scRNAseq analysis (cardiomyopathy vs. control)

2023-07-14

## R Markdown

# Project description

The cells here are from cardiac organoids made by mixing 15% endocardial cells and 85% cardiomyocytes. These organoids are then treated with factors to induce trabeculation. In our analysis so far, we have noticed that all group except for LVNC respond to this stimulation.

The samples are as below: S25: Control GW64: Cardiomyopathy (LVNC: Left Ventricular Non-compaction Cardiomyopathy) GW129: Cardiomyopathy (HCM: Hypertrophic Cardiomyopathy) GW168: Cardiomyopathy (DCM: Dilated Cardiomyopathy)

The first question is to see differences between control and cardiomyopathy group. The second and more important question is that what is the differences between LVNC and other cardiomyopathy groups. Specifically, we are interested in the trabecular/compaction process and how the trabeculations is different between LVNC and two other cardiomyopathy group.

The raw reads from two sub-libraries were processed with Parse Biosciences spilt-pipe v1.0.6p using GRCh38 reference genome and with default parameters.

# 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)  
library(cowplot)

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

library(readr)  
setwd("Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/integrated\_analysis/")  
  
# rm(list = ls())  
  
fig\_path <- 'Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/integrated\_analysis/'  
# fig\_path <- "/research\_jude/rgs01\_jude/groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/integrated\_analysis/"  
  
data\_path <- 'Z:/ResearchHome/Groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures//integrated\_analysis/'  
# data\_path <- "/research\_jude/rgs01\_jude/groups/sapkogrp/projects/RNAseq/common/scRNAseq\_Paul\_cardiomyopathy/newvolume/analysis/combined\_figures/integrated\_analysis/"

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

# Reading in data

After reading in the data we’ll perform basic filtering on our expression matrix to remove low quality cells and uninformative genes. The parameter “min\_genes” will keep cells that have at least 100 genes, and similarly, “min\_cells” will keep genes that are expressed in at least 2 cells. Note: Seurat version 4.1 includes a convenience function to read Parse data from the DGE folder. If we 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)

When we create our Seurat object with 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)  
plot  
# SaveFigure(plot, "vln\_QC", width = 12, height = 6)  
  
plot1 <- FeatureScatter(pbmc, feature1 = "nCount\_RNA", feature2 = "percent.mt")  
plot1  
plot2 <- FeatureScatter(pbmc, feature1 = "nCount\_RNA", feature2 = "nFeature\_RNA")  
plot2  
  
# plot\_grid(plot1, plot2)  
# SaveFigure((plot1 + plot2),"scatter\_QC", width = 12, height = 6, res = 300)

Let’s break down the conditions in the subset argument below:

nFeature\_RNA is the number of genes detected in each cell. nCount\_RNA is the total number of molecules detected within a cell. Low nFeature\_RNA for a cell indicates that it may be dead/dying or an empty droplet. High nCount\_RNA and/or nFeature\_RNA indicates that the “cell” may in fact be a doublet (or multiplet). In combination with % of mitochondrial reads, removing outliers from these groups removes most doublets/dead cells/empty droplets, hence why filtering is a common pre-processing step.

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

# QC 1. 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.

# QC 2. 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.

# split the dataset into a list of seurat objects by samples  
ifnb.list <- SplitObject(pbmc, split.by = "sample")  
  
sample\_name <- names(ifnb.list)  
## For question 1  
for (sample\_name in names(ifnb.list)) {  
 seurat\_obj <- ifnb.list[[sample\_name]]  
  
 # Assuming you have a condition column in the Seurat object that indicates "Case" or "Control"  
 # Replace "condition\_column\_name" with the actual name of your condition column  
 # Assuming the condition is defined as "control" if the sample name contains "S\_25"  
 seurat\_obj$grouping\_var1 <- ifelse(grepl("S\_25", sample\_name), "control", "cardiomyopathy")  
 Idents(seurat\_obj) <- ifelse(grepl("S\_25", sample\_name), "control", "cardiomyopathy")  
 ifnb.list[[sample\_name]] <- seurat\_obj  
}  
  
  
  
# normalize and identify variable features for each dataset independently  
ifnb.list <- lapply(X = ifnb.list, FUN = function(x) {  
 x <- subset(x, subset = nFeature\_RNA < 5000 & nCount\_RNA < 20000 & percent.mt < 15)  
 x <- NormalizeData(x, normalization.method = "LogNormalize", scale.factor = 10000)  
 x <- FindVariableFeatures(x, selection.method = "vst", nfeatures = 2000)  
})  
  
  
# select features that are repeatedly variable across datasets for integration  
features <- SelectIntegrationFeatures(object.list = ifnb.list)

immune.anchors <- FindIntegrationAnchors(object.list = ifnb.list, anchor.features = features)  
# this command creates an 'integrated' data assay  
immune.combined <- IntegrateData(anchorset = immune.anchors)  
  
# SaveObject(immune.combined, "seurat\_obj\_post\_integration")  
# immune.combined <- ReadObject("seurat\_obj\_post\_integration")

# Perform an integrated analysis

Now we can run a single integrated analysis on all cells!

specify that we will perform downstream analysis on the corrected data note that the original unmodified data still resides in the ‘RNA’ assay

DefaultAssay(immune.combined) <- "integrated"  
  
# Run the standard workflow for visualization and clustering  
immune.combined <- ScaleData(immune.combined, verbose = FALSE)  
immune.combined <- RunPCA(immune.combined, npcs = 30, verbose = FALSE)  
immune.combined <- RunUMAP(immune.combined, reduction = "pca", dims = 1:30)  
immune.combined <- FindNeighbors(immune.combined, reduction = "pca", dims = 1:30)  
immune.combined <- FindClusters(immune.combined, resolution = 0.5)  
  
# SaveObject(immune.combined, "seurat\_obj\_post\_integration\_postCluster")  
# immune.combined <- ReadObject("seurat\_obj\_post\_integration\_postCluster")  
  
# Visualization  
p1 <- DimPlot(immune.combined, reduction = "umap", group.by = "grouping\_var1")  
p2 <- DimPlot(immune.combined, reduction = "umap", label = TRUE, repel = TRUE)  
# p1 + p2  
SaveFigure(p1,"dimplot\_cluster", width = 12, height = 6, res = 300)

To visualize the two conditions side-by-side, we can use the split.by argument to show each condition colored by cluster.

p1 <- DimPlot(immune.combined, reduction = "umap", split.by = "grouping\_var1")  
SaveFigure((p1),"dimplot\_cluster\_conditions\_side\_by\_side", width = 12, height = 6, res = 300)

# Identify conserved cell type markers

To identify canonical cell type marker genes that are conserved across conditions, we provide the FindConservedMarkers() function. This function performs differential gene expression testing for each dataset/group and combines the p-values using meta-analysis methods from the MetaDE R package

We can explore these marker genes for each cluster and use them to annotate our clusters as specific cell types.

# For performing differential expression after integration, we switch back to the original data  
DefaultAssay(immune.combined) <- "RNA"  
# loop over clusters  
top.markers <- {}  
top.markers.df <- list()  
identities <- sort(unique(Idents(immune.combined)))  
for(i in 1:length(identities)){  
ident <- identities[i]  
print(paste0("Doing identity: ", ident))  
immune.combined\_markers <- FindConservedMarkers(immune.combined, ident.1 = ident, grouping.var = "grouping\_var1", verbose = FALSE)  
top.markers.tmp <- rownames(head(immune.combined\_markers,1))  
top.markers <- c(top.markers, top.markers.tmp)  
top.markers.df.tmp <- immune.combined\_markers  
top.markers.df.tmp$cluster <- ident  
top.markers.df[[i]] <- top.markers.df.tmp  
}  
  
  
# [1] "Doing identity: 0"  
# [1] "Doing identity: 1"  
# [1] "Doing identity: 2"  
# [1] "Doing identity: 3"  
# [1] "Doing identity: 4"  
# [1] "Doing identity: 5"  
# [1] "Doing identity: 6"  
# [1] "Doing identity: 7"  
# [1] "Doing identity: 8"  
# [1] "Doing identity: 9"  
# [1] "Doing identity: 10"  
# [1] "Doing identity: 11"  
# [1] "Doing identity: 12"  
# Warning: Identity: 12 not present in group control. Skipping controlWarning: Only a single group was tested[1] "Doing identity: 13"  
# Warning: control has fewer than 3 cells in Identity: 13. Skipping controlWarning: Only a single group was tested[1] "Doing identity: 14"  
# [1] "Doing identity: 15"  
# Warning: Identity: 15 not present in group control. Skipping controlWarning: Only a single group was tested  
  
# SaveObject(top.markers, "seurat\_obj\_top\_markers")  
# SaveObject(top.markers.df, "seurat\_obj\_top\_markers\_dataframe\_list")  
  
top.markers <- ReadObject("seurat\_obj\_top\_markers")  
top.markers.df <- ReadObject("seurat\_obj\_top\_markers\_dataframe\_list")  
  
lapply(top.markers.df, head, 4)  
  
# > top.markers  
top.markers <- c("FBN2", "TTN", "ADGRL3", "PCDH9", "CLSPN", "NR2F1-AS1", "ANLN", "CDH19", "CENPF", "CNTNAP2", "SLC7A11", "CDH19", "LUM", "SHOX2", "FGR", "NOS2")  
# FBN2= "ICCs"  
# TTN = "Cardiomyocyte"  
# ADGRL3 = "Pericyte"  
# PCDH9 = "Neuro\_endocrine"  
# CLSPN = ""  
# NR2F1-AS1 = ""  
# ANLN = "Epithelial"  
# CDH19 = "Enterocytes\_Schwann\_cell"  
# CENPF = "Gamma\_delta\_T\_cells"  
# CNTNAP2 = "Neuronal"  
# SLC7A11 = ""  
# CDH19 = "Enterocytes\_Schwann\_cell"  
# LUM = "Fibroblast"  
# SHOX2 = "Interneurons"  
# FGR = "Fibroblast"  
# NOS2 = "Enteric\_glia\_cells"  
  
#0 FBN2= "ICCs"  
#1 TTN = "Cardiomyocyte"  
#2 ADGRL3 = "Pericyte"  
#3 PCDH9 = "Neuro\_endocrine"  
#4 CLSPN = ""  
#5 NR2F1-AS1 = ""  
#6 ANLN = "Epithelial"  
#7 CDH19 = "Enterocytes\_Schwann\_cell"  
#8 CENPF = "Gamma\_delta\_T\_cells"  
#9 CNTNAP2 = "Neuronal"  
#10 SLC7A11 = ""  
#11 CDH19 = "Enterocytes\_Schwann\_cell"  
#12 LUM = "Fibroblast"  
#13 SHOX2 = "Interneurons"  
#14 FGR = "Fibroblast"  
#15 NOS2 = "Enteric\_glia\_cells"

We can explore these marker genes for each cluster and use them to annotate our clusters as specific cell types.

DefaultAssay(immune.combined) <- "RNA"  
p1 <- FeaturePlot(immune.combined, features = top.markers, min.cutoff = "q9")  
  
SaveFigure(p1,"dimplot\_individual\_markers", width = 12, height = 12, res = 300)

immune.combined <- RenameIdents(immune.combined, `0` = "ICCs", `1` = "Cardiomyocyte", `2` = "Pericyte",  
 `3` = "Neuro\_endocrine", `4` = "Unknown\_1", `5` = "Unknown\_2", `6` = "Epithelial", `7` = "Enterocytes\_Schwann\_cell", `8` = "Gamma\_delta\_T\_cells",  
 `9` = "Neuronal", `10` = "Unknown\_3", `11` = "Enterocytes\_Schwann\_cell", `12` = "Fibroblast", `13` = "Interneurons", `14` = "Fibroblast",   
 `15` = "Enteric\_glia\_cells")  
p1 <- DimPlot(immune.combined, label = TRUE, repel = TRUE)  
SaveFigure(p1,"dimplot\_annotated", width = 12, height = 12, res = 300)

The DotPlot() function with the split.by parameter can be useful for viewing conserved cell type markers across conditions, showing both the expression level and the percentage of cells in a cluster expressing any given gene. Here we plot 2-3 strong marker genes for each of our 14 clusters.

markers.to.plot <- unique(c(top.markers, "ADCY5", "TECRL", "CACNA1C", "SORBS2"))  
p1 <- DotPlot(immune.combined, features = markers.to.plot, cols = c("blue", "red"), dot.scale = 8, split.by = "grouping\_var1") +  
 RotatedAxis()  
  
SaveFigure(p1,"dotplot\_top\_markers", width = 12, height = 12, res = 300)

Identify differential expressed genes across conditions Now that we’ve aligned the stimulated and control cells, we can start to do comparative analyses and look at the differences induced by stimulation. One way to look broadly at these changes is to plot the average expression of both the stimulated and control cells and look for genes that are visual outliers on a scatter plot. Here, we take the average expression of both the stimulated and control Cardiomyocytes and ICCs populations and generate the scatter plots, highlighting genes that exhibit dramatic responses.

library(ggplot2)  
library(cowplot)  
theme\_set(theme\_cowplot())  
Cardiomyocytes <- subset(immune.combined, idents = "Cardiomyocyte")  
Idents(Cardiomyocytes) <- "grouping\_var1"  
avg.Cardiomyocytes <- as.data.frame(log1p(AverageExpression(Cardiomyocytes, verbose = FALSE)$RNA))  
avg.Cardiomyocytes$gene <- rownames(avg.Cardiomyocytes)  
  
ICCs <- subset(immune.combined, idents = "ICCs")  
Idents(ICCs) <- "grouping\_var1"  
avg.ICCs <- as.data.frame(log1p(AverageExpression(ICCs, verbose = FALSE)$RNA))  
avg.ICCs$gene <- rownames(avg.ICCs)  
  
# genes.to.label = c("ISG15", "LY6E", "IFI6", "ISG20", "MX1", "IFIT2", "IFIT1", "CXCL10", "CCL8")  
genes.to.label = c("TTN", "BAG3", "MYH7", "MYBPC3", "TNNT2", "TNNI3")  
p1 <- ggplot(avg.Cardiomyocytes, aes(control, cardiomyopathy)) + geom\_point() + ggtitle("cardiomyocytes")  
p1 <- LabelPoints(plot = p1, points = genes.to.label, repel = TRUE)  
p2 <- ggplot(avg.ICCs, aes(control, cardiomyopathy)) + geom\_point() + ggtitle("ICCs")  
p2 <- LabelPoints(plot = p2, points = genes.to.label, repel = TRUE)  
p1 + p2

As you can see, many of the same genes are upregulated in both of these cell types and likely represent a conserved pathway.

Because we are confident in having identified common cell types across condition, we can ask what genes change in different conditions for cells of the same type. First, we create a column in the meta.data slot to hold both the cell type and stimulation information and switch the current ident to that column. Then we use FindMarkers() to find the genes that are different between cardiomyopathy and control cardiomyocyte cells.

immune.combined$celltype.cardiomyopathy <- paste(Idents(immune.combined), immune.combined$grouping\_var1, sep = "\_")  
immune.combined$celltype <- Idents(immune.combined)  
Idents(immune.combined) <- "celltype.cardiomyopathy"  
trt\_control.markers <- FindMarkers(immune.combined, ident.1 = "Cardiomyocyte\_cardiomyopathy", ident.2 = "Cardiomyocyte\_control", verbose = FALSE)  
head(trt\_control.markers, n = 15)  
  
SaveObject(trt\_control.markers, "trt\_control.DGE")

Another useful way to visualize these changes in gene expression is with the split.by option to the FeaturePlot() or VlnPlot() function. This will display FeaturePlots of the list of given genes, split by a grouping variable (stimulation condition here).

p1 <- FeaturePlot(immune.combined, features = genes.to.label, split.by = "grouping\_var1", max.cutoff = 3,  
 cols = c("grey", "red"))  
  
SaveFigure(p1,"DGE\_feature\_plot", width = 12, height = 12, res = 300)