Trajectory Analysis of scRNA-seq Data

Identifying Differentially Expressed Genes Along a Lineage

Swarnali Dasgupta

2024-12-22

Contents

0.1 Lineage based differential expression analysis

When we do single-cell RNA sequencing, biological processes like cell differentiation or even disease progression can be studied as a trajectory which maps how cells progress through different stages of cell differentiation of disease progression. This journey of a cell from one state to other is called lineage and these lineages can show particular brances or pathways that cells take during their differentiation process. We can do lineage based differential gene expression analysis to determine genes whose expression levels change:

- In a lineage over a particular time period (pseudotime): Maps changes in gene expression over different time intervals
- Between two lineages: Differences in gene expression between two branches of the trajectory

Such trajectory analysis is valuable because it can help us study:

- How cells differentiate
- How cells deviate from typical differentiation

However, trajectory analysis can be quite challenging because:

- Genes can sometimes show non-linear shifts in expression, which are tough to model
- There is difficulty in obtaining robust statistical techniques that can accurately build pseudotime trajectories

0.2 What method(s)/package(s) can you find to perform the proposed analysis?

There are a few R based bioinformatics packages that can be used for trajectory building and differential expression analysis in scRNA-seq data:

- **Seurat**: Seurat is one of the most widely used tools for finding differentially expressed genes along a lineage in scRNA-seq data, since it allows efficient pre-processing of data and seemless integration with other tools which can construct pseudotime trajectories.
- **Slingshot + tradeSeq**: Slingshot is a widely used trajectory construction tool that identifies cell lineages by fitting curves for cells in the data. Once Slingshot has compelted building the trajectories, we can use tradeSeq to fit models that can accurately capture genes behaving differentially at different points along the trajectory (completing the pseudotime analysis portion). Slingshot can use Seurat to initially pre-process the data and perform initiial processing like normalization, scaling, clustering as well.
- **Monocle3**: Monocle3 is also a popular package for trajectory building and pseudotime analysis. It can pre-process data and perform initial processing on its own.
- There are a few more packages like Scorpius, dynverse.

0.3 Which of these method(s)/package(s) would you choose to perform the analysis and why?

For the task at hand, I have opted for a combination of **Seurat + Slingshot + tradeSeq**.

Slingshot does really well in identifying cell lineages and building cell trajectories by fitting principal curves through the data. It is quite flexible and compatible since it can easily take input from a variety of other packages like Seurat, which can be handy in performing the pre-processing and initial processing of the data. Slingshot also does not impose a predefined trajectory, making it my choice for capturing a wide variety of biological processes.

tradeSeq's use of additive models to model gene expression patterns along pseudotime allows it to capture even non-linear changes, which are quite common in biological processes.

Monocle3 is also a powerful tool for this task, but I did not select it because: - Monocle3 doesn't integrate as smoothly with widely used tools like Seurat, which can make the workflow more complicated and less streamlined. - Slingshot gives more control over how trajectories are inferred, whereas Monocle3 tends to be more automated, which can sometimes limit customization.

Both Slingshot + tradeSeq and Monocle3 have their own strengths and can be used on a case-by-case basis; the bioinformatician needs to use their own discretion to select the best package for their particular task. For this task, the combination of Slingshot and tradeSeq provides a better balance of flexibility, statistical power, and ease of use.

0.4 What considerations, if any, are there for the input data? Necessary pre-processing, sample replicate number, etc.

Before we get started with the sc-RNA seq data, we must do some pre-processing to ensure that results are reliable and accurate.

1. Data Type: We start with the cellranger software output, which generates a raw counts matrix. The rows represent genes and columns represent cells.

2. Quality Control: We need to remove poor quality cells and genes to avoid unnecessary noise, cells exhibiting high mitochondrial gene expression (generally indicates stressed or dying cells), genes expressed

in very few cells and also cells with unusually low or high number of genes or counts. A lot of this quality control can be done using Seurat in R.

3. Pre-Processing: Seurat package provides an easy workflow for pre-processing of data and mainly consists of the following steps: - Raw counts need to be normalized to neutralize differences in sequencing depths across cells. Here, we use log normalization. - Next, we do feature selection where we find out highly variable genes that capture maximum variation among cells. This helps reduce noise. - Next, we perform scaling of the data to bring all the parameters to the same range. - This is followed by principal component analysis (PCA). PCA is typically performed to reduce the dimensionality and complexity of the data without compromising on the variance, followed by UMAP or TSNE projections to visualize clusters and relations between clusters. - Batch correction is important especially if the data comes from different batches or conditions. Harmony or Seurat's 'integrate' vignette allow us to do this step.

4. Replicates: It is important to know the difference between biological replicates and technical replicates and it is generally prudent to include biological replicates and collapse technical replicates.

5. Experimental Design: Ideally, a balance between condition and control must be maintained to remove any possible bias which can hamper the trajectory inference.

0.5 Using code from any vignettes found online (Github, etc), write a sample script for this analysis using the selected method(s)/package(s). This script should include enough annotations that members of the lab with little bioinformatic experience should be able to understand generally how the method works and what steps/parameters may need to be adjusted when working on data from a different project.

0.5.1 Prerequisites:

0.5.1.1 Softwares and packages Before running this analysis, ensure you have: - R (version >= 4.0) installed on your system. - The following R packages: - CRAN: `dplyr`, `ggplot2`, `Matrix`, `Seurat`, `patchwork` - Bioconductor: `slingshot`, `tradeSeq`, `DelayedMatrixStats`

0.5.1.2 File and directory paths Knowledge of your data directory structure (update paths accordingly in the script).

0.5.1.3 Input Data The input for this pipeline is a raw counts matrix, such as the output from a sequencing machine that has been quantified by suitable software. - Rows: Each row is a gene. - Columns: Each column is a cell. - File format: `.txt` or `.csv` with a header.

0.5.1.4 Workflow overview This pipeline processes single-cell RNA-seq data to identify genes that change along pseudotime trajectories. The steps include:

1. **Pre-processing**:
    - Remove poor-quality cells and genes.
    - Normalize the data to account for sequencing depth differences.
    - Identify highly variable genes.
    - Scale the data.
    - Perform PCA.
2. **Trajectory Inference**:
    - Use UMAP to cluster and visualize the clusters.

- Use Slingshot to construct cell lineages and pseudotime trajectories.

3. **Differential Expression Analysis**:
   - Use tradeSeq to identify genes with expression changes along trajectories.
   - Further use the output for functional enrichment analysis.

0.5.1.5 Expected Outputs

1. **UMAP Plot**: A plot showing cell clusters with their dimensions reduced.
2. **Pseudotime Trajectories**: Lineage curves constructed along pseudotime.
3. **Differential Expression Results**: Significant differentially expressed genes in a .csv file.
4. **Plots**: Plots visualizing journey of the highest differentially expressed gene along the trajectory.

0.5.2 The proposed study

In the bone marrow, stem cells differentiate into various types. This study used single-cell RNA sequencing to explore myeloid progenitors, showing distinct subgroups aimed to differentiate for seven different fates. The data, collected from bone marrow samples, shows how these cells progress along the lineage.

Design: Bone marrow Lin- cKit+ Sca1- myeloid progenitors mRNA profiles from single cells were generated by deep sequencing of thousands of single cells, sequenced in several batches in an Illumina NextSeq. Raw data files were processed as single-ended file since second read files contain only cell/molecule barcodes and therefore, not provided. This information was appended to the fastq entry header.

First, we install (if not already installed) and load the packages/libraries.
IMP: Add other packages here that you might need in your analysis and do not have installed!

We now have a code section where we keep note of all paths used.
IMP: Change these paths to make it suitable to your system!

```
#TODO: Modify paths to reflect your directory structure!
data_dir <- "~/scrna_project/data"
data_file <- paste0(data_dir, "/GSE72857_umitab.txt")
scripts_dir <- "~/scrna_project/scripts"
results_dir <- "~/scrna_project/results"
```

We have already downloaded the dataset and now we will load it here.

IMP: - Change the paths to the data as per your directory structure!
- Also remember to change the cell quality filter thresholds as per your dataset

```
#Read in the data
data <- read.delim(data_file,
                   header=TRUE,
                   row.names = 1)

#Save it as a matrix
comp_matrix <- Matrix::Matrix(as.matrix(data), sparse = TRUE)
saveRDS(comp_matrix, file = paste0(data_dir, "/counts.rds"))

#Load the data- select cells so that large dataset is easier to process
umi_counts <- readRDS(paste0(data_dir, "/counts.rds"))

#Check the structure and dimensions of the data
str(umi_counts)
```

```
## Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   ..@ i       : int [1:13646796] 453 455 503 553 657 1078 1171 1380 3205 3847 ...
##   ..@ p       : int [1:10369] 0 215 1608 1756 3229 5875 7344 7460 9184 11286 ...
##   ..@ Dim     : int [1:2] 27297 10368
##   ..@ Dimnames:List of 2
##   .. ..$ : chr [1:27297] "0610007C21Rik;Apr3" "0610007L01Rik" "0610007P08Rik;Rad26l" "0610007P14Rik"
##   .. ..$ : chr [1:10368] "W29953" "W29954" "W29955" "W29956" ...
##   ..@ x       : num [1:13646796] 1 1 1 1 1 1 1 1 1 1 ...
##   ..@ factors : list()
```

```
dim(umi_counts)
```

```
## [1] 27297 10368
```

```
#Create a Seurat object and filter for only good quality cells
data.seurat <- CreateSeuratObject(counts = umi_counts, project = "trajectory_scrna",
                                  min.cells = 3, min.features = 200)
```

0.5.3 Pre-processing Seurat workflow

Since pre-processing is a crucial step in this pipeline, we must do it using Seurat to make the file suitable for input into Slingshot. It consists of the following steps:

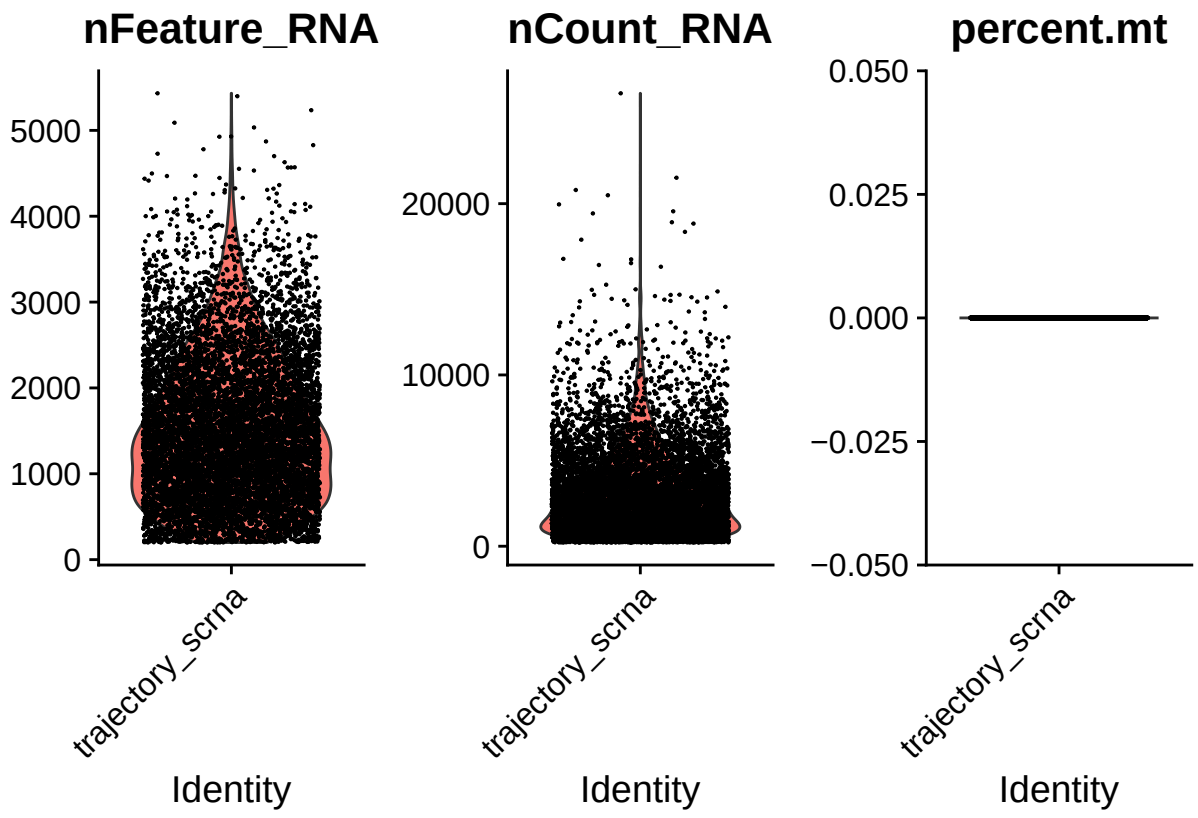
1. **Quality control**
2. **Normalize the data**
3. **Find variable features**
4. **Scale the data**
5. **PCA**

Quality control aims to remove cells which need to be excluded from the analysis. - Low quality cells which have very few genes (done while creating Seurat object) - Cells with high mitochondrial contamination, a feature typical in dying cells

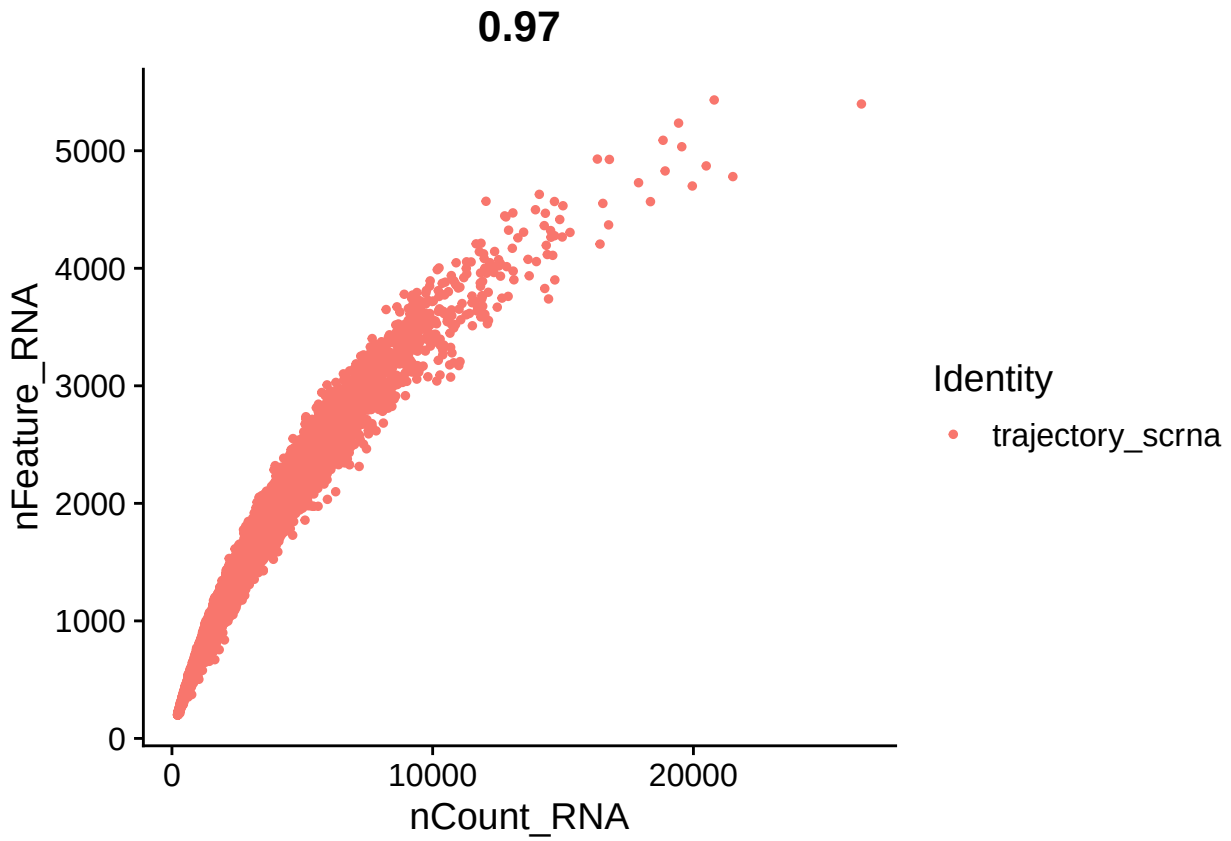
```
#Quality control- detect and remove high mitochondrial content cells
data.seurat[["percent.mt"]] <- PercentageFeatureSet(data.seurat, pattern = "^MT-")
#unique(data.seurat@meta.data$percent.mt)
```

The dataset selected has either already been quality checked for mitochondrial contamination or it naturally does not have cells contaminated with mitochondria.

```
#Visualize the quality control metrics using Violin plot
VlnPlot(data.seurat, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```




```
#Visualize the relation between the number of detected RNA molecules and detected genes
FeatureScatter(data.seurat, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
```



From the two plots above, we can observe a positive correlation between RNA and number of features/genes.

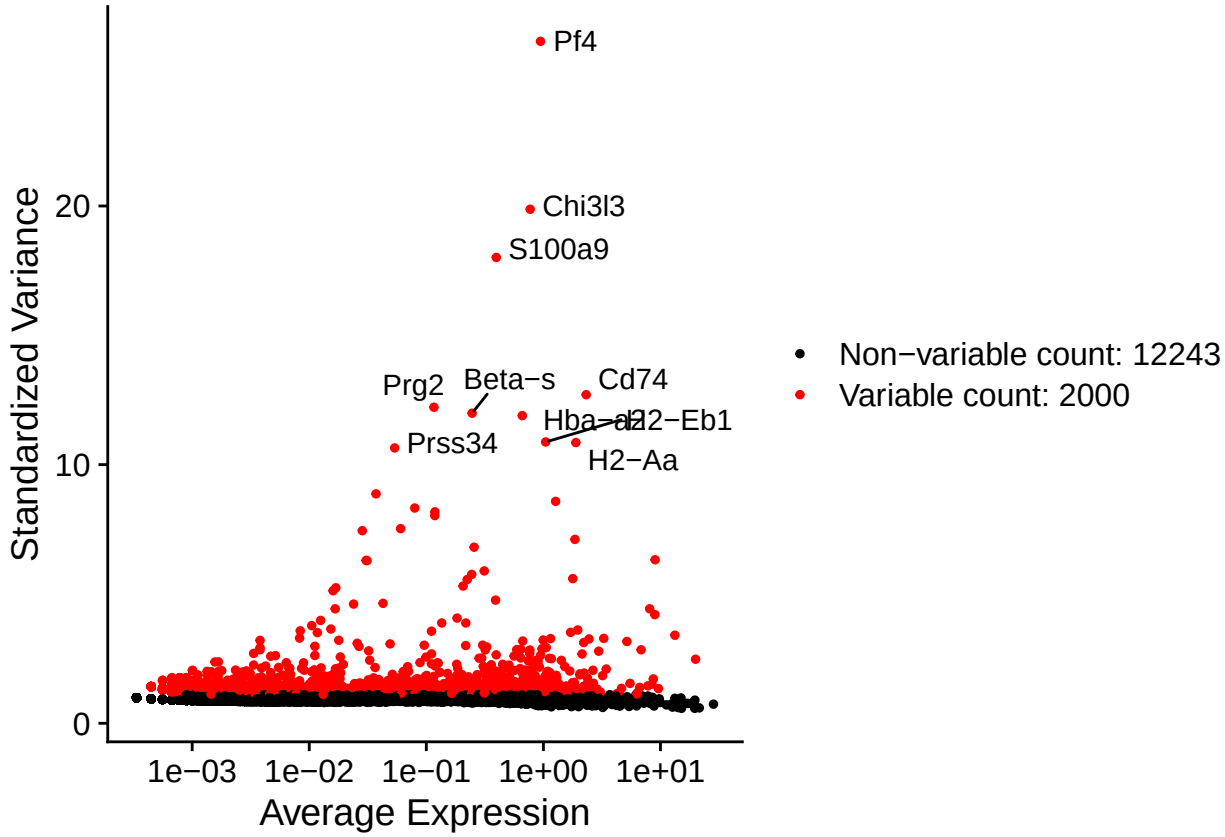
Cells with mitochondrial content have been removed already. Genes in red are the identified variable genes while the genes in black are non variable genes that are not of significance further downstream. In all likelihood, these genes are responsible for biological differences between the cells. The genes with highest variance are Pf4, Chi3l3, S100a9.

Normalize the data: We use log normalization to normalize the data

```
#Log normalize the data
data.seurat <- NormalizeData(data.seurat, normalization.method = "LogNormalize",
                             scale.factor = 10000)
```

Find variable features: Here, we identify genes/features that show high variation between cells

```
#Find the variable features
data.seurat <- FindVariableFeatures(data.seurat, selection.method = "vst",
                                    nfeatures = 2000)
#Identify the 10 most variable genes
top10 <- head(VariableFeatures(data.seurat), 10)

#Visualize the variable features
plot1 <- VariableFeaturePlot(data.seurat)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
plot2
```



Scale the data

```
#Linearly transform the data
all.genes <- rownames(data.seurat)
data.seurat <- ScaleData(data.seurat, features = all.genes)
```

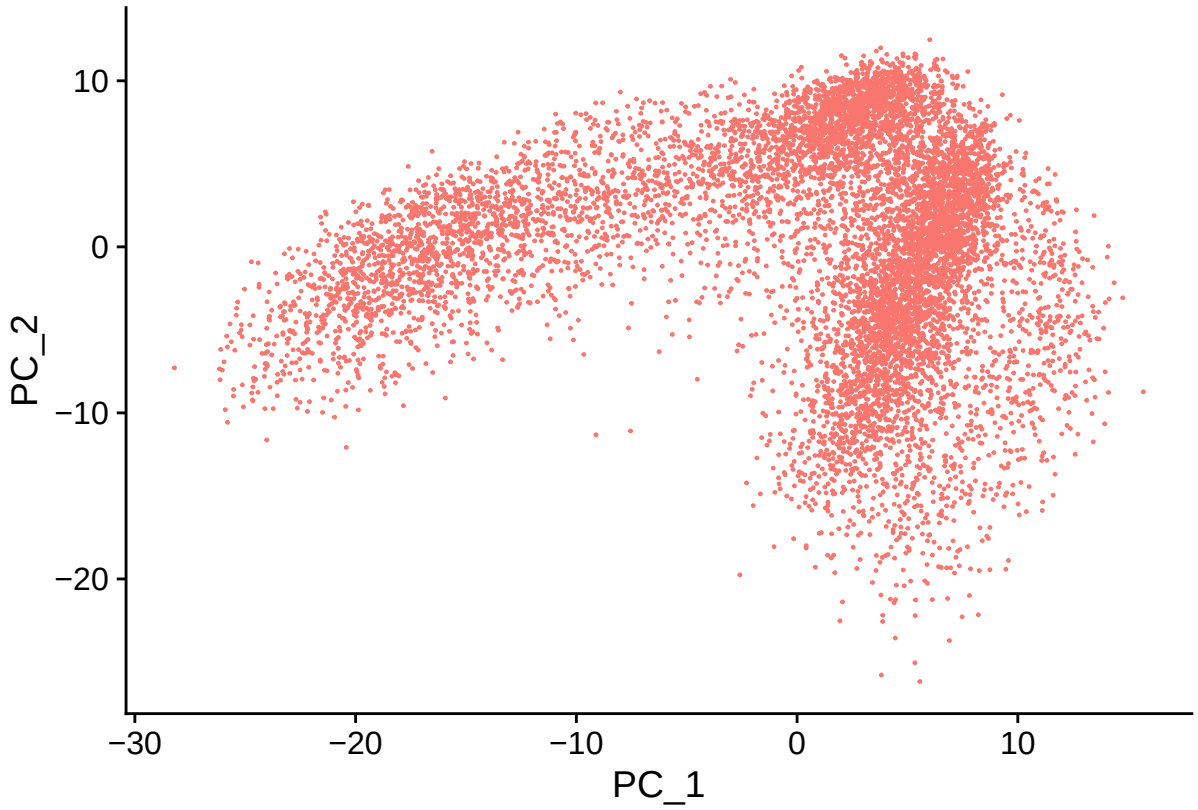
Run PCA on the scaled data

```
#Run PCA on the data
data.seurat <- RunPCA(data.seurat, features = VariableFeatures(object = data.seurat))
print(data.seurat[["pca"]], dims = 1:5, nfeatures = 5)
```

```
## PC_ 1
## Positive:  Coro1a, Vim, Cd52, Pkm2, Cst3
## Negative:  Ermap, Rhd, Car2, Blvrb, Fam132a
## PC_ 2
## Positive:  Apoe, Vamp5, Mfsd2b, Gata2, Nrgn
## Negative:  Hsp90b1, Tmsb4x, Ly6c2, Hspa5, 2810417H13Rik
## PC_ 3
## Positive:  Elane, Hp, Gstm1, Prtn3, Ly6c2
## Negative:  Cd74, H2-Aa, H2-Eb1, H2-Ab1, Plbd1
## PC_ 4
## Positive:  1190002H23Rik, Elane, Hp, C3, Fcnb
## Negative:  Eef1a1, Gpr56, Rpl23, Rpl22, Ifitm1
## PC_ 5
```

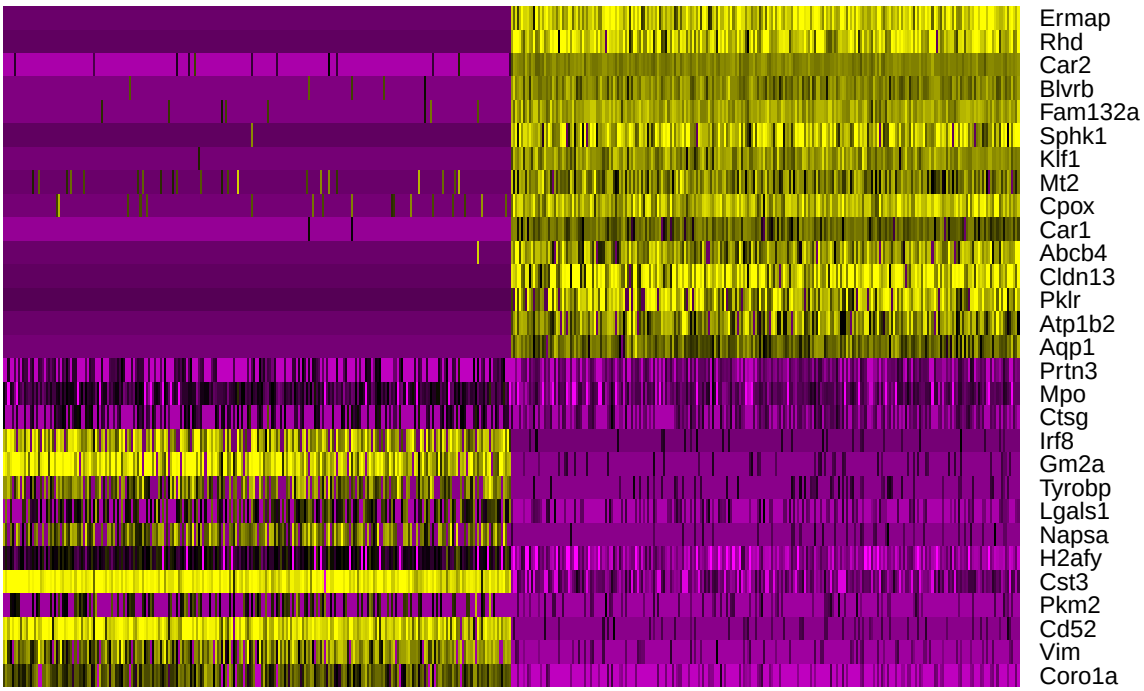
```
## Positive:  1100001G20Rik, Chi3l3, S100a9, Lcn2, Fcnb
## Negative:  Cd34, Irf8, Emb, Lgals1, H2afy
```

```
#Visualize the principal components and select how many PCs capture maximum variation
DimPlot(data.seurat, reduction = "pca") + NoLegend()
```



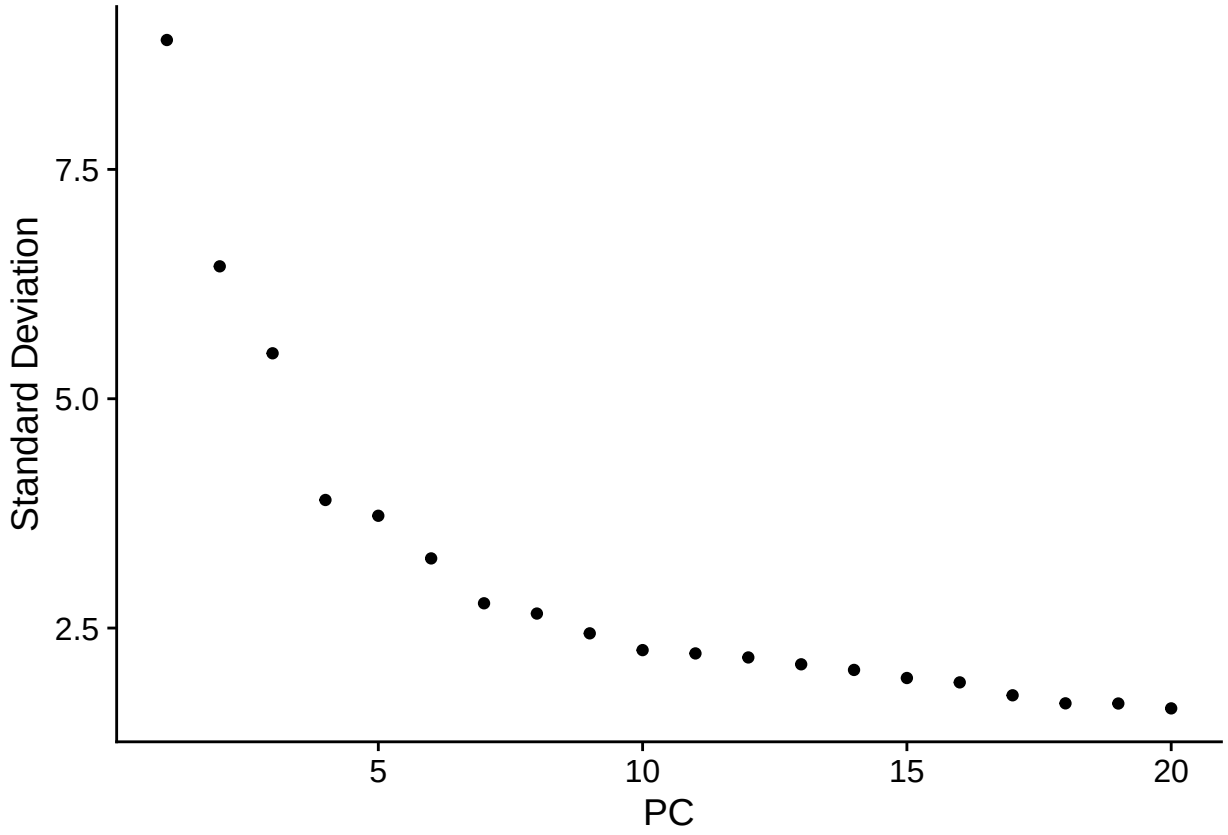
The dimensionality of the data has been successfully reduced to principal components, which should be able to capture the most amount of variance.

```
DimHeatmap(data.seurat, dims = 1, cells = 500, balanced = TRUE)
```



This heatmap shows genes that are responsible for most of the variance as captured by the first principal component. It shows how some genes are highly expressed (in yellow) in some cells and downregulated (in purple) in other cells . This indicates good differential expression.

```
ElbowPlot(data.seurat)
```



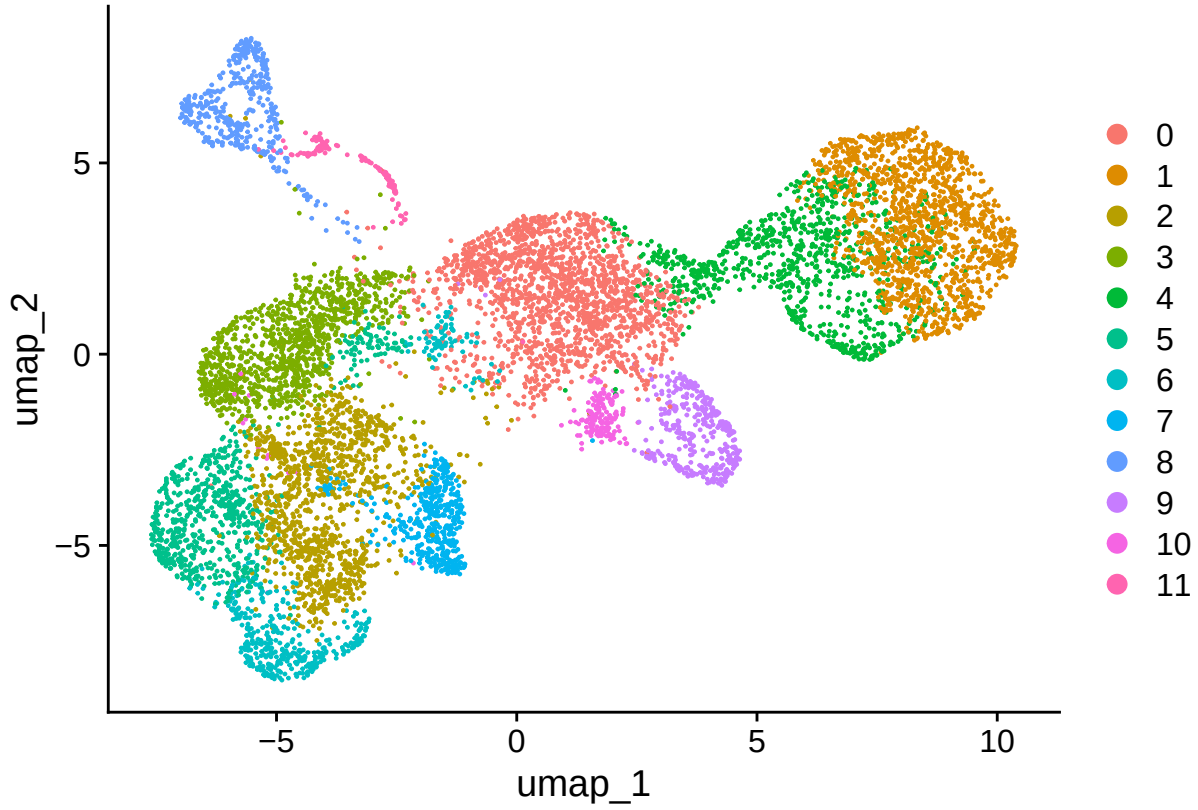
We can observe an 'elbow' around principal component 10, showing that the majority of true variation signal is captured in the first 10 PCs. Hence, we will use 10 dimensions when we find neighbors for clustering the cells in the next step.

0.5.4 Clustering of cells

```
#Find neighbors for clustering
data.seurat <- FindNeighbors(data.seurat, dims = 1:10)
data.seurat <- FindClusters(data.seurat, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 8949
## Number of edges: 280983
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8844
## Number of communities: 12
## Elapsed time: 1 seconds
```

```
data.seurat <- RunUMAP(data.seurat, dims = 1:10)
DimPlot(data.seurat, reduction = "umap")
```




```
#Save the objects as input suitable for slingshot
var.features <- VariableFeatures(data.seurat)
rownames(data.seurat@assays$RNA@layers$counts) <- rownames(data.seurat)

dimred <- data.seurat@reductions$umap@cell.embeddings
clustering <- data.seurat$RNA_snn_res.0.5
counts <- as.matrix(data.seurat@assays$RNA@layers$counts[var.features, ])
```

0.5.5 Annotating the clusters

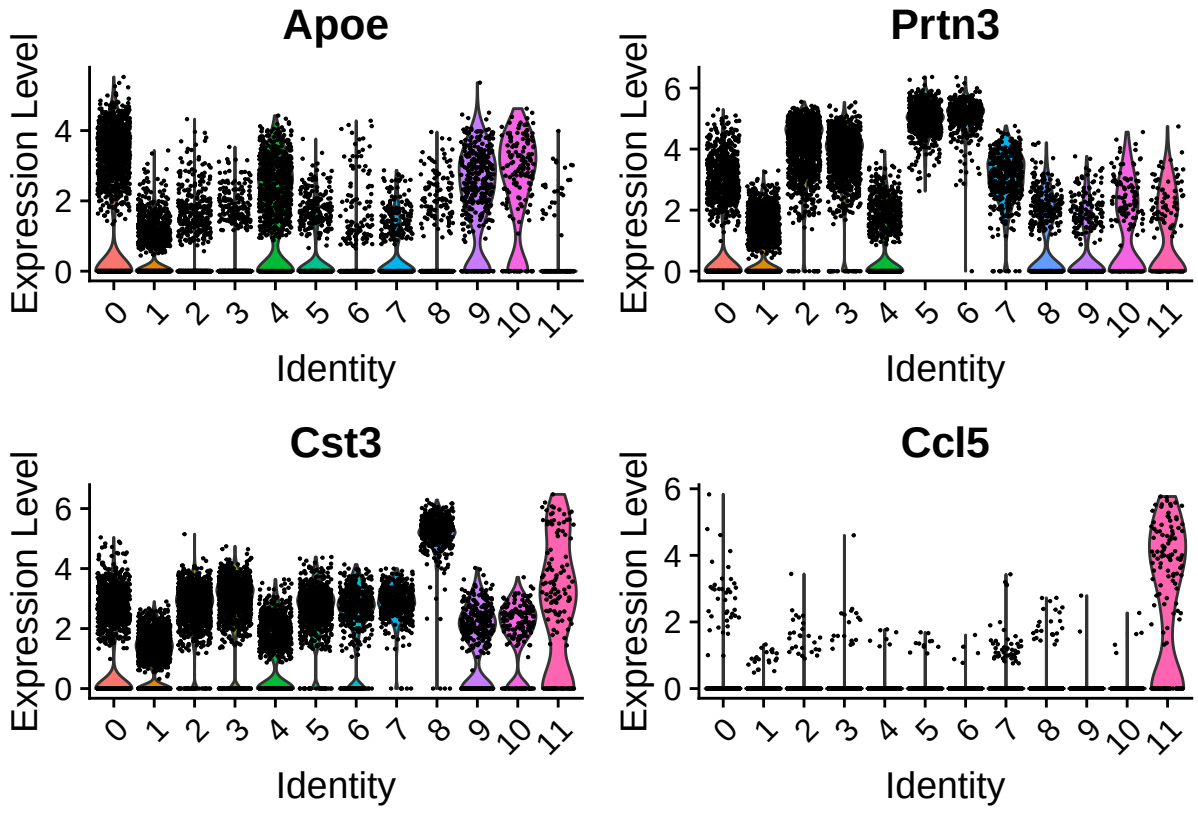
We can find marker genes for each cluster and annotate the clusters as per their cell type.

```
#Annotate the clusters by finding marker genes
markers <- FindAllMarkers(data.seurat, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)
head(markers)
```

```
##                 p_val avg_log2FC pct.1 pct.2     p_val_adj cluster    gene
## Apoe    1.662661e-204  2.5335474 0.628 0.375 2.368128e-200       0    Apoe
## Gata2    1.440081e-75  2.5272378 0.346 0.187  2.051107e-71       0   Gata2
## Ifitm1   9.884608e-71  2.8190436 0.377 0.231  1.407865e-66       0  Ifitm1
## Prpf38b  9.899069e-56  0.2699884 0.159 0.426  1.409924e-51       0 Prpf38b
## Myg1     2.108373e-53  0.2595361 0.128 0.367  3.002955e-49       0    Myg1
## Gpatch4  2.715448e-53  0.2525595 0.121 0.355  3.867612e-49       0 Gpatch4
```

Let's make a violin plot for the top gene in each cluster

```r
#Select the top marker for each cluster
top_gene <- markers %>%
  group_by(cluster) %>%
  top_n(1, wt = avg_log2FC)

#Get the marker names
#top_gene$gene

#Make the violin plot
plot1 <- VlnPlot(data.seurat, features = "Apoe", group.by = "seurat_clusters") + NoLegend()
plot2 <- VlnPlot(data.seurat, features = "Prtn3", group.by = "seurat_clusters") + NoLegend()
plot3 <- VlnPlot(data.seurat, features = "Cst3", group.by = "seurat_clusters") + NoLegend()
plot4 <- VlnPlot(data.seurat, features = "Ccl5", group.by = "seurat_clusters") + NoLegend()

(plot1 | plot2) /
(plot3 | plot4)
```



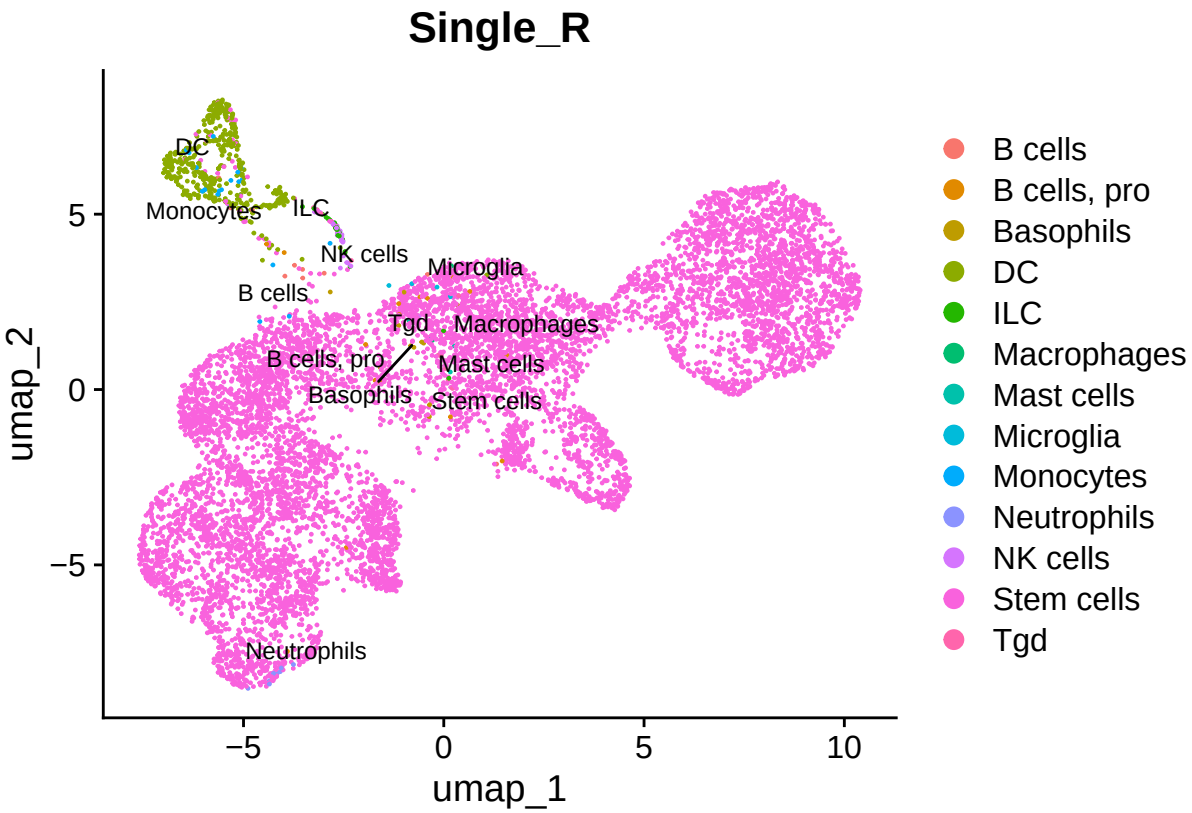

```r
#Select the top 5 markers for each cluster
top_genes <- markers %>%
  group_by(cluster) %>%
  top_n(5, wt = avg_log2FC)
```

The expression of these genes in each cluster is visible through this violin plot. It is important to note that these genes are statistically considered as top markers, but in a biological context, they can be present in

multiple clusters since the cells are following a lineage. We can now manually annotate the cells or use a package to annotate.

SingleR annotation

```
#Load ImmGen reference
ref <- celldex::ImmGenData()

#Run SingleR
pred <- SingleR(test = data.seurat@assays$RNA$data,
                ref = ref,
                labels = ref$label.main)

#Attach the annotations to the seurat object
data.seurat$Single_R <- pred$labels

#Visualize
DimPlot(data.seurat, group.by = "Single_R", label = TRUE, label.size = 3, repel = TRUE)
```



As we can see here, even though the clusters have been annotated, there seem to be clusters with mixed cells, because of which stem cells are dominating over all the clusters. This is where biological interpretation is important. Automated tools, like SingleR, are helpful for initial annotation, but they can mislabel clusters when cells share similar gene expression patterns, like stem or progenitor cells often do. This is where knowledge from research papers and established marker genes comes in. By comparing our results to other papers, we can resolve such tough to assign clusters, identify specific cell types, and ensure that the findings findings make sense in terms of our study. (This might also be a good checkpoint to check in with mentors!)

0.5.6 Slingshot mainly consists of two steps:

- Define the lineages
- Fit a curve through the data that defines a trajectory

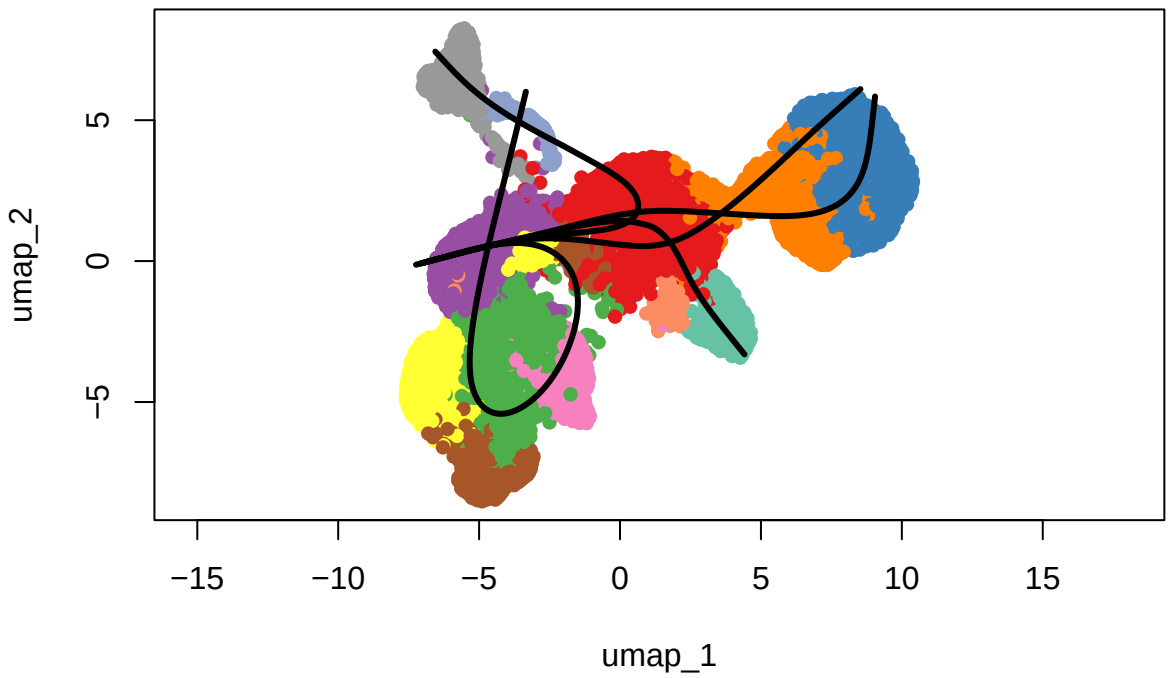
```
#Define the lineages
set.seed(1)
lineages <- getLineages(data = dimred,
                        clusterLabels = clustering,
                        start.clus = "0")
lineages
```

```
## class: PseudotimeOrdering
## dim: 8949 5
## metadata(3): lineages mst slingParams
## pathStats(2): pseudotime weights
## cellnames(8949): W29953 W29954 ... W76333 W76335
## cellData names(2): reducedDim clusterLabels
## pathnames(5): Lineage1 Lineage2 Lineage3 Lineage4 Lineage5
## pathData names(0):
```

```
#Extract curves from lineages
curves <- getCurves(lineages)
curves <- as.SlingshotDataSet(curves)
curves
```

```
## class: SlingshotDataSet
##
##  Samples Dimensions
##     8949          2
##
## lineages: 5
## Lineage1: 0  10  6  2  5  3
## Lineage2: 0  10  6  7
## Lineage3: 0  10  9
## Lineage4: 0  4  1
## Lineage5: 0  11  8
##
## curves: 5
## Curve1: Length: 24.06    Samples: 4096.3
## Curve2: Length: 17.867   Samples: 2387.65
## Curve3: Length: 14.042   Samples: 2281.61
## Curve4: Length: 19.152   Samples: 3682.09
## Curve5: Length: 17.306   Samples: 2285.48
```

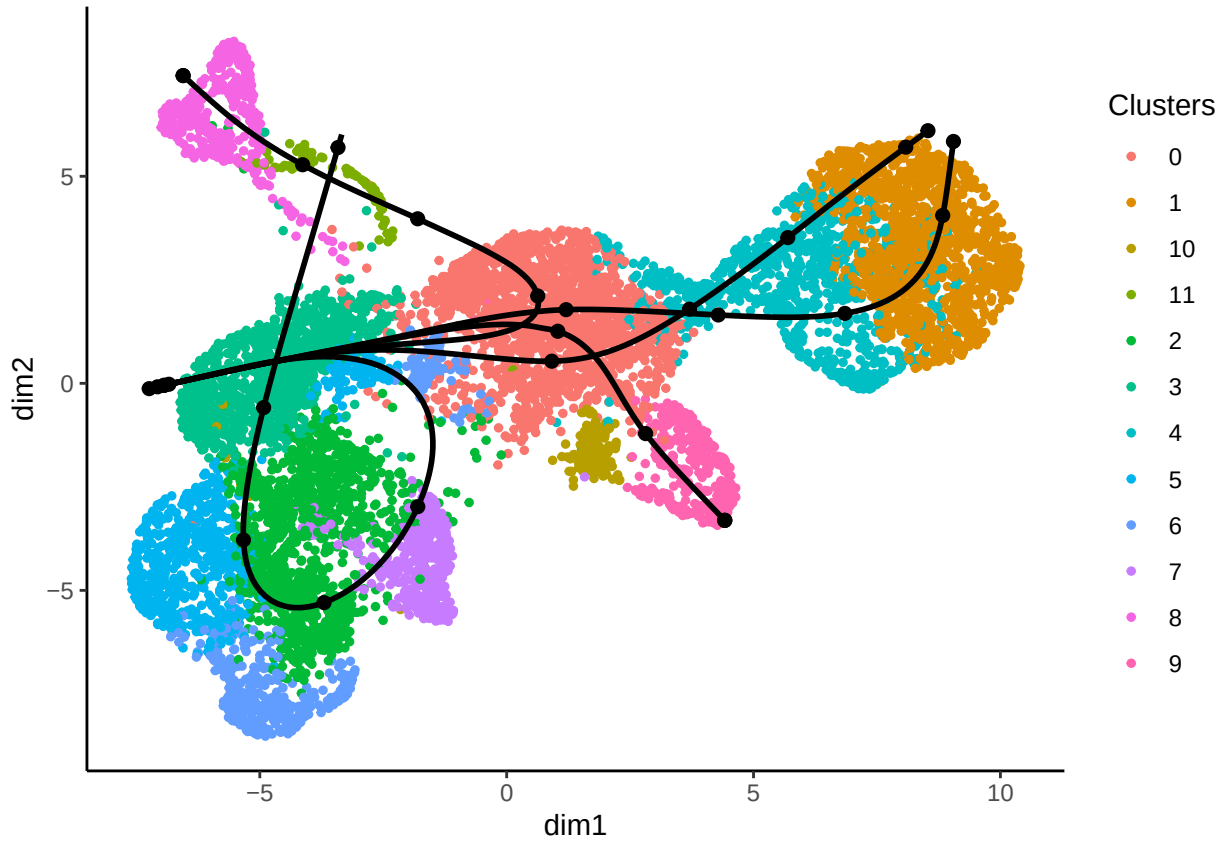
```
#Plot the curves
plot(dimred, col = pal[clustering], asp = 1, pch = 16)
lines(curves, lwd = 3, col = "black")
```



0.5.7 tradeSeq

Once Slingshot has identified pseudotime trajectories, we can go ahead with finding the differentially expressed genes. For this, we will use tradeSeq, which uses a GAM (Generalized Additive Model) to model gene expression changes along trajectories constructed by Slingshot. It takes input of the raw counts matrix and the lineage curves generated by Slingshot.

```
#Create a subset of the counts matrix so processing is finished at a reasonable time;
#skip if time is not a constraint
filt_counts <- counts[rowSums(counts > 5) > ncol(counts)/100, ]
#dim(filt_counts)
#Fit the model!
sce <- fitGAM(counts = as.matrix(filt_counts), sds = curves)
plotGeneCount(curves, filt_counts, clusters = clustering, models = sce)
```



Here we write a function that will help create a plot to visualize the most differentially expressed gene.

```
#Generate a plotting function
plot_differential_expression <- function(feature_id) {
    feature_id <- pseudotime_association %>%
      filter(pvalue < 0.05) %>%
      top_n(1, -waldStat) %>%
      pull(feature_id)

    cowplot::plot_grid(plotGeneCount(curves, filt_counts, gene = feature_id[1],
                                     clusters = clustering, models = sce)
                       + ggplot2::theme(legend.position = "none"),
                       plotSmoothers(sce, as.matrix(counts), gene = feature_id[1]))
}
```

This is followed by performing association tests, which check if a gene's expression changes along any trajectory. The input is the fitted model for each gene. We can find:

- Genes that change with pseudotime
- Genes that change between two pseudotime points
- Genes that are different between lineages

```
pseudotime_association <- associationTest(sce)
pseudotime_association$fdr <- p.adjust(pseudotime_association$pvalue, method = "fdr")
pseudotime_association <- pseudotime_association[order(pseudotime_association$pvalue), ]
pseudotime_association$feature_id <- rownames(pseudotime_association)
```

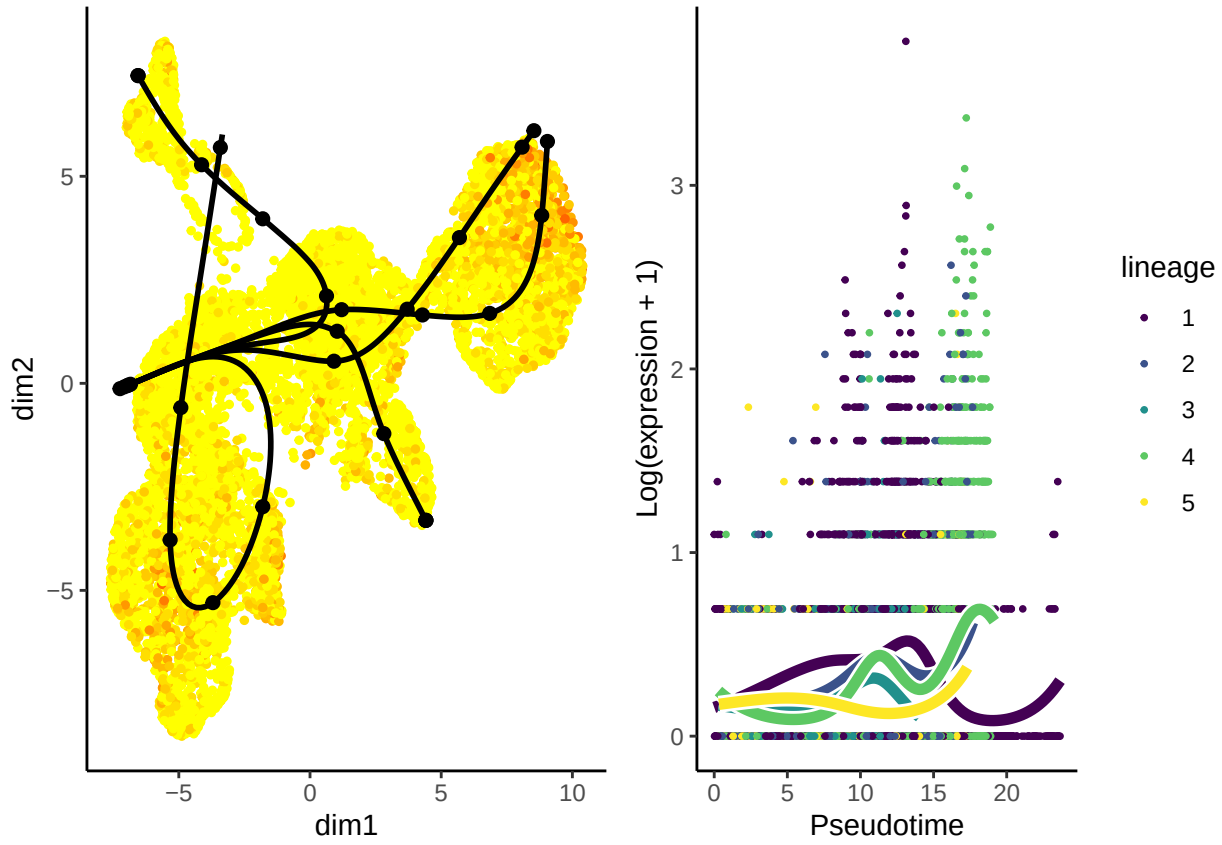
To find genes that change between pseudotime points, we can use:

- **`startVsEndTest()`**: Measures whether the differences in expression levels of a gene at the start of the trajectory are different from those at the end of the trajectory.

```
#TODO: Change the file paths to your output directory!#
#Using startVsEndTest
pseudotime_start_end_association <- startVsEndTest(sce, pseudotimeValues = c(0, 1))
pseudotime_start_end_association$feature_id <- rownames(pseudotime_start_end_association)
pseudotime_start_end_association <- pseudotime_start_end_association %>%
  dplyr::filter(pvalue < 0.05) %>%
  dplyr::arrange(desc(waldStat))

#Write the degs into a .csv file
write.csv(pseudotime_start_end_association, file = paste0(results_dir,
                                                          "/pseudotime_start_end_association_degs.csv"))
#Isolate highest differentially expressed gene
feature_id <- pseudotime_start_end_association %>%
  dplyr::slice(1) %>%
  pull(feature_id)
cat("The gene that is most differentially expressed is: ", feature_id)
```

```
## The gene that is most differentially expressed is:  Prtn3
```

```
plot_differential_expression(feature_id)
```



The black lines in the UMAP plot show the trajectories while the black dots are the trajectory nodes.The different shades of yellow color show the expression levels of this differentially expressed gene in different clusters. The plot on the right side shows the expression of the gene along different lineages.

To find the genes different between lineages, we can use two functions:

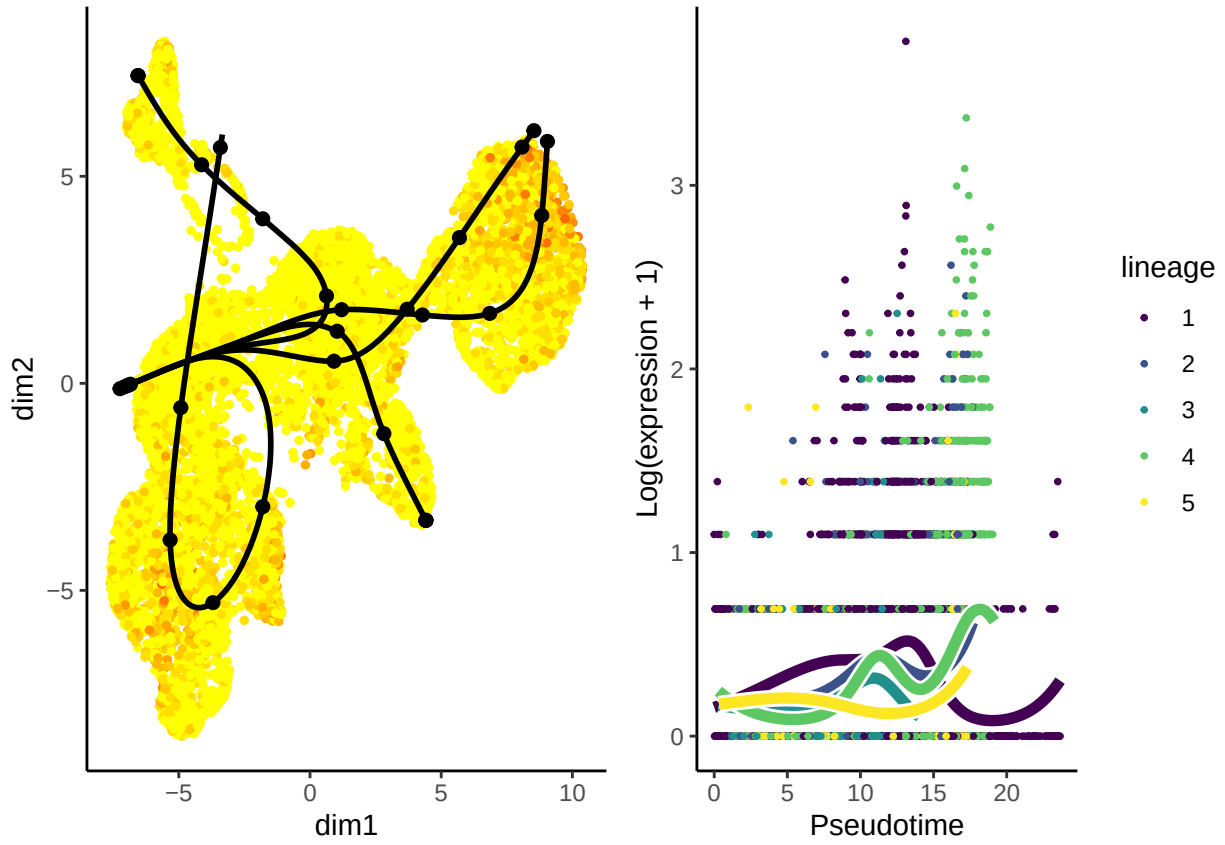
- **`diffEndTest()`**: Measures expression differences at the endpoints of the trajectory (terminal points). For example, identify genes differentiating two mature cell types.
- **`earlyDETest()`**: Measures expression differences early near branch points (near bifurcation). For example, identify genes regulating early trajectory splits.

Once we have the list of significant differentially expressed genes (filtered based on `pvalue < 0.05` and high test statistic), we can look at the trajectory of the most differentially expressed gene.

```
#TODO: Change the file paths to your output directory!#

#Using diffEndTest
different_end_association <- diffEndTest(sce)
different_end_association$feature_id <- rownames(different_end_association)
different_end_association <- different_end_association %>%
  dplyr::filter(pvalue < 0.05) %>%
  dplyr::arrange(desc(waldStat))
#Write the degs into a .csv file
write.csv(different_end_association, file = paste0(results_dir,
                                                   "/different_end_lineage_degs.csv"))

#Isolate highest differentially expressed gene
feature_id <- different_end_association %>%
  dplyr::slice(1) %>%
  pull(feature_id)

cat("The gene that is most differentially expressed is: ", feature_id)
```

```
## The gene that is most differentially expressed is:  Cst3
```

```
plot_differential_expression(feature_id)
```



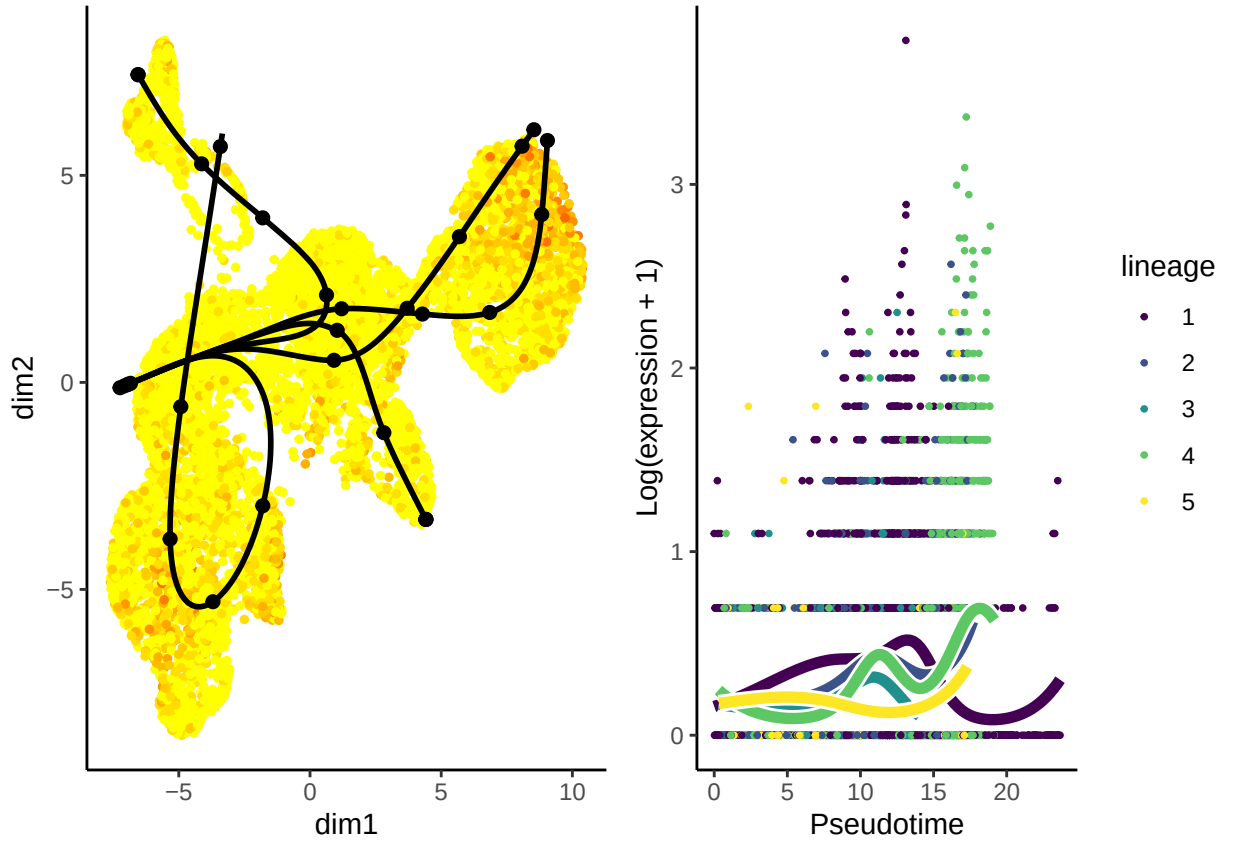
The UMAP plot with trajectories shows that there are a bunch of lineages which originate from the same starting point. The black lines show the trajectories while the black dots are the trajectory nodes.The different shades of yellow color show the expression levels of this differentially expressed gene in different clusters. These trajectories, in all likelihood, show different paths of cellular differentiation. Different colors along different trajectories shows that this gene is important in specific lineages.

The plot on the right side shows the expression of the gene along different lineages; for example, lineages 4 and 5 show a pretty sharp increase in this gene's expression towards the later parts of the pseudotime trajectory. This gene probably plays an important regulatory role in cellular differentiation along some lineages, especially lineages 4 and 5. It could be a regulatory gene that has important role towards the end of cellular differentiation.

```
#Using earlyDETest
early_DE_association <- earlyDETest(sce)
early_DE_association$feature_id <- rownames(early_DE_association)
early_DE_association <- early_DE_association %>%
  dplyr::filter(pvalue < 0.05) %>%
  dplyr::arrange(desc(waldStat))
#Write the degs into a .csv file
write.csv(early_DE_association, file = paste0(results_dir, "/early_DE_lineage_degs.csv"))

#Isolate highest differentially expressed gene
feature_id <- early_DE_association %>%
  dplyr::slice(1) %>%
  pull(feature_id)

cat("The gene that is most differentially expressed is: ", feature_id)
```

```
## The gene that is most differentially expressed is:  Prtn3
```

```
plot_differential_expression(feature_id)
```



We can draw similar conclusions for these plots as well; The black nodes in the UMAP plot indicate important points of the pseudotime, and cells in this plot are colored as per their expression for the gene being studied here. Some clusters and branches have higher expression of that gene. Very bright cell clusters (indicating higher gene expression) are the early clusters, indicating upregulation during initial parts of the cell differentiation process. It can possibly be inferred that this gene is involved in early regulation of cell differentiation.

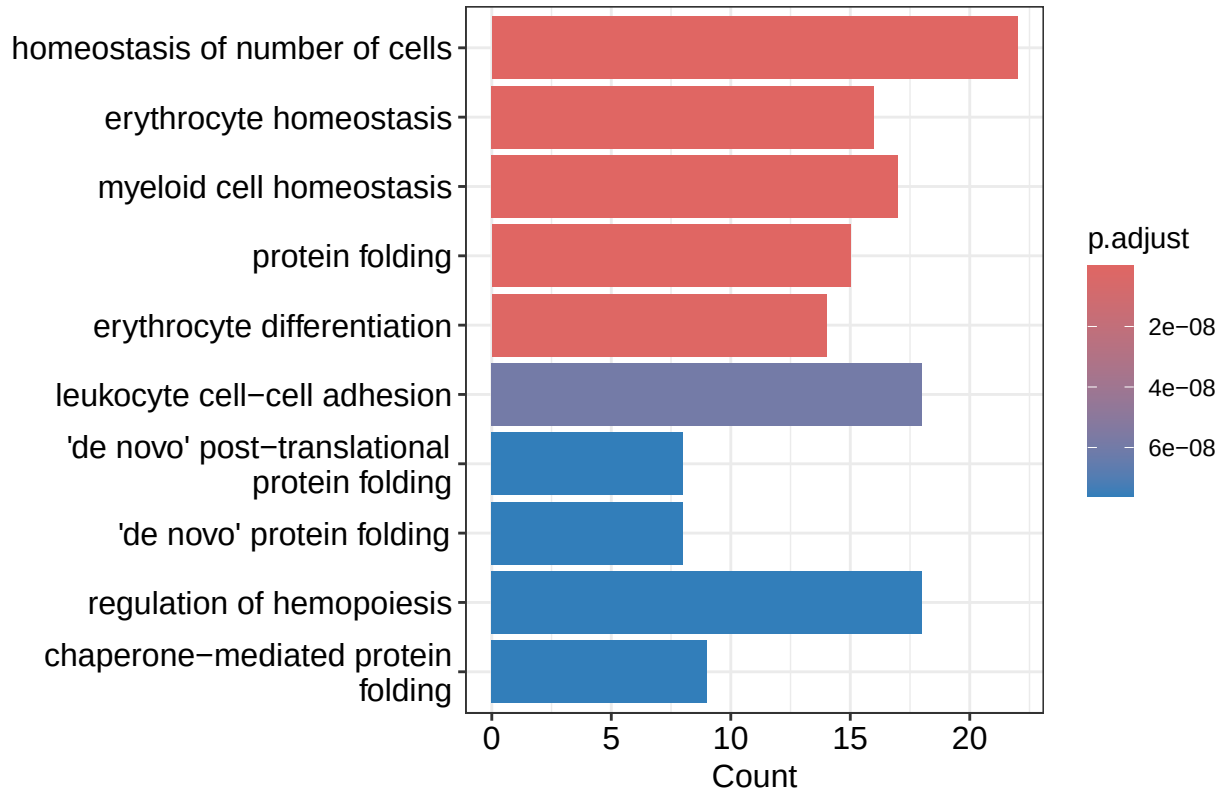
The plot on the right, like the one above, shows the pseudotime on the x axis and the gene expression on the y axis. Different trajectories/lineages are indicated by different colors like purple, yellow and green. The gene's expression is quite high in the early lineages, which is in accordance with the UMAP plot and indicates the gene's importance in early regulation of cell differentiation.

0.5.8 Functional enrichment

Functional annotation helps us understand the biological context of these differentially expressed genes by linking them to biological processes that they have been seen to affect. This allows us to uncover the biology behind cellular differentiation processes. We can use gene Ontology (GO) or KEGG pathway analysis for this purpose, and here we are using GO.

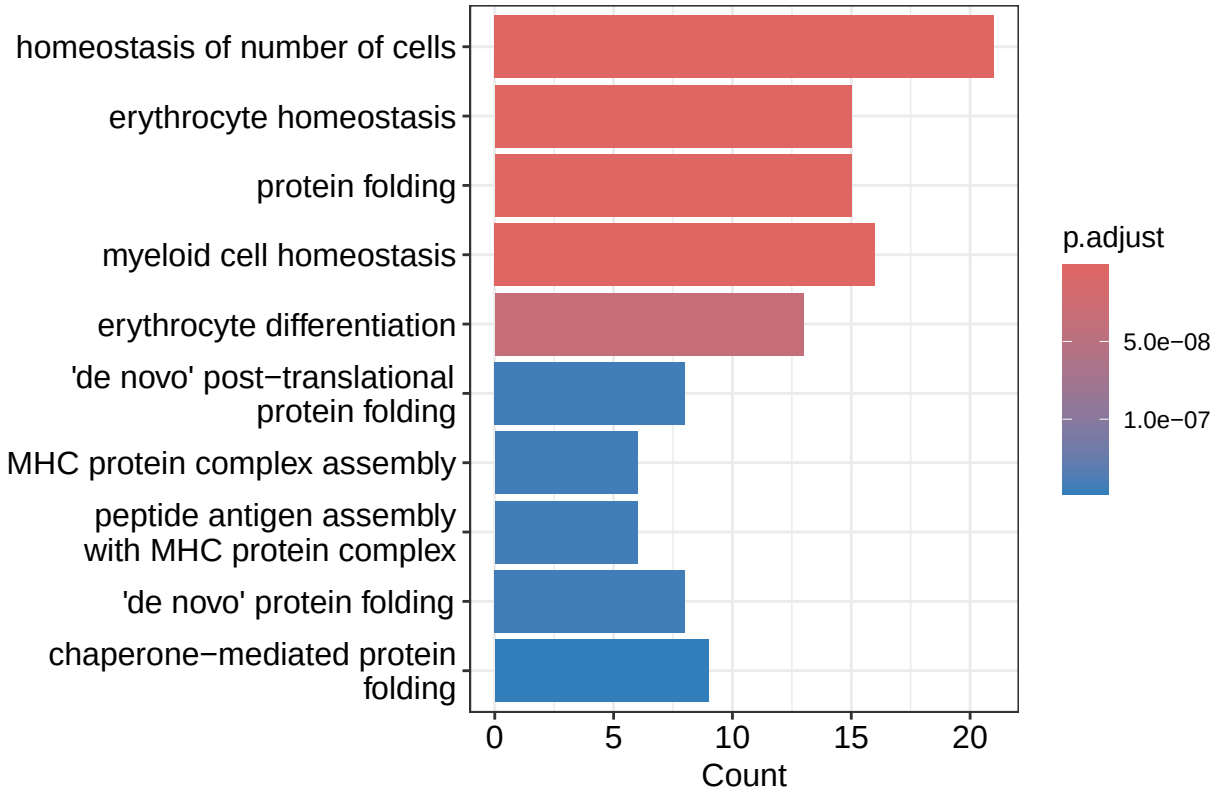
```
#####For startVsEndTest results#####
```

```r
#Ensure that the genome wide annotation for mouse is installed and loaded!
#Functional enrichment using Gene Ontology terms
go_results_startVsEnd <- enrichGO(
  gene          = pseudotime_start_end_association$feature_id,
  OrgDb         = org.Mm.eg.db,
  keyType       = "SYMBOL",
  ont           = "BP",
  pAdjustMethod = "BH",
  pvalueCutoff  = 0.05
)

#head(go_results_diffend)
barplot(go_results_startVsEnd, showCategory = 10)
```

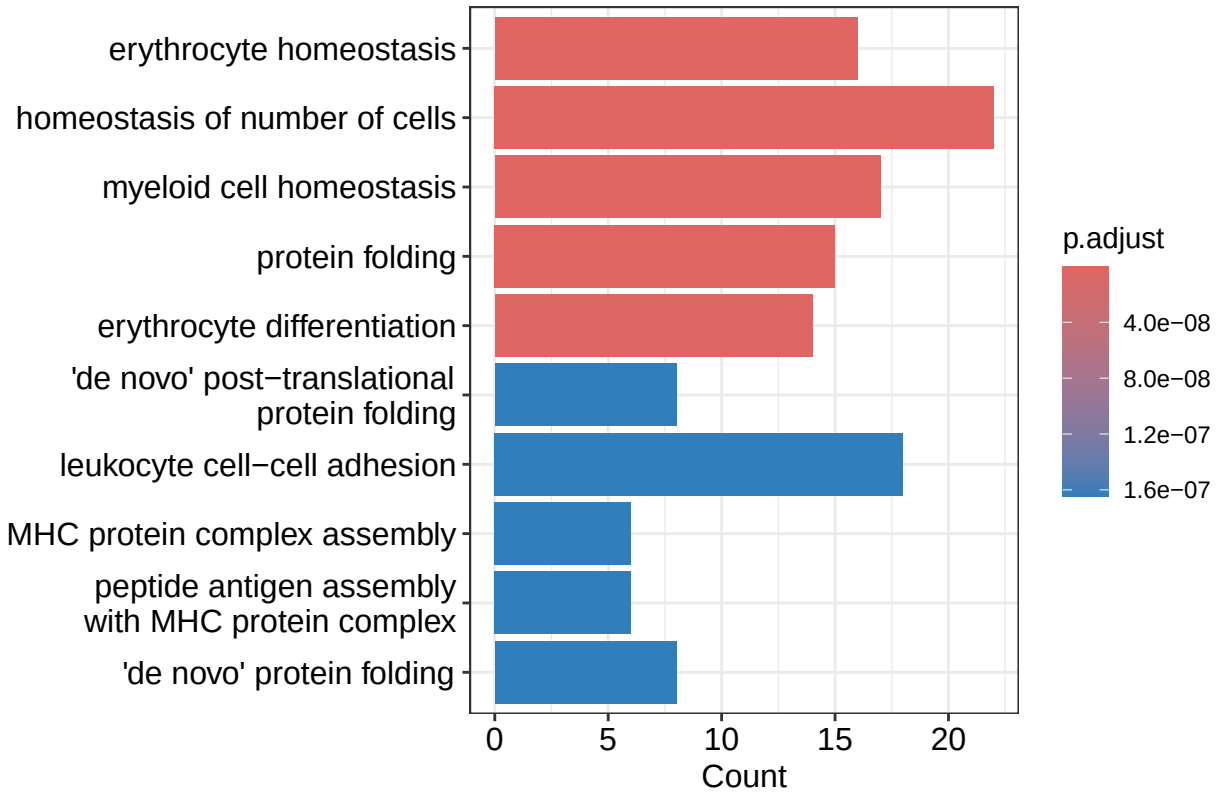



```r
#####For diffEndTest results#####

#Ensure that the genome wide annotation for mouse is installed and loaded!
#Functional enrichment using Gene Ontology terms
go_results_diffend <- enrichGO(
  gene          = different_end_association$feature_id,
  OrgDb         = org.Mm.eg.db,
  keyType       = "SYMBOL",
  ont           = "BP",
  pAdjustMethod = "BH",
  pvalueCutoff  = 0.05
```

```
)

#head(go_results_diffend)
barplot(go_results_diffend, showCategory = 10)
```




```
#####For earlyDEtest results#####
#Ensure that the genome wide annotation for mouse is installed and loaded!
#Functional enrichment using Gene Ontology terms
go_results_earlyde <- enrichGO(
  gene          = early_DE_association$feature_id,
  OrgDb         = org.Mm.eg.db,
  keyType       = "SYMBOL",
  ont           = "BP",
  pAdjustMethod = "BH",
  pvalueCutoff  = 0.05
)

#head(go_results_earlyde)
barplot(go_results_earlyde, showCategory = 10)
```



As we can see in these plots, the differentially expressed genes point towards upregulation of biological activities like erythrocyte differentiation and homeostasis. This brings us to the end of this analysis, but more work can be done on this to understand how trasncription factors could possibly cause cell differentiation as well. If possible, valuable insights could also be gained from ATAC-seq data.

0.6 References

Bioconductor. (2024, October 29). Slingshot: Trajectory Inference for Single-Cell data. Retrieved December 16, 2024, from https://bioconductor.org/packages/devel/bioc/vignettes/slingshot/inst/doc/vignette.html#identifying-global-lineage-structure

Butler, A., Hoffman, P., Smibert, P., Papalexi, E., & Satija, R. (2018). Integrating single-cell transcriptomic data across different conditions, technologies, and species. Nature Biotechnology, 36(5), 411–420. https://doi.org/10.1038/nbt.4096

Cannoodt, R., Saelens, W., & Saeys, Y. (2016). Computational methods for trajectory inference from single-cell transcriptomics. European Journal of Immunology, 46(11), 2496–2506. https://doi.org/10.1002/eji.201646347

Hao, Y., Stuart, T., Kowalski, M. H., Choudhary, S., Hoffman, P., Hartman, A., Srivastava, A., Molla, G., Madad, S., Fernandez-Granda, C., & Satija, R. (2023). Dictionary learning for integrative, multimodal and scalable single-cell analysis. Nature Biotechnology, 42(2), 293–304. https://doi.org/10.1038/s41587-023-01767-y

Hastie, T., & Stuetzle, W. (1989). Principal curves. Journal of the American Statistical Association, 84(406), 502–516. https://doi.org/10.1080/01621459.1989.10478797

NBI Sweden. (n.d.). Trajectory inference analysis: Slingshot. Retrieved December 15, 2024, from https://nbisweden.github.io/workshop-archive/workshop-scRNAseq/2020-01-27/labs/compiled/slingshot/slingshot.html#trajectory_inference_analysis:_slingshot

Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Winter, D., Lara-Astiaso, D., Gury, M., Weiner, A., David, E., Cohen, N., Lauridsen, F. K. B., Haas, S., Schlitzer, A., Mildner, A., Ginhoux, F., Jung, S., Trumpp, A., . . . Amit, I. (2015). Transcriptional heterogeneity and lineage commitment in myeloid progenitors. Cell, 163(7), 1663–1677. https://doi.org/10.1016/j.cell.2015.11.013

Saelens, W., Cannoodt, R., Todorov, H., & Saeys, Y. (2019). A comparison of single-cell trajectory inference methods. Nature Biotechnology, 37(5), 547–554. https://doi.org/10.1038/s41587-019-0071-9

Street, K., Risso, D., Fletcher, R. B., Das, D., Ngai, J., Yosef, N., Purdom, E., & Dudoit, S. (2018). Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. BMC Genomics, 19(1). https://doi.org/10.1186/s12864-018-4772-0

Van Den Berge, K., De Bézieux, H. R., Street, K., Saelens, W., Cannoodt, R., Saeys, Y., Dudoit, S., & Clement, L. (2020). Trajectory-based differential expression analysis for single-cell sequencing data. Nature Communications, 11(1). https://doi.org/10.1038/s41467-020-14766-3

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T., Miller, E., Bache, S., Müller, K., Ooms, J., Robinson, D., Seidel, D., Spinu, V., . . . Yutani, H. (2019). Welcome to the Tidyverse. The Journal of Open Source Software, 4(43), 1686. https://doi.org/10.21105/joss.01686