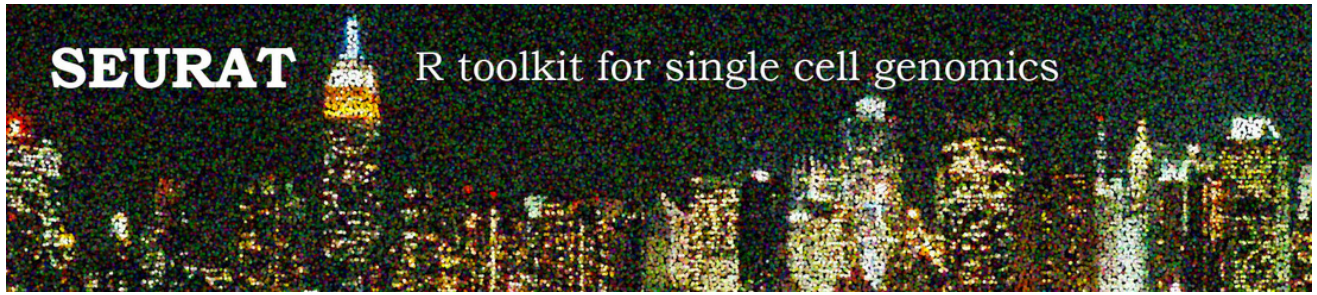
Analysis and interpretation of single-cell RNA-seq data Part II

Bioinformatics Workshop
February 22, 2021

Allegra Petti, PhD
Departments of Neurosurgery, Medicine, and Genetics
allegra.petti@wustl.edu



- R package (R3.5+) containing functions and data structures for single-cell data, including scRNA-seq, scATAC-seq, CITE-seq, etc.
- Streamlined Seurat pipeline from the homework: https://rnabio.org/module-08-scrna/0008/02/01/scRNA/
- Useful references:
  - https://satijalab.org/seurat/
  - Basic workflow and command list: https://satijalab.org/seurat/essential_commands.html
  - Details on getting information and data into and out of the Seurat object: https://github.com/satijalab/seurat/wiki
- Key terms:
  - Features = Genes (and/or proteins if using CITE-seq)
  - Counts = UMIs
  - Barcodes = Cells

Getting Started

- Option 1: RStudio
- Option 2: Use Docker container (e.g. satijalab/seurat)
- Open a text editor where you'll write and save an R script (e.g. as "script.r")
- Run your R script in the R command window by typing:
  `source("script.r")`
- Set working directory using `setwd("/path/to/data")`
- Load the libraries by typing the text below. (type `install.packages("libraryname")` if something is missing)

Libraries that are required and/or useful for running Seurat:

```
library("Seurat");
library("dplyr");
library("RColorBrewer");
library("ggthemes");
library("ggplot2");
library("cowplot");
```

Getting Help on R (and Seurat) functions

1. In R terminal, type:
   ?FunctionName
   example: ?FindAllMarkers

2. Code available on github, e.g.:
https://github.com/satijalab/seurat/blob/master/man/FindClusters.Rd

Seurat V3 workflow overview (for a single sample or pre-integrated samples)

Seurat functions in boldface blue font

```
scrna.counts <- Read10X(data.dir = "/yourpath/outs/filtered_feature_bc_matrix")
scrna <- CreateSeuratObject(counts = scrna.counts)
scrna <- SCTransform(scrna) # Find variable genes, scale, and normalize
     scrna <- NormalizeData(object = scrna) # older pipeline
     scrna <- FindVariableFeatures(object = scrna) # older pipeline
     scrna <- ScaleData(object = scrna) # older pipeline
scrna <- RunPCA(object = scrna) # Principal Component Analysis
scrna <- FindNeighbors(object = scrna)
scrna <- FindClusters(object = scrna)
scrna <- RunTSNE(object = scrna)
scrna <- RunUMAP(object = scrna)
DimPlot(object = scrna, reduction = "tsne")
UMAPPlot(object = scrna)
```

Introduction to the Seurat object

- In tutorial, the object name is “scrna”
- Best saved as an rds file:

```
saveRDS(scrna, file = “seurat.obj.rds”)
```

- A Seurat object saved as an rds file can be loaded as follows:

```
scrna <- readRDS(“seurat.obj.rds”)
```

- Contents (and how to access them):
  - Assay data (i.e. RNA-seq measurements): e.g. ‘RNA’
  - Meta data: summary statistics, sample name, cluster membership, cell cycle phase, batch, and other custom labels for each cell
  - Dimensionality reduction data, e.g. PCA, tSNE, UMAP data
- Overview of scrna contents:

```
str(scrna)
```

Seurat object: meta data

- Meta data:
  - summary statistics
  - ”Identities” for each cell
    - sample name
    - cluster membership
    - cell cycle phase
    - batch or sample
    - other custom labels
- Access using:

  `scrna[[]]`

  `scrna@meta.data`

  `str(scrna@meta.data)`

```
> str(scrna@meta.data)
'data.frame':   13049 obs. of  14 variables:
 $ orig.ident           : Factor w/ 1 level "452198": 1 1 1 1 1 1 1 1 1 1 ...
 $ nCount_RNA           : num  11846 3535 2850 9158 5607 ...
 $ nFeature_RNA         : int  3557 1740 1617 2810 2298 3834 1737 2526 2745 2947 ...
 $ percent.mito         : num  0.0386 0.0526 0.0589 0.0717 0.0462 ...
 $ percent.ribo         : num  0.1033 0.097 0.0821 0.1902 0.0885 ...
 $ S.Score              : num  -0.00337 -0.06351 0.30666 -0.09634 -0.0508 ...
 $ G2M.Score            : num  -0.428 -0.112 0.109 -0.28 -0.233 ...
 $ Phase                : Factor w/ 3 levels "G1","G2M","S": 1 1 3 1 1 1 3 1 1 1 ...
 $ CC.Difference        : num  0.4243 0.0486 0.1976 0.1833 0.182 ...
 $ Sample               : Factor w/ 2 levels "452198_P","452198_R": 1 1 1 1 1 1 1 1 1 1 ...
 $ RNA_snn_res.0.7      : Factor w/ 14 levels "0","1","2","3",..: 3 4 7 3 3 7 3 6 3 3 ...
 $ seurat_clusters      : Factor w/ 14 levels "0","1","2","3",..: 3 4 7 3 3 7 3 6 3 3 ...
 $ ClusterNames_0.7_17PC: Factor w/ 14 levels "0","1","2","3",..: 3 4 7 3 3 7 3 6 3 3 ...
 $ CellType             : Factor w/ 15 levels "B-CELL","BASO",..: 11 4 11 13 13 4 11 11 11 11 ...
```

- Combine with R commands such as head and str, e.g. str(scrna[[]])
- Access number of genes (aka “Features”) for each cell: `head(scrna@meta.data$nFeature_RNA)`
- Access number of UMIs for each cell: `head(scrna@meta.data$nCount_RNA)`
- What is the current default identity class? `unique(Idents(scrna))`
- What are the items in the current default cell identity class? `levels(x=scrna)`
- How many clusters are there?

  `length(unique(scrna@meta.data$seurat_clusters))`

  `levels(x=scrna)`
- What batches are included in this data set? `unique(scrna@meta.data$Batch)`
- Can add meta data

Seurat object cont'd: Assay data and Dimensionality reduction

- Assay data (i.e. RNA-seq measurements): e.g. 'RNA'
  - **Assay** = data type
  - Each assay has multiple "**slots**" that hold the raw data ('counts'), scaled data ('scale.data') or normalized data ('data')
  - Data transformation, such as scaling and normalization, adds new 'slots' to the assay data
  - Access values in the slots as follows:
    - raw RNA counts: `scrna[['RNA']]@counts[1:3,1:3]`
    - scaled data after SCTransform: `scrna[['SCT']]@scale.data[1:3,1:3]`
    - corrected UMI count data after SCTransform: `scrna[['SCT']]@counts[1:3,1:3]`
    - log-normalized data after SCTransform: scrna[['SCT']]@data[1:3,1:3]
  - Alternatively, use the function GetAssayData:
    - `GetAssayData(object = scrna, slot = 'scale.data')[1:3, 1:3]`
    - `GetAssayData(object = scrna, slot = 'counts')[1:3, 1:3]`
  - It's often important to know what 'slot' a function is using. Sometimes you can change it.

- Dimensionality reduction data, e.g. PCA, tSNE, UMAP data
  - Access using scrna[['pca']], scrna[['tsne']], scrna[['umap']]

Summary of common functions for the Seurat object

```
GetAssayData(object = scrna, slot = "counts")
GetAssayData(object = scrna, slot = "scale.data")
FetchData(object = scrna) # returns a data frame
colnames(x = scrna)
rownames(x = scrna)
VariableFeatures(object = scrna)
HVFInfo(object = scrna)
scrna[["assay.name"]] eg scrna[[“RNA”]]
scrna[["pca"]]
Embeddings(object = scrna, reduction = "pca")
Loadings(object = scrna, reduction = "pca")
Idents(object = scrna) # get default cell identities
Idents(object = scrna) <- "new.idents” # change the identities
Idents(object = scrna, cells = 1:10) <- "new.idents” # change the identities for specific cells
scrna$name <- vector # assign a vector of identities
scrna$name # e.g. scrna$seurat_clusters
scrna$saved.idents <- Idents(object = scrna) # change current identities for an existing identity class
levels(x = scrna) # get a list of the default identities (e.g. a list of clusters)
RenameIdents(object = scrna, "old.ident" = "new.ident") # change name of identity classes
WhichCells(object = scrna, idents = "ident.keep") # which cells are in cluster x?
WhichCells(object = scrna, idents = "ident.remove", invert = TRUE)
WhichCells(object = scrna, downsample = 500)
WhichCells(object = scrna, expression = name > low & name < high)
subset(x = scrna, cells = cellist, idents=‘keep’); # subset seurat object and return seurat object
subset(x = scrna, subset = name > low & name < high)
merge(x = object1, y = object2)
```

Additional functions and strategies for accessing data

```
Cells(scrna) # get list of cells

subcells <- Cells(scrna)[(which(scrna[["Sample"]]$Sample == sample1))] # choose cells
that have a given characteristic, here, that are in “sample1”


# alternative approach, choosing cells in cluster 4:

        Idents(object=scrna) <- "ClusterNames” # set default identity to ClusterNames

        WhichCells(scrna,idents="4") # use WhichCells

as.matrix(GetAssayData(scrna, slot = "counts")[, WhichCells(scrna, ident = '4')]) #
extract raw expression matrix for all cells in cluster 4

FetchData # subset Seurat object and return a data frame

subset # subset Seurat object and return a Seurat object

levels(scrna) # reorder the columns in Do.Heatmap
```

Note on printing plots to files:

Many Seurat plots are ggplot objects. To print them to a file you must explicitly call the print command (e.g. ‘print(plot)’)

```
# example:
jpeg(sprintf("%s/UMAP.10.%s.jpg", outdir, date), width = 10,
height = 8, units="in",res=300);
p1 <- DimPlot(object = scrna, reduction = "tsne", group.by =
"DataSet", pt.size=0.1)
p2 <- DimPlot(object = scrna, reduction = "umap", group.by =
"DataSet", pt.size=0.1)
print(plot_grid(p1, p2));
```

Some plotting functions

```
VlnPlot # for diagnostics
FeatureScatter # for diagnostics
DimPlot # for PCA, UMAP, or tSNE
UMAPPlot
PCAPlot
DimHeatmap # plot heatmap of top genes in given set of, eg, principle components
ElbowPlot # for PCA selection
JackStrawPlot # for PCA selection
Do.Heatmap # plot a heatmap of all cells, subsampling
* heatmap.2 in R is also useful
```

Read data, create a Seurat object, and perform initial filtering

Purpose:
- Eliminate genes with essentially no expression
- Eliminate cells with very few genes
- Reduce noise from rarely-expressed genes
- Make matrix smaller

Caution!
- Rare genes may be expressed in only a few cells
- Different cell types have different numbers of genes. Example: Red blood cells express only ~200 genes in some 10x data sets. -> Don't filter out potentially important cells.


```
scrna.counts <- Read10X(data.dir = "/yourpath/outs/filtered_feature_bc_matrix")
scrna <- CreateSeuratObject(counts = scrna.counts, min.cells = 10, min.features
= 100, project = Project1)
```

Multiplexed version: Combining multiple samples in Seurat

Purpose: Read, filter, and merge an arbitrary number of samples into a single Seurat object

```
# matrix.dirs is a list of directories containing 10x data

data.10x = list(); # declare a list of 10x data objects

for (i in 1:nsamples) { # nsamples = number of samples
    data.10x[[i]] <- Read10X(data.dir = matrix.dirs[i]);

}

scrna.list = list(); # a list of Seurat objects

for (i in 1:length(data.10x)) {
     scrna.list[[i]] = CreateSeuratObject(counts = data.10x[[i]], min.cells=10, min.feat
ures=100, project = “Project1”);
     scrna.list[[i]][["Sample"]] = samples[i]; # optional: assign a sample name from a v
ector ‘samples’, e.g. samples = c(“A”,”B”,”C”)

}
scrna <- merge(x=scrna.list[[1]], y=c(scrna.list[[2]],scrna.list[[3]]), add.cell.ids = c
(”name1","name2","name3”…)) # create merged Seurat object
```

Alternative: Combining multiple samples in cellranger

cellranger downsamples the data sets to the same [lowest] sequencing depth

```
cellranger aggr --id=$AggOutName --csv=$af --normalize=mapped --mempercore=64"
```

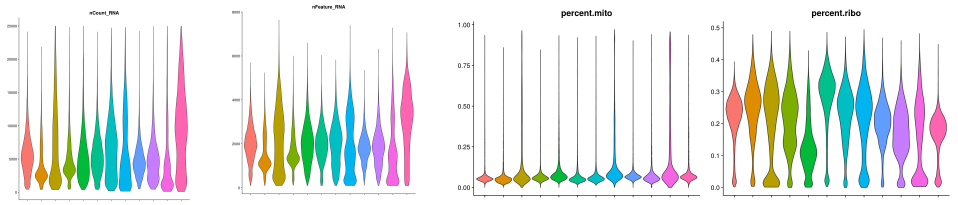
Definitions:
$AggOutName = Name of the output directory for the aggregated samples
$af = Full path to aggregation file, e.g. /path/to/aggregationfile.csv

Aggregation file format:
library_id,molecule_h5
Name1,/PathToData/Name1/outs/molecule_info.h5
Name2,/PathToData/Name2/outs/molecule_info.h5

Plot key parameters (per sample) to choose filtering cutoffs

Goals of Plotting and Filtering:

1. Eliminate apoptosing or leaky cells (high percentage of mitochondrial transcripts)
2. Eliminate free-floating transcripts or under-sequenced cells (too few UMIs or genes per cell)
3. Eliminate doublets (too many UMIs or genes per cell)
4. Know what's in your data
    - Is it dominated by a few genes?
    - Is it dominated by ribosomal protein-encoding genes?
    - How heterogeneous are the cells?
    - Does the data suggest that the sample preparation protocol need to be adjusted?



Sample R and Seurat code for parameter plots

NB: The number of genes and UMIs (nGene and nUMI) are automatically calculated for every object by Seurat and stored in the meta data.

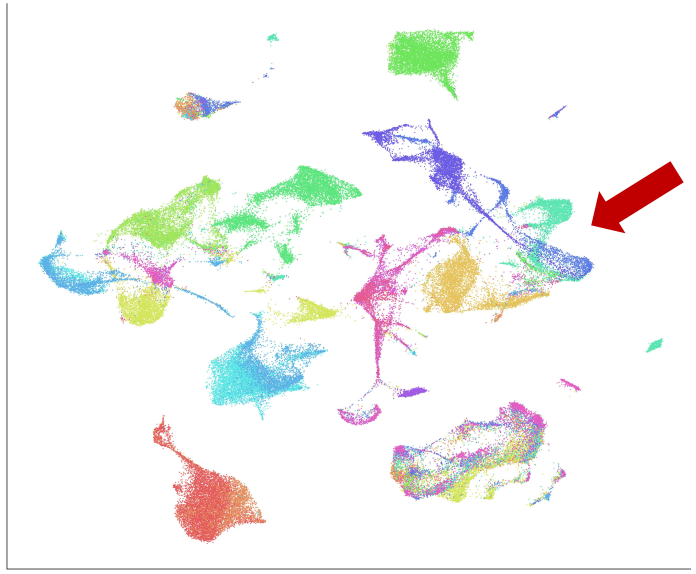
```
# Calculate percentage of mitochondrial genes
mito.genes <- grep(pattern = "^MT-", x = rownames(x = scrna), value = TRUE);
percent.mito <- Matrix::colSums(x = GetAssayData(object = scrna, slot = 'counts')[mito.genes, ]) /
Matrix::colSums(x = GetAssayData(object = scrna, slot = 'counts'));
scrna[['percent.mito']] <- percent.mito; # assign it to the meta data

# ribosomal genes
ribo.genes <- grep(pattern = "^RP[SL][[:digit:]]", x = rownames(x = scrna), value = TRUE);
percent.ribo <- Matrix::colSums(x = GetAssayData(object = scrna, slot = 'counts')[ribo.genes, ]) /
Matrix::colSums(x = GetAssayData(object = scrna, slot = 'counts'));
scrna[['percent.ribo']] <- percent.ribo; # assign it to the meta data

vln <- VlnPlot(object = scrna, features = c("percent.mito", "percent.ribo"), pt.size=0, ncol = 2,
group.by="Batch"); # make a violin plot, and color by batch or sample

vln <- VlnPlot(object = scrna, features = "nCount_RNA", pt.size=0, group.by="Batch", y.max=25000)

vln <- VlnPlot(object = scrna, features = "nFeature_RNA", pt.size=0, group.by="Batch")
```

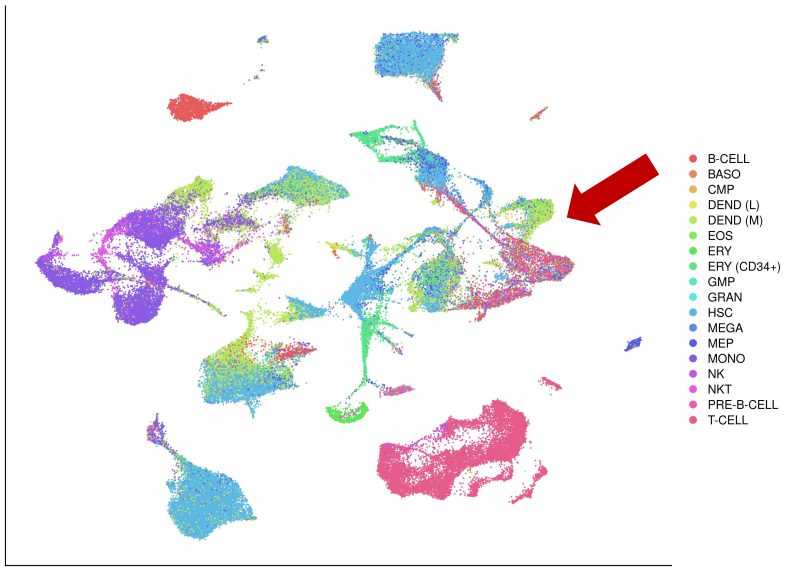
Filter data

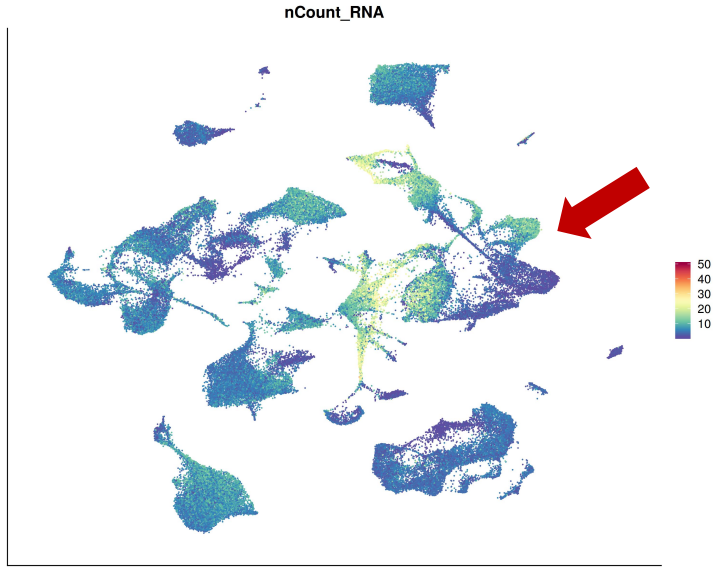
- Purpose: Remove cells with high mitochondrial content, too many genes, too few genes, etc
- Use the plots on the previous slides to choose appropriate thresholds for your data
- Some example thresholds:
  - nFeature_RNA between 200 and 4000, or between 200 and 95th percentile (for example)
  - nFeature > x, nUMI < 93rd percentile
  - Mitochondrial content below 5% or 10%
  - Ribosomal content below 50%

```
scrna <- subset(x = scrna, subset = nFeature_RNA > 200 & nFeature_RNA < Feature95 & percent.mito < mc.hi)
```

Example of possibly inadequate filtering



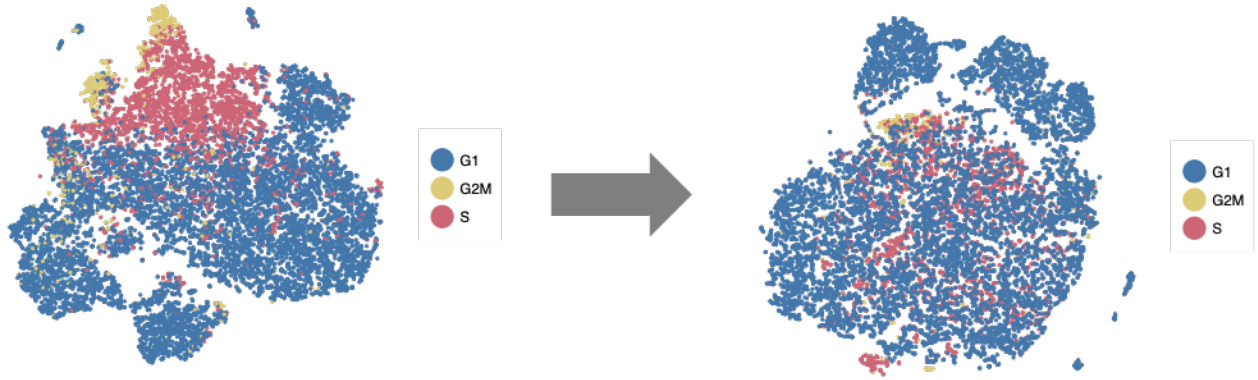




Metric	Cutoff/threshold/range
Genes	>700
UMI	< 93 percentile
Mitochondrial %	<10%

Calculate cell cycle phase of each cell

- Purpose: Cell cycle signature can dominate tSNE/UMAP plots and may need to be removed
- Seurat has a built-in function that calculates relative expression level of G1/S and G2/M genes defined in Tirosh et al 2016



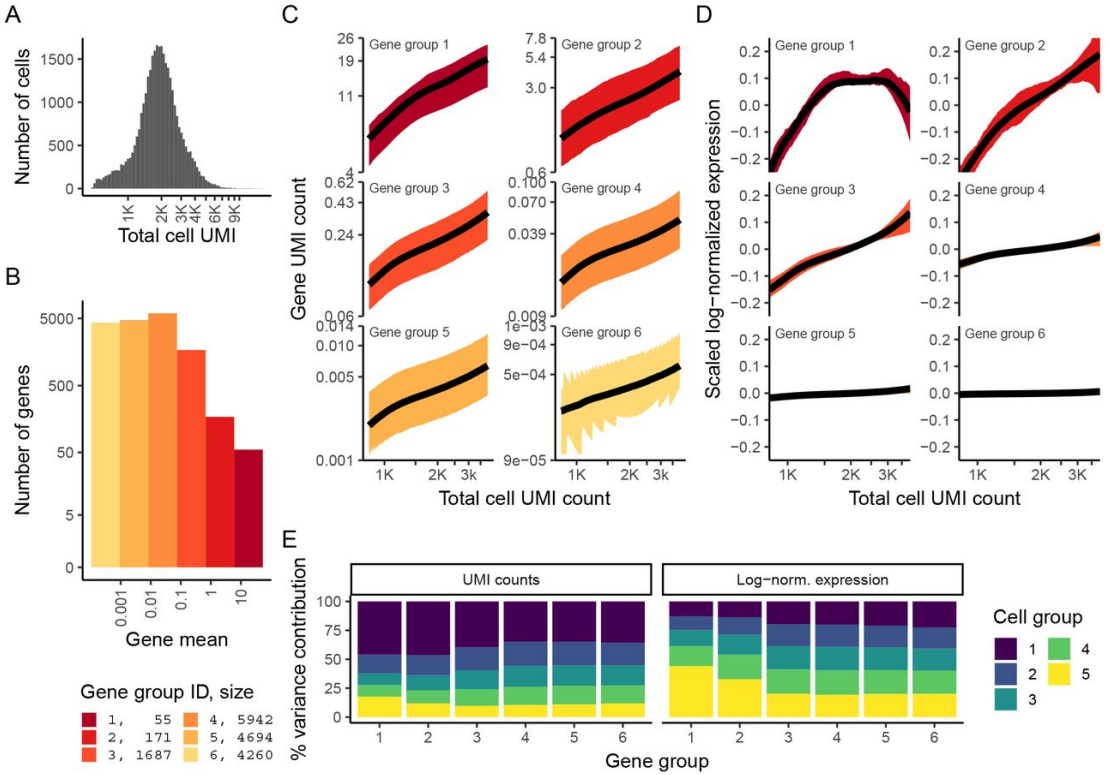

```
cell.cycle.tirosh <- read.table("CellCycleTirosh.txt", sep='\t', header=FALSE);
s.genes = cell.cycle.tirosh$V2[which(cell.cycle.tirosh$V1 == "G1/S")];
g2m.genes = cell.cycle.tirosh$V2[which(cell.cycle.tirosh$V1 == "G2/M")];
scrna <- CellCycleScoring(object=scrna, s.features=s.genes, g2m.features=g2m.genes, set.ident=FALSE)
```

Step 5: Normalize, scale, control for unwanted variation

- **Goal: remove technical effects while preserving biological variation**
- Sequencing depth of a cell: total UMI (nCounts)
- Even in the same experiment, different cells have very different sequencing depths
- Expression level of a gene in a cell is proportional to the sequencing depth of the cell, unless the data is normalized to sequencing depth
- Older normalization approach(es) scale every gene in the cell by the same factor (sequencing depth)
  - **NormalizeData**:
    - Divide by total counts in each cell
    - Scale to fixed counts (default is $1x10^4$ use $1x10^6$ for CPM)
    - Add 1
    - natural log
  - **FindVariableFeatures:** use a variance stabilizing transformation
  - **ScaleData:** subtract mean, divide by standard deviation, remove unwanted signal using multiple regression
- *But this affects different genes differently.* 😮

Regularized negative binomial normalization with Seurat's SCTransform function

- Highly and lowly expressed genes (B, C) are affected differently by scaling factor (D)
- Variance related to sequencing depth, and traditional normalization unevenly changes contribution of each gene to overall variance
- Solution: Use negative binomial regression (with parameters derived from groups of genes with similar expression) to remove impact of sequencing depth. Seurat function SCTransform.



Hafemeister and Satija. bioRxiv 2019

Two approaches to normalization and scaling in Seurat:

```
# I. Variance Stabilizing Transformation (with removal of cell cycle signal):

scrna <- NormalizeData(object = scrna, normalization.method = "LogNormalize", scale.factor = 1e4) # fea
ture counts divided by total, multiplied by scale factor, add 1, ln-transformed

scrna <- FindVariableFeatures(object = scrna, selection.method = 'vst', mean.cutoff = c(0.1,8), dispers
ion.cutoff = c(1, Inf)) # designed to find ~2000 variable genes

scrna <- ScaleData(object = scrna, features = rownames(x = scrna), vars.to.regress = c("S.Score","G2M.S
core"), display.progress=FALSE) # center and regress out unwanted variation

# II. SCTransform (with removal of cell cycle signal):

scrna <- SCTransform(scrna, vars.to.regress = c("S.Score", "G2M.Score"), verbose=FALSE)
```

The fine print:

• scrna[["SCT"]]@scale.data contains the residuals (normalized values), and is used directly as input to PCA. To save memory, we store these values only for variable genes, by setting the return.only.var.genes = TRUE by default in the SCTransform function call.

• To assist with visualization and interpretation, we also convert Pearson residuals back to 'corrected' UMI counts. You can interpret these as the UMI counts we would expect to observe if all cells were sequenced to the same depth.

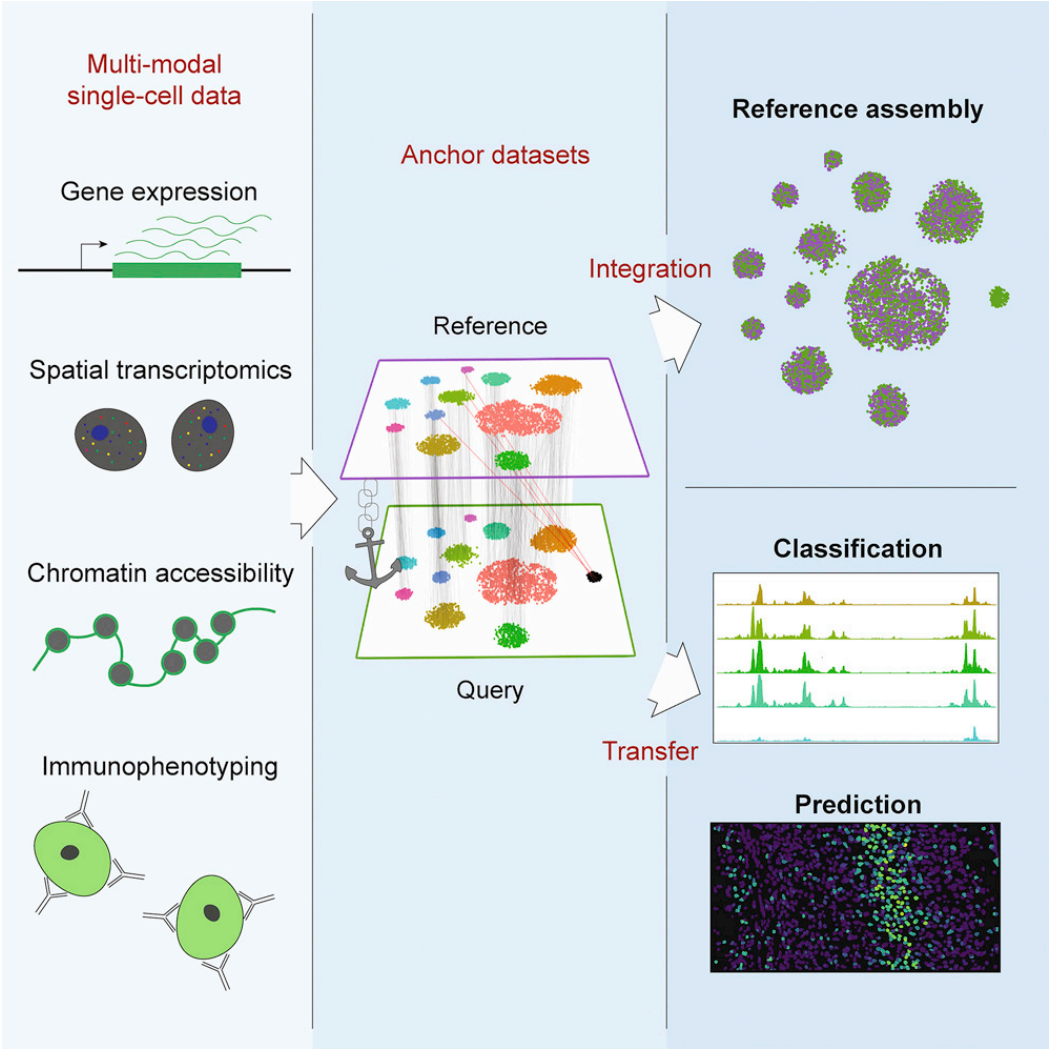
• The 'corrected' UMI counts are stored in scrna[["SCT"]]@counts. We store log-normalized versions of these corrected counts in scrna[["SCT"]]@data, which are very helpful for visualization.

• You can use the corrected log-normalized counts for differential expression and integration. However, in principle, it would be most optimal to perform these calculations directly on the residuals (stored in the scale.data slot) themselves. This is not currently supported in Seurat v3, but will be soon.

Batch correct/integrate instead of merging (optional)

When to use it:

- Correct for different technologies (e.g. 3’ and 5’)
- Correct for different batches with observable biases
- Discover conserved biology by finding corresponding cells across different data sets
- Combining data of different types (e.g. scRNA-seq, ATAC-seq)



Batch correct/integrate instead of merging (cont'd)

```
# matrix.dirs is a list of directories containing 10x data
data.10x = list(); # declare list of 10x data sets
for (i in 1:length(matrix.dirs.)) { # for each data set
     data.10x[[i]] <- Read10X(data.dir = matrix.dirs[i]); # add it to the list
}
scrna.list = list(); # declare list of Seurat objects
for (i in 1:length(data.10x)) { # for each data set…
    scrna.list[[i]] = CreateSeuratObject(counts = data.10x[[i]], min.cells=10, min.features=100,project
=batch[i]); # …create a Seurat object for each data set
    scrna.list[[i]][["Batch"]] = batch[i]; # assign a batch label from 'batch'
    scrna.list[[i]][["Sample"]] = samples[i]; # assign a sample name from 'samples'
}
# Not shown: Filter each sample separately and normalize (using same method for all samples)
anchors <- FindIntegrationAnchors(object.list = scrna.list, dims = 1:30) # find anchors
scrna.int <- IntegrateData(anchorset = anchors, dims = 1:30) # Integrate data
DefaultAssay(object = scrna.int) <- "integrated" # make integrated data the default for downstream anal
yses
```

Batch correction and integration: The fine print

- Integrated values not intended for use with differential expression calculations.
    - We recommend running your differential expression tests on the “unintegrated” data. By default this is stored in the “RNA” Assay. There are several reasons for this.
    - The integration procedure inherently introduces dependencies between data points. This violates the assumptions of the statistical tests used for differential expression.
- TransferData function uses data integration to classify cells based on a reference data set.
- Cellranger does faux batch-correction (corrected values are discarded), but batch-corrected tSNE can be visualized in the loupe browser.

Dimensionality reduction using PCA

Purpose: Approximate original data using fewer dimensions. Define new axes that capture as much "information" as possible in as few dimensions as possible.

```
scrna <- RunPCA(object = scrna, npcs = 100, verbose = FALSE);
```

Ancillary useful functions:

- Which genes contribute to most to each component? `VizDimLoadings`
- Vizualize PCs as heatmaps: `DimHeatmap`
- Rank genes by weight to perform functional enrichment:

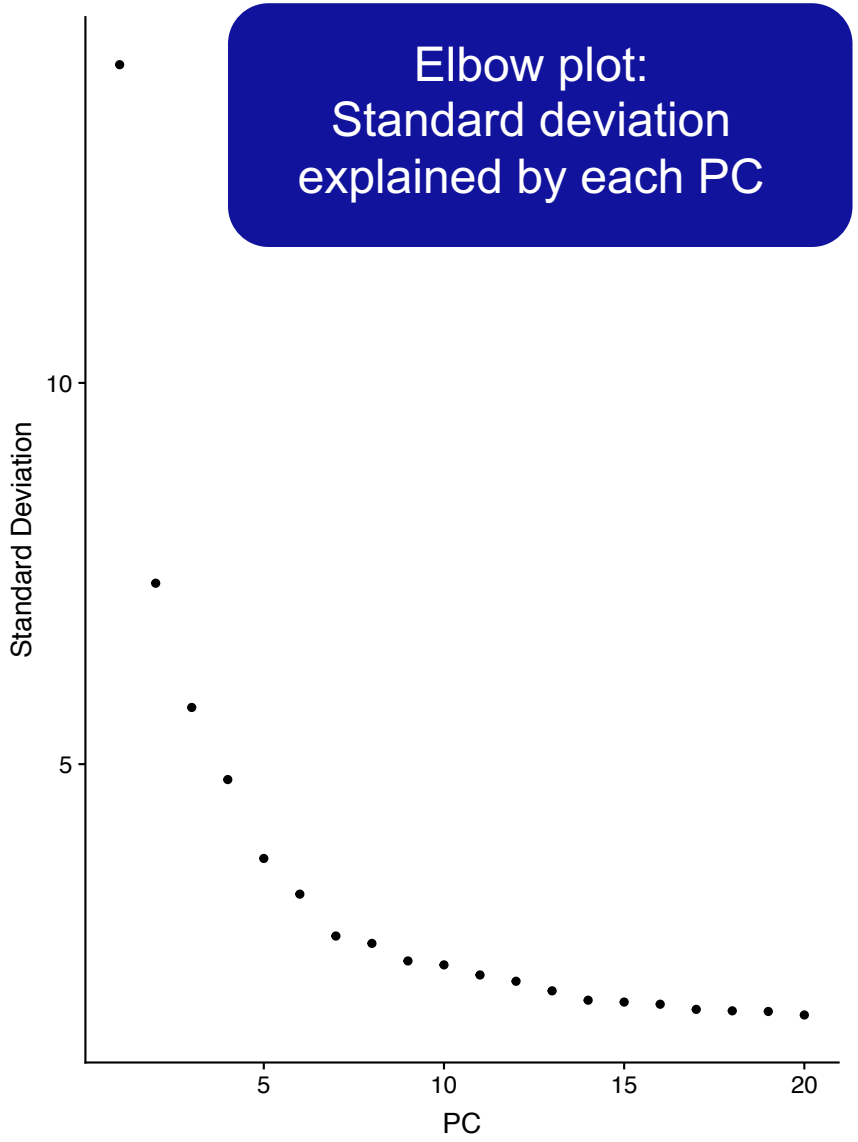
```
PClist_1 <- names(sort(Loadings(object=scrna, reduction="pca")[,1], decreasing=TRUE));
```

Selecting Principal Components

All downstream calculations are done on PCs, not raw data

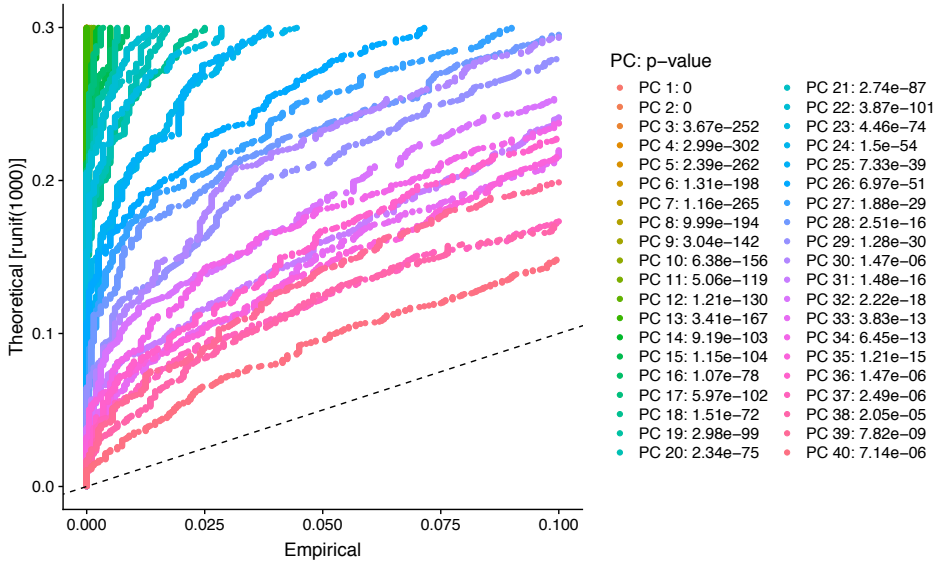
- Retain and plot key information about each principal component (PC):
  - Percentage of standard deviation explained
  - P-value (obtained from bootstrapping “Jackstraw”)
  - Plot gene expression heatmaps for each of the top ~12 principal components
- Choose Principal Components:
  - Purpose: choose relative importance of minor expression signatures
  - Discontinuity in elbow plot
  - All PCs that explain >= x% of SD (e.g. 2%)
  - P-value from JackStraw analysis < $1x10^{-n}$ (e.g. $1x10^{-100}$)
  - Clarity of PC heatmaps
- Number of PCs:
  - Single samples: 5-10
  - Multiple samples: 20-50
  - Cellranger default: 10
  - Partek default: 50

Elbow plot:
Standard deviation
explained by each PC



Selecting PCs: Jackstraw analysis

- Randomly permute data numerous times
- Recalculate PCs of randomized data
- Compare “real” PCs to “random” PCs to derive significance




```
# NB: jackstraw analysis is slow
scrna <- JackStraw(object = scrna, num.replicate = 100, dims=N) # specify N:
30, 50, etc
scrna <- ScoreJackStraw(object = scrna, dims = 1:50)
js <- JackStrawPlot(object = scrna, dims = 1:20) # make plot shown above
pc.pval <- scrna@reductions$pca@jackstraw@overall.p.values # get overall
pvalues for each PC
```

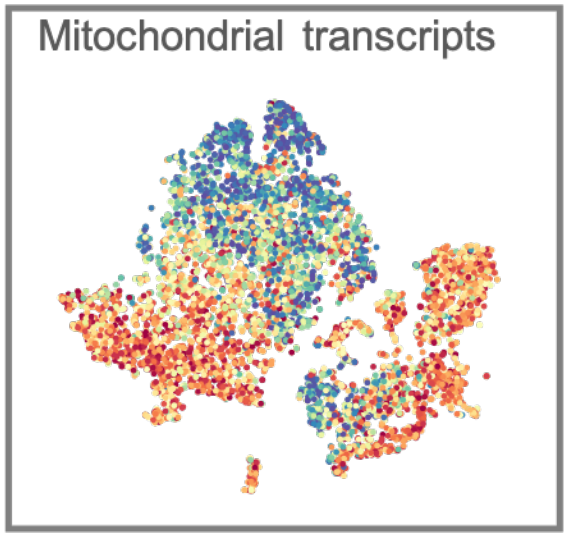
Chung and Storey. Statistical significance of variables driving systematic variation in high-dimensional data. Bioinformatics 2015

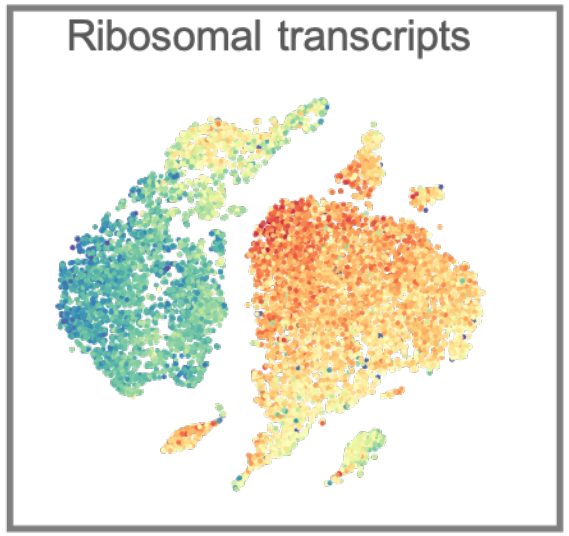
Plotting using t-SNE/UMAP

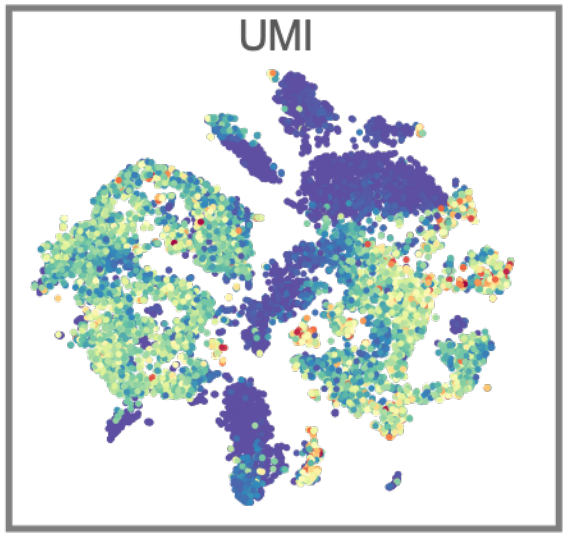
```
n = 10; # choose number of dimensions - experiment-specific!
scrna <- RunUMAP(object = scrna, reduction = "pca", dims = 1:n) # calculate
UMAP
scrna <- RunTSNE(object = scrna, reduction = "pca", dims = 1:n) # calculate
tSNE

# make some plots:
DimPlot(object = scrna, reduction = "tsne", group.by = ”Batch", pt.size=0.1) #
color by Batch
FeaturePlot(object = scrna, features = c("nCount_RNA"), reduction=“tsne”) #
plot one or more genes or variable on t-SNE
```

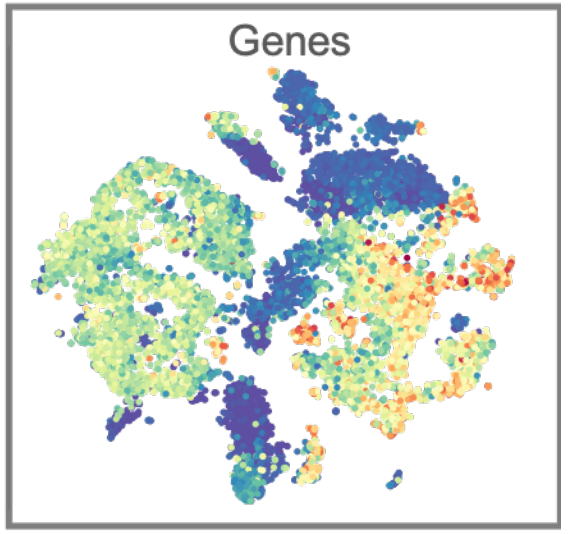
“How to use t-SNE effectively” https://distill.pub/2016/misread-tsne/

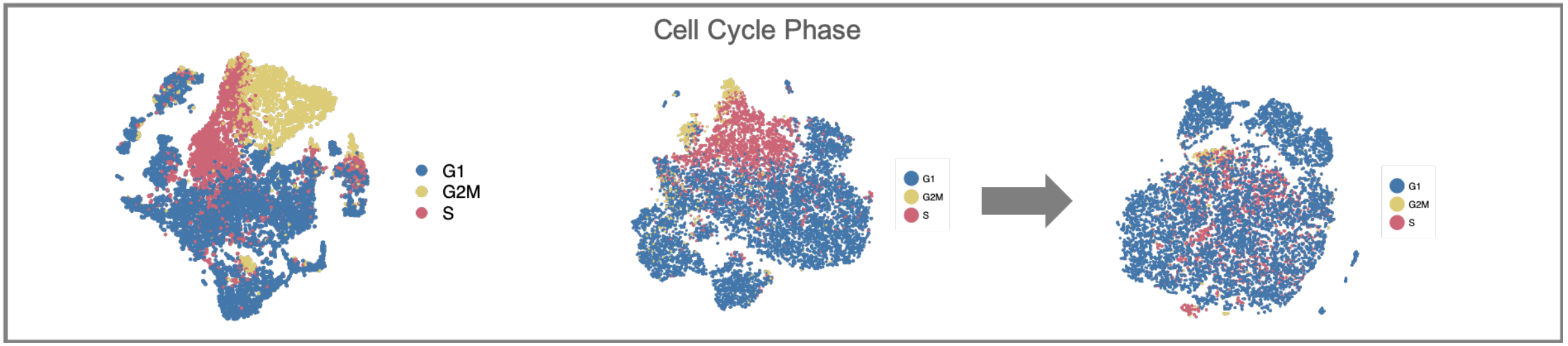
Interpreting the t-SNE/UMAP, Part I: Potentially misleading sources of variation





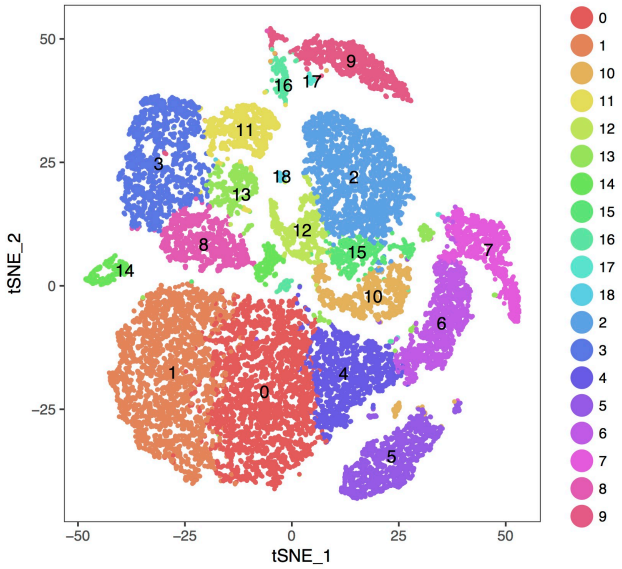


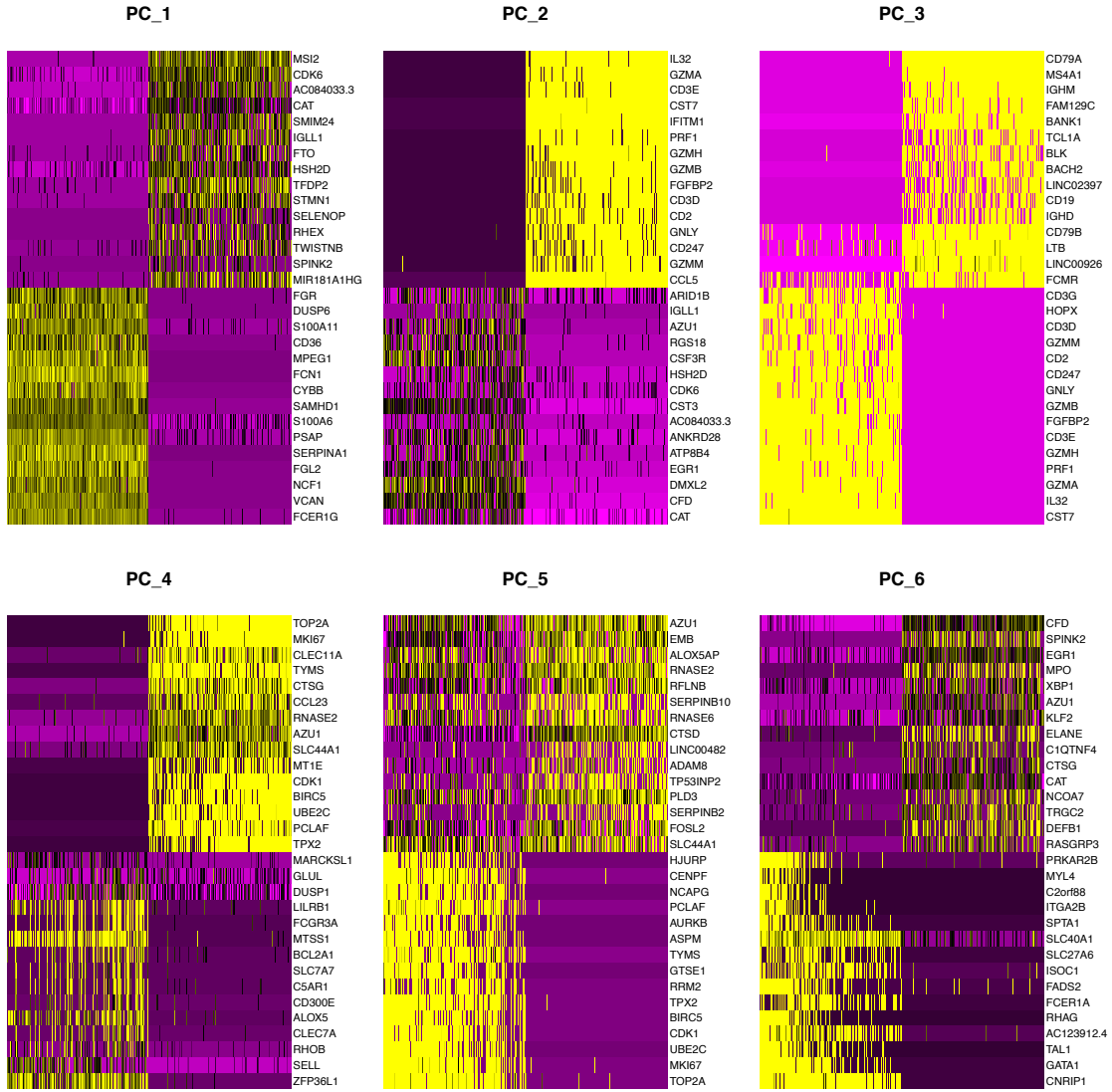




Interpreting the t-SNE/UMAP, Part II: Systematic analysis of variation

- What is driving the t-SNE/UMAP layout?
- Find genes that vary:
  - Principal components
  - Individual cluster-specific genes
- Examine across clusters/t-SNE/UMAP



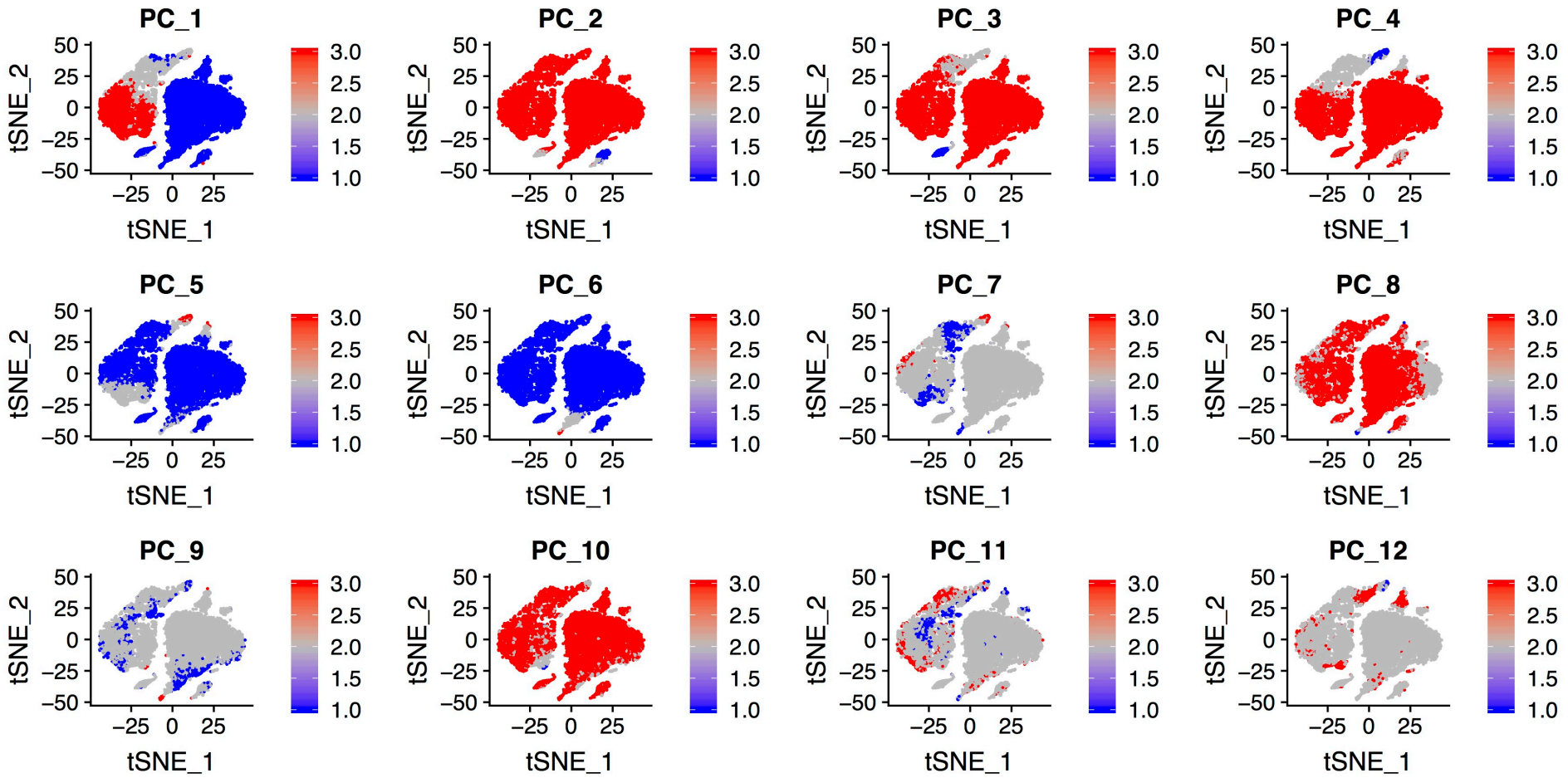



```
DimHeatmap(object = scrna, dims = 1, cells = 500, balanced = TRUE)
```

Part II, cont'd: Visualizing sources of variation

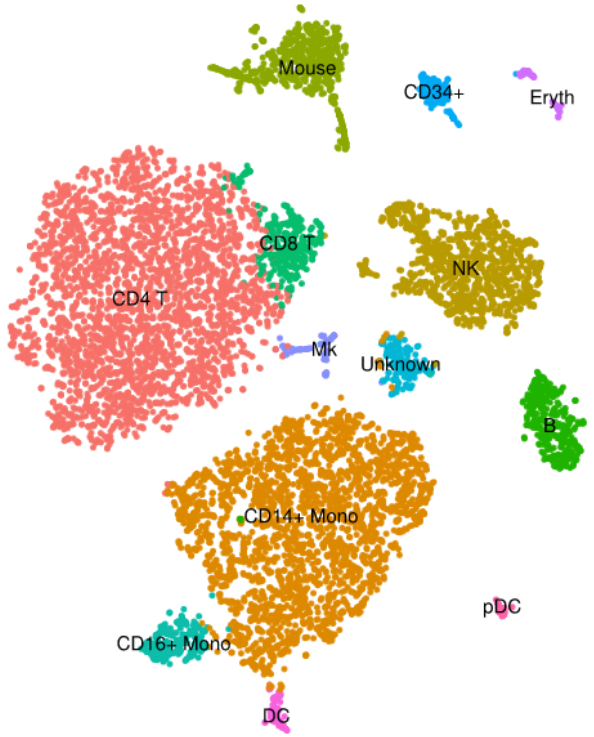
Small but distinct cell clusters may contribute heavily to the layout (may need to be removed)

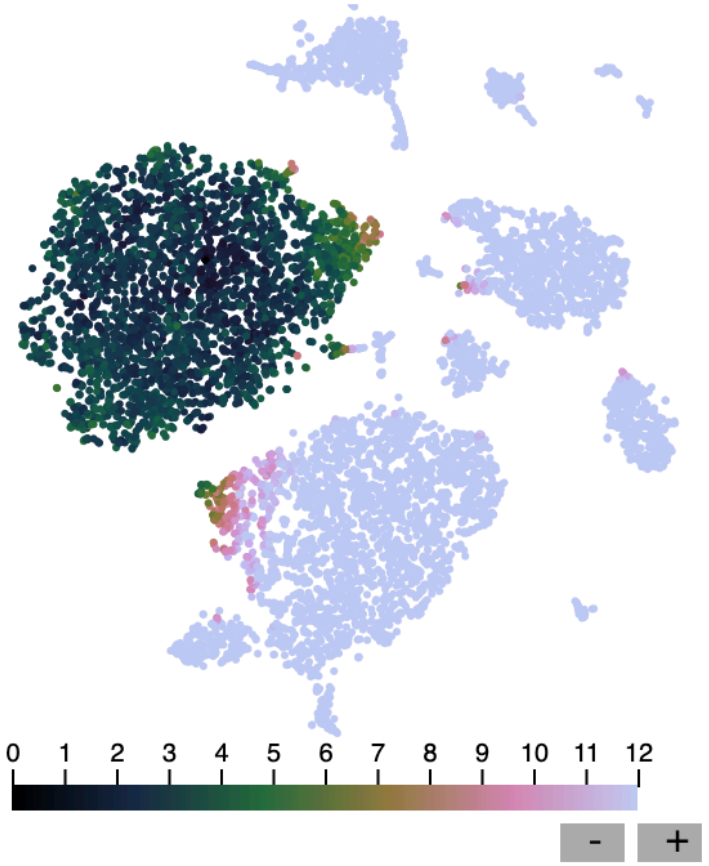
PC #1 captures the biggest source of variation

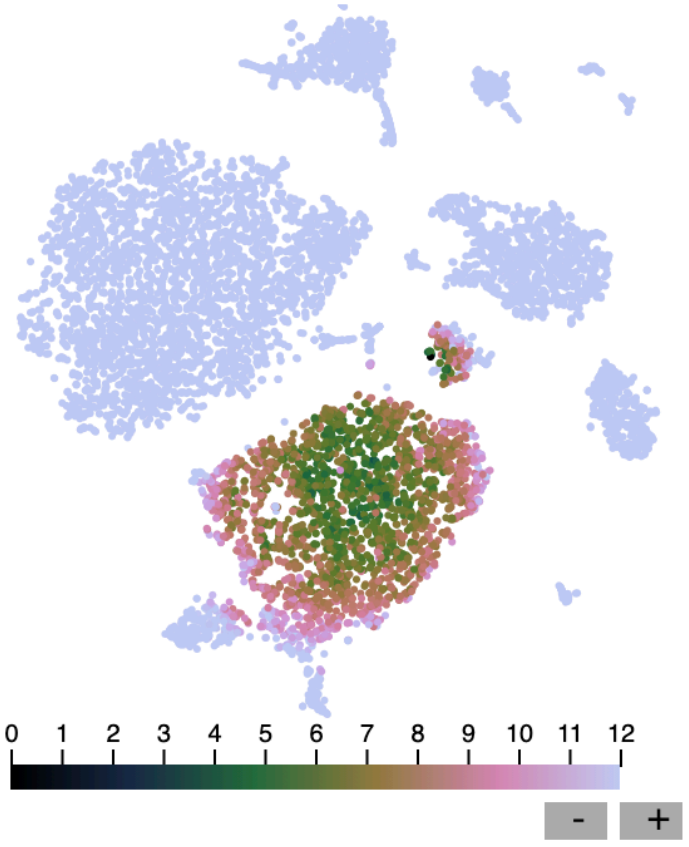


Interpreting the t-SNE/UMAP: Evaluating layout using Sleepwalk

How well does the 2-D layout reflect distances in the original high-dimensional space? https://anders-biostat.github.io/sleepwalk/





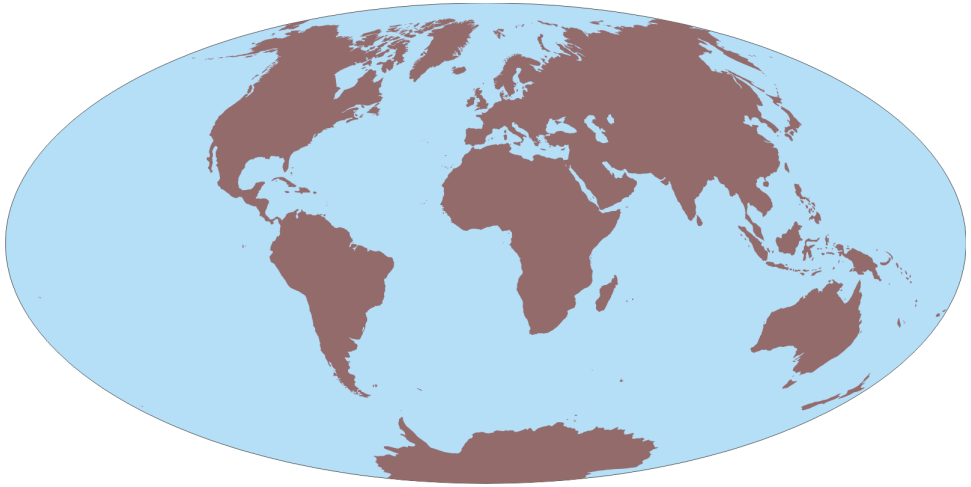


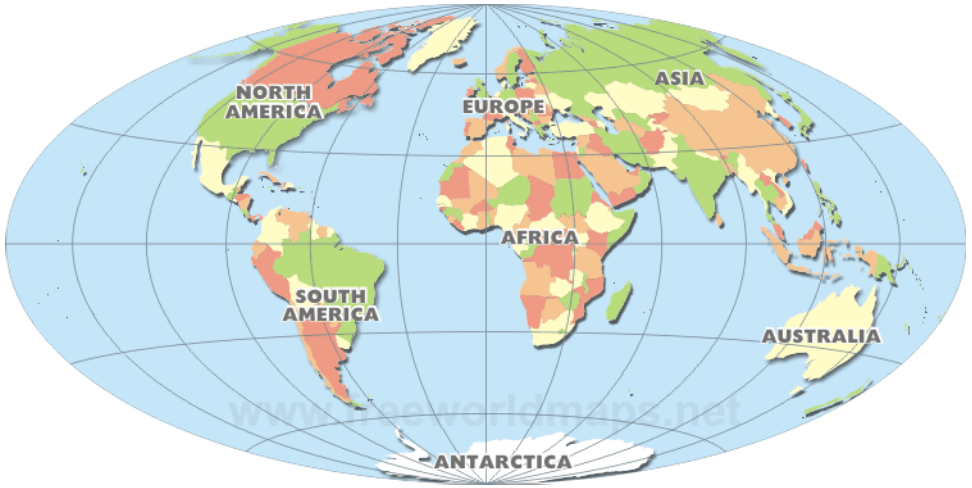
Post-PCA analysis of the gene x cell matrix

- Compute t-SNE and UMAP layouts on *n* Principal Components (NB: not raw data)
- **Clustering**
  - A tool for finding patterns in the data
  - Graph-based (unsupervised, must specify resolution (0.7))
  - Alternative: k-means (supervised, must specify k)
- Characterizing Clusters in terms of individual genes
  - Differentially expressed genes (numerous methods)
  - PC-perspective
    - Choose genes that contribute heavily to top principal components
    - Plot heatmaps of these genes in each cluster
    - Independent of clustering
    - Shows relationships of clusters to each other
- Cell lineage inference

2-D layout vs. Clustering

- Terminology: “clusters” in a tSNE or UMAP are not clusters
- tSNE and UMAP reflects natural organization of data by approximating high-dimensional relationships in low-dimensional space
- Clustering imposes structure by assigning cells to non-overlapping groups based on relative expression similarity





Clustering [Cells]

- Clustering helps organize and identify patterns in data.
- There is no “correct” or “perfect” clustering of any data set.
- Corollary: Even the best clustering may be misleading (aka “wrong”).
- Don’t take clusters too seriously – they don’t prove anything.

Common methods:

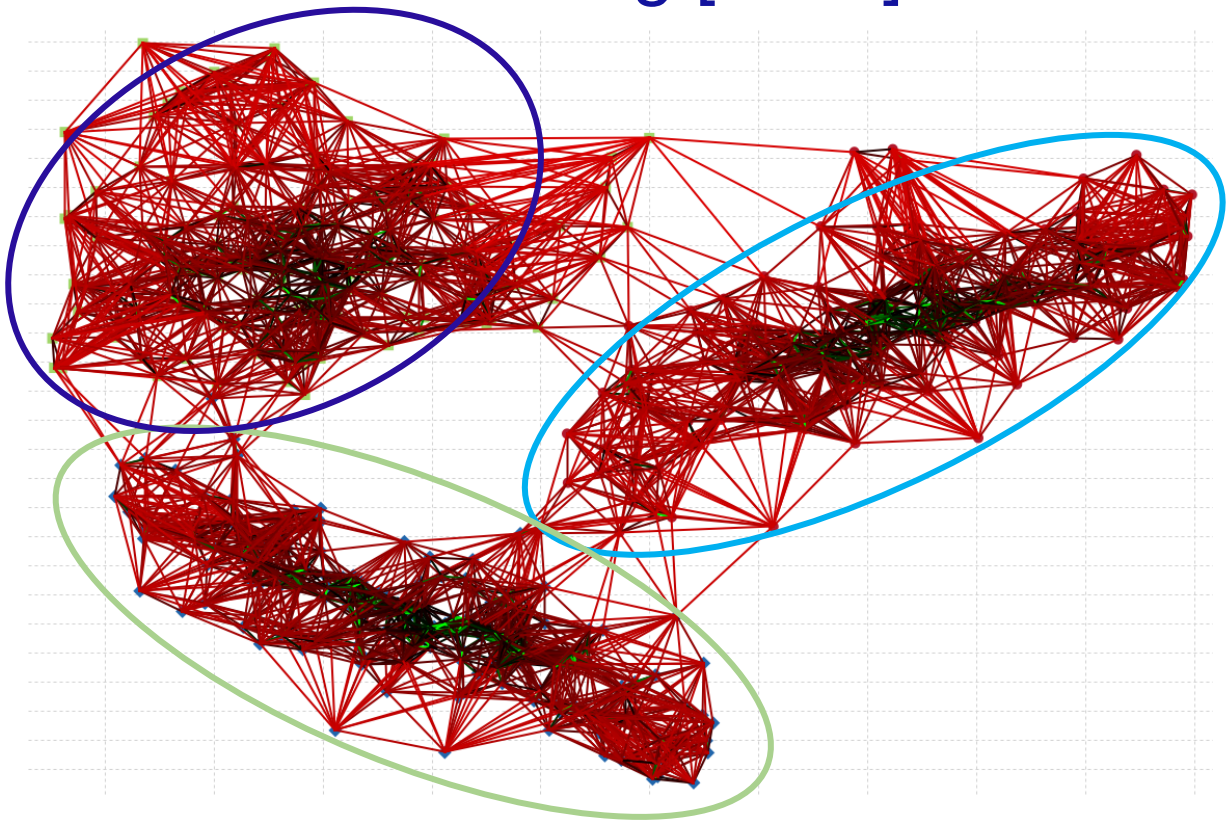
- Graph-based: Unsupervised, but “adjustable” using ”resolution” parameter
  - Calculates k-nearest neighbors for each cell using Euclidian distance in PCA-space; constructs shared nearest-neighbor (SNN) graph; optimize graph modularity (Waltman and van Eck algorithm)
- K-means: Supervised: specify number of clusters. (available in cellranger, not Seurat)

```
nPC = 20; # specify number of PCs to use
cluster.res = 0.7; # specify resolution of clustering.
scrna <- FindNeighbors(object=scrna, dims=1:nPC);
scrna <- FindClusters(object=scrna, resolution=cluster.res);
scrna <- StashIdent(object = scrna, save.name = sprintf("ClusterNames_%.1f_%dPC",
cluster.res, nPC)) # save cluster names in a new identity (ClusterNames_0.7_20 in this
example) if desired
```

Clustering [Cells]

Step 1: Build KNN graph using Euclidian distance in PCA-space

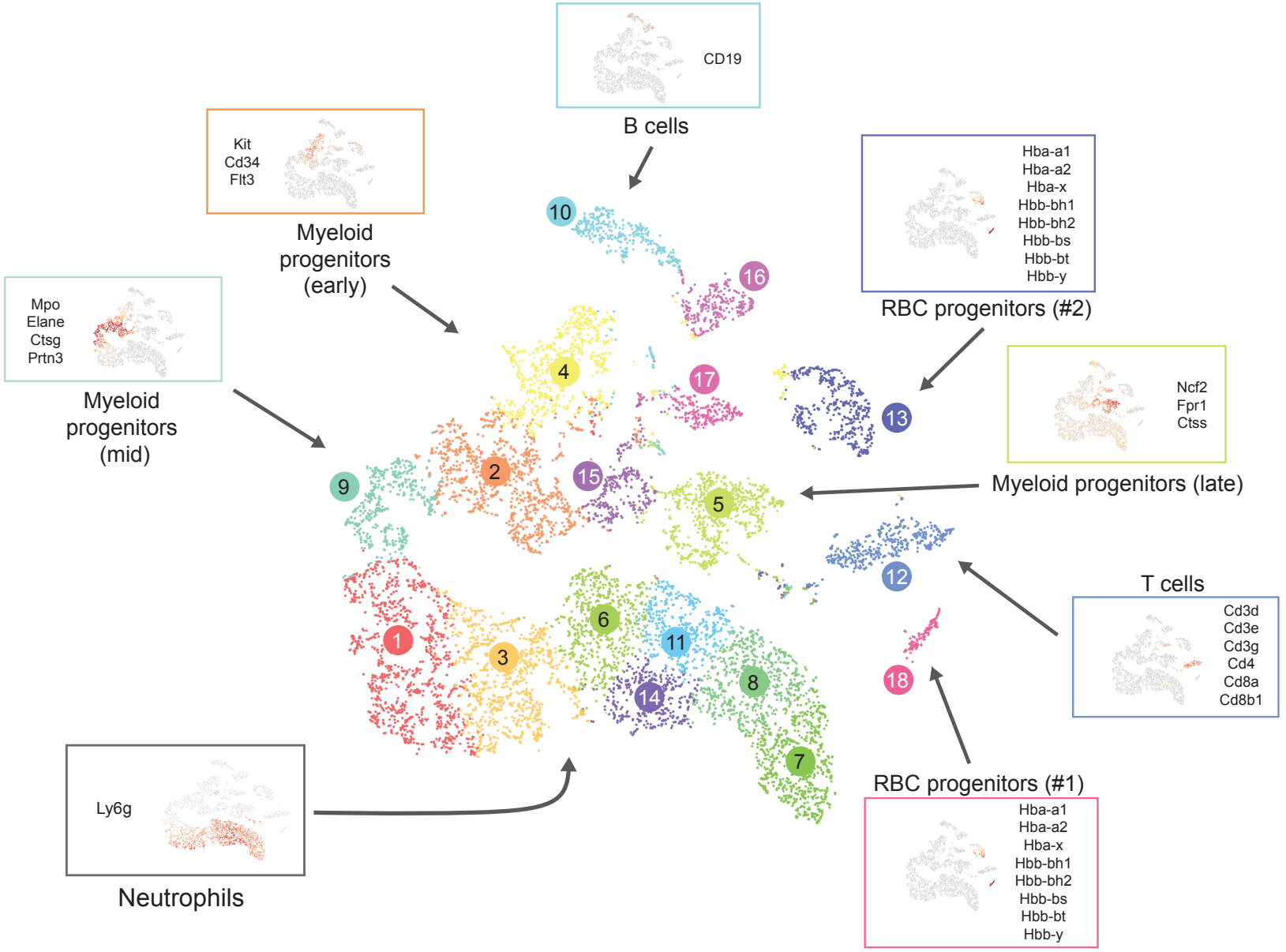
Step 2: Find clusters, or cliques, or modules using Louvain modularity optimization algorithm (alternatives available)



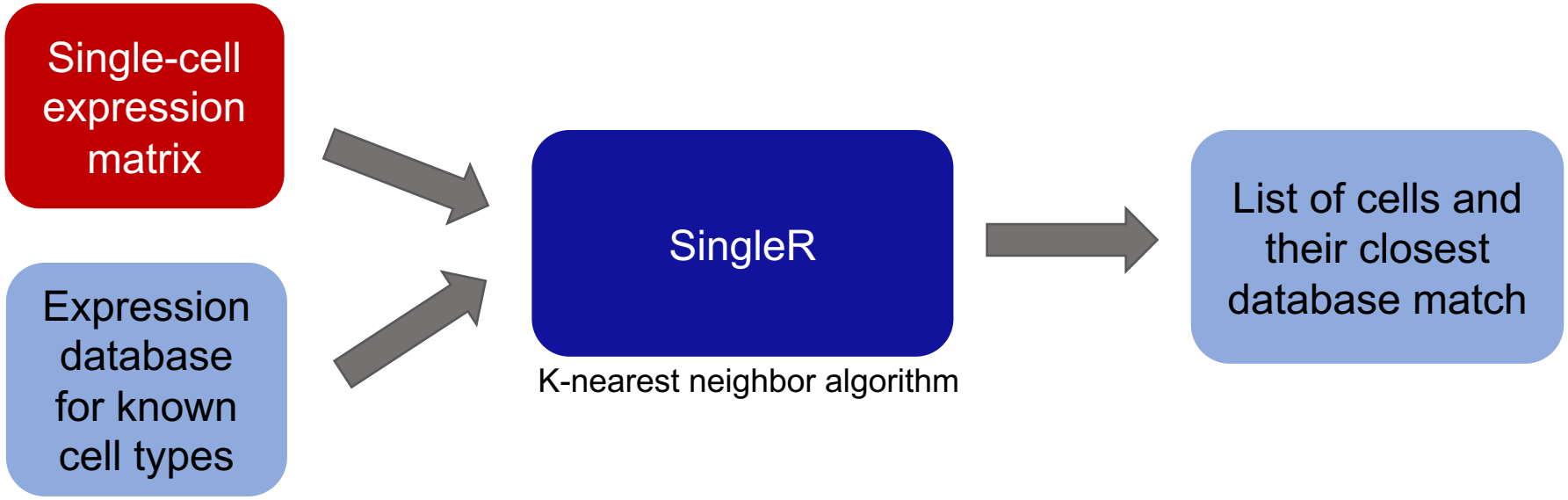
```
nPC = 20; # specify number of PCs to use
cluster.res = 0.7; # specify resolution of clustering.
scrna <- FindNeighbors(object=scrna, dims=1:nPC);
scrna <- FindClusters(object=scrna, resolution=cluster.res);
scrna <- StashIdent(object = scrna, save.name = sprintf("ClusterNames_%.1f_%dPC",
cluster.res, nPC)) # save cluster names in a new identity (ClusterNames_0.7_20 in this
example) if desired
```

Characterizing clusters using marker genes

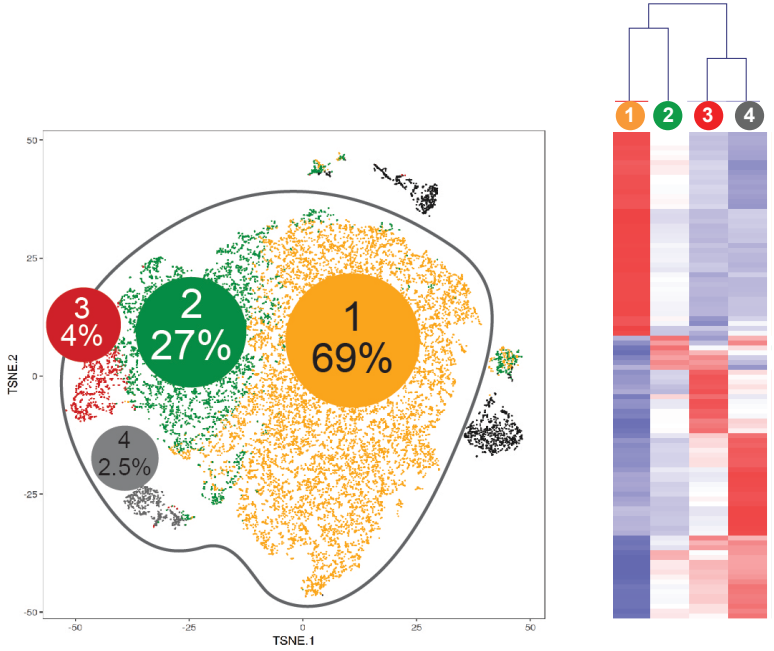
Cells colored by graph-based cluster



SingleR: Marker-free, Cluster-free Cell Lineage Inference



Characterizing clusters using differential gene expression

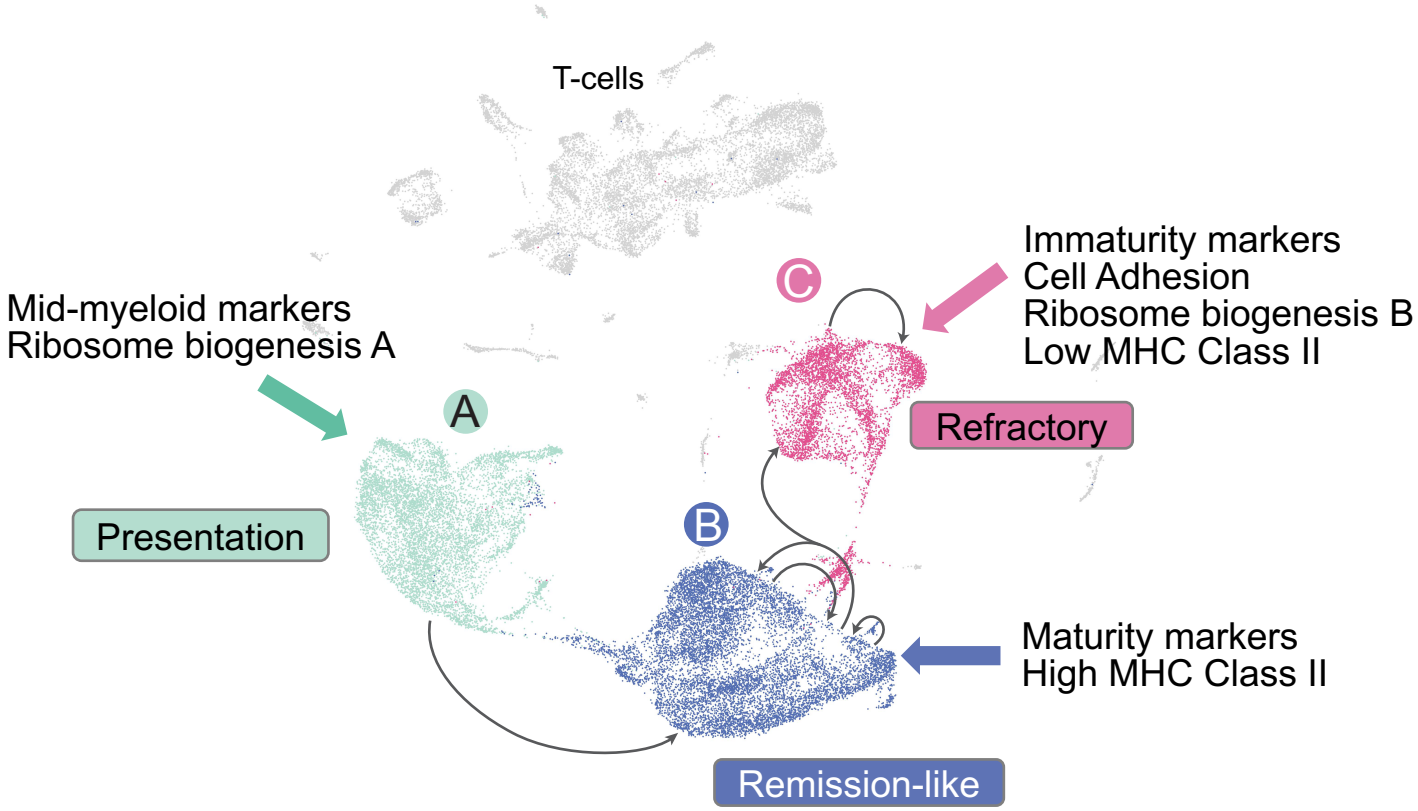


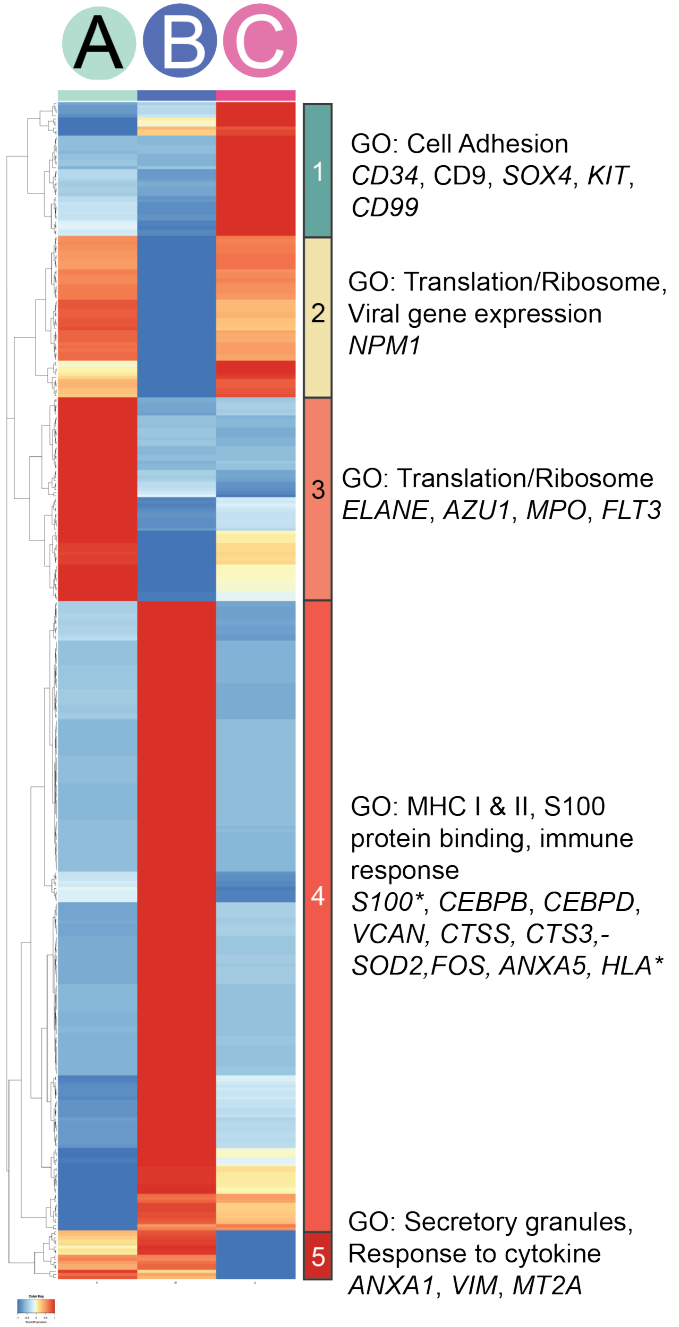
- Data has zero-inflated negative binomial distribution (lots of zeros, overdispersed) so can't use bulk methods
- Default in Seurat: Wilcoxon rank-sum test
- Nonparametric version of t-test
- For two clusters (A and B), and one gene, rank each cell in each cluster according to expression
- Determine whether sum-of-ranks for cluster A is significantly different than sum-of-ranks for cluster B
- Clear explanation of Wilcoxon rank-sum test: http://statweb.stanford.edu/~susan/courses/s141/hononpara.pdf
- Numerous other tests in Seurat and other packages

Characterizing clusters using differential gene expression

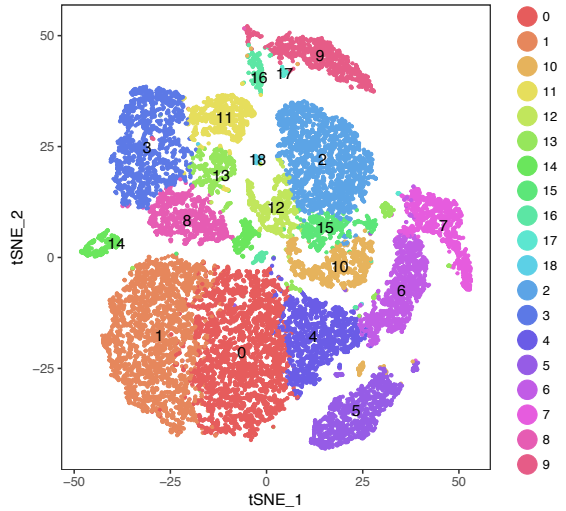
```
DEGs <- FindAllMarkers(object=scrna); # Compare each clusters to all
other cells. output is a matrix.
de.markers <- FindMarkers(scrna, ident.1 = ”1", ident.2 = ”2") #
compare identity 1 to identity 2
# NB: May first need to set default identities, e.g.:
Idents(object = scrna) <- ”seurat_clusters”; # sets the default
identity to Seurat_clusters

# Do the DEGs make sense? Plot them
FeaturePlot(object = scrna, features = ‘CD34’)
# Prettier version:
FeaturePlot(object = scrna, features = genesToPlot, cols =
c("gray","red"), ncol=2, reduction = "umap") +
theme(axis.title.x=element_blank(),axis.title.y=element_blank(),axis.
text.x=element_blank(),axis.text.y=element_blank(),axis.ticks.x=eleme
nt_blank(),axis.ticks.y=element_blank())
```

Example of differential expression on user-defined clusters

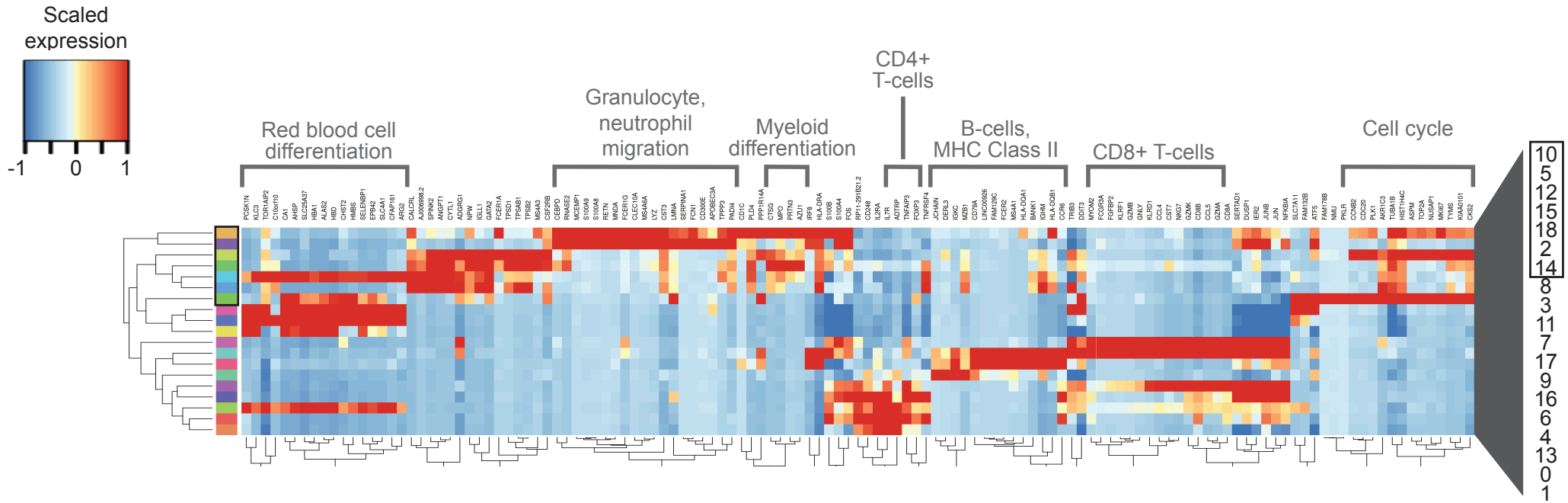




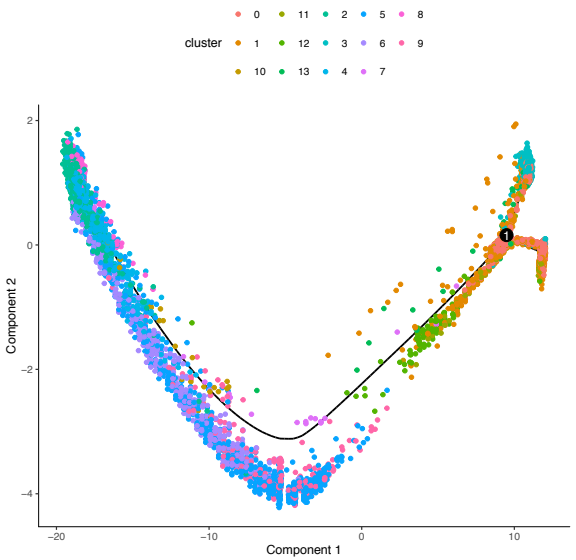
Characterizing clusters using principle components



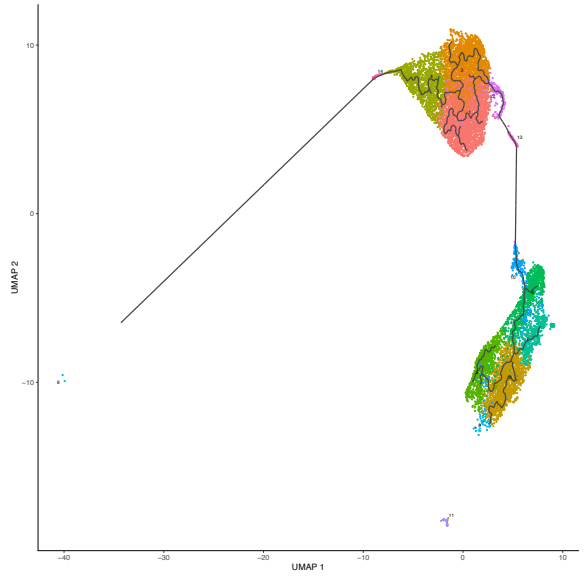
- Cluster-specific DEGs are problematic: require “perfect” clusters, oversimplify, can miss signal, too stringent for the data.
- Instead, visualize principle components across clusters to view relationships among clusters
- Choose top x (e.g. 10) most highly-weighted genes in the top y (e.g. 25) principal components
- Calculate average expression of each gene in each cluster
- Use hierarchical clustering to cluster the clusters based on these genes



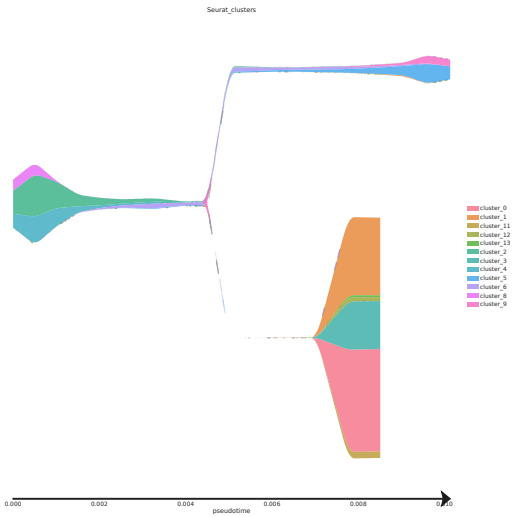
Characterizing gene expression changes with respect to pseudotime



Monocle 2



Monocle 3
(Slingshot is similar)

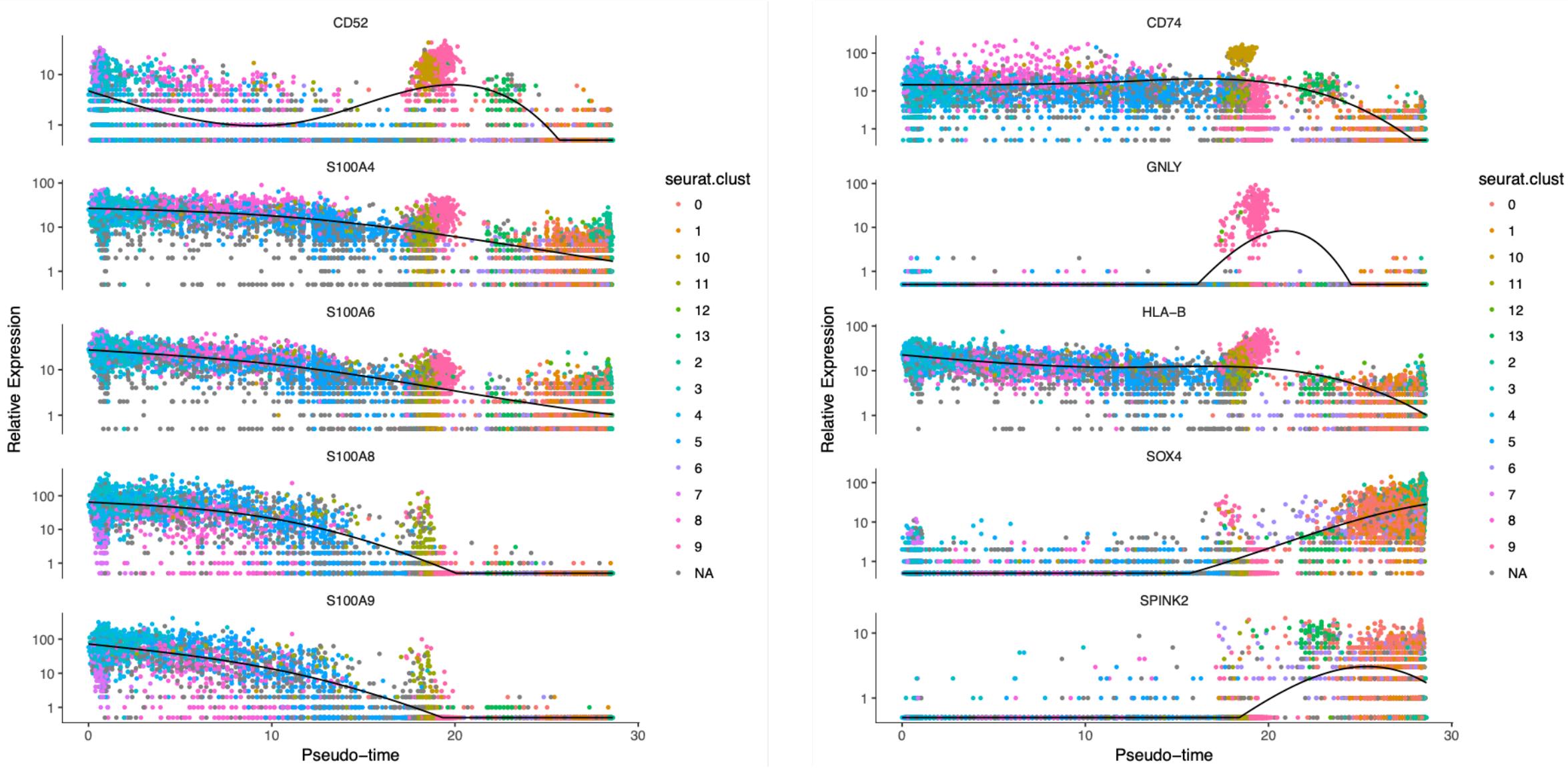


STREAM

Best to benchmark these methods with data from cells with known developmental relationships, e.g. hematopoiesis

Characterizing gene expression changes with respect to pseudotime

(Monocle)

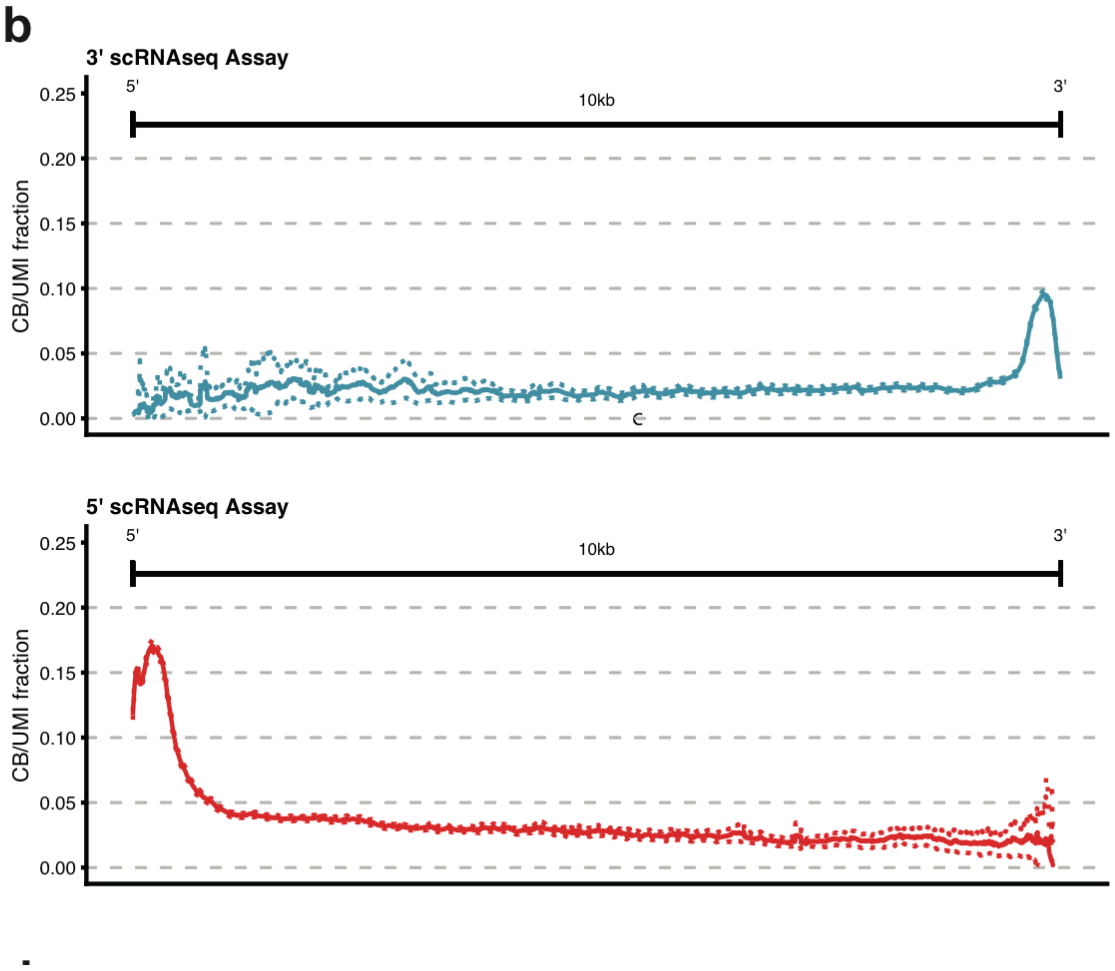


Other questions to consider when comparing two samples/clusters

- Pairwise differential gene expression
- What fraction of cells in each sample express a given gene?
- Of the cells in each sample that express a given gene, does the mean expression in those cells differ?
- Does the distribution of cell types differ between samples?
- Do the samples exhibit cell-type-specific differential gene expression?

Finding mutations:
Whole-transcript coverage from end-biased scRNA-seq

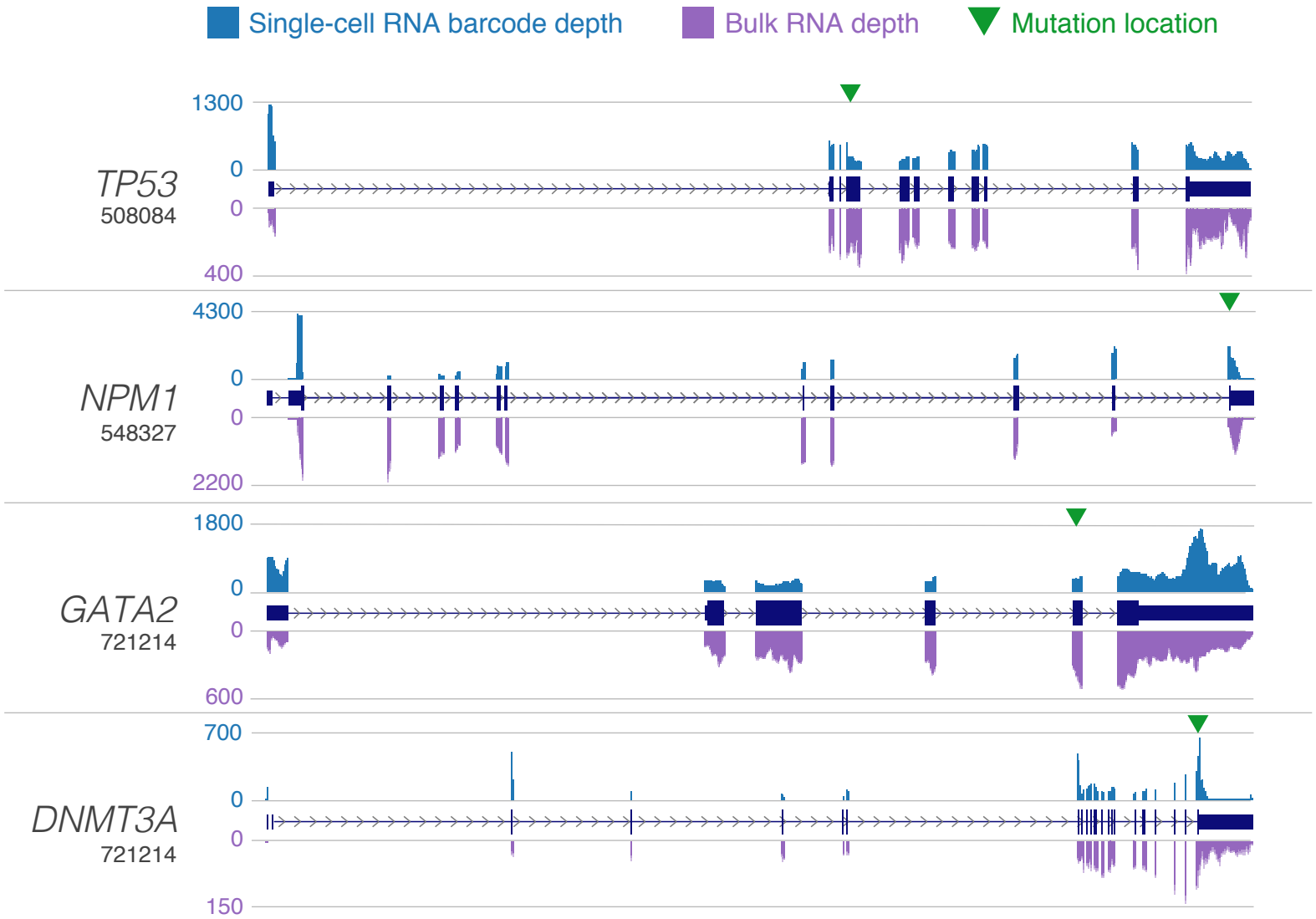
Fraction of UMIs with coverage at indicated position



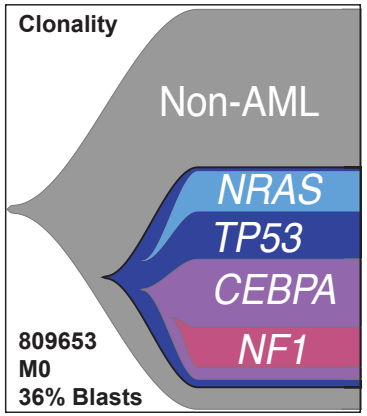
Distance from TTS

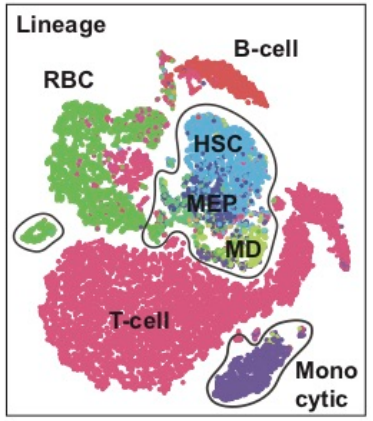
Distance from TSS

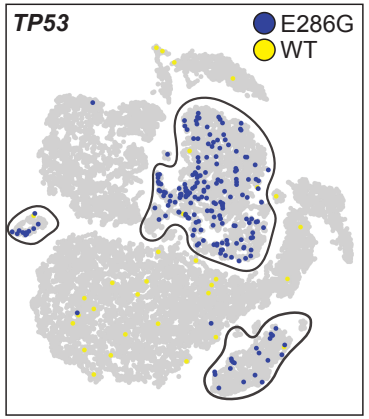
scRNA-seq recapitulates bulk transcript coverage



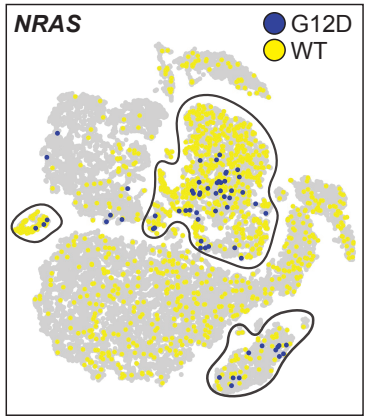
Mutations clarify scRNA-seq data interpretation by marking tumor cell clusters

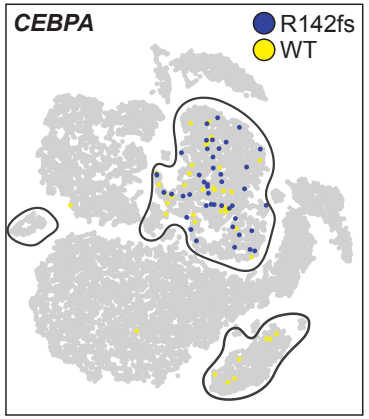




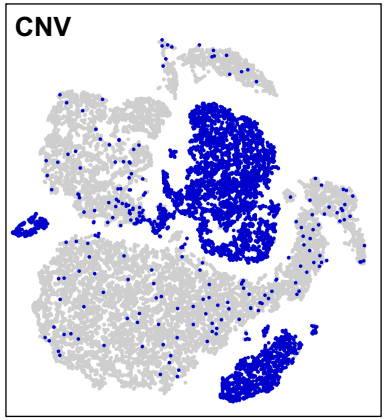


Founding clone
=> AML clusters*





Independent subclones, overlapping signatures



CNVs mark AML clusters

- vartrix
- cb_sniffer
- CONICSmat (CNV only)
- HoneyBADGER (CNV, LOH)

*AML clusters: significantly enriched for somatic mutations (Fisher exact test)

Further Reading

- http://bioconductor.org/books/release/OSCA/
- https://broadinstitute.github.io/2020_scWorkshop/introduction-rbioconductor.html