Single cell course 2018

Heli Pessa

September 5, 2018

For final version: remove knit opts from setup change file locations Todo: H: Quality control with Scater H: Normalization & PCA with Seurat

R Markdown

This R Markdown document contains examples and exercises for Single cell RNA-Seq analysis course at CSC on 21.9.2018. The code is partially based on Scater and Seurat package vignettes.

R Markdown is best opened in RStudio so that the code can be run and the results viewed within the document.

To run a line of code at the cursor, type Ctrl+Enter. To run a whole chunk, click on the green arrow in the upper right corner of the chunk. You can type the code for the exercises directly into the R console or in this document, which you can save. To insert a new code chunk for your code, type Ctrl+Alt+i.

Bioconductor

Bioconductor is a collection of R packages for the analysis of biological data. Bioconductor packages are installed and updated using biocLite():

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

QC with Scater

```
knitr::opts_chunk$set(echo = TRUE, cache = TRUE)
## Add cache.lazy = FALSE if getting errors
library(SingleCellExperiment)
```

```
## Loading required package: SummarizedExperiment
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: stats4
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind,
##     colMeans, colnames, colSums, dirname, do.call, duplicated,
##     eval, evalq, Filter, Find, get, grep, grepl, intersect,
##     is.unsorted, lapply, lengths, Map, mapply, match, mget, order,
##     paste, pmax, pmax.int, pmin, pmin.int, Position, rank, rbind,
##     Reduce, rowMeans, rownames, rowSums, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which, which.max,
##     which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

## Loading required package: GenomeInfoDb

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
```

```
## Loading required package: BiocParallel

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply
```

```
library(scater)
```

```
## Loading required package: ggplot2

##
## Attaching package: 'scater'

## The following object is masked from 'package:S4Vectors':
##
##     rename

## The following object is masked from 'package:stats':
##
##     filter
```

```
library(Seurat)
```

```
## Loading required package: cowplot

##
## Attaching package: 'cowplot'

## The following object is masked from 'package:ggplot2':
##
##     ggsave

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following object is masked from 'package:S4Vectors':
##
##     expand
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:scater':
##
##     arrange, filter, mutate, rename

## The following object is masked from 'package:matrixStats':
##
##     count

## The following object is masked from 'package:Biobase':
##
##     combine

## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union

## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect

## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
set.seed(1234567)
a9_path <- "/wrk/hpessa/DONOTREMOVE/sc_course/10X/A9/GRCh38/"
```

The datasets used here are from last year's course. They are human iPSCs induced towards pancreatic islet fate (A9: control, B9: treated). Timo Otonkoski group has kindly given us the permission to use them as examples, but please do not copy the data and take it with you.

We start by reading in the data. 10X count data is in MatrixMarket format and can be read into R using readMM() from Matrix package. Then we create the SingleCellExperiment object that scater functions work on.

```
cellbarcodes_A9 <- read.table(paste(a9_path, "barcodes.tsv", sep = ""))
genenames_A9 <- read.table(paste(a9_path, "genes.tsv", sep = ""))
counts_A9 <- Matrix::readMM(paste(a9_path, "matrix.mtx", sep = ""))

rownames(counts_A9) <- genenames_A9[,1]
colnames(counts_A9) <- cellbarcodes_A9[,1]
A9sce <- SingleCellExperiment(assays = list(counts = as.matrix(counts_A9)))
A9sce
```

```
## class: SingleCellExperiment
## dim: 33694 1083
## metadata(0):
## assays(1): counts
## rownames(33694): ENSG00000243485 ENSG00000237613 ...
##   ENSG00000277475 ENSG00000268674
## rowData names(0):
## colnames(1083): AAACGGGTCAACCATG-1 AAAGCAAAGGCCGAAT-1 ...
##   TTTGTCAAGGGCACTA-1 TTTGTCAGTTCACGGC-1
## colData names(0):
## reducedDimNames(0):
## spikeNames(0):
```

Remove genes that are not expressed in any cell:

```
keep_feature <- rowSums(counts(A9sce) > 0) > 0
A9sce <- A9sce[keep_feature,]
```

High proportion of mitochondrial transcripts is often a sign of a broken cell. We define a handful of commonly expressed mitochondrial genes as feature controls, which will later be used for QC.

```
mito.ids <- c("ENSG00000198899", "ENSG00000198727", "ENSG00000198888",
    "ENSG00000198886", "ENSG00000212907", "ENSG00000198786",
    "ENSG00000198695", "ENSG00000198712", "ENSG00000198804",
    "ENSG00000198763", "ENSG00000228253", "ENSG00000198938",
    "ENSG00000198840")
isSpike(A9sce, "MT") <- rownames(A9sce) %in% mito.ids

A9sce <- calculateQCMetrics(A9sce, feature_controls = list(MT = isSpike(A9sce, "MT")))
```

```
## Warning in calculateQCMetrics(A9sce, feature_controls = list(MT =
## isSpike(A9sce, : spike-in set 'MT' overwritten by feature_controls set of
## the same name
```

```
## Note that the names of some metrics have changed, see 'Renamed metrics' in ?calculateQCMetrics.
## Old names are currently maintained for back-compatibility, but may be removed in future releases.
```

```
A9sce
```

```
## class: SingleCellExperiment
## dim: 20317 1083
## metadata(0):
## assays(1): counts
## rownames(20317): ENSG00000238009 ENSG00000279457 ...
##   ENSG00000276345 ENSG00000271254
## rowData names(10): is_feature_control is_feature_control_MT ...
##   n_cells_counts pct_dropout_counts
## colnames(1083): AAACGGGTCAACCATG-1 AAAGCAAAGGCCGAAT-1 ...
##   TTTGTCAAGGGCACTA-1 TTTGTCAGTTCACGGC-1
## colData names(44): is_cell_control total_features_by_counts ...
##   total_features_MT log10_total_features_MT
## reducedDimNames(0):
## spikeNames(1): MT
```

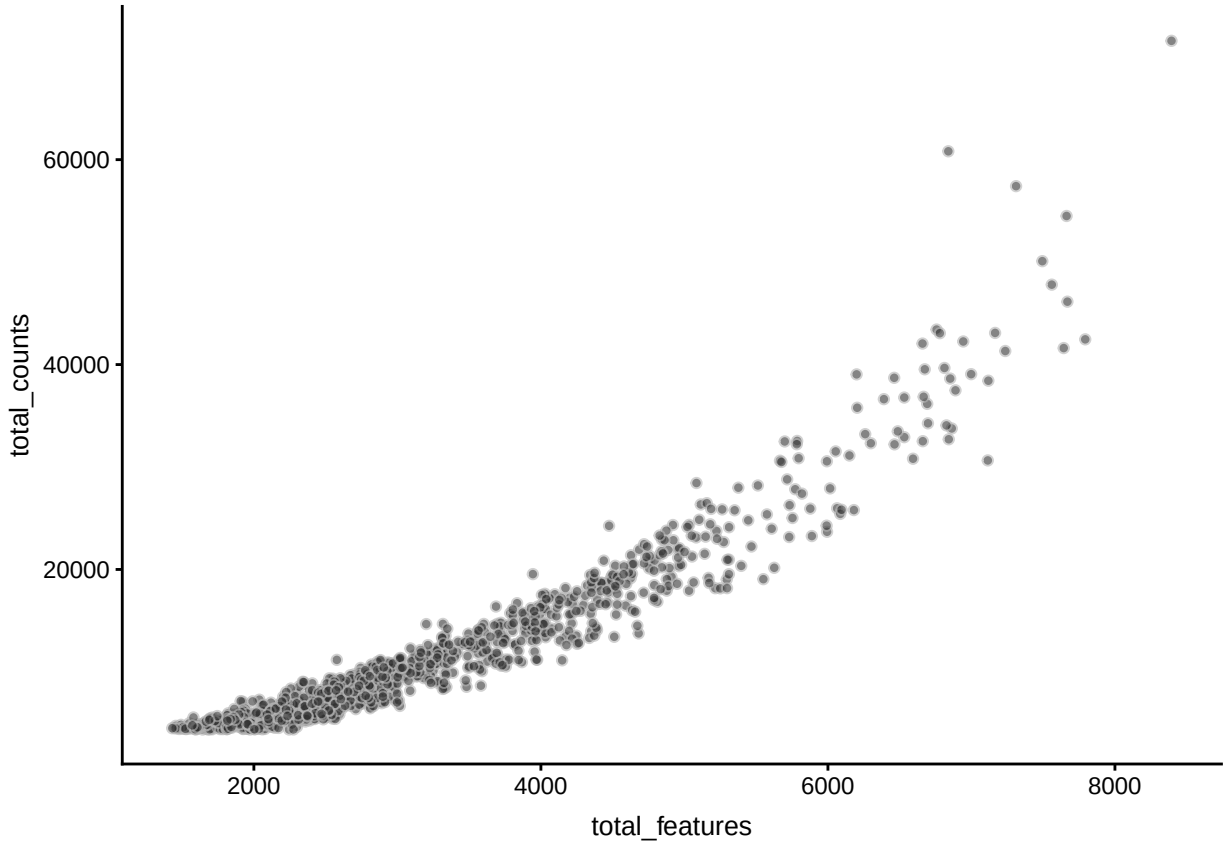
As you see above, calculateQCMetrics() has produced many useful calculations from the data and stored them in the SCE object.

```
names(colData(A9sce))
```

```
##  [1] "is_cell_control"
##  [2] "total_features_by_counts"
##  [3] "log10_total_features_by_counts"
##  [4] "total_counts"
##  [5] "log10_total_counts"
##  [6] "pct_counts_in_top_50_features"
##  [7] "pct_counts_in_top_100_features"
##  [8] "pct_counts_in_top_200_features"
##  [9] "pct_counts_in_top_500_features"
## [10] "total_features"
## [11] "log10_total_features"
## [12] "pct_counts_top_50_features"
## [13] "pct_counts_top_100_features"
## [14] "pct_counts_top_200_features"
## [15] "pct_counts_top_500_features"
## [16] "total_features_by_counts_endogenous"
## [17] "log10_total_features_by_counts_endogenous"
## [18] "total_counts_endogenous"
## [19] "log10_total_counts_endogenous"
## [20] "pct_counts_endogenous"
## [21] "pct_counts_in_top_50_features_endogenous"
## [22] "pct_counts_in_top_100_features_endogenous"
## [23] "pct_counts_in_top_200_features_endogenous"
## [24] "pct_counts_in_top_500_features_endogenous"
## [25] "total_features_endogenous"
## [26] "log10_total_features_endogenous"
## [27] "pct_counts_top_50_features_endogenous"
## [28] "pct_counts_top_100_features_endogenous"
## [29] "pct_counts_top_200_features_endogenous"
## [30] "pct_counts_top_500_features_endogenous"
## [31] "total_features_by_counts_feature_control"
```

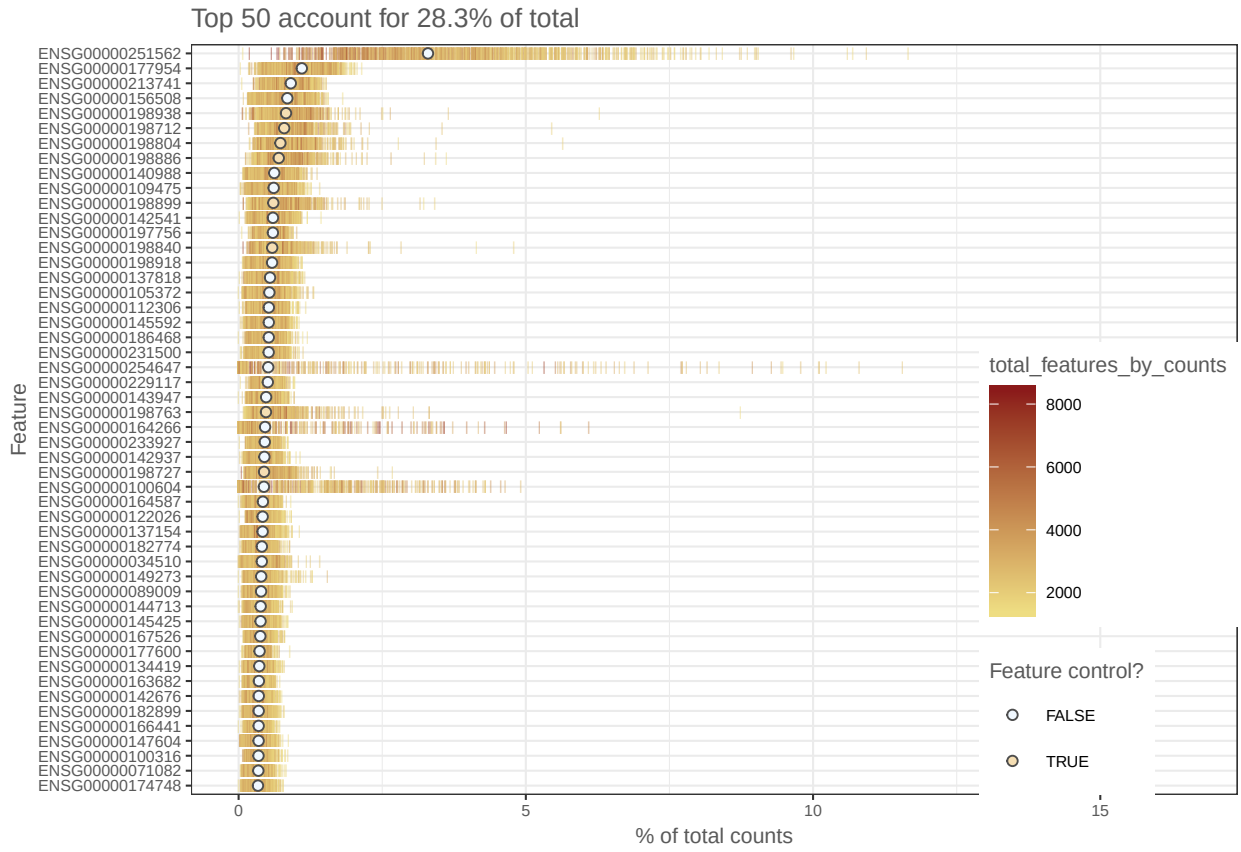
```
## [32] "log10_total_features_by_counts_feature_control"
## [33] "total_counts_feature_control"
## [34] "log10_total_counts_feature_control"
## [35] "pct_counts_feature_control"
## [36] "total_features_feature_control"
## [37] "log10_total_features_feature_control"
## [38] "total_features_by_counts_MT"
## [39] "log10_total_features_by_counts_MT"
## [40] "total_counts_MT"
## [41] "log10_total_counts_MT"
## [42] "pct_counts_MT"
## [43] "total_features_MT"
## [44] "log10_total_features_MT"
```

```
plotColData(A9sce, x = "total_features", y = "total_counts")
```



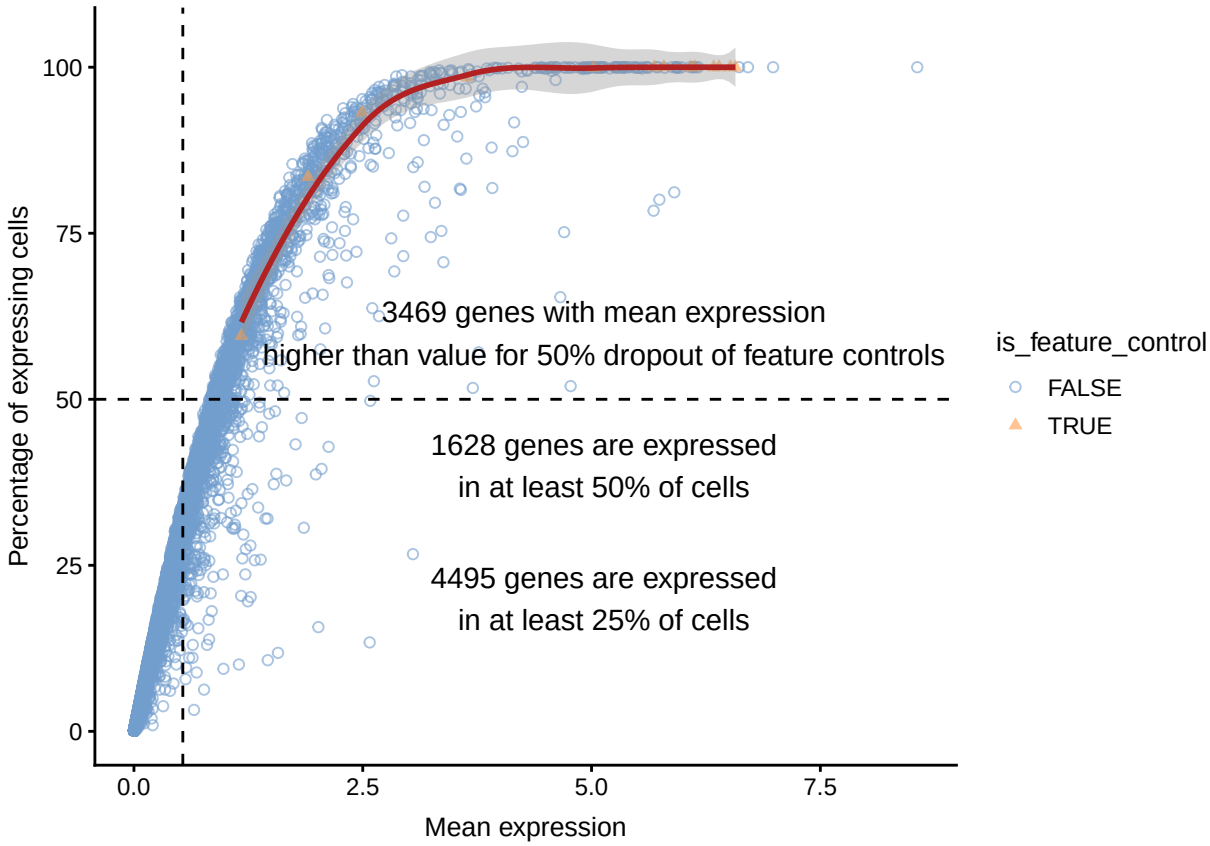
Genes with highest expression:

```
plotQC(A9sce, type = "highest-expression")
```




```
plotQC(A9sce, type = "exprs-freq-vs-mean")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

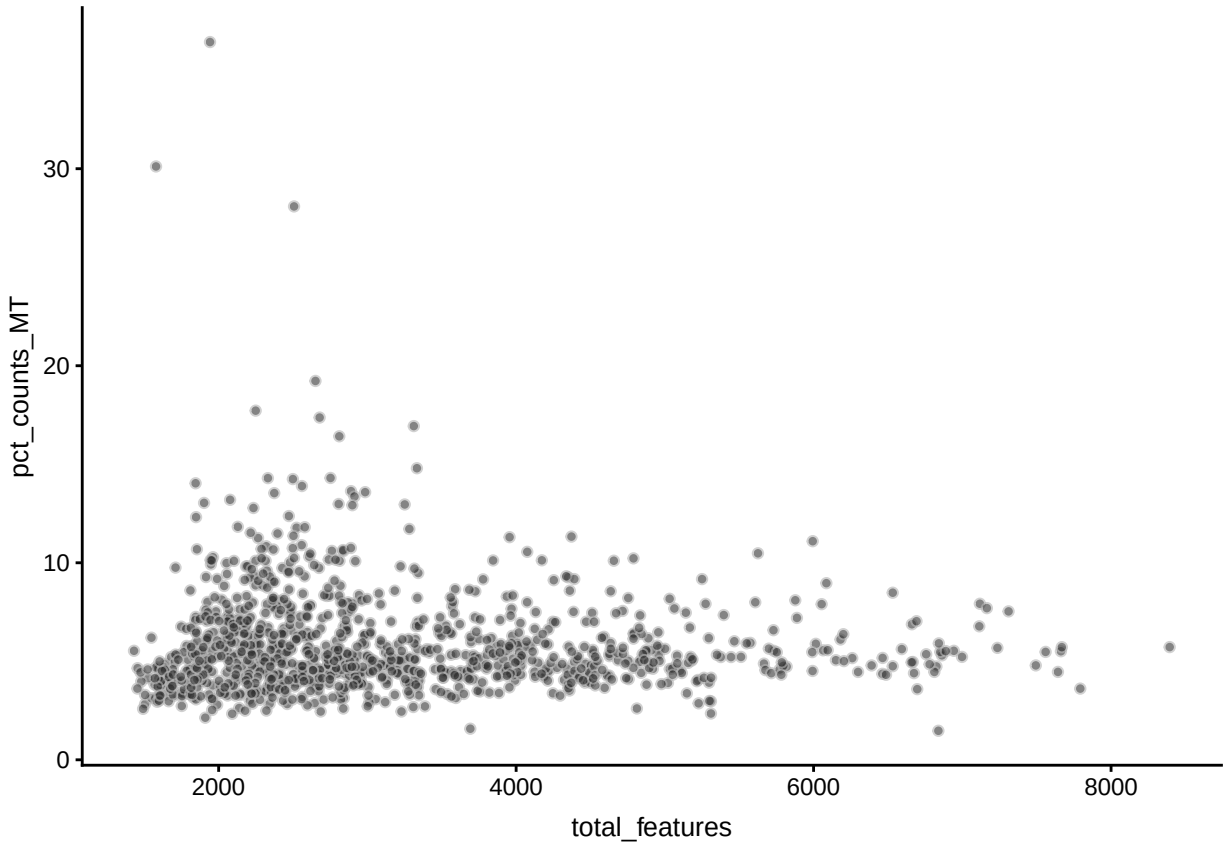


Exercise: explore some more colData slots using plotColData().

Cell QC

Barcodes with very low amounts of RNA may have arisen from beads that did not capture a cell but RNA from the sample buffer. On the other hand, outliers at the upper end of the total counts distribution can be from a clump of cells captured by one barcode bead. In addition, some cells contain a high percentage of mitochondrial transcripts:

```
plotColData(A9sce, x = "total_features", y = "pct_counts_MT")
```



As you can see from the plots, this data is already filtered to remove cells with lowest number of counts and features. Here, we will filter the data retaining only cells that have at least 5000 total counts and at least 500 expressed features:

```
keep.total <- A9sce$total_counts > 5000
keep.n <- A9sce$total_features_by_counts > 500
keep.mt <- A9sce$pct_counts_MT < 10
A9_filtered <- A9sce[,keep.total & keep.n & keep.mt]
```

How many cells were filtered? Do you think the thresholds are sensible? Did the top expressed genes change?

Gene QC

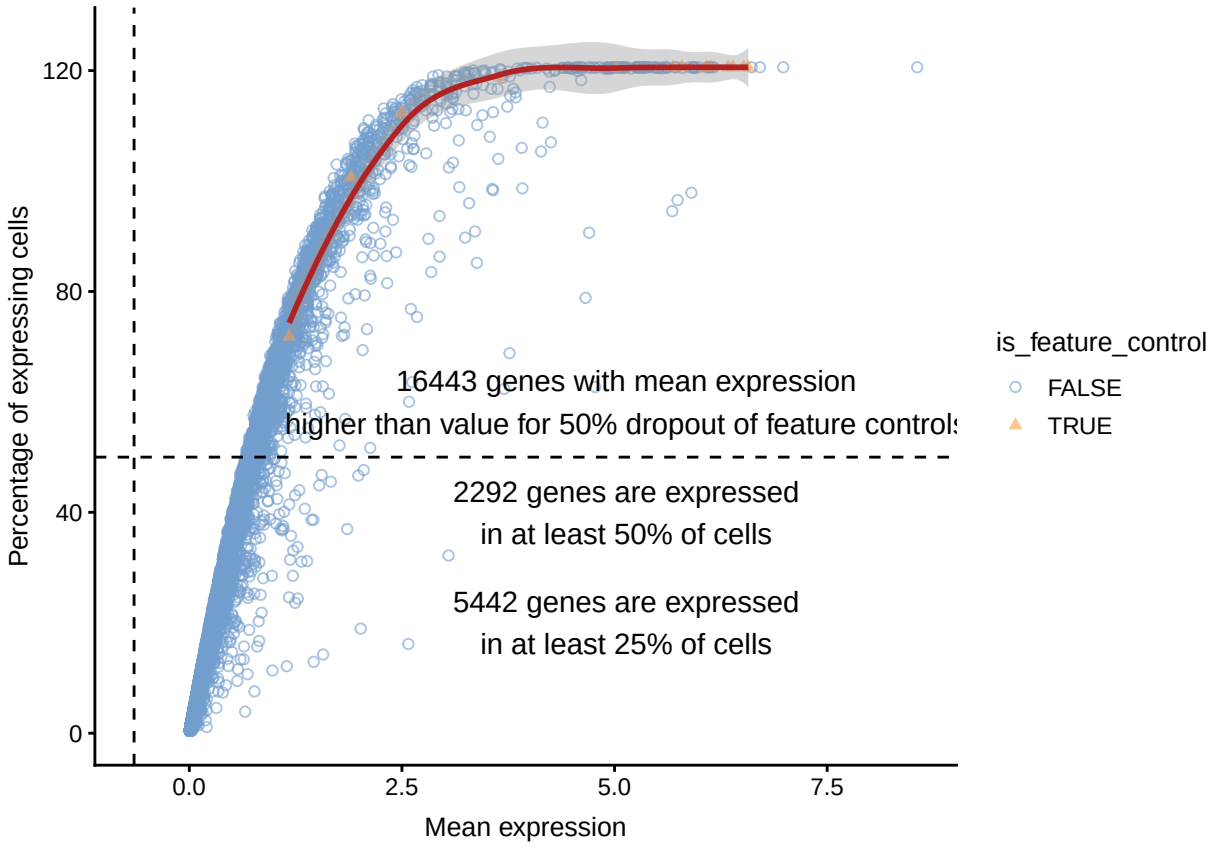
We retain genes only if they are expressed in at least four cells:

```
keep_feature <- nexprs(A9_filtered, byrow = TRUE) >= 4
A9_filtered <- A9_filtered[keep_feature,]
```

How many genes were filtered?

```
plotQC(A9_filtered, type = "exprs-freq-vs-mean")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Examine the filtered dataset. Do you see differences compared to the unfiltered data?

Extra exercise, if there is time: perform QC to the Seq-Well dataset. It is a tsv-separated text file and can be read in using read.table(). Tip: you may have to change the filtering thresholds.

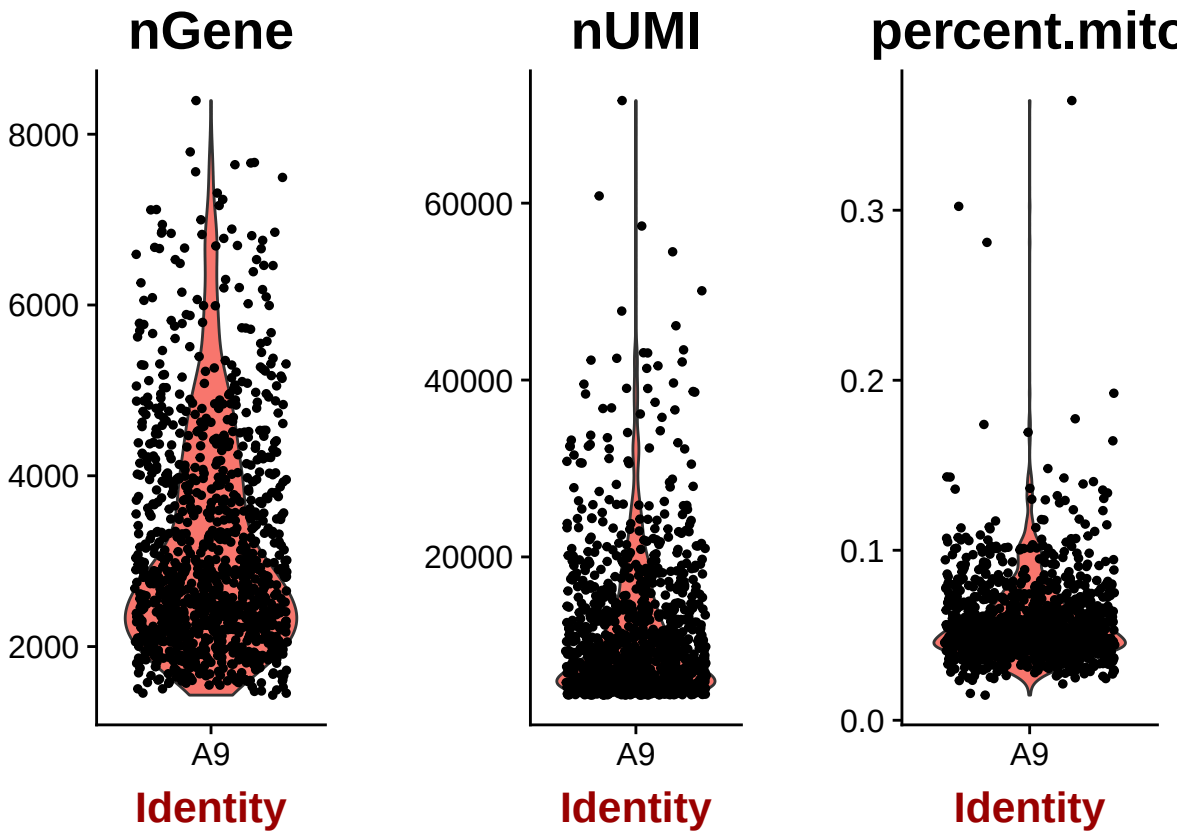
Filtering and normalization with Seurat

We could use the already filtered count matrix from the SingleCellExperiment object as raw data for Seurat. Here we will anyway start from the original data and perform filtering again using Seurat's own functionality. We keep cells with at least 500 transcripts and genes that are expressedd in at least 4 cells.

```
a9.mat <- Read10X(a9_path)

min.genes <- 200
min.cells <- 4

a9 <- CreateSeuratObject(raw.data = a9.mat, min.cells = min.cells, min.genes = min.genes,
    project = "A9")
```

We calculate percentage of mitochondrial transcripts per cell and store it in the metadata slot of the Seurat object:

```
mito.genes = grep("^MT-", rownames(a9@data), value = TRUE)
percent.mito = Matrix::colSums(a9@raw.data[mito.genes,]) / Matrix::colSums(a9@raw.data)
a9 <- AddMetaData(a9, metadata = percent.mito, col.name = "percent.mito")
VlnPlot(a9, features.plot = c("nGene", "nUMI", "percent.mito"), nCol = 3)
```



Metadata can also be used in filtering:

```
a9 <- FilterCells(a9, subset.names = c("nGene", "percent.mito"),
    low.thresholds = c(500, -Inf), high.thresholds = c(Inf, 0.1))
```

The number of transcripts captured per cell varies widely, so normalization is required to make the cells comparable. Seurat implements several normalization methods. We will use LogNormalize, which normalizes the gene expression measurements for each cell by total counts, multiplies this by a scale factor, and log-transforms the result.

```
a9 <- NormalizeData(a9, normalization.method = "LogNormalize", scale.factor = 10000)
```

Removing unwanted sources of variation

Your data may contain uninteresting variation, such as batch effects or cell cycle-induced variation. They can be regressed out by Seurat using linear models to predict gene expression based on user-defined variables. Here we will just remove the variation caused by different number of detected molecules per cell as well as the percentage of mitochondrial transcripts.

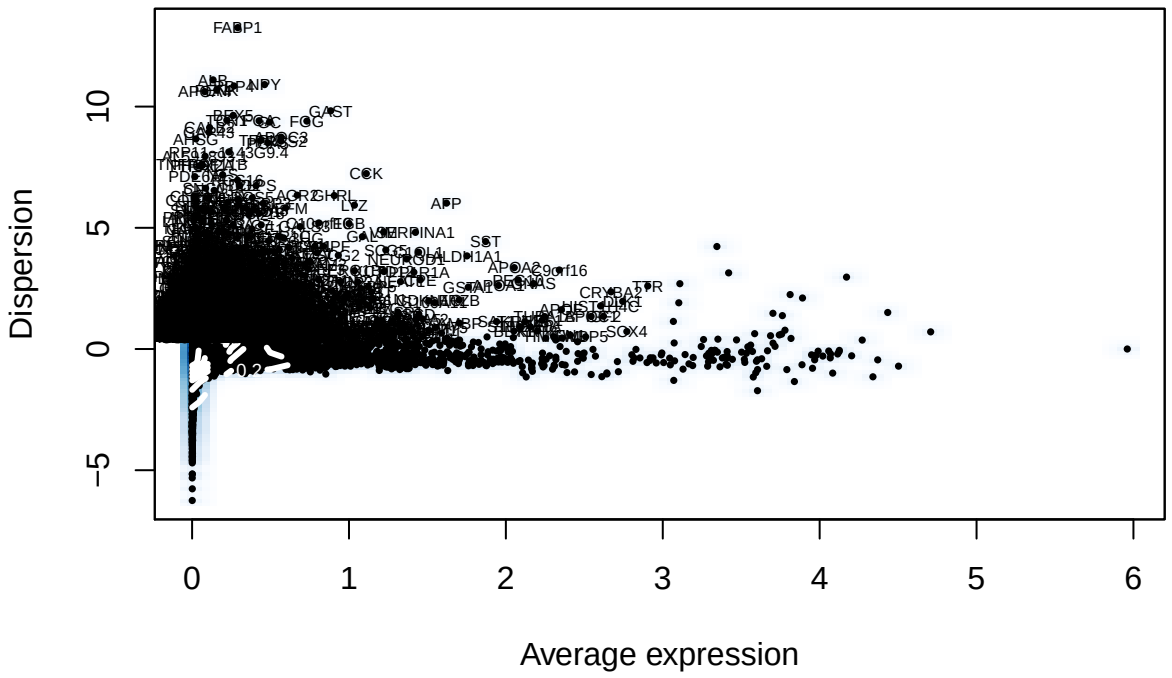
```
a9 <- ScaleData(a9, vars.to.regress = c("nUMI", "percent.mito"))
```

```
## Regressing out: nUMI, percent.mito

##
## Time Elapsed:  18.6997449398041 secs

## Scaling data matrix
```

Choosing variable genes

Most genes in the data are expressed at very low levels or do not show significant variation above noise. For downstream analyses, we need to identify highly variable genes.

```
a9 <- FindVariableGenes(a9, mean.function = ExpMean, dispersion.function = LogVMR,
    x.low.cutoff = 0.0125, x.high.cutoff = 3, y.cutoff = 0.5)
```



How is the relationship between expression level and variation? Do you see interesting genes?

```
length(a9@var.genes)
```

```
## [1] 2472
```

PCA

We examine the variation in the data using principal component analysis (PCA).

```
a9 <- RunPCA(a9, pc.genes = a9@var.genes, do.print = TRUE, pcs.print = 1:5,
    genes.print = 5)
```

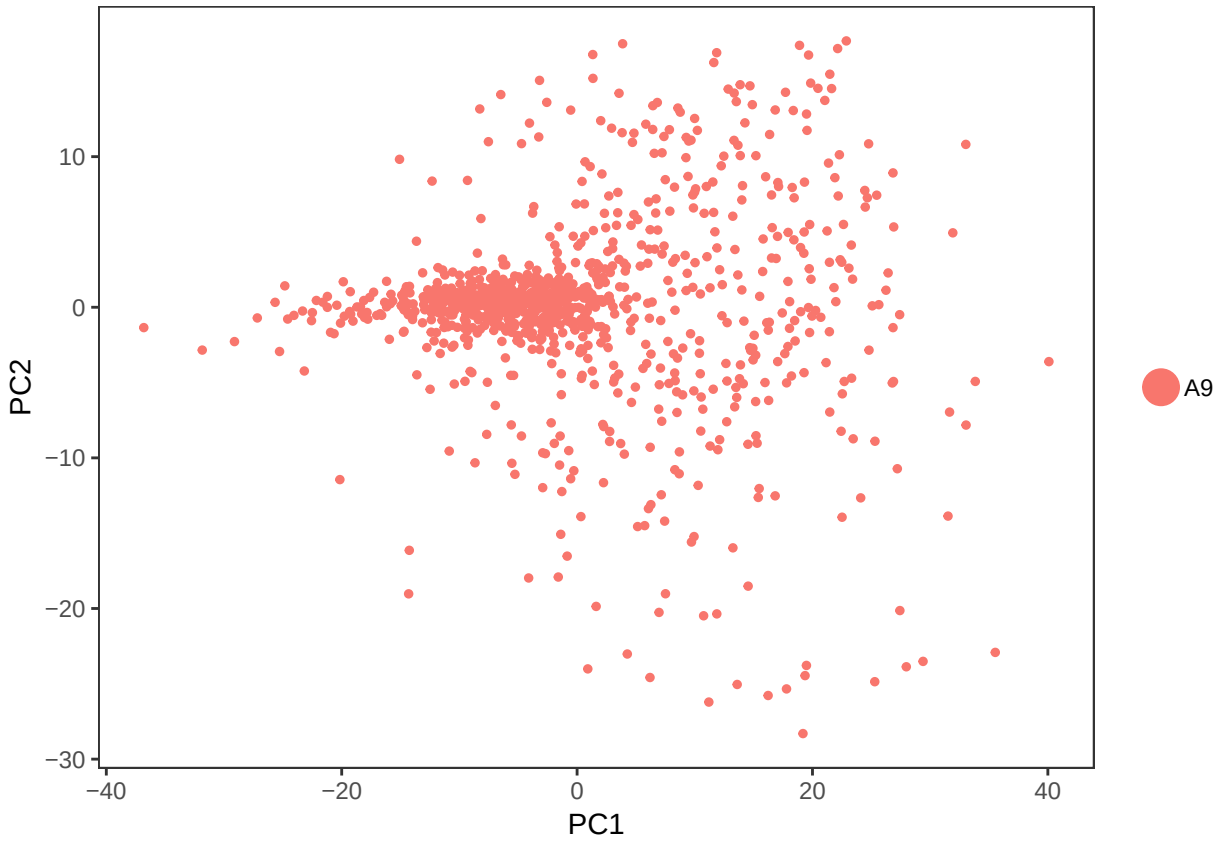
```
## [1] "PC1"
## [1] "HMGA1" "HES1"  "KRT18" "ANXA2" "FN1"
## [1] ""
## [1] "NEUROD1" "C1QL1"   "CRYBA2"  "PPP1R1A" "KCNK16"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "SOX4"    "USP18"   "PGA5"    "SUSD2"   "NEUROG3"
## [1] ""
```

```
## [1] "SLC30A8" "AKAP12"  "FAM46A"  "PTGER3"  "APOA2"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "AFP"    "RNASE1" "FGB"    "FGG"    "LGALS3"
## [1] ""
## [1] "SLC30A8" "STMN2"   "ARX"     "ERO1B"   "SCG3"
## [1] ""
## [1] ""
## [1] "PC4"
## [1] "DDC"    "SYT13"  "PRLHR"  "TGFBR3" "ASCL2"
## [1] ""
## [1] "TTR"      "CLU"      "C10orf10" "PGA5"     "MLLT11"
## [1] ""
## [1] ""
## [1] "PC5"
## [1] "SERPINF1" "MYLIP"    "IGF2"     "APOE"     "FRZB"
## [1] ""
## [1] "TOP2A"  "UBE2C"  "CENPF"  "HMGB2"  "NUSAP1"
## [1] ""
## [1] ""
```

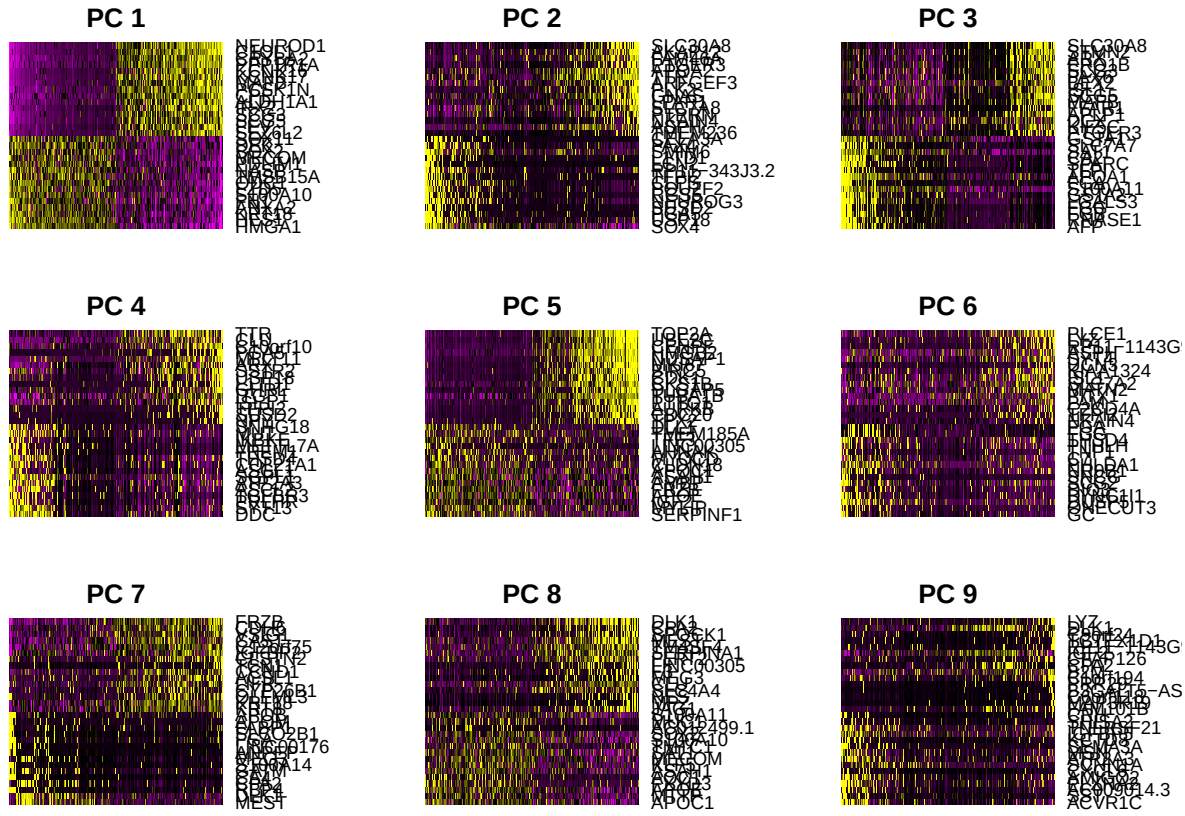
```
PrintPCA(a9, pcs.print = 1:4, genes.print = 5, use.full = FALSE)
```

```
## [1] "PC1"
## [1] "HMGA1" "HES1"  "KRT18" "ANXA2" "FN1"
## [1] ""
## [1] "NEUROD1" "C1QL1"   "CRYBA2"  "PPP1R1A" "KCNK16"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "SOX4"    "USP18"   "PGA5"    "SUSD2"   "NEUROG3"
## [1] ""
## [1] "SLC30A8" "AKAP12"  "FAM46A"  "PTGER3"  "APOA2"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "AFP"    "RNASE1" "FGB"    "FGG"    "LGALS3"
## [1] ""
## [1] "SLC30A8" "STMN2"   "ARX"     "ERO1B"   "SCG3"
## [1] ""
## [1] ""
## [1] "PC4"
## [1] "DDC"    "SYT13"  "PRLHR"  "TGFBR3" "ASCL2"
## [1] ""
## [1] "TTR"      "CLU"      "C10orf10" "PGA5"     "MLLT11"
## [1] ""
## [1] ""
```

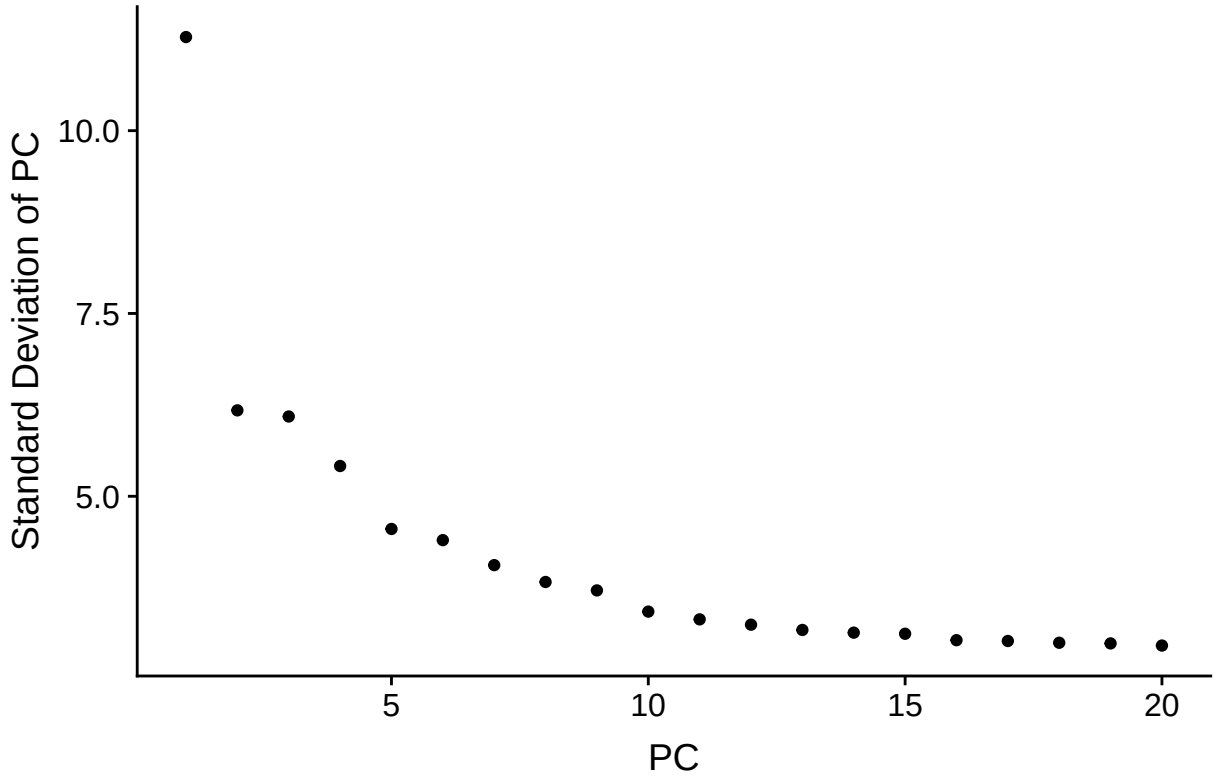
```
PCAPlot(object = a9, dim.1 = 1, dim.2 = 2)
```




```
PCHeatmap(a9, pc.use = 1:9, cells.use = 500, do.balanced = TRUE,
    label.columns = FALSE, use.full = FALSE)
```



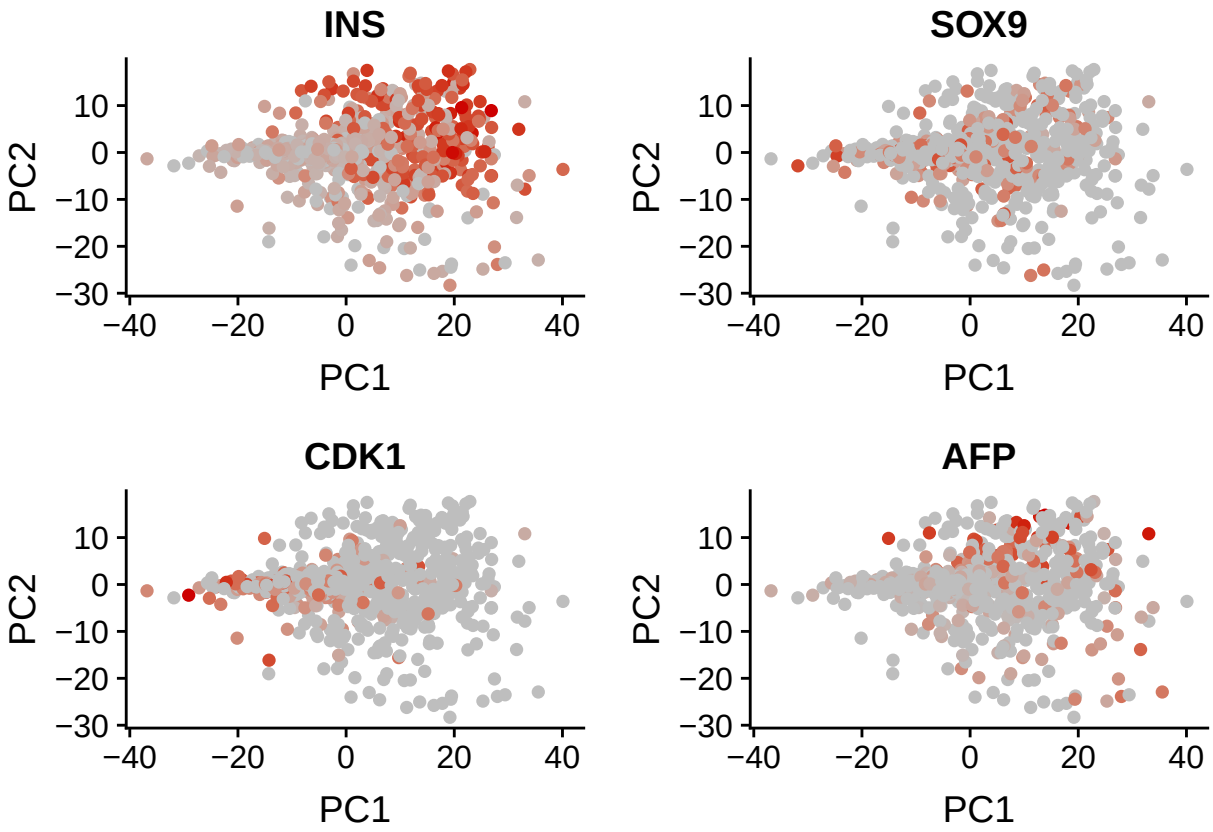

```
PCElbowPlot(a9)
```



Plot the most promising-looking PCs with PCAPlot(). Which ones seem to represent the overall variation best? Are there others that separate a distinct cluster of cells?

Several marker genes should be upregulated in differentiating cells at least in the treated sample, possibly even in the control.

```
FeaturePlot(a9, features.plot = c("INS", "SOX9", "CDK1", "AFP"), cols.use = c("grey", "red3"),
    reduction.use = "pca", pt.size = 2)
```



Repeat the above using the treated sample B9. Compare the genes driving the first PCs between the two samples.

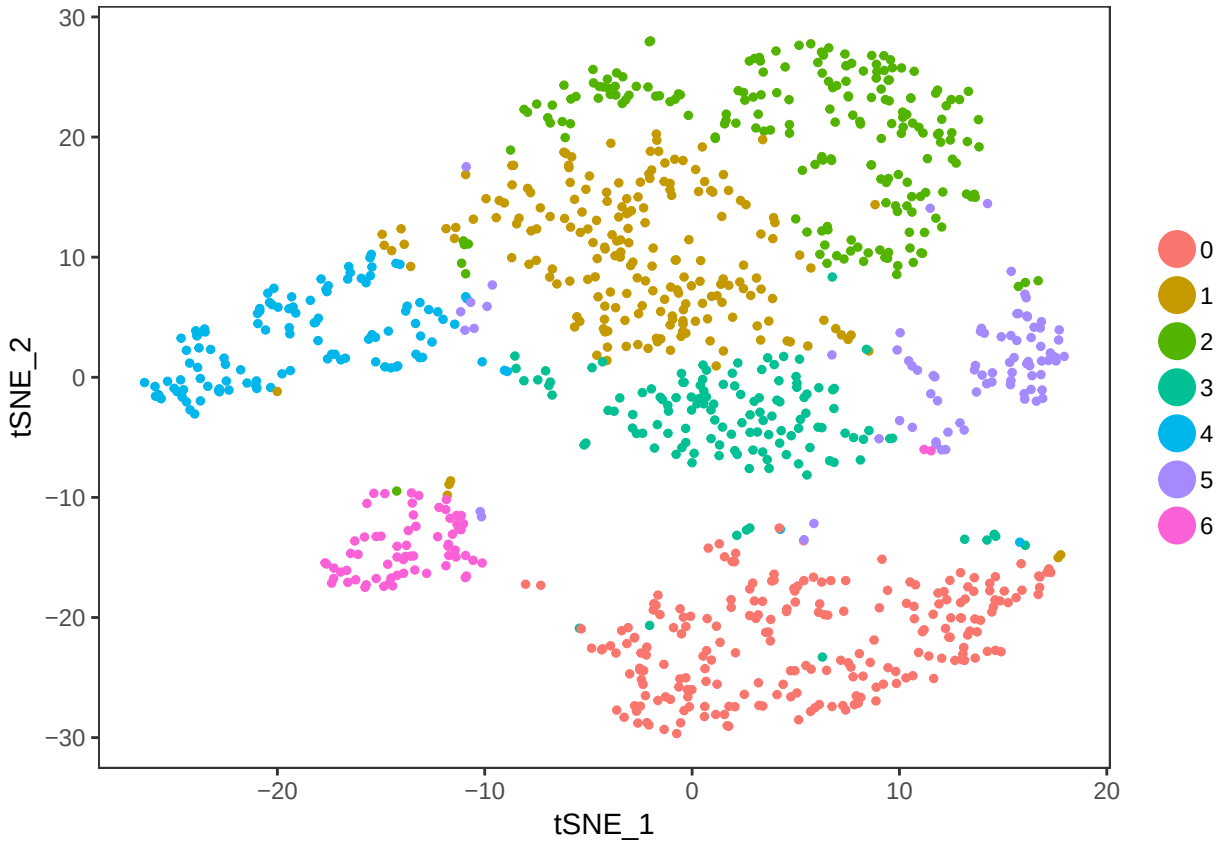
Clustering cells

Choosing PCs is critical for clustering. Use the methods above to examine the PCs to find the ones with most biologically relevant variation and decide which ones to use.

```
a9 <- FindClusters(a9, reduction.type = "pca", dims.use = 1:9,
    resolution = 0.6, print.output = 0, save.SNN = TRUE)
a9 <- RunTSNE(a9, dims.use = 1:9, do.fast = TRUE)
```

T-SNE is a great way to visualize clustering results but it can also mislead. This paper presents some of the difficulties in interpreting t-SNE plots.

```
TSNEPlot(a9)
```



Genes enriched in specific clusters can be used as cluster biomarkers.

```
a9.markers = FindAllMarkers(a9, only.pos = TRUE, min.pct = 0.25, thresh.use = 0.25)
a9.markers %>% group_by(cluster) %>% top_n(4, avg_logFC)
```

```
## # A tibble: 28 x 7
## # Groups:   cluster [7]
##        p_val avg_logFC pct.1 pct.2 p_val_adj cluster gene
##        <dbl>     <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 3.43e-121      2.91  0.995 0.352 5.73e-117 0       CRYBA2
##  2 1.13e-110      3.05  1     0.709 1.89e-106 0       CHGA
##  3 3.77e- 84      3.65  0.964 0.761 6.29e- 80 0       INS
##  4 2.93e- 14      3.44  0.609 0.493 4.88e- 10 0       GCG
##  5 1.00e- 25      0.612 0.96  0.683 1.68e- 21 1       HES1
##  6 7.60e- 25      0.482 0.965 0.749 1.27e- 20 1       MAGI1
##  7 6.33e- 23      0.520 0.876 0.569 1.06e- 18 1       SOX2
##  8 4.32e- 22      0.479 0.945 0.698 7.21e- 18 1       ZFP36L1
##  9 9.00e-111      1.74  0.925 0.175 1.50e-106 2       TOP2A
## 10 2.30e-101      1.52  0.899 0.201 3.85e- 97 2       CENPF
## # ... with 18 more rows
```

Exercise: perform the above steps on the treated sample B9. Are the resulting clusters similar or different?

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
```

```
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: CentOS release 6.10 (Final)
##
## Matrix products: default
## BLAS/LAPACK: /homeappl/appl_taito/opt/mkl/11.3.0/compilers_and_libraries_2016.0.109/linux/mkl/lib/in
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] bindrcpp_0.2.2              dplyr_0.7.6
##  [3] Seurat_2.3.4                Matrix_1.2-14
##  [5] cowplot_0.9.3               scater_1.8.4
##  [7] ggplot2_3.0.0               SingleCellExperiment_1.2.0
##  [9] SummarizedExperiment_1.10.1 DelayedArray_0.6.4
## [11] BiocParallel_1.14.2         matrixStats_0.54.0
## [13] Biobase_2.40.0              GenomicRanges_1.32.6
## [15] GenomeInfoDb_1.16.0         IRanges_2.14.10
## [17] S4Vectors_0.18.3            BiocGenerics_0.26.0
##
## loaded via a namespace (and not attached):
##   [1] snow_0.4-2               backports_1.1.2
##   [3] Hmisc_4.1-1              igraph_1.2.2
##   [5] plyr_1.8.4               lazyeval_0.2.1
##   [7] shinydashboard_0.7.0     splines_3.5.1
##   [9] digest_0.6.16            foreach_1.4.4
##  [11] htmltools_0.3.6          viridis_0.5.1
##  [13] lars_1.2                 fansi_0.2.3
##  [15] gdata_2.18.0             magrittr_1.5
##  [17] checkmate_1.8.5          cluster_2.0.7-1
##  [19] mixtools_1.1.0           ROCR_1.0-7
##  [21] limma_3.36.2             R.utils_2.6.0
##  [23] colorspace_1.3-2         jsonlite_1.5
##  [25] crayon_1.3.4             RCurl_1.95-4.11
##  [27] tximport_1.8.0           bindr_0.1.1
##  [29] zoo_1.8-3                survival_2.42-3
##  [31] iterators_1.0.10         ape_5.1
##  [33] glue_1.3.0               gtable_0.2.0
##  [35] zlibbioc_1.26.0          XVector_0.20.0
##  [37] kernlab_0.9-26           Rhdf5lib_1.2.1
##  [39] prabclus_2.2-6           DEoptimR_1.0-8
##  [41] scales_1.0.0             mvtnorm_1.0-8
##  [43] edgeR_3.22.3             bibtex_0.4.2
##  [45] Rcpp_0.12.18             metap_1.0
##  [47] dtw_1.20-1               viridisLite_0.3.0
```

```
##  [49] xtable_1.8-2             htmlTable_1.12
##  [51] reticulate_1.10          bit_1.1-14
##  [53] foreign_0.8-70           proxy_0.4-22
##  [55] mclust_5.4.1             SDMTools_1.1-221
##  [57] Formula_1.2-3            tsne_0.1-3
##  [59] httr_1.3.1               htmlwidgets_1.2
##  [61] gplots_3.0.1             RColorBrewer_1.1-2
##  [63] fpc_2.1-11.1             acepack_1.4.1
##  [65] modeltools_0.2-22        ica_1.0-2
##  [67] pkgconfig_2.0.1          R.methodsS3_1.7.1
##  [69] flexmix_2.3-14           nnet_7.3-12
##  [71] utf8_1.1.4               locfit_1.5-9.1
##  [73] labeling_0.3             tidyselect_0.2.4
##  [75] rlang_0.2.2              reshape2_1.4.3
##  [77] later_0.7.3              munsell_0.5.0
##  [79] tools_3.5.1              cli_1.0.0
##  [81] ggridges_0.5.0           evaluate_0.11
##  [83] stringr_1.3.1            yaml_2.2.0
##  [85] npsurv_0.4-0             bit64_0.9-7
##  [87] knitr_1.20               fitdistrplus_1.0-11
##  [89] robustbase_0.93-2        caTools_1.17.1.1
##  [91] purrr_0.2.5              RANN_2.6
##  [93] pbapply_1.3-4            nlme_3.1-137
##  [95] mime_0.5                 R.oo_1.22.0
##  [97] hdf5r_1.0.0              compiler_3.5.1
##  [99] rstudioapi_0.7           png_0.1-7
## [101] beeswarm_0.2.3           lsei_1.2-0
## [103] tibble_1.4.2             stringi_1.2.4
## [105] lattice_0.20-35          trimcluster_0.1-2.1
## [107] pillar_1.3.0             lmtest_0.9-36
## [109] Rdpack_0.9-0             irlba_2.3.2
## [111] data.table_1.11.4        bitops_1.0-6
## [113] gbRd_0.4-11              httpuv_1.4.5
## [115] R6_2.2.2                 latticeExtra_0.6-28
## [117] promises_1.0.1           KernSmooth_2.23-15
## [119] gridExtra_2.3            vipor_0.4.5
## [121] codetools_0.2-15         MASS_7.3-50
## [123] gtools_3.8.1             assertthat_0.2.0
## [125] rhdf5_2.24.0             rprojroot_1.3-2
## [127] rjson_0.2.20             withr_2.1.2
## [129] GenomeInfoDbData_1.1.0   diptest_0.75-7
## [131] doSNOW_1.0.16            grid_3.5.1
## [133] rpart_4.1-13             tidyr_0.8.1
## [135] class_7.3-14             rmarkdown_1.10
## [137] DelayedMatrixStats_1.2.0 segmented_0.5-3.0
## [139] Rtsne_0.13               shiny_1.1.0
## [141] base64enc_0.1-3          ggbeeswarm_0.6.0
```