R code for Parada lab

By Elsa Vera and Ben Young

June 8th 2018

Summary

This is a detailed documentation of the code used for analyzing our single cell RNA sequencing data (10X Chromium, done by Cornell Epigenomics Core) of adult mouse dentate gyrus.

For the first part of the analysis we used the Seurat package for R. Before installing the package, make sure you install the latest version of R and R Studio in advance.

The Seurat package was used for pre-processing the data, clustering the cells, identifying the clusters and the markers that constitute the cluster-specific signatures. The Seurat package also includes very useful ways of visualizing the data including heatmaps, violin plots, dot plots and feature plots.

For the "pseudotime" analysis we used the Monocle package. The key of combining these two packages is transforming our original "Seurat object" (Seurat database) into a "Monocle object" to leverage the pre-processing (filtering) and clustering done with Seurat.

NOTE: This tutorial includes our own code, tips, and some of Seurat/Monocle's code and documentation adapted to meet our needs. It is intended to be a guide to help other scientists leverage these tools to extract insights from their data. This guide includes unpublished data and is **not** intended to be used for submissions or citations. A more detailed formal submission based on the tools used in this guide is forthcoming.

Seurat

Load libraries and Setup

Install the package 'Seurat' (one time) and load the libraries every time you want to run this code:

```
#install.packages('Seurat')
library(Seurat)
library(dplyr)
library(Matrix)
library(stringr)
```

These are the location where we are going to pull our data from and store our plots in. It assumes your script is in a folder called "src" at the same level of your 10X results folder.

```
project_folder <- "/Users/elsa/Documents/Work/Current/10X/src/.."
analysis_folder <- file.path(project_folder,"Results/Sample234")
```

Data Load

Load Seurat object if it exist (skip this step the first time you are running the code)

```
pbmc_filename = file.path(project_folder,
                          'Negative_Low_High_filtering_h_0417.Robj')
pbmc_markers_filename = file.path(project_folder,
                                  'Negative_Low_High_filtering_h_AllMarkers_res1_4_0418.Robj')
```

```
# Uncomment to load a previous analysis
# load( file = pbmc_filename )
load( file = pbmc_markers_filename )
```

Load the dataset (load the filtered_gene_matrices of your 10X experiment) and create a Seurat object. Upon creation we keep all genes expressed in >= 3 cells (~0.1% of the data) and all cells with at least 200 detected genes per the Seurat guidelines. Note that we preserved the name of the Seurat object "pbmc" in the Seurat tutorial for code reuse purposes.

```
pbmc.data <-
  Read10X("/Volumes/Elsa Work/10X/Results/Sample234/outs/filtered_gene_bc_matrices_mex/mm10ECEV4806")
pbmc <-
  CreateSeuratObject(raw.data = pbmc.data, min.cells = 3, min.genes = 200, project = "10X_DG")
```

Define our valid cells

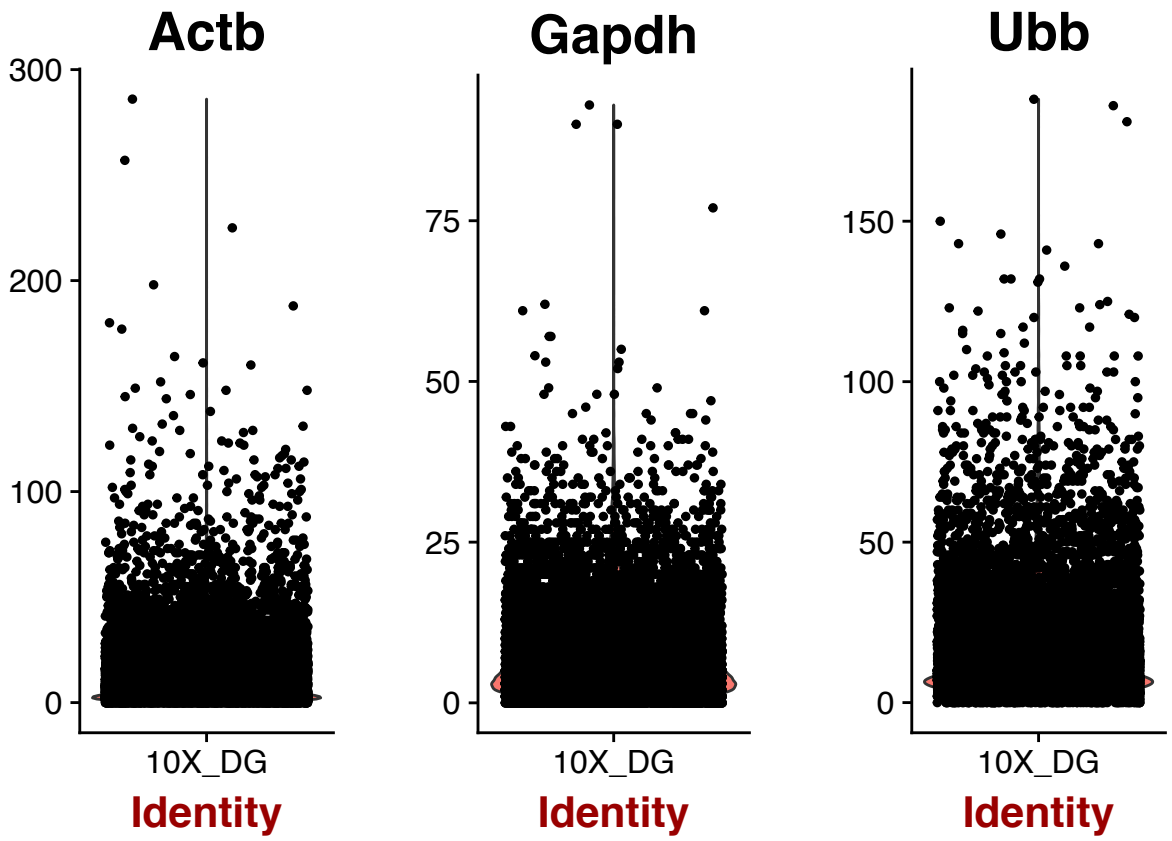
The idea here is to filter out the non viable cells. The Seurat documentation suggests using the percentage of mitochondrial genes as an indicator of viability keeping only the cells with low percentage of mitochondrial genes. The number of genes and UMIs (nGene and nUMI) are automatically calculated for every object by Seurat. We calculate the percentage of mitochondrial genes here and store it in percent.mito using AddMetaData. We use `object@raw.data` since this represents non-transformed and non-log-normalized counts. The % of UMI mapping to MT-genes is a common scRNA-seq QC metric. NOTE: You must have the Matrix package loaded to calculate the percent.mito values.

Lastly we add columns to `object@meta.data` for later usage and graph it to determine cutoffs.

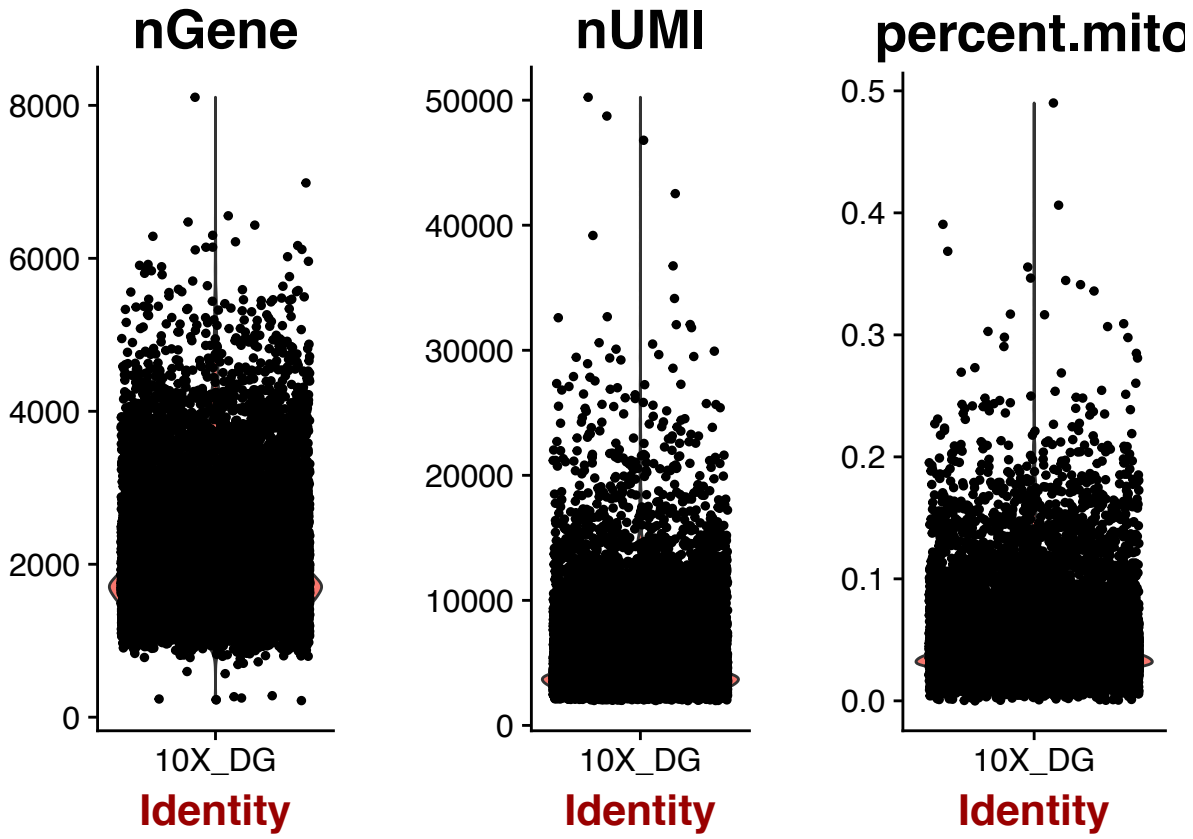
```
mito.genes <- grep(pattern = "^mt-", x = rownames(x = pbmc@data), value = TRUE)
percent.mito <- Matrix::colSums(pbmc@raw.data[mito.genes, ]) /
  Matrix::colSums(pbmc@raw.data)
pbmc <- AddMetaData(object = pbmc, metadata = percent.mito,
                    col.name = "percent.mito")
```

Now we use violin plots to visualize three universally expressed genes ("Actb","Gapdh", "Ubb") to filter our data in addition to the number of genes and nUMI. GenePlot is typically used to visualize gene-gene relationships, but can be used for anything calculated by the object, i.e. columns in `object@meta.data`, PC scores etc.

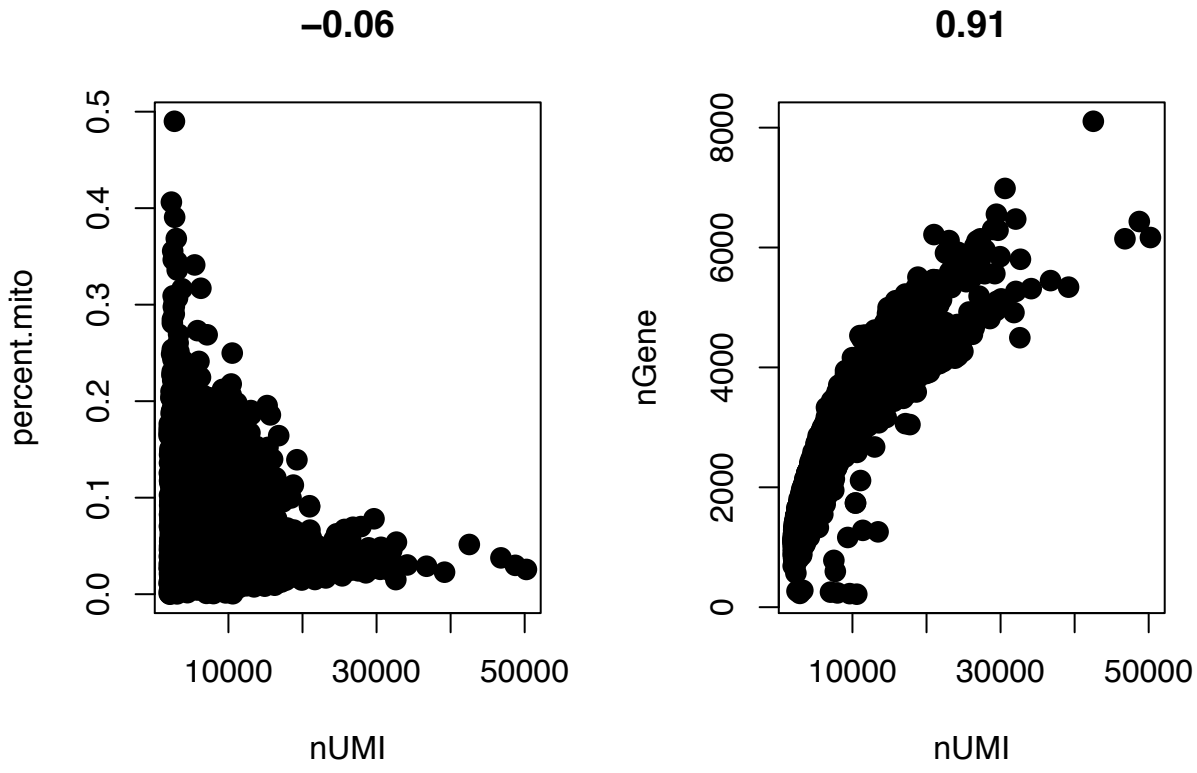
```
VlnPlot(object = pbmc, features.plot = c("Actb","Gapdh", "Ubb"), nCol = 3)
```




```
VlnPlot(object = pbmc, features.plot = c("nGene", "nUMI", "percent.mito"), nCol = 3)
```




```
par(mfrow = c(1, 2))
GenePlot(object = pbmc, gene1 = "nUMI", gene2 = "percent.mito")
GenePlot(object = pbmc, gene1 = "nUMI", gene2 = "nGene")
```



Based on the previous graphs we decide how do we want to define our "valid cells". Seurat's documentation recommends to filter cells with an outlier level of high mitochondrial percentage, low UMI Content and high number of genes.

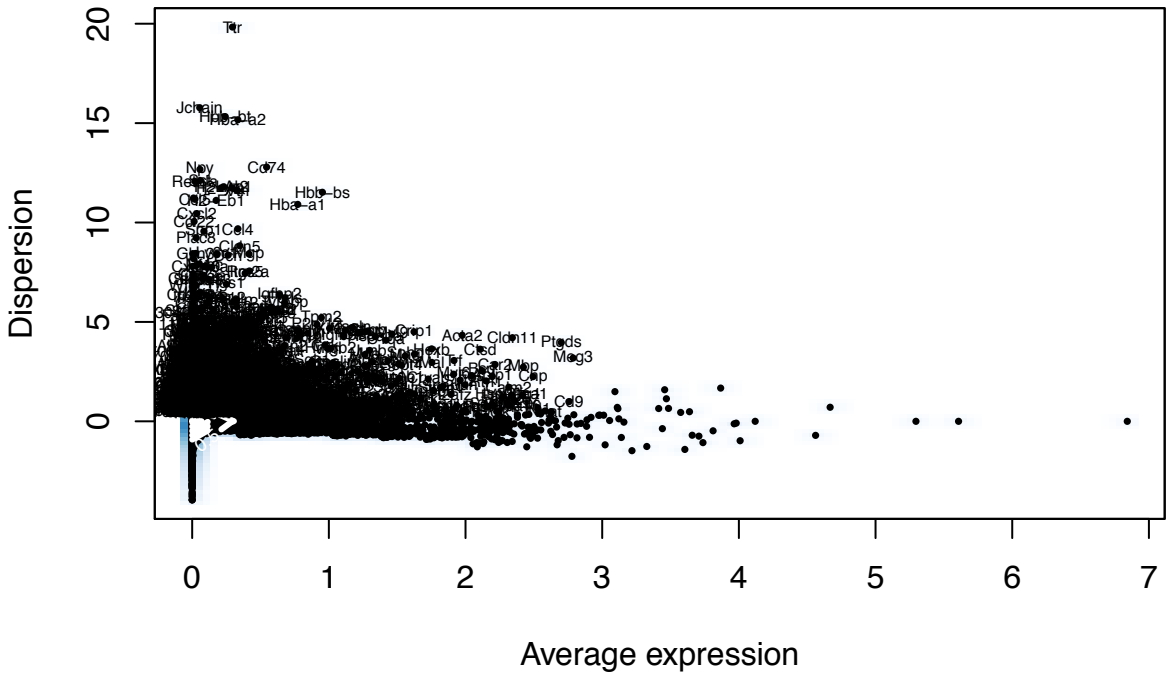
Here we decided (based on Hochgerner et al. 2018) to be more conservative and define our valid cells as ones that have more than 600 genes sequenced and more than 800 UMIs. In addition, we filtered the cells with low expression of universally expressed genes. Please note that filtering based on gene expression must be done after normalizing the data.

```
# Filter cells with low number of genes and UMIs
pbmc <- FilterCells(object = pbmc, subset.names = c("nGene","nUMI"),
                    low.thresholds = c(600, 800), high.thresholds = c(Inf, 20000))

# Before we filter based on gene expression, we must normalize the data
pbmc <- NormalizeData(object = pbmc, normalization.method = "LogNormalize",
                      scale.factor = 10000)

# Filter out cells that have low expression of housekeeping genes
pbmc <- FilterCells(object = pbmc, subset.names = c("Actb","Gapdh", "Ubb"),
                    low.thresholds = c(0.5, 0.5, 0.5),
                    high.thresholds = c(Inf, Inf, Inf))
```

Detect variable genes and Establish Number of Principal Components

We now detect variable genes and determine the number of principal components to be used in our analysis to prepare for clustering.

```
# Find variable genes
pbmc <- FindVariableGenes(pbmc, mean.function=ExpMean, dispersion.function=LogVMR,
                          x.low.cutoff = 0.0125, x.high.cutoff = 3, y.cutoff = 0.5)
```




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

```
## [1] 1894
```

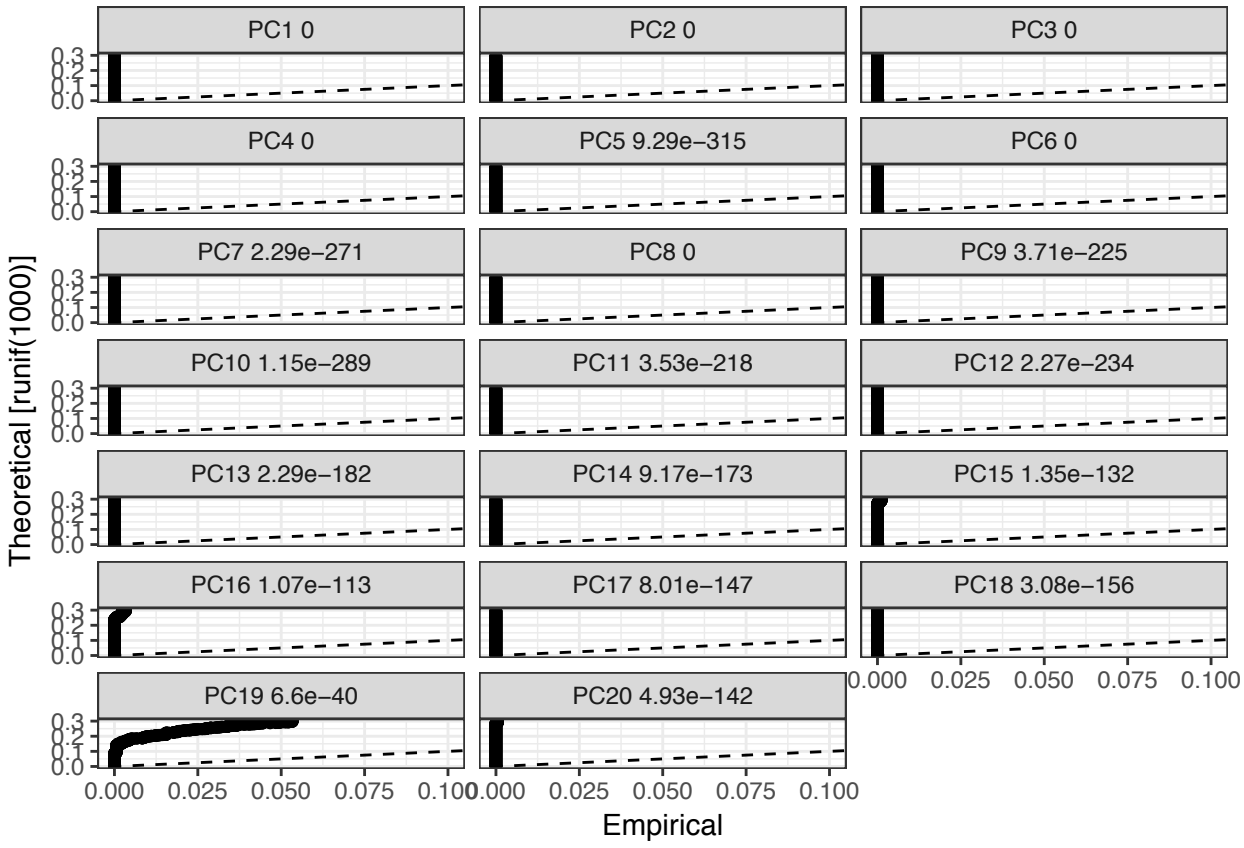
Next we scale the data to regress out cell-cell variation driven by any unwanted source of variability such as the number of detected molecules or mitochondrial gene expression. Seurat's documentation suggests regressing on the number of detected molecules per cell as well as the percentage mitochondrial gene for post-mitotic cells. In our case we regressed exclusively on nUMI.

```
# Scale the data by regressing nUMI
pbmc <- ScaleData(pbmc, vars.to.regress = c("nUMI"))
```

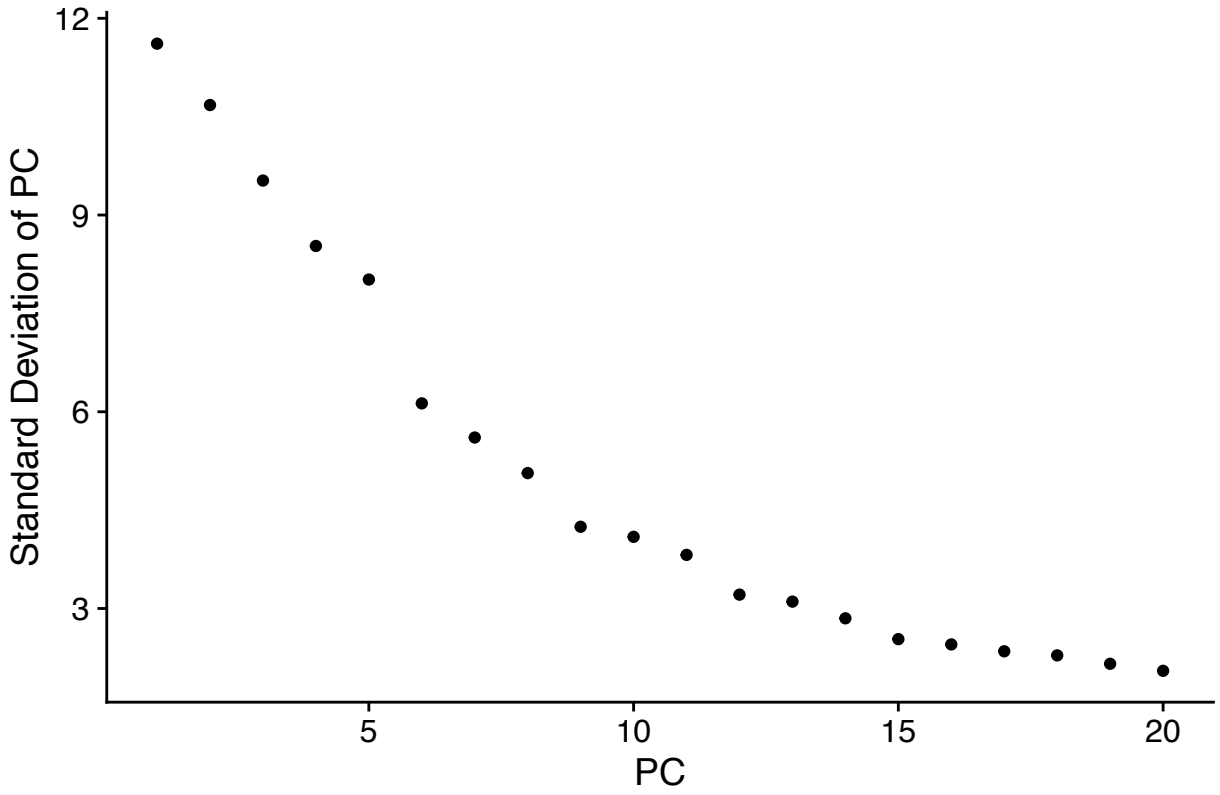
We want to define the number of Principal Component (PCs) to be used in our clustering analysis. For this we calculate the Principal Components (PCs) and visualize them in different ways. The "elbow plot" is particularly useful. The number of PCs that corresponds to the "elbow" of the "y" variable should be the chosen number of PCs. The logic behind it is that adding more PCs to our analysis does not reduce the SD of the PC and therefore does not add significant information to our analysis. Whatever number you choose you should carry over to further steps of the analysis.

```
# Run and Predict PCA
pbmc <- RunPCA(pbmc, pc.genes = pbmc@var.genes,
               do.print = TRUE, pcs.print = 1:5, genes.print = 5)
pbmc <- JackStraw(pbmc, num.replicate = 100)
```

```
# Find the number of PCs to be used in our analysis
JackStrawPlot(pbmc, PCs = 1:20)
```




```
PCElbowPlot(pbmc)
```



Clustering and plotting tSNE graphs

Seurat includes a graph-based clustering in which they first construct a K-nearest neighbor (KNN) graph and then refine the edge weights between cells.

The FindClusters function implements the procedure, and contains a "resolution" parameter that sets the "granularity" of the downstream clustering, with increased values leading to a greater number of clusters. They recommend setting this parameter between 0.6-1.2 typically for single cell data sets of around 3K cells. Optimal resolution often increases for larger data sets. Note: The clusters are saved in the `object@ident` variable.

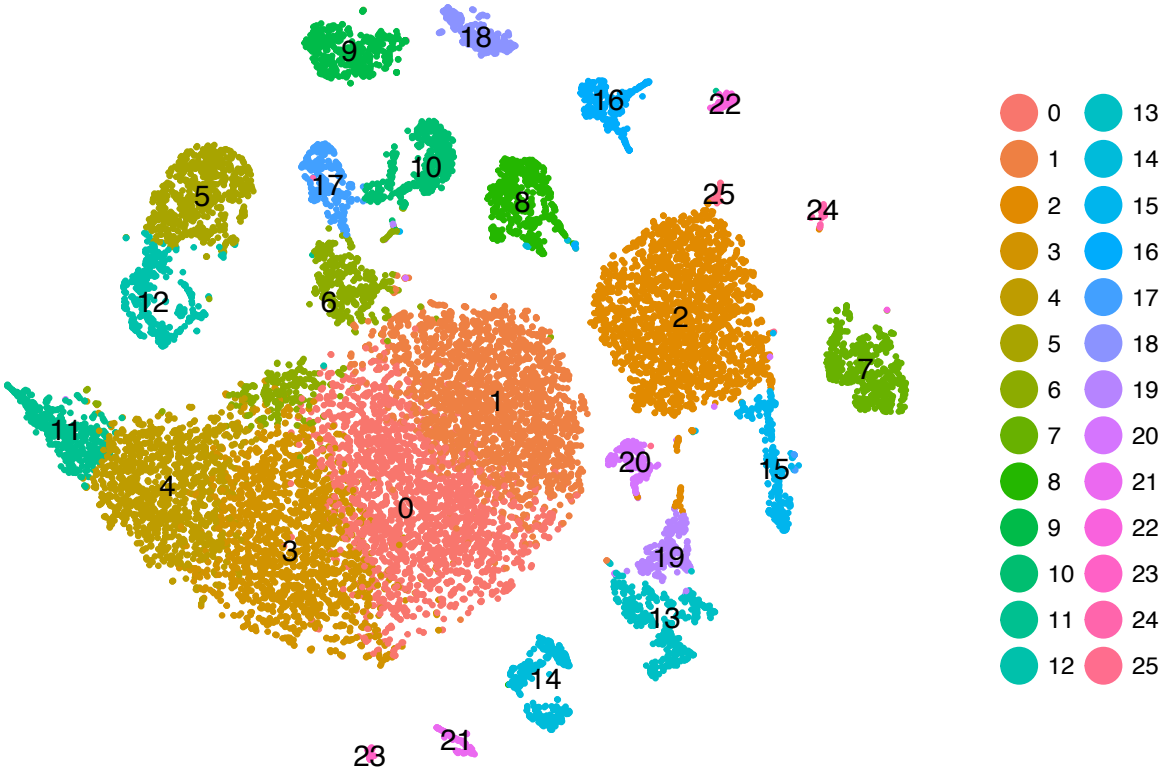
Make sure that the number of PCs you chose (based on the elbow plot above) matches the dims.use parameter. You can run this line of code several times changing the resolution parameter. Multiple runs won't overwrite each other but the calculations will be stored as a different "identity" in the pbmc object. The resulting identity will be called "res.n" where n is the chosen resolution.

```
# Find Clusters
pbmc <- FindClusters(pbmc, reduction.type = "pca", dims.use = 1:15,
                     resolution = 1.4, print.output = 0, save.SNN = TRUE)
PrintFindClustersParams(pbmc)
```

```
## Parameters used in latest FindClusters calculation run on: 2018-06-06 02:58:17
## ============================================================================
## Resolution: 1.4
## ----------------------------------------------------------------------------
## Modularity Function    Algorithm         n.start         n.iter
##      1                   1                 100             10
## ----------------------------------------------------------------------------
## Reduction used          k.param          prune.SNN
```

```
##      pca                  30                 0.0667
## ------------------------------------------------------------------------
## Dims used in calculation
## ========================================================================
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
# Run and Plot tSNE
pbmc <- RunTSNE(pbmc, dims.use = 1:15, do.fast = TRUE)
TSNEPlot(pbmc, do.label=TRUE, no.axes=TRUE, pt.size = 0.5 )
```



Adding a factor column with different levels (GFP High, Low and Negative) and plotting tSNE based on that factor

Executing this section assumes you have sequenced different samples with different libraries and aggregated them with cellranger aggr resulting in a depth normalization – see Aggregating Multiple Libraries with cellranger aggr for more details.

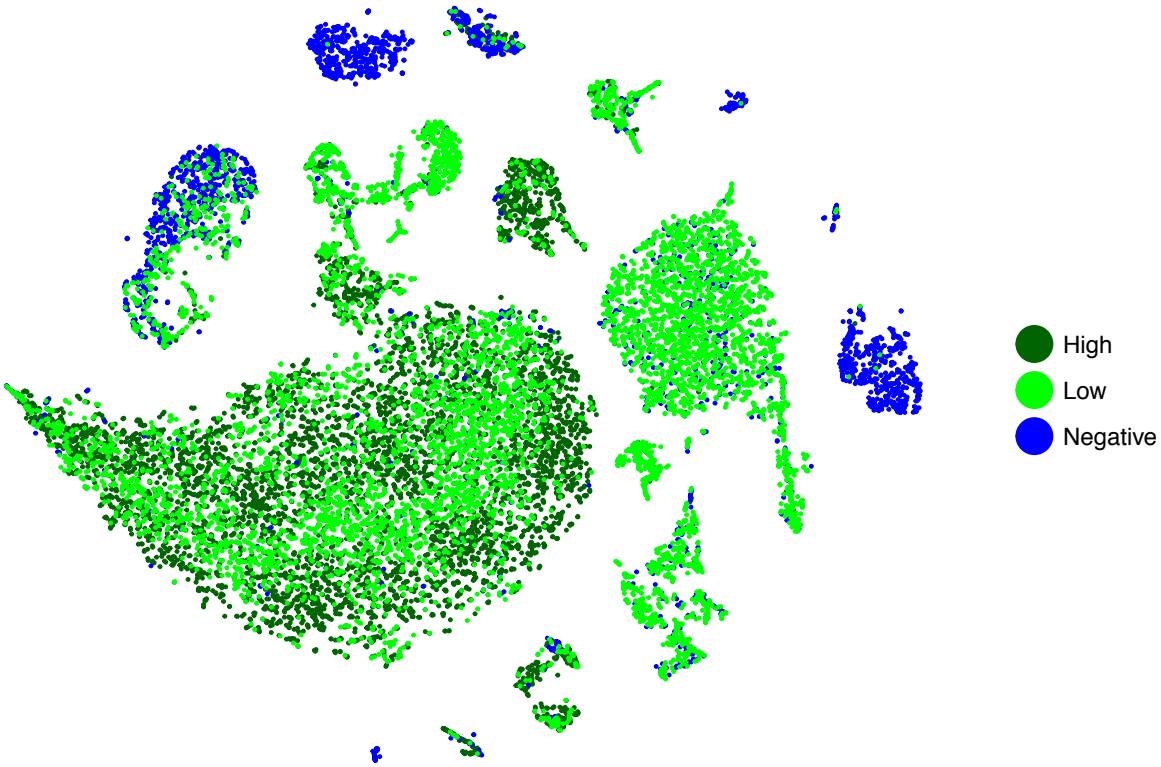
In your normalized dataset you will find a small integer identifying the library next to each nucleotide sequence. This nucleotide sequence plus library ID can be used as a unique identifier in the gene-barcode matrix. For example, AGACCATTGAGACTTA-1 and AGACCATTGAGACTTA-2 are distinct cell barcodes from different libraries (different samples), despite having the same nucleotide sequence. We will parse this number and store it in a another column for later use in our analysis.

```
# Add sample data to our Seurat object
sample.numbers <- as.numeric(str_extract(colnames(pbmc@data), "[^-]+$"))
names(sample.numbers) <- colnames(pbmc@data)
pbmc <- AddMetaData(pbmc, sample.numbers, "sample.number.column")

# Map the sample numbers into a human readable form with the chosen sample names. Make sure you asign t
current.sample.ids <- c(1, 2, 3)
```

```
new.sample.ids <- c("High", "Negative", "Low")
sample.number.names <- plyr::mapvalues(pbmc@meta.data$sample.number.column, from = current.sample.ids,
names(sample.number.names) <- colnames(pbmc@data)
pbmc <- AddMetaData(pbmc, sample.number.names, "sample.number.names")
```

Now we set the identity to sample number to show our clusters with our samples highlighted.

```
# To set new identity to samples
pbmc <- SetAllIdent(object = pbmc, id = "sample.number.names")

# Plot tSNE based on the identity sample.number.names. Choose sample colors.
pbmc <- SetAllIdent(object = pbmc, id = "sample.number.names")
TSNEPlot(object = pbmc, colors.use=c("dark green", "green", "blue"),
         no.axes=TRUE,  pt.size = 0.2, do.label=FALSE )
```




```
# Display a table with the number of cells in each sample
table(pbmc@meta.data$sample.number.column)
```

```
##
##    1    2    3
## 6164 2751 7305
```

Visualization of Gene Expression for Cluster Identification

This block of code is used on two occasions. First, we use it to identify the clusters based on cell type specific marker genes from the literature. Once we have identified the clusters and renamed them, we run this code again to visualize the plots with the final cluster names.

You can run this block of code with any list of genes you want to test. These are just some examples of known cell-type specific markers. Make sure you "uncomment" the line of code you want to run and add a

double parenthesis at the end. You can also run several lists of names at once.

The output for each list of genes are 6 graphs: one feature plot (tSNE with grey dot and purple for the cells that express the gene of interest), a heatmap (feature plot for the different samples), two violin plots and two dot plots. One set of graphs are base on the clusters and the other set of graphs are based on the different samples (GFP High, Low and Negative).

The plots don't show up in R Studio, but they are stored in the "Plots" folder indicated at the beginning of the analysis.

```
genes_of_interest <- list(
  'NSC lineage'               = c("Gfap", "Sox9", "Id4", "Sox2", "Lpar1", "Nes",
                                  "Cdk1","Ccnd2","Neurod1", "Eomes", "Dcx", "Sox11"))
  #'OL lineage'                = c("Sox10", "Olig2", "Sox9", "Pdgfra", "Cspg4",
  #                                "Bmp4", "Mal", "Mog" )
  #'Transgene'                 = c("ECEV4806transgene"),
  #'Stem Cells'                = c("Aldoc", "Apoe", "Id4", "Hopx", "Sox9", "Gfap",
  #                                "Slc1a3", "Sox2", "Fabp7", "Clu"),
  #'NSC lineage'               = c("Sox9", "Id4", "Clu", "Hmgn2", "Ccnd2", "Neurod1",
  #                                "Syt5", "ECEV4806transgene" ),
  #'Early Neural progenitors' = c("Eomes","Ccnd2","Hmgn2"),
  #'Late Neural progenitors'  = c('Neurod1', 'Dcx', "Sox11"),
  #'Neural progenitors'       = c("Eomes","Ccnd2", 'Neurod1', 'Dcx', "Sox11", "Hmgn2"),
  #'Young neurons'            = c("Dcx","Calb2",'Cd24a'),
  #'Mature granular neurons'  = c("Calb1", "Gria1"),
  #'Mature neurons'           = c("Rbfox3","Syt1","Syt5","Myt1l"),
  #'OPC'                      = c("Olig1", "Olig2", "Sox10", "Pdgfra", "Cspg4"),
  #'Commited OL'              = c("Bmp4", "Fyn", "Gpr17"),
  #'Mature Oligodendrocytes'  = c("Plp1", "Mal", "Mog", "Mbp"),
  #'Microglia'                = c("Csf1r", "Cx3cr1"),
  #'Macrophages'              = c('Ptprc'),
  #'Endothelial cells'        = c('Vwf'),
  #'Astrocytes'               = c("S100b", "Fzd2", "Aldh1l1"),
  #'Proliferation'            = c("Ccnd2", "Mki67", "Pcna", "Mcm2"))
  #'Interneurons'             = c("Reln", "Ndnf"),
  #'Pericytes'                = c("Tbx18", "Vtn", "Tagln", "Des"),
  #'Universally expressed'    = c("Actb", "Gapdh", "Ubb"),
  #'CD4 T Cells'              = c("Il7r"),
  #'RBC'                      = c("Hba-a1", "Hba-a2"),
  #'Astrocytes_h'             = c("Gfap", "Hes5", "Sox9", "Aqp4"),
  #'RGL_h'                    = c("Lpar1", "Nes", "Prom1", "Hes5", "Tfap2c", "Rhcg", "Wnt8b"),
  #'nIPC_h'                   = c("Cdk1","Aurkb", 'Top2a', "Neurog2", "Eomes", "Neurod4", "Tfap2c"),
  #'NB1_h'                    = c( "Eomes", "Tac2", "Calb2", "Igfbpl1", "Neurod4"),
  #'NB2_h'                    = c("Gal","Sox11","Dcx", "Igfbpl1"),
  #'Granule inmature_h'       = c("Fxyd7"),
  #'Granule mature_h'         = c("Plk5", "Ntng1"),
  #'Cell Cycle genes_h'       = c("Cdk1","Aurkb", 'Top2a', 'Mki67'))

for ( focus_genes in names(genes_of_interest) ) {
  print(paste('Generating graphs for', focus_genes ))
  genes <- genes_of_interest[[focus_genes]]
  number_of_colums <- max(round(length(genes)/3),1)
  vln_plot_width <- 1000*number_of_colums
  print(paste('Auto calculating columns and plot width to be',
              number_of_colums, vln_plot_width ))
```

```
  # Generate the cluster plot
  png( file.path(analysis_folder,"Plots", paste(focus_genes, "Cluster.png")),
       res=100, height=1000, width=1600)
  FeaturePlot(object = pbmc, features.plot = genes, cols.use = c("grey", "blue"),
              reduction.use = "tsne")
  dev.off()

  # Generate the feature plot heatmap
  png( file.path(analysis_folder,"Plots", paste(focus_genes, "Heatmap.png")),
       res=100, height=1000, width=2200)
  FeatureHeatmap(pbmc, features.plot = genes, group.by = "sample.number.names",
                 pt.size = 0.25,  key.position = "top", max.exp = 3)
  dev.off()

  # Here make sure the identity (here "res.1.4") corresponds to your clustering
  # identity (res. + resolution used for clustering)
  for ( identity in c( "sample.number.names", "res.1.4")) {
    pbmc <- SetAllIdent(object = pbmc, id = identity)

    #generate the dot plot
    png( file.path(analysis_folder,"Plots", paste(focus_genes, identity, "Dot.tiff")),
         res=100, height=800, width=800)
    DotPlot(pbmc,genes.plot = genes, x.lab.rot = T, dot.scale = 8, plot.legend=T, col.min=0)
    dev.off()

    # Generate the violin plot
    png(file.path(analysis_folder,"Plots", paste(focus_genes, identity, "Violn.png")),
        res=100, height=1000, width=vln_plot_width)
    print(VlnPlot(pbmc, do.sort=T, x.lab.rot = T, point.size.use = .5,
                  features.plot=genes, nCol=number_of_colums))
    dev.off()
  }
}
```

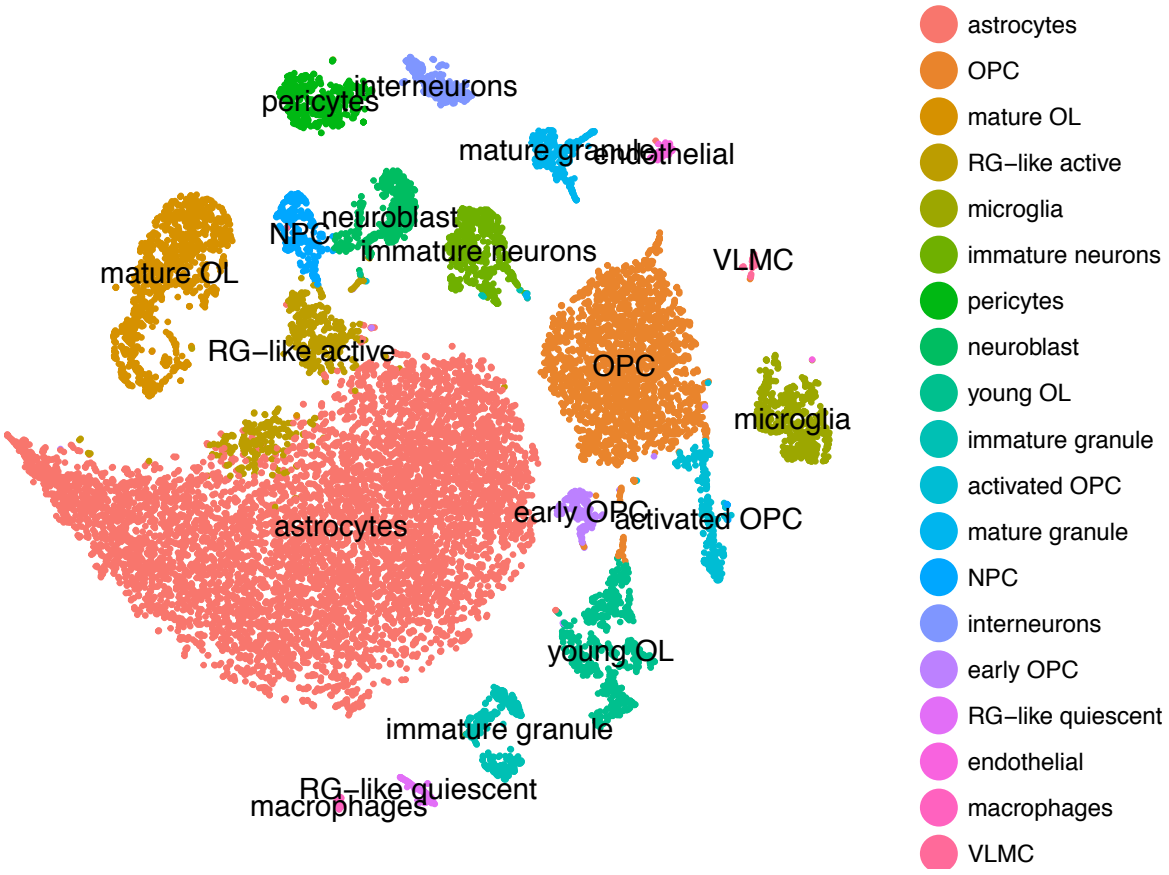
```
## [1] "Generating graphs for NSC lineage"
## [1] "Auto calculating columns and plot width to be 4 4000"
```

Rename the clusters and plot the tSNE plots with the new names

Once we identified the identity of each cluster based on their gene expression we rename the clusters and create a new identity. You should substitute your cluster names in the vector "new.names". We call the new identity "Clusters". With the new identity added to our Seurat object we plot a new tSNE with the new names and save the new Seurat object. You can directly load this seurat object next time you run your code and skip to the analysis below. By saving the object we avoid having to re-run 15-30 minutes of analysis every time.

```
# Name the clusters For Negative, Low, High
pbmc <- SetAllIdent(pbmc, "res.1.4")
current.clusters <- 0:25
new.clusters <- c("astrocytes", "astrocytes", "OPC", "astrocytes", "astrocytes",
                  "mature OL", "RG-like active", "microglia", "immature neurons", "pericytes",
                  "neuroblast", "astrocytes", "mature OL", "young OL", "immature granule",
                  "activated OPC","mature granule", "NPC", "interneurons", "young OL", "early OPC",
```

```
                  "RG-like quiescent", "endothelial", "macrophages", "VLMC", "OPC")

for (i in current.clusters) {
  pbmc <- RenameIdent(object = pbmc, old.ident.name = i,
                      new.ident.name = new.clusters[i + 1])
}

# Add a new identity to our data (called "Clusters")
cluster.names <- plyr::mapvalues(pbmc@meta.data$res.1.4,
                                 from = current.clusters, to = new.clusters)
names(cluster.names) <- colnames(pbmc@data)
pbmc <- AddMetaData(pbmc, cluster.names, "Clusters")

# Plot tSNE with the new names (for dark background in the tSNE plot dark.theme= TRUE)
TSNEPlot(pbmc, do.label = T, pt.size = 0.5, no.axes=TRUE)
```




```
# Save our new Seurat object
# save (pbmc, file = file.path(project_folder, 'Negative_Low_High_filtering_h_0417.Robj') )
```

Count Cells by Sample and Cluster

Now we count the groups and save the results in a csv file.

```
pbmc <- SetAllIdent(pbmc, "Clusters")
Transcriptpercluster<- with(pbmc@meta.data, table(sample.number.column, Clusters))
```

```
# Save the results in a csv file
write.csv(Transcriptpercluster,
          file = file.path(project_folder,
                           'Negative_Low_High_filtering_h_TranscriptsClusters0418.csv') )
```

Finding differentially expressed genes (Cluster Biomarker)

Seurat can help you find markers that define clusters via differential expression. By default, it identifies positive and negative markers of a single cluster, compared to all other cells. FindAllMarkers automates this process for all clusters, but you can also test groups of clusters vs. each other, or against all cells.

The min.pct argument requires a gene to be detected at a minimum percentage in either of the two groups of cells, and the thresh.test argument requires a gene to be differentially expressed (on average) by some amount between the two groups.

Make sure you set the data to the correct identity. The newly generated Seurat object "pbmc.markers" that includes the markers names and expression levels will be saved for future use as this processing can take up to an hour. If you run the two load lines at the start of this file after you save pbmc and pbmc.markers, you can skip to the next code chunk.

```
# Find markers for every cluster compared to all remaining cells, report only the positive ones
pbmc.markers <- FindAllMarkers(object = pbmc, only.pos = TRUE, min.pct = 0.25,
                               thresh.use = 0.25)

# Save the new "markers" seurat object
save (pbmc.markers, file = file.path(project_folder,
                      'Negative_Low_High_filtering_h_AllMarkers_res1_4_0418.Robj'))
```

Finding the top marker genes

Here we are going to define the top (positive) marker genes per cluster.

```
# Get the top 15 genes per cluster
pbmc <- SetAllIdent(pbmc, "Clusters")
top15 <- pbmc.markers %>% group_by(cluster) %>% top_n(15, avg_logFC)

# Save the top15 genes by cluster to a CSV file
write.csv( top15,
           file = file.path(project_folder,
                            'Negative_Low_High_filtering_h_top15markers_Clusters_0418.csv') )

# Save all the markers to a csv file
write.csv( pbmc.markers, file = file.path(project_folder,
                                          'allmarkers.csv') )
```

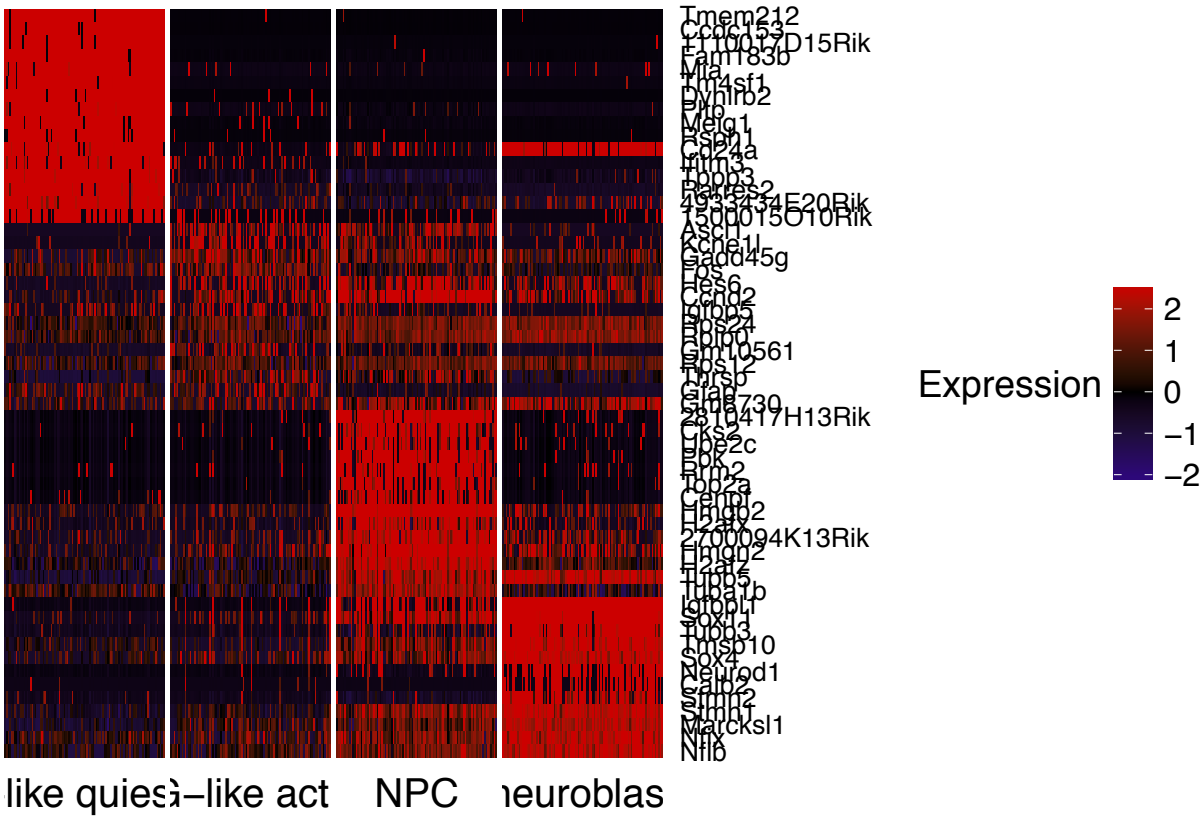
Visualizing cell-type specific signature with a heatmap

Heatmap of Specific Clusters

Seurat's "DoHeatmap" generates an expression heatmap for given cells and genes. Their documentation shows how to generate a heatmap for all the cells in all the clusters. When your database is big and the clusters are very different in size it can result in a not very informative graph. To improve the visualization

of this graph we enabled the selection of specific clusters and the random sampling of n number of cells per clusters (max.cells.per.identity) resulting in a much more informative and proportionate graph.

```
# Make sure you substitute the name of the clusters you want to plot in focus.clusters
focus.clusters <- c("RG-like quiescent", "RG-like active", "NPC", "neuroblast")

# top_n should be previously calculated with the same number of top marker genes per cluster
top_genes <- pbmc.markers %>% group_by(cluster) %>%
  filter(cluster %in% focus.clusters) %>% mutate(order = match(cluster, focus.clusters)) %>%
  top_n(15,  avg_logFC) %>% arrange(order)

# Number of cells per cluster, in our case we chose 100
focus.cells <- WhichCells( pbmc, ident = focus.clusters, max.cells.per.ident = 100)

# Plot Heatmap
DoHeatmap(object = pbmc, cells.use = focus.cells, genes.use = top_genes$gene,
          slim.col.label = TRUE, group.order = focus.clusters,
          col.low="#330099", col.mid = "#000000", col.high = "#CC0000")
```



Generate 2D plots for gene expression between clsuters

This graph is useful to make pairwise comparisons between clusters. The diagonal line represents the genes that are equally expressed in both clusters. Comparing two clusters with similar gene expression signature will result in all the genes located in and around the diagonal line. The genes that are away from the diagonal are differentially expressed.

```
#RG-like quiescent versus RG-like active
cluster_1 <- SubsetData(pbmc, ident.use = "RG-like quiescent", subset.raw = T)
```

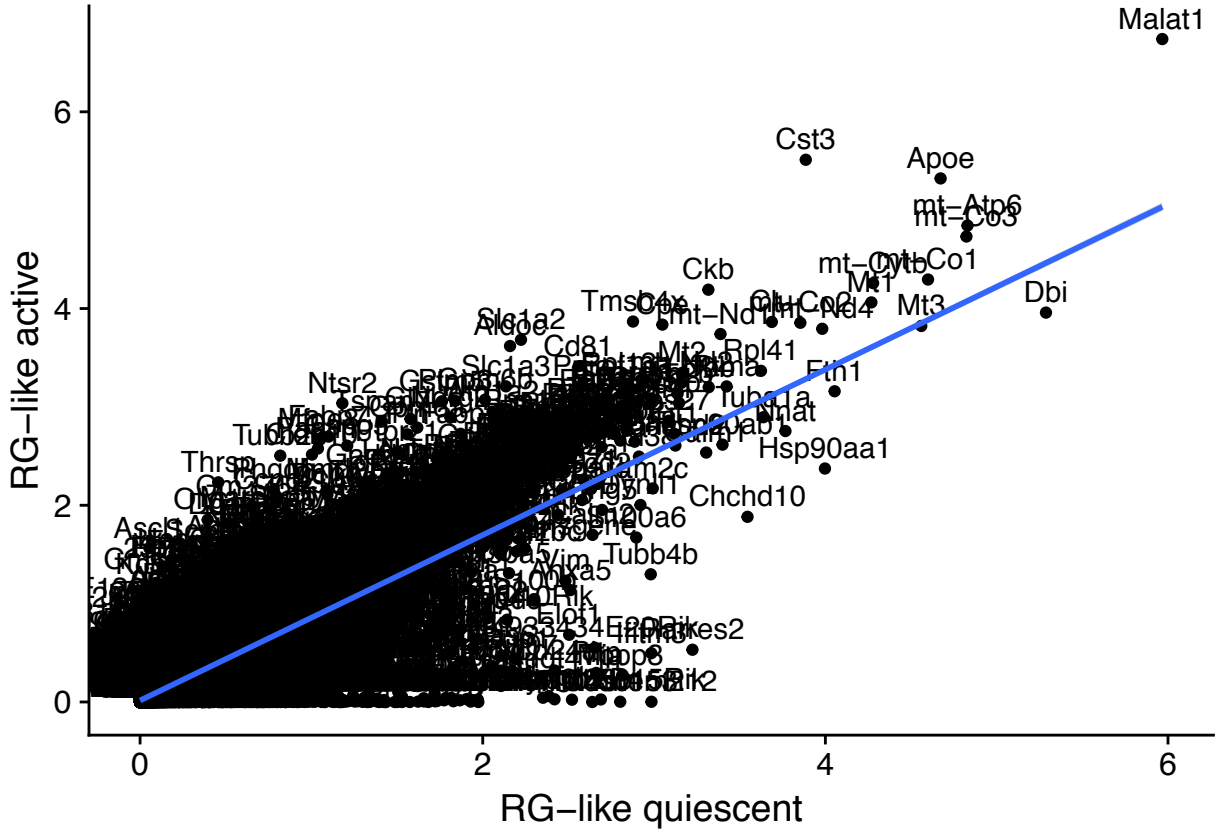
```
avg.cluster_1 <- log1p(AverageExpression(cluster_1))
```

```
## [1] "Finished averaging RNA for cluster RG-like quiescent"
```

```
avg.cluster_1$gene <- rownames(avg.cluster_1)

cluster_2 <- SubsetData(pbmc, ident.use = "RG-like active", subset.raw = T)
avg.cluster_2 <- log1p(AverageExpression(cluster_2))
```

```
## [1] "Finished averaging RNA for cluster RG-like active"
```

```
avg.cluster_2$gene <- rownames(avg.cluster_2)

merged_clusters = merge(avg.cluster_1,avg.cluster_2,by="gene")

merged_clusters['x'] = merged_clusters["RG-like quiescent"]
merged_clusters['y'] = merged_clusters["RG-like active"]
# Dot plot with gene labels. If you want to remove the gene labels delete "geom_text"
ggplot(merged_clusters, aes(x,y)) + geom_point() + geom_text(aes(label=gene), vjust = -0.5) +
  xlab("RG-like quiescent") + ylab("RG-like active") + stat_smooth(method='lm', formula=y ~ x)
```



Visualization of Gene Expression in the identified clusters

The code chunk of the section "Visualization of Gene Expression for Cluster Identification" should be run again with the new identity of the clusters. Make sure you define the identity of the identified clusters ("Clusters" for us) as explained above. Then you can select the specific clusters you want to include in your plots or make the plots for all the clusters as we did previously for "res.1.4" identity.

```
# Set the right identity
pbmc <- SetAllIdent(pbmc, "Clusters")
# Use this to grab a subset of clusters
pbmc <- SubsetData(pbmc,
                   ident.use = c("astrocytes", "RG-like quiescent", "RG-like active", "NPC", "neuroblas
                   subset.raw = T)
```

```
for ( focus_genes in names(genes_of_interest) ) {
  print(paste('Generating graphs for', focus_genes ))
  genes <- genes_of_interest[[focus_genes]]
  number_of_colums <- max(round(length(genes)/3),1)
  vln_plot_width <- 1000*number_of_colums
  print(paste('Auto calculating columns and plot width to be',
              number_of_colums, vln_plot_width ))
  # Generate the feature plots plot
  png( file.path(analysis_folder,"Plots", paste(focus_genes, "Cluster.png")),
       res=100, height=1000, width=1600)
  FeaturePlot(object = pbmc, features.plot = genes, cols.use = c("grey", "blue"),
              reduction.use = "tsne")
  dev.off()
  # Generate the heatmap
  png( file.path(analysis_folder,"Plots", paste(focus_genes, "Heatmap.png")),
       res=100, height=1000, width=2200)
  FeatureHeatmap(pbmc, features.plot = genes, group.by = "sample.number.names",
                 pt.size = 0.25,  key.position = "top", max.exp = 3)
  dev.off()

  # Here make sure the identity (here "Clusters") corresponds to yours.
  for ( identity in c( "sample.number.names", "Clusters")) {
    pbmc <- SetAllIdent(object = pbmc, id = identity)

    #generate the dot plot
    png( file.path(analysis_folder,"Plots", paste(focus_genes, identity, "Dot.tiff")),
         res=100, height=800, width=800)
    DotPlot(pbmc,genes.plot = genes, x.lab.rot = T, dot.scale = 8, plot.legend=T, col.min=0)
    dev.off()

    # Generate the violin plot
    png( file.path(analysis_folder,"Plots", paste(focus_genes, identity, "Violn.png")),
         res=100, height=1000, width=vln_plot_width)
    print(VlnPlot(object = pbmc, do.sort=T, x.lab.rot = T, point.size.use = .5,
                  features.plot = genes, nCol = number_of_colums))
    dev.off()
  }
}
```

```
## [1] "Generating graphs for NSC lineage"
## [1] "Auto calculating columns and plot width to be 4 4000"
```

Monocle

Monocle introduced the strategy of using RNA-Seq for single cell trajectory analysis. Rather than purifying cells into discrete states experimentally, Monocle uses an algorithm to learn the sequence of gene expression

changes each cell must go through as part of a dynamic biological process. Once it has learned the overall "trajectory" of gene expression changes, Monocle can place each cell at its proper position in the trajectory. You can then use Monocle's differential analysis toolkit to find genes regulated over the course of the trajectory, as described in the section Finding Genes that Change as a Function of Pseudotime . If there are multiple outcome for the process, Monocle will reconstruct a "branched" trajectory. These branches correspond to cellular "decisions", and Monocle provides powerful tools for identifying the genes affected by them and involved in making them. Monocle relies on a machine learning technique called reversed graph embedding to construct single-cell trajectories.

Monocle Setup and library load

Installing Monocle:

```
#Run the first time using Monocle
#source("http://bioconductor.org/biocLite.R")
#biocLite()
#biocLite("monocle")
#library(reshape2)
#library(stringr)
#install.packages("devtools")
#devtools::install_github("cole-trapnell-lab/monocle-release@develop")
#biocLite(c("DDRTree", "pheatmap"))
```

Once the previous software and packages are installed. Every time you want to want to use Monocle you have to load the libraries of Monocle and Seurat.

```
library(monocle)
library(Seurat)
```

If this is the first time that you are running Monocle you have to load your Seurat objects

```
# Load Seurat or Monocle object if it exist
# load( file = file.path(project_folder, 'HSMM_Negative_Low_High_filtering_oligo_0510.Robj') )
load( file = file.path(project_folder, 'HSMM_Negative_Low_High_filtering_h_0418.Robj') )
# load( file = file.path(project_folder, 'Negative_Low_High_filtering_h_0417.Robj') )
```

Once our Seurat pbmc object is loaded we can import it into a Monocle object. We maintained the name of the object used in the Monocle tutorial ("HSMM").

```
#Import Seurat Object into a Monocle Object
HSMM <- importCDS(pbmc, import_all = T)

# We must calcualate these every time we run Monocle unless you load the HSMM from an Robj file.
HSMM <- estimateSizeFactors(HSMM)
HSMM <- estimateDispersions(HSMM)
HSMM <- detectGenes(HSMM, min_expr = 0.1)

# Check the head of our Moncole data for structure.
head(fData(HSMM))
head(pData(HSMM))

# Select clusters to focus on for Pseudotime analysis.
neurogenic.lineage.clusters <- c( "RG-like quiescent", "RG-like active", "NPC", "neuroblast")

valid_cells <- row.names(subset(pData(HSMM), is.element(Clusters, c(neurogenic.lineage.clusters))))
HSMM <- HSMM[,valid_cells]
```

```
expressed_genes <- row.names(subset(fData(HSMM), num_cells_expressed >= 10))

HSMM <- detectGenes(HSMM, min_expr = 0.1)
```

Constructing Sinngle Cell Trajectories with Monocle's Semi-supervised ordering mode

Unsupervised ordering is desirable because it avoids introducing bias into the analysis. If you wish to try unsupervised ordering on your data follow Monocle guidelines. However, unsupervised machine learning will sometimes fix on a strong feature of the data that's not the focus of your experiment. For example, where each cell is in the cell cycle has a major impact on the shape of the trajectory when you use unsupervised learning. But what if you wish to focus on cycle-independent effects in your biological process? Monocle's "semi-supervised" ordering mode can help you focus on the aspects of the process you're interested in. We consider semi-supervised ordering more appropriate to model neural differentiation.

Ordering your cells in a semi-supervised manner is very simple. You first define genes that mark progress using the CellTypeHierchy system. Then, you use it to select ordering genes (1000 genes) that co-vary with these markers. Finally, you order the cell based on these genes. The only difference between unsupervised and semi-supervised ordering is in which genes we use for ordering.

Run "expressed genes" and NSC genes_id

```
expressed_genes <- row.names(subset(fData(HSMM), num_cells_expressed >= 10))
```

Define marker genes

```
Nes_id <- row.names(subset(fData(HSMM), gene_short_name == "Nes"))
Lpar1_id <- row.names(subset(fData(HSMM), gene_short_name == "Lpar1"))
Cdk1_id <- row.names(subset(fData(HSMM), gene_short_name == "Cdk1"))
Sox4_id <- row.names(subset(fData(HSMM), gene_short_name == "Sox4"))

cth <- newCellTypeHierarchy()

cth <- addCellType(cth,
                   "RG-like quiescent",
                   classify_func = function(x) { x[Nes_id,] >= 0.5 })
cth <- addCellType(cth,
                   "RG-like active",
                   classify_func = function(x) { x[Lpar1_id,] >=0.5 })
cth <- addCellType(cth,
                   "NPC",
                   classify_func = function(x) { x[Cdk1_id,] >=1 })
cth <- addCellType(cth,
                   "neuroblast",
                   classify_func = function(x) { x[Sox4_id,] >=1 })
```

Classify

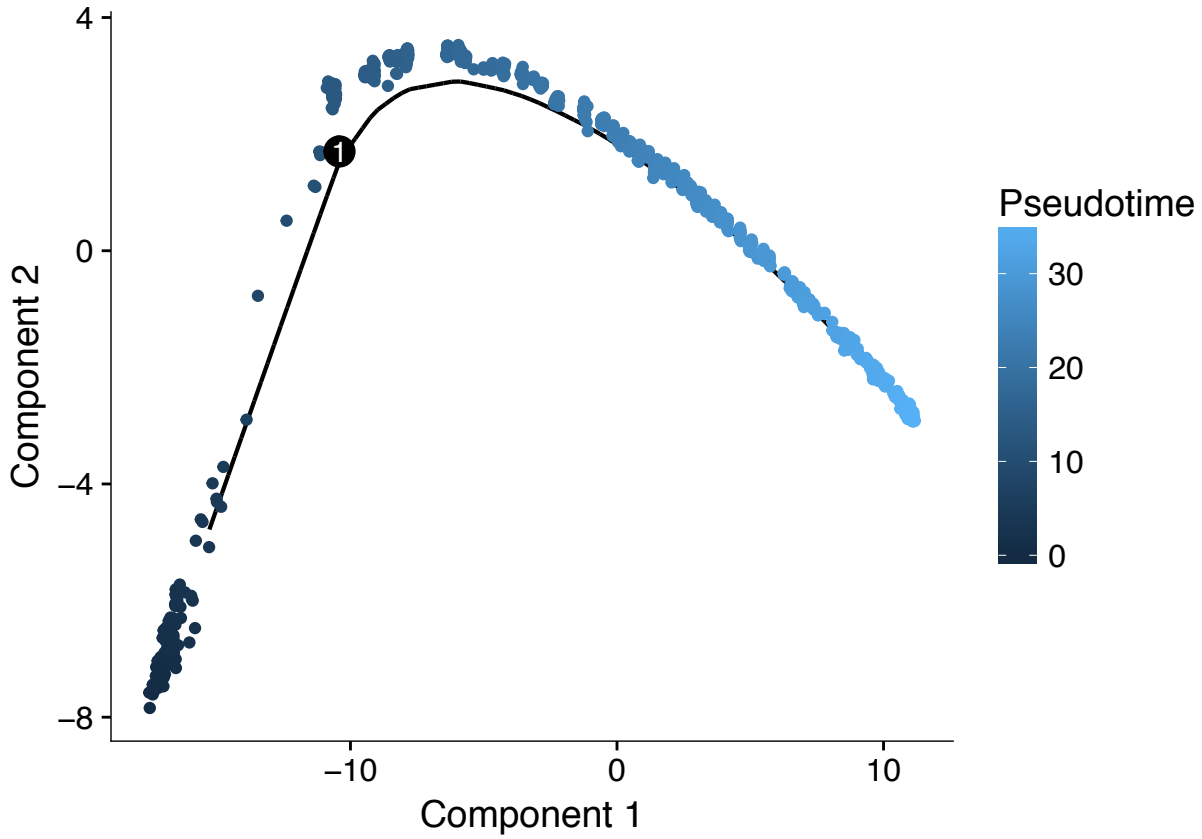
```
HSMM <- classifyCells(HSMM, cth)
#Change cores based on the CPU and system memory availible. ~4GB memory per core for ~5k cells and 100%
marker_diff <- markerDiffTable(HSMM[expressed_genes,], cth, cores = 4)
```

Add in 1,000 more genes to help Moncole order based on the supervised genes above

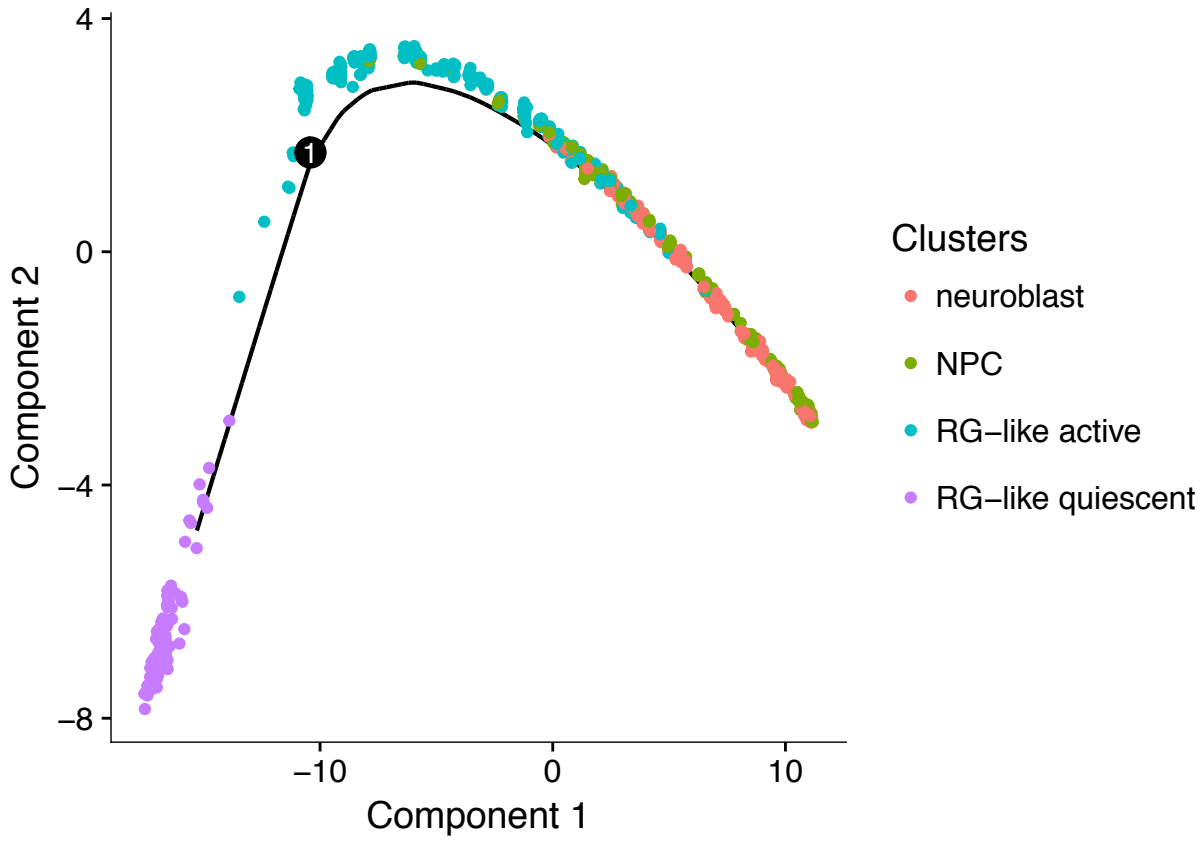
```
semisup_clustering_genes <- row.names(marker_diff)[order(marker_diff$qval)][1:1000]
HSMM <- setOrderingFilter(HSMM, semisup_clustering_genes)
HSMM <- reduceDimension(HSMM, max_components = 2, method = 'DDRTree', norm_method = 'log')
HSMM <- orderCells(HSMM)
```

Construct and plot single cell trajectories

```
# By Pseudotime
plot_cell_trajectory(HSMM, color_by = "Pseudotime") +
  theme(legend.position = "right") +
  ggsave(file.path(analysis_folder,"Plots", "RGL_Pseudotime.png"), height=5, width=9)
```



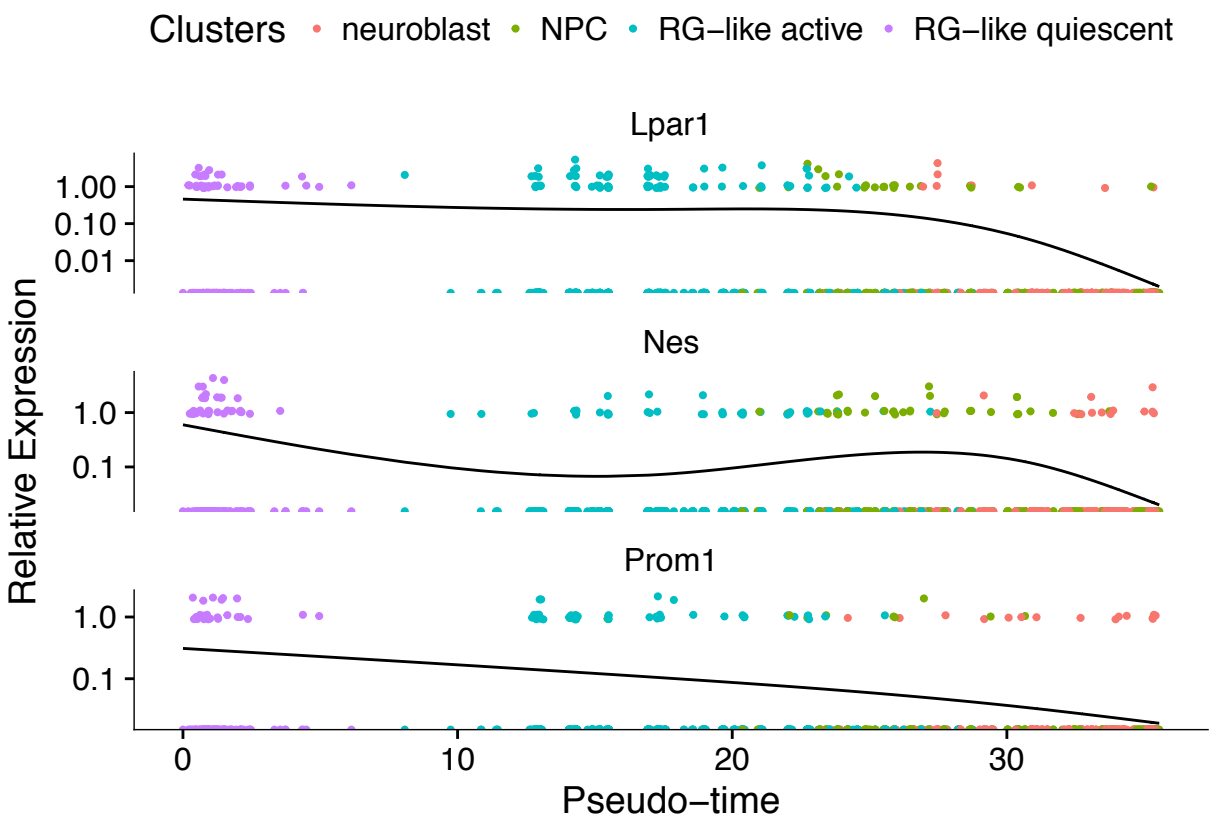

```
# By cluster
plot_cell_trajectory(HSMM, color_by = "Clusters") +
  theme(legend.position = "right") +
  ggsave(file.path(analysis_folder,"Plots", "RGL_Clusters.png"), height=5, width=9)
```



Generate pseudotime plots for specific genes

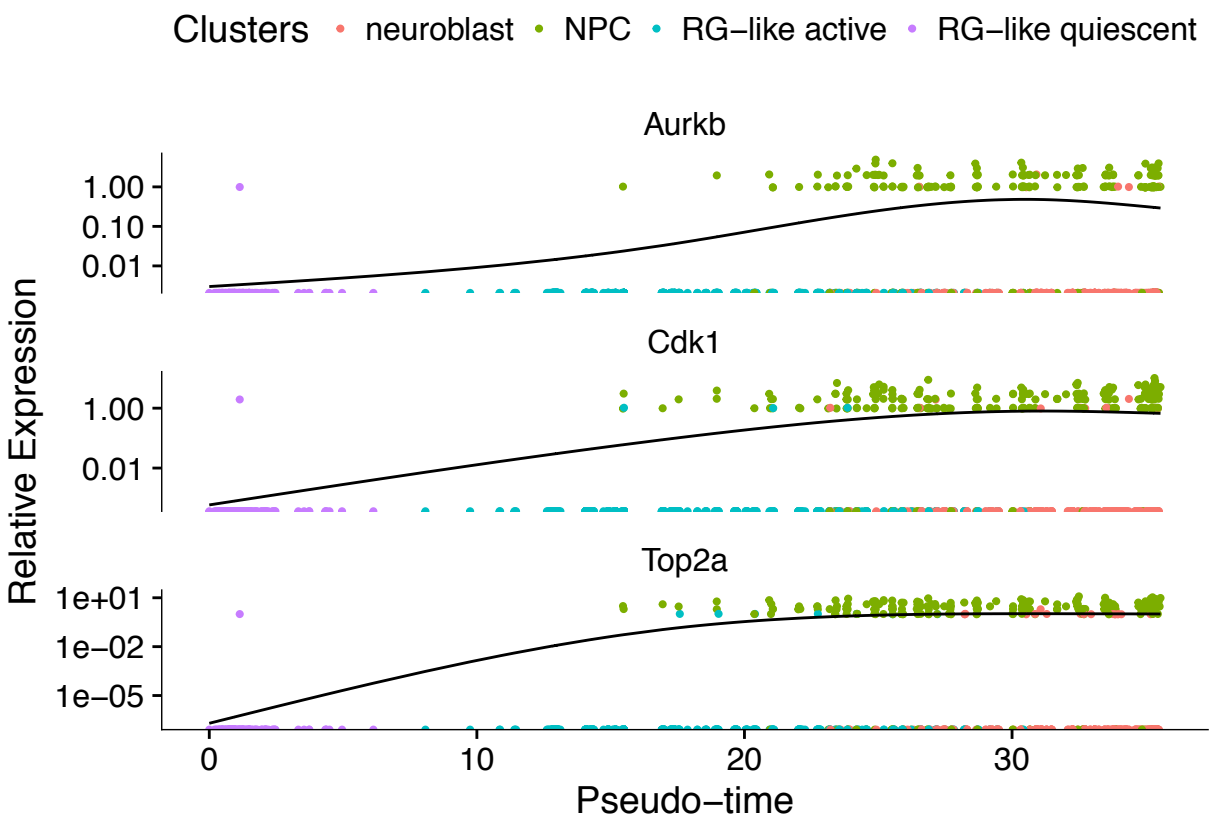
```
generate.pseudo.plot <- function( genes ) {
  #Finding Genes that Change as a Function of Pseudotime
  to_be_tested <- row.names(subset(fData(HSMM), gene_short_name %in% genes ))
  cds_subset <- HSMM[to_be_tested,]
  diff_test_res <- differentialGeneTest(cds_subset, fullModelFormulaStr = "~sm.ns(Pseudotime)")
  diff_test_res[,c("gene_short_name", "pval", "qval")]

  file.name <- file.path(analysis_folder,"Plots", paste(genes[1], "Pseudotime.png"))
  print(paste("Saving plot to",file.name))

  plot_genes_in_pseudotime(cds_subset, color_by = "Clusters") +
    theme(legend.position = "top") + ggsave(file.name, height=9, width=9)
}
# Generate Pseudotime Plots
generate.pseudo.plot(c("Lpar1", "Nes", "Prom1"))
```

```
## [1] "Saving plot to /Users/elsa/Documents/Work/Current/10X/src/../Results/Sample234/Plots/Lpar1 Pseu
```



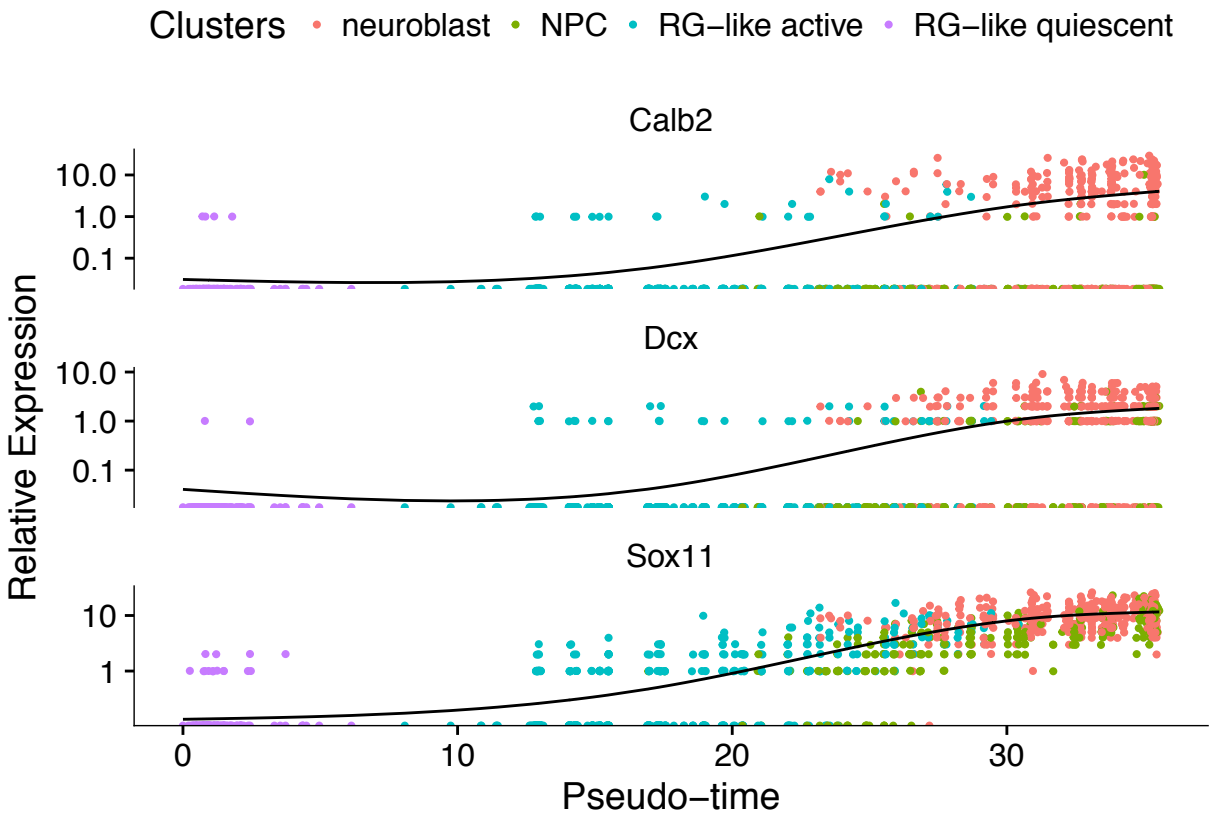

```
generate.pseudo.plot(c("Cdk1","Aurkb", 'Top2a' ))
```

```
## [1] "Saving plot to /Users/elsa/Documents/Work/Current/10X/src/../Results/Sample234/Plots/Cdk1 Pseudo
```




```
generate.pseudo.plot(c("Dcx","Calb2", "Sox11"))
```

```
## [1] "Saving plot to /Users/elsa/Documents/Work/Current/10X/src/../Results/Sample234/Plots/Dcx Pseudo
```



Clustering Genes by Pseudotemporal Expression Pattern

A common question that arises when studying time-series gene expression studies is: "which genes follow similar kinetic trends"? Monocle can help you answer this question by grouping genes that have similar trends, so you can analyze these groups to see what they have in common. Monocle provides a convenient way to visualize all pseudotime-dependent genes. The function plot_pseudotime_heatmap takes a CellDataSet object (usually containing a only subset of significant genes) and generates smooth expression curves much like plot_genes_in_pseudotime. Then, it clusters these genes and plots them using the pheatmap package. This allows you to visualize modules of genes that co-vary across pseudotime.

In our case we used the top marker genes per cluster generated by Seurat for the clusters of interest in order to see which of our marker genes is pseudotime-dependent.

```
# Make sure you substitute the name of the clusters you want to plot in focus.clusters
focus.clusters <- c("RG-like quiescent", "RG-like active", "NPC", "neuroblast")

# top_n should be previously calculated with the same number of top marker genes per cluster (see Seura
top_genes <- pbmc.markers %>% group_by(cluster) %>%
  filter(cluster %in% focus.clusters) %>% mutate(order = match(cluster, focus.clusters)) %>%
  top_n(15,  avg_logFC) %>% arrange(order)

#Run the following line to get the top genes for the specificied clusters
top_15 <- (top_genes %>% filter(cluster %in% focus.clusters ))$gene
marker_genes <- row.names(subset(fData(HSMM), gene_short_name %in% top_15 ))
diff_test_res <- differentialGeneTest(HSMM[marker_genes,],
                                      fullModelFormulaStr = "~sm.ns(Pseudotime)")
sig_gene_names <- row.names(subset(diff_test_res, qval < 0.1))
#Here you can modify num_clusters with the number of expected gene clusters
```

```
plot_pseudotime_heatmap(HSMM[sig_gene_names,],
                        num_clusters = 3,
                        cores = 1,
                        show_rownames = T)
```