **not** provide insights into the stochastic nature of gene expression

3.2 scRNA-seq

- A **new** technology, first publication by (Tang et al., 2009)
- Did not gain widespread popularity until ~2014 when new protocols and lower sequencing costs made it more accessible
- Measures the **distribution of expression levels** for each gene across a population of cells
- Allows to study new biological questions in which **cell-specific changes in transcriptome are important**, e.g. cell type identification, heterogeneity of cell responses, stochasticity of gene expression, inference of gene regulatory networks across the cells.
- Datasets range **from** 10^2 **to** 10^5 **cells** and increase in size every year
- Currently there are several different protocols in use, e.g. SMART-seq2 (Picelli et al., 2013), CELL-seq (Hashimshony et al., 2012) and Drop-seq (Macosko et al., 2015)
- Several computational analysis methods from bulk RNA-seq **can** be used
- **In most cases** computational analysis requires adaptation of the existing methods or development of new ones

3.3 Protocol

Overall, experimental scRNA-seq protocols are similar to the methods used for bulk RNA-seq. For a discussion on experimental methods, please see reviews by (Saliba et al., 2014), (Handley et al., 2015) or (Kolodziejczyk et al., 2015).

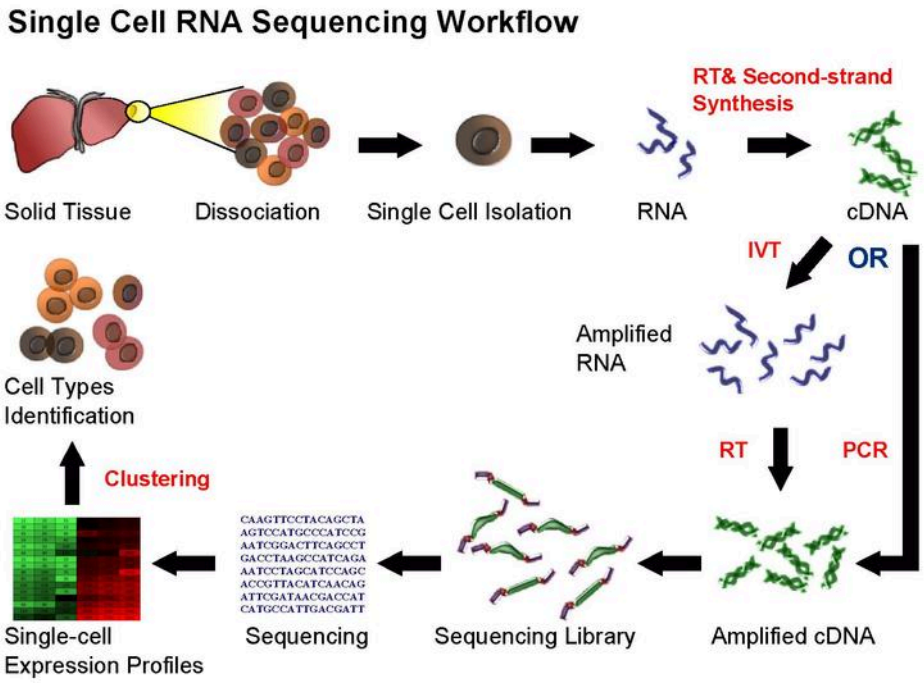


Figure 3.1: Single cell sequencing (taken from Wikipedia)

3.4 Computational Analysis

This course is concerned with the computational analysis of the data obtained from scRNA-seq experiments. The first steps (yellow) are general for any highthroughput sequencing data. Later steps (orange) require a mix of existing RNASeq analysis methods and novel methods to address the technical difference of scRNASeq. Finally the biological interpretation **should** be analyzed with methods specifically developed for scRNASeq.

There are several reviews of the scRNA-seq analysis available including (Stegle et al., 2015).

3.5 Challenges

The main difference between bulk and single cell RNA-seq is that each sequencing library represents a single cell, instead of a population of cells. Therefore, significant attention has to be paid to comparison of the results from different cells (sequencing libraries). The main sources of discrepancy between the libraries are:

- **Amplification** (up to 1 million fold)
- **Gene 'dropouts'** in which a gene is observed at a moderate expression level in one cell but is not detected in another cell (Kharchenko et al., 2014).

In both cases the discrepancies are introduced due to low starting amounts of transcripts since the RNA comes from one cell only. Improving the transcript capture efficiency and reducing the amplification bias are currently active areas of research.

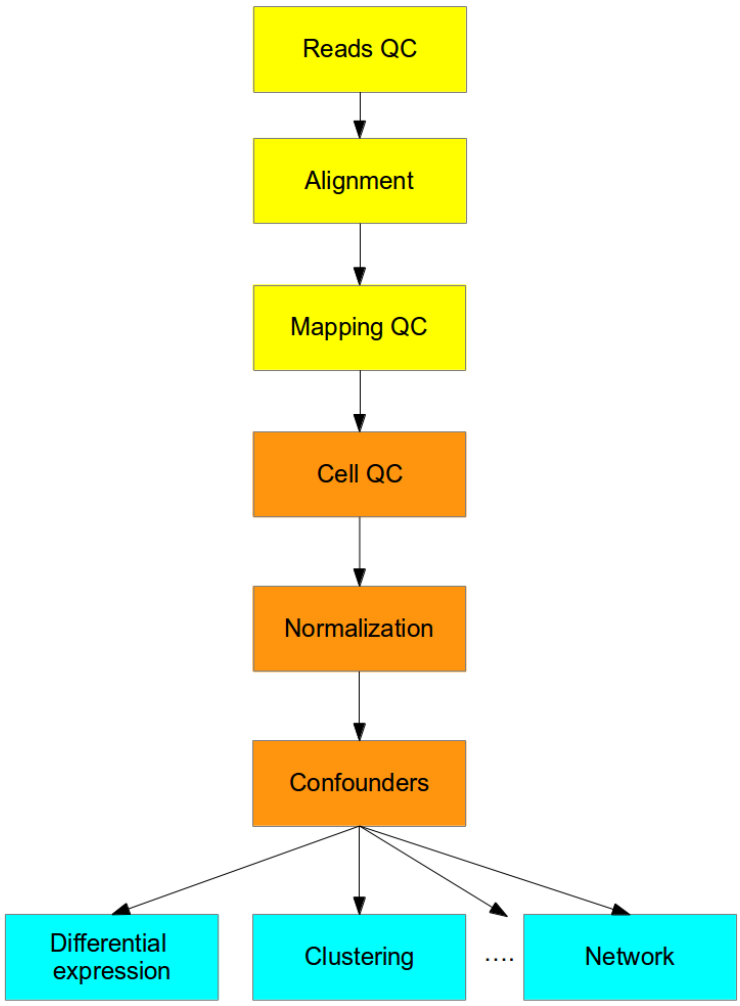


Figure 3.2: Flowchart of the scRNA-seq analysis

3.6 Controls

To provide better estimates of the technical variation between scRNA sequencing libraries two quantitative standards are frequently used. The aim of using spike-ins and UMIs is to facilitate normalization of gene expression levels across different cells.

3.6.1 Spike-ins

Spike-ins are extrinsic RNA molecules of known concentration which are added to the lysate of each cell prior to the reverse transcription reaction. The most popular and widely used spike-ins are synthetic spikes from the External RNA Control Consortium (ERCC). This set of 96 synthetic mRNAs of differing length and GC content based on bacterial sequences (Jiang et al., 2011).

3.6.2 UMIs

Another method of standardisation is to use Unique Molecular Identifiers (UMIs) (Kivioja et al., 2012). These are 4-20 bp barcode sequences which are added to the 3' or 5' end of each transcript prior to amplification (typically during reverse transcription). This is usually followed by targetted sequencing of the respective end of the transcripts. The barcodes make it possible to quantify the number of transcripts prior to the amplification step.

Chapter 4

Construction of expression matrix

4.1 Reads QC

The output from a scRNA-seq experiment is a large collection of cDNA reads. The first step is to ensure that the reads are of high quality. The quality control can be performed by using standard tools, such as FastQC or Kraken.

Assuming that our reads are in experiment.bam, we run FastQC as

```
$<path_to_fastQC>/fastQC experiment.bam
```

Below is an example of the output from FastQC for a dataset of 125 bp reads. The plot reveals a technical error which resulted in a couple of bases failing to be read correctly in the centre of the read. However, since the rest of the read was of high quality this error will most likely have a negligible effect on mapping efficiency.

Additionally, it is often helpful to visualize the data using the Integrative Genomics Browser (IGV) or SeqMonk.

4.2 Reads alignment

After trimming low quality bases from the reads, the remaining sequences can be mapped to a reference genome. Again, there is no need for a special purpose method for this, so we can use the STAR or the TopHat aligner. For large full-transcript datasets from well annotated organisms (e.g. mouse, human) pseudo-alignment methods (e.g. Kallisto, Salmon) may out-perform conventional alignment.

An example of how to map reads.bam to using STAR is

```
$<path_to_STAR>/STAR --runThreadN 1 --runMode alignReads
--readFilesIn reads1.fq.gz reads2.fq.gz --readFilesCommand zcat --genomeDir <path>
--parametersFiles FileOfMoreParameters.txt --outFileNamePrefix <outpath>/output
```

Note, if the *spike-ins* are used, the reference sequence should be augmented with the DNA sequence of the *spike-in* molecules prior to mapping.

Note, when UMIs are used, their barcodes should be removed from the read sequence. A common practice is to add the barcode to the read name.

Once the reads for each cell have been mapped to the reference genome, we need to make sure that a sufficient number of reads from each cell could be mapped to the reference genome. In our experience, the fraction of mappable reads for mouse or human cells is 60-70%. However, this result may vary depending on protocol,

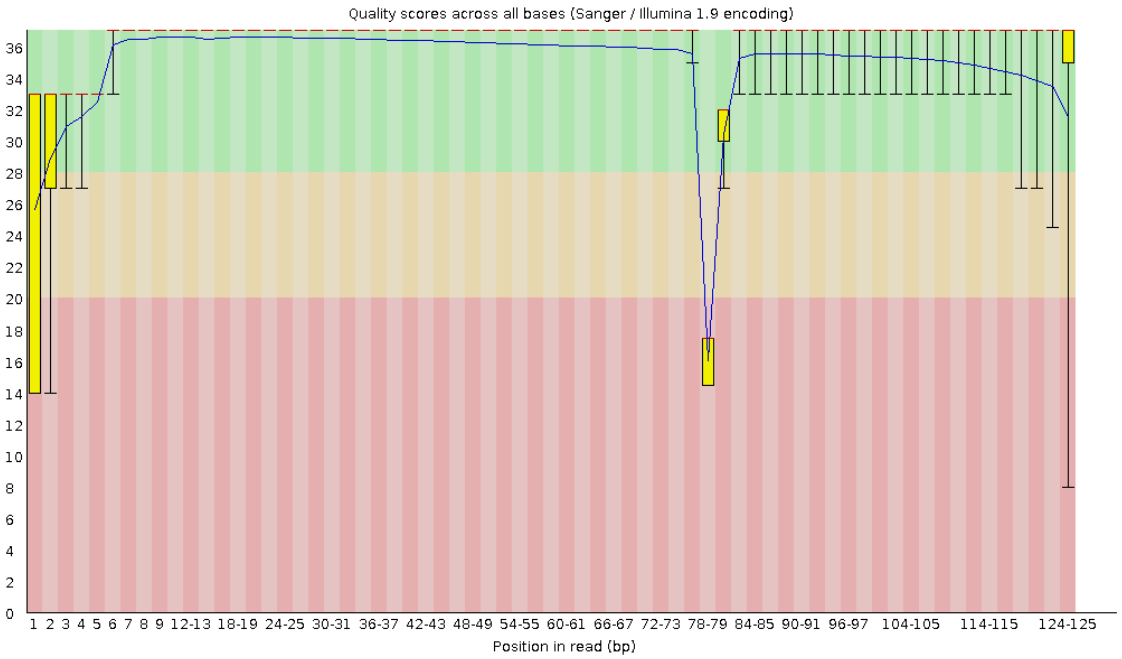


Figure 4.1: Example of FastQC output

read length and settings for the read alignment. As a general rule, we expect all cells to have a similar fraction of mapped reads, so any outliers should be inspected and possibly removed. A low proportion of mappable reads usually indicates contamination.

4.3 Alignment example

The histogram below shows the total number of reads mapped to each cell for an scRNA-seq experiment. Each bar represents one cell, and they have been sorted in ascending order by the total number of reads per cell. The three red arrows indicate cells that are outliers in terms of their coverage and they should be removed from further analysis. The two yellow arrows point to cells with a surprisingly large number of unmapped reads. In this example we kept the cells during the alignment QC step, but they were later removed during cell QC due to a high proportion of ribosomal RNA reads.

4.4 Mapping QC

After mapping the raw sequencing to the genome we need to evaluate the quality of the mapping. There are many ways to measure the mapping quality, including: amount of reads mapping to rRNA/tRNAs, proportion of uniquely mapping reads, reads mapping across splice junctions, read depth along the transcripts. Methods developed for bulk RNA-seq, such as RSeQC, are applicable to single-cell data:

```
python <RSeQCpath>/geneBody_coverage.py -i input.bam -r genome.bed -o output.txt
python <RSeQCpath>/bam_stat.py -i input.bam -r genome.bed -o output.txt
python <RSeQCpath>/split_bam.py -i input.bam -r rRNAmask.bed -o output.txt
```

However the expected results will depend on the experimental protocol, e.g. many scRNA-seq methods use poly-A selection to avoid sequencing rRNAs which results in a 3' bias in the read coverage across the genes

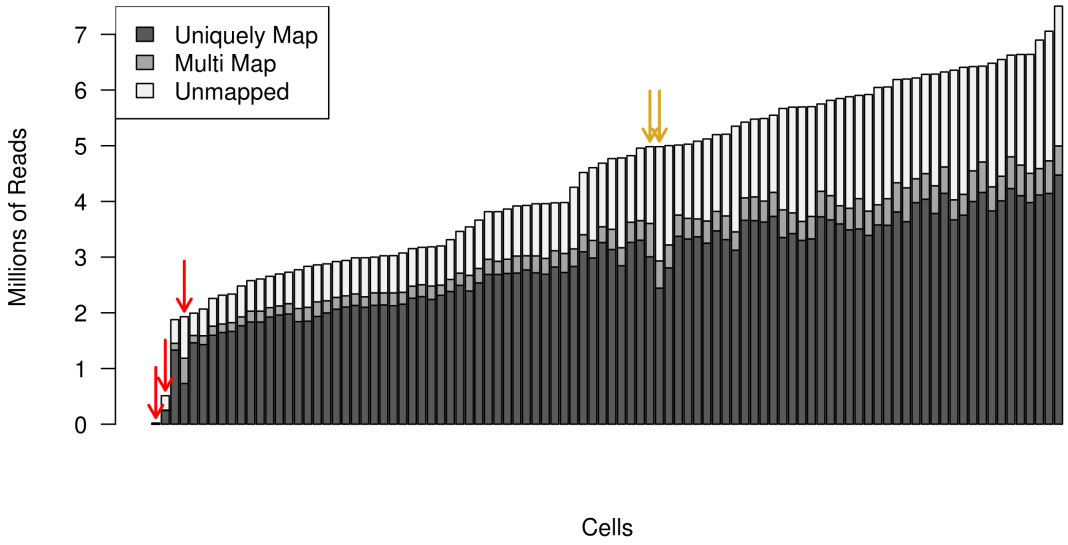


Figure 4.2: Example of the total number of reads mapped to each cell.

(aka gene body coverage). The figure below shows this 3' bias as well as three cells which were outliers and removed from the dataset:

4.5 Reads quantification

The next step is to quantify the expression level of each gene for each cell. For mRNA data, we can use one of the tools which has been developed for bulk RNA-seq data, e.g. HT-seq or FeatureCounts

```
# include multimapping
<featureCounts_path>/featureCounts -O -M -Q 30 -p -a genome.gtf -o outputfile input.bam
# exclude multimapping
<featureCounts_path>/featureCounts -Q 30 -p -a genome.gtf -o outputfile input.bam
```

Unique molecular identifiers (UMIs) make it possible to count the absolute number of molecules and they have proven popular for scRNA-seq. We will discuss how UMIs can be processed in the next chapter.

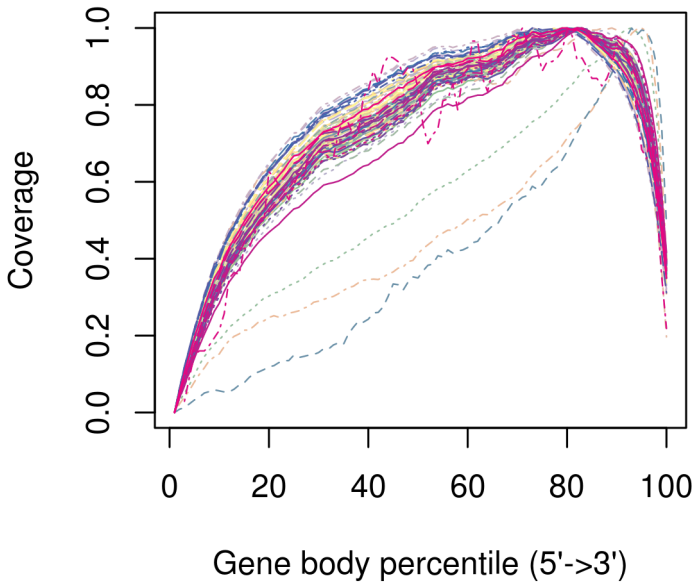


Figure 4.3: Example of the 3' bias in the read coverage.

Chapter 5

Unique Molecular Identifiers (UMIs)

Thanks to Andreas Buness from EMBL Monterotondo for collaboration on this section.

5.1 Introduction

Unique Molecular Identifiers are short (4-10bp) random barcodes added to transcripts during reverse-transcription. They enable sequencing reads to be assigned to individual transcript molecules and thus the removal of amplification noise and biases from scRNASeq data.

When sequencing UMI containing data, techniques are used to specifically sequence only the end of the transcript containing the UMI (usually the 3' end).

5.2 Mapping Barcodes

Since the number of unique barcodes (4^N, N=length of UMI) is much smaller than the total number of molecules per cell (~10^6), each barcode will typically be assigned to multiple transcripts. Hence, to identify unique molecules both barcode and mapping location (transcript) must be used. The first step is to map UMI reads, for which we recommend using STAR since it is fast and outputs good quality BAM-alignments. Moreover, mapping locations can be useful for eg. identifying poorly-annotated 3' UTRs of transcripts.

UMI-sequencing typically consists of paired-end reads where one read from each pair captures the cell and UMI barcodes while the other read consists of exonic sequence from the transcript (Figure 5.2). Note that trimming and/or filtering to remove reads containing poly-A sequence is recommended to avoid erors due to these read mapping to genes/transcripts with internal poly-A/poly-T sequences.

After processing the reads from a UMI experiment, the following conventions are often used:

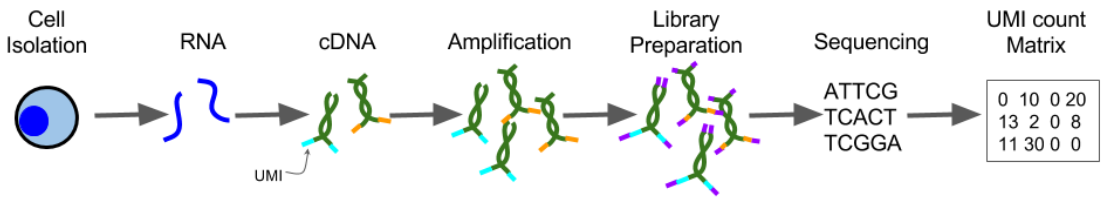


Figure 5.1: UMI sequencing protocol

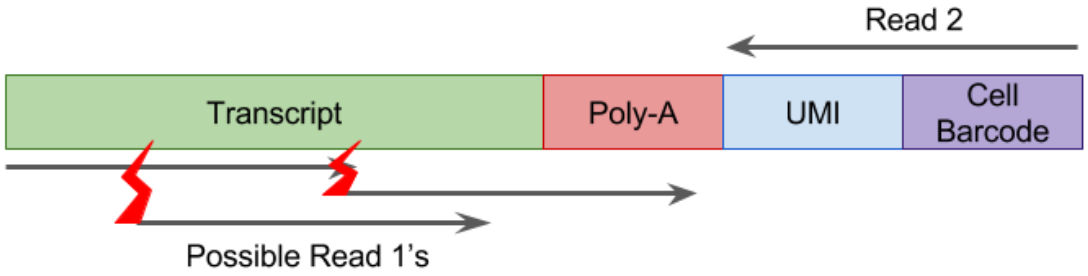


Figure 5.2: UMI sequencing reads, red lightning bolts represent different fragmentation locations

1. The UMI is added to the read name of the other paired read.
2. Reads are sorted into separate files by cell barcode
    - For extremely large, shallow datasets, the cell barcode may be added to the read name as well to reduce the number of files.

5.3 Counting Barcodes

In theory, every unique UMI-transcript pair should represent all reads originating from a single RNA molecule. However, in practice this is frequently not the case and the most common reasons are:

1. **Different UMI doesn't necessarily mean different molecule**
    - Due to PCR or sequencing errors, base-pair substitution events can result in new UMI sequences. Longer UMIs give more opportunity for errors to arise and based on estimates from cell barcodes we expect 7-10% of 10bp UMIs to contain at least one error. If not corrected for, this type of error will result in an overestimate of the number of transcripts.
2. **Different transcript doesn't necessarily mean different molecule**
    - Mapping errors and/or multimapping reads may result in some UMIs being assigned to the wrong gene/transcript. This type of error will also result in an overestimate of the number of transcripts.
3. **Same UMI doesn't necessarily mean same molecule**
    - Biases in UMI frequency and short UMIs can result in the same UMI being attached to different mRNA molecules from the same gene. Thus, the number of transcripts may be underestimated.

5.4 Correcting for Errors

How to best account for errors in UMIs remains an active area of research. The best approaches that we are aware of for resolving the issues mentioned above are:

1. UMI-tools' directional-adjacency method implements a procedure which considers both the number of mismatches and the relative frequency of similar UMIs to identify likely PCR/sequencing errors.
2. Currently an open question. The problem may be mitigated by removing UMIs with few reads to support their association with a particular transcript, or by removing all multi-mapping reads.
3. Simple saturation (aka "collision probability") correction proposed by Grun, Kester and van Oudenaarden (2014) :

$$True \approx -N * log(1 - \frac{n}{N})$$

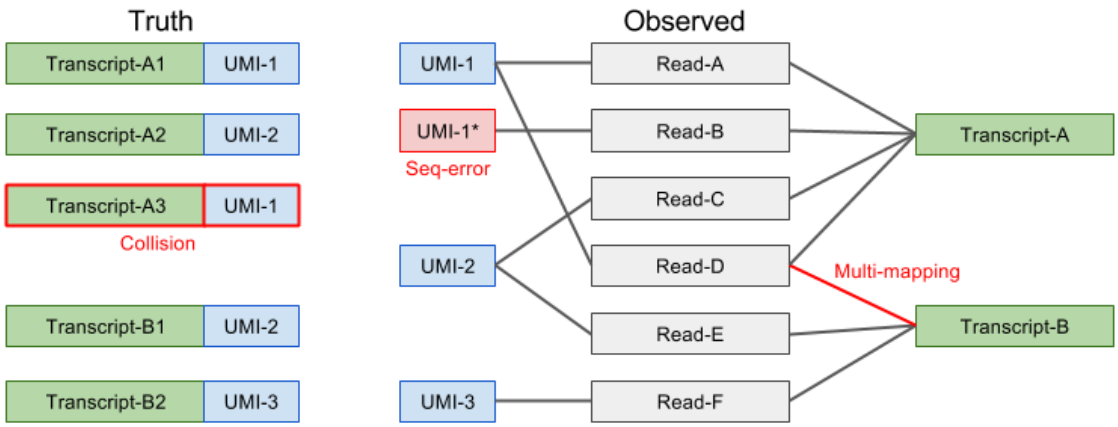


Figure 5.3: Potential Errors in UMIs

where N = total number of unique UMI barcodes and n = number of observations of a specific barcode. + An important caveat of this method is that it assumes that all UMIs are equally frequent. In most cases this is incorrect, since there is often a bias related to the GC content.

Exercise 1 Blischak et al. used 6bp UMI barcodes for their experiments and you can load this data using the command below.

```
molecules <- read.table("blischak/molecules.txt", sep = "\t")
```

Correct this data for collisions and sequencing errors assuming a 1% per base-pair sequencing error rate.

5.5 Downstream Analysis

Current UMI platforms (DropSeq, InDrop, ICell8) exhibit low and highly variable capture efficiency as shown in the figure below.

This variability can introduce strong biases and it needs to be considered in downstream analysis. Recent analyses often pool cells/genes together based on cell-type or biological pathway to increase the power. Robust statistical analyses of this data is still an open research question and it remains to be determined how to best adjust for biases.

Exercise 2 Load the read counts from the Blischak data using the command below.

```
reads <- read.table("blischak/reads.txt", sep = "\t")
```

Using this data and the unadjusted molecule counts from above:

1. Plot the variability in capture efficiency
2. Determine the amplification rate: average number of reads per UMI.

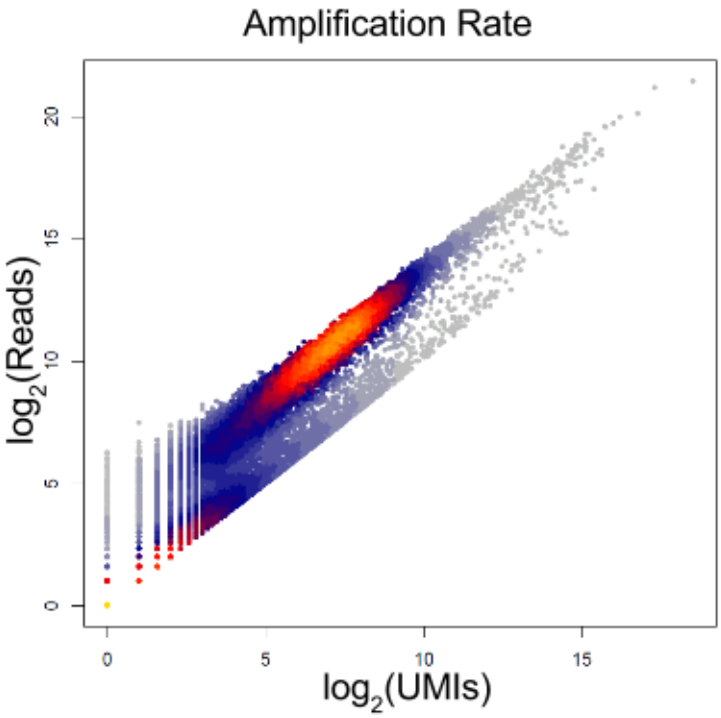


Figure 5.4: Per gene amplification rate

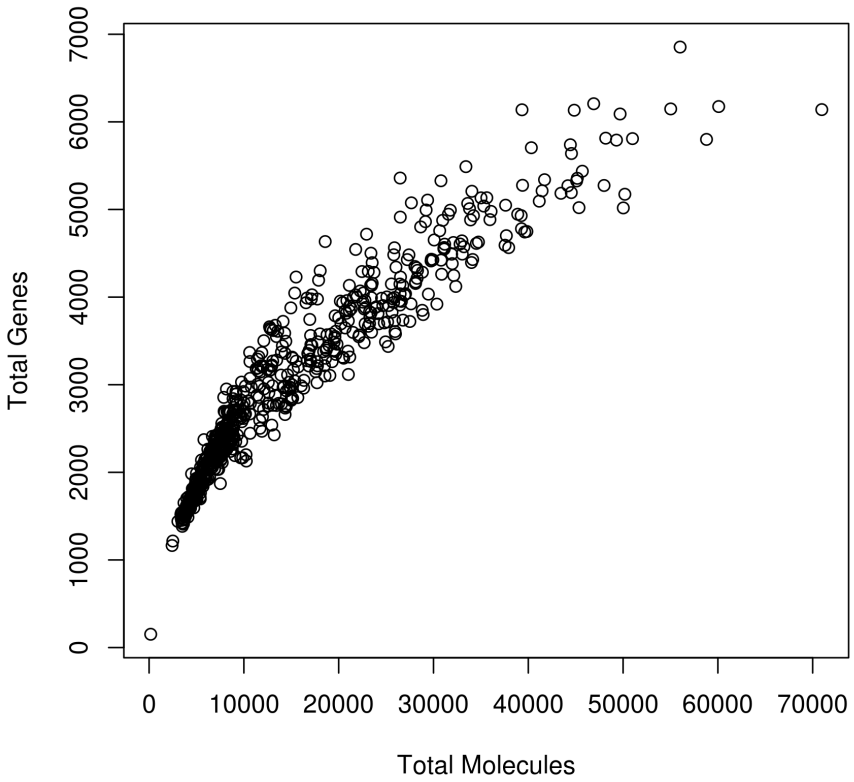


Figure 5.5: Variability in Capture Efficiency

Chapter 6

scater package

6.1 Introduction

scater is a R package single-cell RNA-seq analysis. The package contains several useful methods for quality control, visualisation and pre-processing of data prior to further downstream analysis.

scater features the following functionality:

- Automated computation of QC metrics
- Transcript quantification from read data with pseudo-alignment
- Data format standardisation
- Rich visualizations for exploratory analysis
- Seamless integration into the Bioconductor universe
- Simple normalisation methods

We highly recommend to use scater for all single-cell RNA-seq analyses and scater is the basis of the first part of the course (disclaimer: one of the teachers on this course, Davis McCarthy, is the primary developer of scater).

6.2 scater workflow

6.3 Terminology

(this chapter is taken from the scater vignette)

- The capabilities of scater are built on top of Bioconductor's Biobase.
- In Bioconductor terminology we assay numerous **"features"** for a number of **"samples"**.
- Features, in the context of scater, correspond most commonly to genes or transcripts, but could be any general genomic or transcriptomic regions (e.g. exon) of interest for which we take measurements.
- In the following chapters it may be more intuitive to mentally replace **"feature"** with **"gene"** or **"transcript"** (depending on the context of the study) wherever **"feature"** appears.
- In the scater context, **"samples"** refer to individual cells that we have assayed.

6.4 SCESet class

(this chapter is taken from the scater vignette)

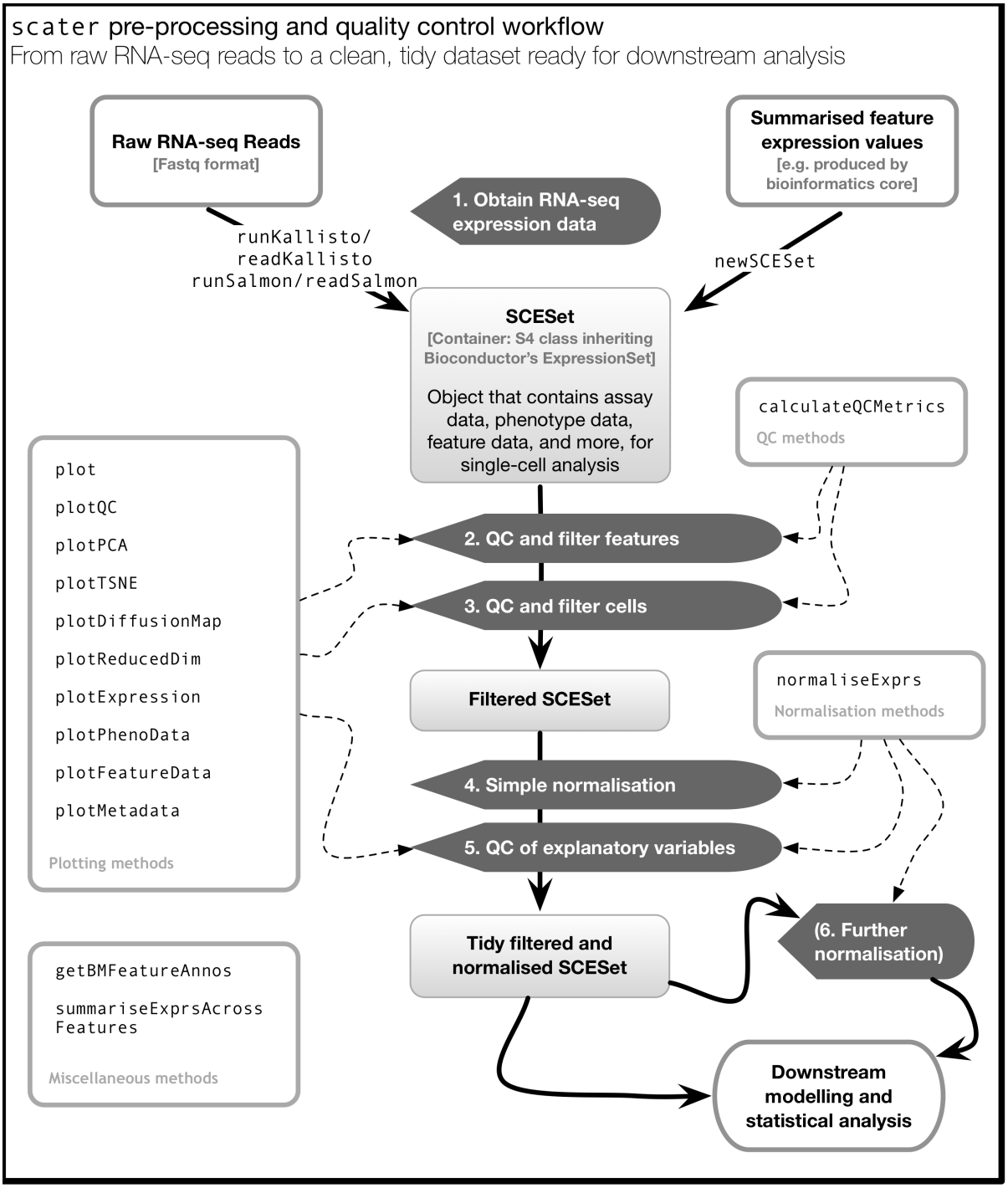

scater pre-processing and quality control workflow
From raw RNA-seq reads to a clean, tidy dataset ready for downstream analysis
Raw RNA-seq Reads
[Fastq format]
Summarised feature expression values
[e.g. produced by bioinformatics core]
1. Obtain RNA-seq expression data
runKallisto/ readKallisto runSalmon/readSalmon
newSCESet
SCESet
[Container: S4 class inheriting Bioconductor's ExpressionSet]
Object that contains assay data, phenotype data, feature data, and more, for single-cell analysis
calculateQCMetrics
QC methods
plot
plotQC
plotPCA
plotTSNE
plotDiffusionMap
plotReducedDim
plotExpression
plotPhenoData
plotFeatureData
plotMetadata
Plotting methods
2. QC and filter features
3. QC and filter cells
Filtered SCESet
normaliseExprs
Normalisation methods
4. Simple normalisation
5. QC of explanatory variables
Tidy filtered and normalised SCESet
(6. Further normalisation)
getBMFeatureAnnos
summariseExprsAcrossFeatures
Miscellaneous methods
Downstream modelling and statistical analysis

Figure 6.1:

In scater we organise single-cell expression data in objects of the **SCESet** class. The class inherits the Bioconductor ExpressionSet class, which provides a common interface familiar to those who have analyzed microarray experiments with Bioconductor. The class requires three input files:

- **exprs**, a numeric matrix of expression values, where rows are features, and columns are cells
- **phenoData**, an *AnnotatedDataFrame* object, where rows are cells, and columns are cell attributes (such as cell type, culture condition, day captured, etc.)
- **featureData**, an *AnnotatedDataFrame* object, where rows are features (e.g. genes), and columns are feature attributes, such as biotype, gc content, etc.

For more details about other features inherited from Bioconductor's **ExpressionSet** class, type `?ExpressionSet` at the R prompt.

When the data are encapsulated in the **SCESet** class, scater will automatically calculate several different properties. This will be demonstrated in the subsequent chapters.

Chapter 7

Expression QC (UMI)

7.1 Introduction

Once gene expression has been quantified it is summarized as an **expression matrix** where each row corresponds to a gene (or transcript) and each column corresponds to a single cell. This matrix should be examined to remove poor quality cells which were not detected in either read QC or mapping QC steps. Failure to remove low quality cells at this stage may add technical noise which has the potential to obscure the biological signals of interest in the downstream analysis.

Since there is currently no standard method for performing scRNASeq the expected values for the various QC measures that will be presented here can vary substantially from experiment to experiment. Thus, to perform QC we will be looking for cells which are outliers with respect to the rest of the dataset rather than comparing to independent quality standards. Consequently, care should be taken when comparing quality metrics across datasets collected using different protocols.

7.2 Blischak dataset

To illustrate cell QC, we consider a dataset of induced pluripotent stem cells generated from three different individuals (Tung et al., 2016) by John Blischak in Yoav Gilad's lab at the University of Chicago. The experiments were carried out on the Fluidigm C1 platform and to facilitate the quantification both unique molecular identifiers (UMIs) and ERCC *spike-ins* were used. The data files are located in the `blischak` folder in your working directory. These files are the copies of the original files made on the 15/03/16. We will use these copies for reproducibility purposes.

```
library(scater, quietly = TRUE)
library(knitr)
options(stringsAsFactors = FALSE)
```

Load the data and annotations:

```
molecules <- read.table("blischak/molecules.txt", sep = "\t")
anno <- read.table("blischak/annotation.txt", sep = "\t", header = TRUE)
```

Inspect a small portion of the expression matrix

```
knitr::kable(
    head(molecules[ , 1:3]), booktabs = TRUE,
    caption = 'A table of the first 6 rows and 3 columns of the molecules table.'
)
```

Table 7.1: A table of the first 6 rows and 3 columns of the molecules table.

	NA19098.r1.A01	NA19098.r1.A02	NA19098.r1.A03
ENSG00000237683	0	0	0
ENSG00000187634	0	0	0
ENSG00000188976	3	6	1
ENSG00000187961	0	0	0
ENSG00000187583	0	0	0
ENSG00000187642	0	0	0

Table 7.2: A table of the first 6 rows of the anno table.

individual	replicate	well	batch	sample_id
NA19098	r1	A01	NA19098.r1	NA19098.r1.A01
NA19098	r1	A02	NA19098.r1	NA19098.r1.A02
NA19098	r1	A03	NA19098.r1	NA19098.r1.A03
NA19098	r1	A04	NA19098.r1	NA19098.r1.A04
NA19098	r1	A05	NA19098.r1	NA19098.r1.A05
NA19098	r1	A06	NA19098.r1	NA19098.r1.A06

```
knitr::kable(
    head(anno), booktabs = TRUE,
    caption = 'A table of the first 6 rows of the anno table.'
)
```

The data consists of 3 individuals and 3 replicates and therefore has 9 batches in total.

We standardize the analysis by using the scater package. First, create the scater SCESet classes:

```
pheno_data <- new("AnnotatedDataFrame", anno)
rownames(pheno_data) <- pheno_data$sample_id
umi <- scater::newSCESet(
    countData = molecules,
    phenoData = pheno_data
)
```

Remove genes that are not expressed in any cell:

```
keep_feature <- rowSums(counts(umi) > 0) > 0
umi <- umi[keep_feature, ]
```

Define control features (genes) - ERCC spike-ins and mitochondrial genes (provided by the authors):

```
ercc <- featureNames(umi)[grepl("ERCC-", featureNames(umi))]
mt <- c("ENSG00000198899", "ENSG00000198727", "ENSG00000198888",
        "ENSG00000198886", "ENSG00000212907", "ENSG00000198786",
        "ENSG00000198695", "ENSG00000198712", "ENSG00000198804",
        "ENSG00000198763", "ENSG00000228253", "ENSG00000198938",
        "ENSG00000198840")
```

Calculate the quality metrics:

```
umi <- scater::calculateQCMetrics(
    umi,
```

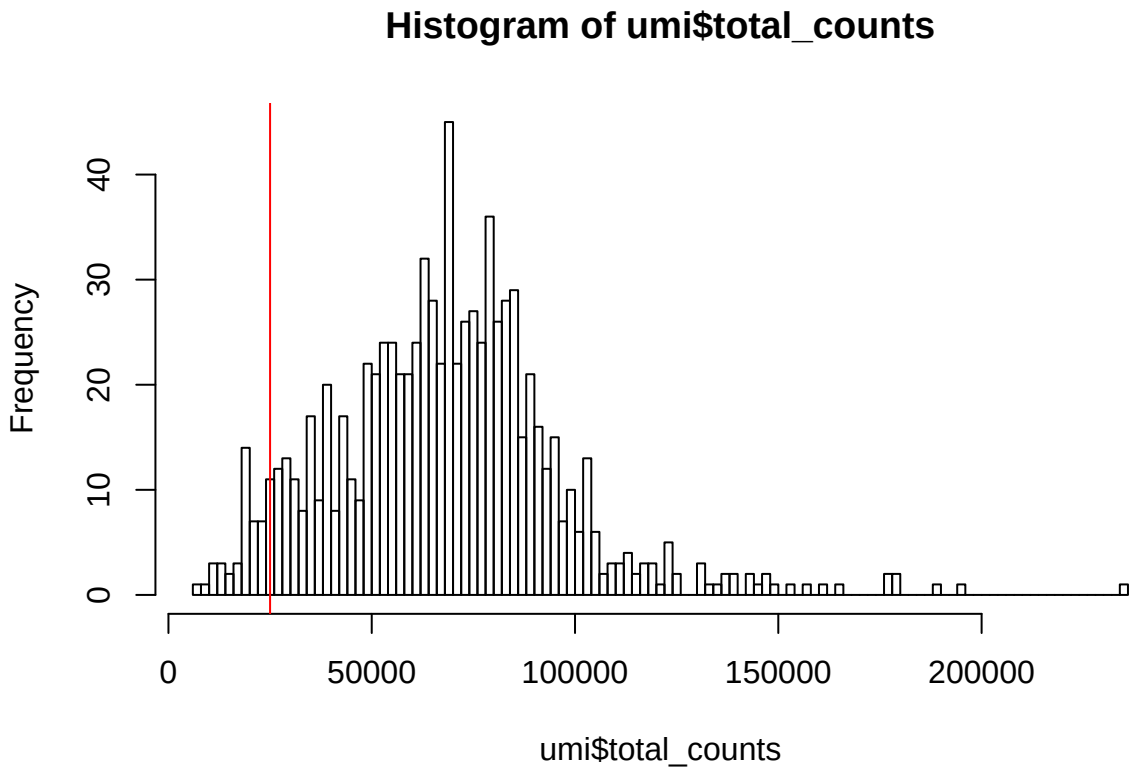


Figure 7.1: Histogram of library sizes for all cells

```
    feature_controls = list(ERCC = ercc, MT = mt)
)
```

7.3 Cell QC

7.3.1 Library size

Next we consider the total number of RNA molecules detected per sample (if we were using read counts rather than UMI counts this would be the total number of reads). Wells with few reads/molecules are likely to have been broken or failed to capture a cell, and should thus be removed.

```
hist(
    umi$total_counts,
    breaks = 100
)
abline(v = 25000, col = "red")
```

Exercise 1

1. How many cells does our filter remove?
2. What distribution do you expect that the total number of molecules for each cell should follow?

Our answer

Table 7.3: The number of cells removed by total counts filter (FALSE)

filter_by_total_counts	Freq
FALSE	46
TRUE	818

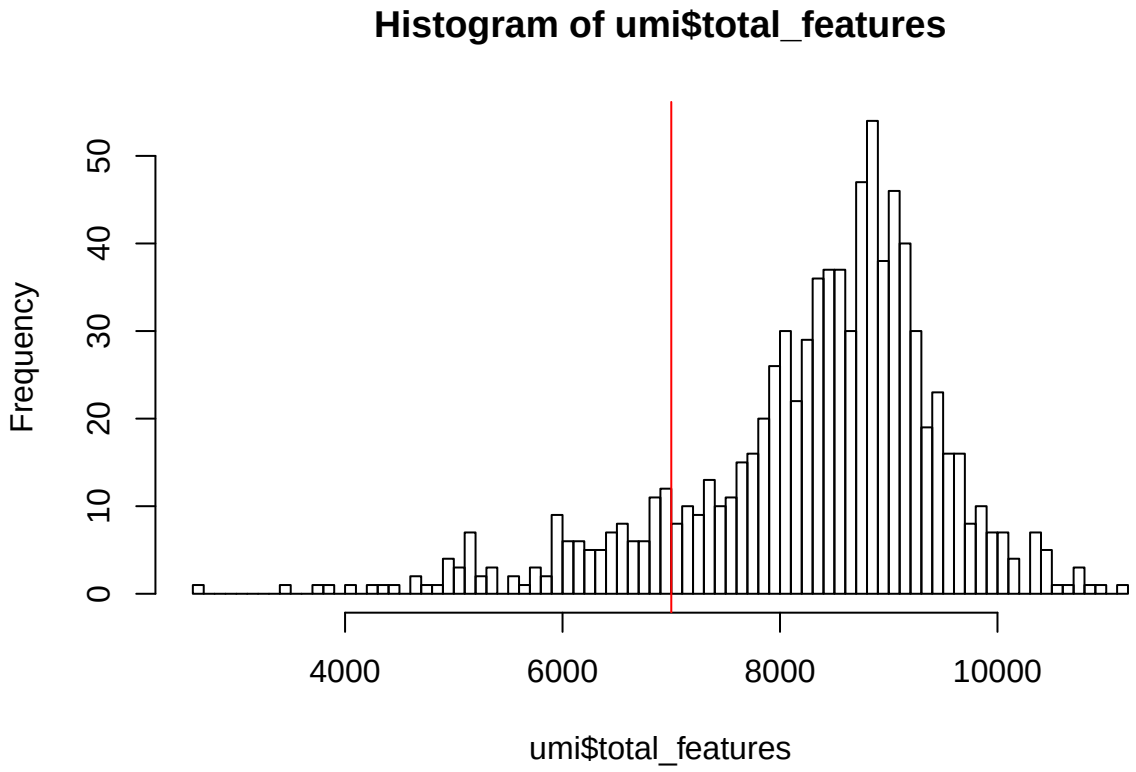


Figure 7.2: Histogram of the number of detected genes in all cells

7.3.2 Detected genes (1)

In addition to ensuring sufficient sequencing depth for each sample, we also want to make sure that the reads are distributed across the transcriptome. Thus, we count the total number of unique genes detected in each sample.

```
hist(
    umi$total_features,
    breaks = 100
)
abline(v = 7000, col = "red")
```

From the plot we conclude that most cells have between 7,000-10,000 detected genes, which is normal for high-depth scRNA-seq. However, this varies by experimental protocol and sequencing depth. For example, droplet-based methods or samples with lower sequencing-depth typically detect fewer genes per cell. The most notable feature in the above plot is the **"heavy tail"** on the left hand side of the distribution. If detection rates were equal across the cells then the distribution should be approximately normal. Thus we remove those cells in the tail of the distribution (fewer than 7,000 detected genes).

Exercise 2

Table 7.4: The number of cells removed by total features filter (FALSE)

filter_by_expr_features	Freq
FALSE	120
TRUE	744

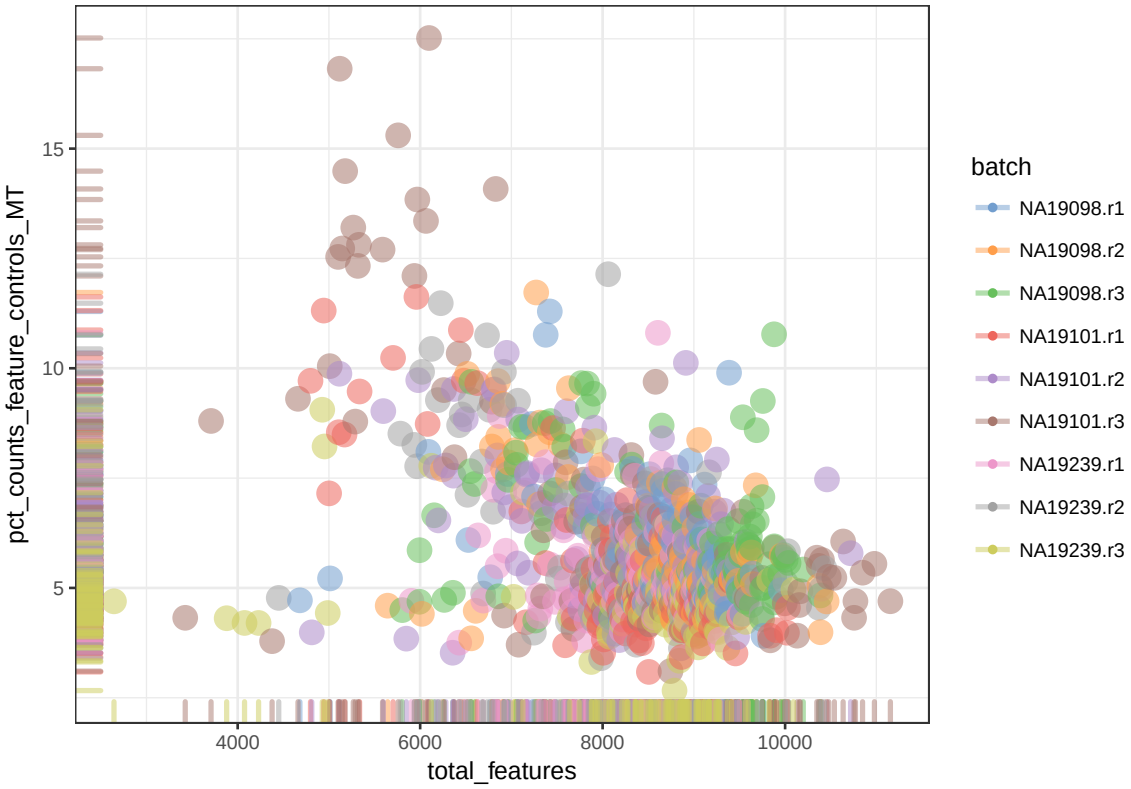


Figure 7.3: Percentage of counts in MT genes

How many cells does our filter remove?

Our answer

7.3.3 ERCCs and MTs

Another measure of cell quality is the ratio between ERCC *spike-in* RNAs and endogenous RNAs. This ratio can be used to estimate the total amount of RNA in the captured cells. Cells with a high level of *spike-in* RNAs had low starting amounts of RNA, likely due to the cell being dead or stressed which may result in the RNA being degraded.

```
scater::plotPhenoData(
    umi,
    aes_string(x = "total_features",
               y = "pct_counts_feature_controls_MT",
               colour = "batch")
)
```

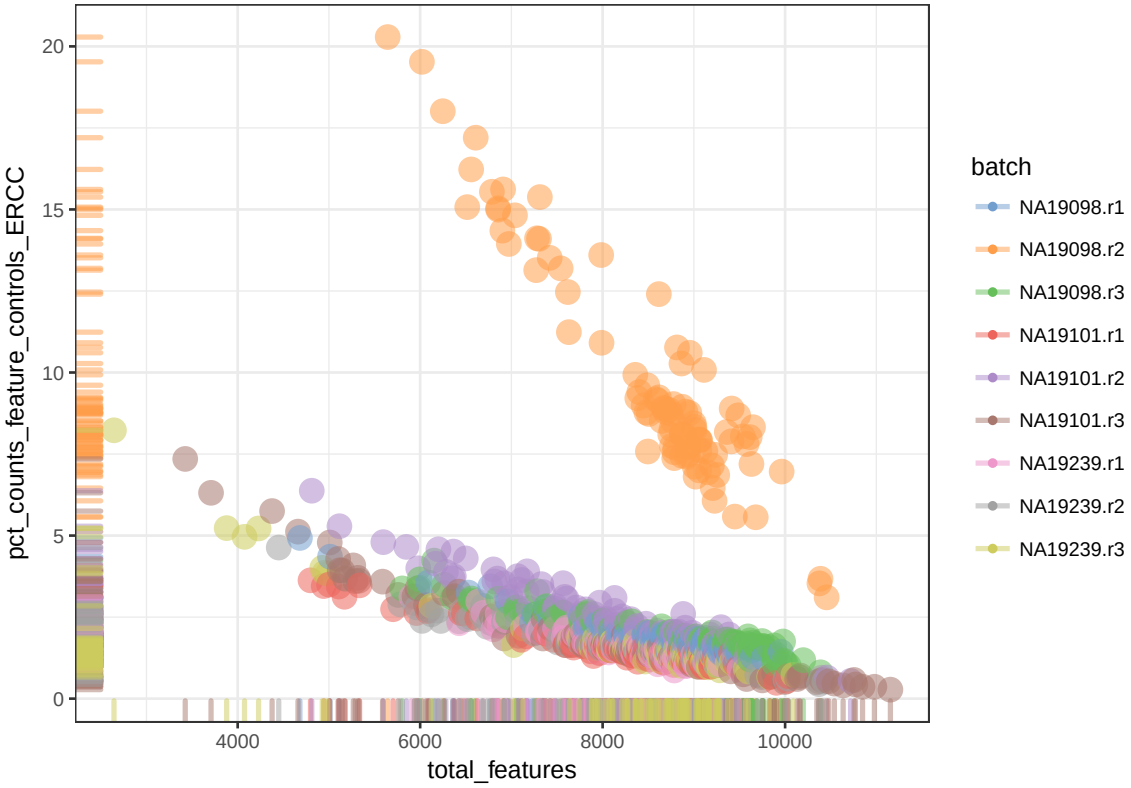


Figure 7.4: Percentage of counts in ERCCs

Table 7.5: The number of cells removed by ERCC filter (FALSE)

filter_by_ERCC	Freq
FALSE	96
TRUE	768

```
scater::plotPhenoData(
    umi,
    aes_string(x = "total_features",
               y = "pct_counts_feature_controls_ERCC",
               colour = "batch")
)
```

The above analysis shows that majority of the cells from NA19098.r2 batch have a very high ERCC/Endo ratio. Indeed, it has been shown by the authors that this batch contains cells of smaller size.

Exercise 3

Create filters for removing batch NA19098.r2 and cells with high expression of mitochondrial genes (>10% of total counts in a cell).

Our answer

Exercise 4

What would you expect to see in the ERCC vs counts plot if you were examining a dataset containing cells of different sizes (eg. normal & senescent cells)?

Table 7.6: The number of cells removed by MT filter (FALSE)

filter_by_MT	Freq
FALSE	31
TRUE	833

Table 7.7: The number of cells removed by manual filter (FALSE)

Var1	Freq
FALSE	210
TRUE	654

Answer

You would expect to see a group corresponding to the smaller cells (normal) with a higher fraction of ERCC reads than a separate group corresponding to the larger cells (senescent).

7.4 Cell filtering

7.4.1 Manual

Now we can define a cell filter based on our previous analysis:

```
umi$use <- (
    # sufficient features (genes)
    filter_by_expr_features &
    # sufficient molecules counted
    filter_by_total_counts &
    # sufficient endogenous RNA
    filter_by_ERCC &
    # remove cells with unusual number of reads in MT genes
    filter_by_MT
)
```

```
knitr::kable(
  as.data.frame(table(umi$use)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by manual filter (FALSE)'
)
```

7.4.2 Default Thresholds

Results from the biological analysis of single-cell RNA-seq data are often strongly influenced by outliers. Thus, it is important to include multiple filters. A robust way of detecting outliers is through the median absolute difference which is defined as $d_i = |r_i - m|$, where r_i is the number of reads in cell i and m is the median number of reads across the all cells in the sample. By default, scater removes all cells where $d_i > 5{*}\text{median}(d_i)$. A similar filter is used for the number of detected genes. Furthermore, scater removes all cells where >80% of counts were assigned to control genes and any cells that have been marked as controls.

Table 7.8: The number of cells removed by default filter (FALSE)

Var1	Freq
FALSE	6
TRUE	858

```
umi$use_default <- (
    # remove cells with unusual numbers of genes
    !umi$filter_on_total_features &
    # remove cells with unusual numbers molecules counted
    !umi$filter_on_total_counts &
    # < 80% ERCC spike-in
    !umi$filter_on_pct_counts_feature_controls_ERCC &
    # < 80% mitochondrial
    !umi$filter_on_pct_counts_feature_controls_MT &
    # controls shouldn't be used in downstream analysis
    !umi$is_cell_control
)
```

```
knitr::kable(
  as.data.frame(table(umi$use_default)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by default filter (FALSE)'
)
```

7.4.3 Automatic

Another option available in **scater** is to conduct PCA on a set of QC metrics and then use automatic outlier detection to identify potentially problematic cells.

By default, the following metrics are used for PCA-based outlier detection:

- **pct_counts_top_100_features**
- **total_features**
- **pct_counts_feature_controls**
- **n_detected_feature_controls**
- **log10_counts_endogenous_features**
- **log10_counts_feature_controls**

scater first creates a matrix where the rows represent cells and the columns represent the different QC metrics. Here, the PCA plot provides a 2D representation of cells ordered by their quality metrics. The outliers are then detected using methods from the mvoutlier package.

```
umi <-
scater::plotPCA(umi,
                size_by = "total_features",
                shape_by = "use",
                pca_data_input = "pdata",
                detect_outliers = TRUE,
                return_SCESet = TRUE)
```

```
## The following cells/samples are detected as outliers:
```

```
## NA19098.r2.A01
## NA19098.r2.A02
## NA19098.r2.A06
## NA19098.r2.A09
## NA19098.r2.A10
## NA19098.r2.A12
## NA19098.r2.B01
## NA19098.r2.B03
## NA19098.r2.B04
## NA19098.r2.B05
## NA19098.r2.B07
## NA19098.r2.B11
## NA19098.r2.B12
## NA19098.r2.C01
## NA19098.r2.C02
## NA19098.r2.C03
## NA19098.r2.C04
## NA19098.r2.C05
## NA19098.r2.C06
## NA19098.r2.C07
## NA19098.r2.C08
## NA19098.r2.C09
## NA19098.r2.C10
## NA19098.r2.C11
## NA19098.r2.C12
## NA19098.r2.D01
## NA19098.r2.D02
## NA19098.r2.D03
## NA19098.r2.D04
## NA19098.r2.D07
## NA19098.r2.D08
## NA19098.r2.D09
## NA19098.r2.D10
## NA19098.r2.D12
## NA19098.r2.E01
## NA19098.r2.E02
## NA19098.r2.E03
## NA19098.r2.E04
## NA19098.r2.E05
## NA19098.r2.E06
## NA19098.r2.E07
## NA19098.r2.E12
## NA19098.r2.F01
## NA19098.r2.F02
## NA19098.r2.F07
## NA19098.r2.F08
## NA19098.r2.F09
## NA19098.r2.F10
## NA19098.r2.F11
## NA19098.r2.F12
## NA19098.r2.G01
## NA19098.r2.G02
## NA19098.r2.G03
## NA19098.r2.G05
```

Table 7.9: The number of cells removed by automatic filter (FALSE)

Var1	Freq
FALSE	791
TRUE	73

```
## NA19098.r2.G06
## NA19098.r2.G08
## NA19098.r2.G09
## NA19098.r2.G10
## NA19098.r2.G11
## NA19098.r2.H01
## NA19098.r2.H02
## NA19098.r2.H03
## NA19098.r2.H04
## NA19098.r2.H05
## NA19098.r2.H06
## NA19098.r2.H07
## NA19098.r2.H08
## NA19098.r2.H10
## NA19098.r2.H12
## NA19101.r3.A02
## NA19101.r3.C12
## NA19101.r3.D01
## NA19101.r3.E08
## Variables with highest loadings for PC1 and PC2:
## \begin{tabular}{l|r|r}
## \hline
##   & PC1 & PC2\\
## \hline
## pct\_counts\_top\_100\_features & 0.4771343 & 0.3009332\\
## \hline
## pct\_counts\_feature\_controls & 0.4735839 & 0.3309562\\
## \hline
## n\_detected\_feature\_controls & 0.1332811 & 0.5367629\\
## \hline
## log10\_counts\_feature\_controls & -0.1427373 & 0.5911762\\
## \hline
## total\_features & -0.5016681 & 0.2936705\\
## \hline
## log10\_counts\_endogenous\_features & -0.5081855 & 0.2757918\\
## \hline
## \end{tabular}
```

```
knitr::kable(
  as.data.frame(table(umi$outlier)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by automatic filter (FALSE)'
)
```

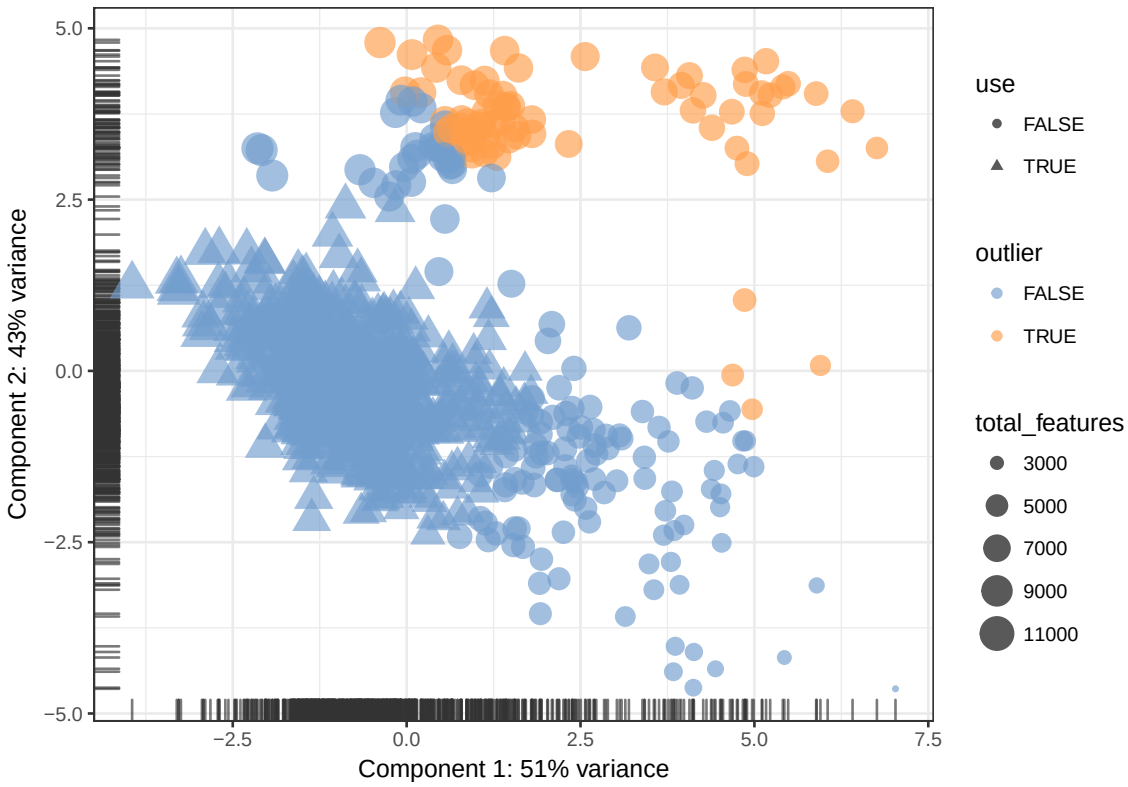


Figure 7.5: PCA plot used for automatic detection of cell outliers

7.5 Compare filterings

Exercise 5

Compare the default, automatic and manual cell filters. Plot a Venn diagram of the outlier cells from these filterings.

Hint: Use `limma::vennCounts` and `limma::vennDiagram` functions from the limma package to make a Venn diagram.

Answer

7.6 Gene analysis

7.6.1 Gene expression

In addition to removing cells with poor quality, it is usually a good idea to exclude genes where we suspect that technical artefacts may have skewed the results. Moreover, inspection of the gene expression profiles may provide insights about how the experimental procedures could be improved.

It is often instructive to consider the number of reads consumed by the top 50 expressed genes.

```
scater::plotQC(umi, type = "highest-expression")
```

The distributions are relatively flat indicating (but not guaranteeing!) good coverage of the full transcriptome of these cells. However, there are several spike-ins in the top 15 genes which suggests a greater dilution of

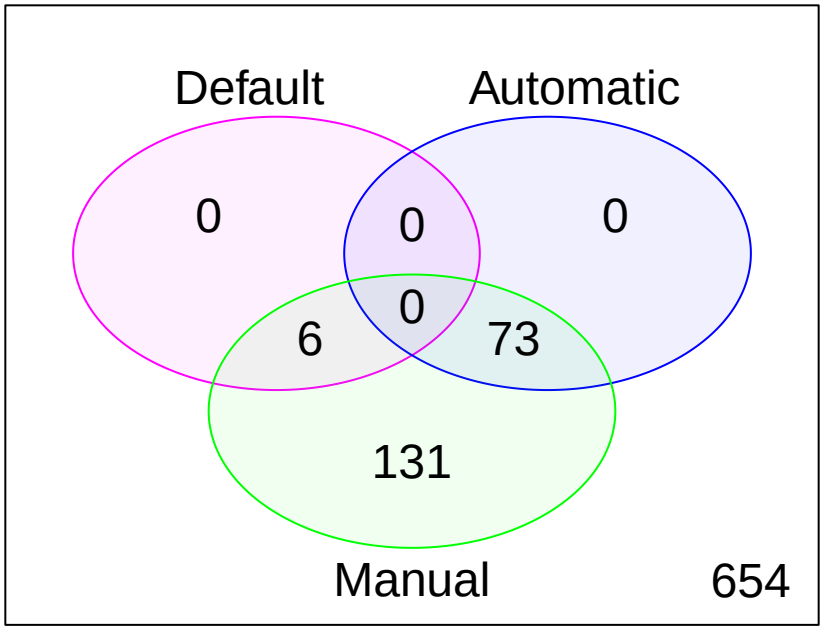


Figure 7.6: Comparison of the default, automatic and manual cell filters

Table 7.10: The number of genes removed by gene filter (FALSE)

filter_genes	Freq
FALSE	4663
TRUE	14063

the spike-ins may be preferrable if the experiment is to be repeated.

7.6.2 Gene filtering

It is typically a good idea to remove genes whose expression level is considered **"undetectable"**. We define a gene as detectable if at least two cells contain more than 1 transcript from the gene. If we were considering read counts rather than UMI counts a reasonable threshold is to require at least five reads in at least two cells. However, in both cases the threshold strongly depends on the sequencing depth. It is important to keep in mind that genes must be filtered after cell filtering since some genes may only be detected in poor quality cells (**note** `pData(umi)$use` filter applied to the `umi` dataset).

```
filter_genes <- apply(counts(umi[ , pData(umi)$use]), 1,
                      function(x) length(x[x > 1]) >= 2)
fData(umi)$use <- filter_genes
```

```
knitr::kable(
    as.data.frame(table(filter_genes)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of genes removed by gene filter (FALSE)'
)
```

Depending on the cell-type, protocol and sequencing depth, other cut-offs may be appropriate.

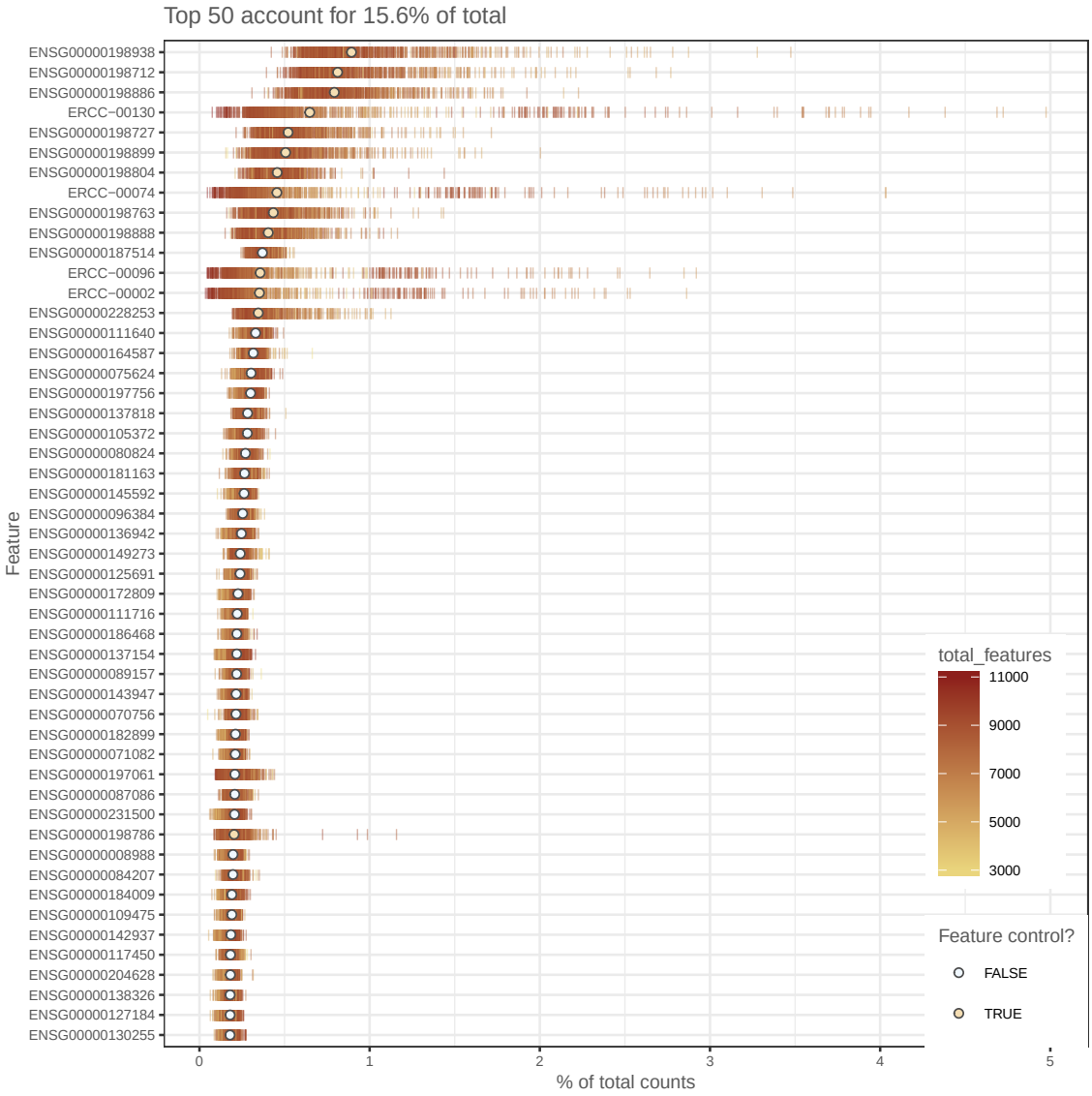


Figure 7.7: Number of total counts consumed by the top 50 expressed genes

7.7 Save the data

Dimensions of the QCed dataset (do not forget about the gene filter we defined above):

```
dim(umi[fData(umi)$use, pData(umi)$use])
```

```
## Features  Samples
##    14063      654
```

Save the data:

```
saveRDS(umi, file = "blischak/umi.rds")
```

If you want to further check yourself you can download our `umi` object. If you followed the steps above it should be exactly the same as yours.

7.8 Big Exercise

Perform exactly the same QC analysis with read counts of the same Blischak data. Use `blischak/reads.txt` file to load the reads. Once you have finished please compare your results to ours (next chapter).

Chapter 8

Expression QC (Reads)

This chapter contains the summary plots and tables for the QC exercise based on the reads for the Bischak data discussed in the previous chapter.

```
library(scater, quietly = TRUE)
library(knitr)
options(stringsAsFactors = FALSE)
```

```
reads <- read.table("blischak/reads.txt", sep = "\t")
anno <- read.table("blischak/annotation.txt", sep = "\t", header = TRUE)
```

```
knitr::kable(
    head(reads[ , 1:3]), booktabs = TRUE,
    caption = 'A table of the first 6 rows and 3 columns of the molecules table.'
)
```

```
knitr::kable(
    head(anno), booktabs = TRUE,
    caption = 'A table of the first 6 rows of the anno table.'
)
```

```
pheno_data <- new("AnnotatedDataFrame", anno)
rownames(pheno_data) <- pheno_data$sample_id
reads <- scater::newSCESet(
    countData = reads,
    phenoData = pheno_data
)
```

Table 8.1: A table of the first 6 rows and 3 columns of the molecules table.

	NA19098.r1.A01	NA19098.r1.A02	NA19098.r1.A03
ENSG00000237683	0	0	0
ENSG00000187634	0	0	0
ENSG00000188976	57	140	1
ENSG00000187961	0	0	0
ENSG00000187583	0	0	0
ENSG00000187642	0	0	0

Table 8.2: A table of the first 6 rows of the anno table.

individual	replicate	well	batch	sample_id
NA19098	r1	A01	NA19098.r1	NA19098.r1.A01
NA19098	r1	A02	NA19098.r1	NA19098.r1.A02
NA19098	r1	A03	NA19098.r1	NA19098.r1.A03
NA19098	r1	A04	NA19098.r1	NA19098.r1.A04
NA19098	r1	A05	NA19098.r1	NA19098.r1.A05
NA19098	r1	A06	NA19098.r1	NA19098.r1.A06

Table 8.3: The number of cells removed by total counts filter (FALSE)

filter_by_total_counts	Freq
FALSE	180
TRUE	684

```
keep_feature <- rowSums(counts(reads) > 0) > 0
reads <- reads[keep_feature, ]
```

```
ercc <- featureNames(reads)[grepl("ERCC-", featureNames(reads))]
mt <- c("ENSG00000198899", "ENSG00000198727", "ENSG00000198888",
        "ENSG00000198886", "ENSG00000212907", "ENSG00000198786",
        "ENSG00000198695", "ENSG00000198712", "ENSG00000198804",
        "ENSG00000198763", "ENSG00000228253", "ENSG00000198938",
        "ENSG00000198840")
```

```
reads <- scater::calculateQCMetrics(
    reads,
    feature_controls = list(ERCC = ercc, MT = mt)
)
```

```
hist(
    reads$total_counts,
    breaks = 100
)
abline(v = 1.3e6, col = "red")
```

```
filter_by_total_counts <- (reads$total_counts > 1.3e6)
```

```
knitr::kable(
    as.data.frame(table(filter_by_total_counts)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of cells removed by total counts filter (FALSE)'
)
```

```
hist(
    reads$total_features,
    breaks = 100
)
abline(v = 7000, col = "red")
```

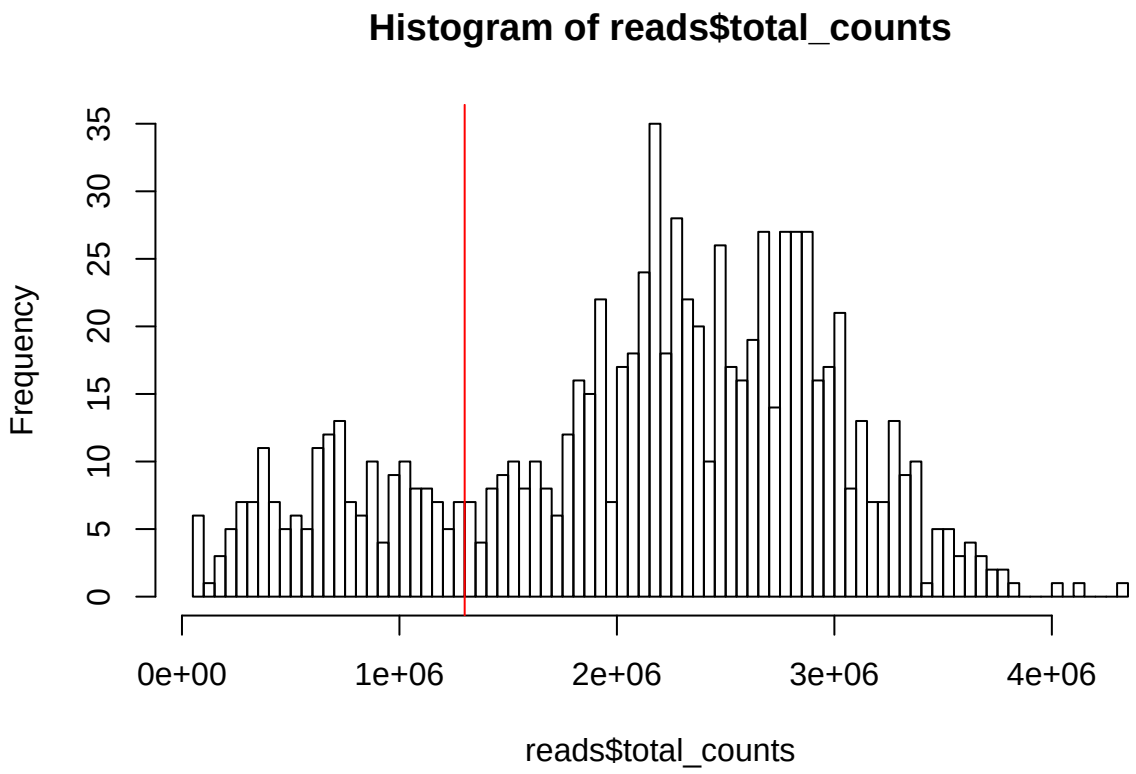


Figure 8.1: Histogram of library sizes for all cells

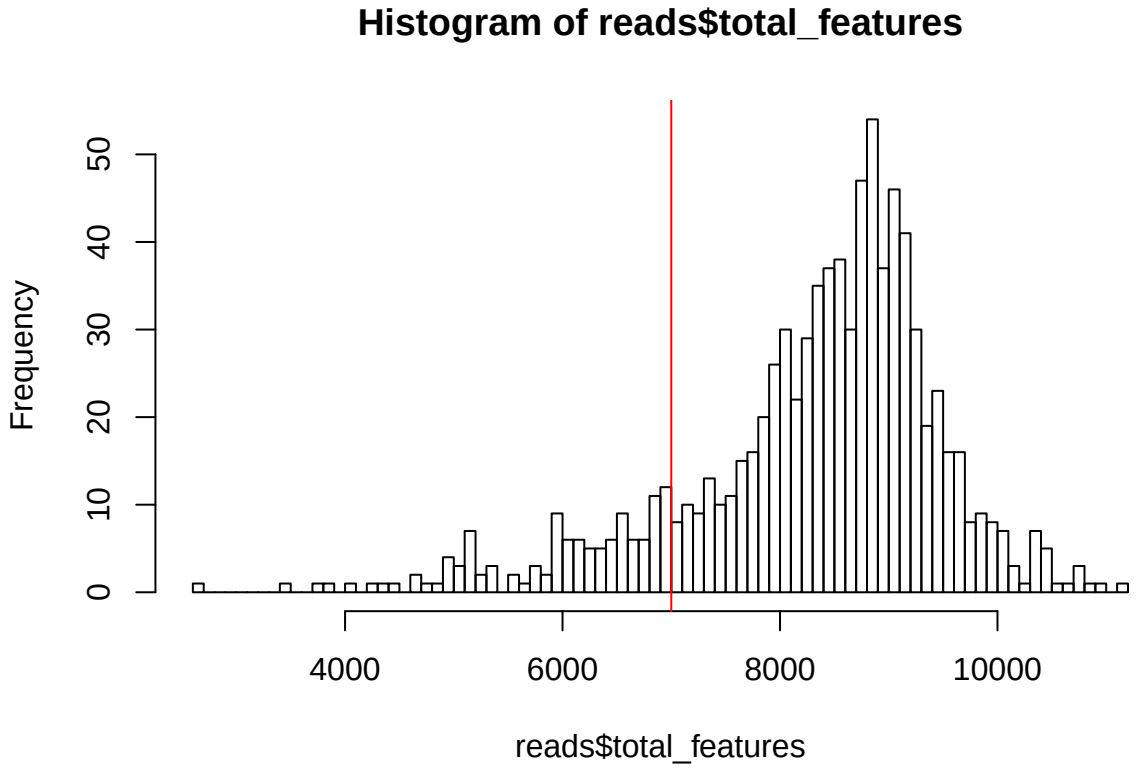


Figure 8.2: Histogram of the number of detected genes in all cells

Table 8.4: The number of cells removed by total features filter (FALSE)

filter_by_expr_features	Freq
FALSE	120
TRUE	744

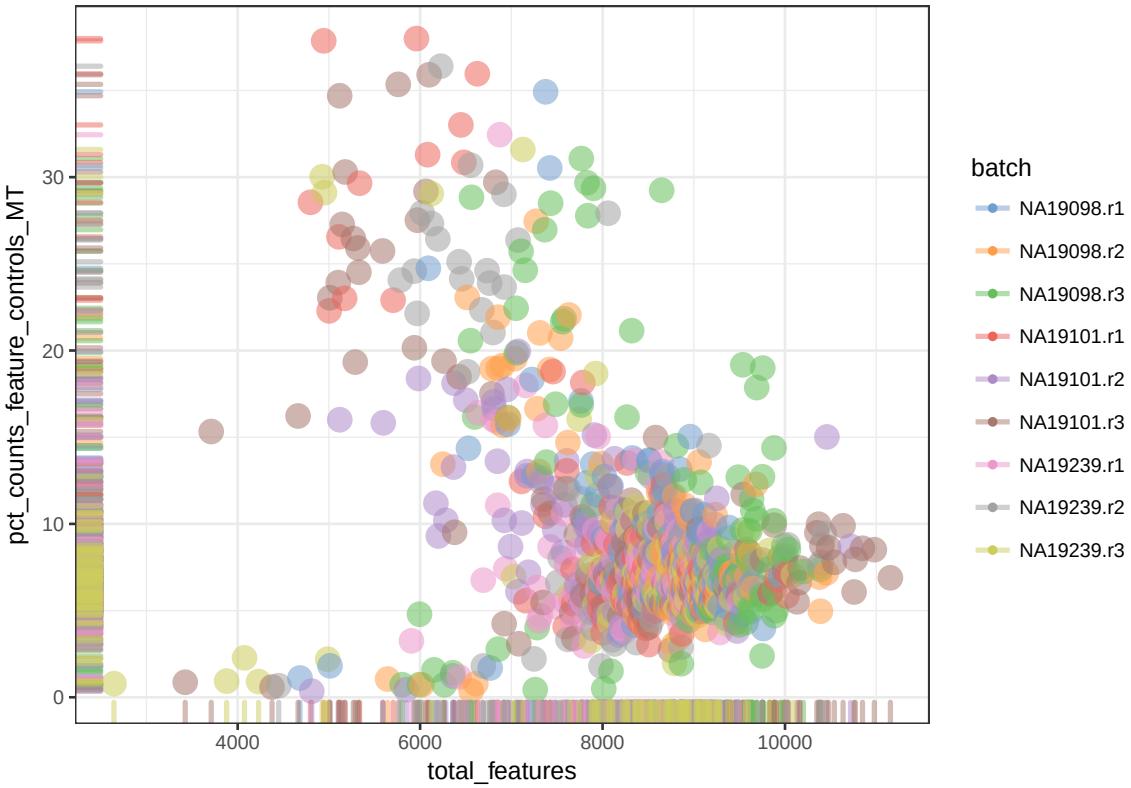


Figure 8.3: Percentage of counts in MT genes

```
filter_by_expr_features <- (reads$total_features > 7000)
```

```
knitr::kable(
    as.data.frame(table(filter_by_expr_features)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of cells removed by total features filter (FALSE)'
)
```

```
scater::plotPhenoData(
    reads,
    aes_string(x = "total_features",
               y = "pct_counts_feature_controls_MT",
               colour = "batch")
)
```

```
scater::plotPhenoData(
    reads,
    aes_string(x = "total_features",
```

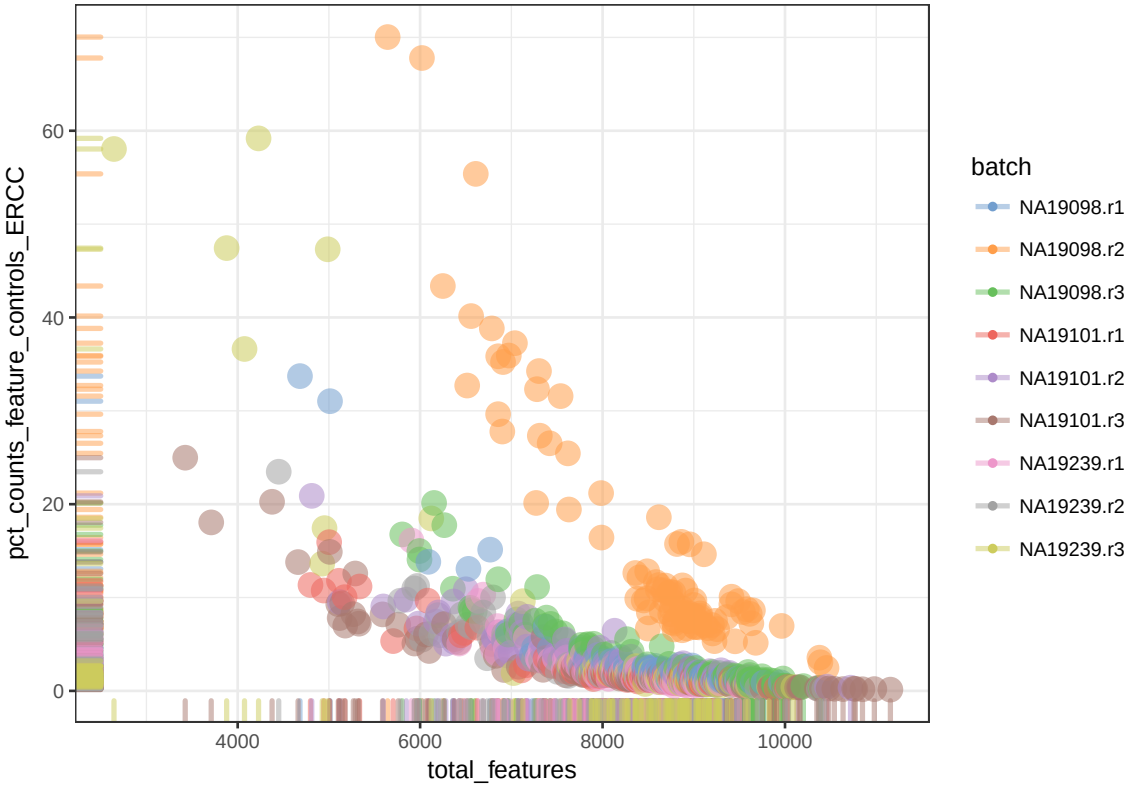


Figure 8.4: Percentage of counts in ERCCs

Table 8.5: The number of cells removed by ERCC filter (FALSE)

filter_by_ERCC	Freq
FALSE	103
TRUE	761

```
                y = "pct_counts_feature_controls_ERCC",
                colour = "batch")
)
```

```
filter_by_ERCC <- reads$batch != "NA19098.r2" &
    reads$pct_counts_feature_controls_ERCC < 25
```

```
knitr::kable(
  as.data.frame(table(filter_by_ERCC)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by ERCC filter (FALSE)'
)
```

```
filter_by_MT <- reads$pct_counts_feature_controls_MT < 30
```

```
knitr::kable(
  as.data.frame(table(filter_by_MT)),
  booktabs = TRUE,
```

Table 8.6: The number of cells removed by MT filter (FALSE)

filter_by_MT	Freq
FALSE	18
TRUE	846

Table 8.7: The number of cells removed by manual filter (FALSE)

Var1	Freq
FALSE	259
TRUE	605

```
  row.names = FALSE,
  caption = 'The number of cells removed by MT filter (FALSE)'
)
```

```
reads$use <- (
    # sufficient features (genes)
    filter_by_expr_features &
    # sufficient molecules counted
    filter_by_total_counts &
    # sufficient endogenous RNA
    filter_by_ERCC &
    # remove cells with unusual number of reads in MT genes
    filter_by_MT
)
```

```
knitr::kable(
  as.data.frame(table(reads$use)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by manual filter (FALSE)'
)
```

```
reads$use_default <- (
    # remove cells with unusual numbers of genes
    !reads$filter_on_total_features &
    # sufficient molecules counted
    !reads$filter_on_total_counts &
    # sufficient endogenous RNA
    !reads$filter_on_pct_counts_feature_controls_ERCC &
    # remove cells with unusual number of reads in MT genes
    !reads$filter_on_pct_counts_feature_controls_MT &
    # controls shouldn't be used in downstream analysis
    !reads$is_cell_control
)
```

```
knitr::kable(
  as.data.frame(table(reads$use_default)),
  booktabs = TRUE,
  row.names = FALSE,
```

Table 8.8: The number of cells removed by default filter (FALSE)

Var1	Freq
FALSE	37
TRUE	827

```
  caption = 'The number of cells removed by default filter (FALSE)'
)
```

```
reads <-
scater::plotPCA(reads,
                size_by = "total_features",
                shape_by = "use",
                pca_data_input = "pdata",
                detect_outliers = TRUE,
                return_SCESet = TRUE)
```

```
## The following cells/samples are detected as outliers:
## NA19098.r1.B10
## NA19098.r1.D07
## NA19098.r1.E04
## NA19098.r1.F06
## NA19098.r1.H08
## NA19098.r1.H09
## NA19098.r2.A01
## NA19098.r2.A06
## NA19098.r2.A09
## NA19098.r2.A12
## NA19098.r2.B01
## NA19098.r2.B11
## NA19098.r2.B12
## NA19098.r2.C04
## NA19098.r2.C09
## NA19098.r2.D02
## NA19098.r2.D03
## NA19098.r2.D09
## NA19098.r2.E04
## NA19098.r2.E07
## NA19098.r2.F01
## NA19098.r2.F11
## NA19098.r2.G01
## NA19098.r2.G05
## NA19098.r2.G10
## NA19098.r2.H01
## NA19098.r2.H07
## NA19098.r2.H08
## NA19098.r2.H12
## NA19098.r3.A05
## NA19098.r3.A07
## NA19098.r3.B02
## NA19098.r3.C07
## NA19098.r3.E05
```

```
## NA19098.r3.E08
## NA19098.r3.E09
## NA19098.r3.F11
## NA19098.r3.F12
## NA19098.r3.G02
## NA19098.r3.G03
## NA19098.r3.G04
## NA19098.r3.G11
## NA19098.r3.G12
## NA19098.r3.H08
## NA19101.r1.A01
## NA19101.r1.A12
## NA19101.r1.B01
## NA19101.r1.B06
## NA19101.r1.E09
## NA19101.r1.E11
## NA19101.r1.F05
## NA19101.r1.F10
## NA19101.r1.G01
## NA19101.r1.G06
## NA19101.r1.H04
## NA19101.r1.H09
## NA19101.r2.A03
## NA19101.r2.C10
## NA19101.r2.E05
## NA19101.r2.F02
## NA19101.r2.H04
## NA19101.r2.H10
## NA19101.r3.A02
## NA19101.r3.A03
## NA19101.r3.A05
## NA19101.r3.A09
## NA19101.r3.B05
## NA19101.r3.C01
## NA19101.r3.C09
## NA19101.r3.C12
## NA19101.r3.D01
## NA19101.r3.D04
## NA19101.r3.D07
## NA19101.r3.D09
## NA19101.r3.E08
## NA19101.r3.F09
## NA19101.r3.G09
## NA19101.r3.H01
## NA19101.r3.H03
## NA19101.r3.H07
## NA19101.r3.H09
## NA19239.r1.F05
## NA19239.r1.G05
## NA19239.r2.B01
## NA19239.r2.B03
## NA19239.r2.B10
## NA19239.r2.B11
## NA19239.r2.C03
```

```
## NA19239.r2.C06
## NA19239.r2.C08
## NA19239.r2.D07
## NA19239.r2.D09
## NA19239.r2.E09
## NA19239.r2.F04
## NA19239.r2.F06
## NA19239.r2.F07
## NA19239.r2.F12
## NA19239.r2.G03
## NA19239.r2.G08
## NA19239.r2.H02
## NA19239.r2.H03
## NA19239.r2.H07
## NA19239.r3.A01
## NA19239.r3.B09
## NA19239.r3.C04
## NA19239.r3.C07
## NA19239.r3.E01
## NA19239.r3.E03
## NA19239.r3.E12
## NA19239.r3.H02
## NA19239.r3.H10
## Variables with highest loadings for PC1 and PC2:
## \begin{tabular}{l|r|r}
## \hline
##   & PC1 & PC2\\
## \hline
## pct\_counts\_feature\_controls & 0.5057646 & 0.2473134\\
## \hline
## pct\_counts\_top\_100\_features & 0.4888852 & 0.2277068\\
## \hline
## n\_detected\_feature\_controls & 0.0231277 & 0.6235516\\
## \hline
## log10\_counts\_feature\_controls & -0.1226860 & 0.6576822\\
## \hline
## total\_features & -0.4655518 & 0.2219694\\
## \hline
## log10\_counts\_endogenous\_features & -0.5223679 & 0.1278782\\
## \hline
## \end{tabular}
```

```
knitr::kable(
  as.data.frame(table(reads$outlier)),
  booktabs = TRUE,
  row.names = FALSE,
  caption = 'The number of cells removed by automatic filter (FALSE)'
)
```

```
def <- colnames(reads)[!reads$use_default]
auto <- colnames(reads)[reads$outlier]
man <- colnames(reads)[!reads$use]
venn.diag <- limma::vennCounts(cbind(colnames(reads) %in% def,
                                     colnames(reads) %in% auto,
                                     colnames(reads) %in% man))
```

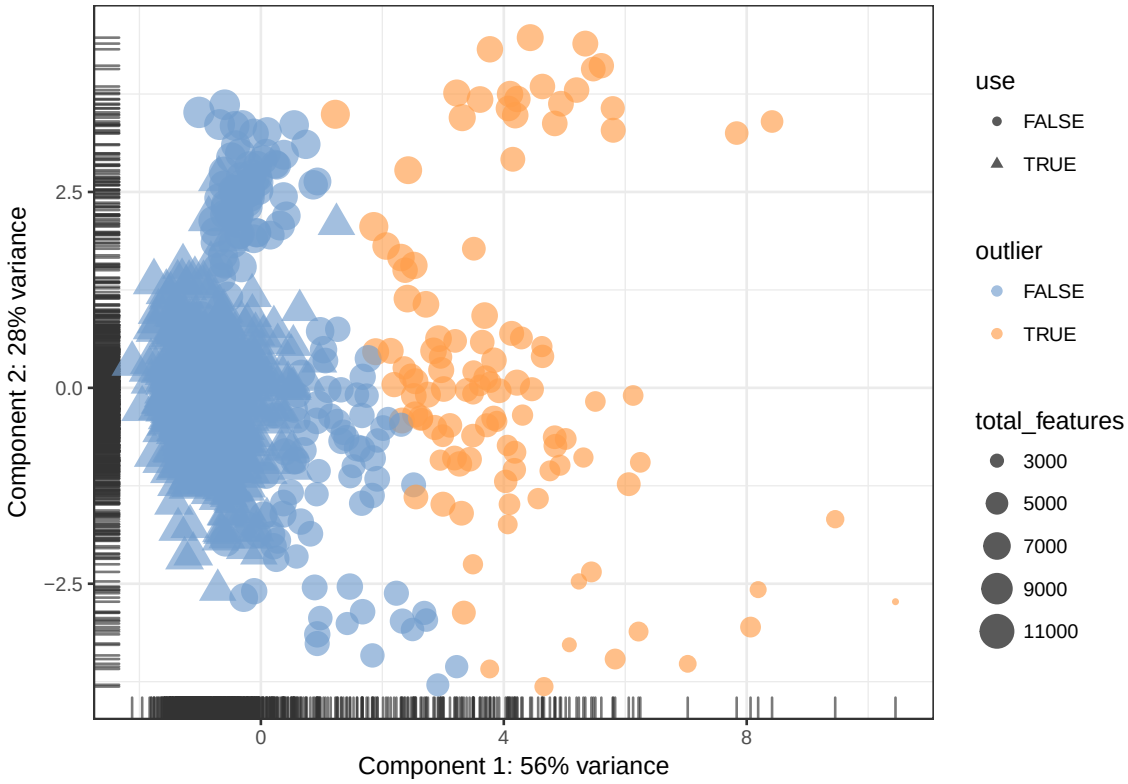


Figure 8.5: PCA plot used for automatic detection of cell outliers

Table 8.9: The number of cells removed by automatic filter (FALSE)

Var1	Freq
FALSE	753
TRUE	111

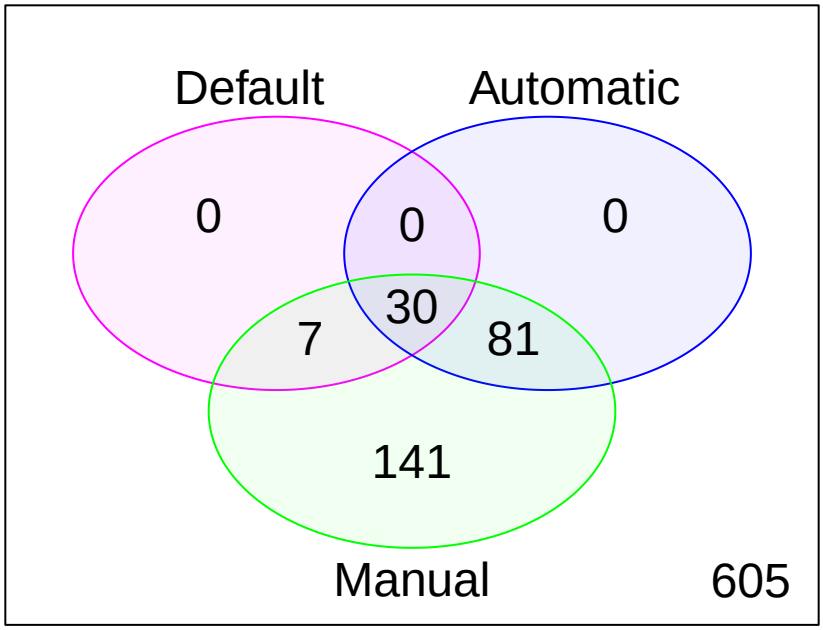


Figure 8.6: Comparison of the default, automatic and manual cell filters

Table 8.10: The number of genes removed by gene filter (FALSE)

filter_genes	Freq
FALSE	2665
TRUE	16061

```
limma::vennDiagram(venn.diag,
                   names = c("Default", "Automatic", "Manual"),
                   circle.col = c("magenta", "blue", "green"))
```

```
scater::plotQC(reads, type = "highest-expression")
```

```
filter_genes <- apply(counts(reads[, pData(reads)$use]), 1,
                      function(x) length(x[x > 1]) >= 2)
fData(reads)$use <- filter_genes
```

```
knitr::kable(
    as.data.frame(table(filter_genes)),
    booktabs = TRUE,
    row.names = FALSE,
    caption = 'The number of genes removed by gene filter (FALSE)'
)
```

```
dim(reads[fData(reads)$use, pData(reads)$use])
```

```
## Features  Samples
##    16061      605
```

```
saveRDS(reads, file = "blischak/reads.rds")
```

If you want to further check yourself you can download our `reads` object. If you followed the steps above it

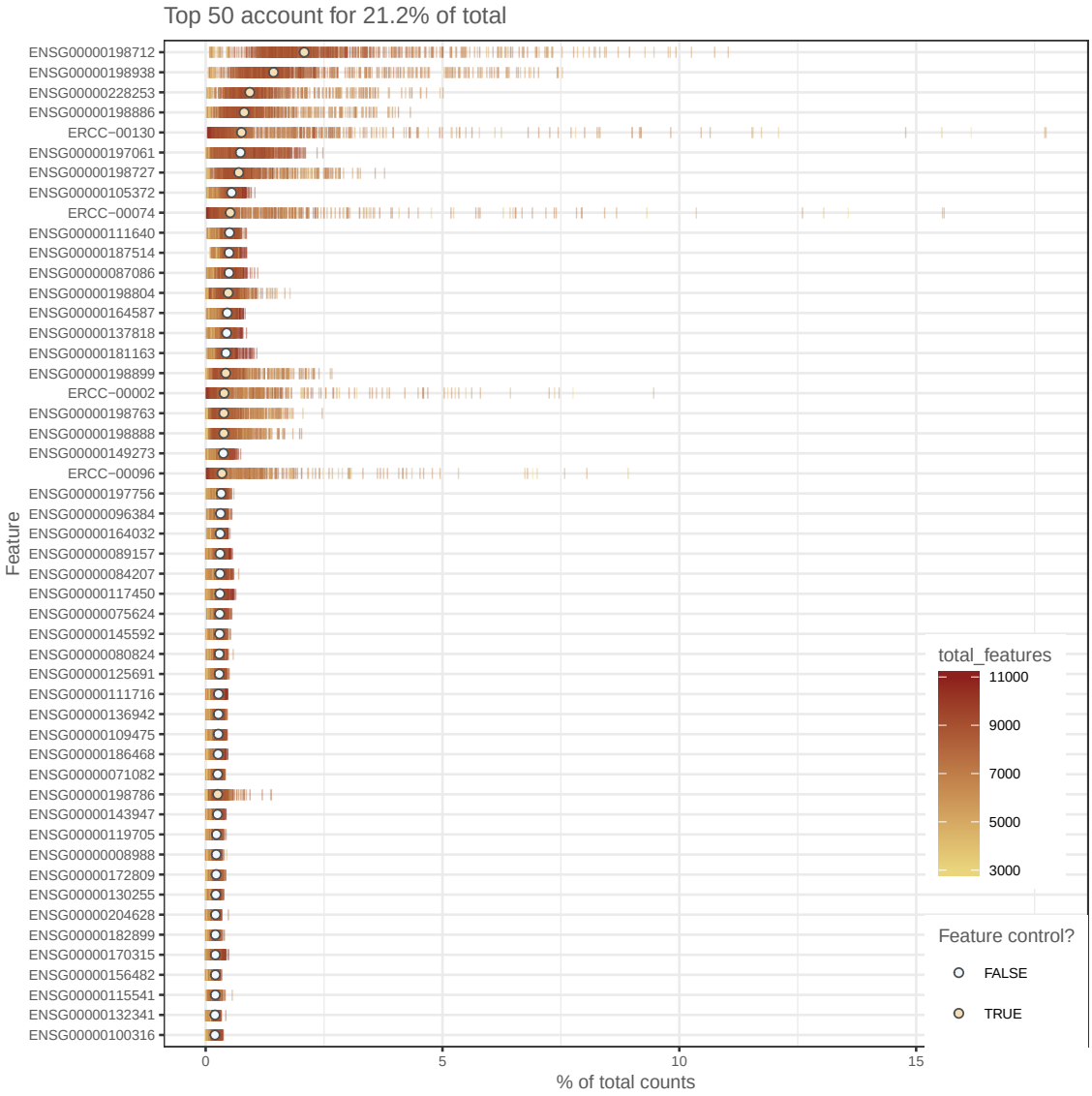


Figure 8.7: Number of total counts consumed by the top 50 expressed genes

should be exactly the same as yours.

By comparing Figure 7.6 and Figure 8.6, it is clear that the reads based filtering removed 49 more cells than the UMI based analysis. If you go back and compare the results you should be able to conclude that the ERCC and MT filters are more strict for the reads-based analysis.

Chapter 9

Data visualization

9.1 Introduction

In this chapter we will continue to work with the filtered **blischak** dataset produced in the previous chapter. We will explore different ways of visualizing the data to allow you to asses what happened to the expression matrix after the quality control step. scater package provides several very useful functions to simplify visualisation.

One important aspect of single-cell RNA-seq is to control for batch effects. Batch effects are technical artefacts that are added to the samples during handling. For example, if two sets of samples were prepared in different labs or even on different days in the same lab, then we may observe greater similarities between the samples that were handled together. In the worst case scenario, batch effects may be mistaken for true biological variation. The Blischak data allows us to explore these issues in a controlled manner since some of the salient aspects of how the samples were handled have been recorded. Ideally, we expect to see batches from the same individual grouping together and distinct groups corresponding to each individual.

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

9.2 PCA plot

The easiest thing to overview the data is to transform it using the principal component analysis and then visualize the first two principal components.

Principal component analysis (PCA) is a statistical procedure that uses a transformation to convert a set of observations into a set of values of linearly uncorrelated variables called principal components (PCs). The number of principal components is less than or equal to the number of original variables.

Mathematically, the PCs correspond to the eigenvectors of the covariance matrix. Typically, the eigenvectors are sorted by eigenvalue so that the first principal component accounts for as much of the variability in the data as possible, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components (the figure below is taken from here).

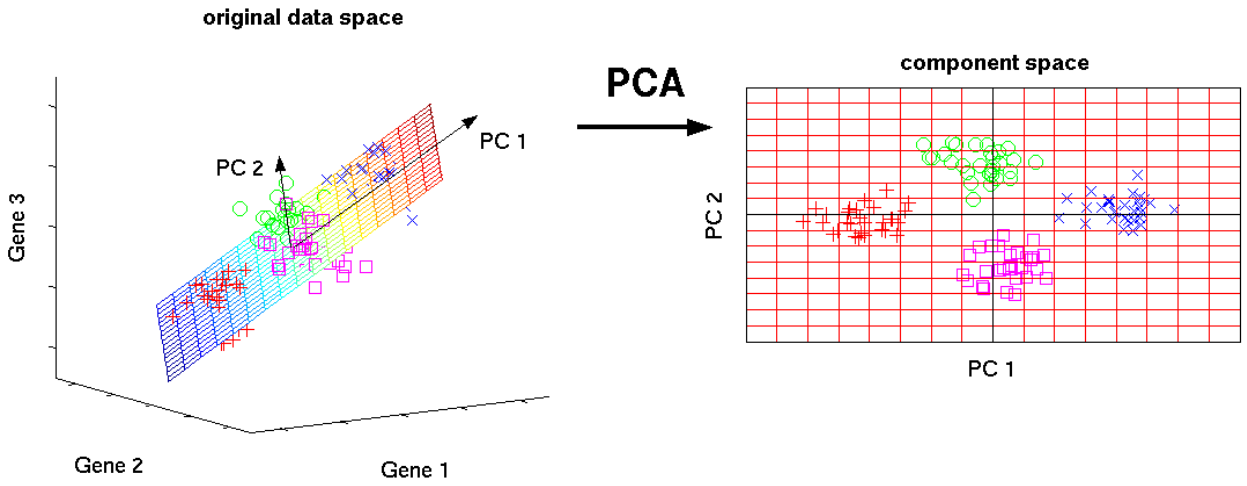


Figure 9.1: Schematic representation of PCA dimensionality reduction

9.2.1 Before QC

```
scater::plotPCA(umi[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "counts")
```

9.2.2 After QC

```
scater::plotPCA(umi.qc[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "counts")
```

Comparing Figure 9.2 and Figure 9.3, it is clear that after quality control the NA19098.r2 cells no longer form a group of outliers.

By default only the top 500 most variable genes are used by scater to calculate the PCA. This can be adjusted by changing the `ntop` argument.

Exercise 1 How do the PCA plots change if when all 14,214 genes are used? Or when only top 50 genes are used?

Our answer

If your answers are different please compare your code with ours (you need to search for this exercise in the opened file).

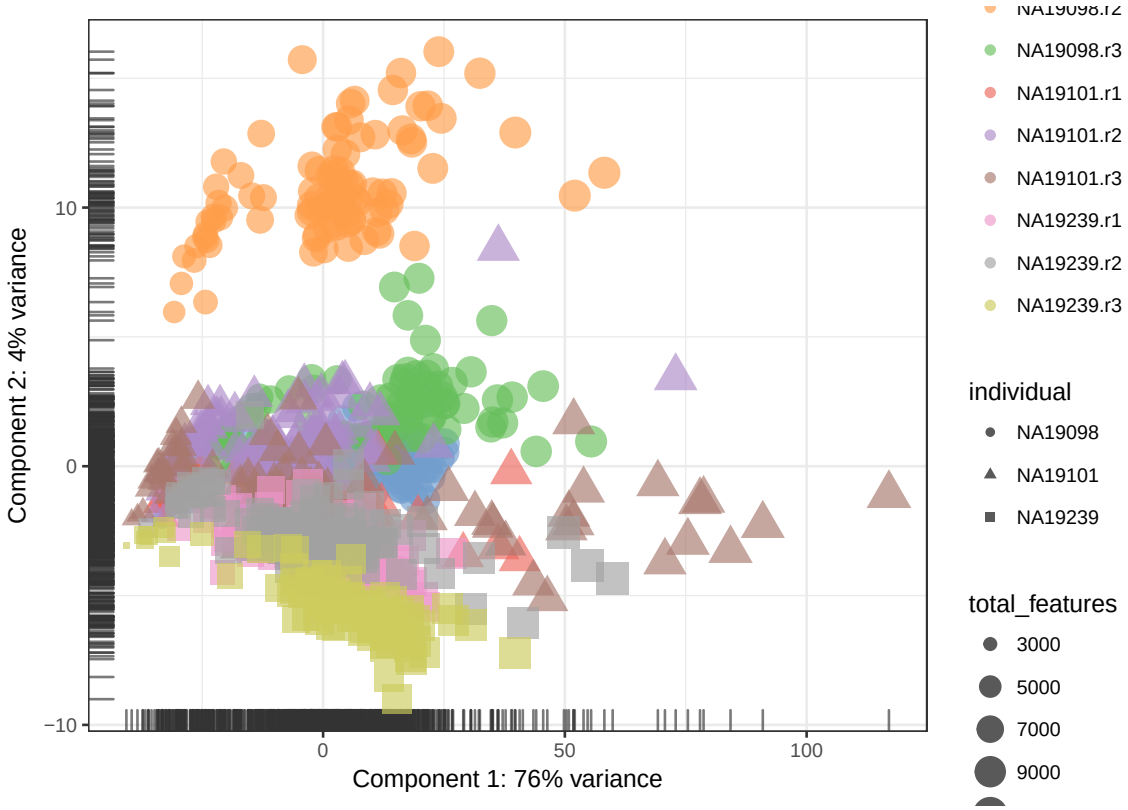


Figure 9.2: PCA plot of the blischak data

9.3 tSNE map

An alternative to PCA for visualizing scRNASeq data is a tSNE plot. tSNE (t-Distributed Stochastic Neighbor Embedding) combines dimensionality reduction (e.g. PCA) with random walks on the nearest-neighbour network to map high dimensional data (i.e. our 14,214 dimensional expression matrix) to a 2-dimensional space while preserving local distances between cells. In contrast with PCA, tSNE is a stochastic algorithm which means running the method multiple times on the same dataset will result in different plots. Due to the non-linear and stochastic nature of the algorithm, tSNE is more difficult to intuitively interpret tSNE. To ensure reproducibility, we fix the "seed" of the random-number generator in the code below so that we always get the same plot.

9.3.1 Before QC

```
scater::plotTSNE(umi[endog_genes, ],
                 ntop = 500,
                 perplexity = 130,
                 colour_by = "batch",
                 size_by = "total_features",
                 shape_by = "individual",
                 exprs_values = "counts",
                 rand_seed = 123456)
```

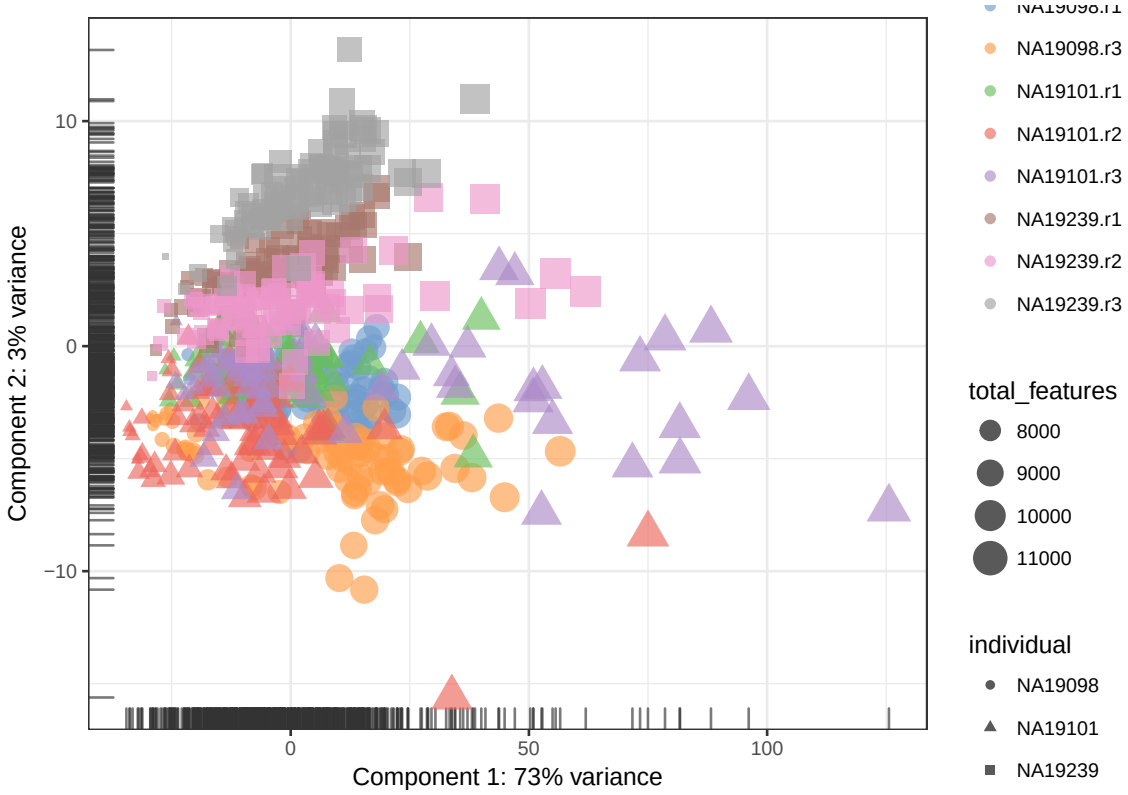


Figure 9.3: PCA plot of the blischak data

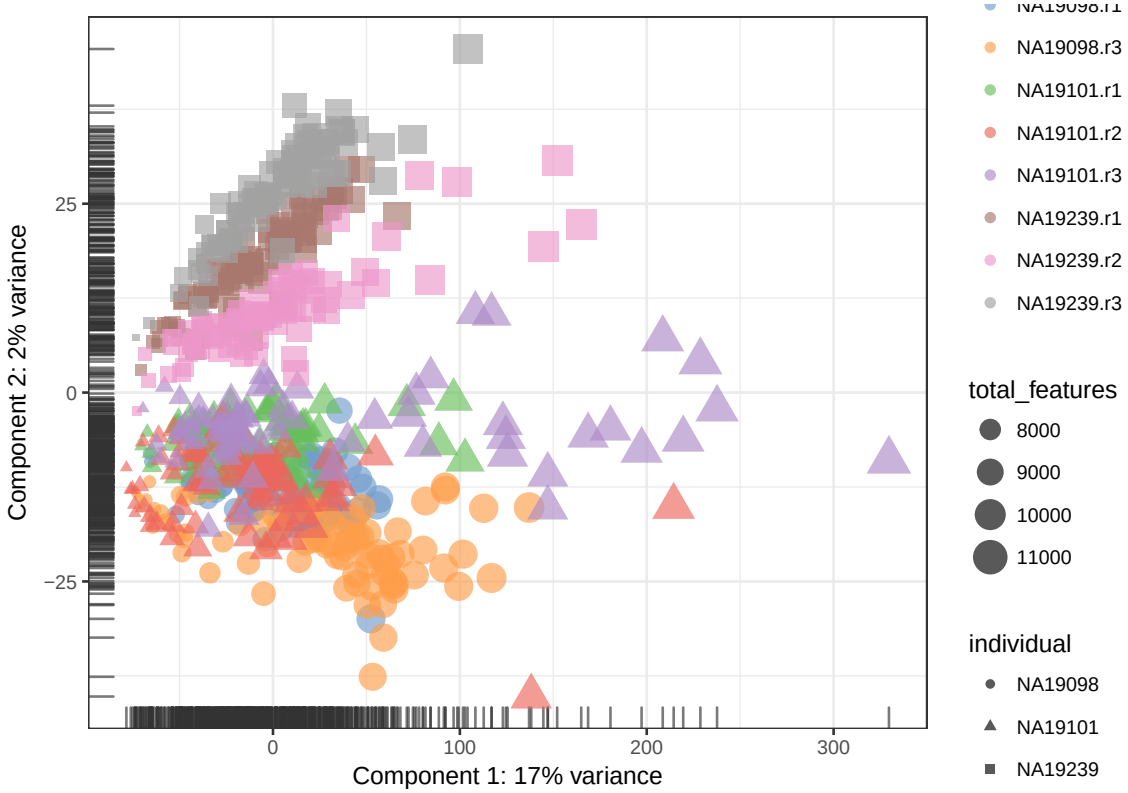


Figure 9.4: PCA plot of the blischak data (14214 genes)

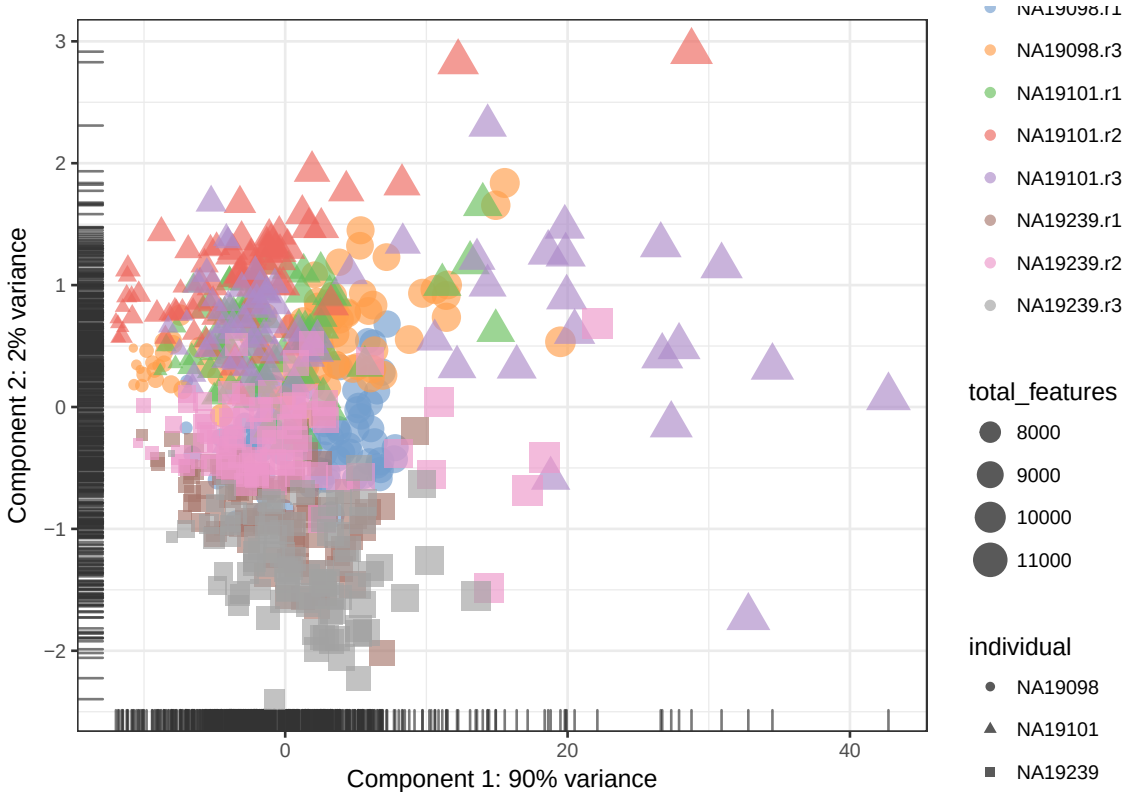


Figure 9.5: PCA plot of the blischak data (50 genes)

9.3.2 After QC

```
scater::plotTSNE(umi.qc[endog_genes, ],
                 ntop = 500,
                 perplexity = 130,
                 colour_by = "batch",
                 size_by = "total_features",
                 shape_by = "individual",
                 exprs_values = "counts",
                 rand_seed = 123456)
```

Interpreting PCA and tSNE plots is often challenging and due to their stochastic and non-linear nature, they are less intuitive. However, in this case it is clear that they provide a similar picture of the data. Comparing Figure 9.6 and 9.7, it is again clear that the samples from NA19098.r2 are no longer outliers after the QC filtering.

Furthermore tSNE requires you to provide a value of "perplexity" which reflects the number of neighbours used to build the nearest-neighbour network; a high value creates a dense network which clumps cells together while a low value makes the network more sparse allowing groups of cells to separate from each other. **scater** uses a default perplexity of the total number of cells divided by five (rounded down).

You can read more about the pitfalls of using tSNE here.

Exercise 2 How do the tSNE plots change when a perplexity of 10 or 200 is used?

Our answer

If your answers are different please compare your code with ours (you need to search for this exercise in the

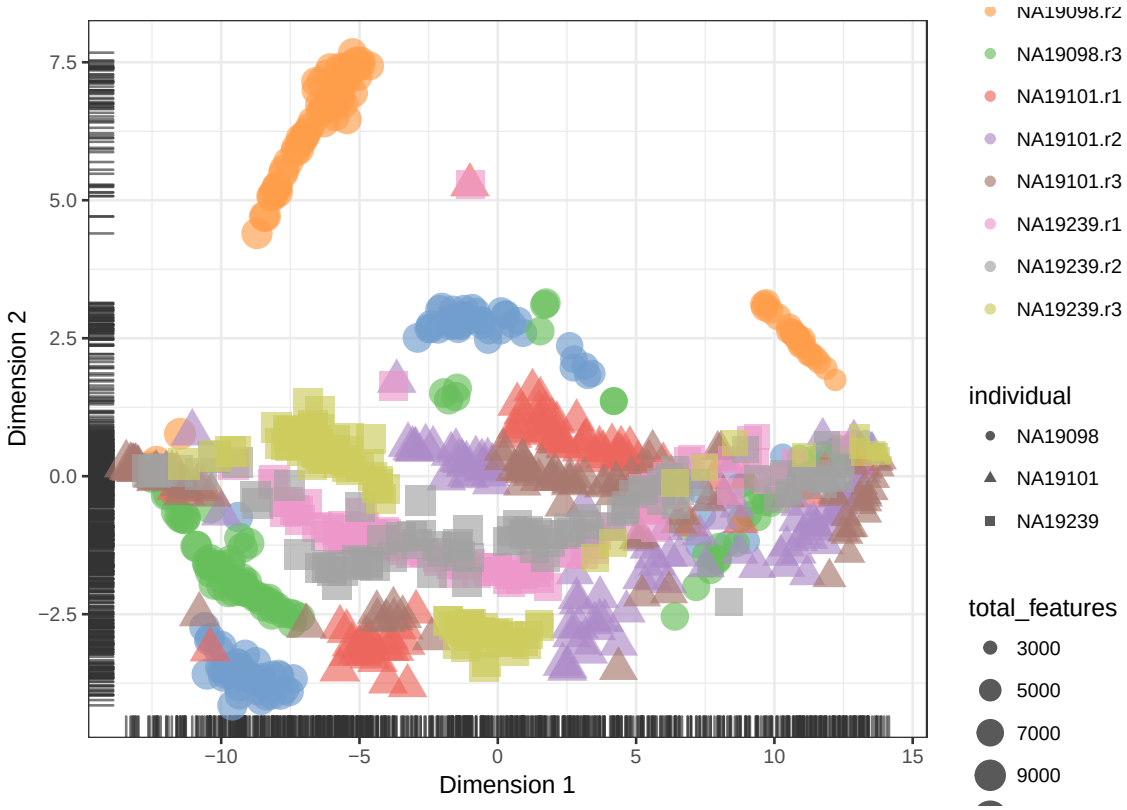


Figure 9.6: tSNE map of the blischak data

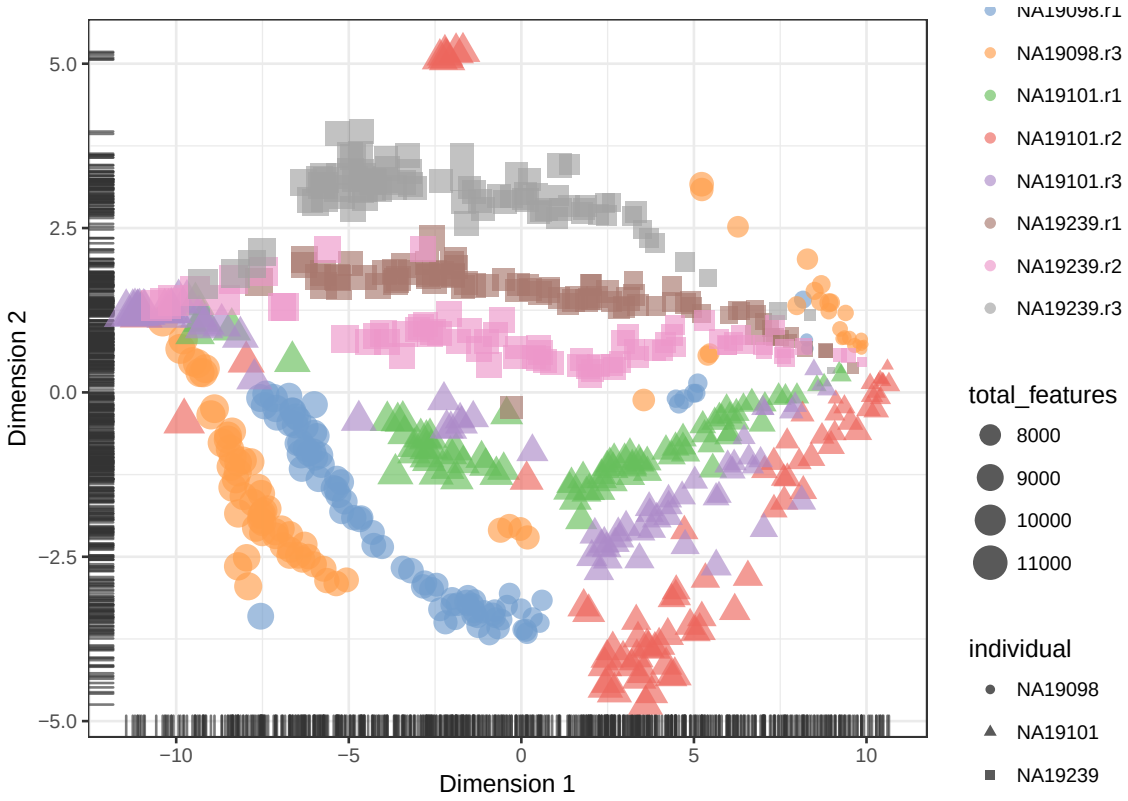


Figure 9.7: tSNE map of the blischak data

opened file).

9.4 Big Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter).

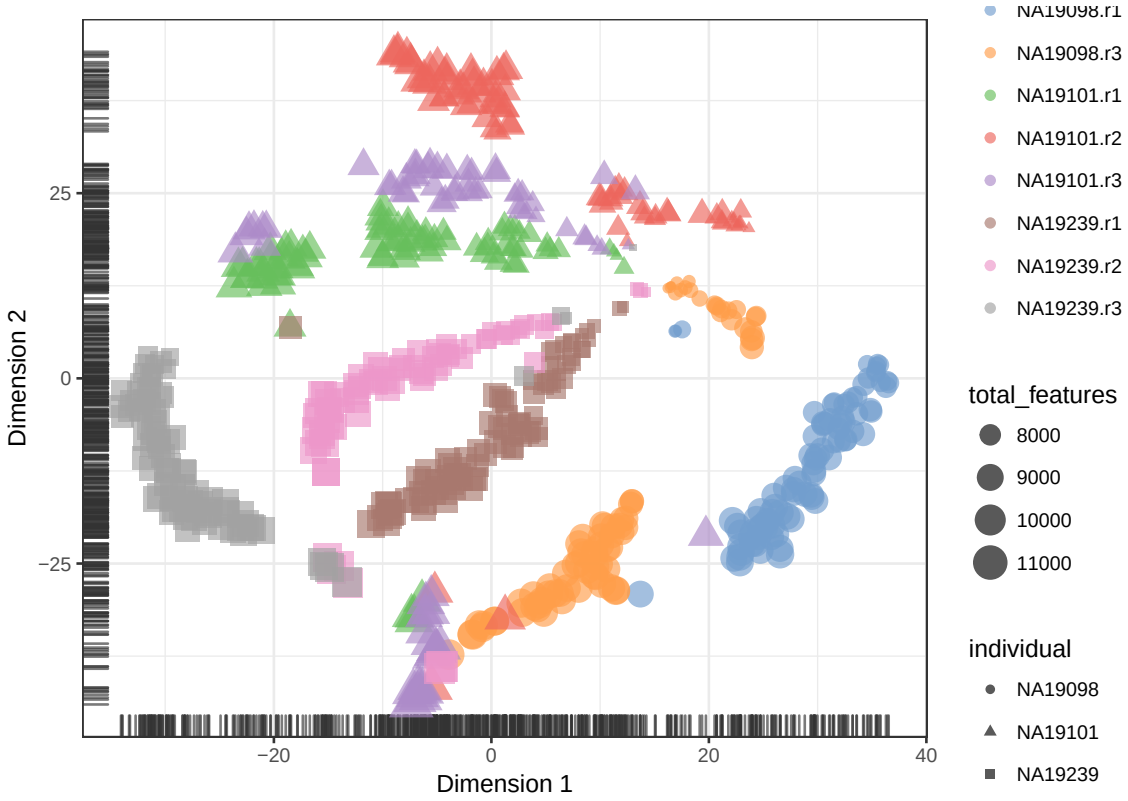


Figure 9.8: tSNE map of the blischak data (perplexity = 10)

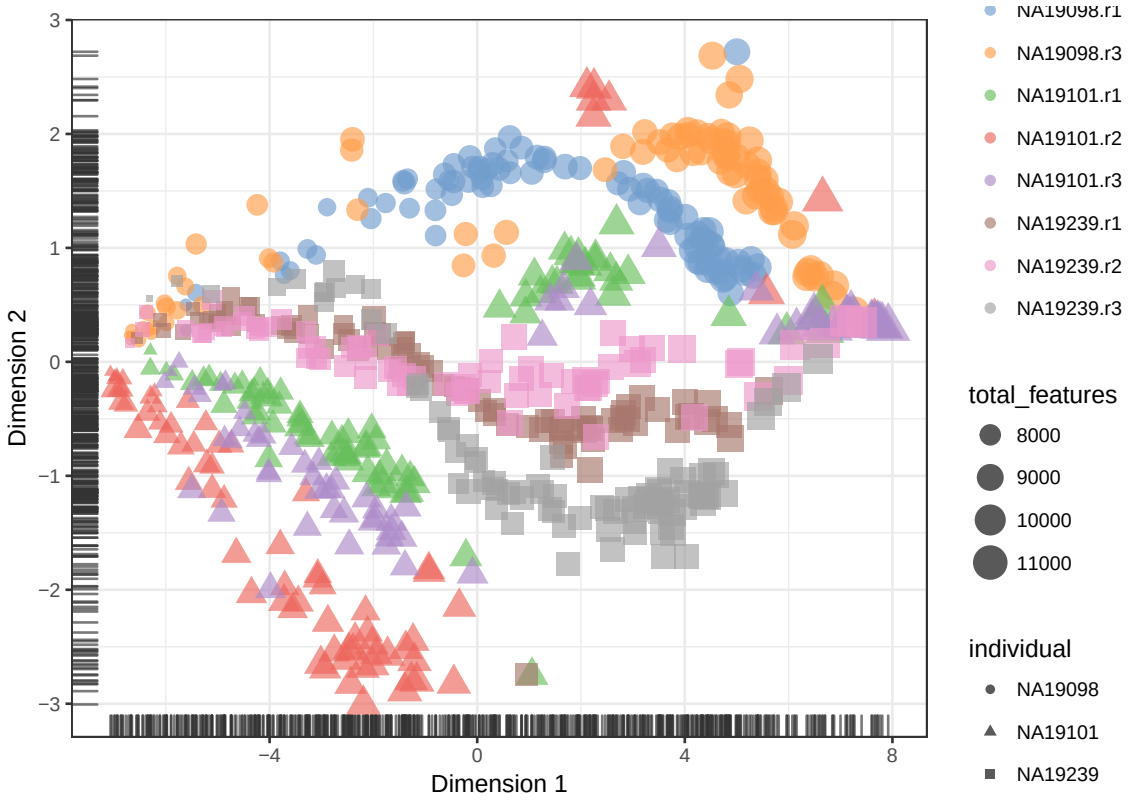


Figure 9.9: tSNE map of the blischak data (perplexity = 200)

Chapter 10

Data visualization (Reads)

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
reads <- readRDS("blischak/reads.rds")
reads.qc <- reads[fData(reads)$use, pData(reads)$use]
endog_genes <- !fData(reads.qc)$is_feature_control
```

```
scater::plotPCA(reads[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "counts")
```

```
scater::plotPCA(reads.qc[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "counts")
```

```
scater::plotTSNE(reads[endog_genes, ],
                 ntop = 500,
                 perplexity = 130,
                 colour_by = "batch",
                 size_by = "total_features",
                 shape_by = "individual",
                 exprs_values = "counts",
                 rand_seed = 123456)
```

```
scater::plotTSNE(reads.qc[endog_genes, ],
                 ntop = 500,
                 perplexity = 130,
                 colour_by = "batch",
                 size_by = "total_features",
                 shape_by = "individual",
                 exprs_values = "counts",
                 rand_seed = 123456)
```

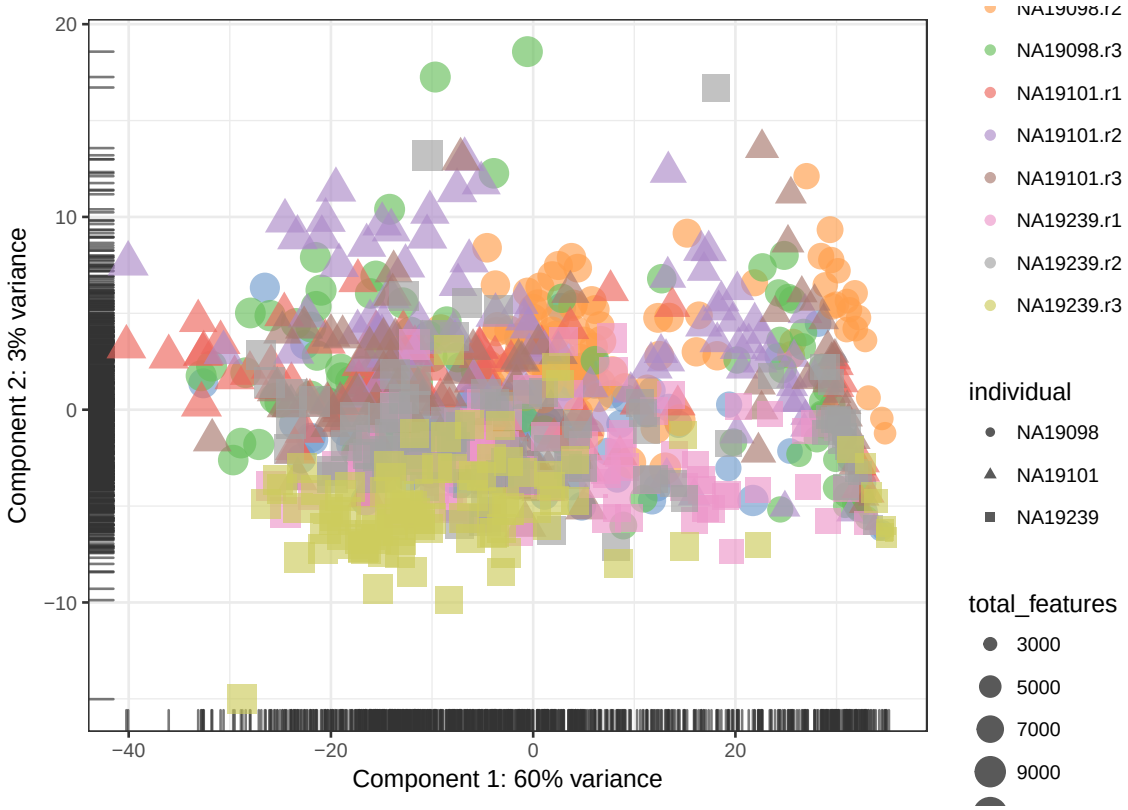


Figure 10.1: PCA plot of the blischak data

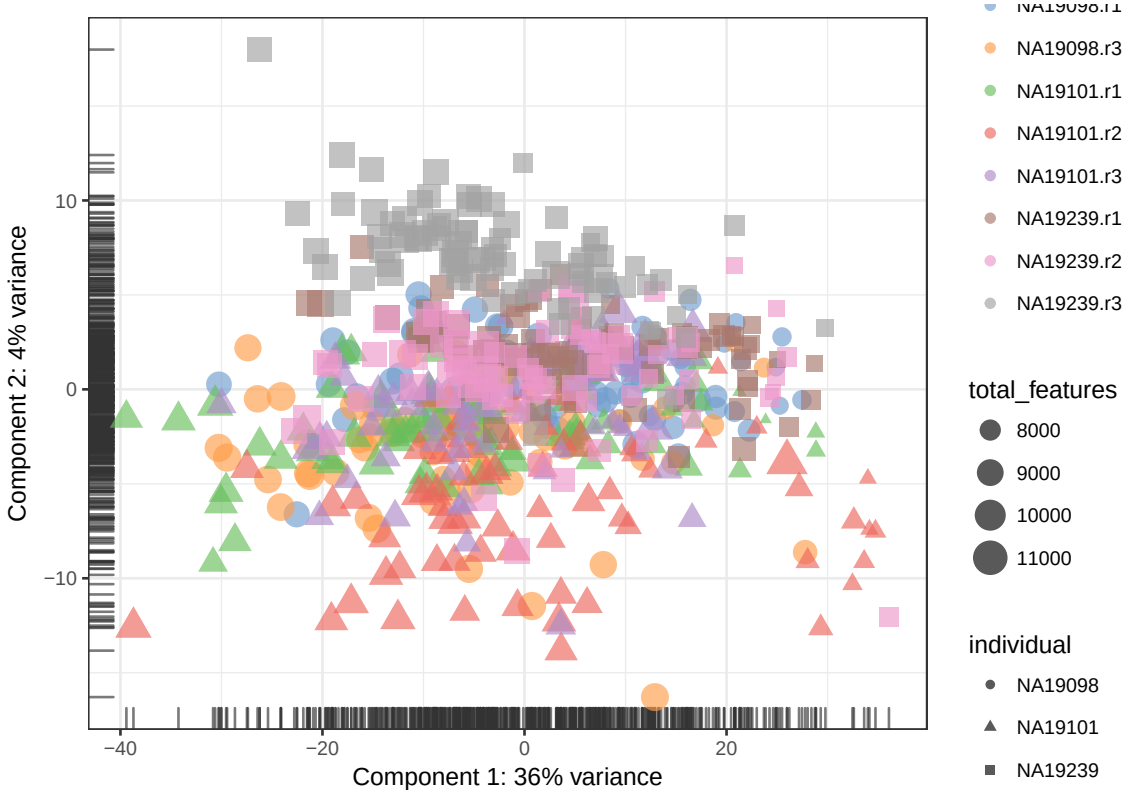


Figure 10.2: PCA plot of the blischak data

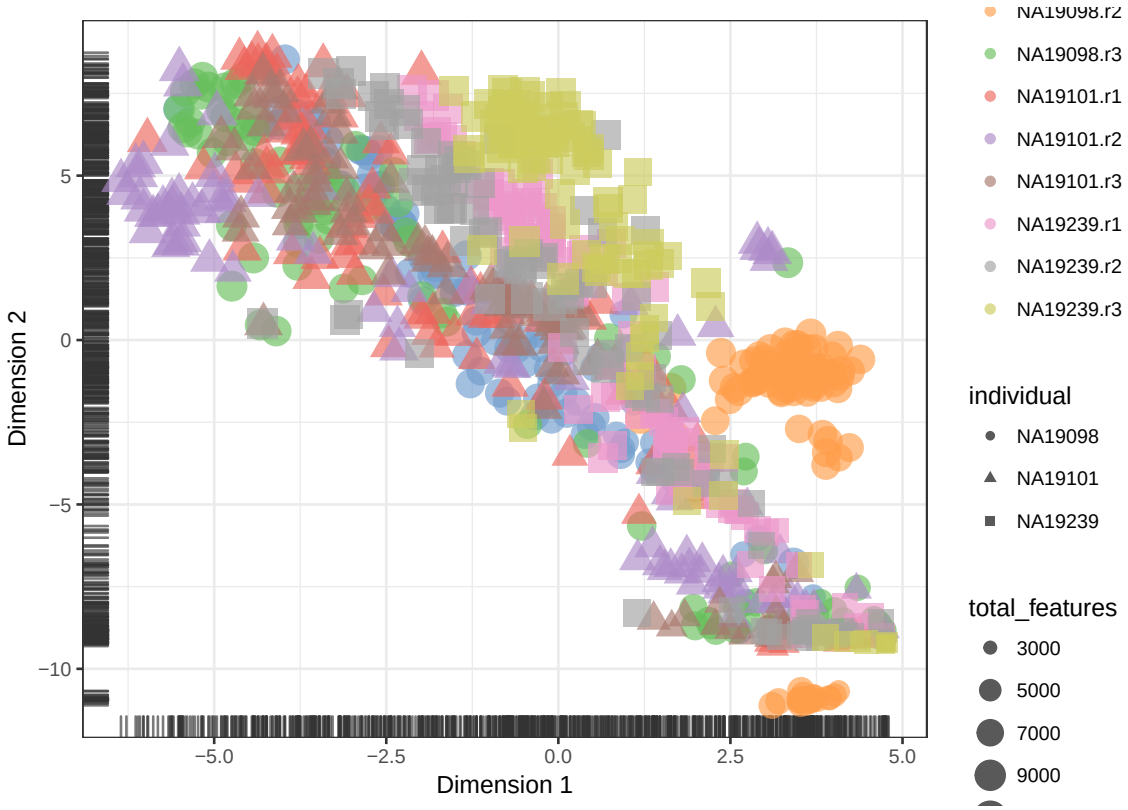


Figure 10.3: tSNE map of the blischak data

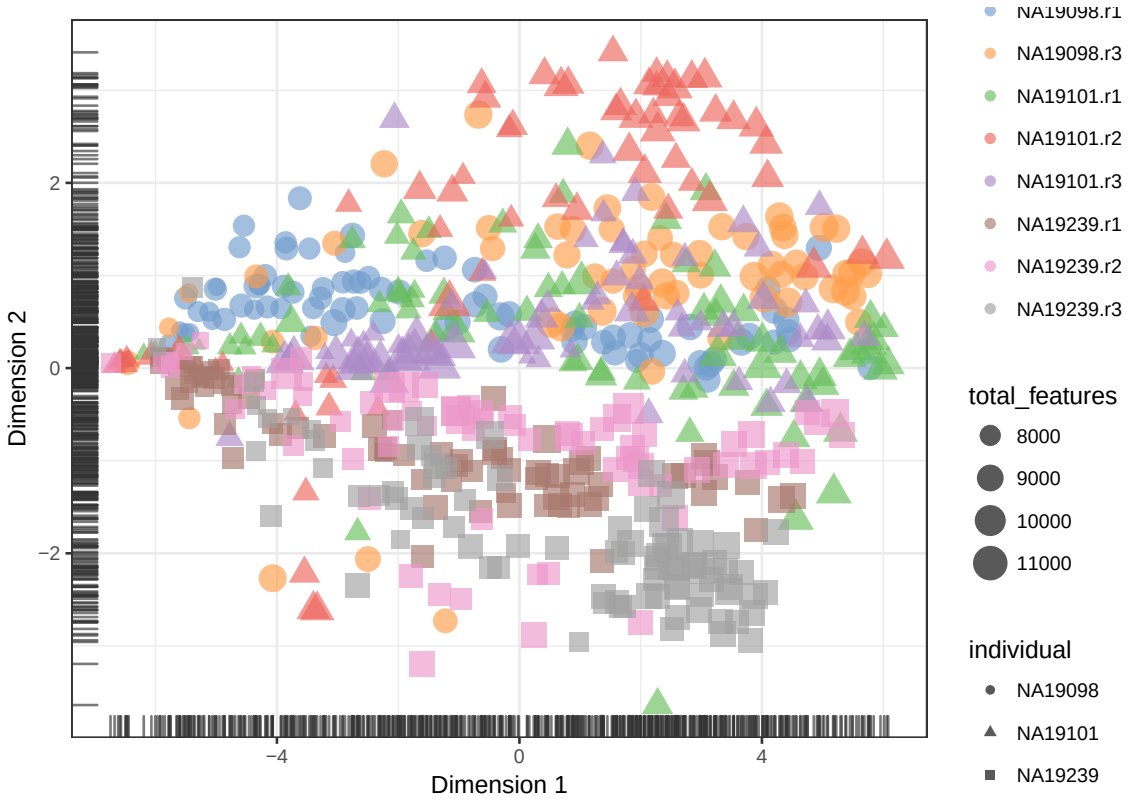


Figure 10.4: tSNE map of the blischak data

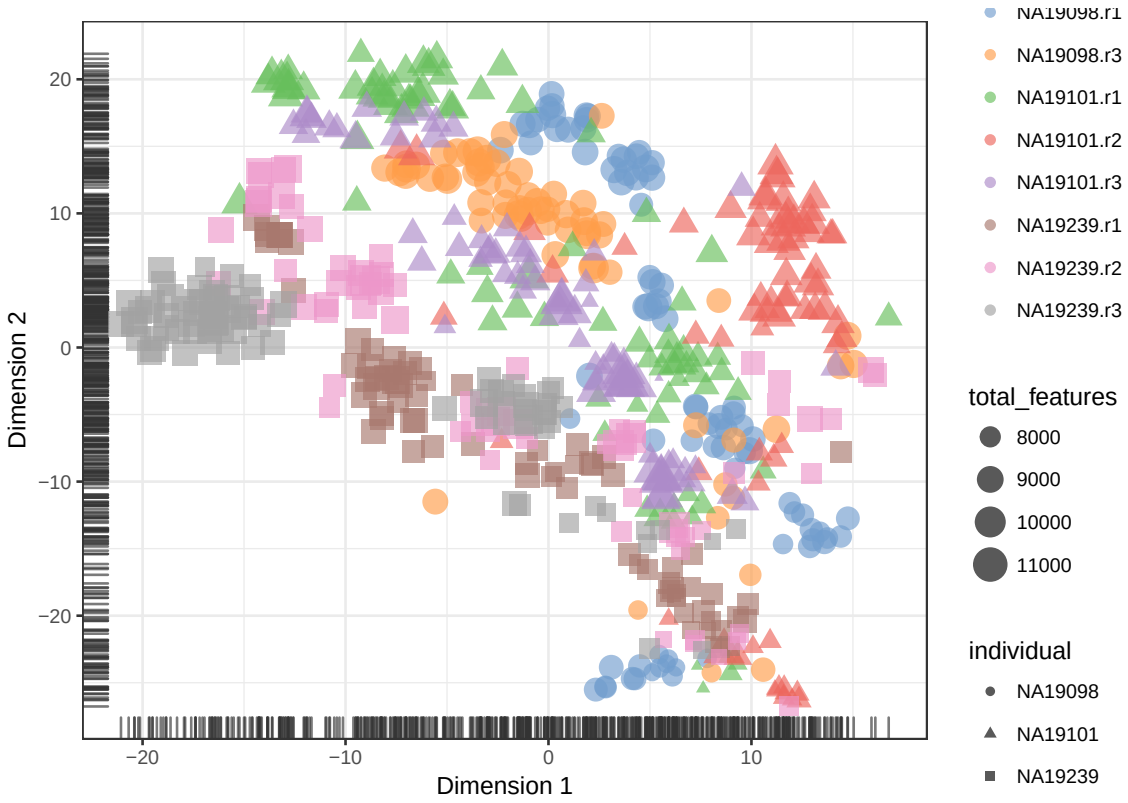


Figure 10.5: tSNE map of the blischak data (perplexity = 10)

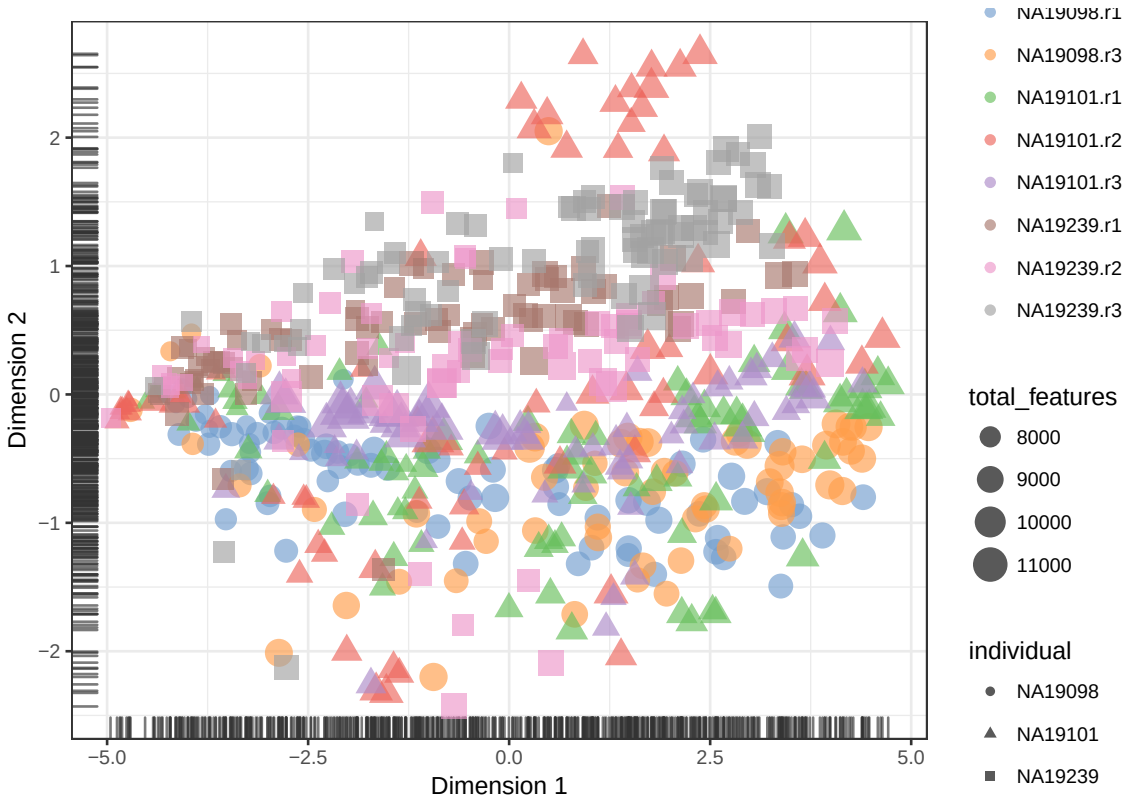


Figure 10.6: tSNE map of the blischak data (perplexity = 200)

Chapter 11

Confounding factors

11.1 Introduction

There is a large number of potential confounders, artifacts and biases in sc-RNA-seq data. One of the main challenges in analyzing scRNA-seq data stems from the fact that it is difficult to carry out a true technical replicate (why?) to distinguish biological and technical variability. In the previous chapters we considered batch effects and in this chapter we will continue to explore how experimental artifacts can be identified and removed. We will continue using the scater package since it provides a set of methods specifically for quality control of experimental and explanatory variables. Moreover, we will continue to work with the Blischak data that was used in the previous chapter.

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

The `umi.qc` dataset contains filtered cells and genes. Our next step is to explore technical drivers of variability in the data to inform data normalisation before downstream analysis.

11.2 Correlations with PCs

Let's first look again at the PCA plot of the QCed dataset:

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                exprs_values = "counts")
```

scater allows one to identify principle components that correlate with experimental and QC variables of interest (it ranks principle components by R^2 from a linear model regressing PC value against the variable of interest).

Let's test whether some of the variables correlate with any of the PCs.

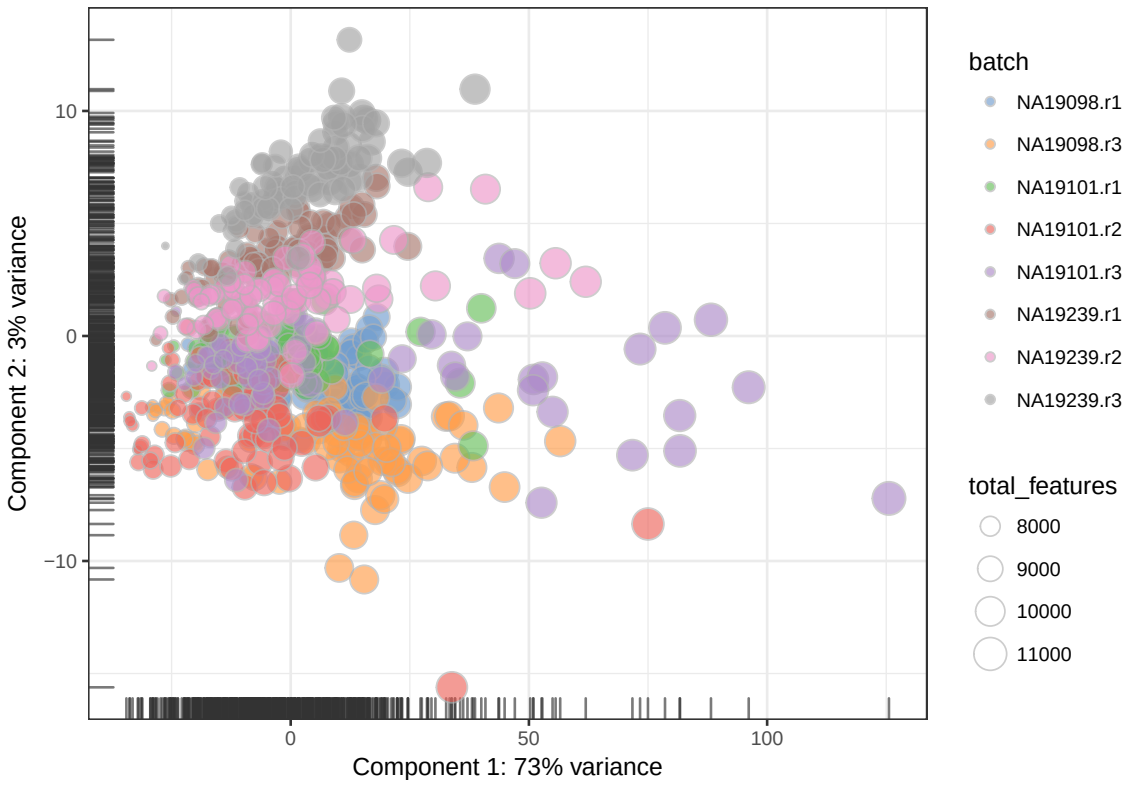


Figure 11.1: PCA plot of the blischak data

11.2.1 Detected genes

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "find-pcs",
               variable = "total_features",
               exprs_values = "counts")
```

Indeed, we can see that PC1 can be completely explained by the number of the detected genes. In fact, it was also visible on the PCA plot above. This is a well-known issue in scRNA-seq and was described here.

11.3 Explanatory variables

scater can also compute the marginal R^2 for each variable when fitting a linear model regressing expression values for each gene against just that variable, and display a density plot of the gene-wise marginal R^2 values for the variables.

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "expl",
               exprs_values = "counts",
               variables = c("total_features",
                             "total_counts",
                             "batch",
                             "individual",
                             "pct_counts_feature_controls_ERCC",
```

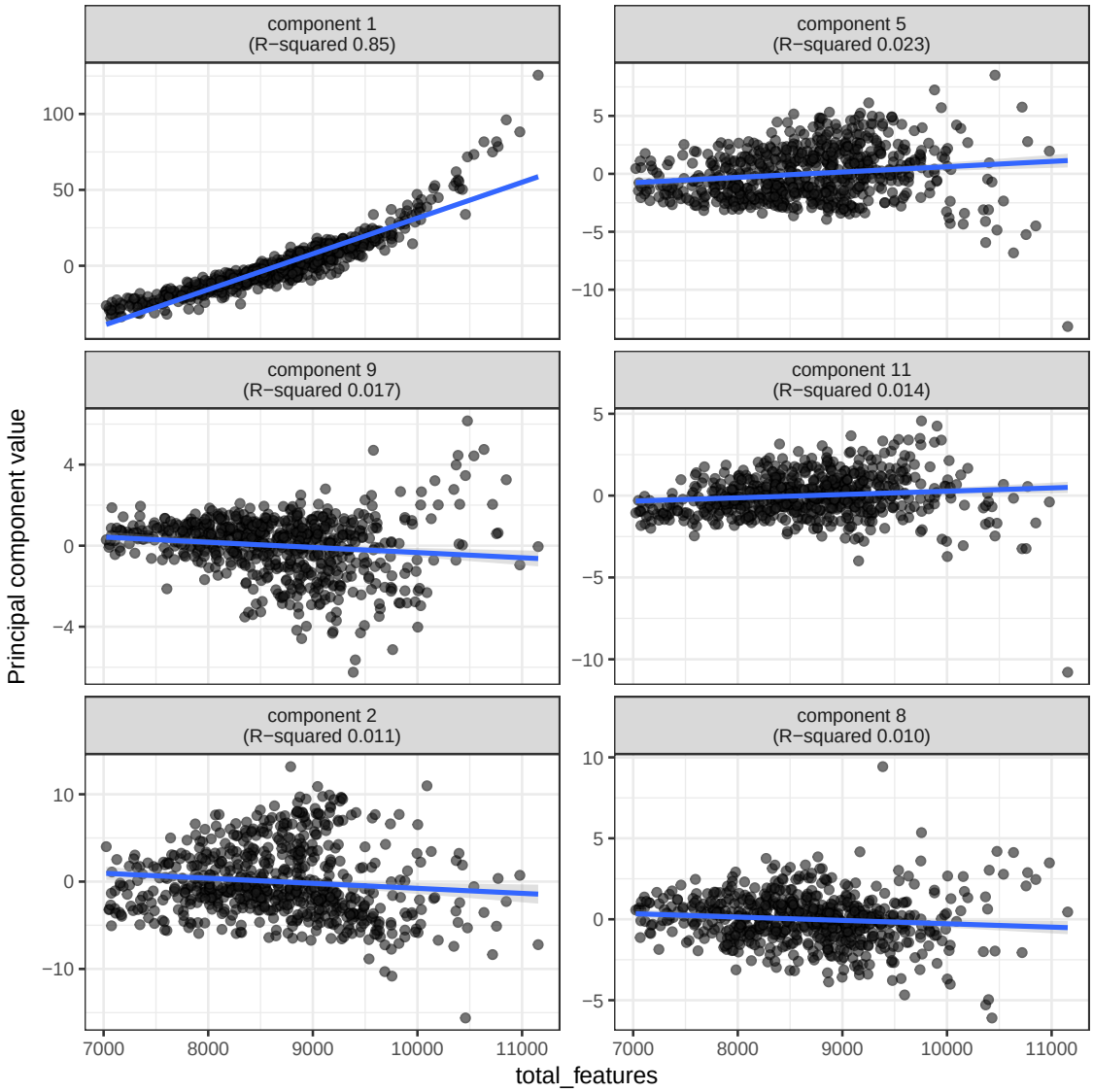


Figure 11.2: PC correlation with the number of detected genes

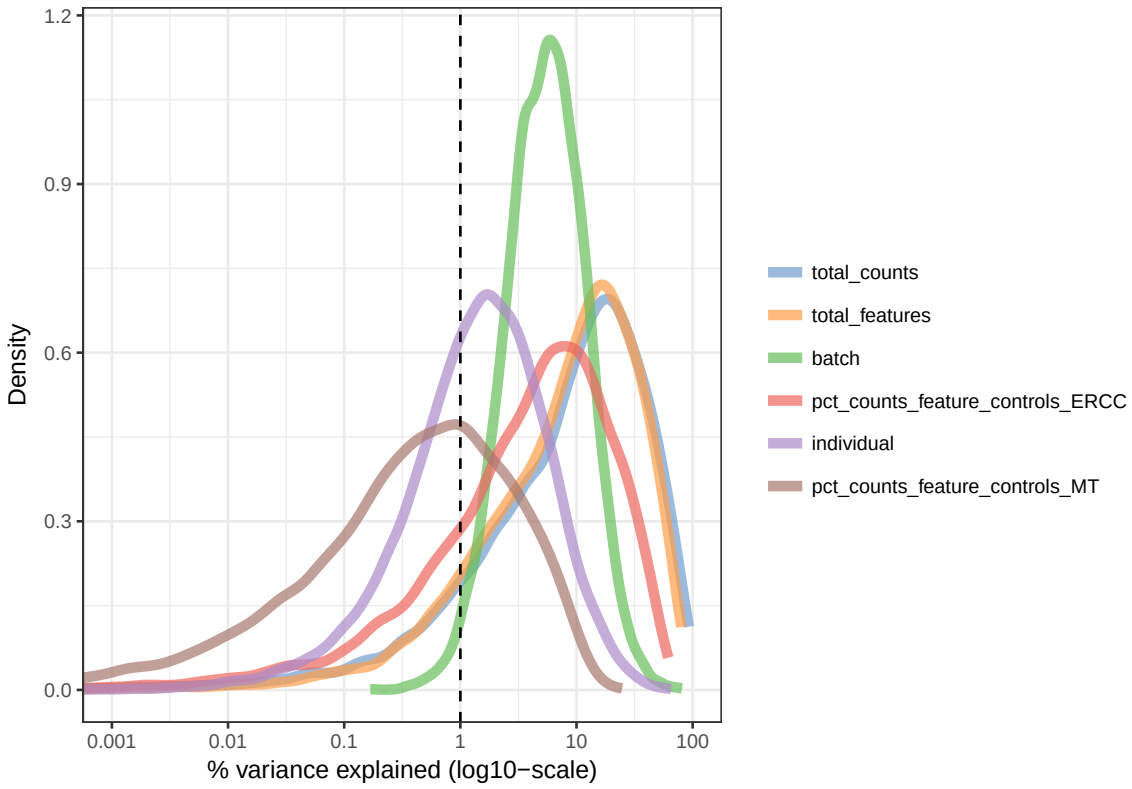


Figure 11.3: Explanatory variables

```
                                    "pct_counts_feature_controls_MT"))
```

This analysis indicates that the number of detected genes (again) and also the sequencing depth (number of counts) have substantial explanatory power for many genes, so these variables are good candidates for conditioning out in a normalisation step, or including in downstream statistical models. Expression of ERCCs also appears to be an important explanatory variable.

11.4 Other confounders

In addition to correcting for batch, there are other factors that one may want to compensate for. As with batch correction, these adjustments require extrinsic information. One popular method is scLVM which allows you to identify and subtract the effect from processes such as cell-cycle or apoptosis.

In addition, protocols may differ in terms of their coverage of each transcript, their bias based on the average content of **A/T** nucleotides, or their ability to capture short transcripts. Ideally, we would like to compensate for all of these differences and biases.

11.5 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter).

Chapter 12

Confounding factors (Reads)

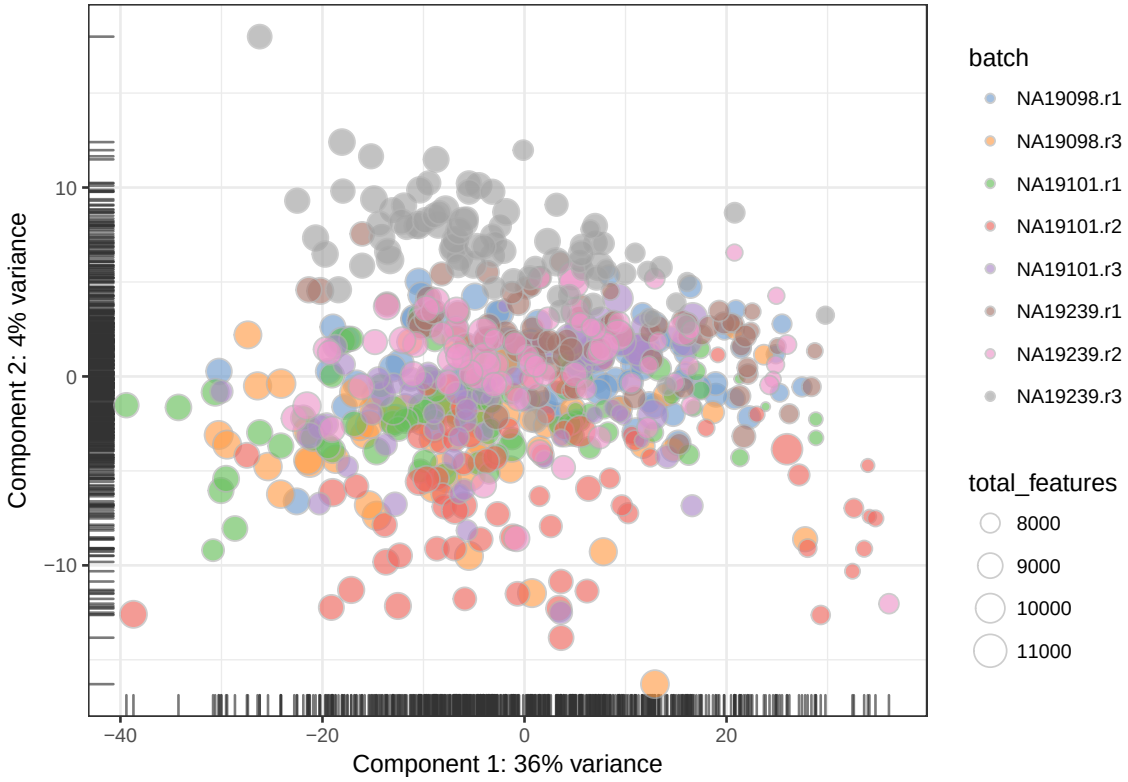


Figure 12.1: PCA plot of the blischak data

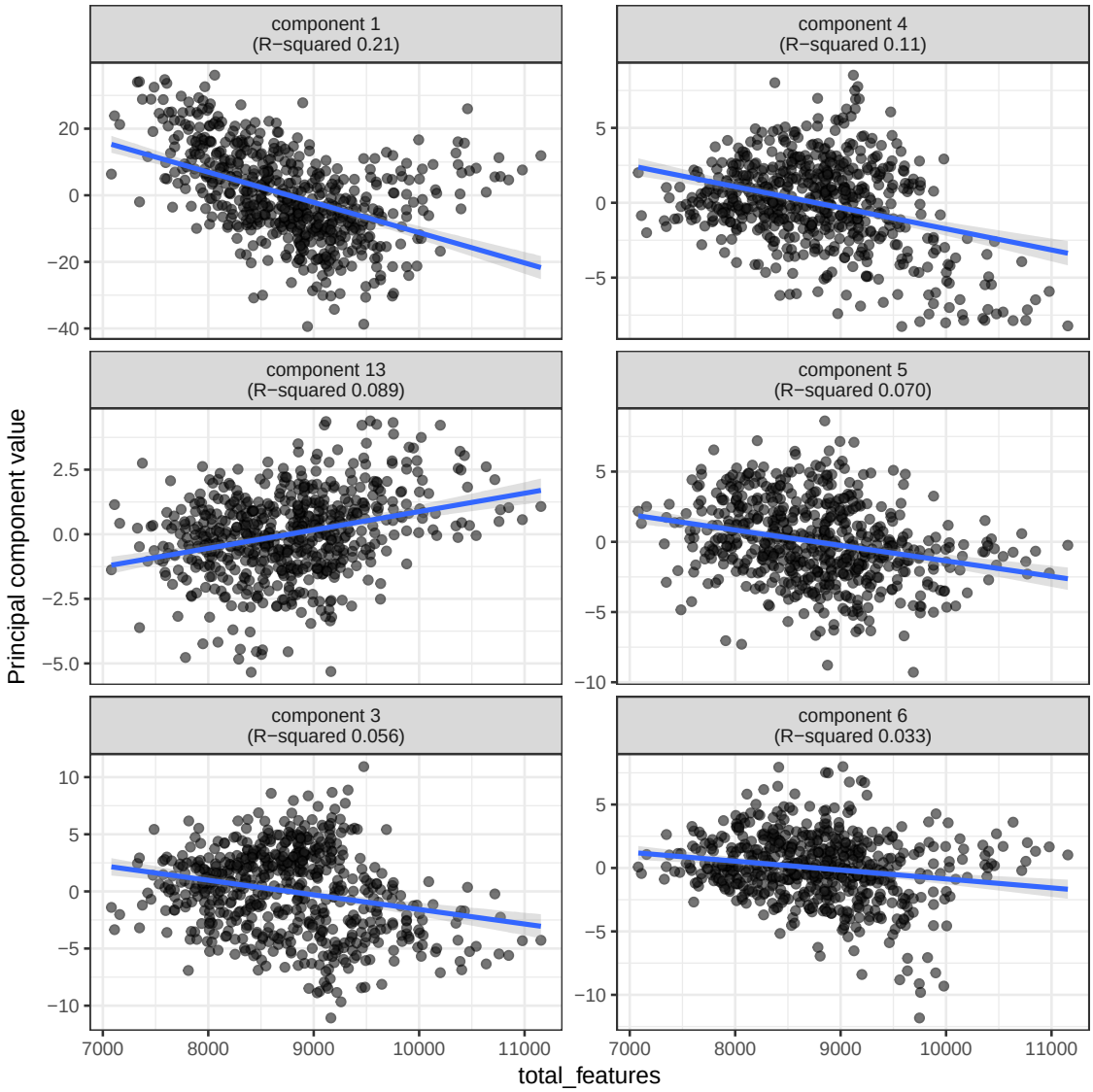


Figure 12.2: PC correlation with the number of detected genes

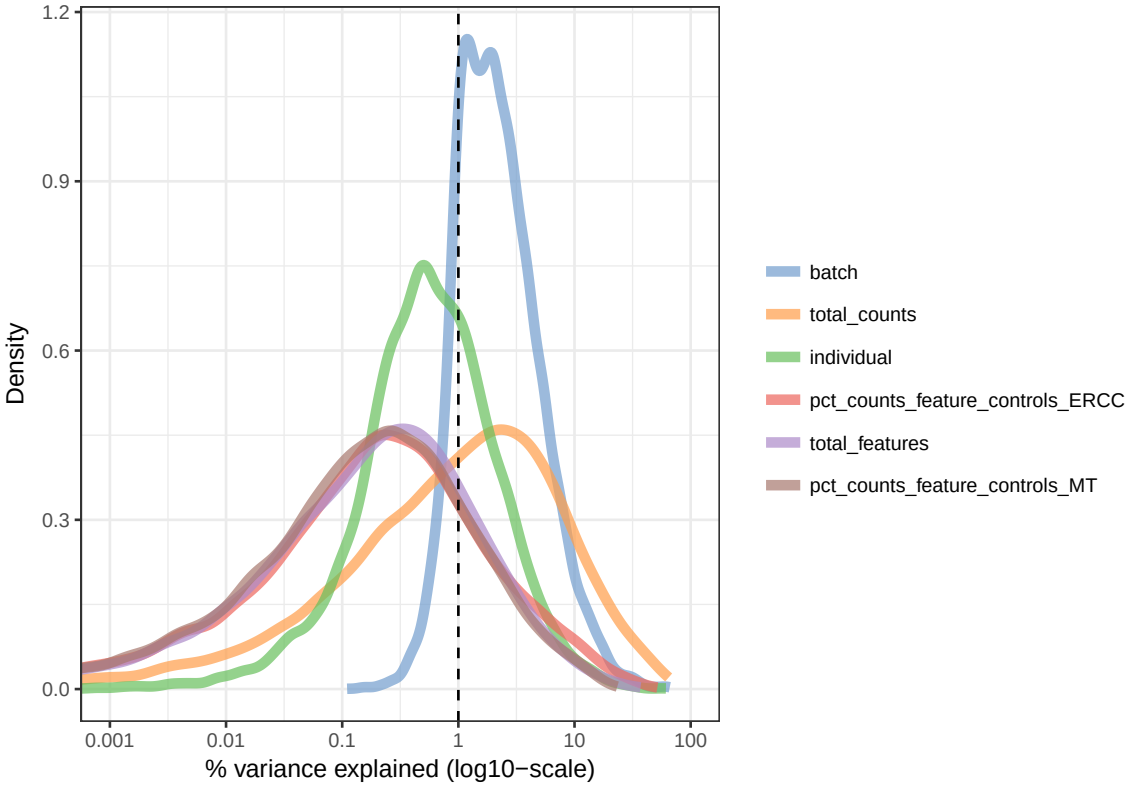


Figure 12.3: Explanatory variables

Chapter 13

Normalization for library size

13.1 Introduction

In the previous chapter we identified important confounding factors and explanatory variables. scater allows one to account for these variables in subsequent statistical models or to condition them out using `normaliseExprs()`, if so desired. This can be done by providing a design matrix to `normaliseExprs()`. We are not covering this topic here, but you can try to do it yourself as an exercise.

Instead we will explore how simple size-factor normalisations correcting for library size can remove the effects of some of the confounders and explanatory variables.

13.2 Library size

Library sizes vary because scRNA-seq data is often sequenced on highly multiplexed platforms the total reads which are derived from each cell may differ substantially. Some quantification methods (eg. Cufflinks, RSEM) incorporated library size when determining gene expression estimates thus do not require this normalization.

However, if another quantification method was used then library size must be corrected for by multiplying or dividing each column of the expression matrix by a "normalization factor" which is an estimate of the library size relative to the other cells. Many methods to correct for library size have been developped for bulk RNA-seq and can be equally applied to scRNA-seq (eg. UQ, SF, CPM, RPKM, FPKM, TPM).

13.3 Normalisations

The simplest way to normalize this data is to convert it to counts per million (**CPM**) by dividing each column by its total then multiplying by 1,000,000. Note that spike-ins should be excluded from the calculation of total expression in order to correct for total cell RNA content, therefore we will only use endogenous genes.

```
calc_cpm <-
function (expr_mat, spikes = NULL)
{
    norm_factor <- colSums(expr_mat[-spikes, ])
    return(t(t(expr_mat)/norm_factor)) * 10^6
}
```

One potential drawback of **CPM** is if your sample contains genes that are both very highly expressed and differentially expressed across the cells. In this case, the total molecules in the cell may depend of whether

such genes are on/off in the cell and normalizing by total molecules may hide the differential expression of those genes and/or falsely create differential expression for the remaining genes.

Note: RPKM, FPKM and TPM are variants on CPM which further adjust counts by the length of the respective gene/transcript.

To deal with this potentiality several other measures were devised:

The **size factor (SF)** was proposed and popularized by DESeq (Anders and Huber (2010)). First the geometric mean of each gene across all cells is calculated. The size factor for each cell is the median across genes of the ratio of the expression to the gene's geometric mean. A drawback to this method is that since it uses the geometric mean only genes with non-zero expression values across all cells can be used in its calculation, making it unadvisable for large low-depth scRNASeq experiments. edgeR & scater call this method **RLE** for "relative log expression".

```
calc_sf <-
function (expr_mat, spikes = NULL)
{
    geomeans <- exp(rowMeans(log(expr_mat[-spikes, ])))
    SF <- function(cnts) {
        median((cnts/geomeans)[(is.finite(geomeans) & geomeans >
            0)])
    }
    norm_factor <- apply(expr_mat[-spikes, ], 2, SF)
    return(t(t(expr_mat)/norm_factor))
}
```

The **upperquartile (UQ)** was proposed by Bullard et al (2010). Here each column is divided by the 75% quantile of the counts for each library. Often the calculated quantile is scaled by the median across cells to keep the absolute level of expression relatively consistent. A drawback to this method is that for low-depth scRNASeq experiments the large number of undetected genes may result in the 75% quantile being zero (or close to it). This limitation can be overcome by generalizing the idea and using a higher quantile (eg. the 99% quantile is the default in scater) or by excluding zeros prior to calculating the 75% quantile.

```
calc_uq <-
function (expr_mat, spikes = NULL)
{
    UQ <- function(x) {
        quantile(x[x > 0], 0.75)
    }
    uq <- unlist(apply(expr_mat[-spikes, ], 2, UQ))
    norm_factor <- uq/median(uq)
    return(t(t(expr_mat)/norm_factor))
}
```

Another method is called **TMM** is the weighted trimmed mean of M-values (to the reference) proposed by Robinson and Oshlack (2010). The M-values in question are the gene-wise log2 fold changes between individual cells. One cell is used as the reference then the M-values for each other cell is calculated compared to this reference. These values are then trimmed by removing the top and bottom ~30%, and the average of the remaining values is calculated by weighting them to account for the effect of the log scale on variance. Each non-reference cell is multiplied by the calculated factor. Two potential issues with this method are insufficient non-zero genes left after trimming, and the assumption that most genes are not differentially expressed.

Finally the `scran` package implements a variant on CPM specialized for single-cell data (Lun et al 2016). Briefly this method deals with the problem of vary large numbers of zero values per cell by pooling cells together calculating a normalization factor (similar to CPM) for the sum of each pool. Since each cell is found in many different pools, cell-specific factors can be deconvoluted from the collection of pool-specific

factors using linear algebra.

We will use visual inspection of PCA plots and calculation of cell-wise relative log expression (`calc_cell_RLE()`) to compare the efficiency of different normalization methods. Cells with many[few] reads have higher[lower] than median expression for most genes resulting in a positive[negative] RLE across the cell, whereas normalized cells have an RLE close to zero.

```
calc_cell_RLE <-
function (expr_mat, spikes = NULL)
{
    RLE_gene <- function(x) {
        if (median(unlist(x)) > 0) {
            log((x + 1)/(median(unlist(x)) + 1))/log(2)
        }
        else {
            rep(NA, times = length(x))
        }
    }
    if (!is.null(spikes)) {
        RLE_matrix <- t(apply(expr_mat[-spikes, ], 1, RLE_gene))
    }
    else {
        RLE_matrix <- t(apply(expr_mat, 1, RLE_gene))
    }
    cell_RLE <- apply(RLE_matrix, 2, median, na.rm = T)
    return(cell_RLE)
}
```

The **RLE**, **TMM**, and **UQ** size-factor methods were developed for bulk RNA-seq data and, depending on the experimental context, may not be appropriate for single-cell RNA-seq data, as their underlying assumptions may be problematically violated. Lun et al (2016) recently published a size-factor normalisation method specifically designed for scRNA-seq data and accounting for single-cell biases, which we will call **LSF** (Lun Sum Factors). Briefly, expression values are summed across pools of cells and the summed values are used to compute normalization size-factors per pool. The pool-based size factors can then be deconvolved into cell-specific size factors, which can be used for normalization in the same way as other size factors.

scater acts as a wrapper for the `calcNormFactors` function from edgeR which implements several library size normalization methods making it easy to apply any of these methods to our data. The **LSF** method is implementated in the Bioconductor package scran, which allows seamless integrationinto the `scater` workflow. (The `scran` package itself depends on `scater`).

Note: edgeR makes extra adjustments to some of the normalization methods which may result in somewhat different results than if the original methods are followed exactly, e.g. edgeR's and scater's "RLE" method which is based on the "size factor" used by DESeq may give different results to the `estimateSizeFactorsForMatrix` method in the DESeq/DESeq2 packages. In addition, some versions of edgeR will not calculate the normalization factors correctly unless `lib.size` is set at 1 for all cells.

We will continue to work with the Blischak data that was used in the previous chapter.

```
library(scRNA.seq.funcs)
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

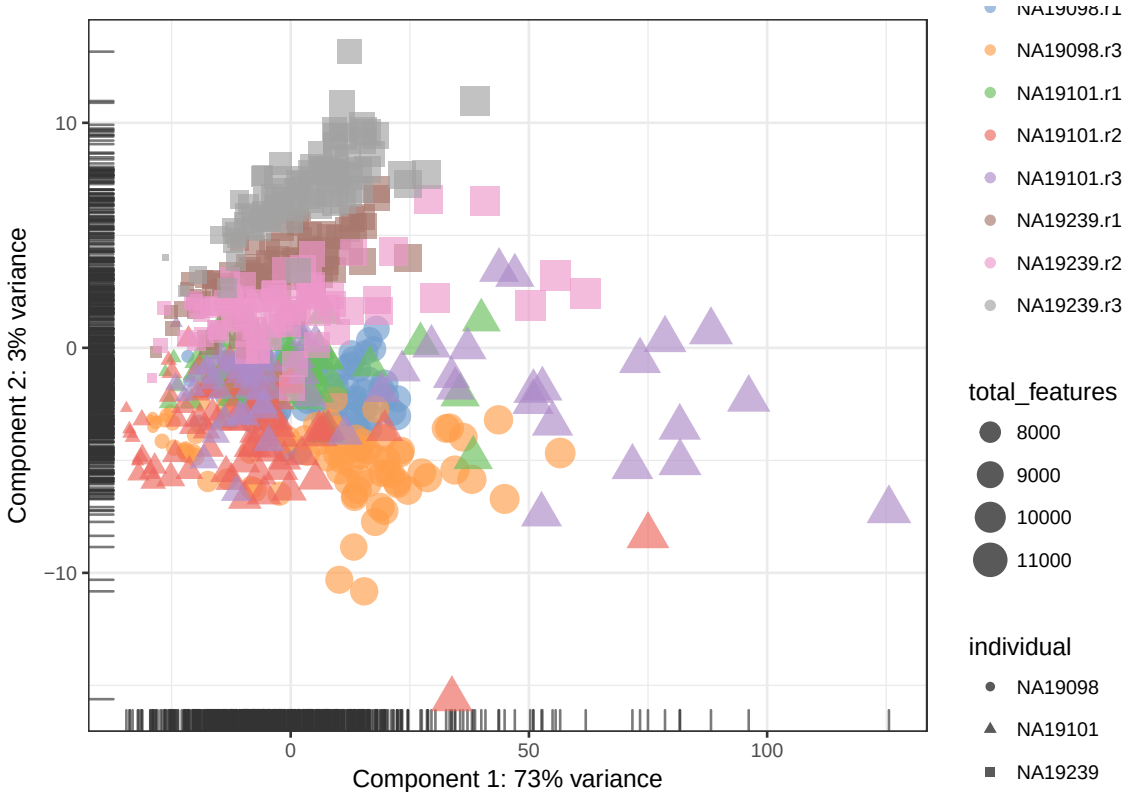


Figure 13.1: PCA plot of the blischak data

13.3.1 Raw

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "counts")
```

```
boxplot(calc_cell_RLE(counts(umi.qc[endog_genes, ])),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim=c(-1,1))
```

13.3.2 CPM

scater performs this normalisation by default, you can control it by changing `exprs_values` parameter to `"exprs"`.

```
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "exprs")
```

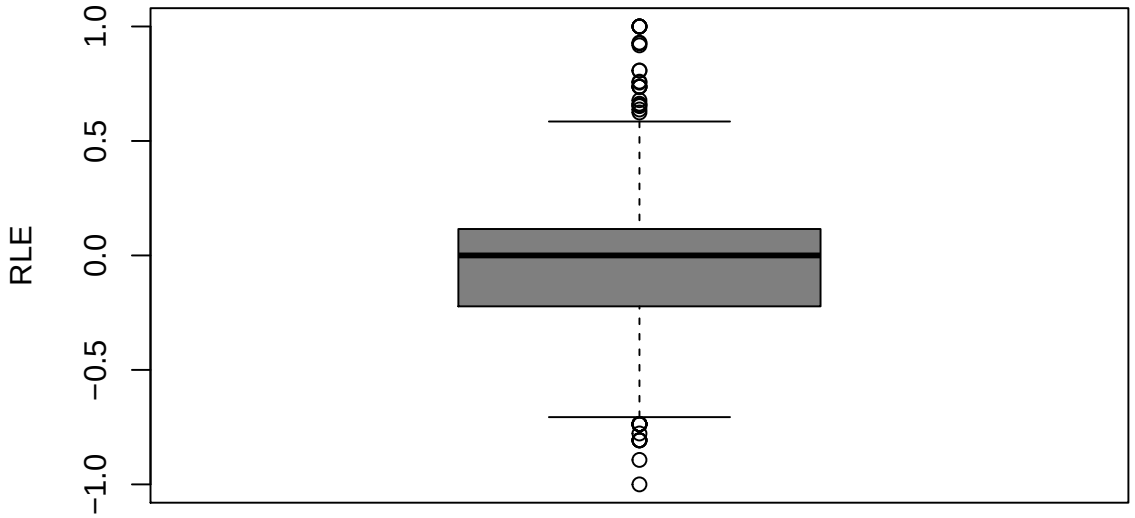


Figure 13.2: Cell-wise RLE of the blischak data

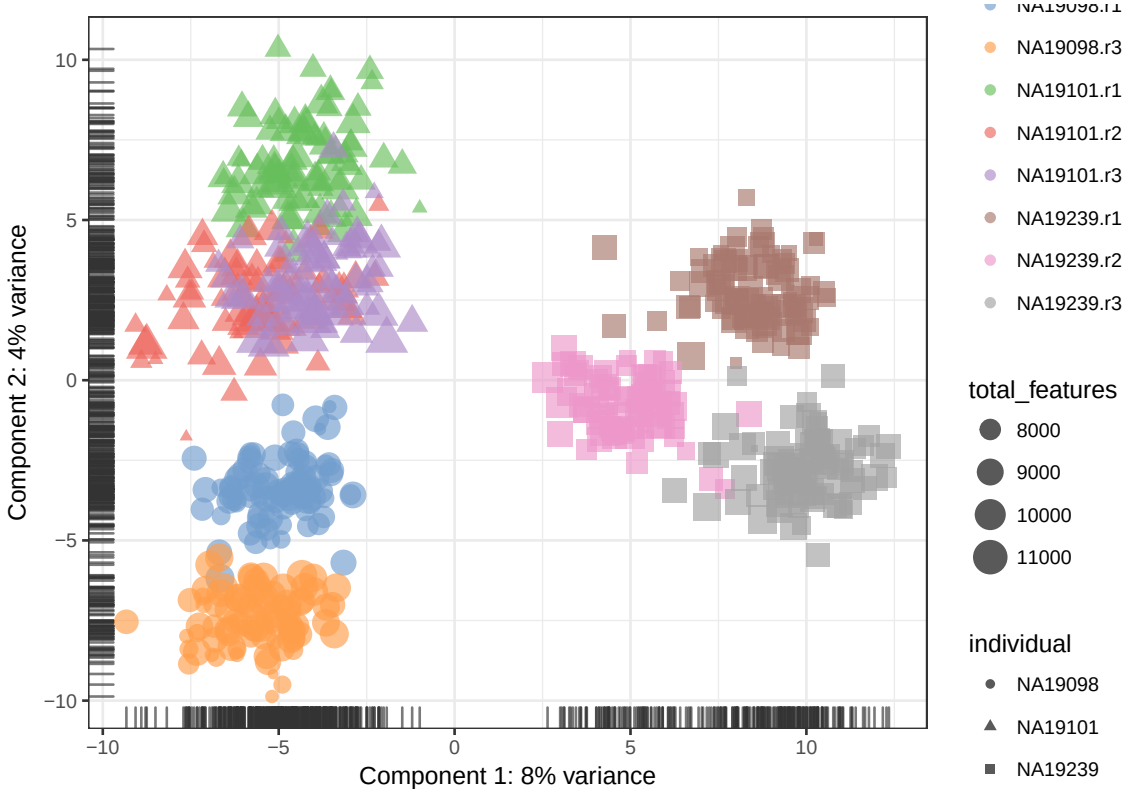


Figure 13.3: PCA plot of the blischak data after CPM normalisation

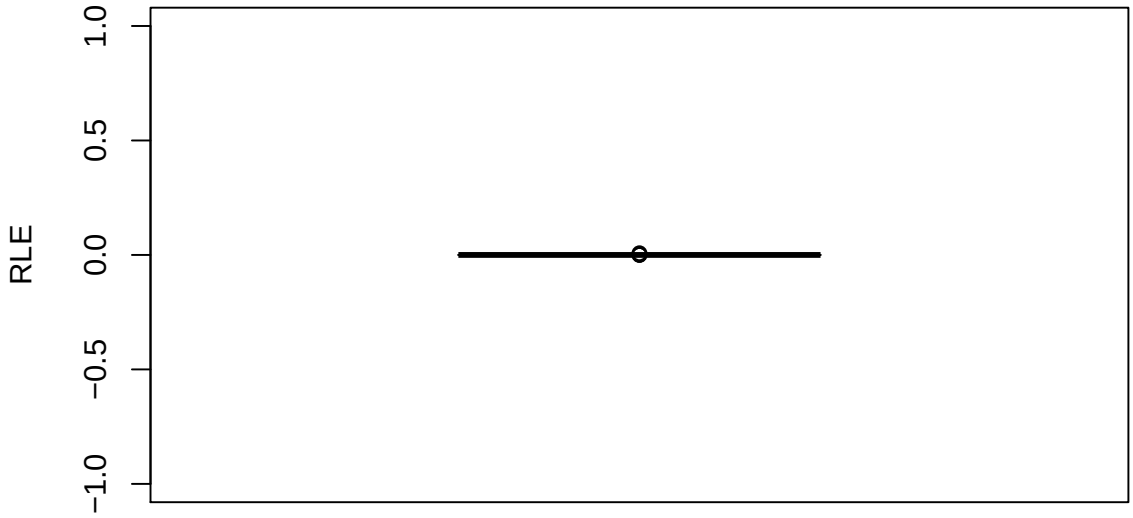


Figure 13.4: Cell-wise RLE of the blischak data

```
boxplot(calc_cell_RLE(exprs(umi.qc[endog_genes, ])),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim = c(-1,1))
```

13.3.3 TMM

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "TMM",
                           feature_set = endog_genes)
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

```
boxplot(calc_cell_RLE(norm_counts(umi.qc[endog_genes, ])),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim=c(-1,1))
```

13.3.4 scran

```
qclust <- scran::quickCluster(umi.qc, min.size = 30)
umi.qc <- scran::computeSumFactors(umi.qc, sizes = 15, clusters = qclust)
umi.qc <- scater::normalize(umi.qc)
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
```

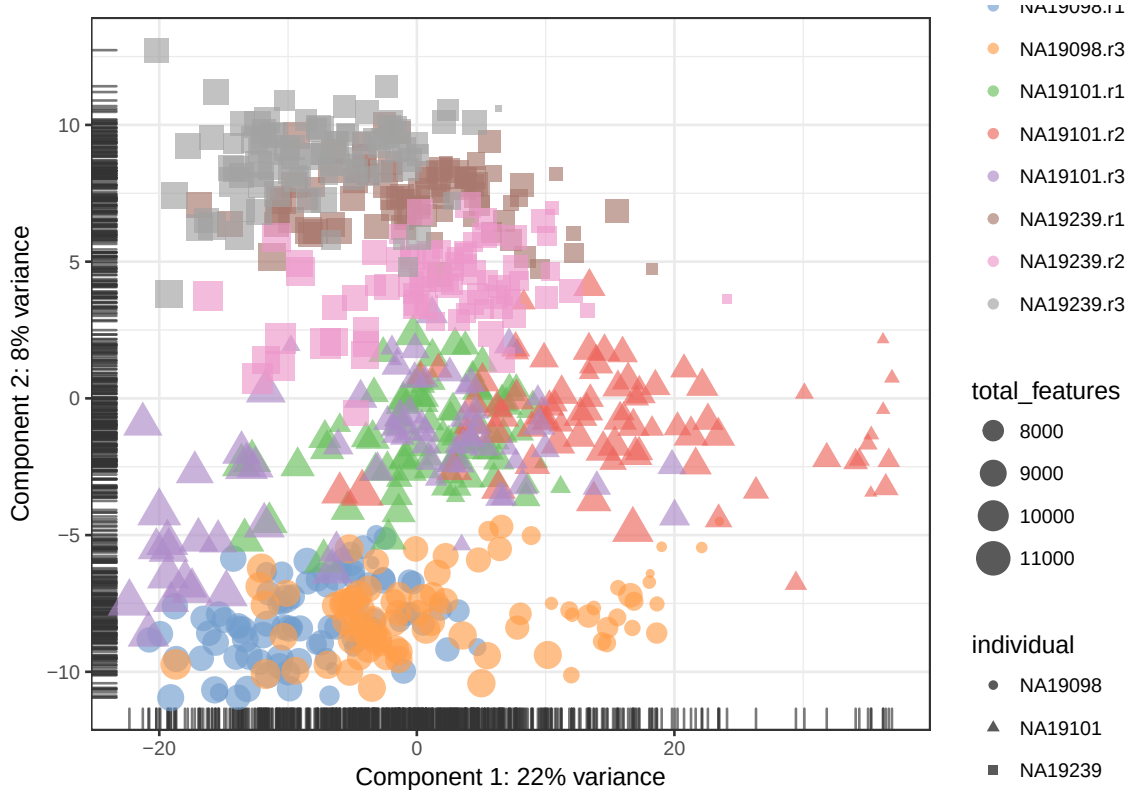


Figure 13.5: PCA plot of the blischak data after TMM normalisation

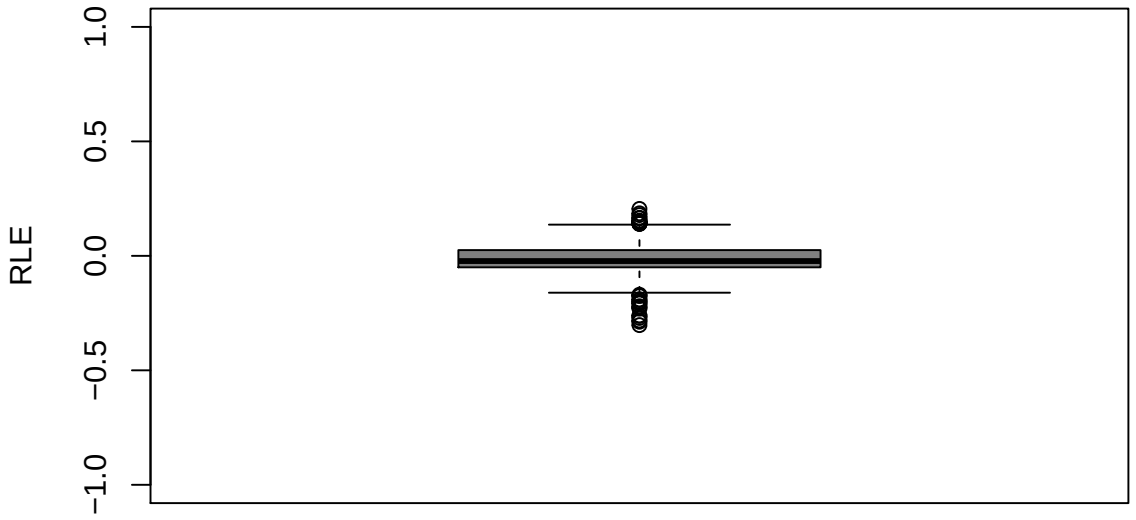


Figure 13.6: Cell-wise RLE of the blischak data

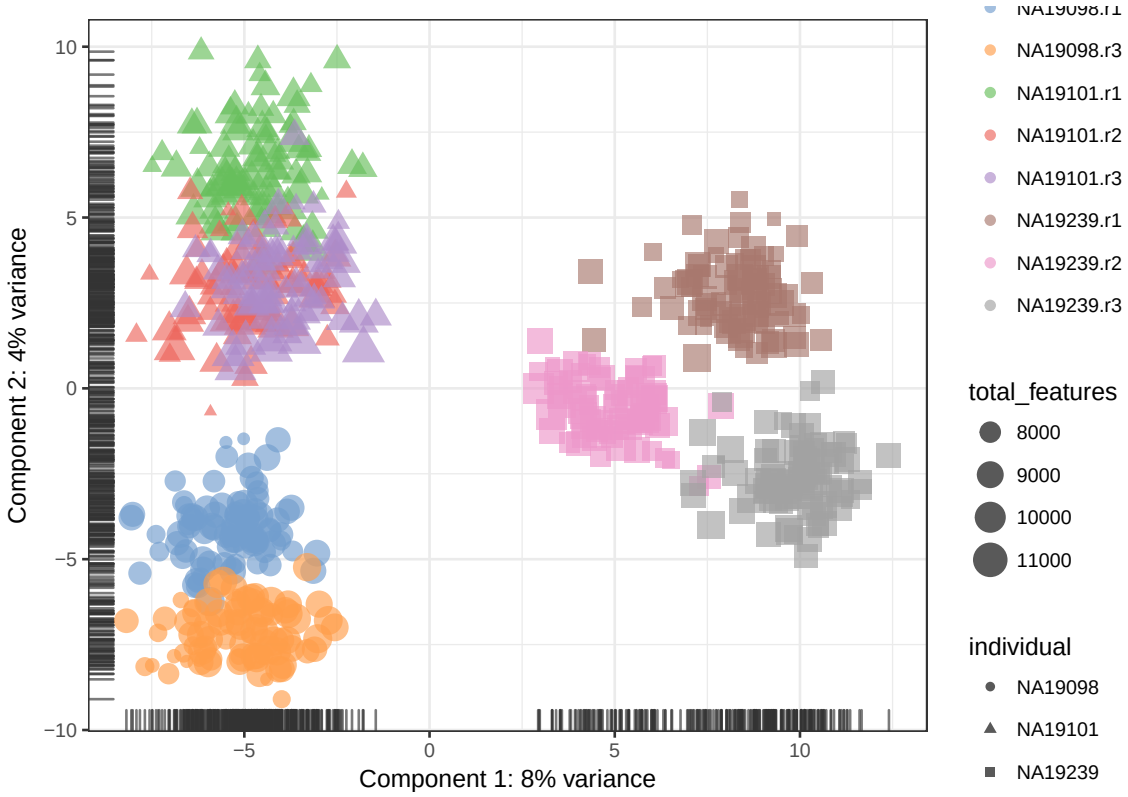


Figure 13.7: PCA plot of the blischak data after LSF normalisation

```
                shape_by = "individual",
                exprs_values = "exprs")
```

```
boxplot(calc_cell_RLE(exprs(umi.qc[endog_genes, ])),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim=c(-1,1))
```

13.3.5 Size-factor (RLE)

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "RLE",
                           feature_set = endog_genes)
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

```
boxplot(calc_cell_RLE(norm_counts(umi.qc[endog_genes, ])),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim=c(-1,1))
```

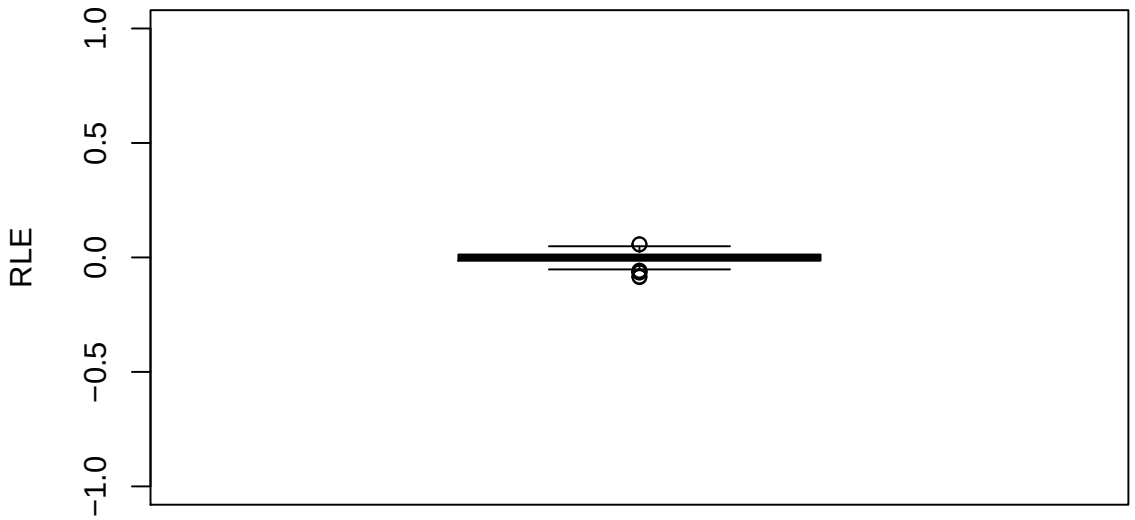


Figure 13.8: Cell-wise RLE of the blischak data

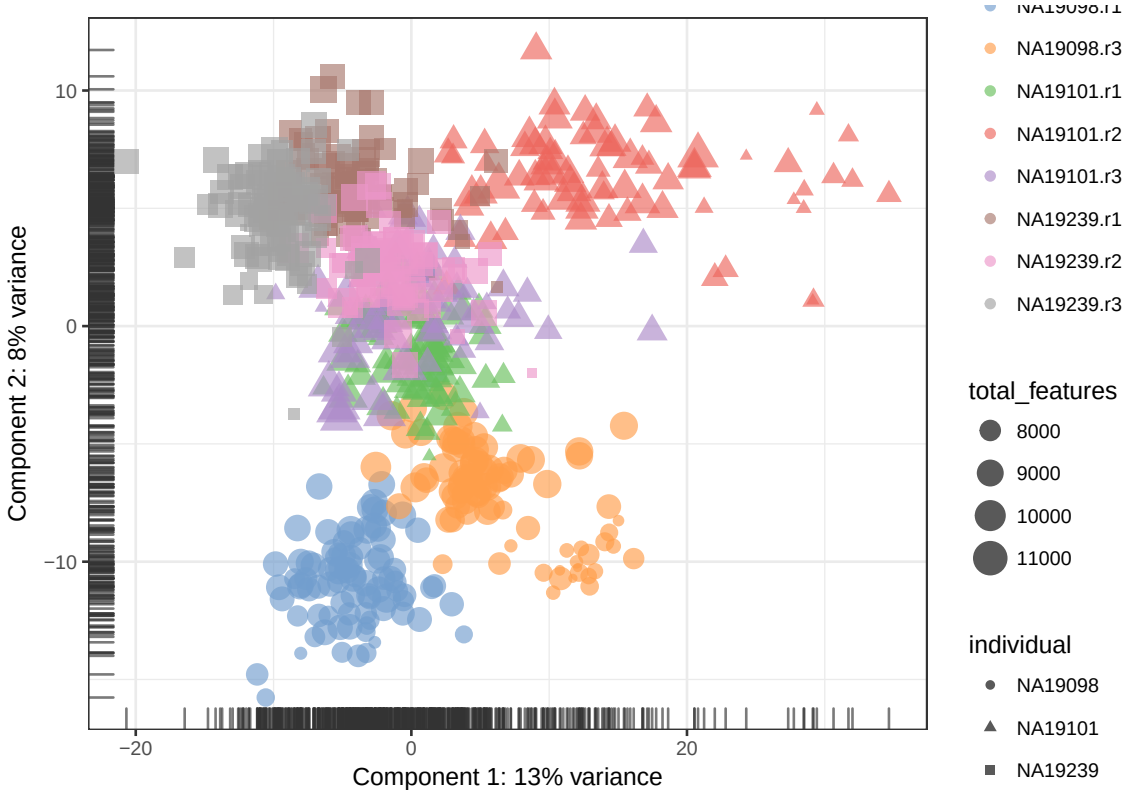


Figure 13.9: PCA plot of the blischak data after RLE normalisation

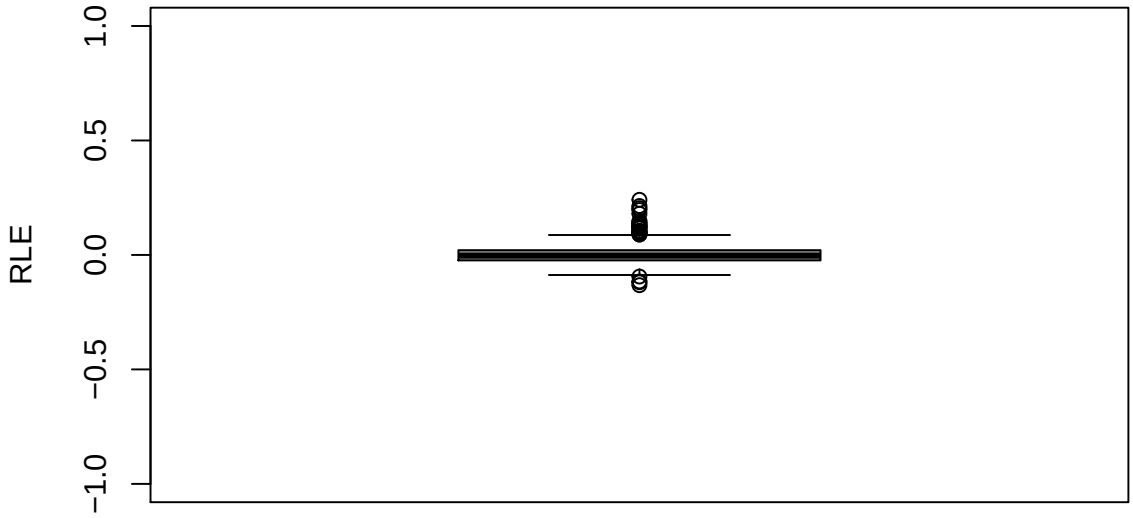


Figure 13.10: Cell-wise RLE of the blischak data

13.3.6 Upperquantile

```
umi.qc <-
    scater::normaliseExprs(umi.qc,
                           method = "upperquartile",
                           feature_set = endog_genes,
                           p = 0.99)
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

```
boxplot(calc_cell_RLE(norm_counts(umi.qc[endog_genes, ])),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim = c(-1, 1))
```

13.4 Downsampling

A final way to correct for library size is to downsample the expression matrix so that each cell has approximately the same total number of molecules. The benefit of this method is that zero values will be introduced by the down sampling thus eliminating any biases due to differing numbers of detected genes. However, the major drawback is that the process is not deterministic so each time the downsampling is run the resulting expression matrix is slightly different. Thus, often analyses must be run on multiple downsamplings to ensure results are robust.

```
Down_Sample_Matrix <-
function (expr_mat)
{
    min_lib_size <- min(colSums(expr_mat))
    down_sample <- function(x) {
        prob <- min_lib_size/sum(x)
```

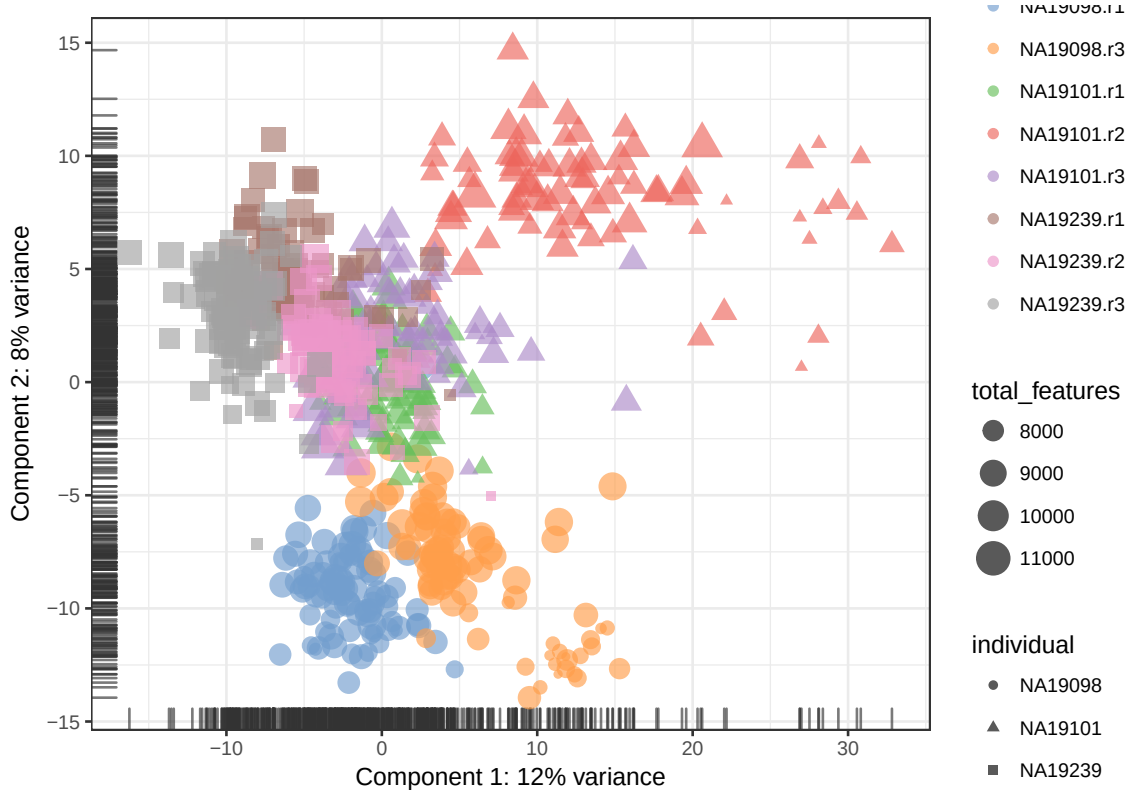


Figure 13.11: PCA plot of the blischak data after UQ normalisation

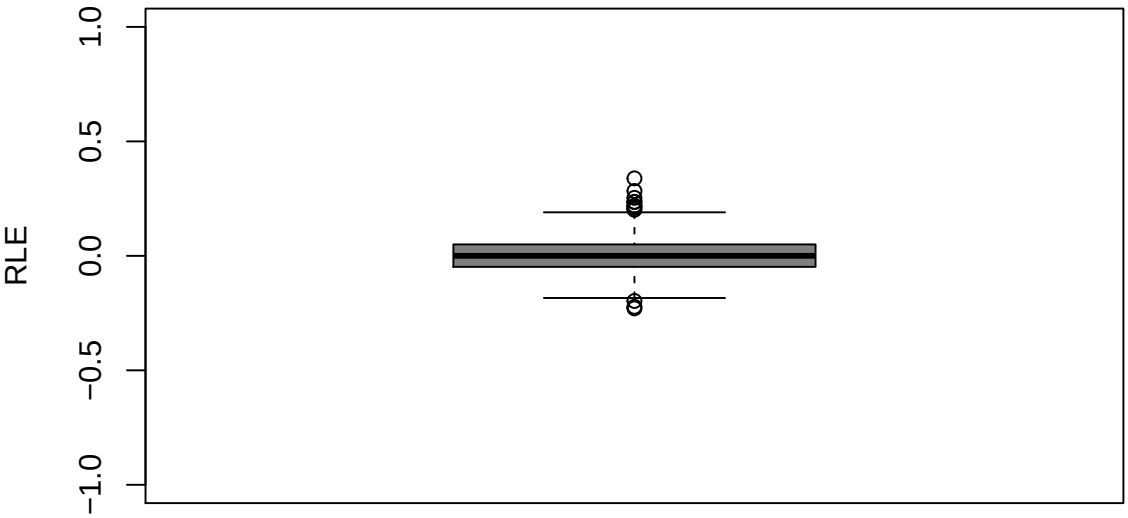


Figure 13.12: Cell-wise RLE of the blischak data

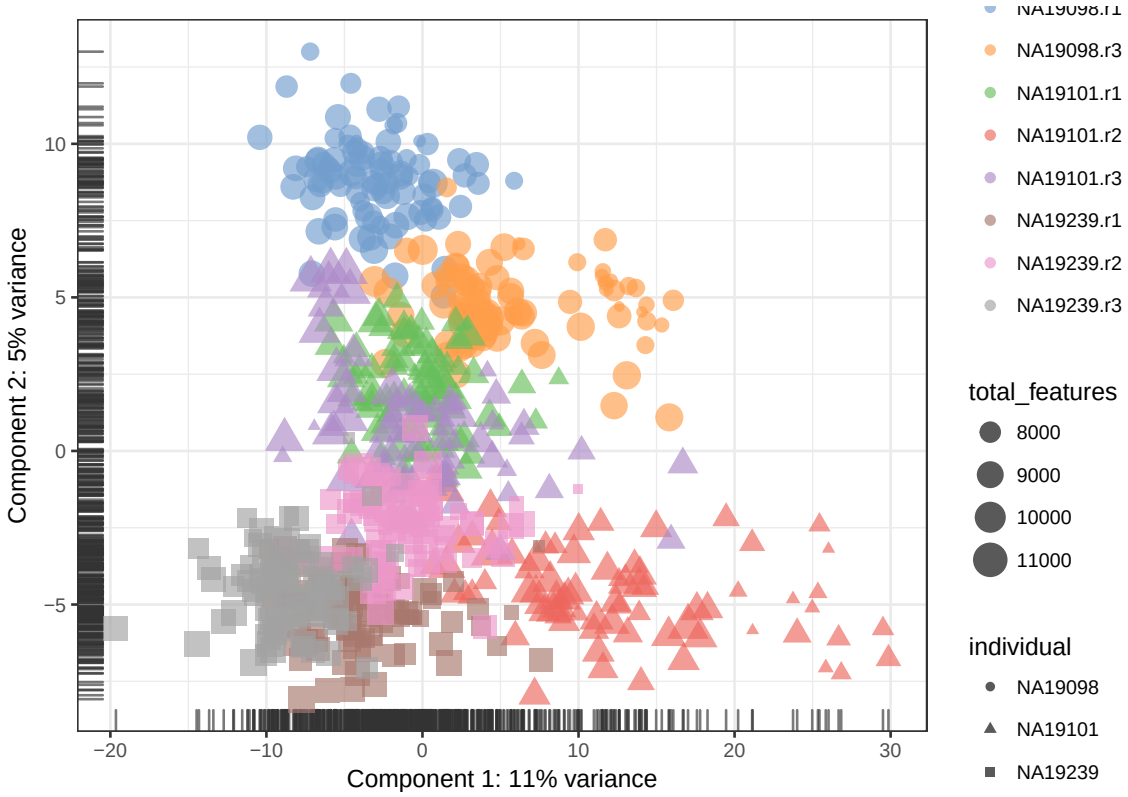


Figure 13.13: PCA plot of the blischak data after downsampling

```
        return(unlist(lapply(x, function(y) {
            rbinom(1, y, prob)
        })))
    }
    down_sampled_mat <- apply(expr_mat, 2, down_sample)
    return(down_sampled_mat)
}
```

```
norm_counts(umi.qc) <-
    scRNA.seq.funcs::Down_Sample_Matrix(counts(umi.qc))
scater::plotPCA(umi.qc[endog_genes, ],
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "norm_counts")
```

```
tmp <- norm_counts(umi.qc[endog_genes, ])
# ignore genes which are not detected in any cells following downsampling
boxplot(calc_cell_RLE(tmp[rowMeans(tmp) > 0, ]),
        col = "grey50",
        ylab = "RLE",
        main = "", ylim = c(-1, 1))
```

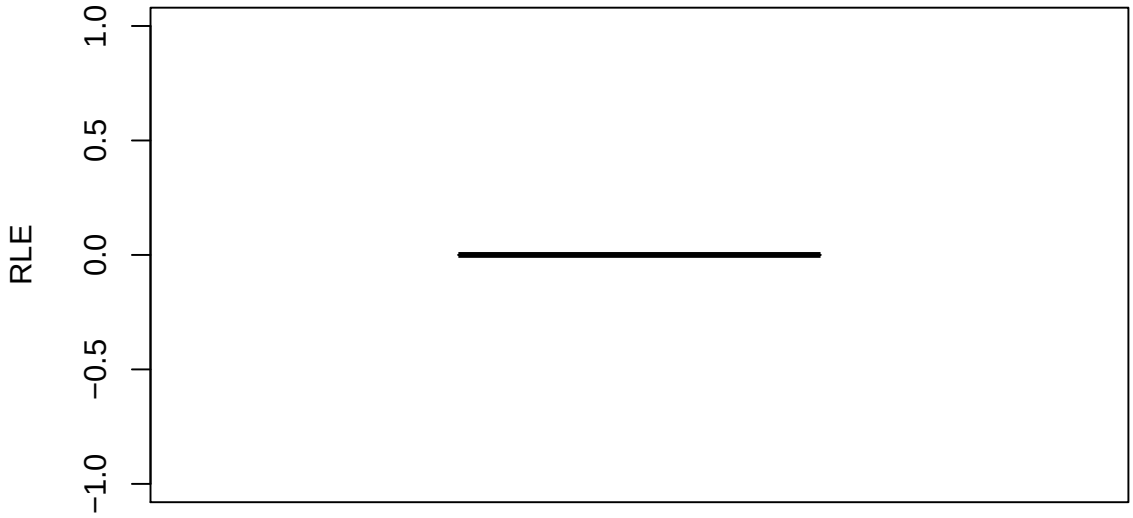


Figure 13.14: Cell-wise RLE of the blischak data

13.5 Normalizing for gene/transcript length

Some methods combine library size and fragment/gene length normalization such as:

- **RPKM** - Reads Per Kilobase Million (for single-end sequencing)
- **FPKM** - Fragments Per Kilobase Million (same as **RPKM** but for paired-end sequencing, makes sure that paired ends mapped to the same fragment are not counted twice)
- **TPM** - Transcripts Per Kilobase Million (same as **RPKM**, but the order of normalizations is reversed - length first and sequencing depth second)

These methods are not applicable to our dataset since the end of the transcript which contains the UMI was preferentially sequenced. Furthermore in general these should only be calculated using appropriate quantification software from aligned BAM files not from read counts since often only a portion of the entire gene/transcript is sequenced, not the entire length. If in doubt check for a relationship between gene/transcript length and expression level.

However, here we show how these normalisations can be calculated using scater. First, we need to find the effective transcript length in Kilobases. However, our dataset containes only gene IDs, therefore we will be using the gene lengths instead of transcripts. scater uses the biomaRt package, which allows one to annotate genes by other attributes:

```
umi.qc <-
    scater::getBMFeatureAnnos(umi.qc,
                              filters = "ensembl_gene_id",
                              attributes = c("ensembl_gene_id",
                                             "hgnc_symbol",
                                             "chromosome_name",
                                             "start_position",
                                             "end_position"),
                              feature_symbol = "hgnc_symbol",
                              feature_id = "ensembl_gene_id",
                              biomart = "ENSEMBL_MART_ENSEMBL",
                              dataset = "hsapiens_gene_ensembl",
                              host = "www.ensembl.org")

# If you have mouse data, change the arguments based on this example:
```

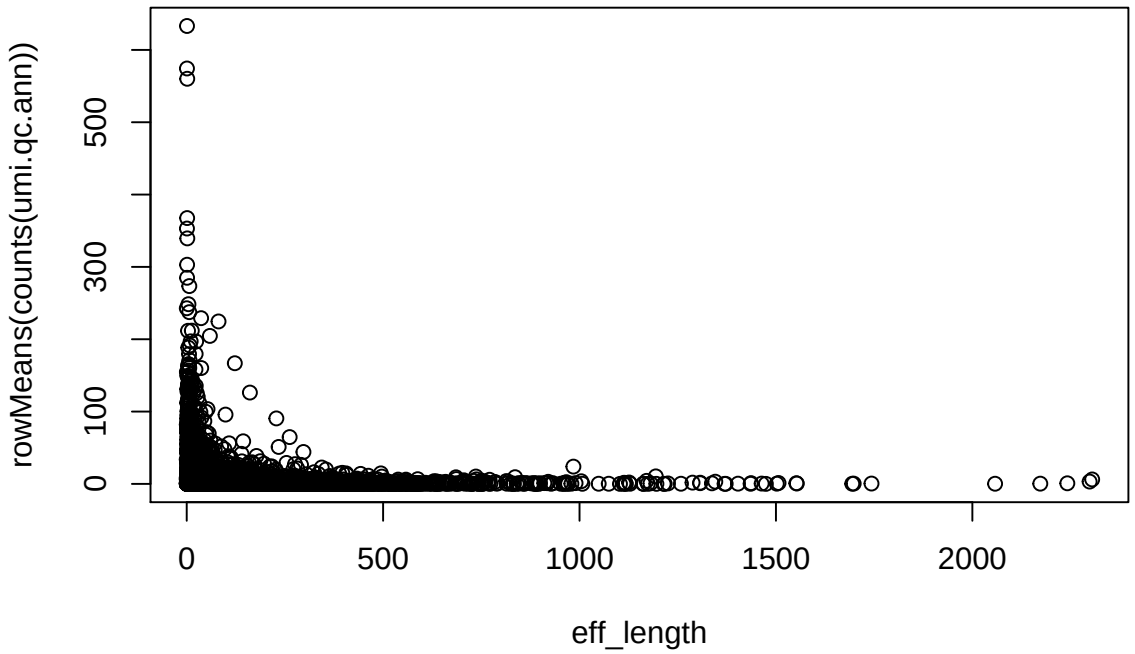


Figure 13.15: Gene length vs Mean Expression for the raw data

```
# scater::getBMFeatureAnnos(object,
#                           filters = "ensembl_transcript_id",
#                           attributes = c("ensembl_transcript_id",
#                                          "ensembl_gene_id", "mgi_symbol",
#                                          "chromosome_name",
#                                          "transcript_biotype",
#                                          "transcript_start",
#                                          "transcript_end",
#                                          "transcript_count"),
#                           feature_symbol = "mgi_symbol",
#                           feature_id = "ensembl_gene_id",
#                           biomart = "ENSEMBL_MART_ENSEMBL",
#                           dataset = "mmusculus_gene_ensembl",
#                           host = "www.ensembl.org")
```

Some of the genes were not annotated, therefore we filter them out:

```
umi.qc.ann <-
    umi.qc[!is.na(fData(umi.qc)$ensembl_gene_id), ]
```

Now we compute the total gene length in Kilobases by using the `end_position` and `start_position` fields:

```
eff_length <- abs(fData(umi.qc.ann)$end_position -
                      fData(umi.qc.ann)$start_position)/1000
```

```
plot(eff_length, rowMeans(counts(umi.qc.ann)))
```

There is no relationship between gene length and mean expression so FPKMs & TPMs are inappropriate for this dataset. But we will demonstrate them anyway.

Now we are ready to perform the normalisations:

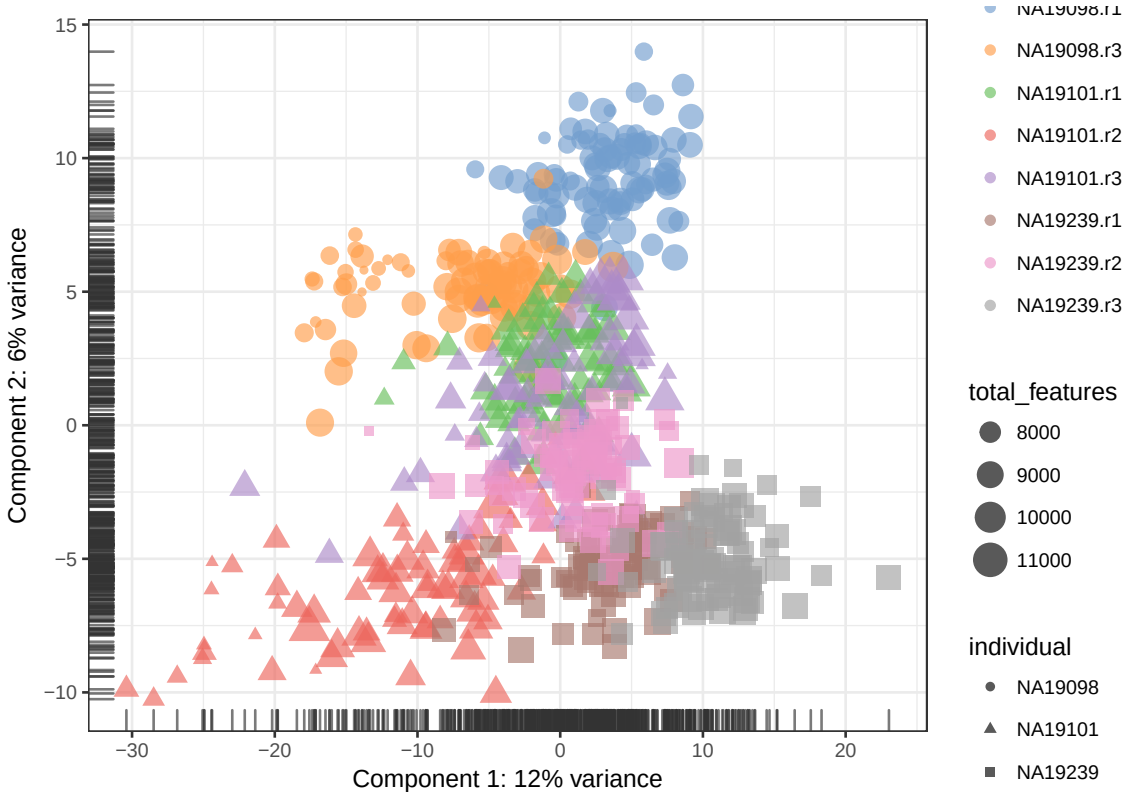


Figure 13.16: PCA plot of the blischak data after FPKM normalisation

```
tpm(umi.qc.ann) <-
    calculateTPM(
        umi.qc.ann,
        eff_length
    )
fpkm(umi.qc.ann) <-
    calculateFPKM(
        umi.qc.ann,
        eff_length
    )
```

Plot the results as a PCA plot:

```
scater::plotPCA(umi.qc.ann,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "fpkm")
```

```
scater::plotPCA(umi.qc.ann,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "tpm")
```

Note: The PCA looks for differences between cells. Gene length is the same across cells for each gene thus

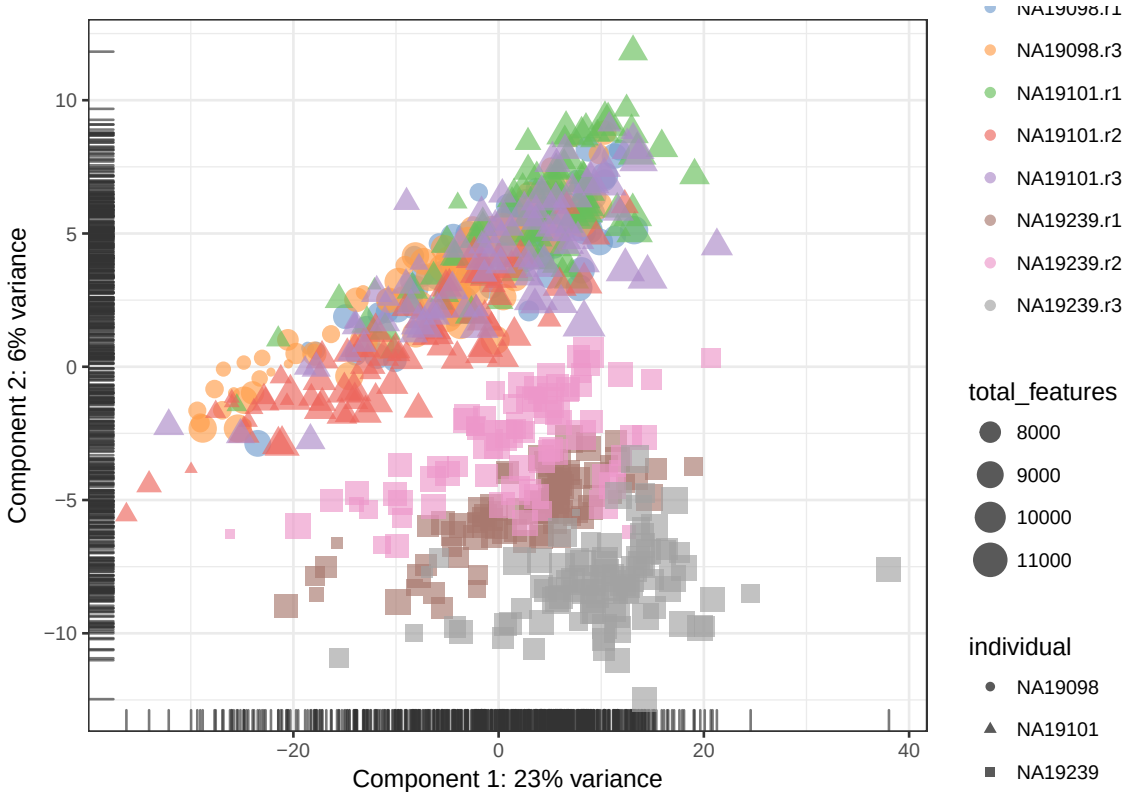


Figure 13.17: PCA plot of the blischak data after TPM normalisation

FPKM is almost identical to the CPM plot (it is just rotated) since it performs CPM first then normalizes gene length. Whereas, TPM is different because it weights genes by their length before performing CPM.

13.6 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter).

Chapter 14

Normalization for library size (Reads)

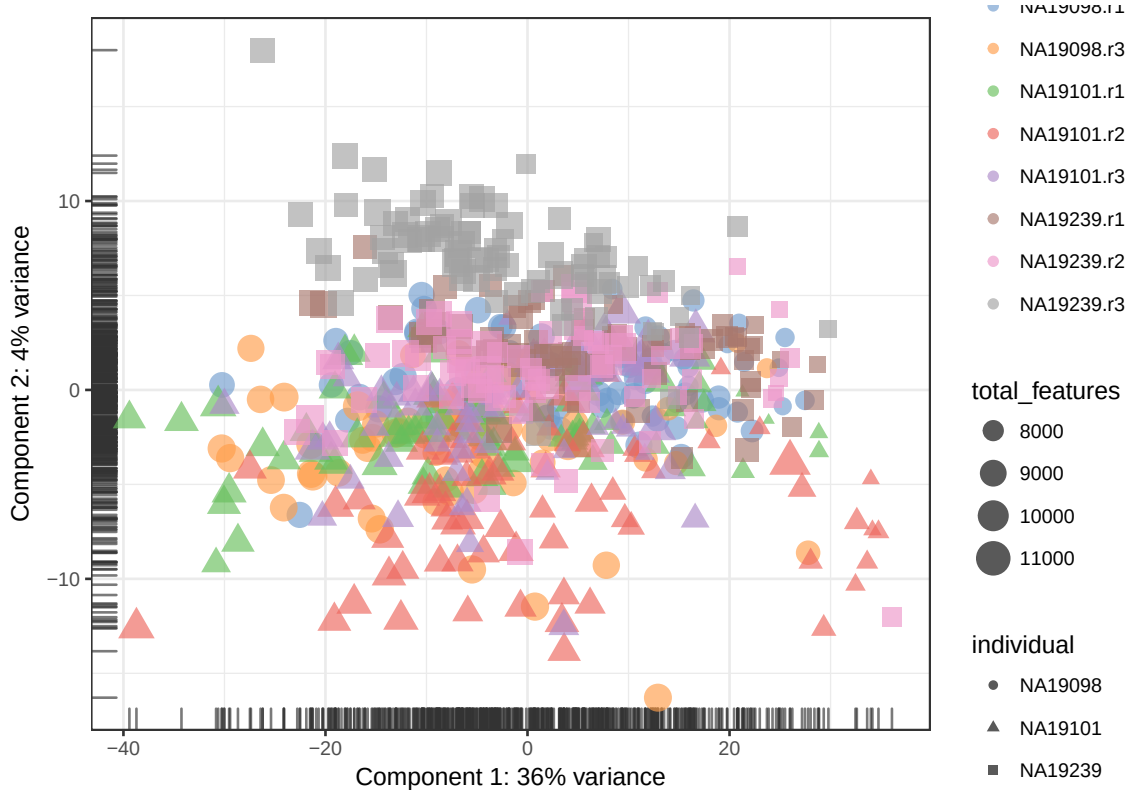


Figure 14.1: PCA plot of the blischak data

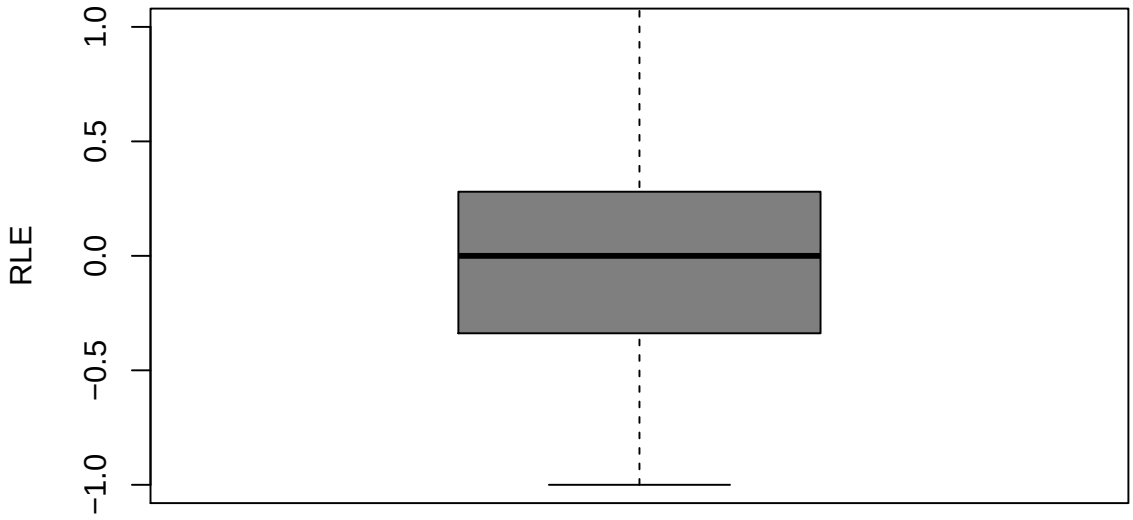


Figure 14.2: Cell-wise RLE of the blischak data

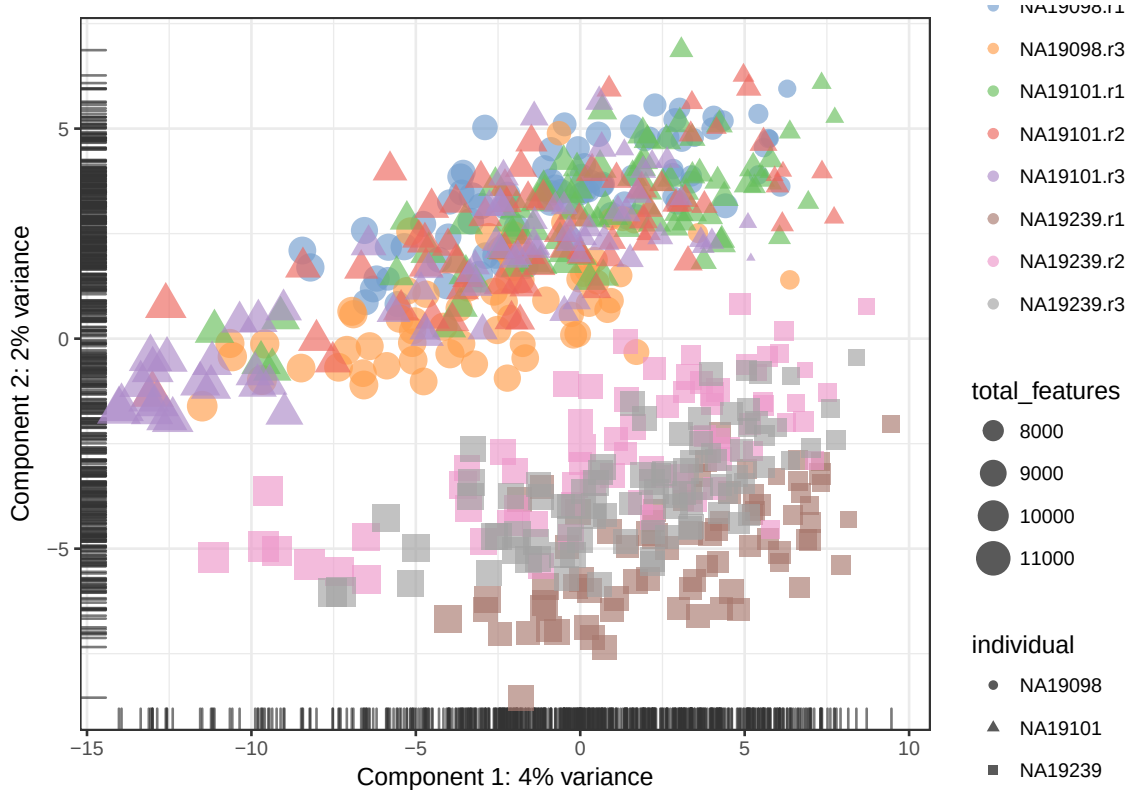


Figure 14.3: PCA plot of the blischak data after CPM normalisation

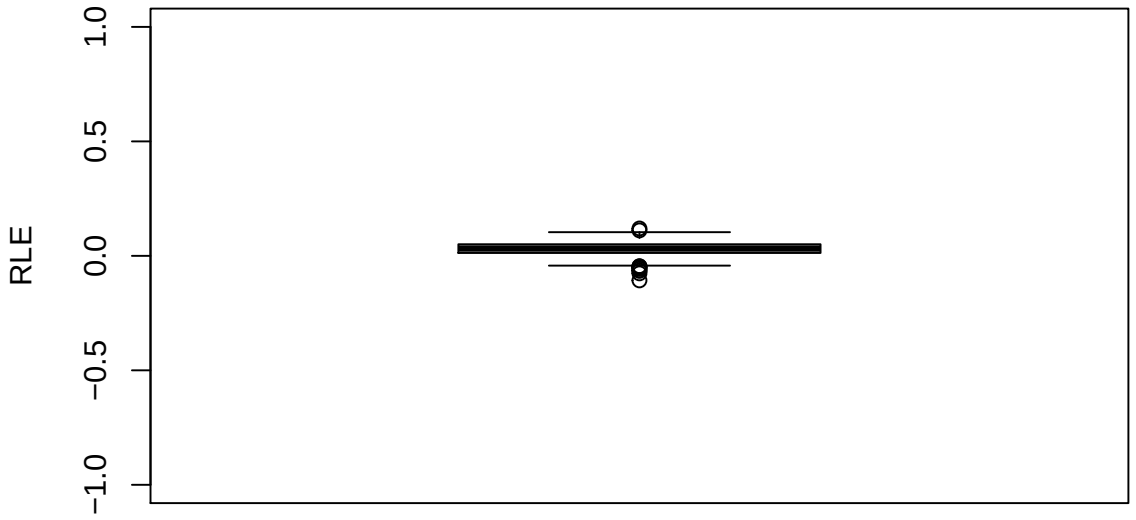


Figure 14.4: Cell-wise RLE of the blischak data

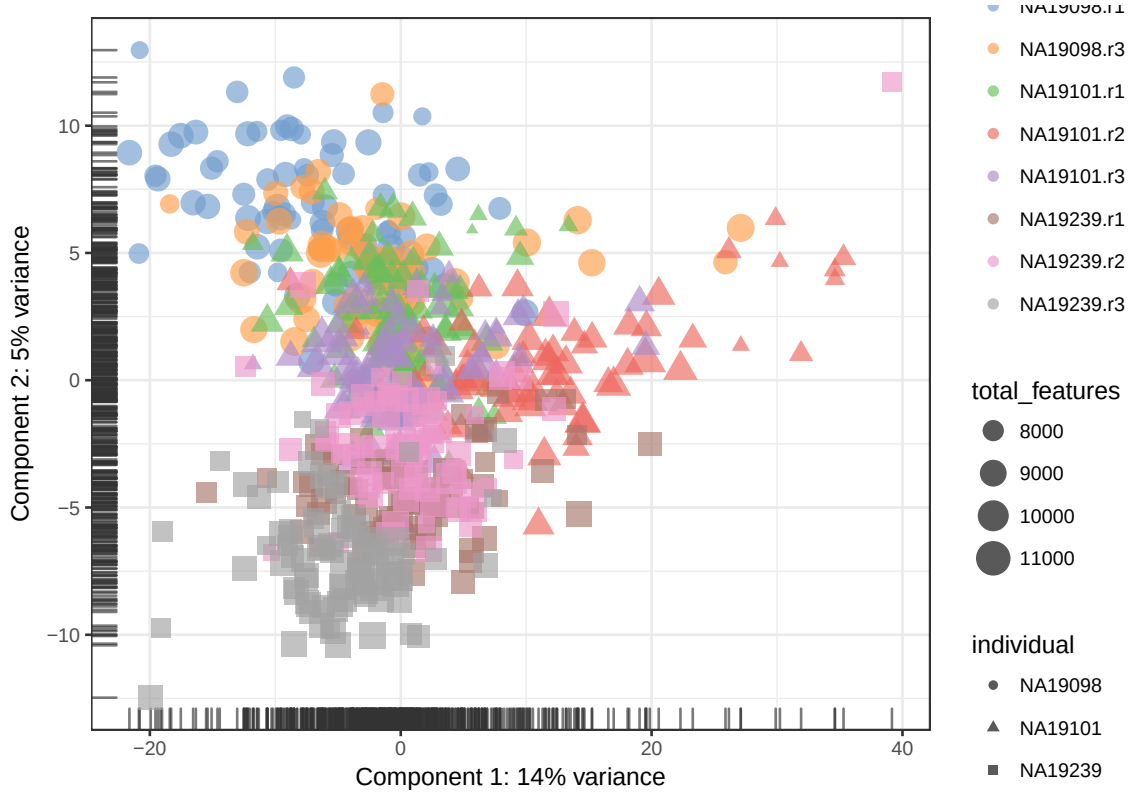


Figure 14.5: PCA plot of the blischak data after TMM normalisation

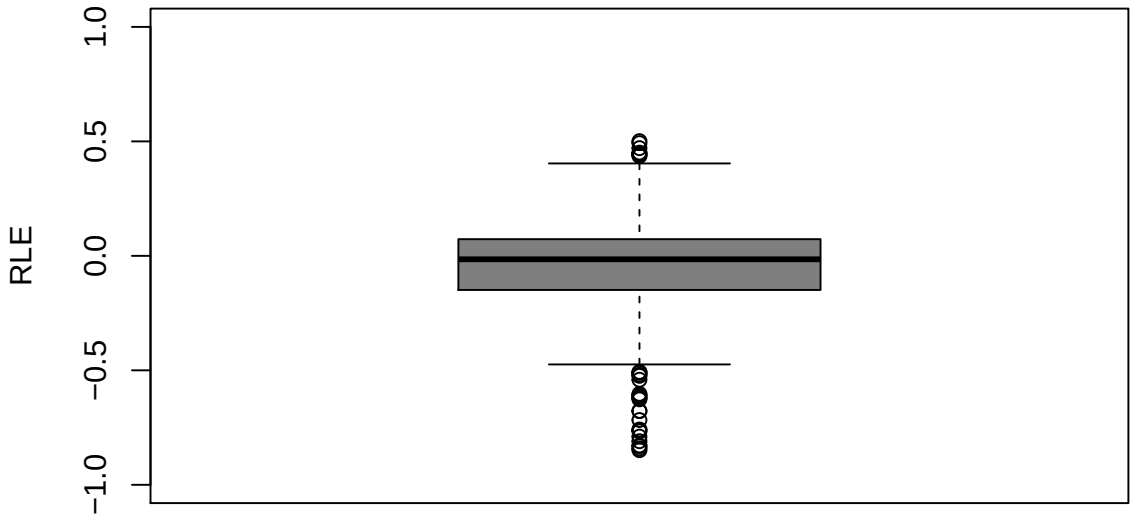


Figure 14.6: Cell-wise RLE of the blischak data

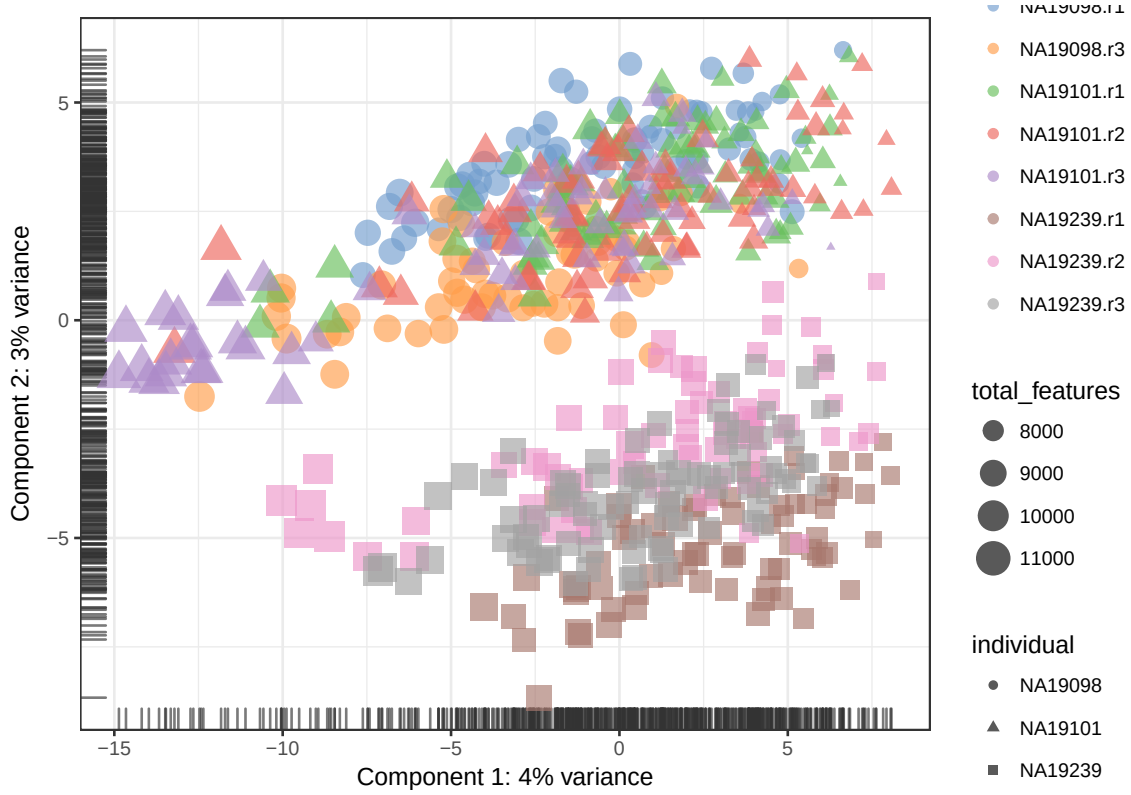


Figure 14.7: PCA plot of the blischak data after LSF normalisation

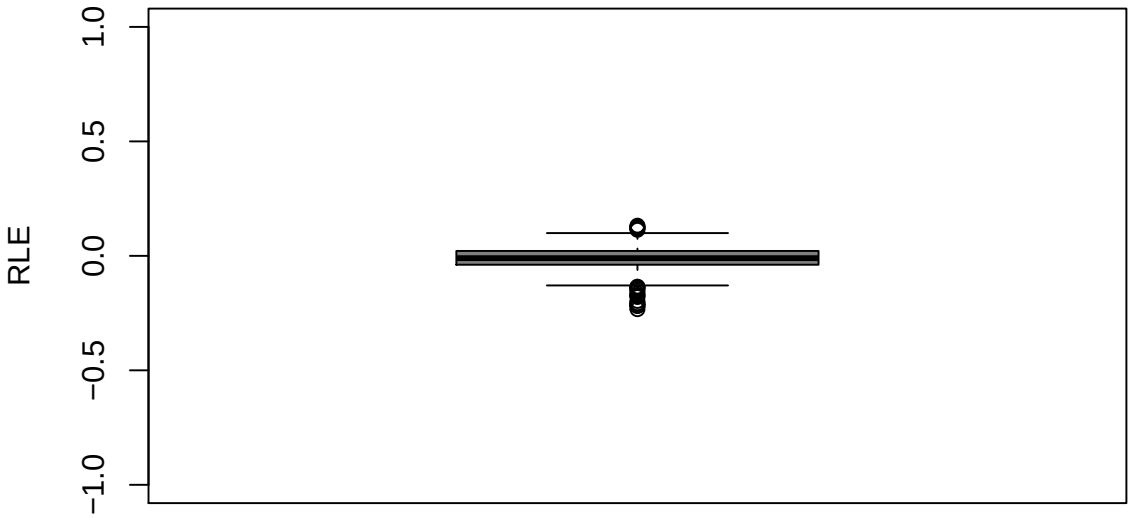


Figure 14.8: Cell-wise RLE of the blischak data

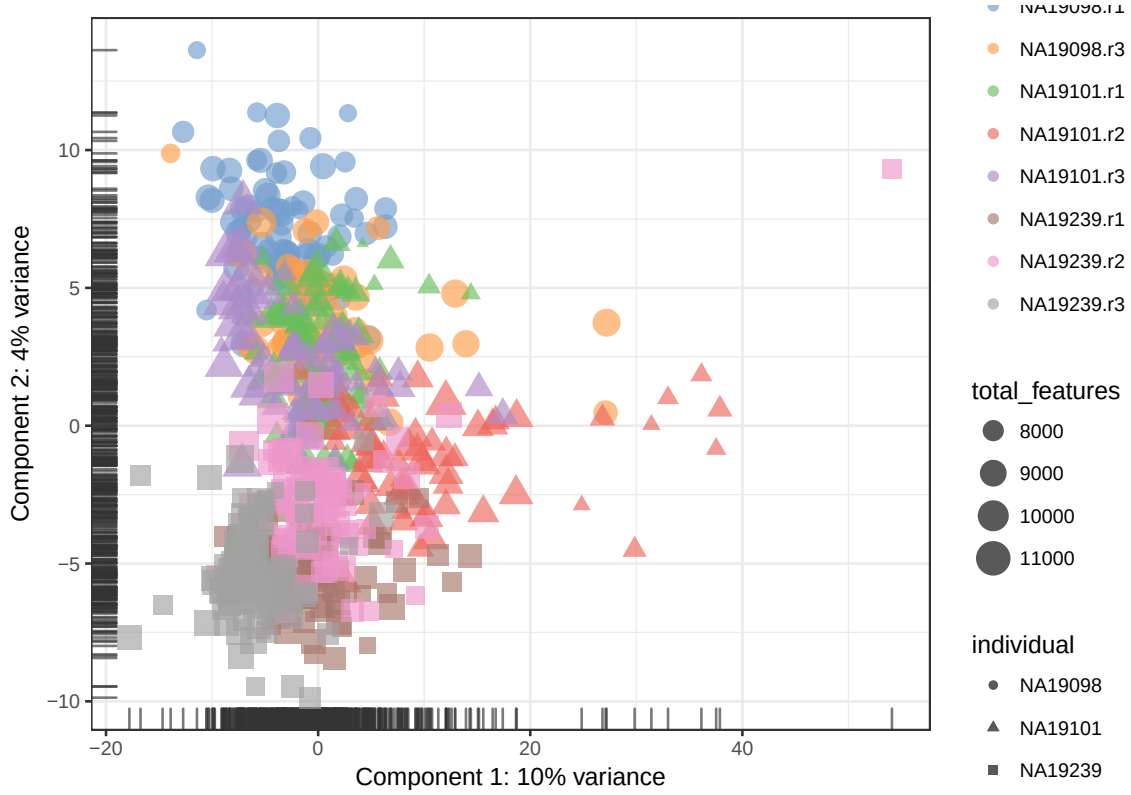


Figure 14.9: PCA plot of the blischak data after RLE normalisation

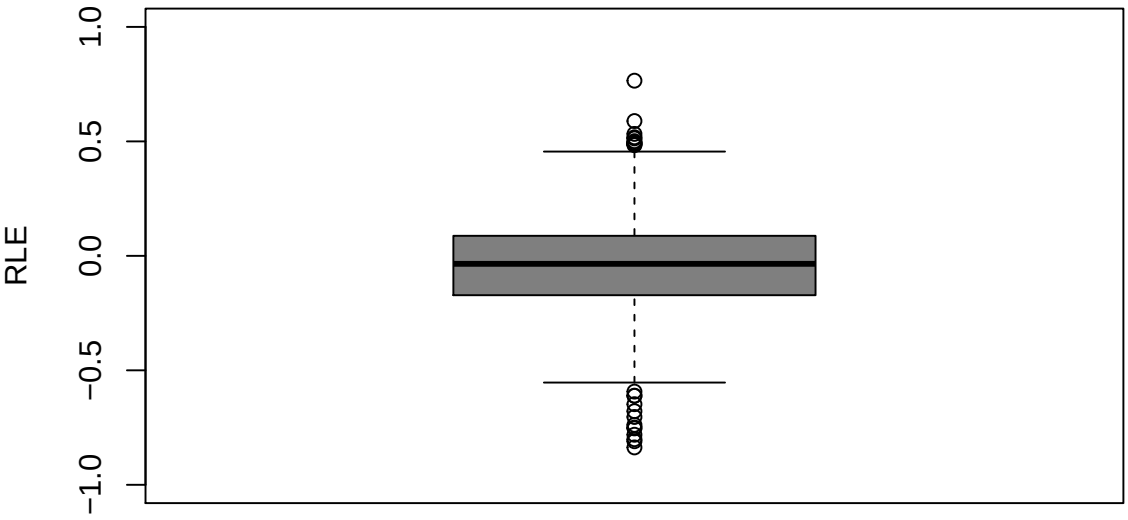


Figure 14.10: Cell-wise RLE of the blischak data

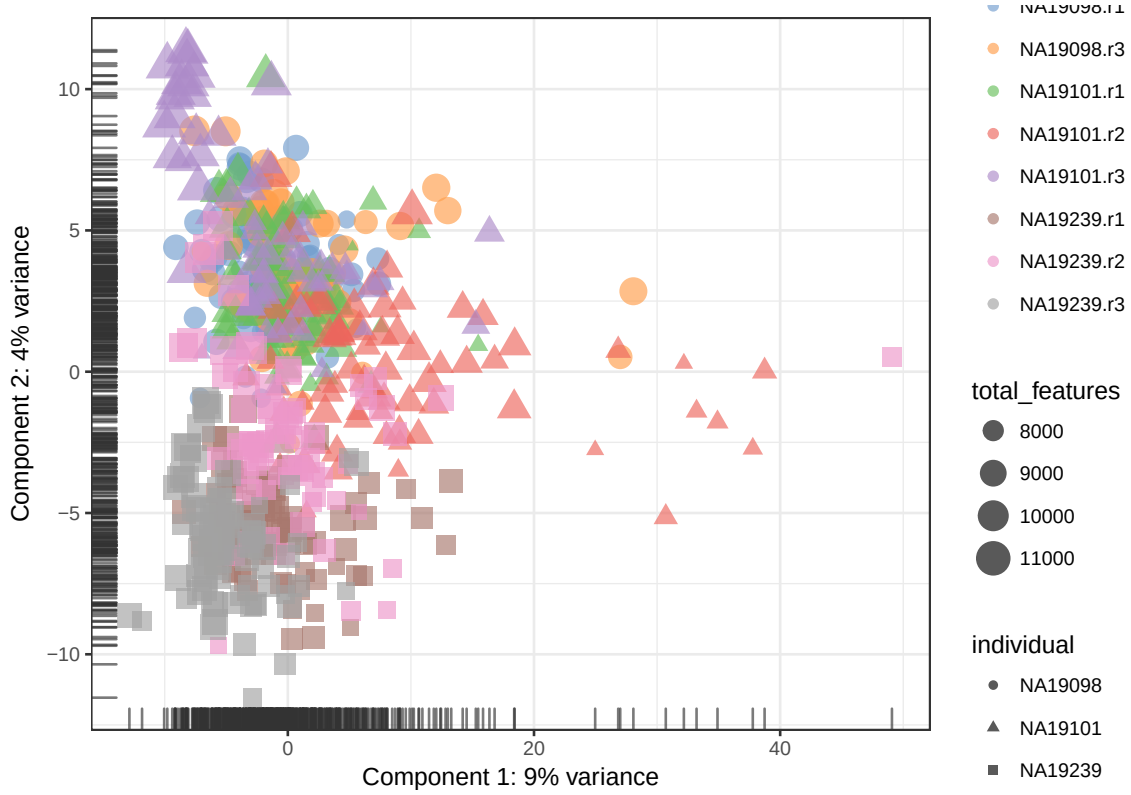


Figure 14.11: PCA plot of the blischak data after UQ normalisation

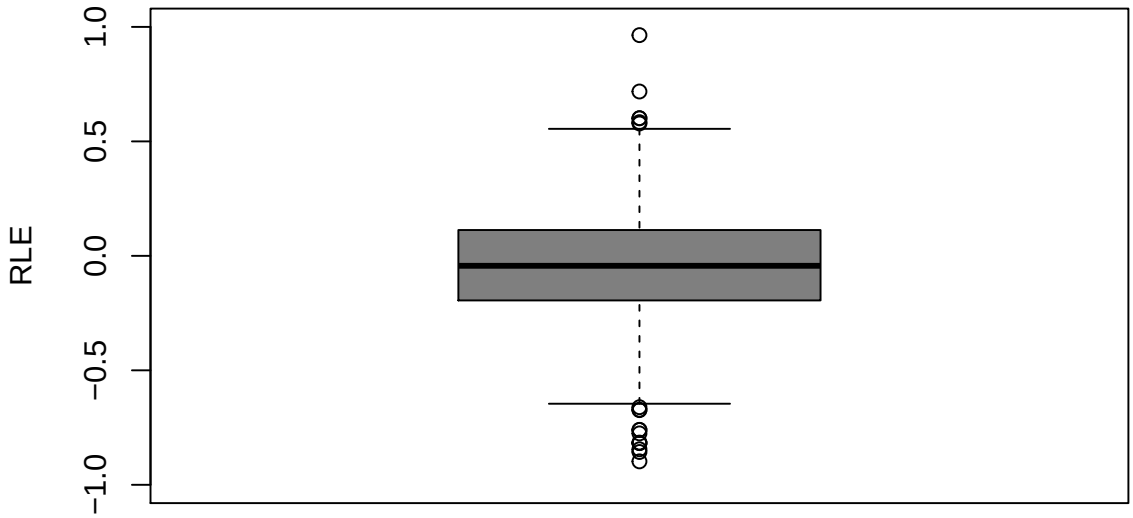


Figure 14.12: Cell-wise RLE of the blischak data

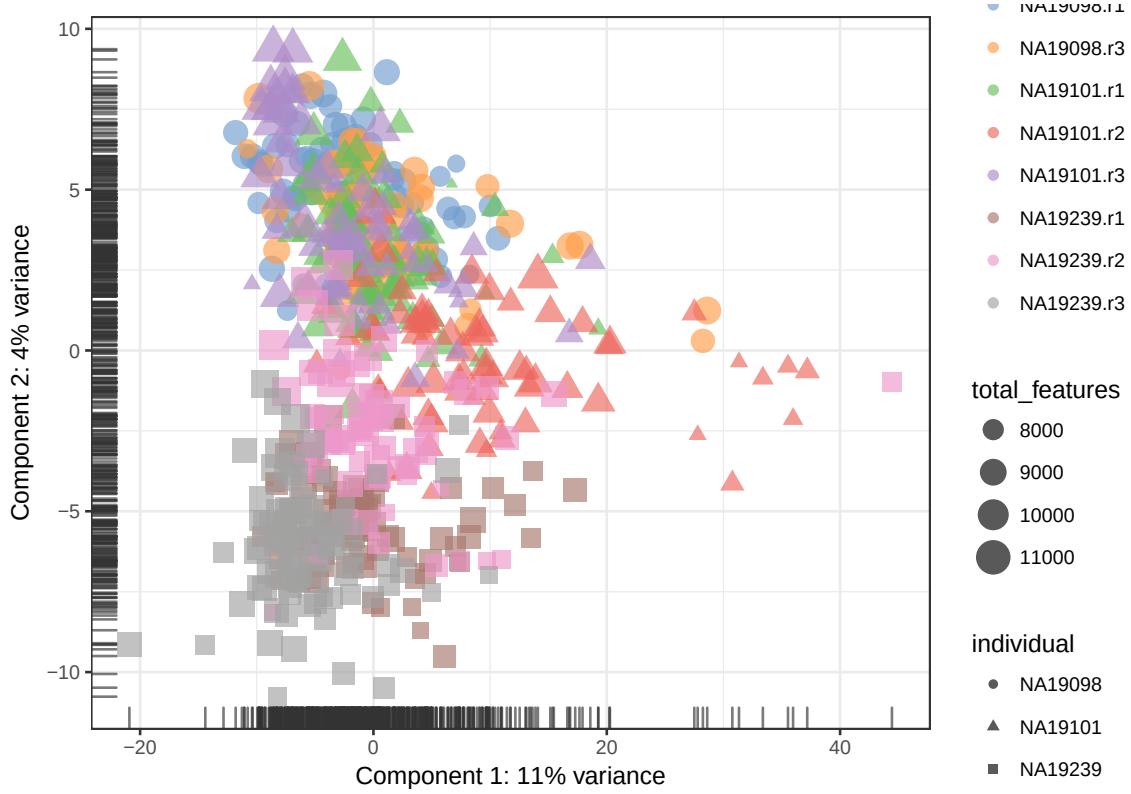


Figure 14.13: PCA plot of the blischak data after downsampling

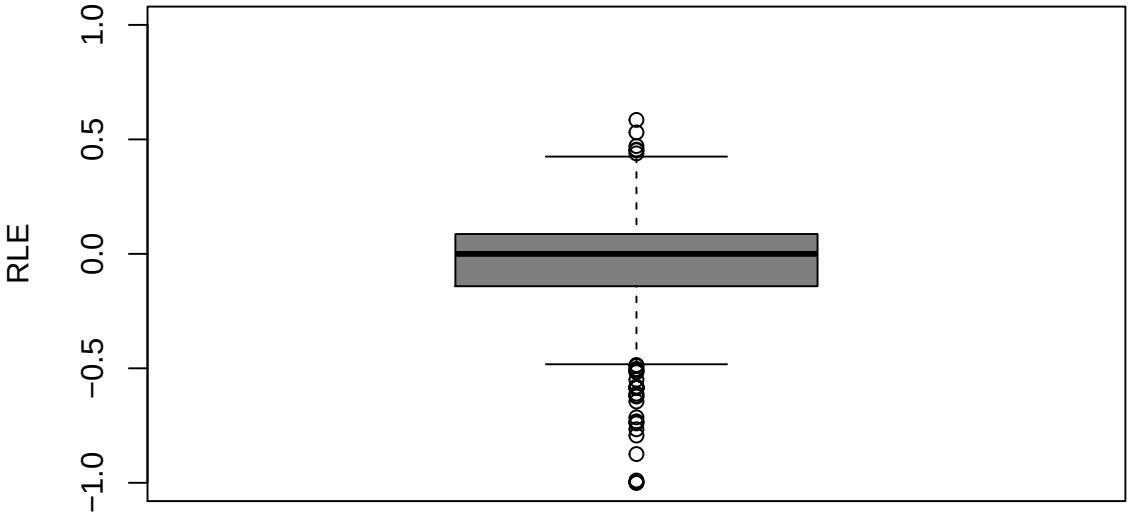


Figure 14.14: Cell-wise RLE of the blischak data

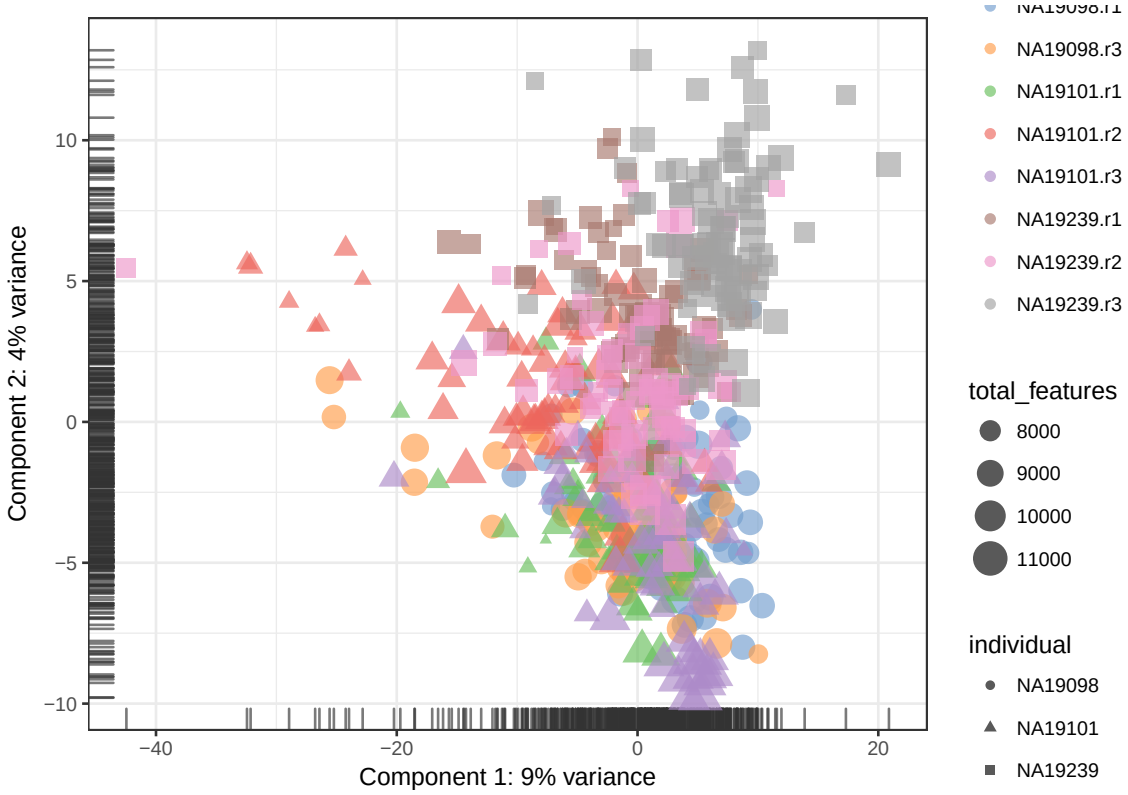


Figure 14.15: PCA plot of the blischak data after FPKM normalisation

Figure 14.16: PCA plot of the blischak data after TPM normalisation

Chapter 15

Dealing with confounders

15.1 Introduction

In the previous chapter we normalized for library size, effectively removing it as a confounder. Now we will consider removing other less well defined confounders from our data. Technical confounders (aka batch effects) can arise from difference in reagents, isolation methods, the lab/experimenter who performed the experiment, even which day/time the experiment was performed. Accounting for technical confounders, and batch effects particularly, is a large topic that also involves principles of experimental design. Here we address approaches that can be taken to account for confounders when the experimental design is appropriate.

Fundamentally, accounting for technical confounders involves identifying and, ideally, removing sources of variation in the expression data that are not related to (i.e. are confounding) the biological signal of interest. Various approaches exist, some of which use spike-in or housekeeping genes, and some of which use endogenous genes.

The use of spike-ins as control genes is appealing, since the same amount of ERCC (or other) spike-in was added to each cell in our experiment. In principel, all the variablity we observe for these genes is due to technical noise; whereas endogenous genes are affected by both technical noise and biological variability. Technical noise can be removed by fitting a model to the spike-ins and "substracting" this from the endogenous genes. There are several methods available based on this premise (eg. BASiCS, scLVM, RUVg); each using different noise models and different fitting procedures. Alternatively, one can identify genes which exhibit significant variation beyond technical noise (eg. Distance to median, Highly variable genes). However, there are issues with the use of spike-ins for normalisation (particularly ERCCs, derived from bacterial sequences), including that their variability can, for various reasons, actually be *higher* than that of endogenous genes.

Given the issues with using spike-ins, better results can often be obtained by using endogenous genes instead. Where we have a large number of endogenous genes that, on average, do not vary systematically between cells and where we expect technical effects to affect a large number of genes (a very common and reasonable assumption), then such methods (for example, the RUVs method) can perform well.

We explore both general approaches below.

```
library(scRNA.seq.funcs)
library(RUVSeq)
library(scater, quietly = TRUE)
library(scran)
library(edgeR)
options(stringsAsFactors = FALSE)
umi <- readRDS("blischak/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
```

```
endog_genes <- !fData(umi.qc)$is_feature_control
erccs <- fData(umi.qc)$is_feature_control
```

15.2 Remove Unwanted Variation

Factors contributing to technical noise frequently appear as "batch effects" where cells processed on different days or by different technicians systematically vary from one another. Removing technical noise and correcting for batch effects can frequently be performed using the same tool or slight variants on it. We will be considering the Remove Unwanted Variation (RUVSeq). Briefly, RUVSeq works as follows. For n samples and J genes, consider the following generalized linear model (GLM), where the RNA-Seq read counts are regressed on both the known covariates of interest and unknown factors of unwanted variation:

$$\log E[Y|W, X, O] = W\alpha + X\beta + O$$

Here, Y is the $n \times J$ matrix of observed gene-level read counts, W is an $n \times k$ matrix corresponding to the factors of "unwanted variation" and O is an $n \times J$ matrix of offsets that can either be set to zero or estimated with some other normalization procedure (such as upper-quartile normalization). The simultaneous estimation of W, α, β, and k is infeasible. For a given k, instead the following three approaches to estimate the factors of unwanted variation W are used:

- *RUVg* uses negative control genes (e.g. ERCCs), assumed to have constant expression across samples;
- *RUVs* uses centered (technical) replicate/negative control samples for which the covariates of interest are constant;
- *RUVr* uses residuals, e.g., from a first-pass GLM regression of the counts on the covariates of interest.

We will concentrate on the first two approaches.

15.2.1 RUVg

```
ruvg <- RUVg(counts(umi.qc), erccs, k = 1)
set_exprs(umi.qc, "ruvg1") <- ruvg$normalizedCounts
ruvg <- RUVg(counts(umi.qc), erccs, k = 2)
set_exprs(umi.qc, "ruvg2") <- ruvg$normalizedCounts
set_exprs(umi.qc, "ruvg2_logcpm") <- log2(t(t(ruvg$normalizedCounts) /
                                        colSums(ruvg$normalizedCounts)) + 1)
```

15.2.2 RUVs

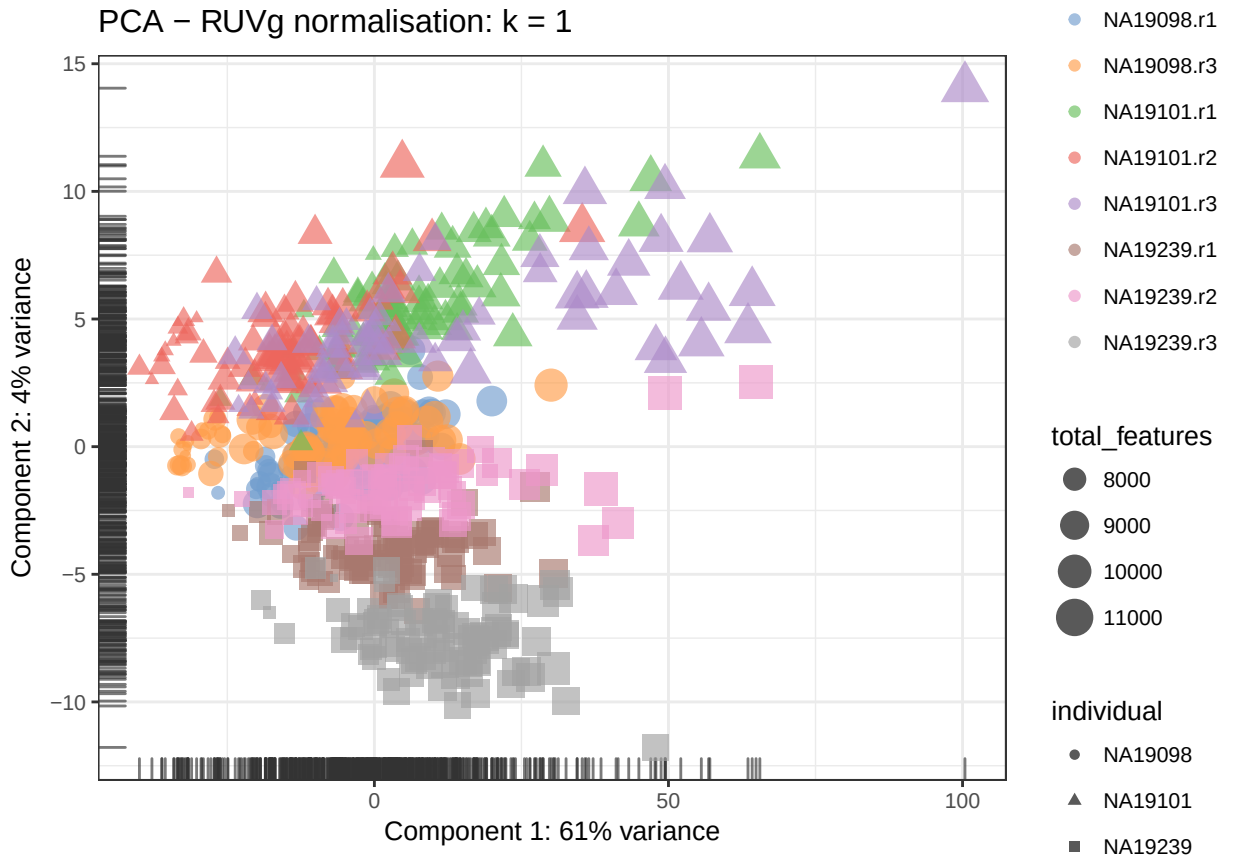
```
scIdx <- matrix(-1, ncol = max(table(umi.qc$individual)), nrow = 3)
tmp <- which(umi.qc$individual == "NA19098")
scIdx[1, 1:length(tmp)] <- tmp
tmp <- which(umi.qc$individual == "NA19101")
scIdx[2, 1:length(tmp)] <- tmp
tmp <- which(umi.qc$individual == "NA19239")
scIdx[3, 1:length(tmp)] <- tmp
cIdx <- rownames(umi.qc)
ruvs <- RUVs(counts(umi.qc), cIdx, k = 1, scIdx = scIdx, isLog = FALSE)
set_exprs(umi.qc, "ruvs1") <- ruvs$normalizedCounts
ruvs <- RUVs(counts(umi.qc), cIdx, k = 2, scIdx = scIdx, isLog = FALSE)
set_exprs(umi.qc, "ruvs2") <- ruvs$normalizedCounts
```

```
set_exprs(umi.qc, "ruvs2_logcpm") <- log2(t(t(ruvs$normalizedCounts) /
                                            colSums(ruvs$normalizedCounts)) + 1)
```

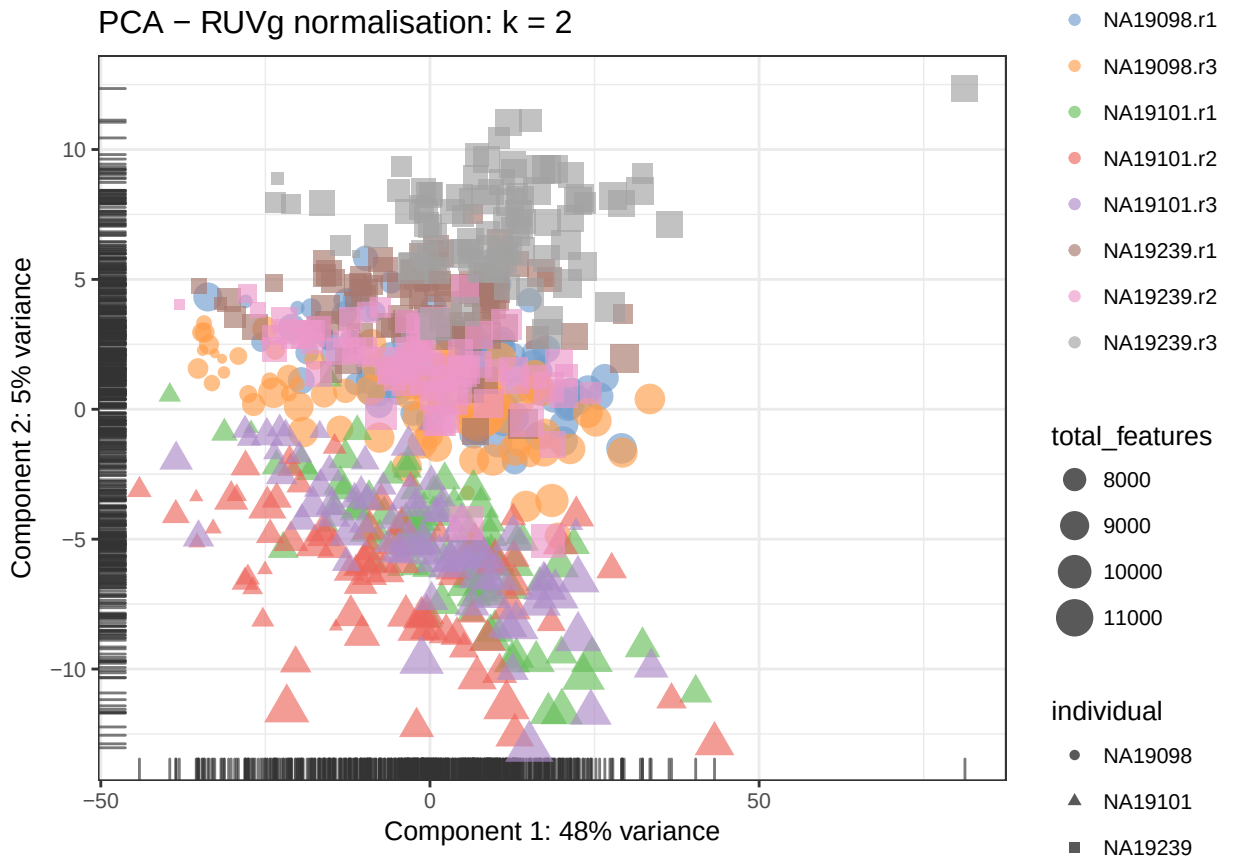
15.3 Effectiveness 1

We evaluate the effectiveness of the normalization by inspecting the PCA plot where colour corresponds the technical replicates and shape corresponds to different biological samples (individuals). Separation of biological samples and interspersed batches indicates that technical variation has been removed.

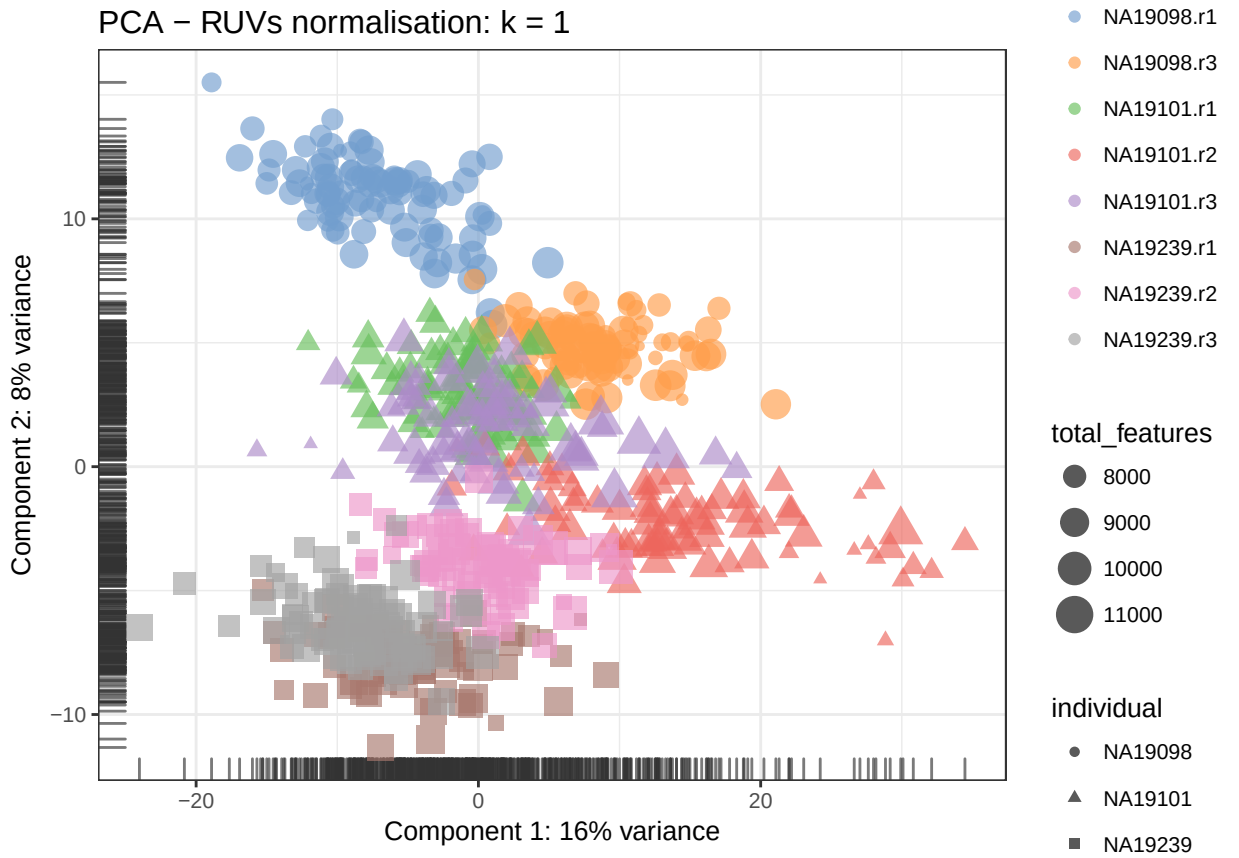
```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg1") +
    ggtitle("PCA - RUVg normalisation: k = 1")
```



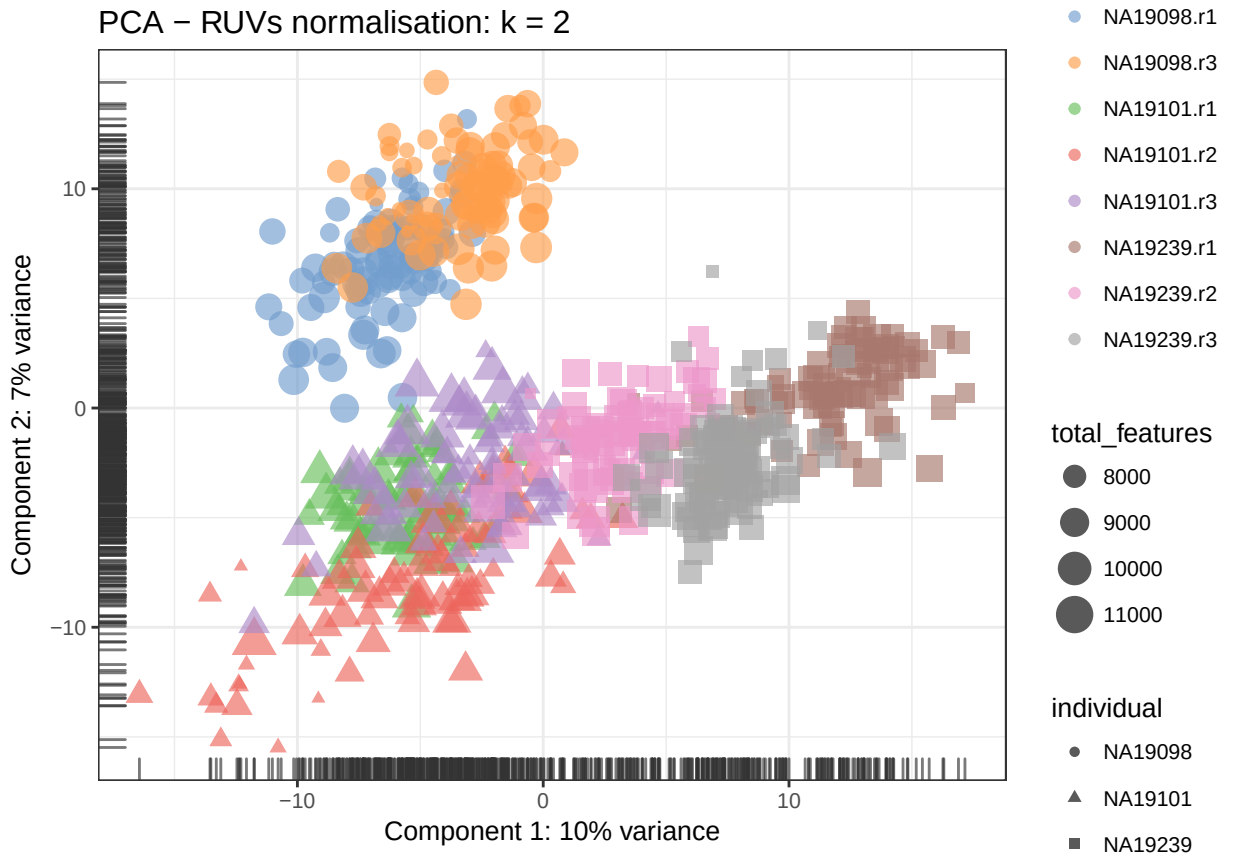

```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg2") +
    ggtitle("PCA - RUVg normalisation: k = 2")
```



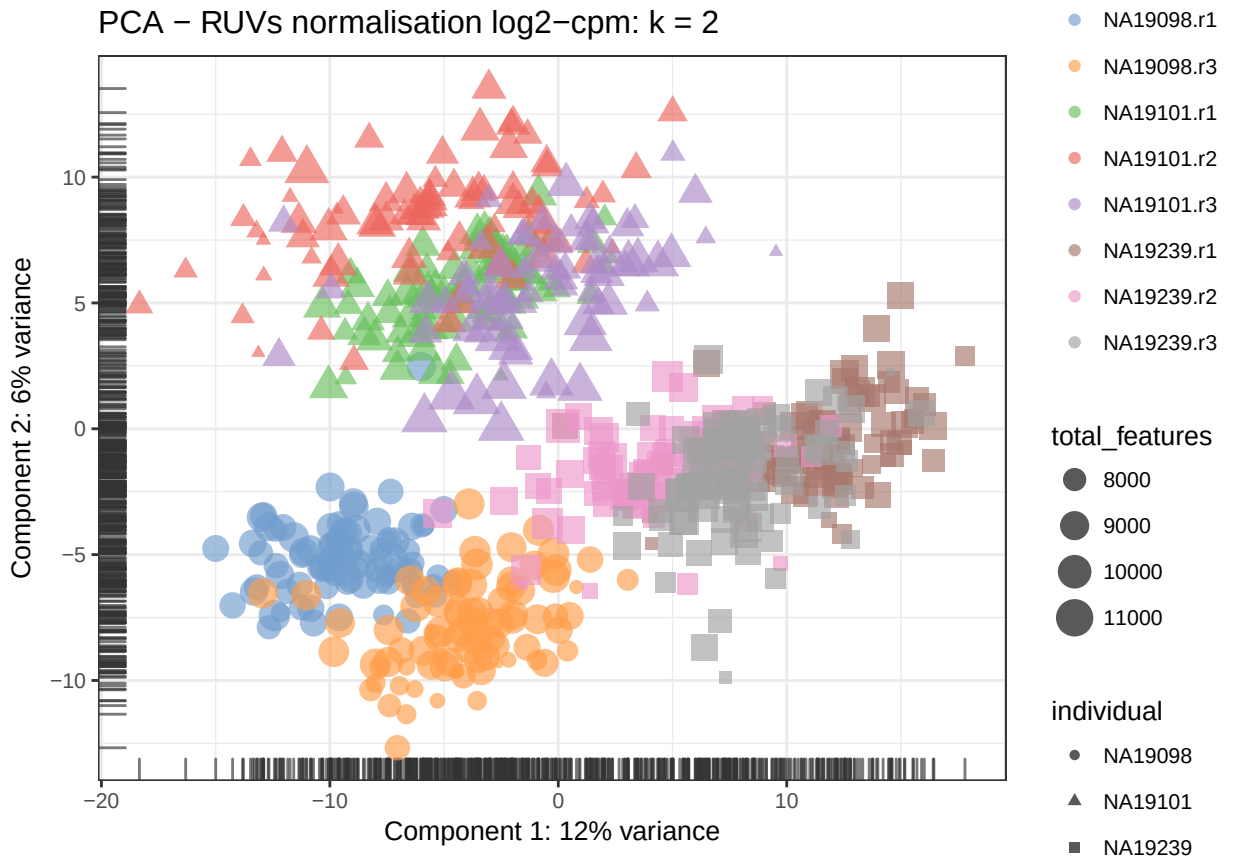

```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs1") +
    ggtitle("PCA - RUVs normalisation: k = 1")
```




```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs2") +
    ggtitle("PCA - RUVs normalisation: k = 2")
```




```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs2_logcpm") +
    ggtitle("PCA - RUVs normalisation log2-cpm: k = 2")
```

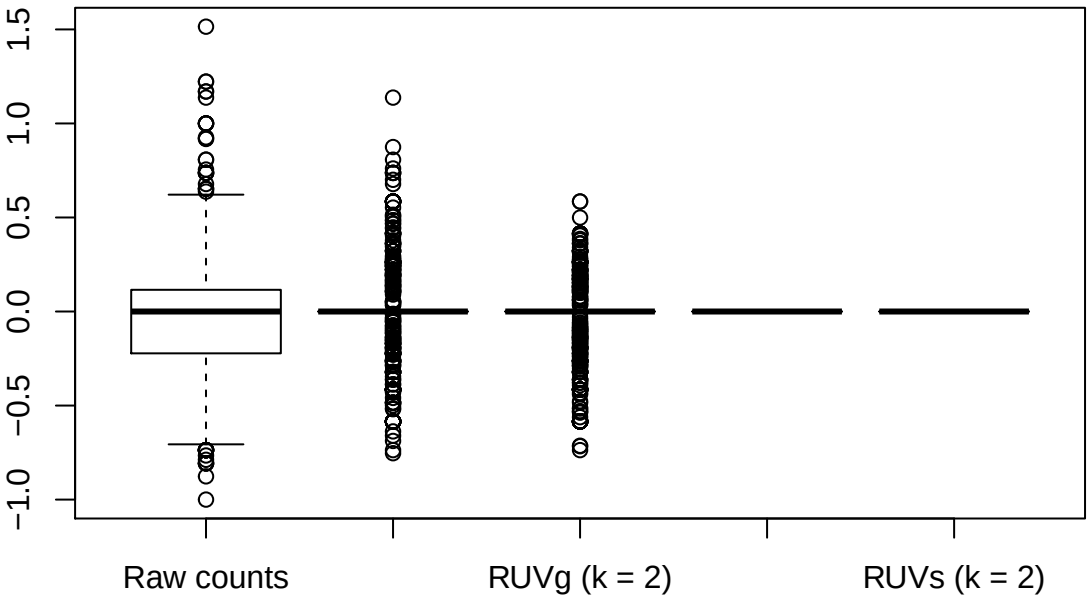


Plotting log2-normalized CPM from RUVs with k = 2 looks to give the best separation of cells by individual.

15.4 Effectiveness 2

We can also examine the effectiveness of correction using the relative log expression (RLE) across cells to confirm technical noise has been removed from the dataset.

```
boxplot(
    list(
        "Raw counts" = calc_cell_RLE(counts(umi.qc), erccs),
        "RUVg (k = 1)" = calc_cell_RLE(assayData(umi.qc)$ruvg1, erccs),
        "RUVg (k = 2)" = calc_cell_RLE(assayData(umi.qc)$ruvg2, erccs),
        "RUVs (k = 1)" = calc_cell_RLE(assayData(umi.qc)$ruvs1, erccs),
        "RUVs (k = 2)" = calc_cell_RLE(assayData(umi.qc)$ruvs2, erccs)
    )
)
```



15.5 Effectiveness 3

Another way of evaluating the effectiveness of correction is to look at the differentially expressed (DE) genes among the batches of the same individual Theoretically, these batches should not differ from each other. Let's take the most promising individual (**NA19101**, whose batches are the closest to each other) and check whether it is true.

For demonstration purposes we will only use a subset of cells. You should not do that with your real dataset, though.

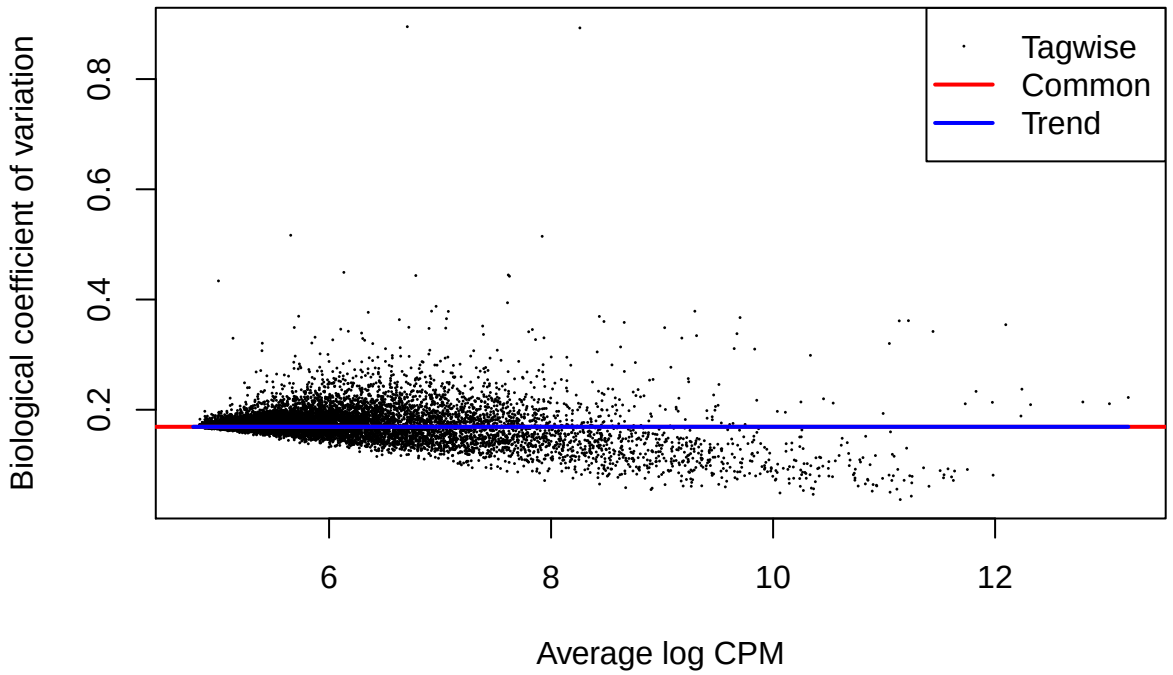
```
keep <- c(
    sample(which(umi.qc$batch == "NA19101.r1"), 20),
    sample(which(umi.qc$batch == "NA19101.r2"), 20),
    sample(which(umi.qc$batch == "NA19101.r3"), 20)
)
design <- model.matrix(~umi.qc[, keep]$batch)
```

We will use the edgeR package to calculate DE genes between plates for this particular individual. Recall that the input data for edgeR (and similar methods like DESeq2) must always be raw counts.

The particular coefficient that we test for DE in each case below tests to for genes that show a difference in expression between replicate plate 3 and replicate plate 1.

15.5.1 DE (raw counts)

```
dge1 <- DGEList(
    counts = counts(umi.qc[, keep]),
    norm.factors = rep(1, length(keep)),
    group = umi.qc[, keep]$batch
)
dge1 <- estimateDisp(dge1, design = design, trend.method = "none")
plotBCV(dge1)
```



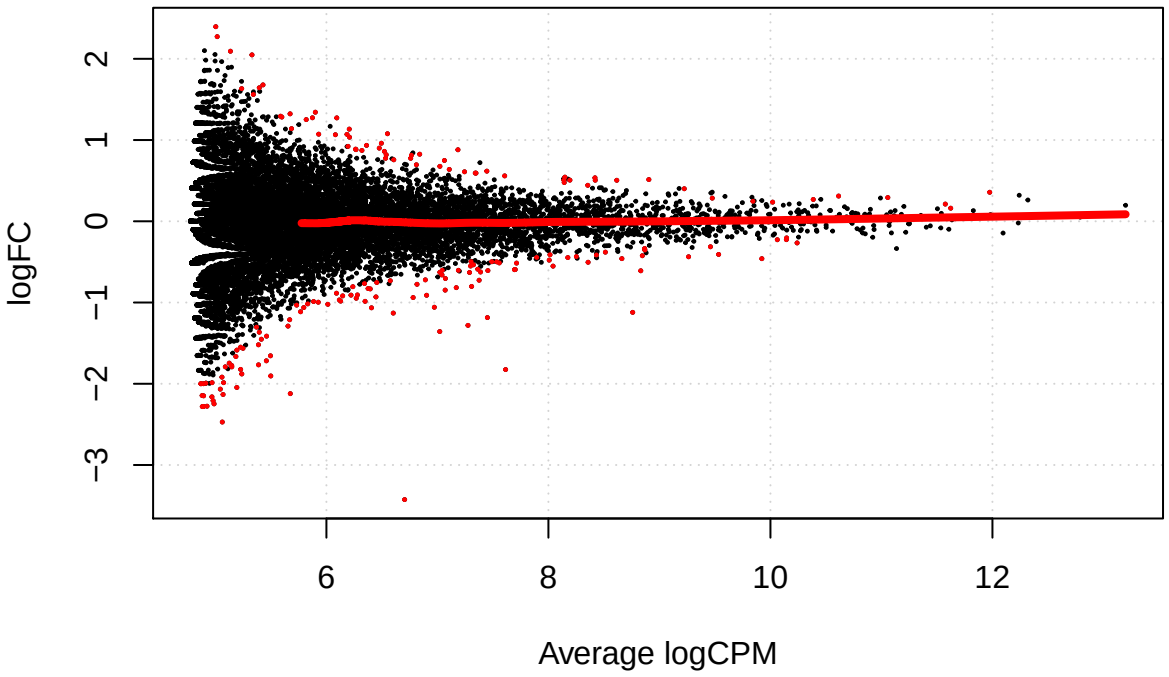

```
fit1 <- glmFit(dge1, design)
res1 <- glmLRT(fit1)
topTags(res1)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                      logFC   logCPM       LR       PValue          FDR
## ENSG00000187193 -1.8249610 7.614579 55.34873 1.009350e-13 1.419449e-09
## ENSG00000185885 -1.1208382 8.759197 52.63725 4.012078e-13 2.821092e-09
## ENSG00000163106 -1.3572375 7.021377 49.61840 1.867538e-12 8.754396e-09
## ENSG00000177105 -3.4255189 6.704809 47.40319 5.778854e-12 2.031701e-08
## ENSG00000034510 -0.4600144 9.922678 43.52045 4.195515e-11 1.180031e-07
## ENSG00000136160 -2.1215083 5.674802 42.06083 8.847781e-11 2.073772e-07
## ENSG00000131969 -1.2813327 7.275737 41.33951 1.279565e-10 2.570646e-07
## ENSG00000214265 -0.4364961 9.262806 38.42973 5.676101e-10 9.977876e-07
## ENSG00000008311 -1.1853929 7.450302 36.80701 1.304206e-09 2.037894e-06
## ENSG00000213719 -0.6090371 8.832039 31.17235 2.361051e-08 3.320346e-05
```

```
summary(decideTestsDGE(res1))
```

```
##    [,1]
## -1   121
## 0  13877
## 1     65
```

```
plotSmear(
    res1, lowess = TRUE,
    de.tags = rownames(topTags(res1, n = sum(abs(decideTestsDGE(res1))))$table)
)
```

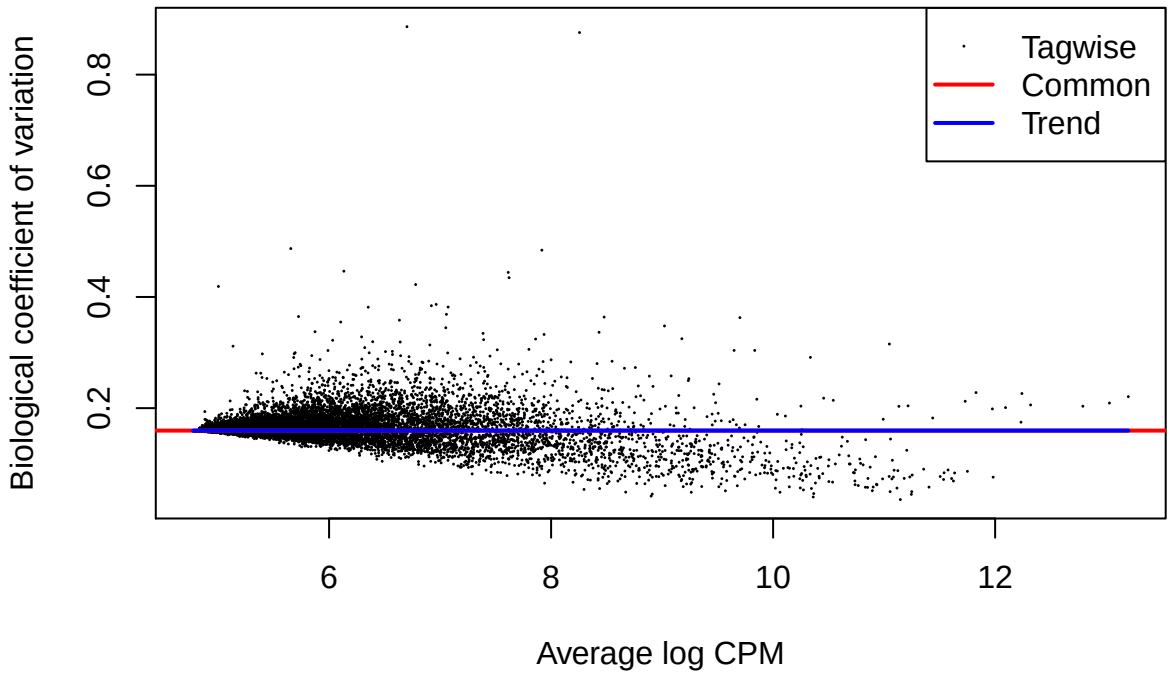


15.5.2 DE (RUVg, k = 2)

```
design_ruvg <- model.matrix(~ruvg$W[keep,] + umi.qc[, keep]$batch)
head(design_ruvg)
```

```
##   (Intercept) ruvg$W[keep, ]W_1 ruvg$W[keep, ]W_2
## 1           1       0.008323015       0.031336158
## 2           1       0.040433646       0.020314190
## 3           1      -0.003190918       0.042458389
## 4           1       0.017779972       0.009123186
## 5           1       0.056209769       0.024911550
## 6           1      -0.007517362       0.001527605
##   umi.qc[, keep]$batchNA19101.r2 umi.qc[, keep]$batchNA19101.r3
## 1                              0                              0
## 2                              0                              0
## 3                              0                              0
## 4                              0                              0
## 5                              0                              0
## 6                              0                              0
```

```
dge_ruvg <- estimateDisp(dge1, design = design_ruvg, trend.method = "none")
plotBCV(dge_ruvg)
```



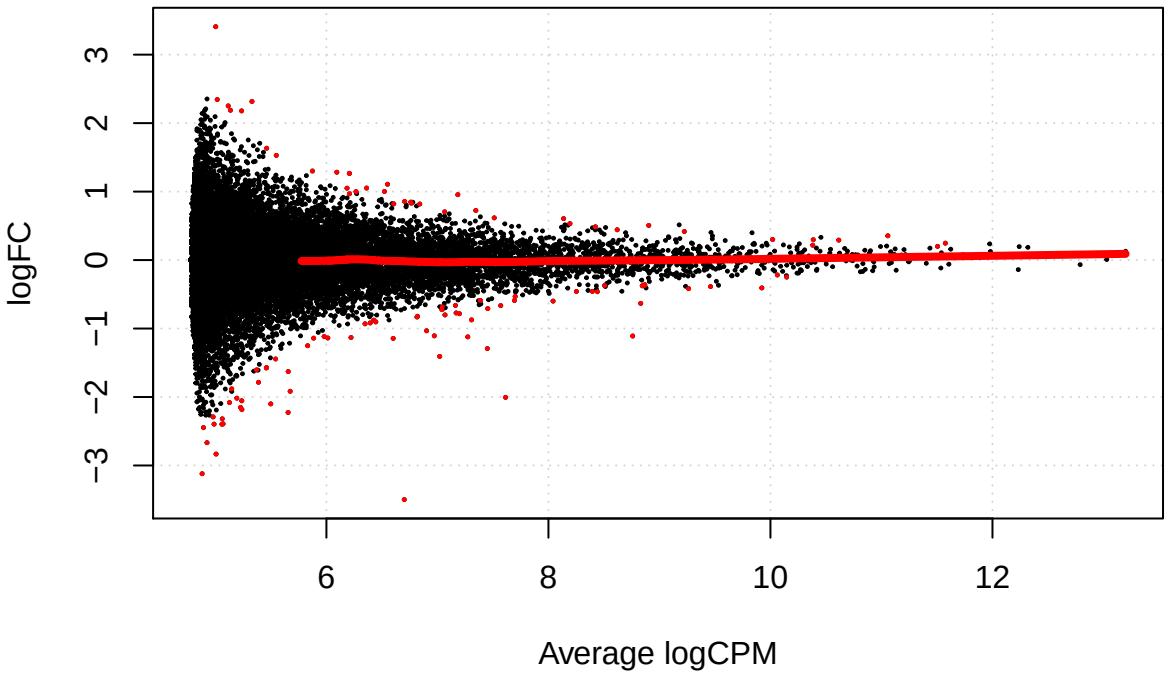

```
fit2 <- glmFit(dge_ruvg, design_ruvg)
res2 <- glmLRT(fit2)
topTags(res2)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                      logFC   logCPM       LR       PValue          FDR
## ENSG00000187193 -2.0050066 7.613732 53.21786 2.985318e-13 4.198253e-09
## ENSG00000185885 -1.1092275 8.758829 44.52763 2.507922e-11 1.763446e-07
## ENSG00000163106 -1.4068563 7.020663 42.92247 5.695273e-11 2.669754e-07
## ENSG00000177105 -3.4979127 6.701437 38.08253 6.781473e-10 2.384196e-06
## ENSG00000008311 -1.2922991 7.449479 35.66504 2.343321e-09 6.590825e-06
## ENSG00000034510 -0.4064682 9.922458 30.67318 3.053575e-08 6.525206e-05
## ENSG00000214265 -0.4170589 9.262833 30.55342 3.247987e-08 6.525206e-05
## ENSG00000131969 -1.1216389 7.273307 29.67683 5.104116e-08 8.972398e-05
## ENSG00000136160 -1.9165521 5.673737 28.95364 7.413149e-08 1.158346e-04
## ENSG00000213719 -0.6325498 8.831550 27.28054 1.759737e-07 2.474719e-04
```

```
summary(decideTestsDGE(res2))
```

```
##    [,1]
## -1    70
## 0  13953
## 1     40
```

```
plotSmear(
    res2, lowess = TRUE,
    de.tags = rownames(topTags(res2, n = sum(abs(decideTestsDGE(res2))))$table)
)
```

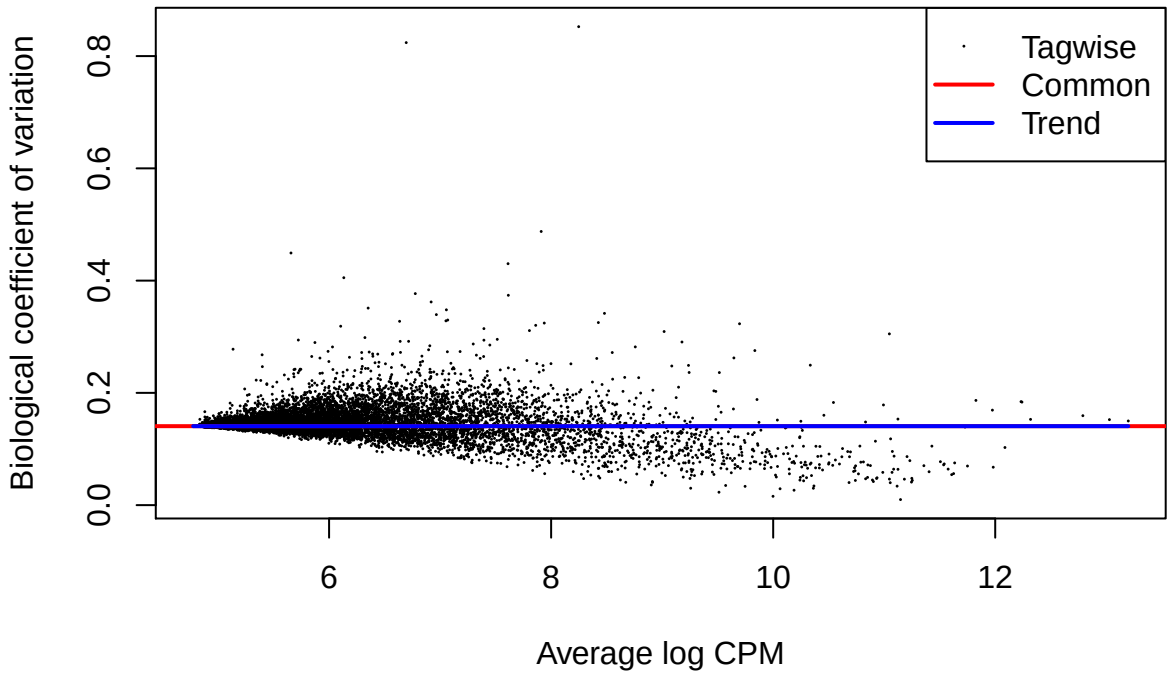


15.5.3 DE (RUVs, k = 2)

```
design_ruvs <- model.matrix(~ruvs$W[keep,] + umi.qc[, keep]$batch)
head(design_ruvs)
```

```
##   (Intercept) ruvs$W[keep, ]W_1 ruvs$W[keep, ]W_2
## 1           1         0.2825106       -0.09287973
## 2           1         0.2458254       -0.09005935
## 3           1         0.3028084       -0.09330027
## 4           1         0.2695799       -0.09943721
## 5           1         0.2381896       -0.09439478
## 6           1         0.2769587       -0.07453027
##   umi.qc[, keep]$batchNA19101.r2 umi.qc[, keep]$batchNA19101.r3
## 1                              0                              0
## 2                              0                              0
## 3                              0                              0
## 4                              0                              0
## 5                              0                              0
## 6                              0                              0
```

```
dge_ruvs <- estimateDisp(dge1, design = design_ruvs, trend.method = "none")
plotBCV(dge_ruvs)
```



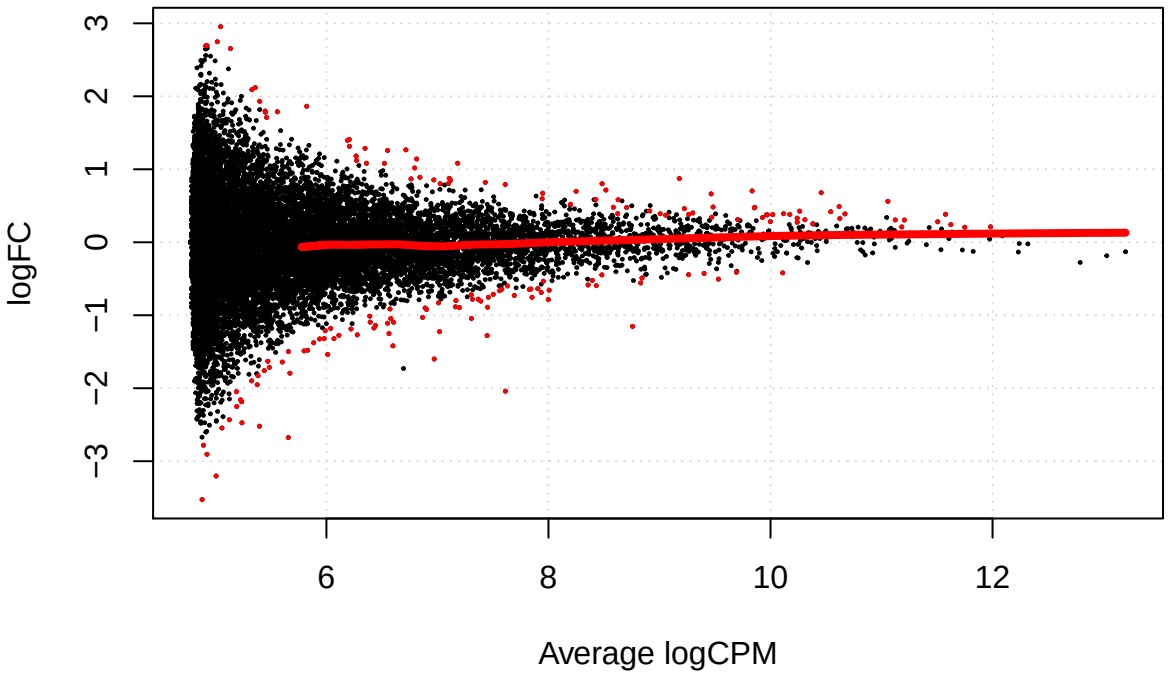

```
fit3 <- glmFit(dge_ruvs, design_ruvs)
res3 <- glmLRT(fit3)
topTags(res3)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                      logFC    logCPM       LR       PValue          FDR
## ENSG00000105372  0.3835537 11.576086 50.75977 1.043918e-12 1.468062e-08
## ENSG00000177954  0.5601185 11.057744 46.72800 8.155517e-12 5.734552e-08
## ENSG00000144713  0.4891221 10.618516 43.34386 4.591741e-11 1.728851e-07
## ENSG00000187193 -2.0397348  7.611963 43.20977 4.917445e-11 1.728851e-07
## ENSG00000165434 -1.5993126  6.971160 40.67757 1.795365e-10 5.049643e-07
## ENSG00000240972  0.6786264 10.456649 37.59573 8.703479e-10 2.039950e-06
## ENSG00000185885 -1.1526624  8.758077 33.96730 5.604618e-09 1.125968e-05
## ENSG00000110931 -1.4188280  6.599484 27.61558 1.479817e-07 2.601333e-04
## ENSG00000231500  0.3064817 11.124266 27.24091 1.796175e-07 2.624483e-04
## ENSG00000008311 -1.2790001  7.447851 26.99367 2.041230e-07 2.624483e-04
```

```
summary(decideTestsDGE(res3))
```

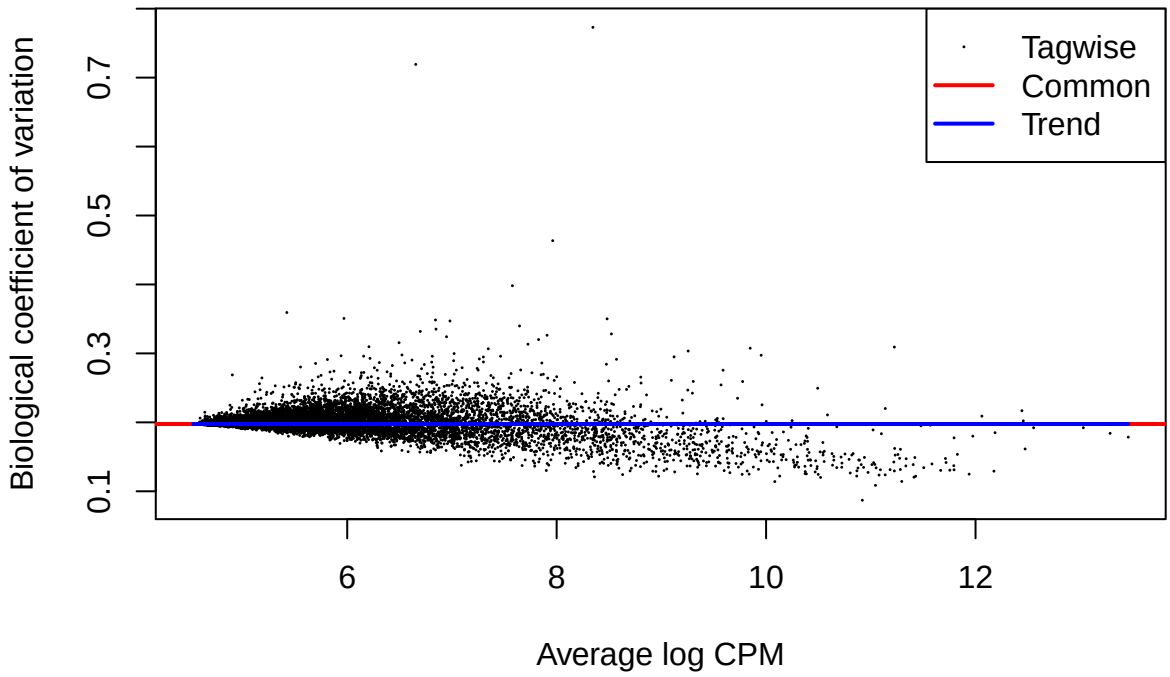
```
##    [,1]
## -1    87
## 0  13890
## 1     86
```

```
plotSmear(
    res3, lowess = TRUE,
    de.tags = rownames(topTags(res3, n = sum(abs(decideTestsDGE(res3))))$table)
)
```



In the above analyses, we have ignored size factors between cells. A typical edgeR analysis would always include these.

```
umi.qc <- scran::computeSumFactors(umi.qc, sizes = 15)
dge_ruvs$samples$norm.factors <- sizeFactors(umi.qc)[keep]
dge_ruvs_sf <- estimateDisp(dge_ruvs, design = design_ruvs, trend.method = "none")
plotBCV(dge_ruvs_sf)
```



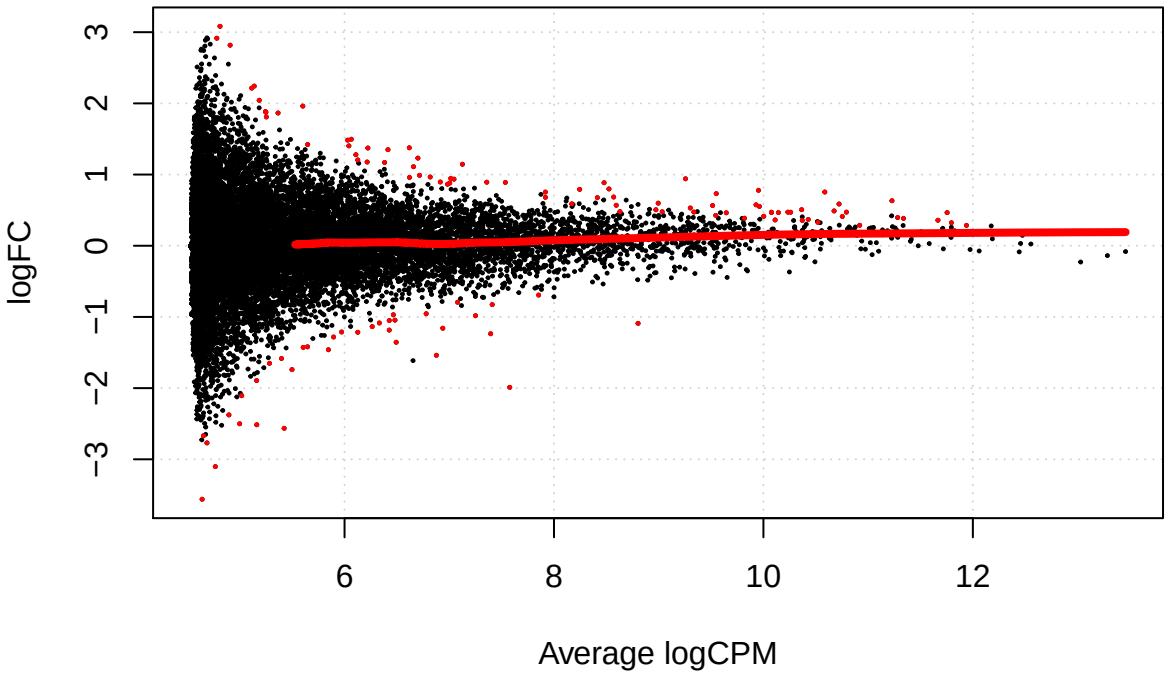

```
fit4 <- glmFit(dge_ruvs_sf, design_ruvs)
res4 <- glmLRT(fit4)
topTags(res4)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                       logFC    logCPM       LR       PValue          FDR
## ENSG00000187193 -1.9887173  7.577384 45.28516 1.703345e-11 2.395415e-07
## ENSG00000144713  0.5865548 10.722464 37.80708 7.809764e-10 4.187302e-06
## ENSG00000177954  0.6327277 11.227681 37.54505 8.932593e-10 4.187302e-06
## ENSG00000165434 -1.5385102  6.876850 35.89755 2.079699e-09 7.311701e-06
## ENSG00000185885 -1.0900427  8.803079 32.81663 1.012748e-08 2.848455e-05
## ENSG00000105372  0.4659271 11.754307 30.74684 2.939814e-08 6.375023e-05
## ENSG00000240972  0.7546141 10.586115 30.59860 3.173232e-08 6.375023e-05
## ENSG00000198918  0.7317812  9.549626 27.43608 1.623737e-07 2.854327e-04
## ENSG00000008311 -1.2362113  7.395390 26.85910 2.188416e-07 3.419522e-04
## ENSG00000125144 -2.5644487  5.423306 25.19082 5.192890e-07 6.750701e-04
```

```
summary(decideTestsDGE(res4))
```

```
##    [,1]
## -1   36
## 0  13944
## 1     83
```

```
plotSmear(
    res4, lowess = TRUE,
    de.tags = rownames(topTags(res4, n = sum(abs(decideTestsDGE(res4))))$table)
)
```



15.6 Exercise

Perform the same analysis with read counts of the Blischak data. Use `blischak/reads.rds` file to load the reads SCESet object. Once you have finished please compare your results to ours (next chapter). Additionally, experiment with other combinations of normalizations and compare the results.

Chapter 16

Dealing with confounders (Reads)

```
library(scRNA.seq.funcs)
library(RUVSeq)
library(scater, quietly = TRUE)
library(scran)
library(edgeR)
options(stringsAsFactors = FALSE)
reads <- readRDS("blischak/reads.rds")
reads.qc <- reads[fData(reads)$use, pData(reads)$use]
endog_genes <- !fData(reads.qc)$is_feature_control
erccs <- fData(reads.qc)$is_feature_control
```

16.1 Remove Unwanted Variation

16.1.1 RUVg

```
ruvg <- RUVg(counts(reads.qc), erccs, k = 1)
set_exprs(reads.qc, "ruvg1") <- ruvg$normalizedCounts
ruvg <- RUVg(counts(reads.qc), erccs, k = 2)
set_exprs(reads.qc, "ruvg2") <- ruvg$normalizedCounts
set_exprs(reads.qc, "ruvg2_logcpm") <- log2(t(t(ruvg$normalizedCounts) /
                                          colSums(ruvg$normalizedCounts)) + 1)
```

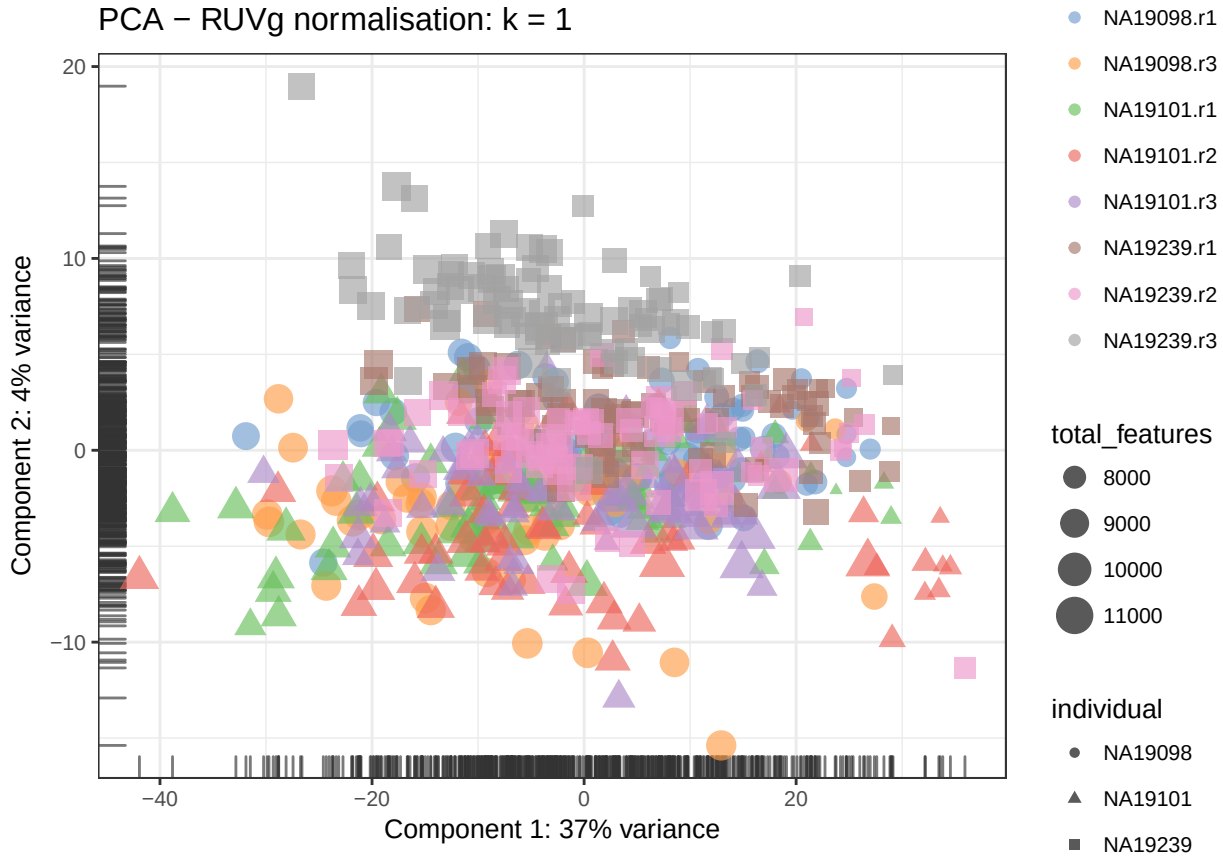
16.1.2 RUVs

```
scIdx <- matrix(-1, ncol = max(table(reads.qc$individual)), nrow = 3)
tmp <- which(reads.qc$individual == "NA19098")
scIdx[1, 1:length(tmp)] <- tmp
tmp <- which(reads.qc$individual == "NA19101")
scIdx[2, 1:length(tmp)] <- tmp
tmp <- which(reads.qc$individual == "NA19239")
scIdx[3, 1:length(tmp)] <- tmp
cIdx <- rownames(reads.qc)
ruvs <- RUVs(counts(reads.qc), cIdx, k = 1, scIdx = scIdx, isLog = FALSE)
```

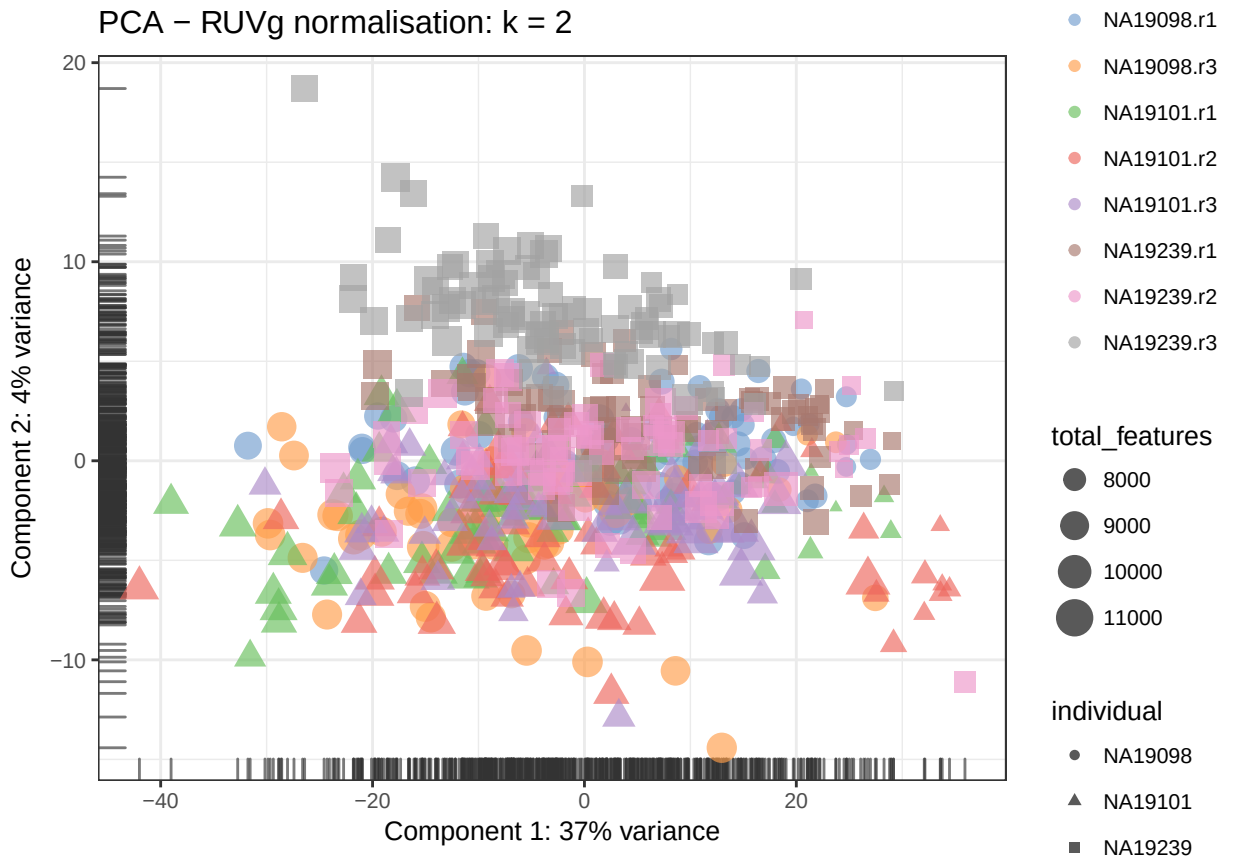
```
set_exprs(reads.qc, "ruvs1") <- ruvs$normalizedCounts
ruvs <- RUVs(counts(reads.qc), cIdx, k = 2, scIdx = scIdx, isLog = FALSE)
set_exprs(reads.qc, "ruvs2") <- ruvs$normalizedCounts
set_exprs(reads.qc, "ruvs2_logcpm") <- log2(t(t(ruvs$normalizedCounts) /
                                              colSums(ruvs$normalizedCounts)) + 1)
```

16.2 Effectiveness 1

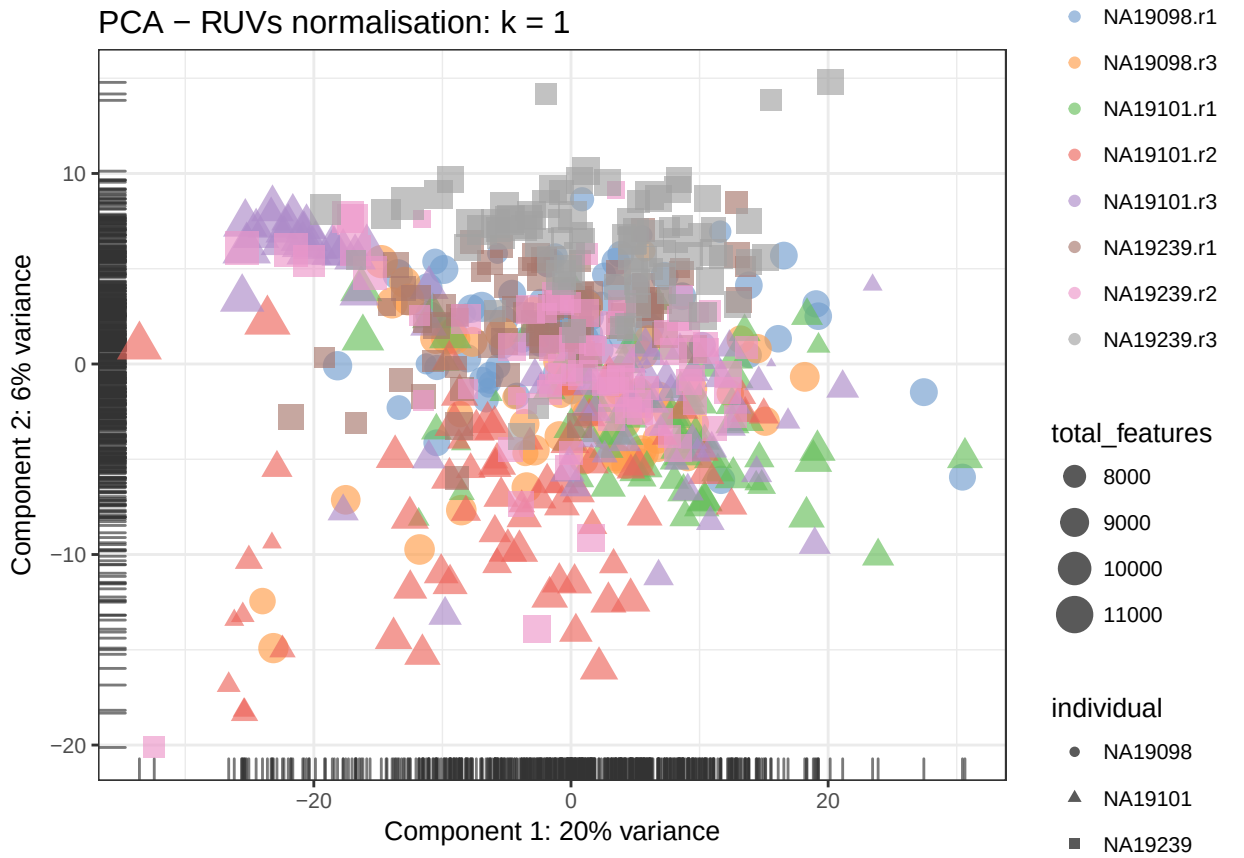
```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg1") +
    ggtitle("PCA - RUVg normalisation: k = 1")
```



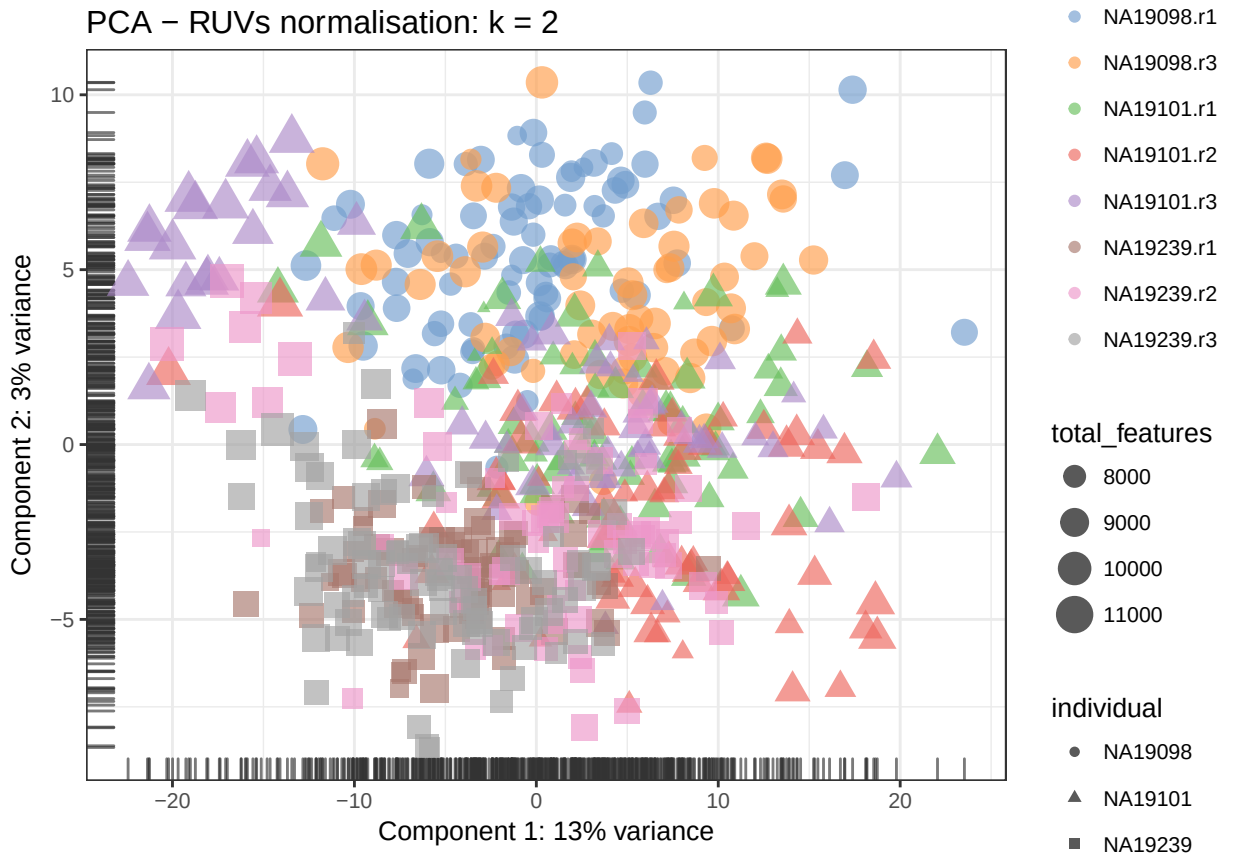

```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg2") +
    ggtitle("PCA - RUVg normalisation: k = 2")
```



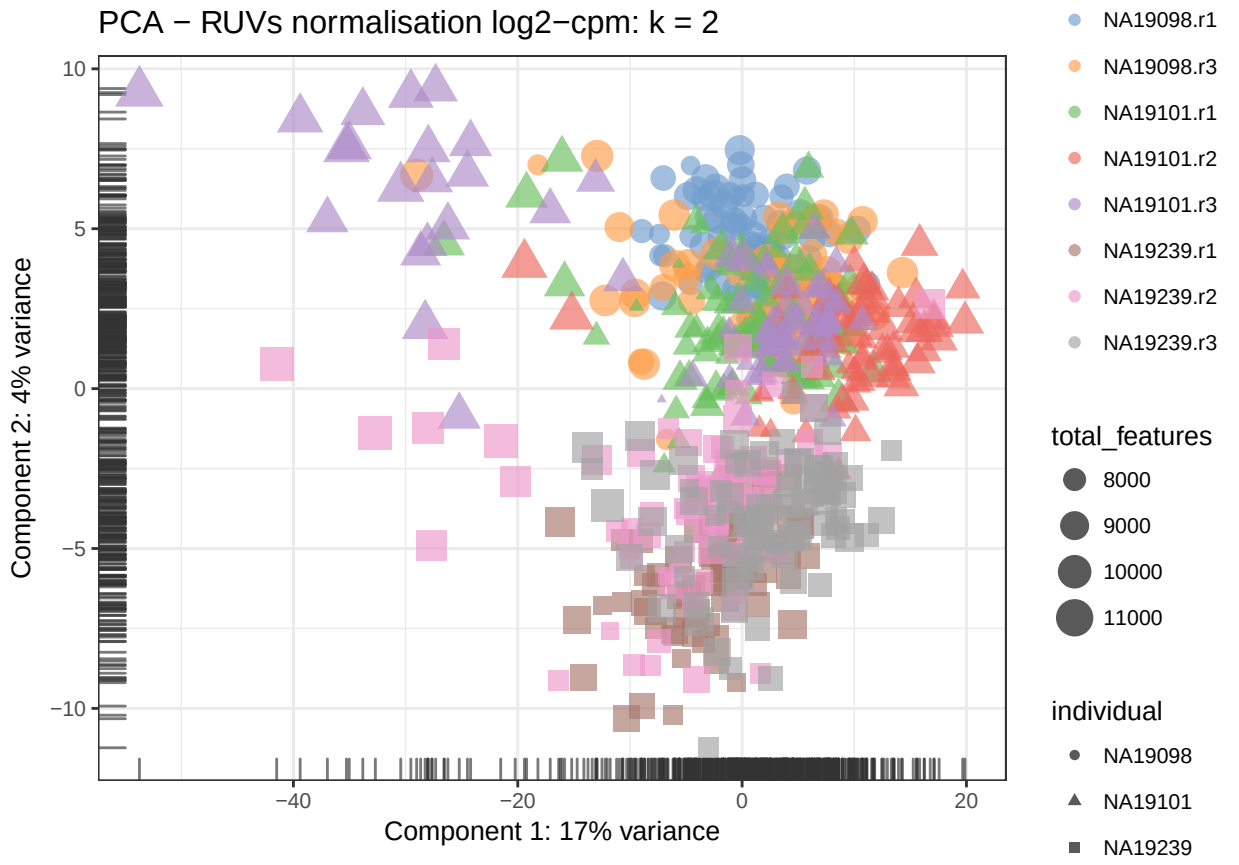

```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs1") +
    ggtitle("PCA - RUVs normalisation: k = 1")
```




```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs2") +
    ggtitle("PCA - RUVs normalisation: k = 2")
```

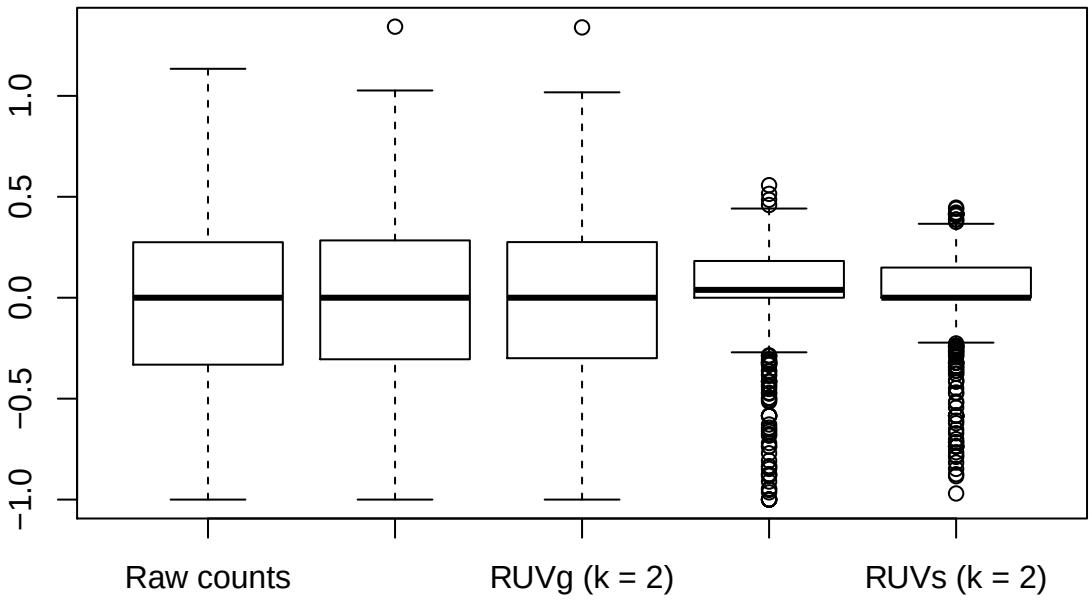



```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs2_logcpm") +
    ggtitle("PCA - RUVs normalisation log2-cpm: k = 2")
```



16.3 Effectiveness 2

```
boxplot(
    list(
        "Raw counts" = calc_cell_RLE(counts(reads.qc), erccs),
        "RUVg (k = 1)" = calc_cell_RLE(assayData(reads.qc)$ruvg1, erccs),
        "RUVg (k = 2)" = calc_cell_RLE(assayData(reads.qc)$ruvg2, erccs),
        "RUVs (k = 1)" = calc_cell_RLE(assayData(reads.qc)$ruvs1, erccs),
        "RUVs (k = 2)" = calc_cell_RLE(assayData(reads.qc)$ruvs2, erccs)
    )
)
```

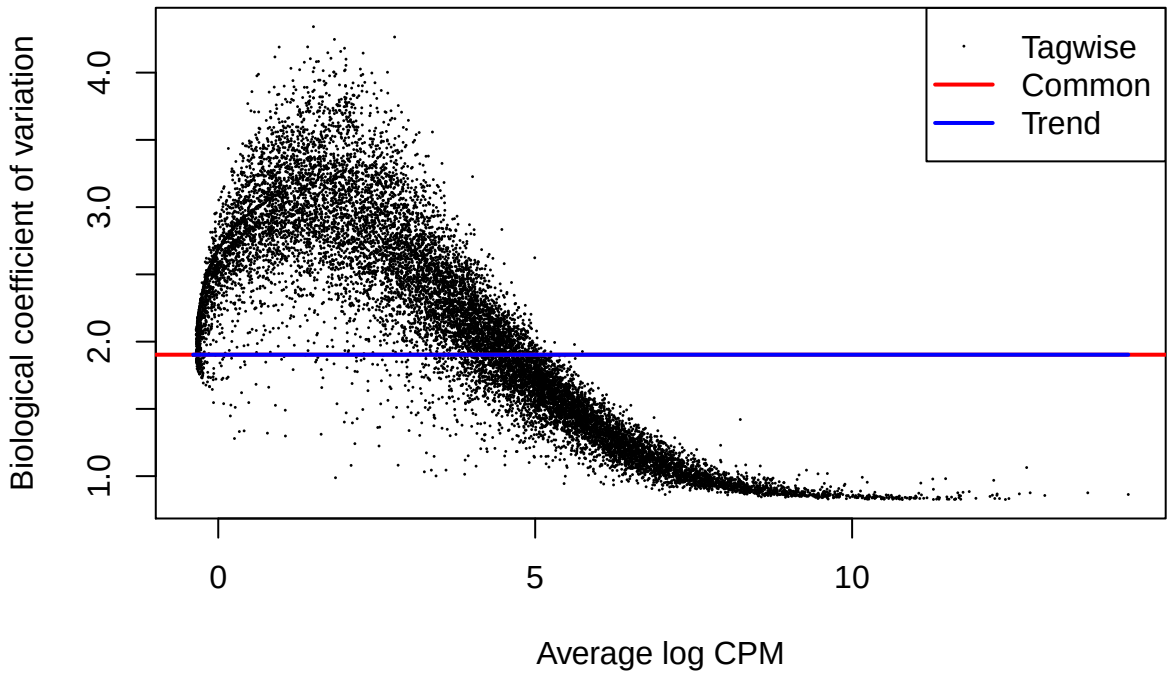


16.4 Effectiveness 3

```
keep <- c(
    sample(which(reads.qc$batch == "NA19101.r1"), 20),
    sample(which(reads.qc$batch == "NA19101.r2"), 20),
    sample(which(reads.qc$batch == "NA19101.r3"), 20)
)
design <- model.matrix(~reads.qc[, keep]$batch)
```

16.4.1 DE (raw counts)

```
dge1 <- DGEList(
    counts = counts(reads.qc[, keep]),
    norm.factors = rep(1, length(keep)),
    group = reads.qc[, keep]$batch
)
dge1 <- estimateDisp(dge1, design = design, trend.method = "none")
plotBCV(dge1)
```



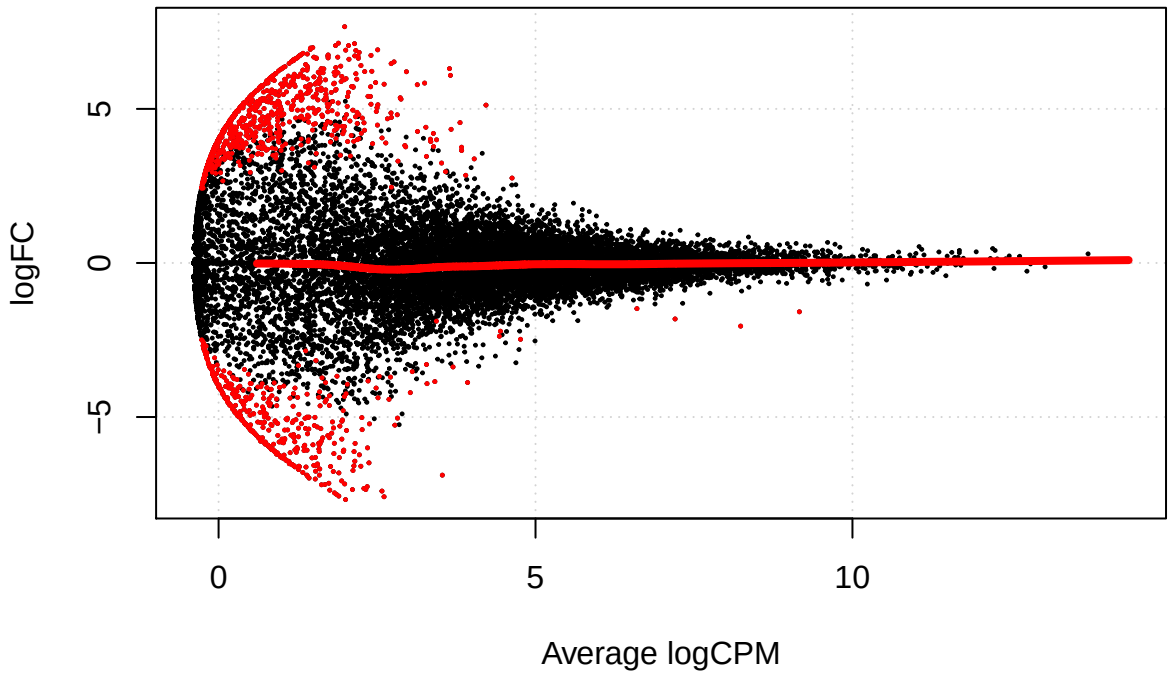

```
fit1 <- glmFit(dge1, design)
res1 <- glmLRT(fit1)
topTags(res1)
```

```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                     logFC   logCPM       LR       PValue          FDR
## ENSG00000138650 -7.678806 2.003747 33.84554 5.966578e-09 9.582922e-05
## ENSG00000109743  7.667826 1.989344 31.97871 1.558716e-08 1.251727e-04
## ENSG00000044446  6.087456 3.657979 29.89003 4.572581e-08 2.448007e-04
## ENSG00000160307  6.768762 1.307775 27.95346 1.242684e-07 4.978484e-04
## ENSG00000146021  5.784734 3.135279 27.52613 1.549867e-07 4.978484e-04
## ENSG00000112137  6.132768 1.292363 27.14609 1.886459e-07 5.049737e-04
## ENSG00000214188  6.988412 1.485779 25.73225 3.922183e-07 8.999170e-04
## ENSG00000171551  6.742559 1.289356 25.12943 5.360833e-07 9.988565e-04
## ENSG00000171208  6.307105 3.643464 25.04622 5.597228e-07 9.988565e-04
## ENSG00000131849  6.618593 1.204518 23.98708 9.698418e-07 1.415977e-03
```

```
summary(decideTestsDGE(res1))
```

```
##    [,1]
## -1   441
## 0  14859
## 1    761
```

```
plotSmear(
    res1, lowess = TRUE,
    de.tags = rownames(topTags(res1, n = sum(abs(decideTestsDGE(res1))))$table)
)
```

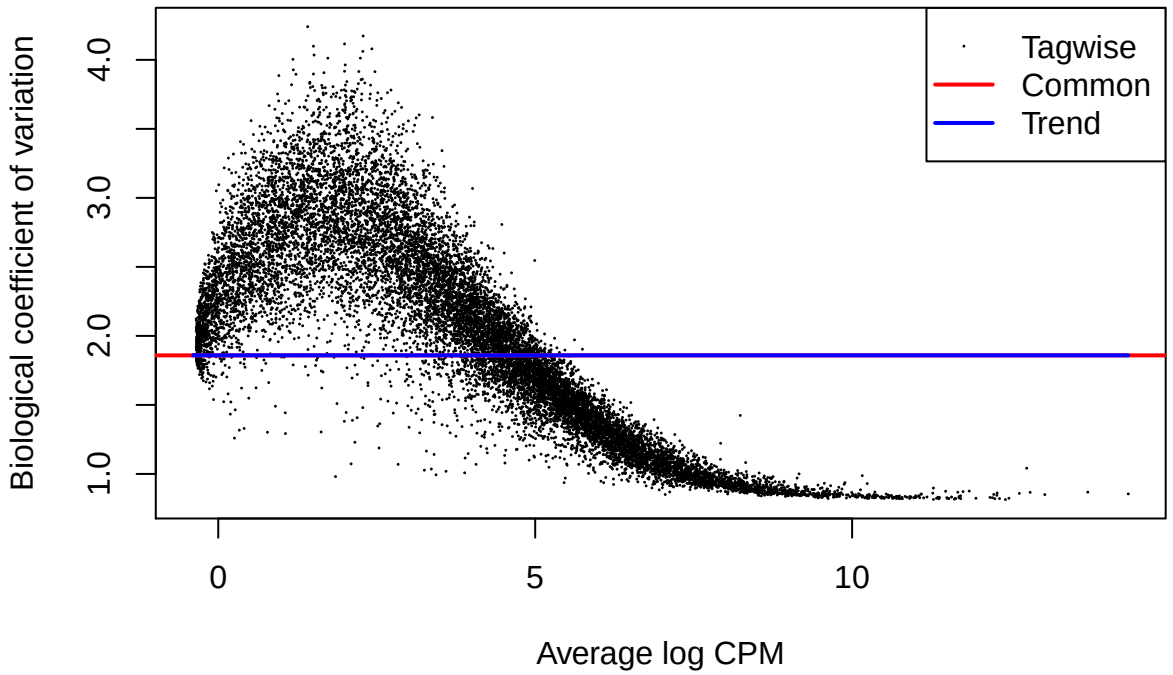


16.4.2 DE (RUVg, k = 2)

```
design_ruvg <- model.matrix(~ruvg$W[keep,] + reads.qc[, keep]$batch)
head(design_ruvg)
```

```
##   (Intercept) ruvg$W[keep, ]W_1 ruvg$W[keep, ]W_2
## 1           1       0.016561929      -0.008540817
## 2           1      -0.025856039       0.027531992
## 3           1      -0.009896783       0.019871106
## 4           1       0.125910579       0.007593017
## 5           1      -0.006315637       0.035765872
## 6           1      -0.049085371      -0.010073119
##   reads.qc[, keep]$batchNA19101.r2 reads.qc[, keep]$batchNA19101.r3
## 1                                0                                0
## 2                                0                                0
## 3                                0                                0
## 4                                0                                0
## 5                                0                                0
## 6                                0                                0
```

```
dge_ruvg <- estimateDisp(dge1, design = design_ruvg, trend.method = "none")
plotBCV(dge_ruvg)
```



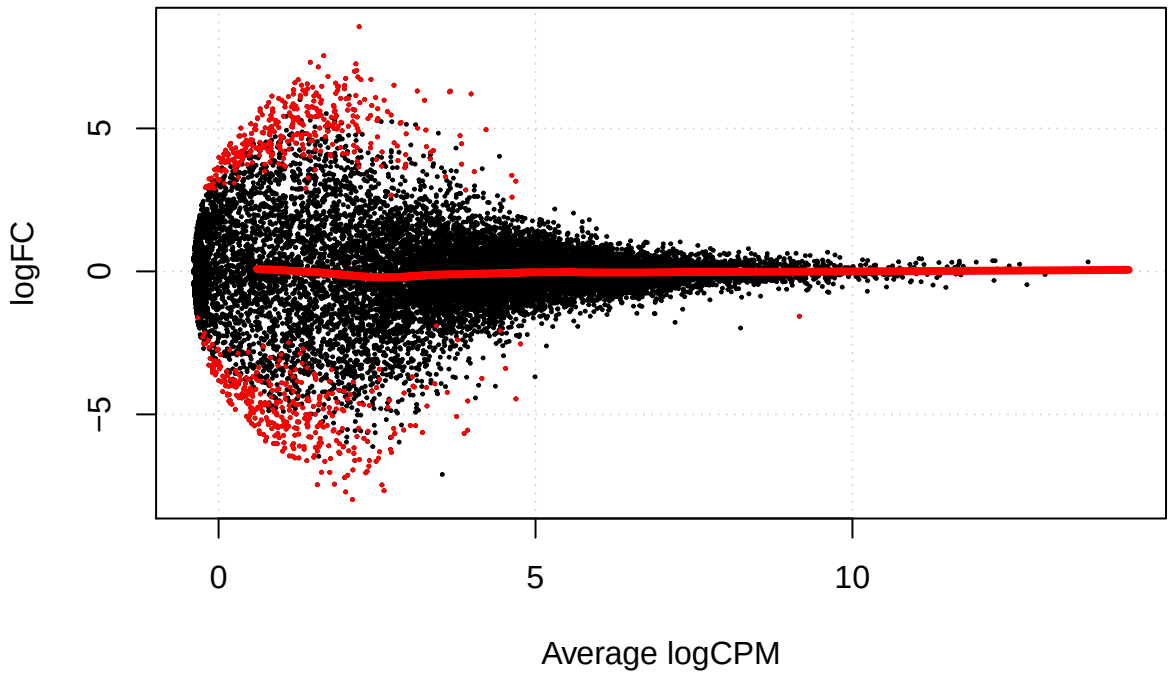

```
fit2 <- glmFit(dge_ruvg, design_ruvg)
res2 <- glmLRT(fit2)
topTags(res2)
```

```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                     logFC    logCPM       LR       PValue          FDR
## ENSG00000138650 -7.710654 2.0040978 33.39202 7.533247e-09 0.0001209915
## ENSG00000146021  6.307282 3.1352154 31.77292 1.732924e-08 0.0001391625
## ENSG00000044446  6.303099 3.6579550 30.61585 3.145133e-08 0.0001683799
## ENSG00000163219 -4.822317 0.8585435 26.69957 2.376748e-07 0.0007734088
## ENSG00000160307  6.512451 1.3076718 26.48642 2.653971e-07 0.0007734088
## ENSG00000112137  6.072095 1.2925307 26.32234 2.889267e-07 0.0007734088
## ENSG00000189134 -4.366128 0.5390822 25.74219 3.902025e-07 0.0008764743
## ENSG00000145431  7.026457 2.1802636 25.52551 4.365727e-07 0.0008764743
## ENSG00000169403  8.557565 2.2185715 25.05101 5.583337e-07 0.0009963775
## ENSG00000109743  6.501223 1.9894723 24.72481 6.612748e-07 0.0010620735
```

```
summary(decideTestsDGE(res2))
```

```
##    [,1]
## -1   371
## 0  15345
## 1    345
```

```
plotSmear(
    res2, lowess = TRUE,
    de.tags = rownames(topTags(res2, n = sum(abs(decideTestsDGE(res2))))$table)
)
```

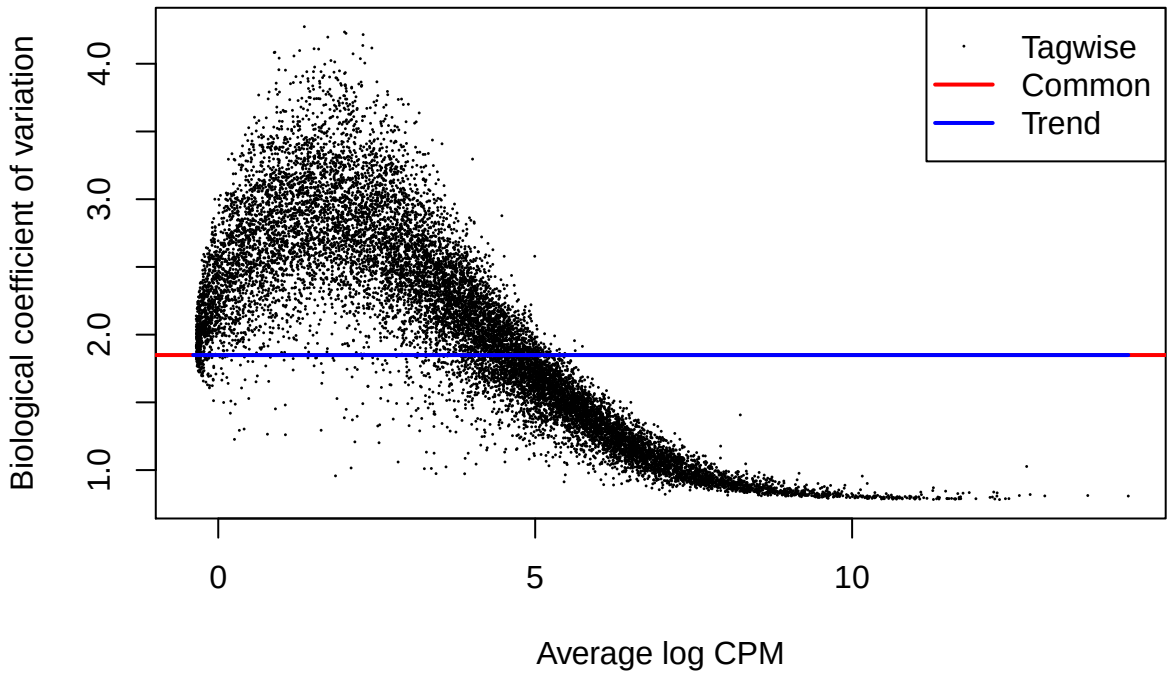


16.4.3 DE (RUVs, k = 2)

```
design_ruvs <- model.matrix(~ruvs$W[keep,] + reads.qc[, keep]$batch)
head(design_ruvs)
```

```
##   (Intercept) ruvs$W[keep, ]W_1 ruvs$W[keep, ]W_2
## 1           1         0.3393820         0.2006604
## 2           1         0.2879353         0.1807737
## 3           1         0.3176779         0.2226902
## 4           1         0.3775897         0.1547174
## 5           1         0.3144554         0.1941110
## 6           1         0.2172195         0.1480125
##   reads.qc[, keep]$batchNA19101.r2 reads.qc[, keep]$batchNA19101.r3
## 1                                0                                0
## 2                                0                                0
## 3                                0                                0
## 4                                0                                0
## 5                                0                                0
## 6                                0                                0
```

```
dge_ruvs <- estimateDisp(dge1, design = design_ruvs, trend.method = "none")
plotBCV(dge_ruvs)
```



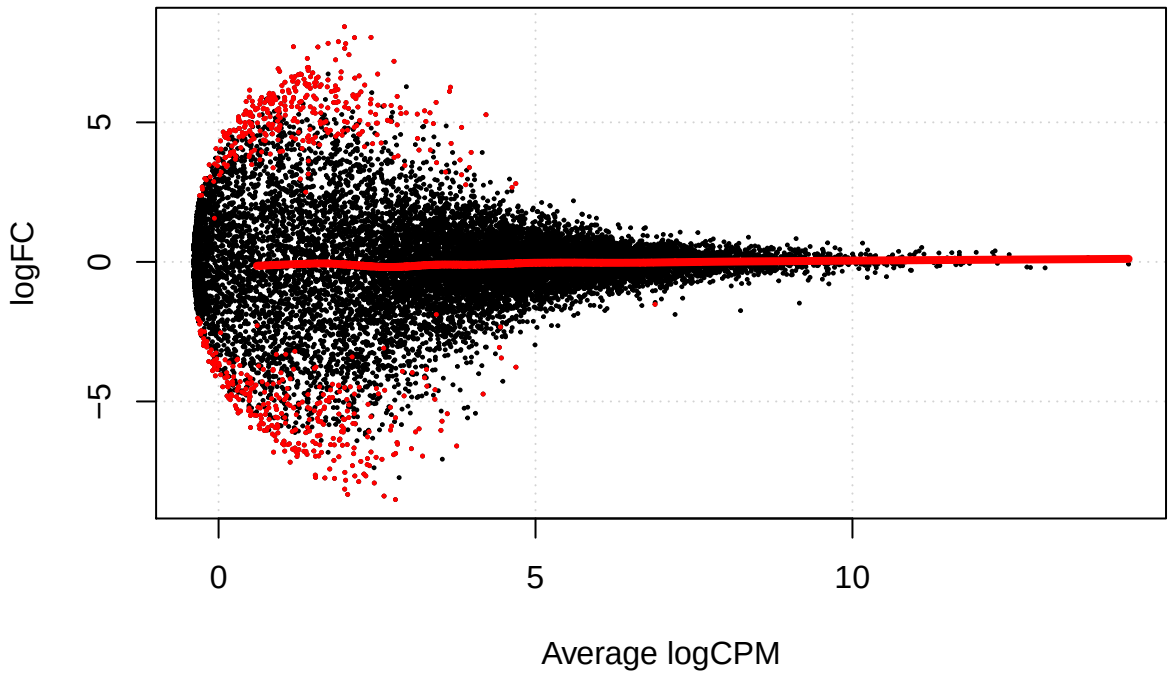

```
fit3 <- glmFit(dge_ruvs, design_ruvs)
res3 <- glmLRT(fit3)
topTags(res3)
```

```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                      logFC    logCPM       LR       PValue          FDR
## ENSG00000138650 -7.822534 2.0041784 35.62401 2.393199e-09 3.843717e-05
## ENSG00000144962 -7.757015 1.8291101 33.01236 9.157473e-09 7.353908e-05
## ENSG00000044446  6.258706 3.6579495 31.79771 1.710940e-08 9.159800e-05
## ENSG00000169071 -6.816668 0.8677284 30.56531 3.228138e-08 9.302654e-05
## ENSG00000128683 -5.977243 0.6514647 30.37054 3.569081e-08 9.302654e-05
## ENSG00000108231  6.633075 1.4514755 30.12366 4.053558e-08 9.302654e-05
## ENSG00000177483 -8.142323 1.9894202 30.00973 4.298845e-08 9.302654e-05
## ENSG00000085276 -7.113673 1.8199658 29.86430 4.633661e-08 9.302654e-05
## ENSG00000112619 -5.232082 0.6578571 28.80205 8.016642e-08 1.345195e-04
## ENSG00000090554 -4.896455 0.8363723 28.71723 8.375540e-08 1.345195e-04
```

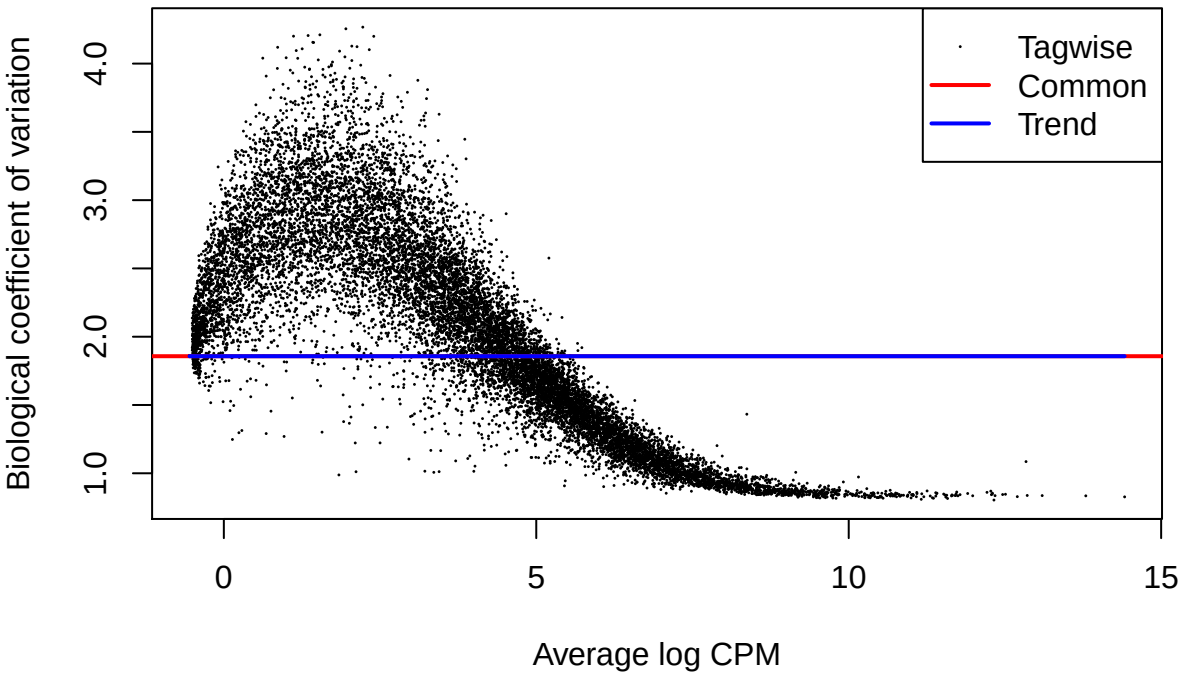
```
summary(decideTestsDGE(res3))
```

```
##    [,1]
## -1   331
## 0  15387
## 1    343
```

```
plotSmear(
    res3, lowess = TRUE,
    de.tags = rownames(topTags(res3, n = sum(abs(decideTestsDGE(res3))))$table)
)
```




```
reads.qc <- scran::computeSumFactors(reads.qc, sizes = 15)
dge_ruvs$samples$norm.factors <- sizeFactors(reads.qc)[keep]
dge_ruvs_sf <- estimateDisp(dge_ruvs, design = design_ruvs, trend.method = "none")
plotBCV(dge_ruvs_sf)
```




```
fit4 <- glmFit(dge_ruvs_sf, design_ruvs)
res4 <- glmLRT(fit4)
topTags(res4)
```

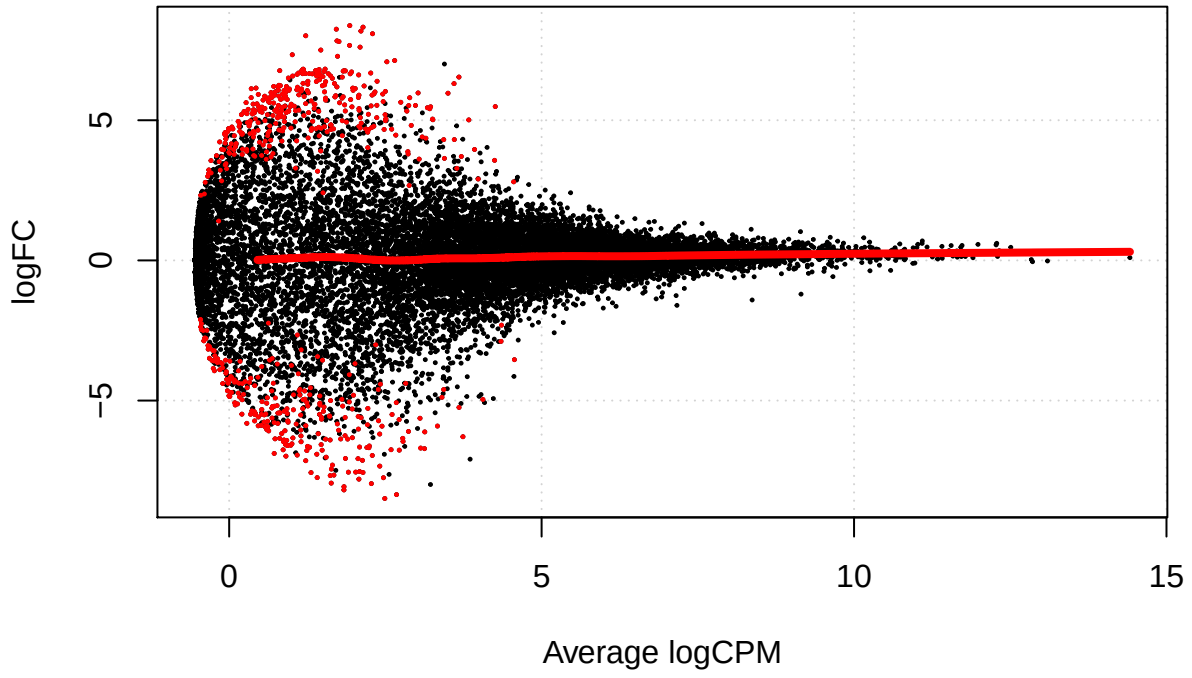
```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                      logFC     logCPM       LR       PValue          FDR
```

```
## ENSG00000138650 -7.659682 1.67575067 33.87807 5.867651e-09 0.0000580736
## ENSG00000044446  6.540458 3.67746787 33.47149 7.231629e-09 0.0000580736
## ENSG00000144962 -7.676915 1.61779567 31.08434 2.470564e-08 0.0001322658
## ENSG00000108231  6.591068 1.46349823 29.94053 4.455014e-08 0.0001788800
## ENSG00000169071 -6.886334 0.72057266 29.36268 6.002242e-08 0.0001927710
## ENSG00000128683 -5.900157 0.46520459 28.96348 7.375594e-08 0.0001927710
## ENSG00000085276 -7.020925 1.55238975 28.62206 8.797410e-08 0.0001927710
## ENSG00000177483 -8.071504 1.83803493 28.45263 9.601941e-08 0.0001927710
## ENSG00000146021  5.446115 3.17203186 27.10201 1.929969e-07 0.0003256494
## ENSG00000162989 -5.075773 0.09068354 26.70524 2.369776e-07 0.0003256494
```

```
summary(decideTestsDGE(res4))
```

```
##    [,1]
## -1   264
## 0  15452
## 1    345
```

```
plotSmear(
    res4, lowess = TRUE,
    de.tags = rownames(topTags(res4, n = sum(abs(decideTestsDGE(res4))))$table)
)
```



Chapter 17

Clustering Introduction

Once we have normalized the data and removed confounders we can carry out analyses that will allow us to interpret the data biologically. The exact nature of the analysis depends on the dataset and the biological question at hand. Nevertheless, there are a few operations which are useful in a wide range of contexts and we will be discussing some of them. We will start with the clustering of scRNA-seq data.

17.1 Introduction

One of the most promising applications of scRNA-seq is the discovery and annotation of cell-types based on the transcription profiles. Computationally, this is a hard problem as it amounts to **unsupervised clustering**. That is, we need to identify groups of cells based on the similarities of the transcriptomes without any prior knowledge of the labels. The problem is made more challenging due to the high level of noise and the large number of dimensions (i.e. genes).

17.2 Dimensionality reductions

When working with large datasets, it can often be beneficial to apply some sort of dimensionality reduction method. By projecting the data onto a lower-dimensional sub-space, one is often able to significantly reduce the amount of noise. An additional benefit is that it is typically much easier to visualize the data in a 2 or 3-dimensional subspace. Here we will introduce some of the popular dimensionality reduction methods.

17.2.1 PCA

PCA analysis was introduced in chapter 9.2.

17.2.2 Spectral

Spectral decomposition is the factorization of a matrix into a canonical form, whereby the matrix is represented in terms of its eigenvalues and eigenvectors.

In application to scRNA-seq data, the matrix can be either an input expression matrix, or matrix of distances between the cells. The computed eigenvectors are similar to the projections of the data to PCs (chapter 9.2.).

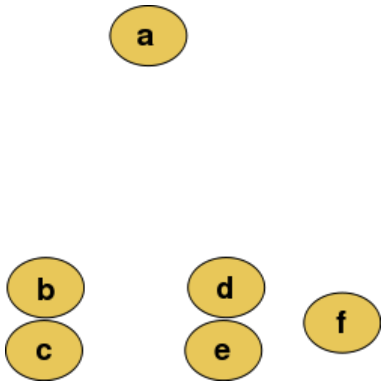


Figure 17.1: Raw data

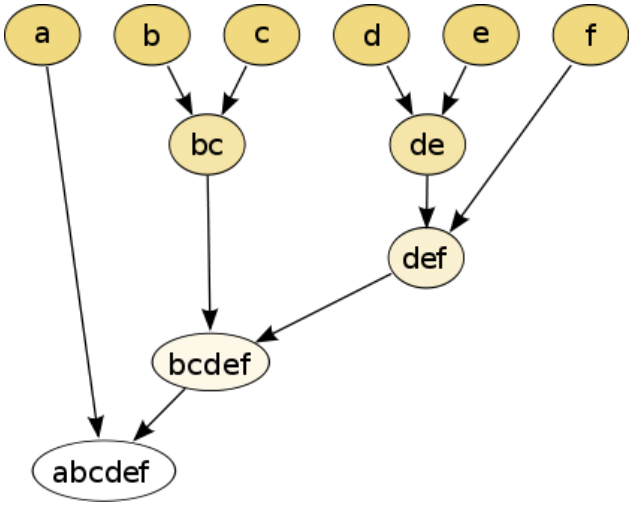


Figure 17.2: The hierarchical clustering dendrogram

17.2.3 tSNE

tSNE analysis was introduced in chapter 9.3.

17.3 Clustering methods

Unsupervised clustering is useful in many different applications and it has been widely studied in machine learning. Some of the most popular approaches are discussed below.

17.3.1 Hierarchical clustering

In hierarchical clustering, one can use either a bottom-up or a top-down approach. In the former case, each cell is initially assigned to its own cluster and pairs of clusters are subsequently merged to create a hiearchy:

With a top-down strategy, one instead starts with all observations in one cluster and then recursively split each cluster to form a hierarchy. One of the advantages of this strategy is that the method is deterministic.

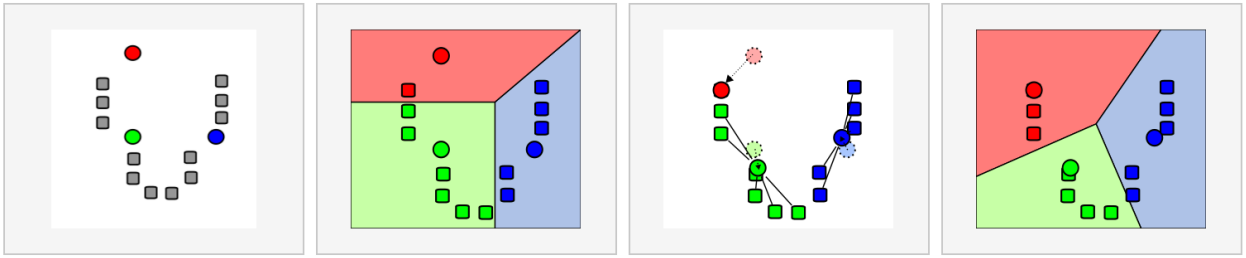


Figure 17.3: Schematic representation of the k-means clustering

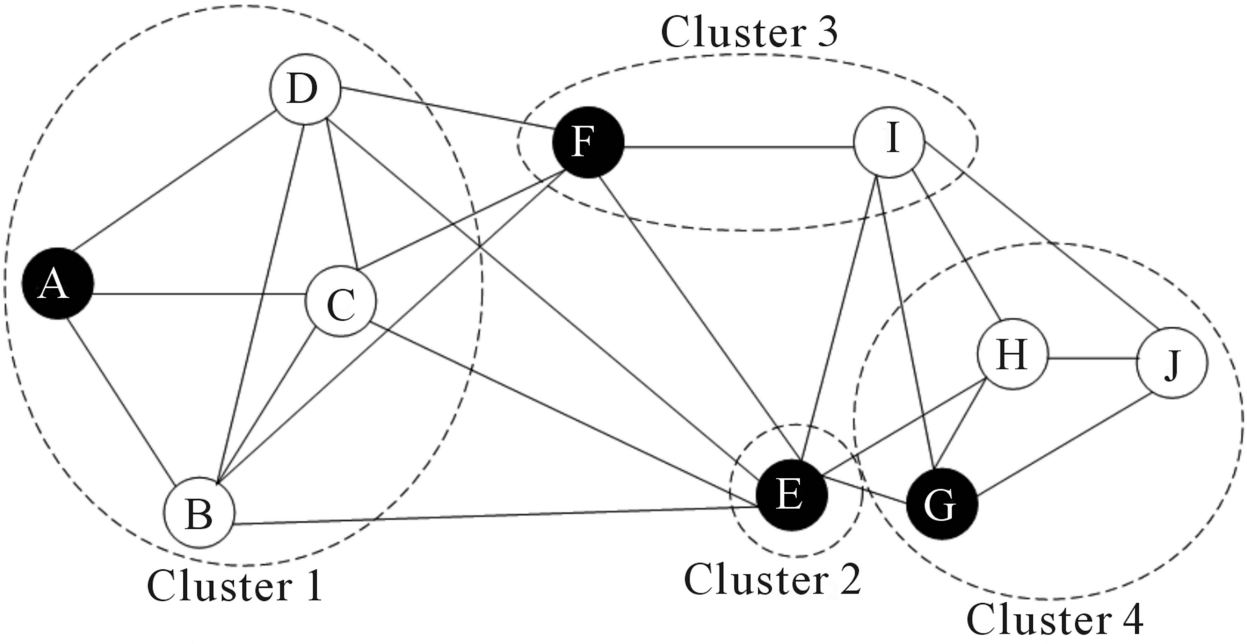


Figure 17.4: Schematic representation of the graph network

17.3.2 k-means

In k-means clustering, the goal is to partition N cells into k different clusters. In an iterative manner, cluster centers are assigned and each cell is assigned to its nearest cluster:

Most methods for scRNA-seq analysis includes a k-means step at some point.

17.3.3 Graph-based methods

Over the last two decades there has been a lot of interest in analyzing networks in various domains. One goal is to identify groups or modules of nodes in a network.

Some of these methods can be applied to scRNA-seq data and one example is the method, which is based on the concept of identifying groups of tightly connected nodes.

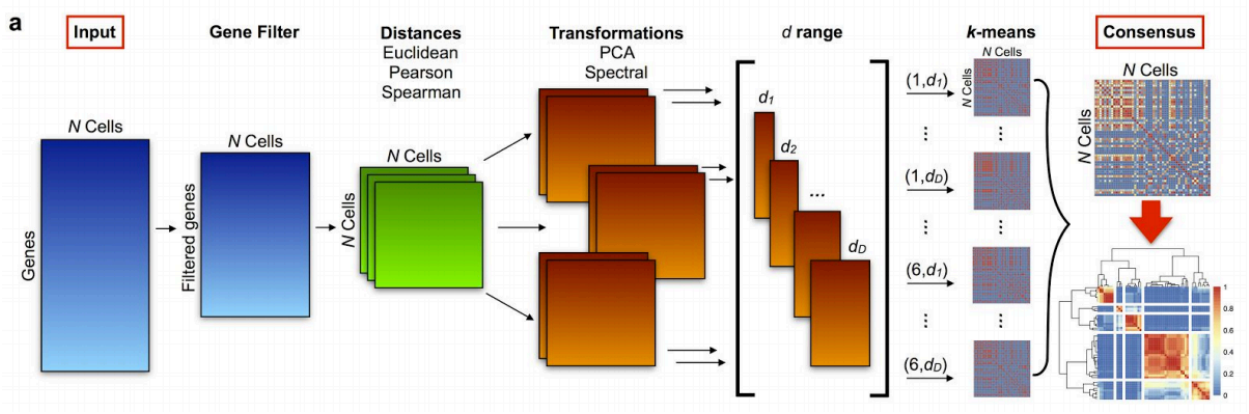


Figure 17.5: SC3 pipeline

17.4 Challenges in clustering

- What is the number of clusters k?
- **Scalability**: in the last 2 years the number of cells in scRNA-seq experiments has grown by 2 orders of magnitude from ~10^2 to ~10^4
- Tools are not user-friendly

17.5 Tools for scRNA-seq data

17.5.1 SINCERA

- SINCERA (Guo et al., 2015) is based on hierarchical clustering
- Data is converted to z-scores before clustering
- Identify k by finding the first singleton cluster in the hierarchy

17.5.2 pcaReduce

pcaReduce (Žurauskienė and Yau, 2016) combines PCA, k-means and "iterative" hierarchical clustering. Starting from a large number of clusters pcaReduce iteratively merges similar clusters; after each merging event it removes the principle component explaning the least variance in the data.

17.5.3 SC3

- SC3 (Kiselev et al., 2016) is based on PCA and spectral dimensionality reductions
- Utilises k-means
- Additionally performs the consensus clustering

17.5.4 tSNE + k-means

- Based on tSNE maps
- Utilises k-means

17.5.5 SEURAT

SEURAT (Macosko et al., 2015) first utilises PCA on a set of cells, then a number of statistically significant PCs is defined. Those PCs are further projected to a 2D space using tSNE. The remaining cells are then projected on the same tSNE map. Density clustering algorithm (DBSCAN) is then used to identify cell clusters in the 2D space.

Note In the newest versions of SEURAT (v. 1.3-1.4) the tSNE is now used exclusively for visualization, and clustering is based on a *community detection* approach similar to one previously proposed for analyzing CyTOF data (Levine et al., 2015).

17.5.6 SNN-Cliq

SNN-Cliq (Xu and Su, 2015) is a graph-based method. First the method identifies the k-nearest-neighbours of each cell according to the *distance* measure. This is used to calculate the number of Shared Nearest Neighbours (SNN) between each pair of cells. A graph is built by placing an edge between two cells If they have at least one SNN. Clusters are defined as groups of cells with many edges between them using a "clique" method. SNN-Cliq requires several parameters to be defined manually.

Chapter 18

Clustering example

```
library(scRNA.seq.funcs)
library(pcaMethods)
library(pcaReduce)
library(Rtsne)
library(SC3)
library(scater)
library(pheatmap)
library(ggplot2)
set.seed(1234567)
```

To illustrate clustering of scRNA-seq data, we consider a dataset of hematopoietic stem cells (HSCs) collected from a patient with myeloproliferative neoplasm (MPN). It is well known that this disease is highly heterogeneous with multiple sub-clones co-existing within the same patient.

18.1 Patient dataset

Traditionally, clonal heterogeneity has been assessed by genotyping sub-clones. Genotyping through Sanger sequencing of two key loci has been carried out for this patient and it has revealed the presence of 3 different sub-clones (WT, WT/Tet2 and Jak2/Tet2). Our goal is to identify clusters corresponding to the three genotypes from the scRNA-seq data.

```
patient.data <- read.table("clustering/patient.txt", sep = "\t")
```

For convenience we have performed all quality control and normalization steps (SF + RUVg) in advance. The dataset contains 51 cells and 8710 genes.

Now we are ready to cluster the data using the methods described in the previous chapter.

18.2 SC3

`SC3` is a purely clustering tool and it does not provide functions for the sequencing quality control (QC) or normalisation. On the contrary it is expected that these preprocessing steps are performed by a user in advance. To encourage the preprocessing, SC3 is built on top of the scater package (see previous chapters). To our knowledge the scater is the most comprehensive toolkit for the QC and normalisation analysis of the single-cell RNA-Seq data.

If you already have an object of `scater` class `SCESet` then you should proceed to the next code chunk. If you have a matrix containing cell/gene expression values then you can create a `scater` object using your matrix:

```
patient.sceset <- newSCESet(countData = patient.data)
patient.sceset <- calculateQCMetrics(patient.sceset)
```

Now we are ready ro run the `SC3` clustering:

```
patient.sceset <- sc3(patient.sceset, ks = 2:5, k_estimator = TRUE, biology = TRUE)
```

When the clustering is done you will be able to visualise the clustering results in many different ways. See the SC3 vignette for more details.

`SC3` also has a useful function for summarising/visualising the results in the interactive `Shiny` session:

```
sc3_interactive(patient.sceset)
```

This command will open `SC3` in a web browser. Once it is opened please perform the following exercises:

- **Exercise 1**: Explore different clustering solutions for k from 2 to 5. Also try to change the consensus averaging by checking and unchecking distance and transformation check boxes in the left panel of **SC3**.
- **Exercise 2**: Based on the genotyping and `SC3` prediction, we strongly believe that $k = 3$ provides the best clustering. What evidence do you find for this in the clustering? How can you use the silhouette plots to motivate choosing the value of k that you think looks best?
- **Exercise 3**: Which clusters are the most stable? You can find out how different cells migrate between clusters using the "Cell Labels"/"Stability" tabs panel.
- **Exercise 4**: Check out differentially expressed genes and marker genes for the obtained clusterings. Please use $k = 3$.
- **Exercise 5**: Change the marker genes threshold (the default is 0.85). Does **SC3** find more marker genes?

18.3 pcaReduce

`pcaReduce` does not provide QC or normalisation function neither. It operates directly on the expression matrix. It is recommended to use a gene filter and log transformation before running `pcaReduce`. We will used the default `SC3` gene filter

```
# use the same gene filter as in SC3
input <- patient.data[fData(patient.sceset)$sc3_gene_filter, ]
# log transform before the analysis
input.log <- log2(input + 1)
```

There are several parameters used by `pcaReduce`: * `nbt` defines a number of `pcaReduce` runs (it is stochastic and may have different solutions after different runs) * `q` defines number of dimensions to start clustering with. The output will contain partitions for all k from 2 to q+1. * `method` defines a method used for clustering. `S` - to perform sampling based merging, `M` - to perform merging based on largest probability.

We will run `pcaReduce` 10 times:

```
# run pcaReduce 10 times creating hierarchies from 1 to 30 clusters
pca.red <- PCAreduce(t(input.log), nbt = 10, q = 30, method = 'S')
```

Let's compare the clusterings (with $k = 3$) provided after running `pcaReduce` 10 times:

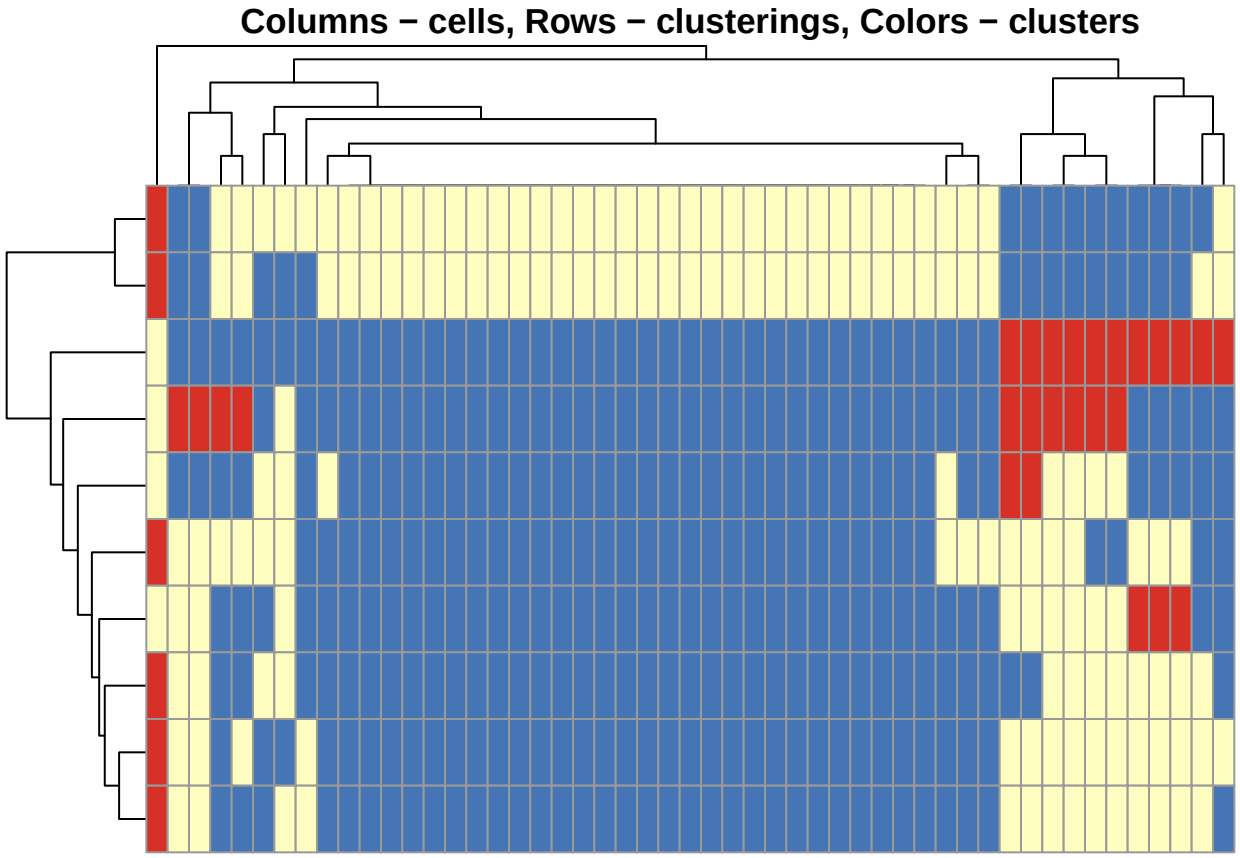


Figure 18.1: Clustering solutions of pcaReduce method after running it for 10 times and selecting $k = 3$

```
res <- unique(scRNA.seq.funcs::merge_pcaReduce_results(pca.red, 3))
pheatmap(res, legend = FALSE, show_rownames = FALSE, main = "Columns - cells, Rows - clusterings, Colors
```

Exercise 6: Run pcaReduce for $k = 2$, $k = 4$ and $k = 5$. Is it easy to choose the optimal k?

Hint: When running pcaReduce for different ks you do not need to rerun pcaReduce::PCAreduce function, just use already calculated `pca.red` object.

Our solutions:

Exercise 7: Compare the results between `SC3` and `pcaReduce`. What is the main difference between the solutions provided by the two different methods?

18.4 tSNE + kmeans

tSNE plots that we saw before (9.3) when used the **scater** package are made by using the Rtsne and ggplot2 packages. We can create a similar plots explicitly:

```
tsne_out <- Rtsne::Rtsne(t(input.log), perplexity = 10)
df_to_plot <- as.data.frame(tsne_out$Y)
comps <- colnames(df_to_plot)[1:2]
ggplot2::ggplot(df_to_plot, aes_string(x = comps[1], y = comps[2])) +
    geom_point() +
    xlab("Dimension 1") +
```

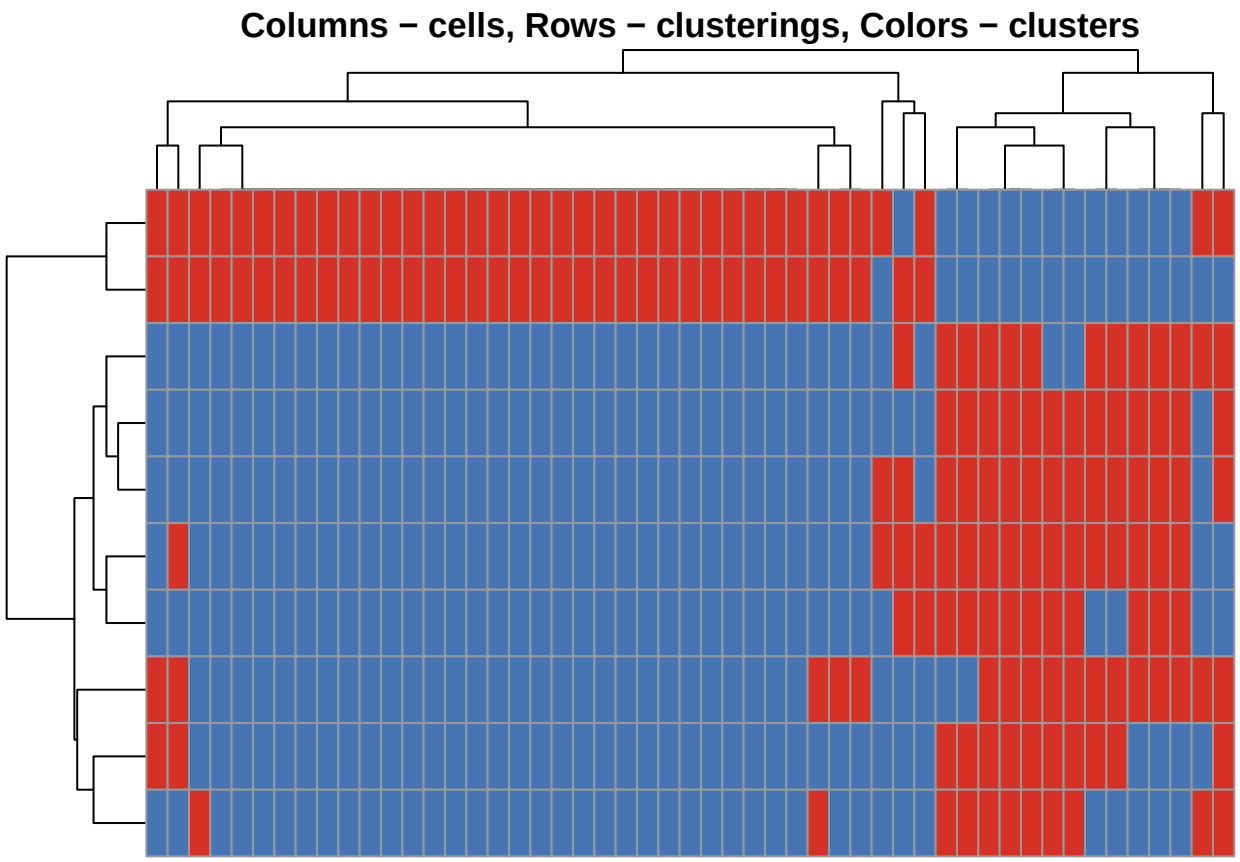


Figure 18.2: Clustering solutions of pcaReduce method after running it for 10 times and selecting $k = 2$.

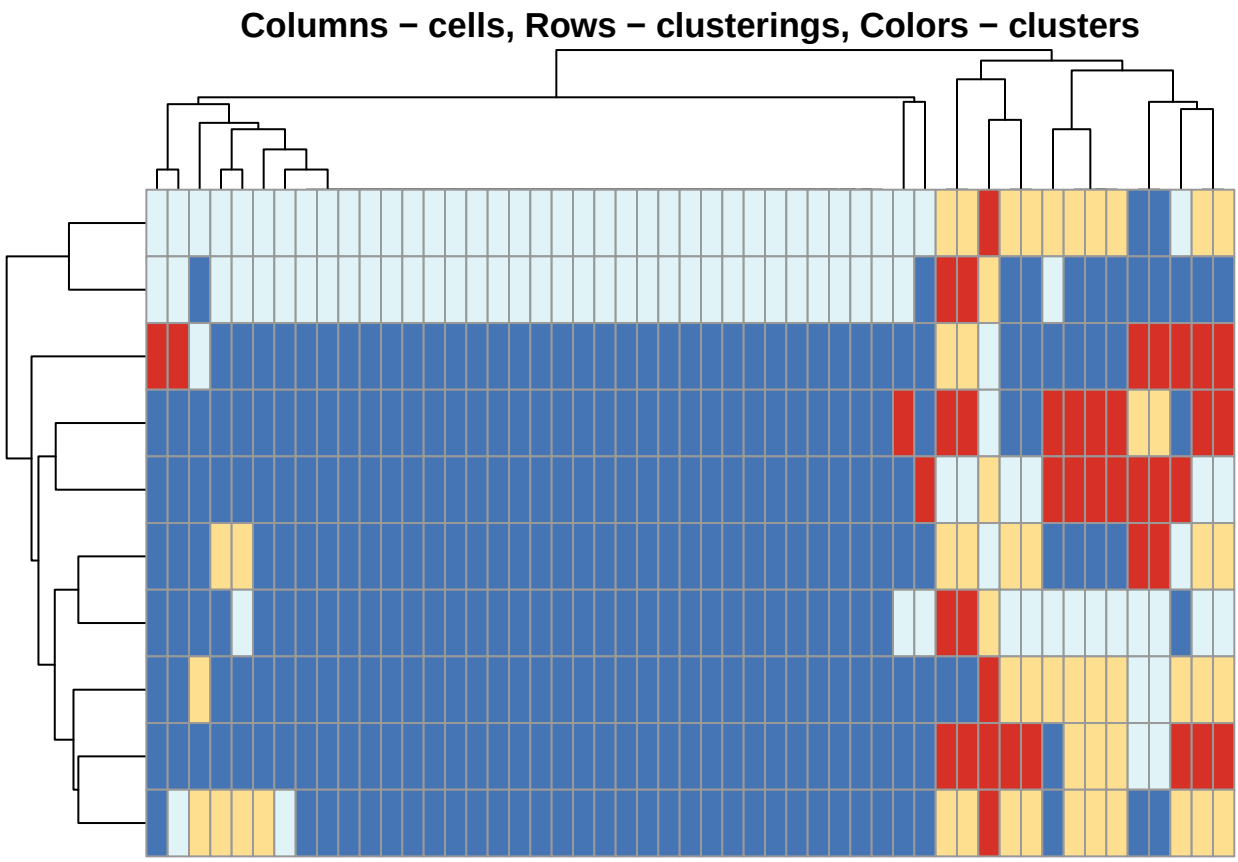


Figure 18.3: Clustering solutions of pcaReduce method after running it for 10 times and selecting $k = 4$.

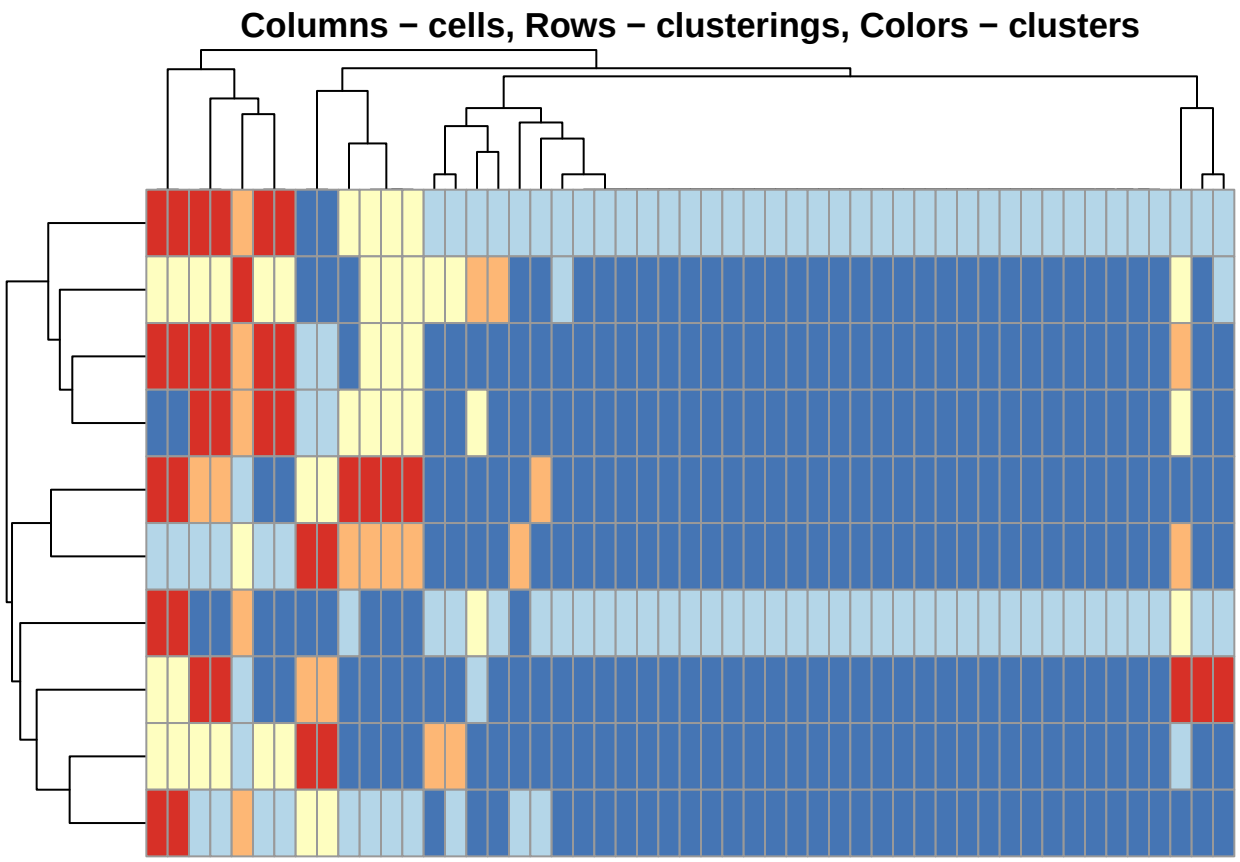


Figure 18.4: Clustering solutions of pcaReduce method after running it for 10 times and selecting $k = 5$.

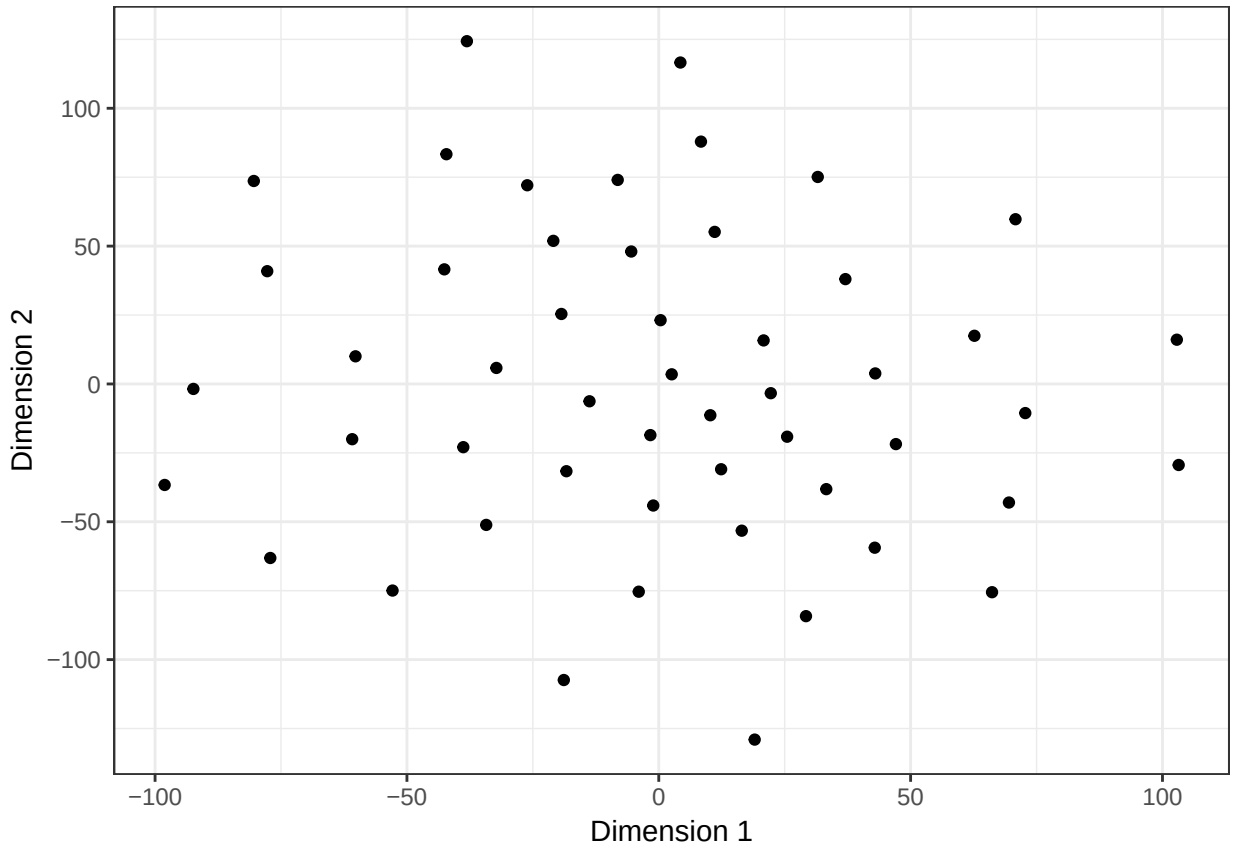


Figure 18.5: tSNE map of the patient data

```
    ylab("Dimension 2") +
    theme_bw()
```

Note that all points on the plot above are black. This is different from what we saw before, when the cells were coloured based on the annotation. Here we do not have any annotation and all cells come from the same batch, therefore all dots are black.

Now we are going to apply k-means clustering algorithm to the cloud of points on the tSNE map. Do you see 2/3/4/5 groups in the cloud?

We will start with $k = 2$:

```
clusts <- kmeans(
    tsne_out$Y,
    centers = 2,
    iter.max = 1e+09,
    nstart = 1000)$clust
df_to_plot$clusts <- factor(clusts, levels = unique(clusts))
comps <- colnames(df_to_plot)[1:3]
ggplot2::ggplot(df_to_plot,
                aes_string(x = comps[1], y = comps[2], color = comps[3])) +
    geom_point() +
    xlab("Dimension 1") +
    ylab("Dimension 2") +
    theme_bw()
```

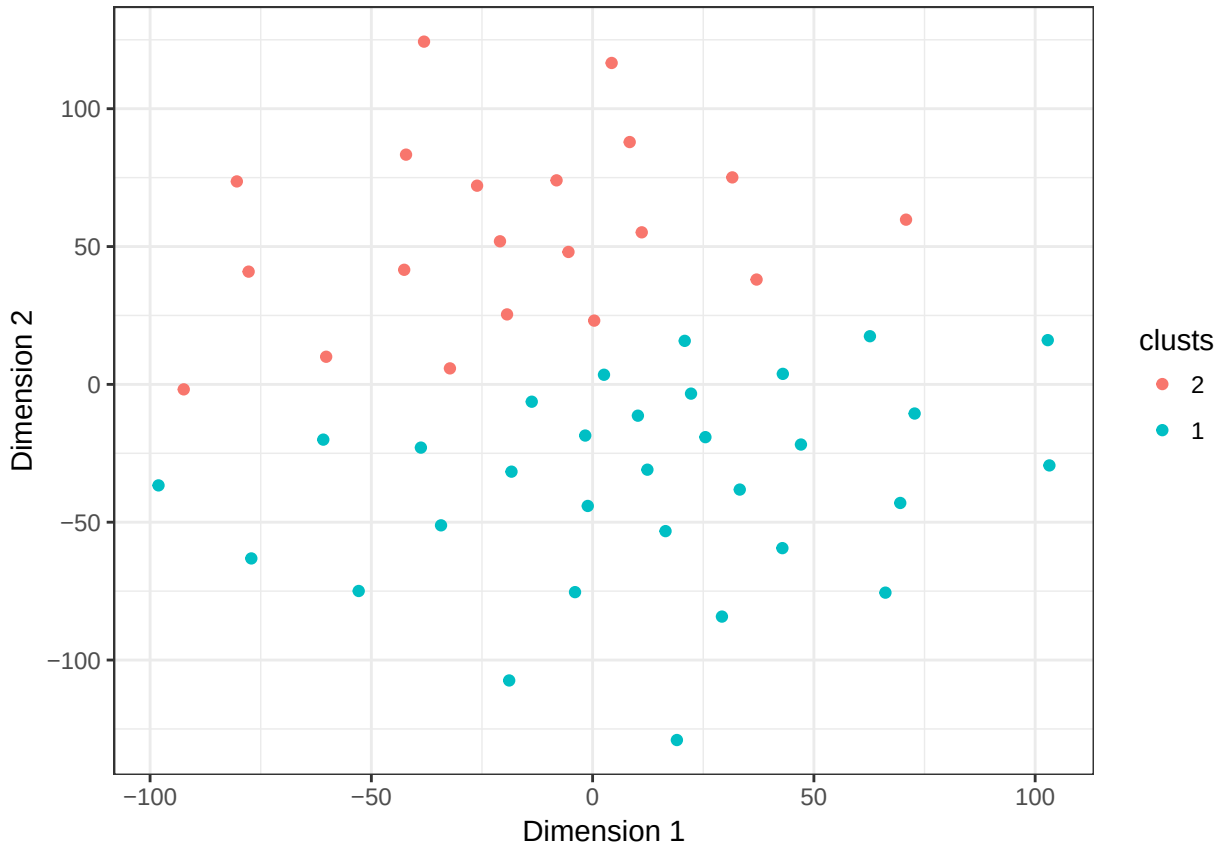


Figure 18.6: tSNE map of the patient data with 2 colored clusters, identified by the k-means clustering algorithm

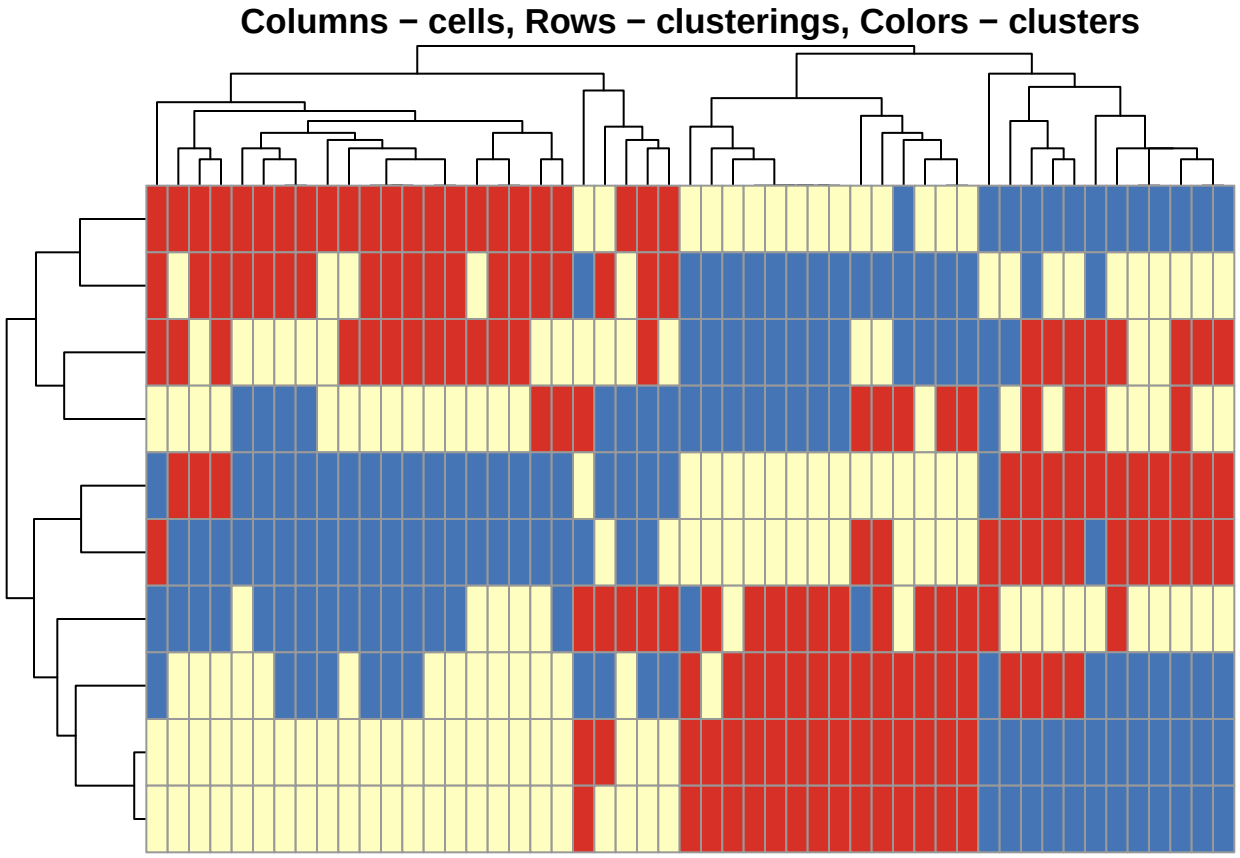


Figure 18.7: Clustering solutions of tSNE+kmeans method after running it for 10 times and using $k = 3$

Exercise 7: Make the same plots for $k = 3$, $k = 4$ and $k = 5$.

Exercise 8: Compare the results between `SC3` and `tSNE+kmeans`. Can the results be improved by changing the perplexity parameter?

As you may have noticed, both `pcaReduce` and `tSNE+kmeans` are stochastic and give different results every time they are run. To get a better overview of the solutions, we need to run the methods multiple times. Here we run **tSNE+kmeans** clustering 10 times with $k = 3$ (with perplexity = 10):

```
tsne.res <- scRNA.seq.funcs::tsne_mult(input.log, 3, 10)
res <- unique(do.call(rbind, tsne.res))
```

Exercise 9: Visualize the different clustering solutions using a heatmap. Then run tSNE+kmeans algorithm with $k = 2$ or $k = 4$ and see how the clustering looks like in these cases.

18.5 SNN-Cliq

Here we run SNN-cliq with te default parameters provided in the author's example:

```
distan <- "euclidean"
par.k <- 3
par.r <- 0.7
par.m <- 0.5
# construct a graph
```

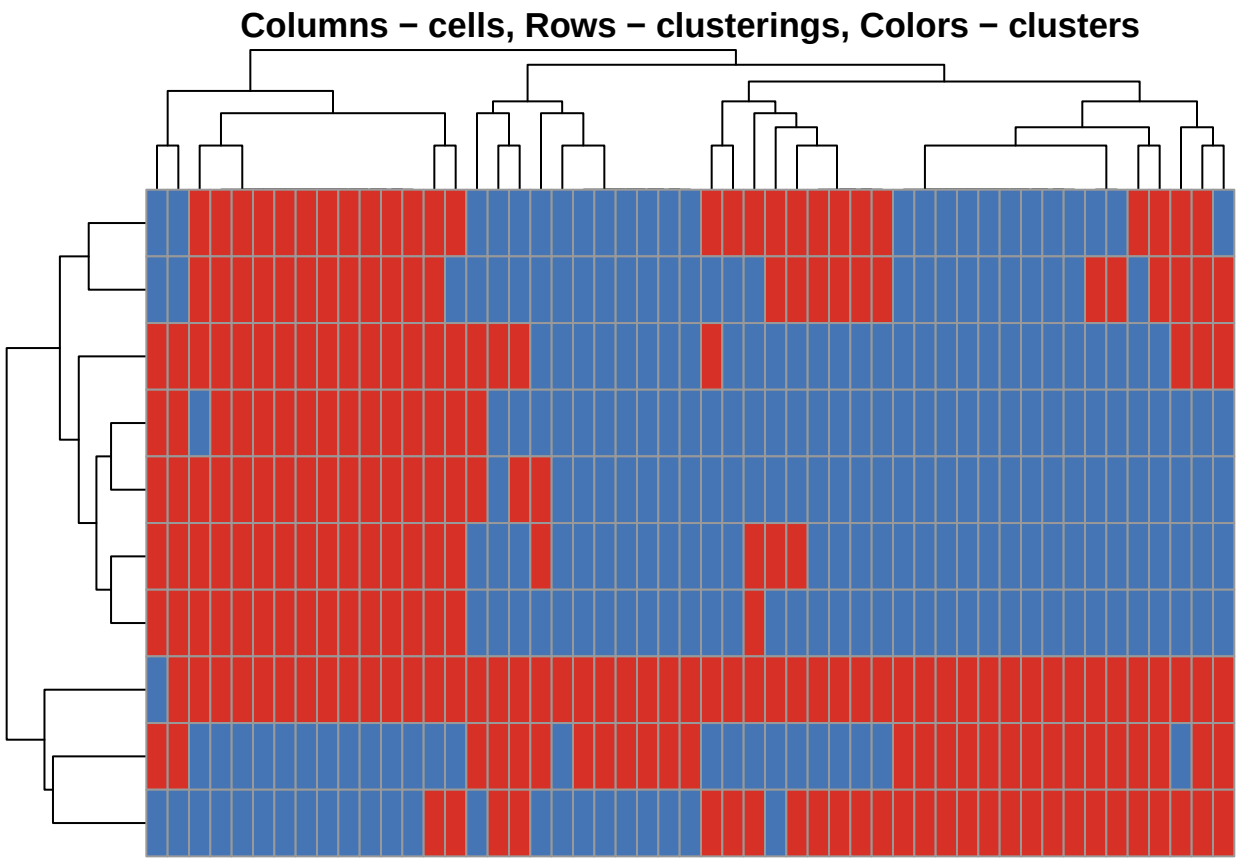


Figure 18.8: Clustering solutions of tSNE+kmeans method after running it for 10 times and using $k = 2$

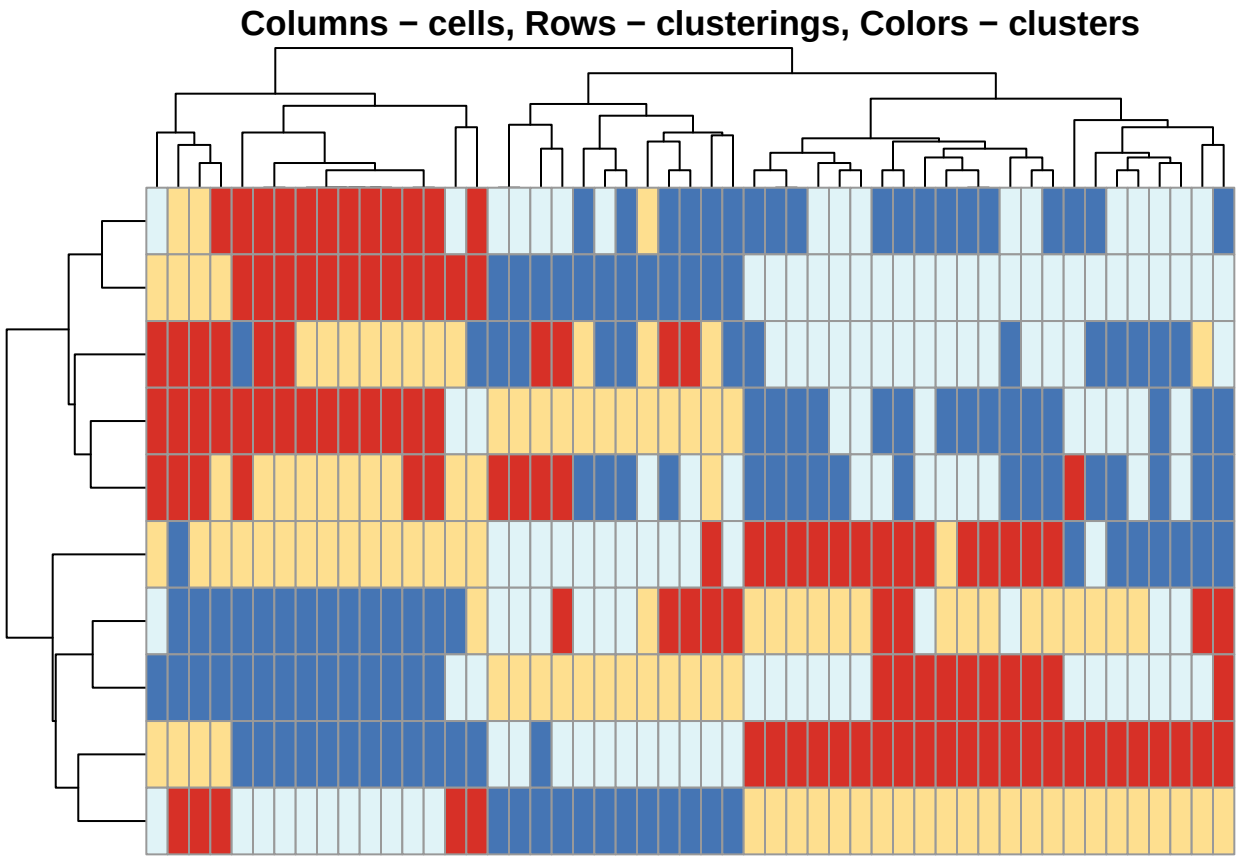


Figure 18.9: Clustering solutions of tSNE+kmeans method after running it for 10 times and using $k = 4$

```
scRNA.seq.funcs::SNN(
    data = t(input.log),
    outfile = "snn-cliq.txt",
    k = par.k,
    distance = distan
)
# find clusters in the graph
snn.res <-
    system(
        paste0(
            "python snn-cliq/Cliq.py ",
            "-i snn-cliq.txt ",
            "-o res-snn-cliq.txt ",
            "-r ", par.r,
            " -m ", par.m
        ),
        intern = TRUE
    )
cat(paste(snn.res, collapse = "\n"))
```

```
## input file snn-cliq.txt
## find 7 quasi-cliques
## merged into 2 clusters
## unique assign done
```

```
snn.res <- read.table("res-snn-cliq.txt")
# remove files that were created during the analysis
system("rm snn-cliq.txt res-snn-cliq.txt")
```

Exercise 10: How can you characterize the solution identified by SNN-Cliq? Run SNN-Cliq algorithm with different values of k, r, m and *distance*, and see how the clustering looks like in these cases.

18.6 SINCERA

As mentioned in the previous chapter SINCERA is based on hierarchical clustering. One important thing to keep in mind is that it performs a gene-level z-score transformation before doing clustering:

```
# perform gene-by-gene per-sample z-score transformation
dat <- apply(input, 1, function(y) scRNA.seq.funcs::z.transform.helper(y))
# hierarchical clustering
dd <- as.dist((1 - cor(t(dat), method = "pearson"))/2)
hc <- hclust(dd, method = "average")
```

If the number of cluster is not known SINCERA can identify **k** as the minimum height of the hierarchical tree that generates no more than a specified number of singleton clusters (clusters containing only 1 cell)

```
num.singleton <- 0
kk <- 1
for (i in 2:dim(dat)[2]) {
    clusters <- cutree(hc, k = i)
    clustersizes <- as.data.frame(table(clusters))
    singleton.clusters <- which(clustersizes$Freq < 2)
    if (length(singleton.clusters) <= num.singleton) {
        kk <- i
```

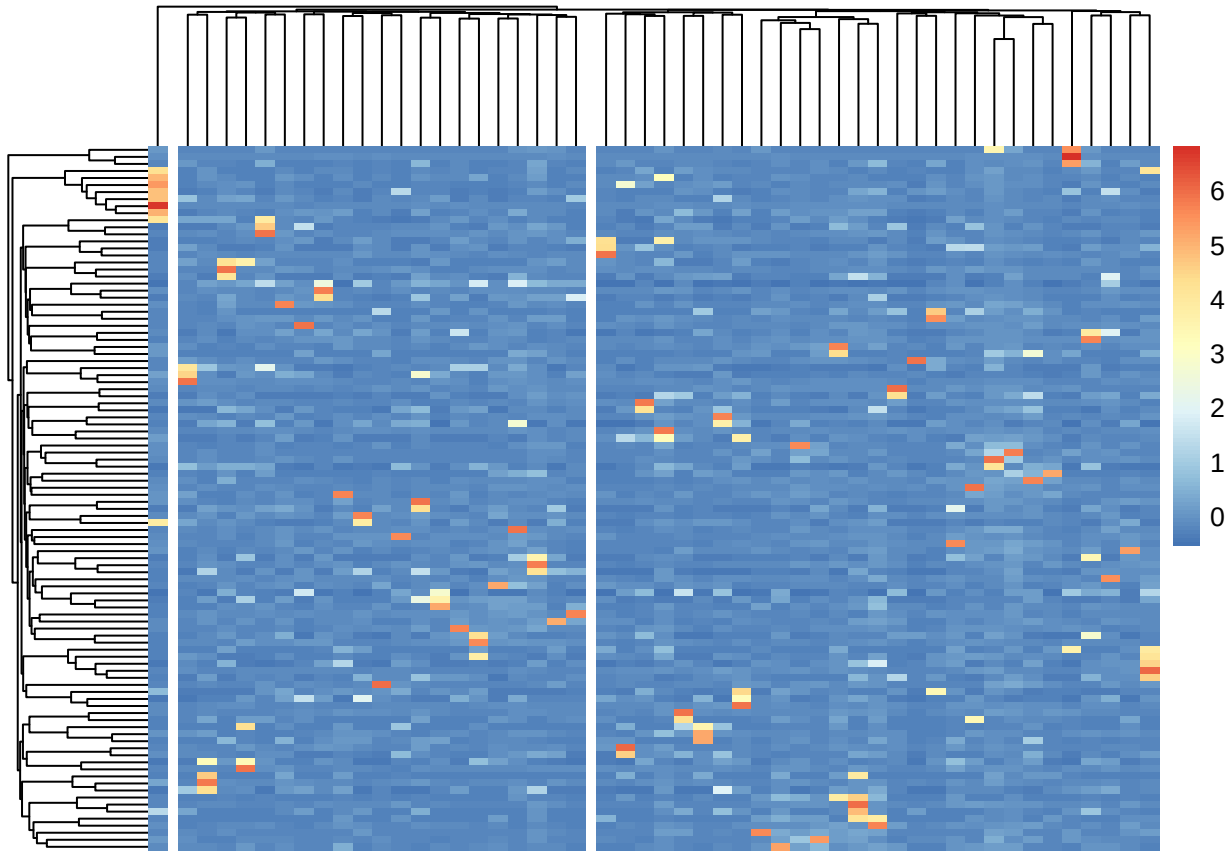


Figure 18.10: Clustering solutions of SINCERA method using $k = 3$

```
    } else {
        break;
    }
}
cat(kk)
```

```
## 1
```

Exercise 11: Visualize the SINCERA results as a heatmap. How do the results compare to the other methods? What happens if you choose $k = 3$?

Our answer:

Exercise 12: Is using the singleton cluster criteria for finding **k** a good idea?

18.7 SEURAT

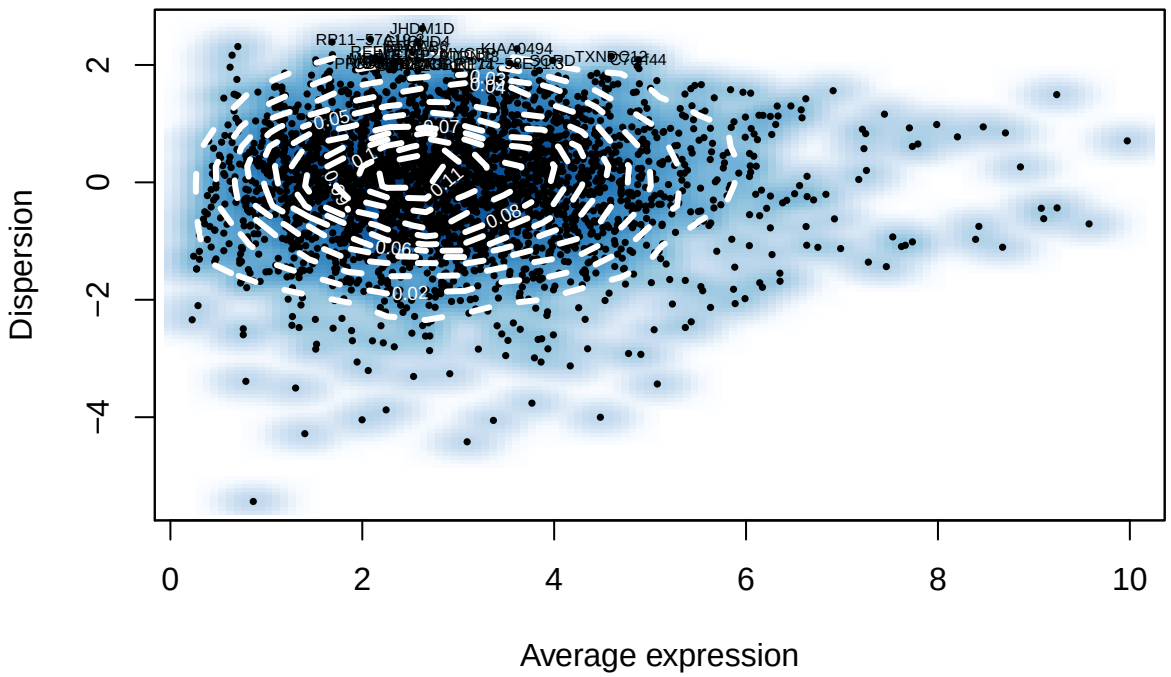
Here we follow an example created by the authors of SEURAT. We had to introduce some modifications due to the errors produced by the original code:

Note In the newest versions of SEURAT (v. 1.3-1.4) the tSNE is now used exclusively for visualization, and clustering is based on a *community detection* approach similar to one previously proposed for analyzing CyTOF data (Levine et al., 2015). In this tutorial we are still using an old version (v. 1.2) of `SEURAT` (this will be updated by the next time this course is run - approximately Spring 2017):

```
library(Seurat)
# Create a SEURAT object
d <- new("seurat", raw.data = as.data.frame(log(patient.data + 1)))

# Setup a SEURAT object
d <- setup(
    d,
    project = "NBT",
    min.cells = 3,
    names.field = 2,
    names.delim = "_",
    min.genes = 10,
    is.expr = 1
)

# Genes placed into 20 bins based on X-axis (average expression).
# Y-axis is within-bin z-score of log(Variance/mean).
d <- mean.var.plot(
    d,
    y.cutoff = 2,
    x.low.cutoff = 2,
    fxn.x = expMean,
    fxn.y = logVarDivMean
)
```




```
# Run a PCA using the variable genes as input
d <- pca(d, do.print = FALSE, pcs.store = 25)

# Do 200 random samplings to find significant genes,
# each time randomly permute 1% of genes.
# This returns a 'p-value' for each gene in each PC,
# based on how likely the gene/PC score would have been observed by chance
```

```
d <- jackStraw(
    d,
    num.replicate = 200,
    do.print = FALSE,
    num.pc = 25
)

# Compare the distribution of P-values for each PC with a uniform distribution.
# 'Significant' PCs will have a strong enrichment of genes with low p-values
pAll <- d@jackStraw.empP
pAll$Contig <- rownames(pAll)
pAll.l <- melt(pAll, id.vars = "Contig")
colnames(pAll.l) <- c("Contig", "PC", "Value")

score.df <- NULL
score.thresh=1e-5
for (i in unique(pAll.l$PC)){
    q <- qqplot(pAll[, i], runif(1000), plot.it = FALSE)
    pc.score <- prop.test(
        c(
            length(which(pAll[, i] <= score.thresh)),
            floor(nrow(pAll) * score.thresh)
        ),
        c(nrow(pAll), nrow(pAll))
    )$p.val
    if (length(which(pAll[, i] <= score.thresh)) == 0) pc.score <- 1

    if(is.null(score.df))
      score.df <- data.frame(PC = i, Score = pc.score)
    else
      score.df <- rbind(score.df, data.frame(PC = i, Score = pc.score))
}

# There are no significant PCs:
head(score.df)
```

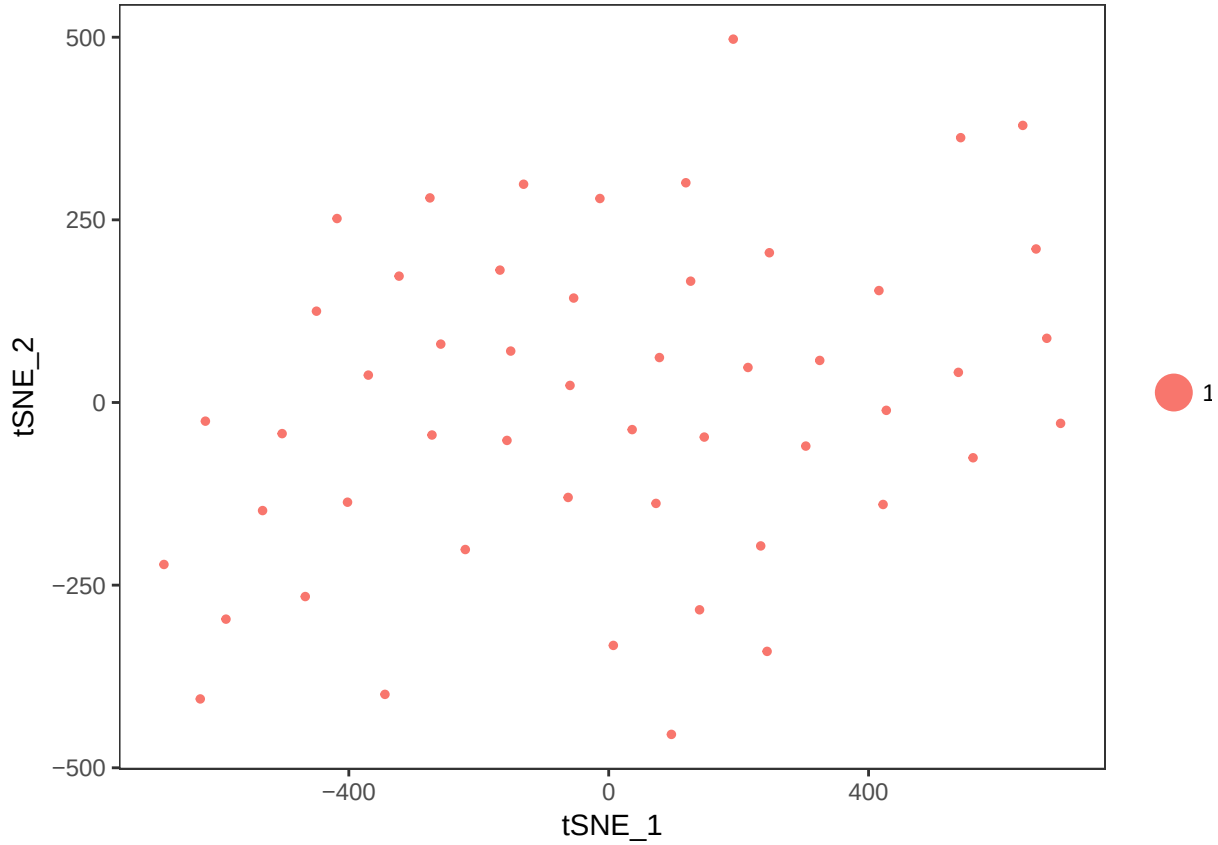
```
##    PC Score
## 1 PC1     1
## 2 PC2     1
## 3 PC3     1
## 4 PC4     1
## 5 PC5     1
## 6 PC6     1
```

```
ndim <- 2

# Project data on a 2D using tSNE
d <- run_tsne(d, dims.use = 1:ndim, max_iter = 2000)

# Find clusters in the tSNE map
d <- DBclust_dimension(
    d,
    1,
    2,
```

```
    reduction.use = "tsne",
    G.use = 8,
    set.ident = TRUE
)
tsne.plot(d, pt.size = 1)
```



Exercise 13: As you can see DBSCAN could find only one cluster. It is known that DBSCAN cannot cluster data sets well with large differences in densities, since the *G.use* parameter cannot then be chosen appropriately for all clusters. Try to change *G.use* to be able to find more than one cluster in the data.

Note We found that in general SEURAT does not work well for small datasets ($N < 200$ cells). For a comprehensive comparisons of the methods please look at the SC3 paper (Kiselev et al., 2016).

Chapter 19

Identification of important genes

```
library(scRNA.seq.funcs)
library(M3Drop)
library(limma)
set.seed(1)
```

One of the key differences between single-cell RNASeq and bulk RNASeq is the large number of dropouts (zero expression) for genes with even moderately high levels of average expression. These zeros violate assumptions made by many statistical tools used for bulk RNASeq, eg. Negative Binomial expression used by DESeq2, normality assumptions of correlation methods, or assumptions of few tied-ranks for many non-parametric tests. Some recent scRNASeq methods model a different dropout rate for each gene (eg. MAST, BASiCS) while other methods try to fit the relationship between expression level and dropout rate across genes for a specific application (eg. SCDE, ZIFA). We will be using our new package Michaelis-Menten Modelling of Dropouts (M3Drop) which specifically focuses on modelling and gaining biological insights from dropouts.

For this section we will be working with the Usoskin et al data. It contains 4 cell types: NP = non-peptidergic nociceptors, PEP = peptidergic nociceptors, NF = neurofilament containing and TH = tyrosine hydroxylase containing neurons.

```
usoskin1 <- readRDS("usoskin/usoskin1.rds")
dim(usoskin1)
```

```
## [1] 25334   622
```

```
table(colnames(usoskin1))
```

```
##
##  NF  NP PEP  TH
## 139 169  81 233
```

19.1 Fitting the models

First we must normalize & QC the dataset. M3Drop contains a built-in function for this which removes cells with few detected genes, removes undetected genes, and converts raw counts to CPM.

```
uso_list <- M3Drop::M3DropCleanData(
    usoskin1,
    labels = colnames(usoskin1),
    min_detected_genes = 2000,
```

```
    is.counts = TRUE
)
```

Exercise 1: How many cells & genes have been removed by this filtering? Do you agree with the 2000 detected genes threshold?

19.2 The Michaelis-Menten Equation

The Michaelis-Menten (MM) Equation is the standard model of enzyme kinetics. We use it to model dropouts in single-cell RNASeq because most dropouts occur as a result of failing to be reverse-transcribed to sufficient levels. The details are omitted here, but in this model the probability that a transcript will not be found is given by

$$P_{dropout} = K/(K+S)$$

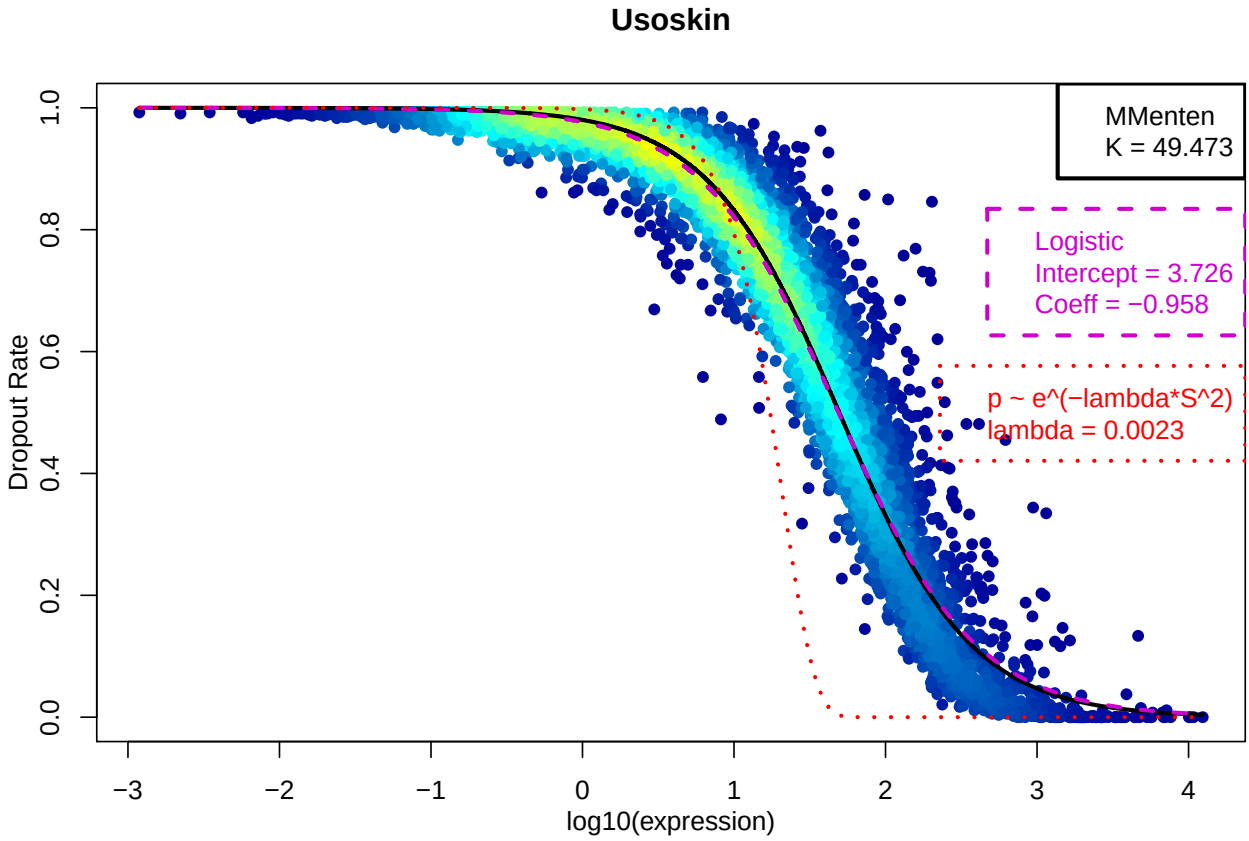
where S is the mRNA concentration in the cell (we will estimate this as average expression) and K is the Michaelis-Menten constant.

Other models that have been proposed are:

- $P = Logistic(\log(S))$ used by SCDE (determining differential expression) and
- $P = e^{-\lambda * S^2}$ used by ZIFA (zero-inflated PCA).

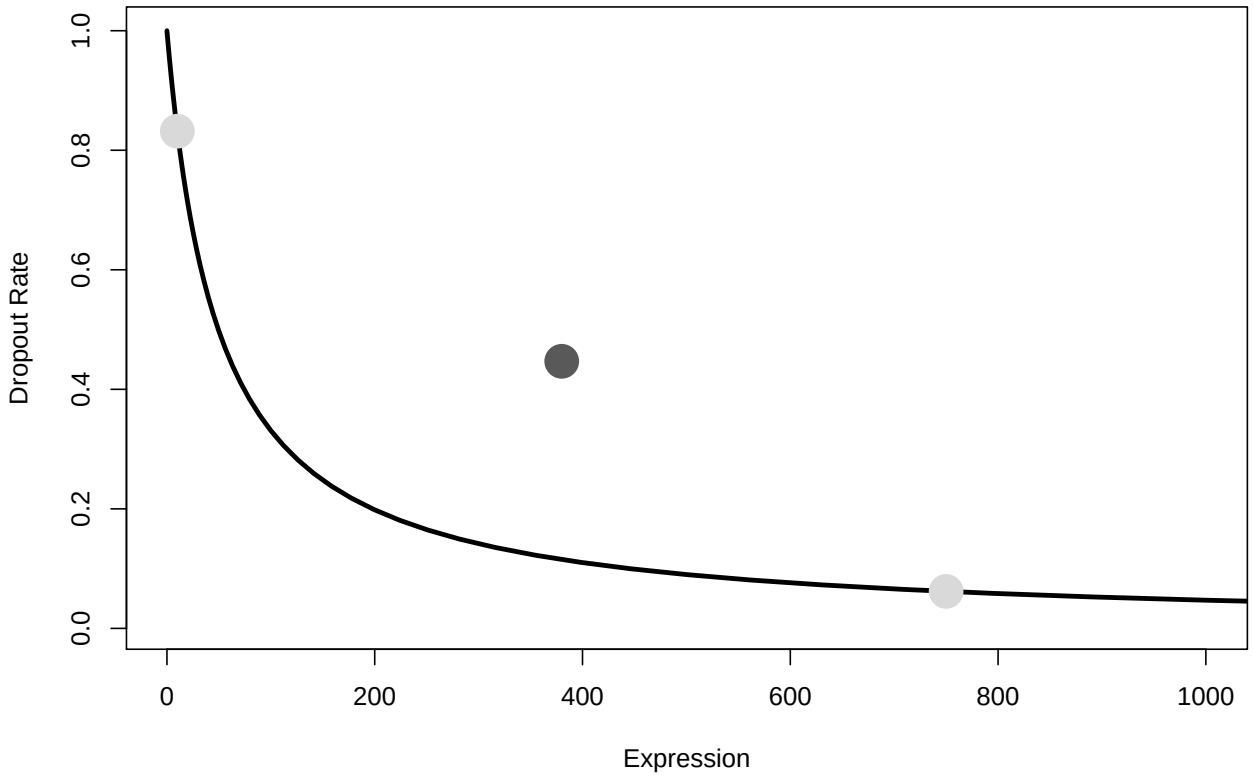
Now we will fit all three models to the normalized Usoskin data:

```
models <- M3Drop::M3DropDropoutModels(uso_list$data)
title(main = "Usoskin")
```



19.3 Right outliers

There are many outliers to the right of the fitted MM curve. Genes which are expressed at different levels in subpopulations of our cells will be shifted to the right of the curve. This happens because the MM curve is a convex function, whereas averaging dropout rate and expression is a linear function.

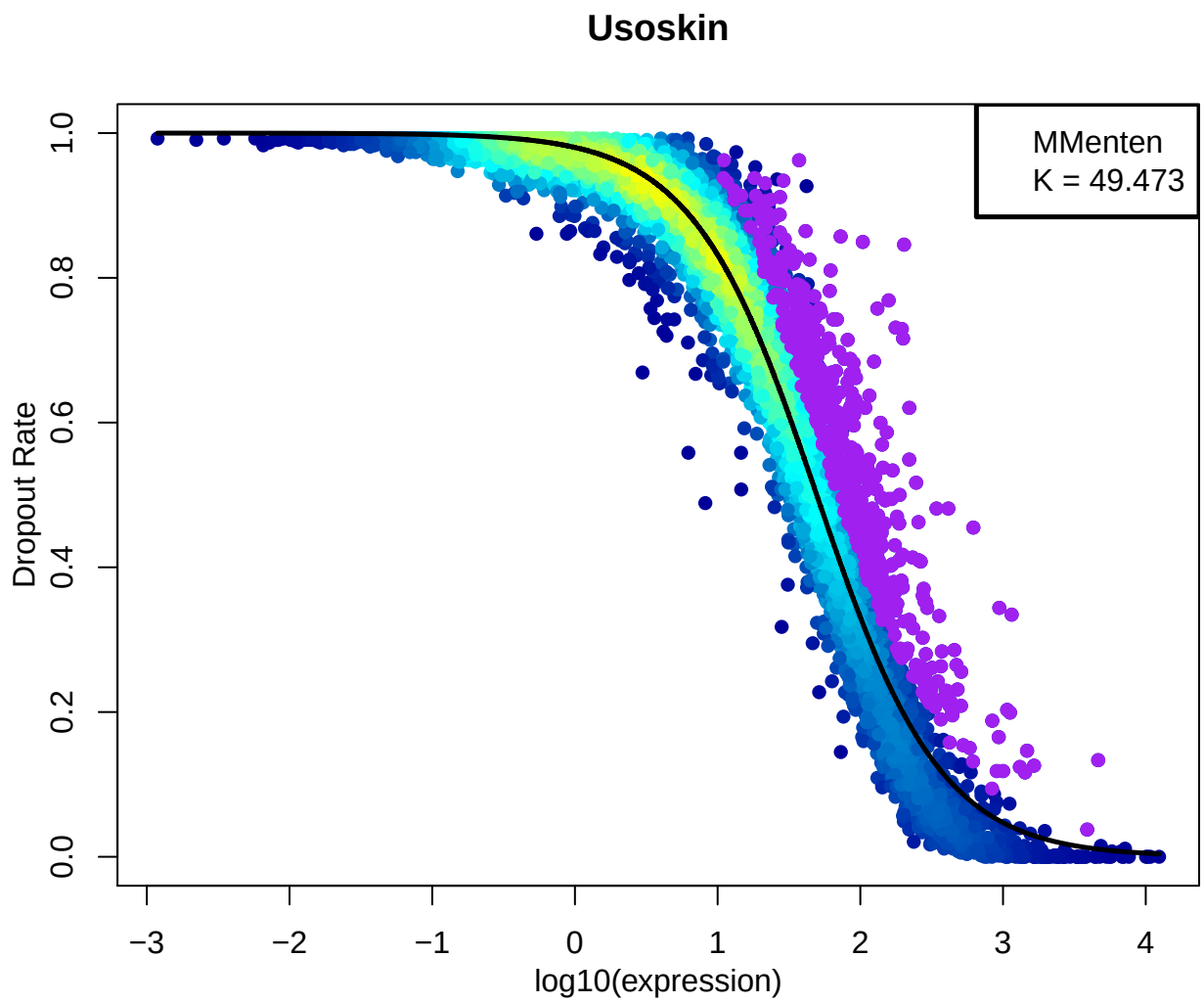


Note: add `log="x"` to the `plot` call above to see how this looks on the log scale, which is used in M3Drop figures.

Exercise 2: Produce the same plot as above with different expression levels (S1 & S2) and/or mixtures (mix).

We use M3Drop to identify significant outliers to the right of the MM curve. We also apply 1% FDR multiple testing correction:

```
DE_genes <- M3Drop::M3DropDifferentialExpression(
    uso_list$data,
    mt_method = "fdr",
    mt_threshold = 0.01
)
title(main = "Usoskin")
```



Check which of the known neuron markers are identified as DE:

```
uso_markers <-
    c("Nefh", "Tac1", "Mrgprd", "Th", "Vim", "B2m",
      "Col6a2", "Ntrk1", "Calca", "P2rx3", "Pvalb")
rbind(uso_markers, uso_markers %in% DE_genes$Gene)
```

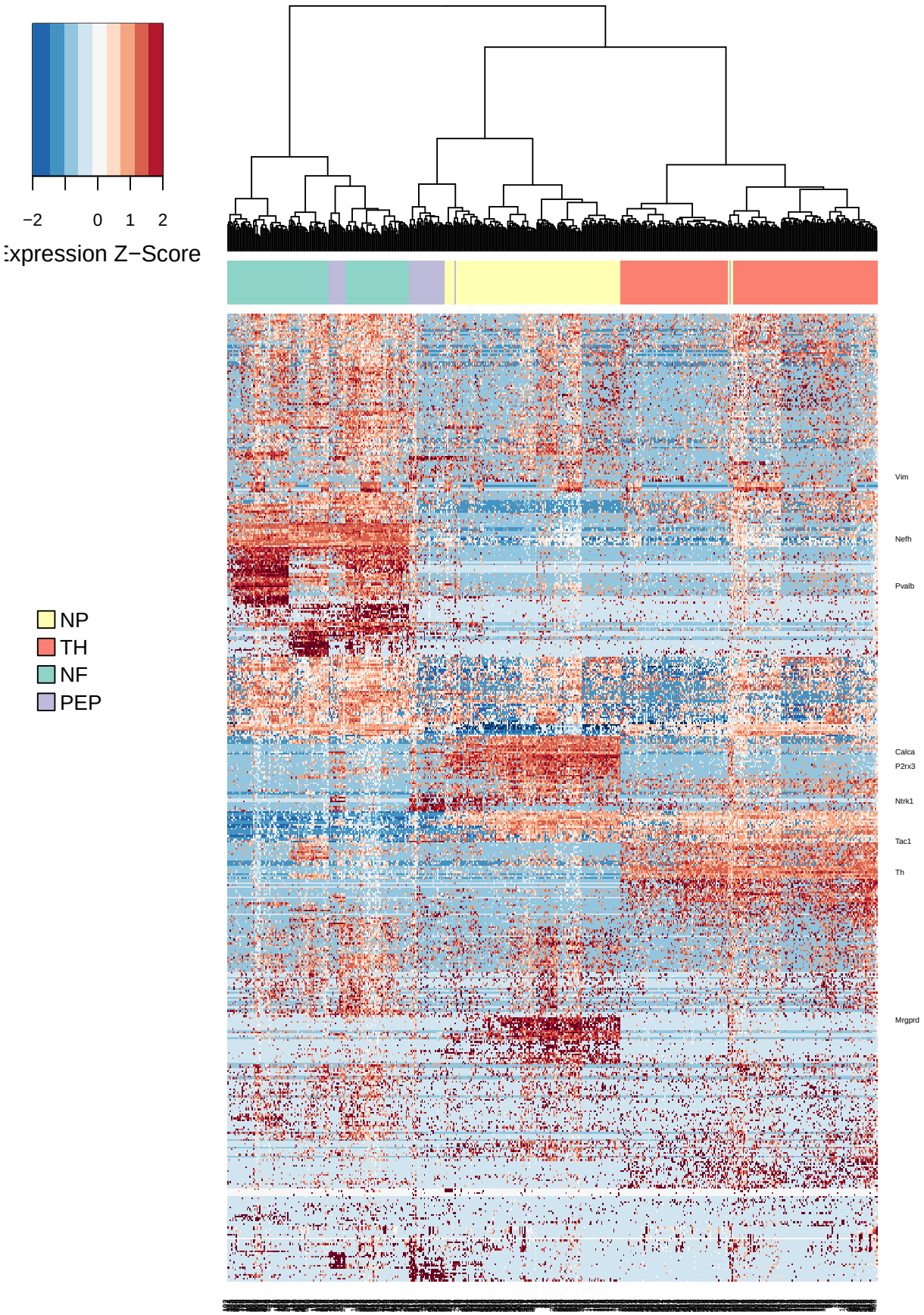
```
##             [,1]   [,2]   [,3]     [,4]   [,5]   [,6]    [,7]      [,8]
## uso_markers "Nefh" "Tac1" "Mrgprd" "Th"   "Vim"  "B2m"   "Col6a2" "Ntrk1"
##             "TRUE" "TRUE" "TRUE"   "TRUE" "TRUE" "FALSE" "FALSE"  "TRUE"
##             [,9]    [,10]   [,11]
## uso_markers "Calca" "P2rx3" "Pvalb"
##             "TRUE"  "TRUE"  "TRUE"
```

19.4 Validation of DE results

We can also plot the expression levels of these genes to check they really are DE genes.

```
M3Drop::M3DropExpressionHeatmap(
    DE_genes$Gene,
    uso_list$data,
```

```
    cell_labels = uso_list$labels,
    key_genes = uso_markers
)
```


−2
0
1
2
Expression Z−Score
NP
TH
NF
PEP
Vim
Nefh
Pvalb
Calca
P2rx3
Ntrk1
Tac1
Th
Mrgprd

19.5 Comparing M3Drop to other methods

We can compare the genes identified as DE using M3Drop to those identified using other methods. Running differential expression methods which compare two groups at a time is slow for this dataset (6 possible pairs of groups x 15,708 genes) thus we have provided you with the output for DESeq. Load it using:

```
DESeq_table <- readRDS("usoskin/DESeq_table.rds")
length(unique(DESeq_table$Gene))
```

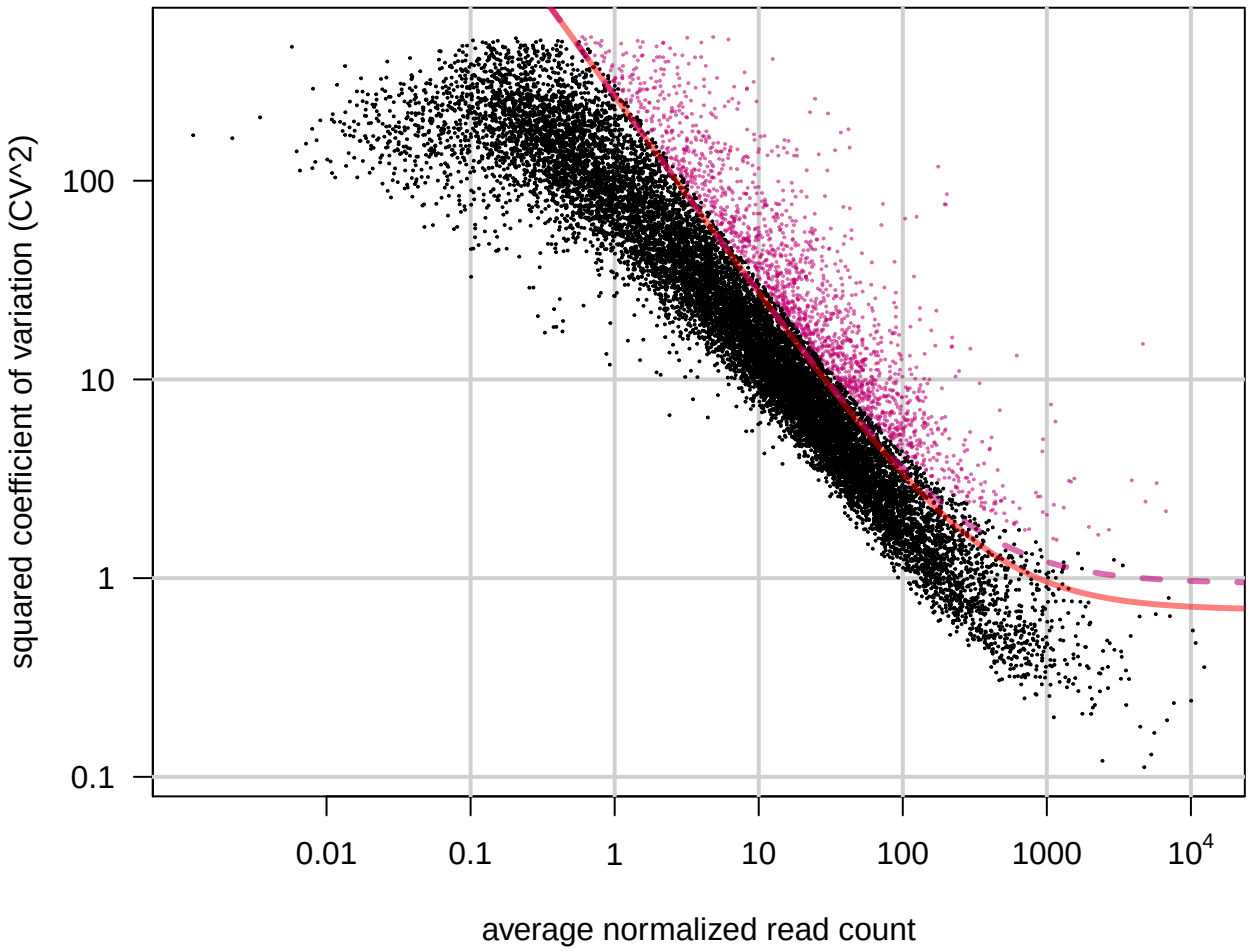
```
## [1] 2604
```

We will demonstrate some of the methods starting from the simplest one proposed by Brennecke et al., which identifies genes with significant variation above technical noise (ERCCs).

To use the method, we first normalize for library size then calculate the mean and the square coefficient of variation (variation divided by the squared mean expression). A quadratic curve is fit to the relationship between these two variables for the ERCC spike-in (subject to just technical variation) then a chi-square test is used to find genes significantly above the curve. This has been provided for you as the Brennecke_getVariableGenes(counts, spikes) function. However, there are only 9 spike-ins detected in this dataset so we will use the entire dataset as spike-ins.

In the figure below the red curve is the fitted technical noise model and the dashed line is the 95% CI. Pink dots are the genes with significant biological variability after multiple-testing correction. Since our dataset is relatively homogeneous only genes are identified as significantly variable.

```
Brennecke_HVG <- M3Drop::BrenneckeGetVariableGenes(
    uso_list$data,
    fdr = 0.01,
    minBiolDisp = 0.5
)
```




```
length(Brennecke_HVG)
```

```
## [1] 4
```

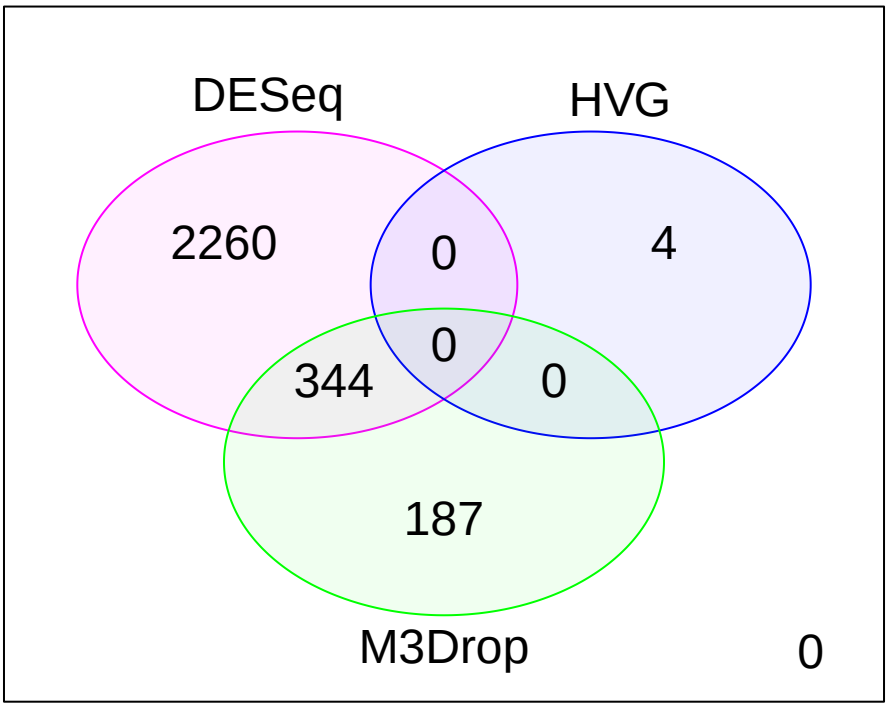
Exercise 3: Plot a heatmap of the expression of the HVGs and DESeq DE genes. Do they look differentially expressed?

Exercise 4: How many of the known markers are identified by Brennecke & DESeq?

Finally, we can compare the overlaps between these three dataset.

```
all.genes <- unique(
    c(
        as.character(DESeq_table$Gene),
        Brennecke_HVG,
        as.character(DE_genes$Gene)
    )
)
venn.diag <- vennCounts(
    cbind(
        all.genes %in% as.character(DESeq_table$Gene),
        all.genes %in% Brennecke_HVG,
        all.genes %in% as.character(DE_genes$Gene)
    )
)
limma::vennDiagram(
```

```
    venn.diag,
    names = c("DESeq", "HVG", "M3Drop"),
    circle.col = c("magenta", "blue", "green")
)
```



Chapter 20

Pseudotime analysis

```
library(TSCAN)
library(M3Drop)
library(monocle)
library(destiny)
set.seed(1)
```

In many situations, one is studying a process where cells change continuously. This includes for example many differentiation processes taking place during development, where following a stimulus, cells will change from one cell-type to another. Ideally, we would like to monitor the expression levels of an individual cell over time. Unfortunately, such monitoring is not possible with scRNA-seq since the cell is lysed (destroyed) when the RNA is extracted.

Instead, we must sample at multiple time-points and obtain snapshots of the gene expression profiles. Since some of the cells will proceed faster along the differentiation than others, each snapshot may contain cells at varying points along the developmental progression. We use statistical methods to order the cells along one or more trajectories which represent the underlying developmental trajectories, this ordering is referred to as "pseudotime".

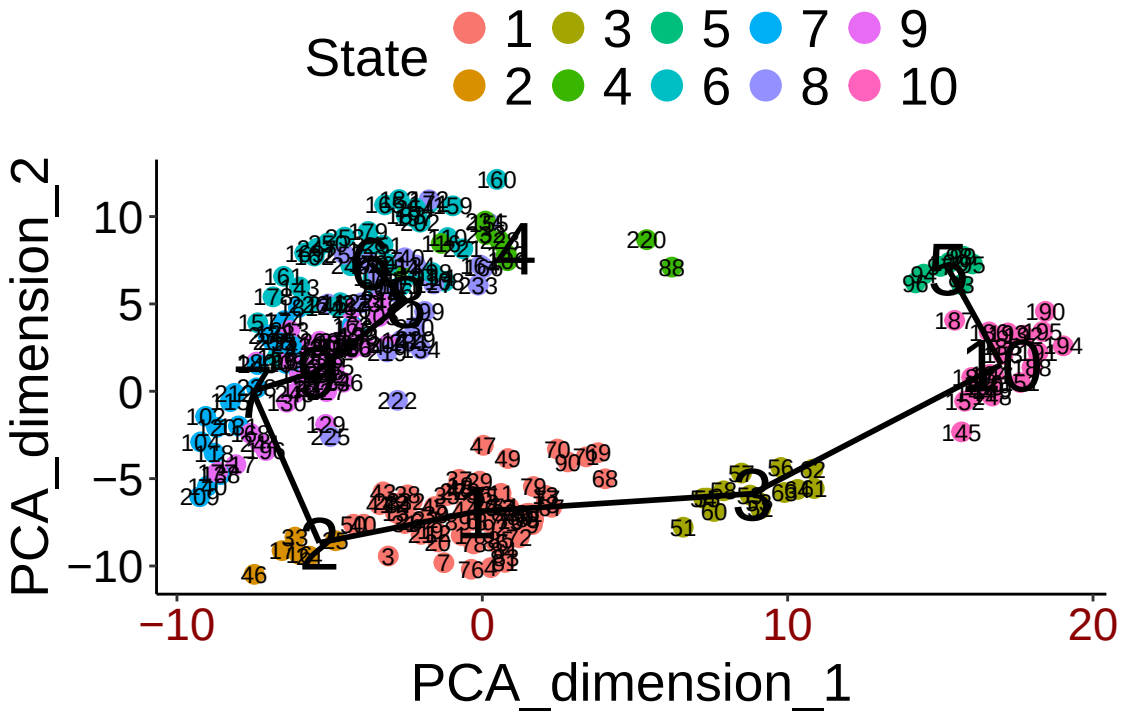
In this chapter we will consider two different tools, Monocle and TSCAN for ordering cells according to their pseudotime development. To illustrate the methods we will be using a dataset on mouse embryonic development (Deng et al., 2014). The dataset consists of 268 cells from 10 different time-points of early mouse development.

20.1 TSCAN

TSCAN combines clustering with pseudotime analysis. First it clusters the cells using `mclust`, which is based on a mixture of normal distributions. Then it builds a minimum spanning tree to connect the clusters together. The branch of this tree that connects the largest number of clusters is the main branch which is used to determine pseudotime.

First we will try to use all genes to order the cells.

```
deng <- readRDS("deng/deng.rds")
cellLabels <- colnames(deng)
procdeng <- TSCAN::preprocess(deng)
colnames(procdeng) <- 1:ncol(deng)
dengclust <- TSCAN::exprmclust(procdeng, clusternum = 10)
TSCAN::plotmclust(dengclust)
```



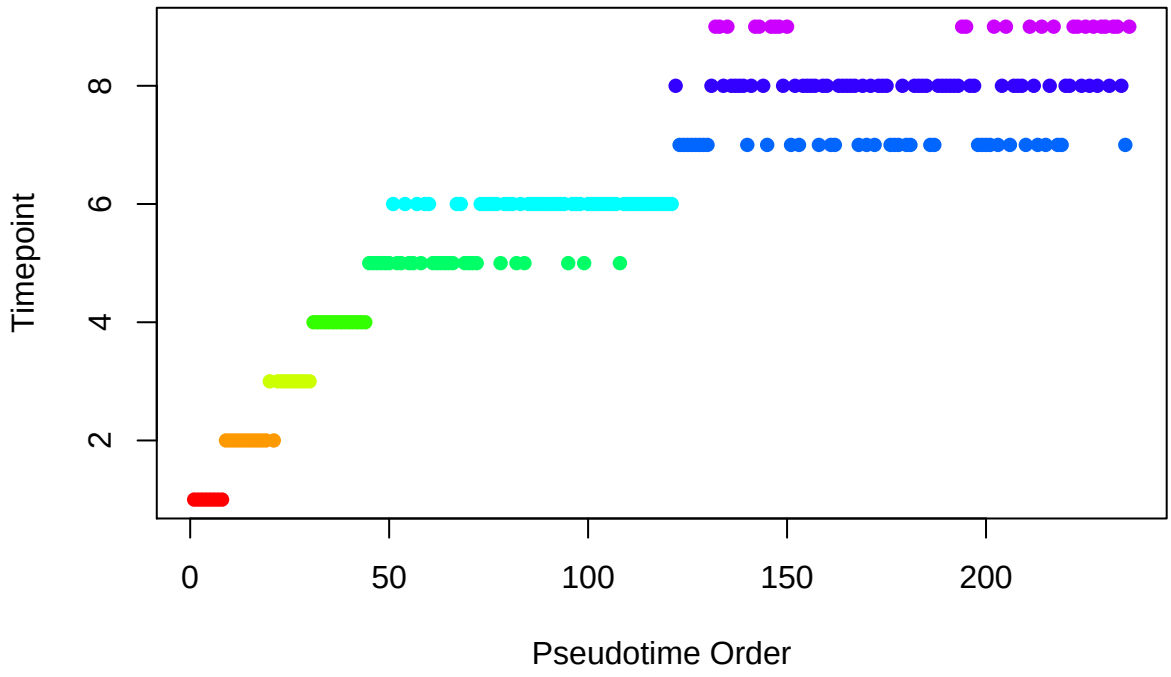

```
dengorderTSCAN <- TSCAN::TSCANorder(dengclust, orderonly = F)
pseudotime_order_tscan <- as.character(dengorderTSCAN$sample_name)
```

We can also examine which timepoints have been assigned to each state:

```
cellLabels[dengclust$clusterid == 10]
```

```
##  [1] "late2cell" "late2cell" "late2cell" "late2cell" "late2cell"
##  [6] "late2cell" "late2cell" "late2cell" "late2cell" "late2cell"
## [11] "mid2cell"  "mid2cell"  "mid2cell"  "mid2cell"  "mid2cell"
## [16] "mid2cell"  "mid2cell"  "mid2cell"  "mid2cell"  "mid2cell"
## [21] "mid2cell"  "mid2cell"
```

```
colours <- rainbow(n = 10) # red = early, violet = late
tmp <-
    factor(
        cellLabels[as.numeric(pseudotime_order_tscan)],
        levels = c("early2cell", "mid2cell", "late2cell", "4cell", "8cell",
                   "16cell", "earlyblast", "midblast", "lateblast")
    )
plot(
    as.numeric(tmp),
    xlab="Pseudotime Order",
    ylab="Timepoint",
    col = colours[tmp],
    pch = 16
)
```



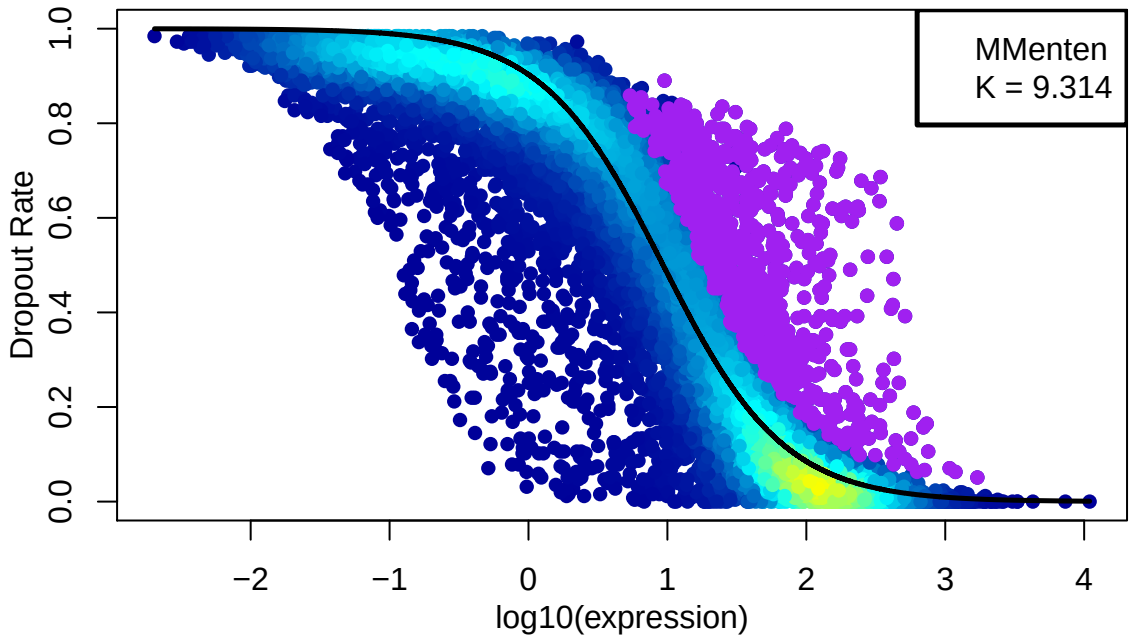
Exercise 1 Compare results for different numbers of clusters (`clusternum`).

20.2 monocle

Monocle skips the clustering stage of TSCAN and directly builds a minimum spanning tree to connect all cells. Monocle then identifies the longest path in this tree to determine pseudotime. If the data contains diverging trajectories (i.e. one cell type differentiates into two different cell-types), monocle can identify alternative long paths in the tree using the argument `num_paths`. Each of the resulting forked paths is defined as a separate cell state, thus `num_paths = 2` will identify three different cell states.

Unfortunately, Monocle does not work when all the genes are used, so we must carry out feature selection. First, we use M3Drop:

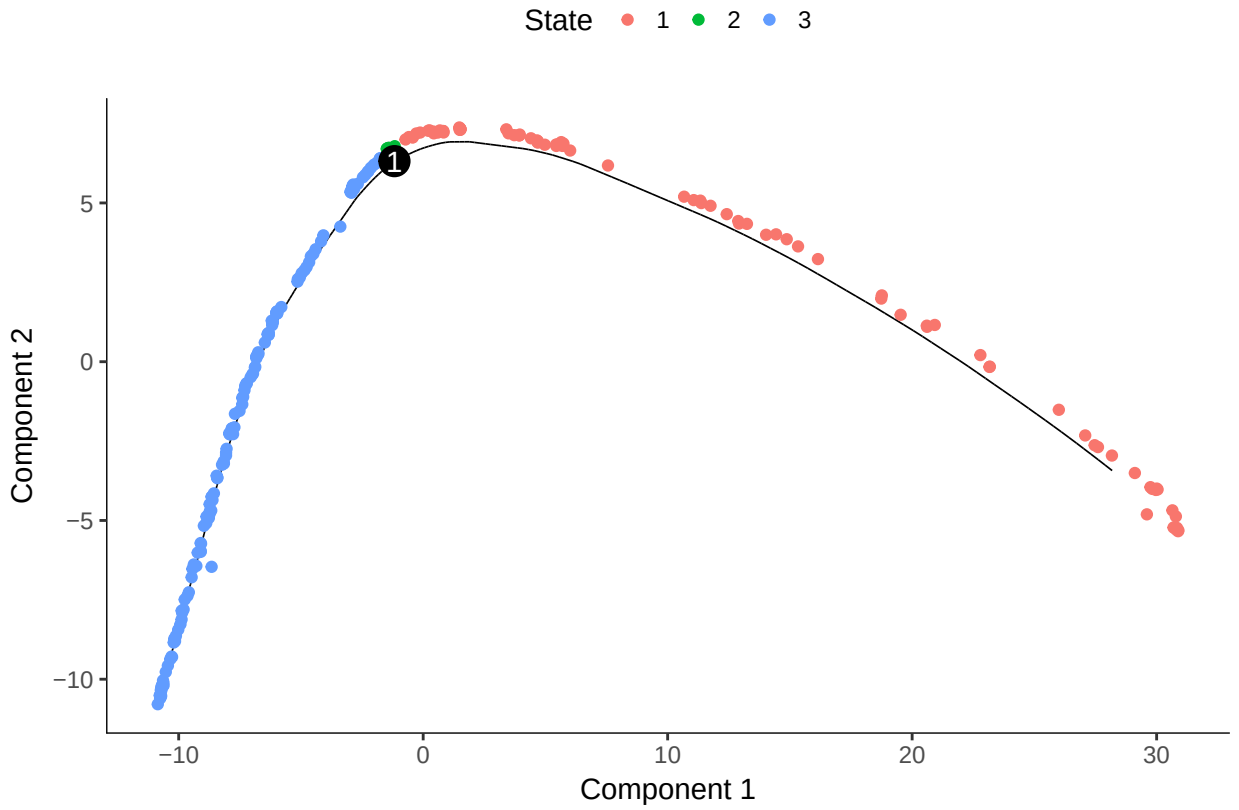
```
m3dGenes <- as.character(
    M3Drop::M3DropDifferentialExpression(deng)$Gene
)
```




```
d <- deng[which(rownames(deng) %in% m3dGenes), ]
d <- d[!duplicated(rownames(d)), ]
```

Now run monocle:

```
pd <- data.frame(timepoint = cellLabels)
pd <- new("AnnotatedDataFrame", data=pd)
fd <- as.data.frame(rownames(d))
names(fd) <- "gene"
fd <- new("AnnotatedDataFrame", data=fd)
colnames(d) <- 1:ncol(d)
rownames(d) <- 1:nrow(d)
dCellData <- monocle::newCellDataSet(d, phenoData = pd, featureData = fd)
dCellData <- monocle::setOrderingFilter(dCellData, 1:length(m3dGenes))
dCellDataSet <- monocle::reduceDimension(dCellData, pseudo_expr = 1)
dCellDataSet <- monocle::orderCells(dCellDataSet, reverse = F, num_paths = 1)
monocle::plot_spanning_tree(dCellDataSet)
```

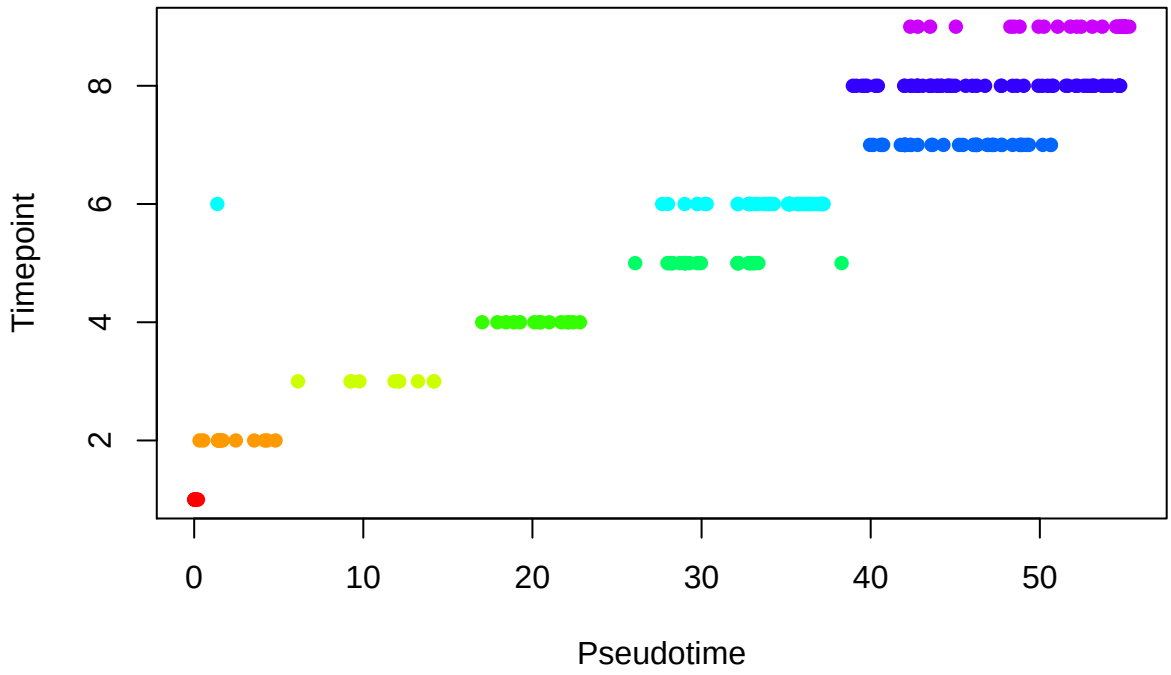



```
# Store the ordering
pseudotime_monocle <-
    data.frame(
        Timepoint = phenoData(dCellDataSet)$timepoint,
        pseudotime = phenoData(dCellDataSet)$Pseudotime,
        State=phenoData(dCellDataSet)$State
    )
rownames(pseudotime_monocle) <- 1:ncol(d)
pseudotime_order_monocle <-
    rownames(pseudotime_monocle[order(pseudotime_monocle$pseudotime), ])
```

We can again compare the inferred pseudotime to the known sampling timepoints.

```
monocle_time_point = factor(
     pseudotime_monocle$Timepoint,
     levels = c("early2cell", "mid2cell", "late2cell", "4cell", "8cell",
                "16cell", "earlyblast", "midblast", "lateblast")
)

plot(
    pseudotime_monocle$pseudotime,
    monocle_time_point,
    xlab="Pseudotime",
    ylab="Timepoint",
    col = colours[monocle_time_point],
    pch = 16
)
```



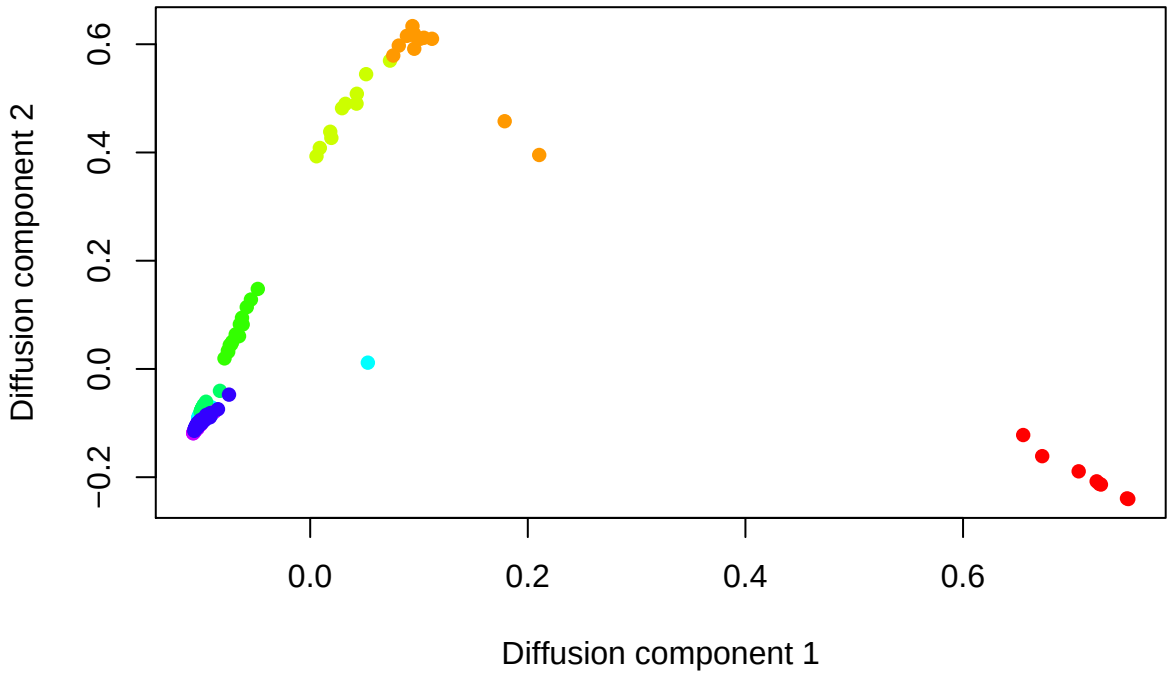
20.3 Diffusion maps

Diffusion maps were introduced by Ronald Coifman and Stephane Lafon. Briefly, the underlying idea is to assume that the data are samples from a diffusion process. The method infers the low-dimensional manifold by estimating the eigenvalues and eigenvectors for the diffusion operator related to the data.

Haghverdi et al have applied the diffusion maps concept to the analysis of single-cell RNA-seq data to create an R package called destiny.

```
dm <- DiffusionMap(t(log2(1+deng)))
tmp <- factor(
    colnames(deng),
    levels = c(
        "early2cell",
        "mid2cell",
        "late2cell",
        "4cell",
        "8cell",
        "16cell",
        "earlyblast",
        "midblast",
        "lateblast"
    )
)
plot(
    eigenvectors(dm)[,1],
    eigenvectors(dm)[,2],
    xlab="Diffusion component 1",
    ylab="Diffusion component 2",
    col = colours[tmp],
    pch = 16
```

```
)
```



Like the other methods, destiny does a good job at ordering the early time-points, but it is unable to distinguish the later ones.

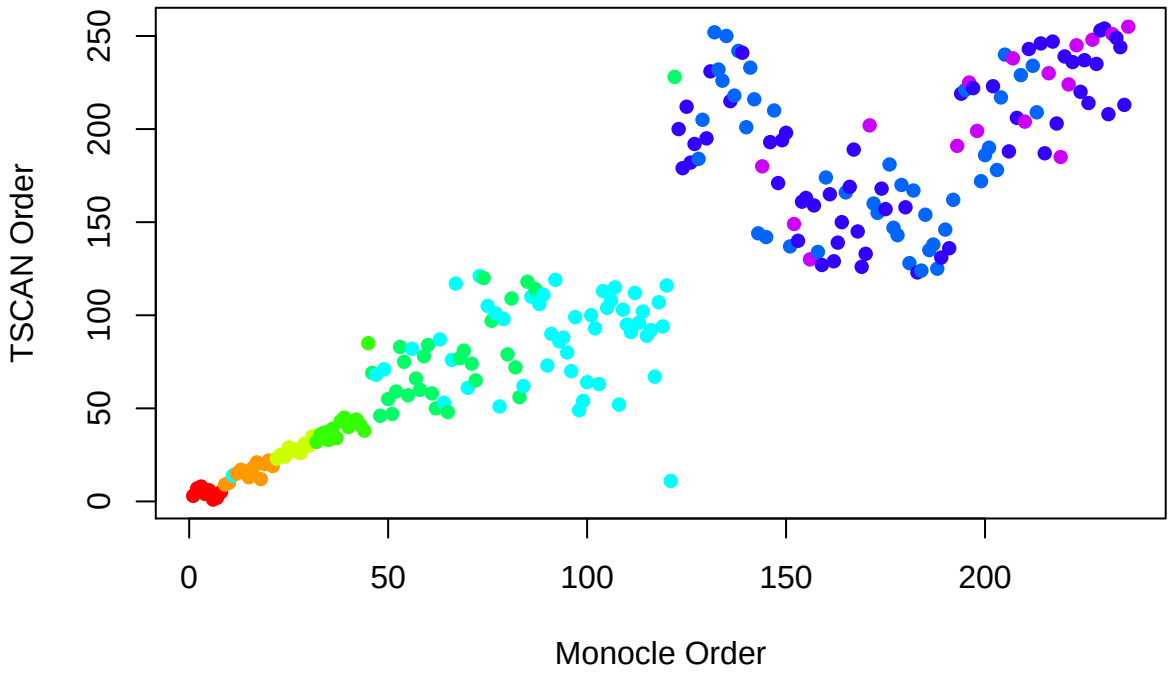
Exercise 2 Do you get a better resolution between the later time points by considering additional eigenvectors?

Exercise 3 How does the ordering change if you only use the genes identified by M3Drop?

20.4 Comparison of the methods

How do the trajectories inferred by TSCAN and Monocle compare?

```
matched_ordering <-
    match(
        pseudotime_order_tscan,
        pseudotime_order_monocle
    )
timepoint_ordered <-
    monocle_time_point[order(pseudotime_monocle$pseudotime)]
plot(
    matched_ordering,
    xlab = "Monocle Order",
    ylab = "TSCAN Order",
    col = colours[timepoint_ordered],
    pch = 16
)
```



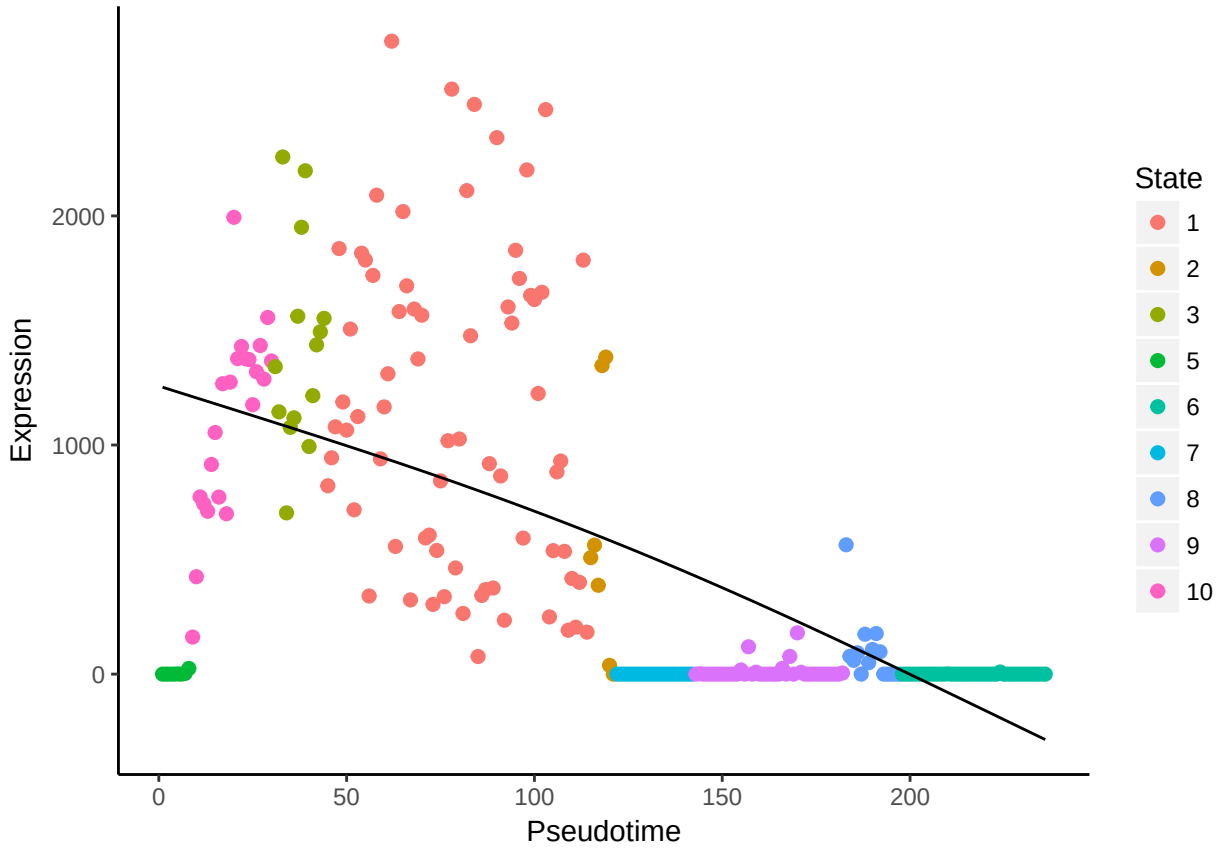
Exercise 4 Compare destiny to TSCAN and Monocle.

20.5 Expression of genes through time

Each package also enables the visualization of expression through pseudotime.

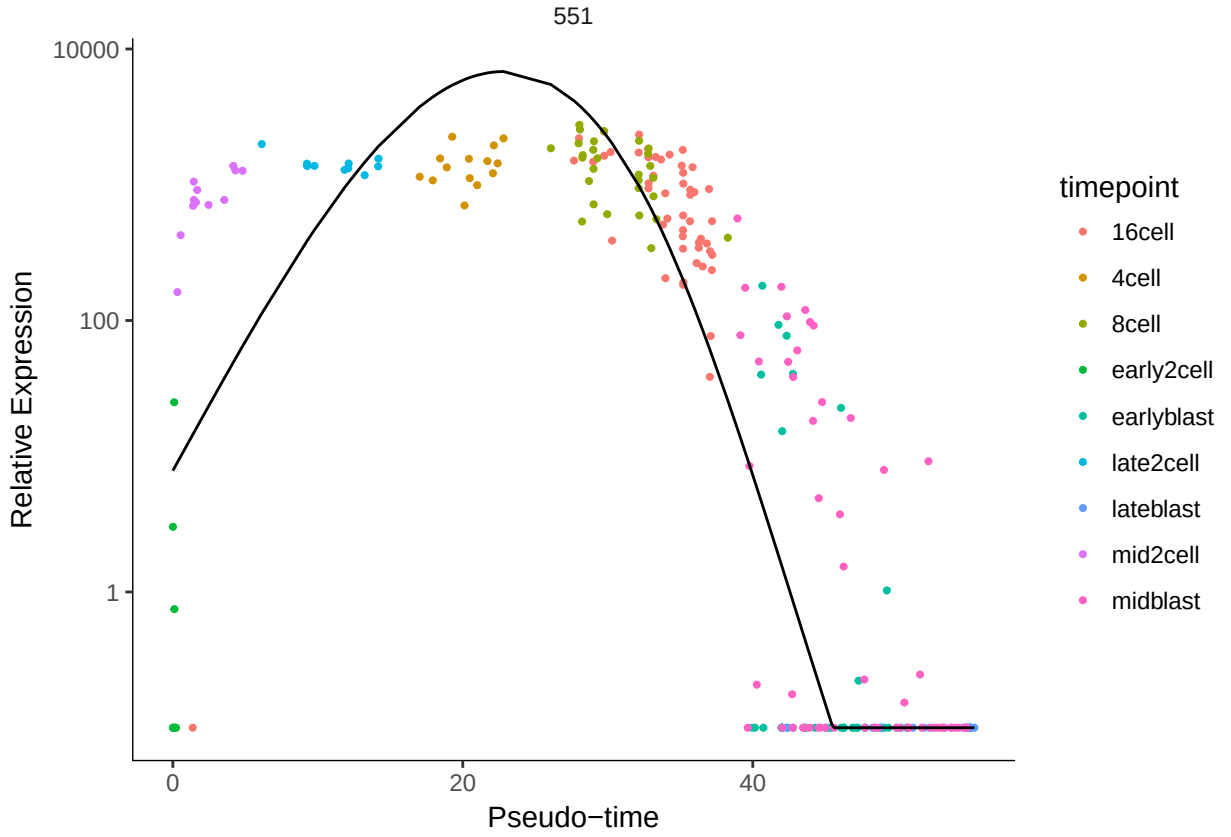
TSCAN

```
colnames(deng) <- 1:ncol(deng)
TSCAN::singlegeneplot(
    deng[rownames(deng) == "Obox6", ],
    dengorderTSCAN
)
```



Monocle

```
monocle::plot_genes_in_pseudotime(
    dCellDataSet[fData(dCellDataSet)$gene == "Obox6",],
    color_by = "timepoint"
)
```



Exercise 5: Repeat the exercise using a subset of the genes, e.g. the set of highly variable genes that can be obtained using M3Drop::Brennecke_getVariableGenes

Chapter 21

Differential Expression (DE) analysis

21.1 Bulk RNA-seq

One of the most common types of analyses when analyzing bulk RNA-seq data is to identify differentially expressed genes. By comparing the genes that change between two conditions, e.g. mutant and wild-type or stimulated and unstimulated, it is possible to characterize the molecular mechanisms underlying the change.

Several different methods, e.g. DESeq2 and edgeR, have been developed for bulk RNA-seq. Moreover, there are also extensive datasets available where the RNA-seq data has been validated using RT-qPCR. These data can be used to benchmark DE finding algorithms.

21.2 Single cell RNA-seq

In contrast to the bulk RNA-seq in scRNA-seq we usually do not have a defined set of experimental conditions, but instead, as was shown in the previous chapter (18) we can identify the cell groups by using the unsupervised clustering approach. Once the groups have been identified one can find differentially expressed genes by either looking at the differencies in variance between the groups (like the Kruskal-Wallis test implemented in SC3), or by comparing gene expression between clusters in a pairwise manner. In the following chapter we will mainly consider tools developed for the comparison of the two groups of cells.

21.3 scRNA-seq synthetic data

One advantage of working with single-cell data is that there is a reliable, analytically tractable mathematical model for the expression levels, the Poisson-Beta distribution. Importantly, the Poisson-Beta distribution has strong experimental support, and it provides a good fit to scRNA-seq data.

In this module, we first discuss the Poisson-Beta distribution. We then use the model to generate synthetic data which we can use to compare different DE finding algorithms. In the final section we investigate a real scRNA-seq dataset.

21.4 Single gene expression

```
library(scRNA.seq.funcs)
set.seed(1)
```

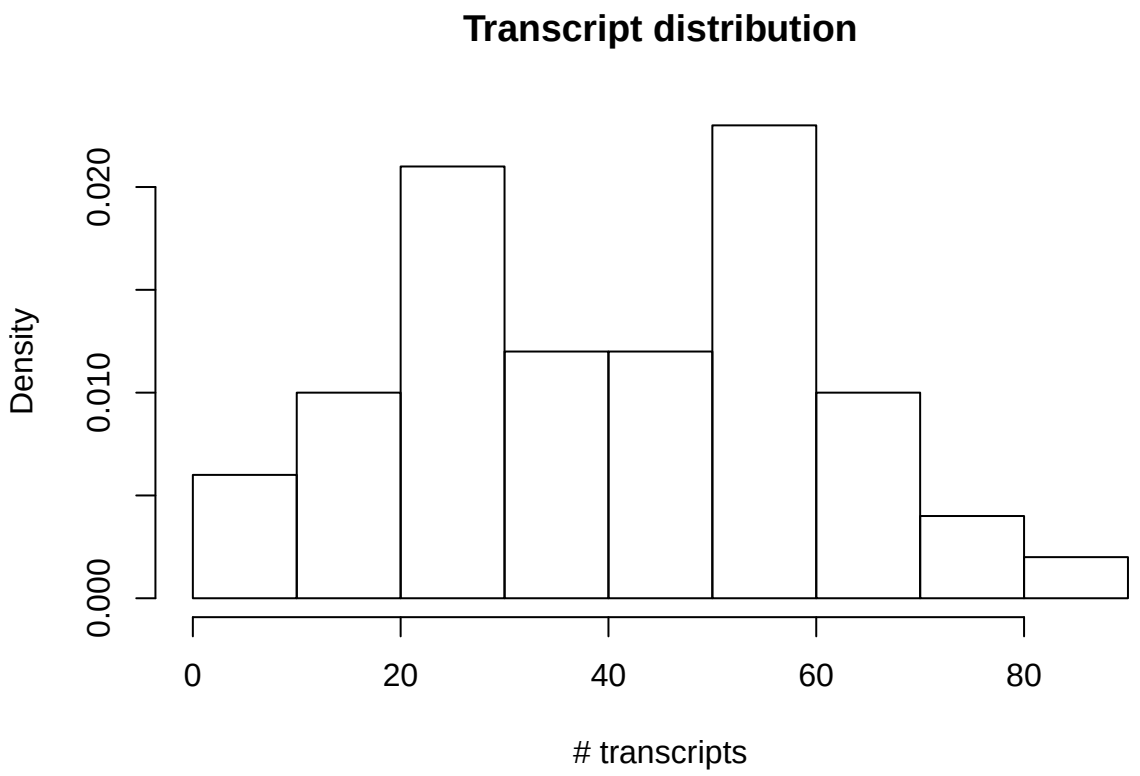


Figure 21.1: Distribution of read counts for a single genes across 100 cells based on the Poisson-Beta model

For single-cell data, the analytically tractable stochastic bursting model provides a good fit. Thus, we can use it to generate realistic data. We start by generating samples from one gene:

```
s <- scRNA.seq.funcs::PoiBeta(100, 2, 3)
```

We then plot the results as a histogram:

```
hist(s,
     freq = FALSE,
     xlab = "# transcripts",
     main = "Transcript distribution")
```

The probability mass function of the Poisson-Beta distribution is challenging to work with since it involves special functions. However, the mean and variance can be calculated as:

$\mu = k \cdot a/(a+b)$

$\sigma^2 = k^2 \cdot a \cdot b/((a+b+1) \cdot (a+b)^2)$

21.5 Poisson-Beta distribution

There are three regimes of the Poisson-Beta distribution and they are determined by the values of the parameters a and b.

When $a < 1$ and $b < 1$ we have a bimodal distribution with one mode at 0 and the other at k, when $a < 1$ and $b > 1$ we have a monotonically decreasing distribution and otherwise we have a unimodal distribution with a mode at $ka/(a+b)$.

```
par(mfrow=c(3,1))
hist(scRNA.seq.funcs::PoiBeta(100, .2, .3),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0, 0, 1, 1/4),
     breaks = seq(0, 120, 10))
hist(scRNA.seq.funcs::PoiBeta(100, .2, 3),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0,1,0,1/4),
     breaks = seq(0, 120, 10))
hist(scRNA.seq.funcs::PoiBeta(100, 2, .3),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(1, 0, 0, 1/4),
     breaks = seq(0, 120, 10))
```

Exercise 1: Vary the parameters a, b and k to explore how the location and shape of the distribution changes.

21.6 Sample size

The difficulty in determining the parameters of the distribution also depends on the sample size. In the example below, it is not clear if the distribution is bimodal when only 10 samples are drawn.

```
hist(scRNA.seq.funcs::PoiBeta(10, .6, 1.2, 10),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(1, 0, 0, 1/4),
     breaks = seq(0, 20, 1))
hist(scRNA.seq.funcs::PoiBeta(10, .6, 1.2, 50),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0, 1, 0, 1/4),
     breaks = seq(0, 20, 1),
     add = TRUE)
```

Exercise 2: Modify the number of samples (i.e. cells) drawn to explore how difficult it is to correctly infer the correct shape.

21.7 Dropout noise

The stochastic bursting model only captures the biological variability. In practice there will also be experimental variability. We model the noise as drop-outs, i.e. we assume that there is a small probality that each transcript is lost. For gene i and cell j it is assumed that the probability of loosing the transcript is given by $p_d = \mu/(d+\mu)$, where μ is the mean expression level of the gene and d is a drop-out parameter. Thus, the probability of drop-outs monotonically decreases as the mean expression level increases.

To visualize the impact of the drop-outs on a sample, we can tune the drop-out parameter:

```
par(mfrow=c(3,1))
hist(scRNA.seq.funcs::PoiBeta(100, 2, 3, 100, 1),
```

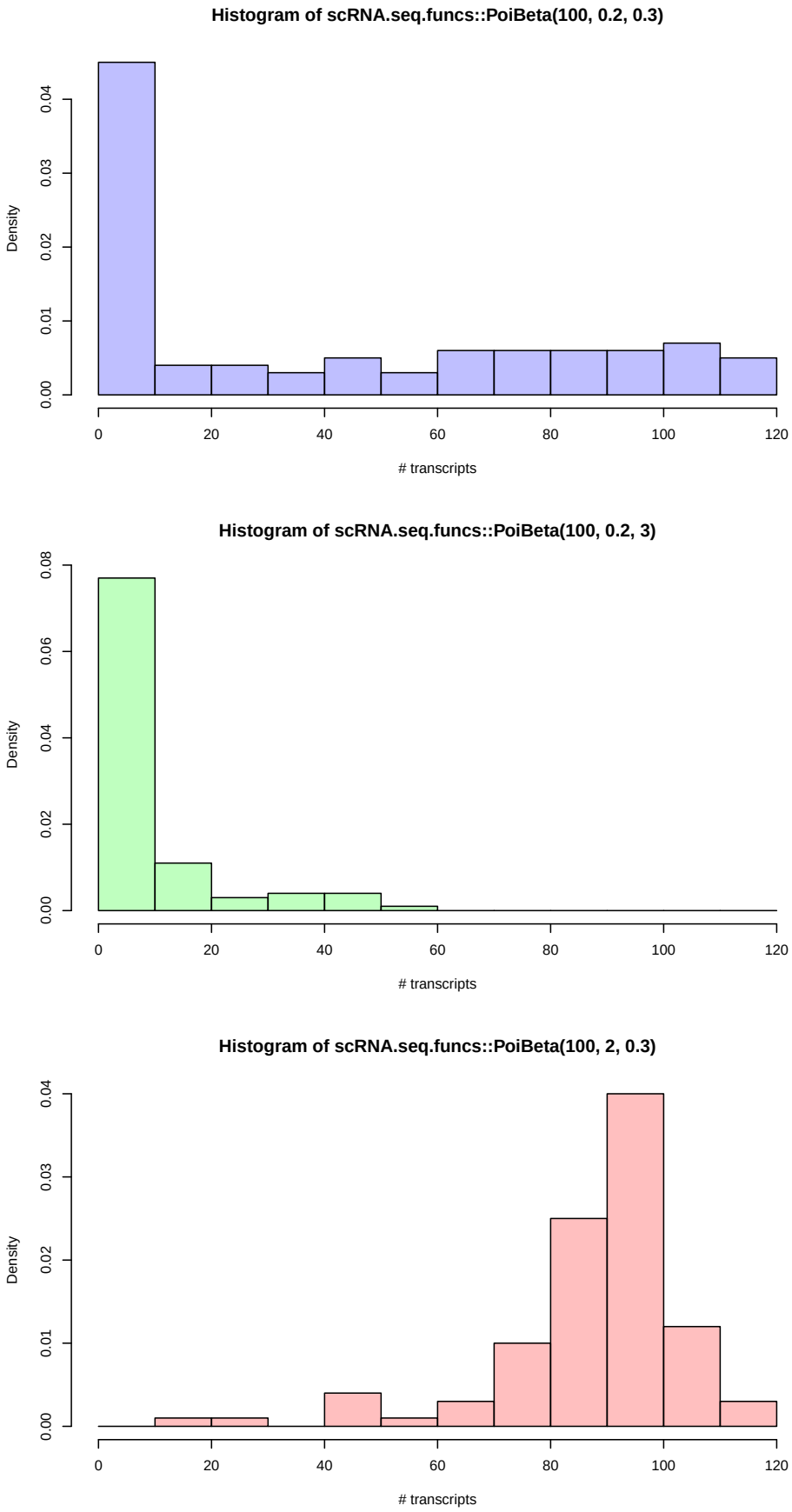


Figure 21.2: Different distributions of read counts for a single genes across 100 cells based on the Poisson-Beta model corresponding to different paramete sets

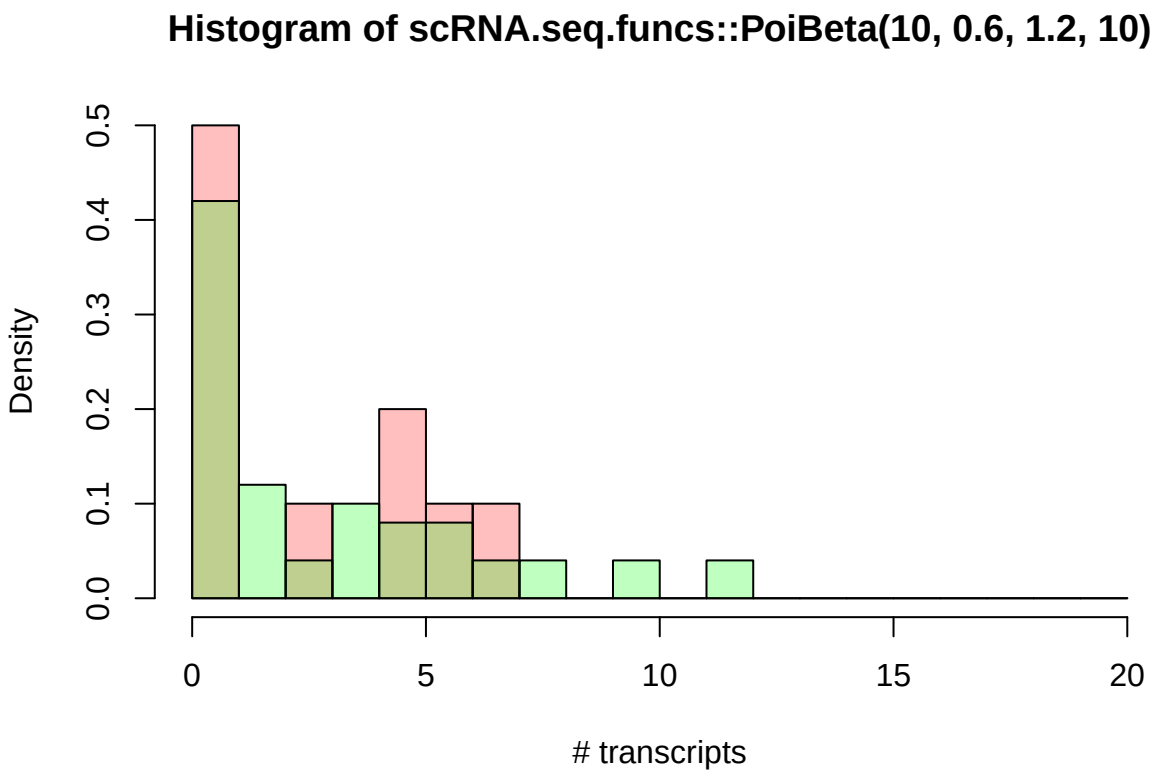


Figure 21.3: Effect of sampling size on the distribution of read counts based on the Poisson-Beta model

```
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(1, 0, 0, 1/4),
     ylim = c(0, .08),
     xlim = c(0, 120))
hist(scRNA.seq.funcs::PoiBeta(100, 2, 3, 100, 10),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0, 1, 0, 1/4),
     ylim = c(0, .08),
     xlim = c(0, 120))
hist(scRNA.seq.funcs::PoiBeta(100, 2, 3, 100, 100),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0, 0, 1, 1/4),
     ylim = c(0, .08),
     xlim = c(0, 120))
```

Exercise 3: Explore the different parameter regimes for the same drop-out rate. Do you think that we are more sensitive to drop-outs in any specific regime?

21.8 Dispersion noise

Another example of noise is under- or over-dispersion. This can be modelled using a single parameter multiplying the parameters a and b in the Poisson-Beta distribution by a scalar s. We can see that the mean

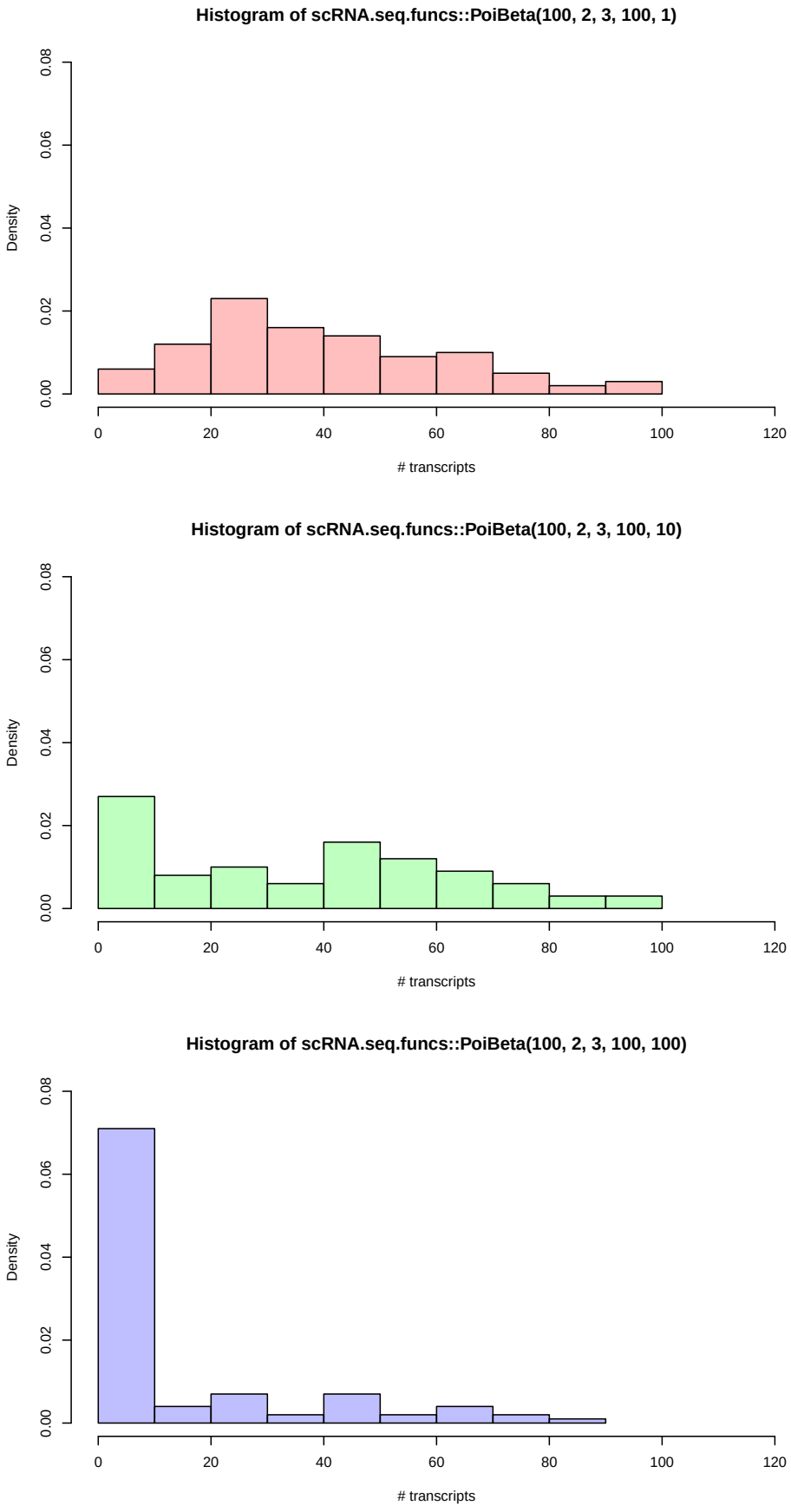


Figure 21.4: Effect of dropouts on the distribution of read counts based on the Poisson-Beta model

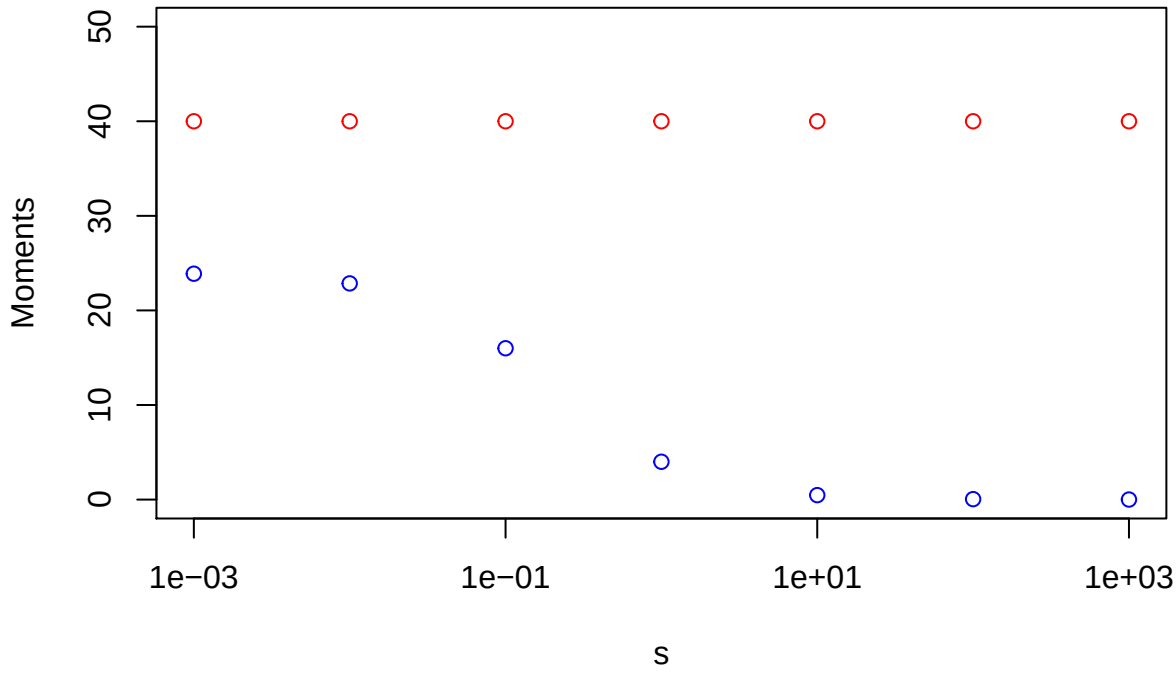


Figure 21.5: The mean and the variance of the distribution of read counts based on the Poisson-Beta model

is left unchanged while the variance is inversely proportional to s.

```
s <- 10^(-3:3)
k <- 100
a <- 2
b <- 3
par(mfrow=c(1,1))
plot(s,
     k*a*s/(a*s + b*s),
     log = "x",
     col = "red",
     ylim = c(0, 50),
     ylab = "Moments")
points(s,
       k*a*s*b*s/((a*s + b*s)^2*(a*s + b*s + 1)),
       col = "blue")
```

To illustrate the effect of the dispersion parameter on the distribution consider:

```
par(mfrow=c(3,1))
hist(scRNA.seq.funcs::PoiBeta(100, 2, 3, 100),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(1, 0, 0, 1/4),
     ylim = c(0, .05),
     xlim = c(0, 120))
hist(scRNA.seq.funcs::PoiBeta(100, 2/10, 3/10, 100),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0, 1, 0, 1/4),
     ylim = c(0, .05),
```

```
     xlim = c(0, 120))
hist(scRNA.seq.funcs::PoiBeta(100, 2*10, 3*10, 100),
     freq = FALSE,
     xlab = "# transcripts",
     col = rgb(0, 0, 1, 1/4),
     ylim = c(0, .05),
     xlim = c(0, 120))
```

Exercise 4: Explore what happens when you have both drop-outs and under/over-dispersion. Can the effects be deconvoluted?

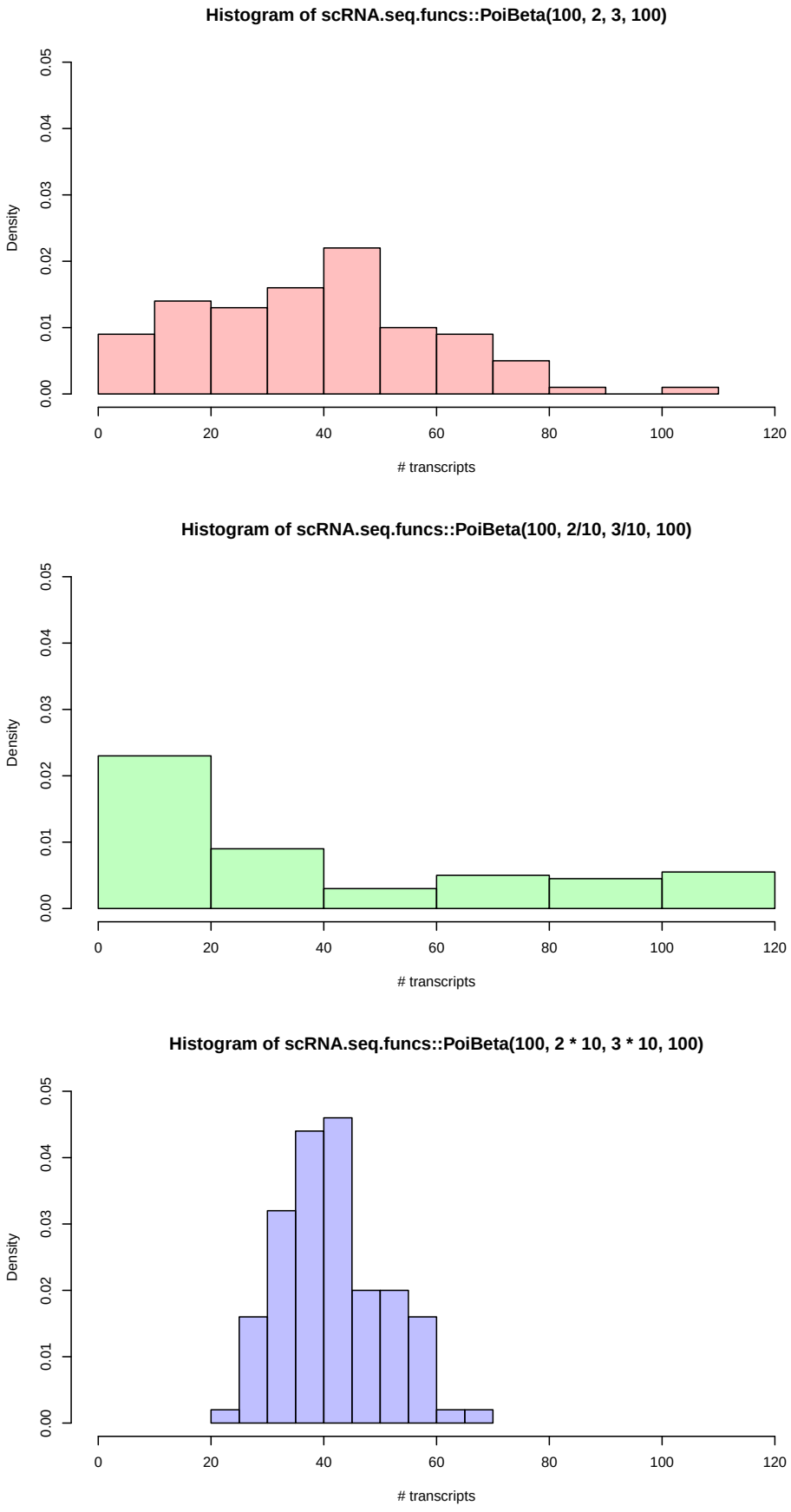


Figure 21.6: Effect of dispersion on the distribution of read counts based on the Poisson-Beta model

Chapter 22

DE in a synthetic dataset

```
library(scRNA.seq.funcs)
library(DESeq2)
library(scde)
library(ROCR)
library(limma)
set.seed(1)
```

22.1 Dataset

We start by generating samples using the Poisson-Beta distribution, using 100 genes from 50 cells. To simulate the second experimental condition we select one of the three parameters for each gene and modify it by multiplying by a normally distributed random factor.

```
nGenes <- 1e2
nCells <- 50
mult <- 2^(rnorm(nGenes, 0, 2))
synData <- scRNA.seq.funcs::GeneratePoiBetaSamples(
    ks = 10^(rnorm(nGenes, 3, .5)),
    as = 10^(rnorm(nGenes, -1, .5)),
    bs = 10^(rnorm(nGenes, 0, .5)),
    mult,
    nGenes,
    nCells
)
g <- synData$sample1
g2 <- synData$sample2
```

22.2 DE in scRNA-seq

For bulk data, each gene is represented by a single value and to identify DEGs we need to identify those genes where the difference in expression between two conditions is sufficiently large. Replicates are needed for us to be able to assess the fold-change as well as its significance.

For single-cell data, the situation is more complicated; instead of comparing two means we are faced with the task of comparing two probability distributions. There are many different functions available for comparing

two probability distributions (e.g. Total variation distance and Kullback-Leibler divergence), and since they emphasize different features, they have different properties.

To establish a ground-truth, we arbitrarily assign genes where one of the parameters has changed by more than a factor of 4 as being true positives. Remaining genes are considered not significantly changed.

```
changedGenes <- abs(log2(mult)) > 2
changedGenesInds <- which(changedGenes)
notChangedGenesInds <- which(abs(log2(mult)) <= 2)
```

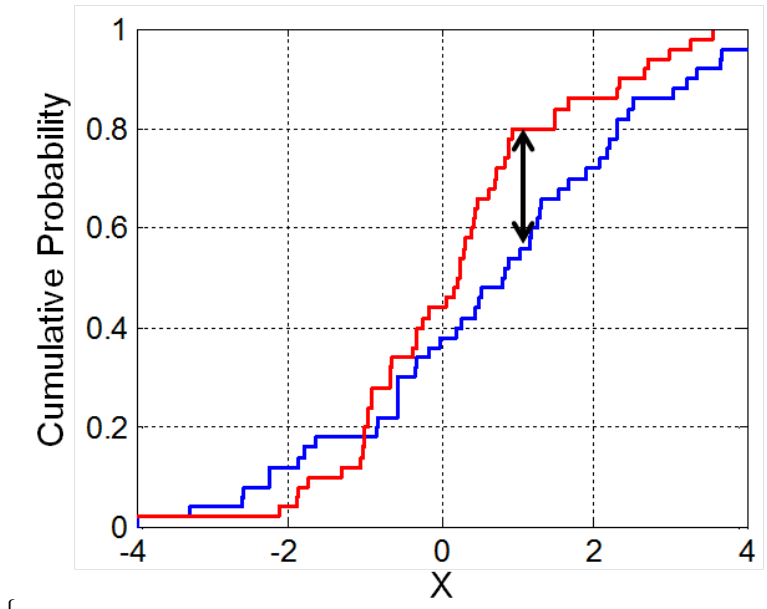
22.3 Kolmogorov-Smirnov test

The types of test that are easiest to work with are non-parametric ones. The most commonly used non-parametric test is the Kolmogorov-Smirnov test (KS-test) and we can use it to compare the distributions for each gene in the two conditions.

The KS-test quantifies the distance between the empirical cummulative distributions of the expression of each gene in each of the two populations. It is sensitive to changes in mean experssion and changes in variability. However it assumes data is continuous and may perform poorly when data contains a large number of identical values (eg. zeros).

```
knitr::include_graphics("figures/KS2_Example.png")
```

\begin{figure}



{

}

\caption{Illustration of the two-sample Kolmogorov–Smirnov statistic. Red and blue lines each correspond to an empirical distribution function, and the black arrow is the two-sample KS statistic. (taken from here)} \end{figure}

```
nGenes <- nrow(g)
pVals <- rep(1, nGenes)
for (i in 1:nGenes) {
    res <- ks.test(g[i,], g2[i,])
    # Bonferroni correction
```

```
    pVals[i] <- res$p.value*nGenes
}
```

Using the standard p-value cut-off .05, we can find out how many genes that were called as significantly different.

```
ksChangedGenes <- which(pVals < 0.05)
ksNotChangedGenes <- which(pVals >= 0.05)
cat(changedGenesInds)
```

```
## 4 11 14 15 24 28 31 35 39 54 55 56 58 61 67 68 70 75 83 84 87 92 93 95 97 99
```

```
cat(ksChangedGenes)
```

```
## 13 28 31 35 40 55 56 61 67 84 87 97
```

```
cat(intersect(changedGenesInds, ksChangedGenes))
```

```
## 28 31 35 55 56 61 67 84 87 97
```

22.4 Performance of KS test

The genes identified by the KS-test differs substantially from the ground truth. Instead of considering the absolute number of identified genes, it is often more informative to consider False positive rate (FPR) and the True positive rate (TPR). The False positive rate is defined as FPR = FP/(FP + TP) and the True positive rate as TPR = TP/(TP + FN), where FP is the number of false positives, TN the number of true negatives, TP the number of true negatives and FN the number of false negatives.

```
tp <- length(intersect(changedGenesInds, ksChangedGenes))
fn <- length(intersect(changedGenesInds, which(pVals >= .05)))
fp <- length(intersect(notChangedGenesInds, ksChangedGenes))
tn <- length(intersect(notChangedGenesInds, which(pVals >= .05)))
tpr <- tp/(tp + fn)
fpr <- fp/(fp + tn)
cat(c(tpr, fpr))
```

```
## 0.3846154 0.02702703
```

As you can see, the p-value cut-off .05 results in a low TPR and a high FPR. That is, the test has failed to identify many of the genes that were truly changed and many of the changed genes were not detected.

Clearly, there is a trade-off between TPR and FPR. If one is willing to accept a higher FPR, then one will be able to achieve a higher TPR. The relationship between FPR and TPR is typically shown as a receiver-operator-characteristic (ROC) curve. To generate and plot the ROC curve, we need to change the p-value cut-off.

To facilitate the plotting, we use the package "ROCR"

```
pred <- ROCR::prediction(pVals, as.numeric(abs(log2(mult)) <= 2))
perf <- ROCR::performance(pred, "tpr", "fpr")
ROCR::plot(perf)
```

Often we are interested in comparing several ROC curves. To carry out such a comparison, we need to summarize the entire curve using only one scalar value. This can be achieved by calculating the area under the ROC curve (AUROC). Since an ROC curve has to stay above the diagonal (why?) the AUROC will be between .5 and 1.

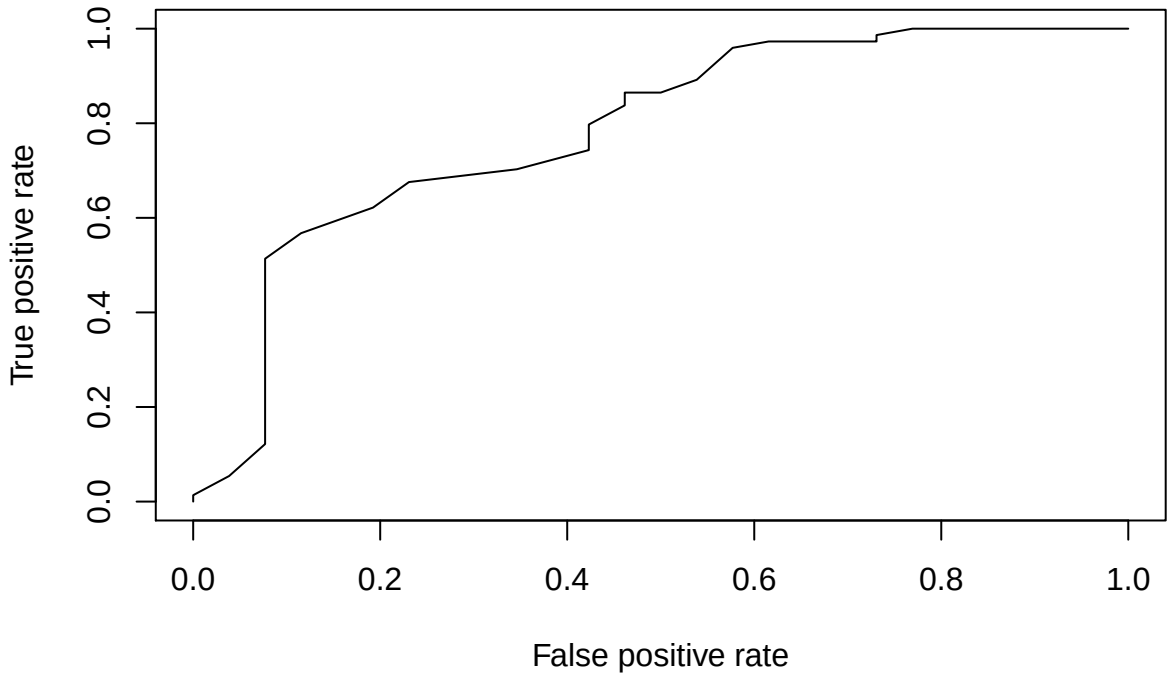


Figure 22.1: Different distributions of read counts for a single genes across 50 cells based on the Poisson-Beta model corresponding to different paramete sets

```
aucObj <- ROCR::performance(pred, "auc")
aucObj@y.values[[1]]
```

```
## [1] 0.7837838
```

Exercise: Compare the AUC values when you change some of the parameters in the analysis, e.g. number of cells, number of genes, threshold for considering a gene differentially expressed, distribution of parameter values, distribution of fold-changes. What factors make it easy or hard to identify differentially expressed genes?

22.5 DESeq2

One could still apply bulk DE methods to scRNA-seq data. One of the most popular methods for differential expression analysis for bulk RNA-Seq data is DESeq2. DESeq2 estimates the variability of each gene using a quadratic relationship between mean expression and variability across all samples fit using Bayesian statistics. It then tests for a difference in mean expression across groups based on a negative binomial distribution.

Let's try it out on our synthetic dataset:

```
cnts <- cbind(g, g2)
cond <- factor(
    c(
        rep("A", ncol(g)),
        rep("B", ncol(g2))
    )
)
# object construction, add a pseudo-count of 1 to make DESeq work
```

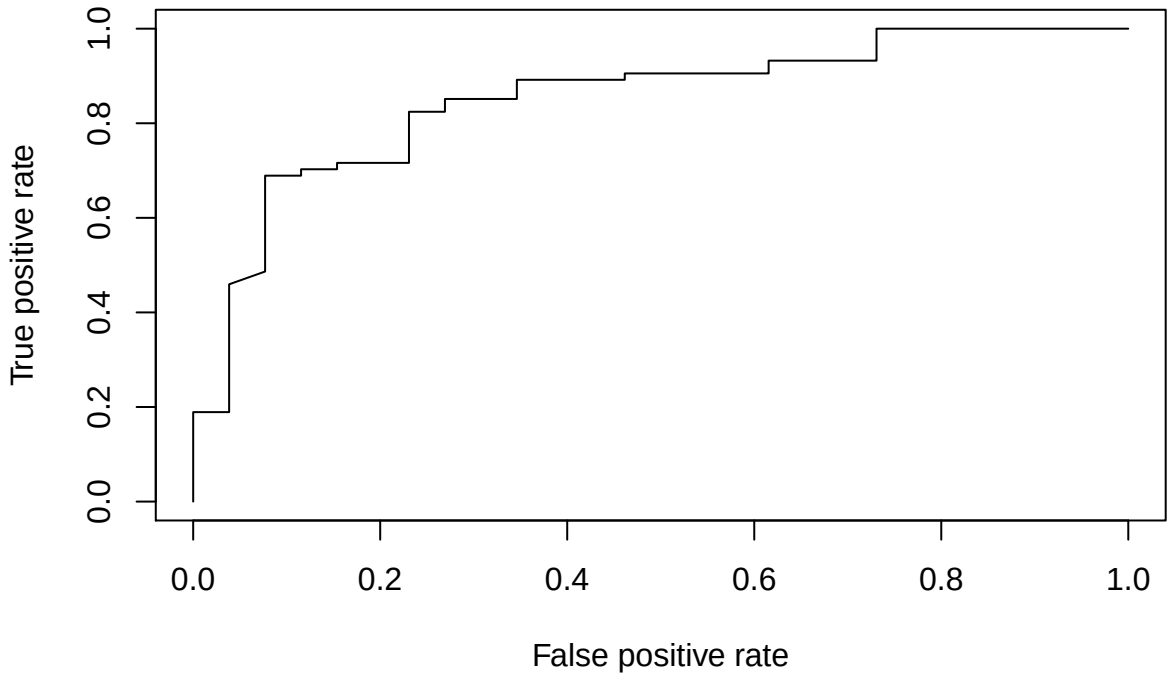


Figure 22.2: Different distributions of read counts for a single genes across 50 cells based on the Poisson-Beta model corresponding to different paramete sets

```
dds <- DESeq2::DESeqDataSetFromMatrix(
    cnts + 1,
    DataFrame(cond),
    ~ cond)
dds <- DESeq2::DESeq(dds)
resDESeq <- results(dds)
```

Check the performance of DESeq2:

```
pValsDESeq <- resDESeq$padj
predDESeq <- prediction(pValsDESeq, as.numeric(abs(log2(mult)) <= 2))
perfDESeq <- performance(predDESeq, "tpr", "fpr")
ROCR::plot(perfDESeq)
```

```
aucObjDESeq <- performance(predDESeq, "auc")
aucObjDESeq@y.values[[1]]
```

```
## [1] 0.8503119
```

Exercise: Based on the AUC-value, does DESeq or the KS-test seem more accurate? Can you find a parameter regime where the ranking is changed?

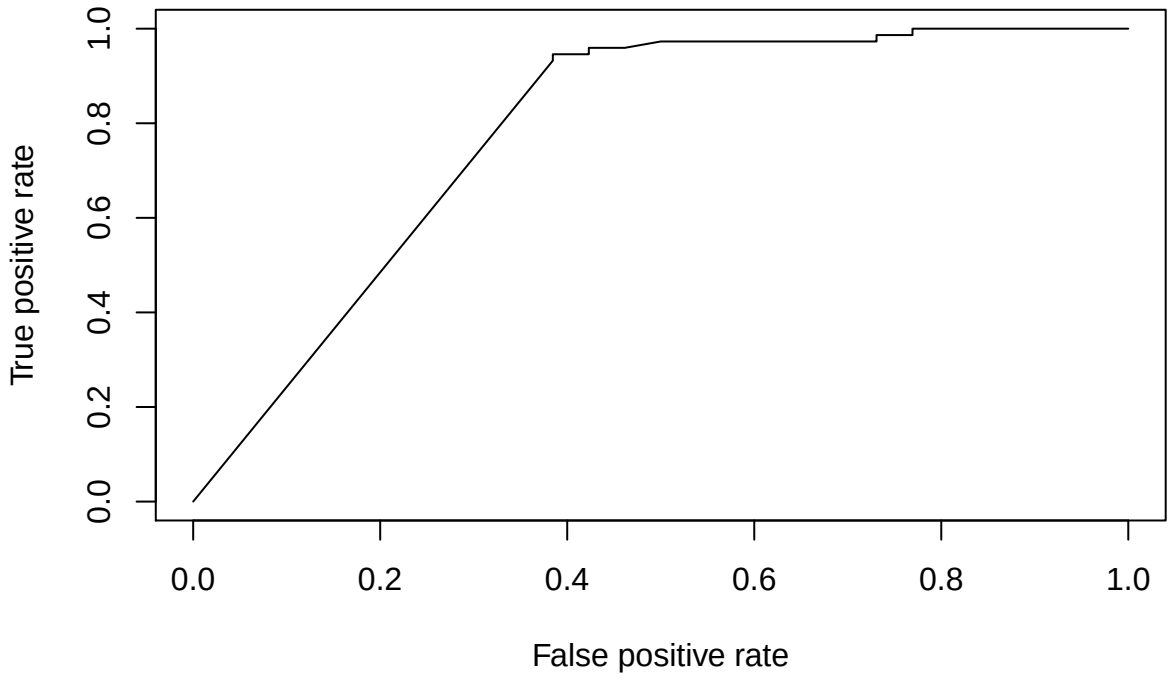
22.6 SCDE

There are yet much fewer methods available for scRNA-seq data than for bulk data, but one better known method is SCDE. SCDE uses Bayesian statistics to fit a zero-inflated negative binomial distribution to the expression of each gene and tests for differences in mean expression level between groups. We can use it on the synthetic data:

```
cnts <- cbind(g, g2)
cnts <- apply(
    cnts,
    2,
    function(x) {
        storage.mode(x) <- 'integer'
        return(x)
    }
)
cond <- factor(
    c(
        rep("A", ncol(g)),
        rep("B", ncol(g2))
    )
)
names(cond) <- 1:length(cnts[1, ])
colnames(cnts) <- 1:length(cnts[1, ])
o.ifm <- scde::scde.error.models(
    counts = cnts,
    groups = cond,
    n.cores = 1,
    threshold.segmentation = TRUE,
    save.crossfit.plots = FALSE,
    save.model.plots = FALSE,
    verbose = 0,
    min.size.entries = 20
)
priors <- scde::scde.expression.prior(
    models = o.ifm,
    counts = cnts,
    length.out = 400,
    show.plot = FALSE)
resSCDE <- scde::scde.expression.difference(
    o.ifm,
    cnts,
    priors,
    groups = cond,
    n.randomizations = 100,
    n.cores = 1,
    verbose = 0)
# Convert Z-scores into 2-tailed p-values
pValsSCDE <- pnorm(abs(resSCDE$cZ), lower.tail = FALSE) * 2
pValsSCDE <- p.adjust(pValsSCDE, method = "bonferroni")
```

Exercise: Calculate an AUROC value and compare to the other methods.

Our answer:

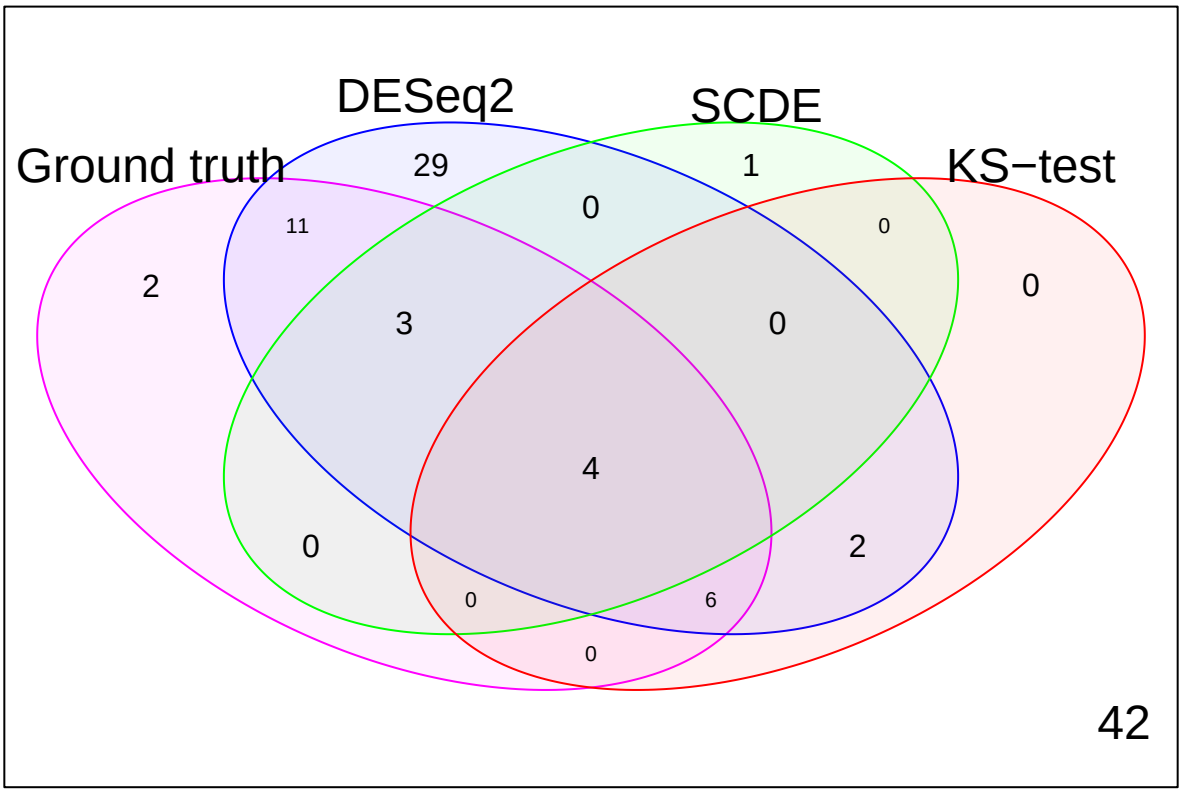



```
## [1] 0.7830042
```

22.7 Comparison of the methods

From the above analyses it is clear that none of the three methods (KS, DESeq2 and SCDE) is able to reliably find the majority of the DE genes. A popular strategy in this situation is to combine two or more methods, in the hope that genes that were identified by more than method are more likely to be true positives.

```
par(mfrow = c(1, 1))
limma::vennDiagram(
    vennCounts(
        cbind(
            changedGenes,
            pValsDESeq < .05,
            pValsSCDE < .05,
            pVals < .05
        )
    ),
    names = c("Ground truth", "DESeq2", "SCDE", "KS-test"),
    circle.col = c("magenta", "blue", "green", "red"))
```




```
allChangedInds <- intersect(
    which(pValsDESeq < .05),
    intersect(which(pValsSCDE < .05),
              which(pVals < .05)
    )
)
tpAll <- length(intersect(changedGenesInds, allChangedInds))
fnAll <- length(intersect(changedGenesInds, setdiff(1:1e3, allChangedInds)))
fpAll <- length(intersect(notChangedGenesInds, allChangedInds))
tnAll <- length(intersect(notChangedGenesInds, setdiff(1:1e3, allChangedInds)))
tprAll <- tpAll / (tpAll + fnAll)
fprAll <- fpAll / (fpAll + tnAll)
cat(c(tprAll, fprAll))
```

```
## 0.1538462 0
```

The more stringent approach results in a lower FPR at the cost of a lower TPR.

Exercise: Calculate the TPR and FPR for the case when we require two out of three methods to agree.

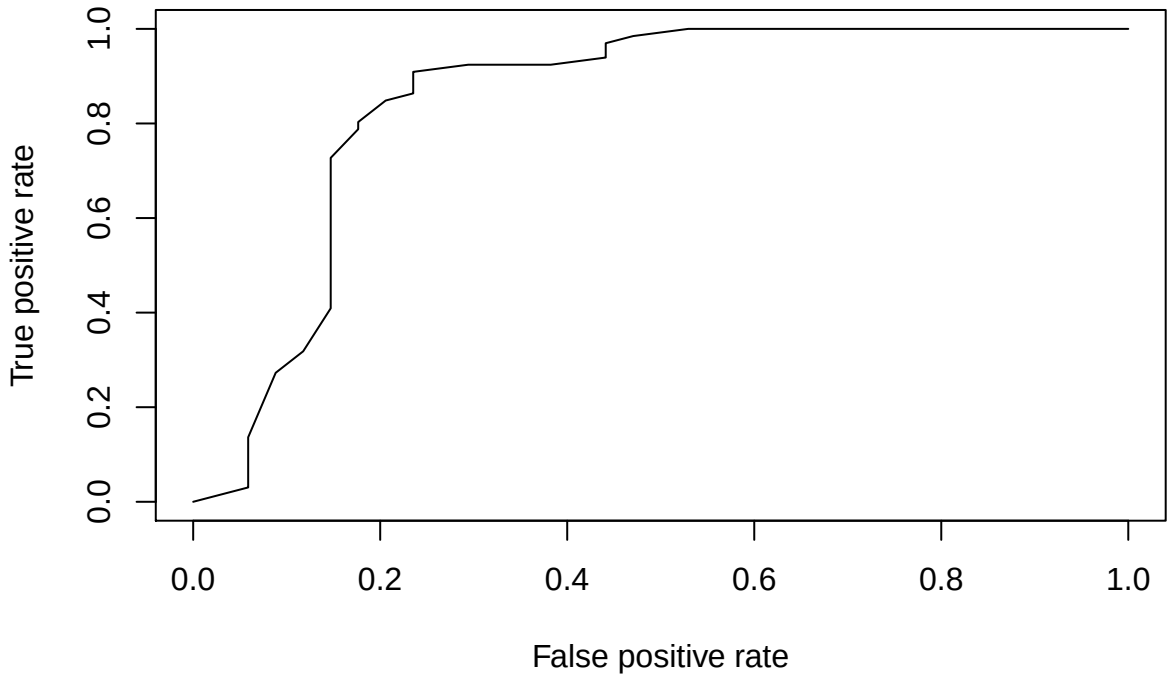
22.8 Beyond changes in the mean

In the realm of single-cell analyses, differential expression is no longer restricted to changes in the mean. As we saw in the previous chapter, it is possible to change the shape of the distribution without changing its mean.

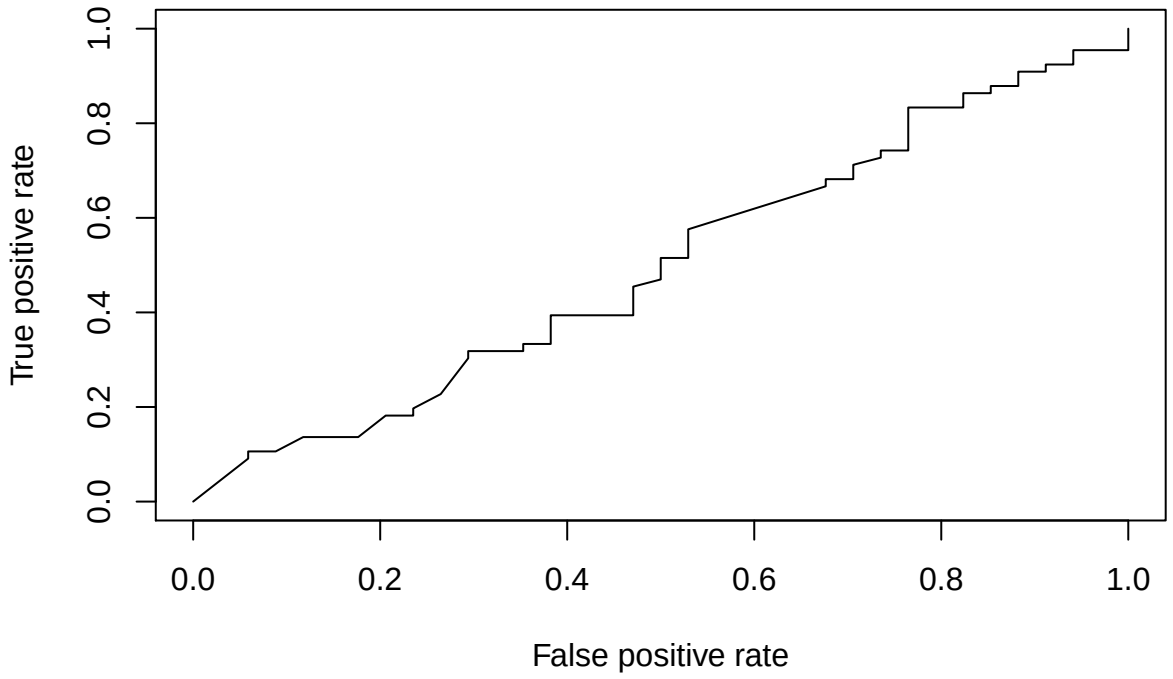
Exercise: Repeat the analysis above, but this time only change the variance, but not the mean. To generate the synthetic data, use the following command:

```
nGenes <- 1e2
nCells <- 50
mult <- 2^(rnorm(nGenes, 0, 2))
synData <- scRNA.seq.funcs::GeneratePoiBetaSamples(
    ks = 10^(rnorm(nGenes, 3, .5)),
    as = 10^(rnorm(nGenes, -1, .5)),
    bs = 10^(rnorm(nGenes, 0, .5)),
    mult,
    nGenes,
    nCells,
    meanFixed = TRUE
)
g <- synData$sample1
g2 <- synData$sample2
```

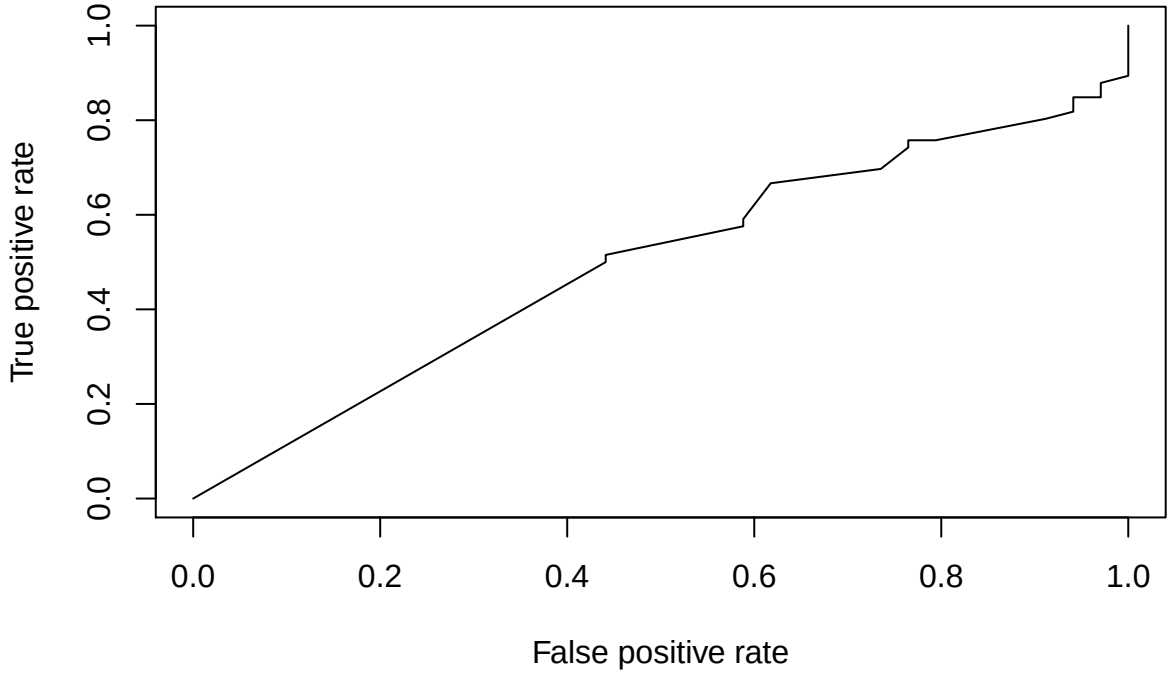
Our answer:




```
## [1] 0.8460339
```




```
## [1] 0.4982175
```




```
## [1] 0.4993316
```

Which method performs the best? Why do some methods perform differently for this scenario? What are the properties of the genes that are identified as DE?

Chapter 23

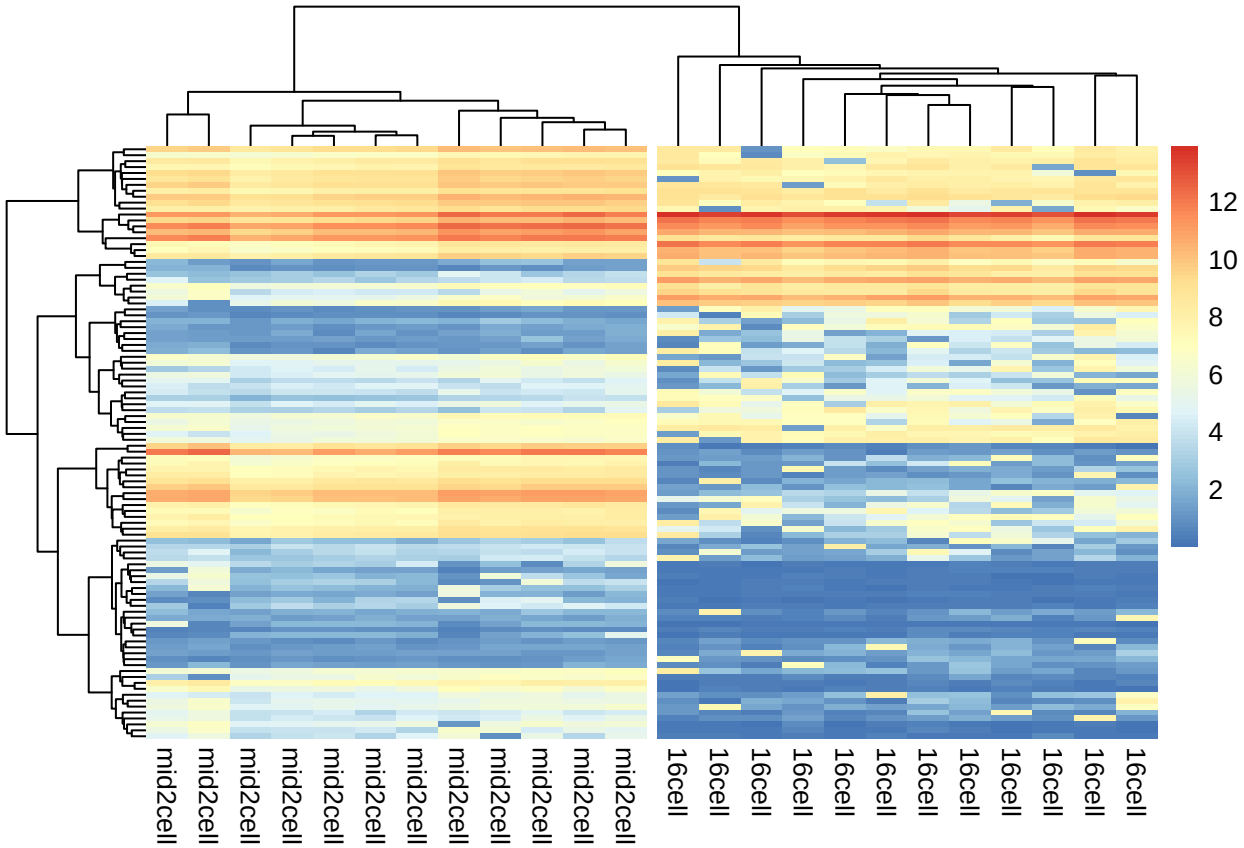
DE in a real dataset

```
library(scRNA.seq.funcs)
library(DESeq2)
library(scde)
library(ROCR)
library(limma)
set.seed(1)
```

23.1 Introduction

The main advantage of using synthetic data is that we have full control over all aspects of the data, and this facilitates the interpretation of the results. However, the transcriptional bursting model is unable to capture the full complexity of a real scRNA-seq dataset. Next, we are going to analyze the difference between the transcriptomes of the two stages of the mouse embryo development (*mid2cell* and *16cell*) again from the `deng`. However, this time we need to use the raw counts instead of the normalized counts. This is a major requirement for the DE analysis tools like `DESeq2`, `edgeR`, `scde` etc.

```
deng <- readRDS("deng/deng_raw.rds")
# select cells at different stages
deng <- deng[ , c(which(colnames(deng) == "mid2cell"),
                  which(colnames(deng) == "16cell")[1:12])]
# keep those genes that are expressed in at least 6 cells
deng <- deng[rowSums(deng > 0) > 5, ]
pheatmap::pheatmap(
    log2(deng + 1),
    cutree_cols = 2,
    kmeans_k = 100,
    show_rownames = FALSE
)
```



As you can see, the cells cluster well by their developmental stage.

We can now use the same methods as before to obtain a list of differentially expressed genes.

Because `scde` is very slow here we will only use a subset of genes. You should not do that with your real dataset, though. Here we do it just for demostration purposes:

```
deng <- deng[sample(1:nrow(deng), 1000), ]
```

23.2 KS-test

```
pVals <- rep(1, nrow(deng))
for (i in 1:nrow(deng)) {
    res <- ks.test(
        deng[i, 1:12],
        deng[i, 13:24]
    )
    pVals[i] <- res$p.value
}
# Bonferroni correction
pVals <- p.adjust(pVals, method = "bonferroni")
```

23.3 DESeq2

```
cond <- factor(
    c(
        rep("mid2cell", 12),
        rep("16cell", 12)
    )
)
colnames(deng) <- 1:ncol(deng)
dds <- DESeq2::DESeqDataSetFromMatrix(
    deng,
    colData = DataFrame(cond),
    design = ~ cond
)
dds <- DESeq2::DESeq(dds)
resDESeq <- DESeq2::results(dds)
pValsDESeq <- resDESeq$padj
```

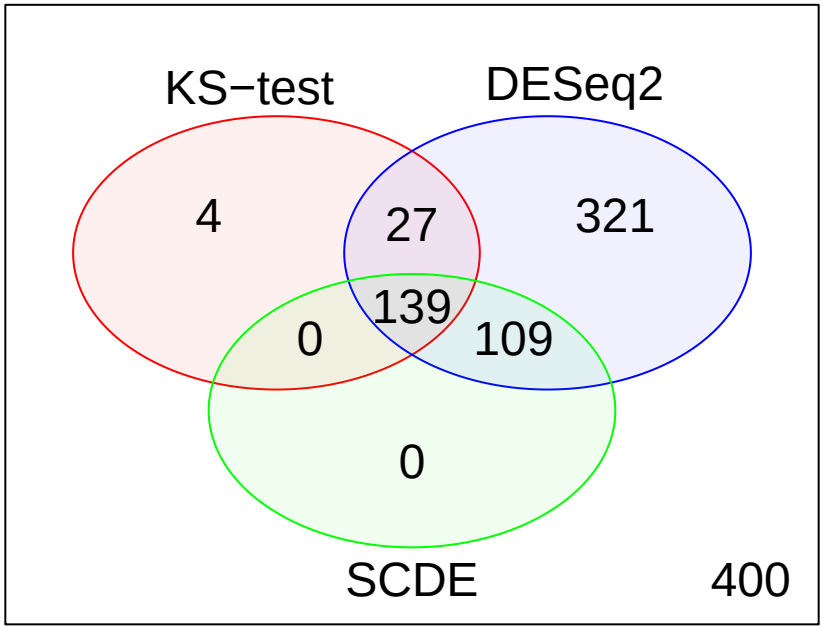
23.4 SCDE

```
cnts <- apply(
    deng,
    2,
    function(x) {
        storage.mode(x) <- 'integer'
        return(x)
    }
)
names(cond) <- 1:length(cond)
colnames(cnts) <- 1:length(cond)
o.ifm <- scde::scde.error.models(
    counts = cnts,
    groups = cond,
    n.cores = 1,
    threshold.segmentation = TRUE,
    save.crossfit.plots = FALSE,
    save.model.plots = FALSE,
    verbose = 0,
    min.size.entries = 2
)
priors <- scde::scde.expression.prior(
    models = o.ifm,
    counts = cnts,
    length.out = 400,
    show.plot = FALSE
)
resSCDE <- scde::scde.expression.difference(
    o.ifm,
    cnts,
    priors,
    groups = cond,
```

```
    n.randomizations = 100,
    n.cores = 1,
    verbose = 0
)
# Convert Z-scores into 2-tailed p-values
pValsSCDE <- pnorm(abs(resSCDE$cZ), lower.tail = FALSE) * 2
pValsSCDE <- p.adjust(pValsSCDE, method = "bonferroni")
```

23.5 Comparison of the methods

```
vennDiagram(
    vennCounts(
        cbind(
            pVals < 0.05,
            pValsDESeq < 0.05,
            pValsSCDE < 0.05
        )
    ),
    names = c("KS-test", "DESeq2", "SCDE"),
    circle.col = c("red", "blue", "green")
)
```



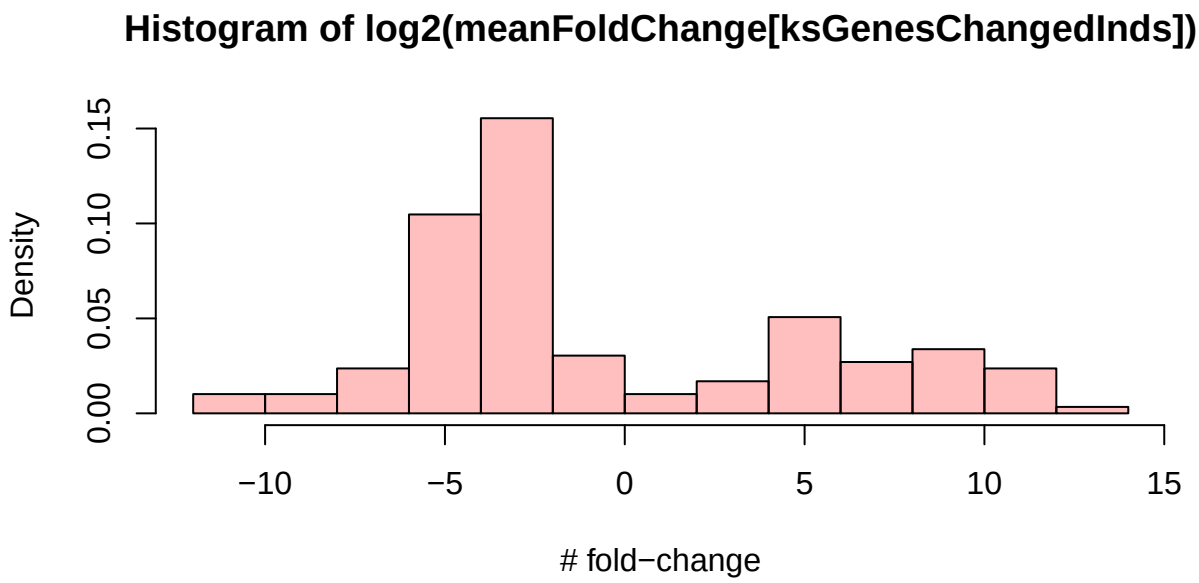
Exercise 1: How does this Venn diagram correspond to what you would expect based on the synthetic data?

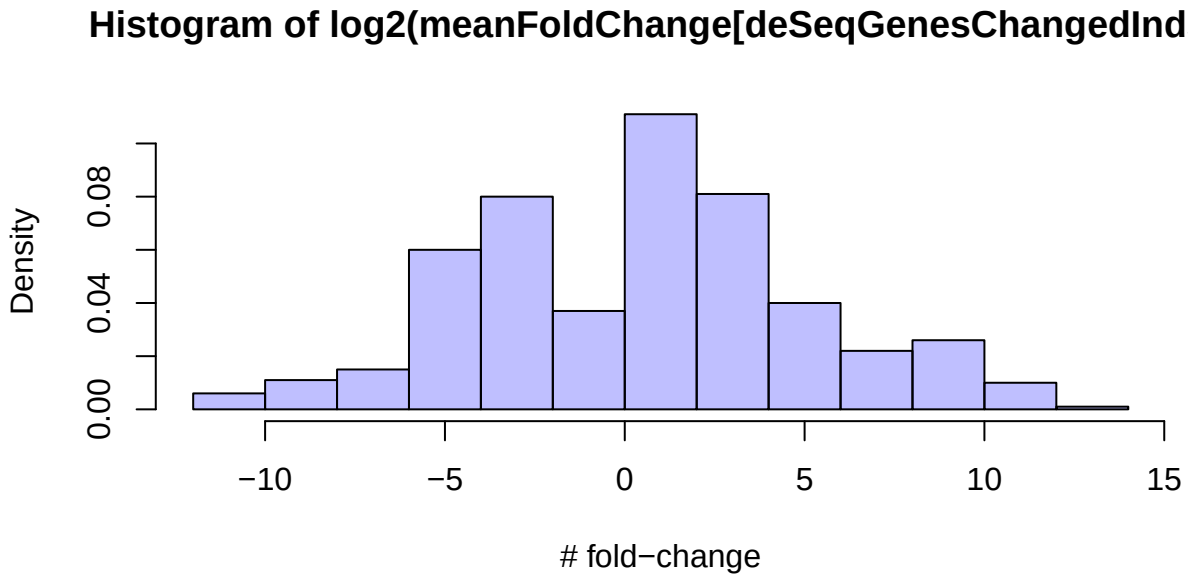
23.6 Visualisation

To further characterize the list of genes, we can calculate the average fold-changes and compare the ones that were called as differentially expressed to the ones that were not.

```
group1 <- deng[, 1:12] # mid2cell stage
group2 <- deng[, 13:24] # 16cell stage
ksGenesChangedInds <- which(pVals<.05)
deSeqGenesChangedInds <- which(pValsDESeq<.05)
scdeGenesChangedInds <- which(pValsSCDE<.05)
ksGenesNotChangedInds <- which(pVals>=.05)
deSeqGenesNotChangedInds <- which(pValsDESeq>=.05)
scdeGenesNotChangedInds <- which(pValsSCDE>=.05)
meanFoldChange <- rowSums(group1)/rowSums(group2)

par(mfrow=c(2,1))
hist(log2(meanFoldChange[ksGenesChangedInds]),
     freq = FALSE,
     xlab = "# fold-change",
     col = rgb(1, 0, 0, 1/4))
hist(log2(meanFoldChange[deSeqGenesChangedInds]),
     freq = FALSE,
     xlab = "# fold-change",
     col = rgb(0, 0, 1, 1/4))
```





Exercise 2: Create the histogram of fold-changes for SCDE. Compare the estimated fold-changes between the different methods? What do the genes where they differ have in common?

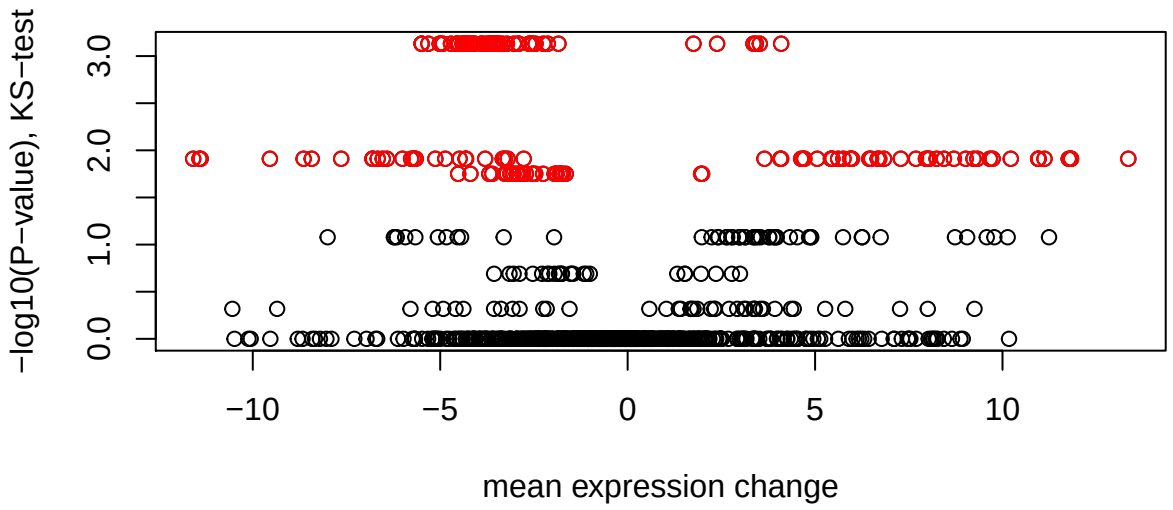
23.6.1 Volcano plots

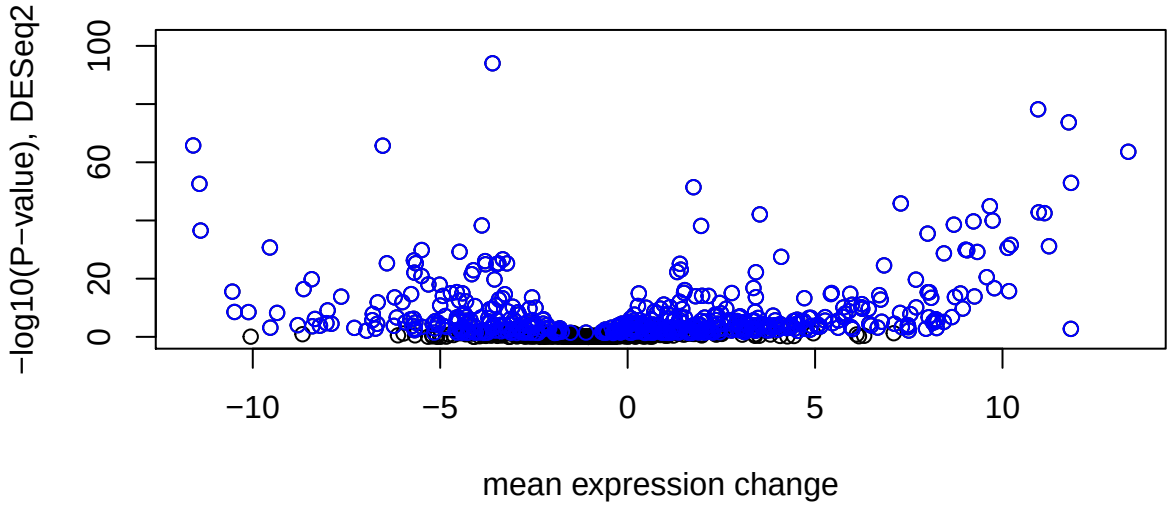
A popular method for illustrating the difference between two conditions is the volcano plot which compares the magnitude of the change and the significance.

```
par(mfrow=c(2,1))
plot(log2(meanFoldChange),
     -log10(pVals),
     xlab = "mean expression change",
     ylab = "-log10(P-value), KS-test")
points(log2(meanFoldChange[ksGenesChangedInds]),
```

```
       -log10(pVals[ksGenesChangedInds]),
       col = "red")

plot(log2(meanFoldChange),
     -log10(pValsDESeq),
     xlab = "mean expression change",
     ylab = "-log10(P-value), DESeq2")
points(log2(meanFoldChange[deSeqGenesChangedInds]),
       -log10(pValsDESeq[deSeqGenesChangedInds]),
       col = "blue")
```



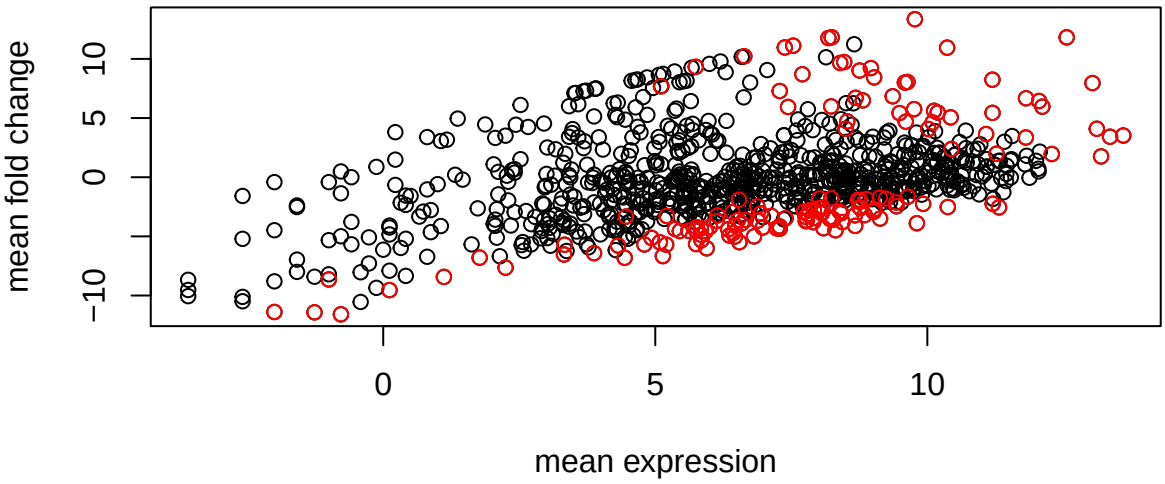


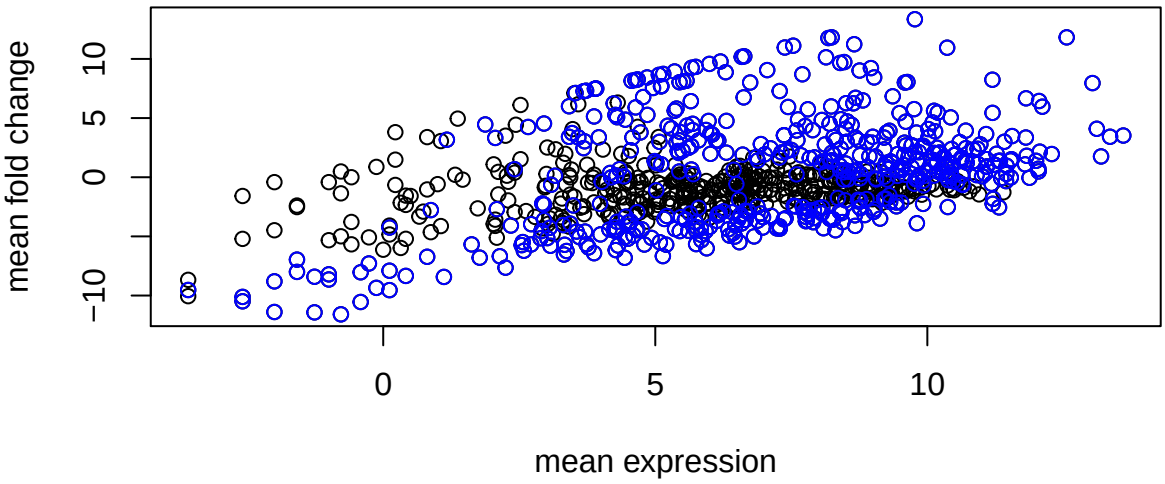
23.6.2 MA-plot

Another popular method for illustrating the difference between two conditions is the MA-plot, in which the data has been transformed onto the M (log ratios) and A (mean average) scale:

```
par(mfrow=c(2,1))
plot(log2(rowMeans(group1)),
     log2(meanFoldChange),
     ylab = "mean fold change",
     xlab = "mean expression")
points(log2(rowMeans(group1[ksGenesChangedInds,])),
       log2(meanFoldChange[ksGenesChangedInds]),
       col = "red")

plot(log2(rowMeans(group1)),
     log2(meanFoldChange),
     ylab = "mean fold change",
     xlab = "mean expression")
points(log2(rowMeans(group1[deSeqGenesChangedInds,])),
       log2(meanFoldChange[deSeqGenesChangedInds]),
       col = "blue")
```



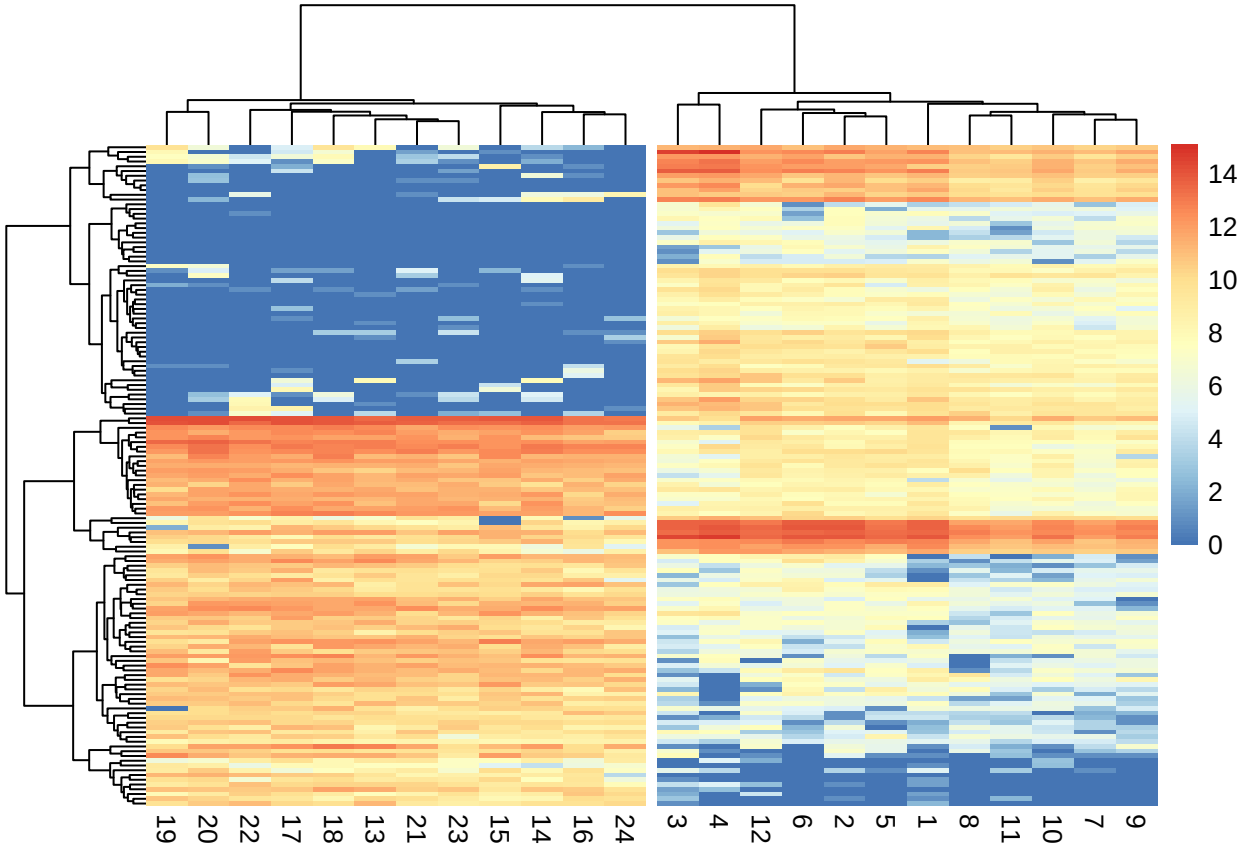


Exercise 3: The volcano and MA-plots for the SCDE are missing - can you generate them? Compare to the synthetic data, what do they tell you about the properties of the genes that have changed?

23.6.3 Heatmap of DE genes

Finally, we can plot heatmaps of the genes that were called as DE by the intersection of the three.

```
allChangedInds <- intersect(which(pValsDESeq<.05),
                            intersect(which(pValsSCDE<.05),
                                      which(pVals<.05)))
pheatmap::pheatmap(log2(1 + deng[allChangedInds,]),
                   cutree_cols = 2,
                   show_rownames = FALSE)
```



Exercise 4: Create heatmaps for the genes that were detected by at least 2/3 methods.

Chapter 24

"Ideal" scRNAseq pipeline (as of Oct 2016)

24.1 Experimental Design

- Avoid confounding biological and batch effects (Figure 24.1)
  - Multiple conditions should be captured on the same chip if possible
  - Perform multiple replicates of each condition where replicates of different conditions should be performed together if possible
  - Statistics cannot correct a completely confounded experiment!
- Unique molecular identifiers
  - Greatly reduce noise in data
  - May reduce gene detection rates (unclear if it is UMIs or other protocol differences)
  - Use longer UMIs (~10bp)
  - Correct for sequencing errors in UMIs using UMI-tools
- Spike-ins
  - Useful for quality control
  - Can be used to approximate cell-size/RNA content (if relevant to biological question)
  - Often exhibit higher noise than endogenous genes (pipetting errors, mixture quality)
  - Requires more sequencing to get enough endogenous reads per cell
- Cell number vs Read depth
  - Gene detection plateaus starting from 1 million reads per cell
  - Transcription factor detection (regulatory networks) require high read depth and most sensitive protocols (i.e. Fluidigm C1)
  - Cell clustering & cell-type identification benefits from large number of cells and doesn't requireas high sequencing depth (~100,000 reads per cell).

24.2 Processing Reads

- Read QC & Trimming
  - FASTQC, cutadapt
- Mapping
  - Small datasets or UMI datasets: align to genome/transcriptome using STAR
  - Large datasets: pseudo-alignment with Salmon
- Quantification
  - Small dataset, no UMIs : Cufflinks or featureCounts

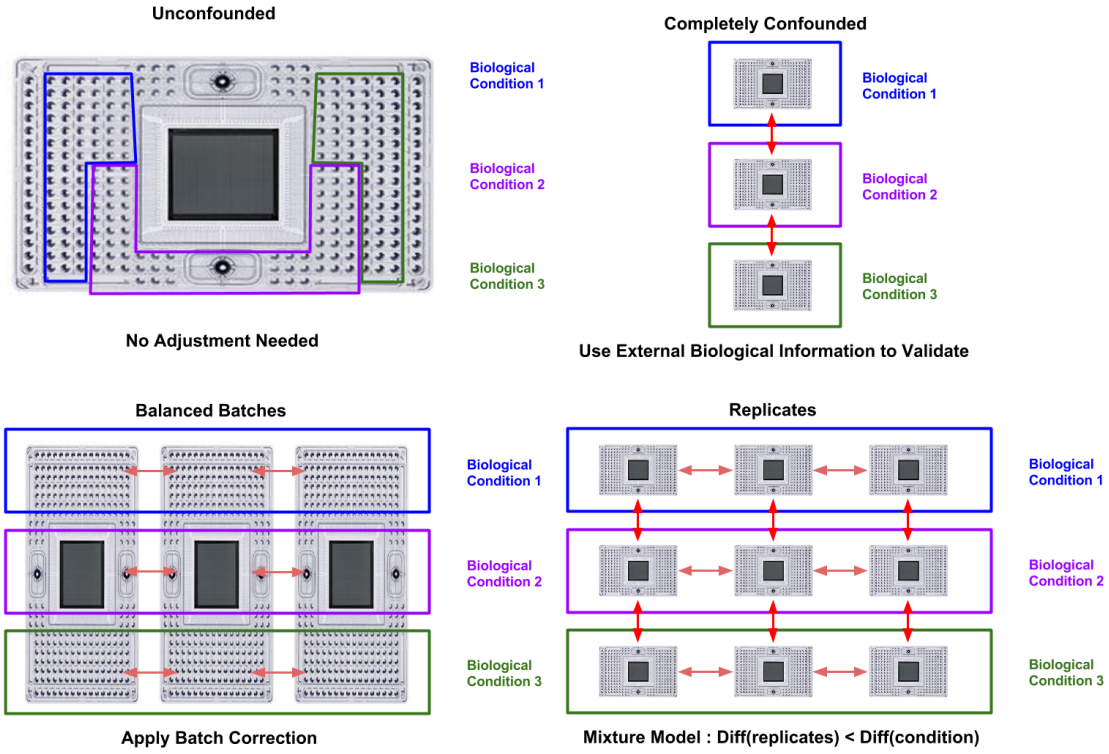


Figure 24.1: Appropriate approaches to batch effects in scRNASeq. Red arrows indicate batch effects which are (pale) or are not (vibrant) correctable through batch-correction.

  - Large datasets, no UMIs: Salmon
  - UMI dataset : UMI-tools' + featureCounts

24.3 Preparing Expression Matrix

- Cell QC
  - scater
  - consider: mtRNA, rRNA, spike-ins (if available), number of detected genes per cell, total reads/molecules per cell
- Library Size Normalization
  - scran
- Batch correction (if appropriate)
  - RUVs

24.4 Biological Interpretation

- Feature Selection
  - M3Drop
- Clustering and Marker Gene Identification
  - SC3
- Pseudotime
  - distinct timepoints: TSCAN
  - continuous data: destiny
- Differential Expression
  - Small number of cells and few groups : scde
  - Replicates with batch effects : mixture/linear models
  - Large datasets: Kruskal-Wallis test (all groups at once), or Kolmogorov-Smirnov (KS)-test (compare 2-groups at a time).

Chapter 25

Advanced exercises

For the final part of the course we would like you to work on more open ended problems. The goal is to carry out the type of analyses that you would be doing for an actual research project.

Participants who have their own dataset that they are interested in should feel free to work with them.

For other participants we recommend downloading a dataset from the conquer resource (consistent quantification of external rna-seq data). conquer uses Salmon to quantify the transcript abundances in a given sample. For a given organism, the fasta files containing cDNA and ncRNA sequences from Ensembl are complemented with ERCC spike-in sequences, and a Salmon quasi-mapping index is built for the entire catalog. Then Salmon is run to estimate the abundance of each transcript. The abundances estimated by Salmon are summarised and provided to the user in the form of a `MultiAssayExperiment` object. This object can be downloaded via the buttons in the *MultiAssayExperiment* column. The provided `MultiAssayExperiment` object contains two "experiments", corresponding to the gene-level and transcript-level values.

The gene-level experiment contains four "assays":

- TPM
- count
- count_lstpm (count-scale length-scaled TPMs)
- avetxlength (the average transcript length, which can be used as offsets in count models based on the count assay, see here).

The transcript-level experiment contains three "assays":

- TPM
- count
- efflength (the effective length estimated by Salmon)

The `MultiAssayExperiment` also contains the phenotypic data (in the `pData` slot), as well as some metadata for the data set (the genome, the organism and the Salmon index that was used for the quantification).

Here we will show you how to create an SCESet from a `MultiAssayExperiment` object. For example, if you download `Shalek2013` dataset you will be able to create an SCESet using the following code:

```
library(MultiAssayExperiment)
library(SummarizedExperiment)
library(scater)
d <- readRDS("~/Desktop/GSE41265.rds")
cts <- assays(experiments(d)[["gene"]])[["count_lstpm"]]
tpms <- assays(experiments(d)[["gene"]])[["TPM"]]
```

```
phn <- pData(d)
sceset <- newSCESet(
    countData = cts,
    tpmData = tpms,
    phenoData = new("AnnotatedDataFrame", data = as.data.frame(phn))
)
```

You can also see that several different QC metrics have already been pre-calculated on the conquer website.

Here are some suggestions for questions that you can explore:

- There are two mESC datasets from different labs (i.e. Xue and Kumar). Can you merge them and remove the batch effects?
- Clustering and pseudotime analysis look for different patterns among cells. How might you tell which is more appropriate for your dataset?
- One of the main challenging in hard clustering is to identify the appropriate value for k. Can you use one or more of the clustering tools to explore the different hierarchies available? What are good mathematical and/or biological criteria for determining k?
- The choice of normalization strategy matters, but how do you determine which is the best method? Explore the effect of different normalizations on downstream analyses.
- scRNA-seq datasets are high-dimensional and since most dimensions (ie genes) are not informative. Consequently, dimensionality reduction and feature selection are important when analyzing and visualizing the data. Consider the effect of different feature selection methods and dimensionality reduction on clustering and/or pseudotime inference.
- One of the main challenges after clustering cells is to interpret the biological relevance of the subpopulations. One approach is to identify gene ontology terms that are enriched for the set of marker genes. Identify marker genes (e.g. using SC3 or M3Drop) and explore the ontology terms using gProfiler, WebGestalt or DAVID.
- Similarly, when ordering cells according to pseudotime we would like to understand what underlying cellular processes are changing over time. Identify a set of changing genes from the aligned cells and use ontology terms to characterize them.

Chapter 26

Resources

26.1 scRNA-seq protocols

- SMART-seq2
- CELL-seq
- Drop-seq
- UMI
- STRT-Seq

26.2 External RNA Control Consortium (ERCC)

ERCCs

26.3 scRNA-seq analysis tools

Extensive list of software packages (and the people developing these methods) for single-cell data analysis:

- awesome-single-cell

Tallulah Andrews' single cell processing scripts:

- scRNASeqPipeline

Bibliography

Deng, Q., Ramsköld, D., Reinius, B., and Sandberg, R. (2014). Single-cell RNA-seq reveals dynamic, random monoallelic gene expression in mammalian cells. *Science*, 343(6167):193–196.

Guo, M., Wang, H., Potter, S. S., Whitsett, J. A., and Xu, Y. (2015). SINCERA: A pipeline for Single-Cell RNA-Seq profiling analysis. *PLoS Comput. Biol.*, 11(11):e1004575.

Handley, A., Schauer, T., Ladurner, A. G., and Margulies, C. E. (2015). Designing Cell-Type-Specific genome-wide experiments. *Mol. Cell*, 58(4):621–631.

Hashimshony, T., Wagner, F., Sher, N., and Yanai, I. (2012). CEL-Seq: single-cell RNA-Seq by multiplexed linear amplification. *Cell Rep.*, 2(3):666–673.

Jiang, L., Schlesinger, F., Davis, C. A., Zhang, Y., Li, R., Salit, M., Gingeras, T. R., and Oliver, B. (2011). Synthetic spike-in standards for RNA-seq experiments. *Genome Res.*, 21(9):1543–1551.

Kharchenko, P. V., Silberstein, L., and Scadden, D. T. (2014). Bayesian approach to single-cell differential expression analysis. *Nat. Methods*, 11(7):740–742.

Kiselev, V. Y., Kirschner, K., Schaub, M. T., Andrews, T., Chandra, T., Natarajan, K. N., Reik, W., Barahona, M., Green, A. R., and Hemberg, M. (2016). SC3 - consensus clustering of single-cell RNA-Seq data.

Kivioja, T., Vähärautio, A., Karlsson, K., Bonke, M., Enge, M., Linnarsson, S., and Taipale, J. (2012). Counting absolute numbers of molecules using unique molecular identifiers. *Nat. Methods*, 9(1):72–74.

Kolodziejczyk, A. A., Kim, J. K., Svensson, V., Marioni, J. C., and Teichmann, S. A. (2015). The technology and biology of Single-Cell RNA sequencing. *Mol. Cell*, 58(4):610–620.

Levine, J. H., Simonds, E. F., Bendall, S. C., Davis, K. L., Amir, E.-A. D., Tadmor, M. D., Litvin, O., Fienberg, H. G., Jager, A., Zunder, E. R., Finck, R., Gedman, A. L., Radtke, I., Downing, J. R., Pe'er, D., and Nolan, G. P. (2015). Data-Driven phenotypic dissection of AML reveals progenitor-like cells that correlate with prognosis. *Cell*, 162(1):184–197.

Macosko, E. Z., Basu, A., Satija, R., Nemesh, J., Shekhar, K., Goldman, M., Tirosh, I., Bialas, A. R., Kamitaki, N., Martersteck, E. M., Trombetta, J. J., Weitz, D. A., Sanes, J. R., Shalek, A. K., Regev, A., and McCarroll, S. A. (2015). Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, 161(5):1202–1214.

Picelli, S., Björklund, A. K., Faridani, O. R., Sagasser, S., Winberg, G., and Sandberg, R. (2013). Smart-seq2 for sensitive full-length transcriptome profiling in single cells. *Nat. Methods*, 10(11):1096–1098.

Saliba, A.-E., Westermann, A. J., Gorski, S. A., and Vogel, J. (2014). Single-cell RNA-seq: advances and future challenges. *Nucleic Acids Res.*, 42(14):8845–8860.

Stegle, O., Teichmann, S. A., and Marioni, J. C. (2015). Computational and analytical challenges in single-cell transcriptomics. *Nat. Rev. Genet.*, 16(3):133–145.

Tang, F., Barbacioru, C., Wang, Y., Nordman, E., Lee, C., Xu, N., Wang, X., Bodeau, J., Tuch, B. B., Siddiqui, A., Lao, K., and Surani, M. A. (2009). mRNA-Seq whole-transcriptome analysis of a single cell. *Nat. Methods*, 6(5):377–382.

Tung, P.-Y., Blischak, J. D., Hsiao, C., Knowles, D. A., Burnett, J. E., Pritchard, J. K., and Gilad, Y. (2016). Batch effects and the effective design of single-cell gene expression studies.

Žurauskienė, J. and Yau, C. (2016). pcareduce: hierarchical clustering of single cell transcriptional profiles. *BMC Bioinformatics*, 17:140.

Xu, C. and Su, Z. (2015). Identification of cell types from single-cell transcriptomes using a novel clustering method. *Bioinformatics*.