scRNAseq\_Hana\_07\_14\_2023\_seurat

2023-07-14

## R Markdown

# Loading libraries and setting location paths

# remove.packages('Matrix')  
# install.packages("Matrix", repos="http://R-Forge.R-project.org")  
library("Matrix")  
library(Seurat)

## Warning: package 'Seurat' was built under R version 4.2.3

## Attaching SeuratObject

# packageVersion("Seurat")  
library(dplyr)

##   
## Attaching package: 'dplyr'

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

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

library(ggplot2)  
setwd("Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/")  
  
rm(list = ls())  
  
fig\_path <- 'Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/'  
data\_path <- 'Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/'

Below we’ve included a few convenience functions for saving images and reading/writing your Seurat object to disk. When you’re working with larger datasets, it’s usually a good idea to save your progress after computationally intensive steps so you can back track if you wish to do so.

# Reading in data

After reading in the data we’ll perform basic filtering a on our expression matrix to remove low quality cells and uninformative genes. The parameter “min\_genes” will keep cells that have at least 300 genes, and similarly, “min\_cells” will keep genes that are expressed in at least 5 cells. Note: Seurat version 4.1 includes a convenience function to read Parse data from the DGE folder. If you would like to use this function, please skip the code block below and see the section “Reading in data with Seurat >= 4.1”

DGE\_folder <- "Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/S\_combined/all-well/DGE\_filtered/"  
  
# split-pipe versions older than 1.1.0 used "DGE.mtx"  
mat <- readMM(paste0(DGE\_folder, "DGE.mtx"))  
  
cell\_meta <- read.delim(paste0(DGE\_folder, "cell\_metadata.csv"),  
 stringsAsFactor = FALSE, sep = ",")  
genes <- read.delim(paste0(DGE\_folder, "all\_genes.csv"),  
 stringsAsFactor = FALSE, sep = ",")  
  
cell\_meta$bc\_wells <- make.unique(cell\_meta$bc\_wells, sep = "\_dup")  
rownames(cell\_meta) <- cell\_meta$bc\_wells  
genes$gene\_name <- make.unique(genes$gene\_name, sep = "\_dup")  
  
# Setting column and rownames to expression matrix  
colnames(mat) <- genes$gene\_name  
rownames(mat) <- rownames(cell\_meta)  
mat\_t <- t(mat)  
  
# Remove empty rownames, if they exist  
mat\_t <- mat\_t[(rownames(mat\_t) != ""),]  
  
# Seurat version 5 or greater uses "min.features" instead of "min.genes"  
pbmc <- CreateSeuratObject(mat\_t, min.features = 100, min.cells = 2, meta.data = cell\_meta)

## Warning: Feature names cannot have underscores ('\_'), replacing with dashes  
## ('-')

When we create our Seurat object the plate well numbers (column names in the expression matrix) from the experiment will automatically be assigned to the cell identity slot. In other words, the program assumes this how we want to initially classify our cells. In general, we would like to avoid this behavior so there isn’t a bias towards a particular cell class when removing outliers.

# Setting our initial cell class to a single type, this will changer after clustering.   
pbmc@meta.data$orig.ident <- factor(rep("pbmc", nrow(pbmc@meta.data)))  
Idents(pbmc) <- pbmc@meta.data$orig.ident  
  
SaveObject(pbmc, "seurat\_obj\_before\_QC")  
pbmc <- ReadObject("seurat\_obj\_before\_QC")

# Cell quality control

In this step we’ll perform cell quality control to prevent outlier cells from influencing downstream analyses. Cells that have unusually high transcript or gene counts are very likely to be multiplets, which is a term for two or more cells that have identical barcodes. Conversely, cells that have very low transcript or genes counts are a consequence of barcoding cells with damage membranes, or barcoding ambient RNA.

Filtering outliers can be accomplished by generating a violin plot for each cell feature and manually selecting the threshold we wish to remove cells. We’ll also add another important cell feature that shows the percentage of mitochondrial genes expressed in each cell. Cells with high mitochondrial gene percentages should be removed, as they are likely to have lost cytoplasmic RNA from lysis or may have increased apoptosis (Luecken and Theis 2019)

pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")  
plot <- VlnPlot(pbmc, pt.size = 0.10,  
features = c("nFeature\_RNA", "nCount\_RNA", "percent.mt"), ncol = 3)  
SaveFigure(plot, "vln\_QC", width = 12, height = 6)

## png   
## 2

plot1 <- FeatureScatter(pbmc, feature1 = "nCount\_RNA", feature2 = "percent.mt")  
plot2 <- FeatureScatter(pbmc, feature1 = "nCount\_RNA", feature2 = "nFeature\_RNA")  
SaveFigure((plot1 + plot2),"scatter\_QC", width = 12, height = 6, res = 200)

## png   
## 2

# Perform the filtering  
pbmc <- subset(pbmc, subset = nFeature\_RNA < 5000 & nCount\_RNA < 20000 & percent.mt < 15)

# Normalizing the data

After removing unwanted cells from the dataset, the next step is to normalize the data. By default, we employ a global-scaling normalization method LogNormalize that normalizes the feature expression measurements for each cell by the total expression, multiplies this by a scale factor (10,000 by default), and log-transforms the result. Normalized values are stored in pbmc[[“RNA”]]@data.

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

# Identification of highly variable features

We next calculate a subset of features that exhibit high cell-to-cell variation in the dataset (i.e, they are highly expressed in some cells, and lowly expressed in others). We and others have found that focusing on these genes in downstream analysis helps to highlight biological signal in single-cell datasets. Our procedure in Seurat3 is described in detail here, and improves on previous versions by directly modeling the mean-variance relationship inherent in single-cell data, and is implemented in the FindVariableFeatures function. By default, we return 2,000 features per dataset. These will be used in downstream analysis, like PCA.

pbmc <- FindVariableFeatures(pbmc, selection.method = "vst", nfeatures = 2000)  
# Identify the 10 most highly variable genes  
top10 <- head(VariableFeatures(pbmc), 10)  
# plot variable features with and without labels  
plot1 <- VariableFeaturePlot(pbmc)  
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)

## When using repel, set xnudge and ynudge to 0 for optimal results

SaveFigure((plot1 + plot2), "var\_features", width = 12, height = 6)

## Warning: Transformation introduced infinite values in continuous x-axis  
## Transformation introduced infinite values in continuous x-axis

## png   
## 2

# Scaling the data

Next, we apply a linear transformation (‘scaling’) that is a standard pre-processing step prior to dimensional reduction techniques like PCA. The ScaleData function:

Shifts the expression of each gene, so that the mean expression across cells is 0 Scales the expression of each gene, so that the variance across cells is 1 This step gives equal weight in downstream analyses, so that highly-expressed genes do not dominate The results of this are stored in pbmc[[“RNA”]]@scale.data

Warning! Generating a scaled data matrix can require a significant amount of RAM for large datasets and potentially crash your session. We therefore chose to use the default setting which only uses the variable gene set in the previous step. It’s important to note that scaling the variable gene subset does not affect clustering; it only limits the number of genes available for plotting in the DoHeatMap function.

pbmc <- ScaleData(pbmc)

## Centering and scaling data matrix

# Perform linear dimensional reduction

Next we perform PCA on the scaled data. By default, only the previously determined variable features are used as input, but can be defined using features argument if you wish to choose a different subset. Seurat provides several useful ways of visualizing both cells and features that define the PCA, including VizDimReduction, DimPlot, and DimHeatmap

pbmc <- RunPCA(pbmc)

## PC\_ 1   
## Positive: SORBS2, FBN2, KCNH7, SPHKAP, ENSG00000289950, PLXNA4, TECRL, EPHA7, MYH7, ENTREP1   
## FRMD4B, TMEM163, AFF3, DGKI, CORIN, ERBB4, INPP4B, SORCS1, SLCO3A1, SIPA1L1   
## DLG2, PLXNA2, ROR1, B4GALNT3, ANKRD1, COL13A1, ENSG00000286071, PLCL1, MAST4, ADCY5   
## Negative: COL1A1, COL1A2, DLK1, COL6A3, ADAM12, PDE1A, COL3A1, IGFBP3, IGF2, BNC2   
## LUM, DCN, TGFBI, PLAGL1, LINGO2, ZFPM2-AS1, ITGA11, PDE7B, CSMD3, COL6A2   
## MMP2, COL14A1, PIEZO2, LDB2, NR2F1-AS1, ADAMTS5, ADAMTS6, AHR, NRK, MFAP4   
## PC\_ 2   
## Positive: FBN2, VCAN, TECRL, SLIT2, SEMA3A, TMEM163, TMTC1, IGFBP5, ADCY5, ROR1   
## KDR, CORIN, LUM, ATP2B1, SIPA1L1, SPHKAP, GJA1, B4GALNT3, MYL3, CSMD3   
## DCN, PDE1A, ZFPM2-AS1, TGFBI, TRPC3, NR2F1-AS1, ENSG00000289950, MYH7, COL13A1, DLK1   
## Negative: KCNIP4, ENSG00000289309, KALRN, ADGRL3, EFNA5, MECOM, ST6GALNAC3, RGS6, SPON1, ENOX1   
## C7, MARCHF1, EPS8, LRRTM4, NKAIN3, TENM3, LTBP1, DOK6, SEMA6A, FLRT2   
## ENSG00000239268, PPP2R2B, FGFR2, VEGFC, ST6GALNAC5, LSAMP, ENSG00000289084, RELN, MAGI2, GDF6   
## PC\_ 3   
## Positive: MYH7B, ENTREP1, SPHKAP, TECRL, MYL3, NEAT1, SYNPO2L, SEMA3A, FBN2, KCNH7   
## PLN, ADAMTS1, CORIN, ENSG00000206549, GJA1, ENSG00000257322, ENSG00000230490, ENSG00000289950, DGKI, COL13A1   
## ADCY5, INPP4B, ENSG00000225689, TMEM163, CADPS, SLIT2, RCAN2, B4GALNT3, EPHA3, FHL2   
## Negative: TOP2A, MKI67, CENPF, CIT, ASPM, DIAPH3, ANLN, CENPE, ENSG00000284906, KIF11   
## KNL1, IQGAP3, KIF14, KIF4A, BUB1B, TPX2, NUSAP1, RRM2, KIF23, NDC80   
## PRC1, SMC4, BUB1, MIR924HG, KIF20B, NCAPG, SPC25, CKAP2L, ARHGAP11A, KIF18B   
## PC\_ 4   
## Positive: NAV3, SORCS3, MARCHF1, SPON1, FREM1, ADGRL3, SLIT3, THBS1, SH3RF2, RGS6   
## HS6ST2, CTNNA2, RSPO3, TNFRSF19, TENM2, ACTA2, ARHGAP6, GRIP1, ENSG00000253471, GRIK1   
## PROM1, DOK6, PRR16, GULP1, PRELID2, MAGI2, SV2C, SORBS1, ENSG00000289084, KDR   
## Negative: CDH19, VCAN, PDZRN4, GNG2, MXRA5, FBN1, LRRC4C, BGN, MAP2, MGP   
## ASPN, SYNPO, DCLK1, STMN2, RHOJ, DOK5, LINC00511, GPC3, LIMCH1, PRRX1   
## SERPINE2, COL9A3, THSD4, FN1, SEMA6D, SEMA3D, THY1, COL9A2, GMDS, PTPRT   
## PC\_ 5   
## Positive: ANKRD1, NALF1, RSPO3, HECW2, SPON1, TNFRSF19, MAMLD1, NGFR, P3H2, ARHGAP6   
## MYL7, HDAC2-AS2, CAMK1D, DAB1, PDLIM3, OXR1, GRIK1, GBE1, MIR4435-2HG, SORCS1   
## RYR3, NCKAP5, KALRN, MIR181A1HG, GRIP1, PRUNE2, NAV3, ROBO1, SPRY4-AS1, LYPD6B   
## Negative: H19, NRXN1, A2M, FBXO32, ABCA1, PDE3A, KCNJ3, IGF2, DLG2, MYOM2   
## PDE1C, MIR100HG, IL1RAPL1, LRRC7, ENSG00000189229, ADAMTSL1, CPNE5, TECRL, SEMA3A, ENSG00000289309   
## TIAM1, UNC5D, DLGAP1, BRINP3, SPOCK1, CORIN, DGKG, C7, SYNPO2, EFNA5

SaveObject(pbmc, "seurat\_obj\_after\_PCA")  
  
# pbmc <- ReadObject("seurat\_obj\_after\_PCA")  
  
# Examine and visualize PCA results a few different ways  
print(pbmc[["pca"]], dims = 1:5, nfeatures = 5)

## PC\_ 1   
## Positive: SORBS2, FBN2, KCNH7, SPHKAP, ENSG00000289950   
## Negative: COL1A1, COL1A2, DLK1, COL6A3, ADAM12   
## PC\_ 2   
## Positive: FBN2, VCAN, TECRL, SLIT2, SEMA3A   
## Negative: KCNIP4, ENSG00000289309, KALRN, ADGRL3, EFNA5   
## PC\_ 3   
## Positive: MYH7B, ENTREP1, SPHKAP, TECRL, MYL3   
## Negative: TOP2A, MKI67, CENPF, CIT, ASPM   
## PC\_ 4   
## Positive: NAV3, SORCS3, MARCHF1, SPON1, FREM1   
## Negative: CDH19, VCAN, PDZRN4, GNG2, MXRA5   
## PC\_ 5   
## Positive: ANKRD1, NALF1, RSPO3, HECW2, SPON1   
## Negative: H19, NRXN1, A2M, FBXO32, ABCA1

plot <- VizDimLoadings(pbmc, dims = 1:2, reduction = "pca")  
SaveFigure(plot, "viz\_PCA\_loadings", width = 10, height = 8)

## png   
## 2

plot <- DimPlot(pbmc, reduction = "pca", group.by = "orig.ident")  
SaveFigure(plot, "pc1\_2\_scatter", width = 8, height = 6)

## png   
## 2

In particular DimHeatmap allows for easy exploration of the primary sources of heterogeneity in a dataset, and can be useful when trying to decide which PCs to include for further downstream analyses. Both cells and features are ordered according to their PCA scores. Setting cells to a number plots the ‘extreme’ cells on both ends of the spectrum, which dramatically speeds plotting for large datasets. Though clearly a supervised analysis, we find this to be a valuable tool for exploring correlated feature sets.

# Image doesn't save as png unless fast = FALSE  
plot <- DimHeatmap(pbmc, dims = 1, cells = 500, balanced = TRUE, fast = FALSE)  
SaveFigure(plot, "dim\_heatmap1", width = 8, height = 6)

## png   
## 2

plot <- DimHeatmap(pbmc, dims = 1:15, cells = 500, balanced = TRUE, fast = FALSE)  
SaveFigure(plot, "dim\_heatmap1\_15", width = 12, height = 18)

## png   
## 2

# Determine the ‘dimensionality’ of the dataset

To overcome the extensive technical noise in any single feature for scRNA-seq data, Seurat clusters cells based on their PCA scores, with each PC essentially representing a ‘metafeature’ that combines information across a correlated feature set. The top principal components therefore represent a robust compression of the dataset. However, how many components should we choose to include? 10? 20? 100?

In Macosko et al, we implemented a resampling test inspired by the JackStraw procedure. We randomly permute a subset of the data (1% by default) and rerun PCA, constructing a ‘null distribution’ of feature scores, and repeat this procedure. We identify ‘significant’ PCs as those who have a strong enrichment of low p-value features.

# NOTE: This process can take a long time for big datasets, comment out for expediency. More  
# approximate techniques such as those implemented in ElbowPlot() can be used to reduce  
# computation time  
# pbmc <- JackStraw(pbmc, num.replicate = 100)  
# pbmc <- ScoreJackStraw(pbmc, dims = 1:20)

An alternative heuristic method generates an ‘Elbow plot’: a ranking of principle components based on the percentage of variance explained by each one (ElbowPlot function). In this example, we can observe an ‘elbow’ around PC9-10, suggesting that the majority of true signal is captured in the first 10 PCs.

For this particular dataset, we chose not to use the jackstraw plot for PC selection due to the amount of time it takes for 65k cells, but instead opted to use the elbow plot where we selected PCs 1-30.

plot <- ElbowPlot(pbmc,ndims = 50)  
SaveFigure(plot, "PC\_elbow\_plot", width = 8, height = 10)

## png   
## 2

Identifying the true dimensionality of a dataset – can be challenging/uncertain for the user. We therefore suggest these three approaches to consider. The first is more supervised, exploring PCs to determine relevant sources of heterogeneity, and could be used in conjunction with GSEA for example. The second implements a statistical test based on a random null model, but is time-consuming for large datasets, and may not return a clear PC cutoff. The third is a heuristic that is commonly used, and can be calculated instantly. In this example, all three approaches yielded similar results, but we might have been justified in choosing anything between PC 7-12 as a cutoff.

We chose 10 here, but encourage users to consider the following:

Dendritic cell and NK aficionados may recognize that genes strongly associated with PCs 12 and 13 define rare immune subsets (i.e. MZB1 is a marker for plasmacytoid DCs). However, these groups are so rare, they are difficult to distinguish from background noise for a dataset of this size without prior knowledge. We encourage users to repeat downstream analyses with a different number of PCs (10, 15, or even 50!). As you will observe, the results often do not differ dramatically. We advise users to err on the higher side when choosing this parameter. For example, performing downstream analyses with only 5 PCs does significantly and adversely affect results.

# Cluster the cells

Seurat v3 applies a graph-based clustering approach, building upon initial strategies in (Macosko et al). Importantly, the distance metric which drives the clustering analysis (based on previously identified PCs) remains the same. However, our approach to partitioning the cellular distance matrix into clusters has dramatically improved. Our approach was heavily inspired by recent manuscripts which applied graph-based clustering approaches to scRNA-seq data [SNN-Cliq, Xu and Su, Bioinformatics, 2015] and CyTOF data [PhenoGraph, Levine et al., Cell, 2015]. Briefly, these methods embed cells in a graph structure - for example a K-nearest neighbor (KNN) graph, with edges drawn between cells with similar feature expression patterns, and then attempt to partition this graph into highly interconnected ‘quasi-cliques’ or ‘communities’.

As in PhenoGraph, we first construct a KNN graph based on the euclidean distance in PCA space, and refine the edge weights between any two cells based on the shared overlap in their local neighborhoods (Jaccard similarity). This step is performed using the FindNeighbors. function, and takes as input the previously defined dimensionality of the dataset (first 10 PCs).

To cluster the cells, we next apply modularity optimization techniques such as the Louvain algorithm (default) or SLM [SLM, Blondel et al., Journal of Statistical Mechanics], to iteratively group cells together, with the goal of optimizing the standard modularity function. The FindClusters function implements this procedure, and contains a resolution parameter that sets the ‘granularity’ of the downstream clustering, with increased values leading to a greater number of clusters. We find that setting this parameter between 0.4-1.2 typically returns good results for single-cell datasets of around 3K cells. Optimal resolution often increases for larger datasets. The clusters can be found using the Idents function.

# install.packages("Matrix", reinstall = TRUE)  
# library(Matrix)  
# update.packages(ask = FALSE)  
pbmc <- FindNeighbors(pbmc, dims = 1:30)

## Computing nearest neighbor graph

## Computing SNN

pbmc <- FindClusters(pbmc, resolution = 0.30)

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 29474  
## Number of edges: 991051  
##   
## Running Louvain algorithm...  
## Maximum modularity in 10 random starts: 0.9349  
## Number of communities: 15  
## Elapsed time: 7 seconds

# Reorder clusters according to their similarity

This step isn’t explicitly required, but can ease the burden of merging cell clusters (discussed further in the section “Merging clusters and labeling cell types”) by reassigning each cluster by their position on a phylogenetic tree.

pbmc <- BuildClusterTree(pbmc, reorder = TRUE, reorder.numeric = TRUE)

## Reordering identity classes and rebuilding tree

# Run non-linear dimensional reduction (UMAP/tSNE)

Seurat offers several non-linear dimensional reduction techniques, such as tSNE and UMAP, to visualize and explore these datasets. The goal of these algorithms is to learn the underlying manifold of the data in order to place similar cells together in low-dimensional space. Cells within the graph-based clusters determined above should co-localize on these dimension reduction plots. As input to the UMAP and tSNE, we suggest using the same PCs as input to the clustering analysis.

pbmc <- RunUMAP(pbmc, dims = 1:30)

## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R-native UWOT using the cosine metric  
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'  
## This message will be shown once per session

## 16:32:27 UMAP embedding parameters a = 0.9922 b = 1.112

## 16:32:27 Read 29474 rows and found 30 numeric columns

## 16:32:27 Using Annoy for neighbor search, n\_neighbors = 30

## 16:32:27 Building Annoy index with metric = cosine, n\_trees = 50

## 0% 10 20 30 40 50 60 70 80 90 100%

## [----|----|----|----|----|----|----|----|----|----|

## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|  
## 16:32:31 Writing NN index file to temp file C:\Users\aneupane\AppData\Local\Temp\Rtmpkl6kJI\file53bc1618194  
## 16:32:31 Searching Annoy index using 1 thread, search\_k = 3000  
## 16:32:41 Annoy recall = 100%  
## 16:32:42 Commencing smooth kNN distance calibration using 1 thread with target n\_neighbors = 30  
## 16:32:44 Initializing from normalized Laplacian + noise (using irlba)  
## 16:32:45 Commencing optimization for 200 epochs, with 1270280 positive edges  
## 16:33:18 Optimization finished

plot <- DimPlot(pbmc, reduction = "umap")  
SaveFigure(plot, "umap\_louvain\_res\_p3", width = 9, height = 8)

## png   
## 2

You can save the object at this point so that it can easily be loaded back in without having to rerun the computationally intensive steps performed above, or easily shared with collaborators.

SaveObject(pbmc, "seurat\_obj\_clustered")  
# pbmc <- ReadObject("seurat\_obj\_clustered")

# Differential gene expression (finding cluster markers)

Seurat can help you find markers that define clusters via differential expression. By default, it identifies positive and negative markers of a single cluster (specified in ident.1), 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 feature to be detected at a minimum percentage in either of the two groups of cells, and the thresh.test argument requires a feature to be differentially expressed (on average) by some amount between the two groups. You can set both of these to 0, but with a dramatic increase in time - since this will test a large number of features that are unlikely to be highly discriminatory. As another option to speed up these computations, max.cells.per.ident can be set. This will downsample each identity class to have no more cells than whatever this is set to. While there is generally going to be a loss in power, the speed increases can be significant and the most highly deferentially expressed features will likely still rise to the top.

pbmc\_markers <- FindAllMarkers(pbmc, min.pct = 0.25, logfc.threshold = 0.25)

## Calculating cluster 1

## For a more efficient implementation of the Wilcoxon Rank Sum Test,  
## (default method for FindMarkers) please install the limma package  
## --------------------------------------------  
## install.packages('BiocManager')  
## BiocManager::install('limma')  
## --------------------------------------------  
## After installation of limma, Seurat will automatically use the more   
## efficient implementation (no further action necessary).  
## This message will be shown once per session

## Calculating cluster 2

## Calculating cluster 3

## Calculating cluster 4

## Calculating cluster 5

## Calculating cluster 6

## Calculating cluster 7

## Calculating cluster 8

## Calculating cluster 9

## Calculating cluster 10

## Calculating cluster 11

## Calculating cluster 12

## Calculating cluster 13

## Calculating cluster 14

## Calculating cluster 15

pbmc\_markers %>% group\_by(cluster) %>% top\_n(n = 2, wt = avg\_log2FC)

## # A tibble: 30 × 7  
## # Groups: cluster [15]  
## p\_val avg\_log2FC pct.1 pct.2 p\_val\_adj cluster gene   
## <dbl> <dbl> <dbl> <dbl> <dbl> <fct> <chr>   
## 1 0 4.19 0.996 0.192 0 1 LINGO2   
## 2 0 4.14 0.997 0.289 0 1 COL1A1   
## 3 0 2.73 0.605 0.087 0 2 GRIK1   
## 4 0 2.51 0.582 0.168 0 2 CNTNAP2   
## 5 0 2.35 0.926 0.419 0 3 EFNA5   
## 6 0 2.07 0.52 0.135 0 3 ENSG00000271945  
## 7 7.64e-92 3.71 0.736 0.068 2.64e-87 4 COL15A1   
## 8 2.07e-83 3.65 0.774 0.086 7.15e-79 4 PLEKHG1   
## 9 0 3.40 0.491 0.013 0 5 PDZRN4   
## 10 0 3.24 0.929 0.282 0 5 FN1   
## # ℹ 20 more rows

We include several tools for visualizing marker expression. VlnPlot (shows expression probability distributions across clusters), and FeaturePlot (visualizes feature expression on a tSNE or PCA plot) are our most commonly used visualizations. We also suggest exploring RidgePlot, CellScatter, and DotPlot as additional methods to view your dataset.

plot <- VlnPlot(pbmc, features = c("HELLS", "MYH7"), group.by = "tree.ident")  
SaveFigure(plot, "vln\_exp1", width = 16, height = 8)

## png   
## 2

# you can plot raw counts as well  
plot <- VlnPlot(pbmc, features = c("HELLS", "MYH7"), slot = "counts", log = TRUE)  
SaveFigure(plot, "vln\_exp2", width = 16, height = 8)

## png   
## 2

# Visualizing the top n genes per cluster

The Seurat PBMC tutorial makes use of the function DoHeatmapfor visualizing the top n genes per cluster in a single figure. This method works well for a few thousand cells, but loses resolution as the number of cells increase because individual columns have to be interpolated. The DotPlot function addresses this by displaying the average expression for all cells in a cluster.

top5 <- pbmc\_markers %>% group\_by(cluster) %>% top\_n(n = 5, wt = avg\_log2FC)  
to\_plot <- unique(top5$gene)  
plot <- DotPlot(pbmc, features = to\_plot, group.by = "tree.ident") + coord\_flip()  
SaveFigure(plot, "dplot\_top5", width = 9, height = 20)

## png   
## 2

# Merging clusters and labeling cell types

A common goal of single cell RNA-seq analysis is to eventually classify all clusters into a cell “type”. But what constitutes a cell type? Does a cluster represent a cell type, or a cell in a temporary state as it transitions from one type to the next? Because the appropriate number of clusters depends on the nature of the dataset and specific analysis goals, it’s usually a good idea to experiment with models that vary in cluster number.

However, there may be situations where one can’t seem to get the “right” number of clusters. That is, you may have a cluster that perfectly describe cells in one population, but another population may have too many subdivisions. In the following example we’ll demonstrate how to merge multiple clusters into a single unit to classify basic PBMC cell types, but it’s important to note that this strategy can be used in any scenario where one wishes to customize cell classification.

First we’ll take a look at the markers from the official tutorial and see which genes correspond to cell type, and then plot each gene on a UMAP plot to see which cluster they localize with.

# Note:

Note that the platelet marker PPBP isn’t present in this dataset. We also have a couple unclassified clusters that aren’t present in the official tutorial. We’ll name these two clusters after their top marker genes, IGHA2 and ZNF385D.

# Plotting PBMC markers  
markers <- c("IL7R", "CCR7", "S100A4", "CD14", "LYZ", "MS4A1", "CD8A",  
"FCGR3A", "MS4A7", "GNLY", "NKG7", "FCER1A", "CST3", "PPBP", "IGHA2", "ZNF385D")  
  
pbmc\_markers <- FeaturePlot(pbmc, features = markers)

## Warning in FetchData.Seurat(object = object, vars = c(dims, "ident", features),  
## : The following requested variables were not found: MS4A1, NKG7, PPBP, IGHA2

SaveFigure(pbmc\_markers, "pbmc\_markers", width = 30, height = 30)

## png   
## 2

Next, lets generate a UMAP plot with the clusters explicitly labeled so it’s a easier to see which cluster corresponds with our PBMC markers. You’ll notice that the cluster numbers tend to follow an ordered pattern (e.g. the green clusters on the bottom left contain 5, 6, and 7). This is a consequence of running the BuildClusterTree function and should make it easier to assign them to a single group (in the next step).

plot <- DimPlot(pbmc, reduction = "umap", label = TRUE) + NoAxes() + NoLegend()  
SaveFigure(plot, "umap\_louvain\_nolegend", width = 8, height = 7)

## png   
## 2

The last step involves generating a list where each cell type is assigned multiple clusters so they can be merged. You may wonder why the cell names in the list look different the label names. It’s because R list names don’t accept spaces or special characters like “+”.The last step involves generating a list where each cell type is assigned multiple clusters so they can be merged. You may wonder why the cell names in the list look different the label names. It’s because R list names don’t accept spaces or special characters like “+”.

# New IDs from official tutorial  
new\_ids <- c("Naive CD4 T", "Memory CD4 T", "CD14+ Mono", "B", "CD8 T",  
 "FCGR3A+ Mono", "NK", "DC", "IGHA", "ZNF385D")  
  
  
new\_id\_list <- list(Naive.CD4.T = c(17,19,20), Memory.CD4.T = c(15,16,18),  
 CD14.Mono = c(2), B = c(5:7), CD8.T = 13, FCGR3A.Mono = 1, NK = c(10:12,14),  
 DC = c(3,8), IGHA = 4, ZNF385D = 9)  
  
  
pbmc@meta.data$collapsed <- ""  
for (i in 1:length(new\_id\_list)) {  
 ind <- which(pbmc@meta.data$tree.ident %in% new\_id\_list[[i]])  
 pbmc@meta.data$collapsed[ind] <- names(new\_id\_list)[i]  
}  
  
pbmc@meta.data$collapsed <- factor(  
 pbmc@meta.data$collapsed, levels = names(new\_id\_list), ordered = TRUE)  
Idents(pbmc) <- pbmc@meta.data$collapsed  
  
names(new\_ids) <- levels(pbmc)  
pbmc <- RenameIdents(pbmc, new\_ids)

## Warning: Cannot find identity NA

plot <- DimPlot(pbmc, reduction = "umap", label = TRUE) + NoAxes() + NoLegend()  
SaveFigure(plot, "umap\_louvain\_nolegend\_names", width = 8, height = 7)

## png   
## 2