RM_HW_W10

Rekha Murali

3/21/2019

Problem 1

1.1

Pull in data, check dimensions

```
readcounts <- read.table("/Users/rekhamurali/Downloads/WT-1.dge.txt", header=TRUE)
#View(readcounts)
```

The rows are the genes and the columns represent the cells. In this dataset we have 25319 genes and data for 1400 single cells.

1.2

Create single cell experiment object

```
suppressPackageStartupMessages(library(SingleCellExperiment))
rownames(readcounts) <- readcounts$GENE
readcounts$GENE <- NULL
counts_matrix <- as.matrix(readcounts)
sce <- SingleCellExperiment(assays = list(counts = counts_matrix))
```

1.3

Show subset of counts matrix

```
counts(sce[1:5,1:5])
```

```
##           GGTCCAGATCAT CCCCCATTATGC TTTACCTAACAG ATTCCCGAGTCA ATTCATTCTTTG
## A1BG                 0            0            0            0            0
## A1BG-AS1             0            0            0            0            0
## A2ML1                0            0            0            0            0
## A2ML1-AS1            0            0            0            0            0
## A2ML1-AS2            0            0            0            0            0
```

1.4

Sequencing Depth

```r
seq_depth <- as.data.frame(colSums(counts(sce[,1:5]), na.rm = TRUE))
seq_depth
```

```
##              colSums(counts(sce[, 1:5]), na.rm = TRUE)
## GGTCCAGATCAT                                     42572
## CCCCCATTATGC                                     33667
## TTTACCTAACAG                                     30704
## ATTCCCGAGTCA                                     28555
## ATTCATTCTTTG                                     25858
```

1.5

Non-zero gene counts

```r
sce_subset <- as.data.frame(counts(sce[,1:5]))
non_zero <-  length(which(rowSums(sce_subset) != 0))
print(paste("There are: ",non_zero, " non-zero gene counts in the first five cells"))
```

```
## [1] "There are:  12066  non-zero gene counts in the first five cells"
```

1.6

Changing row and column names

```r
# change row and column names but keep track of original
# we can do this through metadata

# colData
# set easy to read identities rather than barcodes
cell_metadata <- data.frame(cell_ident = c(paste0("cell_", 1:1400)))
rownames(cell_metadata) <- colnames(sce)
cell_metadata$original_cell_ident <- colnames(sce)

# rowData
# add metadata to the features
gene_metadata <- data.frame(gene_ident = c(paste0("gene_", 1:25319)))
rownames(gene_metadata) <- rownames(sce)
gene_metadata$original_gene_ident <- rownames(sce)

#now append metadata

colData(sce) <- DataFrame(cell_metadata)
# call with sce$some_attribute_in_colData
rowData(sce) <- DataFrame(gene_metadata)
# call with rowData(sce)

# now if we want to change it, the originals are stored
# change col names as example
```

```
colnames(sce) <- sce$cell_ident
# check that it worked
counts(sce[1:5,1:5])
```

```
##           cell_1 cell_2 cell_3 cell_4 cell_5
## A1BG           0      0      0      0      0
## A1BG-AS1       0      0      0      0      0
## A2ML1          0      0      0      0      0
## A2ML1-AS1      0      0      0      0      0
## A2ML1-AS2      0      0      0      0      0
```

1.7

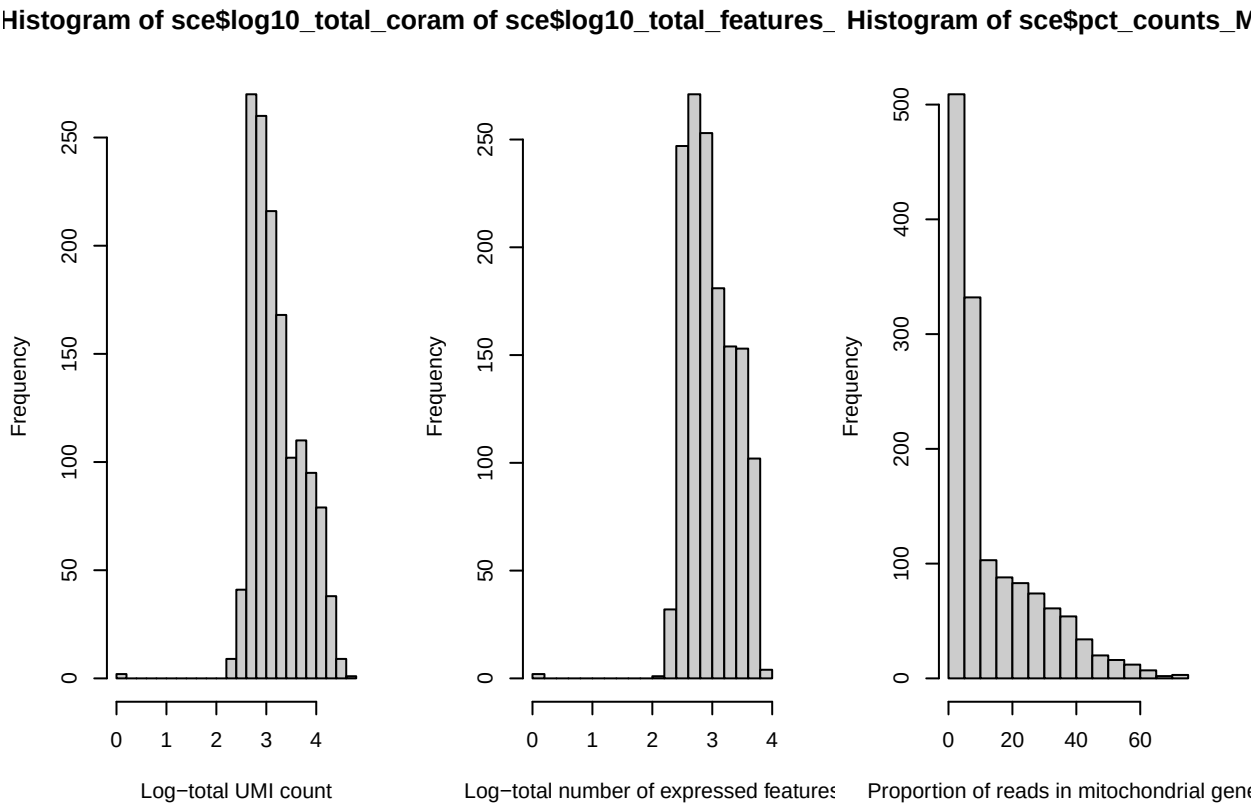
QC

From sce tutorial

```
suppressPackageStartupMessages(library(scater))

mito.genes <- grep(pattern = "^MT-", rownames(sce), value = TRUE)

sce <- calculateQCMetrics(sce, feature_controls=list(Mito=c(mito.genes)))
par(mfrow=c(1,3))
hist(sce$log10_total_counts, breaks=20, col="grey80",
    xlab="Log-total UMI count")
hist(sce$log10_total_features_by_counts, breaks=20, col="grey80",
    xlab="Log-total number of expressed features")
hist(sce$pct_counts_Mito, breaks=20, col="grey80",
    xlab="Proportion of reads in mitochondrial genes")
```



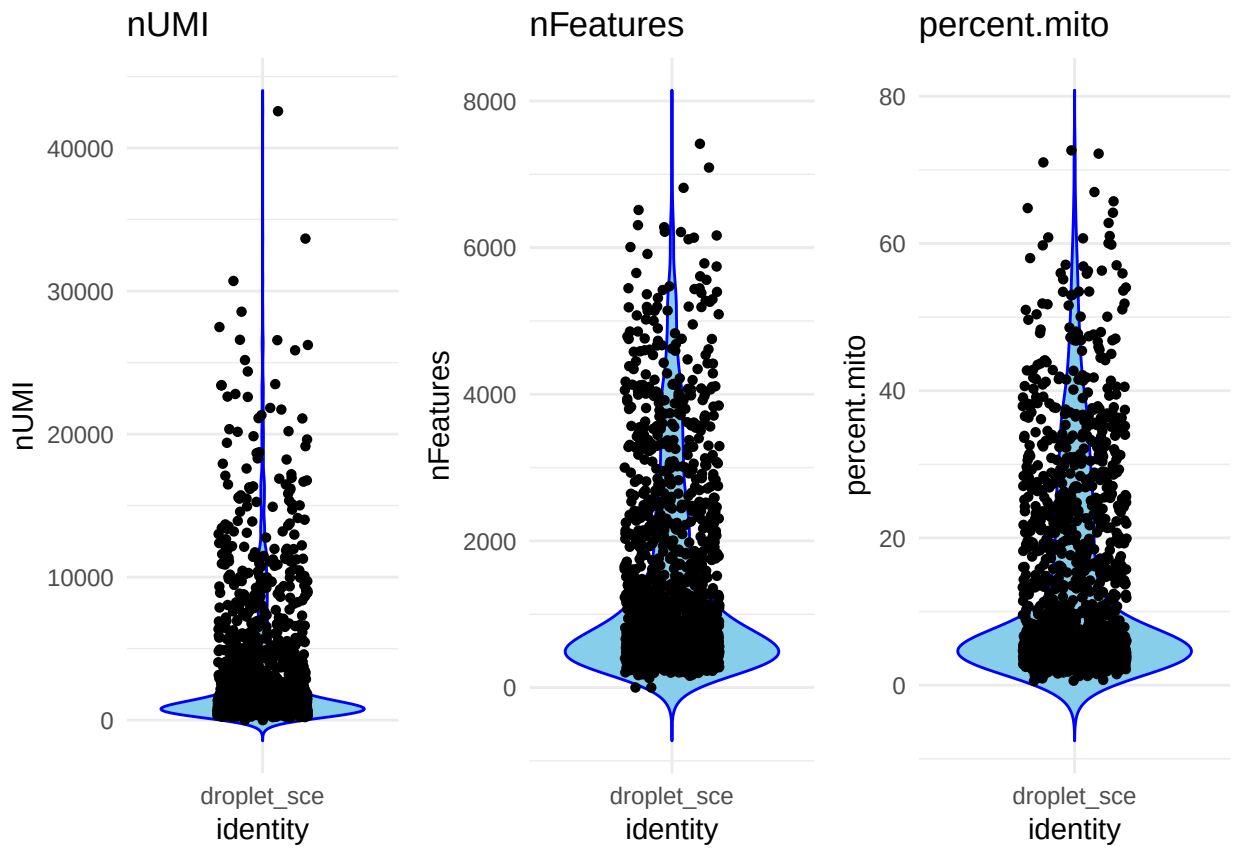
now generate violin plots with ggplot for nicer visualization on a per cell basis

```
suppressPackageStartupMessages(library(ggplot2))
suppressPackageStartupMessages(library(gridExtra))

# make df with QC metrics for input to ggplot2
sce_QC <- as.data.frame(sce$total_counts, row.names = colnames(sce))
sce_QC$nUMI <- sce$total_counts
sce_QC$`sce$total_counts` <- NULL
sce_QC$nFeatures <- sce$total_features_by_counts
sce_QC$identity <- c(rep("droplet_sce",ncol(sce_QC)))
sce_QC$percent.mito <- sce$pct_counts_Mito


# make violin plots
plot1 <- ggplot(sce_QC, aes(x=identity, y=nUMI)) +
  geom_violin(trim=FALSE, fill='skyblue', color="blue")+ theme_minimal()+
  geom_jitter(shape=16, position=position_jitter(0.2))+ggtitle("nUMI")
plot2 <- ggplot(sce_QC, aes(x=identity, y=nFeatures)) +
  geom_violin(trim=FALSE, fill='skyblue', color="blue")+ theme_minimal()+
  geom_jitter(shape=16, position=position_jitter(0.2))+ggtitle("nFeatures")
plot3 <- ggplot(sce_QC, aes(x=identity, y=percent.mito)) +
  geom_violin(trim=FALSE, fill='skyblue', color="blue")+ theme_minimal()+
  geom_jitter(shape=16, position=position_jitter(0.2))+ggtitle("percent.mito")
```

```
suppressWarnings(grid.arrange(plot1, plot2, plot3, ncol=3))
```



Explanation

The total counts, or number of UMIs, represent the total number of transcripts captured for each cell. It is essentially a sum of all the count values in the matrix for each cell. During droplet based sequencing each transcript receives a UMI, so this can also be thought of as library size. The total number of genes (nFeatures) for each cell are essentially all the genes that have at least one count value for each cell.

The log transformed histograms from above actually show a somewhat normal distribution, but maybe it is skewed towards the lower ends. But it is visualized much more clearly in the violin plots that show the values for each cell. You can see where the bulk of the data lies, I think these values for nFeature and nUMI look close to expected for scRNA-seq.

1.8

Filtering

```
sce <- sce[, sce$total_counts > 250]
sce <- sce[, sce$total_counts < 25000]
sce <- sce[, sce$total_features_by_counts > 200]
sce <- sce[, sce$total_features_by_counts < 6000]
sce <- sce[,sce$pct_counts_Mito < 40]
```

Explanation

I am going to filter by: 200 < nFeatures < 6000 and 250 < nUMI < 25000. Additionally, I am removing cells that have over 40% mitochondiral content, which is pretty high. I am not actually doing outlier analysis. But, those values look like outliers when compared to our data. I think this is still on the less strict side when you look at where most of our data lies. I am doing this because cells with too low of genes or transcripts could confound the data. We definitely would not want cells that had zero genes or transcripts. We also discussed doublets during class so the really high values could be because two cells were captured in one droplet.

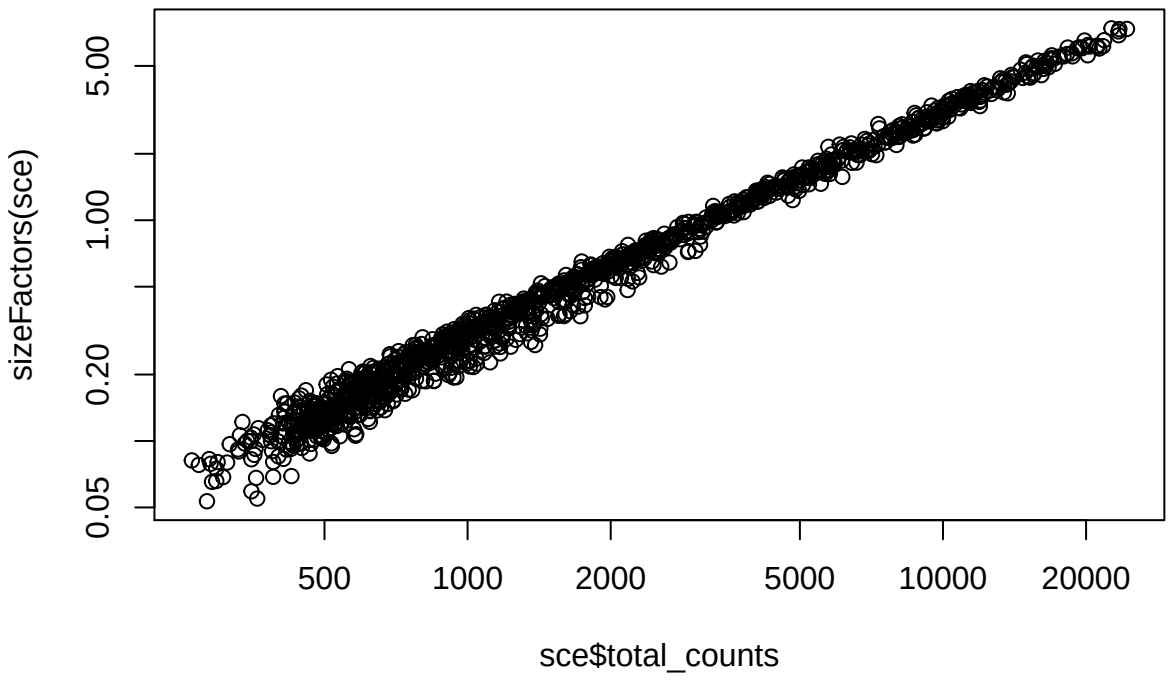
This ends up only removing 119 cells, I still retain ~91% of my cells. Since we discussed gene drop out in class I think it makes more sense to be on the conservative side in terms of lower counts. I lose the most cells through my mitochondrial filtering, in general this dataset has pretty high mitochondrial content. One explanation for overly high mitochondrial content could be cells that are stressed or dying, which is why I set a cut-off for that.

1.9 Normalizing the data

```
suppressPackageStartupMessages(library(scran))
sce <- computeSumFactors(sce, min.mean=0.1)
summary(sizeFactors(sce))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.05334 0.18725 0.38625 1.00000 1.22633 7.39883
```

```
plot(sce$total_counts, sizeFactors(sce), log="xy")
```




```
sce_normalize <- scater::normalize(sce,
  exprs_values = "counts", return_log = TRUE,
  log_exprs_offset = NULL, centre_size_factors = TRUE,
  preserve_zeroes = FALSE)
# to access a subset of normalized count data:
# logcounts(sce_normalize[,1:10])
```

Explanation

My figure shows the same correlation from Figure 6 of the tutorial between size factors and library sizes. As the tutorial states, this indicates that capture efficiency and sequencing depth are major biases.

You can access the log normalized counts by logcounts(sce_normalize).

Problem 2: Seurat (a much better tool than sce!)

note: I use seurat v3

2.1 create a seurat object

```
suppressPackageStartupMessages(library(Seurat))
WT_seurat <- suppressWarnings(CreateSeuratObject(counts = readcounts,
                                                 project = "WT_seurat"))
```

2.2 Filter based on same parameters

```
# Grab mito and keep as metadata for filtering
MT.features <- grep(pattern = "^MT-", x = rownames(x = WT_seurat), value = TRUE)
fraction.mito <- Matrix::colSums(x = GetAssayData(
  object = WT_seurat, slot = 'counts')[MT.features, ])/Matrix::colSums(x = GetAssayData(
    object = WT_seurat, slot = 'counts'))

WT_seurat[['fraction.mito']] <- fraction.mito
# Filter
WT_seurat <- subset(x = WT_seurat, subset = nFeature_RNA > 200&
                      nFeature_RNA < 6000 & fraction.mito < 0.40&
                      nCount_RNA > 250  & nCount_RNA < 25000)
# if you look at object info you can confirm the same amount of cells has been removed
```

2.3 Normalize the data

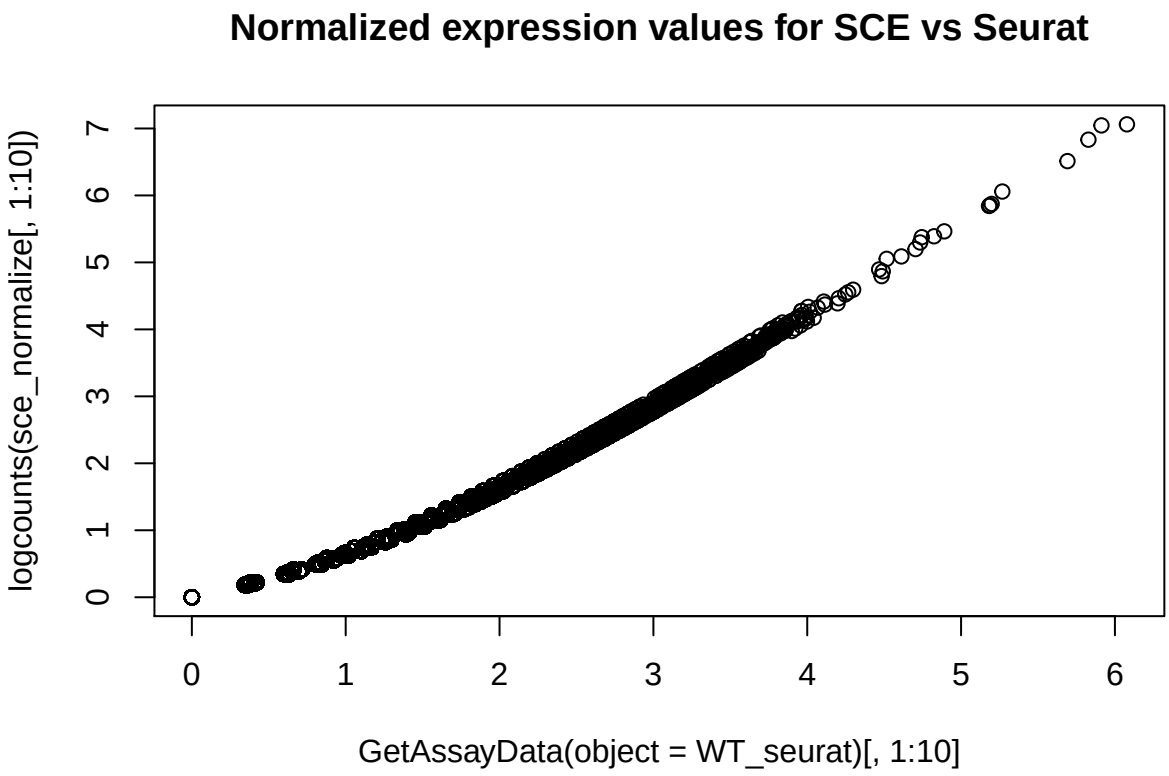
```
WT_seurat <- NormalizeData(object = WT_seurat,
                           normalization.method = "LogNormalize",
                           scale.factor = 1e4)

# normalized counts here
# GetAssayData(object = WT_seurat)
```

You access the normalized counts by GetAssayData(object = WT_seurat).

2.4 Compare normalized values from Seurat and SCE

```
# First I need to change back my column names in the SCE object
colnames(sce_normalize) <- sce_normalize$original_cell_ident
plot(x = GetAssayData(object = WT_seurat)[,1:10],
     y = logcounts(sce_normalize[,1:10]))
title(main = "Normalized expression values for SCE vs Seurat")
```



Explanation

This looks pretty good, it doesn't look that far off from y = x, though I did not actually fit a line. This means the normalization is similar between the two. From what I remember Seurat does normalize by library size (nUMI) and also does a log transformation, which is similar to the SCE workflow. So, the fact they match up well is expected.

Problem 3

identify cell type

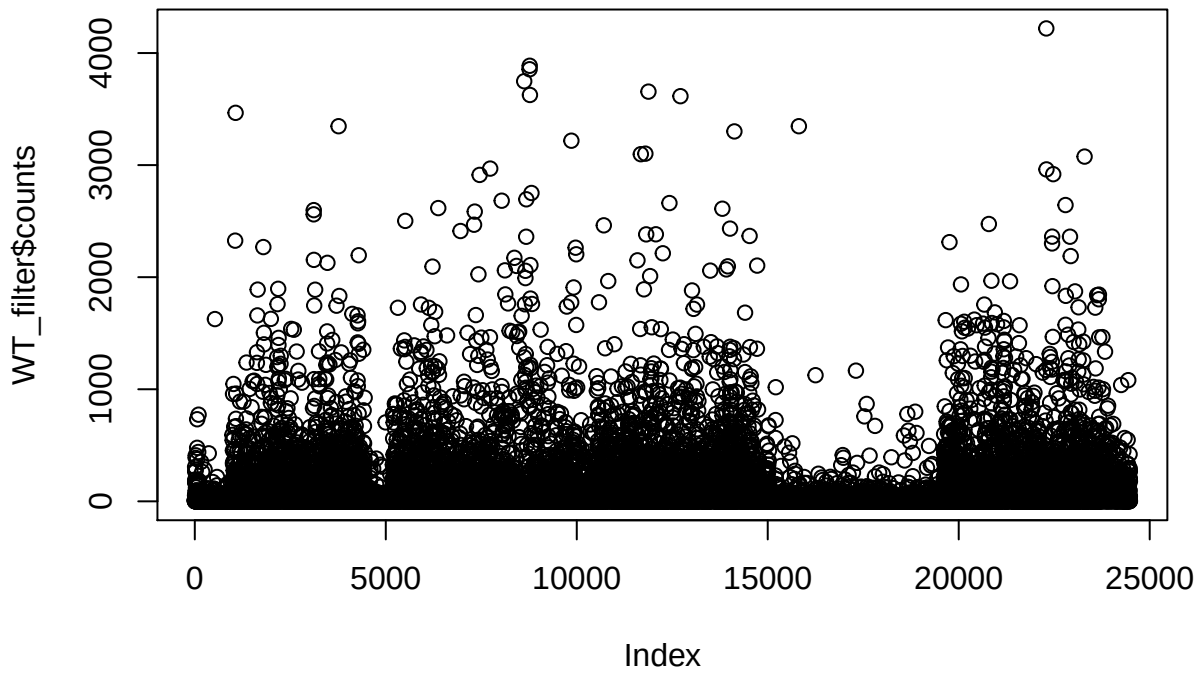
```
suppressPackageStartupMessages(library(mygene))
# aggregate gene counts across cells
WT_counts <- as.data.frame(GetAssayData(object = WT_seurat))
WT_sums <- as.data.frame(rep(0,nrow(WT_counts)))
rownames(WT_sums) <- rownames(WT_counts)
WT_sums$counts <- rowSums(WT_counts)
WT_sums$`rep(0, nrow(WT_counts))` <- NULL
WT_sums$gene <- rownames(WT_sums)

# remove mitochondrial and ribosomal genes from this list
to_remove <- grep(pattern = "^MT-|^RPS|^RPL",
```

```
                  x = rownames(x = WT_sums), value = TRUE)

WT_filter = WT_sums[ !(row.names(WT_sums) %in% to_remove), ]

# look at highest
# probably most will not be cell type specific
WT_filter <- as.data.frame(WT_filter)
plot(WT_filter$counts)
```




```
# investigate highly expressed genes
subset <- WT_filter[c(which(WT_filter$counts > 2000)),]

check.genes <- queryMany(c(row.names(subset)), scopes="symbol",
                         species="human", fields = c("name"))
```

```
## Finished
## Pass returnall=TRUE to return lists of duplicate or missing query terms.
```

```
# look at subset
check.genes[1:5,]
```

```
## DataFrame with 5 rows and 5 columns
##         query         _id   X_score                                 name
##   <character> <character> <numeric>                          <character>
```

```
## 1        ACTB          60  90.10658                             actin beta
## 2       ACTG1          71 90.109825                           actin gamma 1
## 3       APLP2         334   86.5752 amyloid beta precursor like protein 2
## 4       CALD1         800 89.445465                            caldesmon 1
## 5       CALM1         801  89.14005                           calmodulin 1
##    notfound
##   <logical>
## 1        NA
## 2        NA
## 3        NA
## 4        NA
## 5        NA
```

```
# found gene of interest to look at
WT_filter["CHGA",]
```

```
##        counts gene
## CHGA 9.673007 CHGA
```

Explanation:

I took an approach that is not optimal and it is not the approach I would take if I had more time. If I had more time I would consider finding gene to gene correlations in a cluster and finding gene modules which I could then use as a searching point. I would have wanted to compare my expressed genes to known molecular signatures of different cell types, but that is difficult considering those can be thousands of genes for each cell type. Additionally, it is nuanced because the top expressed genes in a cell don't necessarily relate to its identity. That is why I removed all the mitochondrial and ribosomal genes when looking at the highest expressed genes. I also pooled all the gene expression values across the cells since we knew it was a homogenous population. Ideally I would have continued processing it with Seurat and found differentially expressed genes for each cluster (if there were multiple cell types) and that would have been another starting point for narrowing down cell type. Even with removing the mitochondrial and ribosomal genes most of the top markers are not indicitave of cell type at all. I used the mygene library database to query them and could immediately tell that most of them did not relate to cell type.

However, it would be too difficult to look at all my genes which is why I tried to narrow them to top expressed. Once that did not yield results I used this resource: http://biocc.hrbmu.edu.cn/CellMarker/index.jsp. From there, I could browse some of the gene modules for each cell type. But even this is way too broad an approach. Randomly, I decided to look at endocrine because there was only one cell marker that they had listed, as oppose to larger lists from the other cell types. I saw that chromogranin (CHGA) was the only listed marker and I also had some expression in my dataset, though the expression is not very high. But, I don't think expression of cell type specific markers necessarily needs to be high. From there I looked at the protein atlas resource: https://www.proteinatlas.org/ENSG00000100604-CHGA/tissue. I found that while the protein is expressed in several tissues, the RNA expression is limited to endocrine tissues.

Therefore, I am going with endocrine as a guess but I am not confident at all!! My methods were definitely not the correct way to identify cell type.