An introduction to 'ascend' - Processing and analysis of retinal ganglion cells

Anne Senabouth

2018-04-05

The `ascend` package provides a series of tools for the processing and analysis of single cell RNA-seq (scRNA-seq) in R. These tools perform tasks such as filtering, normalisation, clustering and differential expression.

About the dataset

This dataset comprises of 1272 human embryonic stem cell-derived retinal ganglion cells (RGCs). The single cell libraries were prepared with the Chromium Single Cell 3' Solution system by 10x Genomics. Two libraries were prepared - one consisting of THY1-positive cells (Batch 1) and THY1-negative cells (Batch 2). Sequence from these two batches were aggregated and batch normalised using 10x Genomics' Cell Ranger Single Cell Software Suite 1.3.1.

You can read more about this dataset in the paper Single Cell RNA Sequencing of stem cell-derived retinal ganglion cells by Daniszewski et al. 2017.

Loading data for use in 'ascend'

About the Expression and Metadata Set (EMSet)

An `ascend` Expression and Metadata Set (EMSet) is a S4 object that stores information generated and used by the `ascend` package. You can read more about this object in the vignette "An introduction to ascend - Analysis of Single Cell Expression, Normalisation and Differential expression".

SHORTCUT - Load data from Cell Ranger into ascend automatically

If you are using Chromium data, you can load the data into R with the `LoadCellRanger` function. This function loads the data into an EMSet, with the assumption that mitochondrial and ribosomal genes are controls for this experiment.

```
em.set <- LoadCellRanger("RGC_scRNASeq/", "GRCh38")
```

Preparing data manually

This part of the vignette is for those who wish to prepare the data manually. The data for this vignette can be downloaded from the ascend website repository.

You can load the data with the following command:

```
load("RGC_scRNASeq.RData")
```

These objects contain all the information we need to create an `EMSet`.

Expression matrix

The main source of input is an expression matrix, or a gene-cell matrix where each row represents a transcript and each column represents a cell. Cell Ranger - the processing pipeline for the Chromium platform, has stored the expression matrix in a Market Exchange Format (MEX) file called `matrix.mtx`. This was read into R with the `readMM` function from the `Matrix` package.

Let's have a look at a small part of the matrix.

```
matrix <- as.data.frame(as.matrix(matrix))
matrix[1:5,1:3]
```

```
##   V1 V2 V3
## 1  0  0  0
## 2  0  0  0
## 3  0  0  0
## 4  0  0  0
## 5  0  0  0
```

`readMM` reads the data in as a sparse matrix, using less memory than data frames and matrices. The expression matrix can be kept in this format, but as we want to view the contents of the matrix for this tutorial – we have converted it into a data frame. This data frame lacks row and column labels as Cell Ranger has stored them in the two other files - `barcodes.tsv` and `genes.tsv`.

Preparing Cell Information

`barcodes.tsv` is a CSV file containing cell identifier and batch information. Chromium uses actual cell barcodes as cell identifiers and has attached a number to each barcode. This number represents the batch the cell originated from.

```
barcodes[1:5,]
```

```
## [1] AAACCTGAGCTGTTCA-1 AAACCTGCAATTCCTT-1 AAACCTGGTCTACCTC-1
## [4] AAACCTGTCGGAGCAA-1 AAACGGGAGTCGATAA-1
## 1272 Levels: AAACCTGAGACCACGA-2 AAACCTGAGCTGTTCA-1 ... TTTGTCATCTTCATGT-2
```

Extract the batch numbers from the cell identifiers by splitting each string at the '-' symbol and retrieve the second part of the string.

```
batch.information <- unlist(as.numeric(
  lapply(strsplit(as.character(barcodes$V1), "-"), `[`, 2)))
batch.information[1:5]
```

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

Add this batch information to the barcodes data frame, which will become our Cell Information dataframe.

```
colnames(barcodes) <- c("cell_barcode")
barcodes$batch <- as.numeric(batch.information)
barcodes[1:5,]
```

```
##         cell_barcode batch
## 1 AAACCTGAGCTGTTCA-1     1
## 2 AAACCTGCAATTCCTT-1     1
## 3 AAACCTGGTCTACCTC-1     1
## 4 AAACCTGTCGGAGCAA-1     1
## 5 AAACGGGAGTCGATAA-1     1
```

Finally, add the cell identifiers to the expression matrix as column names.

```
colnames(matrix) <- barcodes[,1]
matrix[1:5, 1:5]
```

```
##   AAACCTGAGCTGTTCA-1 AAACCTGCAATTCCTT-1 AAACCTGGTCTACCTC-1
## 1                  0                  0                  0
## 2                  0                  0                  0
## 3                  0                  0                  0
## 4                  0                  0                  0
## 5                  0                  0                  0
##   AAACCTGTCGGAGCAA-1 AAACGGGAGTCGATAA-1
## 1                  0                  0
## 2                  0                  0
## 3                  0                  0
## 4                  0                  0
## 5                  0                  0
```

Gene Information

`genes.tsv` contains the names of transcripts identified by Cell Ranger. This csv file contains ENSEMBL transcript IDs in one column and their corresponding gene name in the other column. Either of these identifiers can be used as row names in the expression matrix.

```
colnames(genes) <- c("ensembl_id", "gene_symbol")
genes[1:5,]
```

```
##        ensembl_id  gene_symbol
## 1 ENSG00000243485    MIR1302-2
## 2 ENSG00000237613       FAM138A
## 3 ENSG00000186092         OR4F5
## 4 ENSG00000238009 RP11-34P13.7
## 5 ENSG00000239945 RP11-34P13.8
```

For this tutorial, we will use gene names. As genes can be associated with more than one transcript, we need to make the names unique with `make.unique` before adding them to the expression matrix. We also need to swap the order of the identifiers, as ascend requires our chosen names to be in the first column of the Gene Information dataframe.

```
genes <- genes[,c("gene_symbol", "ensembl_id")]
gene.names <- make.unique(as.vector(genes$gene_symbol))
rownames(matrix) <- gene.names
matrix[1:5, 1:3]
```

```
##              AAACCTGAGCTGTTCA-1 AAACCTGCAATTCCTT-1 AAACCTGGTCTACCTC-1
## MIR1302-2                     0                  0                  0
## FAM138A                       0                  0                  0
## OR4F5                         0                  0                  0
## RP11-34P13.7                  0                  0                  0
## RP11-34P13.8                  0                  0                  0
```

Now that the gene names have been modified, the gene_names column in the `genes` data frame needs to be updated. This will link the information in this data frame with the rows of the expression matrix.

```
genes$gene_symbol <- gene.names
genes[1:15,]
```

```
##     gene_symbol      ensembl_id
## 1     MIR1302-2 ENSG00000243485
## 2        FAM138A ENSG00000237613
## 3          OR4F5 ENSG00000186092
## 4   RP11-34P13.7 ENSG00000238009
## 5   RP11-34P13.8 ENSG00000239945
## 6  RP11-34P13.14 ENSG00000239906
## 7   RP11-34P13.9 ENSG00000241599
## 8     FO538757.2 ENSG00000279928
## 9     FO538757.1 ENSG00000279457
## 10    AP006222.2 ENSG00000228463
## 11 RP5-857K21.15 ENSG00000236743
## 12  RP4-669L17.2 ENSG00000236601
## 13 RP4-669L17.10 ENSG00000237094
## 14        OR4F29 ENSG00000278566
## 15  RP5-857K21.4 ENSG00000230021
```

Defining Controls

Finally, we need to identify controls for this experiment. Ribosomal and mitochondrial genes are typically used as controls for single-cell experiments, so will use these genes for the tutorial. Spike-ins should be used as controls if they are included in the experiment.

We are using a quick method of identifying mitochondrial and ribosomal genes, by using the `grep` function to identify these genes by their prefix.

```
mito.genes <- rownames(matrix)[grep("^MT-", rownames(matrix),
                                    ignore.case = TRUE)]
ribo.genes <- rownames(matrix)[grep("^RPS|^RPL",
                                    rownames(matrix),
                                    ignore.case = TRUE)]
controls <- list(Mt = mito.genes, Rb = ribo.genes)
controls
```

```
## $Mt
##  [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
##  [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
##
## $Rb
##   [1] "RPL22"          "RPL11"          "RPS6KA1"        "RPS8"
##   [5] "RPL5"           "RPS27"          "RPS6KC1"        "RPS7"
##   [9] "RPS27A"         "RPL31"          "RPL37A"         "RPL32"
##  [13] "RPL15"          "RPSA"           "RPL14"          "RPL29"
##  [17] "RPL24"          "RPL22L1"        "RPL39L"         "RPL35A"
##  [21] "RPL9"           "RPL34-AS1"      "RPL34"          "RPS3A"
##  [25] "RPL37"          "RPS23"          "RPS14"          "RPL26L1"
##  [29] "RPS18"          "RPS10-NUDT3"    "RPS10"          "RPL10A"
##  [33] "RPL7L1"         "RPS12"          "RPS6KA2"        "RPS6KA2-AS1"
##  [37] "RPS6KA3"        "RPS4X"          "RPS6KA6"        "RPL36A"
##  [41] "RPL36A-HNRNPH2" "RPL39"          "RPL10"          "RPS20"
##  [45] "RPL7"           "RPL30"          "RPL8"           "RPS6"
##  [49] "RPL35"          "RPL12"          "RPL7A"          "RPLP2"
##  [53] "RPL27A"         "RPS13"          "RPS6KA4"        "RPS6KB2"
##  [57] "RPS3"           "RPS25"          "RPS24"          "RPS26"
##  [61] "RPL41"          "RPL6"           "RPLP0"          "RPL21"
##  [65] "RPL10L"         "RPS29"          "RPL36AL"        "RPS6KL1"
##  [69] "RPS6KA5"        "RPS27L"         "RPL4"           "RPLP1"
##  [73] "RPS17"           "RPL3L"           "RPS2"            "RPS15A"
##  [77] "RPL13"           "RPL26"           "RPL23A"          "RPL23"
##  [81] "RPL19"           "RPL27"           "RPS6KB1"         "RPL38"
##  [85] "RPL17-C18orf32" "RPL17"           "RPS21"           "RPS15"
##  [89] "RPL36"           "RPS28"           "RPL18A"          "RPS16"
##  [93] "RPS19"           "RPL18"           "RPL13A"          "RPS11"
##  [97] "RPS9"            "RPL28"           "RPS5"            "RPS4Y1"
## [101] "RPS4Y2"          "RPL3"            "RPS19BP1"
```

Building an EMSet

We can now load all of this information into an `EMSet`, using the `NewEMSet` function.

```
em.set <- NewEMSet(ExpressionMatrix = matrix, GeneInformation = genes,
                   CellInformation = barcodes, Controls = controls)
```

To view information about this object, enter the name of the object into the console.

```
em.set
```

```
## [1] "ascend Object - EMSet"
## [1] "Expression Matrix: 33020 genes and 1272 cells"
## [1] "Controls:"
## $Mt
##  [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
##  [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
##
## $Rb
##   [1] "RPL22"           "RPL11"           "RPS6KA1"         "RPS8"
##   [5] "RPL5"            "RPS27"           "RPS6KC1"         "RPS7"
##   [9] "RPS27A"          "RPL31"           "RPL37A"          "RPL32"
##  [13] "RPL15"           "RPSA"            "RPL14"           "RPL29"
##  [17] "RPL24"           "RPL22L1"         "RPL39L"          "RPL35A"
##  [21] "RPL9"            "RPL34-AS1"       "RPL34"           "RPS3A"
##  [25] "RPL37"           "RPS23"           "RPS14"           "RPL26L1"
##  [29] "RPS18"           "RPS10-NUDT3"     "RPS10"           "RPL10A"
##  [33] "RPL7L1"          "RPS12"           "RPS6KA2"         "RPS6KA2-AS1"
##  [37] "RPS6KA3"         "RPS4X"           "RPS6KA6"         "RPL36A"
##  [41] "RPL36A-HNRNPH2" "RPL39"           "RPL10"           "RPS20"
##  [45] "RPL7"            "RPL30"           "RPL8"            "RPS6"
##  [49] "RPL35"           "RPL12"           "RPL7A"           "RPLP2"
##  [53] "RPL27A"          "RPS13"           "RPS6KA4"         "RPS6KB2"
##  [57] "RPS3"            "RPS25"           "RPS24"           "RPS26"
##  [61] "RPL41"           "RPL6"            "RPLP0"           "RPL21"
##  [65] "RPL10L"          "RPS29"           "RPL36AL"         "RPS6KL1"
##  [69] "RPS6KA5"         "RPS27L"          "RPL4"            "RPLP1"
##  [73] "RPS17"           "RPL3L"           "RPS2"            "RPS15A"
##  [77] "RPL13"           "RPL26"           "RPL23A"          "RPL23"
##  [81] "RPL19"           "RPL27"           "RPS6KB1"         "RPL38"
##  [85] "RPL17-C18orf32" "RPL17"           "RPS21"           "RPS15"
##  [89] "RPL36"           "RPS28"           "RPL18A"          "RPS16"
##  [93] "RPS19"           "RPL18"           "RPL13A"          "RPS11"
##  [97] "RPS9"            "RPL28"           "RPS5"            "RPS4Y1"
## [101] "RPS4Y2"          "RPL3"            "RPS19BP1"
```

Adding additional metadata to the EMSet

We can add other information to the EMSet after it is created.

For example, the cells in this dataset were sorted for expression of the THY1 protein. This corresponds to the batch identifiers that we have just pulled out from the barcodes.

```
cell.info <- GetCellInfo(em.set)
thy1.expression <- cell.info$batch
thy1.expression <- thy1.expression == 1
cell.info$THY1 <- thy1.expression
cell.info[1:5, ]
```

```
##         cell_barcode batch THY1
## 1 AAACCTGAGCTGTTCA-1     1 TRUE
## 2 AAACCTGCAATTCCTT-1     1 TRUE
## 3 AAACCTGGTCTACCTC-1     1 TRUE
## 4 AAACCTGTCGGAGCAA-1     1 TRUE
## 5 AAACGGGAGTCGATAA-1     1 TRUE
```

We are also interested in the expression of transcripts from the BRN3 family (POU4F1, POU4F2, POU4F3), that are expressed in retinal ganglion cells. We can identify cells that are expressing these transcripts by looking at the row that contains counts for these genes.

```
# Create a list of transcript names
brn3.transcripts <- c("POU4F1", "POU4F2", "POU4F3")

# Extract expression matrix from the em.set as a data frame
expression.matrix <- GetExpressionMatrix(em.set, format = "data.frame")

# Extract rows from matrix belonging to these transcripts
brn3.transcript.counts <- expression.matrix[brn3.transcripts, ]

# Identify cells (columns) where transcript counts are greater than one
brn3.cells <- colSums(brn3.transcript.counts) > 0

# Add new information to cell information
cell.info$BRN3 <- brn3.cells

# View cell.info
cell.info[1:5,]
```

```
##         cell_barcode batch THY1  BRN3
## 1 AAACCTGAGCTGTTCA-1     1 TRUE FALSE
## 2 AAACCTGCAATTCCTT-1     1 TRUE FALSE
## 3 AAACCTGGTCTACCTC-1     1 TRUE FALSE
## 4 AAACCTGTCGGAGCAA-1     1 TRUE FALSE
## 5 AAACGGGAGTCGATAA-1     1 TRUE FALSE
```

To load the modified cell information dataframe back into the EMSet, use the `ReplaceCellInfo` function.

```
em.set <- ReplaceCellInfo(em.set, cell.info)
```

Single-cell post-processing and normalisation workflow

The filtering workflow is based off A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor by Lun, McCarthy & Marioni 2016.

Preliminary QC

We can assess the quality of the data through a series of plots generated by `PlotGeneralQC`. These plots will be used to guide the filtering process.

Printing plots to PDF

The resulting plots are stored in a named list. You can use the `PlotPDF` function to output the plots in this list to a PDF file.

```r
raw.qc.plots <- PlotGeneralQC(em.set)
```

Classifying cells by cell cycle

To identify the stage of the cell cycle each cell is in, use `scranCellCycle`. This function is a wrapper for `scran`'s `cyclone` function. For more information on how this function works, refer to the `scran` documentation. The `scranCellCycle` and subsequently and `cyclone` function require a training dataset. In this case, we loaded the human dataset that comes packaged with `scran`. We also had to briefly convert the gene annotation used in the EMSet to ENSEMBL IDs, to match the training dataset. Fortunately, Cell Ranger has provided both identifiers in the `genes` data frame, so we can easily switch them with the `ConvertGeneAnnotation` function.

```r
# Convert the EMSet's gene annotation to ENSEMBL IDs stored in the ensembl_id
# column of the GeneInformation dataframe
em.set <- ConvertGeneAnnotation(em.set, "gene_symbol", "ensembl_id")

# Load scran's training dataset
training.data <- readRDS(system.file("exdata", "human_cycle_markers.rds",
                                     package = "scran"))

# Run scranCellCycle
em.set <- scranCellCycle(em.set, training.data)

# View cell information
cell.info <- GetCellInfo(em.set)
cell.info[1:5, ]

# Convert annotation back to gene_symbol
em.set <- ConvertGeneAnnotation(em.set, "ensembl_id", "gene_symbol")
```

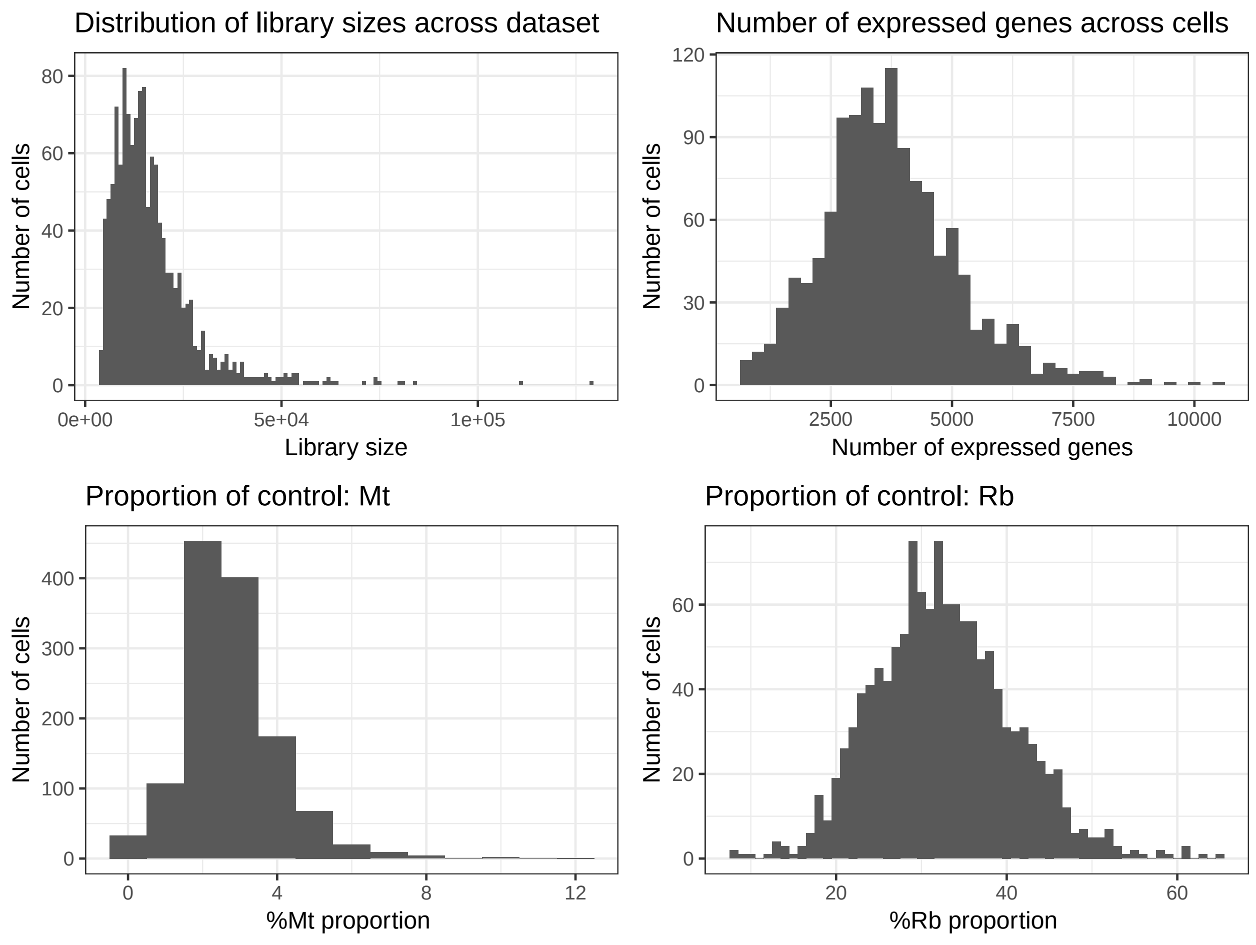
Cell filtering

Filter cells by library size and gene expression

First, we will filter cells based on outliers in term of library size, number of non-control genes expressed and control genes expressed.

We can use the following plots to examine the distributions of these values.

```r
grid.arrange(raw.qc.plots$LibSize,
             raw.qc.plots$FeatureCountsPerCell,
             raw.qc.plots$ControlPercentageTotalCounts$Mt,
             raw.qc.plots$ControlPercentageTotalCounts$Rb, ncol = 2)
```



The `FilterByOutliers` function will remove outliers based on these criteria. The threshold arguments refer to the median absolute deviations (MADs) below the median. These are set to 3 by default, but you can adjust them if required

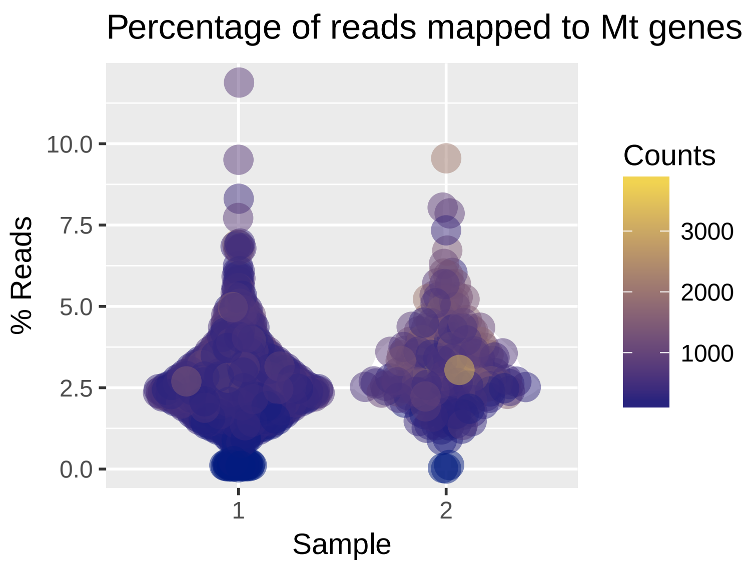
```
em.set <- FilterByOutliers(em.set, cell.threshold = 3, control.threshold = 3)
```

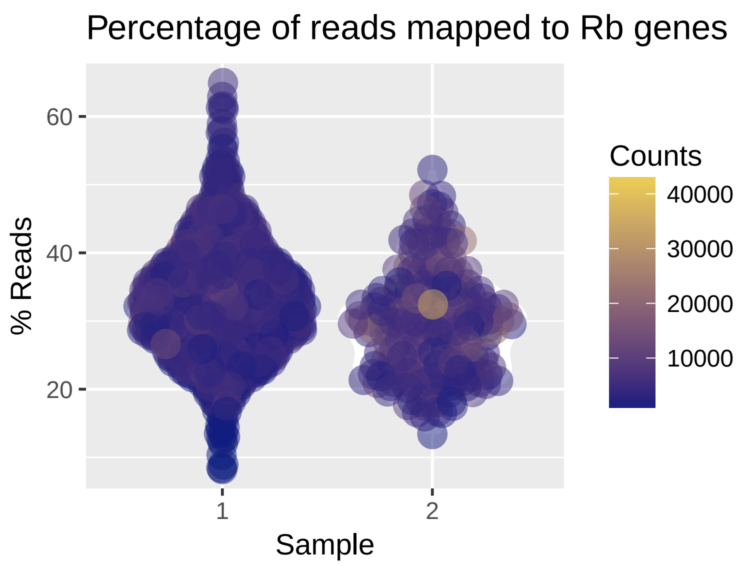
Filter cells by control gene expression

We removed a significant number of cells in the previous step that were expressing too many, or too few control genes. As ribosomal and mitochondrial genes are indicative of a stressed or dying cell, we need to perform some additional filtering and remove cells where they contribute to the bulk of the cell's expression.

The beehive plots below show the percentage of control genes in the transcriptomes of each cell, per sample.

```
grid.arrange(raw.qc.plots$ControlPercentageSampleCounts$Mt,
             raw.qc.plots$ControlPercentageSampleCounts$Rb, ncol = 1)
```





Review the control list by using `GetControls`. As you can see, we have stored the mitochondrial genes under "Mt" and ribosomal genes under "Rb."

```
print(GetControls(em.set))
```

```
## $Mt
##  [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
##  [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
##
## $Rb
##   [1] "RPL22"         "RPL11"         "RPS6KA1"       "RPS8"
##   [5] "RPL5"          "RPS27"         "RPS6KC1"       "RPS7"
##   [9] "RPS27A"        "RPL31"         "RPL37A"        "RPL32"
##  [13] "RPL15"         "RPSA"          "RPL14"         "RPL29"
##  [17] "RPL24"         "RPL22L1"       "RPL39L"        "RPL35A"
##  [21] "RPL9"          "RPL34-AS1"     "RPL34"         "RPS3A"
##  [25] "RPL37"         "RPS23"         "RPS14"         "RPL26L1"
##  [29] "RPS18"         "RPS10-NUDT3"   "RPS10"         "RPL10A"
##  [33] "RPL7L1"        "RPS12"         "RPS6KA2"       "RPS6KA2-AS1"
##  [37] "RPS6KA3"         "RPS4X"           "RPS6KA6"         "RPL36A"
##  [41] "RPL36A-HNRNPH2" "RPL39"           "RPL10"           "RPS20"
##  [45] "RPL7"            "RPL30"           "RPL8"            "RPS6"
##  [49] "RPL35"           "RPL12"           "RPL7A"           "RPLP2"
##  [53] "RPL27A"          "RPS13"           "RPS6KA4"         "RPS6KB2"
##  [57] "RPS3"            "RPS25"           "RPS24"           "RPS26"
##  [61] "RPL41"           "RPL6"            "RPLP0"           "RPL21"
##  [65] "RPL10L"          "RPS29"           "RPL36AL"         "RPS6KL1"
##  [69] "RPS6KA5"         "RPS27L"          "RPL4"            "RPLP1"
##  [73] "RPS17"           "RPL3L"           "RPS2"            "RPS15A"
##  [77] "RPL13"           "RPL26"           "RPL23A"          "RPL23"
##  [81] "RPL19"           "RPL27"           "RPS6KB1"         "RPL38"
##  [85] "RPL17-C18orf32" "RPL17"           "RPS21"           "RPS15"
##  [89] "RPL36"           "RPS28"           "RPL18A"          "RPS16"
##  [93] "RPS19"           "RPL18"           "RPL13A"          "RPS11"
##  [97] "RPS9"            "RPL28"           "RPS5"            "RPS4Y1"
## [101] "RPS4Y2"          "RPL3"            "RPS19BP1"
```

Use `FilterByCustomControl` to remove cells that are mostly expressing control genes. This function takes two arguments - the name of the list of control genes and the minimum percentage expression to filter by.

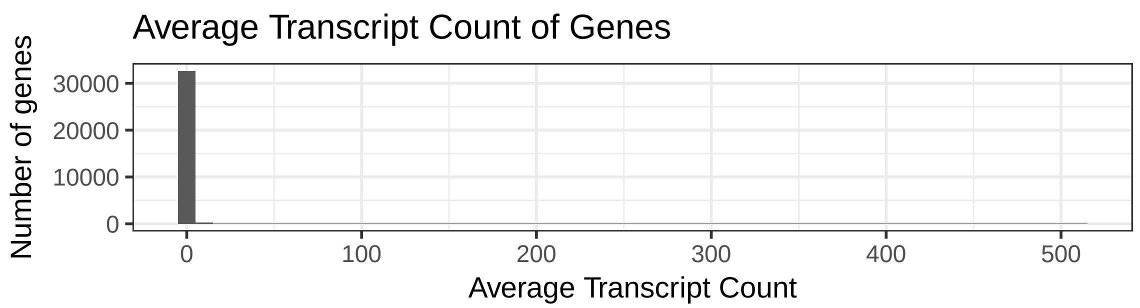
```r
# Filter by mitochondrial genes
em.set <- FilterByControl(control.name = "Mt", pct.threshold = 20, em.set)
# Filter by ribosomal genes
em.set <- FilterByControl(control.name = "Rb", pct.threshold = 50, em.set)
```

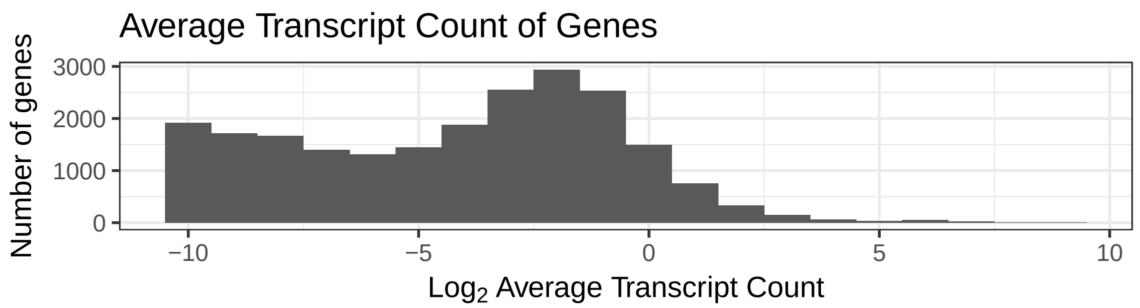
Some analyses will require the removal of these controls. This should not be done at this stage; it is best done after normalisation.

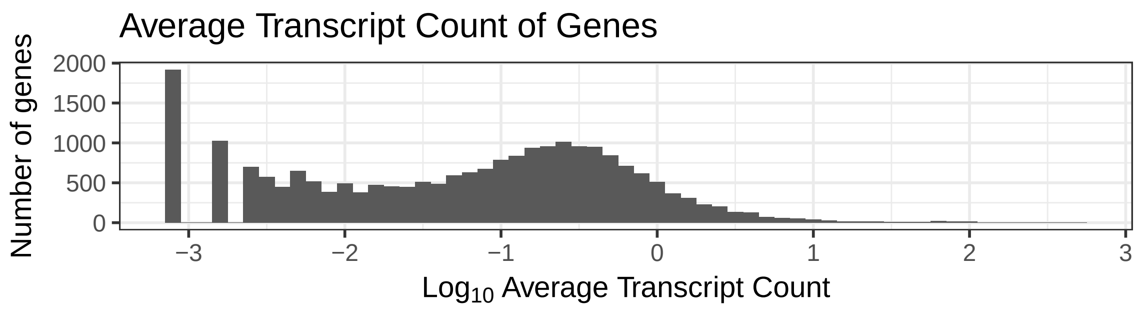
Removing low abundance genes

The average expression of genes can be reviewed on the average transcript count plots.

```r
grid.arrange(raw.qc.plots$AverageGeneCount,
             raw.qc.plots$Log2AverageGeneCount,
             raw.qc.plots$Log10AverageGeneCount,
             ncol = 1)
```







Due to the nature of single-cell RNASeq, many genes will have zero or near-zero expression. Have a closer look at the distribution with the Log2 and Log10 average transcript count plots.

If we wanted to remove genes that are only expressed in a small percentage of cells, we can use the `FilterLowAbundanceGenes` function. This will remove genes that are expressed in at most, a certain percentage of the cell population.

```
em.set <- FilterLowAbundanceGenes(em.set, pct.value = 1)
```

There are experiments where this is not ideal, such as this one as we are trying to characterise cell populations by genes that may only be expressed in a small number of cells. For example, we are interested in transcripts from the BRN3 family but these transcripts are expressed in only a small proportion of the cells.

```
expression.matrix <- GetExpressionMatrix(em.set, "data.frame")
brn3.transcripts <- c("POU4F1", "POU4F2", "POU4F3")
expression.matrix[brn3.transcripts,
                  which(colSums(expression.matrix[brn3.transcripts,]) > 0)[1:3]]
```

```
##        ACAGCCGTCTCGCTTG-1 ACGCCAGGTGGTGTAG-1 AGGGTGATCGCGTTTC-1
## POU4F1                  0                  0                  0
## POU4F2                  1                  2                  1
## POU4F3                  0                  0                  0
```

Other genes that are involved in the differentiation of the stem cells into retinal ganglion cells may also be lowly expressed, so we will omit this filtering step.

Filtering Review

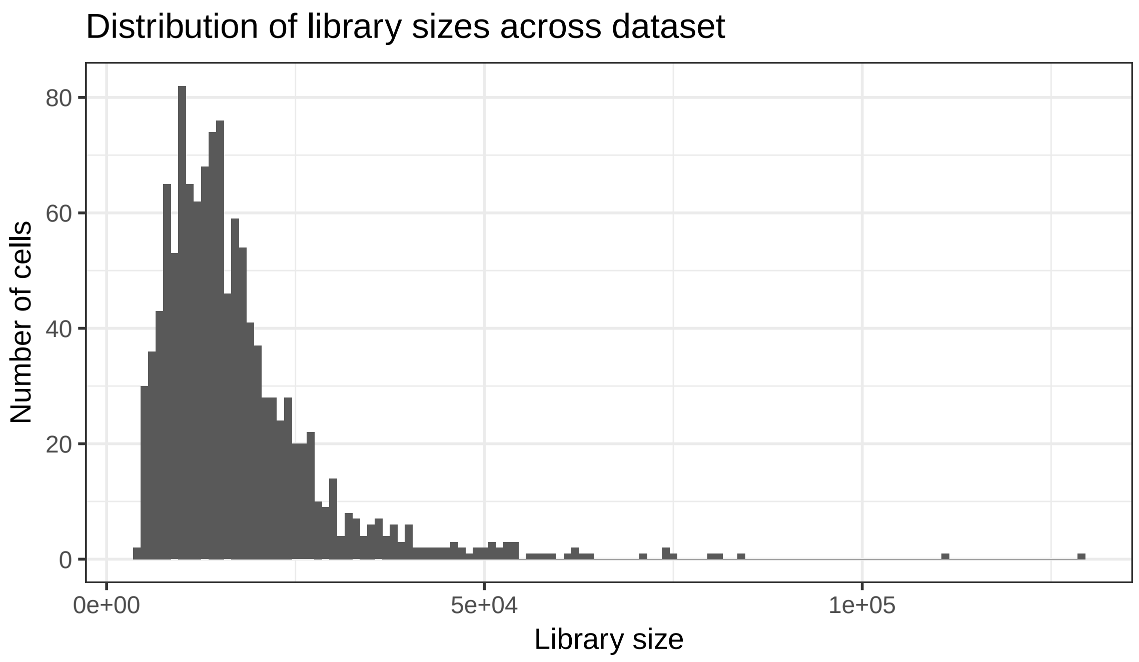
The filtering functions record which barcodes were removed by the function and stores them in the `EMSet`. You can review the number of cells filtered by the functions by using the `DisplayLog` function.

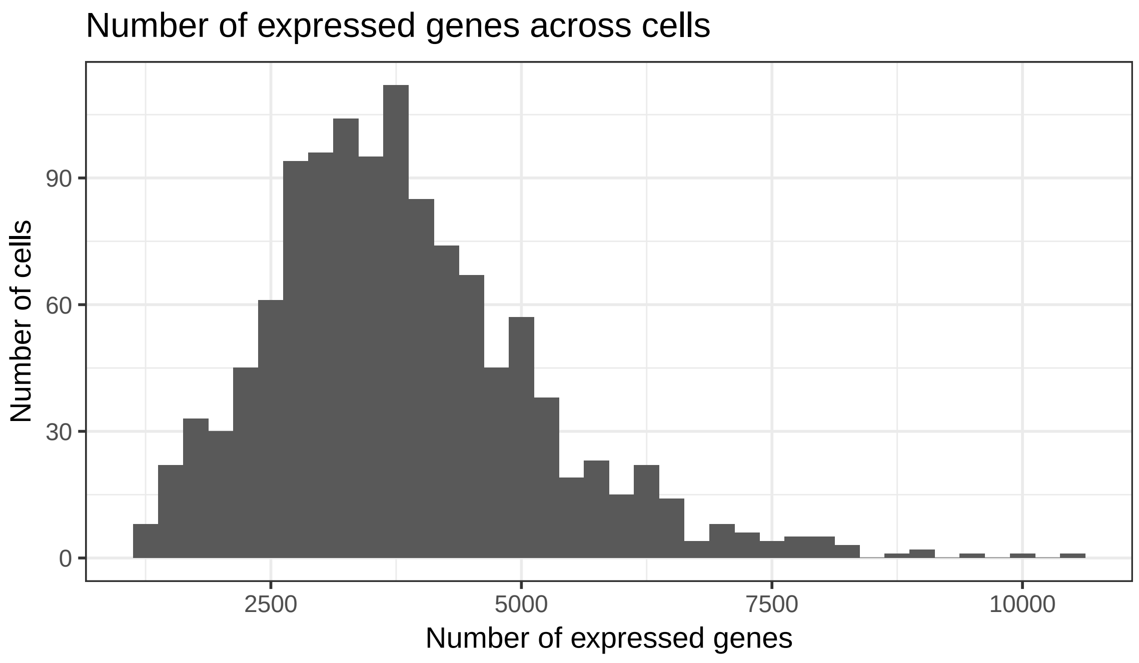
```
DisplayLog(em.set)
```

```
## $Controls
## [1] TRUE
##
## $FilterByOutliers
## $FilterByOutliers$CellsFilteredByLibSize
## NULL
##
## $FilterByOutliers$CellsFilteredByLowExpression
##  [1] "AACTGGTTCATAAAGG-1" "AAGACCTAGACAATAC-1" "ACAGCCGGTCTCTCTG-1"
##  [4] "ACGATACGTCACACGC-1" "ACTTGTTCAAGCGTAG-1" "AGCCTAACAGTAACGG-1"
##  [7] "AGTCTTTTCACCACCT-1" "AGTGTCAAGCCACTAT-1" "ATAACGCGTCGACTAT-1"
## [10] "ATCTACTCATTTGCTT-1" "ATTATCCTCCTTGACC-1" "CAGATCACAATAACGA-1"
## [13] "CCTCTGAAGATGTGGC-1" "CCTCTGATCGCATGAT-1" "CGCTATCGTGGCTCCA-1"
## [16] "CGTGTCTTCAATCACG-1" "CTCGTCATCAAACGGG-1" "CTTAACTCACCACGTG-1"
## [19] "GACCAATGTTCAACCA-1" "GACTAACGTTTGCATG-1" "GGCGTGTTCCAAGTAC-1"
## [22] "TAGACCAAGCTAACAA-1" "TAGCCGGCAGTCCTTC-1" "TCATTACAGGCAATTA-1"
## [25] "TGACTAGCATCACGTA-1" "ACTGAGTGTTAAGTAG-2"
##
## $FilterByOutliers$CellsFilteredByControls
## $FilterByOutliers$CellsFilteredByControls$Mt
##  [1] "AACACGTAGTGGGTTG-1" "AACGTTGTCACGAAGG-1" "AAGACCTGTCTGGAGA-1"
##  [4] "ACGATGTTCCCATTTA-1" "ACGGCCACACCAGCAC-1" "ACTTTCATCCTAAGTG-1"
##  [7] "AGATCTGGTCGACTGC-1" "CACAGGCCATGTTGAC-1" "CATCGGGAGGCGCTCT-1"
## [10] "CTCTACGCAATCCGAT-1" "CTGAAACAGCACAGGT-1" "CTGATAGGTAAACGCG-1"
## [13] "GAAACTCGTTGGTAAA-1" "GAACCTAAGATATGGT-1" "GCACATATCTTCATGT-1"
## [16] "GGACAAGCAACTGCGC-1" "GGACAAGTCCTGCCAT-1" "GTATCTTTCACCAGGC-1"
## [19] "GTGAAGGTCTAACTTC-1" "GTTCTCGGTCGTTGTA-1" "TCATTTGGTCCGAACC-1"
## [22] "TGGCTGGCATAGAAAC-1" "TGTGGTAGTGCAGTAG-1" "AAAGTAGGTTAGTGGG-2"
## [25] "AAGGCAGAGCTAACAA-2" "ACTGAGTGTTAAGTAG-2" "AGCTCCTTCTCCAGGG-2"
## [28] "ATGTGTGAGTCAAGCG-2" "CGCGTTTAGGTGATAT-2" "CTGTGCTCAGCGTTCG-2"
## [31] "GACAGAGTCACAATGC-2" "GACGTTAGTACCCAAT-2" "GCGACCAGTTTGCATG-2"
## [34] "GTGAAGGGTGCTAGCC-2" "TGAGCCGCACAAGACG-2" "TGTGGTAAGTACGCGA-2"
##
## $FilterByOutliers$CellsFilteredByControls$Rb
## [1] "AACTGGTTCATAAAGG-1" "ACGATACGTCACACGC-1" "AGCCTAACAGTAACGG-1"
## [4] "AGTCTTTTCACCACCT-1" "ATAACGCGTCGACTAT-1" "ATCTACTCATTTGCTT-1"
## [7] "ATTATCCTCCTTGACC-1" "CTTAACTCACCACGTG-1" "TCATTACAGGCAATTA-1"
##
##
##
## $FilteringLog
##   CellsFilteredByLibSize CellsFilteredByExpression CellsFilteredByControls
## 1                      0                        26                      45
##   CellsFilteredByMt CellsFilteredByRb
## 1                 0                11
##
## $FilterByControl
## $FilterByControl$Mt
## list()
##
## $FilterByControl$Rb
##  [1] "ACCAGTATCGGTTCGG-1" "CAGCGACAGCAGCGTA-1" "CGAATGTAGGCTCTTA-1"
##  [4] "CTTCTCTAGCACGCCT-1" "GGGAGATGTAAAGGAG-1" "GTATTCTAGTCCATAC-1"
##  [7] "TCGCGTTAGCAGGTCA-1" "TGCCCTAGTCCAGTTA-1" "TGGCTGGTCGAATCCA-1"
## [10] "TTAGTTCAGTTACGGG-1" "TTCGGTCAGGATGGAA-1"
```

You can also run `PlotGeneralQC` again to see how the filtering has altered the dataset.

```
grid.arrange(filtered.qc.plots$LibSize, filtered.qc.plots$FeatureCountsPerCell,
             ncol = 1)
```





Normalisation

Normalisation needs to be done at two levels - between batches and between cells.

The `ascend` package provides the following normalisation functions:

- NormaliseBatches: Normalise library sizes between batches.
- NormaliseByRLE: Normalise library sizes between cells by Relative Log Expression (RLE).
- scranNormalise: Normalise library sizes between cells using *scran*'s deconvolution method.

When and how the functions are applied are dependant on the dataset.

NormaliseBatches

Normalisation between batches needs to be done prior to filtering.

For this tutorial - we do not need to use the `NormaliseBatches` as this dataset was prepared with Cell Ranger's *aggr* pipeline. This pipeline uses a subsampling process to normalise between batches (Zheng et al. 2017).

However, we do need to normalise between cells. This can be done with either the `NormaliseByRLE` or `scranNormalise` functions.

NormaliseByRLE

In this method, each cell is considered as one library and assumes that most genes are not differentially expressed. It uses gene expression values higher than 0 to calculate the geometric means of a gene. The geometric mean is the mean of the expression of the gene across all cells (for cells where the gene is detected). Each gene has one geometric mean value for all cell. For each cell, the gene expression values are divided by the geometric means to get one normalisation factor for a gene in that cell. The median of all the normalisation factors for all genes in that cell is the normalisation factor for the cell. Post RLE normalisation, a gene with 0 expression still has 0 expression. A gene with expression higher than 0 will have an expression value equal the raw expression divided by the calculated normalization factor for the cell. As spike-ins affect library size, they need to be removed prior to normalisation.

This method is relatively quick and can be run on a desktop.

```
norm.set <- NormaliseByRLE(em.set)
```

scranNormalise

This function is a wrapper for the deconvolution method by Lun et al. 2015 that uses the scran and scater packages. This method takes into account the high proportion of zero counts in single-cell data and tackles the zero-inflation problem by applying a pooling strategy to calculate size-factors of each pool. The pooled size factors are then deconvoluted to infer the size factor for each cell, which are used scale the counts within that cell. The scran vignette explains the whole process in greater detail.

To ensure compatibility with `scran` and `scater`, the `EMSet` needs to have mitochondrial and ribosomal genes as controls. The control list also needs to be formatted as follows:

```
print(GetControls(em.set))
```

```
## $Mt
##  [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
##  [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
##
## $Rb
##   [1] "RPL22"          "RPL11"          "RPS6KA1"        "RPS8"
##   [5] "RPL5"            "RPS27"           "RPS6KC1"         "RPS7"
##   [9] "RPS27A"          "RPL31"           "RPL37A"          "RPL32"
##  [13] "RPL15"           "RPSA"            "RPL14"           "RPL29"
##  [17] "RPL24"           "RPL22L1"         "RPL39L"          "RPL35A"
##  [21] "RPL9"            "RPL34-AS1"       "RPL34"           "RPS3A"
##  [25] "RPL37"           "RPS23"           "RPS14"           "RPL26L1"
##  [29] "RPS18"           "RPS10-NUDT3"     "RPS10"           "RPL10A"
##  [33] "RPL7L1"          "RPS12"           "RPS6KA2"         "RPS6KA2-AS1"
##  [37] "RPS6KA3"         "RPS4X"           "RPS6KA6"         "RPL36A"
##  [41] "RPL36A-HNRNPH2" "RPL39"           "RPL10"           "RPS20"
##  [45] "RPL7"            "RPL30"           "RPL8"            "RPS6"
##  [49] "RPL35"           "RPL12"           "RPL7A"           "RPLP2"
##  [53] "RPL27A"          "RPS13"           "RPS6KA4"         "RPS6KB2"
##  [57] "RPS3"            "RPS25"           "RPS24"           "RPS26"
##  [61] "RPL41"           "RPL6"            "RPLP0"           "RPL21"
##  [65] "RPL10L"          "RPS29"           "RPL36AL"         "RPS6KL1"
##  [69] "RPS6KA5"         "RPS27L"          "RPL4"            "RPLP1"
##  [73] "RPS17"           "RPL3L"           "RPS2"            "RPS15A"
##  [77] "RPL13"           "RPL26"           "RPL23A"          "RPL23"
##  [81] "RPL19"           "RPL27"           "RPS6KB1"         "RPL38"
##  [85] "RPL17-C18orf32" "RPL17"           "RPS21"           "RPS15"
##  [89] "RPL36"           "RPS28"           "RPL18A"          "RPS16"
##  [93] "RPS19"           "RPL18"           "RPL13A"          "RPS11"
##  [97] "RPS9"            "RPL28"           "RPS5"            "RPS4Y1"
## [101] "RPS4Y2"          "RPL3"            "RPS19BP1"
```

If the dataset contains less than 10,000 cells, `scranNormalise` will run `scran`'s `computeSumFactors` function with preset sizes of 40, 60, 80 and 100. For larger datasets, `scranNormalise` will run `quickCluster` before `computeSumFactors`. `scran` 1.6.6 introduced an additional argument - *min.mean* to the function `computeSumFactors`. This is the threshold for average counts. By default, it is set by `ascend` to 1e-5 as this value works best for UMI data. If you are working with read counts, please set this value to 1.

This method is computationally intensive; we do not recommend running datasets larger than 5000 cells on a desktop machine. Datasets larger than 10,000 cells should be run on a HPC.

```
norm.set <- scranNormalise(em.set, quickCluster = FALSE, min.mean = 1e-5)
```

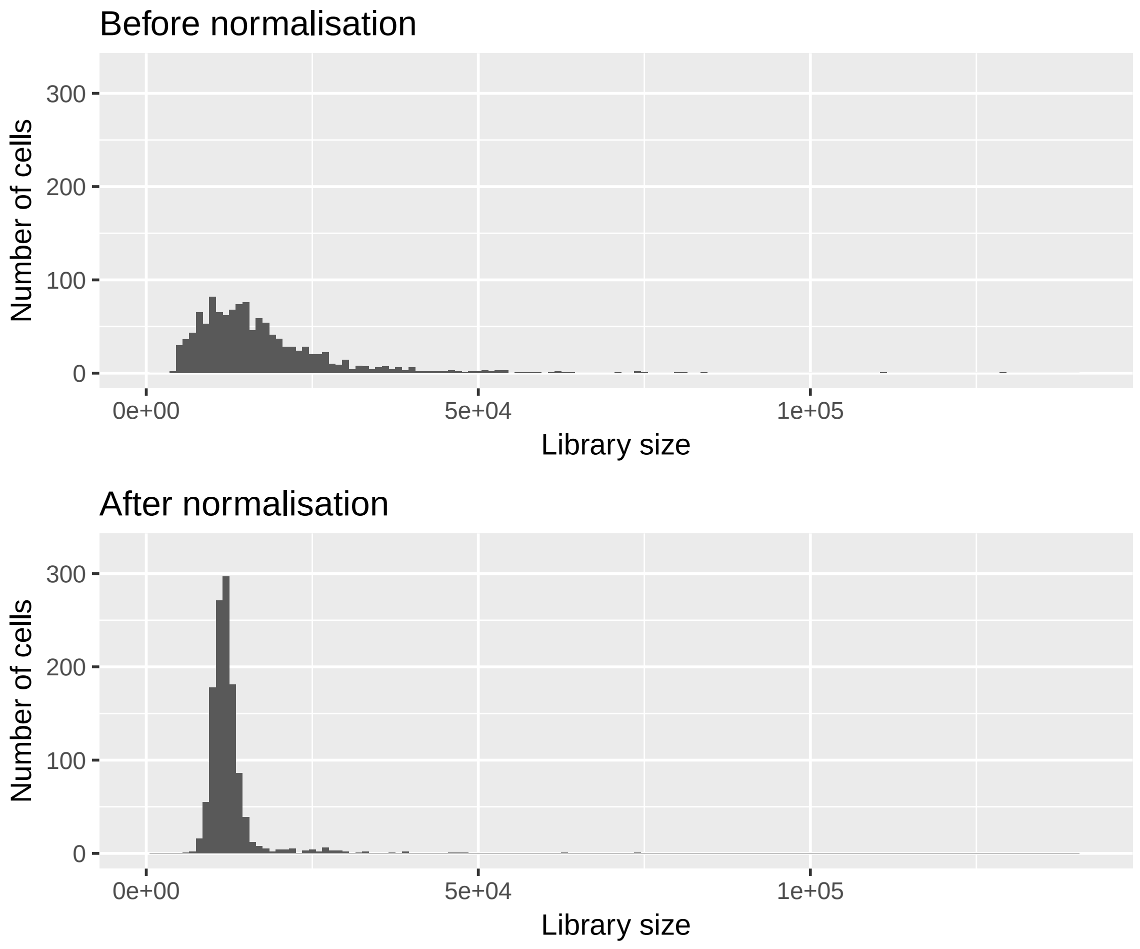
Reviewing the normalisation process

`PlotNormalisationQC` will generate a series of plots for the review of the normalisation process. This function can only be used if you have retained the un-normalised `EMSet`. You can also review the expression of genes you are interested in; in this case, we will look at the expression of GAPDH and MALAT1 as they are considered 'housekeeping' genes.

```
norm.qc <- PlotNormalisationQC(original = em.set, normalised = norm.set,
                               gene.list = c("GAPDH", "MALAT1"))
```

The first set of graphs are library size histograms. The `scranNormalise` affects library size to a greater extent than the `NormaliseByRLE`.

```r
grid.arrange(norm.qc$Libsize$Original, norm.qc$Libsize$Normalised, ncol = 1)
```

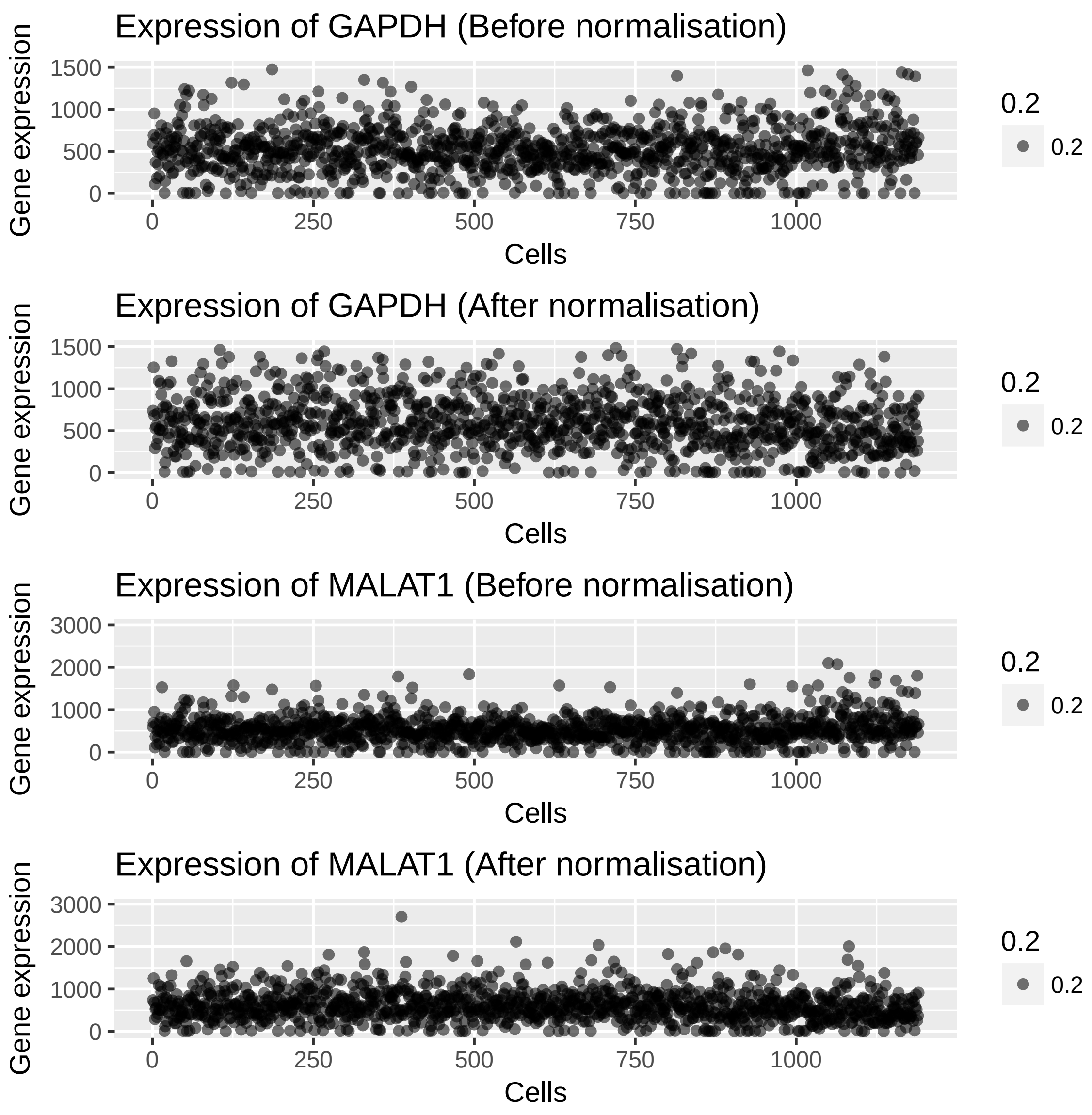


The gene scatter plots show how expression has changed on a gene level. Both genes are strongly expressed in this dataset, and normalisation has enabled us to make a clearer distinction between the expression level of these genes between each cell.

```r
grid.arrange(norm.qc$GeneScatterPlots$GAPDH$Original,
             norm.qc$GeneScatterPlots$GAPDH$Normalised,
             norm.qc$GeneScatterPlots$MALAT1$Original,
             norm.qc$GeneScatterPlots$MALAT1$Normalised,
             ncol = 1)
```

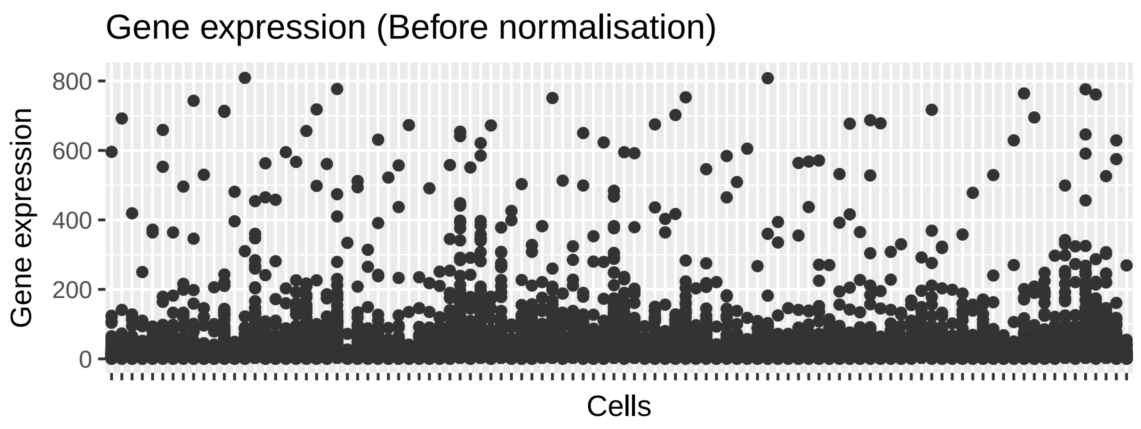
```
## Warning: Removed 18 rows containing missing values (geom_point).
```

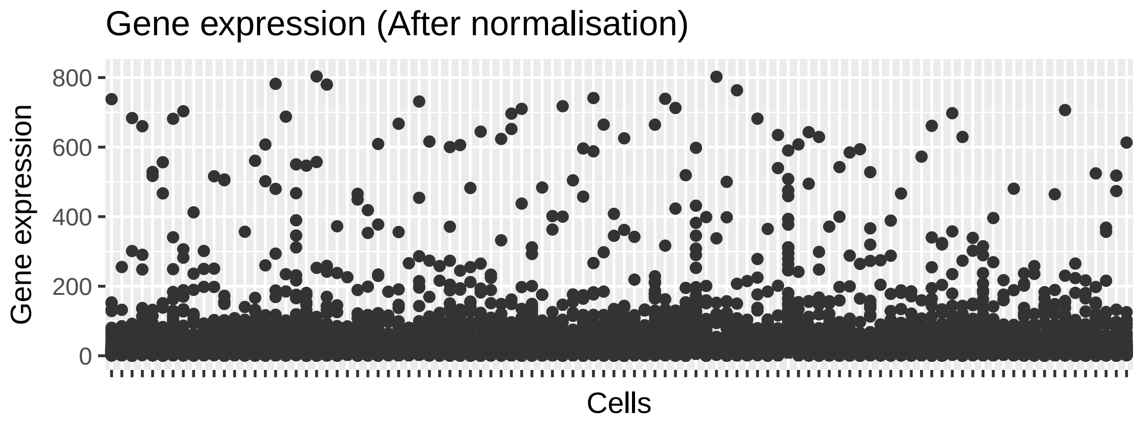
```
## Warning: Removed 24 rows containing missing values (geom_point).
```



The changes to overall gene expression can also be reviewed on gene expression boxplots.

```r
grid.arrange(norm.qc$GeneExpressionBoxplot$Original,
             norm.qc$GeneExpressionBoxplot$Normalised, ncol = 1)
```



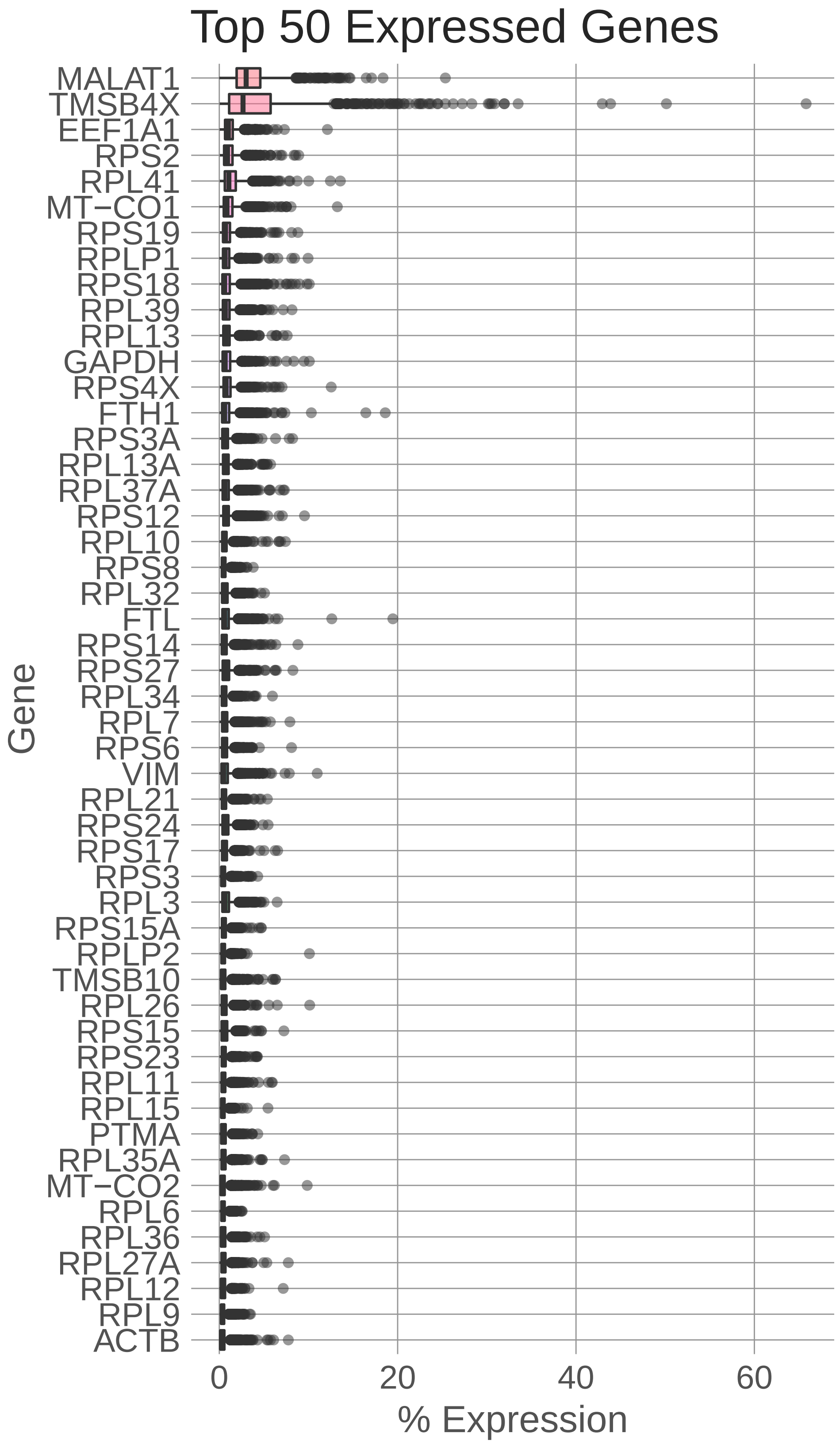


Control Removal

We can review the genes that dominate expression with the `PlotTopGeneExpression` function. This function gets called by the `PlotGeneralQC` function as well.

Let's review the plot generated by the `PlotGeneralQC` function after filtering.

```r
print(filtered.qc.plots$TopGenes)
```

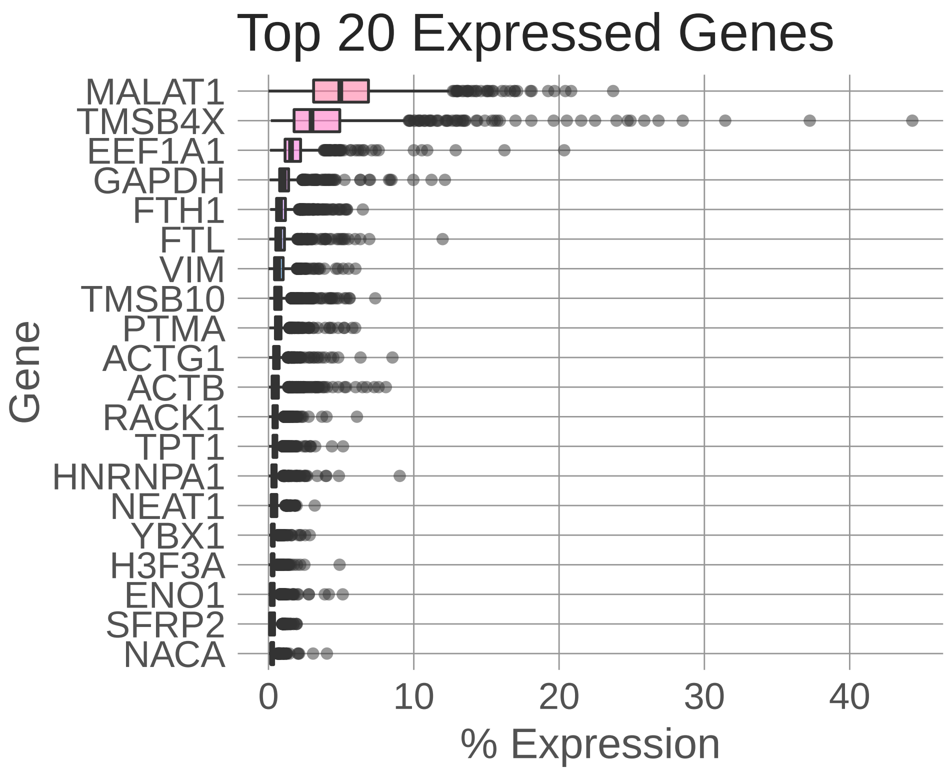


As you can see, ribosomal genes dominate gene expression, even after filtering. What does the dataset look like without these control genes? We will just plot the top 20 most expressed genes.

```
top.20.plot <- PlotTopGeneExpression(norm.set, n = 20, controls = FALSE)
```

```
## [1] "Calculating control metrics..."
```

```
print(top.20.plot)
```



As we are interested in the expression of non-control genes, we will need to remove the controls from the dataset. This can be done with the `ExcludeControl` function.

```
norm.set <- ExcludeControl(norm.set, "Mt")
norm.set <- ExcludeControl(norm.set, "Rb")
```

Please note that this has already been done as a part of the `scranNormalise` process.

Regression of Counfounding Factors

If we suspect there are transcripts that would bias the data and are not relevent to our analysis, we can regress them out with the `RegressConfoundingFactors` function.

```
cell.cycle.genes <- c("CDK4","CCND1","NOC2L","ATAD3C", "CCNL2")
em.set <- RegressConfoundingFactors(em.set, candidate.genes = cell.cycle.genes)
```

Dimension Reduction

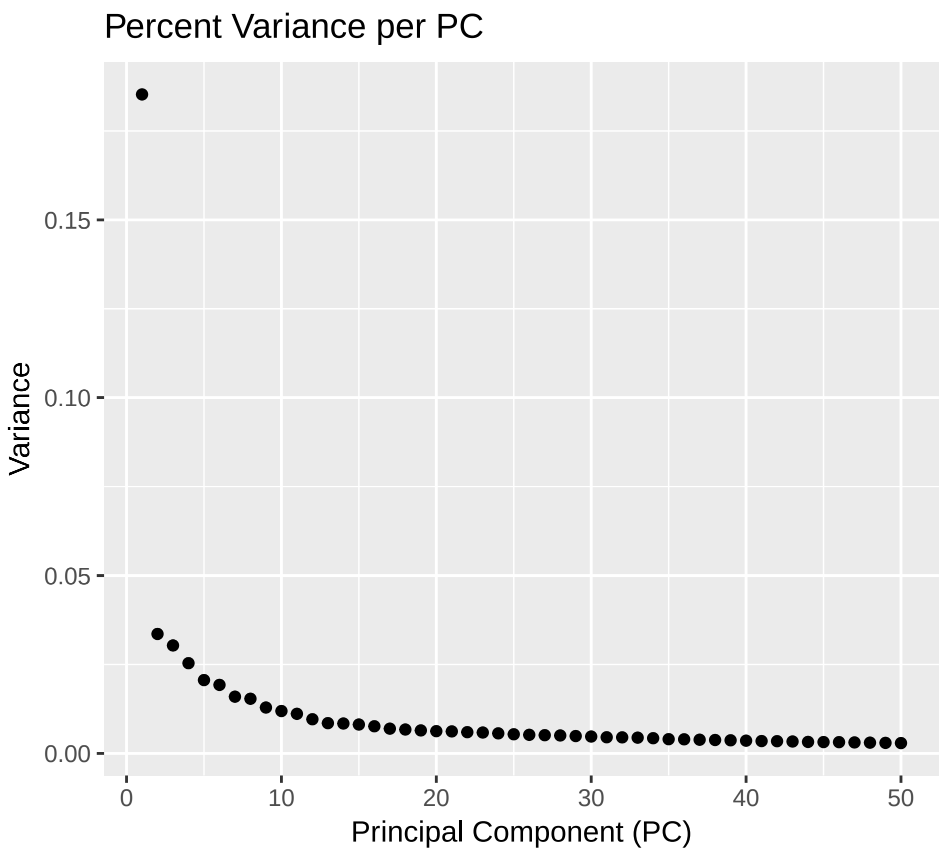
We have filtered our dataset down to 1235 cells and 32904 genes and normalised the transcript counts with `scranNormalise`. We can reduce this dataset further by using *Principal Component Analysis (PCA)* to identify genes that are major contributors to variation.

```
pca.set <- RunPCA(norm.set)
```

```
## [1] "Retrieving data..."
## [1] "Calculating variance..."
## [1] "Computing PCA values..."
## [1] "PCA complete! Returning object..."
```

PlotPCAVariance‘ generates what is known as a *scree plot*, which depicts what percentage each PC contributes to the total variance of the data. This will help determine how many PCs the dataset should be reduced to.

```
pca.variance <- PlotPCAVariance(pca.set, n = 50)
print(pca.variance)
```



The scree plot shows most of the variance is due to the top 20 PCs. Reduce the dataset to 20 PCs with the `ReduceDimensions` function.

```
pca.set <- ReduceDimensions(pca.set, n = 20)
```

Clustering

Clustering can be done on the original expression matrix or the PCA-transformed matrix, which is the preferred input. Use `RunCORE` to identify clusters. You can use the following arguments to finetune the algorithm.

Argument	Description
conservative	Use conservative (more stable) clustering result (TRUE or FALSE). Default: TRUE
nres	Number of resolutions to test Default: 40
remove_outlier	Remove cells that weren’t assigned a cluster with dynamicTreeCut.
This is indicati	ve of outlier cells within the sample. Default: TRUE

The RunCORE function generates a distance matrix based on the input and from this, builds a dendrogram. This dendrogram is then cut with the DynamicTreeCut algorithm to select clusters from the dendrogram based on the shape and size of the branches. This is repeated again, but this time with the tree-height parameter set to the chosen number of resolutions values ranging from 0 (the bottom of the tree) to 1 (the top of the tree).

```
## [1] "Performing unsupervised clustering..."
## [1] "Generating clusters by running dynamicTreeCut at different heights..."
```

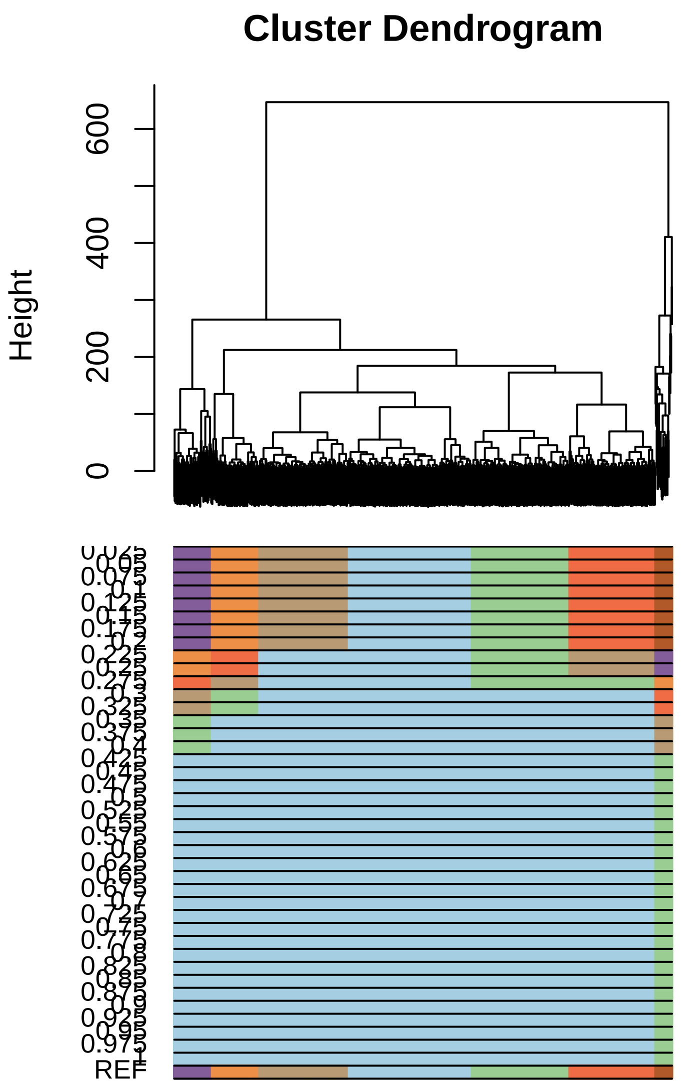
```
## Warning in split.default(windows, 1:nworkers): data length is not a
## multiple of split variable
```

```
##
  |
  |                                                                  |   0%
  |
  |======================                                            |  33%
  |
  |============================================                      |  67%
  |
  |==================================================================| 100%
##
## [1] "Calculating rand indices..."
## [1] "Calculating stability values..."
## [1] "Aggregating data..."
## [1] "Finding optimal number of clusters..."
## [1] "Optimal number of clusters found! Returning output..."
```

The PlotStabilityDendro generates a plot that represents this part of the process. In addition to the dendrogram, it generates the distribution of clusters across the 40 cut heights.

```
PlotStabilityDendro(clustered.set)
```

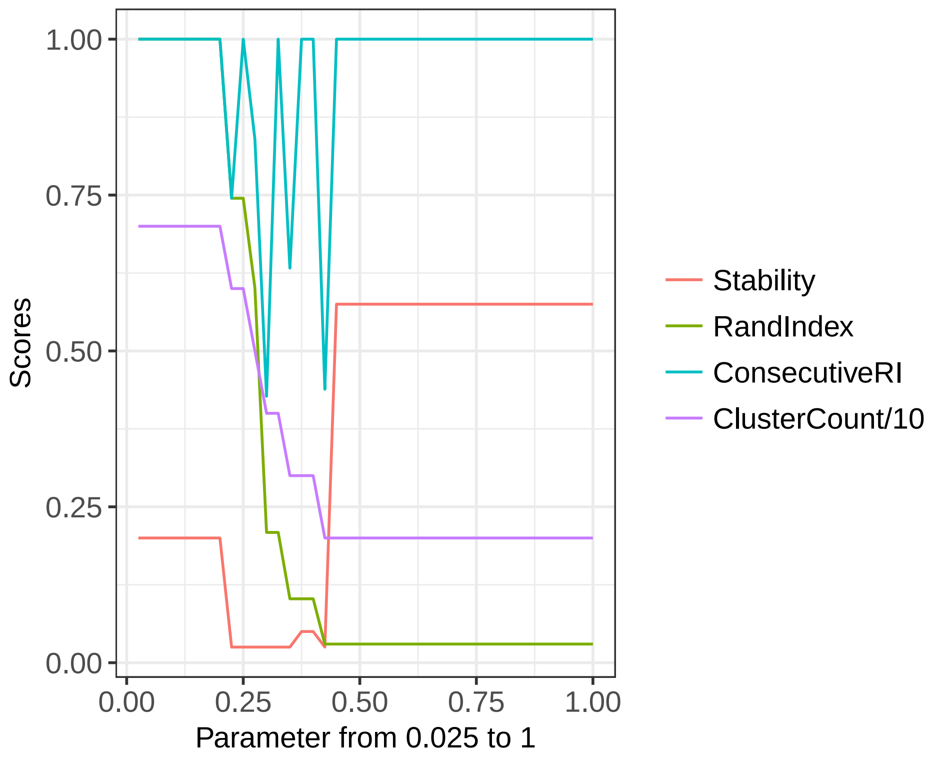
```
## $mar
## [1] 1 5 0 1
```



The clustering results are then compared quantitatively using rand indices, which calculates every pair of cells being in the same cluster or not. It is used as an indicator of the stability of a clustering result. If a rand index is stable across multiple tree-height values, this indicates the tree-height produces the most stable clustering result.

This information is shown on a plot generated by the `PlotStability` function.

```
PlotStability(clustered.set)
```



The rand index is stable in more than 50% of tree-cut heights that correspond to the lowest number of clusters. This indicates that 2 clusters is the most stable cluster assignment.

You can review this information in tabular form by using `GetRandMatrix`.

```r
rand.matrix <- GetRandMatrix(clustered.set)
kable(rand.matrix, digits = 3)
```

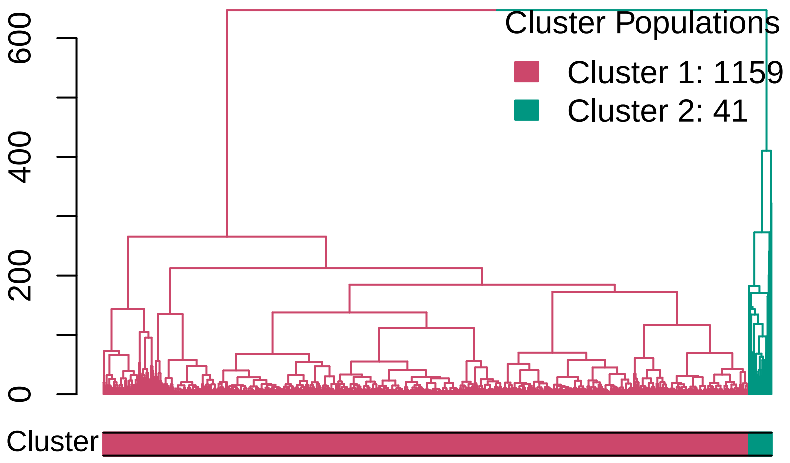
Height	Stability	RandIndex	ConsecutiveRI	ClusterCount
0.025	0.200	1.000	1.000	7
0.05	0.200	1.000	1.000	7
0.075	0.200	1.000	1.000	7
0.1	0.200	1.000	1.000	7
0.125	0.200	1.000	1.000	7
0.15	0.200	1.000	1.000	7
0.175	0.200	1.000	1.000	7
0.2	0.200	1.000	1.000	7
0.225	0.025	0.745	0.745	6
0.25	0.025	0.745	1.000	6
0.275	0.025	0.601	0.840	5
0.3	0.025	0.209	0.427	4
0.325	0.025	0.209	1.000	4
0.35	0.025	0.102	0.633	3
0.375	0.050	0.102	1.000	3
0.4	0.050	0.102	1.000	3
0.425	0.025	0.030	0.439	2
0.45	0.575	0.030	1.000	2
0.475	0.575	0.030	1.000	2
0.5	0.575	0.030	1.000	2
0.525	0.575	0.030	1.000	2

Height	Stability	RandIndex	ConsecutiveRI	ClusterCount
0.55	0.575	0.030	1.000	2
0.575	0.575	0.030	1.000	2
0.6	0.575	0.030	1.000	2
0.625	0.575	0.030	1.000	2
0.65	0.575	0.030	1.000	2
0.675	0.575	0.030	1.000	2
0.7	0.575	0.030	1.000	2
0.725	0.575	0.030	1.000	2
0.75	0.575	0.030	1.000	2
0.775	0.575	0.030	1.000	2
0.8	0.575	0.030	1.000	2
0.825	0.575	0.030	1.000	2
0.85	0.575	0.030	1.000	2
0.875	0.575	0.030	1.000	2
0.9	0.575	0.030	1.000	2
0.925	0.575	0.030	1.000	2
0.95	0.575	0.030	1.000	2
0.975	0.575	0.030	1.000	2
1	0.575	0.030	1.000	2

The `PlotDendrogram` function generates a dendrogram that depicts each cluster and its members.

```
PlotDendrogram(clustered.set)
```

```
## Warning in `labels<-.dendrogram`(dend, value = value, ...): The lengths
## of the new labels is shorter than the number of leaves in the dendrogram -
## labels are recycled.
```



The cluster information has been added as a new column in the Cell Information slot, which can be retrieved with the `GetCellInfo` function.

```
cell.info <- GetCellInfo(clustered.set)
cell.info[1:5,]
```

```
##          cell_barcode batch THY1  BRN3 phase cluster
## 1 AAACCTGAGCTGTTCA-1     1 TRUE FALSE    G1       1
## 2 AAACCTGCAATTCCTT-1     1 TRUE FALSE     S       1
## 3 AAACCTGGTCTACCTC-1     1 TRUE FALSE     S       1
## 4 AAACCTGTCGGAGCAA-1     1 TRUE FALSE    G1       1
## 5 AAACGGGAGTCGATAA-1      1 TRUE FALSE     G1        1
```

Differential Expression

This package uses `DESeq` to perform differential expression, and can be done with or without clustering. Each cell needs to be assigned one of two conditions; for this tutorial, we will use batch information and clustering information. As this step is computationally intensive for larger datasets, you can restrict analysis to a specified number of the most variable genes by using the `ngenes` argument.

The `RunDiffExpression` calls `DESeq` to perform differential expression between two conditions. This function can be run with or without clustering, after PCA reduction. If this function is unable to fit your data, you may adjust the arguments `method` and `fitType`. These arguments are for `DESeq`'s `estimateDispersions` function. If your dataset is very small, you should also reduce the number of genes being analysed with the `ngenes` function.

First, let's compare the expression of THY1-positive cells to THY1-negative cells.

```
thy1.de.result <- RunDiffExpression(clustered.set,
                                    condition.a = "TRUE",
                                    condition.b = "FALSE",
                                    conditions = "THY1",
                                    fitType = "local",
                                    method = "per-condition",
                                    ngenes = NULL)
```

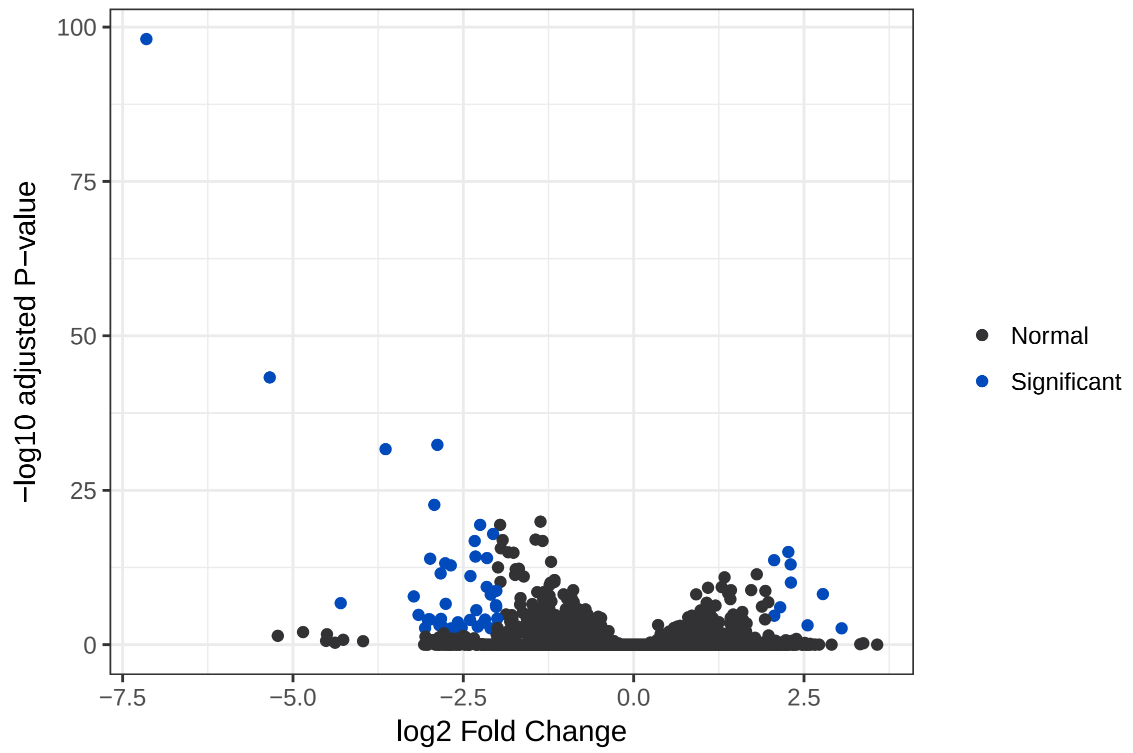
```
kable(thy1.de.result[1:10,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
8572	MGP	1.909	1.041	6.830	142.183	-7.152	0	0
9197	COL3A1	1.570	1.082	4.333	40.529	-5.341	0	0
9716	LGALS1	4.961	3.026	15.925	7.365	-2.881	0	0
18062	CTGF	2.737	1.638	8.964	12.474	-3.641	0	0
5654	TGM2	1.756	1.380	3.886	7.599	-2.926	0	0
18429	ITGB1	4.580	3.894	8.466	2.580	-1.367	0	0
11478	TPM1	10.780	7.823	27.542	3.890	-1.960	0	0
20941	COL1A1	3.978	2.903	10.069	4.767	-2.253	0	0
12769	TNFRSF12A	2.124	1.761	4.181	4.178	-2.063	0	0
7095	FSTL1	2.569	2.248	4.387	2.714	-1.440	0	0

The results are sorted in ascending order, based on the p-value. The fold change values have been adjusted; they represent absolute fold change.

We can view these results as a volcano plot with the `PlotDEVolcano` function.

```
thy1.volcano.plot <- PlotDEVolcano(thy1.de.result, labels = FALSE)
print(thy1.volcano.plot)
```



Let's examine what genes are differentially expressed between clusters 1 and 2.

```
cluster.de.result <- RunDiffExpression(clustered.set,
                                       condition.a = "1",
                                       condition.b = "2",
                                       condition = "cluster",
                                       fitType = "local",
                                       method = "per-condition")
```

```
sig.de.result <- cluster.de.result[which(cluster.de.result$padj < 0.05), ]
sig.de.result <- sig.de.result[order(-abs(sig.de.result$log2FoldChange)),]
kable(sig.de.result[1:20, ], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
7529	DCT	8.465	8.724	1.152	0.020	5.666	0.000	0.000
18494	RSRP1	2.525	2.578	1.032	0.020	5.641	0.002	0.030
15233	AKAP9	2.899	2.963	1.079	0.040	4.638	0.001	0.019
21985	RASGRP2	1.107	1.067	2.239	18.603	-4.217	0.002	0.022
11038	S100A6	1.565	1.361	7.344	17.584	-4.136	0.000	0.003
7870	PYGM	1.132	1.091	2.281	14.078	-3.815	0.002	0.032
278	PVALB	1.178	1.125	2.680	13.435	-3.748	0.003	0.043
20719	DHRS4	1.362	1.264	4.143	11.924	-3.576	0.000	0.000
5687	RP11-197K6.1	1.184	1.140	2.437	10.285	-3.362	0.003	0.035
1711	LEF1	1.238	1.184	2.775	9.672	-3.274	0.000	0.000
15661	CTC-524C5.2	1.310	1.240	3.291	9.548	-3.255	0.000	0.000
3350	TMEM53	1.417	1.324	4.038	9.368	-3.228	0.000	0.000
5884	LINC00941	1.160	1.125	2.134	9.036	-3.176	0.004	0.047
9987	ACOT8	1.390	1.307	3.739	8.914	-3.156	0.000	0.000
4095	NKTR	3.282	3.353	1.274	0.116	3.102	0.001	0.017
20099	KCNQ1OT1	6.922	7.106	1.712	0.117	3.101	0.000	0.000

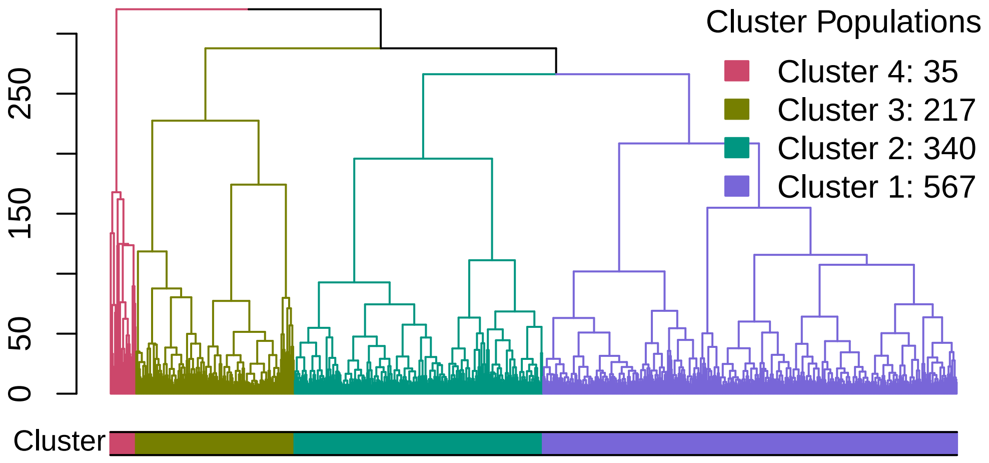
	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
19901	FBXL8	1.139	1.111	1.937	8.458	-3.080	0.004	0.042
10194	CCDC58	1.416	1.335	3.697	8.044	-3.008	0.000	0.000
21910	PINX1	1.194	1.157	2.235	7.859	-2.974	0.000	0.005
974	NME2	1.262	1.213	2.653	7.754	-2.955	0.001	0.017

These results underwent further analysis, and revealed cells in cluster 2 were strongly expressing apoptopic genes. The cells in this cluster were deemed 'low quality' and removed from the dataset. To confirm that the remaining cells were good quality, the dataset re-clustered.

```r
clean.set <- SubsetCluster(clustered.set, clusters = "1")
clean.pca <- RunPCA(clean.set)
clean.cluster <- RunCORE(clean.pca, conservative = TRUE)
```

```r
PlotDendrogram(clean.cluster)
```

```
## Warning in `labels<-.dendrogram`(dend, value = value, ...): The lengths
## of the new labels is shorter than the number of leaves in the dendrogram -
## labels are recycled.
```



Reclustering and differential expression revealed the remaining 1159 cells comprised of four subpopulations, each representing retinal ganglion cells at different stages of differentiation.

Comparing each cluster against all other clusters

The `RunDiffExpression` can be called to run multiple comparisons at once, using standard R functions such as `lapply` and `sapply`. The use of "Others" as condition.b tells the function to compare cells that have condition.a to all other cells that don't. If you need to do a lot of comparisons, you can even use BiocParallel's `bplapply` to run this function.

```r
# List of clusters to compare
cluster.list <- c("1", "2", "3", "4")

# Create a custom function to call RunDiffExpression
customFunction <- function(x, clean.cluster){
  # This is a standard RunDiffExpression call; The only difference is "x" will
  # be inputted by the sapply function
  de.result <- RunDiffExpression(clean.cluster,
                                 condition.a = x,
                                 condition.b = "Others",
                                    conditions = "cluster")
  # This will output the differential expression result as a list of dataframes
  return (de.result)
}

clean.cluster.de.results <- lapply(cluster.list, function(x)
  customFunction(x, clean.cluster))

# Generate volcano plots
cluster.de.1 <- clean.cluster.de.results[[1]]
cluster.de.2 <- clean.cluster.de.results[[2]]
cluster.de.3 <- clean.cluster.de.results[[3]]
cluster.de.4 <- clean.cluster.de.results[[4]]

# Format DE results
cluster.de.1 <- cluster.de.1[which(cluster.de.1$padj < 0.05), ]
cluster.de.1 <- cluster.de.1[order(-abs(cluster.de.1$log2FoldChange)),]

cluster.de.2 <- cluster.de.2[which(cluster.de.2$padj < 0.05), ]
cluster.de.2 <- cluster.de.2[order(-abs(cluster.de.2$log2FoldChange)),]

cluster.de.3 <- cluster.de.3[which(cluster.de.3$padj < 0.05), ]
cluster.de.3 <- cluster.de.3[order(-abs(cluster.de.3$log2FoldChange)),]

cluster.de.4 <- cluster.de.4[which(cluster.de.4$padj < 0.05), ]
cluster.de.4 <- cluster.de.4[order(-abs(cluster.de.4$log2FoldChange)),]
```

Cluster 1 vs Other Clusters

```
kable(cluster.de.1[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
13333	FDCSP	1.339	1.026	1.638	24.216	-4.598	0.000	0.000
4492	MGP	1.939	1.086	2.756	20.414	-4.351	0.000	0.000
16101	TYRP1	1.691	1.074	2.282	17.355	-4.117	0.000	0.000
1948	DCT	8.703	1.928	15.191	15.285	-3.934	0.000	0.000
17866	CSN3	1.211	1.029	1.385	13.492	-3.754	0.000	0.001
17081	HTN1	1.463	1.070	1.840	12.035	-3.589	0.000	0.000
8194	ACTG2	1.205	1.034	1.368	10.897	-3.446	0.000	0.001
552	LUM	1.143	1.024	1.256	10.582	-3.404	0.002	0.023
1383	ATP6V1C2	1.788	1.148	2.401	9.458	-3.242	0.000	0.000
12384	COL8A2	1.144	1.028	1.256	9.164	-3.196	0.000	0.005
19864	IGFBP7	5.819	1.944	9.531	9.034	-3.175	0.000	0.000
15924	NCCRP1	1.320	1.064	1.565	8.880	-3.151	0.000	0.000
17012	KRT17	1.984	1.209	2.726	8.265	-3.047	0.000	0.000
11258	KRT7	2.424	1.311	3.491	8.019	-3.003	0.000	0.000
12290	ELN	2.656	1.367	3.890	7.869	-2.976	0.000	0.000
20073	COL8A1	1.360	1.081	1.627	7.743	-2.953	0.000	0.000
7357	LGALS1	4.877	1.891	7.738	7.566	-2.920	0.000	0.000
2916	PTGDS	3.571	1.596	5.462	7.483	-2.904	0.000	0.000
15858	CTGF	2.795	1.417	4.114	7.463	-2.900	0.000	0.000
13282	DCX	1.247	1.443	1.060	0.135	2.894	0.000	0.000

Cluster 2 vs Other Clusters

```
kable(cluster.de.2[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
4492	MGP	1.939	1.013	2.323	99.466	-6.636	0.000	0.000
3423	S100A9	1.802	1.034	2.121	32.796	-5.035	0.000	0.000
17081	HTN1	1.463	1.029	1.644	22.308	-4.480	0.000	0.000
20340	S100A14	1.208	1.016	1.287	18.289	-4.193	0.001	0.012
17786	COL3A1	1.595	1.047	1.823	17.618	-4.139	0.000	0.000
13333	FDCSP	1.339	1.030	1.467	15.679	-3.971	0.000	0.000
1948	DCT	8.703	1.777	11.578	13.612	-3.767	0.000	0.000
7993	CPLX3	1.226	1.024	1.310	12.889	-3.688	0.000	0.002
17012	KRT17	1.984	1.113	2.345	11.885	-3.571	0.000	0.000
8194	ACTG2	1.205	1.025	1.279	11.260	-3.493	0.001	0.025
6061	C5orf46	1.184	1.025	1.251	9.942	-3.314	0.001	0.031
17866	CSN3	1.211	1.029	1.286	9.907	-3.308	0.001	0.032
15374	TINAGL1	1.228	1.033	1.308	9.227	-3.206	0.001	0.011
16101	TYRP1	1.691	1.108	1.933	8.673	-3.116	0.000	0.000
1383	ATP6V1C2	1.788	1.123	2.064	8.671	-3.116	0.000	0.000
11258	KRT7	2.424	1.251	2.911	7.604	-2.927	0.000	0.000
18022	CDKN2A	1.205	1.038	1.275	7.286	-2.865	0.002	0.040
21827	IGFBP3	1.738	1.140	1.987	7.060	-2.820	0.000	0.000
8648	ITIH5	1.191	1.037	1.255	6.926	-2.792	0.002	0.042
9582	FN1	2.905	1.369	3.542	6.881	-2.783	0.000	0.000

Cluster 3 vs Other Clusters

```
kable(cluster.de.3[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
4492	MGP	1.939	5.713	1.069	0.015	6.086	0.000	0.000
1948	DCT	8.703	38.408	1.860	0.023	5.444	0.000	0.000
13333	FDCSP	1.339	2.621	1.043	0.027	5.222	0.000	0.000
16101	TYRP1	1.691	4.304	1.089	0.027	5.220	0.000	0.000
15248	RP11-247C2.2	1.688	1.029	1.840	28.652	-4.841	0.000	0.000
21675	LEFTY2	3.919	1.127	4.562	28.116	-4.813	0.000	0.000
12384	COL8A2	1.144	1.648	1.028	0.044	4.508	0.000	0.000
3423	S100A9	1.802	1.044	1.977	22.003	-4.460	0.000	0.000
9361	DCN	1.085	1.373	1.018	0.049	4.358	0.001	0.010
12290	ELN	2.656	8.260	1.365	0.050	4.315	0.000	0.000
7993	CPLX3	1.226	1.989	1.050	0.051	4.299	0.000	0.000
1383	ATP6V1C2	1.788	4.433	1.179	0.052	4.264	0.000	0.000
6597	CALB1	3.658	1.170	4.232	19.025	-4.250	0.000	0.000
9271	OC90	1.071	1.302	1.017	0.057	4.143	0.000	0.007
2524	FOXN4	1.268	1.019	1.325	17.315	-4.114	0.000	0.004
21239	FZD5	2.072	1.075	2.302	17.283	-4.111	0.000	0.000
17684	TRPM1	1.694	3.952	1.174	0.059	4.081	0.000	0.000
7268	DKK2	1.096	1.409	1.024	0.059	4.075	0.004	0.045
1347	ASCL1	1.605	1.045	1.733	16.261	-4.023	0.000	0.000
15858	CTGF	2.795	8.560	1.467	0.062	4.018	0.000	0.000

Cluster 4 vs Other Clusters

```r
kable(cluster.de.4[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
1617	XAGE2	1.164	5.387	1.032	0.007	7.088	0.000	0.000
17043	SFN	1.177	4.959	1.060	0.015	6.053	0.000	0.000
3908	TTR	4.345	1.055	4.448	63.086	-5.979	0.000	0.000
3423	S100A9	1.802	17.910	1.301	0.018	5.813	0.000	0.000
10822	LINC00685	2.196	1.023	2.232	53.950	-5.754	0.001	0.021
82	RAB25	1.114	3.328	1.045	0.019	5.695	0.000	0.000
5959	CLDN7	1.117	3.103	1.055	0.026	5.252	0.000	0.001
20340	S100A14	1.208	4.536	1.104	0.029	5.086	0.000	0.000
21335	MIAT	7.265	1.195	7.454	33.028	-5.046	0.000	0.000
12577	LHX2	2.467	1.049	2.511	30.550	-4.933	0.000	0.001
8761	TUBA4A	1.085	2.329	1.046	0.034	4.860	0.001	0.043
11258	KRT7	2.424	22.897	1.787	0.036	4.798	0.000	0.000
15374	TINAGL1	1.228	4.416	1.128	0.038	4.735	0.000	0.000
15399	TTC14	2.407	1.056	2.449	25.755	-4.687	0.000	0.007
4857	LINC00632	2.286	1.052	2.324	25.295	-4.661	0.000	0.006
11459	KCNQ1OT1	7.098	1.278	7.279	22.561	-4.496	0.000	0.000
18752	TUG1	1.986	1.052	2.015	19.443	-4.281	0.001	0.043
20621	ALOX5AP	1.138	2.716	1.088	0.051	4.279	0.002	0.048
8505	EPCAM	1.223	3.697	1.146	0.054	4.212	0.000	0.000
12969	KRT18	6.058	54.648	4.544	0.066	3.920	0.000	0.000

Comparing pairs of clusters

We can also compare pairs of clusters by setting conditions A and B in the `RunDiffExpression` function.

```r
# Run differential expression on pairs
c1c2.de.results <- RunDiffExpression(clean.cluster, condition.a = "1",
                                     condition.b = "2", conditions = "cluster")
c1c3.de.results <- RunDiffExpression(clean.cluster, condition.a = "1",
                                     condition.b = "3", conditions = "cluster")
c1c4.de.results <- RunDiffExpression(clean.cluster, condition.a = "1",
                                     condition.b = "4", conditions = "cluster")
c2c3.de.results <- RunDiffExpression(clean.cluster, condition.a = "2",
                                     condition.b = "3", conditions = "cluster")
c2c4.de.results <- RunDiffExpression(clean.cluster, condition.a = "2",
                                     condition.b = "4", conditions = "cluster")
c3c4.de.results <- RunDiffExpression(clean.cluster, condition.a = "3",
                                     condition.b = "4", conditions = "cluster")

# Format DE results
c1c2.de.results <- c1c2.de.results[which(c1c2.de.results$padj < 0.05), ]
c1c2.de.results <- c1c2.de.results[order(-abs(c1c2.de.results$log2FoldChange)),]

c1c3.de.results <- c1c3.de.results[which(c1c3.de.results$padj < 0.05), ]
c1c3.de.results <- c1c3.de.results[order(-abs(c1c3.de.results$log2FoldChange)),]

c1c4.de.results <- c1c4.de.results[which(c1c4.de.results$padj < 0.05), ]
c1c4.de.results <- c1c4.de.results[order(-abs(c1c4.de.results$log2FoldChange)),]
```

```r
c2c3.de.results <- c2c3.de.results[which(c2c3.de.results$padj < 0.05), ]
c2c3.de.results <- c2c3.de.results[order(-abs(c2c3.de.results$log2FoldChange)),]

c2c4.de.results <- c2c4.de.results[which(c2c4.de.results$padj < 0.05), ]
c2c4.de.results <- c2c4.de.results[order(-abs(c2c4.de.results$log2FoldChange)),]

c3c4.de.results <- c3c4.de.results[which(c3c4.de.results$padj < 0.05), ]
c3c4.de.results <- c3c4.de.results[order(-abs(c3c4.de.results$log2FoldChange)),]
```

Cluster 1 vs Cluster 2

```r
kable(c1c2.de.results[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
20864	S100A9	1.367	1.564	1.037	0.065	3.935	0.000	0.000
8949	COL3A1	1.227	1.334	1.049	0.145	2.784	0.002	0.039
4575	STMN2	2.125	2.648	1.251	0.153	2.713	0.000	0.000
6119	HOXB5	1.295	1.425	1.078	0.183	2.447	0.000	0.006
2923	CDC20	1.607	1.234	2.231	5.271	-2.398	0.000	0.000
17648	DCX	1.315	1.452	1.088	0.194	2.364	0.000	0.001
15233	TTR	3.929	2.176	6.852	4.975	-2.315	0.002	0.041
427	CCNB1	1.936	1.393	2.842	4.690	-2.230	0.000	0.000
3047	NEK2	1.329	1.160	1.610	3.811	-1.930	0.000	0.000
14444	CCNB2	1.748	1.380	2.363	3.588	-1.843	0.000	0.000
2084	TUBB4B	3.238	2.187	4.991	3.363	-1.750	0.000	0.000
19271	PLK1	1.435	1.234	1.770	3.288	-1.717	0.000	0.000
17985	IGFBP3	1.330	1.443	1.141	0.318	1.651	0.001	0.021
7258	KIF20A	1.248	1.138	1.431	3.129	-1.646	0.000	0.005
15180	PTTG1	5.132	3.301	8.186	3.123	-1.643	0.000	0.000
19535	PDIA3	3.806	2.572	5.863	3.093	-1.629	0.000	0.000
20679	DKK1	1.351	1.198	1.607	3.068	-1.617	0.000	0.006
17811	MAD2L1	2.198	1.676	3.069	3.060	-1.613	0.000	0.000
7890	PGK1	11.528	7.024	19.037	2.994	-1.582	0.000	0.000
14694	ANXA5	3.829	2.641	5.810	2.931	-1.551	0.000	0.000

Cluster 1 vs Cluster 3

```r
kable(c1c3.de.results[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
18282	MGP	2.377	1.092	5.734	51.309	-5.681	0.000	0.000
17651	FDCSP	1.478	1.035	2.636	46.415	-5.537	0.000	0.000
11467	TYRP1	1.969	1.075	4.307	44.359	-5.471	0.000	0.000
17803	DCT	12.113	1.941	38.691	40.044	-5.324	0.000	0.000
12636	CSN3	1.256	1.027	1.856	31.990	-5.000	0.000	0.000
14443	RP11-247C2.2	1.690	1.942	1.032	0.034	4.879	0.000	0.000
4263	HTN1	1.496	1.067	2.616	24.100	-4.591	0.000	0.000
14926	ATP6V1C2	2.070	1.156	4.460	22.175	-4.471	0.000	0.000
13015	NCCRP1	1.413	1.063	2.328	21.151	-4.403	0.000	0.000
12634	IGFBP7	6.938	1.949	19.973	19.989	-4.321	0.000	0.000
18140	LEFTY2	2.887	3.559	1.130	0.051	4.297	0.000	0.000

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
2748	ELN	3.295	1.376	8.309	19.428	-4.280	0.000	0.000
5966	FZD5	2.090	2.478	1.078	0.053	4.240	0.000	0.000
8727	COL8A2	1.209	1.036	1.660	18.184	-4.185	0.000	0.000
10471	CTGF	3.391	1.416	8.551	18.146	-4.182	0.000	0.000
17332	TRPM1	1.935	1.164	3.952	18.015	-4.171	0.000	0.000
13759	ACTG2	1.213	1.038	1.672	17.848	-4.158	0.000	0.000
17135	COL8A1	1.481	1.086	2.511	17.475	-4.127	0.000	0.000
16698	DCN	1.121	1.022	1.380	17.310	-4.114	0.001	0.012
10755	OC90	1.097	1.018	1.305	17.236	-4.107	0.001	0.013

Cluster 1 vs Cluster 4

```r
kable(c1c4.de.results[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
17299	KRT7	2.582	1.318	23.057	69.377	-6.116	0.000	0.000
6692	XAGE2	1.320	1.065	5.455	68.895	-6.106	0.000	0.000
14338	SFN	1.287	1.059	4.980	67.993	-6.087	0.000	0.000
4894	HTN1	1.292	1.071	4.873	54.649	-5.772	0.000	0.000
20414	LINC00685	2.482	2.572	1.029	0.018	5.769	0.000	0.001
17105	KRT17	1.764	1.208	10.768	46.925	-5.552	0.000	0.000
8816	MIAT	9.901	10.438	1.204	0.022	5.532	0.000	0.000
16323	ALOX5AP	1.138	1.039	2.745	44.545	-5.477	0.000	0.011
12246	ACTG2	1.153	1.045	2.905	42.260	-5.401	0.000	0.002
12947	BMPR1B	1.973	2.032	1.027	0.026	5.264	0.001	0.037
18752	ZNF397	2.075	2.139	1.032	0.028	5.166	0.001	0.029
13985	RAB25	1.201	1.067	3.365	35.081	-5.133	0.000	0.000
4922	TINAGL1	1.294	1.101	4.413	33.675	-5.074	0.000	0.000
10971	ANKRD26	1.958	2.015	1.033	0.032	4.962	0.001	0.030
3435	TTC14	2.660	2.759	1.057	0.032	4.957	0.000	0.000
6357	LHX2	2.797	2.904	1.062	0.033	4.932	0.000	0.000
7261	S100A9	2.515	1.562	17.955	30.152	-4.914	0.000	0.000
11297	KRT18	5.922	2.879	55.222	28.853	-4.851	0.000	0.000
10510	LINC00632	2.642	2.740	1.062	0.035	4.820	0.000	0.001
10074	CLDN7	1.194	1.077	3.095	27.186	-4.765	0.000	0.000

Cluster 2 vs Cluster 3

```r
kable(c2c3.de.results[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
18640	MGP	2.871	1.023	5.768	208.342	-7.703	0.000	0.000
15611	HTN1	1.655	1.033	2.629	49.839	-5.639	0.000	0.000
11400	DCT	16.115	1.784	38.569	47.927	-5.583	0.000	0.000
13230	COL3A1	1.869	1.050	3.153	43.485	-5.442	0.000	0.000
1005	FDCSP	1.663	1.038	2.642	43.112	-5.430	0.000	0.000
12779	LEFTY2	4.444	6.557	1.132	0.024	5.397	0.000	0.000
5069	CPLX3	1.409	1.030	2.003	33.662	-5.073	0.000	0.000
7591	TYRP1	2.356	1.109	4.309	30.324	-4.922	0.000	0.000
1239	CSN3	1.355	1.031	1.862	28.077	-4.811	0.000	0.000

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
12278	C5orf46	1.278	1.025	1.674	26.875	-4.748	0.000	0.000
5125	CALB1	3.927	5.682	1.177	0.038	4.728	0.000	0.000
760	ATP6V1C2	2.432	1.131	4.469	26.453	-4.725	0.000	0.000
16810	KRT17	2.216	1.112	3.946	26.375	-4.721	0.000	0.000
17693	ACTG2	1.281	1.029	1.676	23.042	-4.526	0.000	0.000
6625	DKK2	1.174	1.020	1.417	21.177	-4.404	0.002	0.037
7751	CRYBB2	1.187	1.022	1.445	19.972	-4.320	0.003	0.037
19398	RP11-247C2.2	1.475	1.755	1.038	0.050	4.313	0.000	0.000
20125	CENPA	1.686	2.088	1.055	0.051	4.294	0.000	0.000
14261	FN1	4.053	1.375	8.250	19.357	-4.275	0.000	0.000
6023	ELN	4.073	1.380	8.293	19.179	-4.261	0.000	0.000

Cluster 2 vs Cluster 4

```r
kable(c2c4.de.results[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
9080	S100A9	2.629	1.043	18.042	400.261	-8.645	0.000	0.000
5877	XAGE2	1.438	1.021	5.488	211.461	-7.724	0.000	0.000
15379	S100A14	1.358	1.025	4.587	142.253	-7.152	0.000	0.000
12634	SFN	1.400	1.029	5.002	137.591	-7.104	0.000	0.000
6765	HTN1	1.395	1.034	4.905	114.350	-6.837	0.000	0.000
13951	RAB25	1.249	1.028	3.400	86.724	-6.438	0.000	0.000
2619	TTR	6.334	6.876	1.068	0.012	6.431	0.000	0.000
14636	KRT7	3.321	1.265	23.295	84.039	-6.393	0.000	0.000
8282	TINAGL1	1.361	1.043	4.451	81.165	-6.343	0.000	0.000
18567	KRT17	2.034	1.122	10.887	80.738	-6.335	0.000	0.000
9441	GABRP	1.154	1.020	2.456	73.507	-6.200	0.000	0.025
14839	CXCL8	1.193	1.025	2.829	73.012	-6.190	0.001	0.022
11556	CLDN7	1.224	1.029	3.115	72.312	-6.176	0.000	0.000
97	ACTG2	1.212	1.038	2.906	50.754	-5.665	0.000	0.001
15227	ALOX5AP	1.199	1.037	2.768	47.688	-5.576	0.000	0.010
3036	TFPI2	1.523	1.116	5.476	38.567	-5.269	0.000	0.000
5506	CDKN2B	1.156	1.036	2.317	36.476	-5.189	0.000	0.003
6156	C19orf33	1.329	1.085	3.699	31.729	-4.988	0.000	0.000
14852	TUBA4A	1.172	1.049	2.368	27.905	-4.802	0.000	0.022
9673	LHX2	2.469	2.614	1.058	0.036	4.798	0.000	0.002

Cluster 3 vs Cluster 4

```r
kable(c3c4.de.results[1:20,], digits = 3)
```

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
358	S100A9	3.416	1.051	18.078	333.960	-8.384	0.000	0.000
9340	XAGE2	1.648	1.028	5.495	160.144	-7.323	0.000	0.000
6424	TTR	5.914	6.696	1.065	0.011	6.457	0.000	0.000
2881	DCT	33.629	38.800	1.565	0.015	6.064	0.000	0.000
3339	GABRP	1.227	1.026	2.469	55.474	-5.794	0.000	0.007
9820	COL9A3	2.377	2.594	1.033	0.021	5.580	0.000	0.001
9339	CLDN7	1.336	1.047	3.130	45.404	-5.505	0.000	0.000

	id	baseMean	baseMeanA	baseMeanB	foldChange	log2FoldChange	pval	padj
4164	RAB25	1.381	1.060	3.375	39.874	-5.317	0.000	0.000
2644	SFN	1.664	1.129	4.980	30.757	-4.943	0.000	0.000
6179	KCNQ1OT1	8.785	9.993	1.297	0.033	4.921	0.000	0.000
3550	ELN	7.396	8.387	1.247	0.033	4.905	0.000	0.000
17280	TYRP1	3.894	4.335	1.157	0.047	4.405	0.000	0.000
13348	CXCL8	1.329	1.087	2.829	21.053	-4.396	0.000	0.010
14140	S100A14	1.663	1.185	4.626	19.577	-4.291	0.000	0.000
12356	CALB1	1.591	1.180	4.143	17.494	-4.129	0.000	0.000
8571	TTC14	2.060	2.220	1.073	0.060	4.067	0.001	0.029
7826	TUBA4A	1.266	1.087	2.374	15.714	-3.974	0.001	0.019
15260	TUG1	1.898	2.032	1.066	0.064	3.973	0.001	0.038
5217	FABP7	1.878	1.292	5.510	15.465	-3.951	0.000	0.000
1467	AHNAK2	2.326	2.523	1.105	0.069	3.853	0.000	0.006

References

Maciej Daniszewski, Anne Senabouth, Quan Nguyen, Duncan E Crombie, Samuel W Lukowski, Tejal Kulkarni, Donald J Zack, Alice Pebay, Joseph E Powell, Alex Hewitt, Single Cell RNA Sequencing of stem cell-derived retinal ganglion cells. bioRxiv 191395; doi: https://doi.org/10.1101/191395

Lun ATL, McCarthy DJ and Marioni JC. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor [version 2; referees: 3 approved, 2 approved with reservations]. F1000Research 2016, 5:2122 (doi: 10.12688/f1000research.9501.2)

Zheng, G. X. Y. et al. Massively parallel digital transcriptional profiling of single cells. Nat. Commun. 8, 14049 doi: 10.1038/ncomms14049 (2017).

McCarthy DJ, Campbell KR, Lun ATL and Wills QF (2017). “Scater: pre-processing, quality control, normalisation and visualisation of single-cell RNA-seq data in R.” Bioinformatics, 14 Jan. doi: 10.1093/bioinformatics/btw777, http://dx.doi.org/10.1093/bioinformatics/btw777.

Lun ATL, McCarthy DJ and Marioni JC (2016). “A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor.” F1000Res., 5, pp. 2122.