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

Anne Senabouth

2018-01-09

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

Preparing data manually

The data for this vignette is included with the package. You can load the data by using the following command:

```
load(system.file("extdata", "RGC_scRNASeq.RData", package = "ascend"))
```

These objects contain all the information we need to create an `EMSet` - the main data object of the ascend package.

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

```
##   V1 V2 V3 V4 V5
## 1  0  0  0  0  0
## 2  0  0  0  0  0
## 3  0  0  0  0  0
## 4  0  0  0  0  0
## 5  0  0  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, 1:5]
```

```
##              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
##              AAACCTGTCGGAGCAA-1 AAACGGGAGTCGATAA-1
## MIR1302-2                    0                  0
## FAM138A                      0                  0
## OR4F5                        0                  0
## RP11-34P13.7                 0                  0
## RP11-34P13.8                 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. To view information about this object, enter the name of the object into the console.

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

```
## [1] "Calculating control metrics..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

```
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"
```

Load data from Cell Ranger into ascend automatically

If you are using Chromium data, you can also 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")
```

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.

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

```
## [1] "Plotting Total Count and Library Size..."
## [1] "Plotting Average Counts..."
```

```
## [1] "Plotting Proportion of Control Histograms..."
## [1] "Using ggplot2 to plot Control Percentages..."
## [1] "Using ggplot2 to plot Library Sizes Per Sample..."
## [1] "Using ggplot2 to plot Top Genes Per Sample..."
## [1] "Using ggplot2 to plot Top Gene Expression..."
```

```
PlotPDF(raw.qc.plots, "RawQC.pdf")
```

```
## pdf
##   2
```

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.

```
# 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")
```

```
## [1] "Calculating control metrics..."
##
  |
  |                                                      |   0%
  |
  |================================                      |  50%
```

```
  |
  |======================================================================| 100%
```

```
# 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)
```

```
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
```

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

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

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

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

```
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

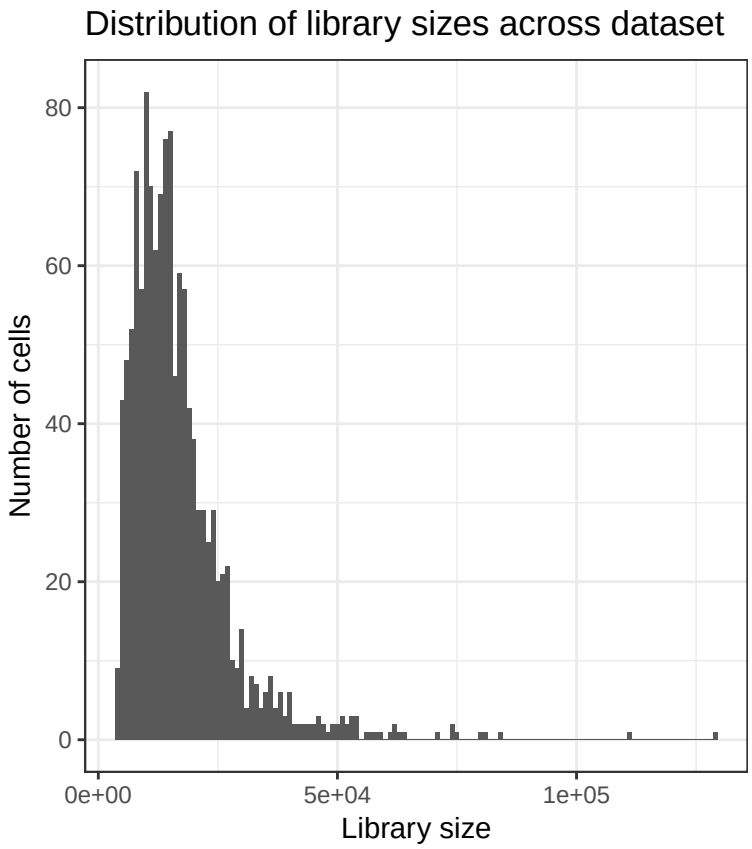
Cell filtering

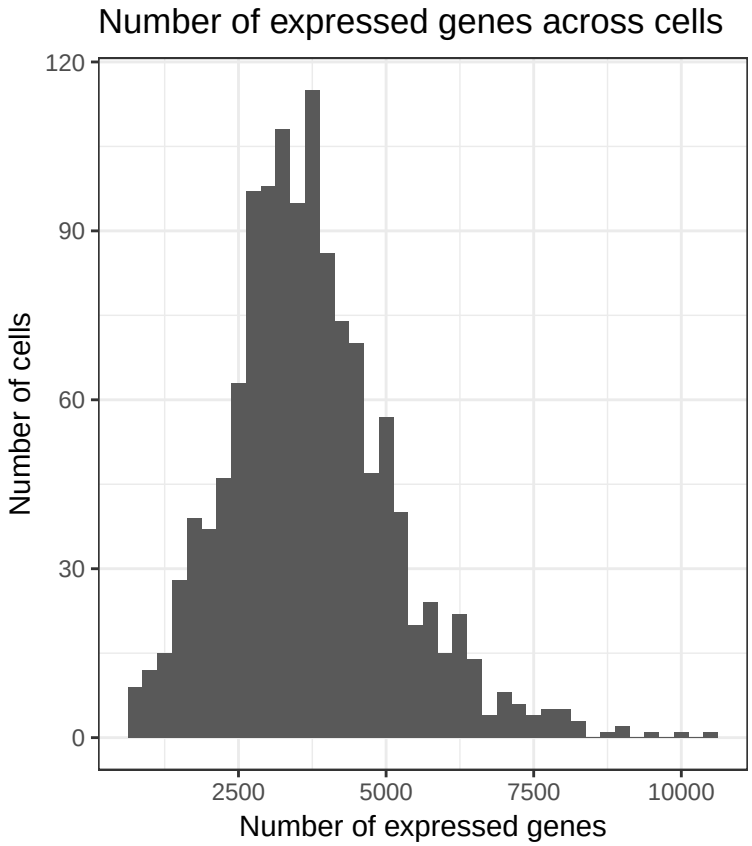
Filter cells by library size and gene expression

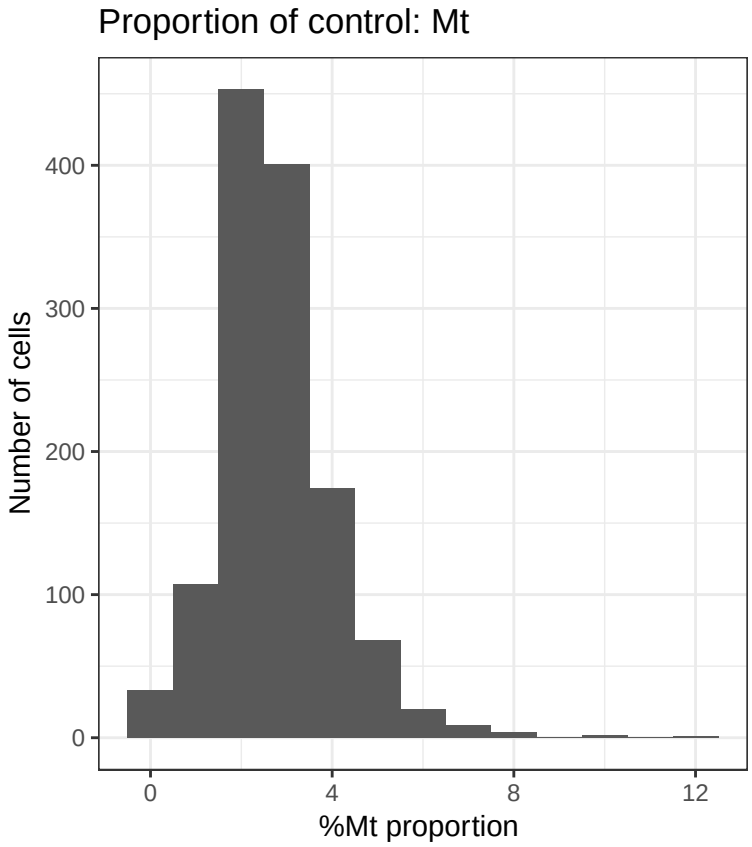
First, we will filter cells based on outliers in term of library size, number of non-control genes expressed and control genes expressed.

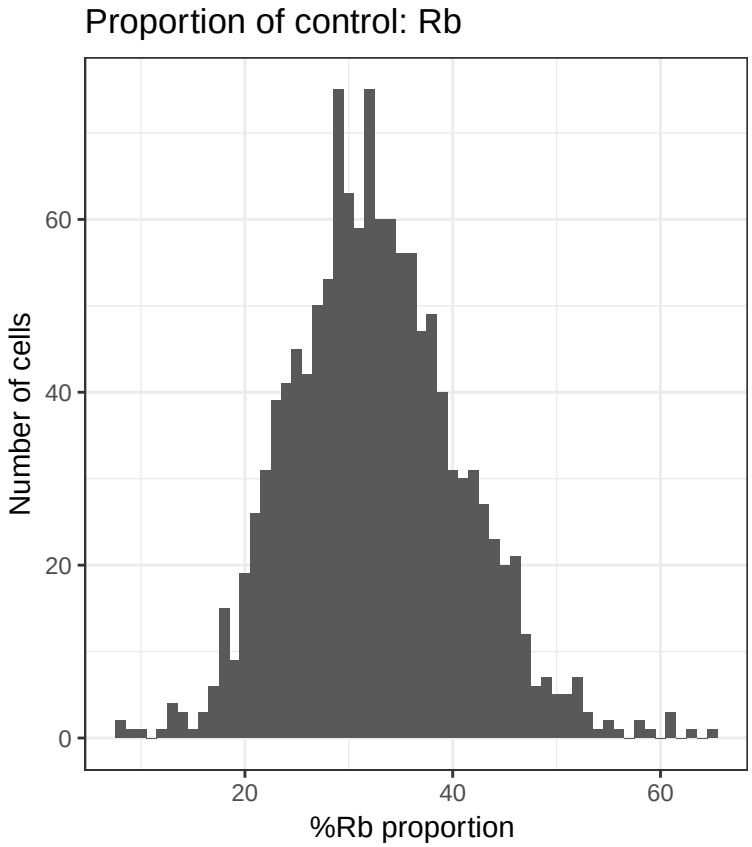
We can use the following plots to examine the distributions of these values.

```
print(raw.qc.plots$LibSize)
print(raw.qc.plots$FeatureCountsPerCell)
print(raw.qc.plots$ControlPercentageTotalCounts$Mt)
print(raw.qc.plots$ControlPercentageTotalCounts$Rb)
```




Number of expressed genes across cells
120
90
60
30
0
Number of cells
2500
5000
7500
10000
Number of expressed genes


Proportion of control: Mt
400
300
200
100
0
Number of cells
0
4
8
12
%Mt proportion



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)
```

```
## [1] "Identifying outliers..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
##
## [1] "Removing cells by library size..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
##
## [1] "Updating object information..."
##
  |
  |                                                                      |   0%
```

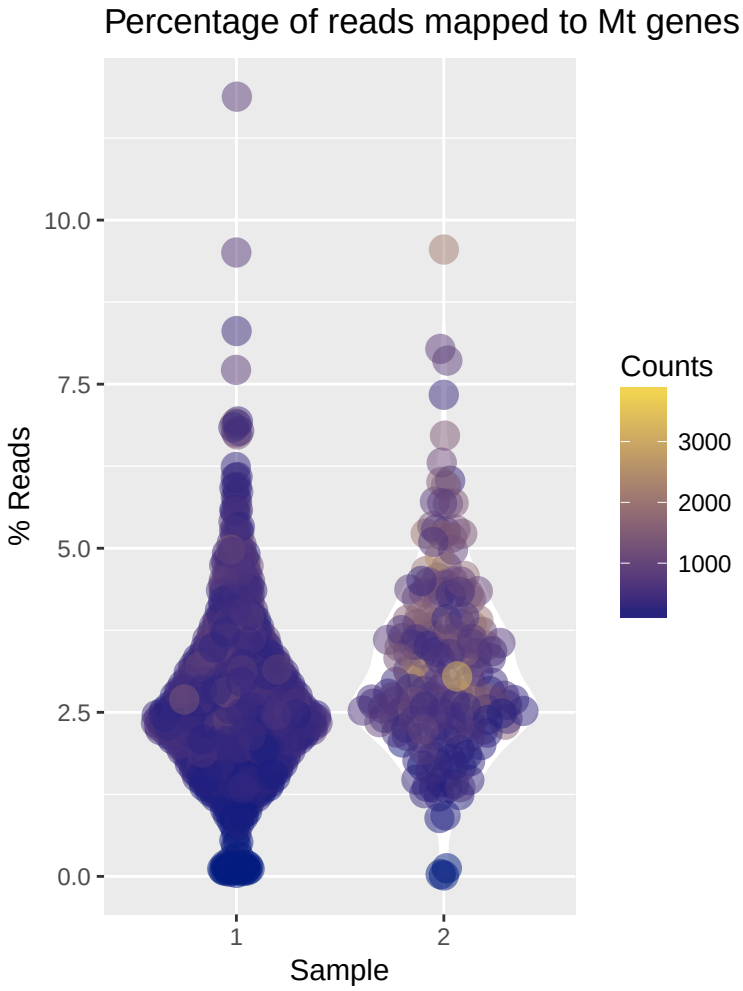
```
  |
  |================================                                 |  50%
  |
  |=================================================================| 100%
##
## [1] "Calculating control metrics..."
##
  |
  |                                                                 |   0%
  |
  |================================                                 |  50%
  |
  |=================================================================| 100%
```

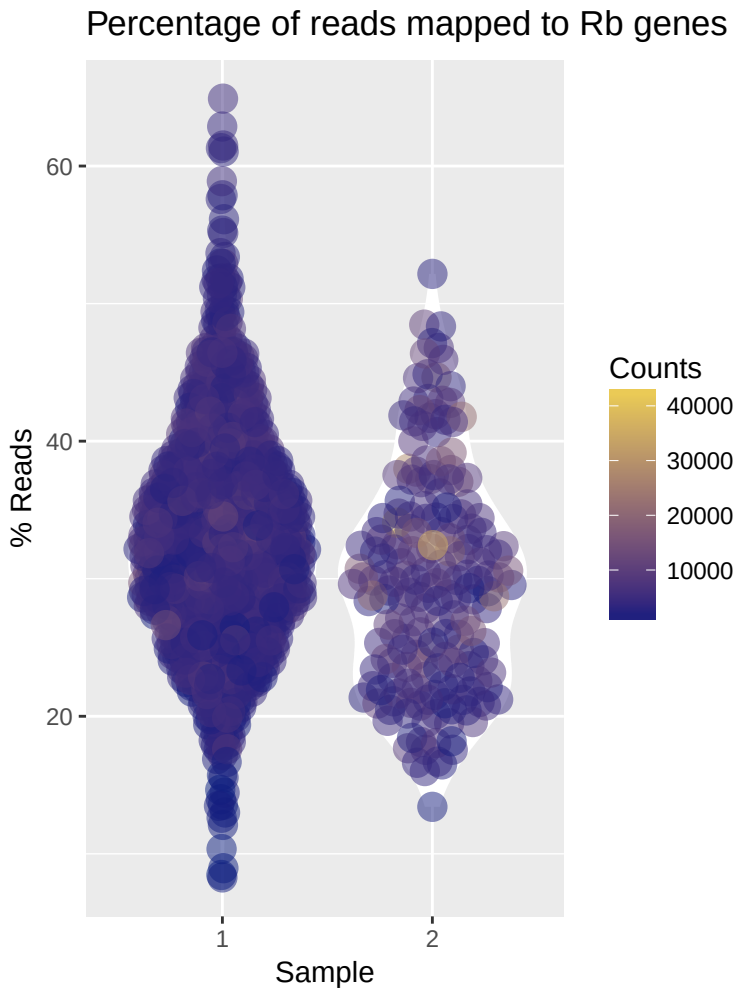
Filter cells by control gene expression

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.

```
print(raw.qc.plots$ControlPercentageSampleCounts$Mt)
print(raw.qc.plots$ControlPercentageSampleCounts$Rb)
```


Percentage of reads mapped to Mt genes
10.0
7.5
5.0
2.5
0.0
% Reads
1
2
Sample
Counts
3000
2000
1000



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.

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

```
## [1] "Calculating control metrics..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

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

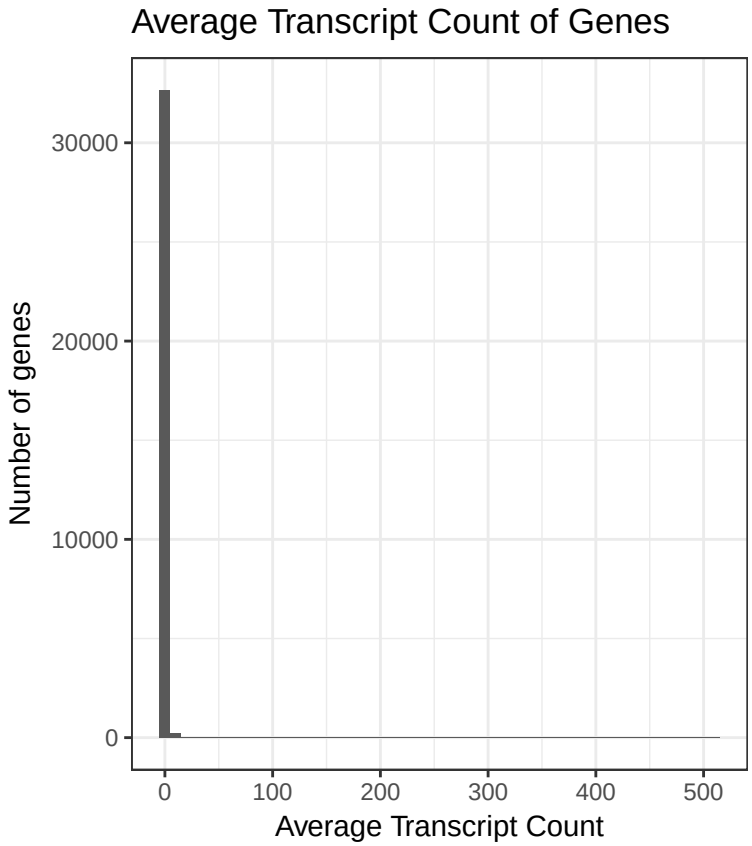
```
## [1] "Calculating control metrics..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

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

Filtering by expression

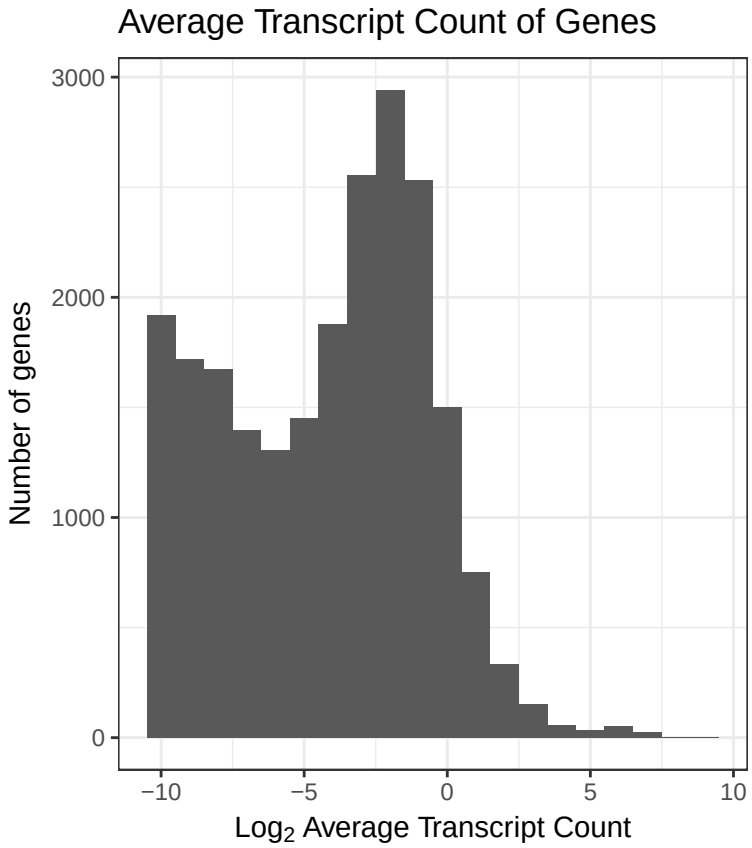
The final step of filtering is to remove low-abundance genes. The average expression of genes can be reviewed on the average transcript count plots.

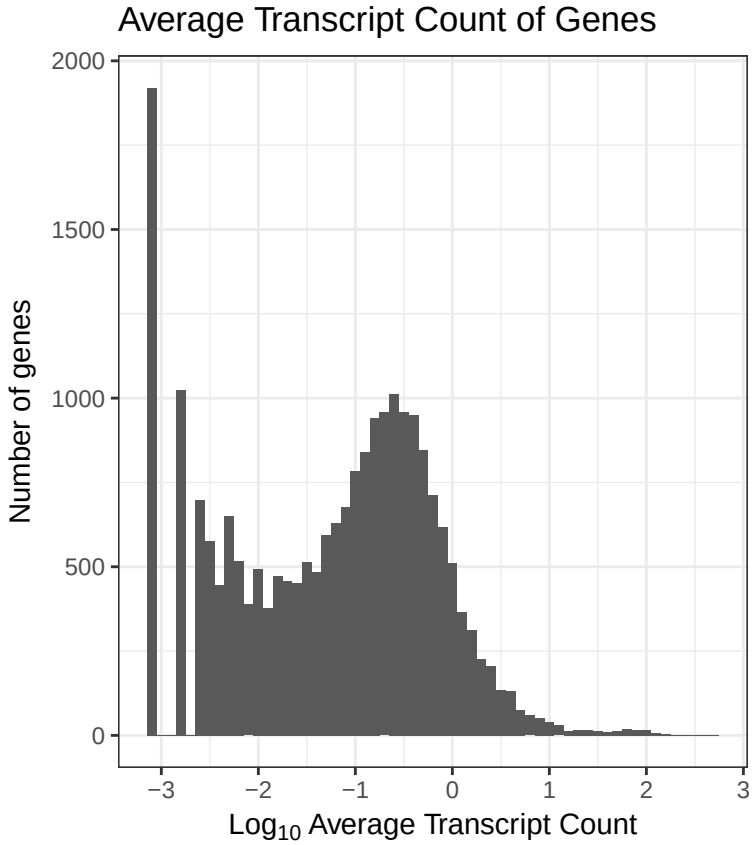
```
print(raw.qc.plots$AverageGeneCount)
```



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.

```
print(raw.qc.plots$Log2AverageGeneCount)
print(raw.qc.plots$Log10AverageGeneCount)
```


Average Transcript Count of Genes
3000
2000
1000
0
Number of genes
−10
−5
0
5
10
Log2 Average Transcript Count


Average Transcript Count of Genes
2000
1500
1000
500
0
Number of genes
−3
−2
−1
0
1
2
3
Log10 Average Transcript Count

If we wanted to remove these genes, we can use `FilterByExpressedGenesPerCell`. This function removes genes that are expressed in at most, a certain percentage of the cell population.

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

However, there are experiments where this is not ideal, such as this one. 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)]
```

```
##        ACAGCCGTCTCGCTTG-1 ACGCCAGGTGGTGTAG-1 AGGGTGATCGCGTTTC-1
## POU4F1                  0                  0                  0
## POU4F2                  1                  2                  1
## POU4F3                  0                  0                  0
##        CAAGTTGTCACGCGGT-1 CACAGGCGTCGACTAT-1 CCAGCGATCAGCACAT-1
## POU4F1                  1                  0                  0
## POU4F2                  0                  3                  1
## POU4F3                  0                  0                  0
##        CGGGTCATCGCAAGCC-1 CGTAGCGCATCACCCT-1 CTACACCCAAAGGAAG-1
## POU4F1                  0                  0                  0
## POU4F2                  4                  1                  4
## POU4F3                  0                  0                  0
##        CTAGCCTGTGCAACGA-1 GAACGGAGTGGTCTCG-1 TAAGTGCGTAGCTTGT-1
## POU4F1                  0                  0                  0
## POU4F2                 10                  7                  5
## POU4F3                  0                  0                  0
##        TCGCGAGGTTCTCATT-1 TTTGCGCGTCTTTCAT-1 AAATGCCAGACAGACC-2
## POU4F1                  0                  0                  0
## POU4F2                  1                  2                  4
## POU4F3                  0                  0                  0
##        AACTTTCGTAGCGTCC-2 AGGCCGTTCACCCGAG-2 AGGGAGTGTTGGGACA-2
## POU4F1                  1                  0                  0
## POU4F2                  0                  1                  3
## POU4F3                  0                  0                  0
##        CGTCACTCATATGAGA-2 GGAAAGCAGGCCGAAT-2
## POU4F1                  0                  0
## POU4F2                  1                  5
## POU4F3                  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
```

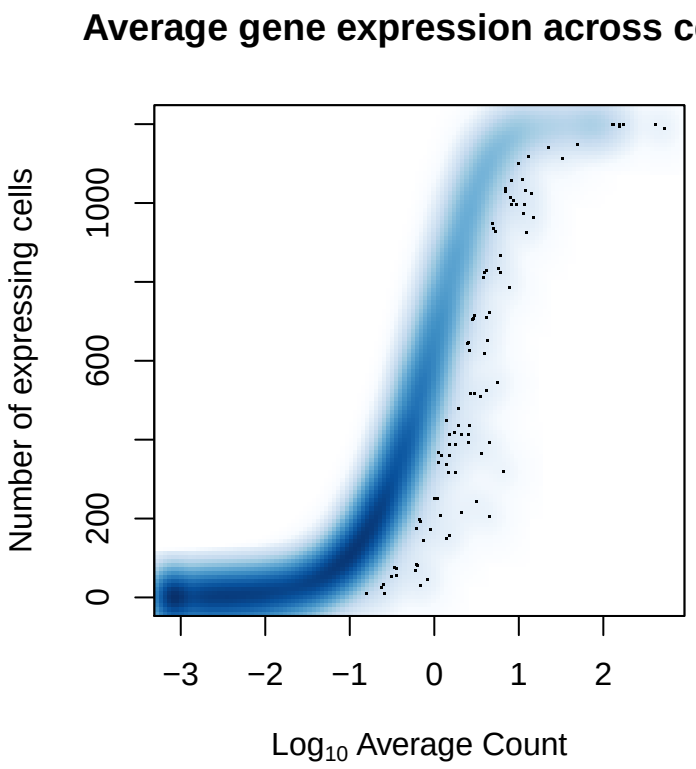
```
## $FilterByOutliers$CellsFilteredByLibSize
## NULL
##
## $FilterByOutliers$CellsFilteredByLowExpression
##  [1] "AACTGGTTCATAAAGG-1" "AAGACCTAGACAATAC-1" "ACAGCCGGTCTCTCTG-1"
##  [4] "ACGATACGTCACACGC-1" "ACTTGTTCAAGCGTAG-1" "AGCCTAACAGTAACGG-1"
##  [7] "AGTCTTTTCACCACCT-1" "AGTGTCAAGCCACTAT-1" "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"
```

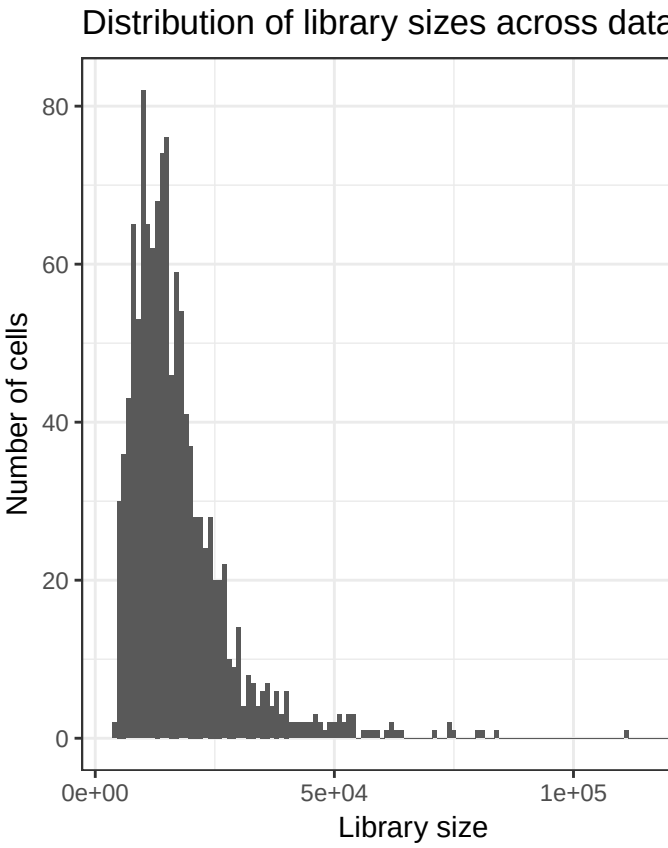
You can also run `PlotGeneralQC` again to see how the filtering has altered the dataset.

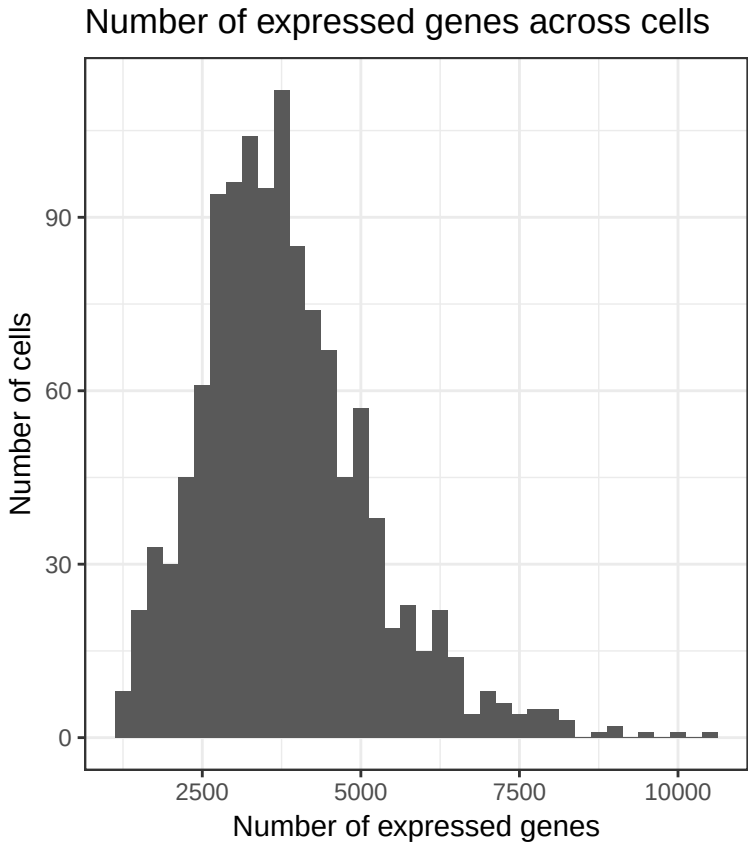
```
## [1] "Plotting Total Count and Library Size..."
```

```
## [1] "Plotting Average Counts..."
```

```
## [1] "Plotting Proportion of Control Histograms..."
## [1] "Using ggplot2 to plot Control Percentages..."
## [1] "Using ggplot2 to plot Library Sizes Per Sample..."
## [1] "Using ggplot2 to plot Top Genes Per Sample..."
## [1] "Using ggplot2 to plot Top Gene Expression..."
```







Normalisation

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

The ascend package contains 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.

How you use these functions depends 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).

We do need to normalise between cells so we can use one of the following functions: `NormaliseByRLE` or `scranNormalise`.

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)
```

```
## [1] "Converting EMSet to SingleCellExperiment..."
## [1] "1200 cells detected. Running computeSumFactors with preset sizes of 40, 60, 80, 100..."
## [1] "scran's computeSumFactors complete. Removing zero sum factors from dataset..."
## [1] "Running scater's normalize method..."
## [1] "Normalisation complete. Converting SingleCellExperiment back to EMSet..."
## [1] "Calculating control metrics..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

Reviewing the normalisation process

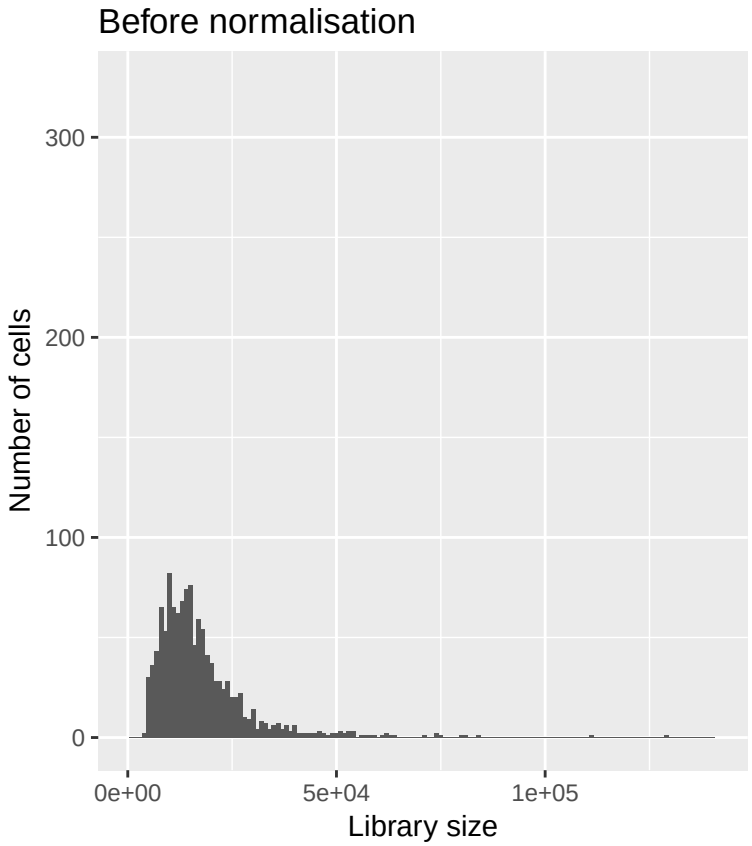
`PlotNormalisationQC` will generate a series of plots for the review of the normalisation process. This function can only be used if you have retained the un-normalised `EMSet`. You can also review the expression of genes you are interested in; in this case, we will look at the expression of GAPDH and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

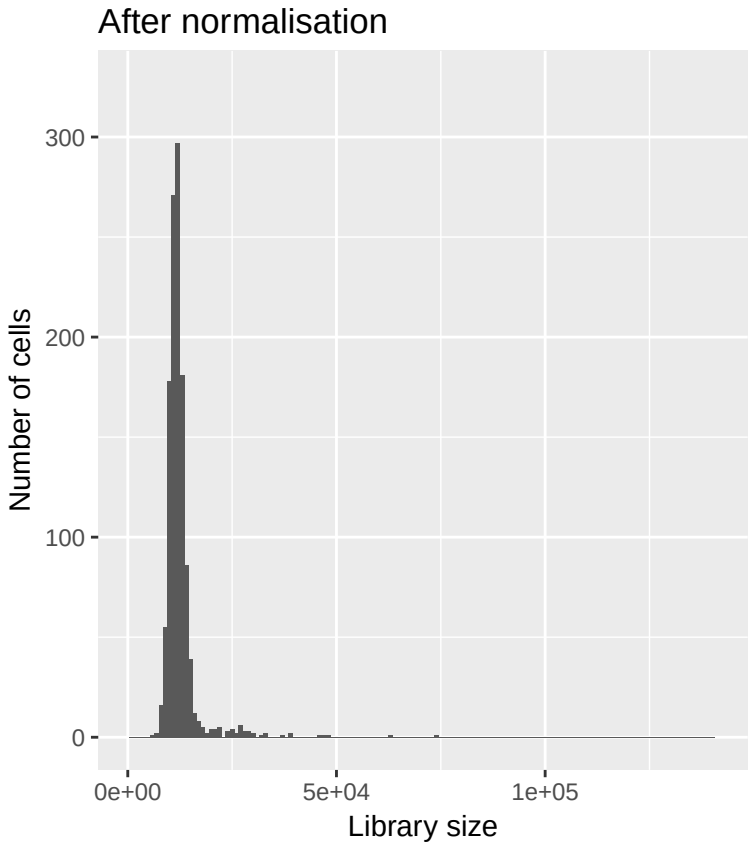
```
norm.qc <- PlotNormalisationQC(original = em.set, normalised = norm.set,
                               gene.list = c("GAPDH", "MALAT1"))
```

```
## [1] "Retrieving data from EMSets..."
## [1] "Plotting libsize histograms..."
## [1] "Plotting GAPDH expression..."
## [1] "Plotting MALAT1 expression..."
## [1] "Plotting gene expression box plots..."
## [1] "Plotting gene expression box plots..."
## [1] "Plots complete!"
```

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

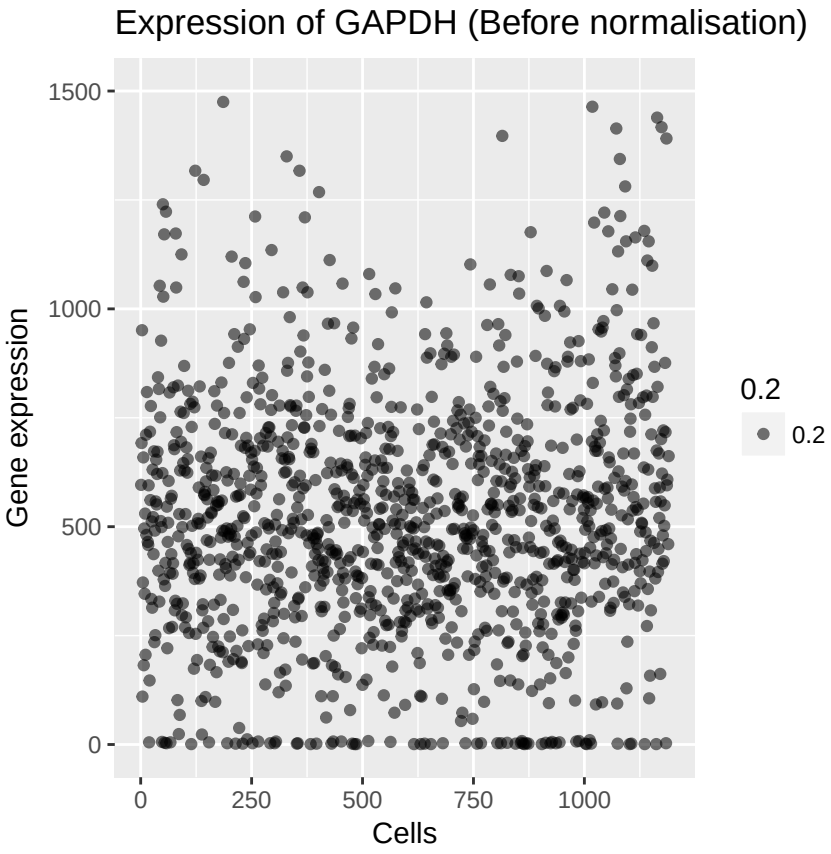
```
print(norm.qc$Libsize$Original)
print(norm.qc$Libsize$Normalised)
```

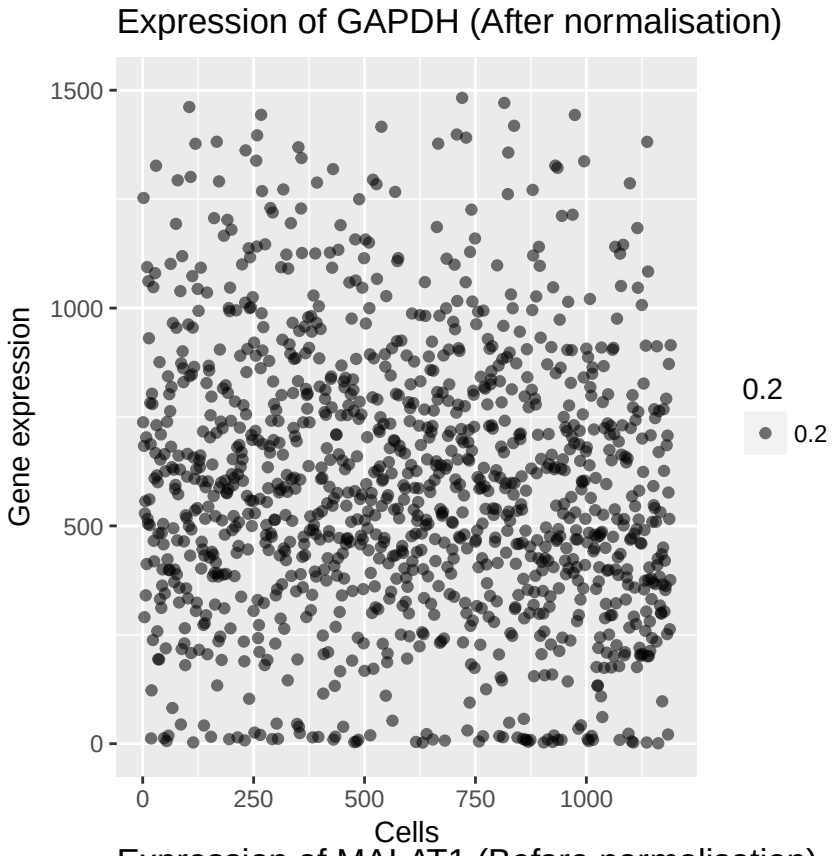

Before normalisation
300
200
100
0
Number of cells
0e+00
5e+04
1e+05
Library size

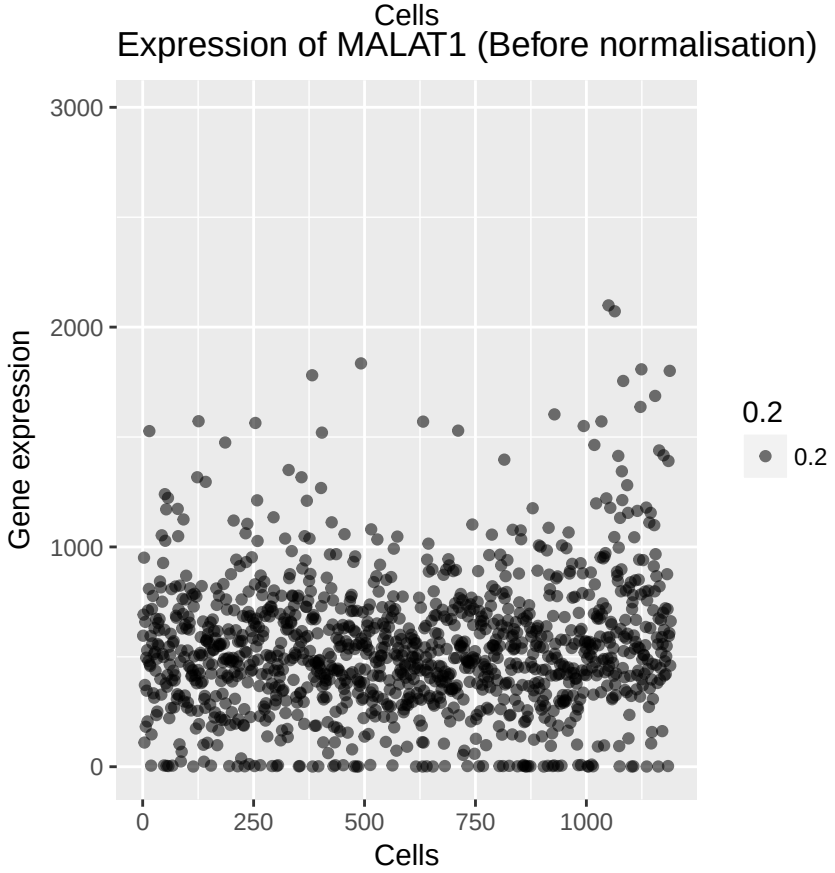

After normalisation
300
200
100
0
Number of cells
0e+00
5e+04
1e+05
Library size

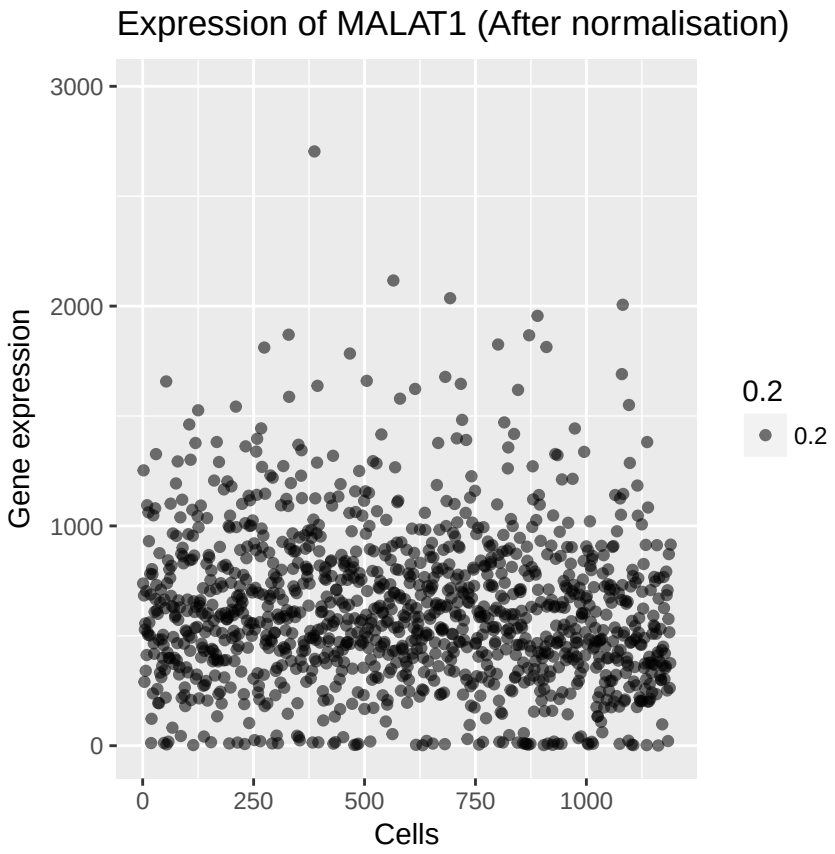
The gene scatter plots show how expression has changed on a gene level. Both genes are strongly expressed in this dataset, and normalisation has enabled us to make a clearer distinction between the expression level of these genes between each cell.

```
print(norm.qc$GeneScatterPlots$GAPDH$Original)
print(norm.qc$GeneScatterPlots$GAPDH$Normalised)
print(norm.qc$GeneScatterPlots$MALAT1$Original)
print(norm.qc$GeneScatterPlots$MALAT1$Normalised)
```



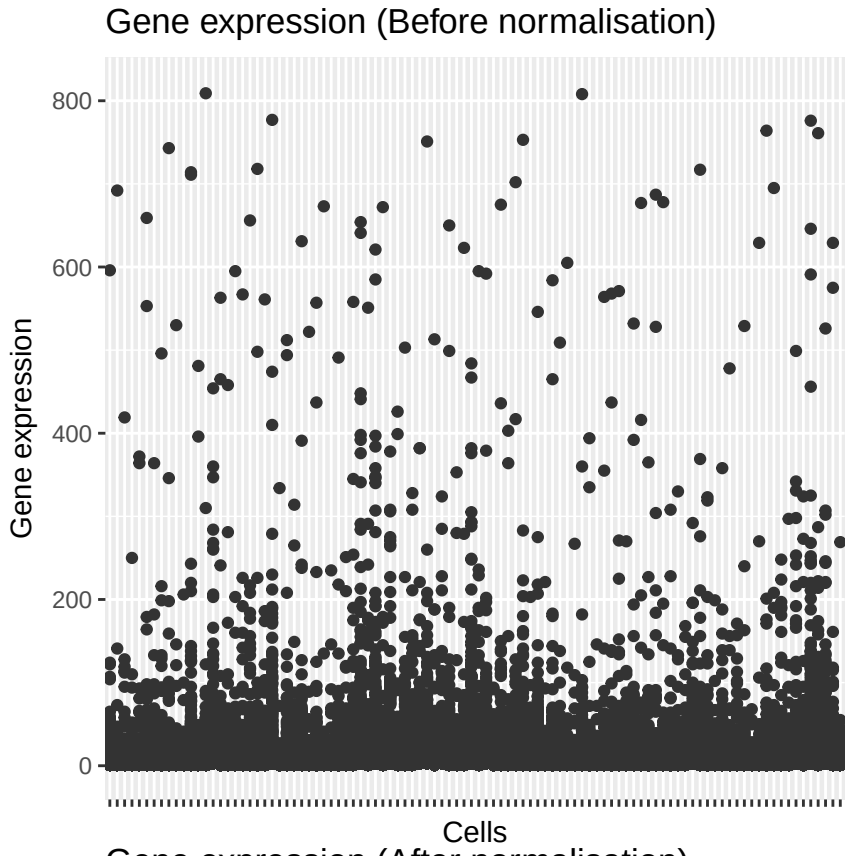

Expression of GAPDH (After normalisation)
Gene expression
1500
1000
500
0
0
250
500
750
1000
Cells
0.2
0.2

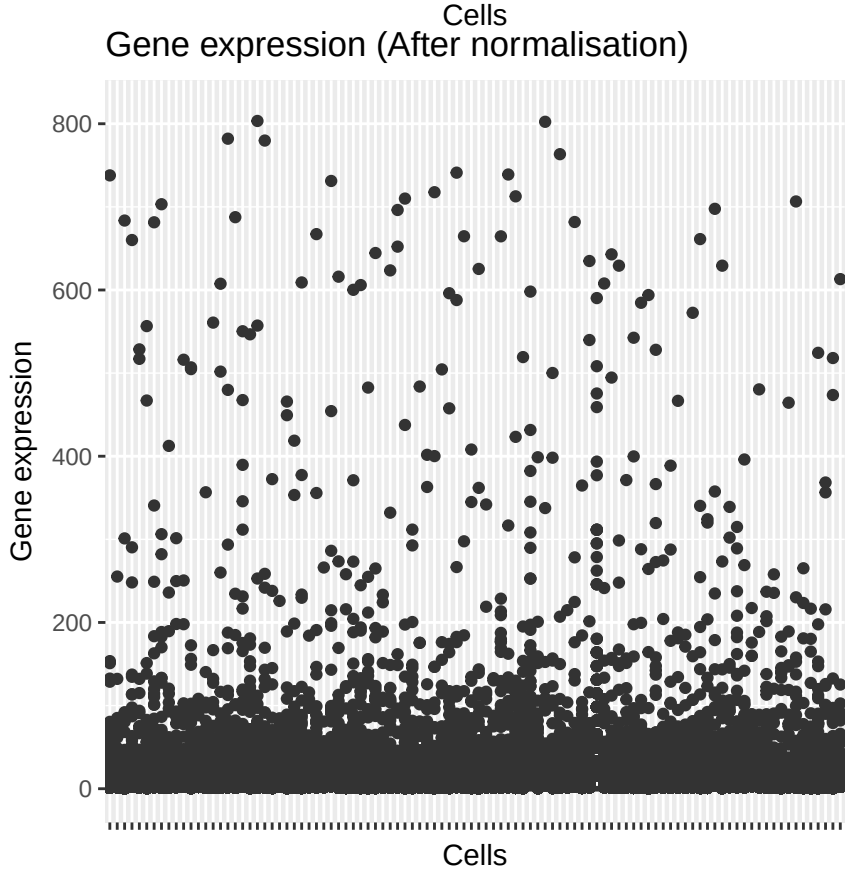

Expression of MALAT1 (Before normalisation)
Gene expression
3000
2000
1000
0
0
250
500
750
1000
Cells
0.2
0.2



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

```
print(norm.qc$GeneExpressionBoxplot$Original)
print(norm.qc$GeneExpressionBoxplot$Normalised)
```


Gene expression (Before normalisation)
Gene expression
800
600
400
200
0
Cells

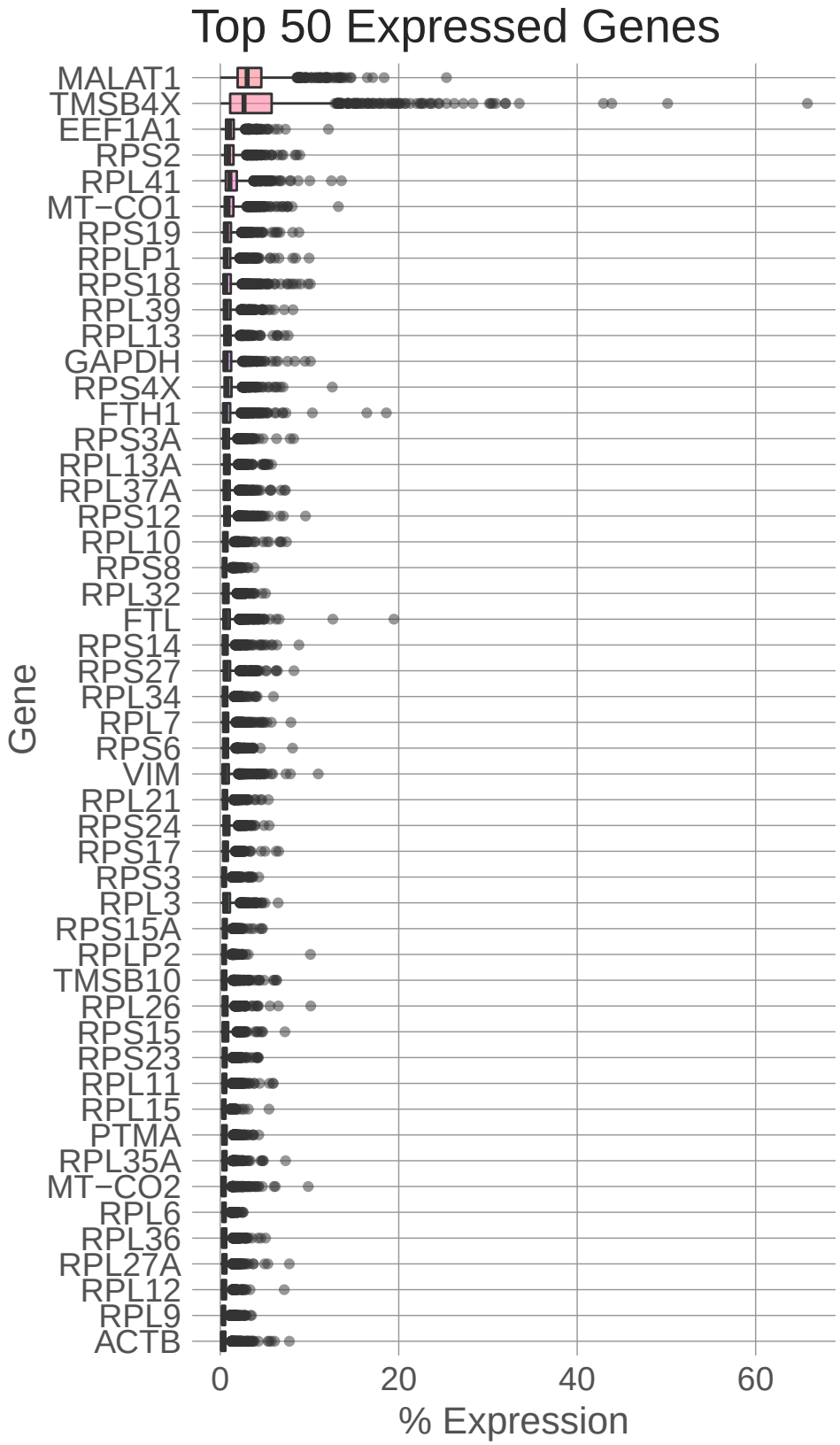

Gene expression (After normalisation)
Gene expression
800
600
400
200
0
Cells

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.

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

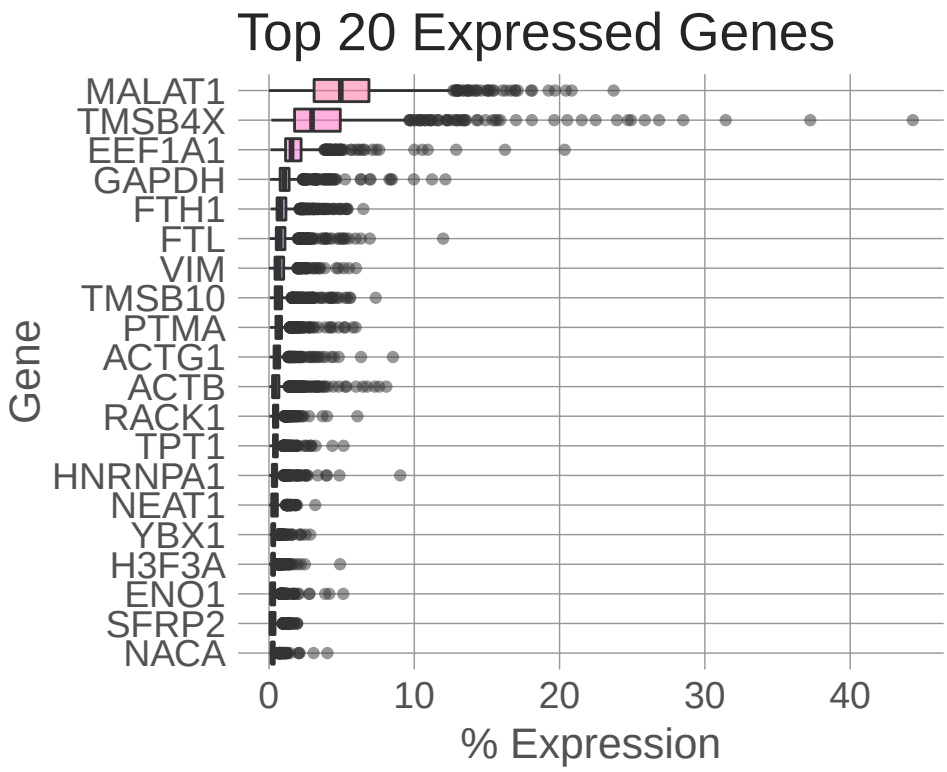


As you can see, ribosomal genes dominate gene expression, even after filtering. What does the dataset look like without these control genes? We will just plot the top 20 most expressed genes.

```
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

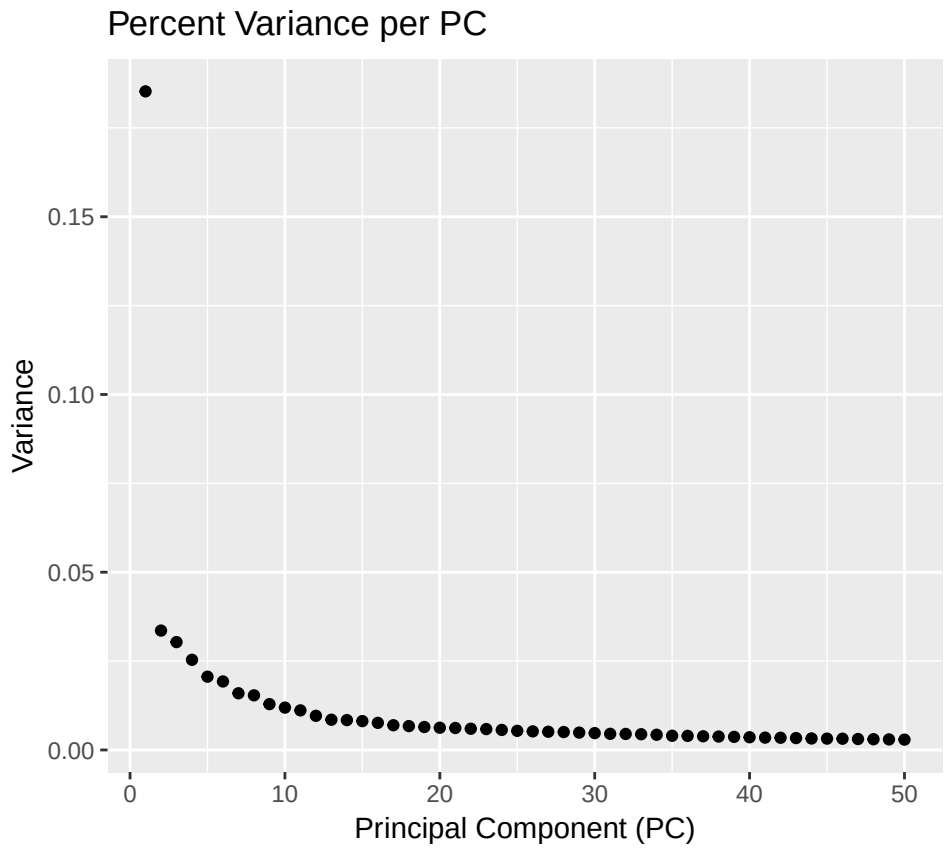
We have filtered our dataset down to 1235 cells and 32904 genes and normalised the transcript counts with `scranNormalise`. We can reduce this dataset further by using *Principal Component Analysis (PCA)* to identify genes that are major contributors to variation.

```
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. This function has only one argumenmt - `conservative`. This argument is only used when the algorithm detects more than one set of stable results. If set to TRUE, the function will choose the result that is the most stable but yields the least number of clusters. If set to FALSE, the function will choose the result that is the least stable, but yields the most number of clusters.

```
clustered.set <- RunCORE(pca.set, conservative = TRUE)
```

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

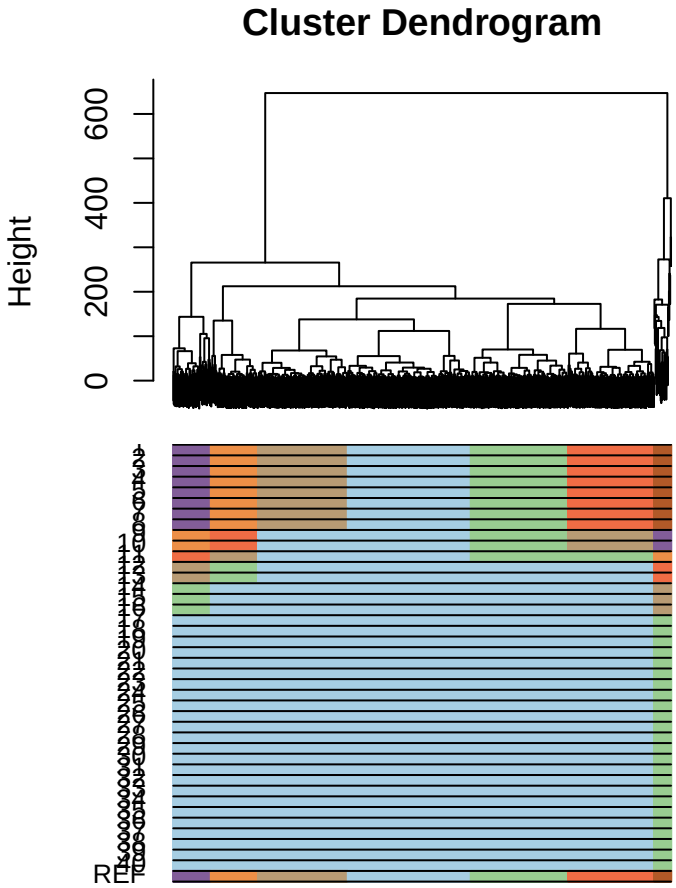
```
## [1] "Calculating rand indices..."
## [1] "Calculating stability values..."
## [1] "Aggregating data..."
##
  |
  |                                                                      |   0%
  |
  |=====================                                                 |  33%
  |
  |=========================================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

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 40 values ranging from 0.025 (the bottom of the tree) to 1 (the top of the tree).

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)
```

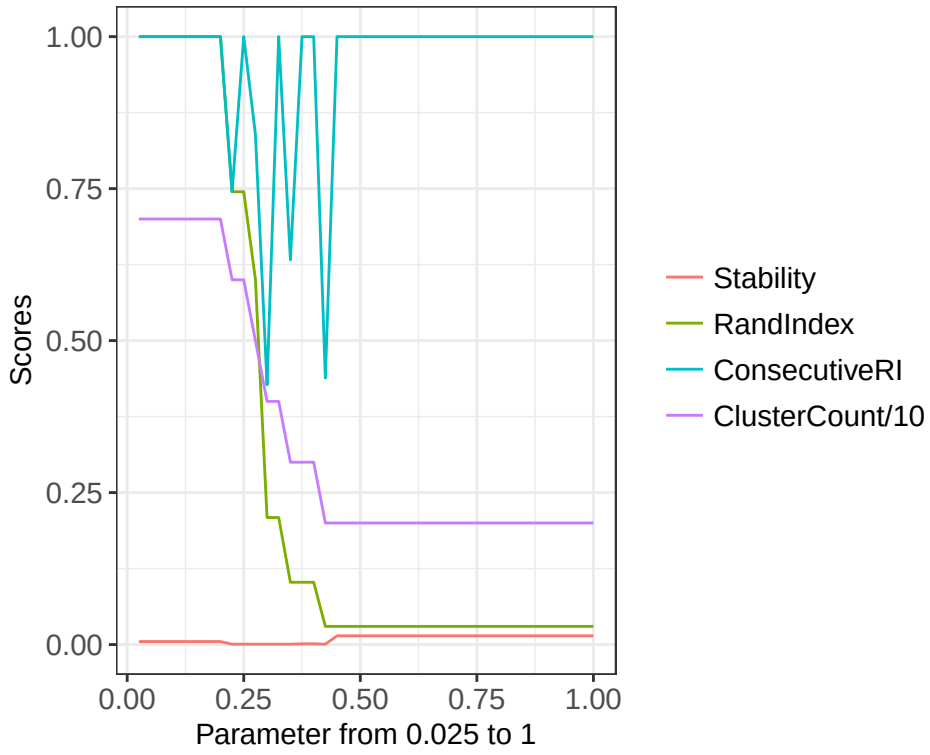
```
## $mar
## [1] 1 5 0 1
```



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`.

```
rand.matrix <- GetRandMatrix(clustered.set)
rand.matrix
```

```
##    Height Stability  RandIndex ConsecutiveRI ClusterCount
## 1   0.025     0.200 1.00000000     1.0000000            7
## 2    0.05     0.200 1.00000000     1.0000000            7
## 3   0.075     0.200 1.00000000     1.0000000            7
## 4     0.1     0.200 1.00000000     1.0000000            7
## 5   0.125     0.200 1.00000000     1.0000000            7
## 6    0.15     0.200 1.00000000     1.0000000            7
## 7   0.175     0.200 1.00000000     1.0000000            7
## 8     0.2     0.200 1.00000000     1.0000000            7
## 9   0.225     0.025 0.74484782     0.7448478            6
## 10   0.25     0.025 0.74484782     1.0000000            6
## 11  0.275     0.025 0.60083697     0.8396588            5
## 12    0.3     0.025 0.20897468     0.4273129            4
## 13  0.325     0.025 0.20897468     1.0000000            4
## 14   0.35     0.025 0.10248122     0.6328317            3
## 15  0.375     0.050 0.10248122     1.0000000            3
## 16    0.4     0.050 0.10248122     1.0000000            3
## 17  0.425     0.025 0.02988621     0.4385739            2
## 18   0.45     0.575 0.02988621     1.0000000            2
## 19  0.475     0.575 0.02988621     1.0000000            2
## 20    0.5     0.575 0.02988621     1.0000000            2
## 21  0.525     0.575 0.02988621     1.0000000            2
## 22   0.55     0.575 0.02988621     1.0000000            2
```

```
## 23  0.575     0.575 0.02988621     1.0000000           2
## 24    0.6     0.575 0.02988621     1.0000000           2
## 25  0.625     0.575 0.02988621     1.0000000           2
## 26   0.65     0.575 0.02988621     1.0000000           2
## 27  0.675     0.575 0.02988621     1.0000000           2
## 28    0.7     0.575 0.02988621     1.0000000           2
## 29  0.725     0.575 0.02988621     1.0000000           2
## 30   0.75     0.575 0.02988621     1.0000000           2
## 31  0.775     0.575 0.02988621     1.0000000           2
## 32    0.8     0.575 0.02988621     1.0000000           2
## 33  0.825     0.575 0.02988621     1.0000000           2
## 34   0.85     0.575 0.02988621     1.0000000           2
## 35  0.875     0.575 0.02988621     1.0000000           2
## 36    0.9     0.575 0.02988621     1.0000000           2
## 37  0.925     0.575 0.02988621     1.0000000           2
## 38   0.95     0.575 0.02988621     1.0000000           2
## 39  0.975     0.575 0.02988621     1.0000000           2
## 40      1     0.575 0.02988621     1.0000000           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. This step is computationally intensive for larger datasets.

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.

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")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |============================================                          |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: TRUE vs FALSE complete!"
```

```
thy1.de.result[1:10,]
```

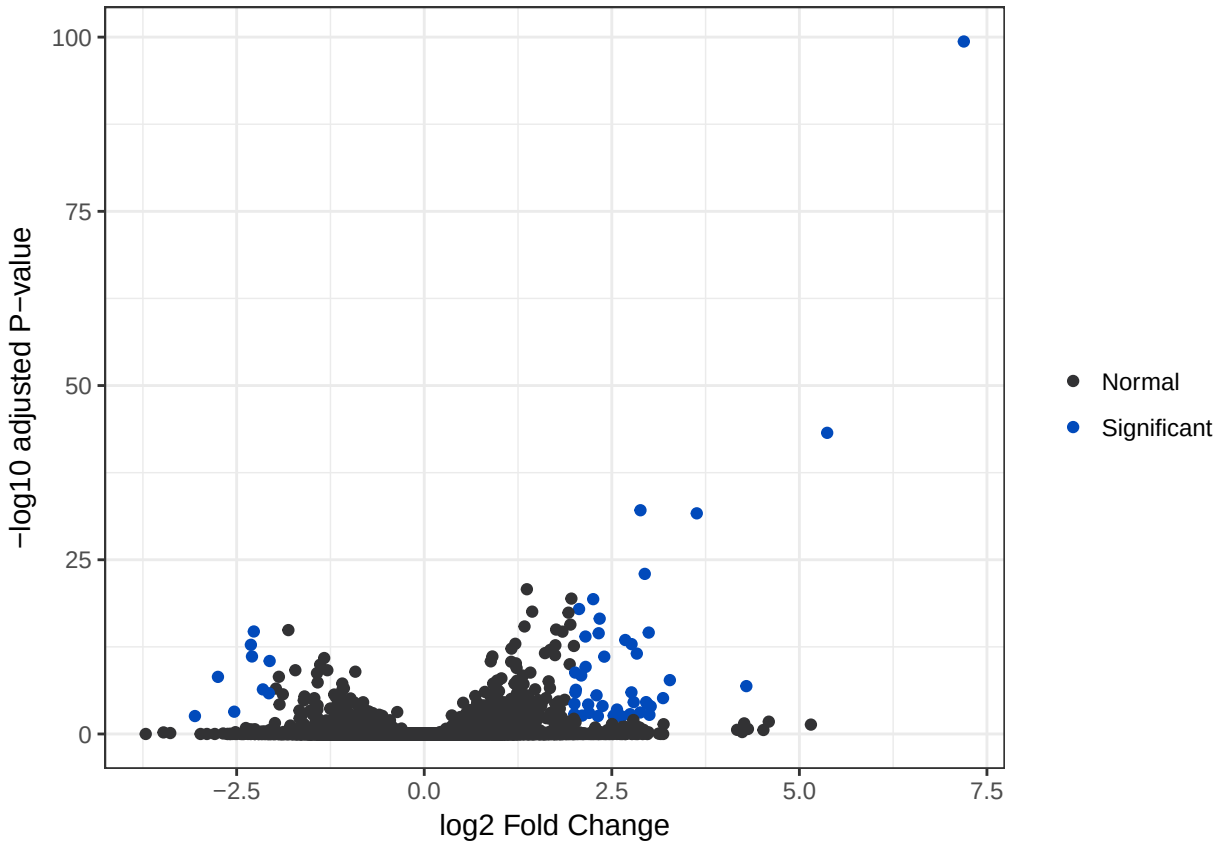
```
##              id  baseMean baseMeanA baseMeanB foldChange log2FoldChange
## 11926       MGP  1.906847  1.039802  6.820100 146.226592       7.192062
## 12388    COL3A1  1.566694  1.080307  4.322886  41.377046       5.370759
## 10388    LGALS1  4.951281  3.020405 15.892912   7.371249       2.881909
## 5334       CTGF  2.742901  1.643053  8.975374  12.402361       3.632543
## 10363      TGM2  1.750495  1.375119  3.877625   7.671226       2.939457
## 5933      ITGB1  4.572514  3.887745  8.452876   2.580864       1.367854
## 8614       TPM1 10.756877  7.803241 27.494144   3.894341       1.961379
## 17413    COL1A1  3.987944  2.909903 10.096845   4.762988       2.251867
## 20010 TNFRSF12A  2.123340  1.760514  4.179352   4.180531       2.063686
## 19839     FSTL1  2.570386  2.249719  4.387500   2.710609       1.438617
```

```
##                pval          padj
## 11926 4.231017e-103 4.252172e-100
## 12388  6.082067e-47  6.112478e-44
## 10388  7.791379e-36  7.830336e-33
## 5334   2.169420e-35  2.180268e-32
## 10363  2.108726e-26  1.059635e-23
## 5933   3.374687e-24  1.695780e-21
## 8614   3.740572e-23  3.759274e-20
## 17413  4.475804e-23  4.498183e-20
## 20010  1.173918e-21  1.178613e-18
## 19839  5.568784e-21  2.795529e-18
```

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")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |============================================                          |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 1 vs 2 complete!"
```

```
cluster.de.result[1:10,]
```

```
##             id baseMean baseMeanA baseMeanB foldChange log2FoldChange
## 3224       TTR 4.314462  4.419727  1.338787 0.09906844     -3.3354306
## 19147     IL32 1.973927  2.000830  1.213398 0.21322056     -2.2295816
## 11773  DNAJC15 1.750583  1.720315  2.606201 2.22985766      1.1569516
## 7448     SUMF2 2.171181  2.199026  1.384037 0.32029043     -1.6425474
## 978     S100A6 1.562109  1.581493  1.014163 0.02435602     -5.3595777
## 18171      FN1 2.863821  2.906379  1.660785 0.34661783     -1.5285822
## 22104     COQ5 1.626951  1.606389  2.208205 1.99245941      0.9945503
## 7604      UPP1 1.591483  1.567226  2.277179 2.25162142      1.1709643
## 3207     MED31 1.710156  1.683641  2.459693 2.13517471      1.0943541
## 5254  IVNS1ABP 2.416935  2.370198  3.738095 1.99832002      0.9987876
##               pval        padj
## 3224  8.292527e-11 8.33399e-08
## 19147 2.062445e-03 1.00000e+00
## 11773 2.712567e-03 1.00000e+00
## 7448  3.753265e-03 1.00000e+00
## 978   3.841124e-03 1.00000e+00
## 18171 5.530897e-03 1.00000e+00
## 22104 5.897854e-03 1.00000e+00
## 7604  6.682398e-03 1.00000e+00
## 3207  8.814417e-03 1.00000e+00
## 5254  8.860888e-03 1.00000e+00
```

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.

```
clean.set <- SubsetCluster(clustered.set, clusters = "1")
```

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

```
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

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

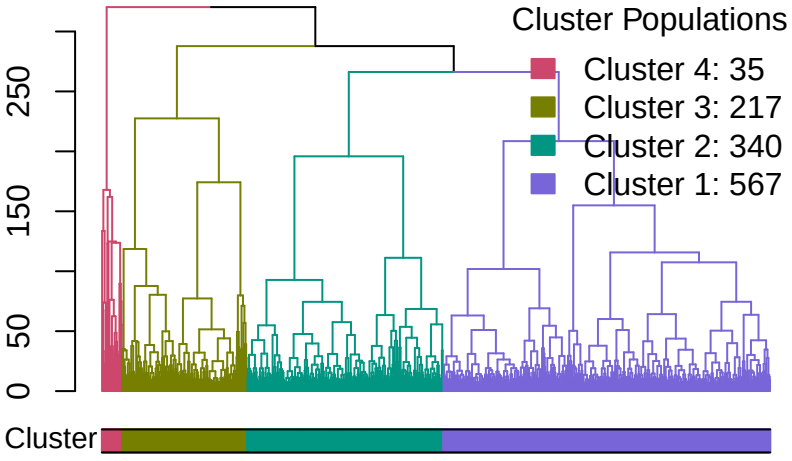
```
## [1] "Retrieving data..."
## [1] "Calculating variance..."
## [1] "Computing PCA values..."
## [1] "PCA complete! Returning object..."
```

```
clean.cluster <- RunCORE(clean.pca, conservative = TRUE)
```

```
## [1] "Performing unsupervised clustering..."
## [1] "Generating clusters by running dynamicTreeCut at different heights..."
## [1] "Calculating rand indices..."
## [1] "Calculating stability values..."
## [1] "Aggregating data..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |============================================                          |  67%
  |
  |======================================================================| 100%
##
## [1] "Finding optimal number of clusters..."
## [1] "Optimal number of clusters found! Returning output..."
```

```
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.

```
# 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))
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 1 vs Others complete!"
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

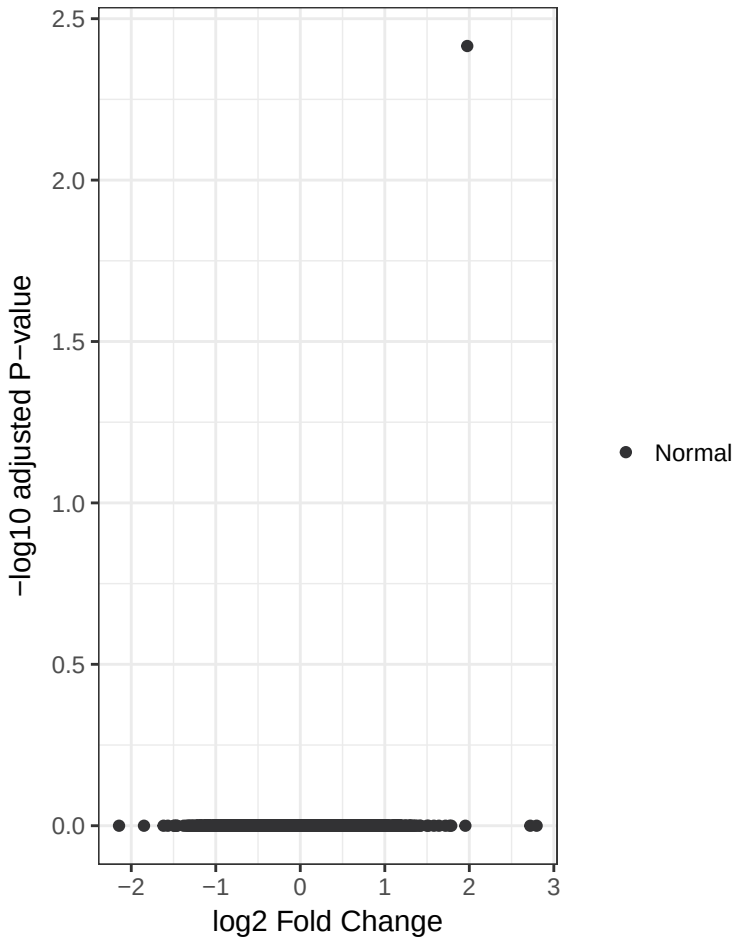
```
## Warning in split.default(sample(1:nelements), 1:chunks): data length is not
## a multiple of split variable
```

```
## [1] "Running DESeq..."
```

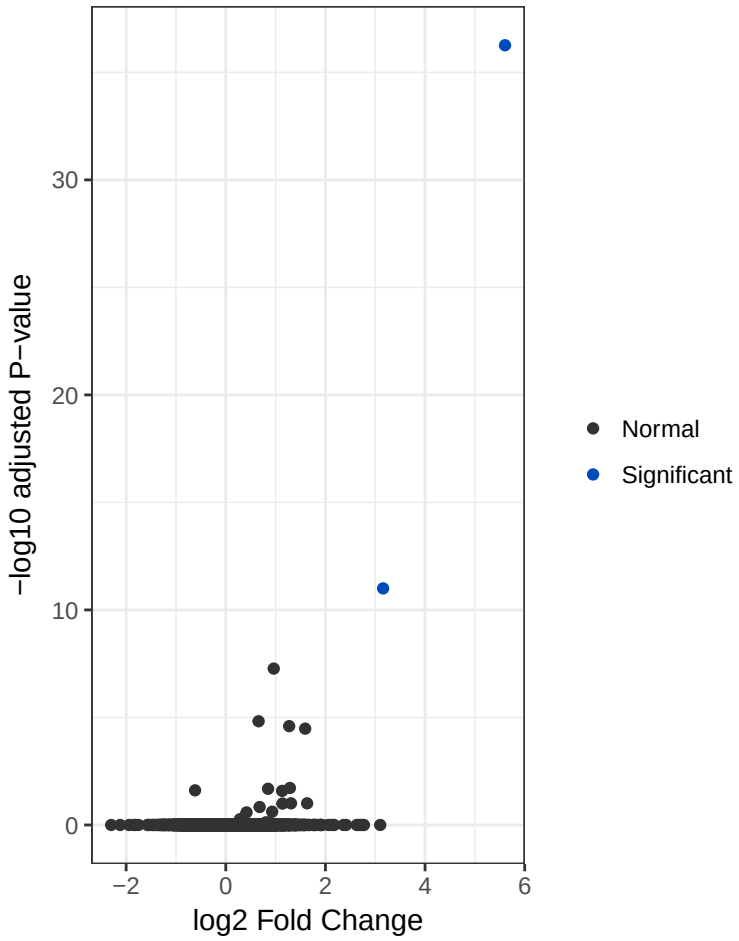
```
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 2 vs Others complete!"
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
## Warning in split.default(sample(1:nelements), 1:chunks): data length is not
## a multiple of split variable
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 3 vs Others complete!"
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
## Warning in split.default(sample(1:nelements), 1:chunks): data length is not
## a multiple of split variable
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
```

```
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 4 vs Others complete!"
```

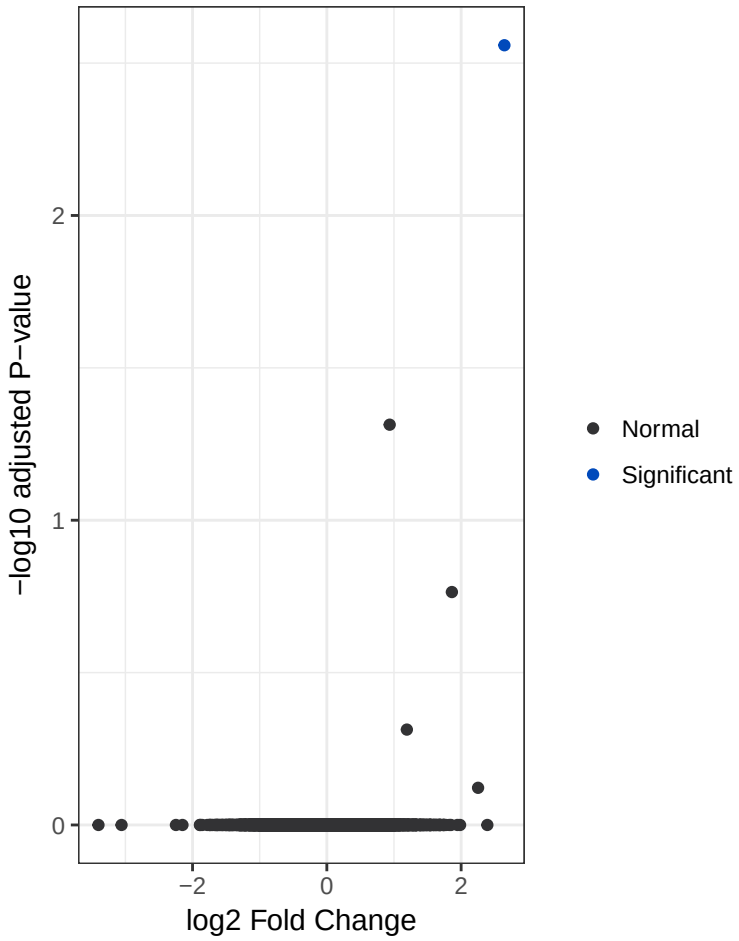
```
# Generate volcano plots
cluster.de.1 <- PlotDEVolcano(clean.cluster.de.results[[1]], labels = FALSE)
cluster.de.2 <- PlotDEVolcano(clean.cluster.de.results[[2]], labels = FALSE)
cluster.de.3 <- PlotDEVolcano(clean.cluster.de.results[[3]], labels = FALSE)
cluster.de.4 <- PlotDEVolcano(clean.cluster.de.results[[4]], labels = TRUE)
print(cluster.de.1)
```



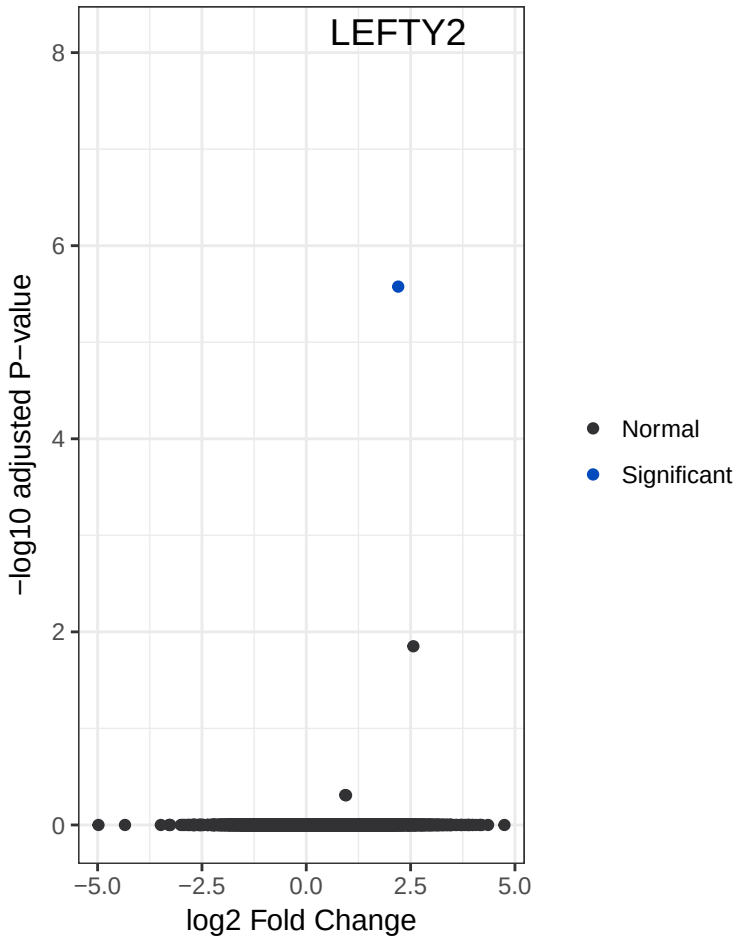

```
print(cluster.de.2)
```




```
print(cluster.de.3)
```




```
print(cluster.de.4)
```



Comparing pairs of clusters

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

```
# Run differential expression on pairs
c1c2.de.results <- RunDiffExpression(clean.cluster, condition.a = "1",
                                     condition.b = "2", conditions = "cluster")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
```

```
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 1 vs 2 complete!"
```

```
c1c3.de.results <- RunDiffExpression(clean.cluster, condition.a = "1",
                                     condition.b = "3", conditions = "cluster")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 1 vs 3 complete!"
```

```
c1c4.de.results <- RunDiffExpression(clean.cluster, condition.a = "1",
                                     condition.b = "4", conditions = "cluster")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
```

```
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 1 vs 4 complete!"
```

```
c2c3.de.results <- RunDiffExpression(clean.cluster, condition.a = "2",
                                     condition.b = "3", conditions = "cluster")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 2 vs 3 complete!"
```

```
c2c4.de.results <- RunDiffExpression(clean.cluster, condition.a = "2",
                                     condition.b = "4", conditions = "cluster")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

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

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 2 vs 4 complete!"
```

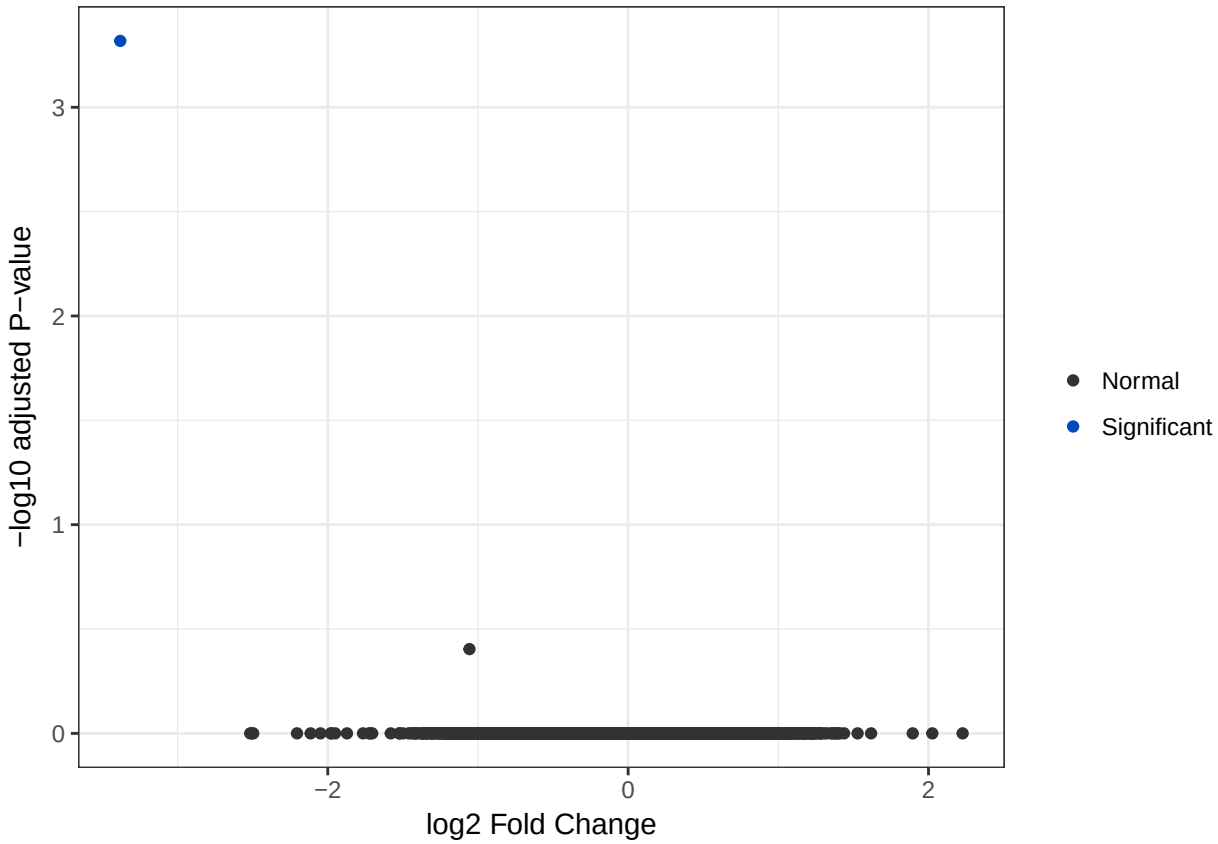
```
c3c4.de.results <- RunDiffExpression(clean.cluster, condition.a = "3",
                                     condition.b = "4", conditions = "cluster")
```

```
## [1] "Processing expression matrix..."
## [1] "Rounding expression matrix values..."
## [1] "Chunking matrix..."
## [1] "Chunking expression matrix by rows..."
```

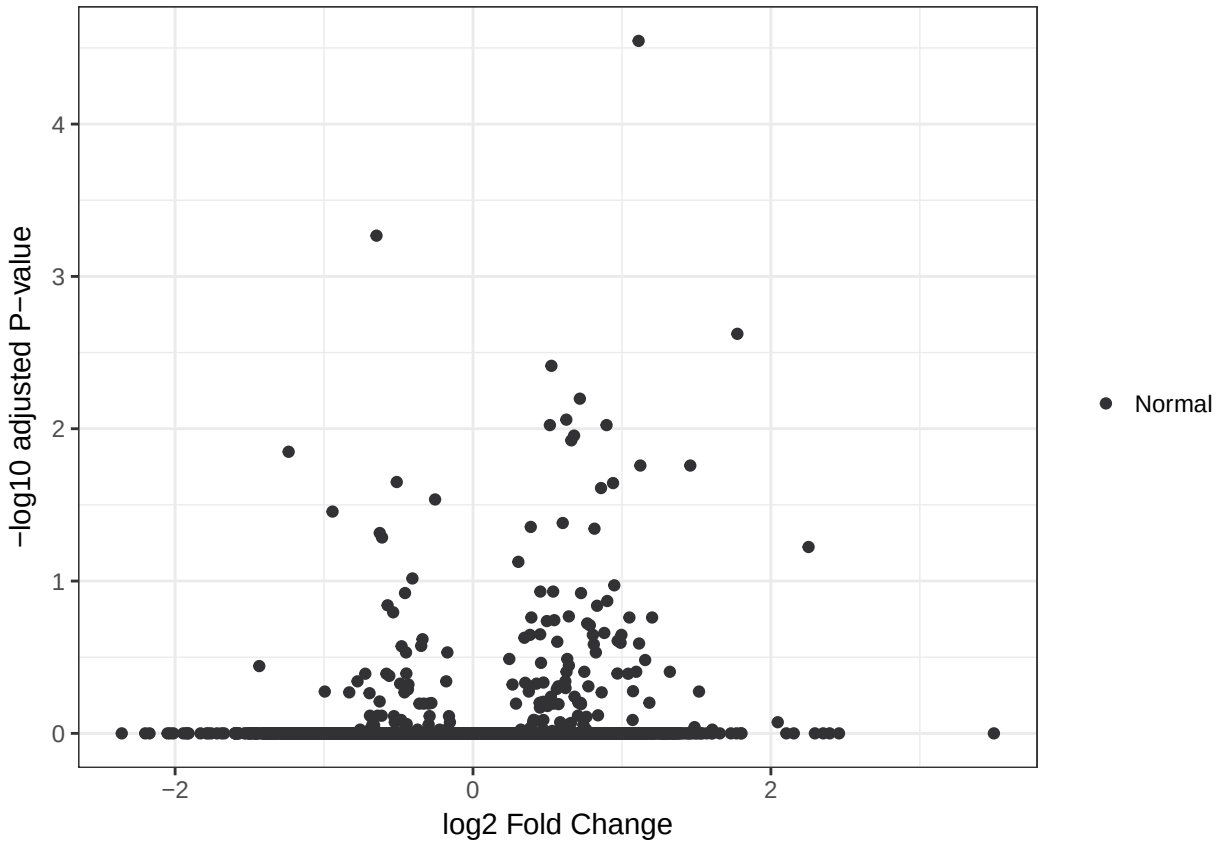
```
## Warning in split.default(sample(1:nelements), 1:chunks): data length is not
## a multiple of split variable
```

```
## [1] "Running DESeq..."
##
  |
  |                                                                      |   0%
  |
  |======================                                                |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## [1] "Differential expression complete!"
## [1] "Combining DE results..."
## [1] "Adjusting fold change values..."
## [1] "Condition: 3 vs 4 complete!"
```

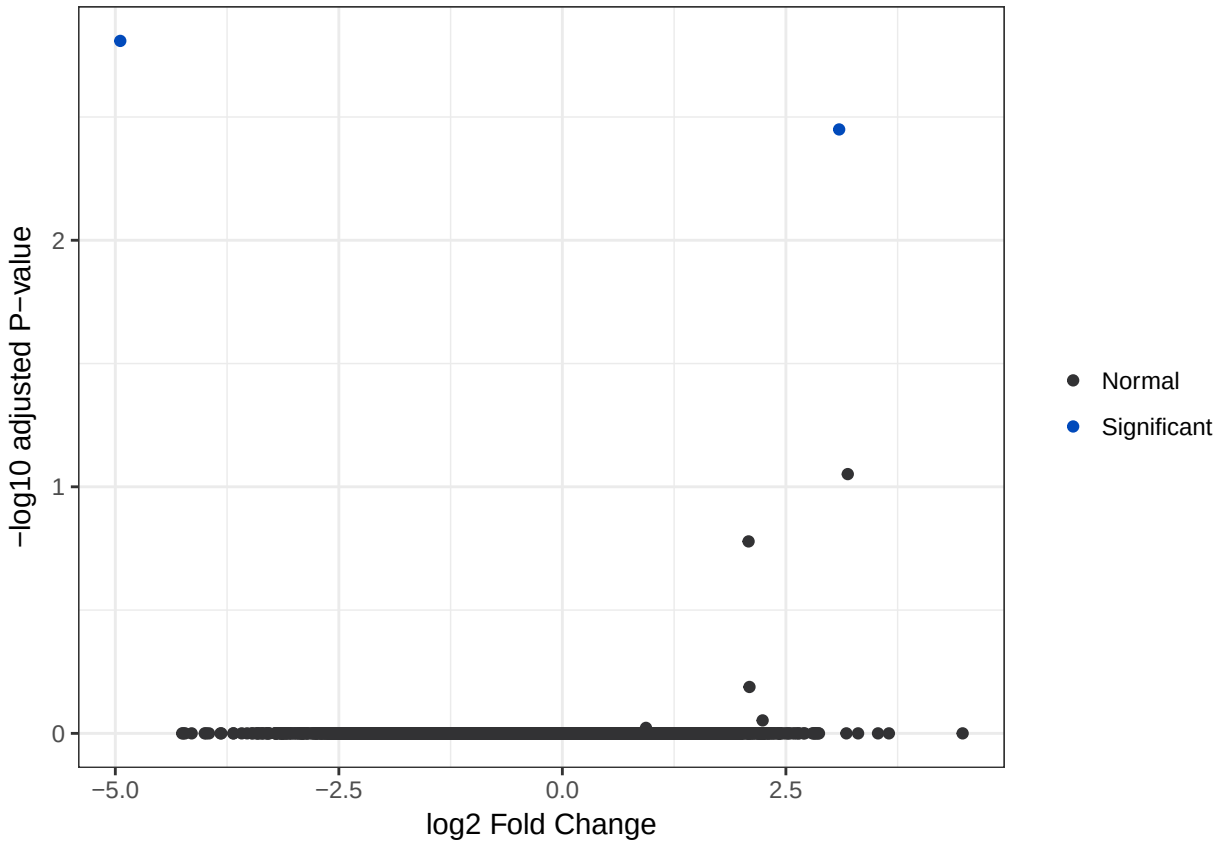
```
# Plot differential expression results
c1c2.plot <- PlotDEVolcano(c1c2.de.results, labels = FALSE)
c1c3.plot <- PlotDEVolcano(c1c3.de.results, labels = FALSE)
c1c4.plot <- PlotDEVolcano(c1c4.de.results, labels = FALSE)
c2c3.plot <- PlotDEVolcano(c2c3.de.results, labels = FALSE)
c2c4.plot <- PlotDEVolcano(c2c4.de.results, labels = FALSE)
c3c4.plot <- PlotDEVolcano(c3c4.de.results, labels = FALSE)

print(c1c2.plot)
```



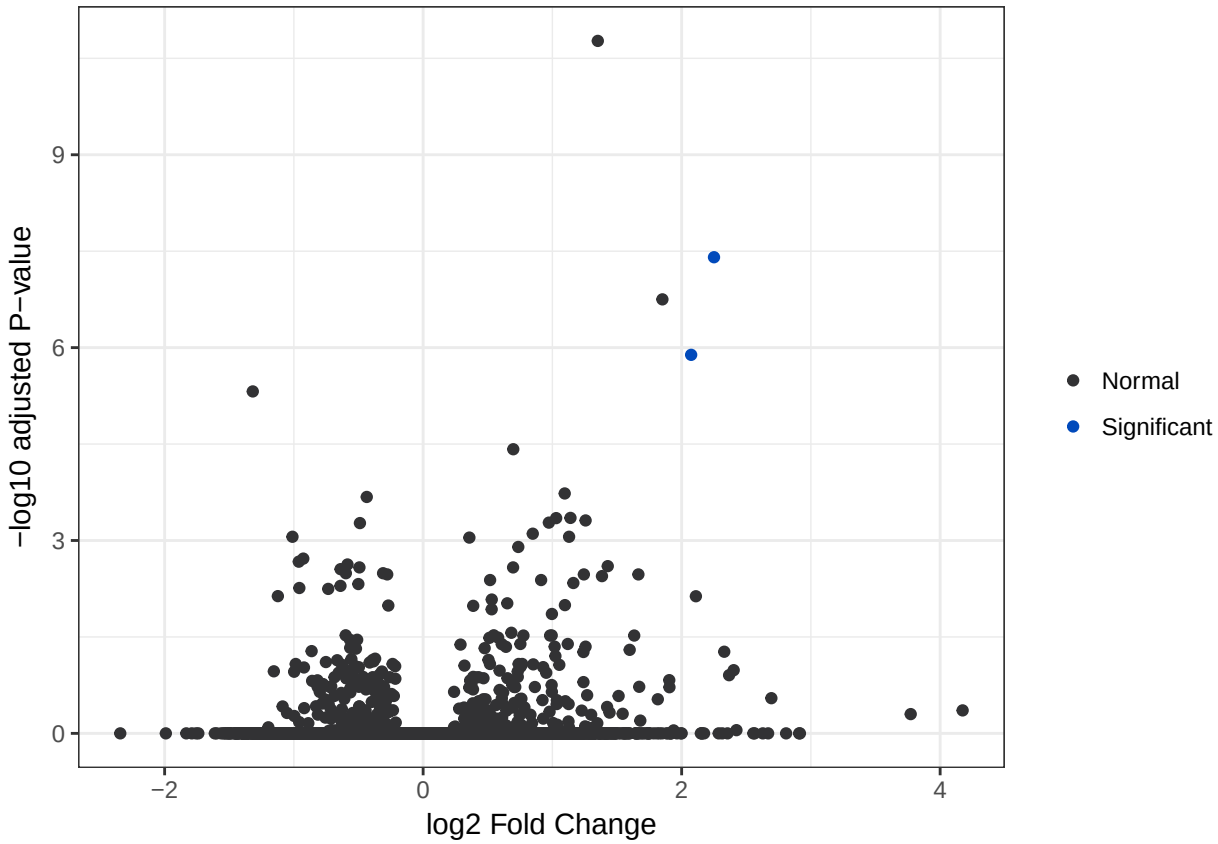

```
print(c1c3.plot)
```



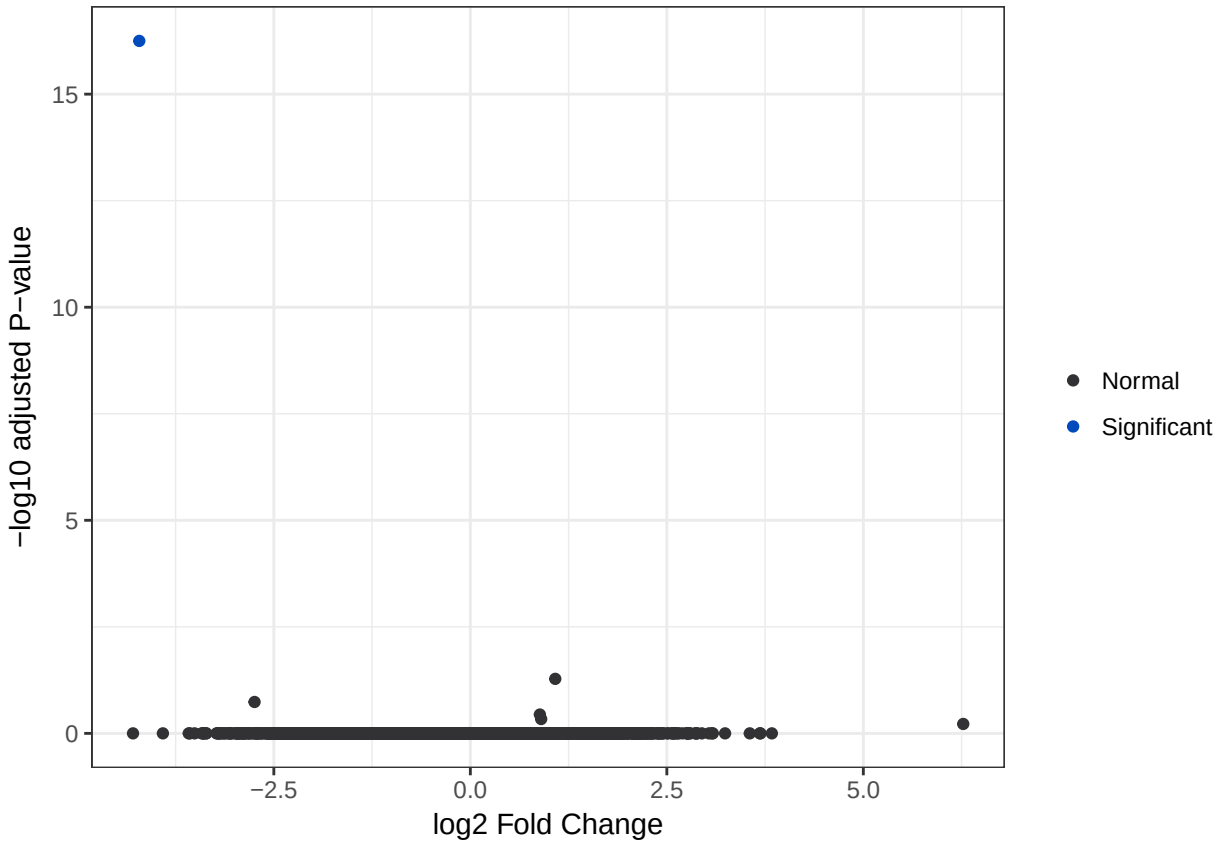

```
print(c1c4.plot)
```



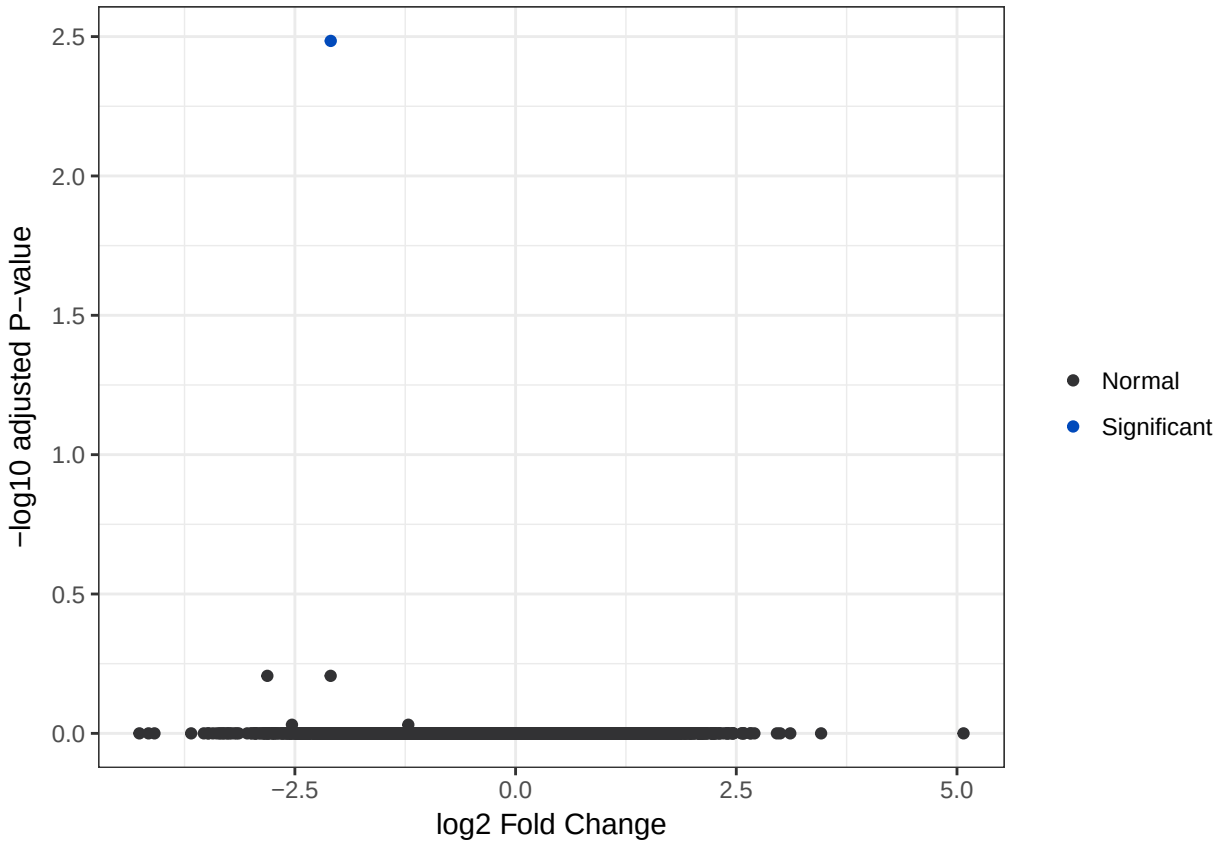

```
print(c2c3.plot)
```




```
print(c2c4.plot)
```




```
print(c3c4.plot)
```



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.