Analysis of single cell RNA-seq data

Vladimir Kiselev, Tallulah Andrews, Davis McCarthy and Martin Hemberg

2017-05-25

Contents

Chapter 1

About the course

Today it is 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methods are required and some of the underlying assumptions for the methods developed for bulk RNA-seq experiments are no longer valid.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 discuss some of the questions that can be addressed using scRNA-seq as well as the available computational and statistical methods avialable. 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. The course is taught twice per year and the material here is updated prior to each event.

The number of computational tools is increasing rapidly and we are doing our best to keep up to date with what is available. One of the main constraints for this course is that we would like to use tools that are implemented in R and that run reasonably fast. Moreover, we will also confess to being somewhat biased towards methods that have been developed either by us or by our friends and colleagues.

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 Docker image

The course can be reproduced without any package installation by running the course docker image which contains all the required packages.

Make sure Docker is installed on your system. If not, please follow these instructions. To run the course docker image:

```
docker run -it quay.io/hemberg-group/scrna-seq-course:latest R
```

It will download the course docker image (may take some time) and start a new R session in a docker container with all packages installed and all data files available.

1.5 License

GPL-3

1.6 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.

1.7 Contact

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

Chapter 2

Technical requirements

2.1 R-based

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.

2.2 Docker image

If you do not want to install all the packages required for the course manually, you can run a course docker image which contains all the required packages.

Make sure Docker is installed on your system. If not, please follow these instructions. To run the course docker image:

```
docker run -it quay.io/hemberg-group/scrna-seq-course:latest R
```

It will download the course docker image (may take some time) and start a new R session in a docker container with all packages installed and all data files available.

2.3 Manual installation

If you are not using a docker image of the course, then to be able to run all code chunks of the course you need to clone or download the course GitHub repository and start an R session in the cloned folder. You will also need to install the following R packages (ordered by purposes):

2.3.1 General

devtools

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

BiocInstaller

```
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")
```

2.3.2 Plotting

pheatmap

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

limma

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

2.3.3 QC and normalisation

scater

```
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")
```

scran

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

RUVSeq

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

2.3.4 Clustering

pcaReduce

```
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

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

SEURAT

```
devtools::install_github('satijalab/seurat')
```

2.3.5 Dropouts

M3Drop

```
source("https://bioconductor.org/biocLite.R")
biocLite("M3Drop")
```

2.3.6 Pseudotime

TSCAN

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

monocle

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

destiny

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

SLICER

```
devtools::install_github('jw156605/SLICER')
```

2.3.7 Differential Expression

ROCR

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

edgeR

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

DESeq2

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

MAST

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

scde (optional)

```
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.

2.4 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 (**?**)
- 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 (**?**), CELL-seq (**?**) and Drop-seq (**?**)
- There are also commercial platforms available, including the Fluidigm C1 and the 10X Genomics Chromium
- 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 Workflow

Overall, experimental scRNA-seq protocols are similar to the methods used for bulk RNA-seq. We will be discussing some of the most common approaches in the next chapter.

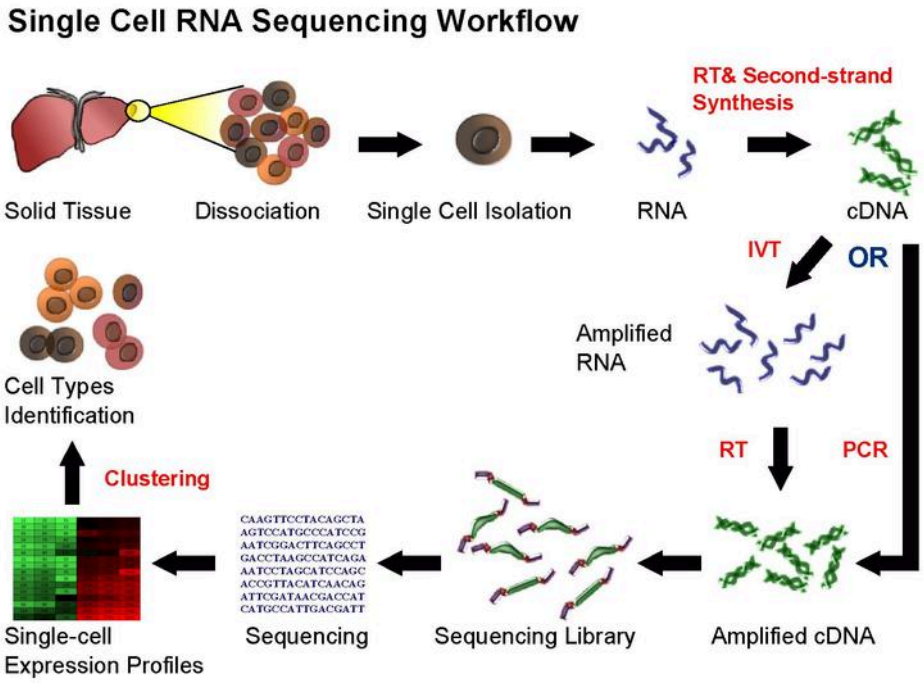


Figure 3.1: Single cell sequencing (taken from Wikipedia)

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 (**?**).

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 (**?**).

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. However, as we shall see in this course, it is possible to alleviate some of these issues through proper normalization and corrections.

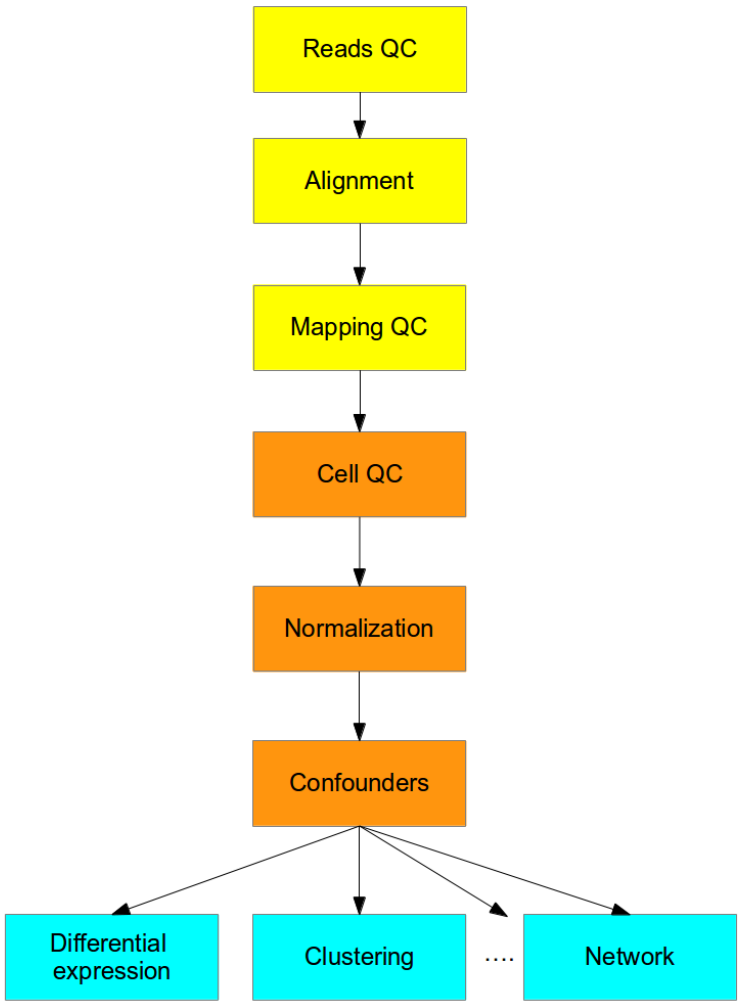


Figure 3.2: Flowchart of the scRNA-seq analysis

Chapter 4

Experimental methods

4.1 Overview of experimental methods for generating scRNA-seq data

Development of new methods and protocols for scRNA-seq is currently a very active area of research, and several protocols have been published over the last few years. An non-comprehensive list includes:

- CEL-seq
- CEL-seq2
- Drop-seq
- InDrop-seq
- MARS-seq
- SCRB-seq
- Seq-well
- Smart-seq
- Smart-seq2
- SMARTer
- STRT-seq

The methods can be categorized in different ways, but the two most important aspects are **quantification** and **capture**.

For quantification, there are two types, **full-length** and **tag-based**. The former tries to achieve a uniform read coverage of each transcript. By contrast, tag-based protocols only capture either the 5'- or 3'-end of each RNA. The choice of quantification method has important implications for what types of analyses the data can be used for. In theory, full-length protocols should provide an even coverage of transcripts, but as we shall see, there are often biases in the coverage. The main advantage of tag-based protocol is that they can be combined with unique molecular identifiers (UMIs) which can help improve the quantification (see chapter 6). On the other hand, being restricted to one end of the transcript may reduce the mappability and it also makes it harder to distinguish different isoforms (**?**).

The strategy used for capture determines throughput, how the cells can be selected as well as what kind of additional information besides the sequencing that can be obtained. The three most widely used options are **microwell-**, **microfluidic-** and **droplet-** based.

For well-based platforms, cells are isolated using for example pipette or laser capture and placed in microfluidic wells. One advantage of well-based methods is that they can be combined with fluorescent activated cell sorting (FACS), making it possible to select cells based on surface markers. This strategy is thus very useful for situations when one wants to isolate a specific subset of cells for sequencing. Another advantage is that one can take pictures of the cells. The image provides an additional modality and a particularly useful

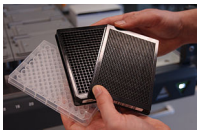


Figure 4.1: Image of microwell plates (image taken from Wikipedia)

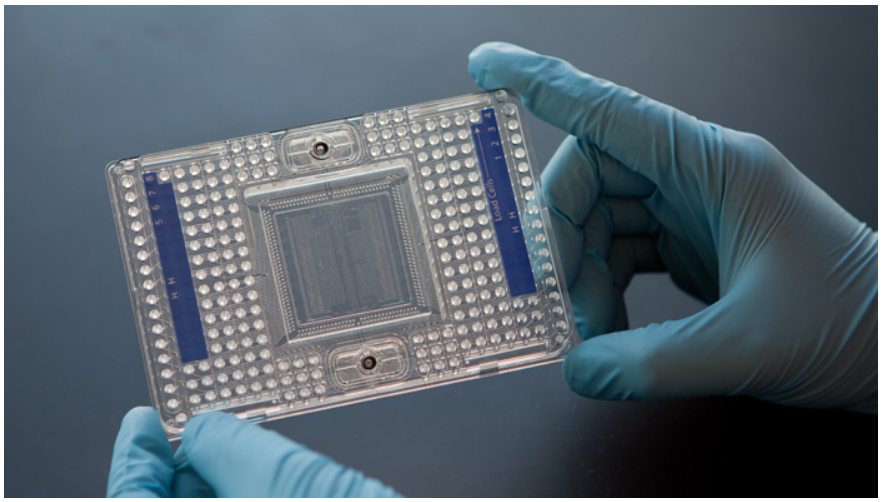


Figure 4.2: Image of a 96-well Fluidigm C1 chip (image taken from Fluidigm)

application is to identify wells containg damaged cells or doublets. The main drawback of these methods is that they are often low-throughput and the amount of work required per cell may be considerable.

Microfluidic platforms, such as Fluidigm's C1, provide a more integrated system for capturing cells and for carrying out the reactions necessary for the library preparations. Thus, they provide a higher throughput than microwell based platforms. Typically, only around 10% of cells are captured in a microfluidic platform and thus they are not appropriate if one is dealing with rare cell-types or very small amounts of input. Moreover, the chip is relatively expensive, but since reactions can be carried out in a smaller volume money can be saved on reagents.

The idea behind droplet based methods is to encapsulate each individual cell inside a nanoliter droplet together with a bead. The bead is loaded with the enzymes required to construct the library. In particular, each bead contains a unique barcode which is attached to all of the reads originating from that cell. Thus, all of the droplets can be pooled, sequenced together and the reads can subsequently be assigned to the cell of origin based on the barcodes. Droplet platforms typically have the highest throughput since the library preparation costs are on the order of .05 USD/cell. Instead, sequencing costs often become the limiting factor and a typical experiment the coverage is low with only a few thousand different transcripts detected (**?**).

4.2 What platform to use for my experiment?

The most suitable platform depends on the biological question at hand. For example, if one is interested in characterizing the composition of a tissue, then a droplet-based method which will allow a very large number of cells to be captured is likely to be the most appropriate. On the other hand, if one is interesting in characterizing a rare cell-population for which there is a known surface marker, then it is probably best to enrich using FACS and then sequence a smaller number of cells.

Clearly, full-length transcript quantification will be more appropriate if one is interested in studying different

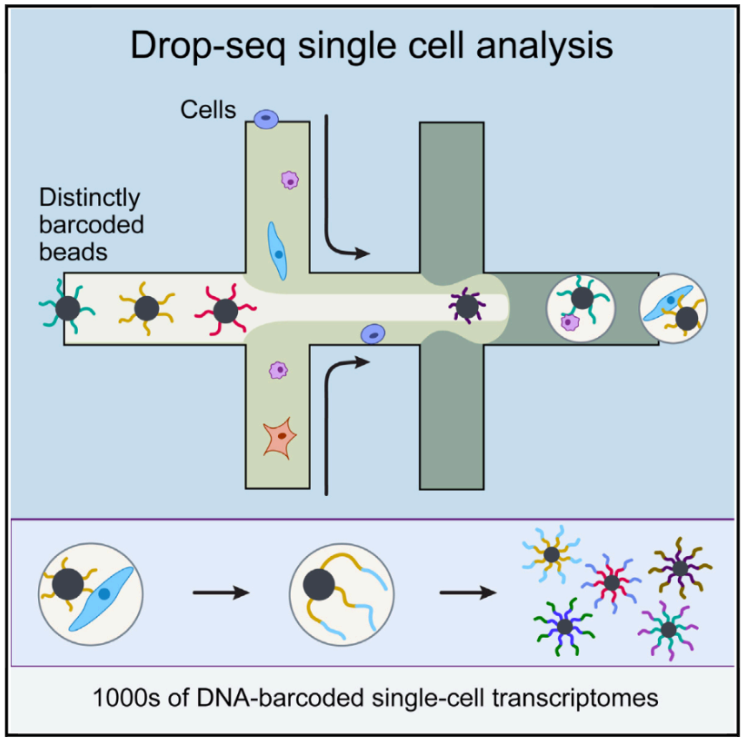


Figure 4.3: Schematic overview of the drop-seq method (Image taken from Macosko et al)

isoforms since tagged protocols are much more limited. By contrast, UMIs can only be used with tagged protocols and they can facilitate gene-level quantification.

Two recent studies from the Enard group (**?**) and the Teichmann group (**?**) have compared several different protocols. In their study, Ziegenhain et al compared five different protocols on the same sample of mouse embryonic stem cells (mESCs). By controlling for the number of cells as well as the sequencing depth, the authors were able to directly compare the sensitivity, noise-levels and costs of the different protocols. One example of their conclusions is illustrated in the figure below which shows the number of genes detected (for a given detection threshold) for the different methods. As you can see, there is almost a two-fold difference between drop-seq and Smart-seq2, suggesting that the choice of protocol can have a major impact on the study

Svensson et al take a different approach by using synthetic transcripts (spike-ins, more about these later) with known concentrations to measure the accuracy and sensitivity of different protocols. Comparing a wide range of studies, they also reported substantial differences between the protocols.

As protocols are developed and computational methods for quantifying the technical noise are improved, it is likely that future studies will help us gain further insights regarding the strengths of the different methods. These comparative studies are helpful not only for helping researchers decide which protocol to use, but also for developing new methods as the benchmarking makes it possible to determine what strategies are the most useful ones.

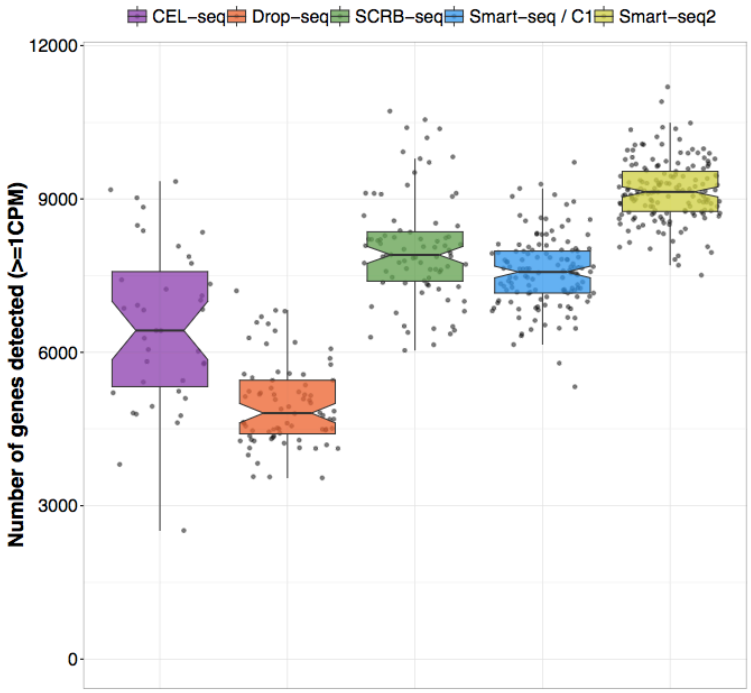


Figure 4.4: Enard group study

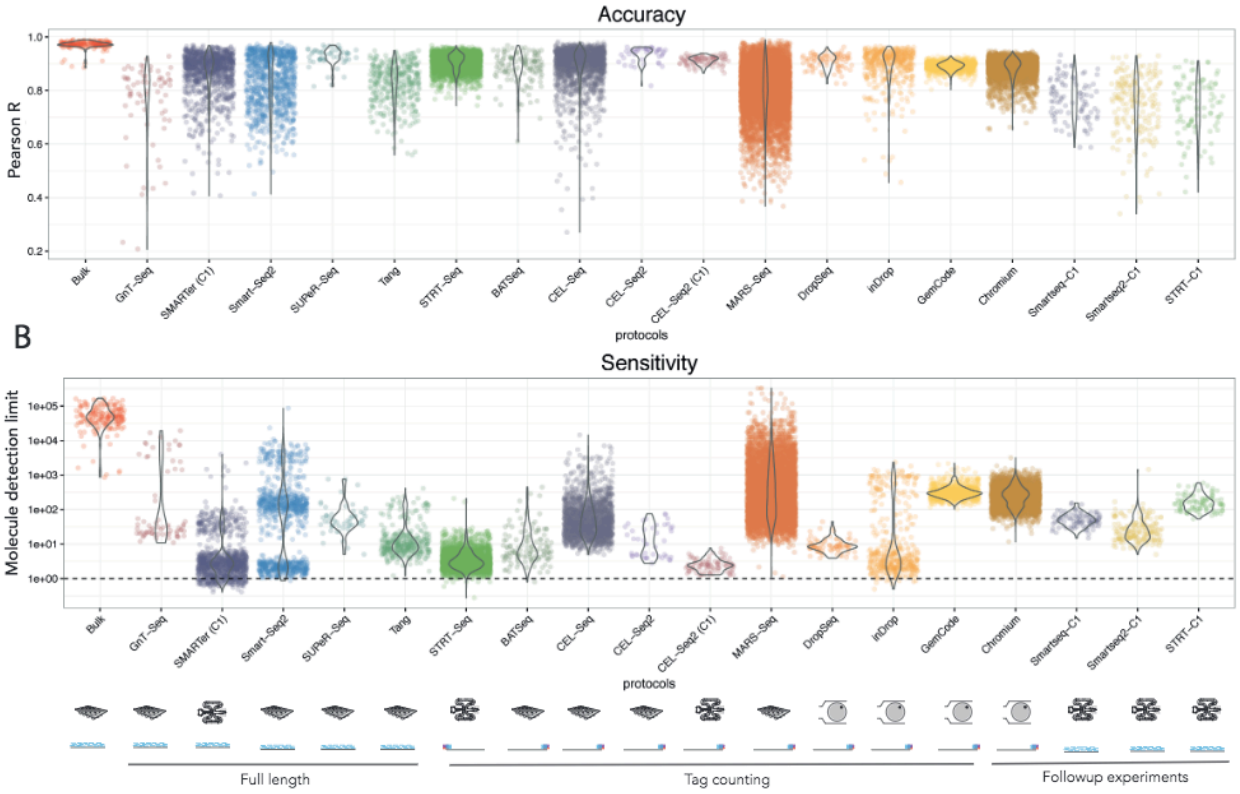


Figure 4.5: Teichmann group study

Chapter 5

Construction of expression matrix

Many analyses of scRNA-seq data take as their starting point an **expression matrix**. By convention, the each row of the expression matrix represents a gene and each column represents a cell (although some authors use the transpose). Each entry represents the expression level of a particular gene in a given cell. The units by which the expression is meassured depends on the protocol and the normalization strategy used.

5.1 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.

5.2 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. For drop-seq based datasets with tens- or hundreds of thousands of reads pseudoaligners become more appealing since their run-time can be several orders of magnitude less than traditional aligners.

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
```

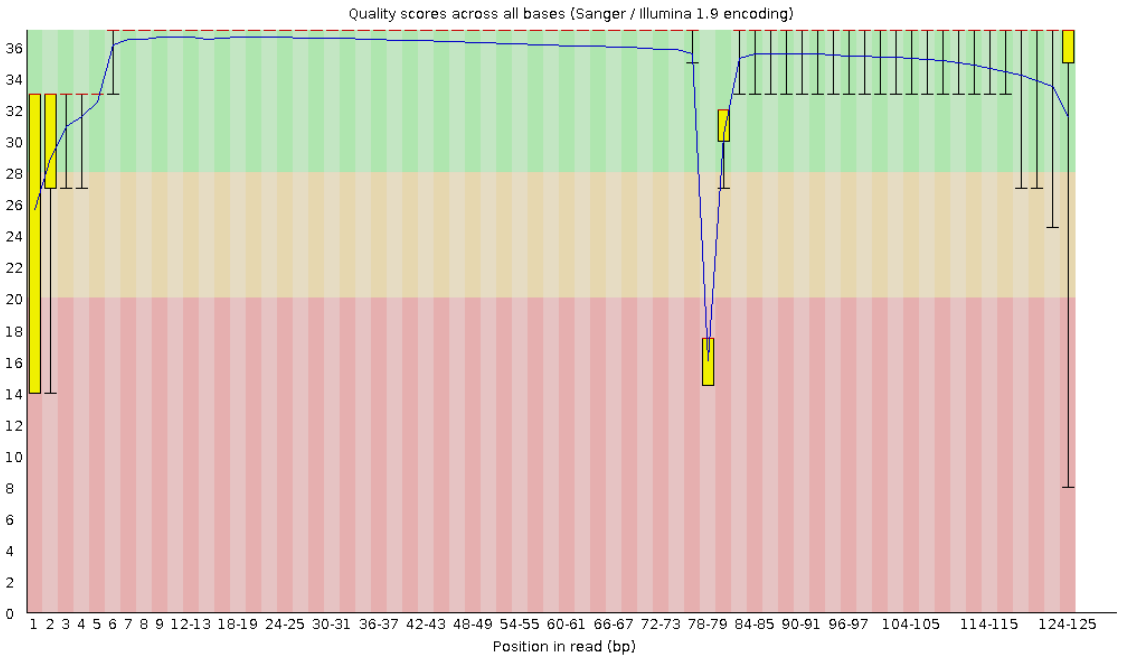


Figure 5.1: Example of FastQC output

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, 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.

An example of how to quantify expression using Salmon is

```
$<path_to_Salmon>/salmon quant -i salmon_transcript_index -1 reads1.fq.gz -2 reads2.fq.gz -p #threads -
```

Note Salmon produces estimated read counts and estimated transcripts per million (tpm) in our experience the latter over corrects the expression of long genes for scRNASeq, thus we recommend using read counts.

5.3 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.

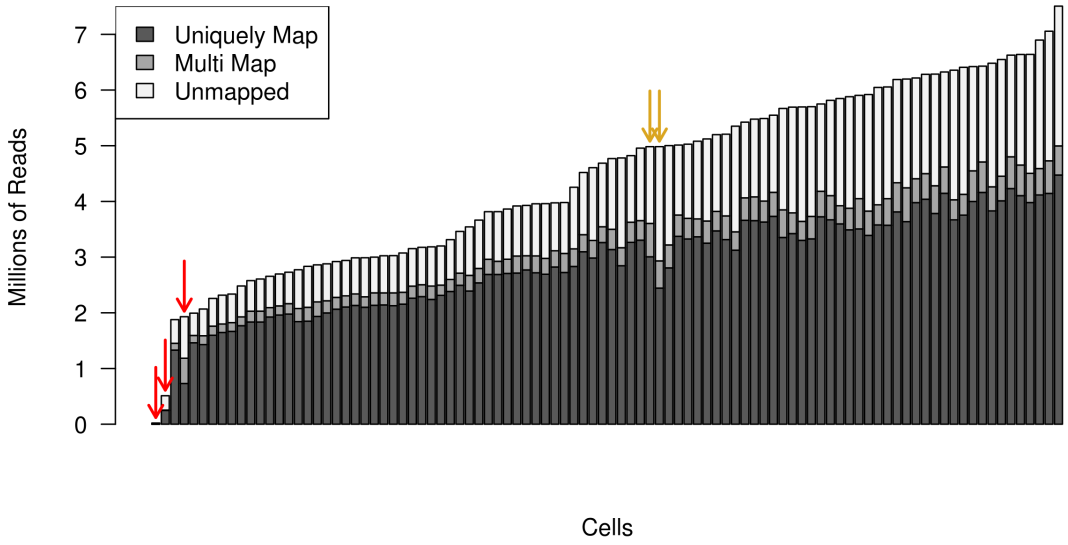


Figure 5.2: Example of the total number of reads mapped to each cell.

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

5.5 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.

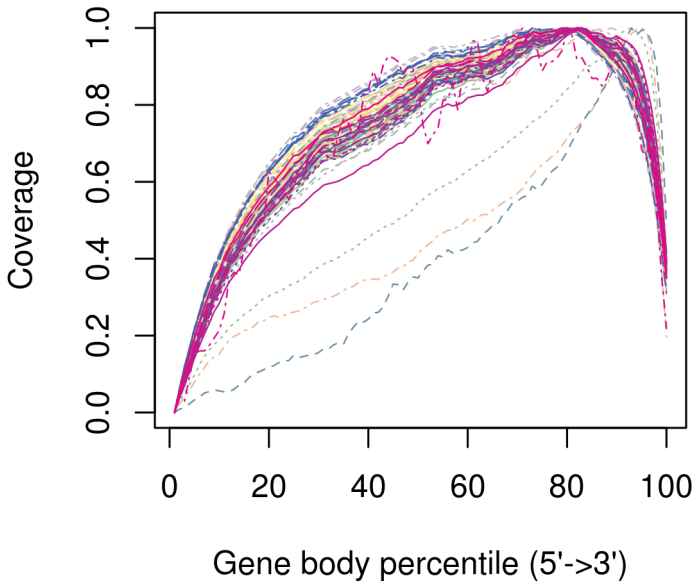


Figure 5.3: Example of the 3' bias in the read coverage.

Chapter 6

Unique Molecular Identifiers (UMIs)

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

6.1 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).

6.2 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 **??**). 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:

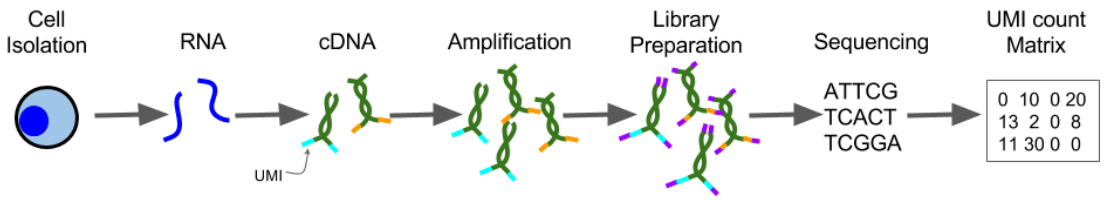


Figure 6.1: UMI sequencing protocol

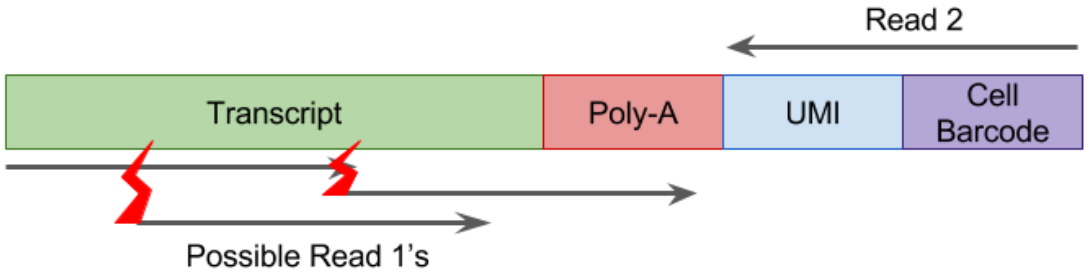


Figure 6.2: UMI sequencing reads, red lightning bolts represent different fragmentation locations

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.

6.3 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.

6.4 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) to estimate the true number of molecules M:

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

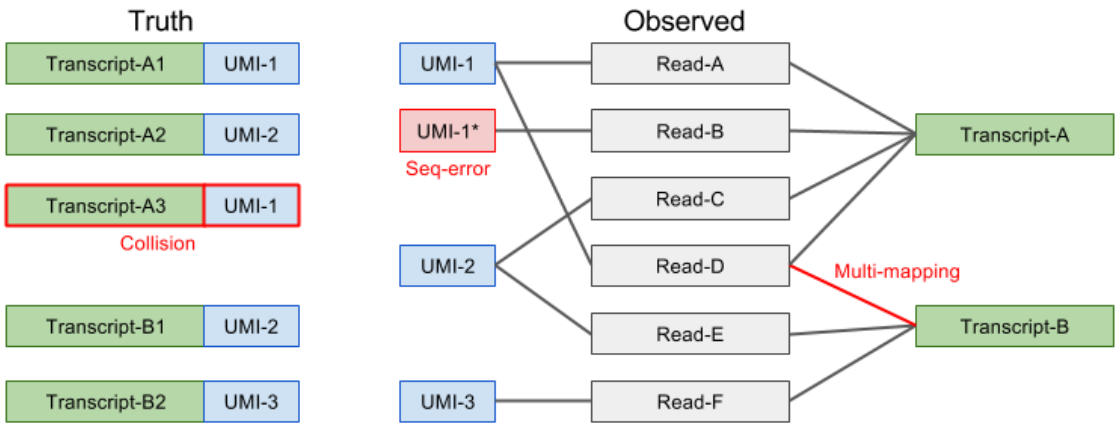


Figure 6.3: Potential Errors in UMIs

where N = total number of unique UMI barcodes and n = number of observed barcodes.

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.

6.5 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 1 We have provided you with UMI counts and read counts from induced pluripotent stem cells generated from three different individuals (**?**) (see: Chapter **??** for details of this dataset).

```
umi_counts <- read.table("tung/molecules.txt", sep = "\t")
read_counts <- read.table("tung/reads.txt", sep = "\t")
```

Using this data:

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

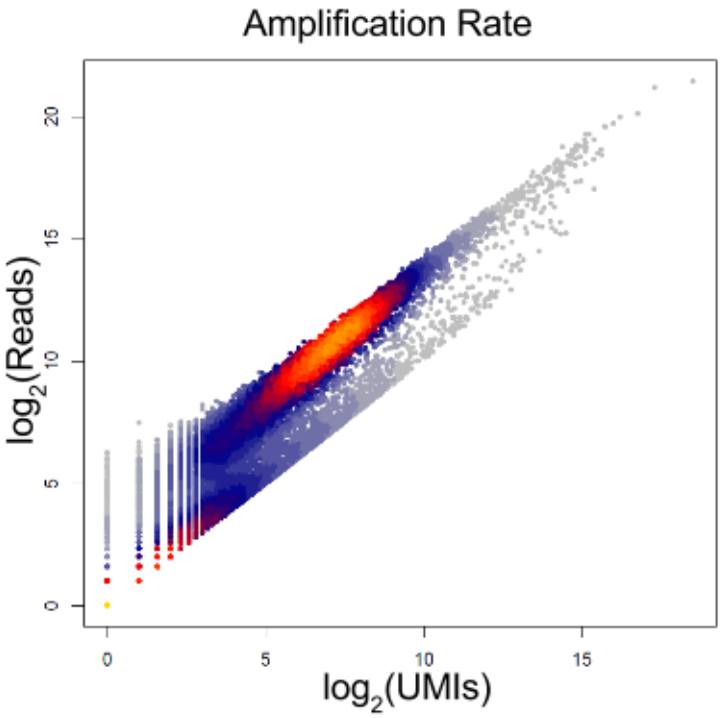


Figure 6.4: Per gene amplification rate

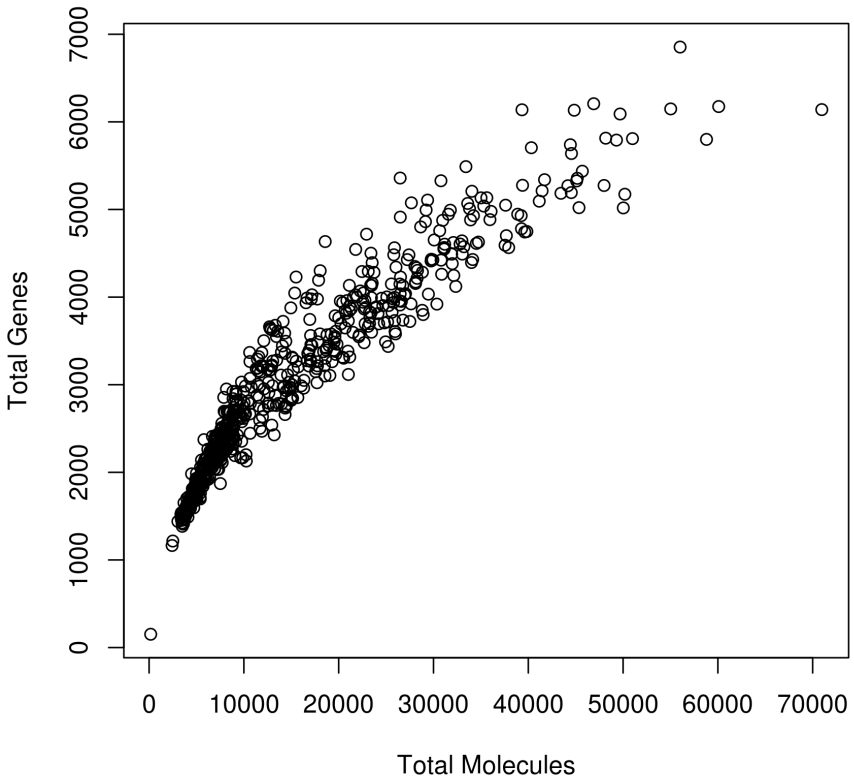


Figure 6.5: Variability in Capture Efficiency

Chapter 7

scater package

7.1 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.

7.2 scater workflow

As illustrated in the figure below, scater will help you with quality control, filtering and normalization of your expression matrix following mapping and alignment.

7.3 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.

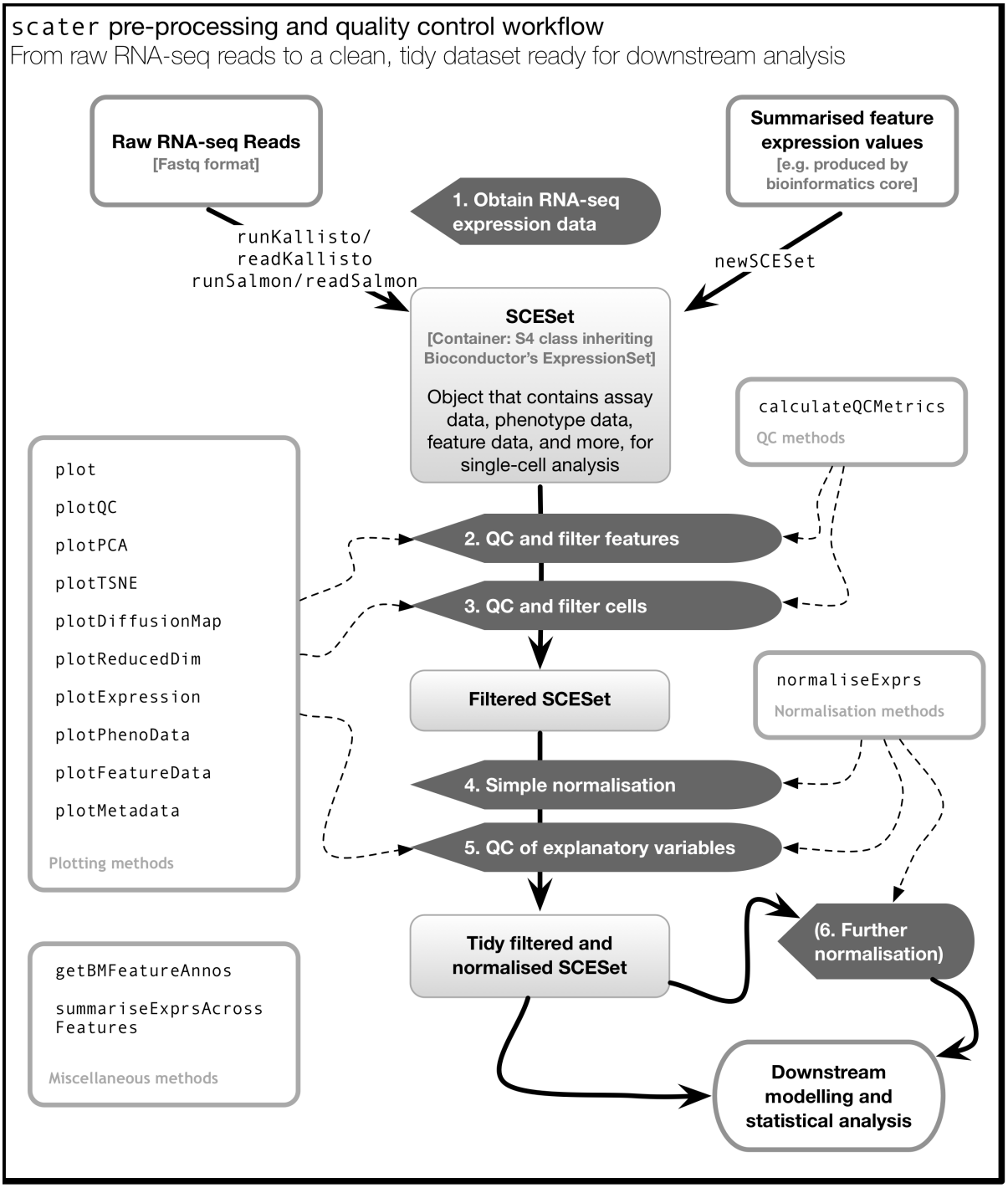

scater pre-processing and quality control workflow
From raw RNA-seq reads to a clean, tidy dataset ready for downstream analysis
Raw RNA-seq Reads
[Fastq format]
Summarised feature expression values
[e.g. produced by bioinformatics core]
1. Obtain RNA-seq expression data
runKallisto/ readKallisto runSalmon/readSalmon
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 7.1:

7.4 SCESet class

(this chapter is taken from the scater vignette)

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 8

Expression QC (UMI)

8.1 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.

8.2 Tung dataset

To illustrate cell QC, we consider a dataset of induced pluripotent stem cells generated from three different individuals (**?**) 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 `tung` 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("tung/molecules.txt", sep = "\t")
anno <- read.table("tung/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 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	3	6	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

```
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,
```

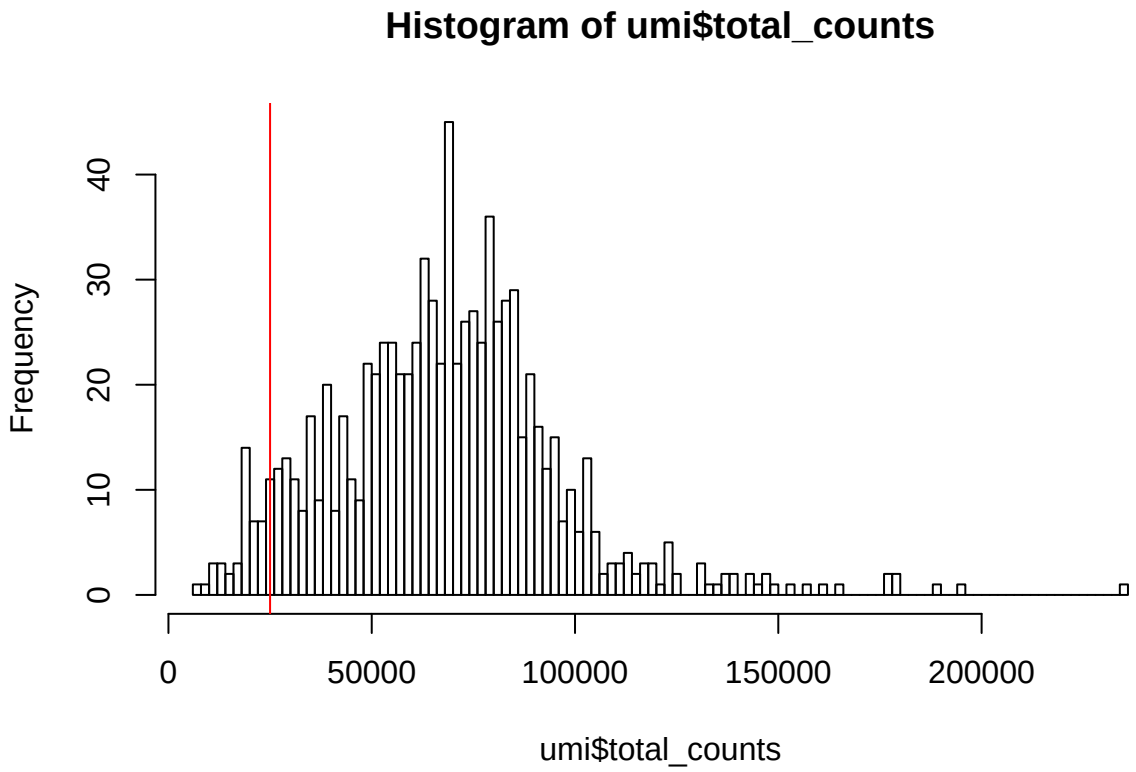


Figure 8.1: Histogram of library sizes for all cells

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

8.3 Cell QC

8.3.1 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 8.3: The number of cells removed by total counts filter (FALSE)

filter_by_total_counts	Freq
FALSE	46
TRUE	818

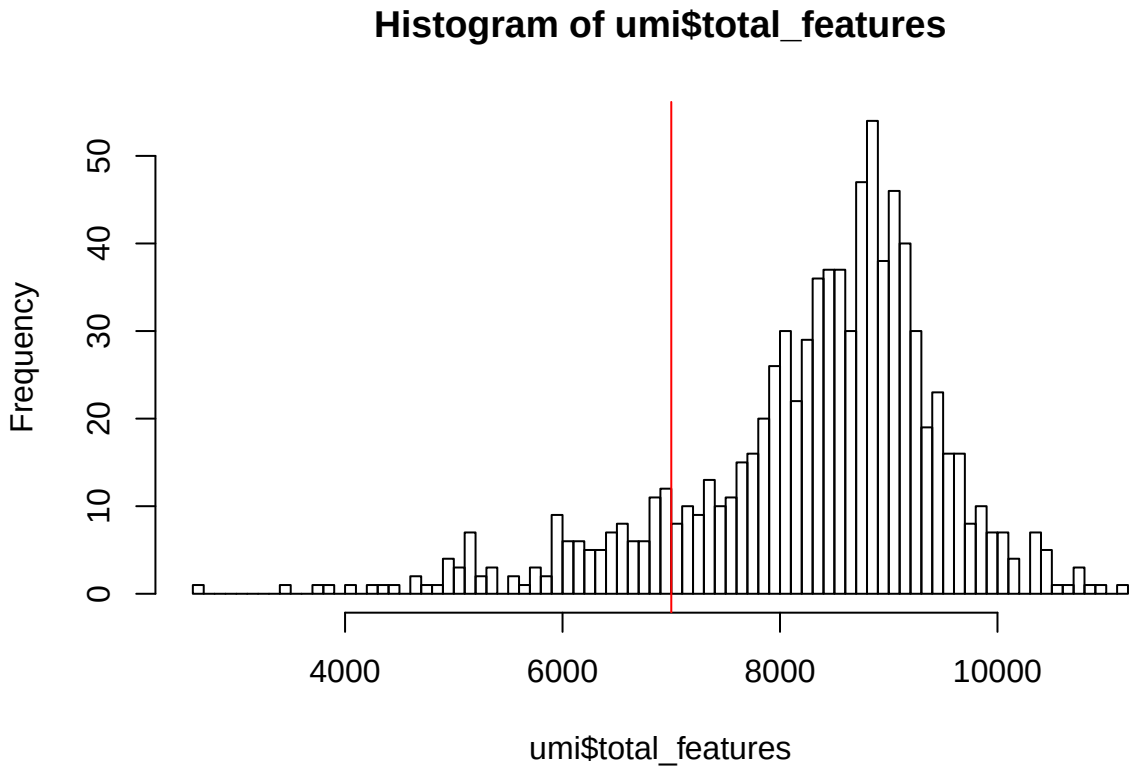


Figure 8.2: Histogram of the number of detected genes in all cells

8.3.2 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 8.4: The number of cells removed by total features filter (FALSE)

filter_by_expr_features	Freq
FALSE	120
TRUE	744

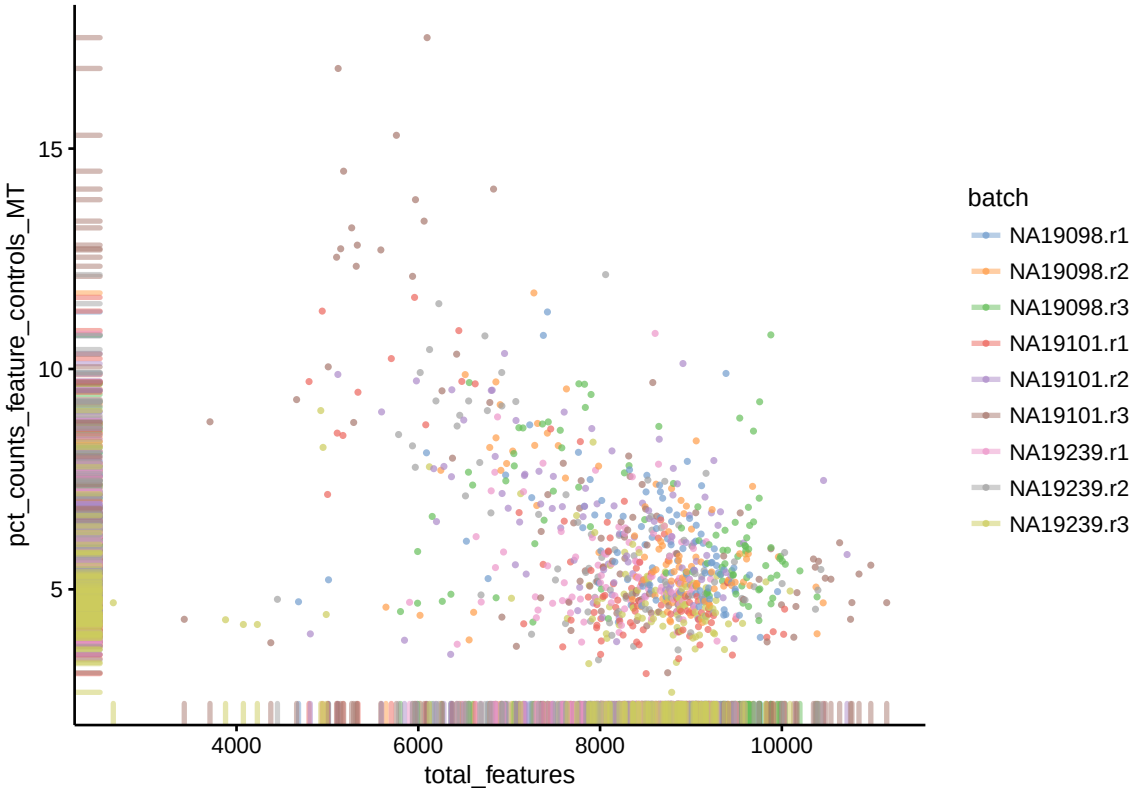


Figure 8.3: Percentage of counts in MT genes

How many cells does our filter remove?

Our answer

8.3.3 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")
)
```

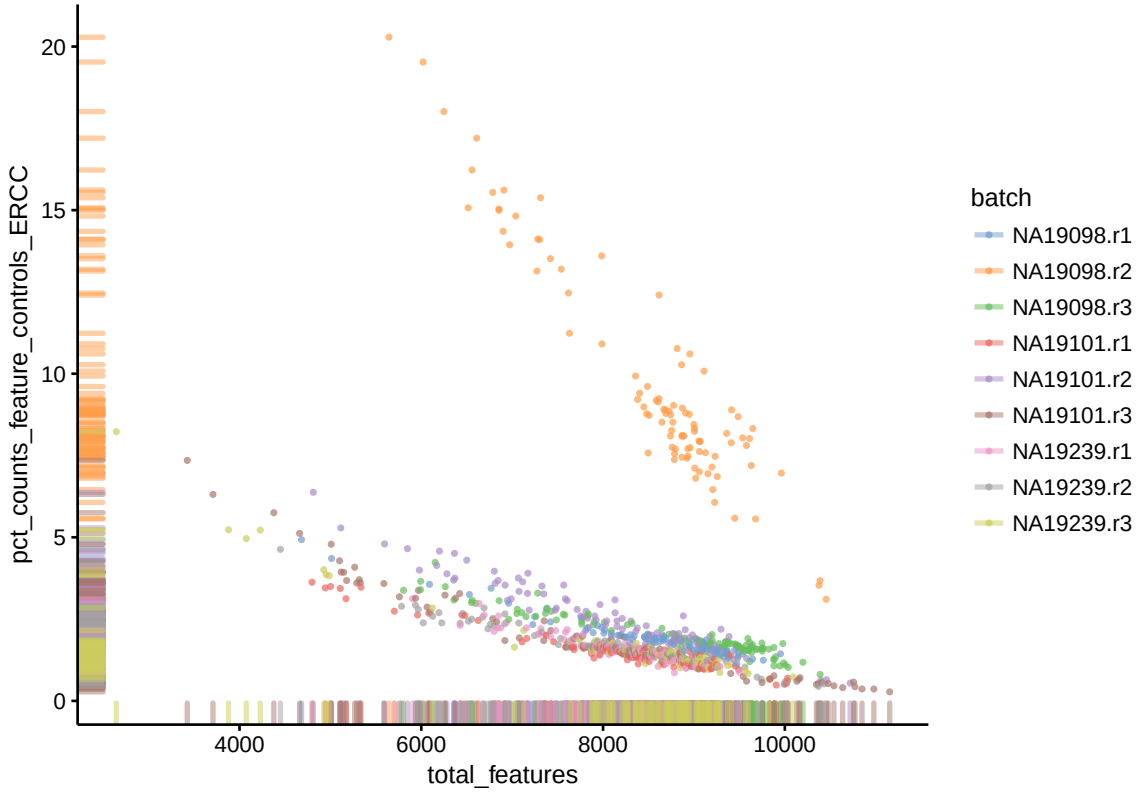


Figure 8.4: Percentage of counts in ERCCs

Table 8.5: 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 8.6: The number of cells removed by MT filter (FALSE)

filter_by_MT	Freq
FALSE	31
TRUE	833

Table 8.7: 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).

8.4 Cell filtering

8.4.1 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)'
)
```

8.4.2 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 8.8: 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)'
)
```

8.4.3 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 8.9: 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)'
)
```

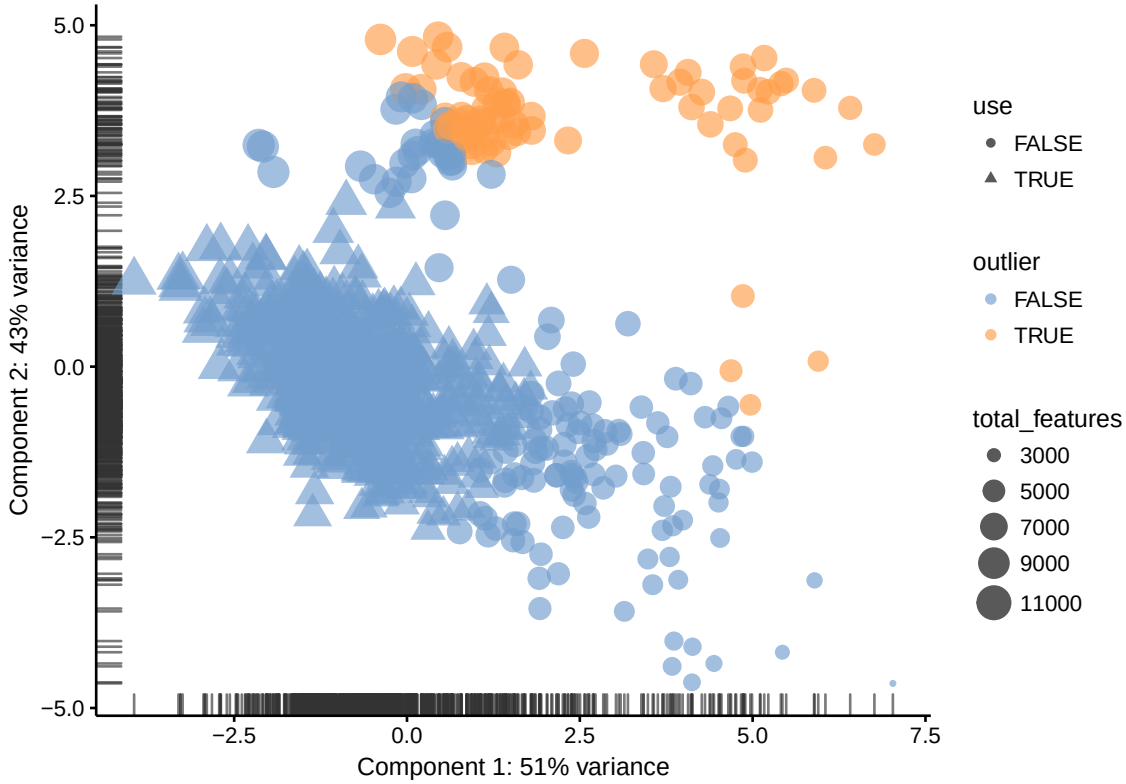


Figure 8.5: PCA plot used for automatic detection of cell outliers

8.5 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

8.6 Gene analysis

8.6.1 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

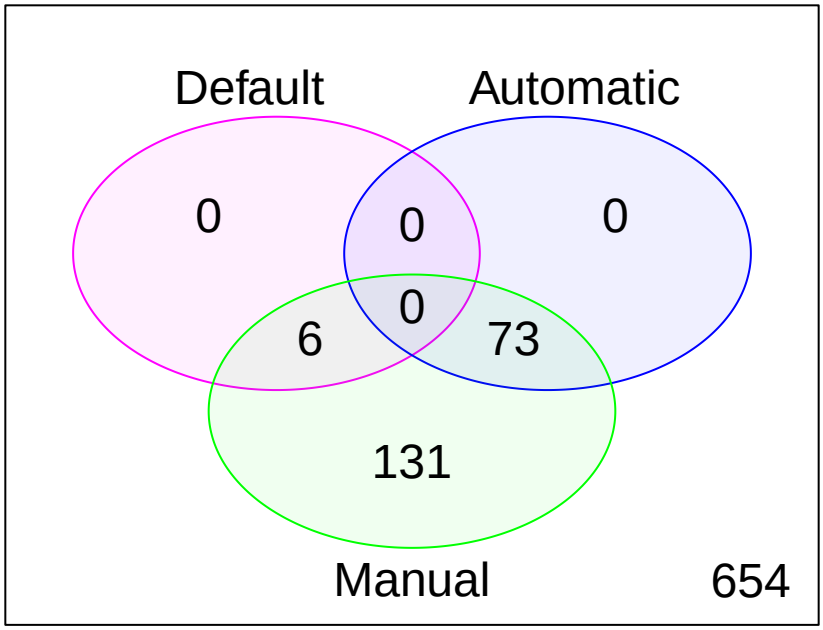


Figure 8.6: Comparison of the default, automatic and manual cell filters

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

filter_genes	Freq
FALSE	4663
TRUE	14063

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

8.6.2 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.

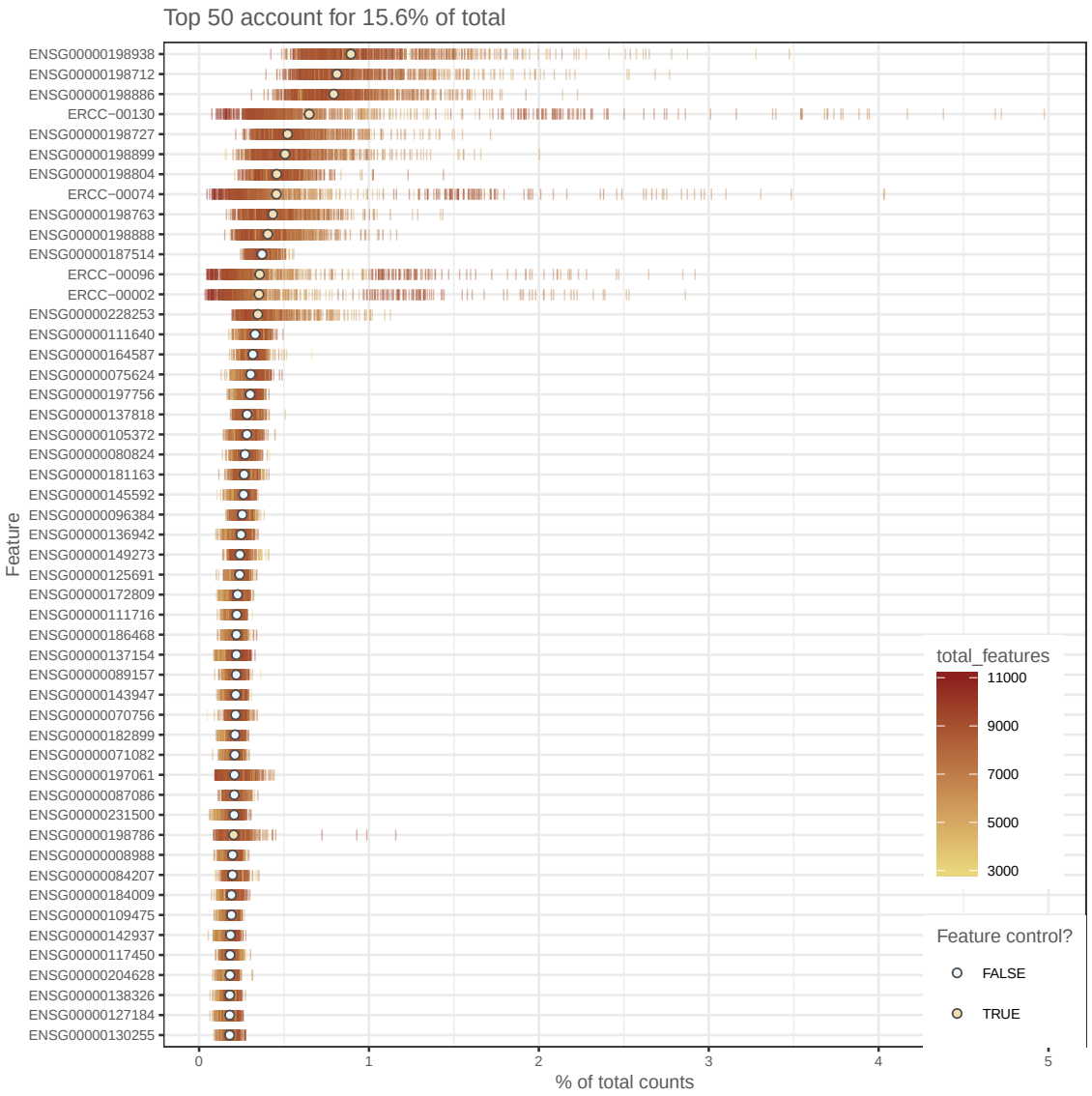


Figure 8.7: Number of total counts consumed by the top 50 expressed genes

8.7 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
```

Let's create an additional slot with log-transformed counts (we will need it in the next chapters):

```
set_exprs(umi, "log2_counts") <- log2(counts(umi) + 1)
```

Save the data:

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

8.8 Big Exercise

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

Chapter 9

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("tung/reads.txt", sep = "\t")
anno <- read.table("tung/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 9.1: 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 9.2: 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 9.3: 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")
```

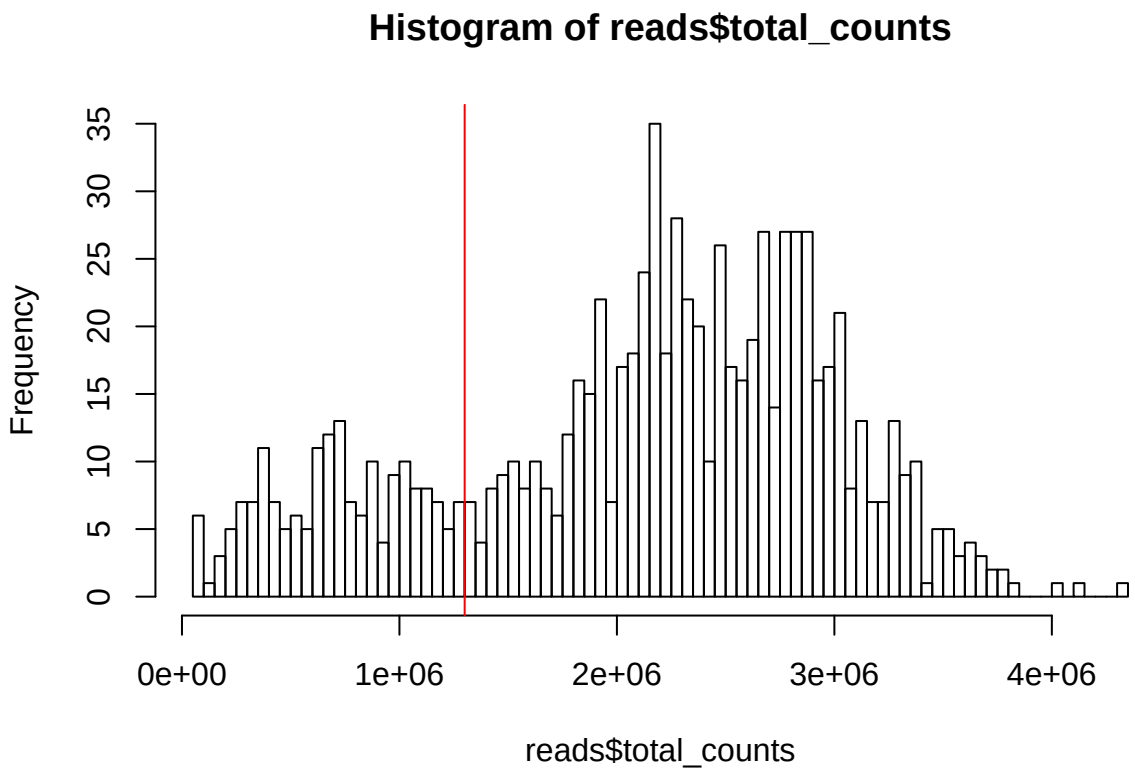


Figure 9.1: Histogram of library sizes for all cells

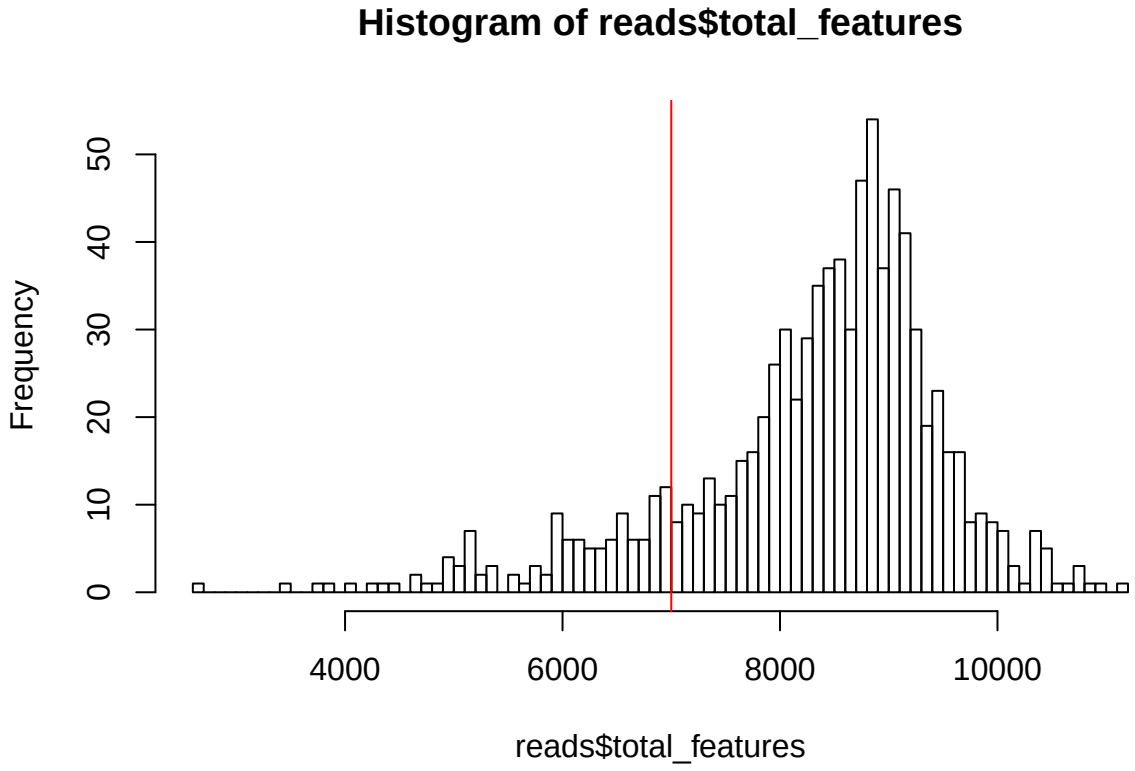


Figure 9.2: Histogram of the number of detected genes in all cells

Table 9.4: The number of cells removed by total features filter (FALSE)

filter_by_expr_features	Freq
FALSE	120
TRUE	744

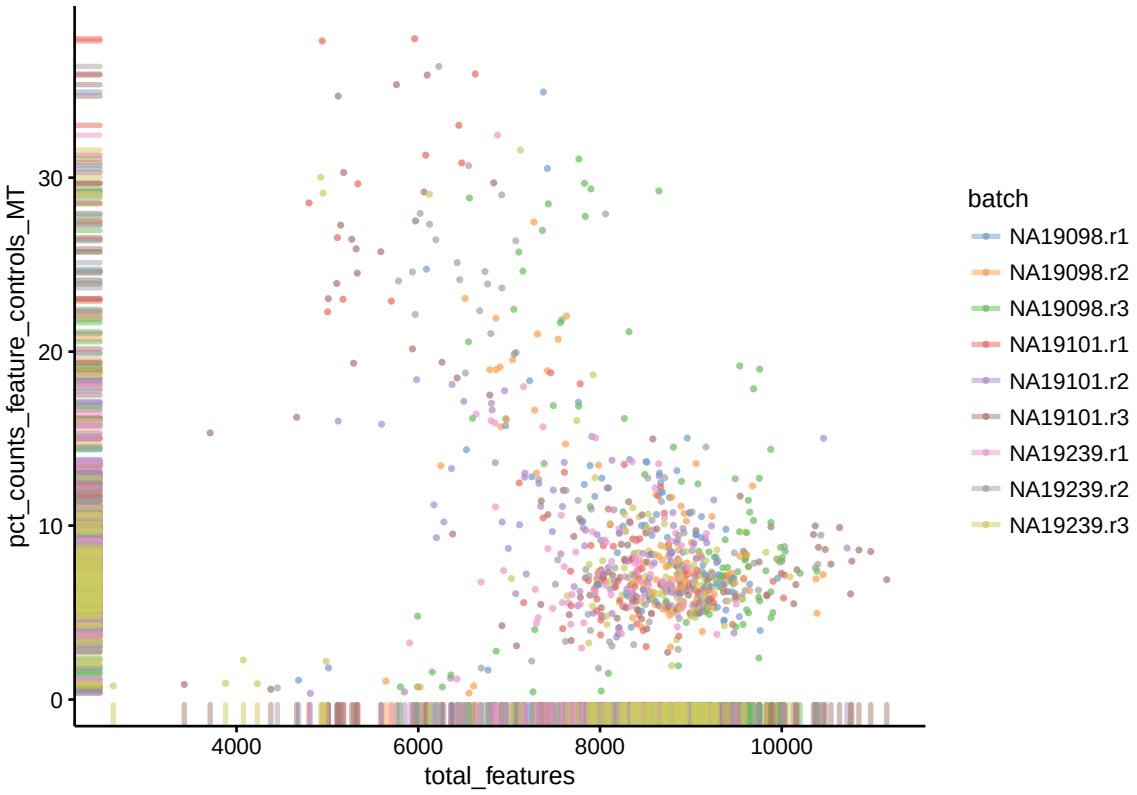


Figure 9.3: Percentage of counts in MT genes

```
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",
```

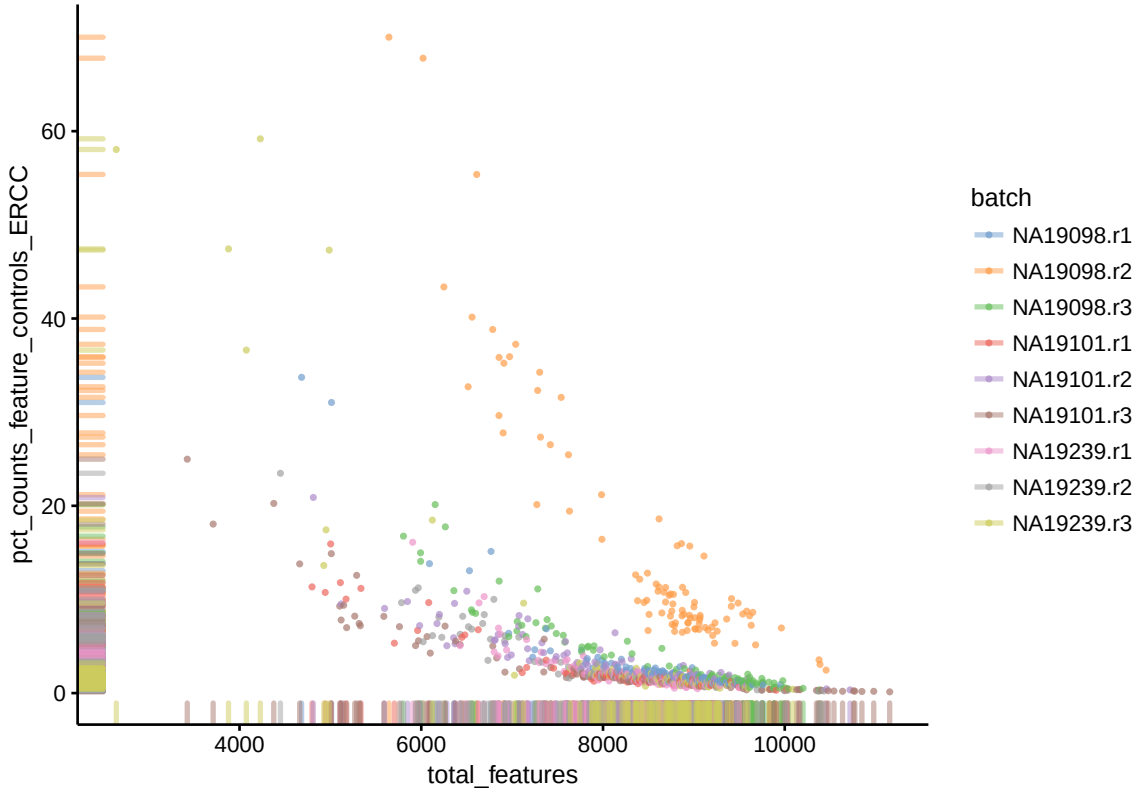


Figure 9.4: Percentage of counts in ERCCs

Table 9.5: 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 9.6: The number of cells removed by MT filter (FALSE)

filter_by_MT	Freq
FALSE	18
TRUE	846

Table 9.7: 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 9.8: 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))
```

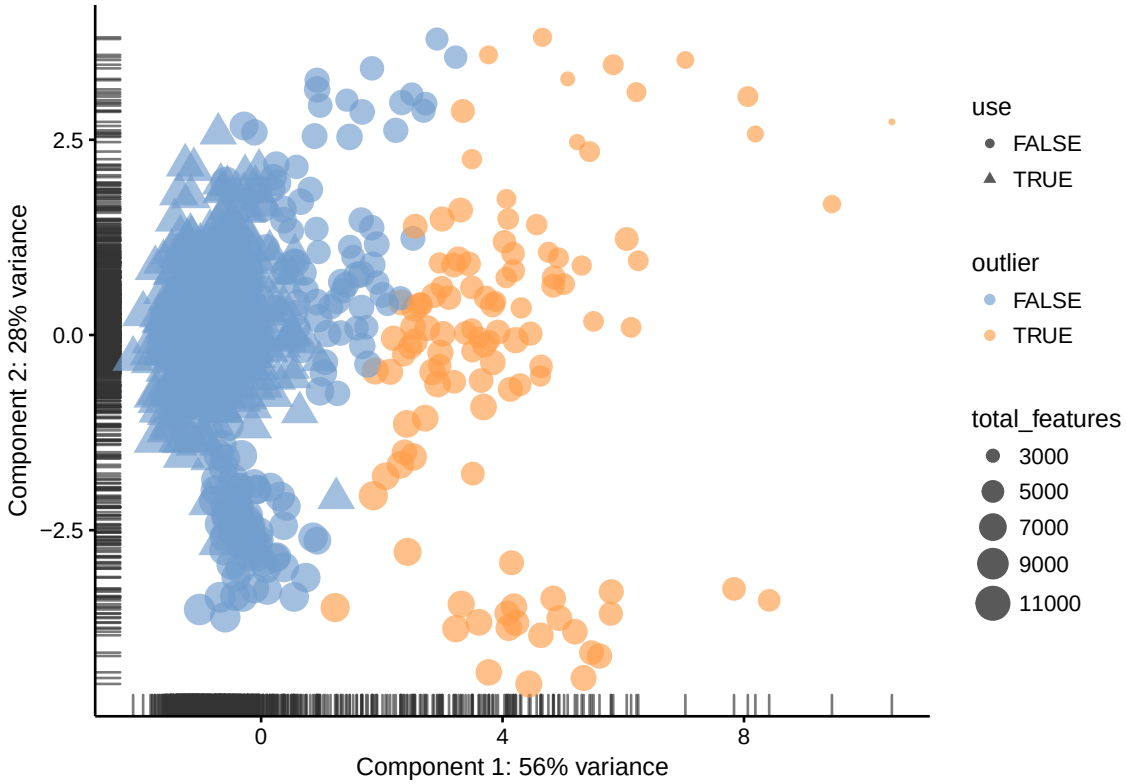


Figure 9.5: PCA plot used for automatic detection of cell outliers

Table 9.9: The number of cells removed by automatic filter (FALSE)

Var1	Freq
FALSE	753
TRUE	111

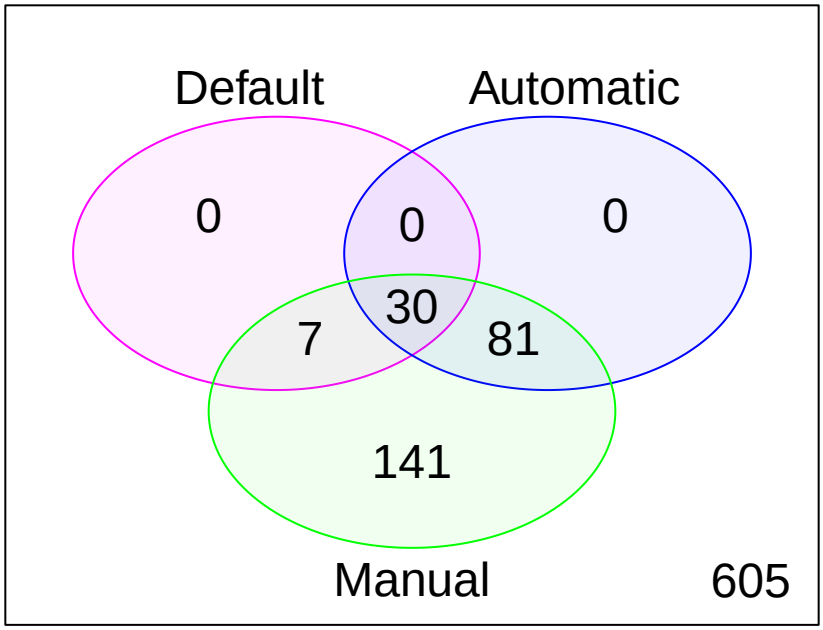


Figure 9.6: Comparison of the default, automatic and manual cell filters

Table 9.10: 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
```

```
set_exprs(reads, "log2_counts") <- log2(counts(reads) + 1)
```

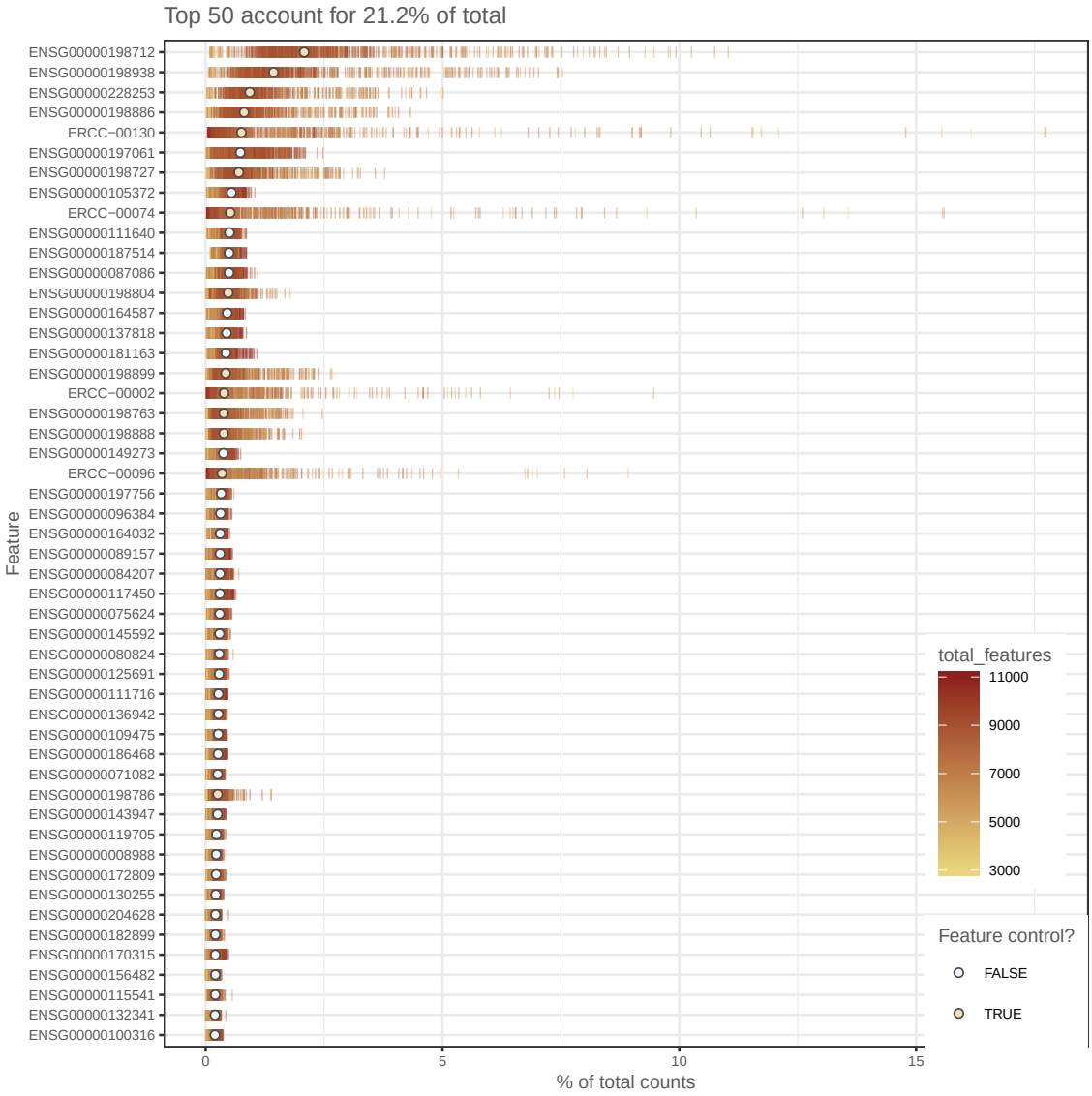


Figure 9.7: Number of total counts consumed by the top 50 expressed genes

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

By comparing Figure **??** and Figure **??**, 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 10

Data visualization

10.1 Introduction

In this chapter we will continue to work with the filtered `Tung` 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 `Tung` 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("tung/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

10.2 PCA plot

The easiest way to overview the data is by transforming 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. 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).

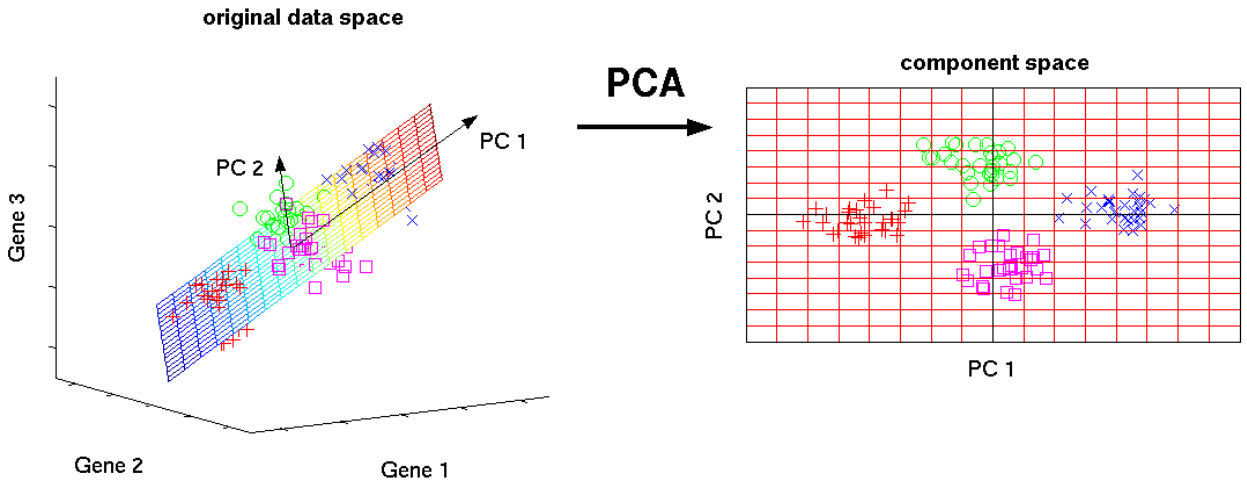


Figure 10.1: Schematic representation of PCA dimensionality reduction

10.2.1 Before QC

Without log-transformation:

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

With log-transformation:

```
scater::plotPCA(umi[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "log2_counts")
```

Clearly log-transformation is benefitial for our data - it reduces the variance on the first principal component and already separates some biological effects. Moreover, it makes the distribution of the expression values more normal. In the following analysis and chapters we will be using log-transformed raw counts by default.

However, note that just a log-transformation is not enough to account for different technical factors between the cells (e.g. sequencing depth). Therefore, please do not use `log2_counts` for your downstream analysis, instead as a minimum suitable data use the `exprs` slot of the scater object, which not just log-transformed, but also normalised by library size (CPM normalisation). In the course we use `log2_counts` only for demonstration purposes!

10.2.2 After QC

```
scater::plotPCA(umi.qc[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
```

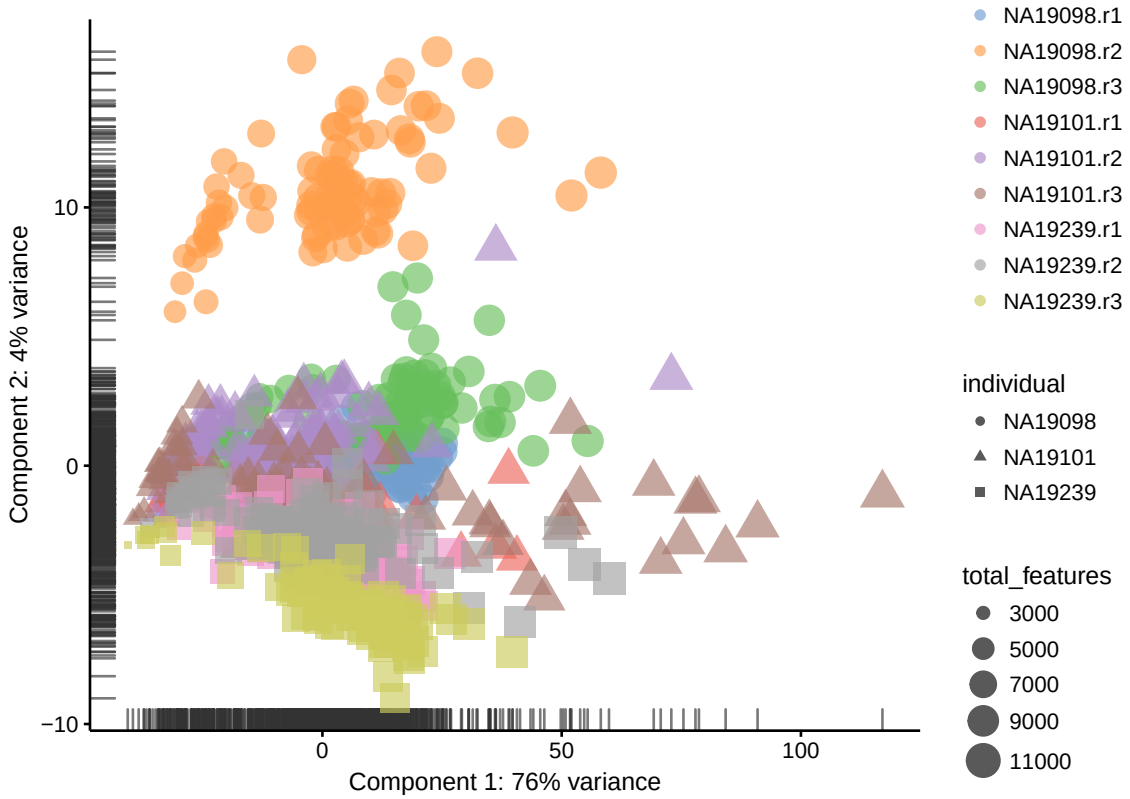


Figure 10.2: PCA plot of the tung data

```
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "log2_counts")
```

Comparing Figure **??** and Figure **??**, 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? Why does the fraction of variance accounted for by the first PC change so dramatically?

Our answer

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

10.3 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

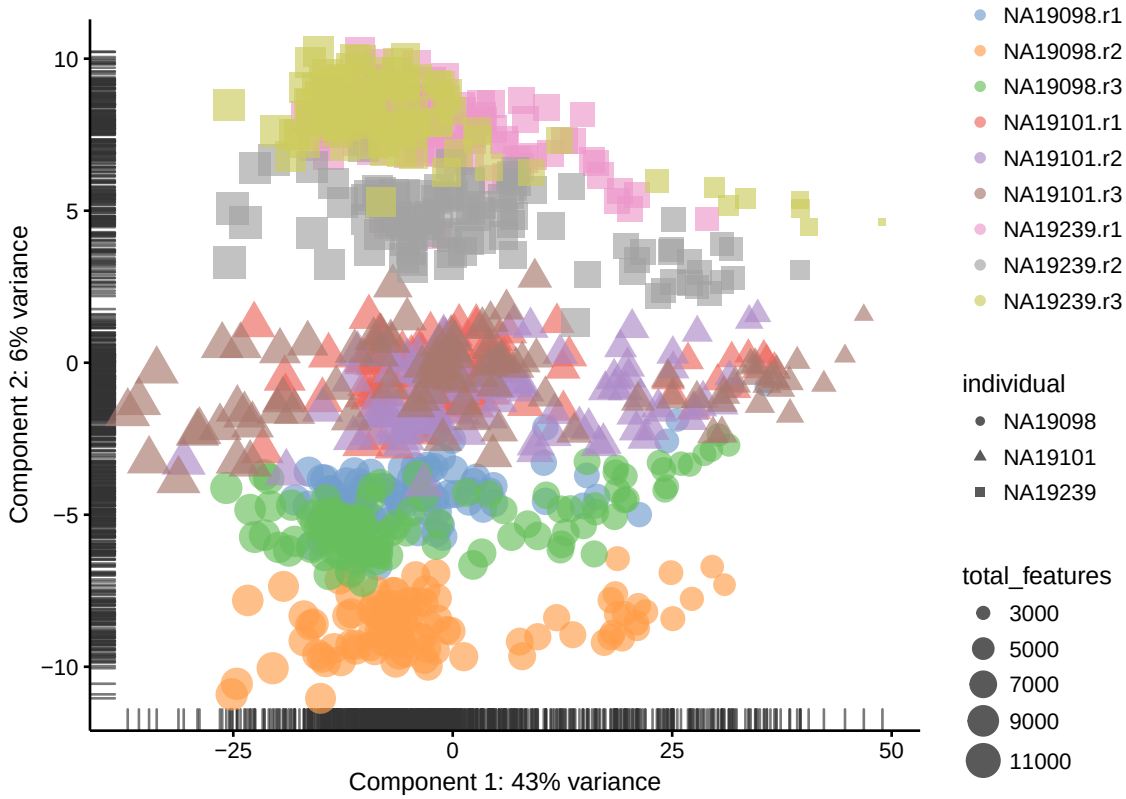


Figure 10.3: PCA plot of the tung data

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.

10.3.1 Before QC

```
scater::plotTSNE(umi[endog_genes, ],
                 ntop = 500,
                 perplexity = 130,
                 colour_by = "batch",
                 size_by = "total_features",
                 shape_by = "individual",
                 exprs_values = "log2_counts",
                 rand_seed = 123456)
```

10.3.2 After QC

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

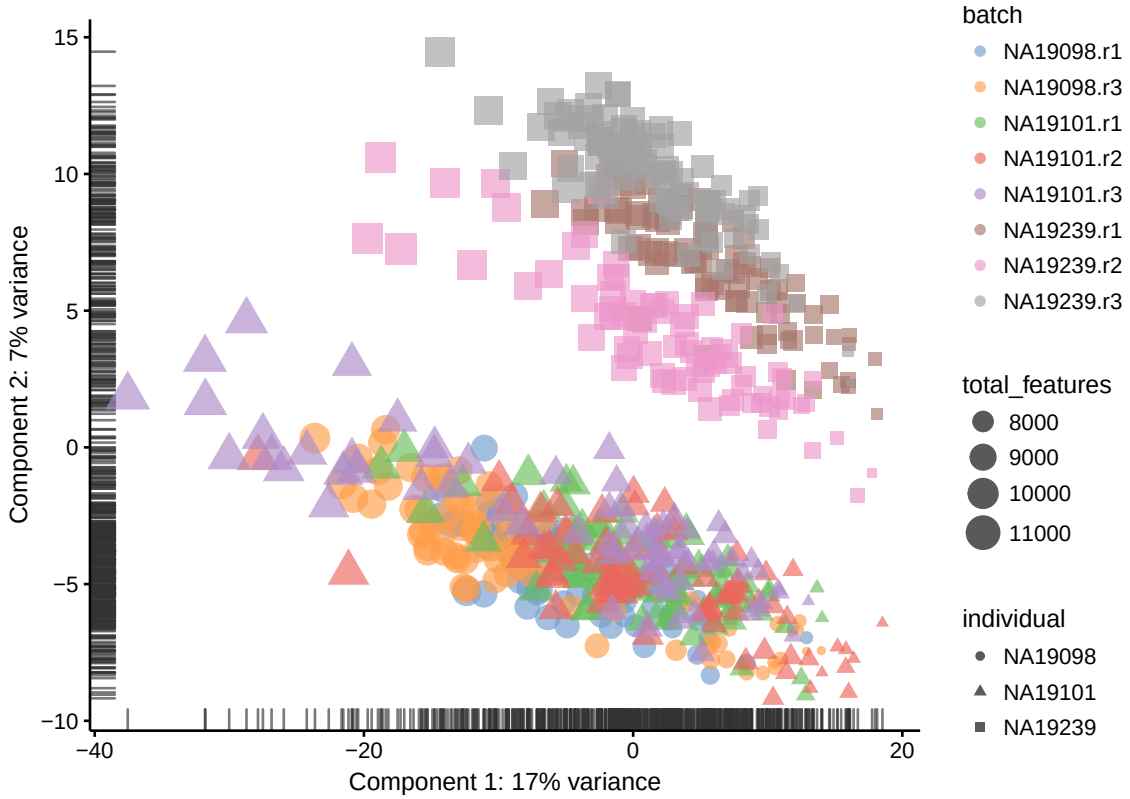


Figure 10.4: PCA plot of the tung data

```
                exprs_values = "log2_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 **??** and **??**, 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? How does the choice of perplexity affect the interpretation of the results?

Our answer

10.4 Big Exercise

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

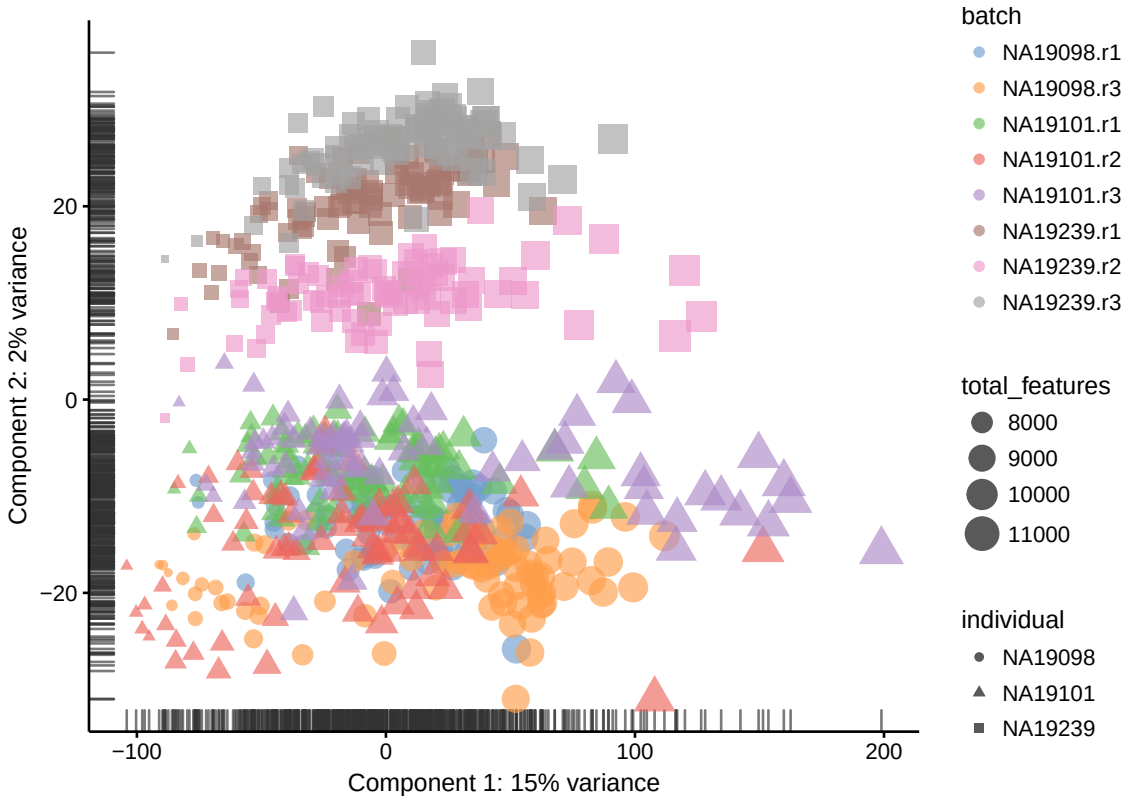


Figure 10.5: PCA plot of the tung data (14214 genes)

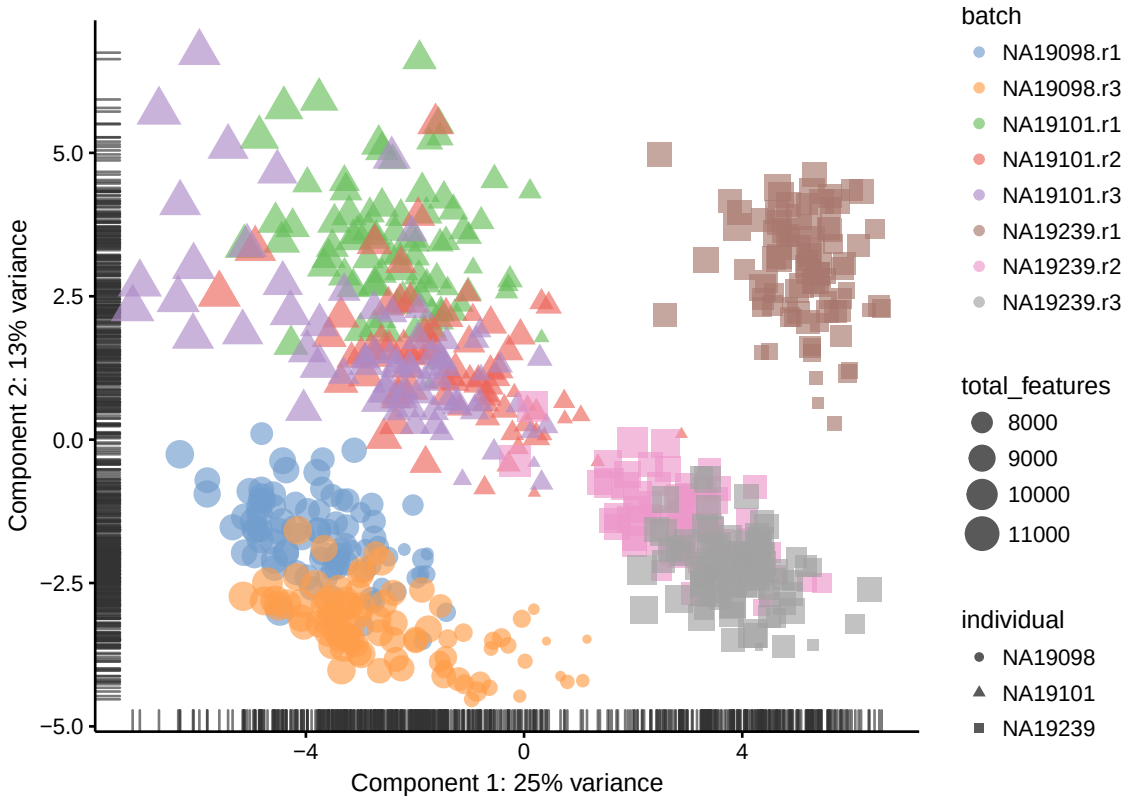


Figure 10.6: PCA plot of the tung data (50 genes)

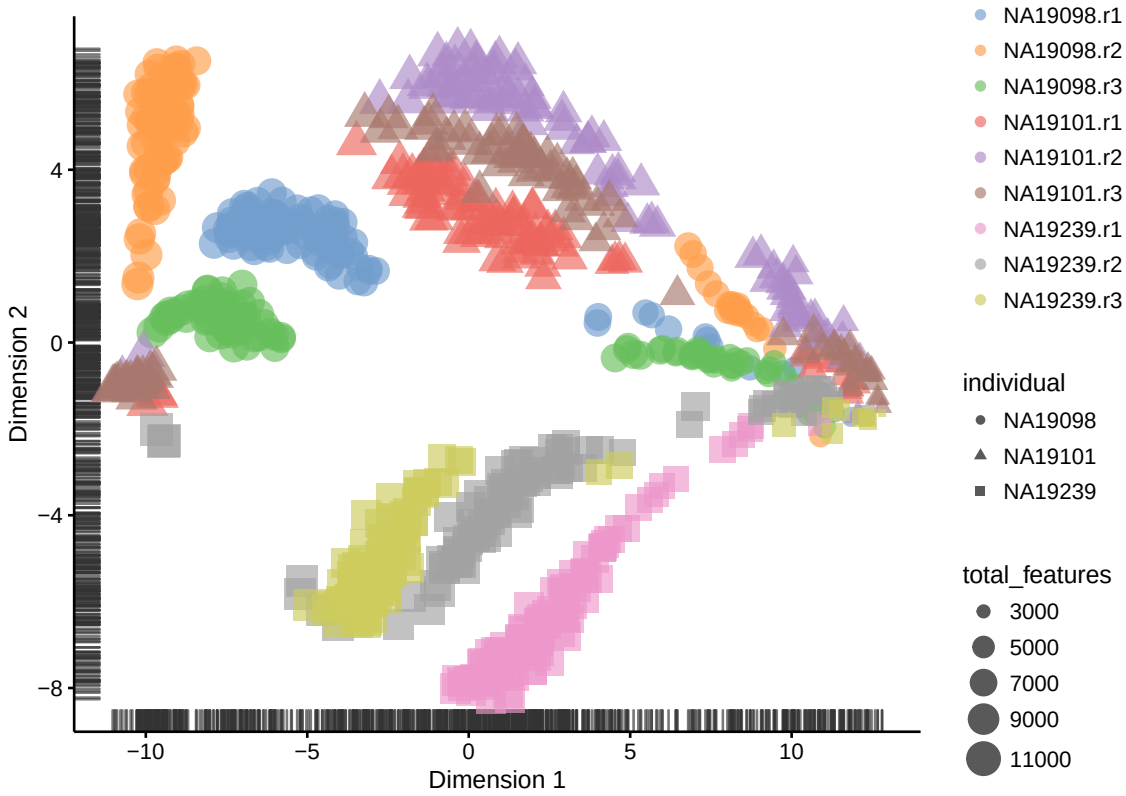


Figure 10.7: tSNE map of the tung data

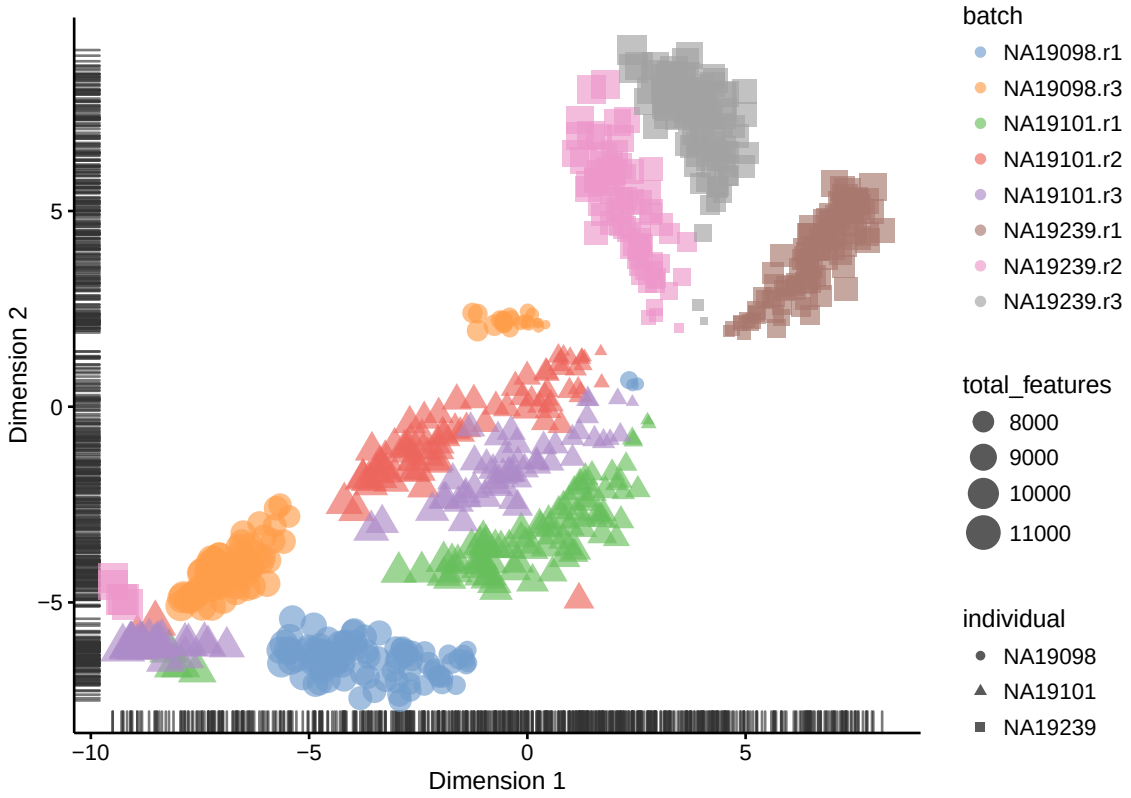


Figure 10.8: tSNE map of the tung data

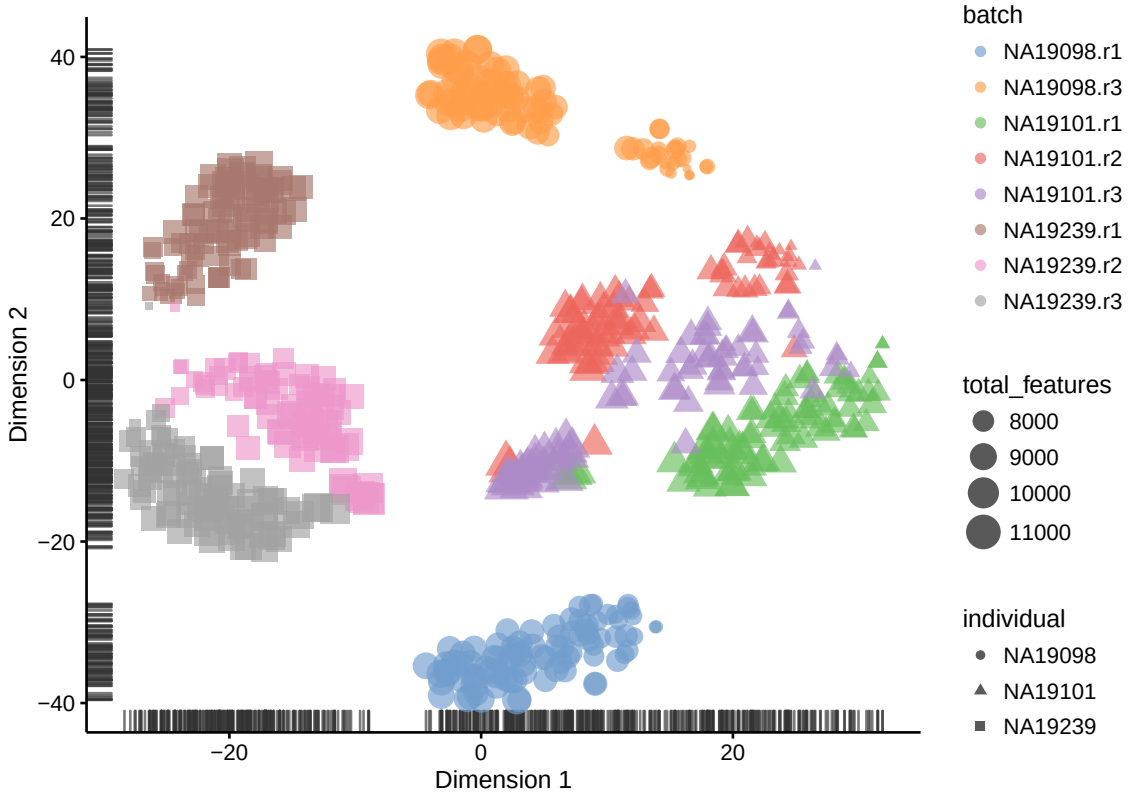


Figure 10.9: tSNE map of the tung data (perplexity = 10)

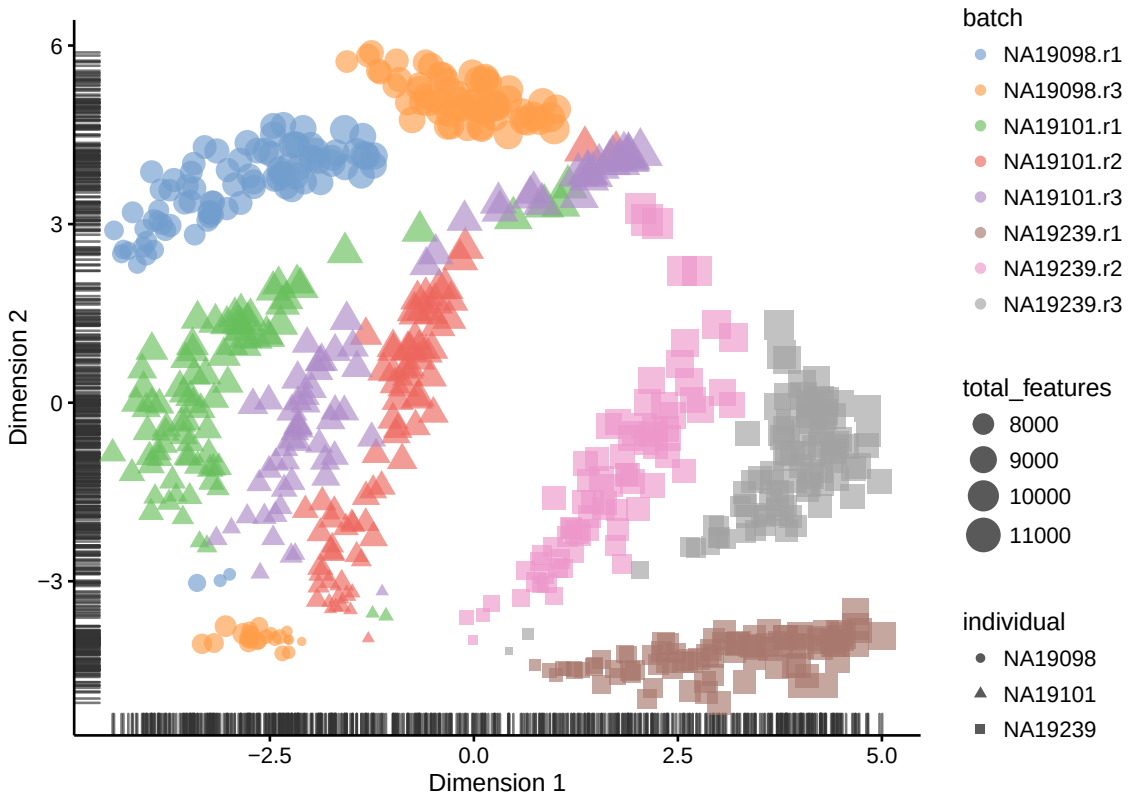


Figure 10.10: tSNE map of the tung data (perplexity = 200)

Chapter 11

Data visualization (Reads)

```
library(scater, quietly = TRUE)
options(stringsAsFactors = FALSE)
reads <- readRDS("tung/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[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "log2_counts")
```

```
scater::plotPCA(reads.qc[endog_genes, ],
                ntop = 500,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "log2_counts")
```

```
scater::plotTSNE(reads[endog_genes, ],
                 ntop = 500,
                 perplexity = 130,
                 colour_by = "batch",
                 size_by = "total_features",
                 shape_by = "individual",
                 exprs_values = "log2_counts",
                 rand_seed = 123456)
```

```
scater::plotTSNE(reads.qc[endog_genes, ],
                 ntop = 500,
```

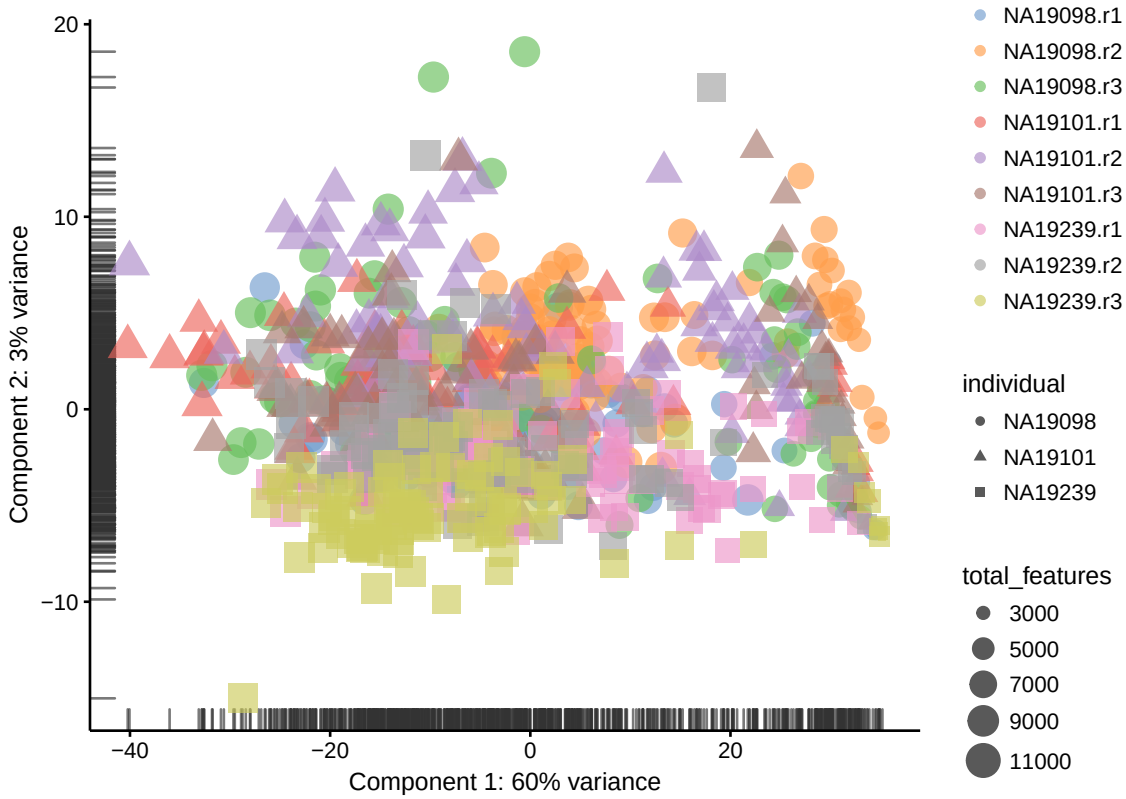


Figure 11.1: PCA plot of the tung data

```
                perplexity = 130,
                colour_by = "batch",
                size_by = "total_features",
                shape_by = "individual",
                exprs_values = "log2_counts",
                rand_seed = 123456)
```

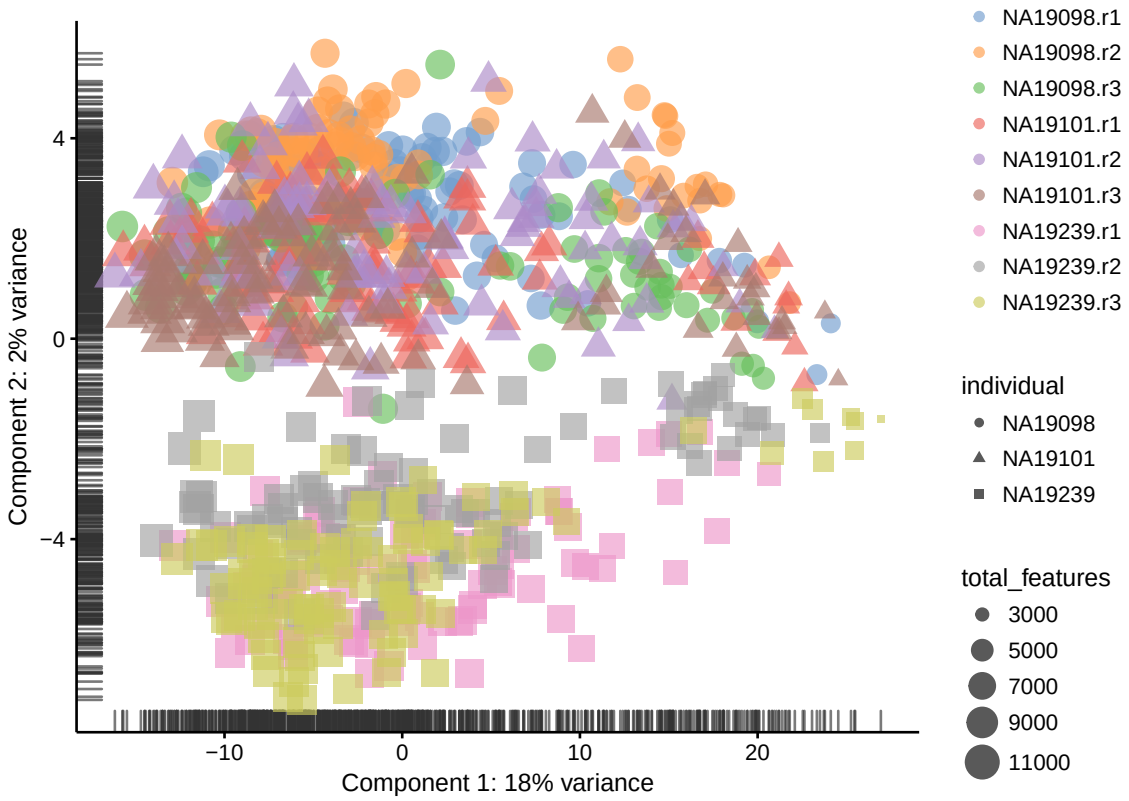


Figure 11.2: PCA plot of the tung data

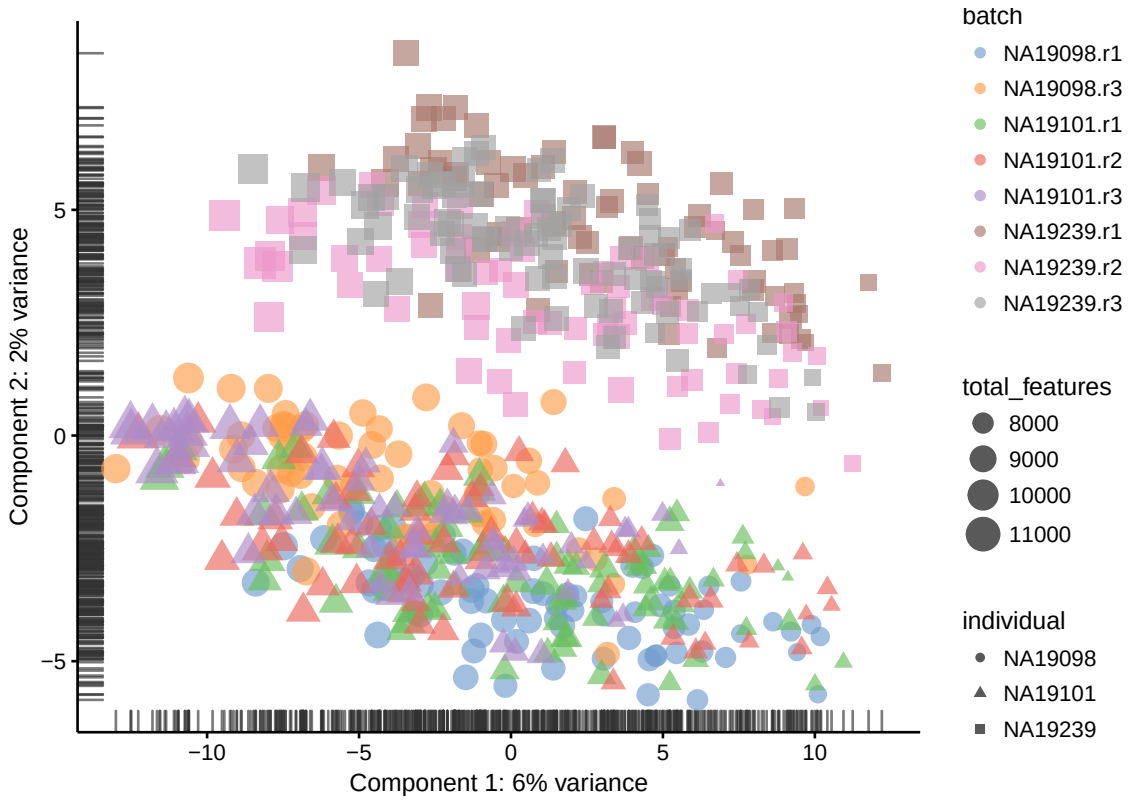


Figure 11.3: PCA plot of the tung data

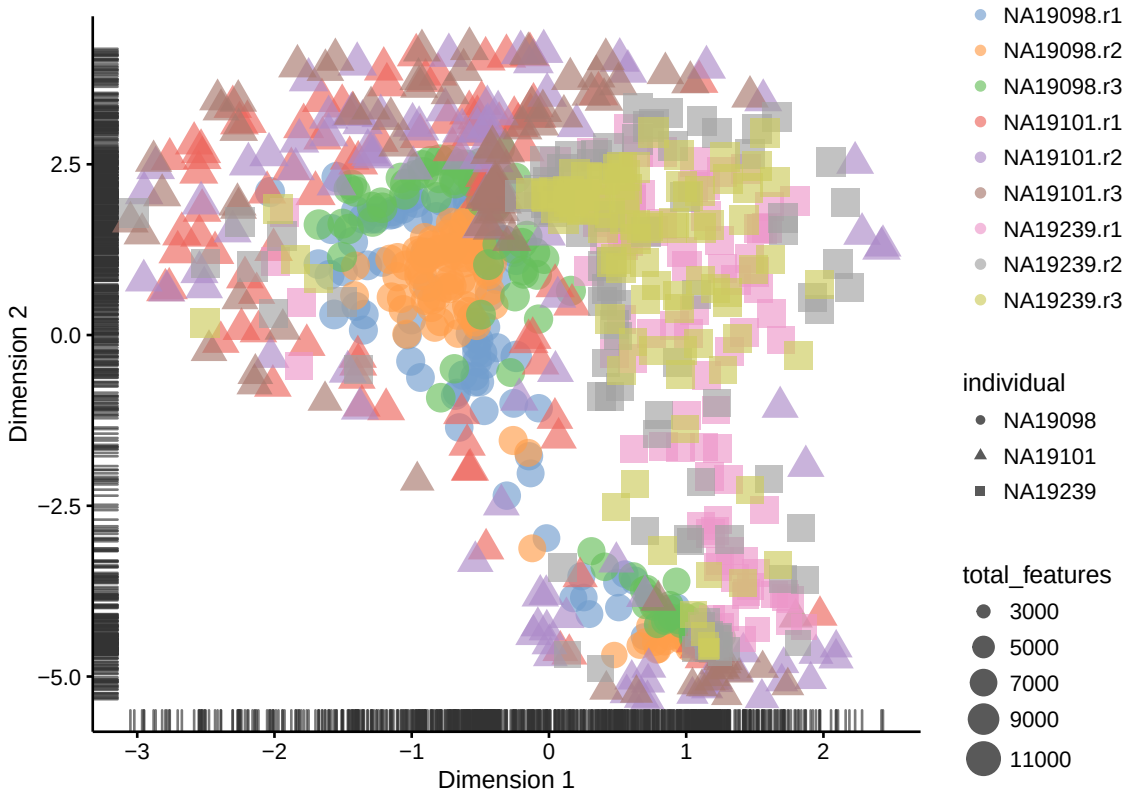


Figure 11.4: tSNE map of the tung data

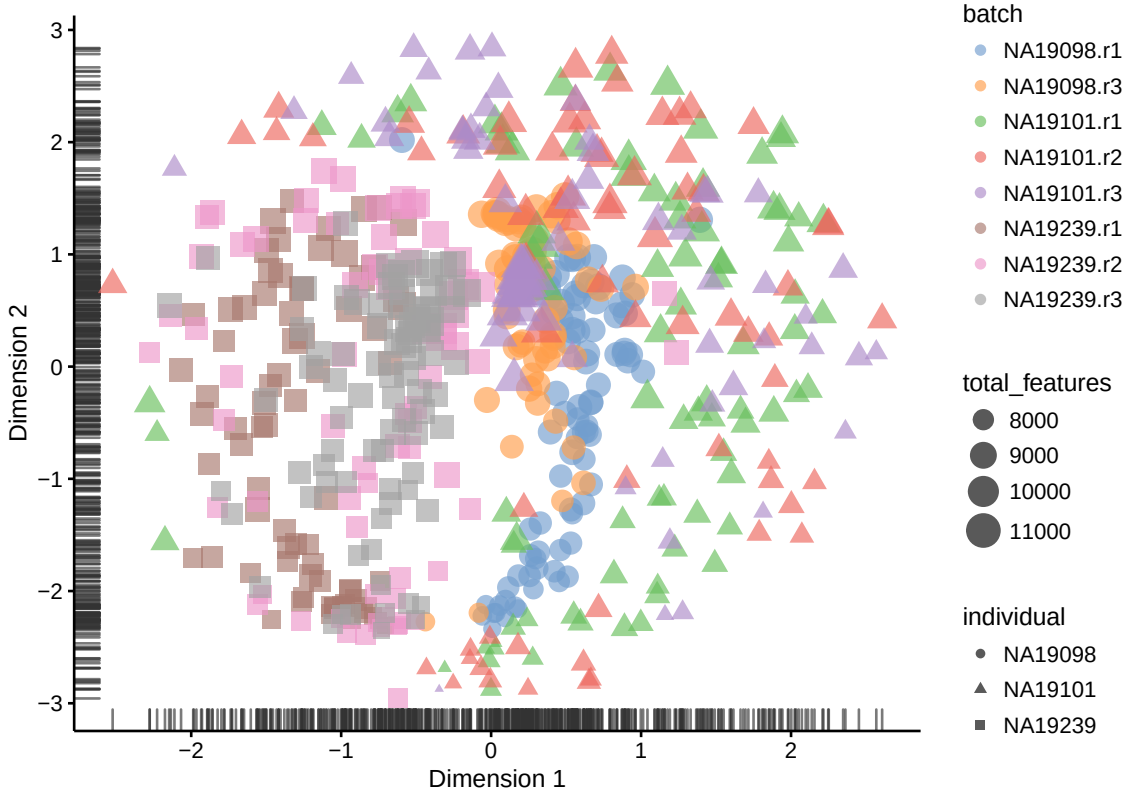


Figure 11.5: tSNE map of the tung data

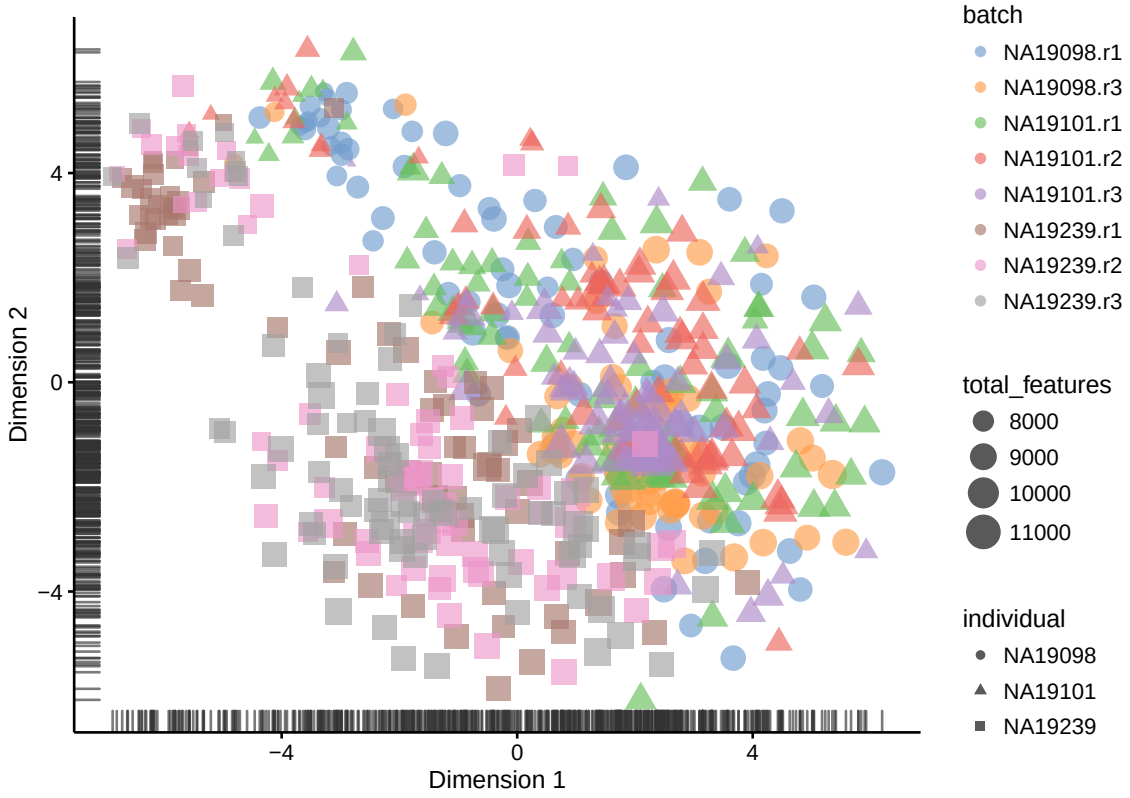


Figure 11.6: tSNE map of the tung data (perplexity = 10)

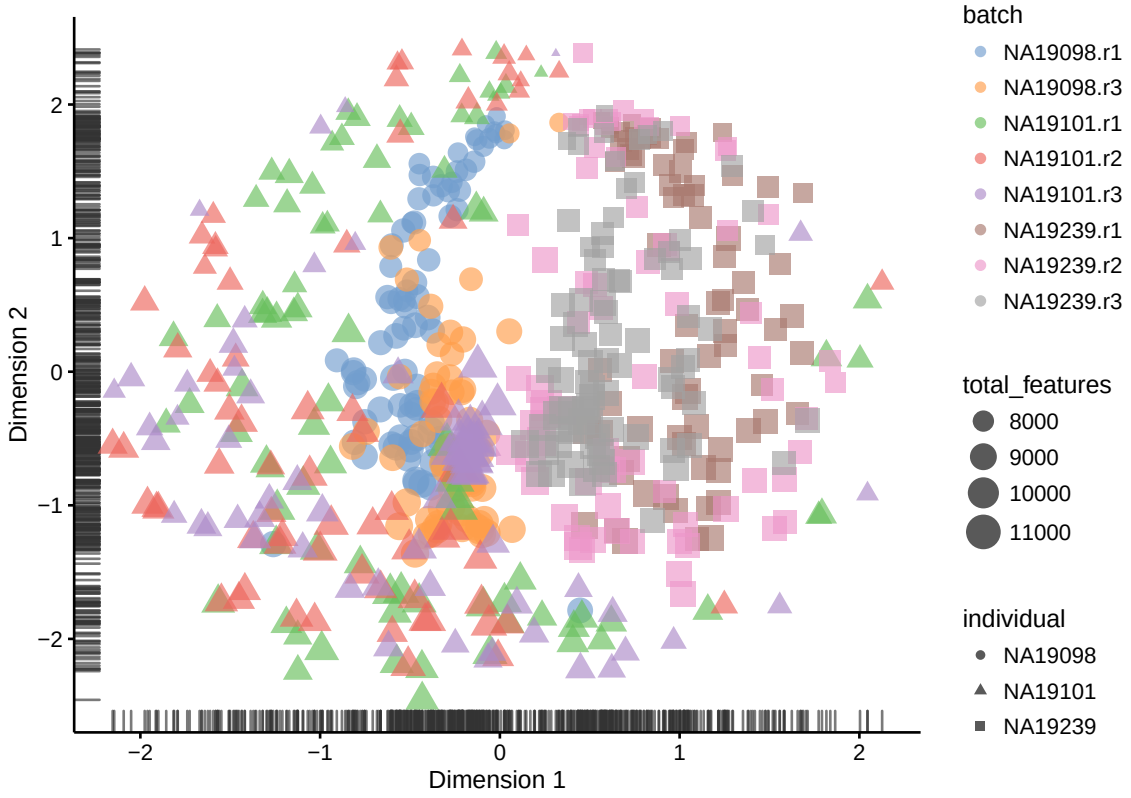


Figure 11.7: tSNE map of the tung data (perplexity = 200)

Chapter 12

Identifying confounding factors

12.1 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("tung/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.

12.2 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 = "log2_counts")
```

scater allows one to identify principal 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.

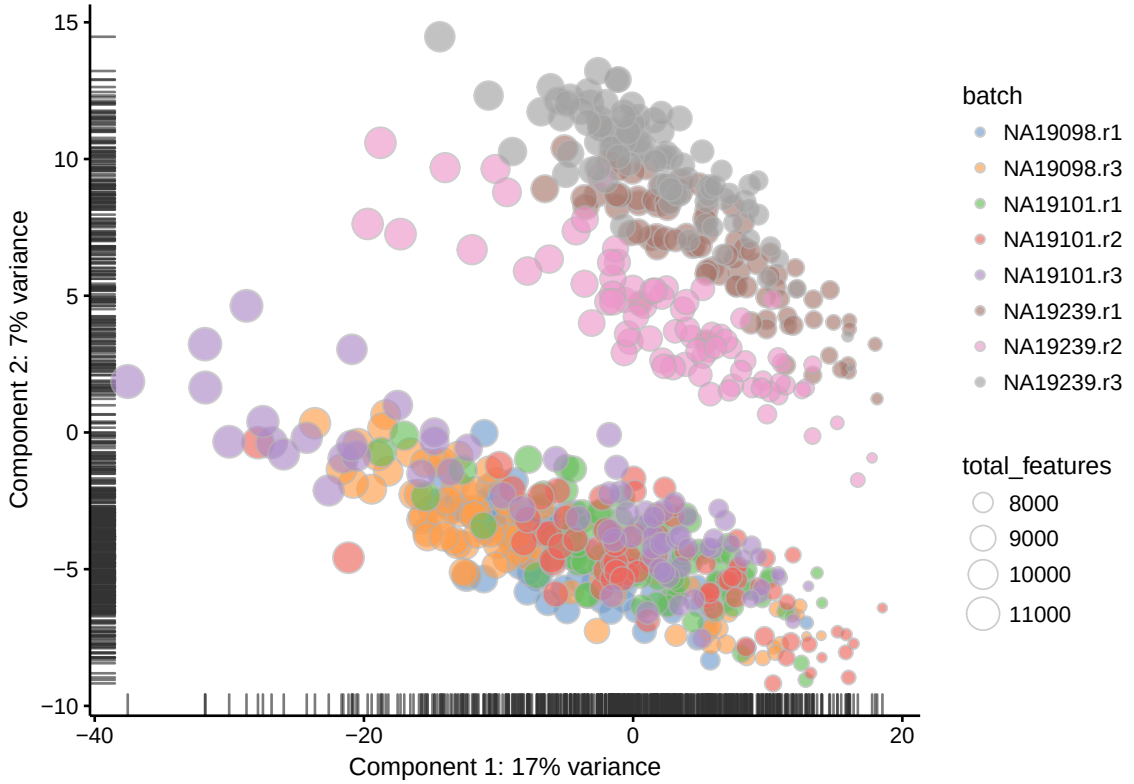


Figure 12.1: PCA plot of the tung data

12.2.1 Detected genes

```
scater::plotQC(umi.qc[endog_genes, ],
               type = "find-pcs",
               variable = "total_features",
               exprs_values = "log2_counts")
```

Indeed, we can see that PC1 can be almost 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.

12.3 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 = "log2_counts",
               variables = c("total_features",
                             "total_counts",
                             "batch",
                             "individual",
```

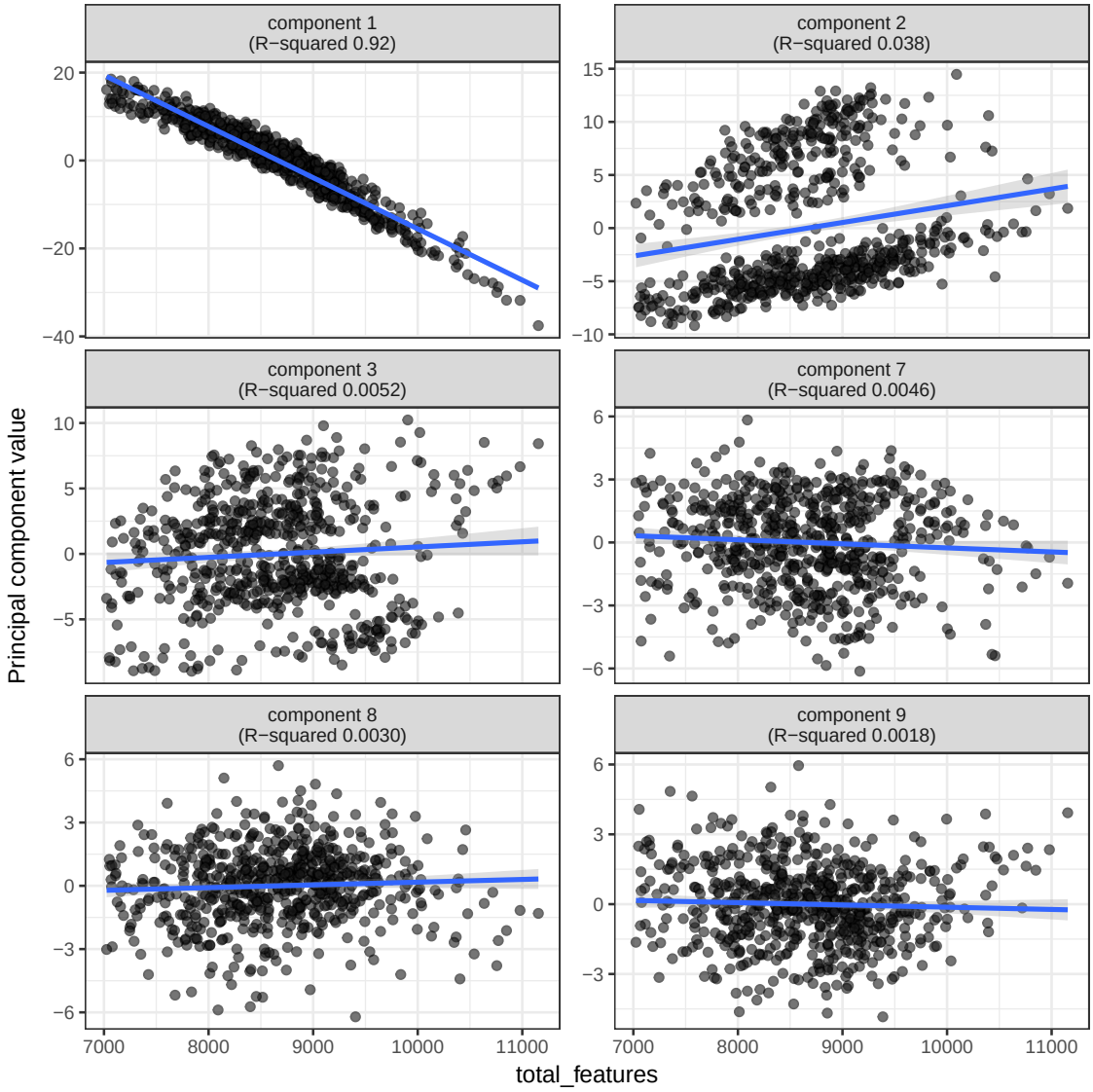


Figure 12.2: PC correlation with the number of detected genes

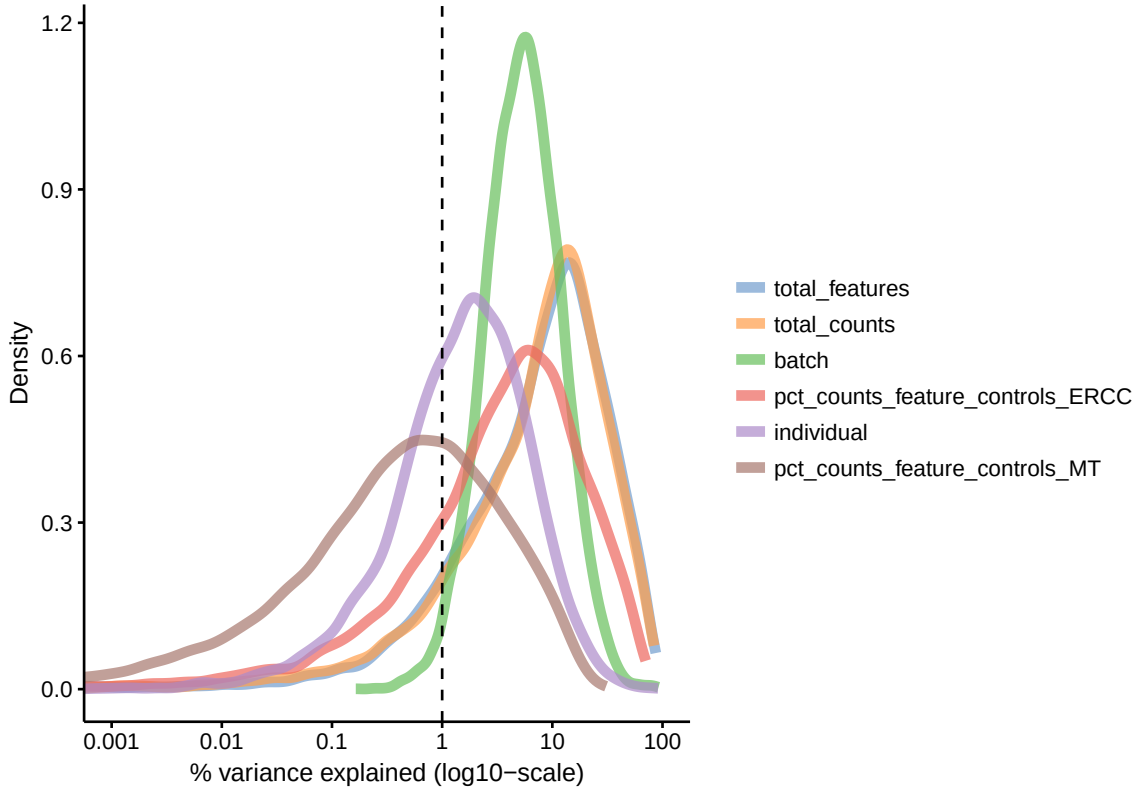


Figure 12.3: Explanatory variables

```
                                "pct_counts_feature_controls_ERCC",
                                "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. One notable feature of the above plot is that batch explains more than individual, what does that tell us about the technical and biological variability of the data?

12.4 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.

12.5 Exercise

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

Chapter 13

Identifying confounding factors (Reads)

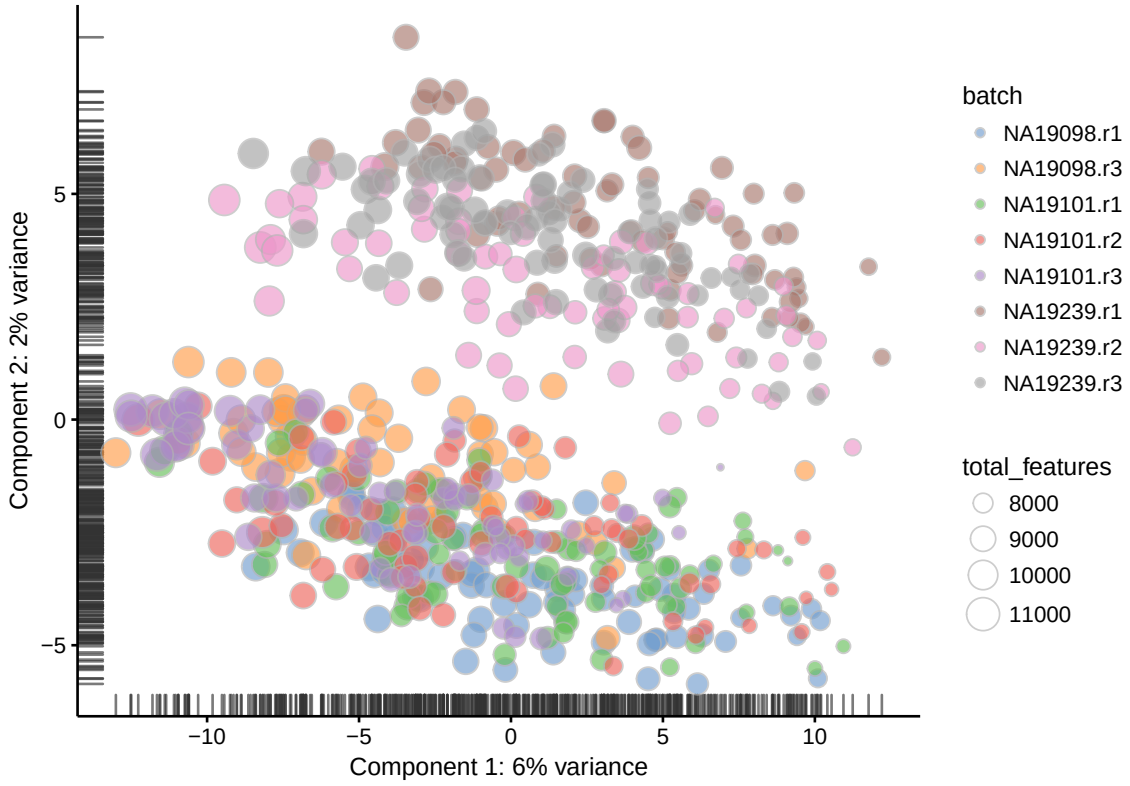


Figure 13.1: PCA plot of the tung data

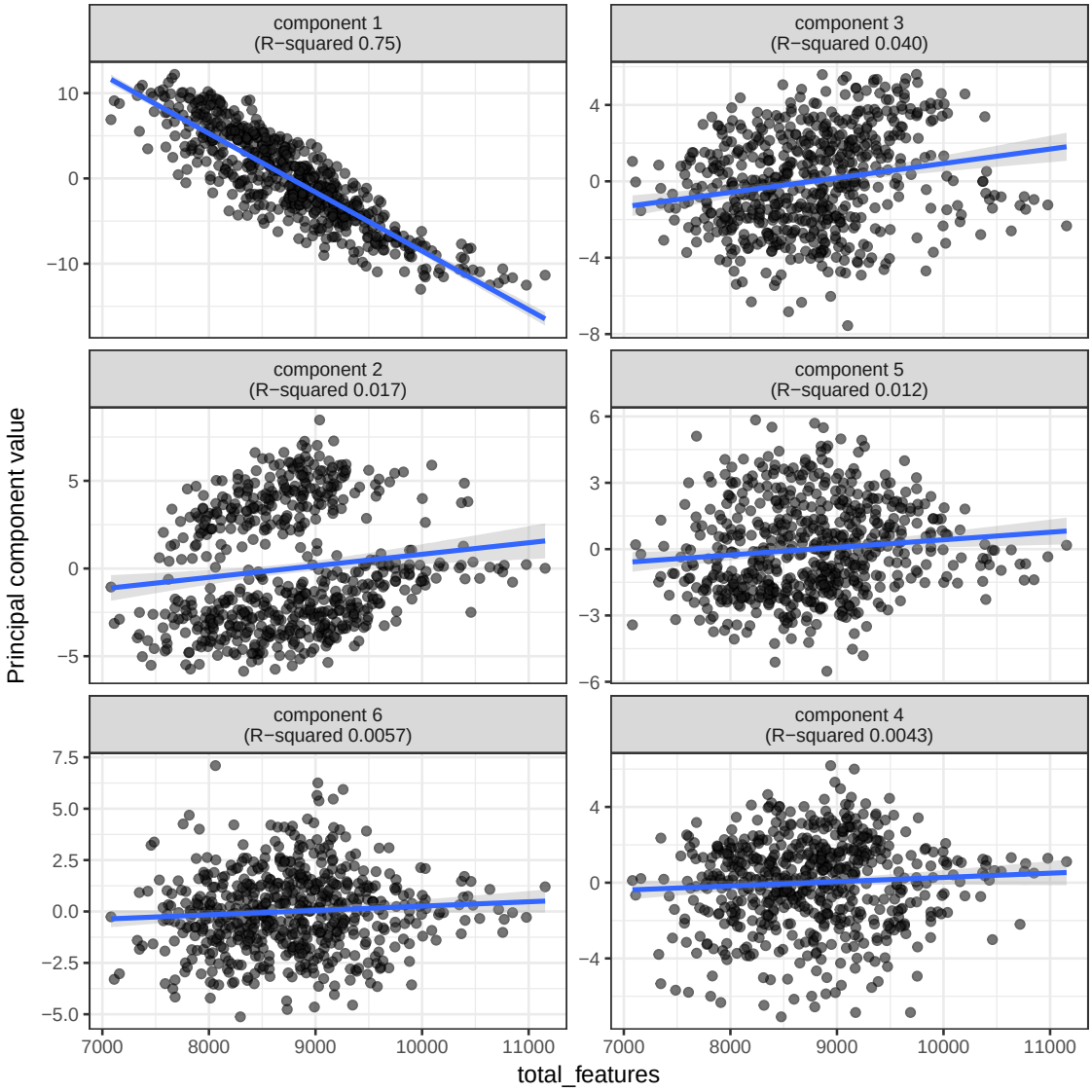


Figure 13.2: PC correlation with the number of detected genes

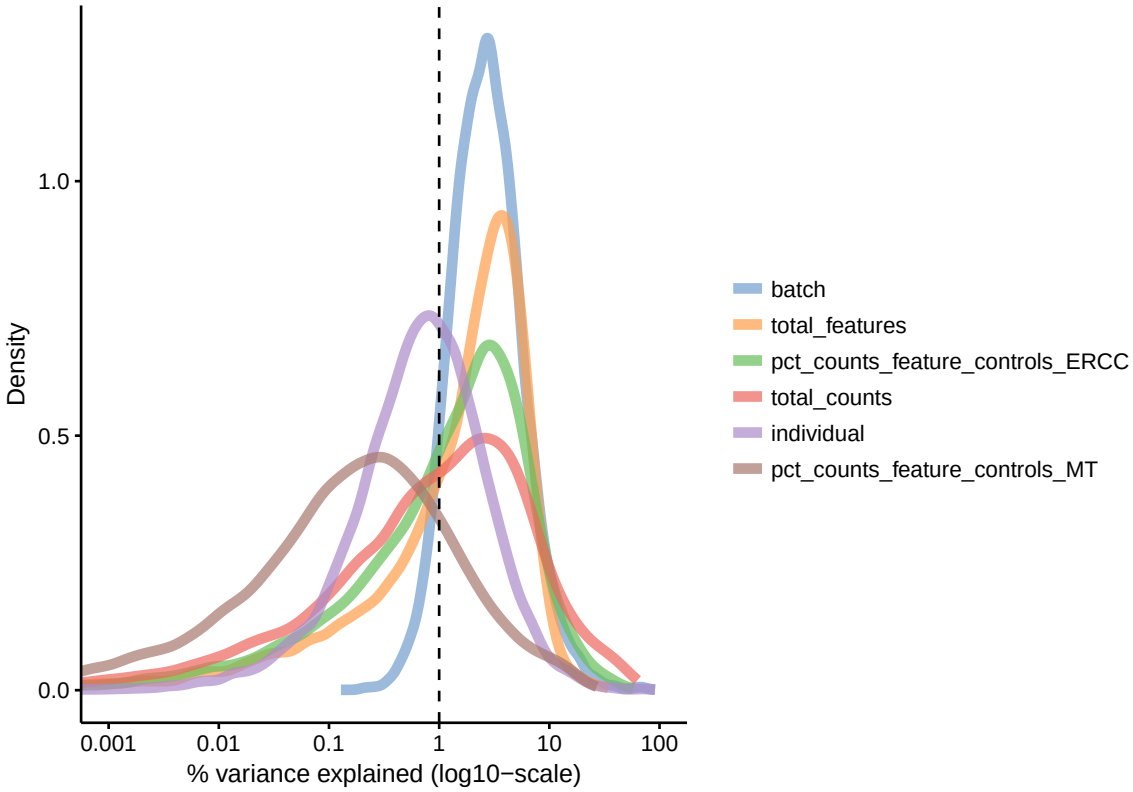


Figure 13.3: Explanatory variables

Chapter 14

Normalization for library size

14.1 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.

14.2 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).

14.3 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 `tung` data that was used in the previous chapter.

```
library(scRNA.seq.funcs)
library(scater)
library(scran)
options(stringsAsFactors = FALSE)
set.seed(1234567)
umi <- readRDS("tung/umi.rds")
umi.qc <- umi[fData(umi)$use, pData(umi)$use]
endog_genes <- !fData(umi.qc)$is_feature_control
```

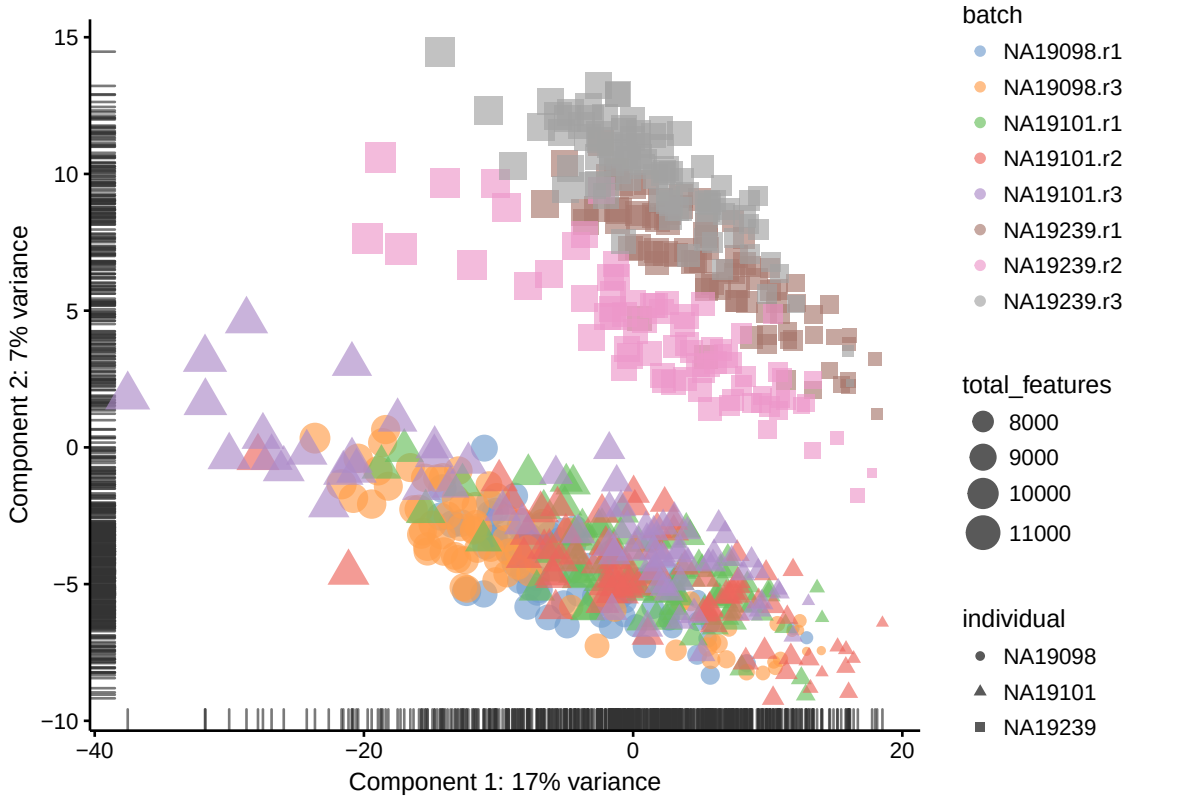


Figure 14.1: PCA plot of the tung data

14.3.1 Raw

```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "log2_counts"
)
```

14.3.2 CPM

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

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

```
plotRLE(
    umi.qc[endog_genes, ],
```

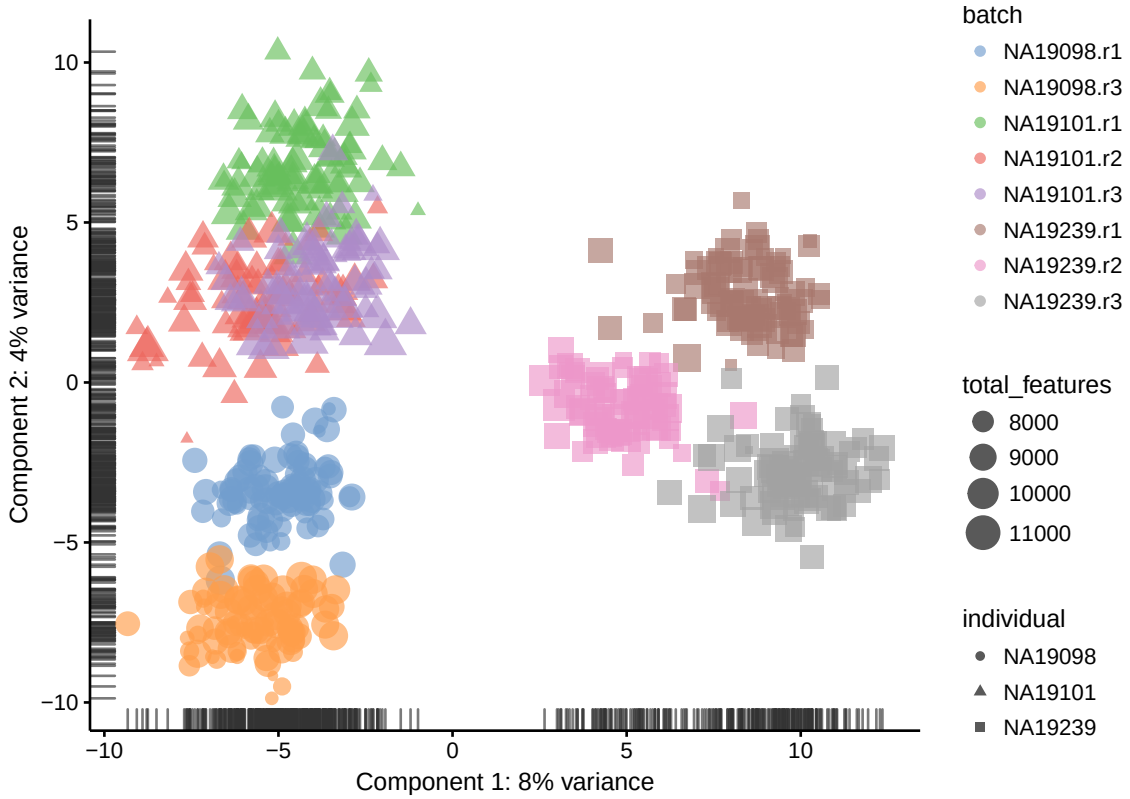


Figure 14.2: PCA plot of the tung data after CPM normalisation

```
    exprs_mats = list(Raw = "log2_counts", CPM = "exprs"),
    exprs_logged = c(TRUE, TRUE),
    colour_by = "batch"
)
```

14.3.3 TMM

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

```
plotRLE(
    umi.qc[endog_genes, ],
    exprs_mats = list(Raw = "log2_counts", TMM = "norm_exprs"),
    exprs_logged = c(TRUE, TRUE),
```

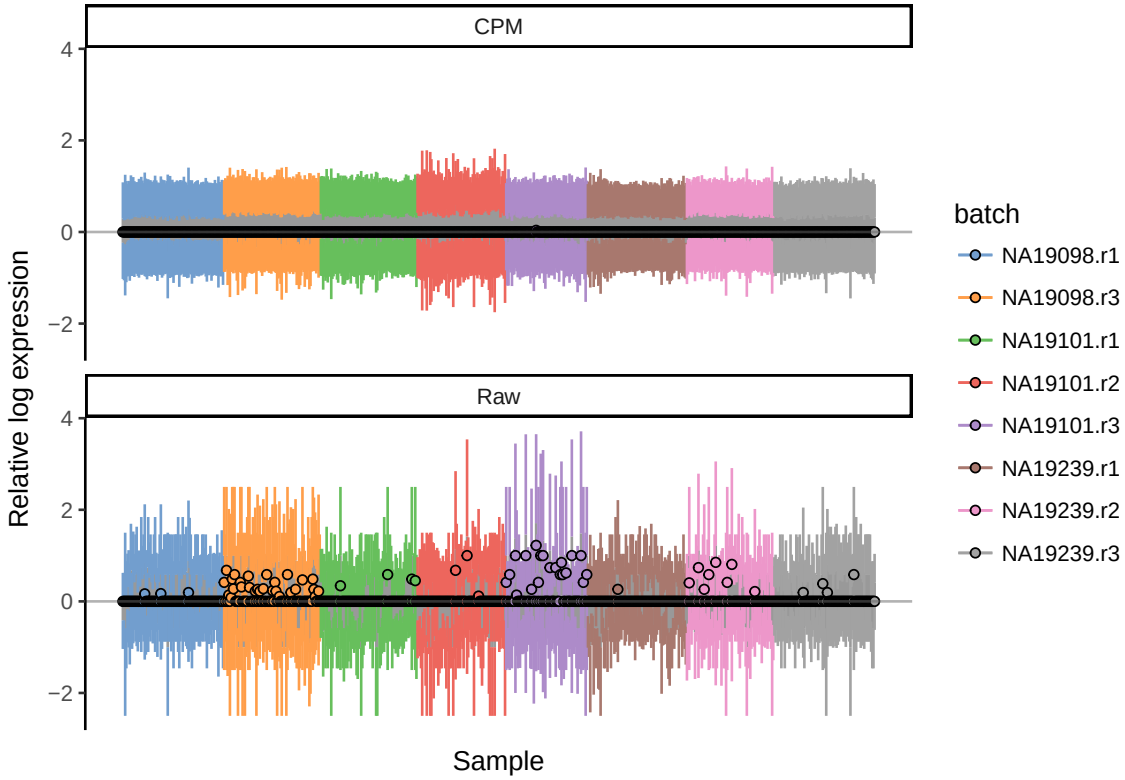


Figure 14.3: Cell-wise RLE of the tung data

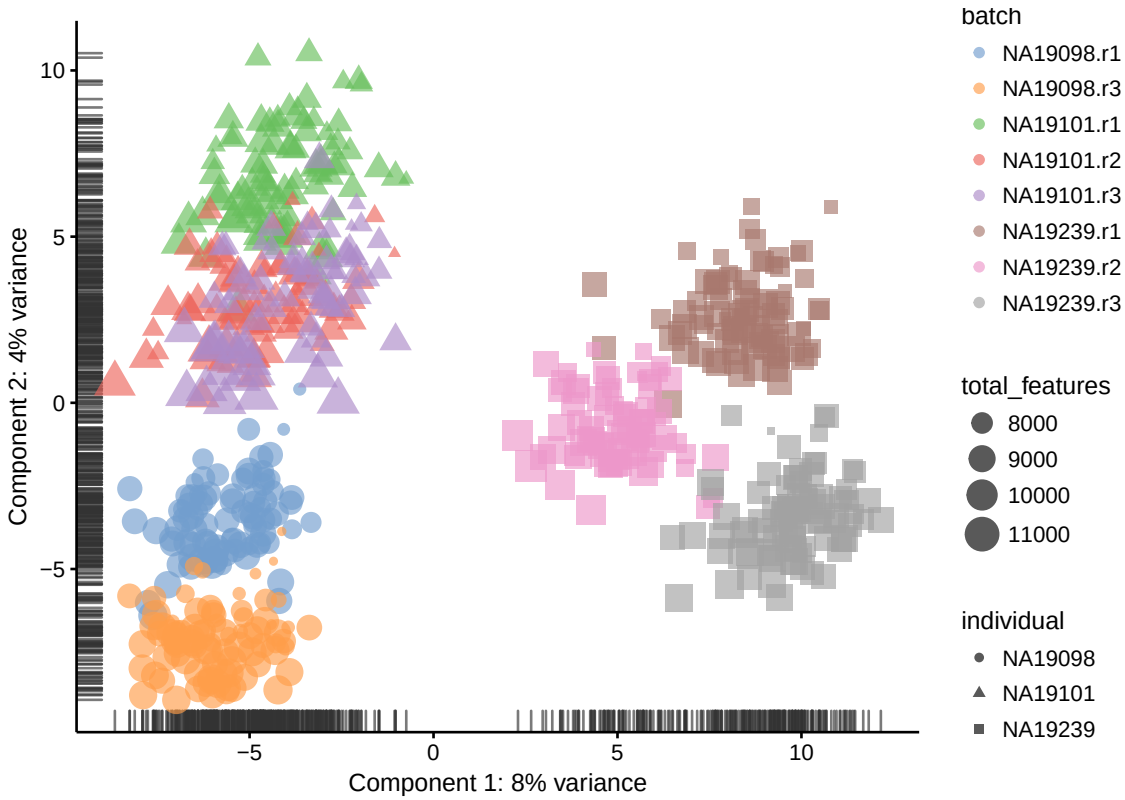


Figure 14.4: PCA plot of the tung data after TMM normalisation

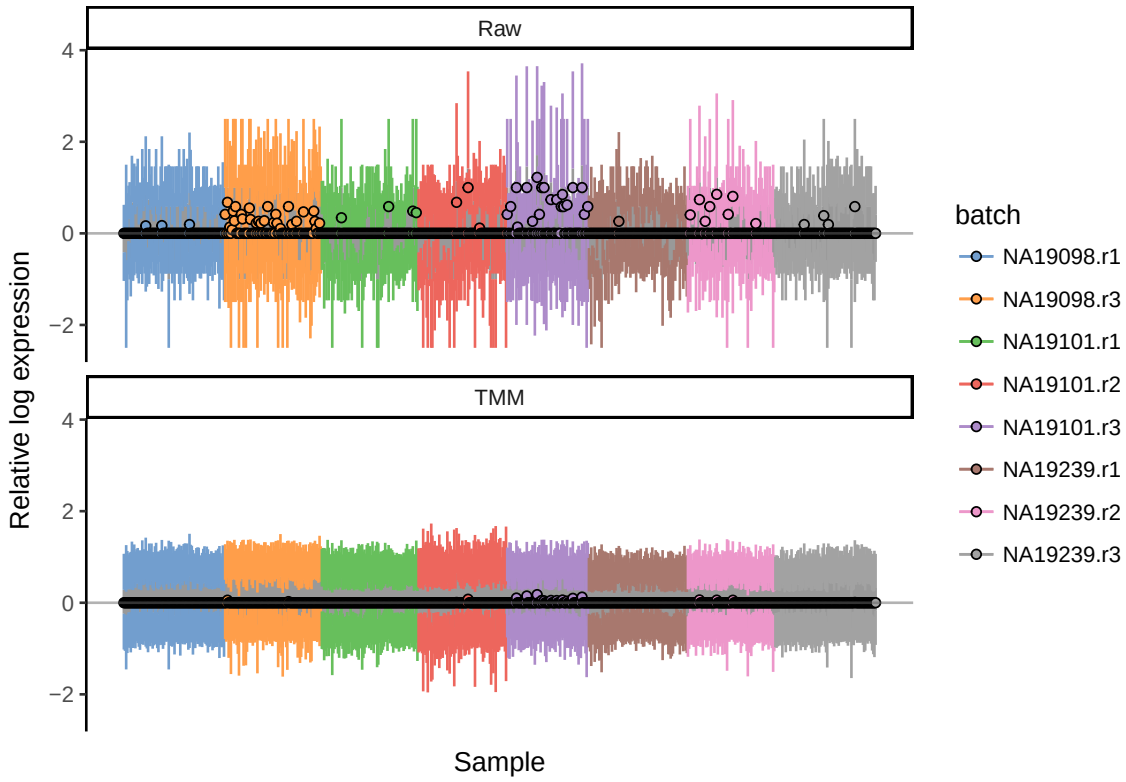


Figure 14.5: Cell-wise RLE of the tung data

```
    colour_by = "batch"
)
```

14.3.4 scran

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

```
plotRLE(
    umi.qc[endog_genes, ],
    exprs_mats = list(Raw = "log2_counts", scran = "exprs"),
    exprs_logged = c(TRUE, TRUE),
    colour_by = "batch"
)
```

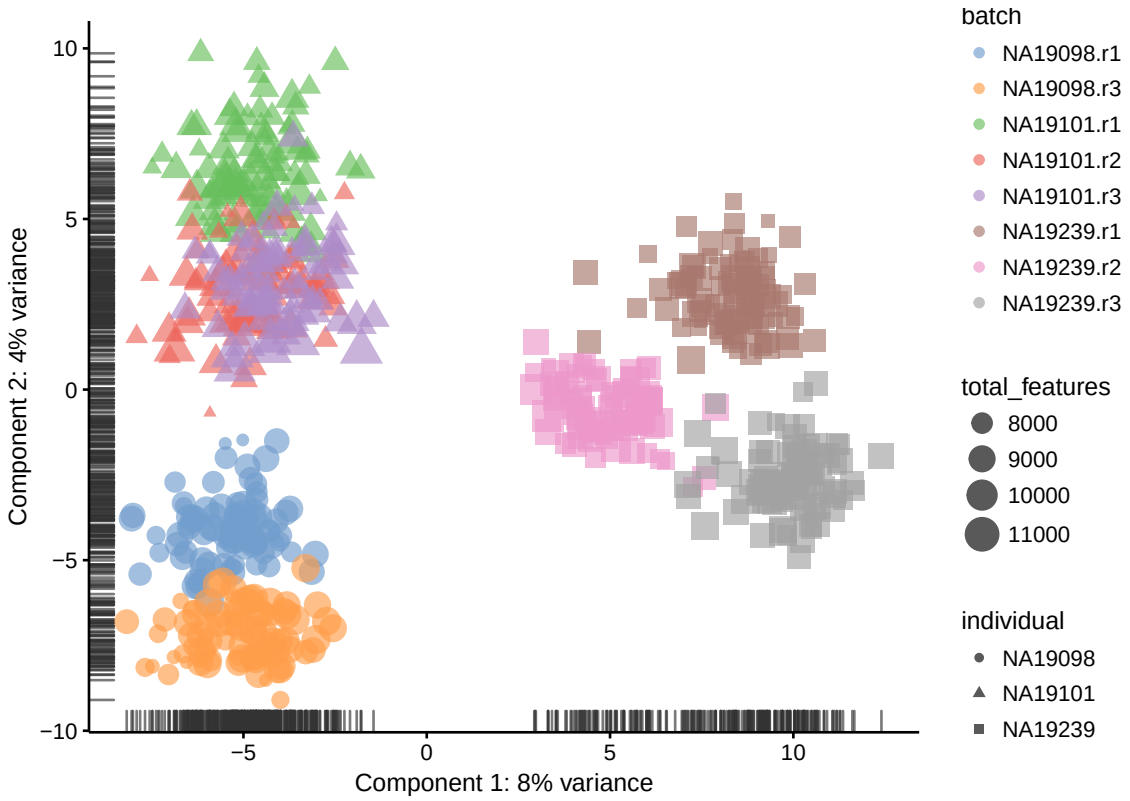


Figure 14.6: PCA plot of the tung data after LSF normalisation

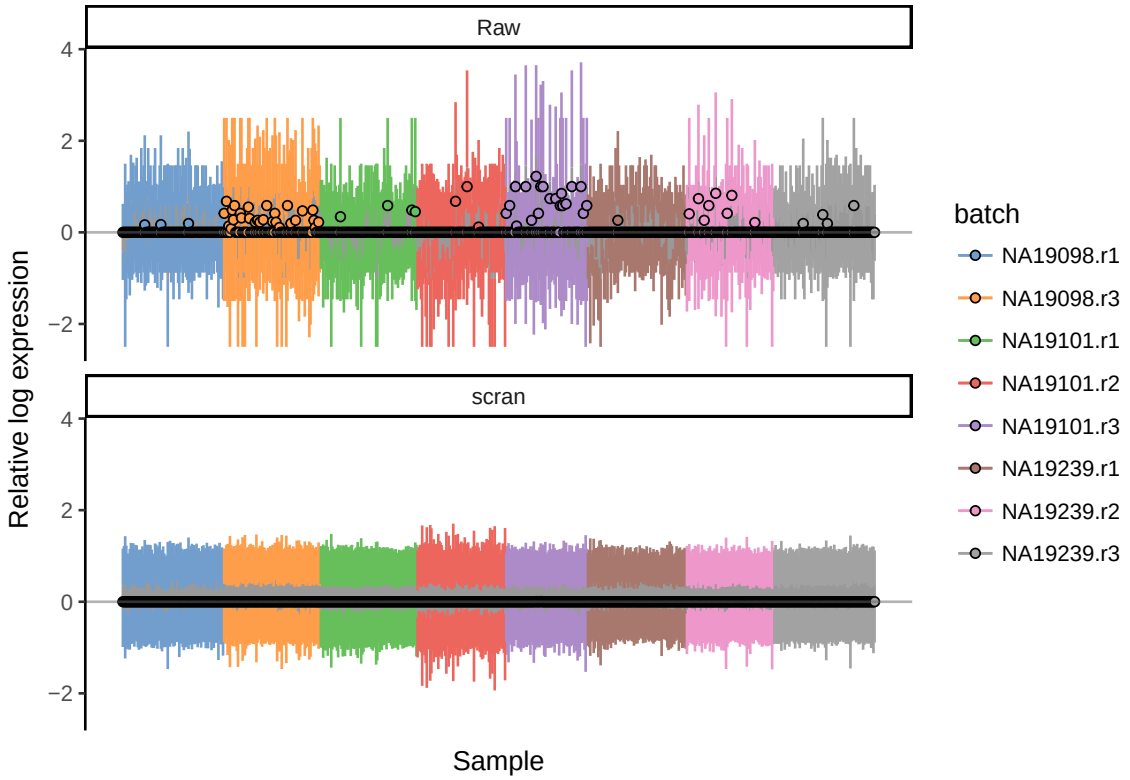


Figure 14.7: Cell-wise RLE of the tung data

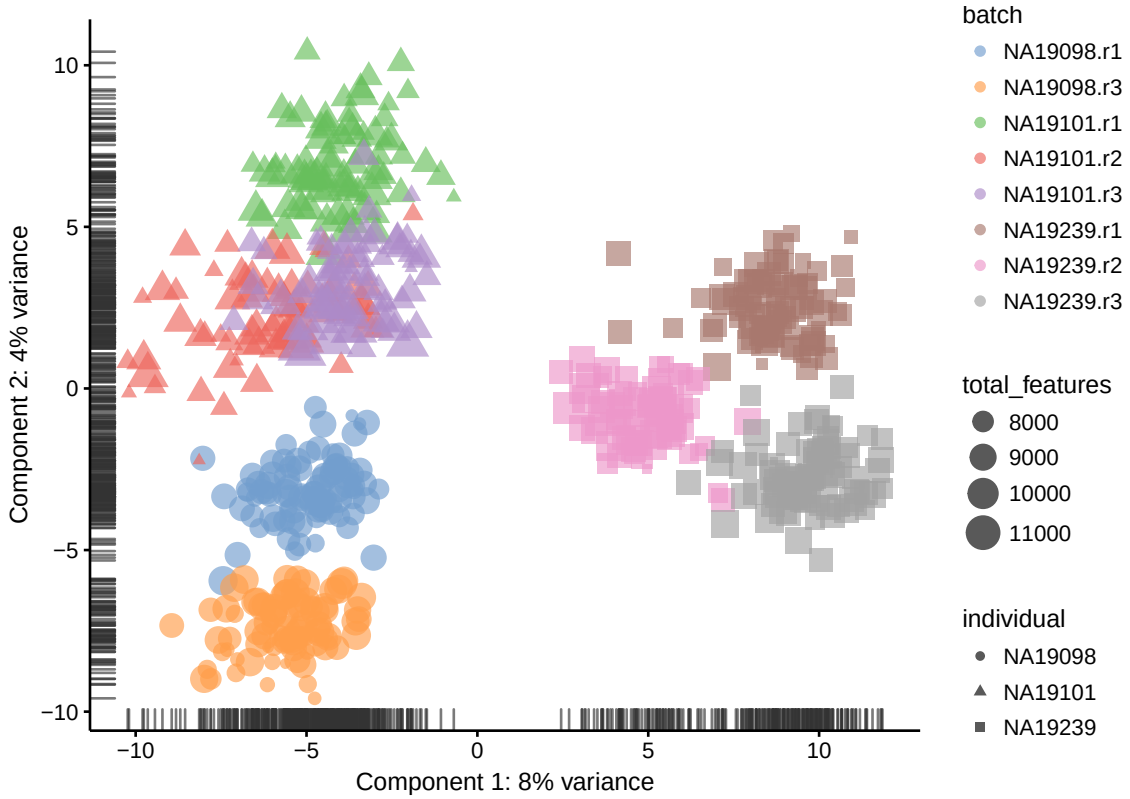


Figure 14.8: PCA plot of the tung data after RLE normalisation

14.3.5 Size-factor (RLE)

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

```
plotRLE(
    umi.qc[endog_genes, ],
    exprs_mats = list(Raw = "log2_counts", RLE = "norm_exprs"),
    exprs_logged = c(TRUE, TRUE),
    colour_by = "batch"
)
```

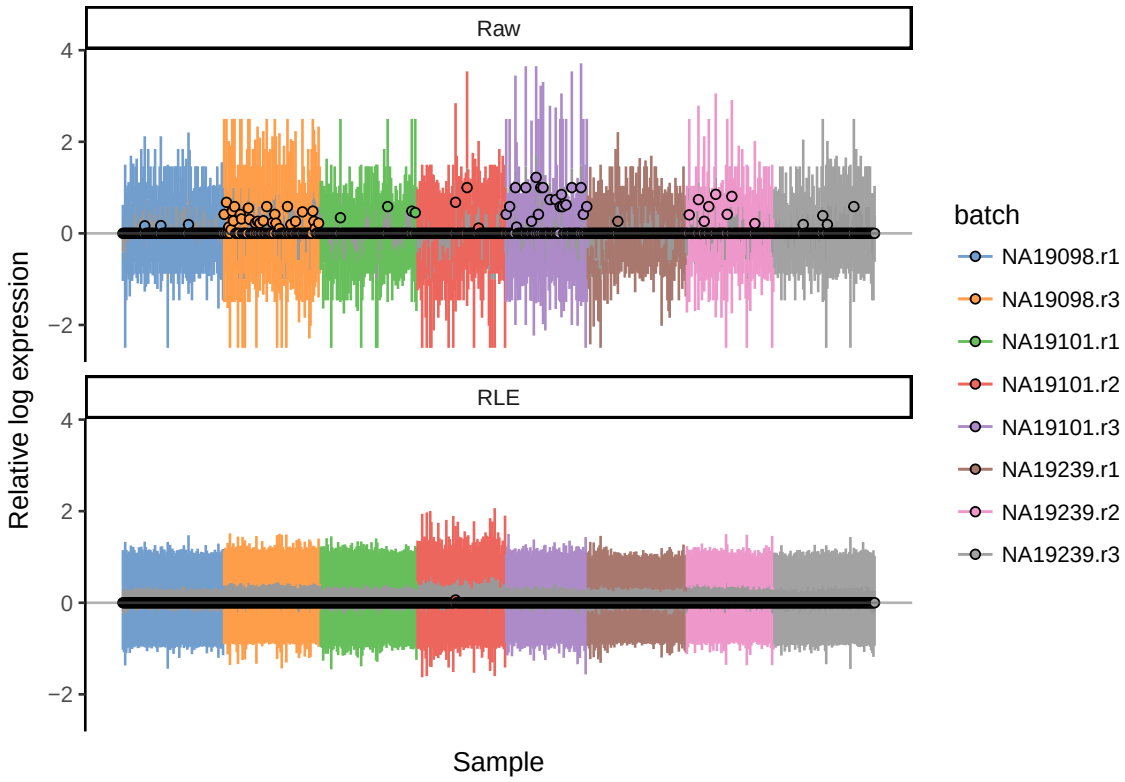


Figure 14.9: Cell-wise RLE of the tung data

14.3.6 Upperquantile

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

```
plotRLE(
    umi.qc[endog_genes, ],
    exprs_mats = list(Raw = "log2_counts", UQ = "norm_exprs"),
    exprs_logged = c(TRUE, TRUE),
    colour_by = "batch"
)
```

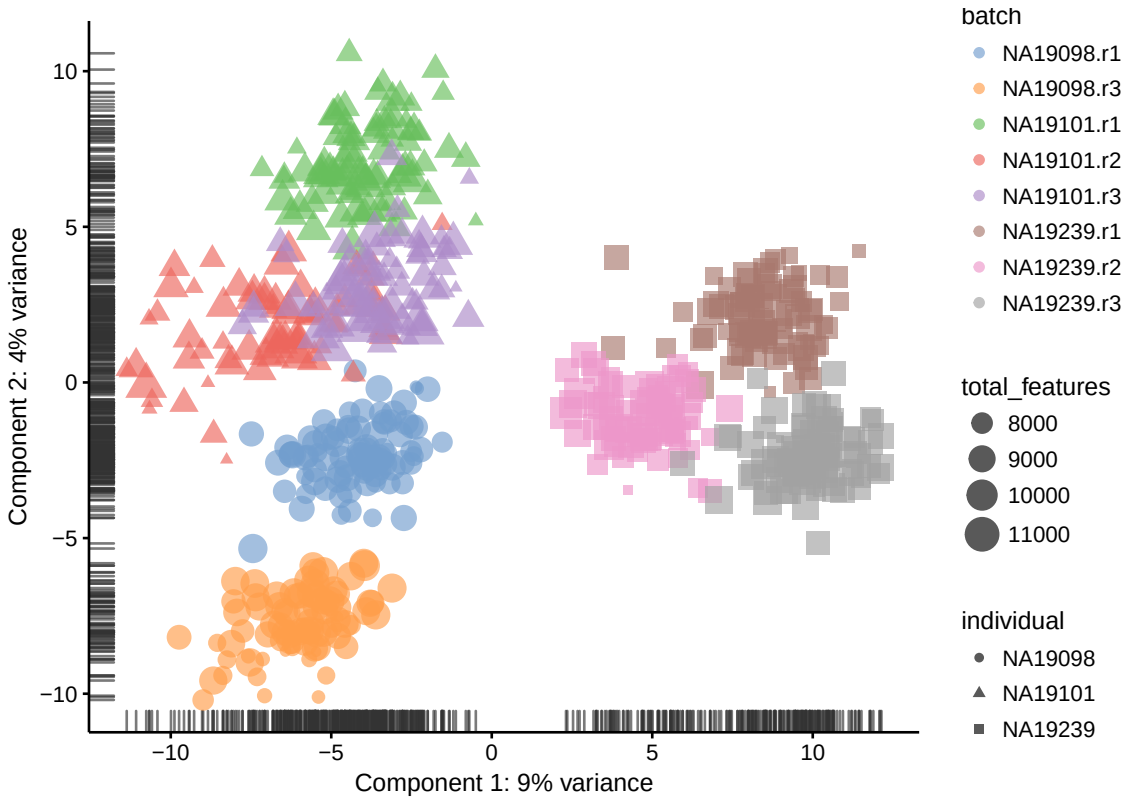


Figure 14.10: PCA plot of the tung data after UQ normalisation

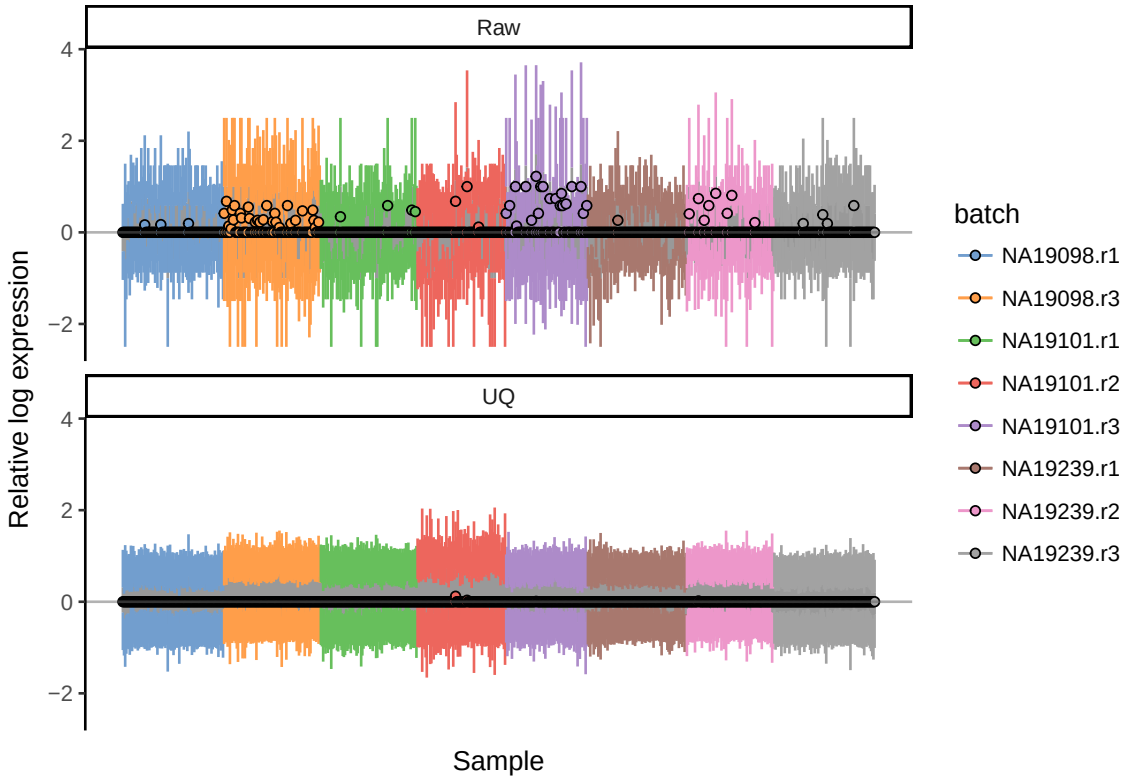


Figure 14.11: Cell-wise RLE of the tung data

14.4 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)
        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) <- log2(Down_Sample_Matrix(counts(umi.qc)) + 1)
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "norm_counts"
)
```

```
plotRLE(
    umi.qc[endog_genes, ],
    exprs_mats = list(Raw = "log2_counts", DownSample = "norm_counts"),
    exprs_logged = c(TRUE, TRUE),
    colour_by = "batch"
)
```

14.5 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.

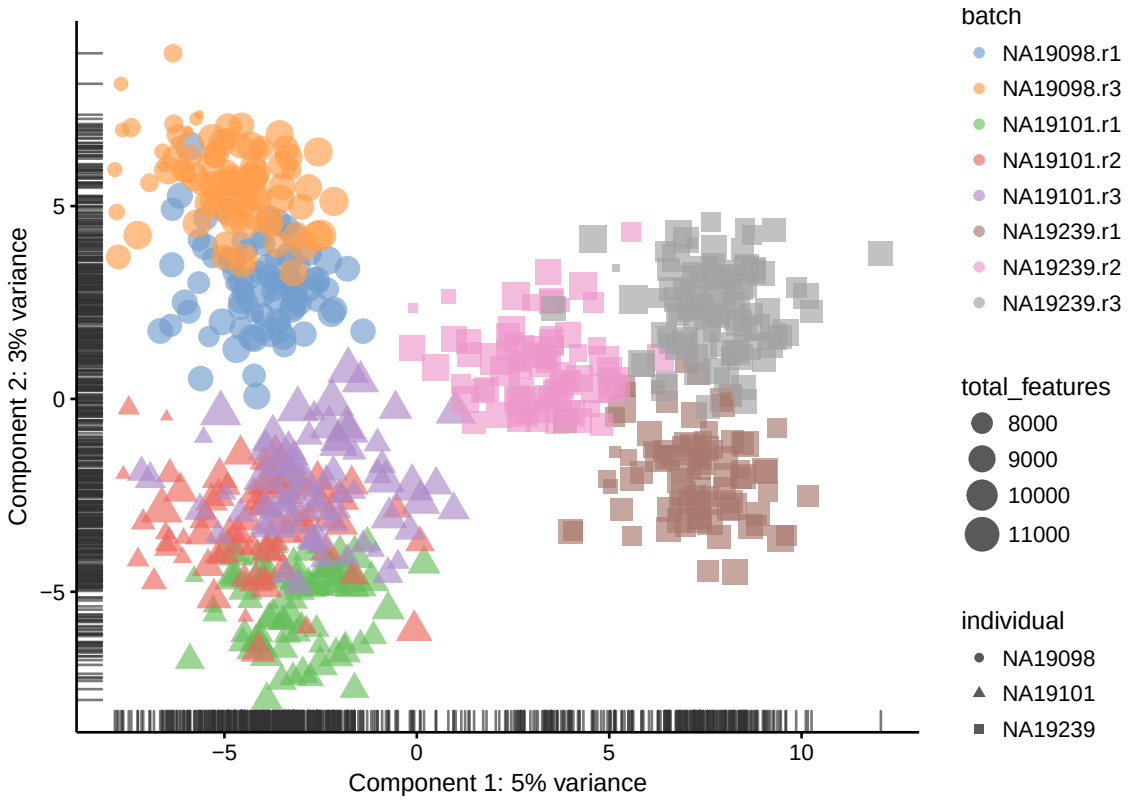


Figure 14.12: PCA plot of the tung data after downsampling

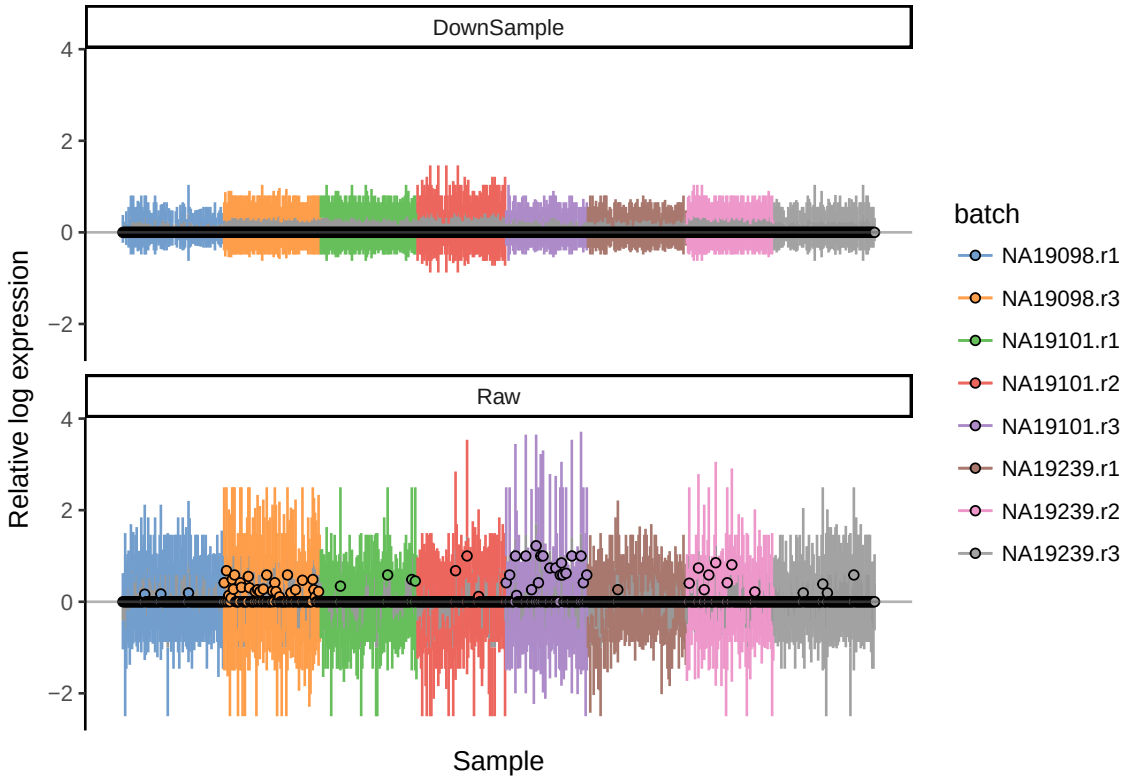


Figure 14.13: Cell-wise RLE of the tung data

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 <- 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:
# 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.

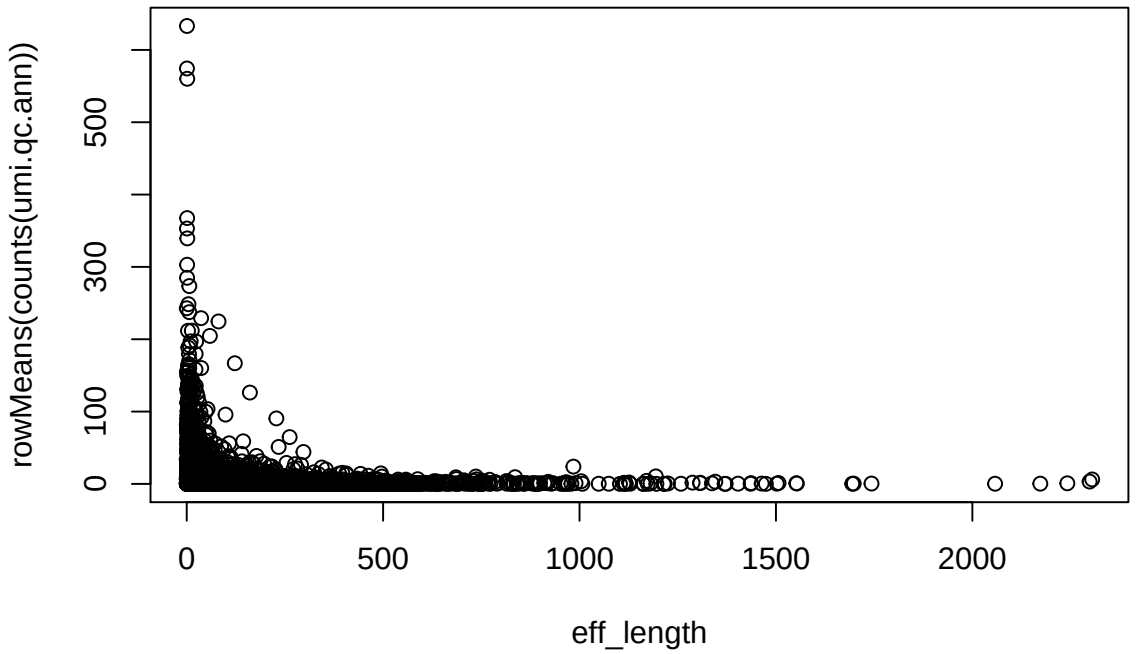


Figure 14.14: Gene length vs Mean Expression for the raw data

Now we are ready to perform the normalisations:

```
tpm(umi.qc.ann) <- log2(calculateTPM(umi.qc.ann, eff_length) + 1)
fpkm(umi.qc.ann) <- log2(calculateFPKM(umi.qc.ann, eff_length) + 1)
```

Plot the results as a PCA plot:

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

```
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 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.

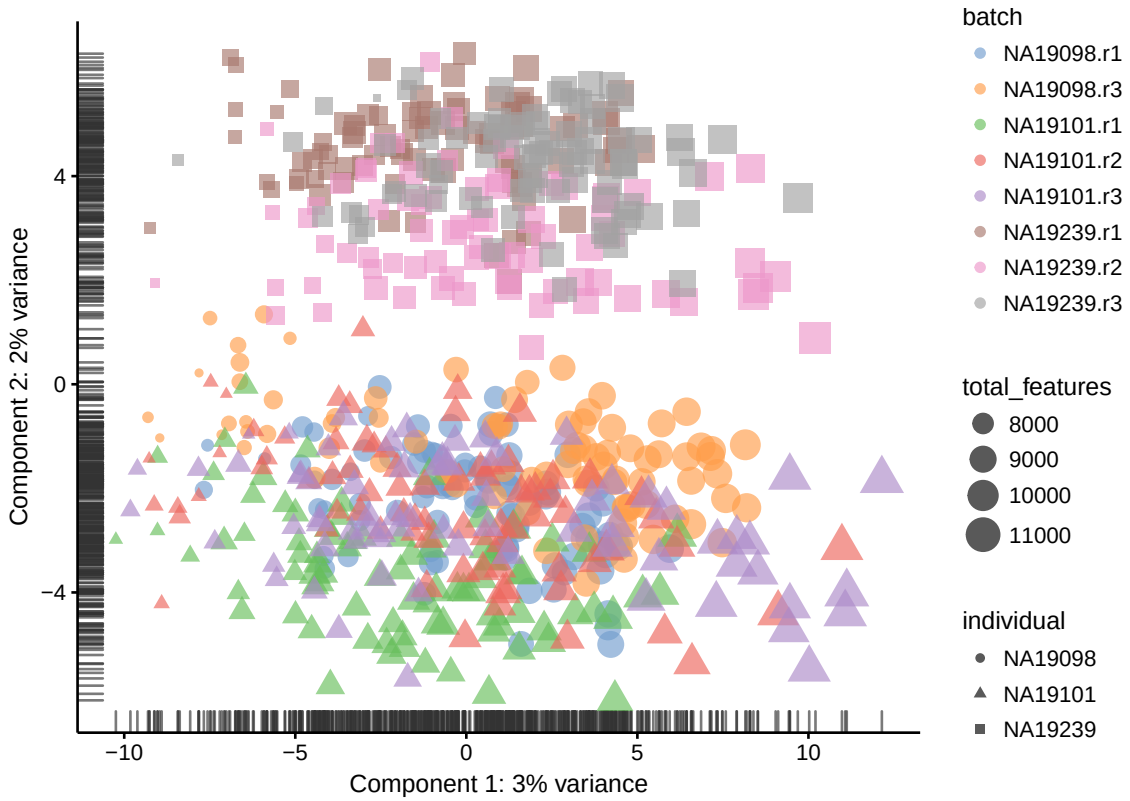


Figure 14.15: PCA plot of the tung data after FPKM normalisation

14.6 Exercise

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

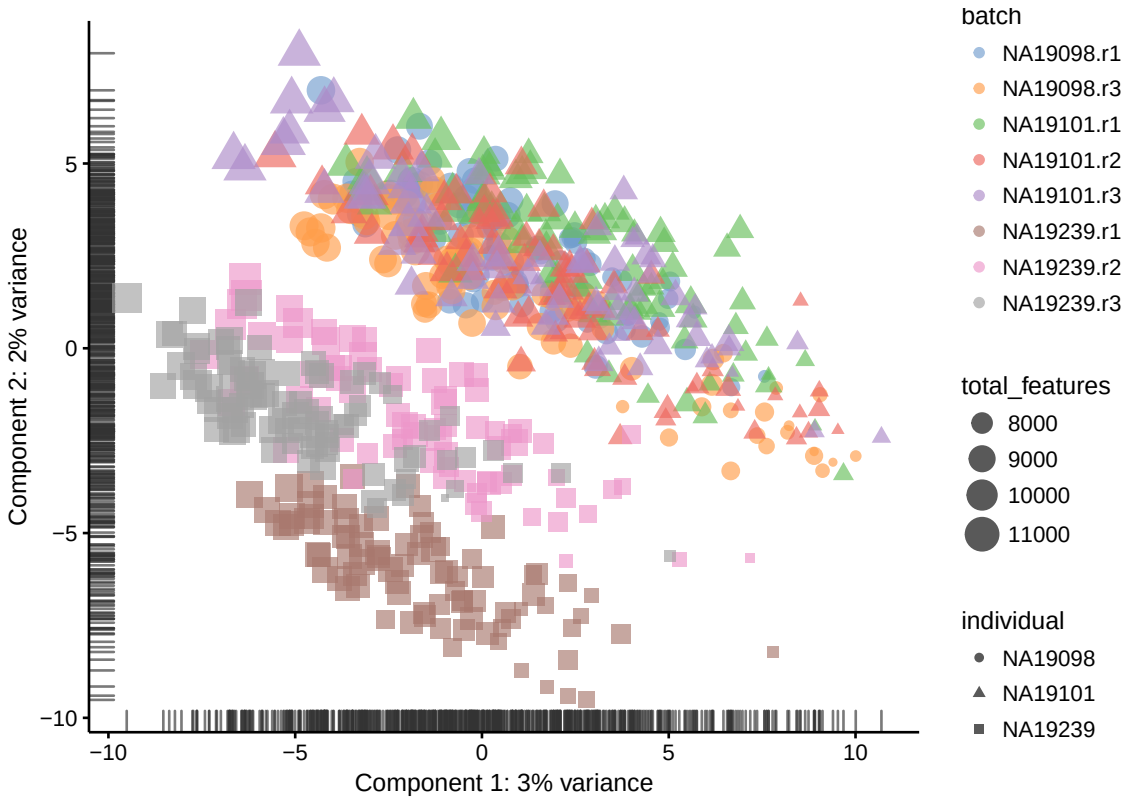


Figure 14.16: PCA plot of the tung data after TPM normalisation

Chapter 15

Normalization for library size (Reads)

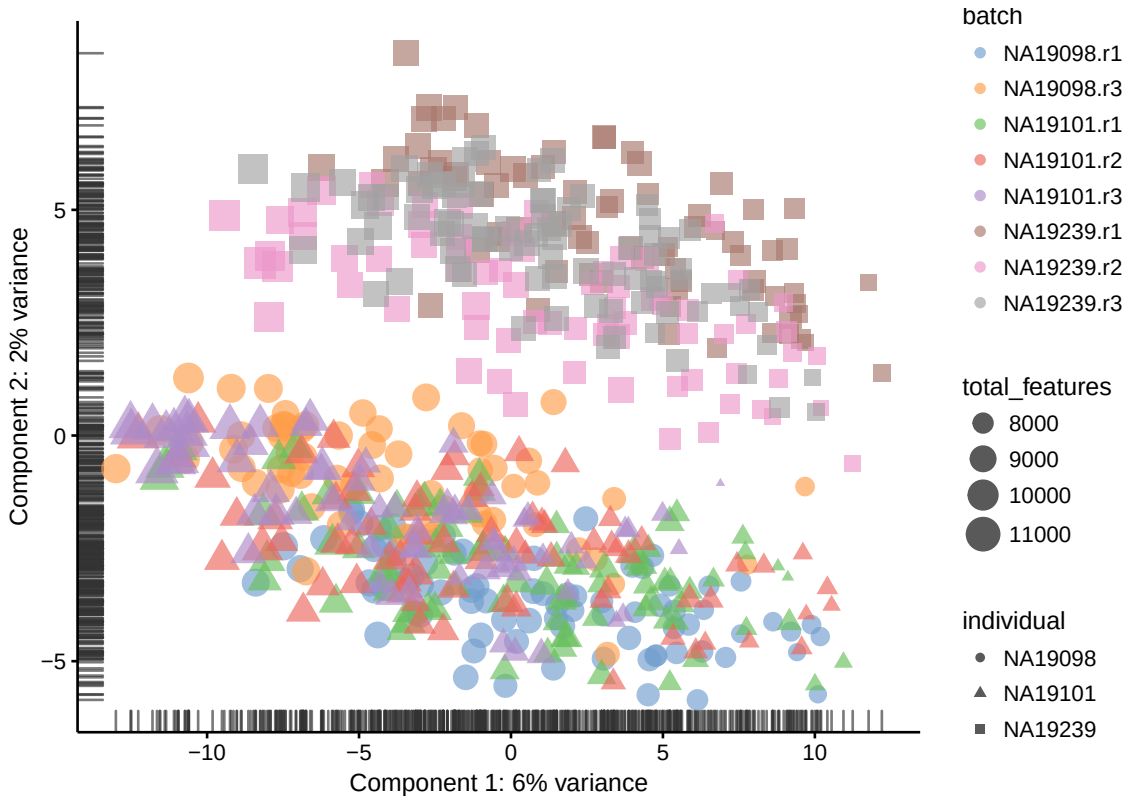


Figure 15.1: PCA plot of the tung data

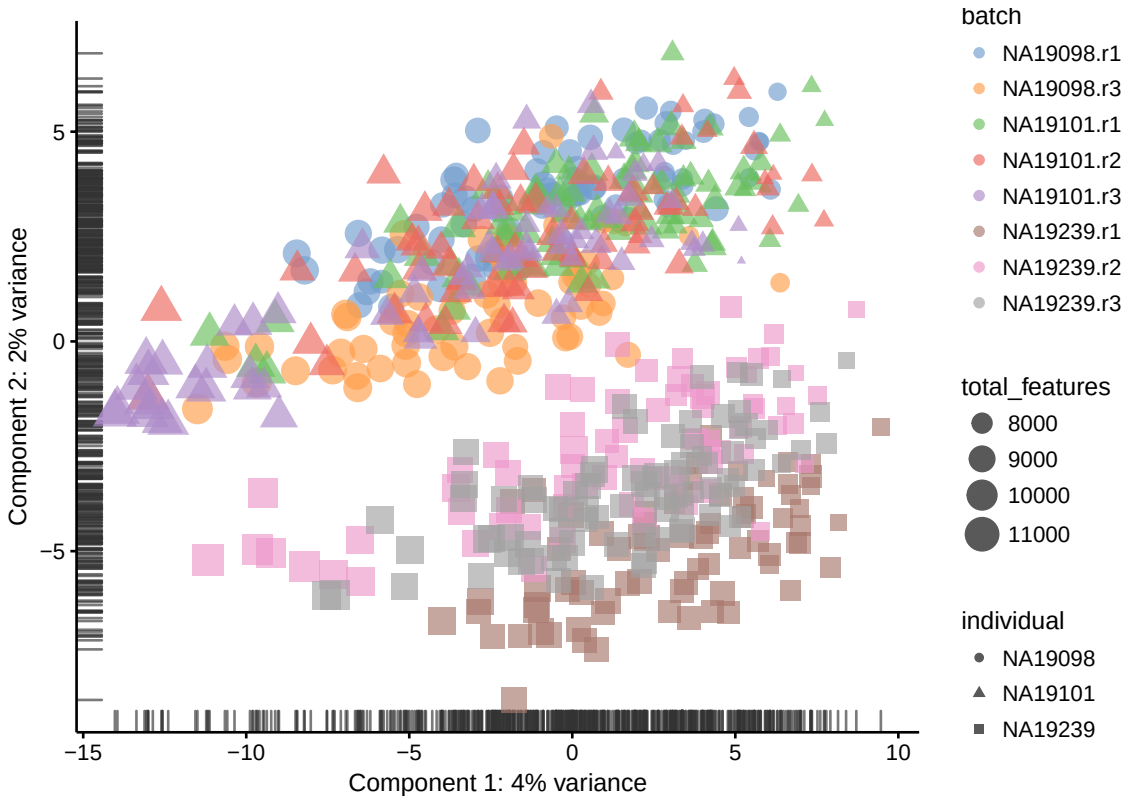


Figure 15.2: PCA plot of the tung data after CPM normalisation

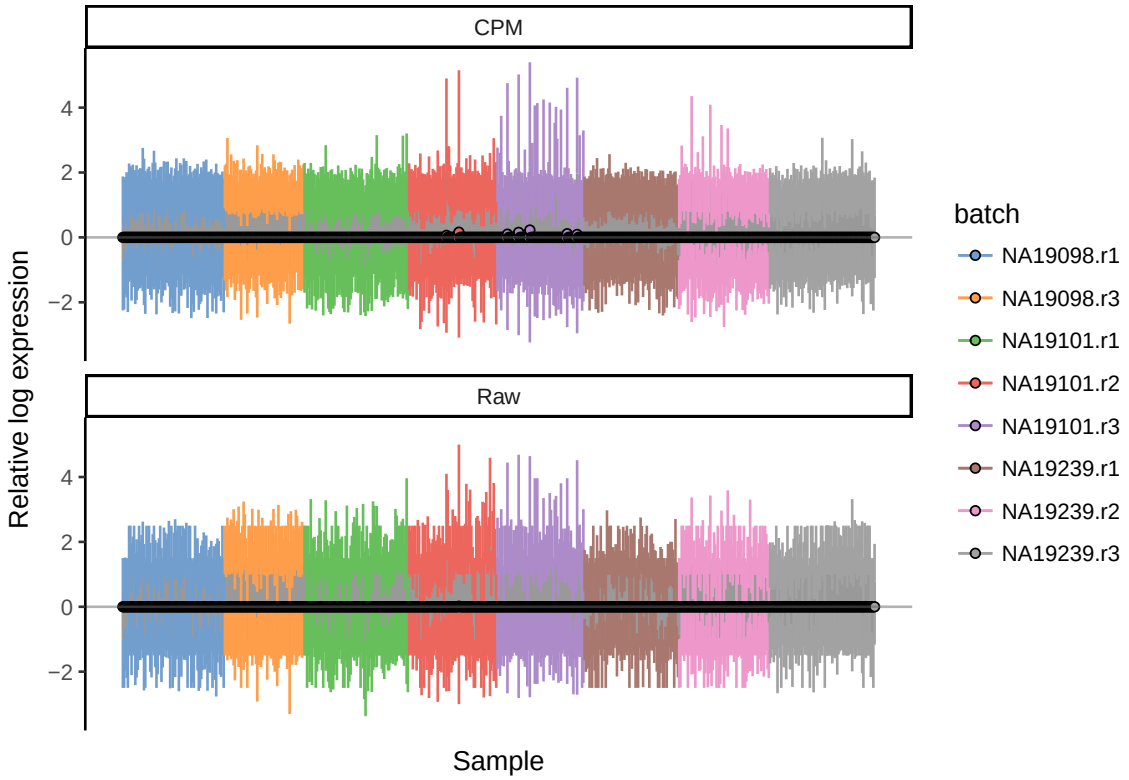


Figure 15.3: Cell-wise RLE of the tung data

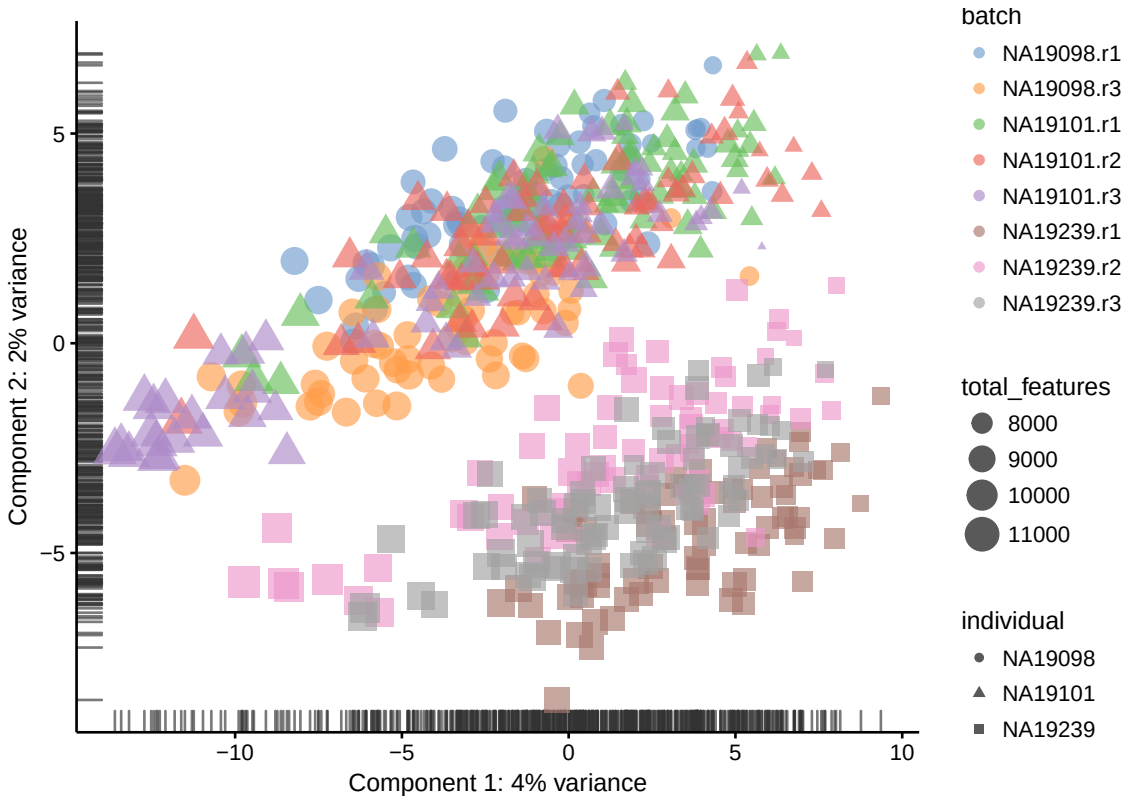


Figure 15.4: PCA plot of the tung data after TMM normalisation

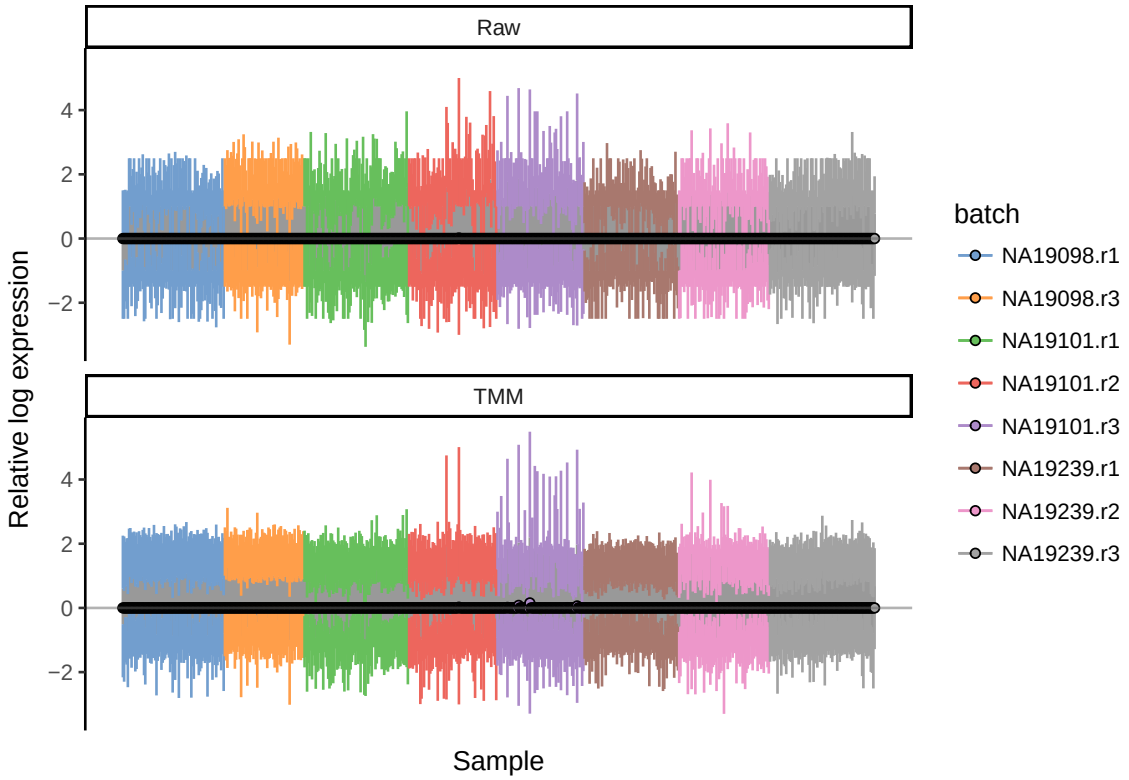


Figure 15.5: Cell-wise RLE of the tung data

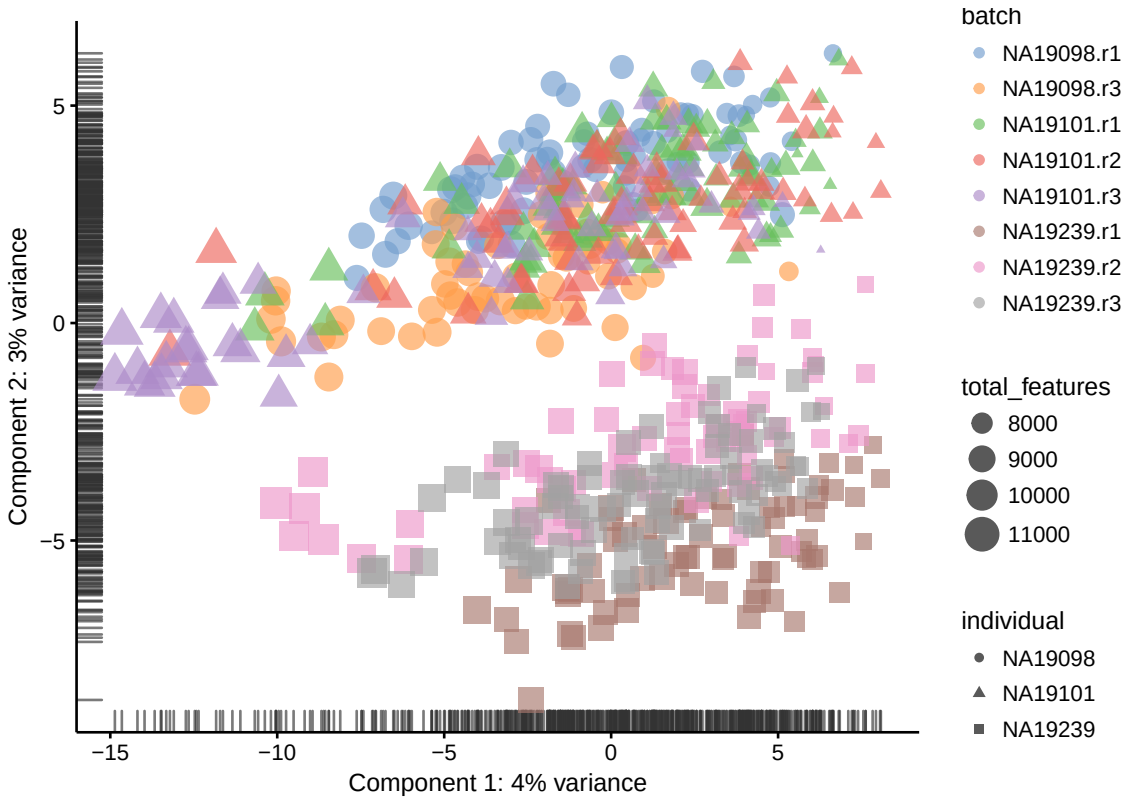


Figure 15.6: PCA plot of the tung data after LSF normalisation

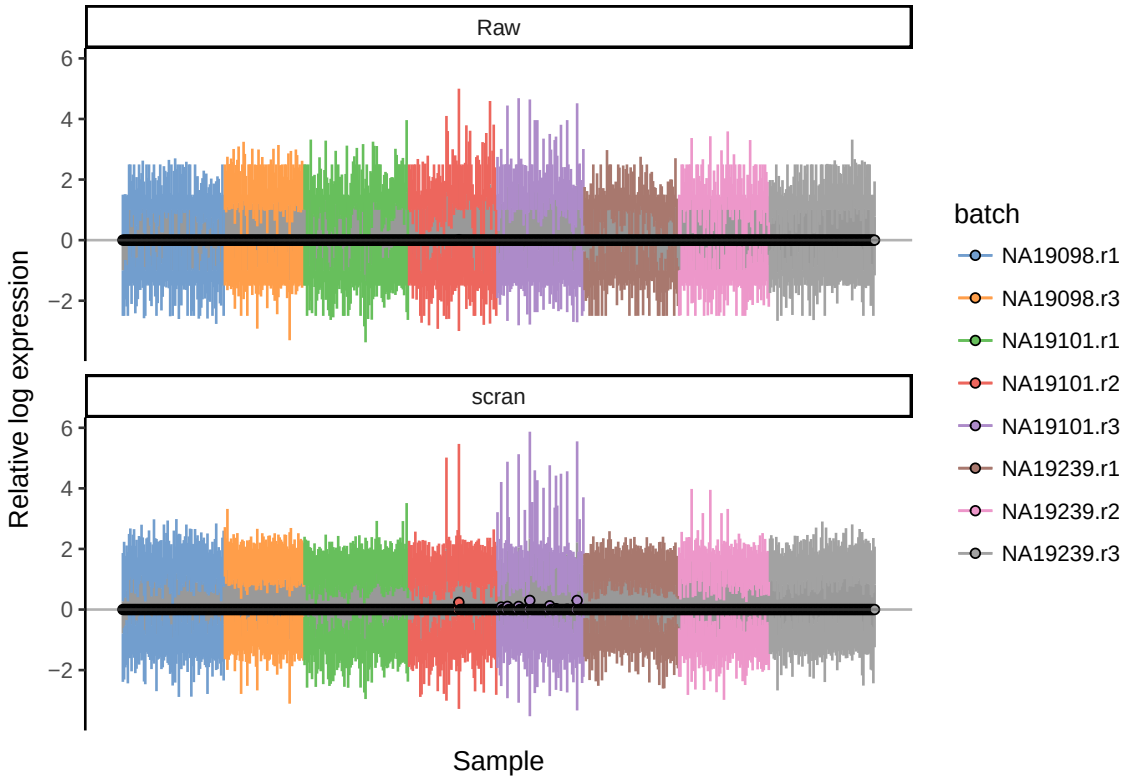


Figure 15.7: Cell-wise RLE of the tung data

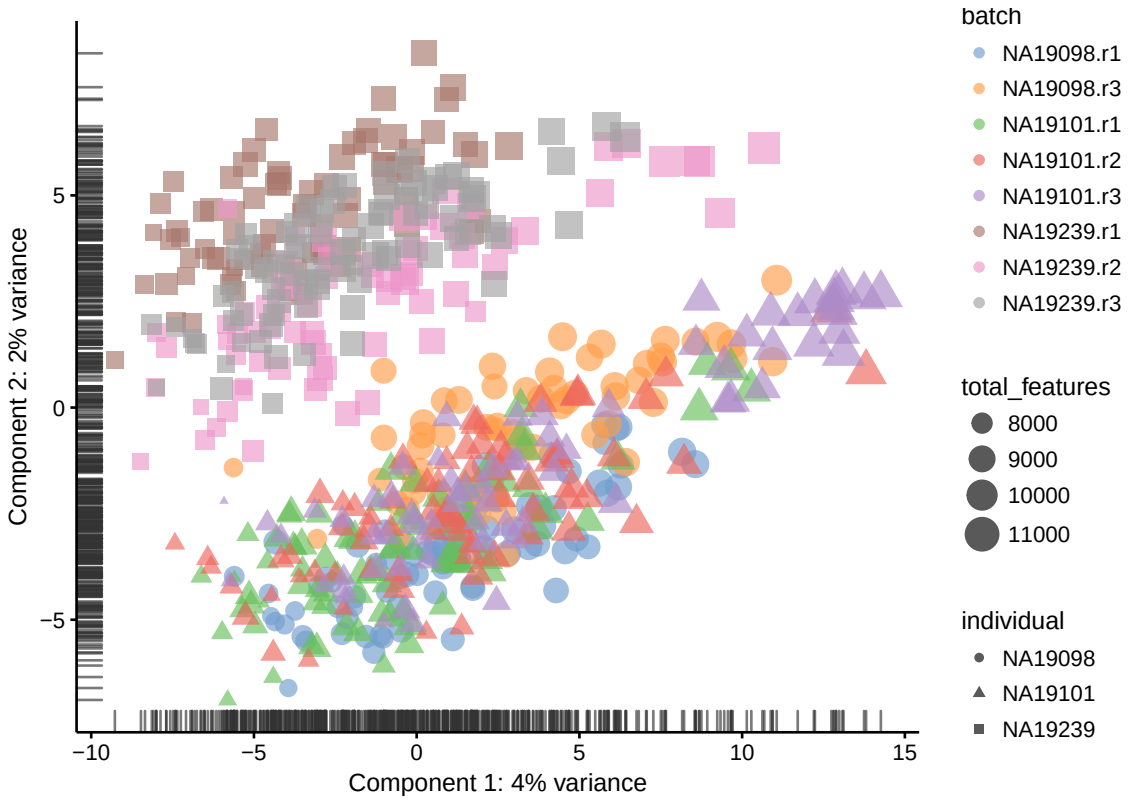


Figure 15.8: PCA plot of the tung data after RLE normalisation

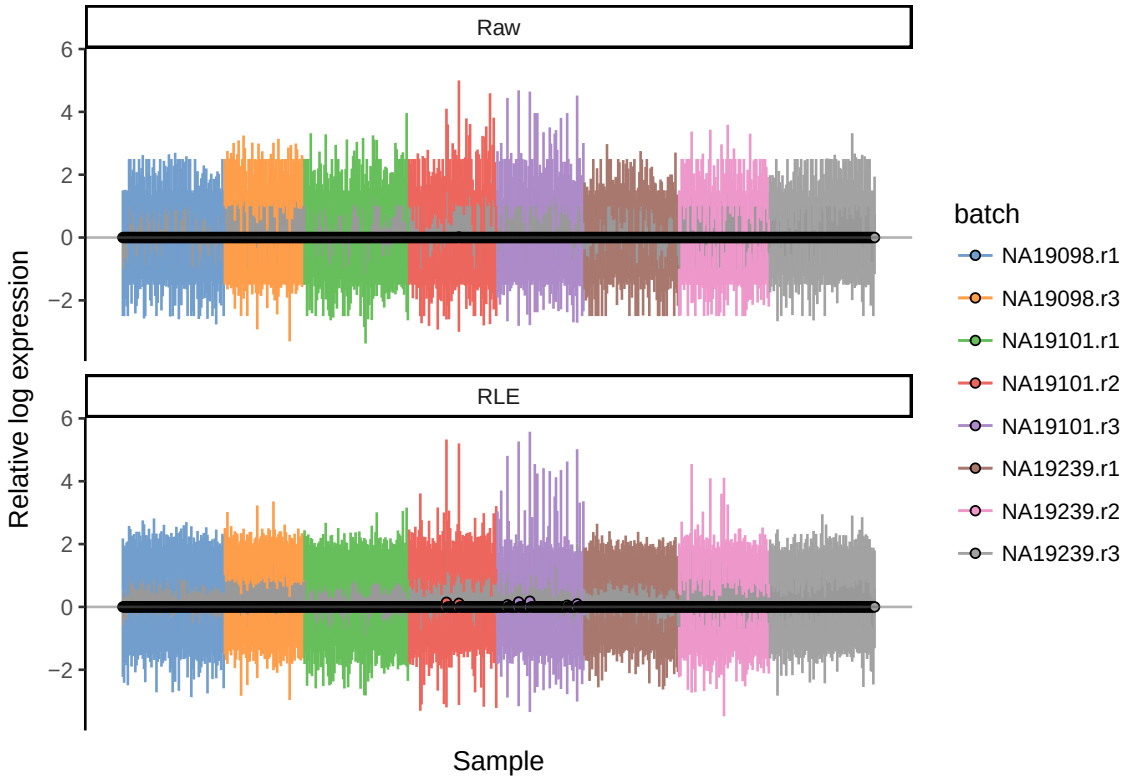


Figure 15.9: Cell-wise RLE of the tung data

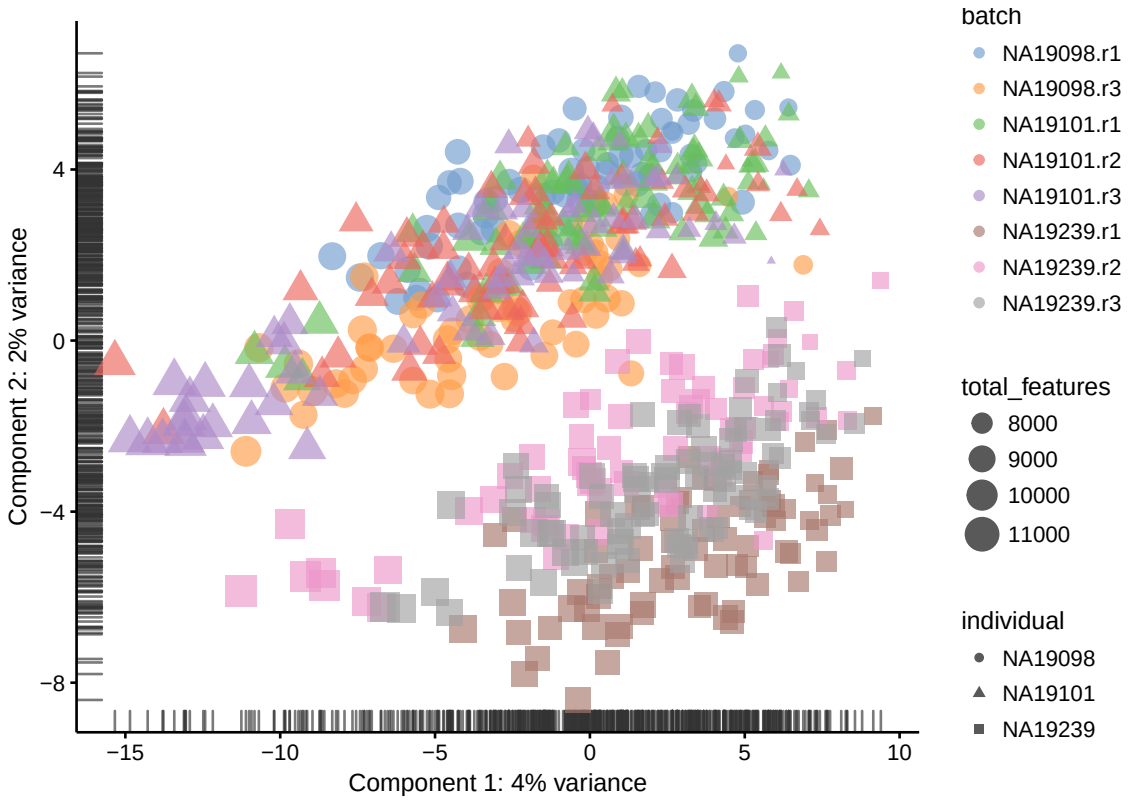


Figure 15.10: PCA plot of the tung data after UQ normalisation

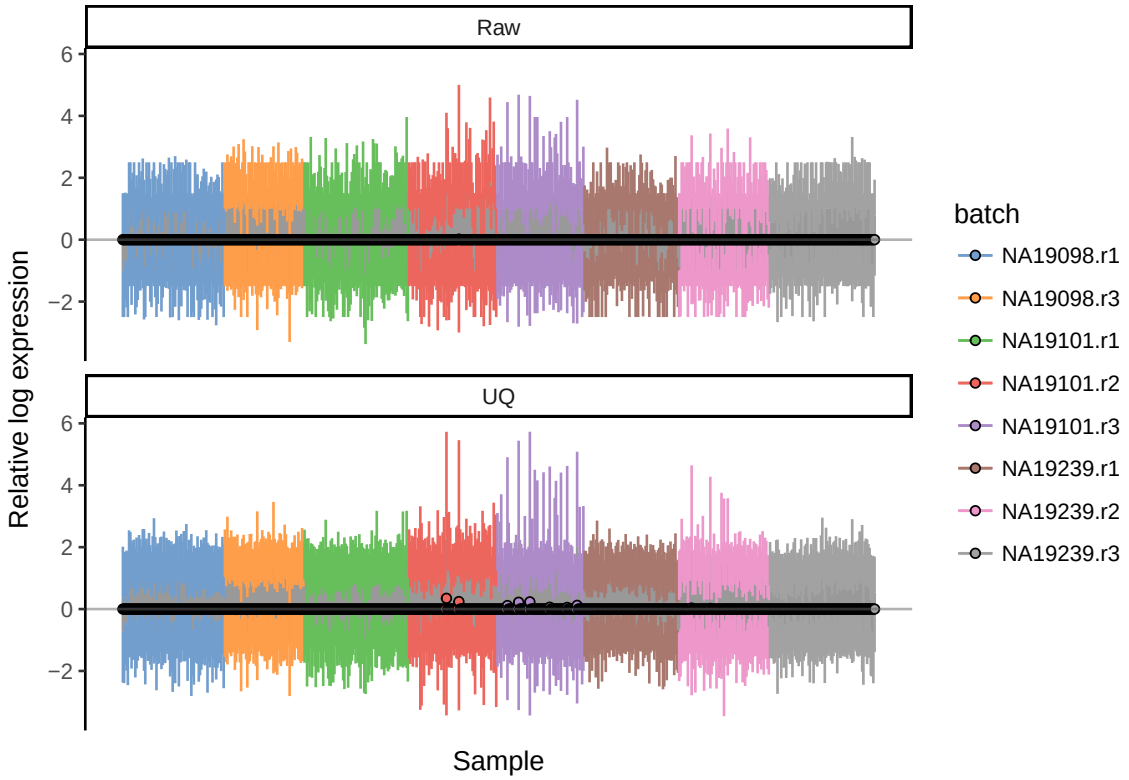


Figure 15.11: Cell-wise RLE of the tung data

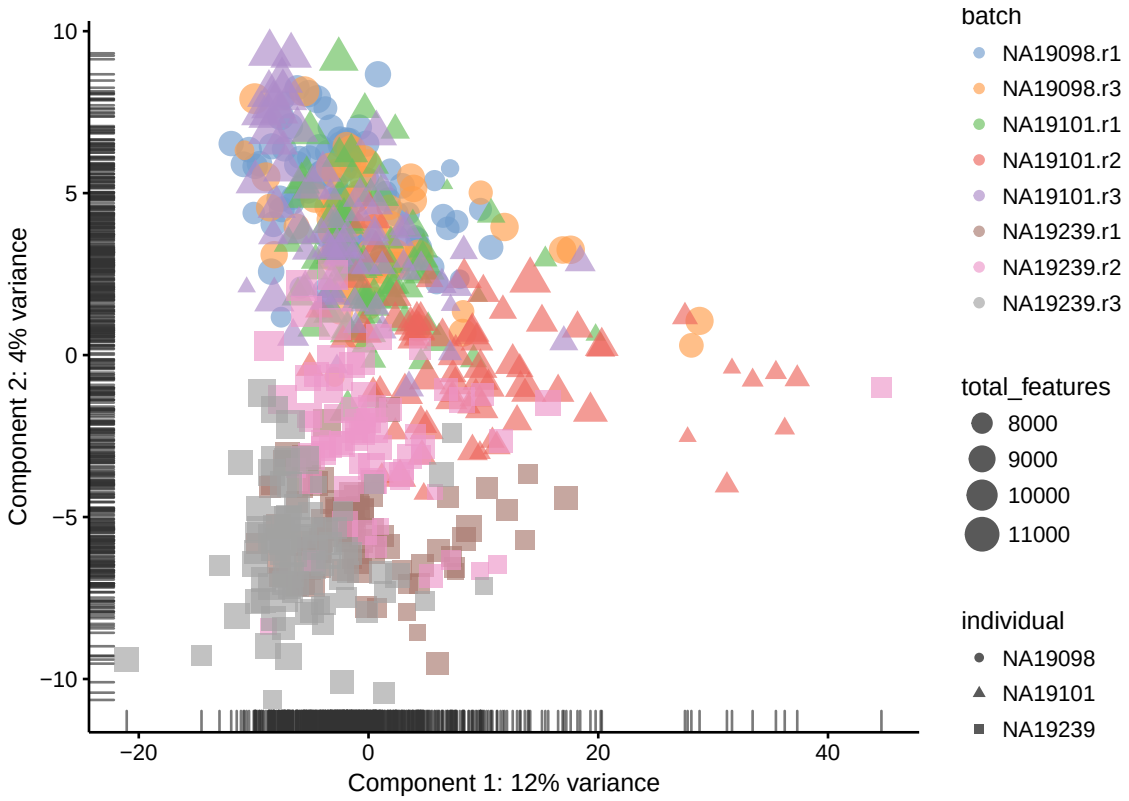


Figure 15.12: PCA plot of the tung data after downsampling

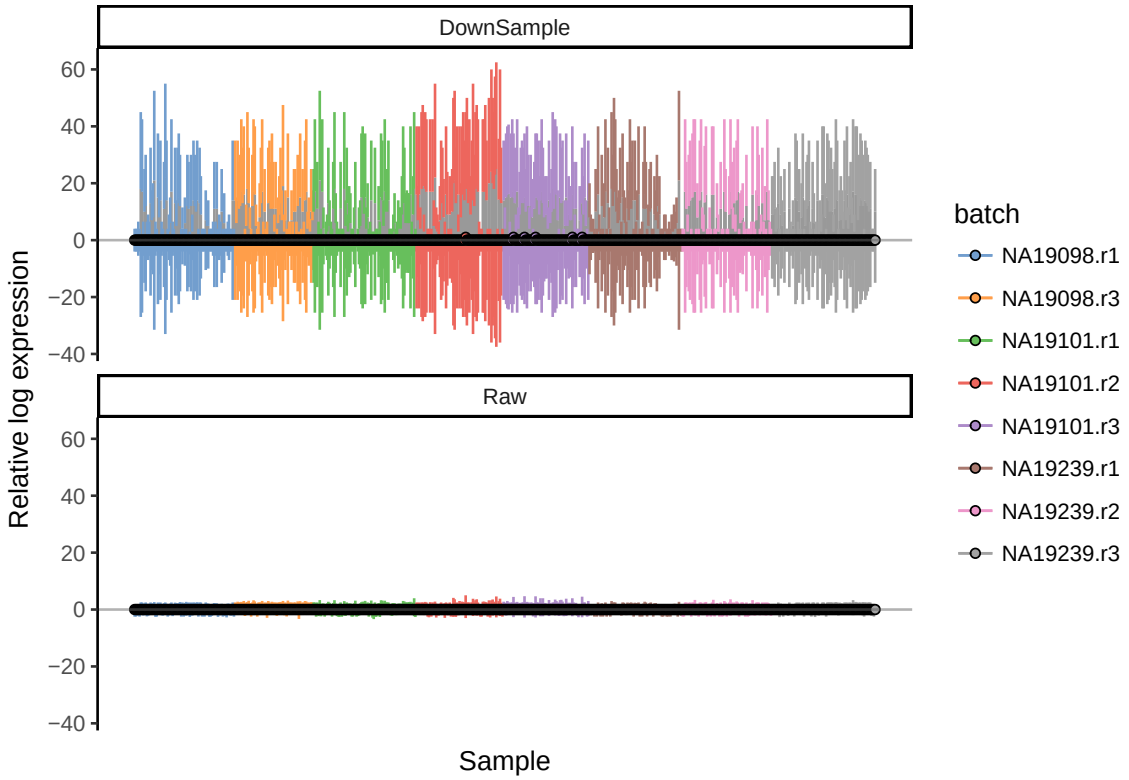


Figure 15.13: Cell-wise RLE of the tung data

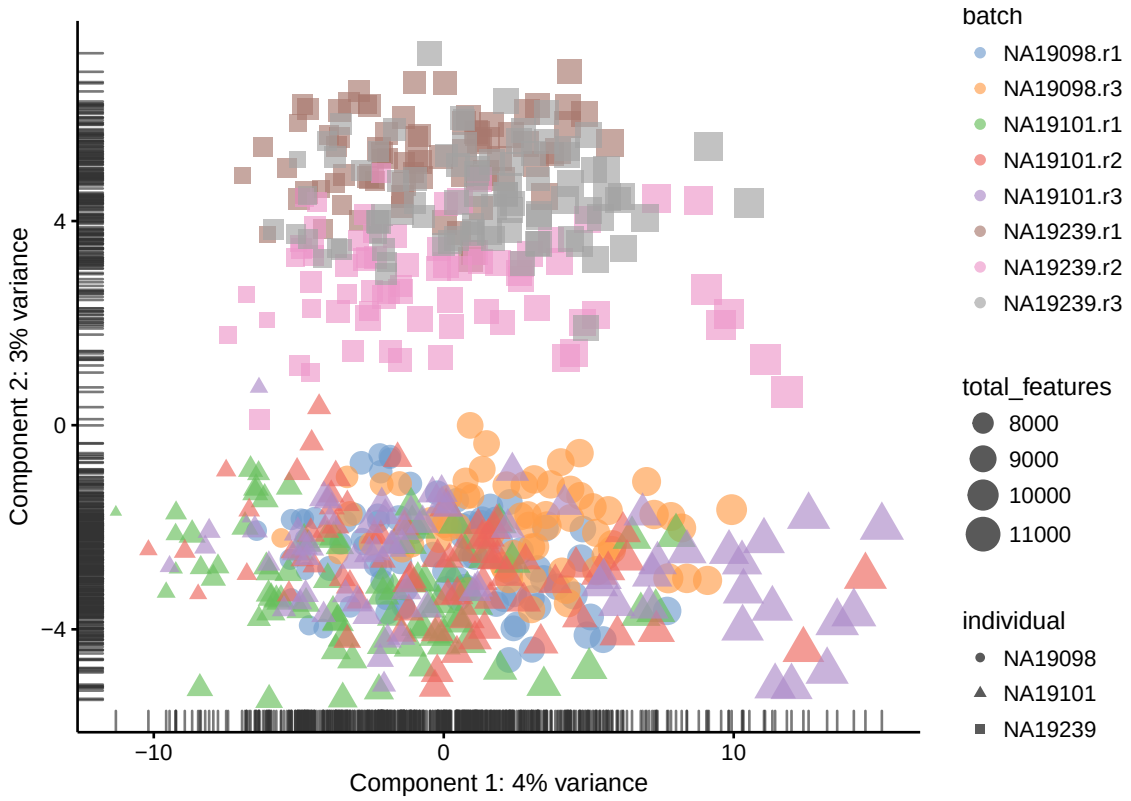


Figure 15.14: PCA plot of the tung data after FPKM normalisation

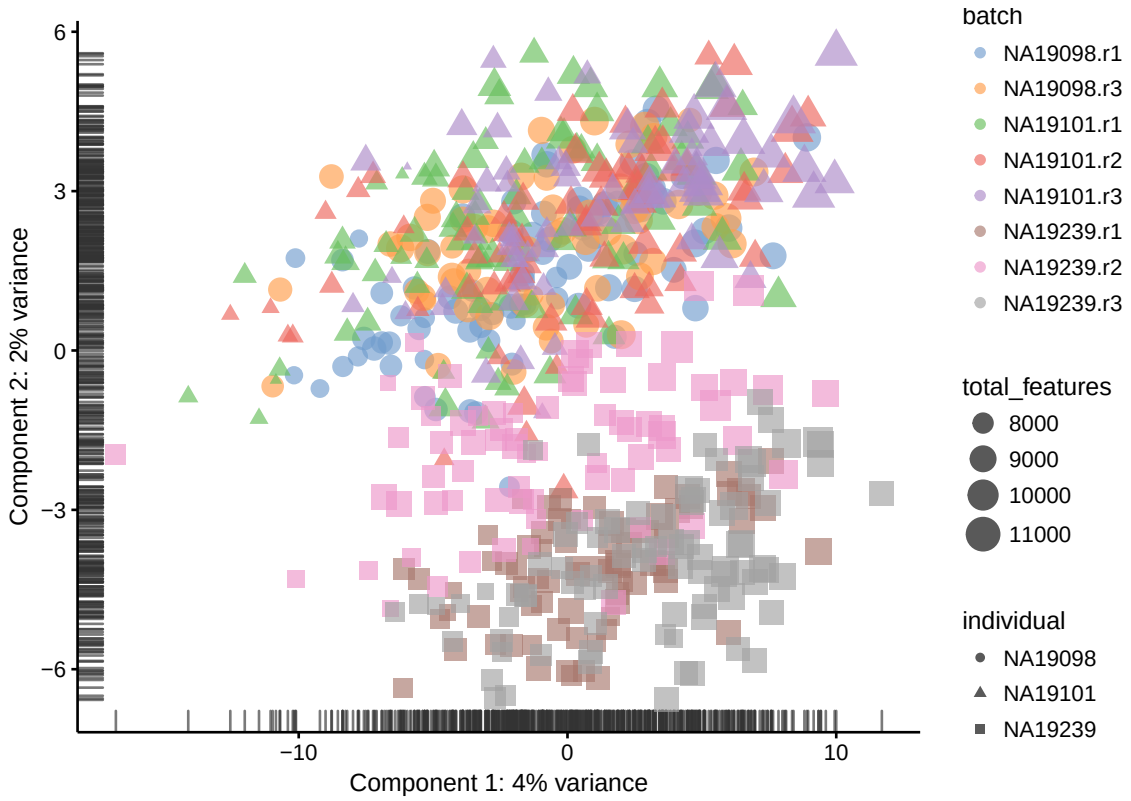


Figure 15.15: PCA plot of the tung data after TPM normalisation

Chapter 16

Dealing with confounders

16.1 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.

16.1.1 Advantages and disadvantages of using spike-ins to remove confounders

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 principle, 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.

16.1.2 How to evaluate and compare confounder removal strategies

A key question when considering the different methods for removing confounders is how to quantitatively determine which one is the most effective. The main reason why comparisons are challenging is because

it is often difficult to know what corresponds to technical counfounders and what is interesting biological variability. Here, we consider three different metrics which are all reasonable based on our knowledge of the experimental design. Depending on the biological question that you wish to address, it is important to choose a metric that allows you to remove the confounders that are likely to be the biggest concern for the given situation.

```
library(scRNA.seq.funcs)
library(RUVSeq)
library(scater, quietly = TRUE)
library(scran)
library(edgeR)
set.seed(1234567)
options(stringsAsFactors = FALSE)
umi <- readRDS("tung/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
```

16.2 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.

16.2.1 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) * 1e6) + 1)
```

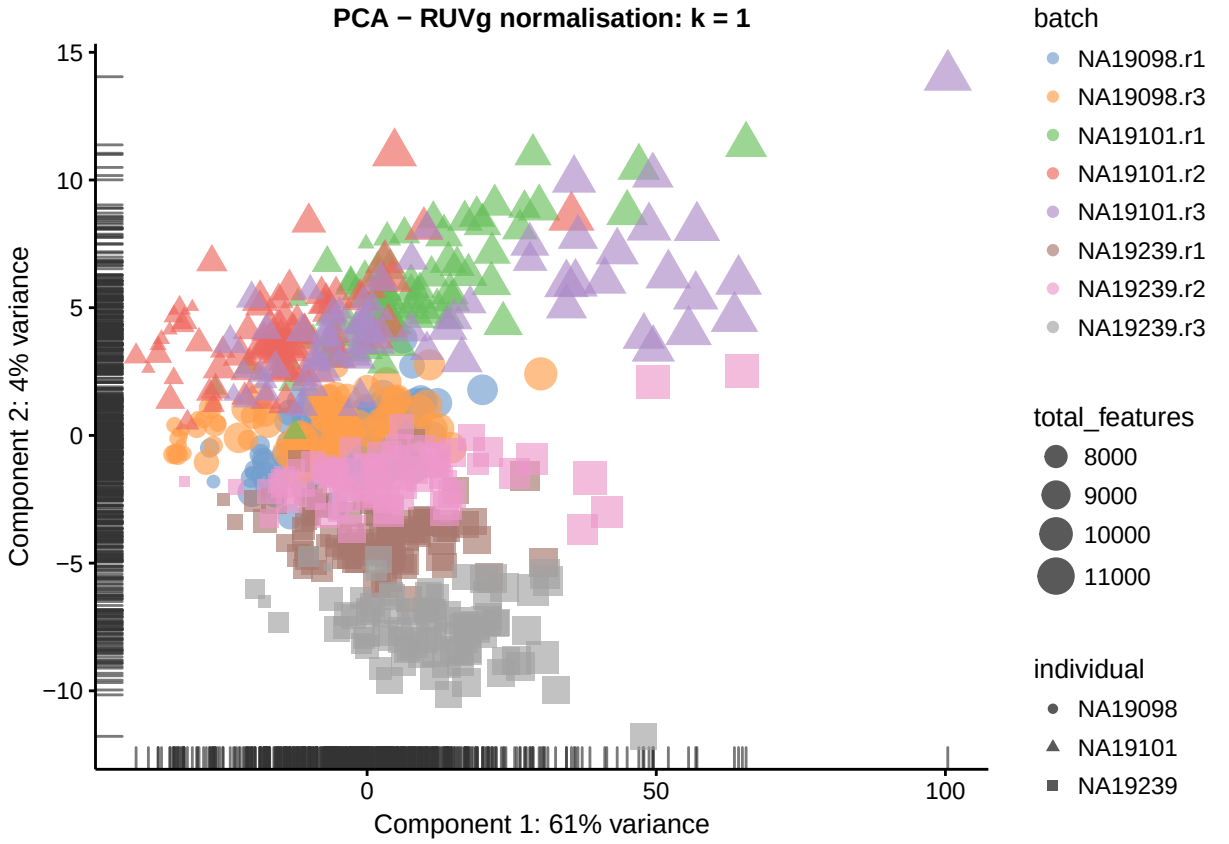
16.2.2 RUVs

```
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) * 1e6) + 1)
```

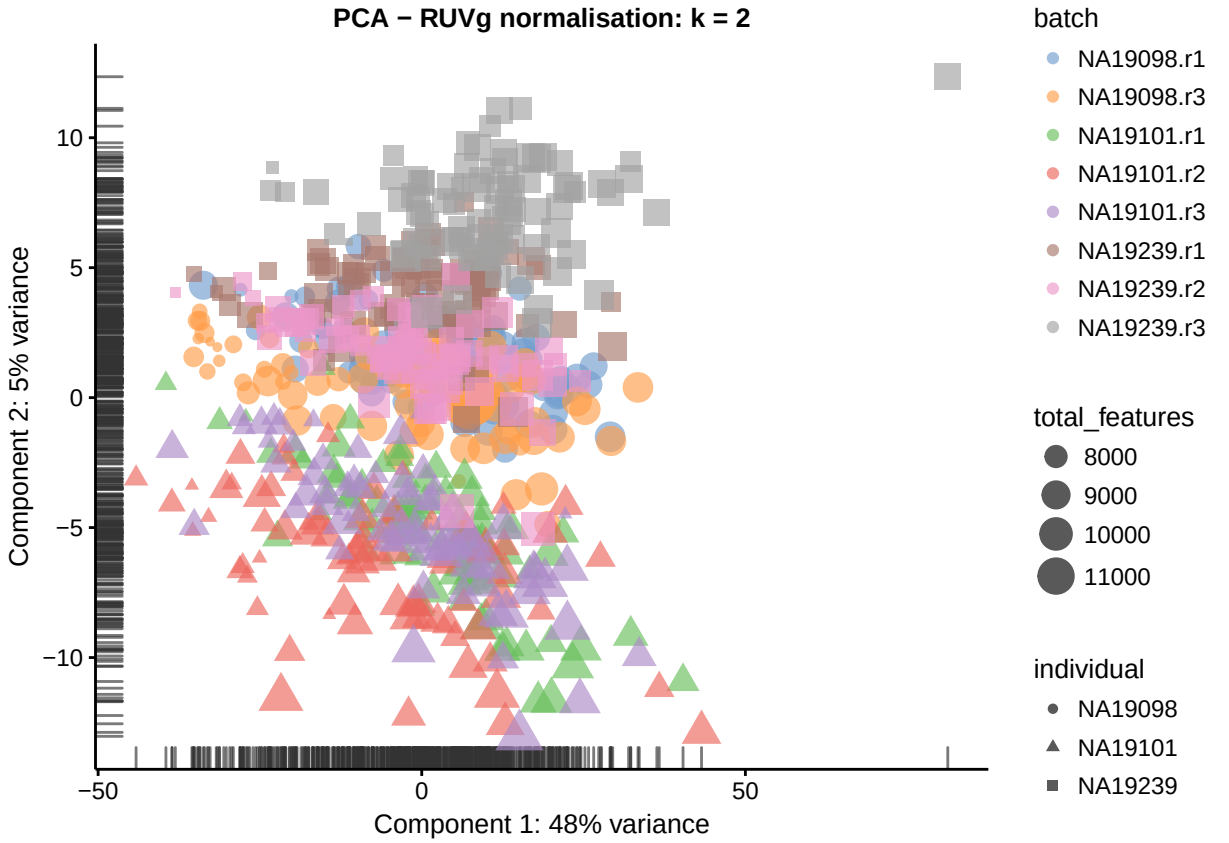
16.3 Effectiveness 1

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.

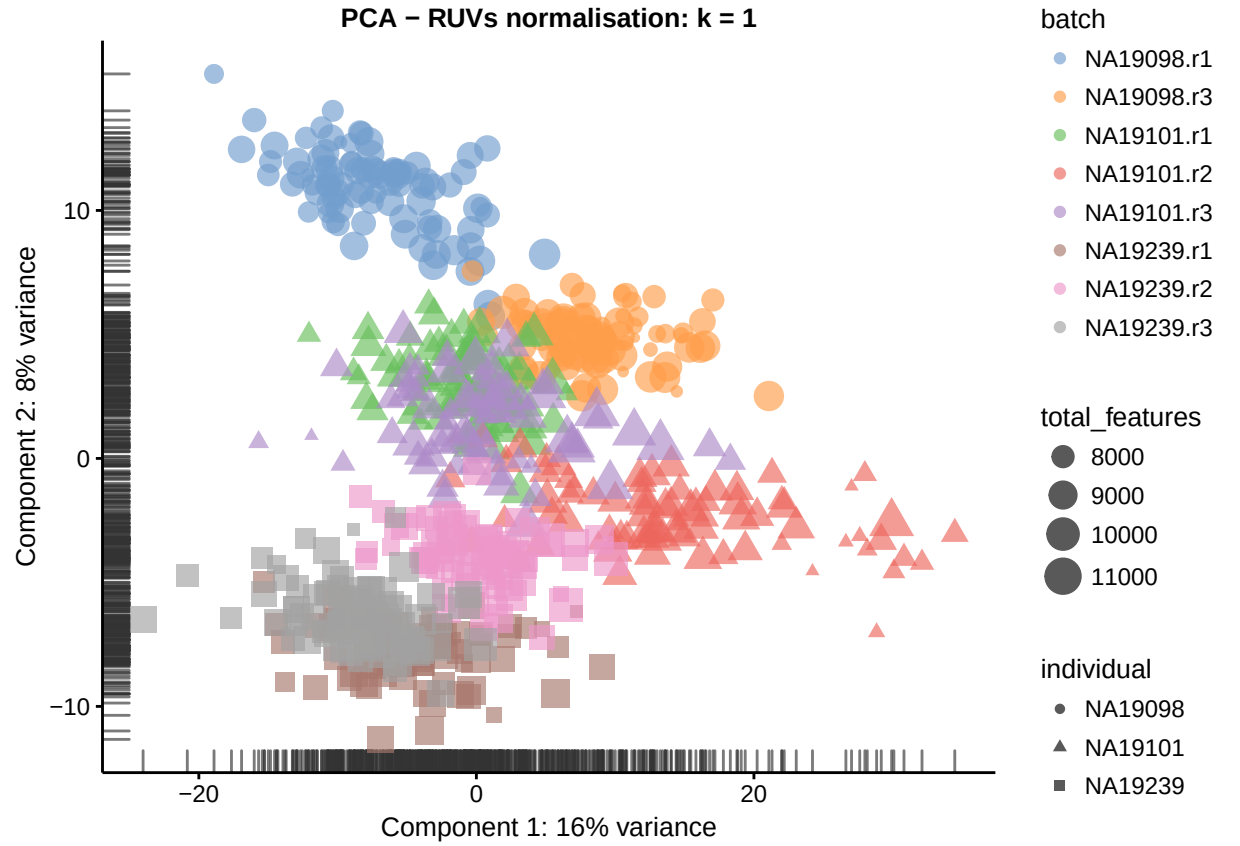
```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg1") +
    ggtitle("PCA - RUVg normalisation: k = 1")
```



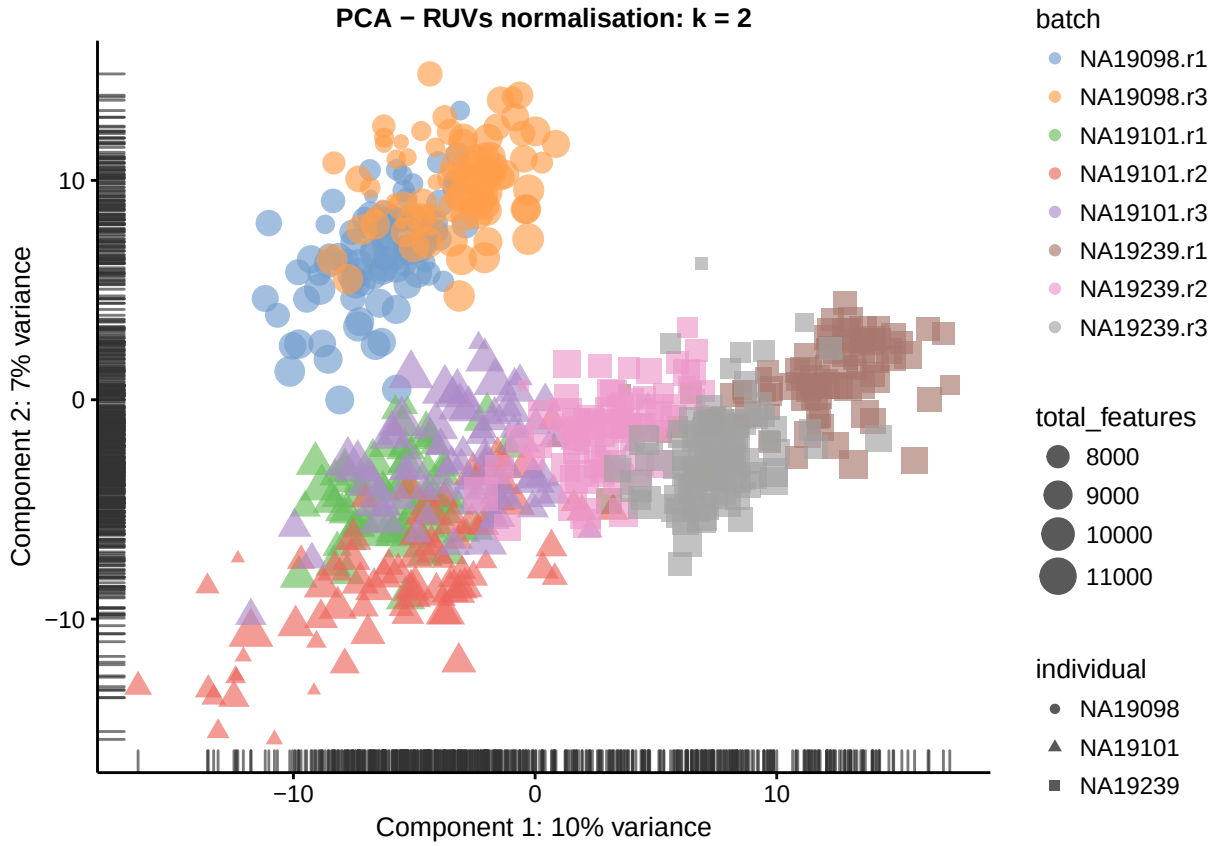

```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg2") +
    ggtitle("PCA - RUVg normalisation: k = 2")
```



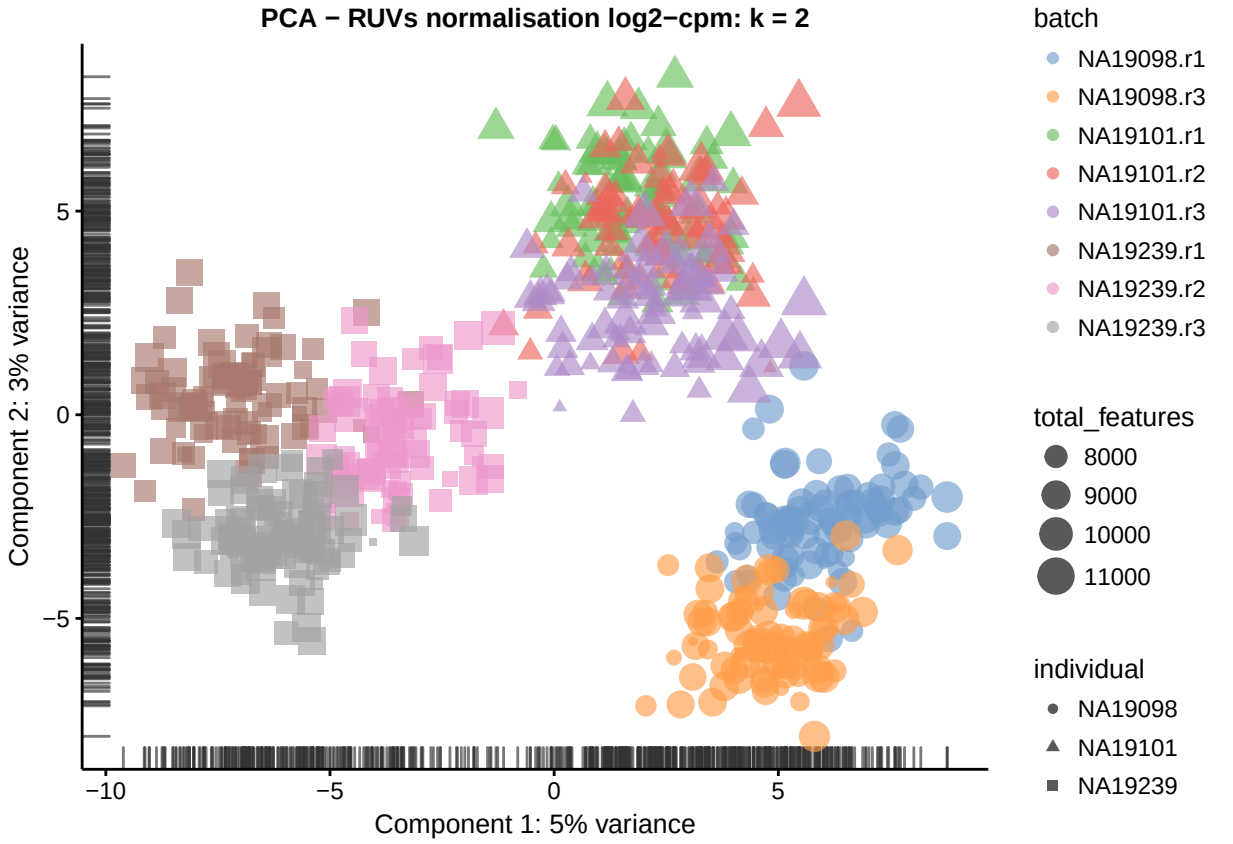

```
plotPCA(
    umi.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs1") +
    ggtitle("PCA - RUVs normalisation: k = 1")
```




```
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")
```

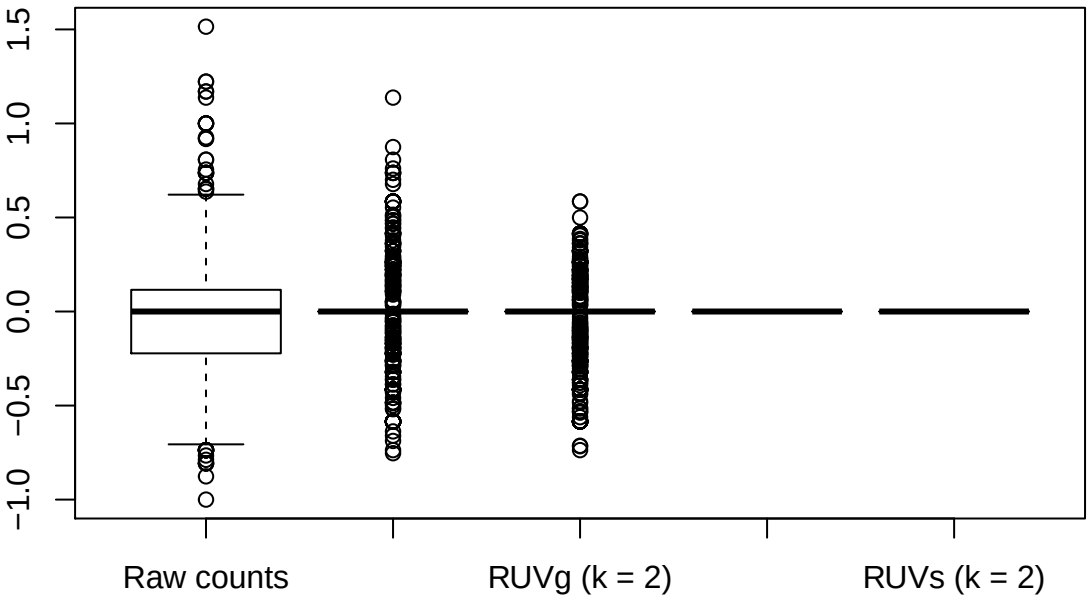


Plotting log2-normalized CPM from RUVs with k = 2 looks to give the best separation of cells by individual.

16.4 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)
    )
)
```



16.5 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.

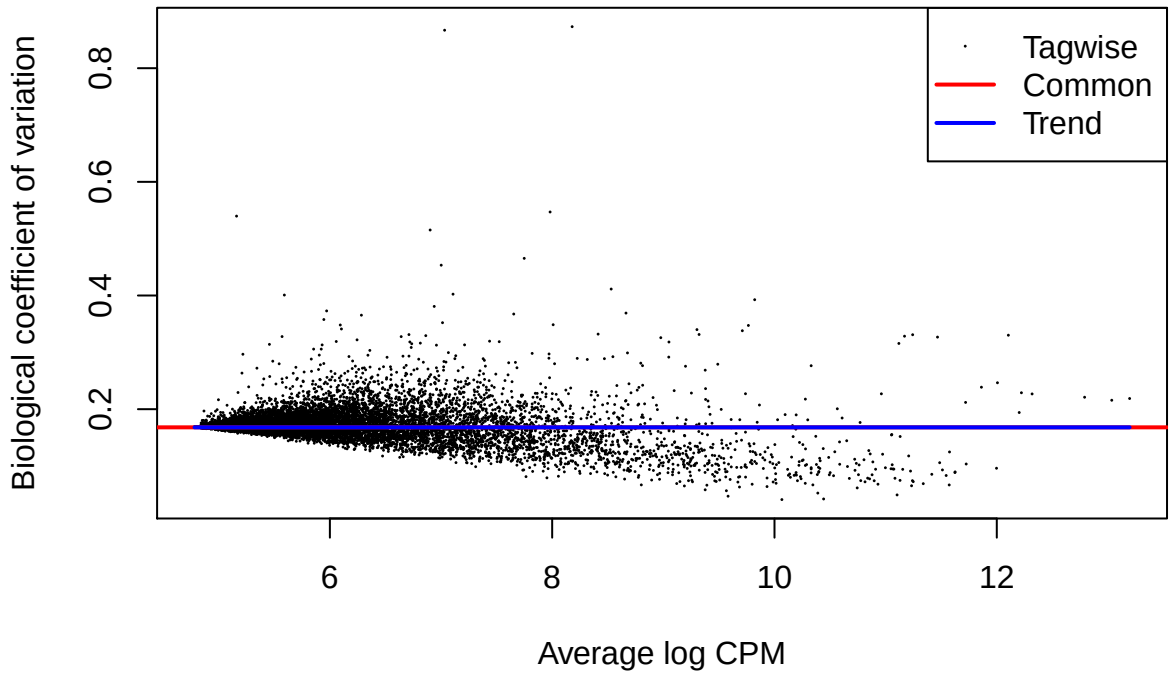
```
keep <- c(
    sample(which(umi.qc$batch == "NA19101.r1"), 20),
    sample(which(umi.qc$batch == "NA19101.r2"), 20),
    sample(which(umi.qc$batch == "NA19101.r3"), 20)
)
design <- model.matrix(~umi.qc[, keep]$batch)
```

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.

16.5.1 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)
```



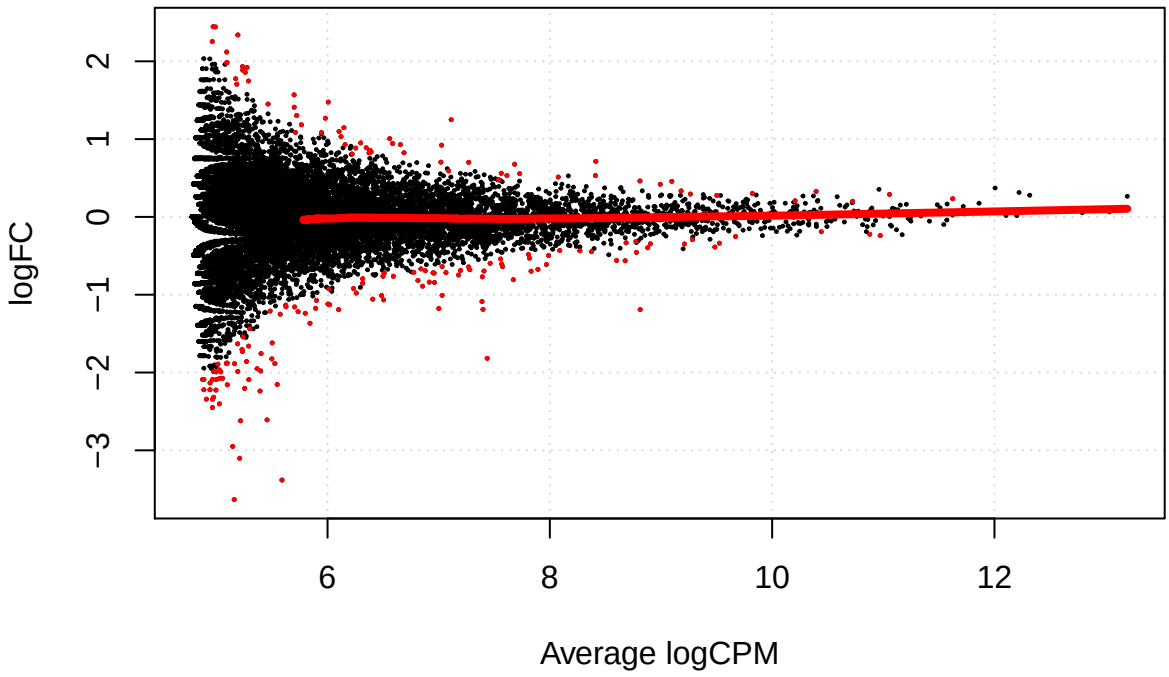

```
fit1 <- glmFit(dge1, design)
res1 <- glmLRT(fit1)
topTags(res1)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                      logFC   logCPM       LR       PValue          FDR
## ENSG00000187193 -1.8179825 7.436706 75.64199 3.400520e-18 4.782151e-14
## ENSG00000185885 -1.1909683 8.812688 62.22578 3.062525e-15 2.153414e-11
## ENSG00000125144 -3.3823146 5.590605 58.16741 2.407340e-14 1.128481e-10
## ENSG00000150459  0.7135016 8.414351 40.20293 2.289043e-10 6.467710e-07
## ENSG00000182463 -3.6331080 5.160311 40.19399 2.299548e-10 6.467710e-07
## ENSG00000008311 -1.1891979 7.398917 37.66873 8.383786e-10 1.965020e-06
## ENSG00000164265 -2.1530458 5.547251 37.26738 1.029932e-09 2.069134e-06
## ENSG00000186439 -3.1023383 5.209191 36.68009 1.391937e-09 2.371266e-06
## ENSG00000134369  1.2492359 7.112593 36.51166 1.517556e-09 2.371266e-06
## ENSG00000198417 -2.6086556 5.456231 34.46438 4.341236e-09 6.105081e-06
```

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

```
##    umi.qc[, keep]$batchNA19101.r3
## -1                            125
## 0                           13875
## 1                              63
```

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

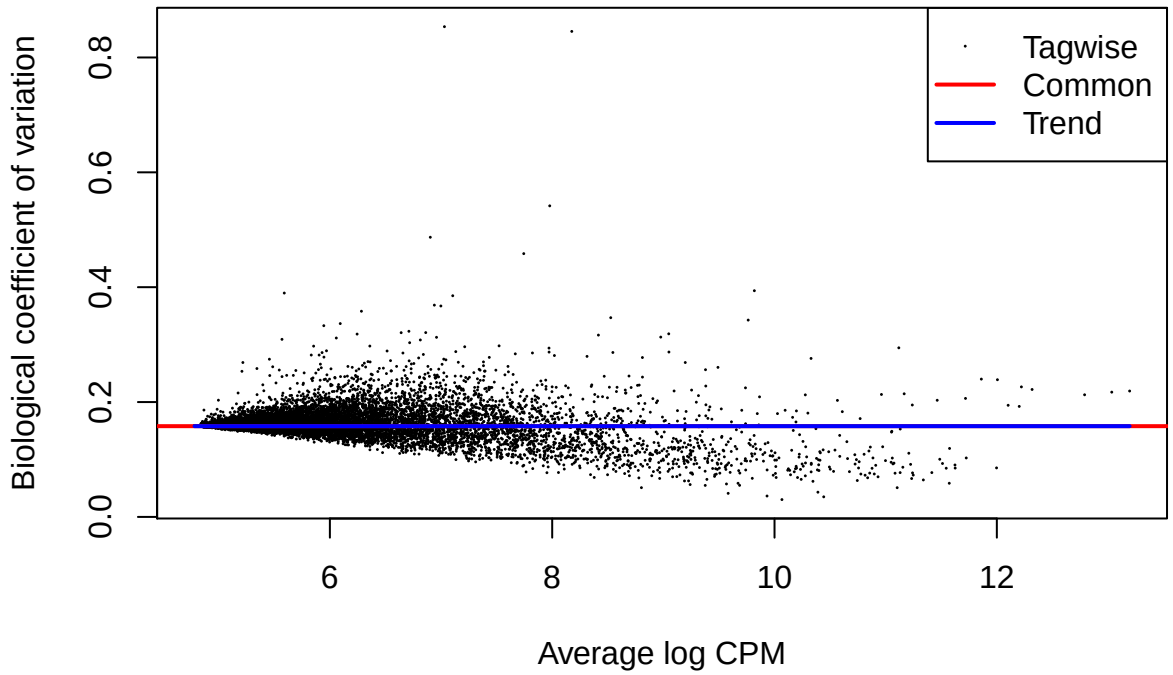


16.5.2 DE (RUVg, k = 2)

```
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       0.031566414       0.028446254
## 2           1       0.008323015       0.031336158
## 3           1       0.010699708      -0.011631828
## 4           1       0.019683744       0.006157921
## 5           1       0.033731033       0.020439301
## 6           1       0.031992504       0.057216310
##   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)
```



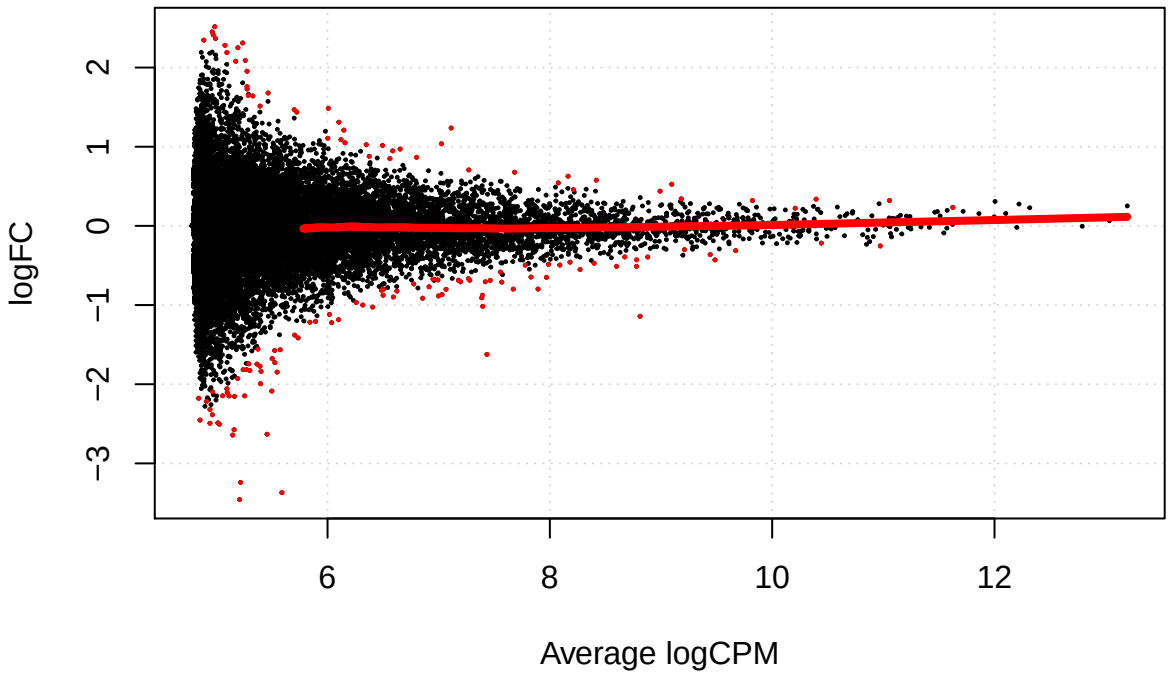

```
fit2 <- glmFit(dge_ruvg, design_ruvg)
res2 <- glmLRT(fit2)
topTags(res2)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                       logFC   logCPM       LR       PValue          FDR
## ENSG00000187193 -1.6237017 7.434075 66.71822 3.132259e-16 4.404895e-12
## ENSG00000185885 -1.1420992 8.811712 50.87408 9.848549e-13 6.925008e-09
## ENSG00000125144 -3.3692260 5.589868 49.87533 1.638318e-12 7.679889e-09
## ENSG00000186439 -3.4569861 5.208899 35.54571 2.491356e-09 8.758984e-06
## ENSG00000134369  1.2360744 7.112423 32.43121 1.234871e-08 3.473198e-05
## ENSG00000196683 -0.4292702 9.485778 30.67591 3.049281e-08 7.147005e-05
## ENSG00000198417 -2.6320892 5.455931 29.86782 4.625256e-08 9.292139e-05
## ENSG00000196591 -0.5111855 8.779698 29.39948 5.889344e-08 1.035273e-04
## ENSG00000143570 -0.7983990 7.670748 28.97874 7.317717e-08 1.143434e-04
## ENSG00000150459  0.5770140 8.417628 28.77421 8.132704e-08 1.143702e-04
```

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

```
##    umi.qc[, keep]$batchNA19101.r3
## -1                             94
## 0                           13920
## 1                              49
```

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

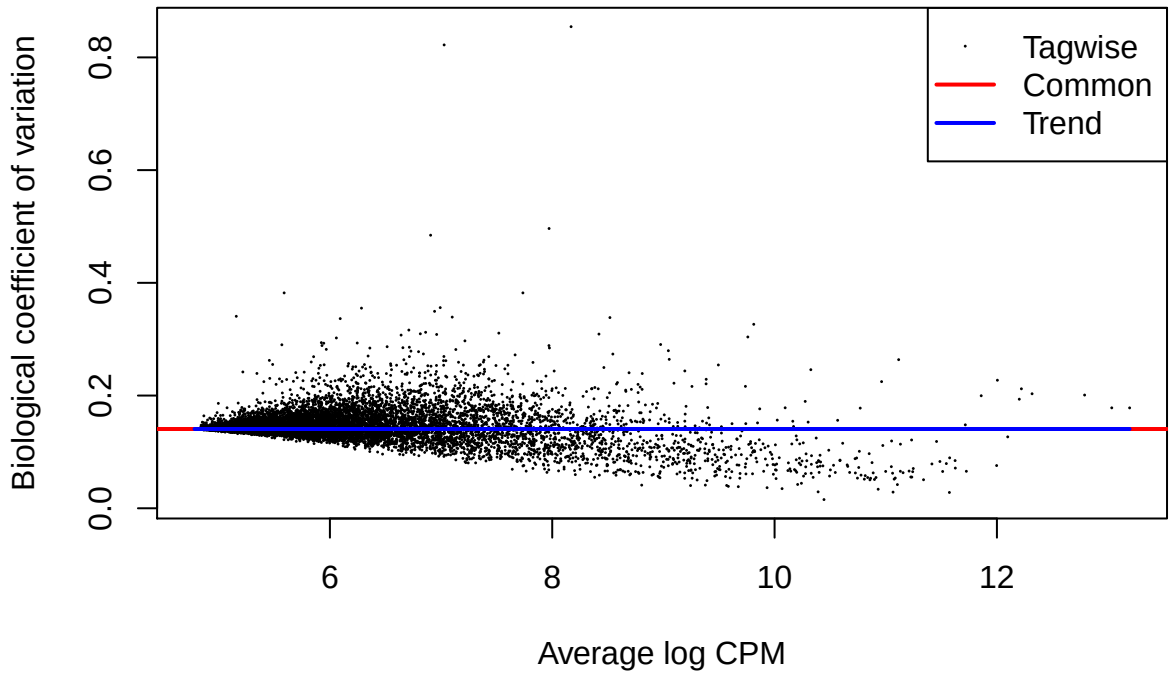


16.5.3 DE (RUVs, k = 2)

```
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         0.2786262       -0.07496082
## 2           1         0.2825106       -0.09287973
## 3           1         0.2688981       -0.08130043
## 4           1         0.2152265       -0.09724690
## 5           1         0.2727856       -0.08117065
## 6           1         0.2351730       -0.07046587
##   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)
```



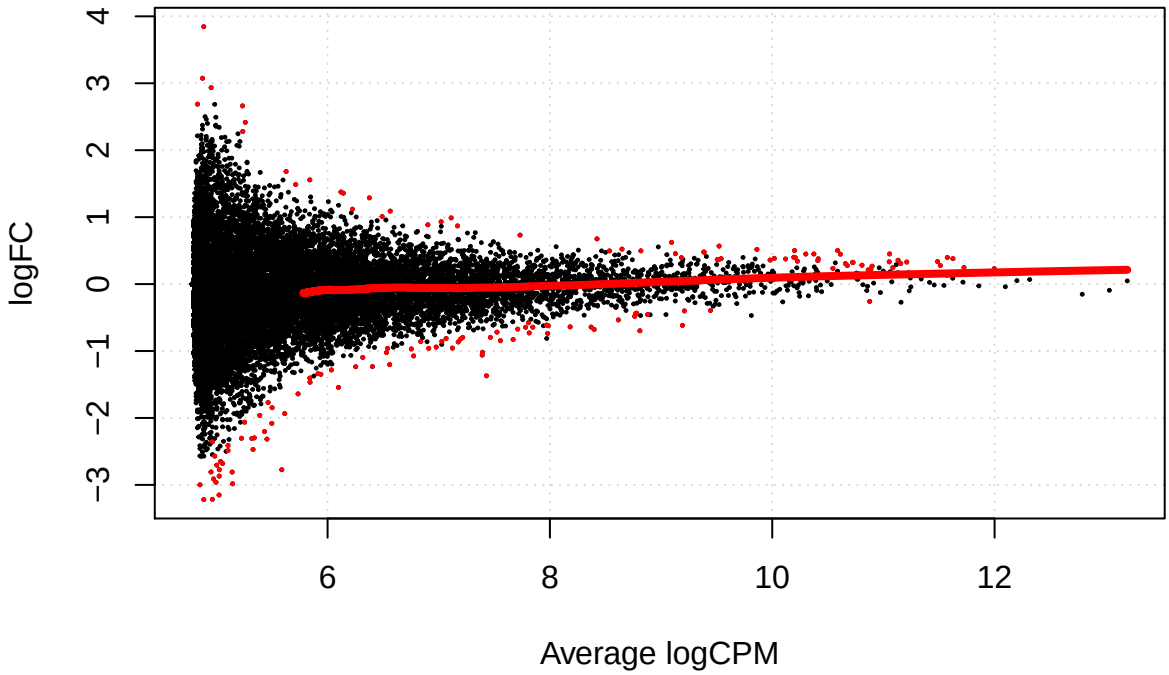

```
fit3 <- glmFit(dge_ruvs, design_ruvs)
res3 <- glmLRT(fit3)
topTags(res3)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                      logFC    logCPM       LR       PValue          FDR
## ENSG00000137818  0.3806669 11.625728 45.53881 1.496440e-11 1.554820e-07
## ENSG00000144713  0.5006206 10.588483 44.77416 2.211221e-11 1.554820e-07
## ENSG00000105372  0.3973437 11.574721 37.81839 7.764625e-10 3.639797e-06
## ENSG00000197958  0.3942581 10.207469 33.58036 6.837956e-09 2.404054e-05
## ENSG00000137154  0.3291951 11.209969 31.28073 2.232847e-08 5.913156e-05
## ENSG00000187193 -1.3710013  7.429212 31.04369 2.522857e-08 5.913156e-05
## ENSG00000181163  0.3355133 11.486981 28.79788 8.033888e-08 1.614008e-04
## ENSG00000089157  0.3079715 11.155625 27.89563 1.280385e-07 2.076654e-04
## ENSG00000150459  0.6754420  8.423888 27.82351 1.329011e-07 2.076654e-04
## ENSG00000114391  0.5009652 10.200433 27.52536 1.550481e-07 2.180441e-04
```

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

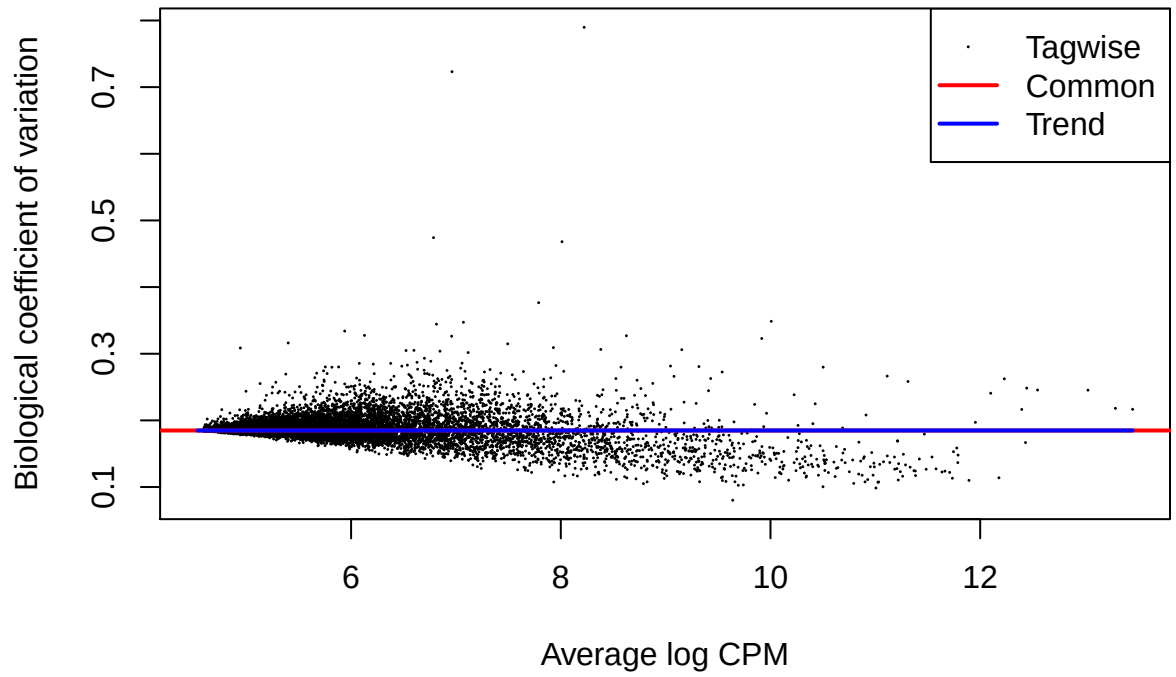
```
##    umi.qc[, keep]$batchNA19101.r3
## -1                             83
## 0                           13911
## 1                              69
```

```
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)
```



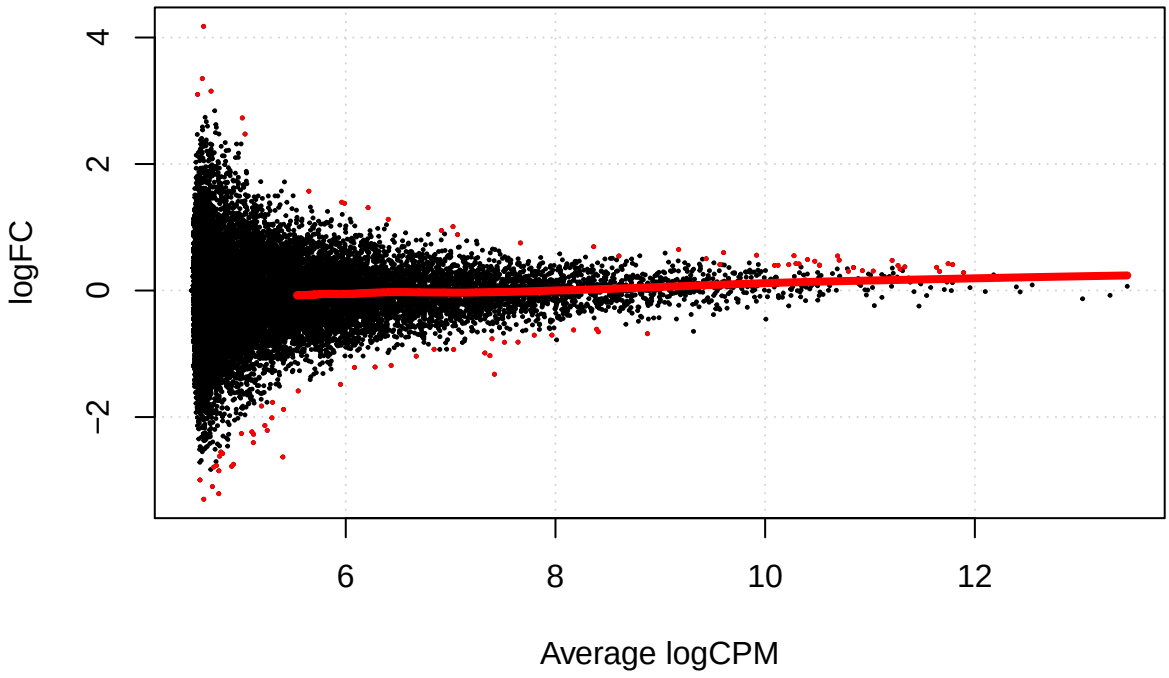

```
fit4 <- glmFit(dge_ruvs_sf, design_ruvs)
res4 <- glmLRT(fit4)
topTags(res4)
```

```
## Coefficient:  umi.qc[, keep]$batchNA19101.r3
##                       logFC    logCPM       LR       PValue          FDR
## ENSG00000187193 -1.3230540  7.418234 31.81968 1.691697e-08 0.0002379033
## ENSG00000144713  0.5465584 10.690938 27.50251 1.568906e-07 0.0011031766
## ENSG00000153246  4.1765109  4.644252 23.73794 1.103848e-06 0.0039359534
## ENSG00000150459  0.6934232  8.365003 23.71081 1.119520e-06 0.0039359534
## ENSG00000125144 -2.6309945  5.400342 22.94655 1.665689e-06 0.0046849159
## ENSG00000162244  0.5584180  9.918262 22.01551 2.704563e-06 0.0063390446
## ENSG00000198918  0.5993234  9.603605 21.15631 4.233047e-06 0.0084570542
## ENSG00000174748  0.4802203 10.706546 20.52616 5.882171e-06 0.0084570542
## ENSG00000181163  0.3668285 11.635978 20.39748 6.291268e-06 0.0084570542
## ENSG00000114391  0.5512702 10.275265 20.07975 7.427918e-06 0.0084570542
```

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

```
##    umi.qc[, keep]$batchNA19101.r3
## -1                             43
## 0                           13973
## 1                              47
```

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



16.6 Exercise

Perform the same analysis with read counts of the `tung` data. Use `tung/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 17

Dealing with confounders (Reads)

```
library(scRNA.seq.funcs)
library(RUVSeq)
library(scater, quietly = TRUE)
library(scran)
library(edgeR)
set.seed(1234567)
options(stringsAsFactors = FALSE)
reads <- readRDS("tung/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
```

17.1 Remove Unwanted Variation

17.1.1 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)
```

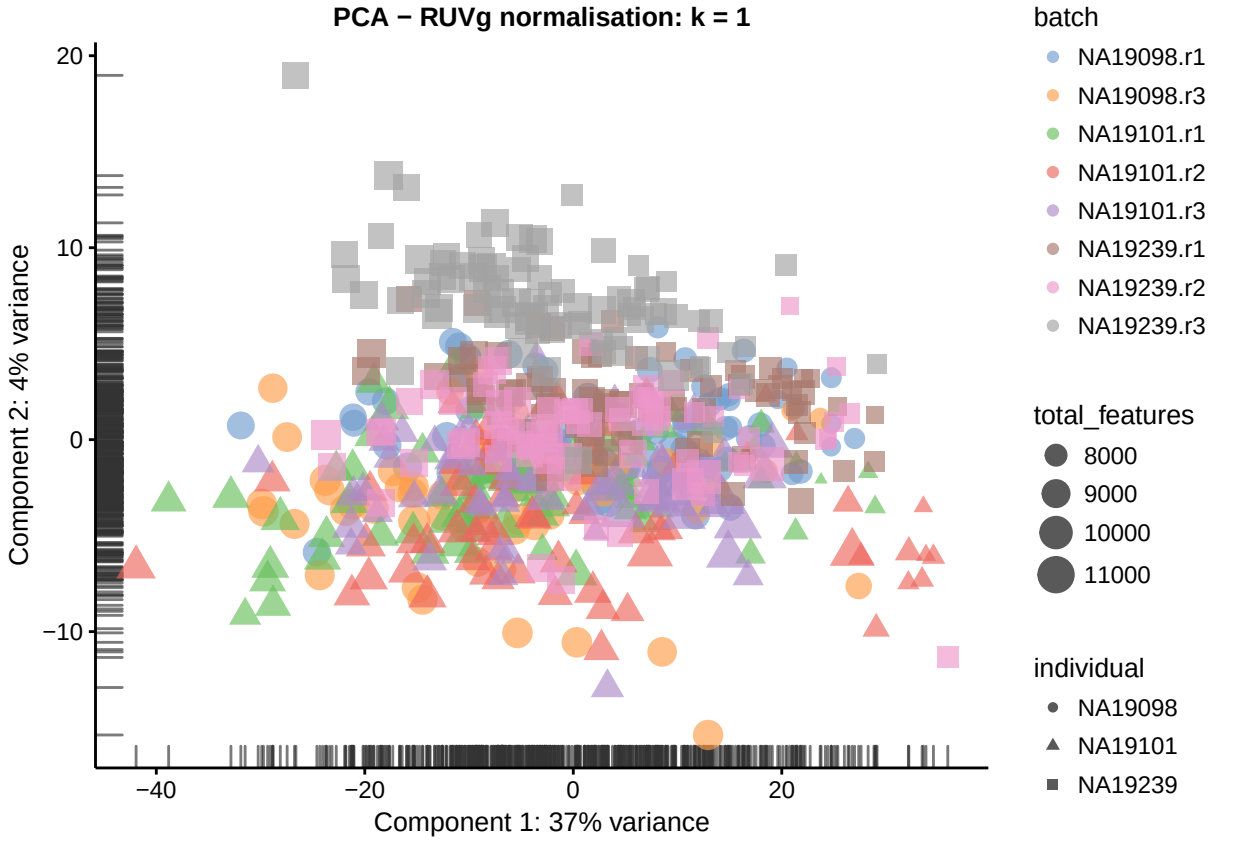
17.1.2 RUVs

```
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)
```

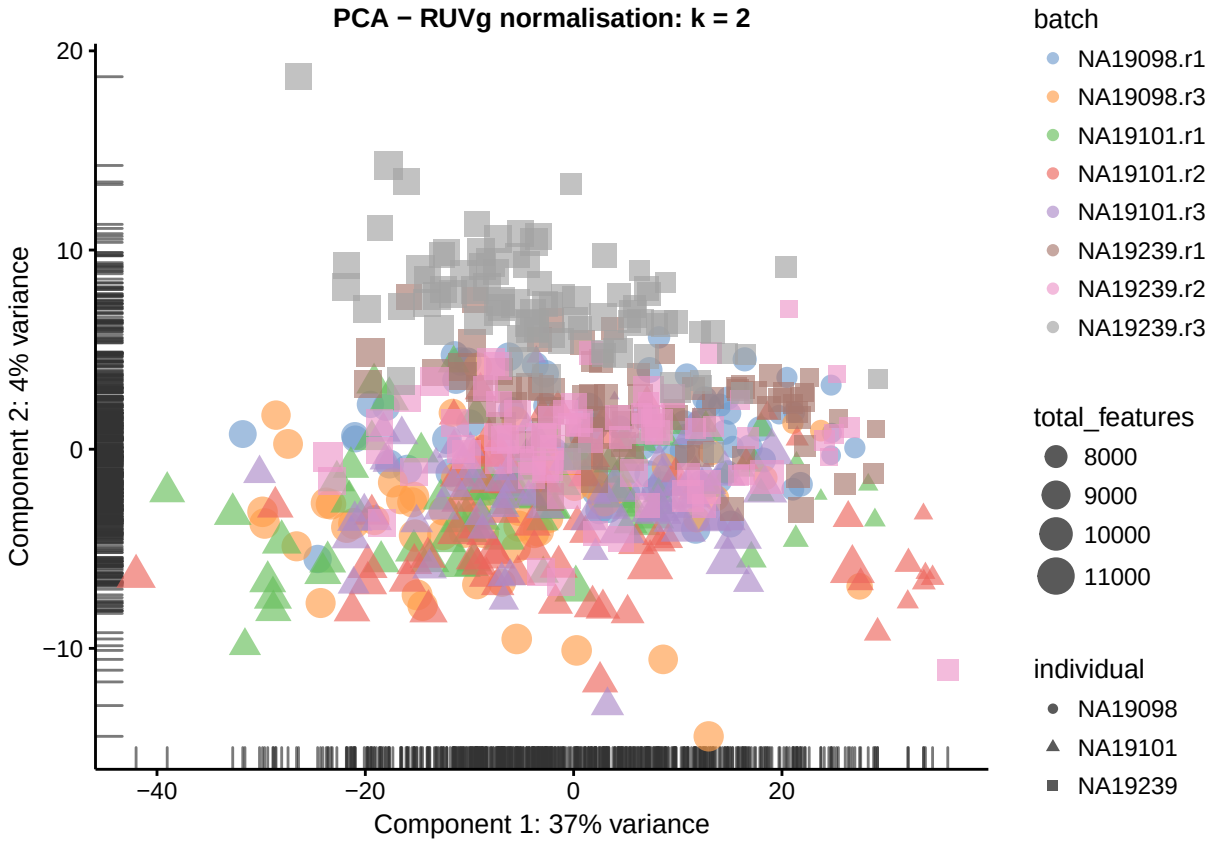
```
ruvs <- RUVs(counts(reads.qc), cIdx, k = 1, scIdx = scIdx, isLog = FALSE)
set_exprs(reads.qc, "ruvs1") <- ruvs$normalizedCounts
ruvs <- RUVs(counts(reads.qc), cIdx, k = 2, scIdx = scIdx, isLog = FALSE)
set_exprs(reads.qc, "ruvs2") <- ruvs$normalizedCounts
set_exprs(reads.qc, "ruvs2_logcpm") <- log2(t(t(ruvs$normalizedCounts)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

17.2 Effectiveness 1

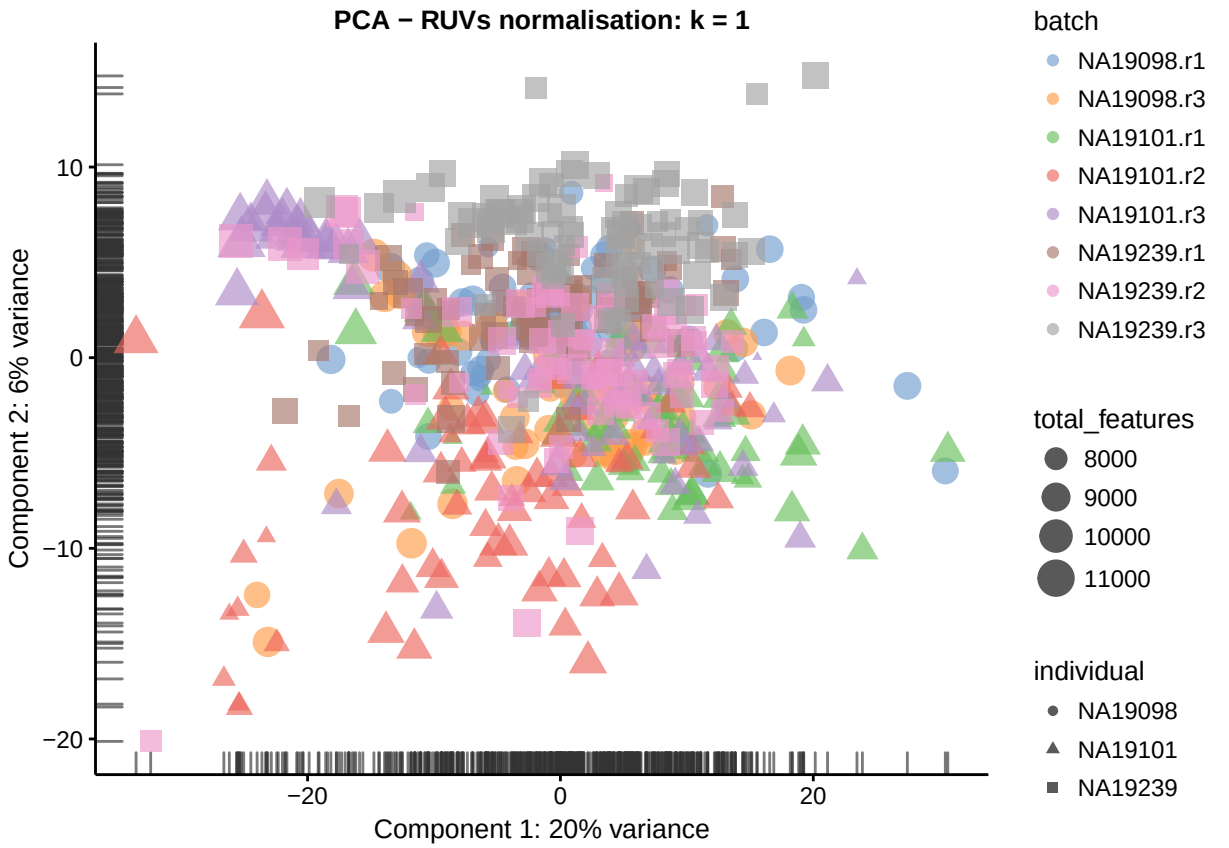
```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg1") +
    ggtitle("PCA - RUVg normalisation: k = 1")
```



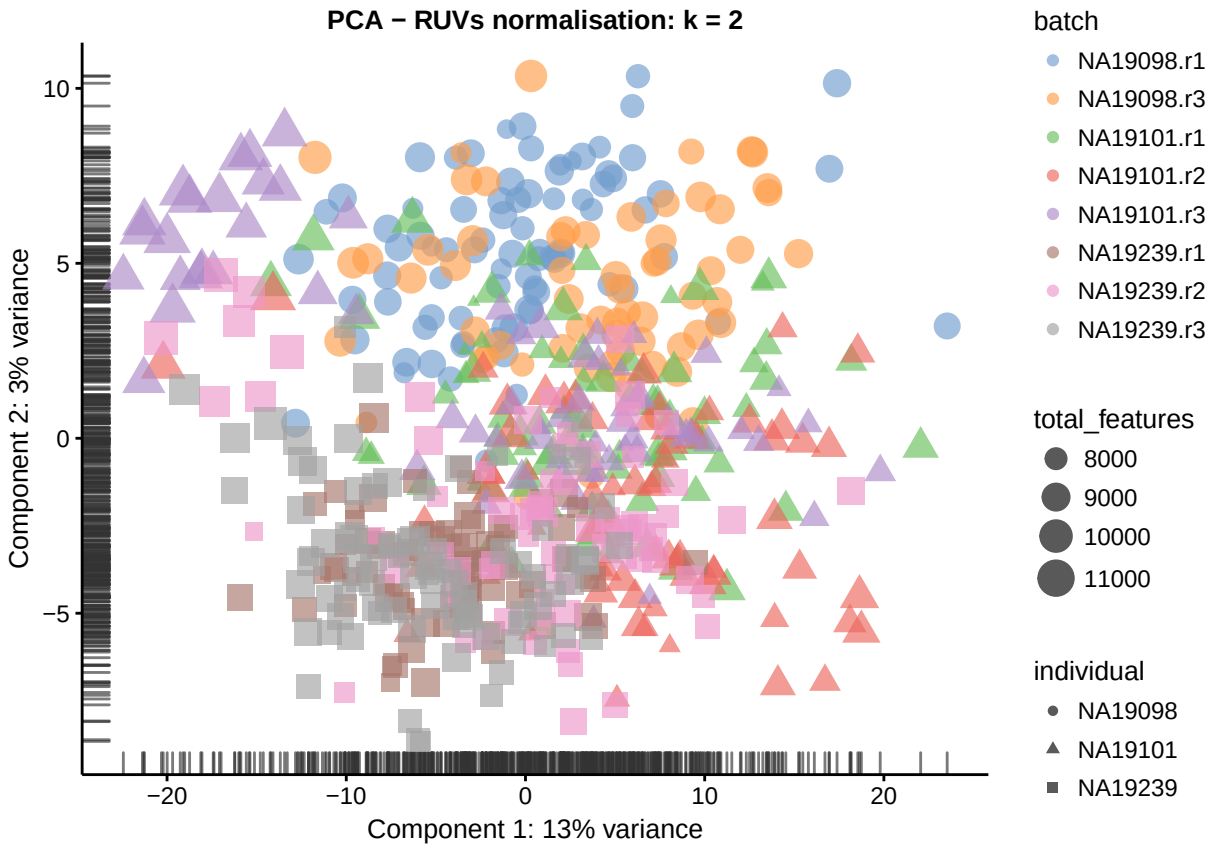

```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvg2") +
    ggtitle("PCA - RUVg normalisation: k = 2")
```



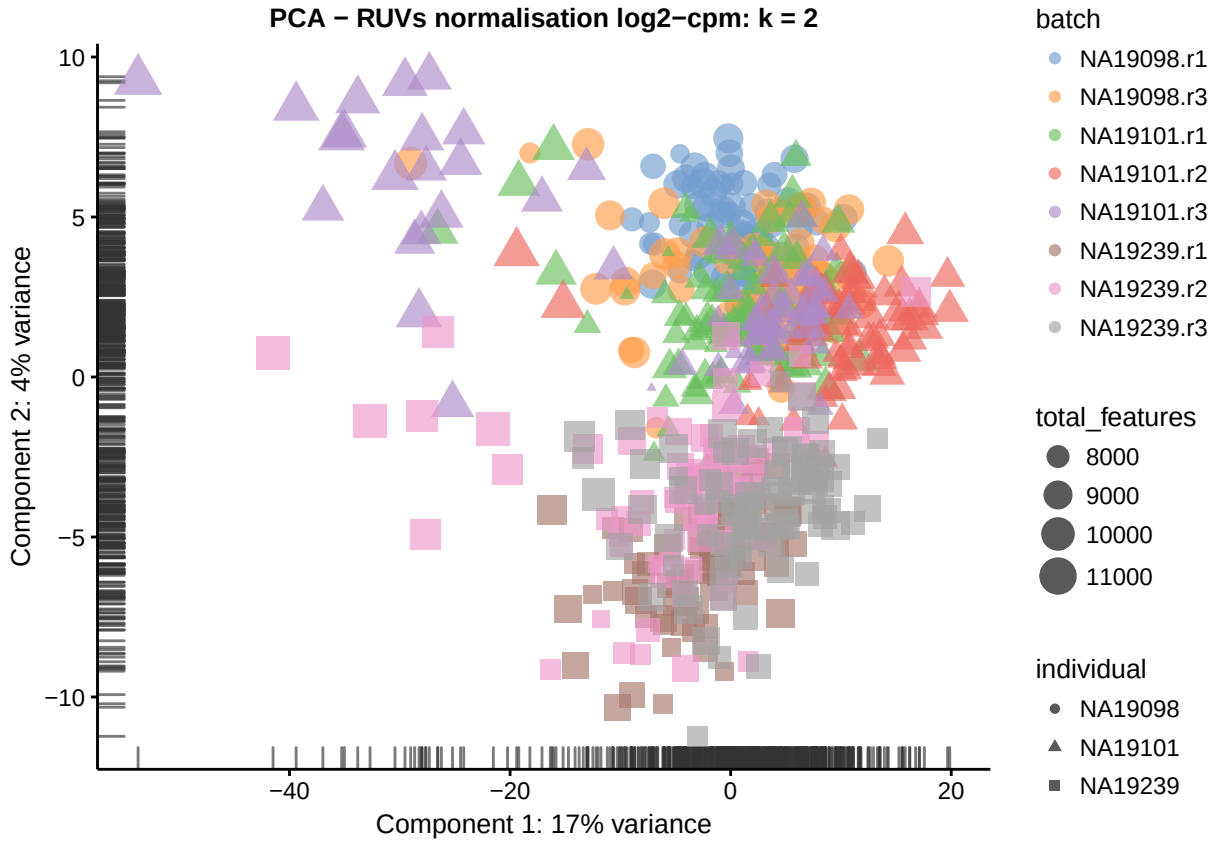

```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs1") +
    ggtitle("PCA - RUVs normalisation: k = 1")
```




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

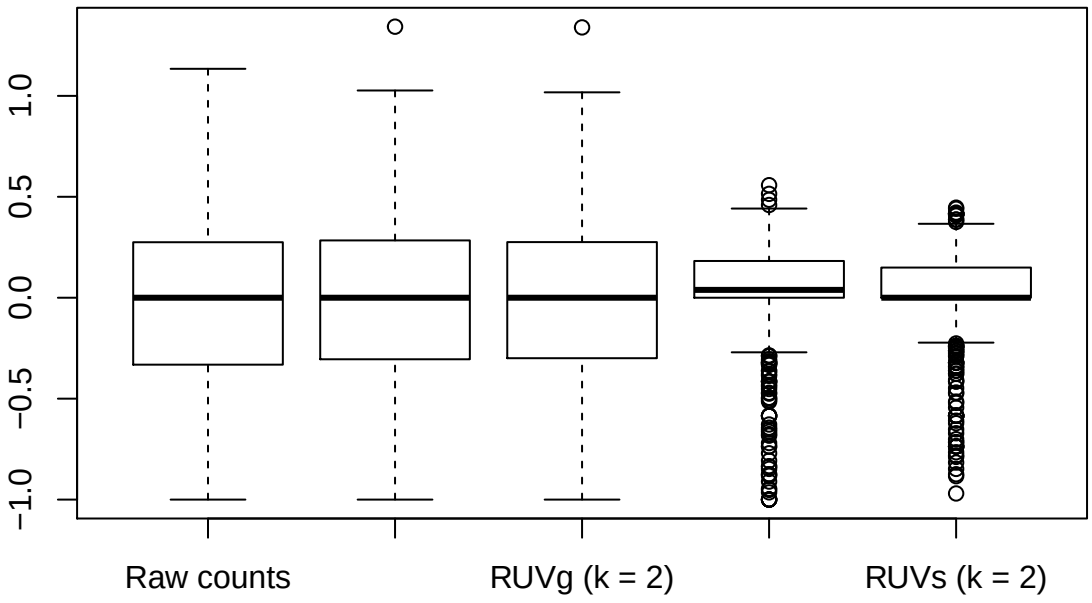



```
plotPCA(
    reads.qc[endog_genes, ],
    colour_by = "batch",
    size_by = "total_features",
    shape_by = "individual",
    exprs_values = "ruvs2_logcpm") +
    ggtitle("PCA - RUVs normalisation log2-cpm: k = 2")
```



17.3 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)
    )
)
```

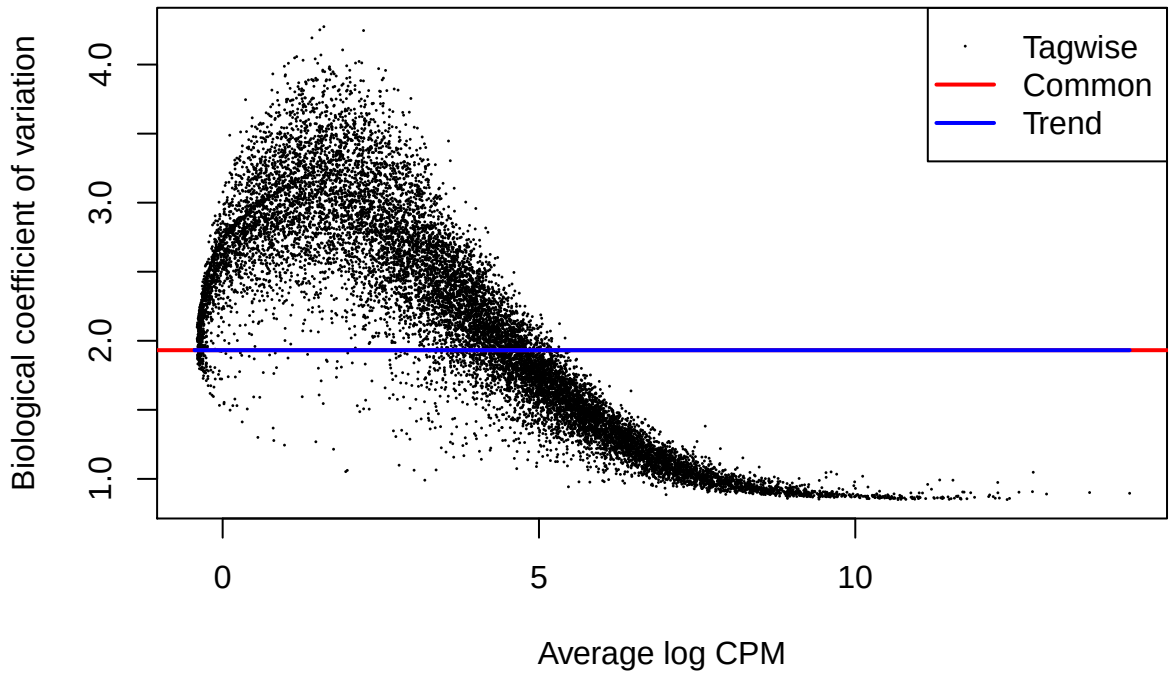


17.4 Effectiveness 3

```
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)
```

17.4.1 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)
```



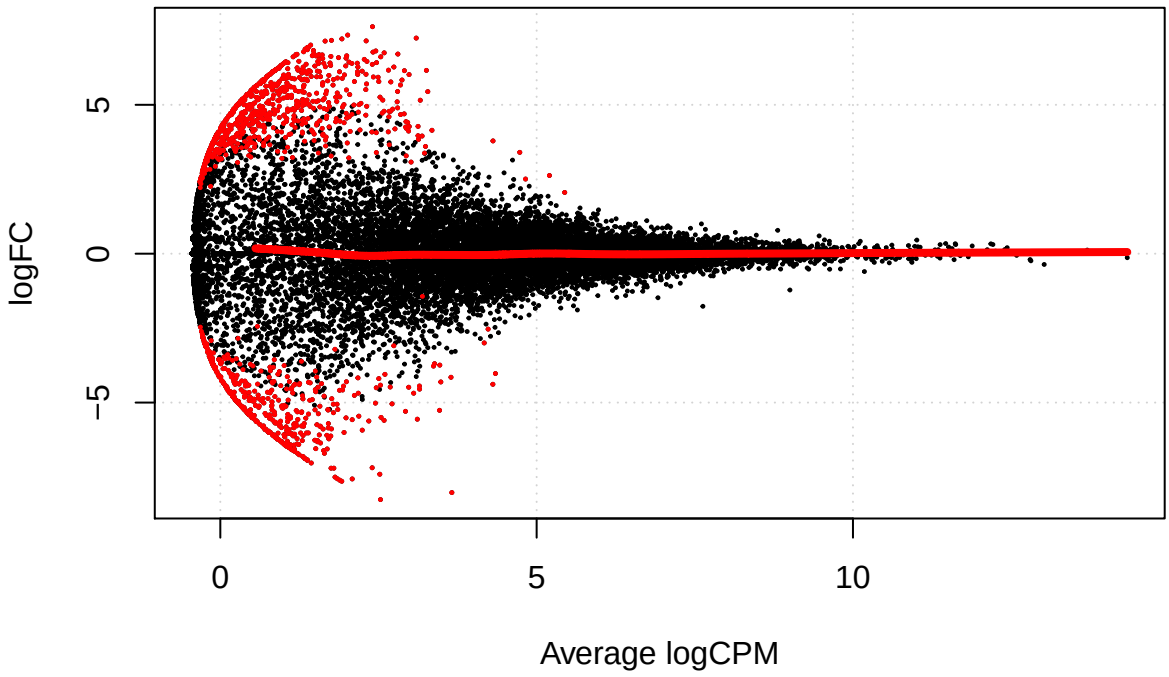

```
fit1 <- glmFit(dge1, design)
res1 <- glmLRT(fit1)
topTags(res1)
```

```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                     logFC    logCPM       LR       PValue          FDR
## ENSG00000245680  6.769933 2.4140394 29.57165 5.388728e-08 0.0008043469
## ENSG00000157680  6.929079 1.3773980 28.37086 1.001615e-07 0.0008043469
## ENSG00000169857  7.241350 3.0968670 27.43852 1.621686e-07 0.0008681968
## ENSG00000151690  6.362547 1.3564536 26.51047 2.621139e-07 0.0008699941
## ENSG00000197261  6.156435 3.2574320 26.44720 2.708406e-07 0.0008699941
## ENSG00000021300  7.140051 1.6568860 25.65818 4.075650e-07 0.0009893710
## ENSG00000123612 -6.807787 1.3010583 25.54937 4.312059e-07 0.0009893710
## ENSG00000078579  6.931629 1.3951041 25.10872 5.418732e-07 0.0010878783
## ENSG00000018869  5.873107 0.7109472 24.74574 6.541323e-07 0.0011090305
## ENSG00000182575  6.251666 0.9288213 24.64143 6.905115e-07 0.0011090305
```

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

```
##    reads.qc[, keep]$batchNA19101.r3
## -1                              464
## 0                             14851
## 1                               746
```

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

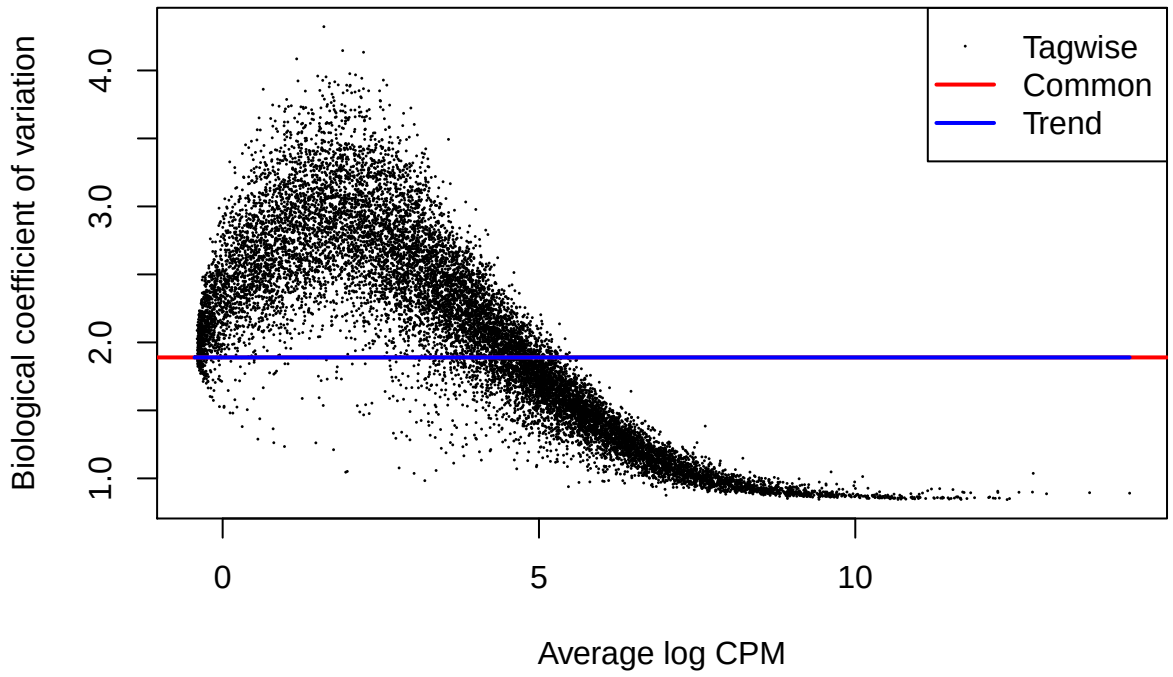


17.4.2 DE (RUVg, k = 2)

```
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       0.018190004        0.04764997
## 2           1       0.023547154        0.03231790
## 3           1      -0.043540927        0.03411069
## 4           1      -0.011047886       -0.05379651
## 5           1      -0.002009755        0.04140351
## 6           1       0.055027809        0.05877063
##   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)
```



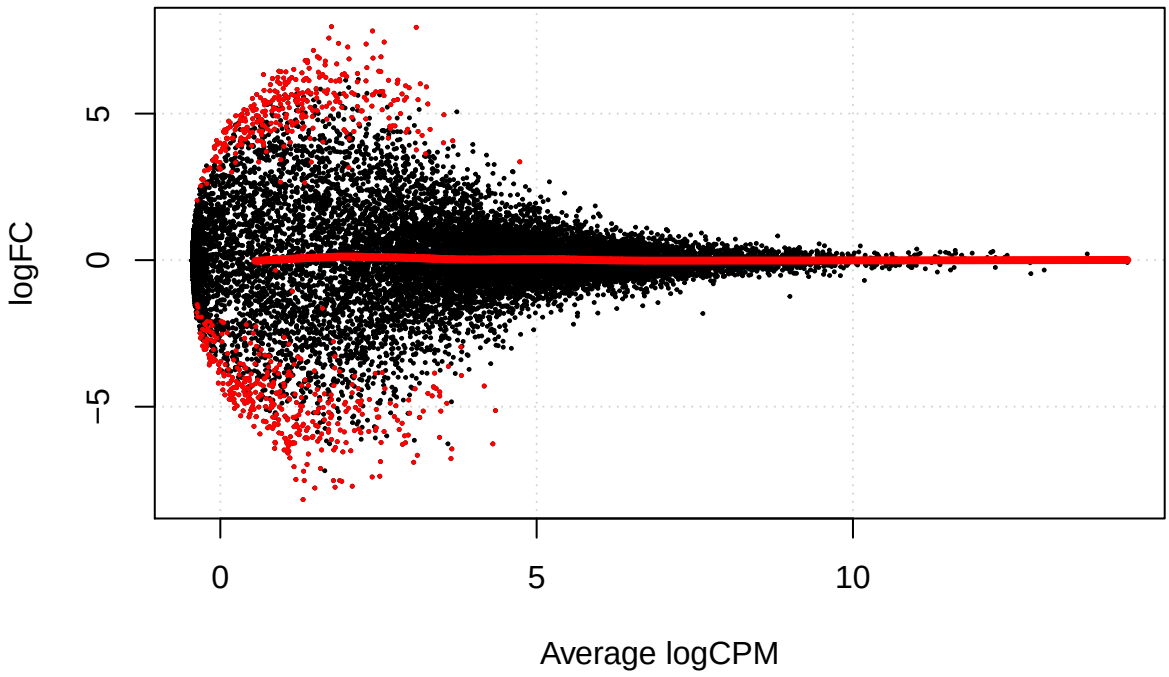

```
fit2 <- glmFit(dge_ruvg, design_ruvg)
res2 <- glmLRT(fit2)
topTags(res2)
```

```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                      logFC    logCPM       LR       PValue          FDR
## ENSG00000088926 -7.520594 1.8872095 31.75667 1.747483e-08 0.0002806632
## ENSG00000169857  7.944401 3.0967722 29.69136 5.066020e-08 0.0002887833
## ENSG00000245680  6.894702 2.4140553 29.56971 5.394121e-08 0.0002887833
## ENSG00000143127  5.641200 1.0238358 27.99202 1.218168e-07 0.0003519555
## ENSG00000153233  6.197221 0.8234713 27.75578 1.376366e-07 0.0003519555
## ENSG00000175318 -6.747507 1.0999870 27.73641 1.390212e-07 0.0003519555
## ENSG00000144962 -7.747163 1.8142340 27.54608 1.533957e-07 0.0003519555
## ENSG00000081923 -7.047319 1.1647043 25.76440 3.857376e-07 0.0005768692
## ENSG00000062282  5.554112 0.8624379 25.73278 3.921092e-07 0.0005768692
## ENSG00000114737 -7.712987 2.0853276 25.54660 4.318272e-07 0.0005768692
```

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

```
##    reads.qc[, keep]$batchNA19101.r3
## -1                              395
## 0                             15341
## 1                               325
```

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

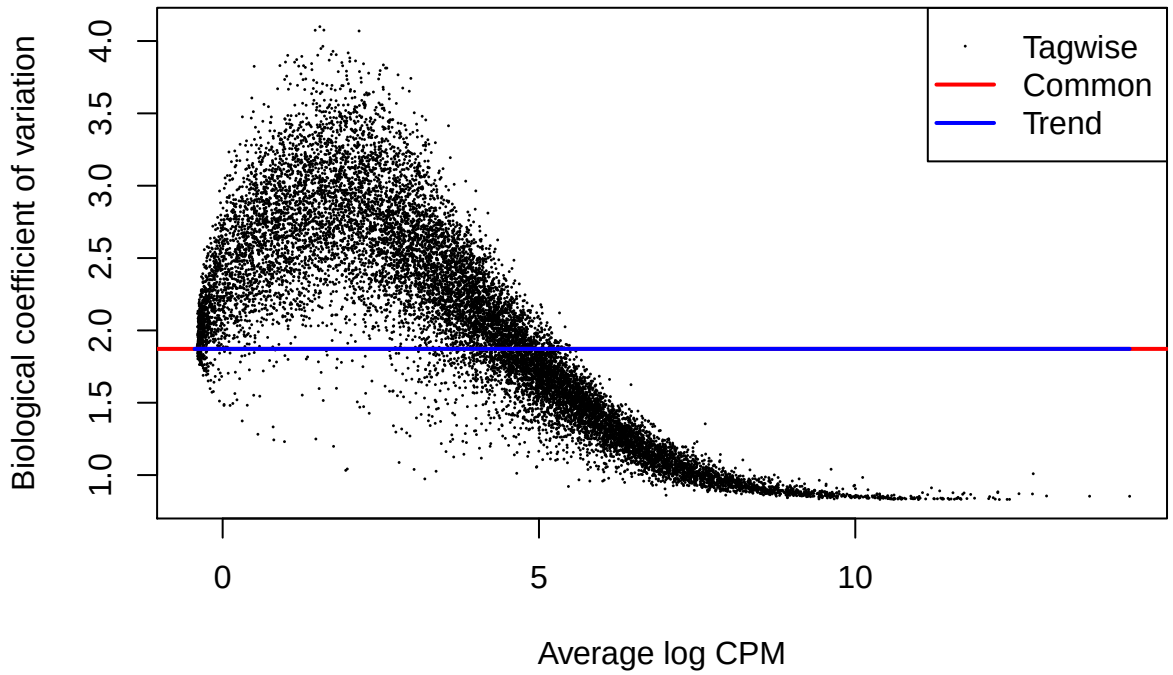


17.4.3 DE (RUVs, k = 2)

```
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         0.3772545         0.1683790
## 2           1         0.3470296         0.2160446
## 3           1         0.3371526         0.1701236
## 4           1         0.2576765         0.1992972
## 5           1         0.3527278         0.2031540
## 6           1         0.3008299         0.1209562
##   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)
```



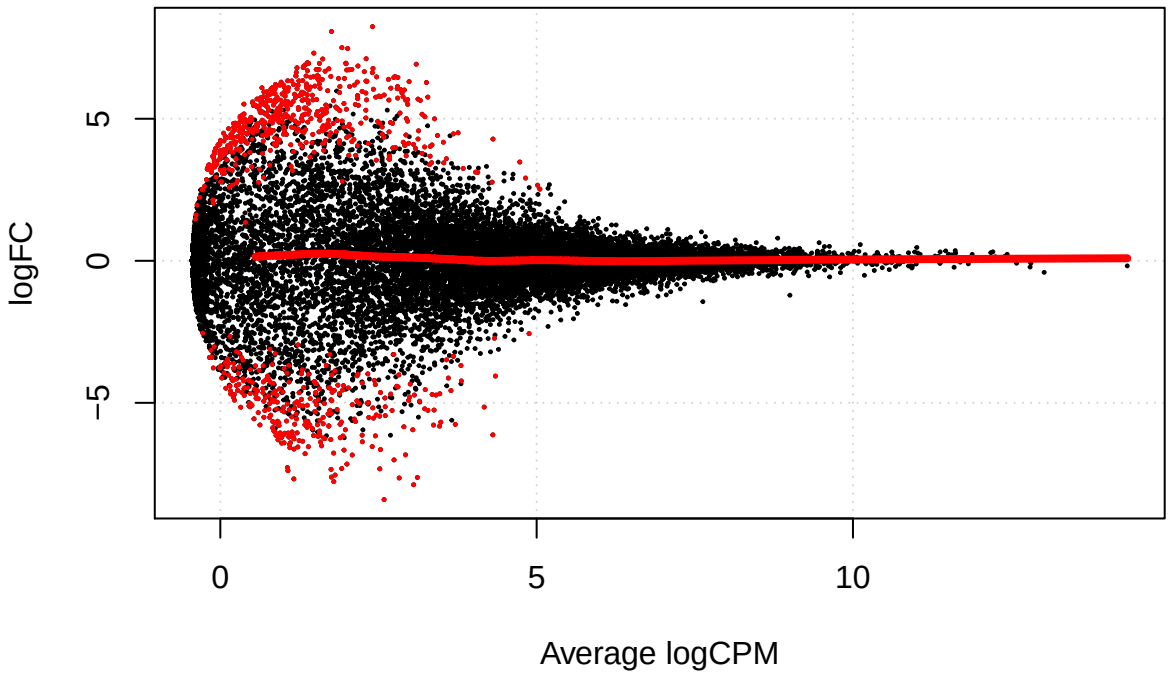

```
fit3 <- glmFit(dge_ruvs, design_ruvs)
res3 <- glmLRT(fit3)
topTags(res3)
```

```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                      logFC   logCPM       LR       PValue          FDR
## ENSG00000175105 -7.881072 3.056358 29.62129 5.252477e-08 0.0003983720
## ENSG00000245680  6.771561 2.414062 29.20563 6.508967e-08 0.0003983720
## ENSG00000138650 -8.404976 2.589017 28.94635 7.441105e-08 0.0003983720
## ENSG00000157680  6.974511 1.377270 28.16394 1.114619e-07 0.0004475476
## ENSG00000127366 -7.311757 1.919586 26.84980 2.198967e-07 0.0005301343
## ENSG00000197261  6.276350 3.257436 26.77877 2.281295e-07 0.0005301343
## ENSG00000151690  6.506839 1.356669 26.75417 2.310529e-07 0.0005301343
## ENSG00000250506  6.738161 2.079191 26.17409 3.119801e-07 0.0005614098
## ENSG00000169228  8.240724 2.405449 25.92722 3.545347e-07 0.0005614098
## ENSG00000143127  5.971608 1.023916 25.47575 4.479788e-07 0.0005614098
```

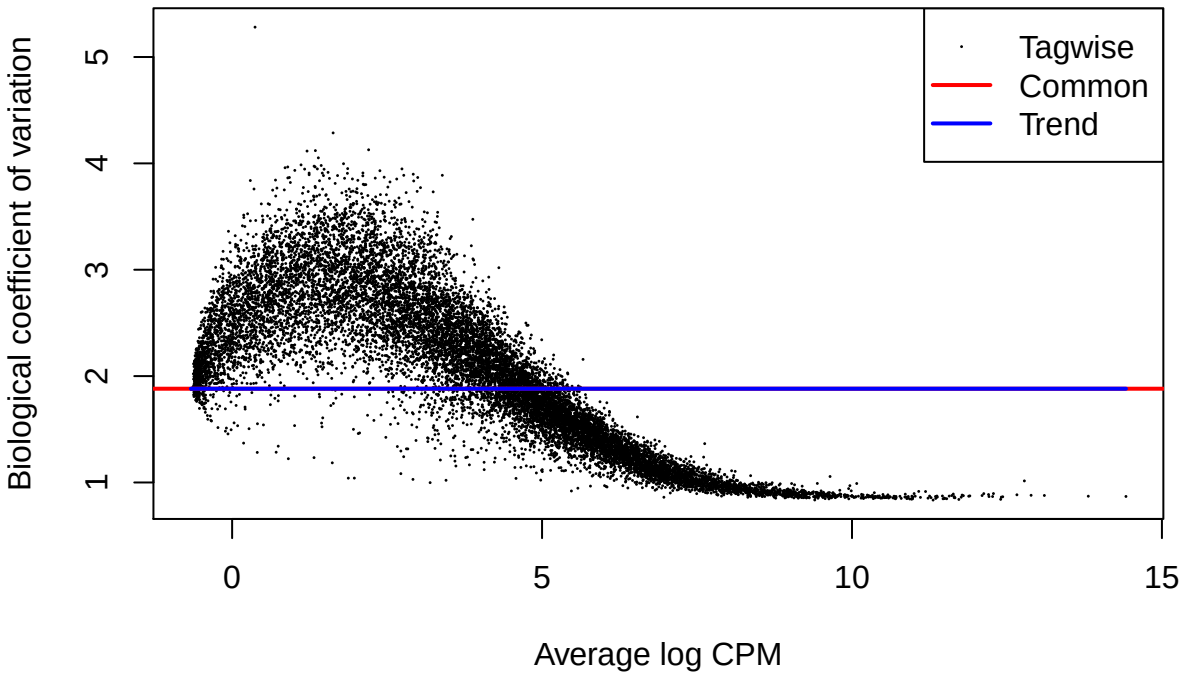
```
summary(decideTestsDGE(res3))
```

```
##    reads.qc[, keep]$batchNA19101.r3
## -1                              303
## 0                             15219
## 1                               539
```

```
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)
```

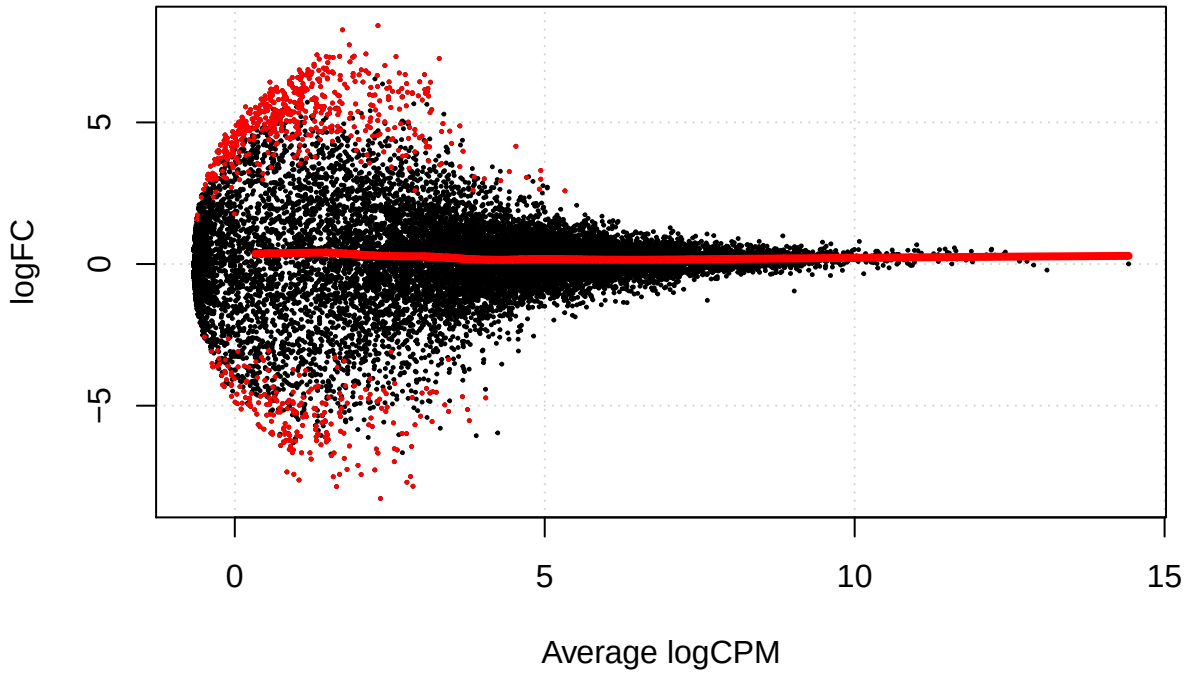
```
## Coefficient:  reads.qc[, keep]$batchNA19101.r3
##                      logFC     logCPM       LR       PValue          FDR
```

```
## ENSG00000245680  7.007457 2.42248075 29.46449 5.695077e-08 0.0007083699
## ENSG00000157680  7.237809 1.36241855 28.17447 1.108570e-07 0.0007083699
## ENSG00000197261  6.418993 3.15686745 27.50494 1.566940e-07 0.0007083699
## ENSG00000138650 -8.274919 2.35084765 27.26492 1.774010e-07 0.0007083699
## ENSG00000151690  6.547575 1.20130072 26.28609 2.944008e-07 0.0007083699
## ENSG00000185686  5.101937 0.73285441 26.00984 3.396824e-07 0.0007083699
## ENSG00000244607  4.559426 0.01356324 25.73511 3.916360e-07 0.0007083699
## ENSG00000175105 -7.497481 2.83568134 25.65137 4.090053e-07 0.0007083699
## ENSG00000127366 -7.248685 1.81131942 25.42882 4.590090e-07 0.0007083699
## ENSG00000169228  8.417010 2.31022748 25.39601 4.668829e-07 0.0007083699
```

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

```
##    reads.qc[, keep]$batchNA19101.r3
## -1                              232
## 0                             15307
## 1                               522
```

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



Chapter 18

Clustering Introduction

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

18.1 Introduction

One of the most promising applications of scRNA-seq is *de novo* discovery and annotation of cell-types based on 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. Moreover, in most situations we do not even know the number of clusters *a priori*. The problem is made even more challenging due to the high level of noise (both technical and biological) and the large number of dimensions (i.e. genes).

18.2 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. We have already discussed PCA (chapter **??**) and t-SNE (chapter **??**).

18.3 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 **hierarchical clustering**, **k-means clustering** and **graph-based clustering**.

18.3.1 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:

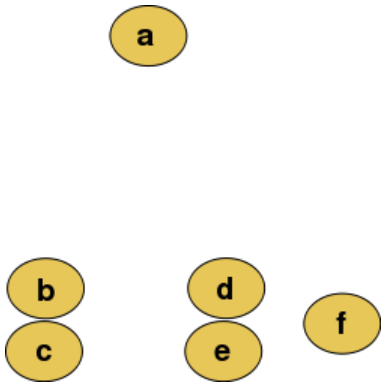


Figure 18.1: Raw data

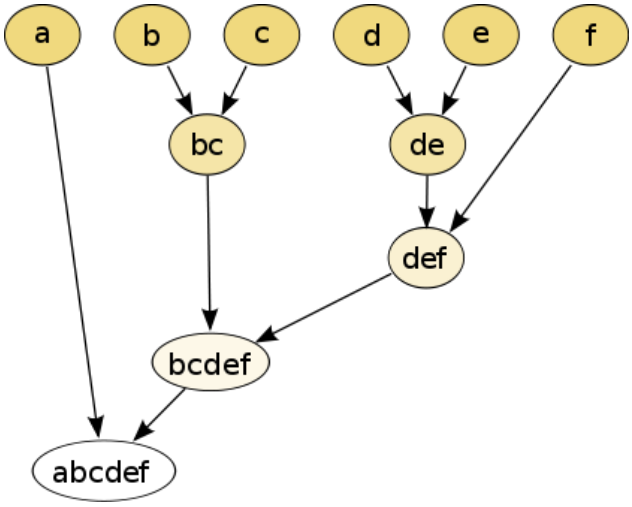


Figure 18.2: The hierarchical clustering dendrogram

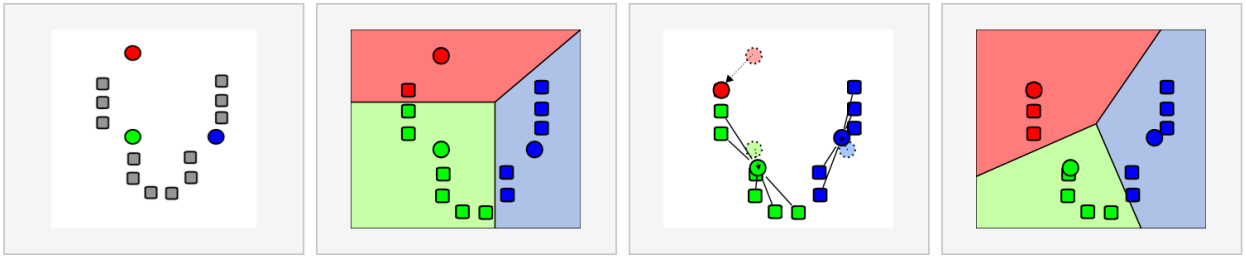


Figure 18.3: Schematic representation of the k-means clustering

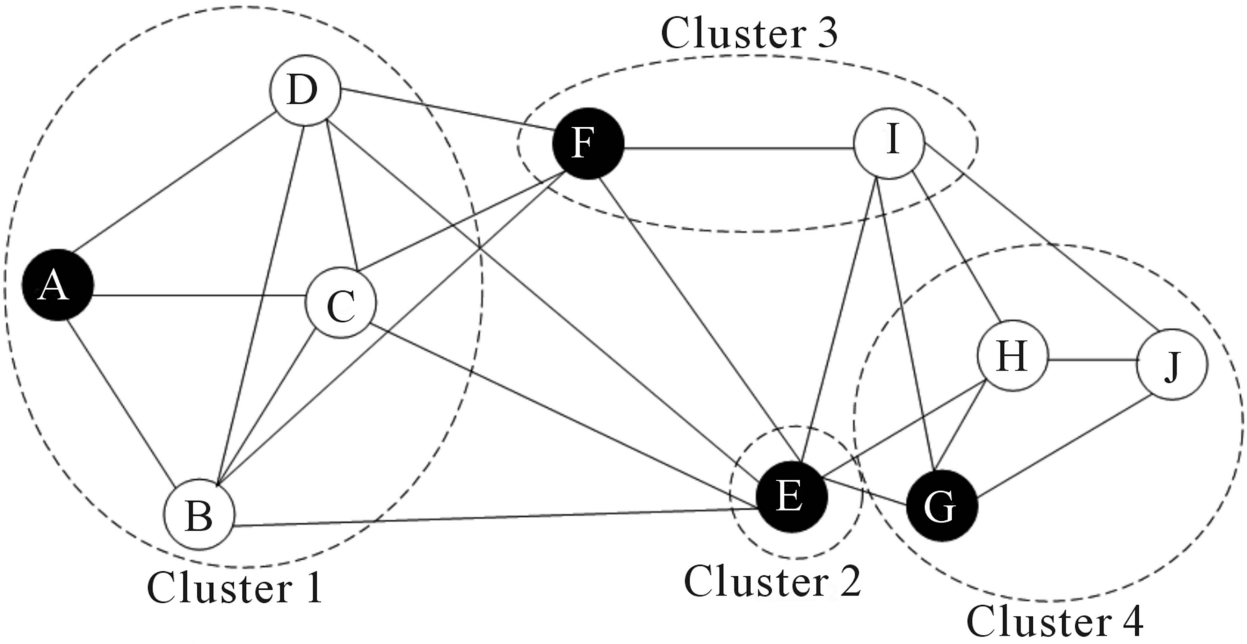


Figure 18.4: Schematic representation of the graph network

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.

18.3.2 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.

18.3.3 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 by building a graph where each node represents a cell. Note that constructing the graph and assigning weights to the edges is not trivial. One advantage

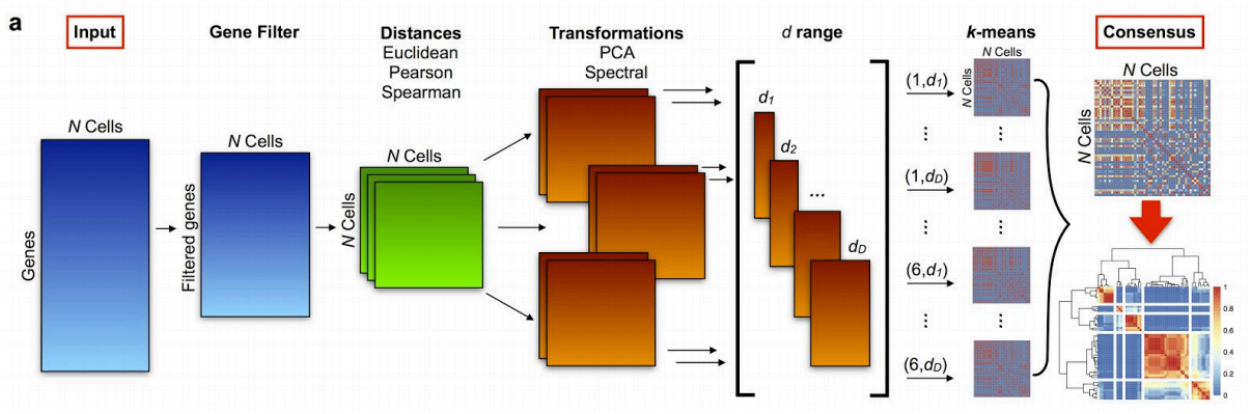


Figure 18.5: SC3 pipeline

of graph-based methods is that some of them are very efficient and can be applied to networks containing millions of nodes.

18.4 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

18.5 Tools for scRNA-seq data

18.5.1 SINCERA

- SINCERA (**?**) is based on hierarchical clustering
- Data is converted to z-scores before clustering
- Identify k by finding the first singleton cluster in the hierarchy

18.5.2 pcaReduce

pcaReduce (**?**) 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.

18.5.3 SC3

- SC3 (**?**) is based on PCA and spectral dimensionality reductions
- Utilises k-means
- Additionally performs the consensus clustering

18.5.4 tSNE + k-means

- Based on **tSNE** maps
- Utilises *k*-means

18.5.5 SEURAT

In the newest versions of SEURAT (v. 1.3-1.4) the clustering is based on a *community detection* approach similar to one previously proposed for analyzing CyTOF data (**?**). **tSNE** is only used exclusively for visualization. In the next chapter we will be using the latest version of `SEURAT`.

Note

In the original version `SEURAT` (**?**) first utilised PCA on a set of cells, then a number of statistically significant PCs were defined. Those PCs were further projected to a 2D space using tSNE. The remaining cells were projected on the same tSNE map. Density clustering algorithm (DBSCAN) was then used to identify cell clusters in the 2D space.

18.5.6 SNN-Cliq

SNN-Cliq (**?**) 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 19

Clustering example

```
library(pcaMethods)
library(pcaReduce)
library(SC3)
library(scater)
library(pheatmap)
set.seed(1234567)
```

To illustrate clustering of scRNA-seq data, we consider the `Pollen` dataset of cells from different human tissues (**?**). We have preprocessed the dataset and created a scater object in advance. We have also annotated the cells with the cell type information (it is the `cell_type1` column in the `phenoData` slot).

19.1 Pollen dataset

Let's load the data and look at it:

```
pollen <- readRDS("pollen/pollen.rds")
pollen
```

```
## SCESet (storageMode: lockedEnvironment)
## assayData: 23730 features, 301 samples
##   element names: exprs, is_exprs, tpm
## protocolData: none
## phenoData
##   rowNames: Hi_2338_1 Hi_2338_2 ... Hi_GW16_26 (301 total)
##   varLabels: cell_type1 cell_type2 ... is_cell_control (33 total)
##   varMetadata: labelDescription
## featureData
##   featureNames: A1BG A1BG-AS1 ... ZZZ3 (23730 total)
##   fvarLabels: mean_exprs exprs_rank ... feature_symbol (11 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
```

Let's look at the cell type annotation:

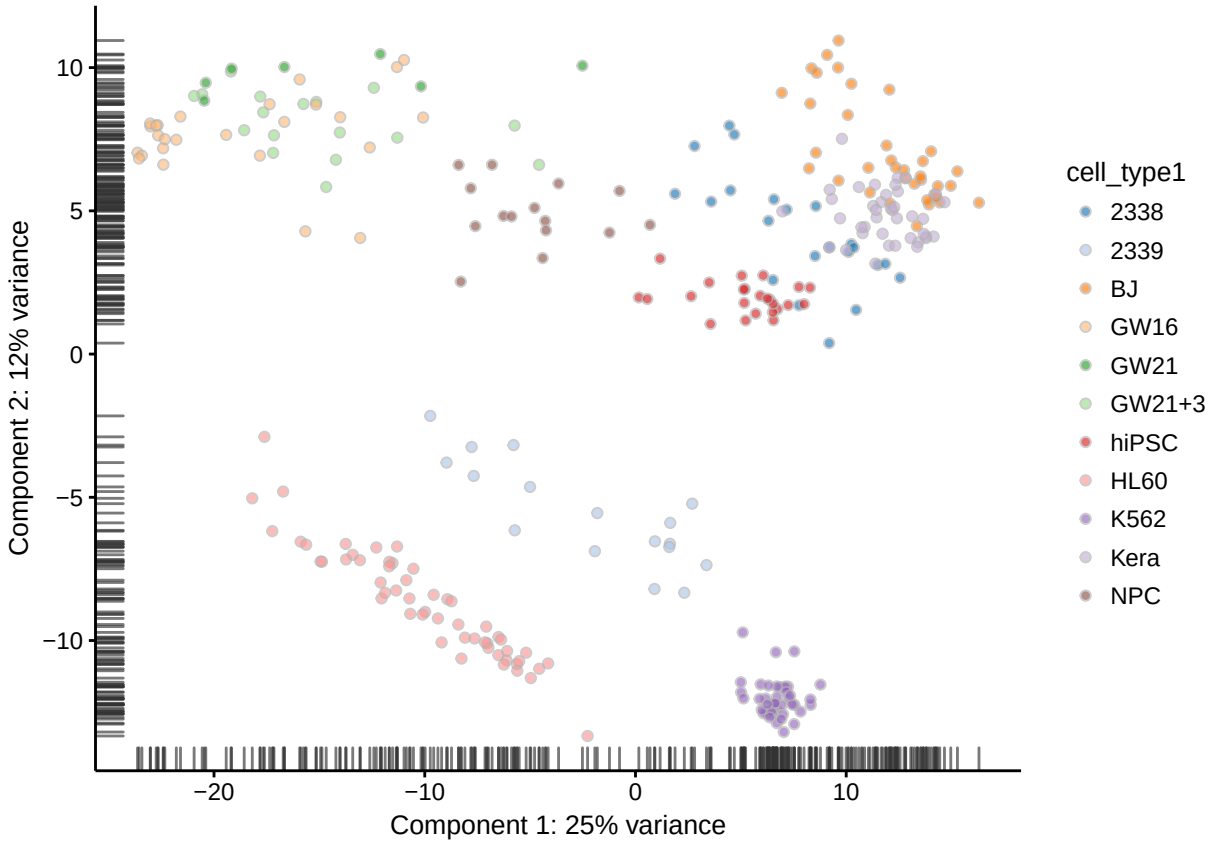
```
table(pData(pollen)$cell_type1)
```

```
##
```

```
##    2338    2339      BJ    GW16    GW21 GW21+3  hiPSC    HL60    K562    Kera
##      22      17      37      26       7      17      24      54      42      40
##     NPC
##      15
```

A simple PCA analysis already separates some strong cell types and provides some insights in the data structure:

```
plotPCA(pollen, colour_by = "cell_type1")
```



19.2 SC3

Let's run `SC3` clustering on the Pollen data. The advantage of the `SC3` is that it can directly take a scater object (see previous chapters) as an input.

Now let's image we do not know the number of clusters k (cell types). `SC3` can estimate a number of clusters for you:

```
pollen <- sc3_prepare(pollen, ks = 2:5)
```

```
## Setting SC3 parameters...
```

```
## Setting a range of k...
```

```
pollen <- sc3_estimate_k(pollen)
```

```
## Estimating k...
```

```
pollen@sc3$k_estimation
```

```
## [1] 11
```

Interestingly, the number of cell types predicted by SC3 is the same as the number of cell types in the Pollen data annotation.

Now we are ready to run SC3 (we also ask it to calculate biological properties of the clusters):

```
pollen <- sc3(pollen, ks = 11, biology = TRUE)
```

```
## Setting SC3 parameters...
```

```
## Setting a range of k...
```

```
## Calculating distances between the cells...
```

```
## Performing transformations and calculating eigenvectors...
```

```
## Performing k-means clustering...
```

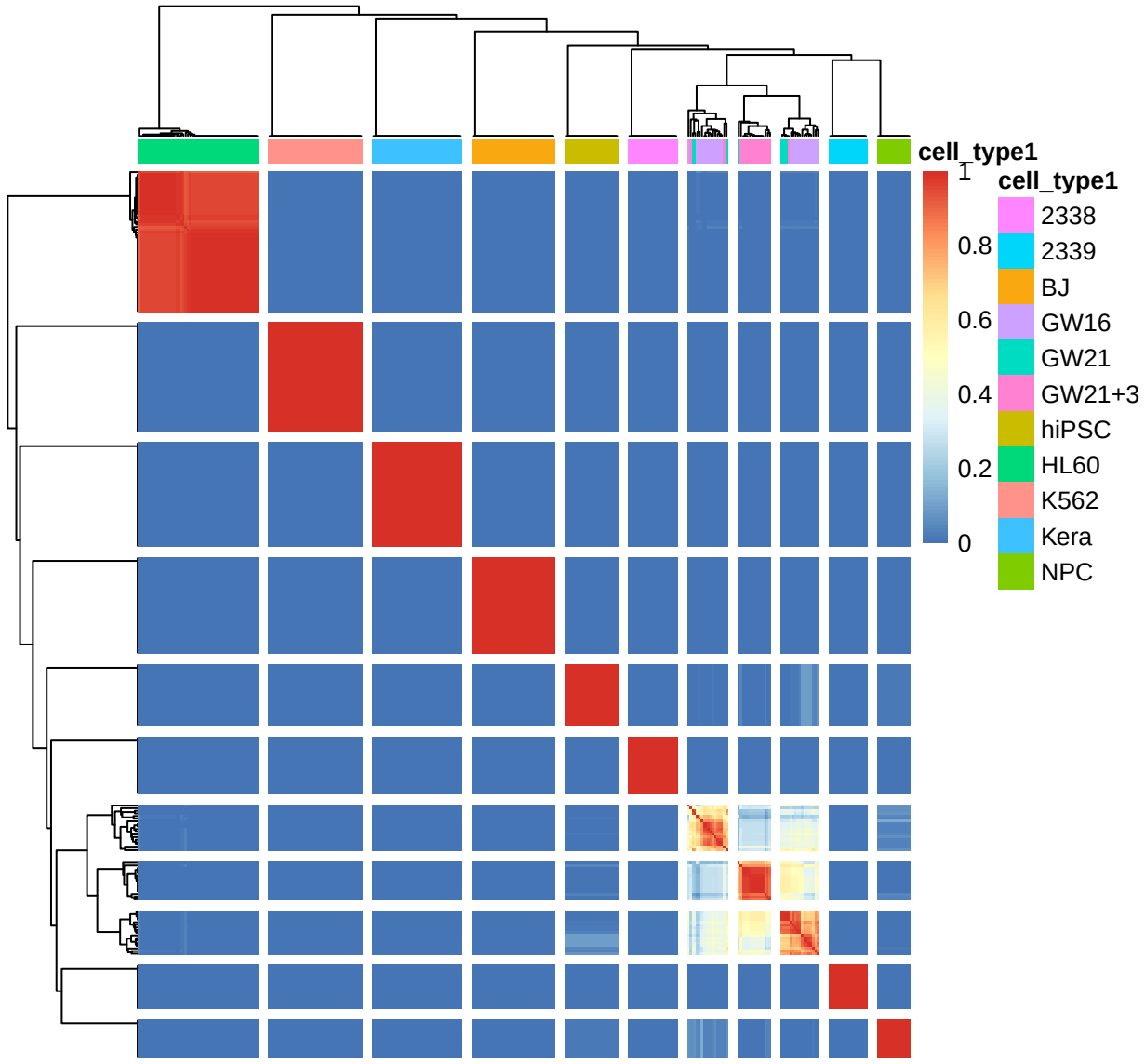
```
## Calculating consensus matrix...
```

```
## Calculating biology...
```

SC3 result consists of several different outputs (please look in (**?**) and SC3 vignette for more details). Here we show some of them:

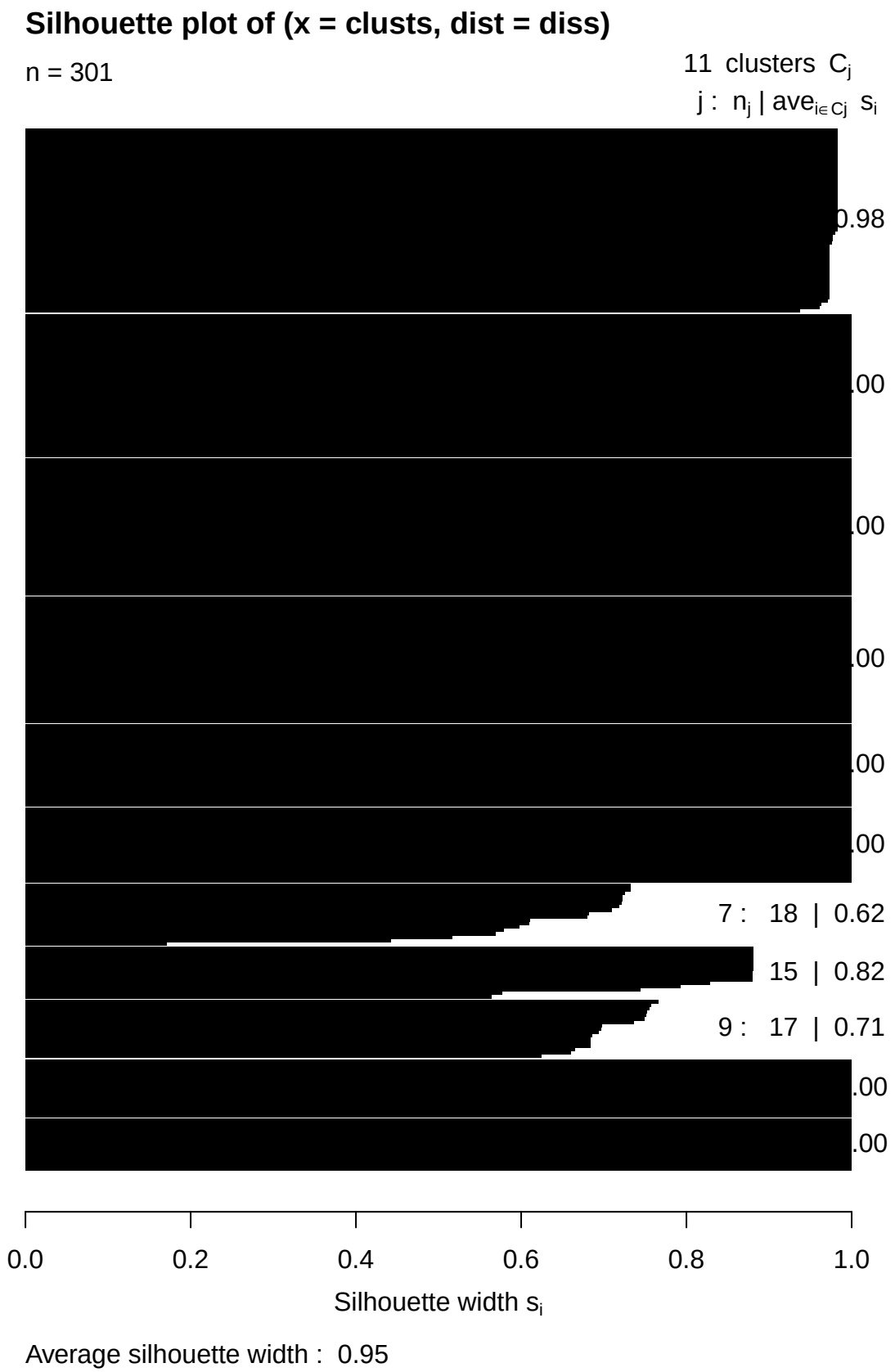
Consensus matrix:

```
sc3_plot_consensus(pollen, k = 11, show_pdata = "cell_type1")
```



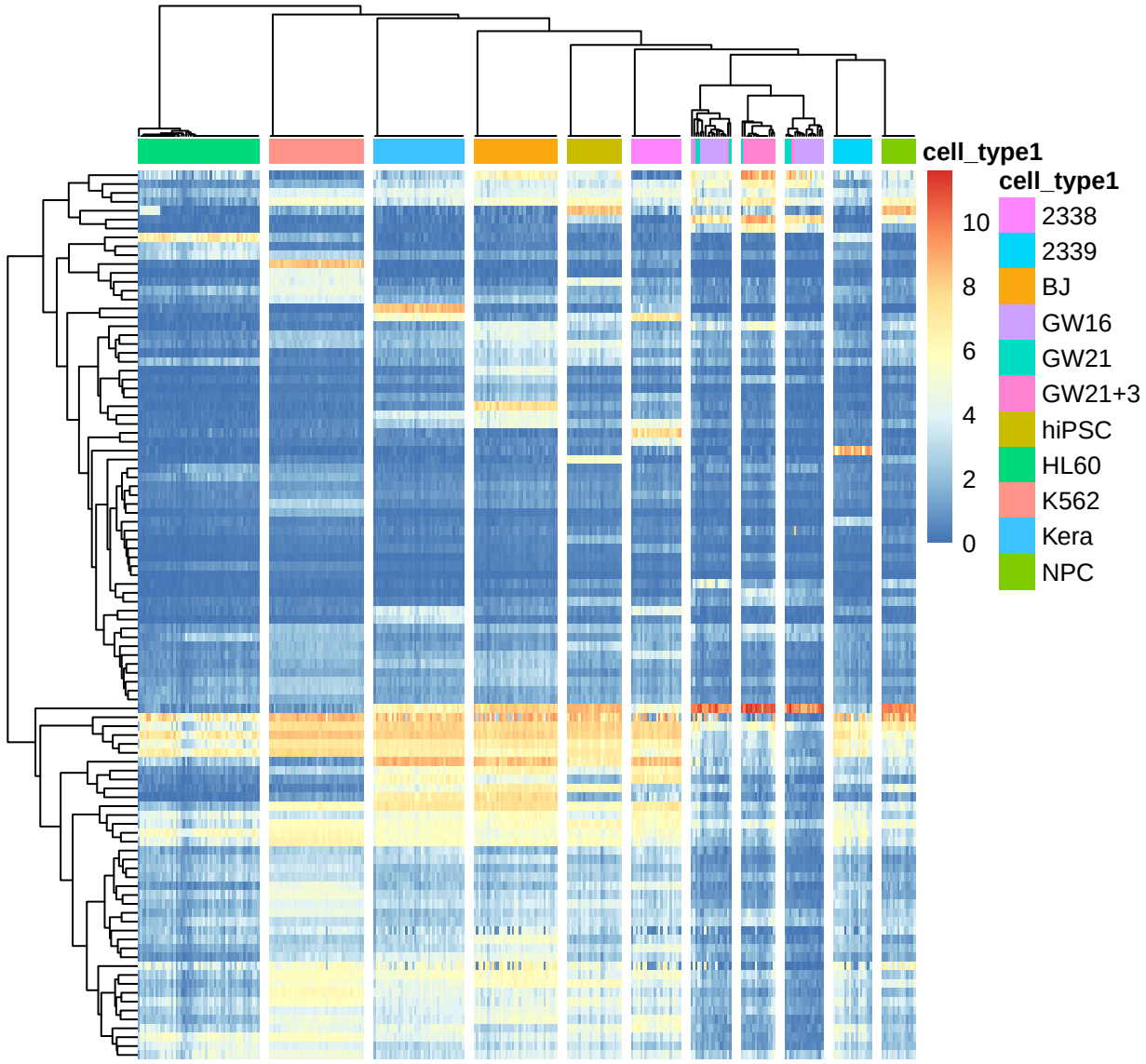
Silhouette plot:

```
sc3_plot_silhouette(pollen, k = 11)
```



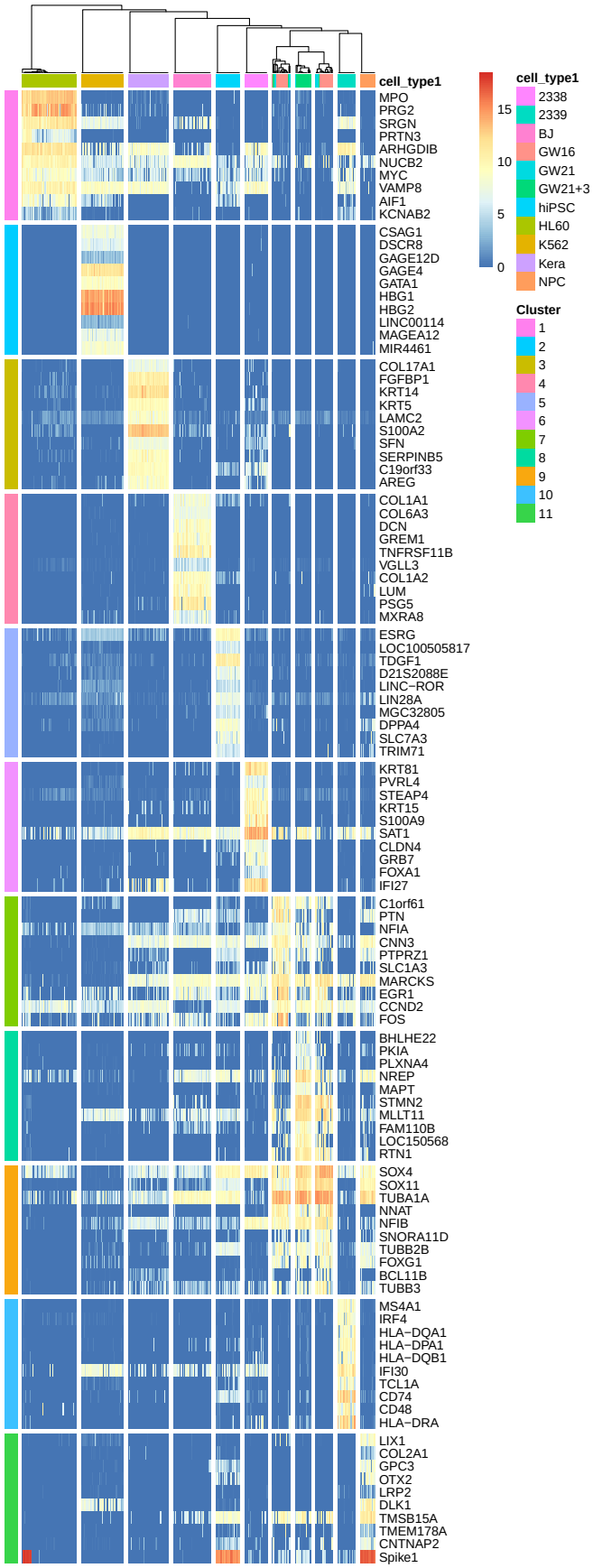
Heatmap of the expression matrix:

```
sc3_plot_expression(pollen, k = 11, show_pdata = "cell_type1")
```



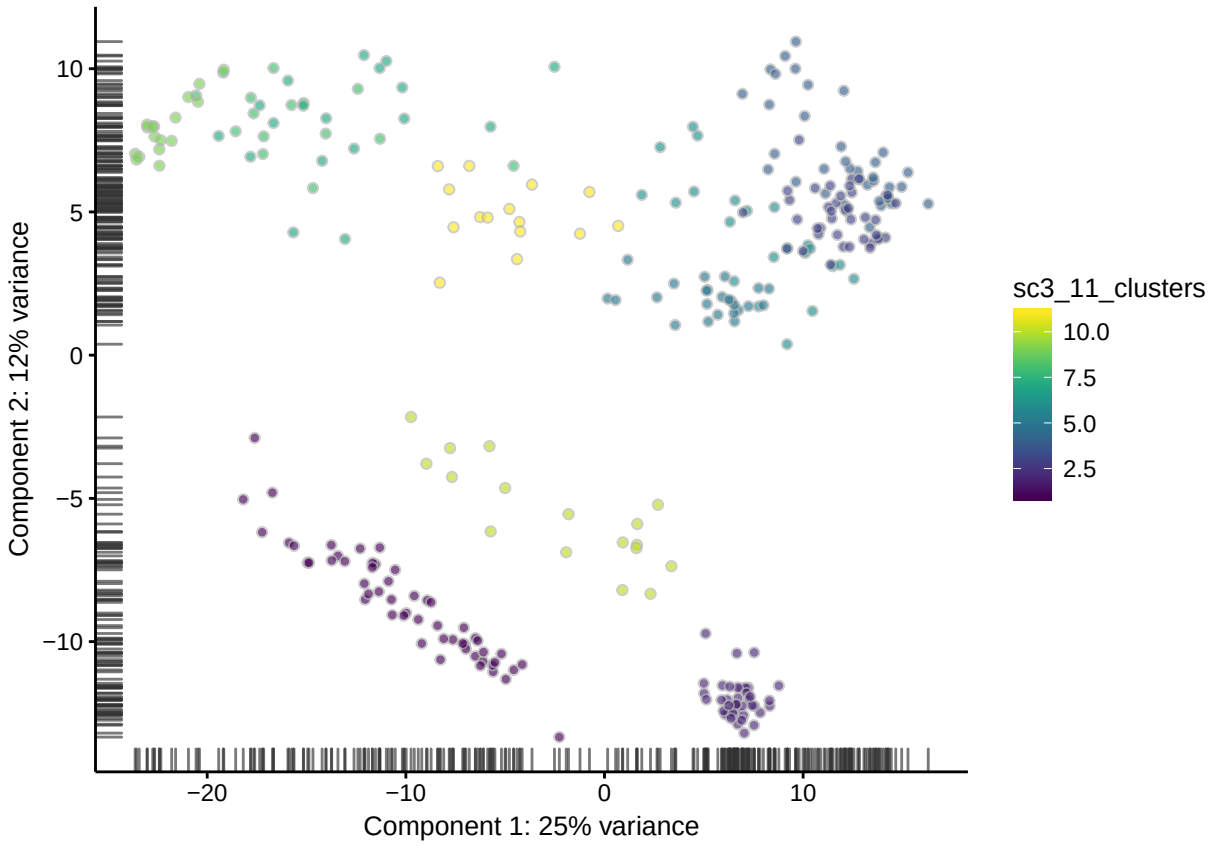
Identified marker genes:

```
sc3_plot_markers(pollen, k = 11, show_pdata = "cell_type1")
```


cell_type1
MPO
PRG2
SRGN
PRTN3
ARHGDIB
NUCB2
MYC
VAMP8
AIF1
KCNAB2
CSAG1
DSCR8
GAGE12D
GAGE4
GATA1
HBG1
HBG2
LINC00114
MAGEA12
MIR4461
COL17A1
FGFBP1
KRT14
KRT5
LAMC2
S100A2
SFN
SERPINB5
C19orf33
AREG
COL1A1
COL6A3
DCN
GREM1
TNFRSF11B
VGLL3
COL1A2
LUM
PSG5
MXRA8
ESRG
LOC100505817
TDGF1
D21S2088E
LINC−ROR
LIN28A
MGC32805
DPPA4
SLC7A3
TRIM71
KRT81
PVRL4
STEAP4
KRT15
S100A9
SAT1
CLDN4
GRB7
FOXA1
IFI27
C1orf61
PTN
NFIA
CNN3
PTPRZ1
SLC1A3
MARCKS
EGR1
CCND2
FOS
BHLHE22
PKIA
PLXNA4
NREP
MAPT
STMN2
MLLT11
FAM110B
LOC150568
RTN1
SOX4
SOX11
TUBA1A
NNAT
NFIB
SNORA11D
TUBB2B
FOXG1
BCL11B
TUBB3
MS4A1
IRF4
HLA−DQA1
HLA−DPA1
HLA−DQB1
IFI30
TCL1A
CD74
CD48
HLA−DRA
LIX1
COL2A1
GPC3
OTX2
LRP2
DLK1
TMSB15A
TMEM178A
CNTNAP2
Spike1
15
10
5
0
cell_type1
2338
2339
BJ
GW16
GW21
GW21+3
hiPSC
HL60
K562
Kera
NPC
Cluster
1
2
3
4
5
6
7
8
9
10
11

PCA plot with highlighted SC3 clusters:

```
plotPCA(pollen, colour_by = "sc3_11_clusters")
```



Note, that one can also run SC3 in an interactive Shiny session:

```
sc3_interactive(pollen)
```

This command will open SC3 in a web browser.

- **Exercise 1**: Run SC3 for k from 9 to 13 and explore different clustering solutions in your web browser.
- **Exercise 2**: Which clusters are the most stable when k is changed from 9 to 13? (Look at the "Stability" tab)
- **Exercise 3**: Check out differentially expressed genes and marker genes for the obtained clusterings. Please use $k = 11$.
- **Exercise 4**: Change the marker genes threshold (the default is 0.85). Does **SC3** find more marker genes?

19.3 pcaReduce

pcaReduce operates directly on the expression matrix. It is recommended to use a gene filter and log transformation before running pcaReduce. We will use the default SC3 gene filter (note that the exprs slot of a scater object is log-transformed by default).

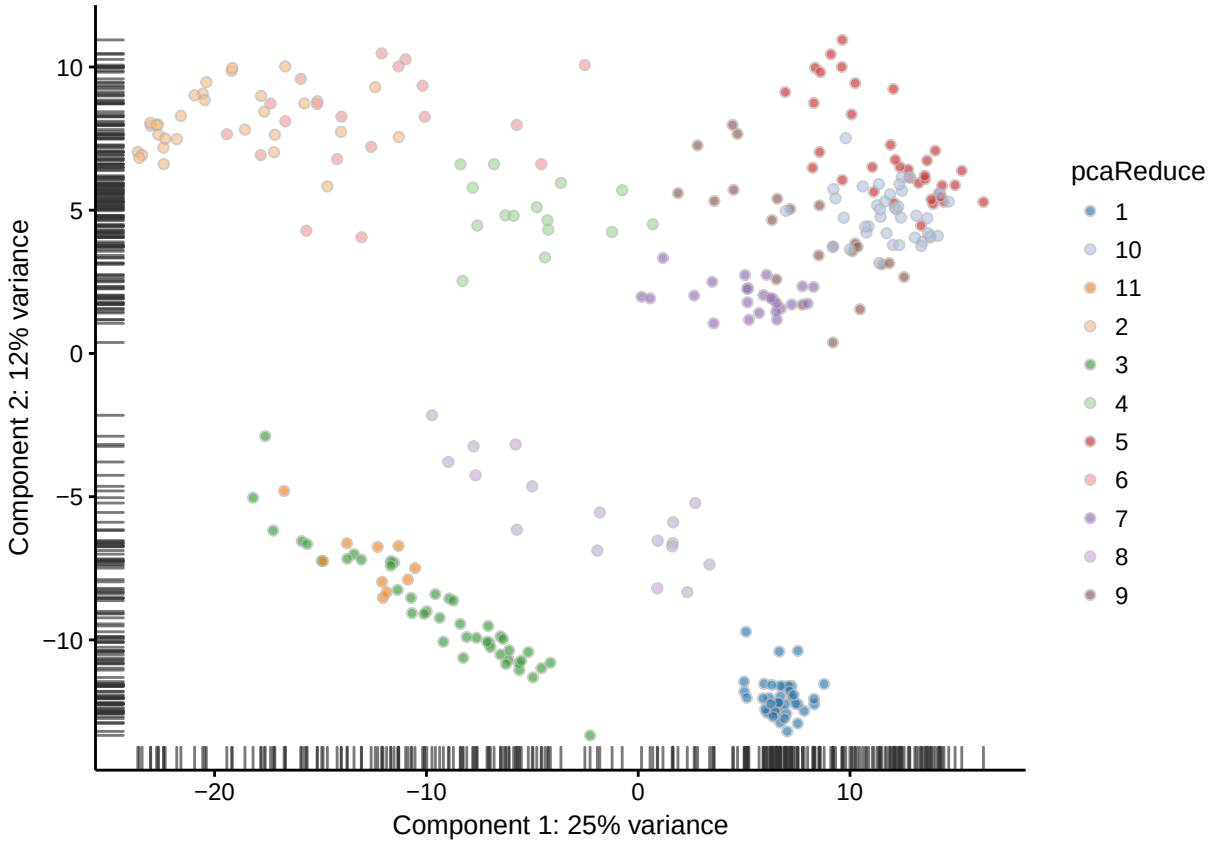
```
# use the same gene filter as in SC3
input <- exprs(pollen[fData(pollen)$sc3_gene_filter, ])
```

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` 1 time:

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

```
pData(pollen)$pcaReduce <- as.character(pca.red[,32 - 11])
plotPCA(pollen, colour_by = "pcaReduce")
```



Exercise 5: Run pcaReduce for $k = 2$ and plot a similar PCA plot. Does it look good?

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

Our solution:

Exercise 6: Compare the results between `SC3` and `pcaReduce` for $k = 11$. What is the main difference between the solutions provided by the two different methods?

Our solution:

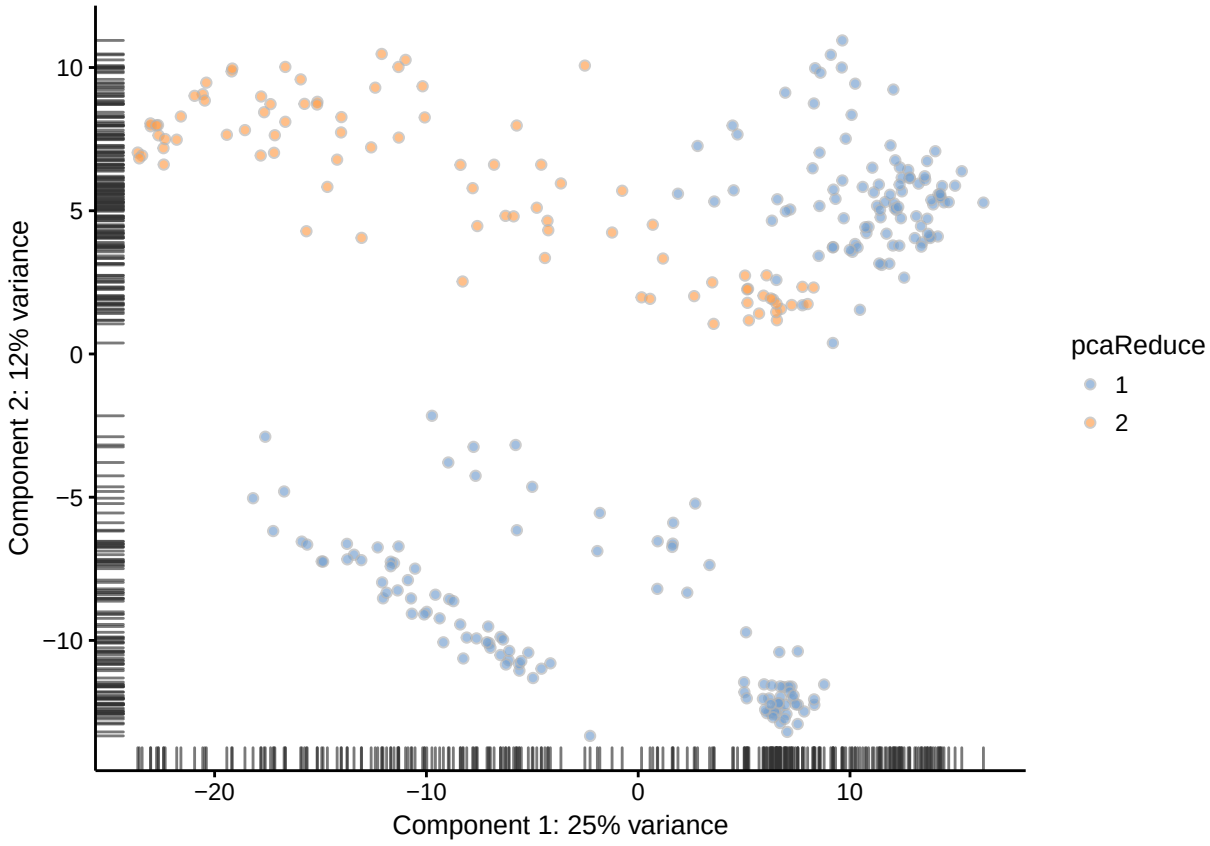
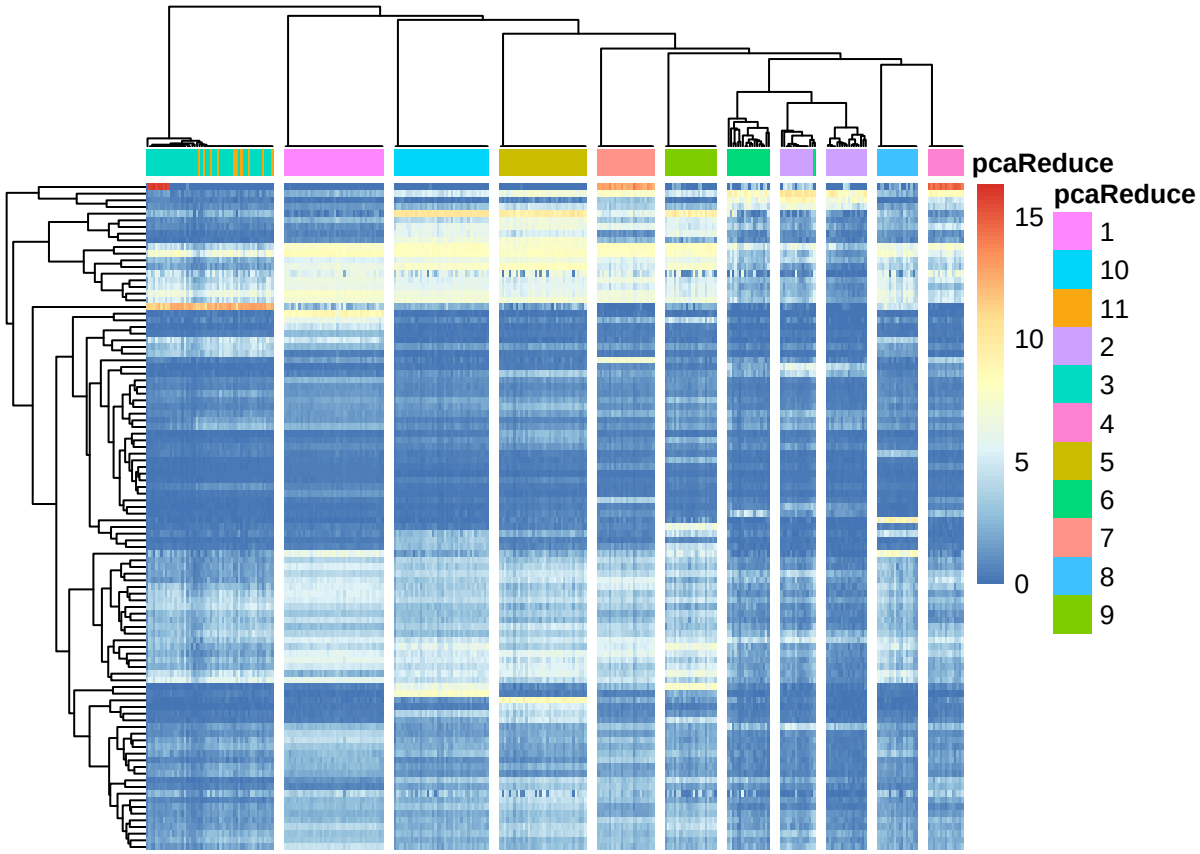


Figure 19.1: Clustering solutions of pcaReduce method for $k = 2$.



19.4 tSNE + kmeans

tSNE plots that we saw before (**??**) when used the **scater** package are made by using the Rtsne and ggplot2 packages. Here we will do the same:

```
pollen <- plotTSNE(pollen, rand_seed = 1, return_SCESet = TRUE)
```

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. How many groups do you see in the cloud?

We will start with $k = 8$:

```
pData(pollen)$tSNE_kmeans <- as.character(kmeans(pollen@reducedDimension, centers = 8)$clust)
plotTSNE(pollen, rand_seed = 1, colour_by = "tSNE_kmeans")
```

Exercise 7: Make the same plot for $k = 11$.

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

Our solution:

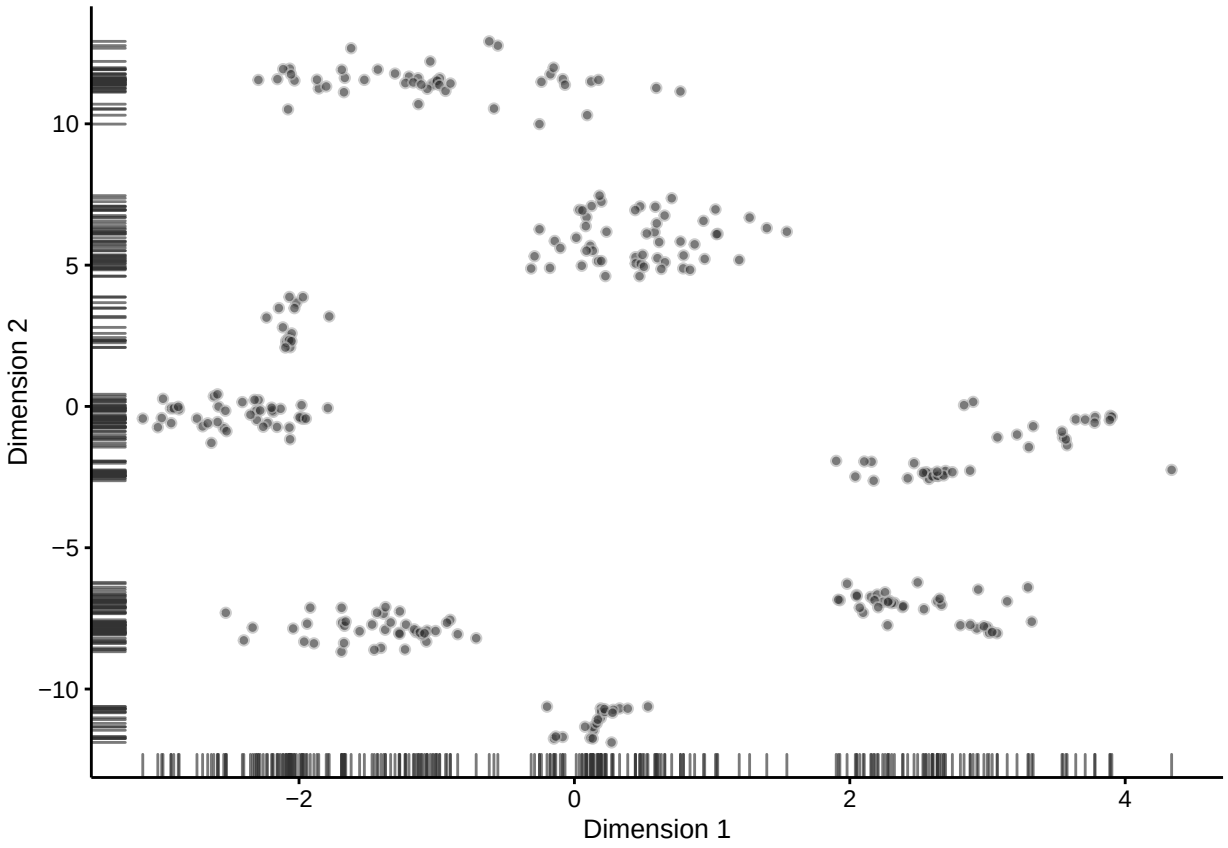


Figure 19.2: tSNE map of the patient data

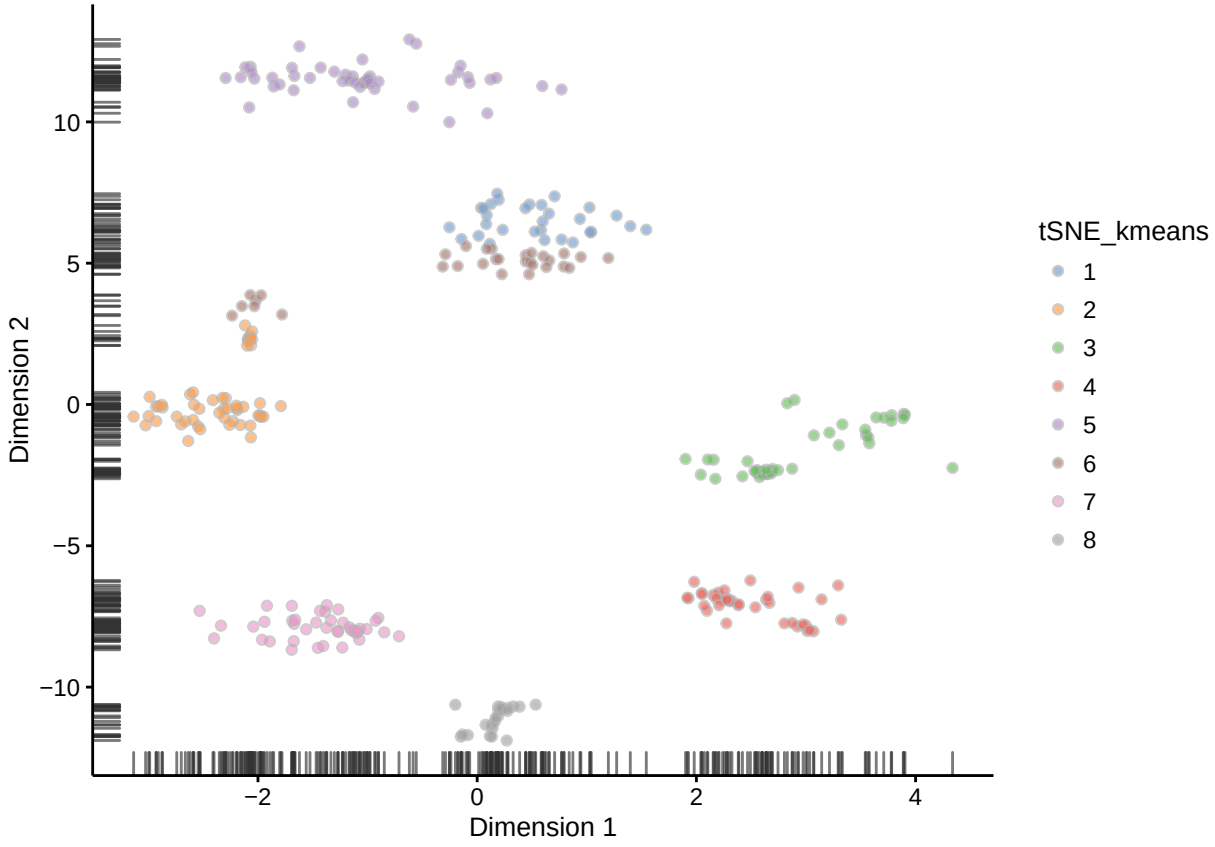
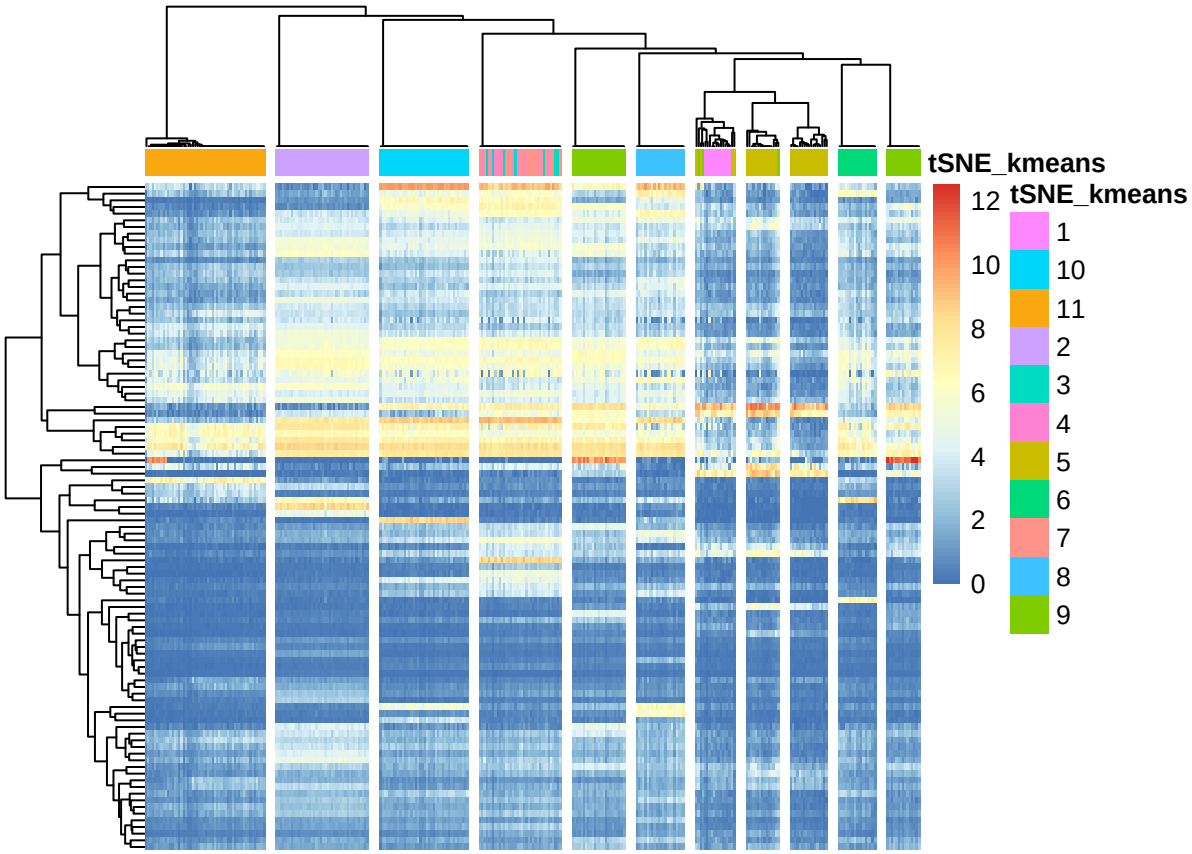


Figure 19.3: tSNE map of the patient data with 8 colored clusters, identified by the k-means clustering algorithm



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. `SC3` is also stochastic, but thanks to the consensus step, it is more robust and less likely to produce different outcomes.

19.5 SNN-Cliq

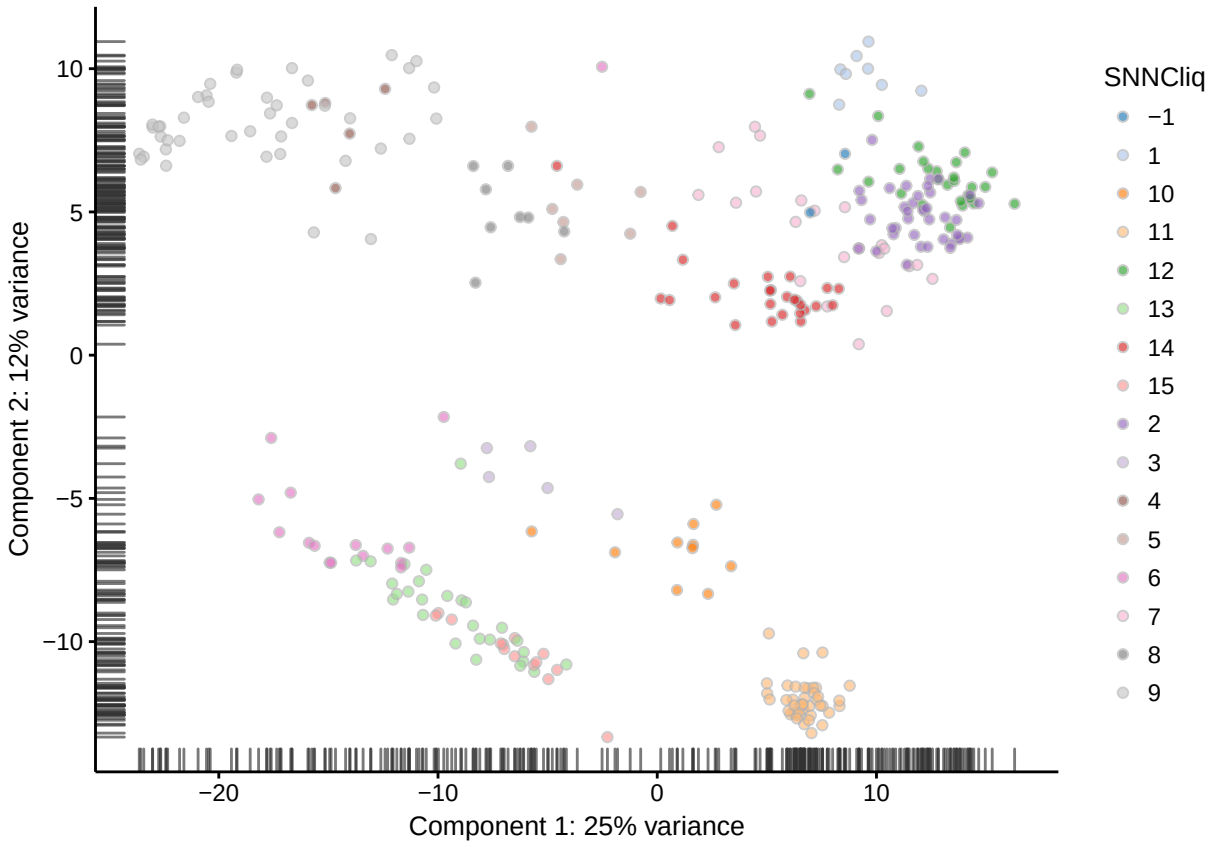
Here we run SNN-cliq with te default parameters provided in the author's example:

```
distan <- "euclidean"
par.k <- 3
par.r <- 0.7
par.m <- 0.5
# construct a graph
scRNA.seq.funcs::SNN(
    data = t(input),
    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"))
```

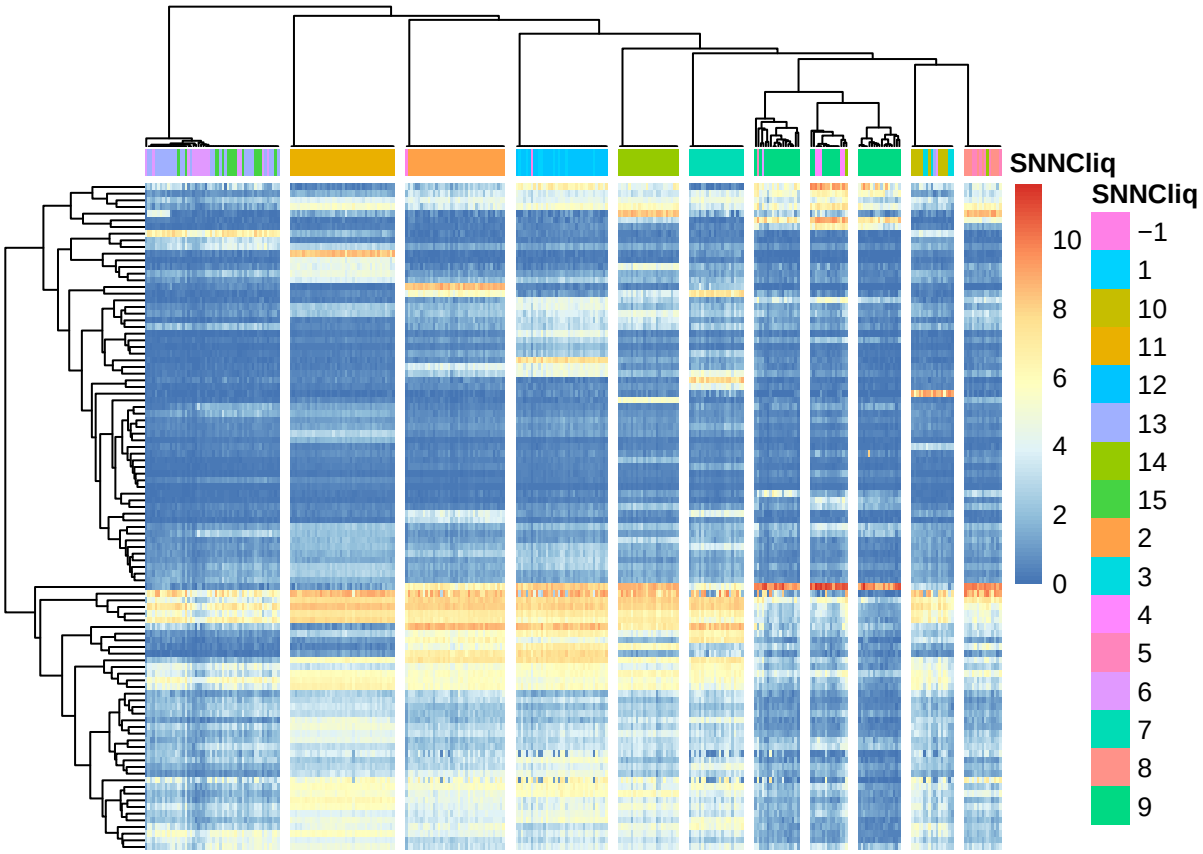
```
## input file snn-cliq.txt
## find 65 quasi-cliques
## merged into 15 clusters
## unique assign done
```

```
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")

pData(pollen)$SNNCliq <- as.character(snn.res[,1])
plotPCA(pollen, colour_by = "SNNCliq")
```



Exercise 9: Compare the results between `SC3` and `SNN-Cliq`.

Our solution:



19.6 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
    } else {
        break;
    }
}
```

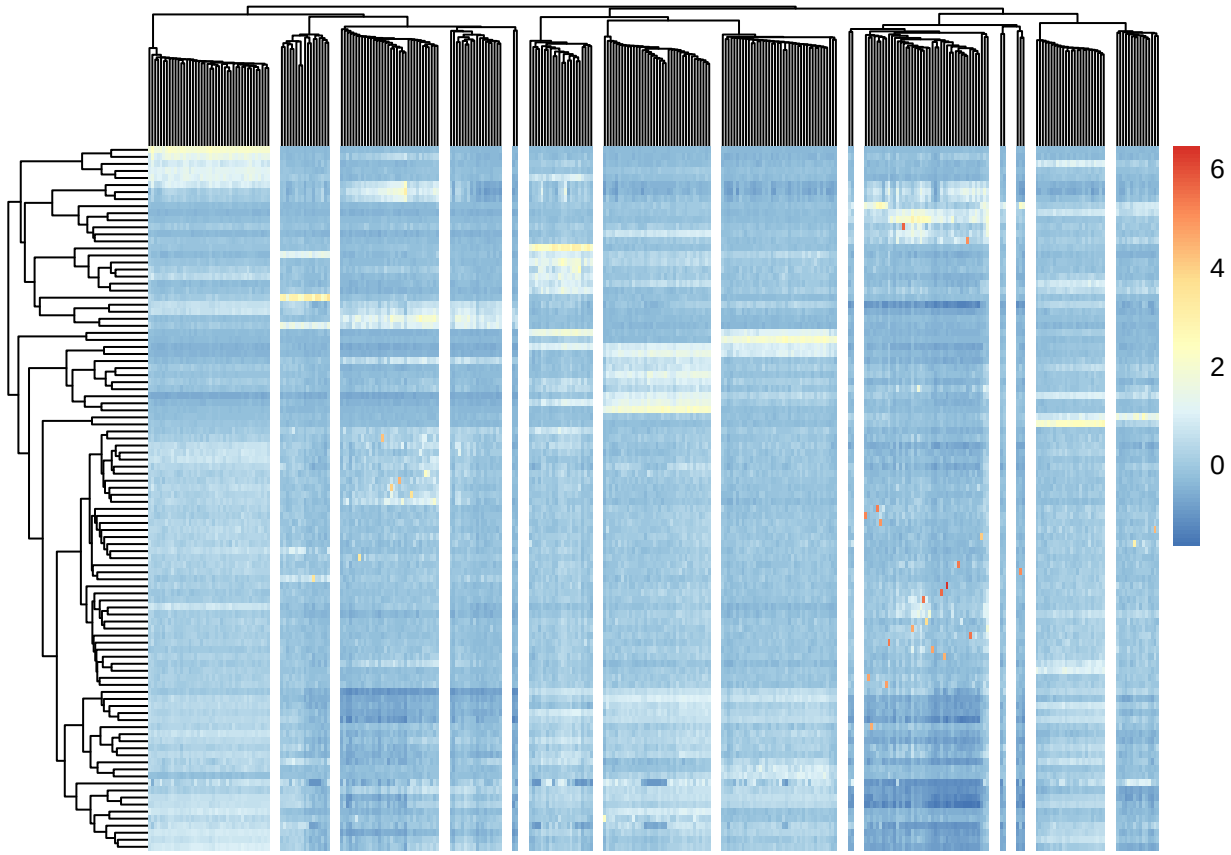


Figure 19.4: Clustering solutions of SINCERA method using $k = 3$

```
cat(kk)
```

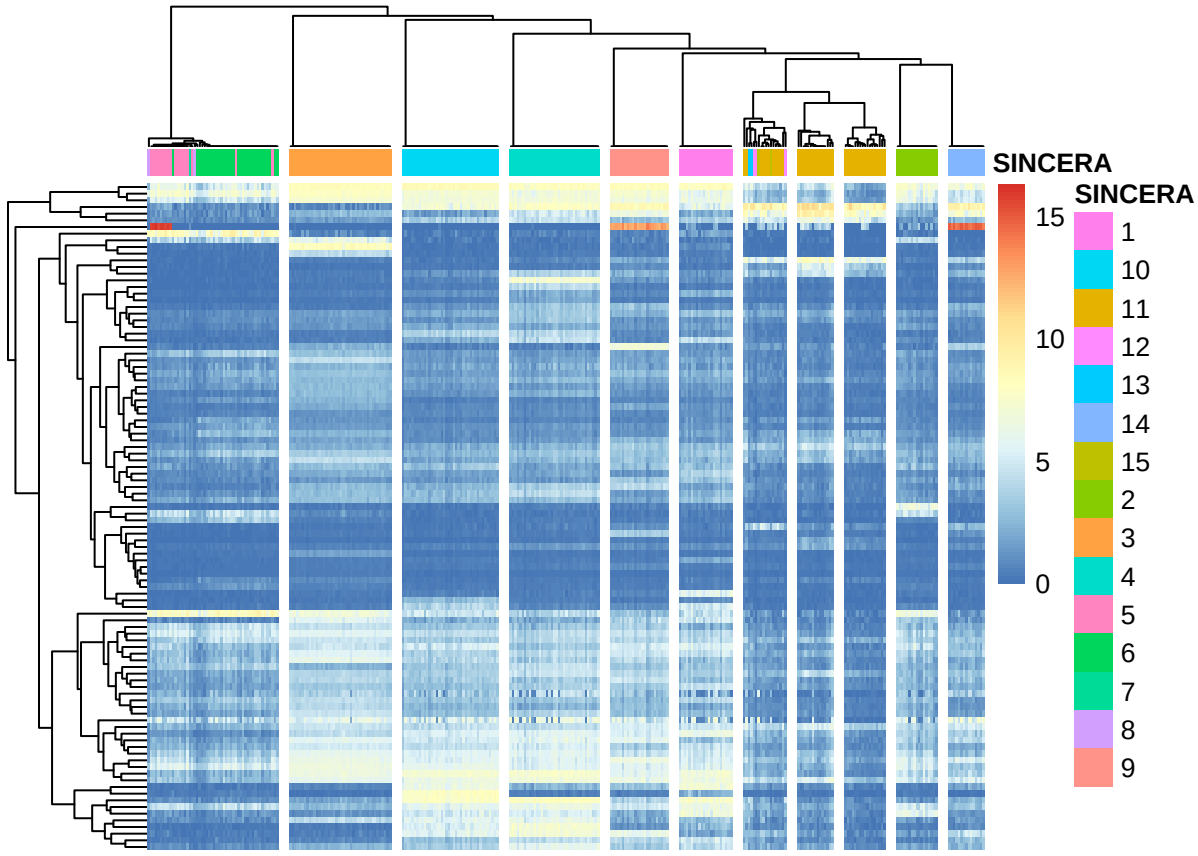
```
## 14
```

Let's now visualize the SINCERA results as a heatmap:

```
pheatmap(
    t(dat),
    cluster_cols = hc,
    cutree_cols = 14,
    kmeans_k = 100,
    show_rownames = FALSE
)
```

Exercise 10: Compare the results between `SC3` and `SNN-Cliq`.

Our solution:

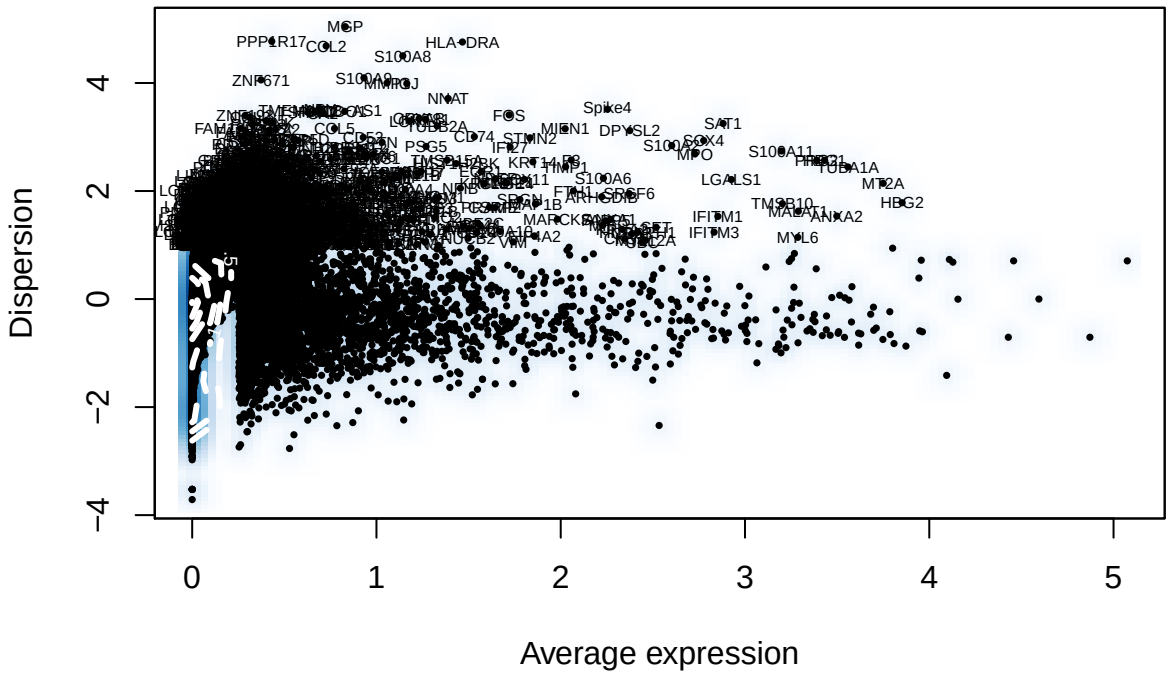


Exercise 11: Is using the singleton cluster criteria for finding **k** a good idea?

19.7 SEURAT

Here we follow an example created by the authors of `SEURAT` (8,500 Pancreas cells). We mostly use default values in various function calls, for more details please consult the documentation and the authors:

```
library(Seurat)
library(Matrix)
pollen_seurat <- new("seurat", raw.data = get_exprs(pollen, exprs_values = "tpm"))
pollen_seurat <- Setup(pollen_seurat, project = "Pollen")
pollen_seurat <- MeanVarPlot(pollen_seurat)
```




```
pollen_seurat <- RegressOut(pollen_seurat, latent.vars = c("nUMI"),
                            genes.regress = pollen_seurat@var.genes)
```

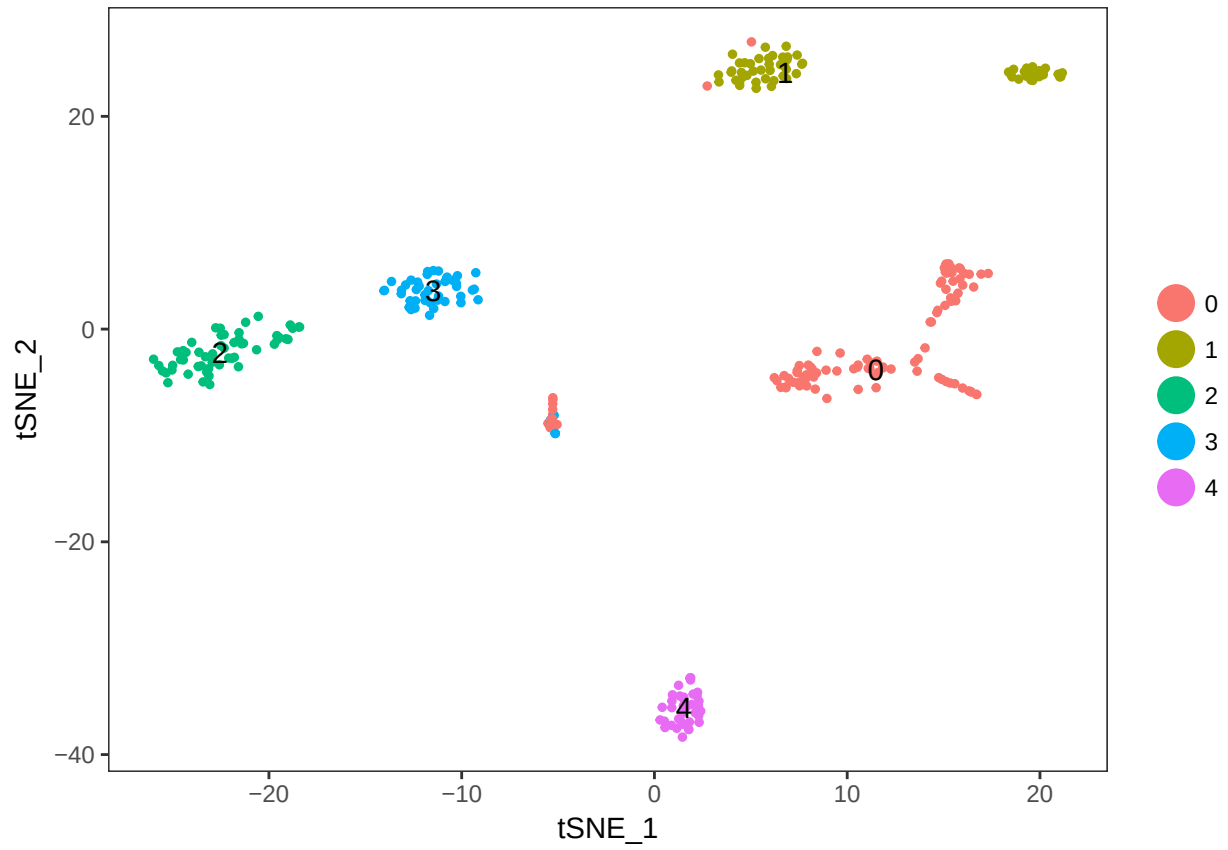
```
## [1] "Regressing out nUMI"
```

```
pollen_seurat <- PCAFast(pollen_seurat)
```

```
## [1] "PC1"
##  [1] "MAP1B"     "TUBA1A"    "SOX11"     "DPYSL2"    "DCX"
##  [6] "STMN2"     "RTN1"      "GPM6A"     "DPYSL3"    "MLLT11"
## [11] "SOX4"      "MAP2"      "CRMP1"     "LOC150568" "NREP"
## [16] "CALM1"     "FAM110B"   "NFIB"      "MAPT"      "TUBB2B"
## [21] "RUFY3"     "NNAT"      "CXADR"     "FXYD6"     "MIR100HG"
## [26] "FOXG1"     "POU3F2"    "KIF5C"     "MN1"       "NCAM1"
## [1] ""
##  [1] "MYL12A"    "ARHGDIB"   "S100A11"   "IFITM1"    "CKS2"
##  [6] "SRGN"      "IFITM3"    "ANXA1"     "IFI30"     "GTSF1"
## [11] "MT2A"      "ISG15"     "KRT8"      "CD53"      "ANXA2"
## [16] "HIST1H1C"  "HIST1H2BK" "LAPTM5"    "LGALS1"    "NOB1"
## [21] "MPO"       "PRG2"      "KRT18"     "AIF1"      "NUCB2"
## [26] "PRAME"     "PSMB9"     "SFXN4"     "TRAP1"     "RNF114"
## [31] "CD52"
## [1] ""
## [1] ""
## [1] "PC2"
##  [1] "S100A6"   "CD44"     "TPM1"     "ANXA2"    "IFITM3"   "ANXA1"
##  [7] "S100A11"  "RND3"     "LGALS1"   "IGFBP3"   "DKK1"     "MT2A"
## [13] "THBS1"    "TGFBI"    "PLS3"     "TMSB10"   "SERPINE1" "KRT7"
## [19] "FN1"      "S100A10"  "TIMP1"    "CTGF"     "SAT1"     "PLAUR"
## [25] "CAV1"     "DDAH1"    "TPM2"     "MYL6"     "EMP1"     "CCND1"
## [1] ""
##  [1] "MPO"          "SRGN"         "PRG2"         "AIF1"
##  [5] "MS4A3"        "GTSF1"        "CTSG"         "TESC"
```

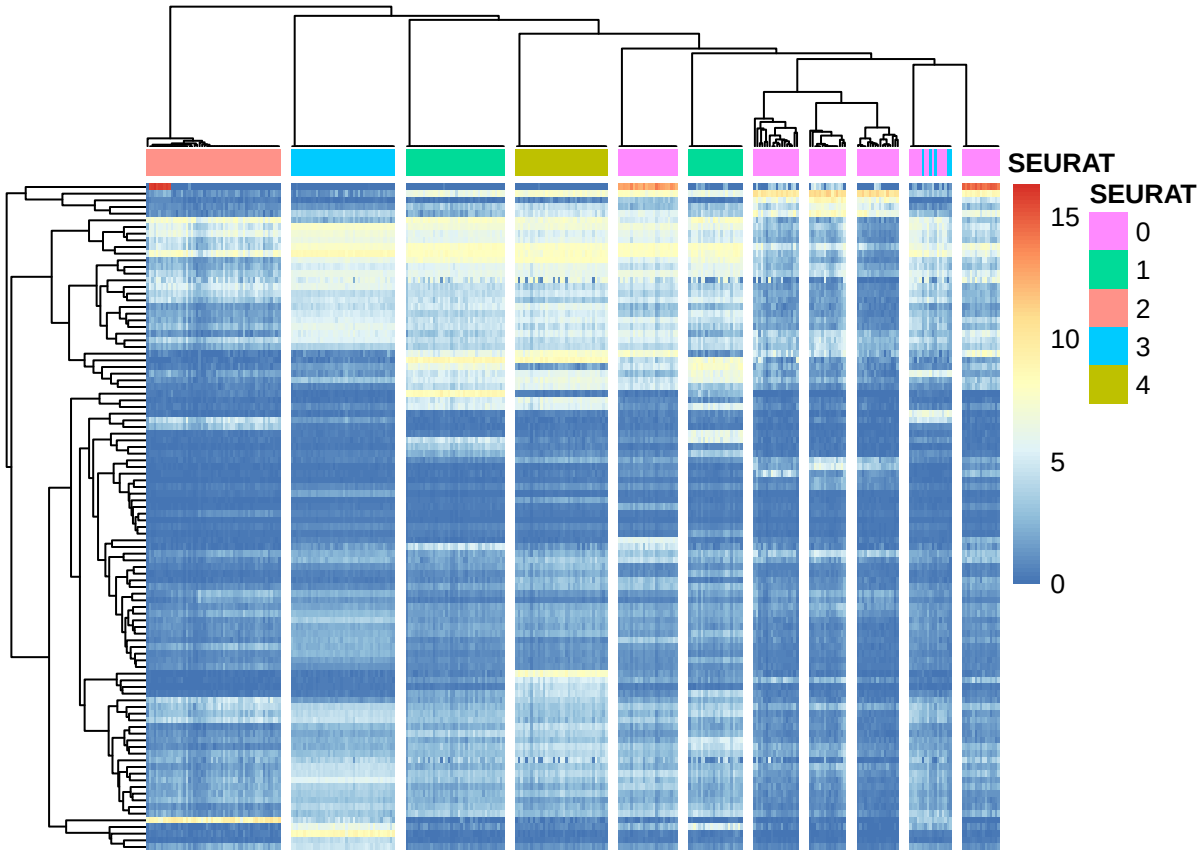
```
##  [9] "MZB1"         "LOC100190986" "CD53"         "LAPTM5"
## [13] "ARHGDIB"      "TRAP1"        "SPNS3"        "RNASE2"
## [17] "PRTN3"        "BTK"          "APOC2"        "CD33"
## [21] "LOC100272216" "HSH2D"        "CLEC5A"       "SLC43A1"
## [25] "PRAME"        "DOK3"         "SERPINB10"    "PLEK"
## [29] "DOCK2"        "CD52"         "NDUFV1"
## [1] ""
## [1] ""
## [1] "PC3"
##  [1] "COL1A2"    "LUM"       "DCN"       "GREM1"     "PSG5"
##  [6] "TNFRSF11B" "SERPINE1"  "TPM2"      "COL1A1"    "LOX"
## [11] "TIMP3"     "VIM"       "TAGLN"     "CRYAB"     "S100A4"
## [16] "SULF1"     "GLIPR1"    "DKK1"      "COX7A1"    "RGS4"
## [21] "CCDC80"    "SERPINE2"  "FN1"       "THBS1"     "KRT34"
## [26] "ALDH1A1"   "KIAA1199"  "FGF7"      "TIMP1"     "WBP5"
## [1] ""
##  [1] "KRT81"    "STEAP4"   "LCN2"     "S100A9"   "KRT15"    "ELF3"
##  [7] "CEACAM6"  "CLDN4"    "RARRES1"  "SLPI"     "KLK5"     "GRB7"
## [13] "DHRS3"    "CXorf61"  "RARRES3"  "KLK6"     "IFI27"    "AZGP1"
## [19] "S100A8"   "MYEOV"    "CXCL17"   "KLK8"     "PDZK1IP1" "BMP3"
## [25] "MUC1"     "FOLR1"    "TNFSF10"  "MIEN1"    "KRT23"    "VGLL1"
## [31] "ALDH1A3"
## [1] ""
## [1] ""
## [1] "PC4"
##  [1] "PPME1"    "S100A11"  "KIAA1598" "NUCB2"    "SEMA3C"   "MRPS10"
##  [7] "MAPT"     "STMN2"    "GTSF1"    "TRIM16"   "DLX6-AS1" "MYT1L"
## [13] "KRT18"    "PQBP1"    "MT2A"     "ANXA1"    "CXADR"    "POLR2C"
## [19] "CDK5R1"   "MPO"      "DLX5"     "INA"      "BCL11B"   "PRG2"
## [25] "KRT8"     "ATAT1"    "PRAME"    "CBWD3"    "S100A10"  "TAF11"
## [1] ""
##  [1] "CD74"     "HLA-DPA1" "HLA-DRA"  "MS4A1"    "HLA-DQA1" "HLA-DRB5"
##  [7] "HLA-DQB1" "HLA-DQA2" "BLNK"     "HLA-DMA"  "HLA-DRB1" "IRF4"
## [13] "HLA-DPB1" "HLA-DMB"  "ELK2AP"   "MIR155HG" "CHI3L2"   "TCL1A"
## [19] "CD48"     "LRMP"     "SLAMF1"   "BCL2A1"   "LY86"     "CLECL1"
## [25] "HLA-A"    "CRIP1"    "CD27"     "CCL3"     "CYTIP"    "JUN"
## [31] "PTN"
## [1] ""
## [1] ""
## [1] "PC5"
##  [1] "GLI3"    "PTN"     "SLC1A3"  "PTPRZ1"  "MDK"     "GPX3"    "CDO1"
##  [8] "CLU"     "ITGB8"   "JUN"     "AIF1L"   "FAM107A" "ATP1B2"  "FOS"
## [15] "TSPAN6"  "HEPN1"   "PAX6"    "HOPX"    "FABP7"   "ID4"     "PON2"
## [22] "SHISA2"  "GPM6B"   "PELI2"   "PMP2"    "ABAT"    "FBXO32"  "NFIA"
## [29] "C1orf61" "CNN3"
## [1] ""
##  [1] "HLA-DPA1" "CD74"     "HLA-DRA"  "HLA-DQA1" "MS4A1"    "HLA-DRB5"
##  [7] "HLA-DQB1" "HLA-DQA2" "BLNK"     "HLA-DMA"  "HLA-DRB1" "IRF4"
## [13] "LPXN"     "MIR155HG" "ELK2AP"   "HLA-DPB1" "HLA-DMB"  "LRMP"
## [19] "CHI3L2"   "CD48"     "BCL2A1"   "TCL1A"    "CLECL1"   "SLAMF1"
## [25] "LY86"     "IFI30"    "CRIP1"    "CD27"     "CCL3"     "IGJ"
## [31] "LAPTM5"
## [1] ""
## [1] ""
```

```
pollen_seurat <- RunTSNE(pollen_seurat)
pollen_seurat <- FindClusters(pollen_seurat)
TSNEPlot(pollen_seurat, do.label = T)
```



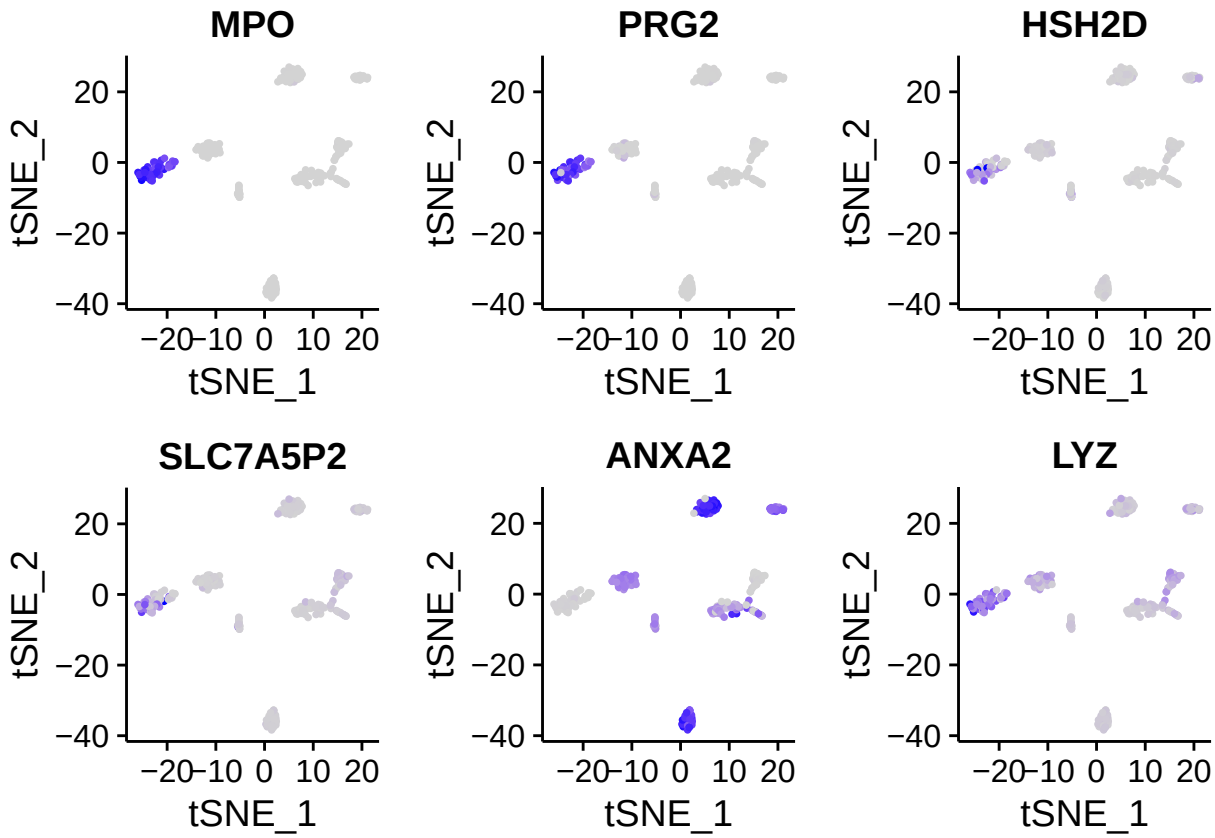
Exercise 12: Compare the results between `SC3` and `SEURAT`.

Our solution:



Seurat can also find marker genes, e.g. marker genes for cluster 2:

```
markers <- FindMarkers(pollen_seurat, 2)
FeaturePlot(pollen_seurat,
            head(rownames(markers)),
            cols.use = c("lightgrey", "blue"),
            nCol = 3)
```



Exercise 13: Compare marker genes provided by `SEURAT` and `SC3`.

Chapter 20

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
```

20.1 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?

20.2 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 present is given by

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

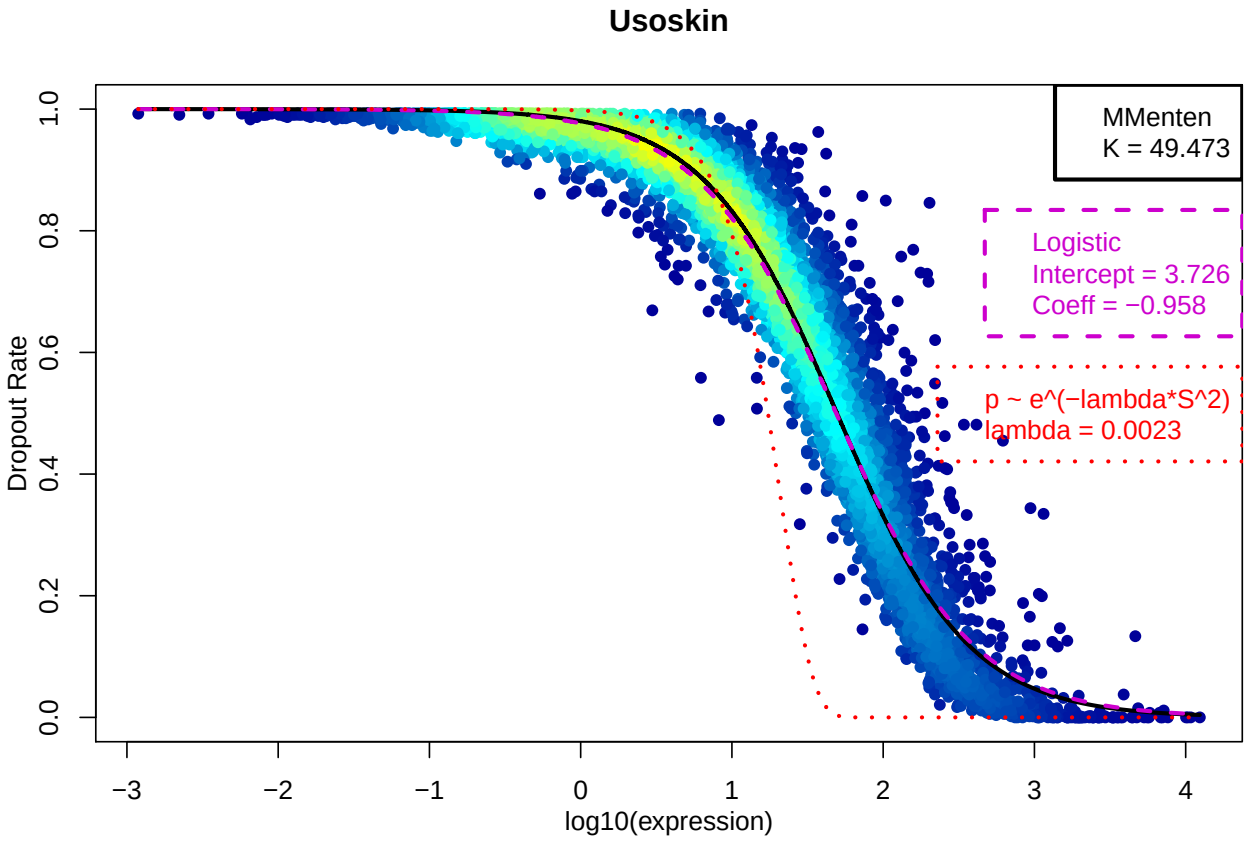
where S is the mRNA concentration in the cell (we will estimate this as average expression) and K is the Michaelis-Menten constant.

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).

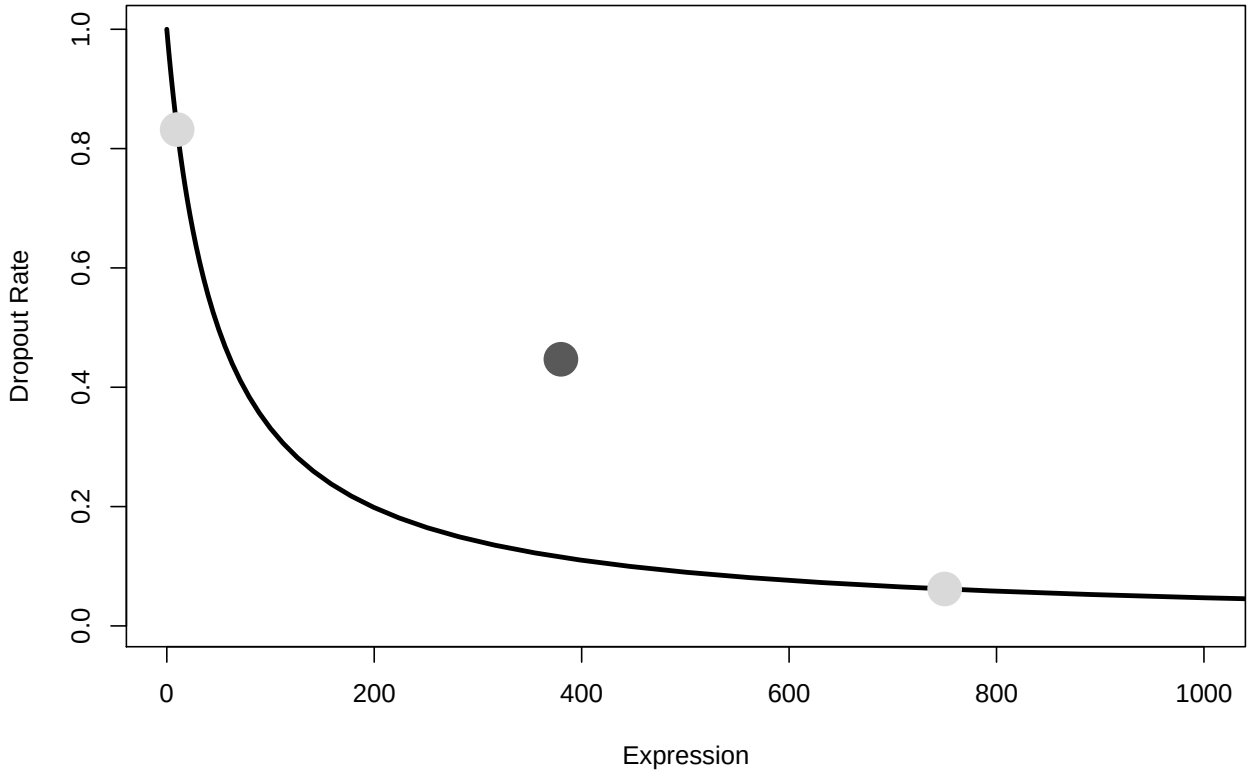
Now we will fit all three models to the normalized Usoskin data:

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



20.3 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.

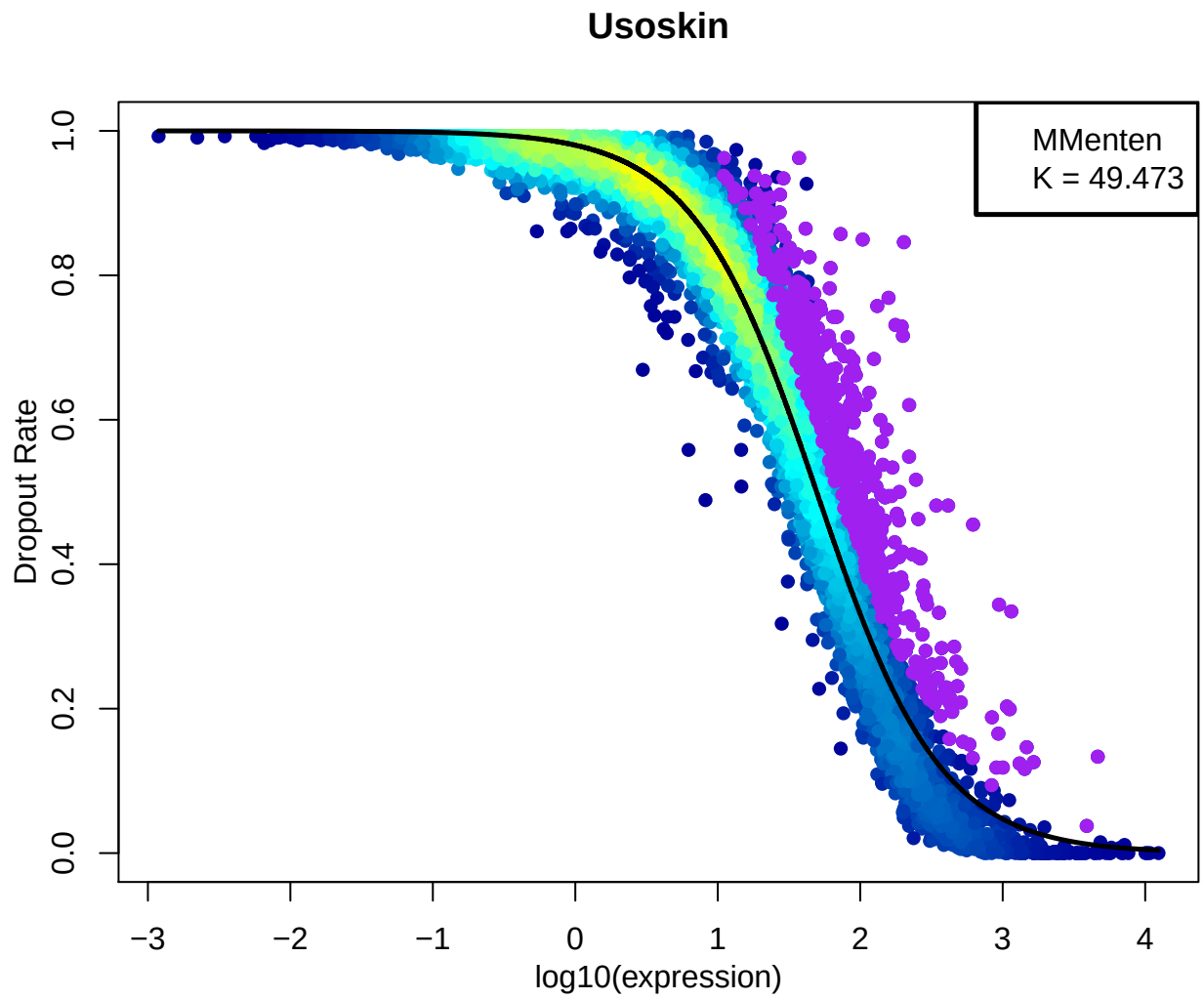


Note: add `log="x"` to the `plot` call above to see how this looks on the log scale, which is used in M3Drop figures.

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)
```

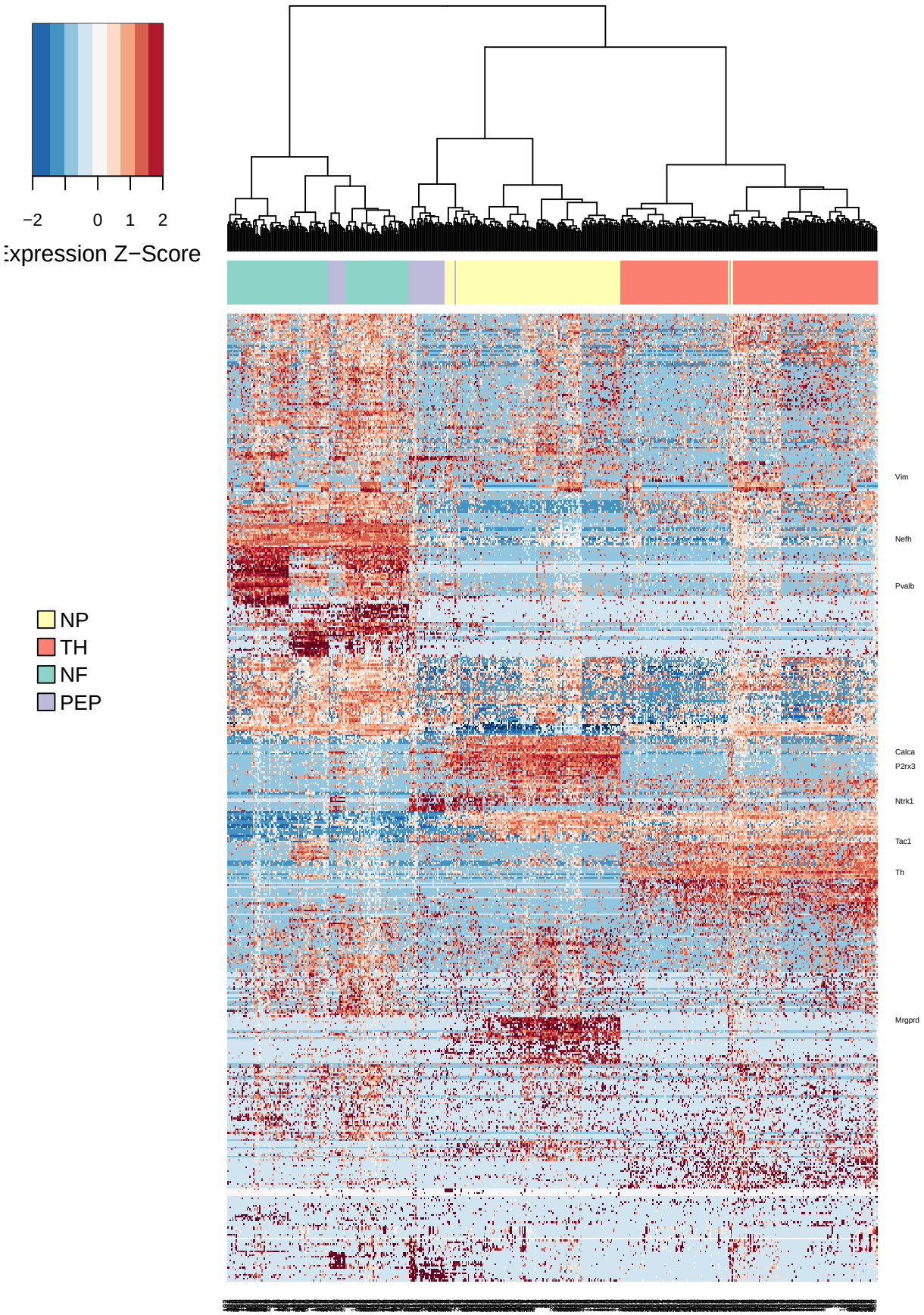
```
##             [,1]   [,2]   [,3]     [,4]   [,5]   [,6]    [,7]      [,8]
## uso_markers "Nefh" "Tac1" "Mrgprd" "Th"   "Vim"  "B2m"   "Col6a2" "Ntrk1"
##             "TRUE" "TRUE" "TRUE"   "TRUE" "TRUE" "FALSE" "FALSE"  "TRUE"
##             [,9]    [,10]   [,11]
## uso_markers "Calca" "P2rx3" "Pvalb"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

20.4 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

20.5 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))
```

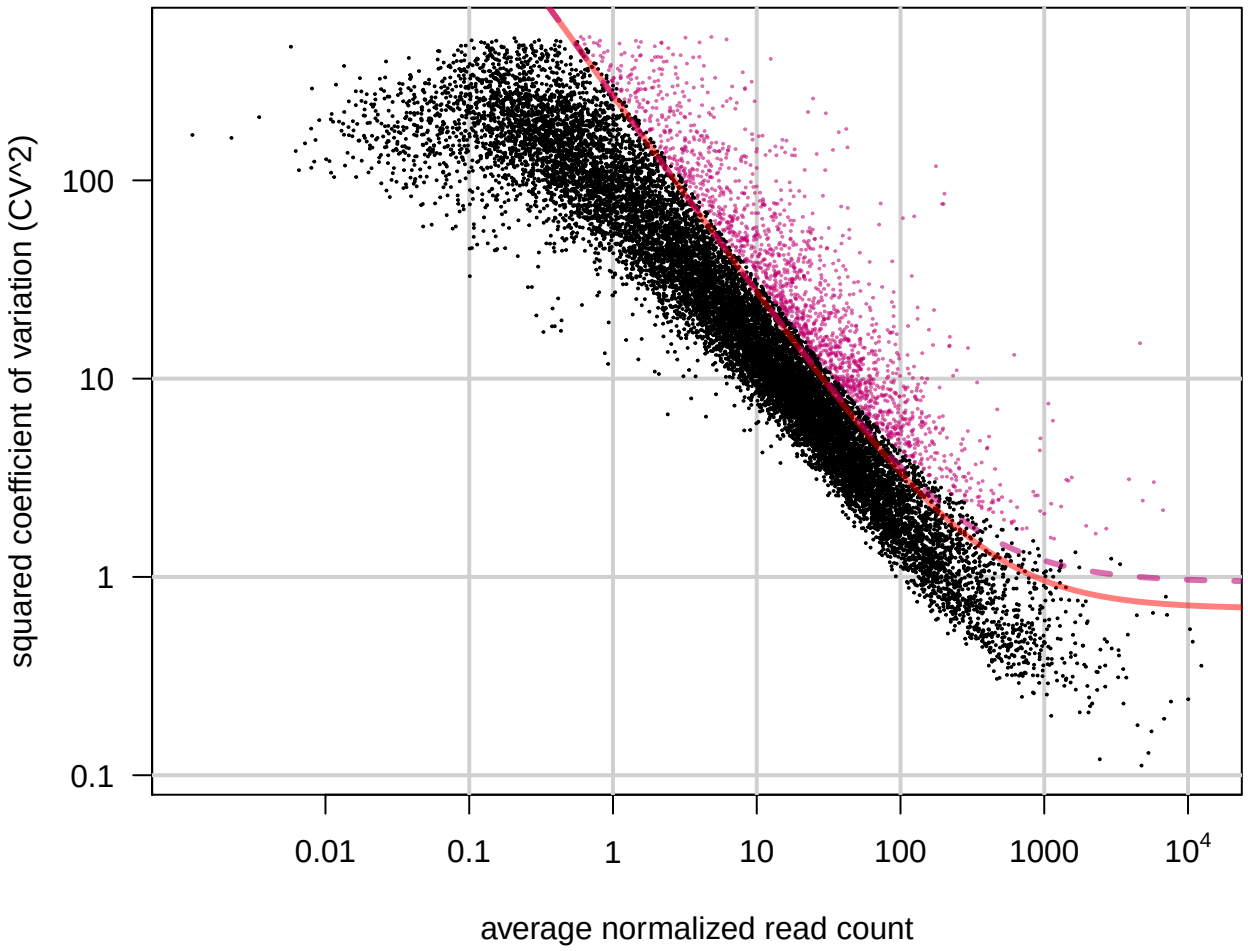
```
## [1] 2604
```

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 (here it is assumed that the technical noise can be estimated using ERCCs).

To use the Brennecke 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, and 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 to estimate the technical noise.

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.

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




```
length(Brennecke_HVG)
```

```
## [1] 1595
```

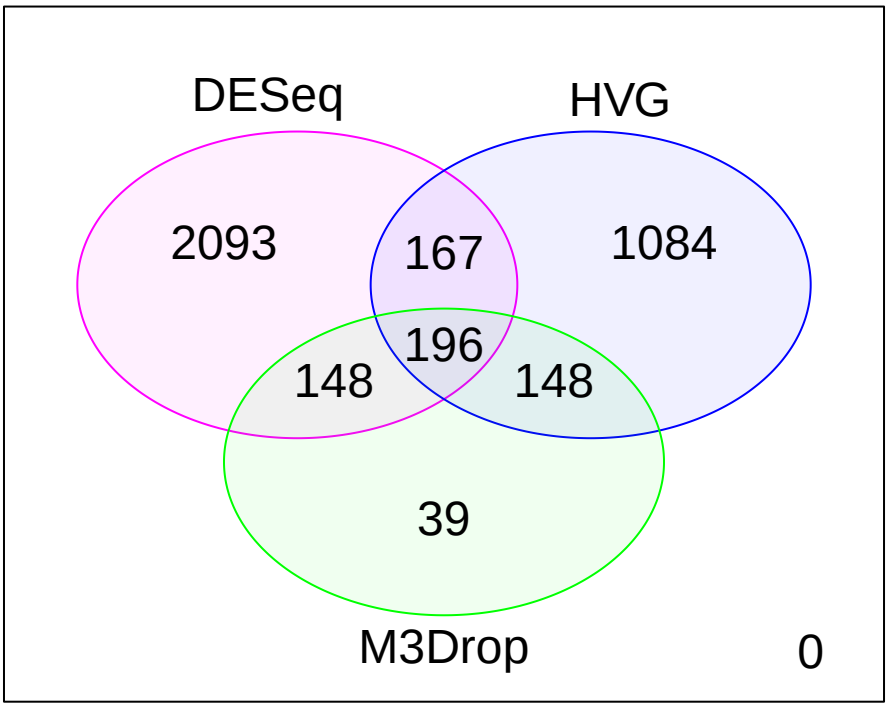
Exercise 3: Plot a heatmap of the expression of the HVGs and DESeq DE genes. Do they look differentially expressed?

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 21

Pseudotime analysis

```
library(TSCAN)
library(M3Drop)
library(monocle)
library(destiny)
library(SLICER)
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: 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 four different tools: Monocle, TSCAN, destiny and SLICER for ordering cells according to their pseudotime development. To illustrate the methods we will be using a dataset on mouse embryonic development (**?**). The dataset consists of 268 cells from 10 different time-points of early mouse development. In this case, there is no need for pseudotime alignment since the cell labels provide information about the development trajectory. Thus, the labels allow us to establish a ground truth so that we can evaluate and compare the different methods.

A recent review by Cannoodt et al provides a detailed summary of the various computational methods for trajectory inference from single-cell transcriptomics (**?**). They discuss several tools, but unfortunately for our purposes many of these tools do not have complete or well-maintained implementations, and/or are not implemented in R.

Cannoodt et al cover:

- SCUBA - Matlab implementation
- Wanderlust - Matlab (and requires registration to even download)
- Wishbone - Python
- SLICER - R, but package only available on Github
- SCOUP - C++ command line tool
- Waterfall - R, but one R script in supplement
- Mpath - R pkg, but available as tar.gz on Github; function documentation but no vignette/workflow

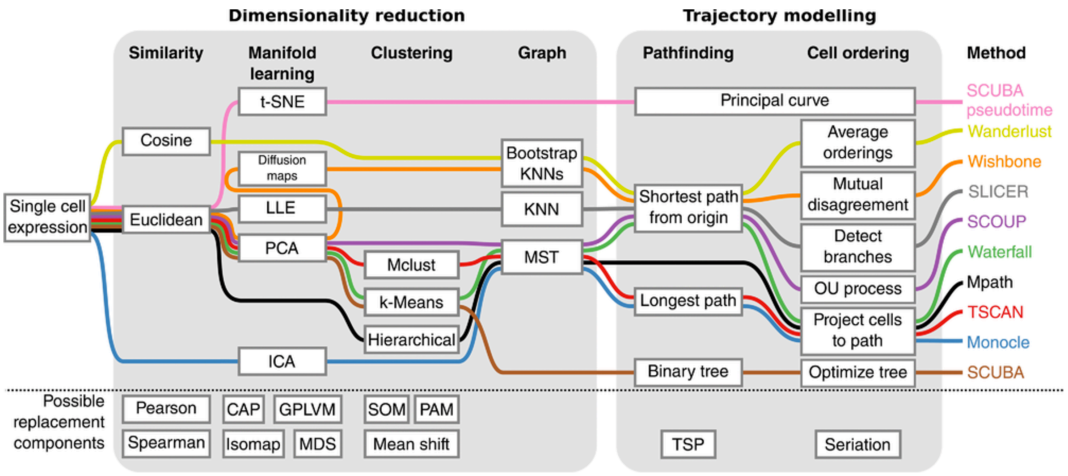


Figure 21.1: Descriptions of trajectory inference methods for single-cell transcriptomics data (Fig. 2 from Cannoodt et al, 2016).

- Monocle - Bioconductor package
- TSCAN - Bioconductor package

Unfortunately only two tools discussed (Monocle and TSCAN) meet the gold standard of open-source software hosted in a reputable repository.

The following figures from the paper summarise some of the features of the various tools.

21.1 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. 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)
```

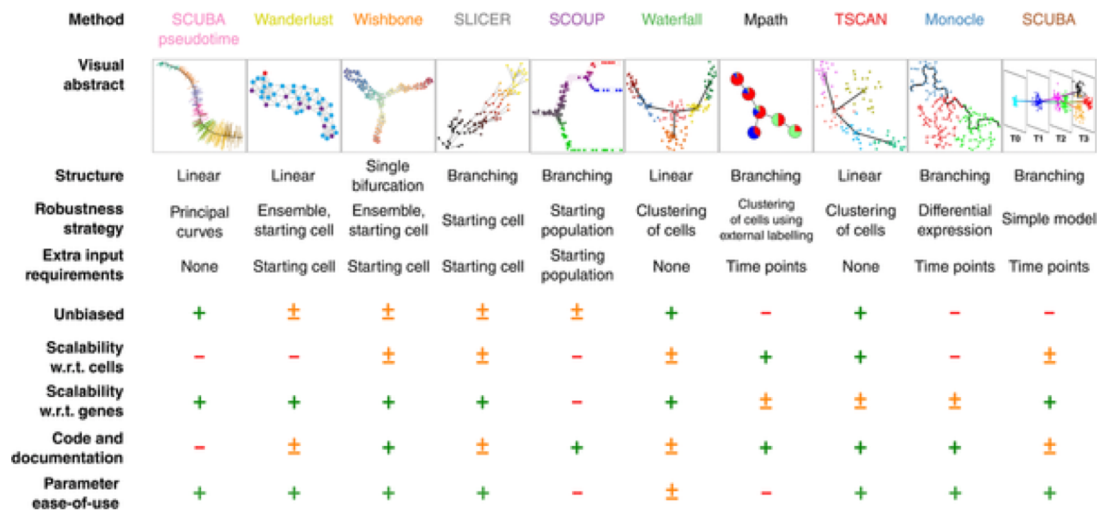
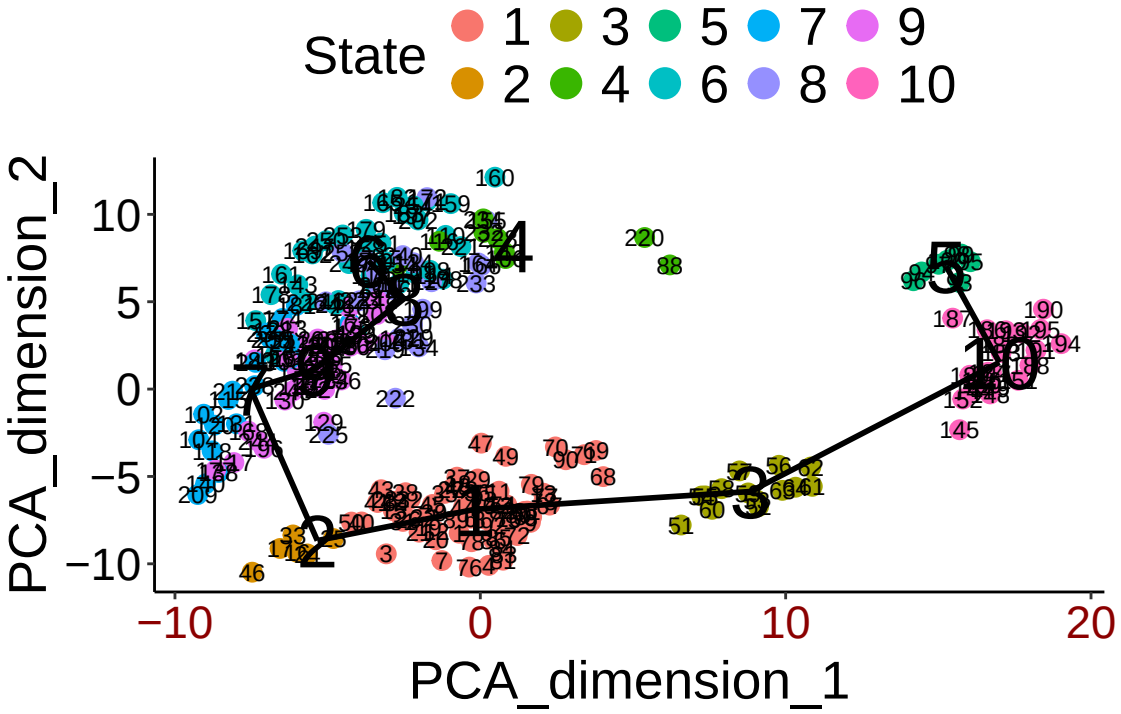


Figure 21.2: Characterization of trajectory inference methods for single-cell transcriptomics data (Fig. 3 from Cannoodt et al, 2016).




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

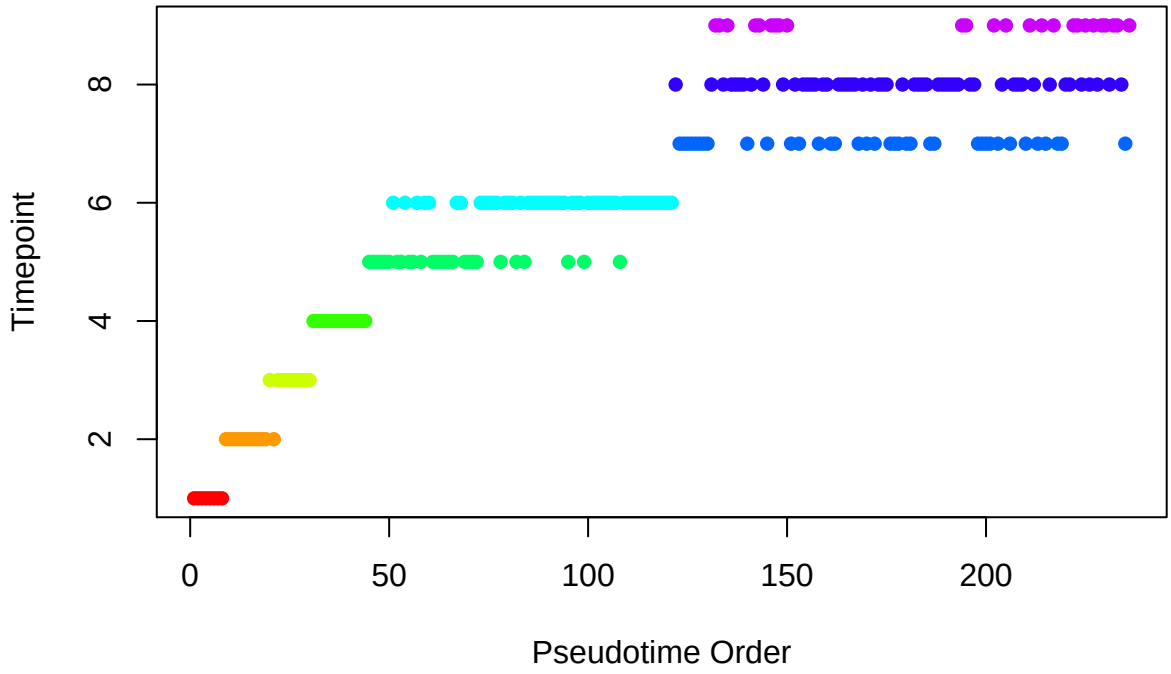
We can also examine which timepoints have been assigned to each state:

```
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
)
```



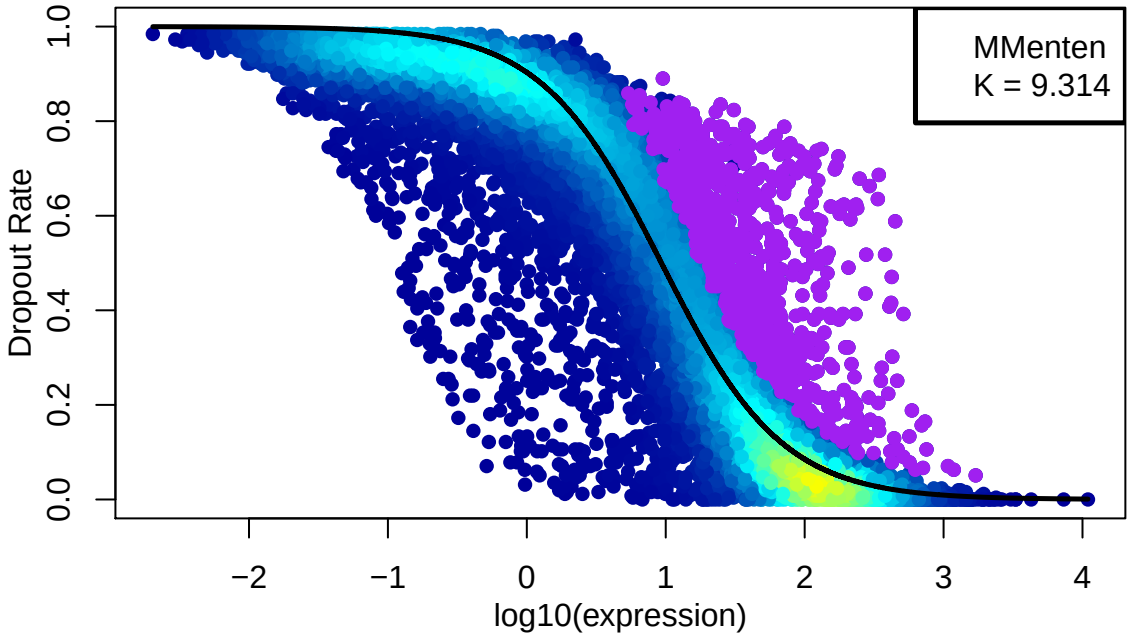
Exercise 1 Compare results for different numbers of clusters (`clusternum`).

21.2 monocle

Monocle skips the clustering stage of TSCAN and directly builds a minimum spanning tree on a reduced dimension representation of the cells 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 these. Each of the resulting forked paths is defined as a separate cell state.

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

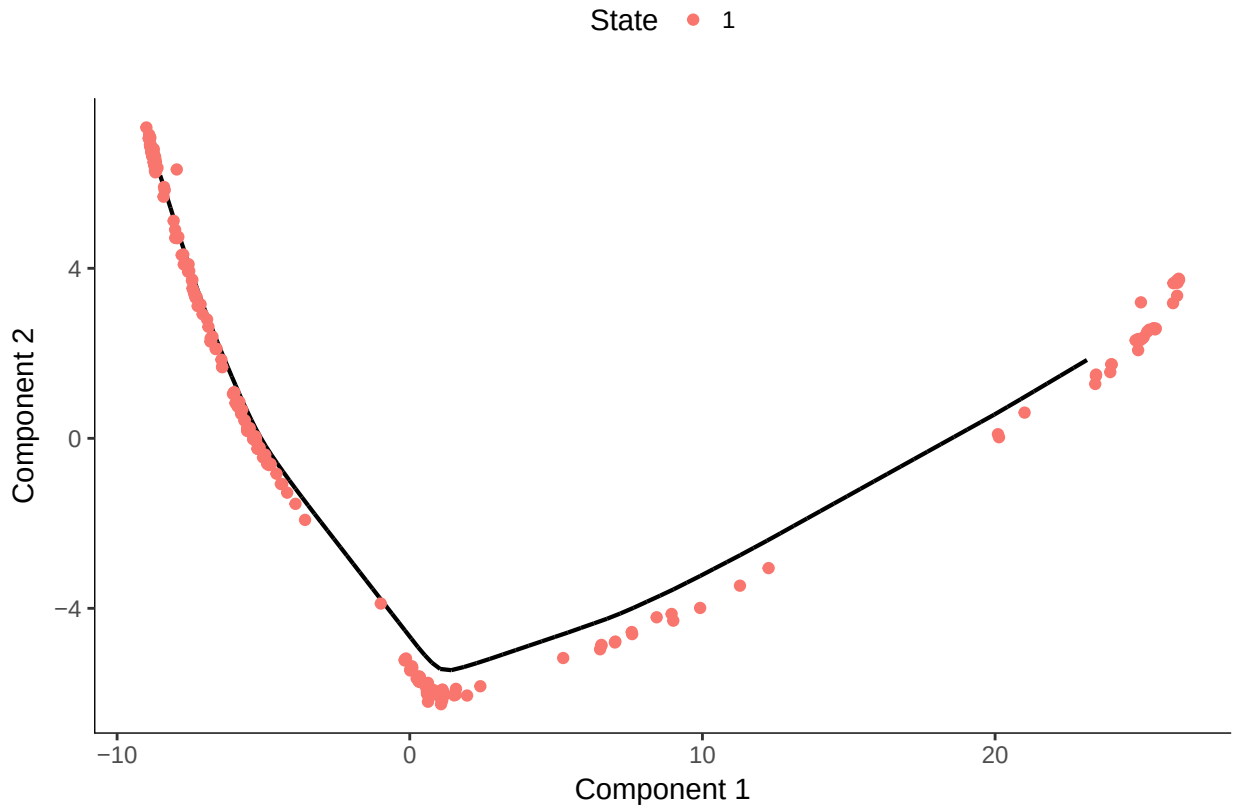
```
m3dGenes <- as.character(
    M3Drop::M3DropDifferentialExpression(deng)$Gene
)
```




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

Now run monocle:

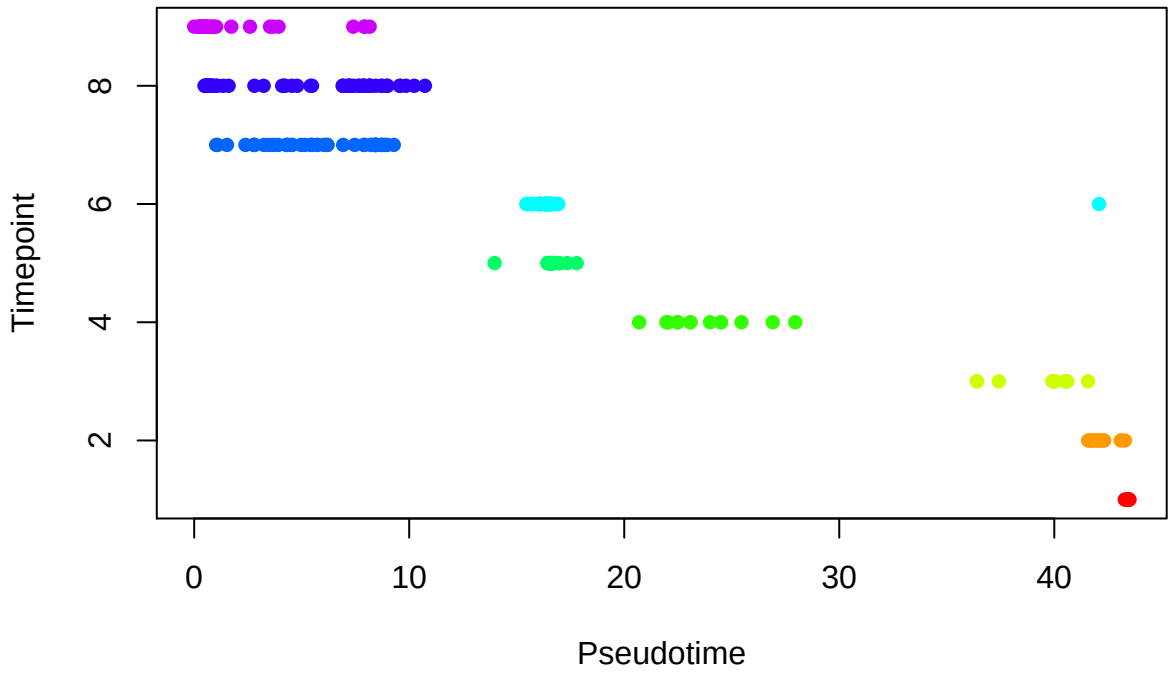
```
colnames(d) <- 1:ncol(d)
geneNames <- rownames(d)
rownames(d) <- 1:nrow(d)
pd <- data.frame(timepoint = cellLabels)
pd <- new("AnnotatedDataFrame", data=pd)
fd <- data.frame(gene_short_name = geneNames)
fd <- new("AnnotatedDataFrame", data=fd)

dCellData <- newCellDataSet(d, phenoData = pd, featureData = fd, expressionFamily = tobit())
dCellData <- setOrderingFilter(dCellData, which(geneNames %in% m3dGenes))
dCellData <- estimateSizeFactors(dCellData)
dCellDataSet <- reduceDimension(dCellData, pseudo_expr = 1)
dCellDataSet <- orderCells(dCellDataSet, reverse = TRUE)
plot_cell_trajectory(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
)
```



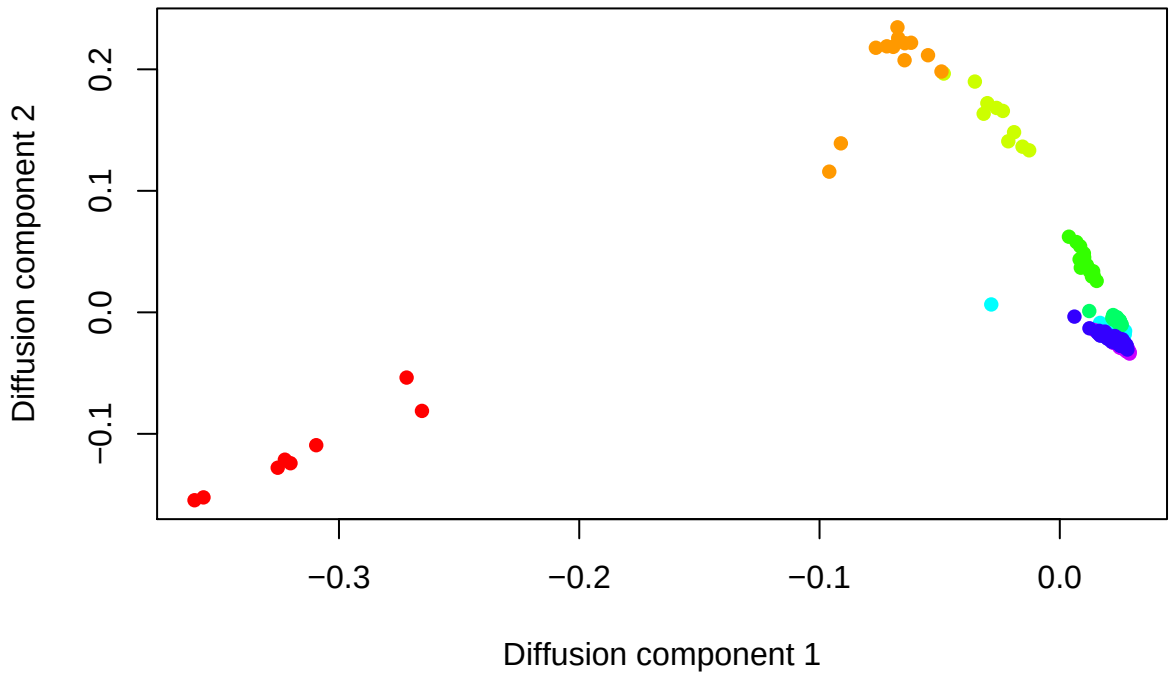
21.3 Diffusion maps

Diffusion maps were introduced by Ronald Coifman and Stephane Lafon, and 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(deng + 1)))
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?

21.4 SLICER

The SLICER method is an algorithm for constructing trajectories that describe gene expression changes during a sequential biological process, just as Monocle and TSCAN are. SLICER is designed to capture highly nonlinear gene expression changes, automatically select genes related to the process, and detect multiple branch and loop features in the trajectory (**?**). The SLICER R package is available from its GitHub repository and can be installed from there using the `devtools` package.

We use the `select_genes` function in SLICER to automatically select the genes to use in builing the cell trajectory. The function uses "neighbourhood variance" to identify genes that vary smoothly, rather than fluctuating randomly, across the set of cells. Following this, we determine which value of "k" (number of nearest neighbours) yields an embedding that most resembles a trajectory. Then we estimate the locally linear embedding of the cells.

```
slicer_genes <- select_genes(t(deng))
k <- select_k(t(deng[slicer_genes,]), kmin = 30, kmax=60)
```

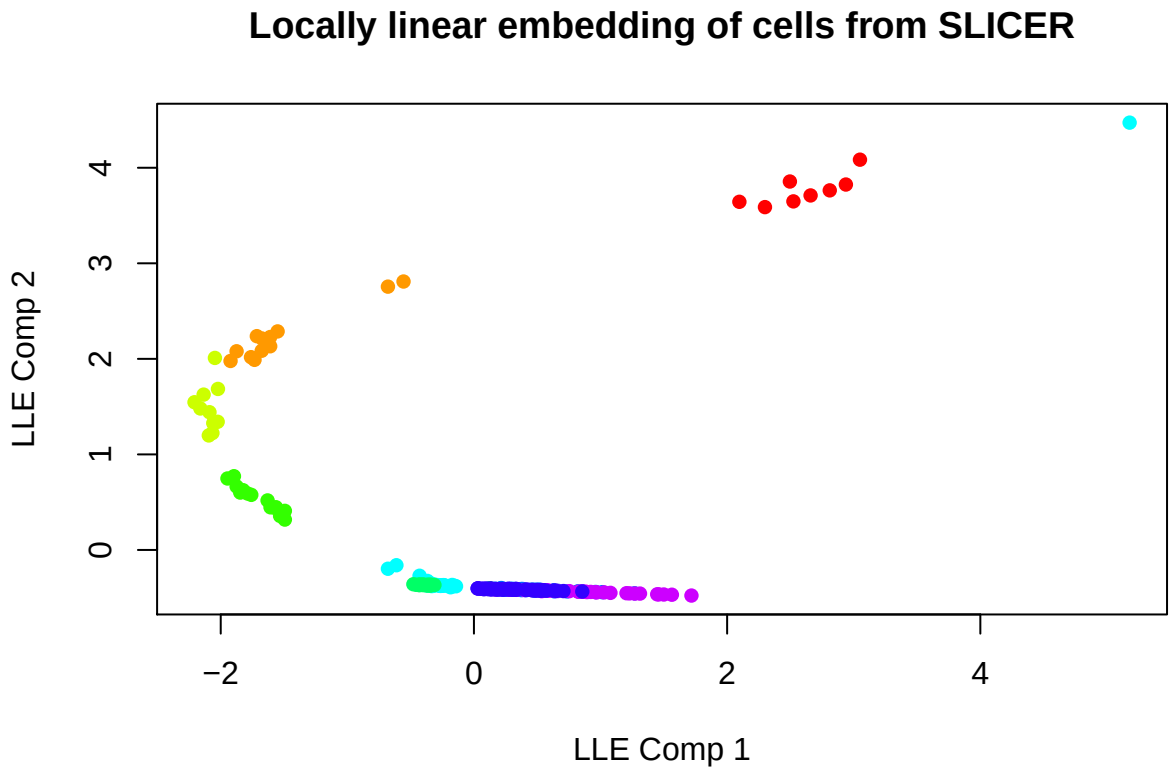
```
## finding neighbours
## calculating weights
## computing coordinates
## finding neighbours
## calculating weights
## computing coordinates
```

```
## finding neighbours
## calculating weights
## computing coordinates
## finding neighbours
## calculating weights
## computing coordinates
## finding neighbours
## calculating weights
## computing coordinates
## finding neighbours
## calculating weights
## computing coordinates
## finding neighbours
## calculating weights
## computing coordinates
```

```
slicer_traj_lle <- lle(t(deng[slicer_genes,]), m = 2, k)$Y
```

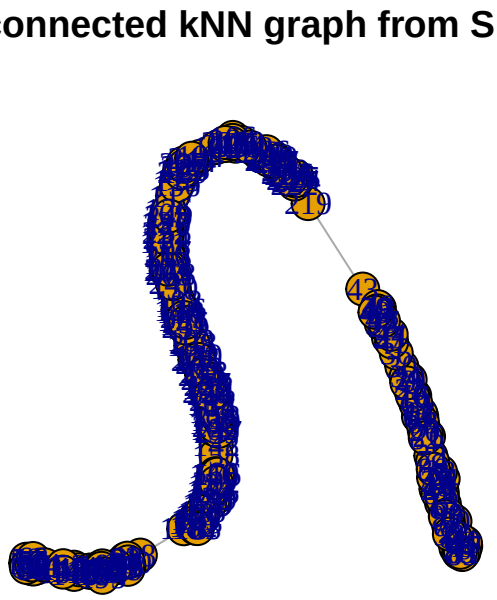
```
## finding neighbours
## calculating weights
## computing coordinates
```

```
plot(slicer_traj_lle, xlab = "LLE Comp 1", ylab = "LLE Comp 2",
     main = "Locally linear embedding of cells from SLICER",
     col=colours[tmp], pch=16)
```



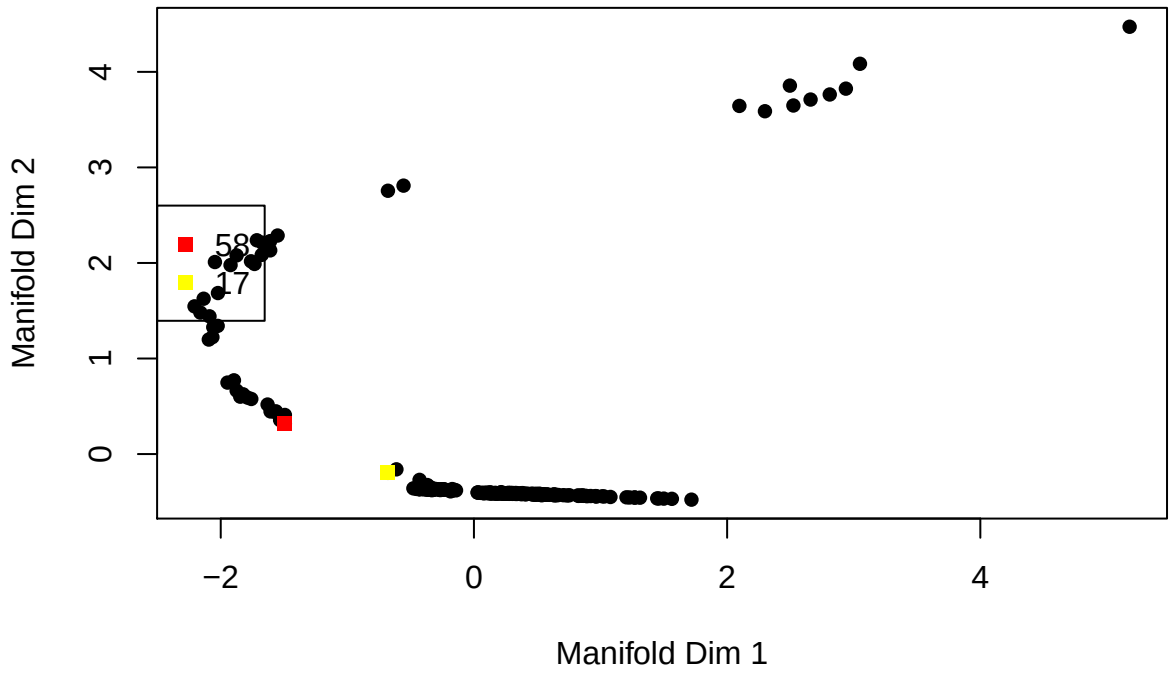
With the locally linear embedding computed we can construct a k-nearest neighbour graph that is fully connected. This plot displays a (yellow) circle for each cell, with the cell ID number overlaid in blue. Here we show the graph computed using 10 nearest neighbours. Here, SLICER appears to detect one major trajectory with one branch.

```
slicer_traj_graph <- conn_knn_graph(slicer_traj_lle, 10)
plot(slicer_traj_graph, main = "Fully connected kNN graph from SLICER")
```



From this graph we can identify "extreme" cells that are candidates for start/end cells in the trajectory.

```
ends <- find_extreme_cells(slicer_traj_graph, slicer_traj_lle)
```




```
start <- ends[1]
```

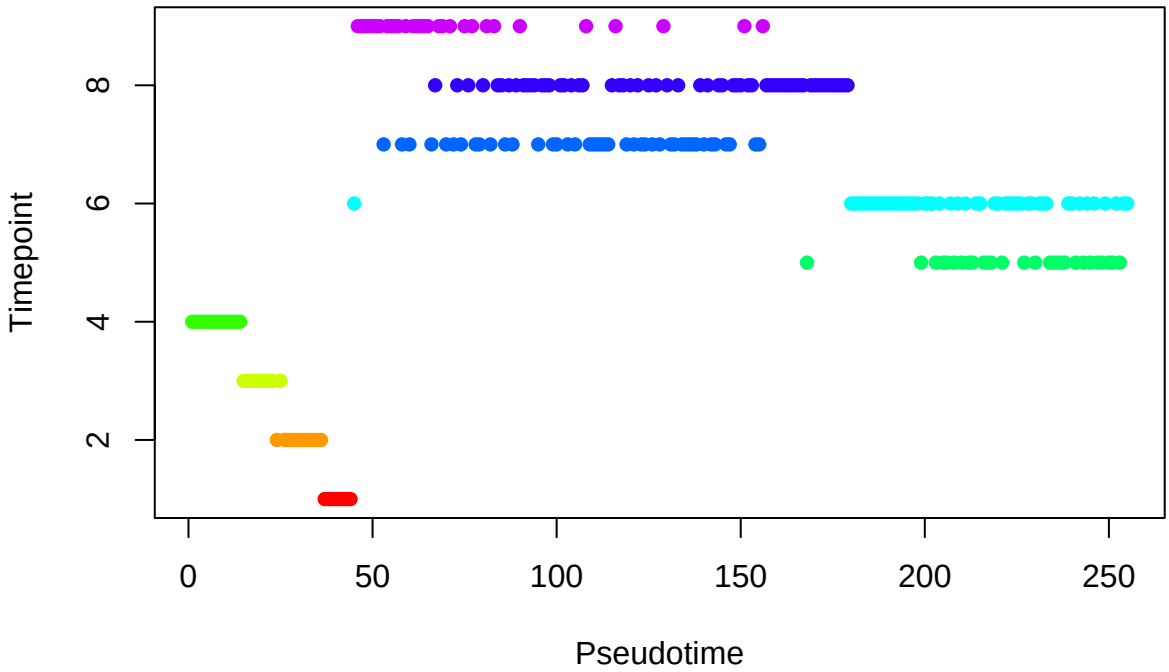
Having defined a start cell we can order the cells in the estimated pseudotime.

```
pseudotime_order_slicer <- cell_order(slicer_traj_graph, start)
branches <- assign_branches(slicer_traj_graph, start)

pseudotime_slicer <-
```

```
    data.frame(
        Timepoint = cellLabels,
        pseudotime = NA,
        State = branches
    )
pseudotime_slicer$pseudotime[pseudotime_order_slicer] <-
    1:length(pseudotime_order_slicer)
```

We can again compare the inferred pseudotime to the known sampling timepoints. SLICER does not provide a pseudotime value per se, just an ordering of cells.

```
slicer_time_point <- factor(
     pseudotime_slicer$Timepoint,
     levels = c("early2cell", "mid2cell", "late2cell", "4cell", "8cell",
                   "16cell", "earlyblast", "midblast", "lateblast")
)

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



Like the previous method, SLICER here provides a good ordering for the early time points and struggles for later time points.

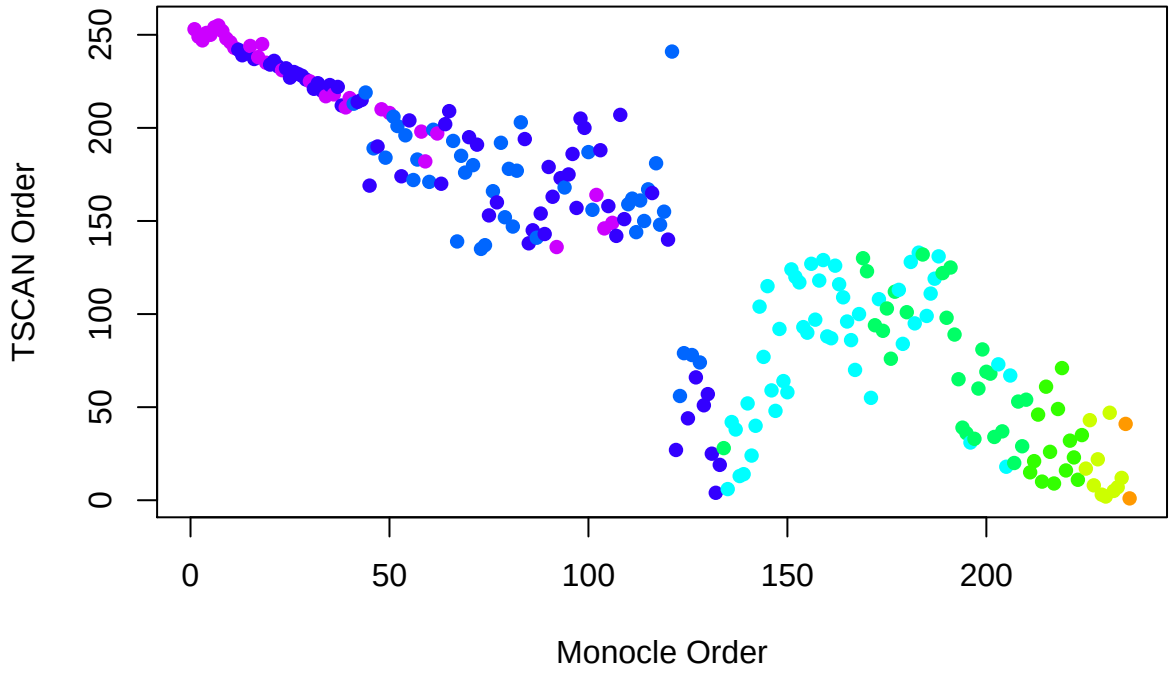
Exercise 4 How do the results change for different k? (e.g. k = 5) What about changing the number of nearest neighbours in the call to `conn_knn_graph`?

Exercise 5 How does the ordering change if you use a different set of genes from those chosen by SLICER (e.g. the genes identified by M3Drop)?

21.5 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 6: Compare destiny and SLICER to TSCAN and Monocle.

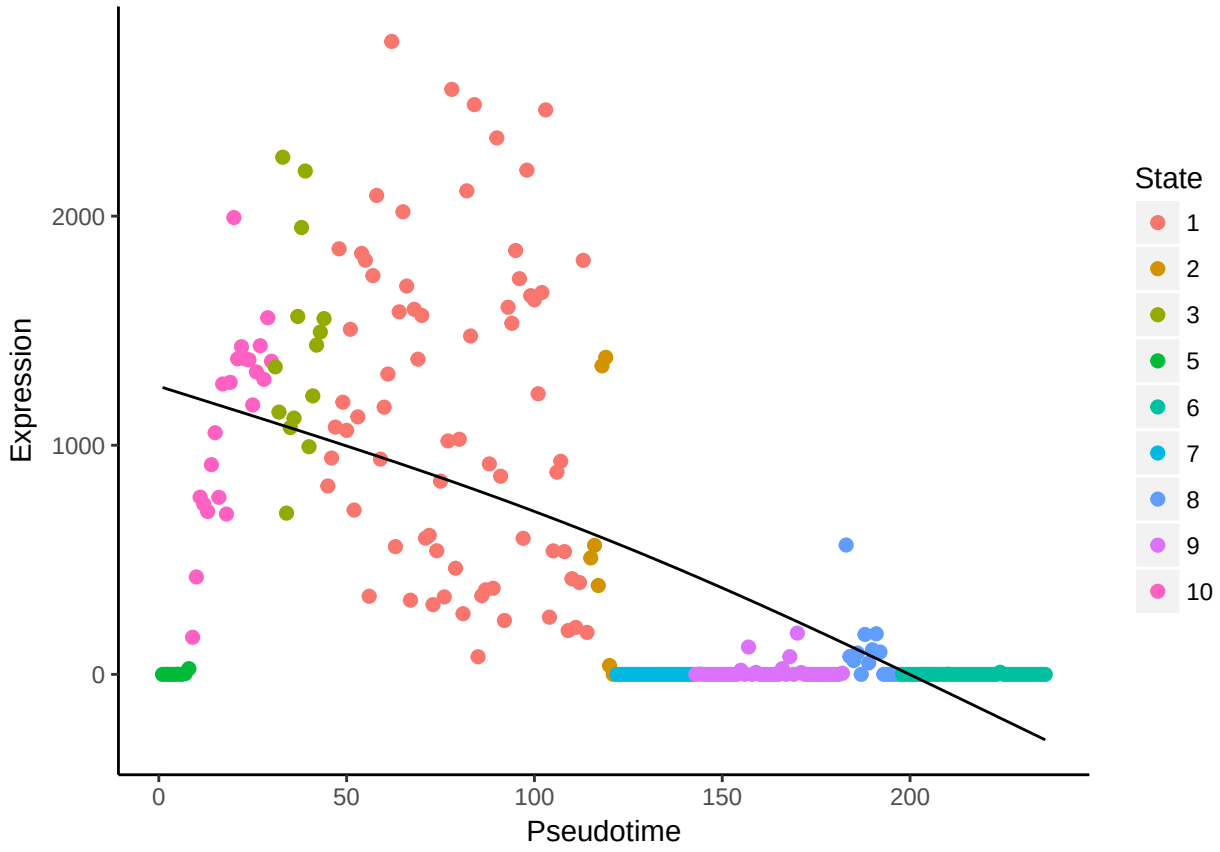
21.6 Expression of genes through time

Each package also enables the visualization of expression through pseudotime. Following individual genes is very helpful for identifying genes that play an important role in the differentiation process. We illustrate the procedure using the Obox6 gene which is known to be important during early development.

TSCAN

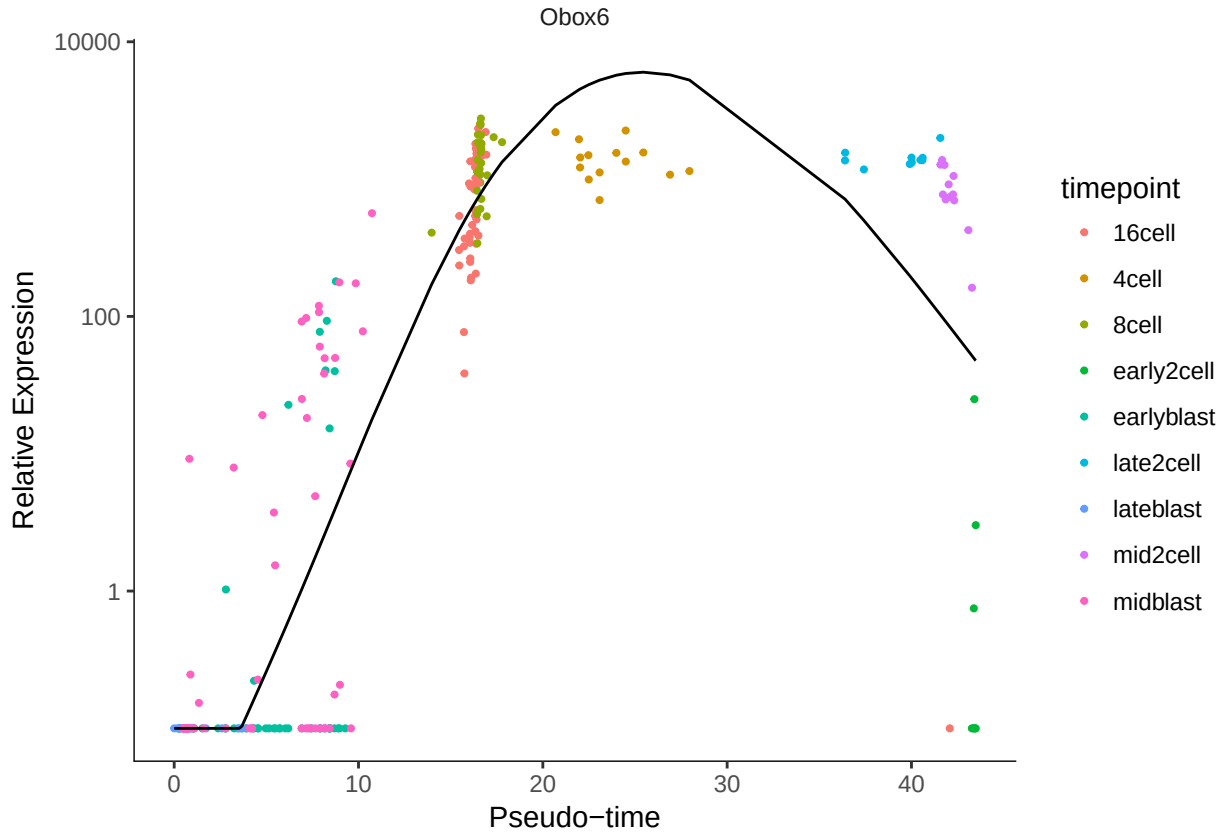
```
colnames(deng) <- 1:ncol(deng)
TSCAN::singlegeneplot(
    deng[rownames(deng) == "Obox6", ],
```

```
    dengorderTSCAN
)
```



Monocle

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



Of course, pseudotime values computed with any method can be added to the `pData` slot of an `SCESet` object. Having done that, the full plotting capabilities of the `scater` package can be used to investigate relationships between gene expression, cell populations and pseudotime. This would be particularly useful for the SLICER results, as SLICER does not provide plotting functions.

Exercise 7: 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 22

Differential Expression (DE) analysis

22.1 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.

22.2 Single cell RNA-seq

In contrast to bulk RNA-seq, in scRNA-seq we usually do not have a defined set of experimental conditions. Instead, as was shown in a previous chapter (**??**) we can identify the cell groups by using an unsupervised clustering approach. Once the groups have been identified one can find differentially expressed genes either by comparing the differences 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 pairwise comparisons.

22.3 Differences in Distribution

Unlike bulk RNA-seq, we generally have a large number of samples (i.e. cells) for each group we are comparing in single-cell experiments. Thus we can take advantage of the whole distribution of expression values in each group to identify differences between groups rather than only comparing estimates of mean-expression as is standard for bulk RNASeq.

There are two main approaches to comparing distributions. Firstly, we can use existing statistical models/distributions and fit the same type of model to the expression in each group then test for differences in the parameters for each model, or test whether the model fits better if a particular paramter is allowed to be different according to group. For instance in Chapter **??** we used edgeR to test whether allowing mean expression to be different in different batches significantly improved the fit of a negative binomial model of the data.

Alternatively, we can use a non-parametric test which does not assume expression values follow any particular distribution, e.g. the Kolmogorov-Smirnov test (KS-test). Non-parametric tests generally convert observed

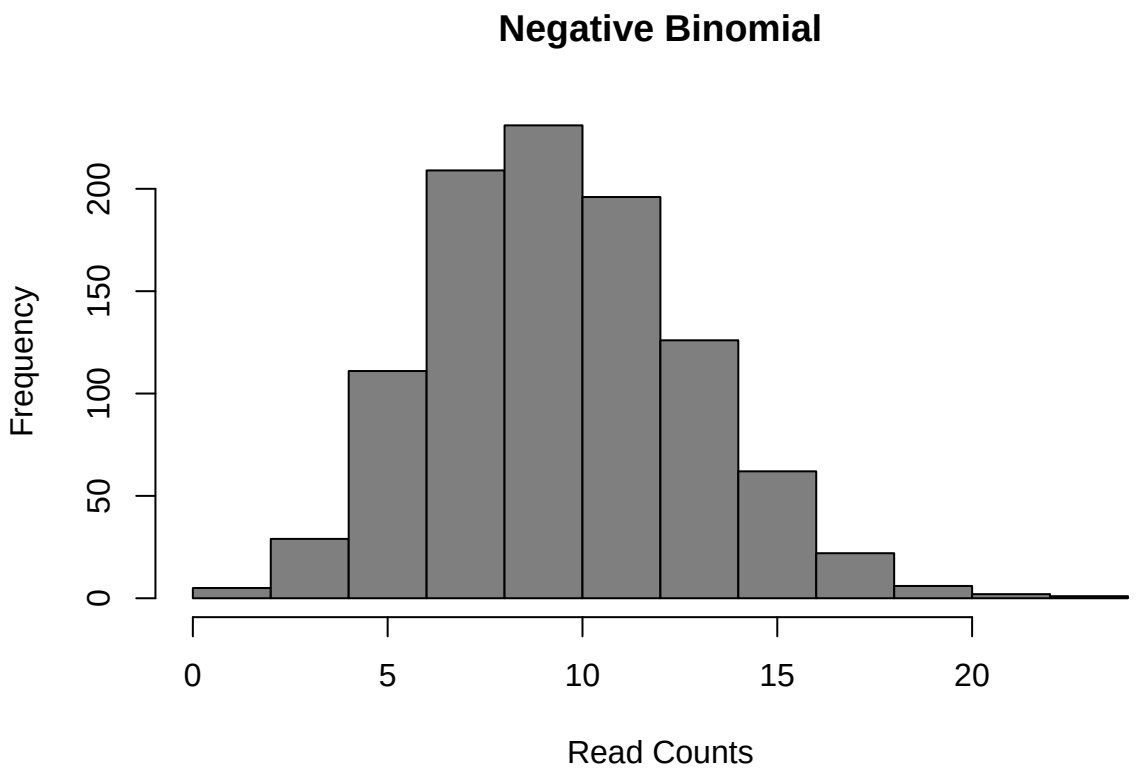


Figure 22.1: Negative Binomial distribution of read counts for a single gene across 1000 cells

expression values to ranks and test whether the distribution of ranks for one group are signficantly different from the distribution of ranks for the other group. However, some non-parametric methods fail in the presence of a large number of tied values, such as the case for dropouts (zeros) in single-cell RNA-seq expression data. Moreover, if the conditions for a parametric test hold, then it will typically be more powerful than a non-parametric test.

22.4 Models of single-cell RNASeq data

The most common model of RNASeq data is the negative binomial model:

```
set.seed(1)
hist(rnbinom(1000, mu=10, size=100), col="grey50", xlab="Read Counts", main="Negative Binomial")
```

Mean: $\mu = mu$

Variance: $\sigma^2 = mu + mu^2/size$

It is parameterized by the mean expression (mu) and the dispersion (size), which is inversely related to the variance. The negative binomial model fits bulk RNA-seq data very well and it is used for most statistical methods designed for such data. In addition, it has been show to fit the distribution of molecule counts obtained from data tagged by unique molecular identifiers (UMIs) quite well (Grun et al. 2014, Islam et al. 2011).

However, a raw negative binomial model does not fit full-length transcript data as well due to the high dropout rates relative to the non-zero read counts. For this type of data a variety of zero-inflated negative binomial models have been proposed (e.g. MAST, SCDE).

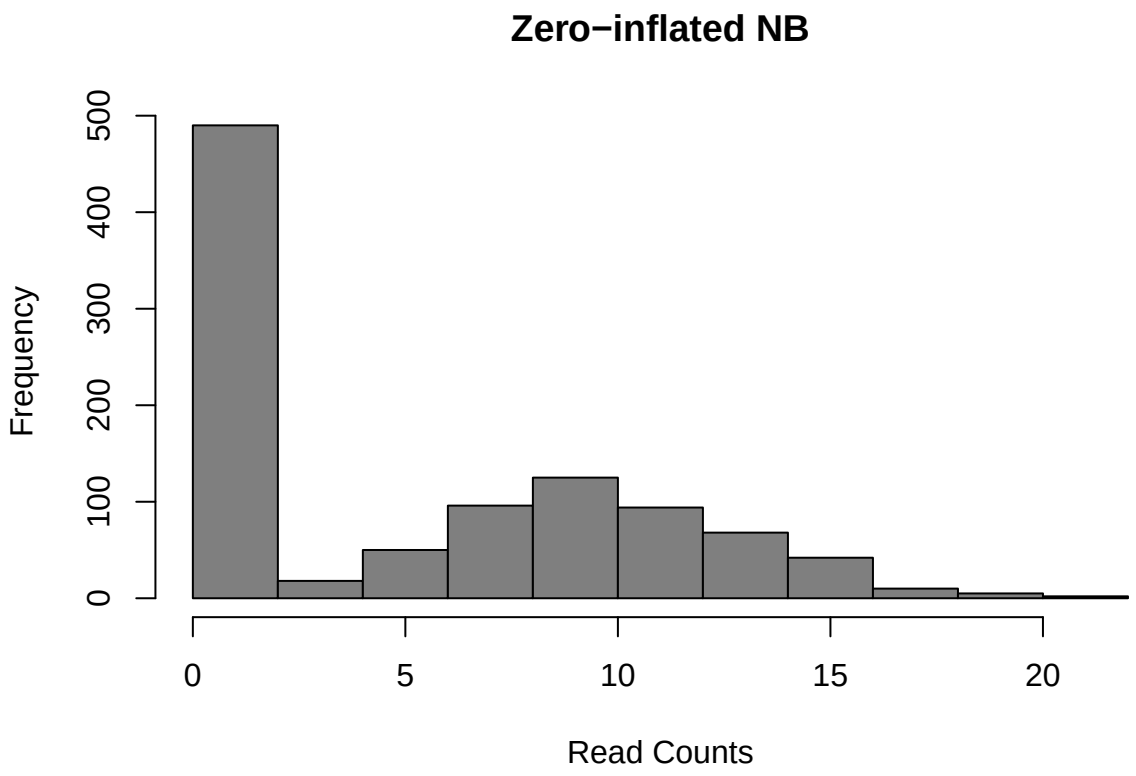


Figure 22.2: Zero-inflated Negative Binomial distribution

```
d = 0.5;
counts <- rnbinom(1000, mu=10, size=100);
counts[runif(1000) < d] = 0;
hist(counts, col="grey50", xlab="Read Counts", main="Zero-inflated NB");
```

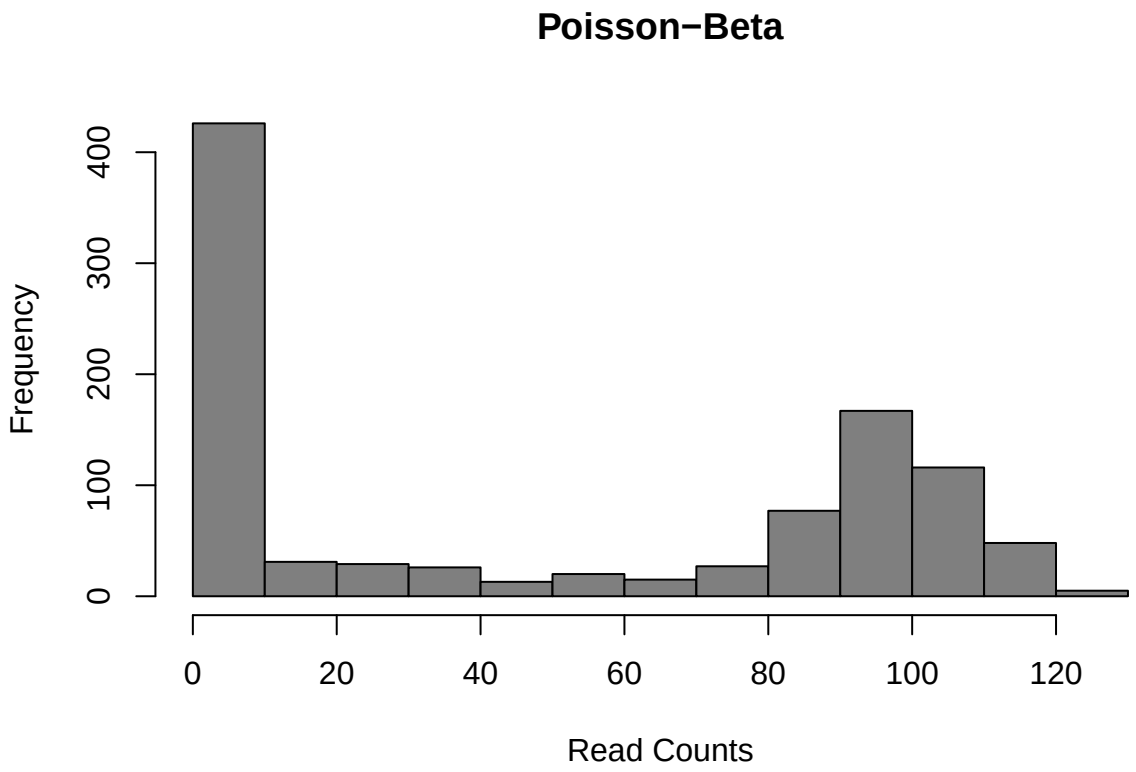
Mean: $\mu = mu \cdot (1 - d)$

Variance: $\sigma^2 = \mu \cdot (1 - d) \cdot (1 + d \cdot \mu + \mu/size)$

These models introduce a new parameter d, for the dropout rate, to the negative binomial model. As we saw in Chapter 19, the dropout rate of a gene is strongly correlated with the mean expression of the gene. Different zero-inflated negative binomial models use different relationships between mu and d and some may fit μ and d to the expression of each gene independently.

Finally, several methods use a Poisson-Beta distribution which is based on a mechanistic model of transcriptional bursting. There is strong experimental support for this model (Kim and Marioni, 2013) and it provides a good fit to scRNA-seq data but it is less easy to use than the negative-binomial models and much less existing methods upon which to build than the negative binomial model.

```
a = 0.1
b = 0.1
g = 100
lambdas = rbeta(1000, a, b)
counts = sapply(g*lambdas, function(l) {rpois(1, lambda=l)})
hist(counts, col="grey50", xlab="Read Counts", main="Poisson-Beta")
```



Mean: $\mu = g \cdot a/(a+b)$

Variance: $\sigma^2 = g^2 \cdot a \cdot b/((a+b+1) \cdot (a+b)^2)$

This model uses three parameters: a the rate of activation of transcription; b the rate of inhibition of transcription; and g the rate of transcript production while transcription is active at the locus. Differential expression methods may test each of the parameters for differences across groups or only one (often g).

All of these models may be further expanded to explicitly account for other sources of gene expression differences such as batch-effect or library depth depending on the particular DE algorithm.

Exercise: Vary the parameters of each distribution to explore how they affect the distribution of gene expression. How similar are the Poisson-Beta and Negative Binomial models?

Chapter 23

DE in a real dataset

```
library(scRNA.seq.funcs)
library(edgeR)
library(monocle)
library(MAST)
library(ROCR)
set.seed(1)
```

23.1 Introduction

To test different single-cell differential expression methods we will be using the Blischak dataset from Chapters 7-17. For this experiment bulk RNA-seq data for each cell-line was generated in addition to single-cell data. We will use the differentially expressed genes identified using standard methods on the respective bulk data as the ground truth for evaluating the accuracy of each single-cell method. To save time we have pre-computed these for you, run the commands below to load these data.

```
DE <- read.table("tung/TPs.txt")
notDE <- read.table("tung/TNs.txt")
GroundTruth <- list(DE=as.character(unlist(DE)), notDE=as.character(unlist(notDE)))
```

This ground truth has been produce for the comparison of individual NA19101 to NA19239. Now load the respective single-cell data:

```
molecules <- read.table("tung/molecules.txt", sep = "\t")
anno <- read.table("tung/annotation.txt", sep = "\t", header = TRUE)
keep <- anno[,1] == "NA19101" | anno[,1] == "NA19239"
data <- molecules[,keep]
group <- anno[keep,1]
batch <- anno[keep,4]
# remove genes that aren't expressed in at least 6 cells
gkeep <- rowSums(data > 0) > 5;
counts <- data[gkeep,]
# Library size normalization
lib_size = colSums(counts)
norm <- t(t(counts)/lib_size*median(lib_size))
# Variant of CPM for datasets with library sizes of fewer than 1 mil molecules
```

Now we will compare various single-cell DE methods. Note that we will only be running methods which are

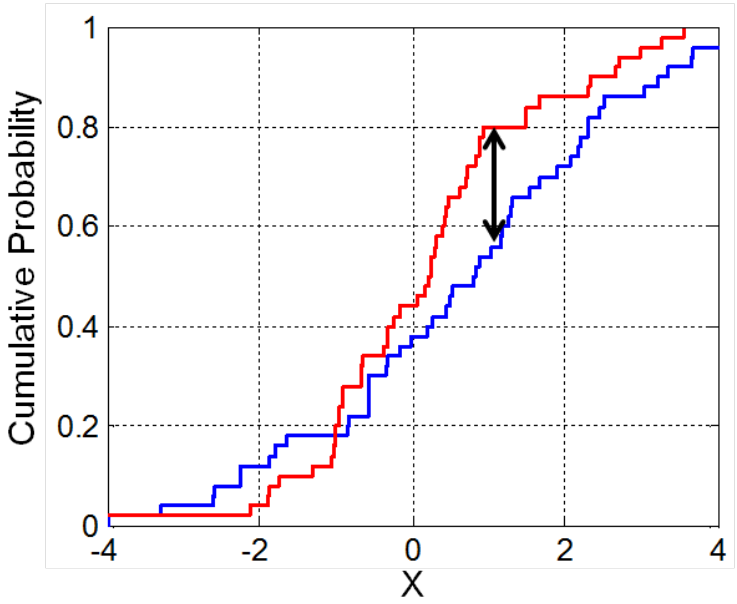
available as R-packages and run relatively quickly.

23.2 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 individuals.

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 experession 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). Another issue with the KS-test is that it can be very sensitive for large sample sizes and thus it may end up as significant even though the magnitude of the difference is very small.

\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}

Now run the test:

```
pVals <- apply(norm, 1, function(x) {
        ks.test(x[group =="NA19101"],
                x[group=="NA19239"])$p.value
         })
# multiple testing correction
pVals <- p.adjust(pVals, method = "fdr")
```

This code "applies" the function to each row (specified by 1) of the expression matrix, data. In the function we are returning just the p.value from the ks.test output. We can now consider how many of the ground truth positive and negative DE genes are detected by the KS-test:

23.2.1 Evaluating Accuracy

```
sigDE <- names(pVals)[pVals < 0.05]
length(sigDE)
```

```
## [1] 5095
```

```
# Number of KS-DE genes
sum(GroundTruth$DE %in% sigDE)
```

```
## [1] 792
```

```
# Number of KS-DE genes that are true DE genes
sum(GroundTruth$notDE %in% sigDE)
```

```
## [1] 3190
```

```
# Number of KS-DE genes that are truly not-DE
```

As you can see many more of our ground truth negative genes were identified as DE by the KS-test (false positives) than ground truth positive genes (true positives), however this may be due to the larger number of notDE genes thus we typically normalize these counts as the True positive rate (TPR), TP/(TP + FN), and False positive rate (FPR), FP/(FP+TP).

```
tp <- sum(GroundTruth$DE %in% sigDE)
fp <- sum(GroundTruth$notDE %in% sigDE)
tn <- sum(GroundTruth$notDE %in% names(pVals)[pVals >= 0.05])
fn <- sum(GroundTruth$DE %in% names(pVals)[pVals >= 0.05])
tpr <- tp/(tp + fn)
fpr <- fp/(fp + tn)
cat(c(tpr, fpr))
```

```
## 0.7346939 0.2944706
```

Now we can see the TPR is much higher than the FPR indicating the KS test is identifying DE genes.

So far we've only evaluated the performance at a single significance threshold. Often it is informative to vary the threshold and evaluate performance across a range of values. This is then plotted as a receiver-operating-characteristic curve (ROC) and a general accuracy statistic can be calculated as the area under this curve (AUC). We will use the ROCR package to facilitate this plotting.

```
# Only consider genes for which we know the ground truth
pVals <- pVals[names(pVals) %in% GroundTruth$DE |
               names(pVals) %in% GroundTruth$notDE]
truth <- rep(1, times = length(pVals));
truth[names(pVals) %in% GroundTruth$DE] = 0;
pred <- ROCR::prediction(pVals, truth)
perf <- ROCR::performance(pred, "tpr", "fpr")
ROCR::plot(perf)
```

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

```
## [1] 0.7954796
```

Finally to facilitate the comparisons of other DE methods let's put this code into a function so we don't need to repeat it:

```
DE_Quality_AUC <- function(pVals) {
        pVals <- pVals[names(pVals) %in% GroundTruth$DE |
```

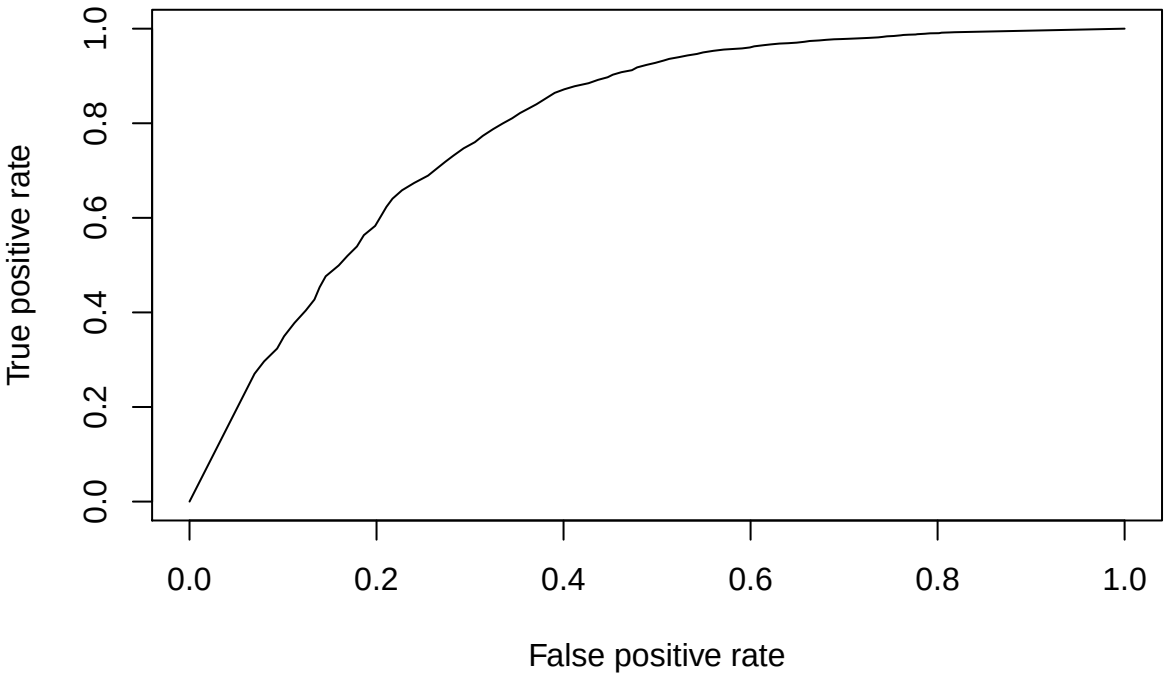


Figure 23.1: ROC curve for KS-test.

```
                        names(pVals) %in% GroundTruth$notDE]
        truth <- rep(1, times = length(pVals));
        truth[names(pVals) %in% GroundTruth$DE] = 0;
        pred <- ROCR::prediction(pVals, truth)
        perf <- ROCR::performance(pred, "tpr", "fpr")
        ROCR::plot(perf)
        aucObj <- ROCR::performance(pred, "auc")
        return(aucObj@y.values[[1]])
}
```

23.3 Wilcox/Mann-Whitney-U Test

The Wilcox-rank-sum test is another non-parametric test, but tests specifically if values in one group are greater/less than the values in the other group. Thus it is often considered a test for difference in median expression between two groups; whereas the KS-test is sensitive to any change in distribution of expression values.

```
pVals <- apply(norm, 1, function(x) {
        wilcox.test(x[group =="NA19101"],
                x[group=="NA19239"])$p.value
        })
# multiple testing correction
pVals <- p.adjust(pVals, method = "fdr")
DE_Quality_AUC(pVals)
```

```
## [1] 0.8320326
```

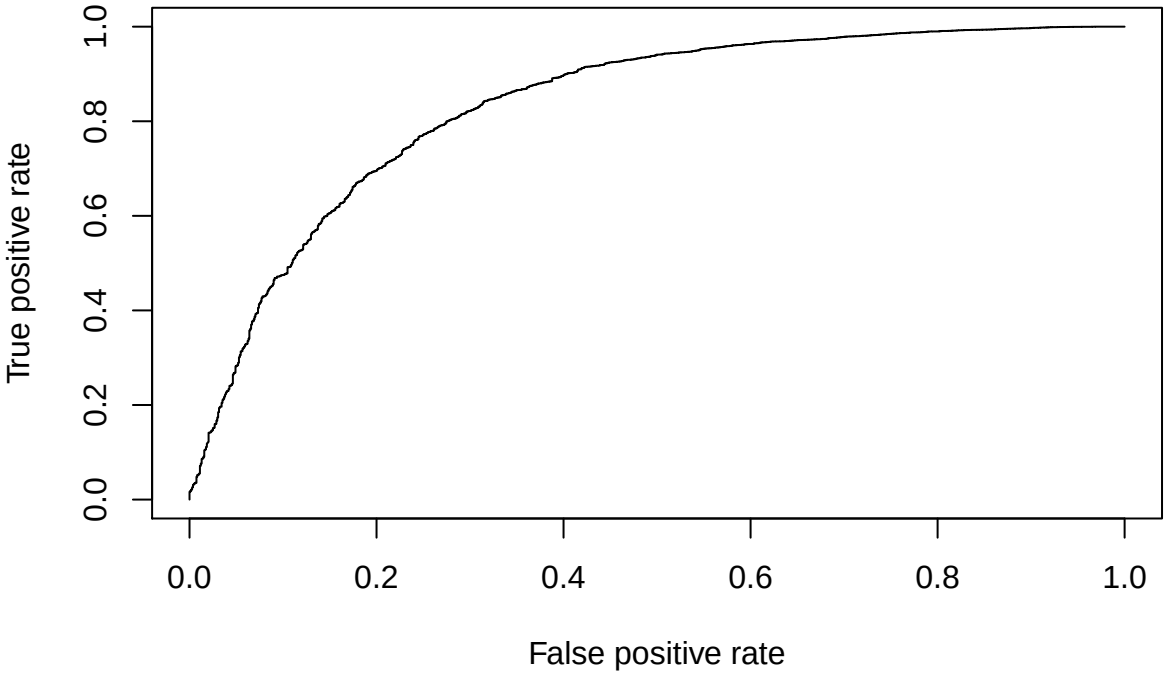


Figure 23.2: ROC curve for Wilcox test.

23.4 edgeR

We've already used edgeR for differential expression in Chapter **??**. edgeR is based on a negative binomial model of gene expression and uses a generalized linear model (GLM) framework, the enables us to include other factors such as batch to the model.

```
dge <- DGEList(counts=counts, norm.factors = rep(1, length(counts[1,])), group=group)
group_edgeR <- factor(group)
design <- model.matrix(~group_edgeR)
dge <- estimateDisp(dge, design = design, trend.method="none")
fit <- glmFit(dge, design)
res <- glmLRT(fit)
pVals <- res$table[,4]
names(pVals) <- rownames(res$table)

pVals <- p.adjust(pVals, method = "fdr")
DE_Quality_AUC(pVals)
```

```
## [1] 0.8477189
```

23.5 Monocle

Monocle can use several different models for DE. For count data it recommends the Negative Binomial model (negbinomial.size). For normalized data it recommends log-transforming it then using a normal distribution (gaussianff). Similar to edgeR this method uses a GLM framework so in theory can account for batches, however in practice the model fails for this dataset if batches are included.

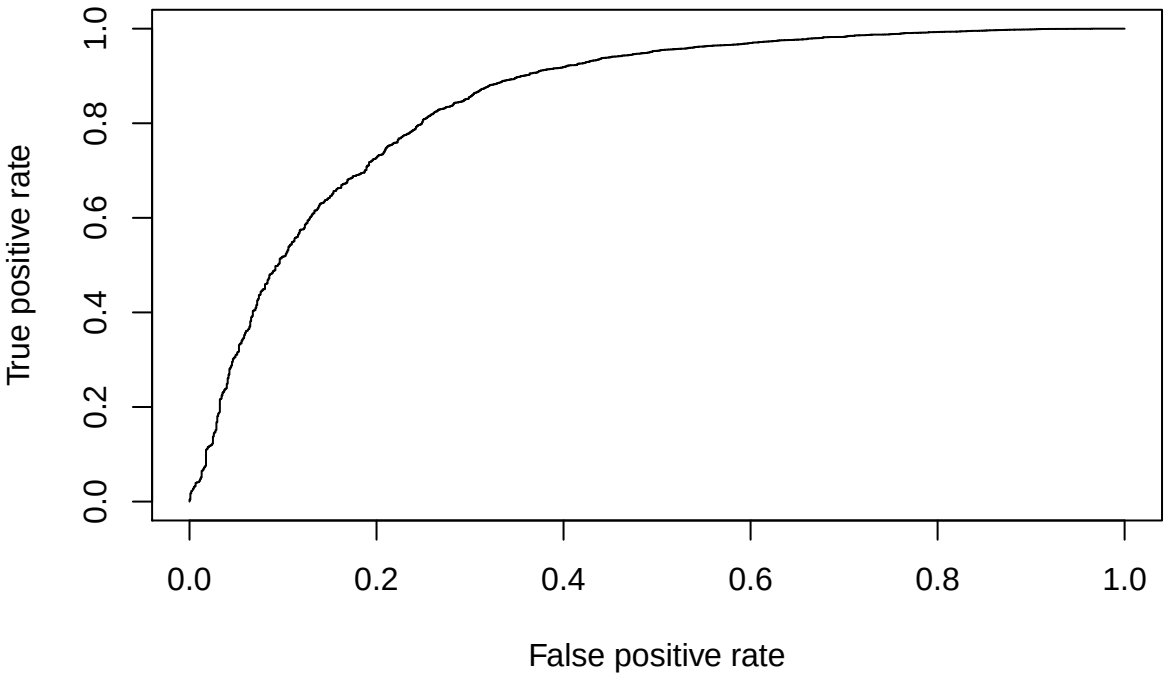


Figure 23.3: ROC curve for edgeR.

```
pd <- data.frame(group=group, batch=batch)
rownames(pd) <- colnames(counts)
pd <- new("AnnotatedDataFrame", data = pd)

Obj <- newCellDataSet(as.matrix(counts), phenoData=pd,
        expressionFamily=negbinomial.size())
Obj <- estimateSizeFactors(Obj)
Obj <- estimateDispersions(Obj)
res <- differentialGeneTest(Obj,fullModelFormulaStr="~group")

pVals <- res[,3]
names(pVals) <- rownames(res)
pVals <- p.adjust(pVals, method = "fdr")
DE_Quality_AUC(pVals)
```

```
## [1] 0.8252662
```

Exercise: Compare the results using the negative binomial model on counts and those from using the normal/gaussian model (gaussianff()) on log-transformed normalized counts.

Answer:

```
## [1] 0.7357829
```

23.6 MAST

MAST is based on a zero-inflated negative binomial model. It tests for differential expression using a hurdle model to combine tests of discrete (0 vs not zero) and continuous (non-zero values) aspects of gene expression. Again this uses a linear modelling framework to enable complex models to be considered.

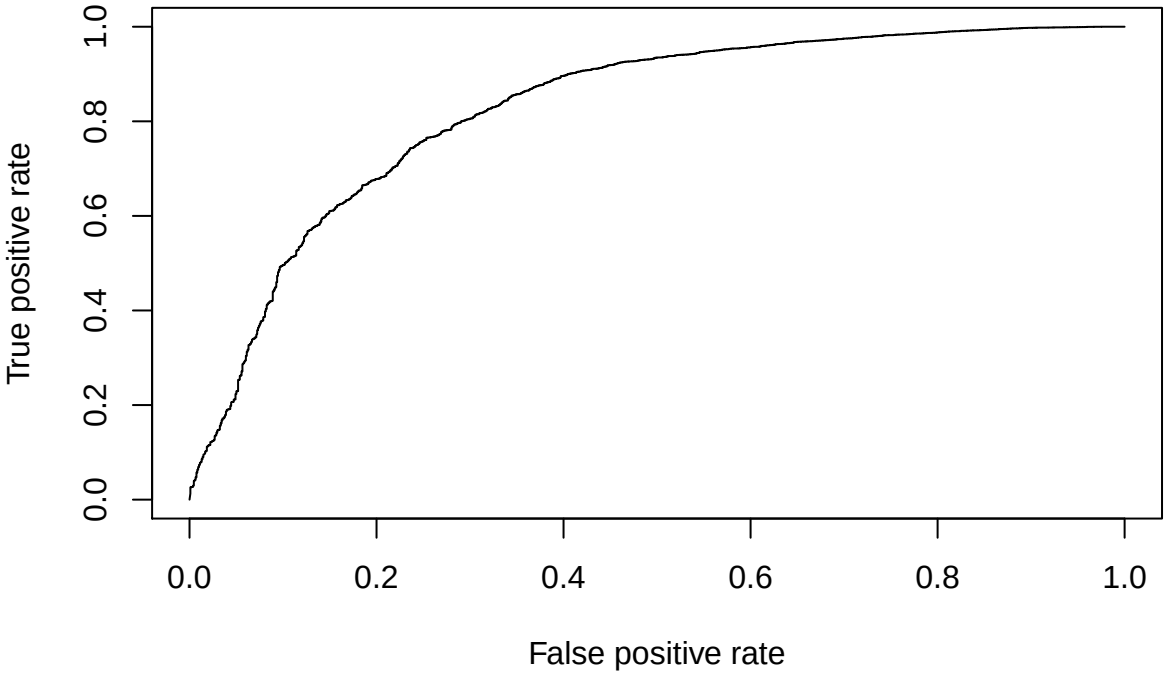


Figure 23.4: ROC curve for Monocle.

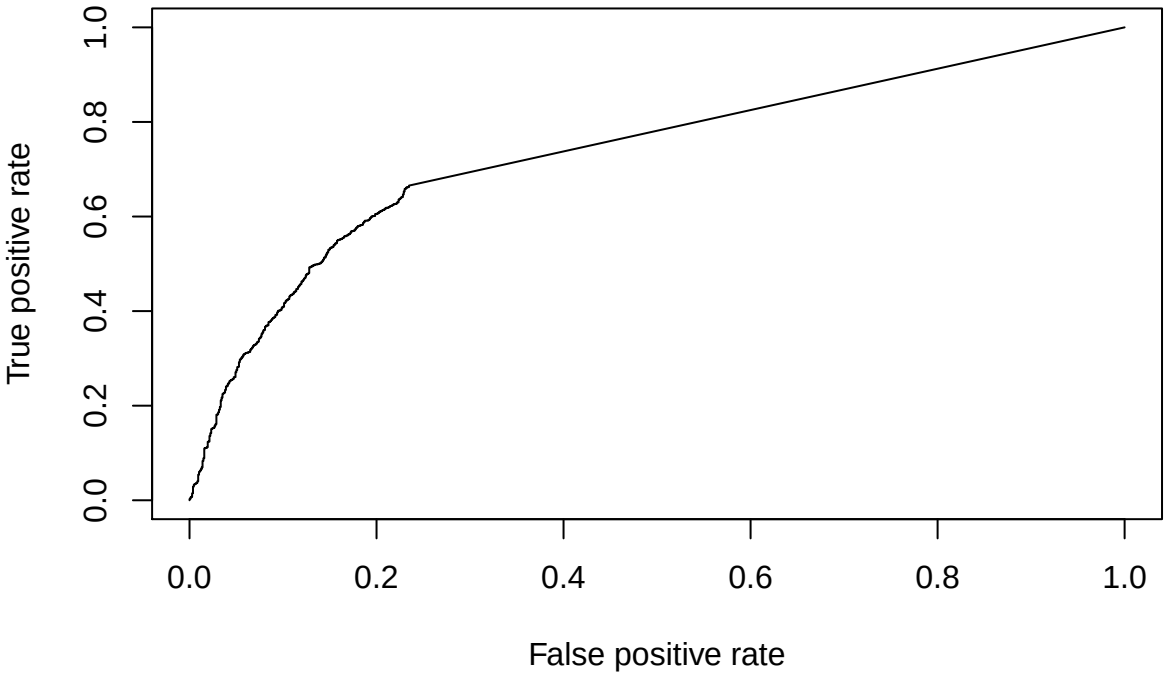


Figure 23.5: ROC curve for Monocle-gaussian.

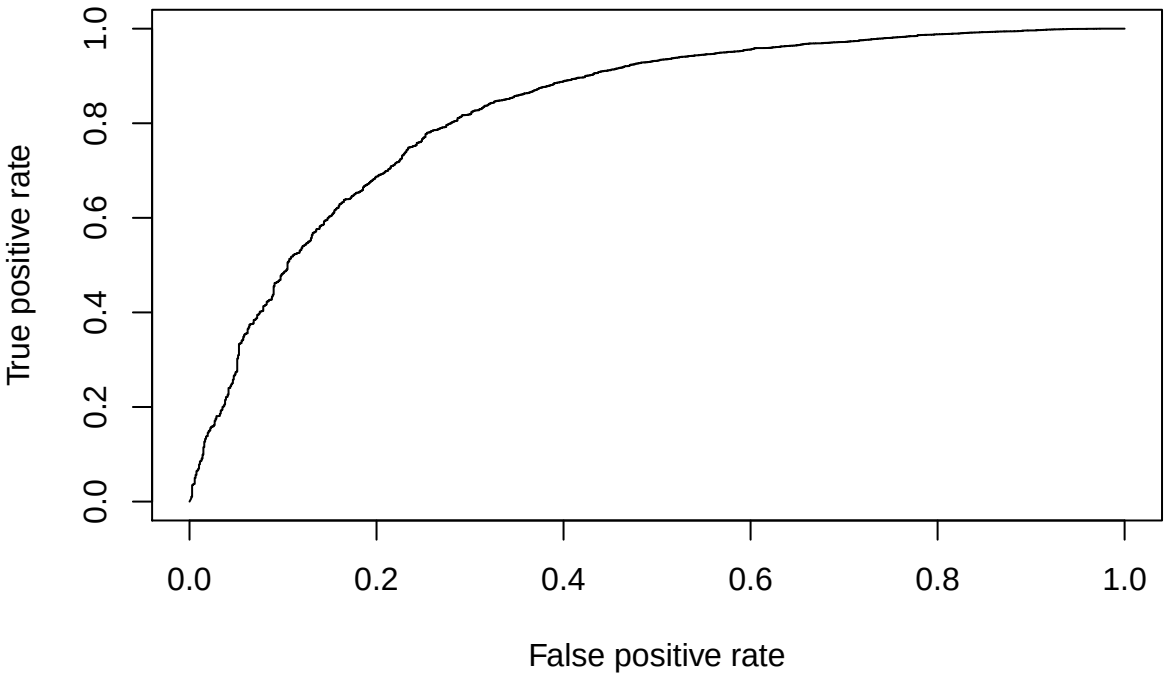


Figure 23.6: ROC curve for MAST.

```
log_counts <- log(counts+1)/log(2)
fData = data.frame(names=rownames(log_counts))
rownames(fData) = rownames(log_counts);
cData = data.frame(cond=group)
rownames(cData) = colnames(log_counts)

obj <- FromMatrix(as.matrix(log_counts), cData, fData)
colData(obj)$cngeneson <- scale(colSums(assay(obj)>0))
cond <- factor(colData(obj)$cond)

# Model expression as function of condition & number of detected genes
zlmCond <- zlm.SingleCellAssay(~cond + cngeneson, obj)
```

```
## Warning: 'zlm.SingleCellAssay' is deprecated.
## Use 'zlm' instead.
## See help("Deprecated")
```

```
## Warning in .nextMethod(object = object, value = value): Coefficients
## condNA19239 are never estimible and will be dropped.
```

```
summaryCond <- summary(zlmCond, doLRT="condNA19101")
summaryDt <- summaryCond$datatable

summaryDt <- as.data.frame(summaryDt)
pVals <- unlist(summaryDt[summaryDt$component == "H",4]) # H = hurdle model
names(pVals) <- unlist(summaryDt[summaryDt$component == "H",1])
pVals <- p.adjust(pVals, method = "fdr")
DE_Quality_AUC(pVals)
```

```
## [1] 0.8284046
```

23.7 Slow Methods (>1h to run)

These methods are too slow to run today but we encourage you to try them out on your own:

23.8 BPSC

BPSC uses the Poisson-Beta model of single-cell gene expression, which we discussed in the previous chapter, and combines it with generalized linear models which we've already encountered when using edgeR. BPSC performs comparisons of one or more groups to a reference group ("control") and can include other factors such as batches in the model.

```
library(BPSC)
bpsc_data <- norm[,batch=="NA19101.r1" | batch=="NA19239.r1"]
bpsc_group = group[batch=="NA19101.r1" | batch=="NA19239.r1"]

control_cells <- which(bpsc_group == "NA19101")
design <- model.matrix(~bpsc_group)
coef=2 # group label
res=BPglm(data=bpsc_data, controlIds=control_cells, design=design, coef=coef,
               estIntPar=FALSE, useParallel = FALSE)
pVals = res$PVAL
pVals <- p.adjust(pVals, method = "fdr")
DE_Quality_AUC(pVals)
```

23.9 SCDE

SCDE is the first single-cell specific DE method. It fits a zero-inflated negative binomial model to expression data using Bayesian statistics. The usage below tests for differences in mean expression of individual genes across groups but recent versions include methods to test for differences in mean expression or dispersion of groups of genes, usually representing a pathway.

```
library(scde)
cnts <- apply(
    counts,
    2,
    function(x) {
        storage.mode(x) <- 'integer'
        return(x)
    }
)
names(group) <- 1:length(group)
colnames(cnts) <- 1:length(group)
o.ifm <- scde::scde.error.models(
    counts = cnts,
    groups = group,
    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 = group,
    n.randomizations = 100,
    n.cores = 1,
    verbose = 0
)
# Convert Z-scores into 2-tailed p-values
pVals <- pnorm(abs(resSCDE$cZ), lower.tail = FALSE) * 2
pVals <- p.adjust(pVals, method = "fdr")
DE_Quality_AUC(pVals)
```

Chapter 24

"Ideal" scRNAseq pipeline (as of Mar 2017)

24.1 Experimental Design

- Avoid confounding biological and batch effects (Figure **??**)
  - 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)
  - Lose splicing information
  - Use longer UMIs (~10bp)
  - Correct for sequencing errors in UMIs using UMI-tools
- Spike-ins
  - Useful for quality control
  - May be useful for normalizing read counts
  - 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

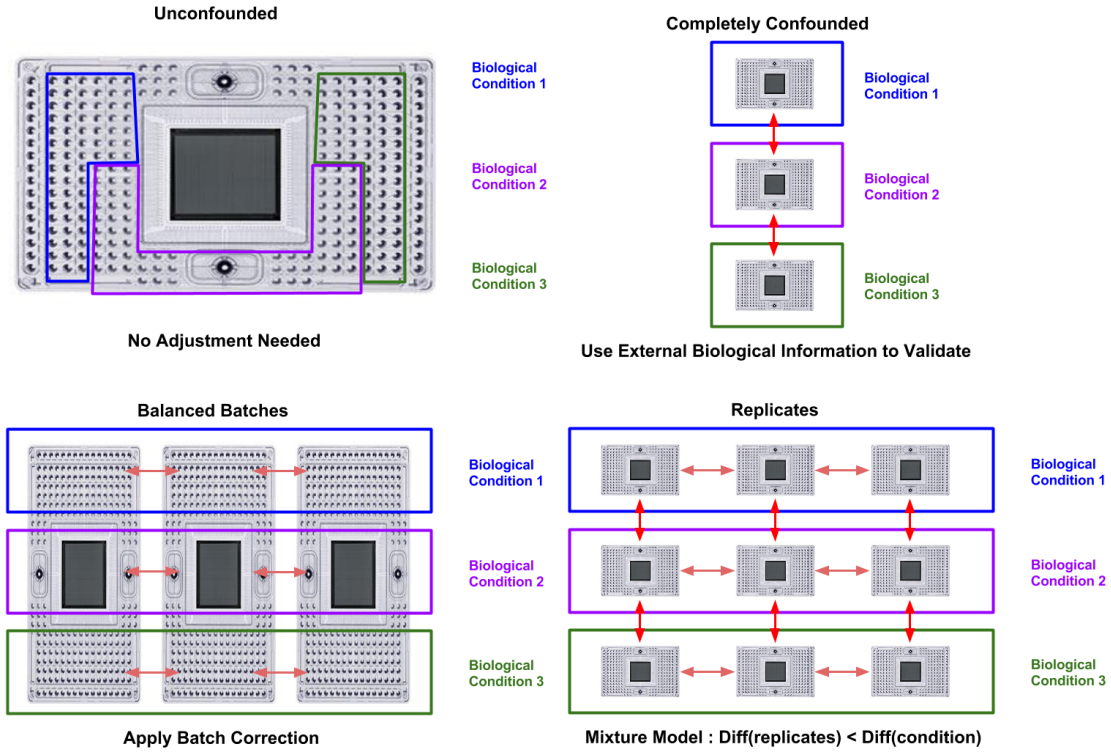


Figure 24.1: Appropriate approaches to batch effects in scRNASeq. Red arrows indicate batch effects which are (pale) or are not (vibrant) correctable through batch-correction.

- Quantification
  - Small dataset, no UMIs : featureCounts
  - 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
  - small dataset/unknown number of branches: Monocle2
  - large continuous dataset: destiny
- Differential Expression
  - Small number of cells and few groups : scde
  - Replicates with batch effects : mixture/linear models
  - Balanced batches: edgeR or MAST
  - Large datasets: Kruskal-Wallis test (all groups at once), or Wilcox-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

26.4 scRNA-seq public datasets

- Hemberg group's public datasets