Single Cell RNA-Seq (scRNA-seq)

Guannan Shen

December 11, 2018

Contents


```
## set up workspace
library(Seurat)
library(knitr)
library(tidyverse)
library(magrittr)
library(stats)
library(data.table)
options(stringsAsFactors = F)
options(dplyr.width = Inf)
getwd()
```

```
## [1] "/home/guanshim/Documents/Stats/CIDA_OMICs/7659Stats_Genetics/Proj_Single_Cell/DataRaw"
```

```
## not in function
"%nin%" <- Negate("%in%")

# ######## clean memory ######################## rm(list =
# ls()) gc() slotNames(x) getSlots(x)
```

1 Setup the Seurat Object

```
# Sample GSM3374613: **Original single cell RNA-seq library**
# from the 10x genomics 3' end library prep. PBMCs were
# isolated from a health human donor.

## C:\Users\hithr\Documents\Stats\7659Genetics\Proj

## write.table(test, file='test.tsv', quote=FALSE, sep='\t',
## col.names = NA)
ctrl.data <- read.table("~/Documents/Stats/7659Genetics/Proj/GSM3374613_kirkpatrick_umis.tsv.gz",
    sep = "\t", header = T)
### ctrl.data <-
### read.table('C:/Users/hithr/Documents/Stats/7659Genetics/Proj/GSM3374613_kirkpatrick_umis.tsv.gz',
### sep = '\t', header = T) #, sep = '\t', header = T)
barcodes.ctrl <- colnames(ctrl.data)
```

```
genes.ctrl <- ctrl.data[, 1]
dim(ctrl.data)
```

```
## [1] 21957  3195
```

```
head(ctrl.data[, 1:6])
```

```
##       gene AAACCTGCACCTCGGA AAACCTGCATGCTAGT AAACCTGCATGGGAAC
## 1  5S_rRNA                0                0                0
## 2      7SK                0                0                0
## 3     A1BG                0                0                0
## 4 A1BG-AS1                0                0                0
## 5      A2M                0                0                0
## 6  A2M-AS1                0                0                0
##   AAACCTGGTGATGTGG AAACGGGAGCGGCTTC
## 1                0                0
## 2                0                0
## 3                0                0
## 4                0                0
## 5                0                0
## 6                0                0
```

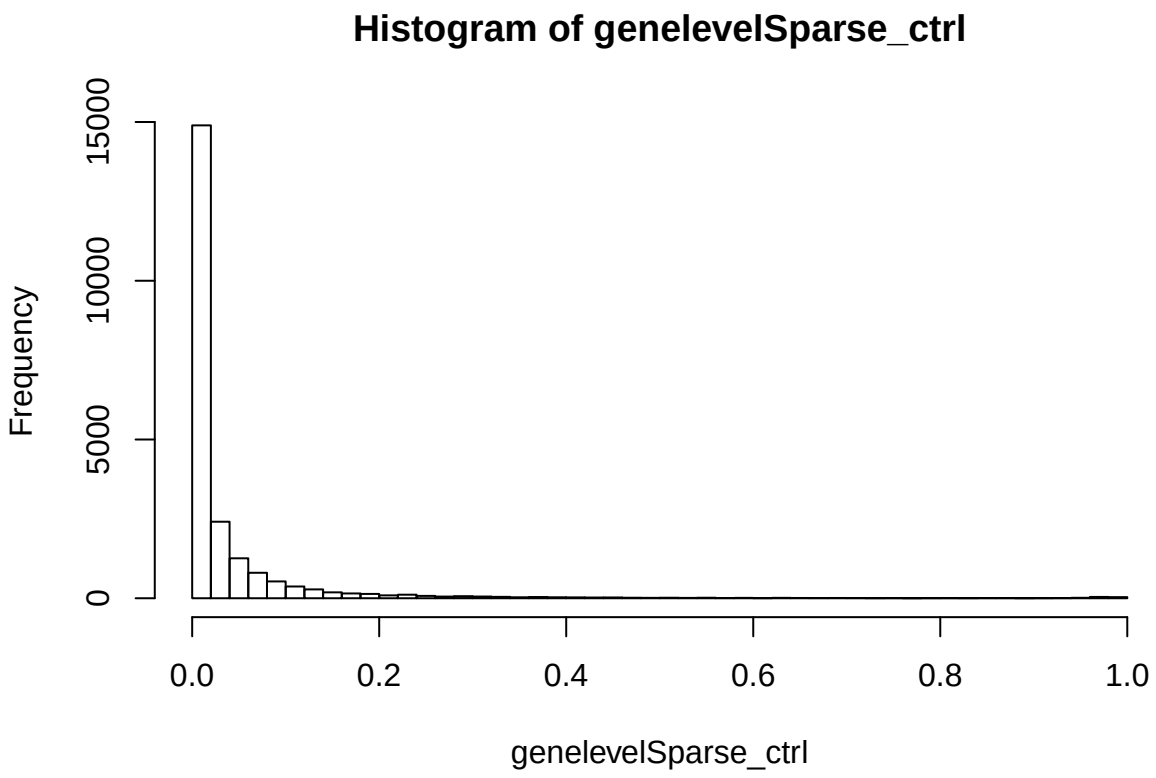
```
## get the pure counts
ctrl <- ctrl.data
colnames(ctrl) <- NULL
ctrl <- ctrl[, -1]
rownames(ctrl) <- NULL
ctrl <- as.matrix(ctrl)
head(ctrl[, 1:6])
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
## [6,]    0    0    0    0    0    0
```

```
## check the sparsity
genelevelSparse_ctrl <- apply(ctrl, 1, function(x) {
    sum(x != 0)/length(x)
})
sum(genelevelSparse_ctrl != 0)
```

```
## [1] 20818
```

```
hist(genelevelSparse_ctrl, freq = T, breaks = 50)
```




```
## boxplot(log2(ctrl.data), main = 'The Counts of Single Cell
## RNA-Seq', ylab= 'log2(Counts)') Sample GSM3374614 PBMCs
## were isolated from a health human donor and a LNA probe was
## used to hybridize selected cells for targeted resequencing.
stim.data <- read.table("~/Documents/Stats/7659Genetics/Proj/GSM3374614_mkcell_pulldown_umis.tsv.gz",
    sep = "\t", header = T)
dim(stim.data)
```

```
## [1] 21215  3195
```

```
barcodes.stim <- colnames(stim.data)
genes.stim <- stim.data[, 1]
dim(stim.data)
```

```
## [1] 21215  3195
```

```
head(stim.data[, 1:6])
```

```
##        gene AAACCTGCACCTCGGA AAACCTGCATGCTAGT AAACCTGCATGGGAAC
## 1       7SK                0                0                0
## 2      A1BG                0                0                0
## 3  A1BG-AS1                0                0                0
## 4       A2M                0                0                0
## 5   A2M-AS1                0                0                0
## 6 A2ML1-AS2                0                0                0
##   AAACCTGGTGATGTGG AAACGGGAGCGGCTTC
## 1                0                0
## 2                0                0
## 3                0                0
## 4                0                0
## 5                0                0
```

```
## 6                0                 0
```

```
## test PF4 megakaryocyte
"PF4" %in% genes.stim
```

```
## [1] TRUE
```

```
"PF4" %in% genes.ctrl
```

```
## [1] TRUE
```

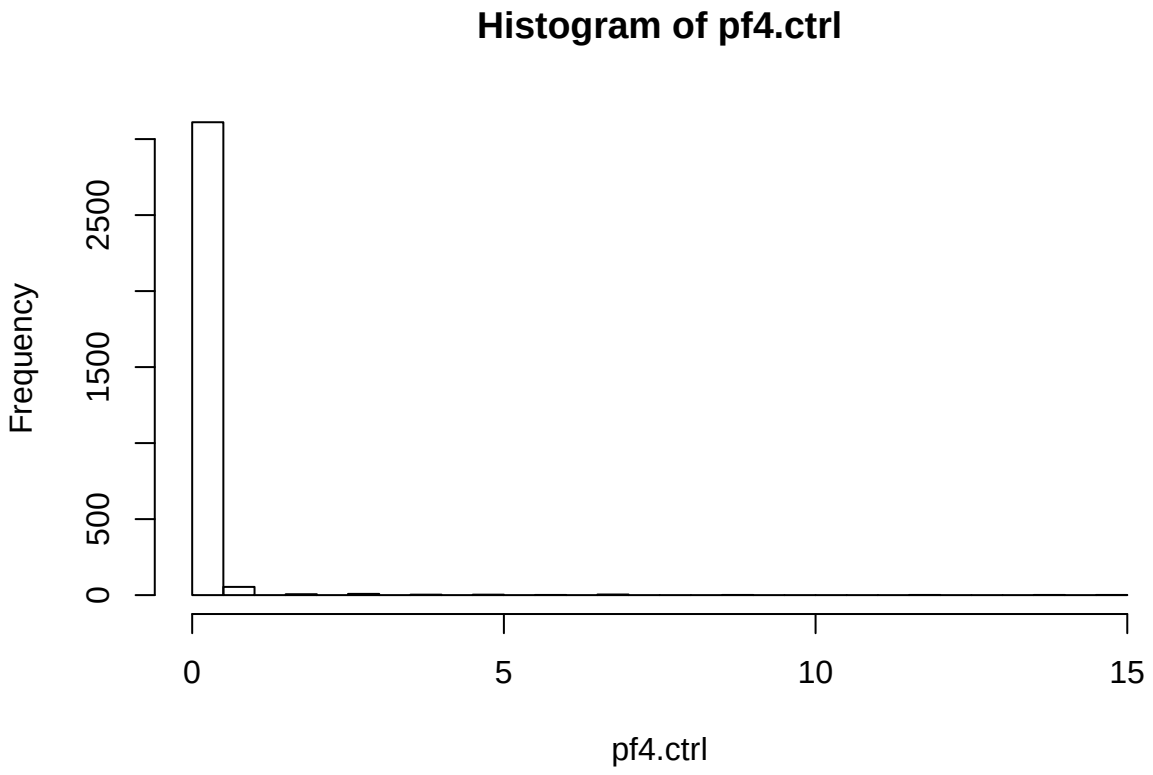
```
"CD8" %in% genes.stim
```

```
## [1] FALSE
```

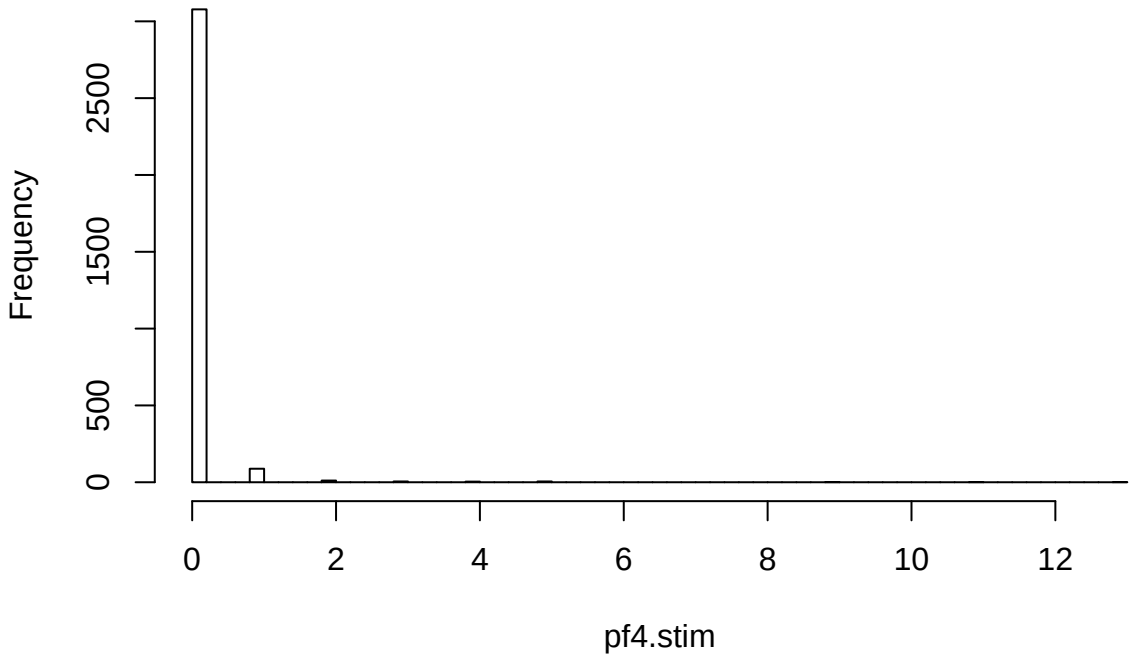
```
"CD8" %in% genes.ctrl
```

```
## [1] FALSE
```

```
pf4.ctrl <- as.numeric(ctrl.data[ctrl.data$gene == "PF4", -1])
pf4.stim <- as.numeric(stim.data[stim.data$gene == "PF4", -1])
hist(pf4.ctrl, breaks = 50)
```




```
hist(pf4.stim, breaks = 50)
```



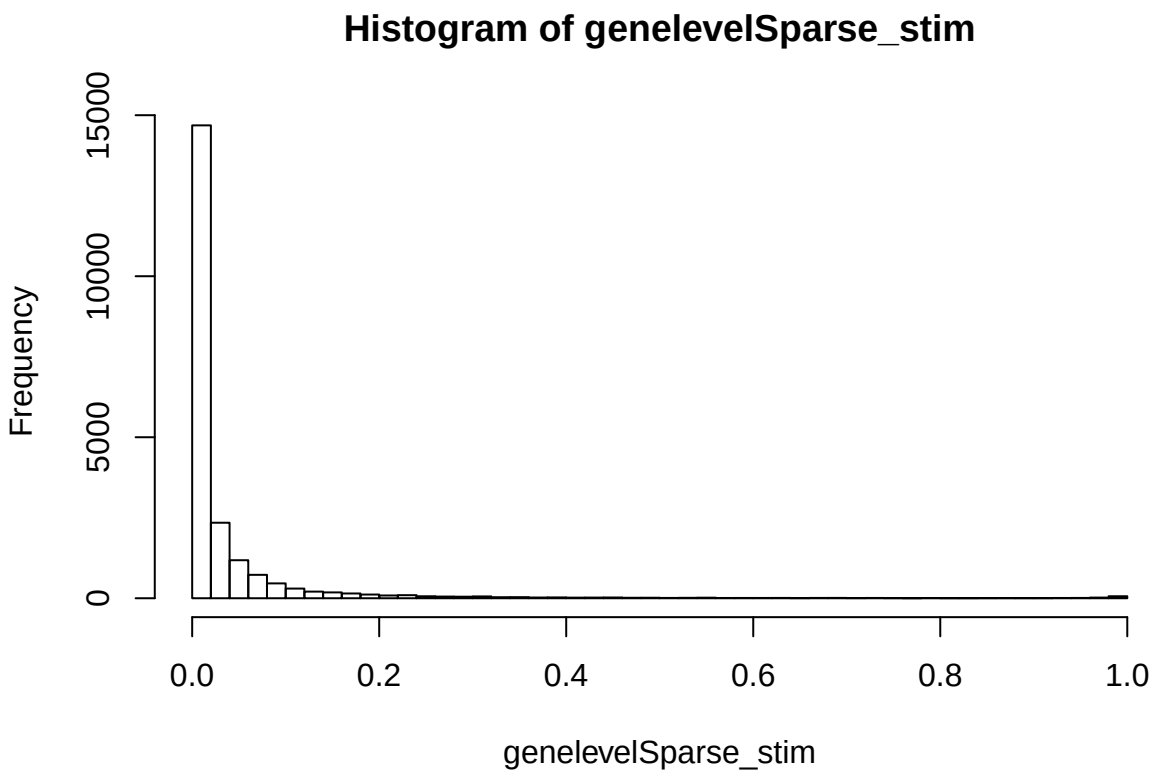

```
## get the pure counts
stim <- stim.data
colnames(stim) <- NULL
stim <- stim[, -1]
rownames(stim) <- NULL
stim <- as.matrix(stim)
head(stim[, 1:6])
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    0    0    0
## [2,]    0    0    0    0    0    0
## [3,]    0    0    0    0    0    0
## [4,]    0    0    0    0    0    0
## [5,]    0    0    0    0    0    0
## [6,]    0    0    0    0    0    0
```

```
genelevelSparse_stim <- apply(stim, 1, function(x) {
    sum(x != 0)/length(x)
})
sum(genelevelSparse_stim != 0)
```

```
## [1] 19910
```

```
hist(genelevelSparse_stim, freq = T, breaks = 50)
```




```
# Initialize the Seurat object with the raw (non-normalized
# data).  Keep all genes expressed in >= 3 cells (~0.1% of
# the data). Keep all cells with at least 200 detected genes
colnames(ctrl) <- as.vector(barcodes.ctrl[-1])
rownames(ctrl) <- genes.ctrl
head(ctrl[, 1:6])
```

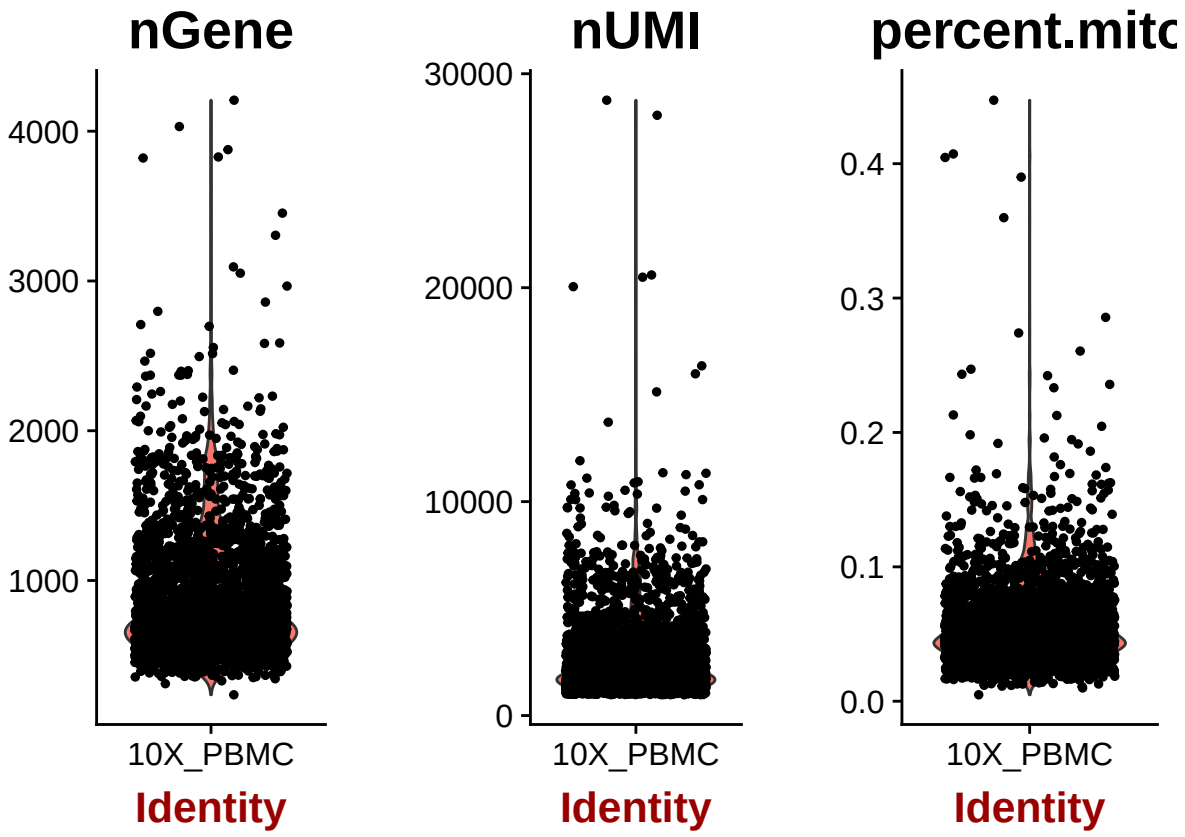
```
##          AAACCTGCACCTCGGA AAACCTGCATGCTAGT AAACCTGCATGGGAAC
## 5S_rRNA                 0                0                0
## 7SK                     0                0                0
## A1BG                    0                0                0
## A1BG-AS1                0                0                0
## A2M                     0                0                0
## A2M-AS1                 0                0                0
##          AAACCTGGTGATGTGG AAACGGGAGCGGCTTC AAACGGGCAATCGGTT
## 5S_rRNA                 0                0                0
## 7SK                     0                0                0
## A1BG                    0                0                0
## A1BG-AS1                0                0                0
## A2M                     0                0                0
## A2M-AS1                 0                0                0
```

```
pbmc <- CreateSeuratObject(raw.data = ctrl, min.cells = 3, min.genes = 200,
    project = "10X_PBMC")

slotNames(pbmc)
```

```
##  [1] "raw.data"     "data"         "scale.data"   "var.genes"
##  [5] "is.expr"      "ident"        "meta.data"    "project.name"
##  [9] "dr"           "assay"        "hvg.info"     "imputed"
```

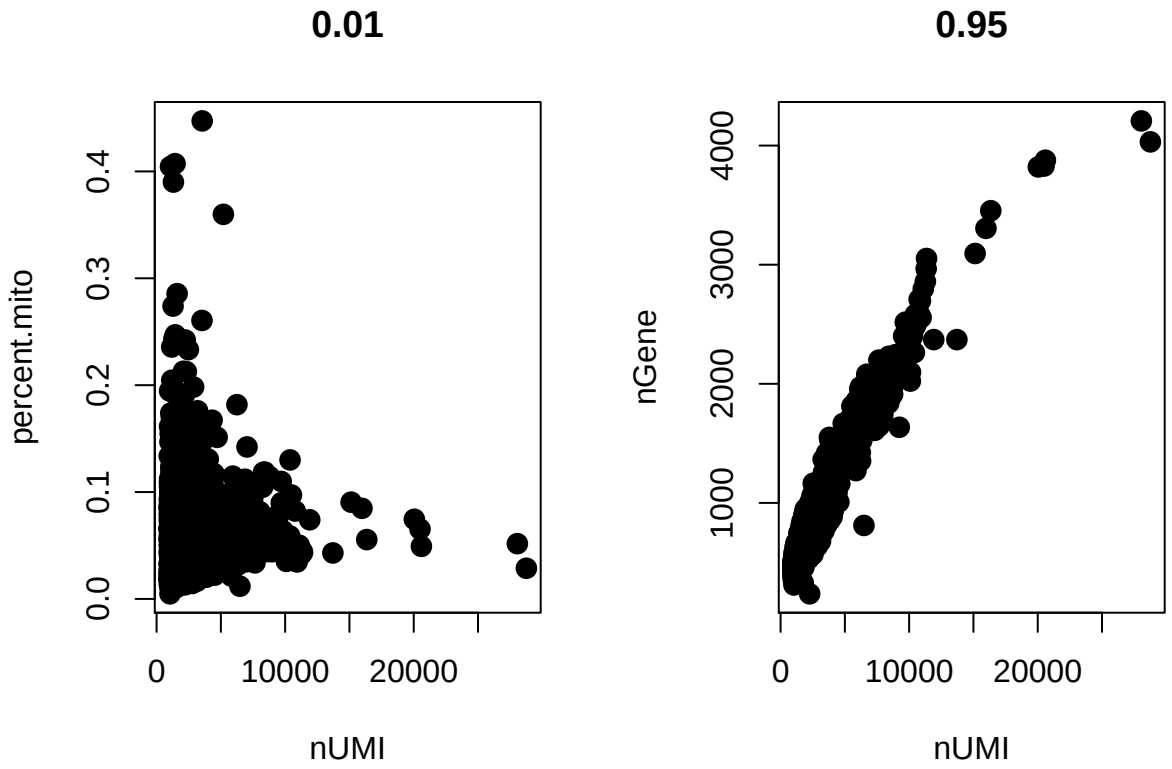
```
## [13] "cell.names"   "cluster.tree" "snn"          "calc.params"
## [17] "kmeans"       "spatial"      "misc"         "version"
```

```
# The number of genes and UMIs (nGene and nUMI) are
# automatically calculated for every object by Seurat.  For
# non-UMI data, nUMI represents the sum of the non-normalized
# values within a cell We calculate the percentage of
# mitochondrial genes here and store it in percent.mito using
# AddMetaData.  We use object@raw.data since this represents
# non-transformed and non-log-normalized counts The % of UMI
# mapping to MT-genes is a common scRNA-seq QC metric.
mito.genes <- grep(pattern = "^MT-", x = rownames(x = pbmc@data),
    value = TRUE)
percent.mito <- Matrix::colSums(pbmc@raw.data[mito.genes, ])/Matrix::colSums(pbmc@raw.data)

# AddMetaData adds columns to object@meta.data, and is a
# great place to stash QC stats
pbmc <- AddMetaData(object = pbmc, metadata = percent.mito, col.name = "percent.mito")
VlnPlot(object = pbmc, features.plot = c("nGene", "nUMI", "percent.mito"),
    nCol = 3)
```




```
# GenePlot is typically used to visualize gene-gene
# relationships, but can be used for anything calculated by
# the object, i.e. columns in object@meta.data, PC scores
# etc.  Since there is a rare subset of cells with an outlier
# level of high mitochondrial percentage and also low UMI
# content, we filter these as well
par(mfrow = c(1, 2))
```

```
GenePlot(object = pbmc, gene1 = "nUMI", gene2 = "percent.mito")
GenePlot(object = pbmc, gene1 = "nUMI", gene2 = "nGene")
```




```
# We filter out cells that have unique gene counts over 2,500
# or less than 200 Note that low.thresholds and
# high.thresholds are used to define a 'gate'.  -Inf and Inf
# should be used if you don't want a lower or upper
# threshold.
pbmc <- FilterCells(object = pbmc, subset.names = c("nGene",
    "percent.mito"), low.thresholds = c(200, -Inf), high.thresholds = c(2500,
    0.05))

# 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 gene expression measurements for each cell
# by the total expression, multiplies this by a scale factor
# (10,000 by default), and log-transforms the result.

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

1.1 Detection of variable genes across the single cells

Seurat calculates highly variable genes and focuses on these for downstream analysis. FindVariableGenes calculates the average expression and dispersion for each gene, places these genes into bins, and then calculates a z-score for dispersion within each bin. This helps control for the relationship between variability and average expression. This function is unchanged from (Macosko et al.), but new methods for variable gene expression identification are coming soon. We suggest that users set these parameters to mark visual outliers

on the dispersion plot, but the exact parameter settings may vary based on the data type, heterogeneity in the sample, and normalization strategy. The parameters here identify ~2,000 variable genes, and represent typical parameter settings for UMI data that is normalized to a total of 1e4 molecules.

1.2 Scaling the data and removing unwanted sources of variation

Your single cell dataset likely contains 'uninteresting' sources of variation. This could include not only technical noise, but batch effects, or even biological sources of variation (cell cycle stage). As suggested in Buettner et al, NBT, 2015, regressing these signals out of the analysis can improve downstream dimensionality reduction and clustering. To mitigate the effect of these signals, Seurat constructs linear models to predict gene expression based on user-defined variables. The scaled z-scored residuals of these models are stored in the scale.data slot, and are used for dimensionality reduction and clustering.

We can regress out cell-cell variation in gene expression driven by batch (if applicable), cell alignment rate (as provided by Drop-seq tools for Drop-seq data), the number of detected molecules, and mitochondrial gene expression. For cycling cells, we can also learn a 'cell-cycle' score (see example here) and regress this out as well. In this simple example here for post-mitotic blood cells, we regress on the number of detected molecules per cell as well as the percentage mitochondrial gene content.

Seurat v2.0 implements this regression as part of the data scaling process. Therefore, the RegressOut function has been deprecated, and replaced with the vars.to.regress argument in ScaleData.

1.3 PC selection –

identifying the true dimensionality of a dataset – is an important step for Seurat, but 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 7-10 as a cutoff. We followed the jackStraw here, admittedly buoyed by seeing the PCHeatmap returning interpretable signals (including canonical dendritic cell markers) throughout these PCs. Though the results are only subtly affected by small shifts in this cutoff (you can test below), we strongly suggest always explore the PCs they choose to include downstream.

1.4 Run Non-linear dimensional reduction (tSNE)

Seurat continues to use tSNE as a powerful tool to visualize and explore these datasets. While we no longer advise clustering directly on tSNE components, cells within the graph-based clusters determined above should co-localize on the tSNE plot. This is because the tSNE aims to place cells with similar local neighborhoods in high-dimensional space together in low-dimensional space. As input to the tSNE, we suggest using the same PCs as input to the clustering analysis, although computing the tSNE based on scaled gene expression is also supported using the genes.use argument.

1.5 Finding differentially expressed genes (cluster biomarkers)

Seurat can help you find markers that define clusters via differential expression. By default, it identifes 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 gene to be detected at a minimum percentage in either of the two groups of cells, and the thresh.test argument requires a gene 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 genes 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 significiant and the most highly differentially expressed genes will likely still rise to the top.

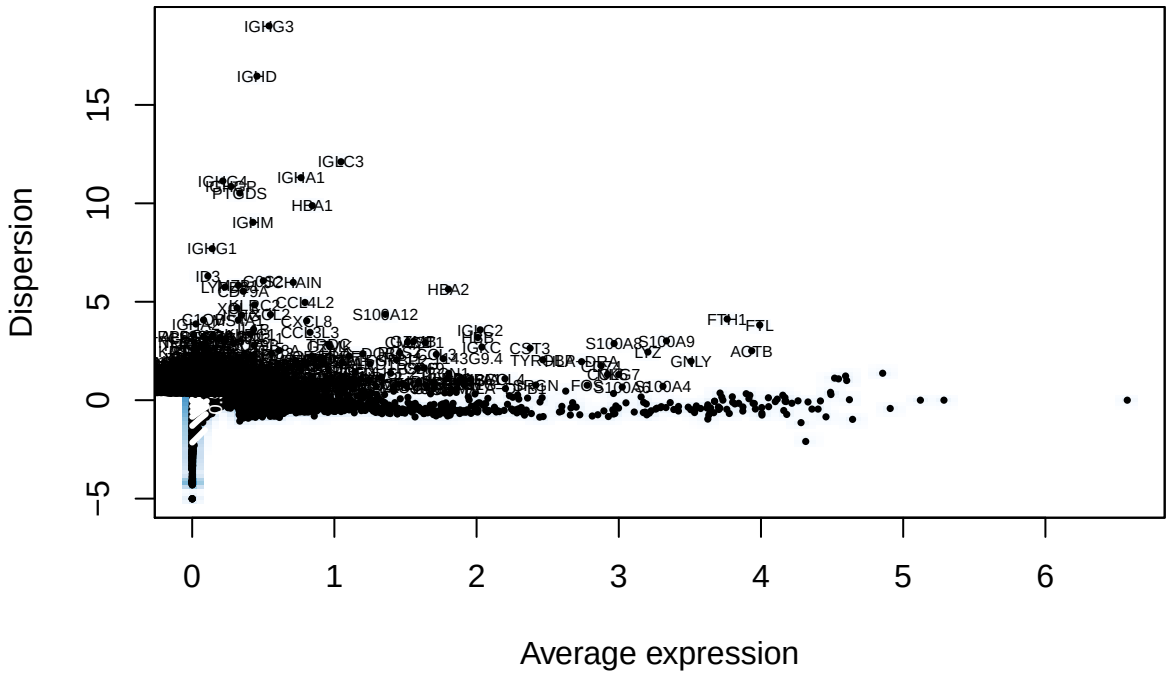
```
pbmc <- FindVariableGenes(object = pbmc, mean.function = ExpMean,
    dispersion.function = LogVMR, x.low.cutoff = 0.0125, x.high.cutoff = 4,
    y.cutoff = 0.5)
length(x = pbmc@var.genes)
```

```
## [1] 2800
```

```
pbmc <- ScaleData(object = pbmc, vars.to.regress = c("nUMI",
    "percent.mito"))
```

```
##
## Time Elapsed:  13.1332397460938 secs
```

```
pbmc <- FindVariableGenes(object = pbmc, mean.function = ExpMean,
    dispersion.function = LogVMR, x.low.cutoff = 0.0125, x.high.cutoff = 4,
    y.cutoff = 0.5)
```




```
length(x = pbmc@var.genes)
```

```
## [1] 2800
```

```
paste("nUMI, percent.mito, these variables won't change the expression spectrum")
```

```
## [1] "nUMI, percent.mito, these variables won't change the expression spectrum"
```

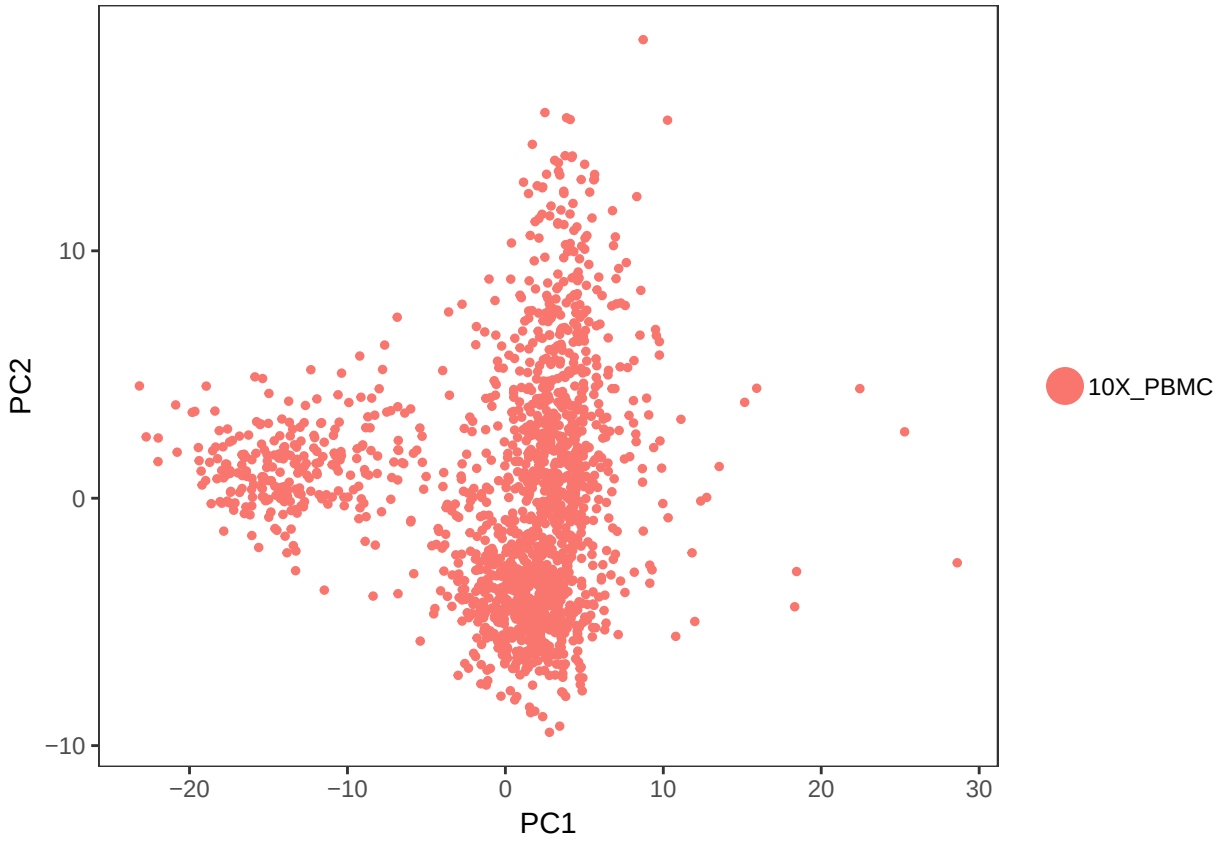
```
# Perform linear dimensional reduction Next we perform PCA on
# the scaled data. By default, the genes in object@var.genes
# are used as input, but can be defined using pc.genes. We
```

```
# have typically found that running dimensionality reduction
# on highly variable genes can improve performance. However,
# with UMI data - particularly after regressing out technical
# variables, we often see that PCA returns similar (albeit
# slower) results when run on much larger subsets of genes,
# including the whole transcriptome.

pbmc <- RunPCA(object = pbmc, pc.genes = pbmc@var.genes, do.print = TRUE,
    pcs.print = 1:10, genes.print = 5)
```

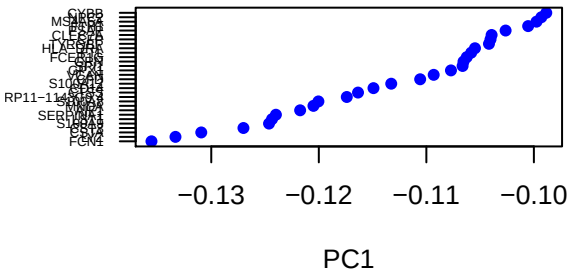
```
## [1] "PC1"
## [1] "FCN1"   "LYZ"    "CSTA"   "CST3"   "S100A9"
## [1] ""
## [1] "CCL5" "NKG7" "CST7" "GZMA" "GNLY"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "IL7R"  "NOSIP" "CD27"  "CCR7"  "CD79A"
## [1] ""
## [1] "NKG7" "GNLY" "CST7" "CCL4" "CCL5"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "SERPINF1" "LILRA4"   "TCF4"     "PLD4"     "JCHAIN"
## [1] ""
## [1] "IL7R"   "NOSIP"  "TRBC2"  "S100A9" "FCN1"
## [1] ""
## [1] ""
## [1] "PC4"
## [1] "MS4A1"  "CD79A"  "CD79B"  "VPREB3" "BANK1"
## [1] ""
## [1] "LILRA4"   "TPM2"     "SERPINF1" "PTCRA"    "SCT"
## [1] ""
## [1] ""
## [1] "PC5"
## [1] "GZMH"   "TRGC2"  "CCL5"   "S100A4" "FGFBP2"
## [1] ""
## [1] "XCL1"   "KLRC1"  "FOS"    "CD69"   "NFKBIA"
## [1] ""
## [1] ""
## [1] "PC6"
## [1] "CD79A"    "TNFRSF17" "IGKC"     "MZB1"     "MS4A1"
## [1] ""
## [1] "FCER1A"  "CD1C"    "CLEC10A" "PKIB"    "ENHO"
## [1] ""
## [1] ""
## [1] "PC7"
## [1] "LYPD2"  "FCGR3A" "C1QA"   "TCF7L2" "IFITM3"
## [1] ""
## [1] "VCAN"    "S100A12" "S100A8"  "CD14"    "CSF3R"
## [1] ""
## [1] ""
## [1] "PC8"
## [1] "SPON2"  "FGFBP2" "GZMB"   "AKR1C3" "CD1C"
```

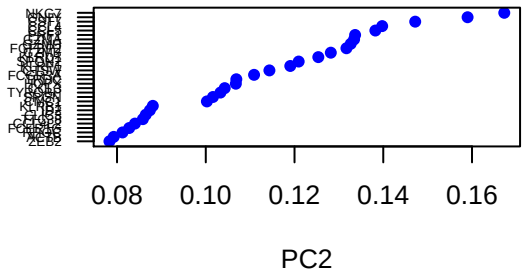
```
## [1] ""
## [1] "GZMK"  "XCL1"  "DUSP2" "RGS1"  "CDCA7"
## [1] ""
## [1] ""
## [1] "PC9"
## [1] "KLF6"   "JUN"    "S100A4" "CCL4"   "CD69"
## [1] ""
## [1] "CMC1"  "KLRC2" "TXK"   "TASP1" "XCL2"
## [1] ""
## [1] ""
## [1] "PC10"
## [1] "KIAA0101" "RGS1"     "CDCA7"    "CD27"     "TOX"
## [1] ""
## [1] "XCL1"   "SPTSSB" "GNLY"   "KLRC1"  "TRGC2"
## [1] ""
## [1] ""
```

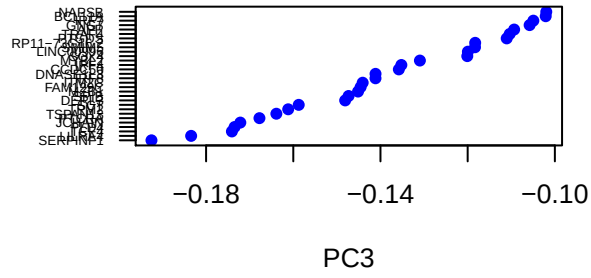
```
PCAPlot(object = pbmc, dim.1 = 1, dim.2 = 2)
```

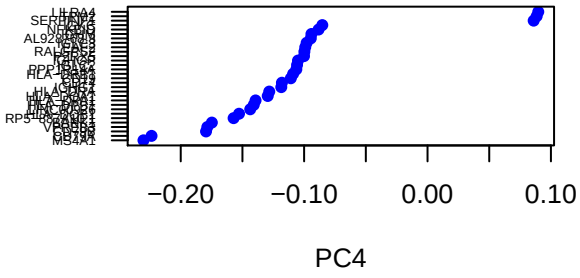



```
VizPCA(object = pbmc, pcs.use = 1:4)
```

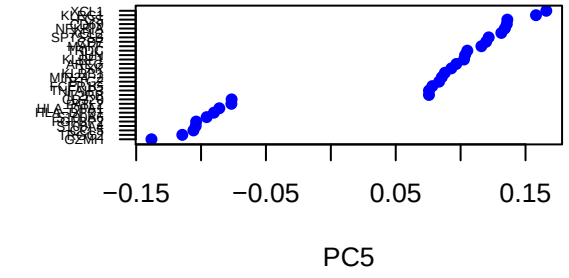


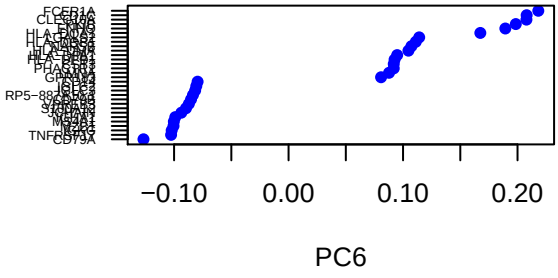


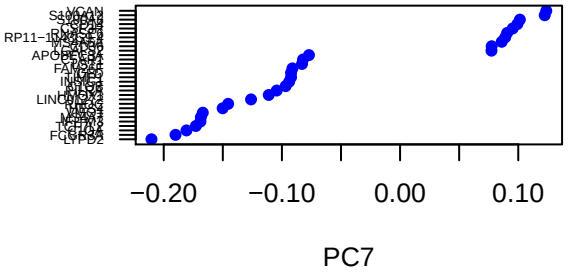


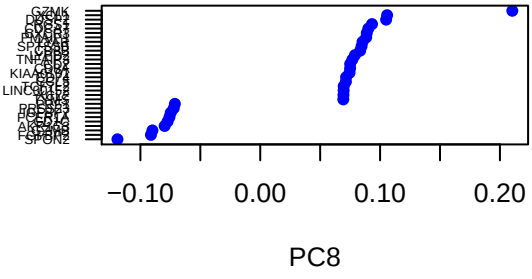



```
VizPCA(object = pbmc, pcs.use = 5:8)
```

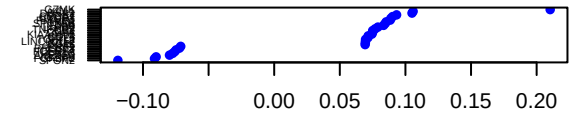




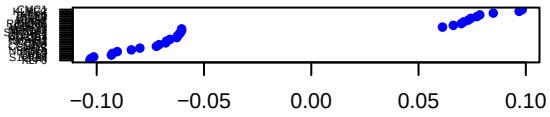




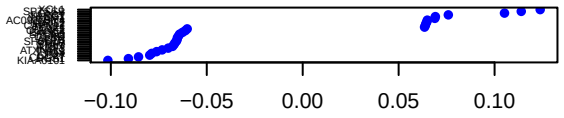

```
VizPCA(object = pbmc, pcs.use = 8:12)
```



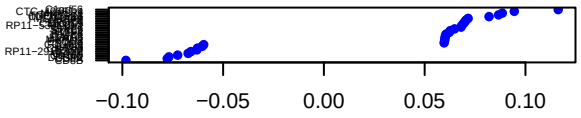
PC8



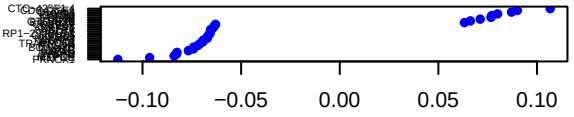
PC9



PC10

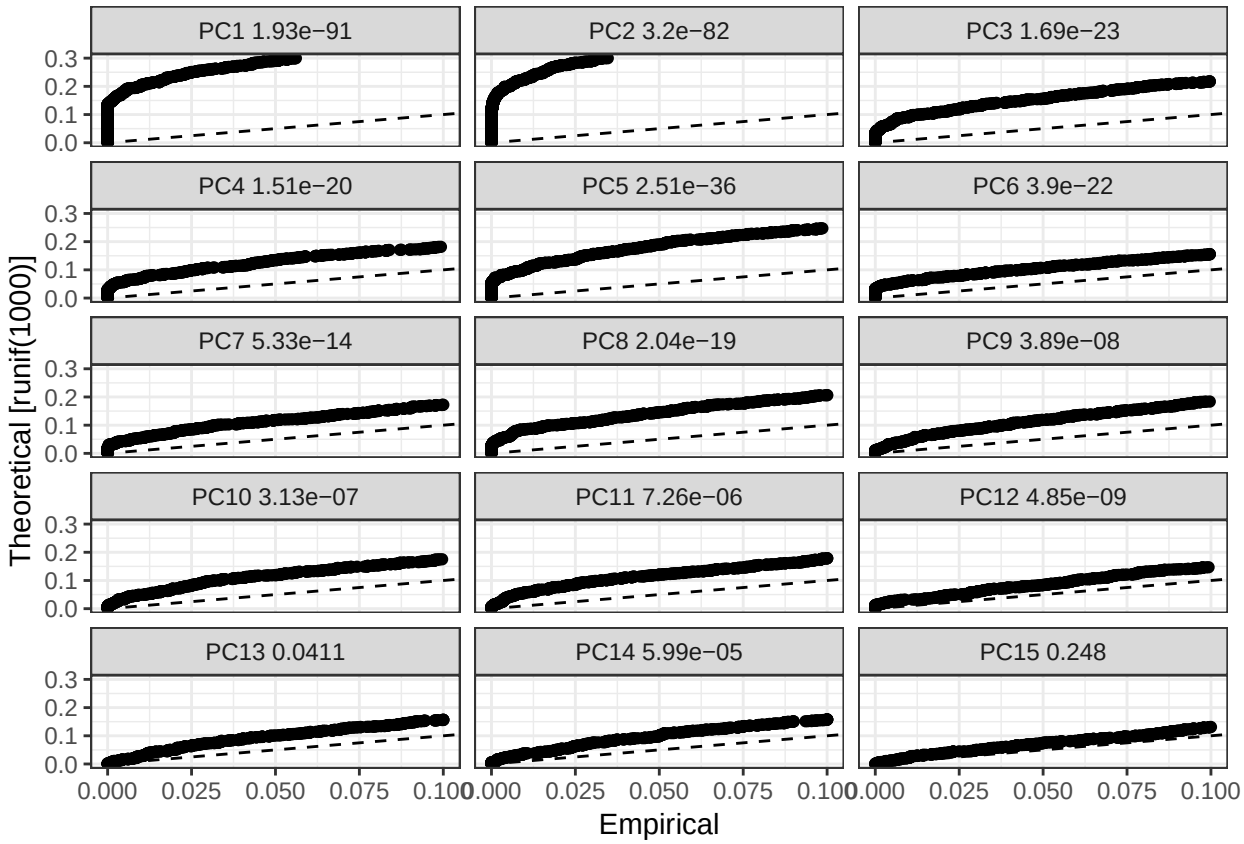


PC11



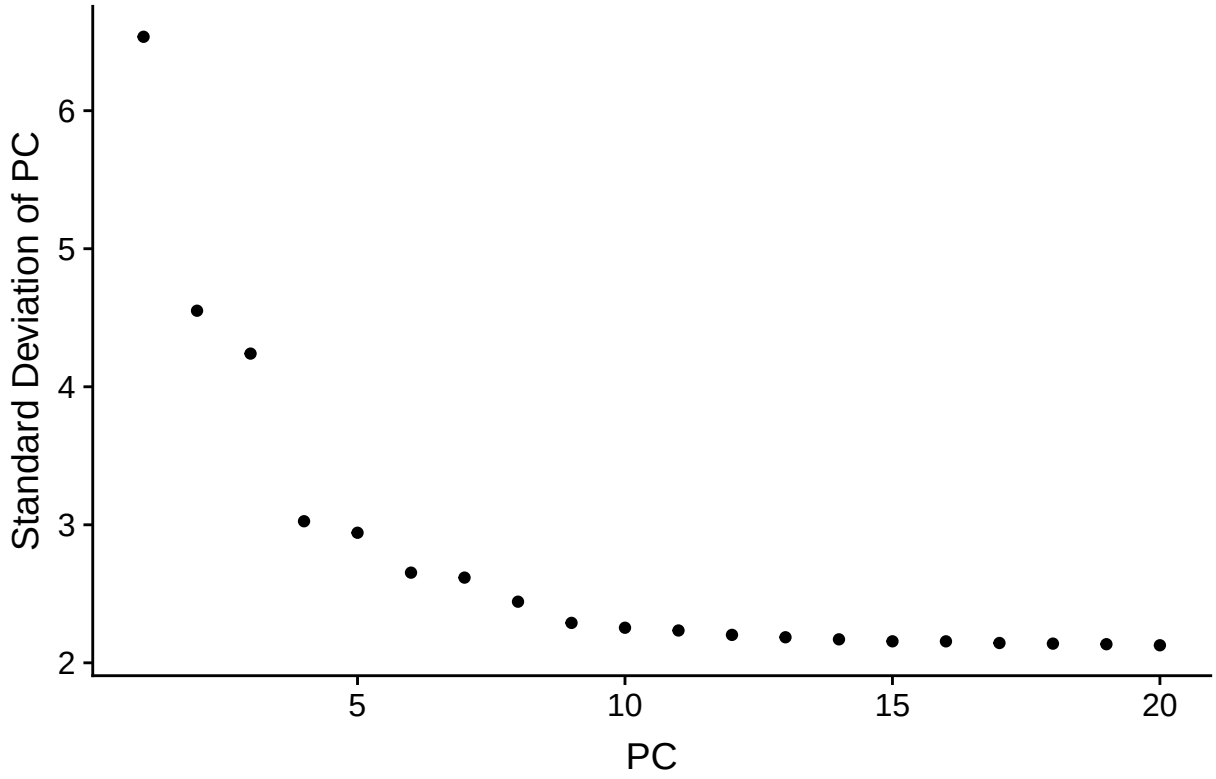
PC12

```
# ProjectPCA scores each gene in the dataset (including genes
# not included in the PCA) based on their correlation with
# the calculated components.  Though we don't use this
# further here, it can be used to identify markers that are
# strongly correlated with cellular heterogeneity, but may
# not have passed through variable gene selection.  The
# results of the projected PCA can be explored by setting
# use.full=T in the functions above
pbmc <- ProjectPCA(object = pbmc, do.print = FALSE)

# NOTE: This process can take a long time for big datasets,
# comment out for expediency.  More approximate techniques
# such as those implemented in PCElbowPlot() can be used to
# reduce computation time
pbmc <- JackStraw(object = pbmc, num.replicate = 100, display.progress = FALSE)
JackStrawPlot(object = pbmc, PCs = 1:15)
```

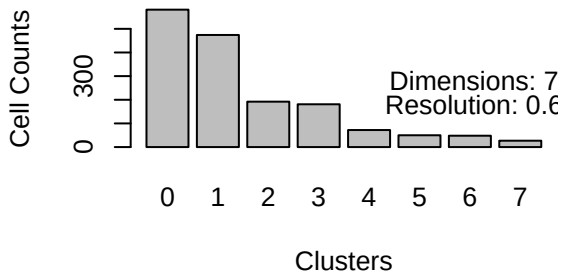


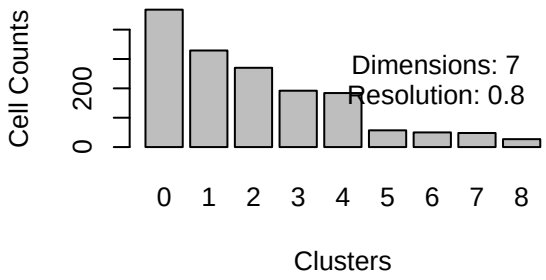

```
## An object of class seurat in project 10X_PBMC
##  16104 genes across 1625 samples.
```

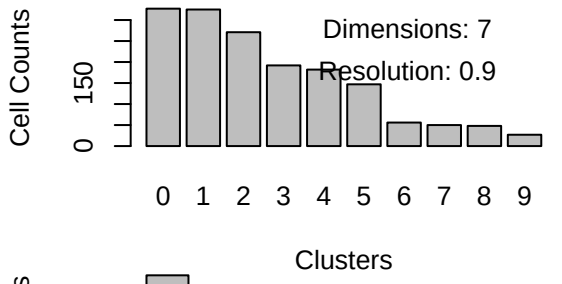
```
PCElbowPlot(object = pbmc)
```

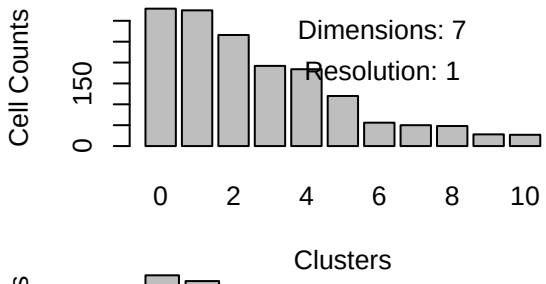


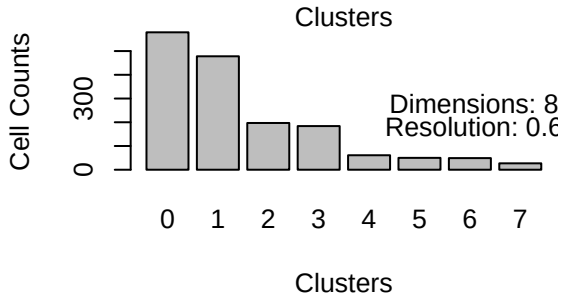

```
# save.SNN = T saves the SNN so that the clustering algorithm
# can be rerun using the same graph but with a different
# resolution value (see docs for full details)
par(mfrow = c(2, 2))
for (i in 7:11) {
    for (j in c(0.6, 0.8, 0.9, 1)) {
        pbmc <- FindClusters(object = pbmc, reduction.type = "pca",
            dims.use = 1:i, resolution = j, print.output = 0,
            save.SNN = TRUE)
        barplot(table(pbmc@ident), xlab = "Clusters", ylab = "Cell Counts")
        text(8, 280, labels = paste("Dimensions: ", i, sep = ""))
        text(8, 180, labels = paste(sep = "", "Resolution: ",
            j))
    }
}
```

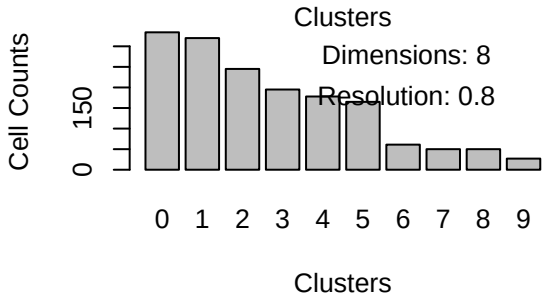

Cell Counts
0
300
Dimensions: 7
Resolution: 0.6
0 1 2 3 4 5 6 7
Clusters

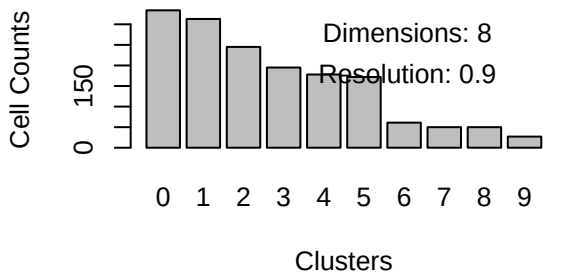

Cell Counts
0
200
Dimensions: 7
Resolution: 0.8
0 1 2 3 4 5 6 7 8
Clusters

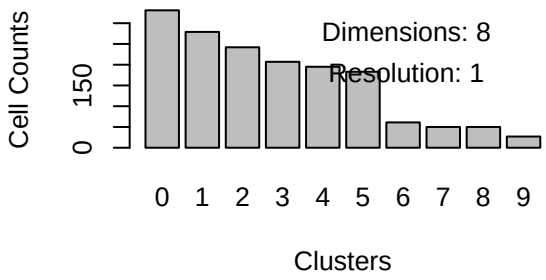

Cell Counts
0
150
Dimensions: 7
Resolution: 0.9
0 1 2 3 4 5 6 7 8 9
Clusters

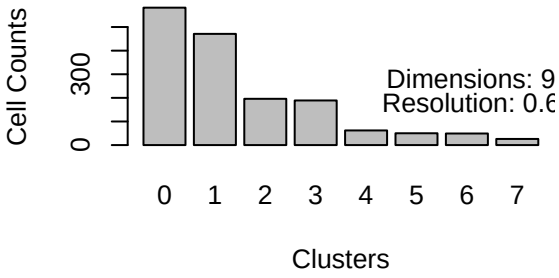

Cell Counts
0
150
Dimensions: 7
Resolution: 1
0 2 4 6 8 10
Clusters

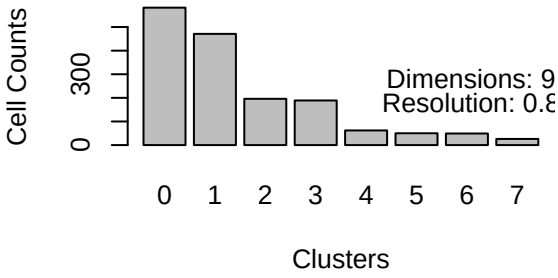

Cell Counts
0
300
Dimensions: 8
Resolution: 0.6
0 1 2 3 4 5 6 7
Clusters

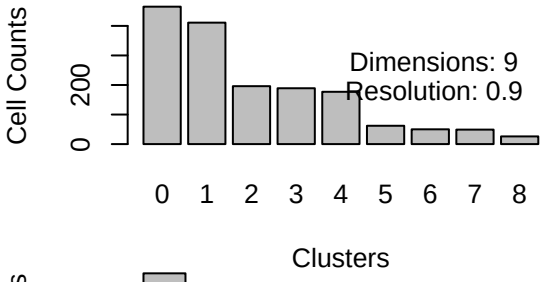

Cell Counts
0
150
Clusters
Dimensions: 8
Resolution: 0.8
0 1 2 3 4 5 6 7 8 9
Clusters

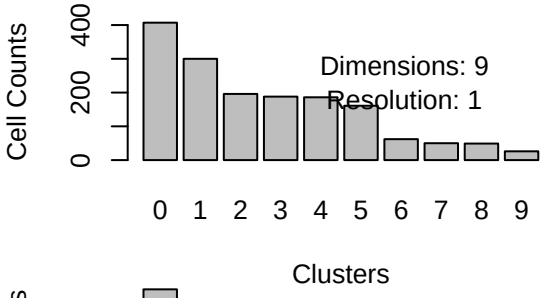

Cell Counts
0
150
Dimensions: 8
Resolution: 0.9
0 1 2 3 4 5 6 7 8 9
Clusters

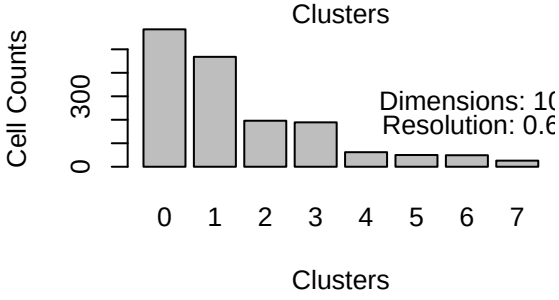

Cell Counts
0
150
Dimensions: 8
Resolution: 1
0 1 2 3 4 5 6 7 8 9
Clusters

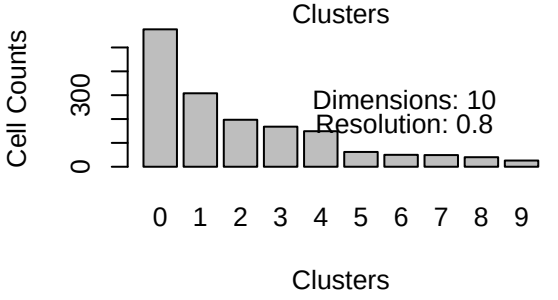

Cell Counts
0
300
Dimensions: 9
Resolution: 0.6
0 1 2 3 4 5 6 7
Clusters

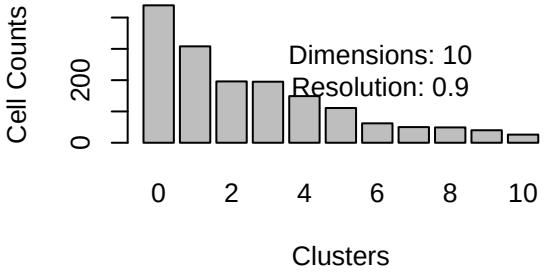

Cell Counts
0
300
Dimensions: 9
Resolution: 0.8
0 1 2 3 4 5 6 7
Clusters

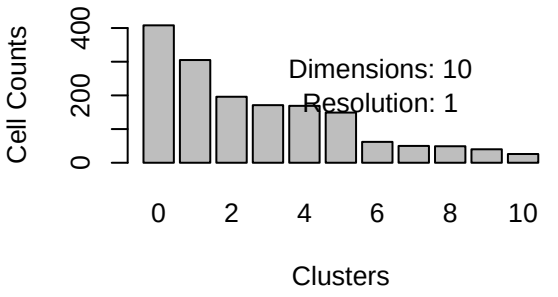

Cell Counts
0
200
Dimensions: 9
Resolution: 0.9
0 1 2 3 4 5 6 7 8
Clusters

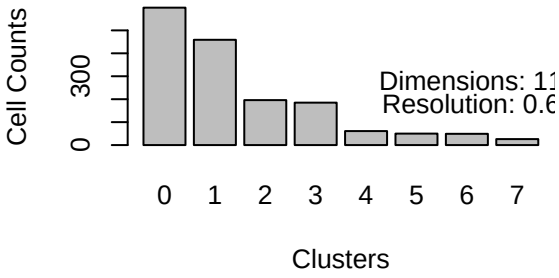

Cell Counts
0
200
400
Dimensions: 9
Resolution: 1
0 1 2 3 4 5 6 7 8 9
Clusters

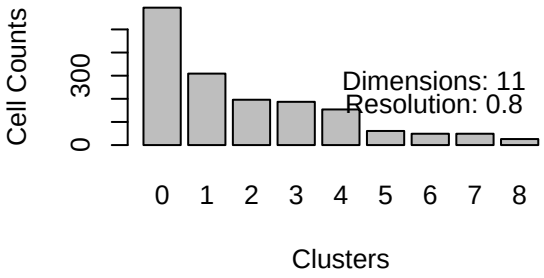

Cell Counts
0
300
Dimensions: 10
Resolution: 0.6
0 1 2 3 4 5 6 7
Clusters

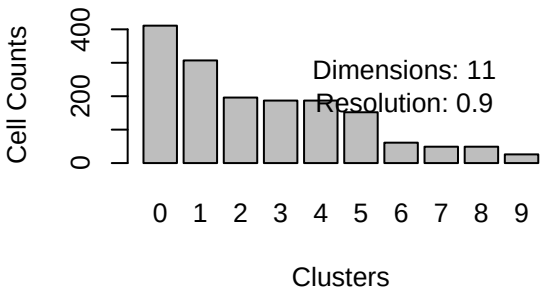

Cell Counts
0
300
Dimensions: 10
Resolution: 0.8
0 1 2 3 4 5 6 7 8 9
Clusters

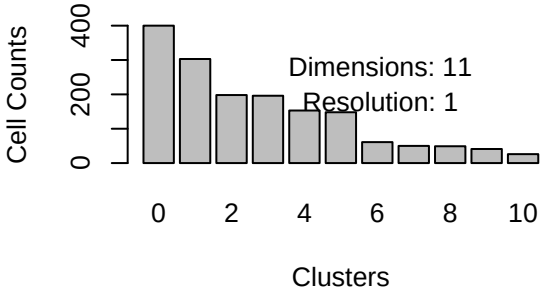

Cell Counts
0
200
Dimensions: 10
Resolution: 0.9
0 2 4 6 8 10
Clusters


Cell Counts
0
200
400
Dimensions: 10
Resolution: 1
0 2 4 6 8 10
Clusters










```
paste("Dimension: 9, resolution 0.9 was chosen before t-SNE")
```

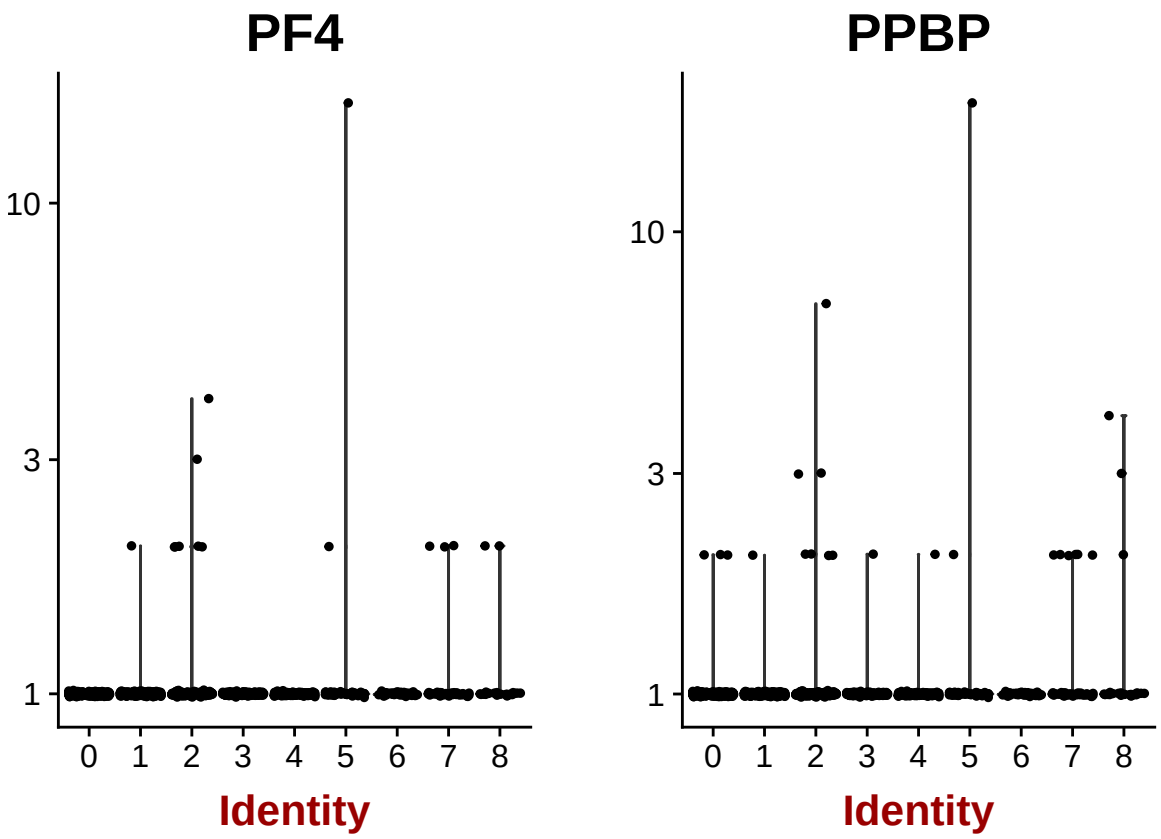
```
## [1] "Dimension: 9, resolution 0.9 was chosen before t-SNE"
```

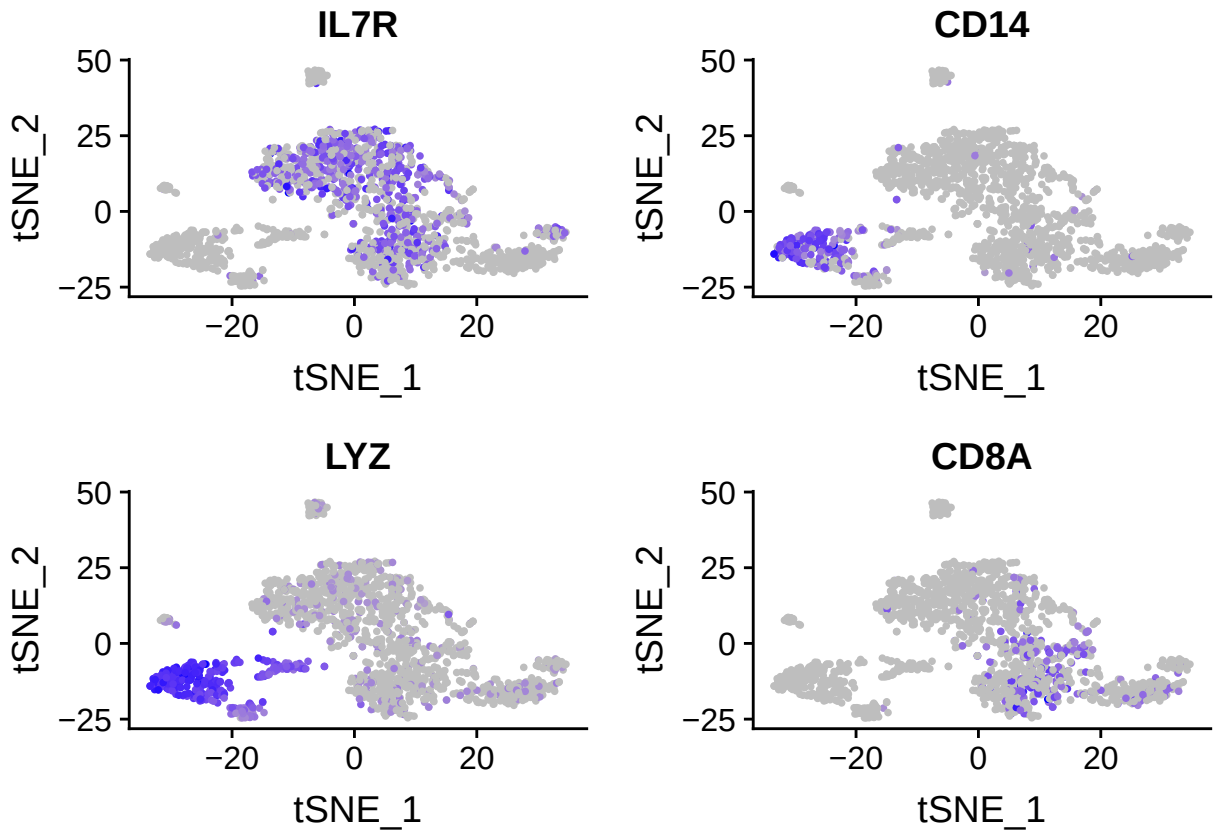
```
## rerun to choose clusters setting
pbmc <- FindClusters(object = pbmc, reduction.type = "pca", dims.use = 1:9,
    resolution = 0.9, print.output = 0, save.SNN = TRUE)
barplot(table(pbmc@ident), xlab = "Clusters", ylab = "Cell Counts")
pbmc <- RunTSNE(object = pbmc, dims.use = 1:9, do.fast = TRUE)

# note that you can set do.label=T to help label individual
# clusters
plot1 <- TSNEPlot(object = pbmc, do.label = T)

## 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.

saveRDS(pbmc, file = "~/Documents/Stats/7659Genetics/Proj/pbmc_GSM3374613.rds")

# find all markers of cluster 1
for (i in 0:8) {
    cluster1.markers <- FindMarkers(object = pbmc, ident.1 = i,
        min.pct = 0.25)
    print(x = head(x = cluster1.markers, n = 5))
    if ("PF4" %in% rownames(cluster1.markers)) {
        print(which(rownames(cluster1.markers %in% "PF4")))
    } else {
        print("No")
    }
}
```

```
##              p_val avg_logFC pct.1 pct.2     p_val_adj
## CCL5 1.494431e-159  1.539566 0.946 0.243 2.406632e-155
## NKG7 9.364109e-123  1.190562 0.925 0.255 1.507996e-118
## GZMH 1.587000e-110  1.575303 0.675 0.134 2.555705e-106
## GZMA 2.669276e-110  1.207880 0.785 0.183 4.298602e-106
## B2M  2.736695e-107  0.463275 1.000 1.000 4.407174e-103
## [1] "No"
##               p_val avg_logFC pct.1 pct.2     p_val_adj
## RPL13 1.078509e-122 0.5344799 1.000 0.998 1.736830e-118
## RPL34 2.060265e-115 0.4955476 1.000 1.000 3.317850e-111
## RPL32 3.679195e-111 0.5014030 1.000 1.000 5.924975e-107
## LTB   4.674150e-105 1.2151749 0.922 0.353 7.527252e-101
## RPS6  4.544892e-100 0.4853576 1.000 0.998  7.319095e-96
## [1] "No"
##                 p_val avg_logFC pct.1 pct.2     p_val_adj
## CD14    1.653946e-235  2.317629 0.842 0.028 2.663514e-231
## VCAN    3.005582e-230  2.161764 0.791 0.020 4.840189e-226
## S100A12 1.038674e-225  3.027795 0.857 0.039 1.672681e-221
## CSTA    8.939605e-208  2.230067 0.934 0.078 1.439634e-203
## FCN1    7.651415e-204  2.787449 0.995 0.115 1.232184e-199
## [1] "No"
##               p_val avg_logFC pct.1 pct.2     p_val_adj
## TRDC  1.877384e-151  1.807467 0.704 0.043 3.023339e-147
## KLRF1 9.547894e-150  1.862712 0.704 0.048 1.537593e-145
## GNLY  9.403107e-108  1.892360 0.984 0.352 1.514276e-103
## GZMB  3.126872e-104  1.804710 0.788 0.134 5.035514e-100
## SPON2  3.602524e-98  1.752347 0.566 0.059  5.801505e-94
## [1] "No"
##               p_val  avg_logFC pct.1 pct.2    p_val_adj
## RPL34  4.454658e-29  0.3524328 1.000 1.000 7.173782e-25
## MT-CO1 1.648899e-27 -0.4290404 0.989 1.000 2.655386e-23
## RPL11  2.142575e-27  0.3350659 1.000 0.999 3.450403e-23
## RPL13  3.457974e-27  0.3353229 1.000 0.999 5.568721e-23
## RPL32  1.586025e-26  0.3205479 1.000 1.000 2.554135e-22
## [1] "No"
##                 p_val avg_logFC pct.1 pct.2     p_val_adj
## FCER1A  2.224141e-209  2.038660 0.839 0.013 3.581757e-205
## CD1C    2.812339e-199  1.762610 0.742 0.009 4.528991e-195
## CLEC10A 3.726833e-182  1.753328 0.742 0.012 6.001692e-178
## PKIB    3.957983e-180  1.052013 0.613 0.004 6.373936e-176
## ENHO    6.952633e-148  1.365314 0.581 0.008 1.119652e-143
## [1] "No"
##                      p_val avg_logFC pct.1 pct.2     p_val_adj
## CD79A        9.955889e-223  2.567627  0.86 0.010 1.603296e-218
## MS4A1        2.020169e-215  2.490556  0.78 0.007 3.253280e-211
## VPREB3       5.148732e-129  1.530683  0.42 0.002 8.291517e-125
## BANK1        1.484239e-118  1.386668  0.48 0.006 2.390219e-114
## RP5-887A10.1 8.734277e-117  1.699321  0.40 0.003 1.406568e-112
## [1] "No"
##                        p_val avg_logFC pct.1 pct.2     p_val_adj
## LYPD2          2.765850e-126 2.2374767 0.429 0.003 4.454125e-122
## C1QA           1.211482e-120 1.5844144 0.490 0.006 1.950971e-116
## CDKN1C         1.659843e-120 0.9244981 0.429 0.003 2.673011e-116
## RP11-1008C21.1 6.496086e-108 0.8250118 0.327 0.001 1.046130e-103
```

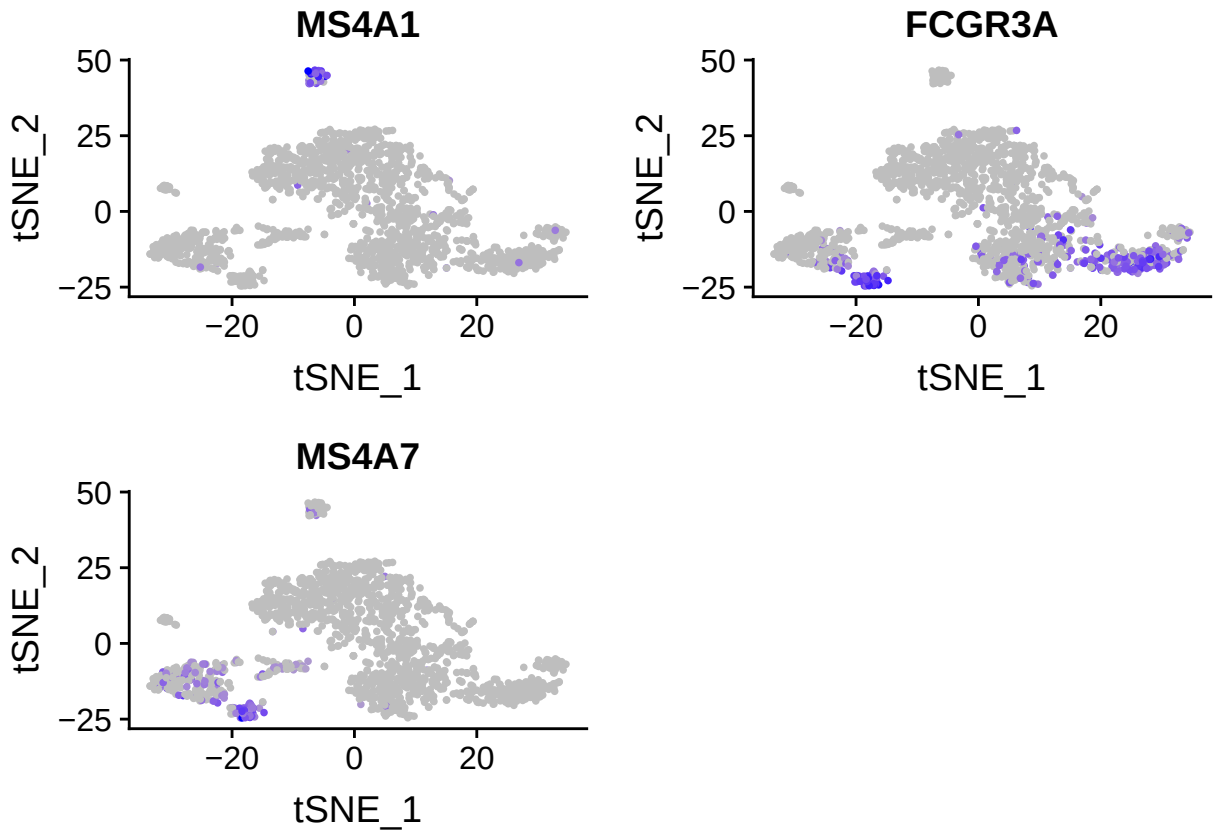
```
## TCF7L2         3.161399e-100 1.1180371 0.531 0.013  5.091117e-96
## [1] "No"
##                  p_val avg_logFC pct.1 pct.2     p_val_adj
## LILRA4   2.840187e-250  2.108120 0.808 0.002 4.573838e-246
## TPM2     1.093019e-220  1.593401 0.692 0.001 1.760197e-216
## PTCRA    4.639778e-218  1.809285 0.769 0.003 7.471899e-214
## SCT      4.974179e-207  1.902493 0.654 0.001 8.010417e-203
## SERPINF1 1.738504e-189  2.503832 0.923 0.011 2.799687e-185
## [1] "No"
```

```r
# you can plot raw UMI counts as well
VlnPlot(object = pbmc, features.plot = c("PF4", "PPBP"), use.raw = TRUE,
    y.log = TRUE)

FeaturePlot(object = pbmc, features.plot = c("IL7R", "CD14",
    "LYZ", "CD8A"), cols.use = c("grey", "blue"), reduction.use = "tsne")
```

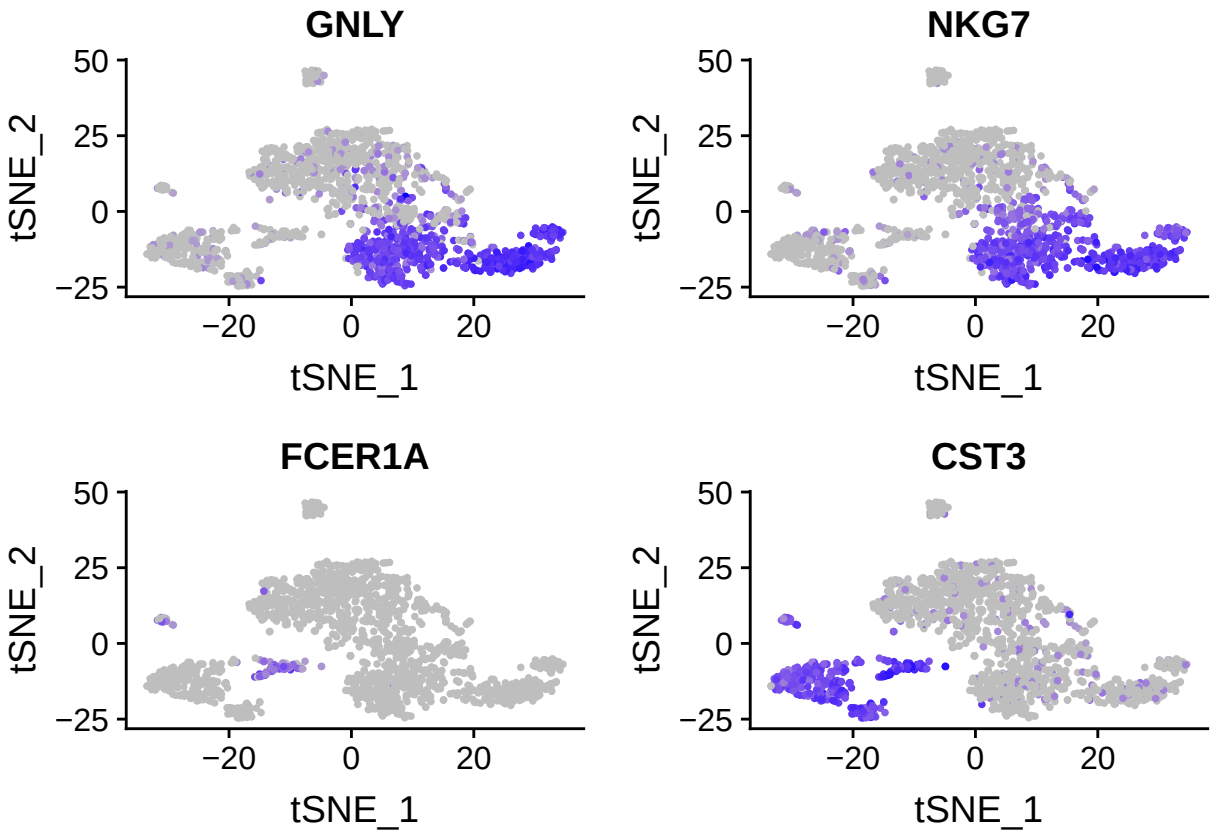




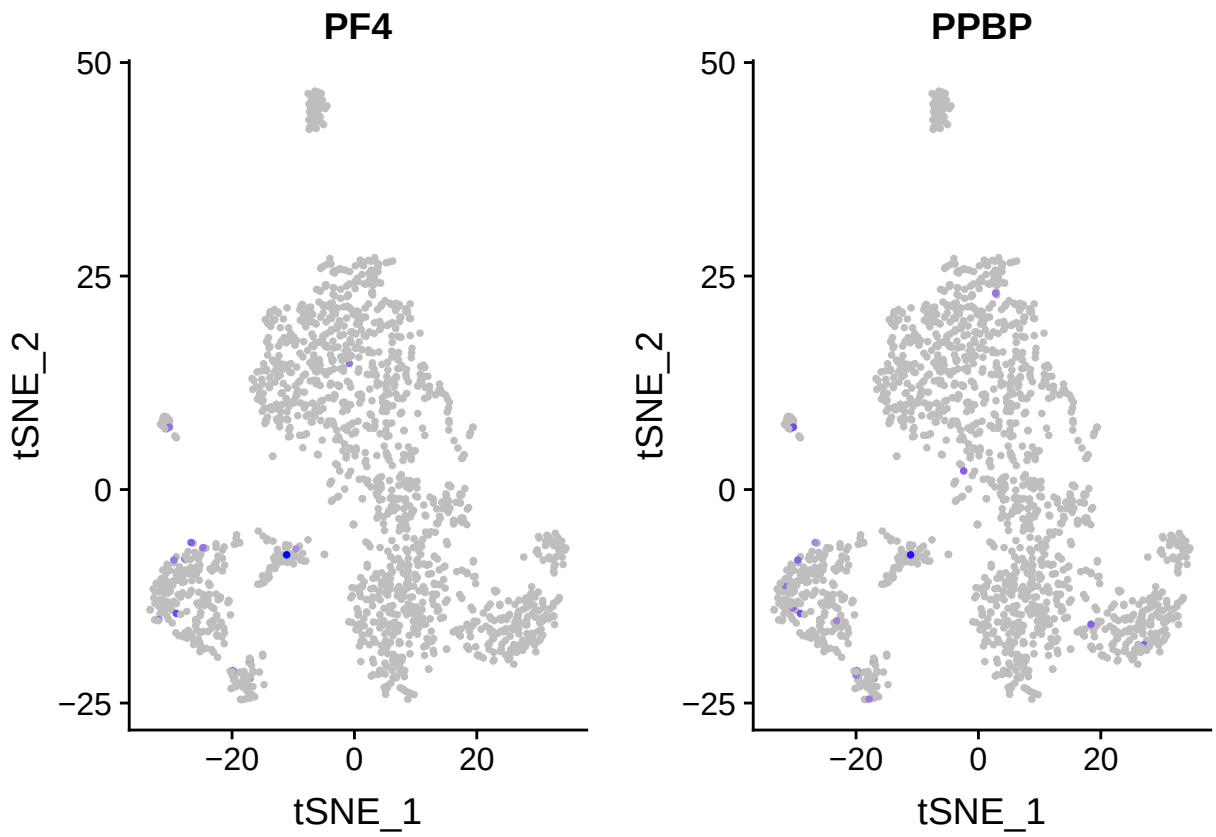

```
FeaturePlot(object = pbmc, features.plot = c("MS4A1", "FCGR3A",
    "MS4A7"), cols.use = c("grey", "blue"), reduction.use = "tsne")
```



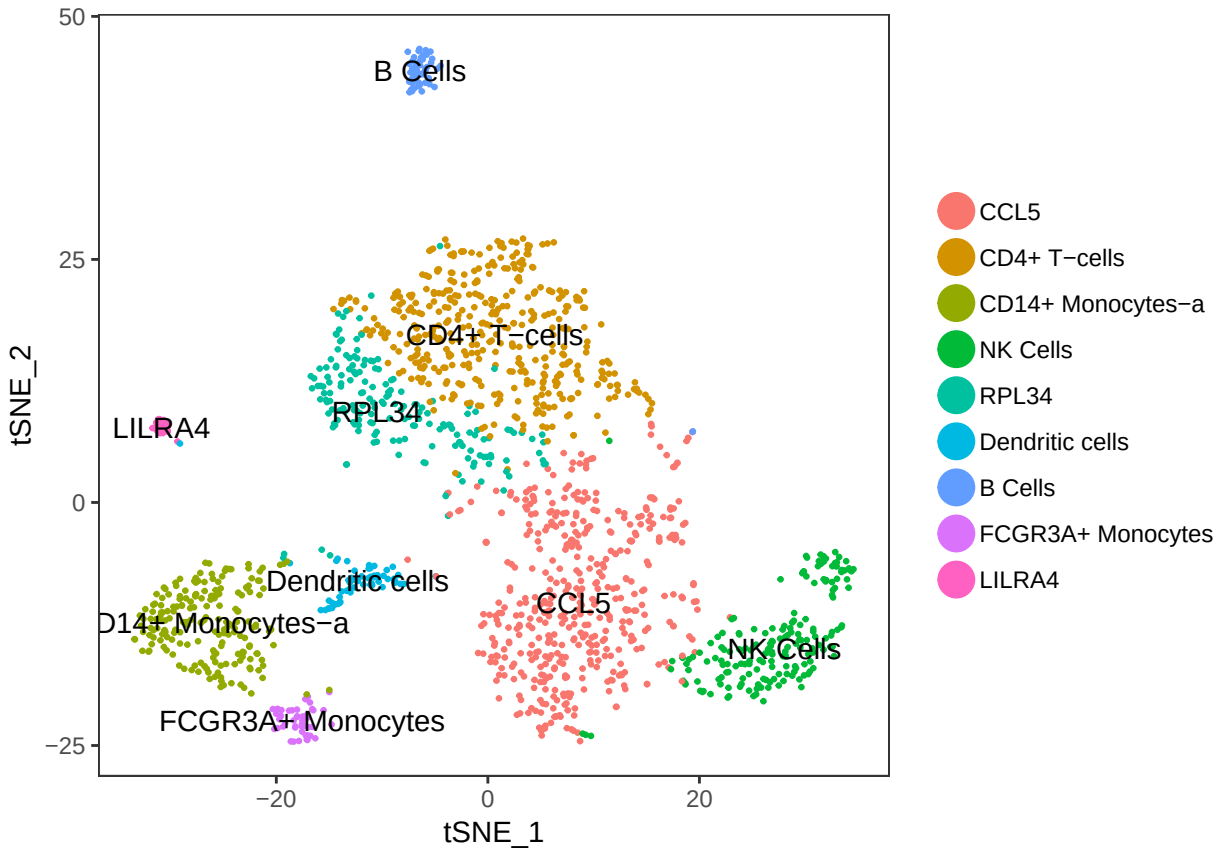

```
FeaturePlot(object = pbmc, features.plot = c("GNLY", "NKG7",
    "FCER1A", "CST3"), cols.use = c("grey", "blue"), reduction.use = "tsne")
```



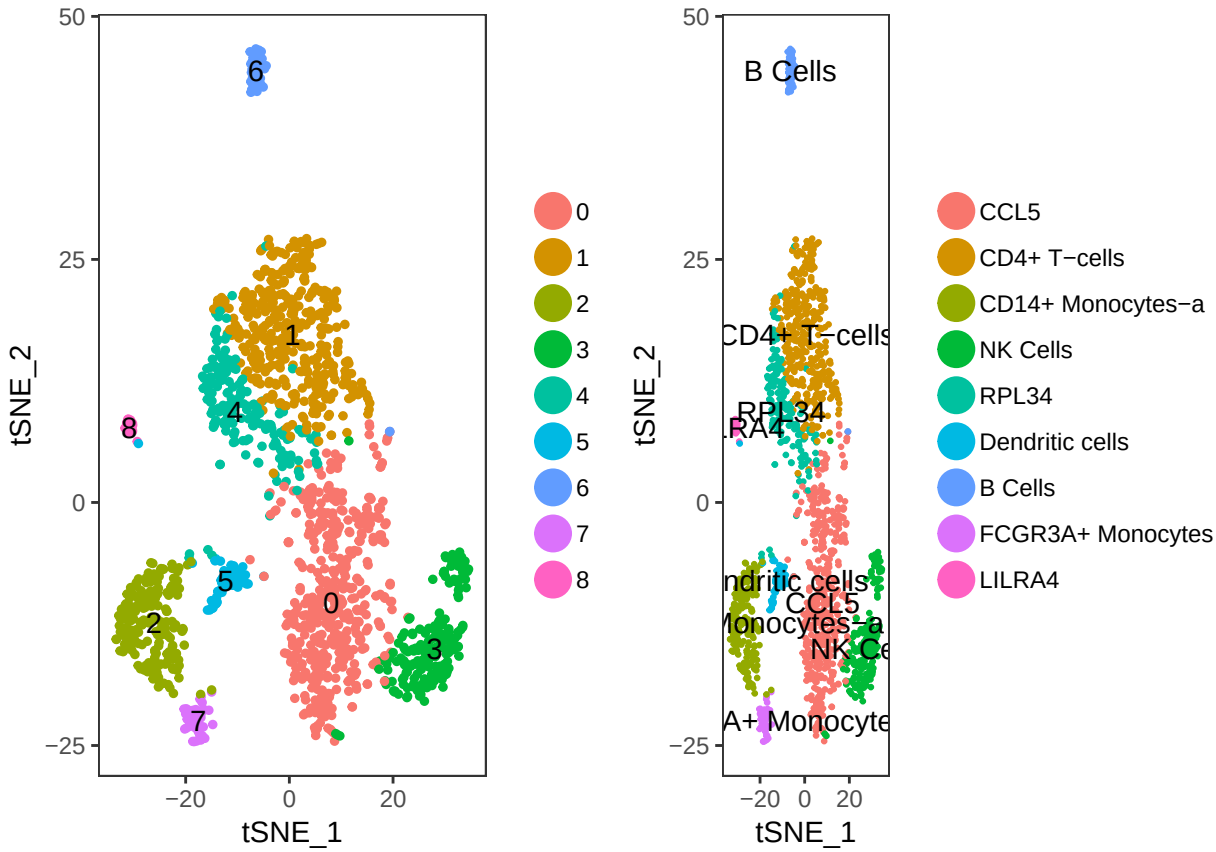

```
FeaturePlot(object = pbmc, features.plot = c("PF4", "PPBP"),
    cols.use = c("grey", "blue"), reduction.use = "tsne")
```




```
current.cluster.ids <- c(0, 1, 2, 3, 4, 5, 6, 7, 8)
new.cluster.ids <- c("CCL5", "CD4+ T-cells", "CD14+ Monocytes-a",
    "NK Cells", "RPL34", "Dendritic cells", "B Cells", "FCGR3A+ Monocytes",
    "LILRA4")
pbmc@ident <- plyr::mapvalues(x = pbmc@ident, from = current.cluster.ids,
    to = new.cluster.ids)
plot2 <- TSNEPlot(object = pbmc, do.label = TRUE, pt.size = 0.5)
```




```
## save name First lets stash our identities for later
pbmc <- StashIdent(object = pbmc, save.name = "ClusterNames_0.9_9")

plot_grid(plot1, plot2)
```




```
saveRDS(pbmc, file = "~/Documents/Stats/7659Genetics/Proj/pbmc_GSM3374613_final.rds")
```

1.6 resample

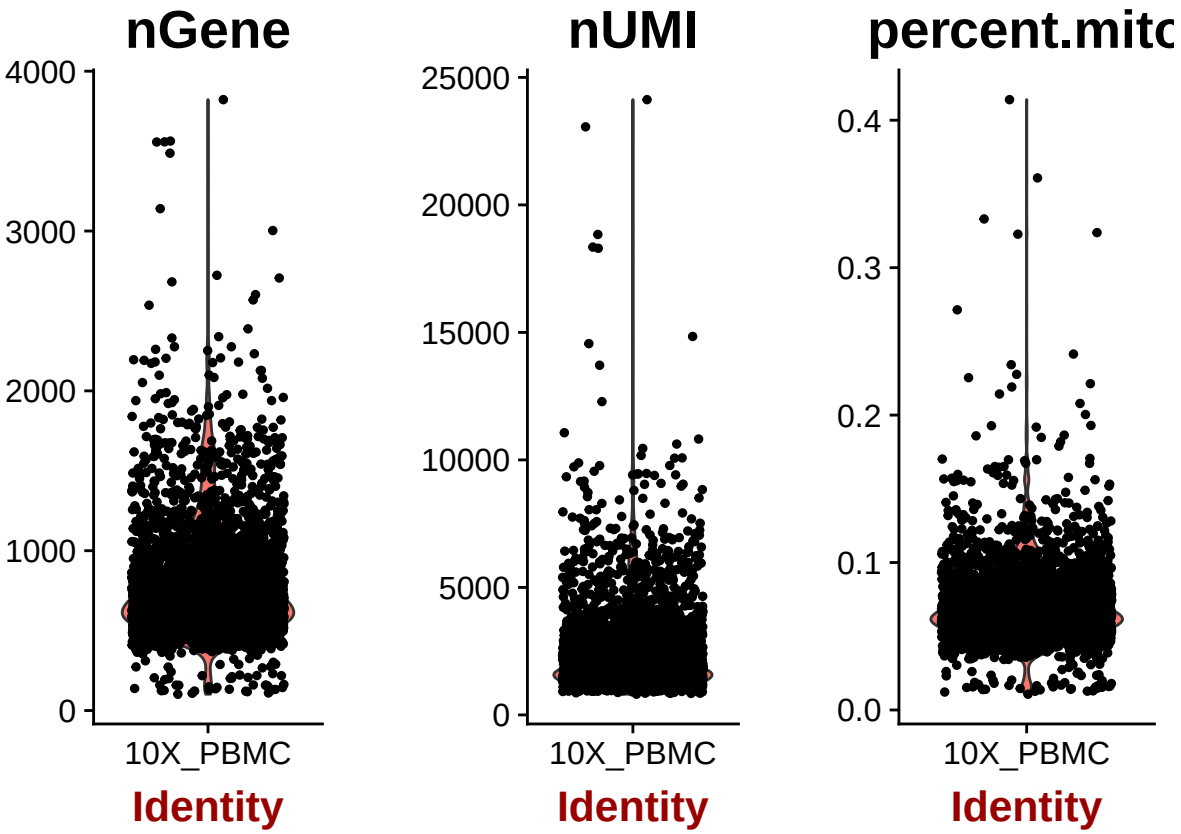
We found that megakaryocytes represented 2.2% of the PBMCs as judged by expression of the megakaryocyte marker PF4

```
colnames(stim) <- as.vector(barcodes.stim[-1])
rownames(stim) <- genes.stim
head(stim[, 1:6])
```

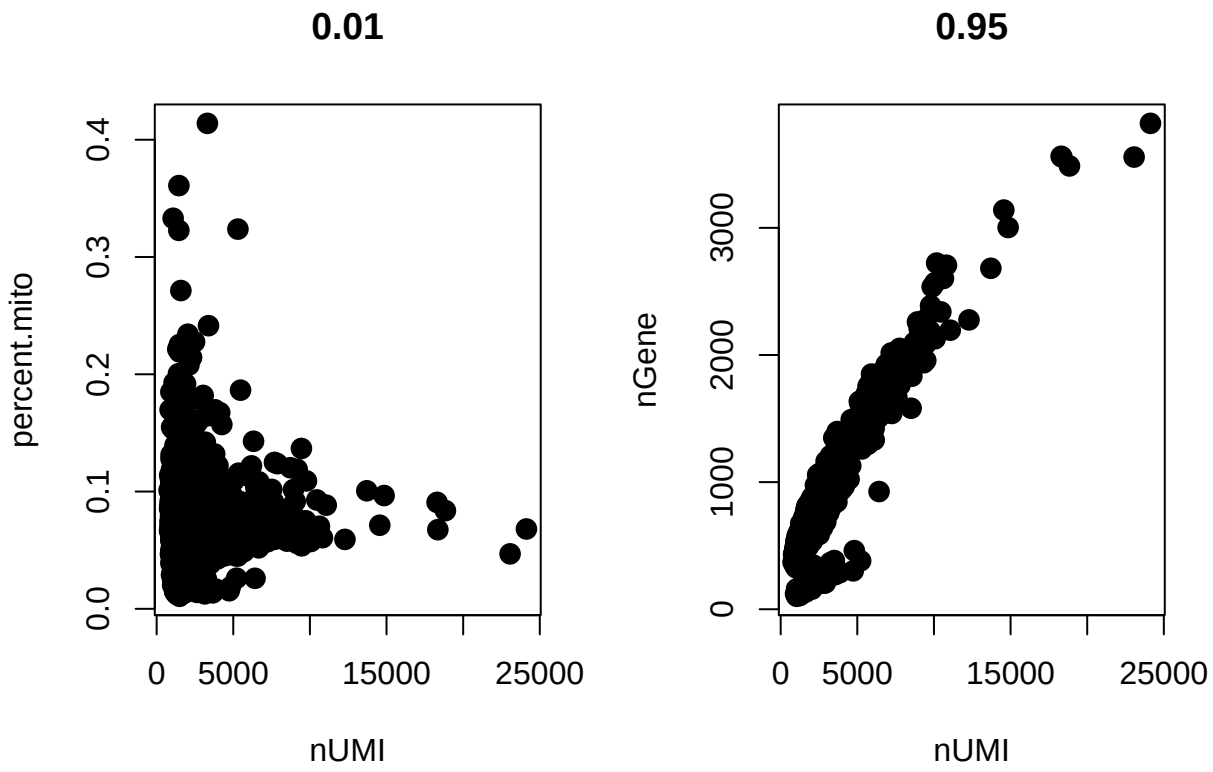
```
##           AAACCTGCACCTCGGA AAACCTGCATGCTAGT AAACCTGCATGGGAAC
## 7SK                      0                0                0
## A1BG                     0                0                0
## A1BG-AS1                 0                0                0
## A2M                      0                0                0
## A2M-AS1                  0                0                0
## A2ML1-AS2                0                0                0
##           AAACCTGGTGATGTGG AAACGGGAGCGGCTTC AAACGGGCAATCGGTT
## 7SK                      0                0                0
## A1BG                     0                0                0
## A1BG-AS1                 0                0                0
## A2M                      0                0                0
## A2M-AS1                  0                0                0
## A2ML1-AS2                0                0                0
```

```
pbmc <- CreateSeuratObject(raw.data = stim, min.cells = 2, min.genes = 100,
    project = "10X_PBMC")
```

```
# scRNA-seq QC metric.
mito.genes <- grep(pattern = "^MT-", x = rownames(x = pbmc@data),
    value = TRUE)
percent.mito <- Matrix::colSums(pbmc@raw.data[mito.genes, ])/Matrix::colSums(pbmc@raw.data)

# AddMetaData adds columns to object@meta.data, and is a
# great place to stash QC stats
pbmc <- AddMetaData(object = pbmc, metadata = percent.mito, col.name = "percent.mito")
VlnPlot(object = pbmc, features.plot = c("nGene", "nUMI", "percent.mito"),
    nCol = 3)
```




```
# GenePlot is typically used to visualize gene-gene
# relationships, but can be used for anything calculated by
# the object, i.e. columns in object@meta.data, PC scores
# etc.  Since there is a rare subset of cells with an outlier
# level of high mitochondrial percentage and also low UMI
# content, we filter these as well
par(mfrow = c(1, 2))
GenePlot(object = pbmc, gene1 = "nUMI", gene2 = "percent.mito")
GenePlot(object = pbmc, gene1 = "nUMI", gene2 = "nGene")
```




```
# We filter out cells that have unique gene counts over 2,500
# or less than 200 Note that low.thresholds and
# high.thresholds are used to define a 'gate'.  -Inf and Inf
# should be used if you don't want a lower or upper
# threshold.
pbmc <- FilterCells(object = pbmc, subset.names = c("nGene",
    "percent.mito"), low.thresholds = c(100, -Inf), high.thresholds = c(5500,
    0.05))

# 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 gene expression measurements for each cell
# by the total expression, multiplies this by a scale factor
# (10,000 by default), and log-transforms the result.

pbmc <- NormalizeData(object = pbmc, normalization.method = "LogNormalize",
    scale.factor = 1000)
```

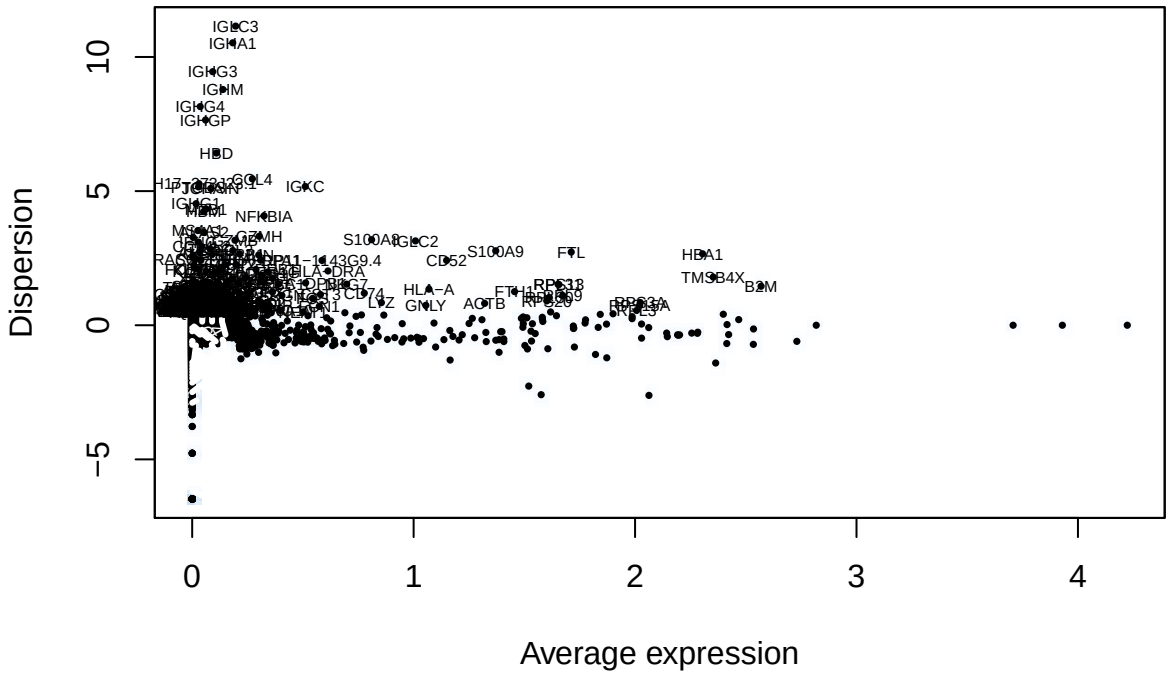
```
pbmc <- FindVariableGenes(object = pbmc, mean.function = ExpMean,
    dispersion.function = LogVMR, x.low.cutoff = 0.0125, x.high.cutoff = 4,
    y.cutoff = 0.5)
length(x = pbmc@var.genes)
```

```
## [1] 301
```

```
pbmc <- ScaleData(object = pbmc, vars.to.regress = c("nUMI",
    "percent.mito"))
```

```
##
## Time Elapsed:  12.455894947052 secs
```

```
pbmc <- FindVariableGenes(object = pbmc, mean.function = ExpMean,
    dispersion.function = LogVMR, x.low.cutoff = 0.0125, x.high.cutoff = 4,
    y.cutoff = 0.5)
```




```
length(x = pbmc@var.genes)
```

```
## [1] 301
```

```
paste("nUMI, percent.mito, these variables won't change the expression spectrum")
```

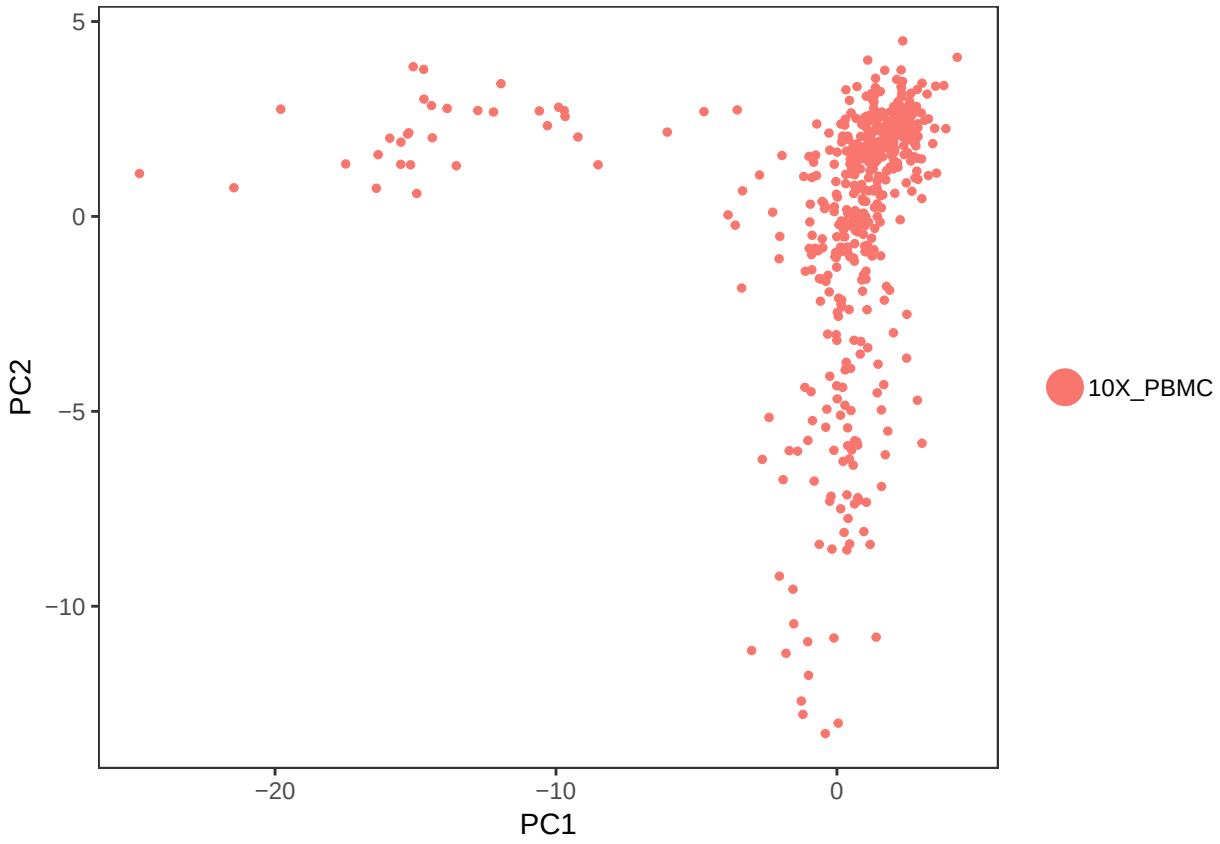
```
## [1] "nUMI, percent.mito, these variables won't change the expression spectrum"
```

```
pbmc <- RunPCA(object = pbmc, pc.genes = pbmc@var.genes, do.print = TRUE,
    pcs.print = 1:10, genes.print = 5)
```

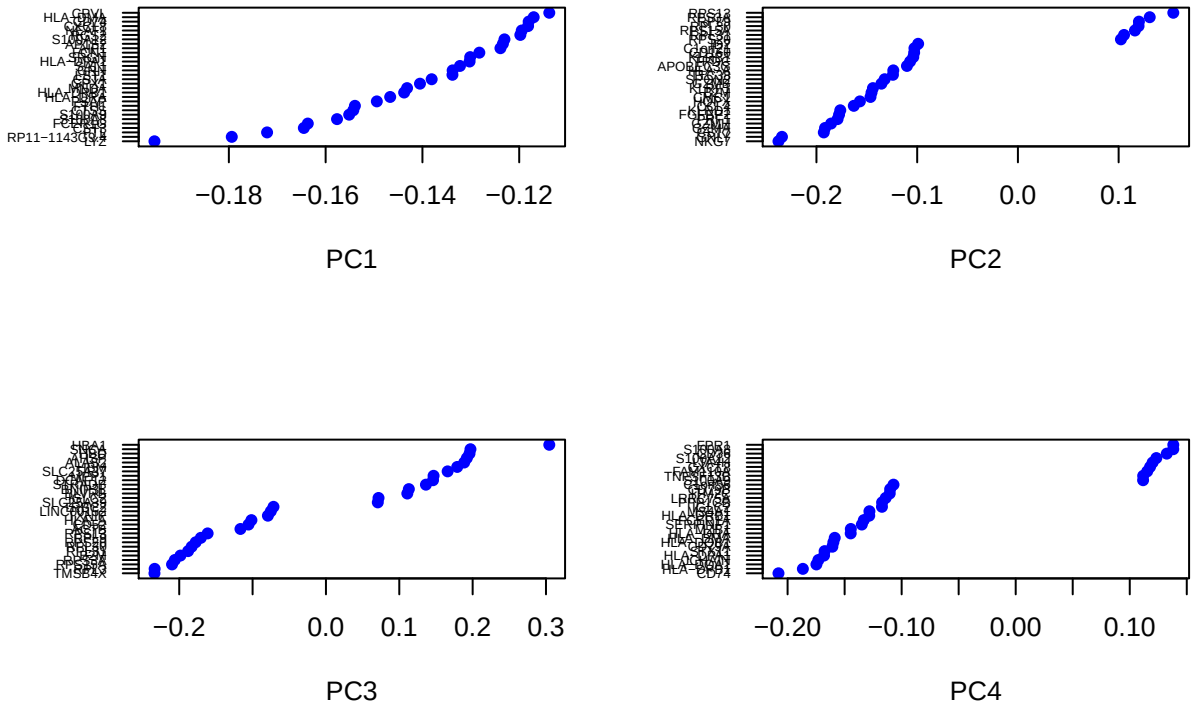
```
## [1] "PC1"
## [1] "LYZ"           "RP11-1143G9.4" "FTL"           "CST3"
## [5] "FCER1G"
## [1] ""
## [1] "RPL3"  "RPL31" "RPL30" "RPL9"  "IL7R"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "NKG7" "GNLY" "CST7" "GZMA" "GZMH"
## [1] ""
## [1] "RPS13"  "RPS3A"  "RPL9"   "RPL30"  "RPS15A"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "TMSB4X" "RPL3"   "RPS15A" "RPS3A"  "B2M"
## [1] ""
## [1] "HBA1"  "SNCA"  "HBD"   "AHSP"  "ALAS2"
## [1] ""
## [1] ""
```

```
## [1] "PC4"
## [1] "CD74"     "HLA-DPB1" "HLA-DQA1" "JCHAIN"   "HLA-DPA1"
## [1] ""
## [1] "FPR1"    "S100A8"  "CD36"    "S100A12" "LTA4H"
## [1] ""
## [1] ""
## [1] "PC5"
## [1] "NFKBIA" "KLRC1"  "MAFF"   "CD69"   "JUN"
## [1] ""
## [1] "FGFBP2" "PATL2"  "CD52"   "GZMH"   "TRGC2"
## [1] ""
## [1] ""
## [1] "PC6"
## [1] "JCHAIN"   "SERPINF1" "MZB1"     "IGLC2"    "ITM2C"
## [1] ""
## [1] "MLXIP"  "STX11"  "IFITM3" "CD14"   "R3HDM4"
## [1] ""
## [1] ""
## [1] "PC7"
## [1] "LINC00152" "SERPINA1"  "GZMK"      "IL2RG"     "C19orf38"
## [1] ""
## [1] "MKNK1"   "LRRC75A" "SDCCAG8" "LITAF"   "ANXA4"
## [1] ""
## [1] ""
## [1] "PC8"
## [1] "FBXW5"      "CSGALNACT2" "ATXN1"      "RASGEF1B"   "ATIC"
## [1] ""
## [1] "FLOT1"    "SDCCAG8"  "HNRNPUL2" "LGALS2"   "IL1B"
## [1] ""
## [1] ""
## [1] "PC9"
## [1] "PPP1CB"  "LRRC75A" "CHFR"    "CHP1"    "MRPL3"
## [1] ""
## [1] "ANXA4"    "C1orf21"  "LILRB2"   "SERPINB9" "IFITM3"
## [1] ""
## [1] ""
## [1] "PC10"
## [1] "CCDC124" "SYNE2"   "ANKZF1"  "CD79A"   "CCL4L2"
## [1] ""
## [1] "C1orf56"     "MMP25-AS1"   "CDC42SE1"    "CCDC22"      "CTC-425F1.4"
## [1] ""
## [1] ""
```

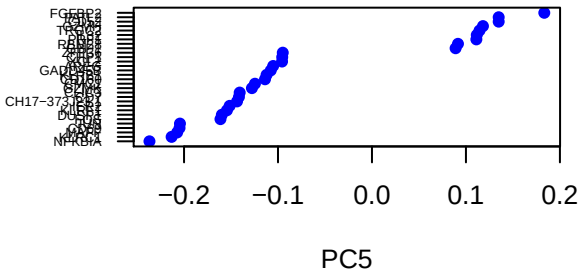
```
PCAPlot(object = pbmc, dim.1 = 1, dim.2 = 2)
```

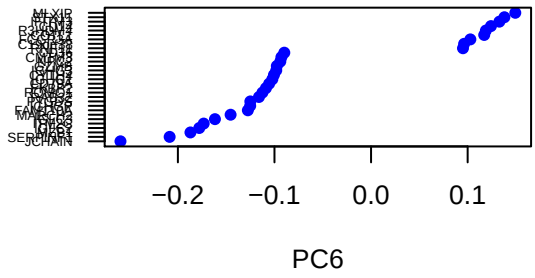


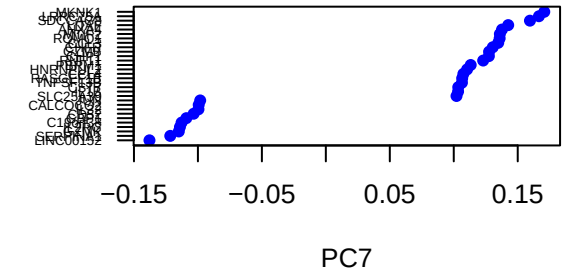

```
VizPCA(object = pbmc, pcs.use = 1:4)
```

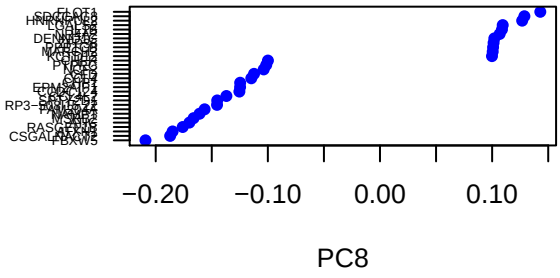



```
VizPCA(object = pbmc, pcs.use = 5:8)
```

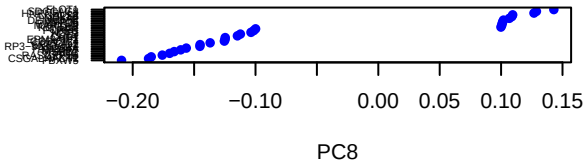


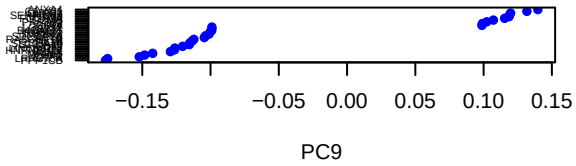


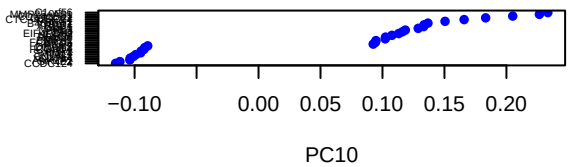


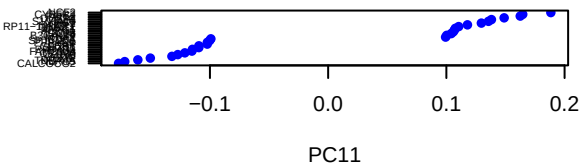


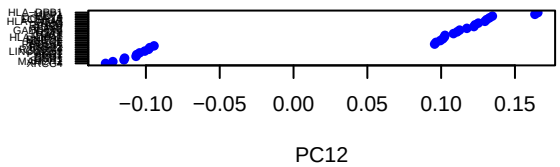

```
VizPCA(object = pbmc, pcs.use = 8:12)
```





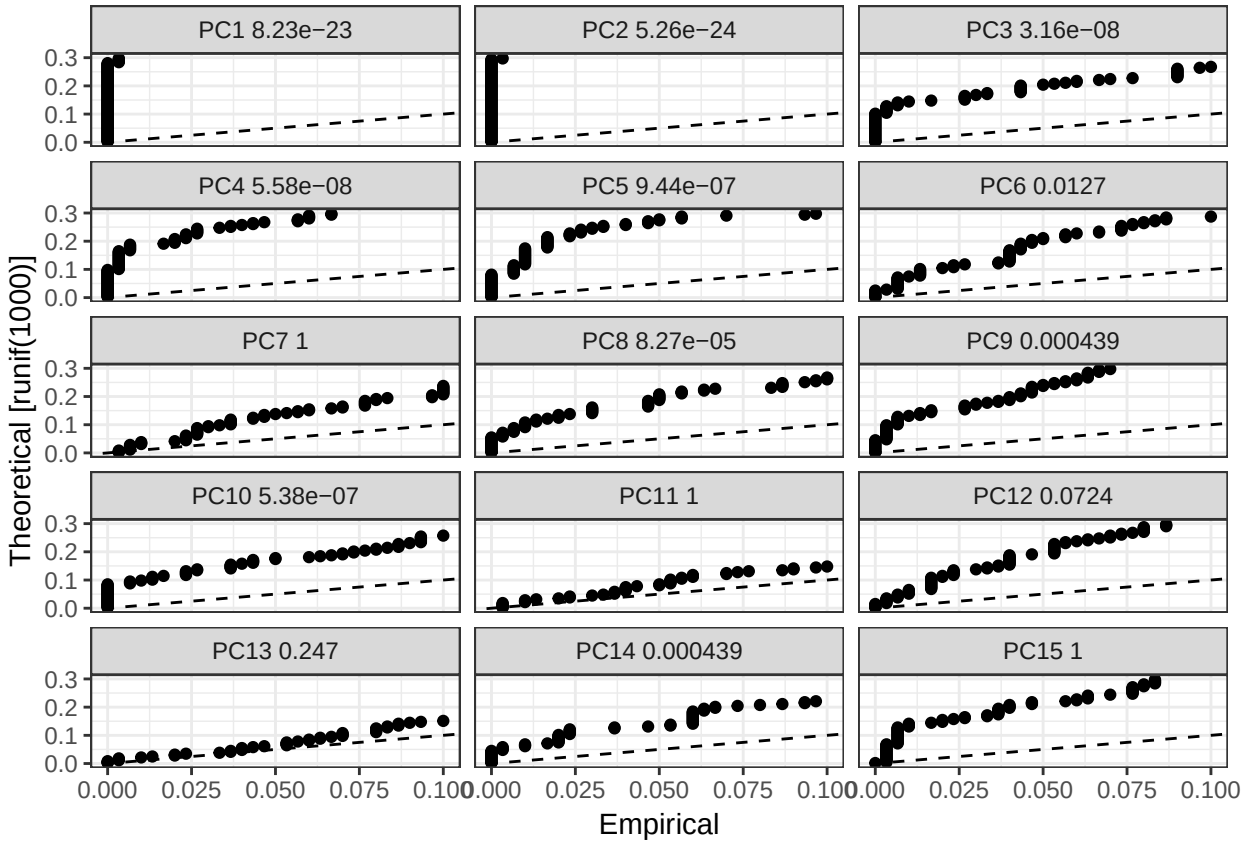






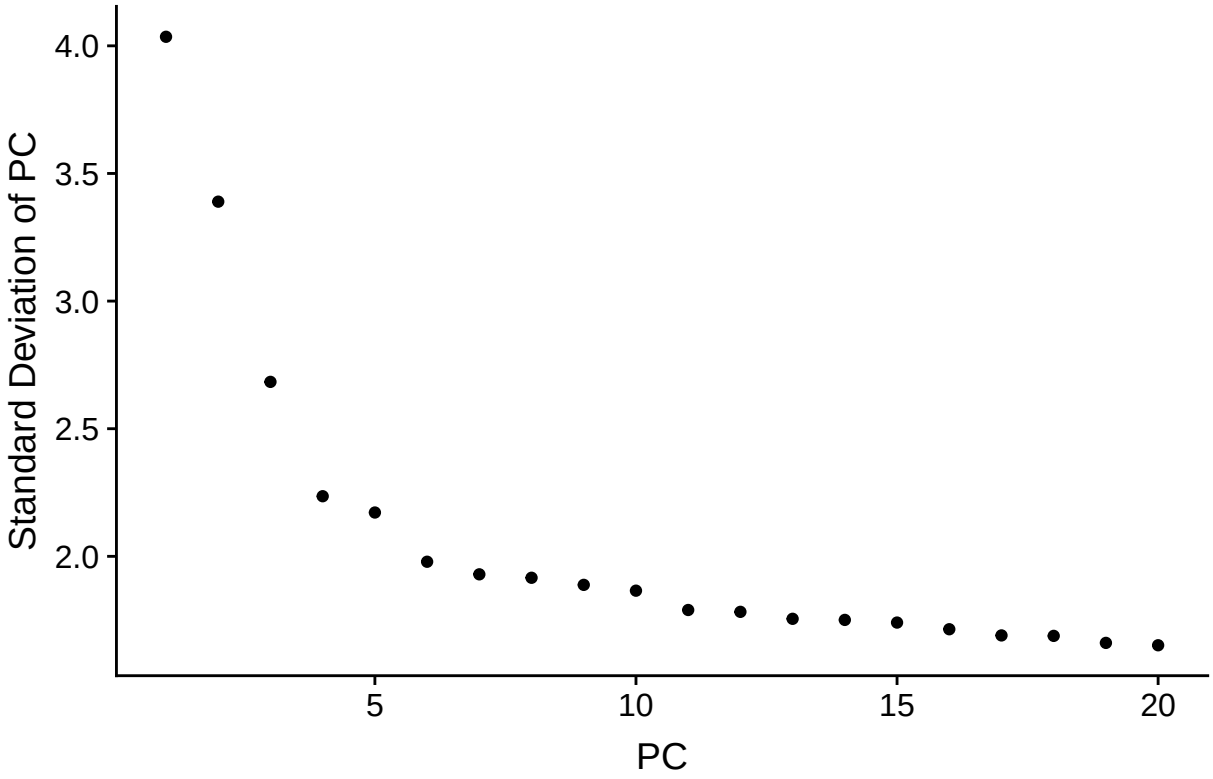

```
pbmc <- ProjectPCA(object = pbmc, do.print = FALSE)

# NOTE: This process can take a long time for big datasets,
# comment out for expediency.  More approximate techniques
# such as those implemented in PCElbowPlot() can be used to
# reduce computation time
```

```
pbmc <- JackStraw(object = pbmc, num.replicate = 100, display.progress = FALSE)
JackStrawPlot(object = pbmc, PCs = 1:15)
```




```
## An object of class seurat in project 10X_PBMC
##  16949 genes across 479 samples.
```

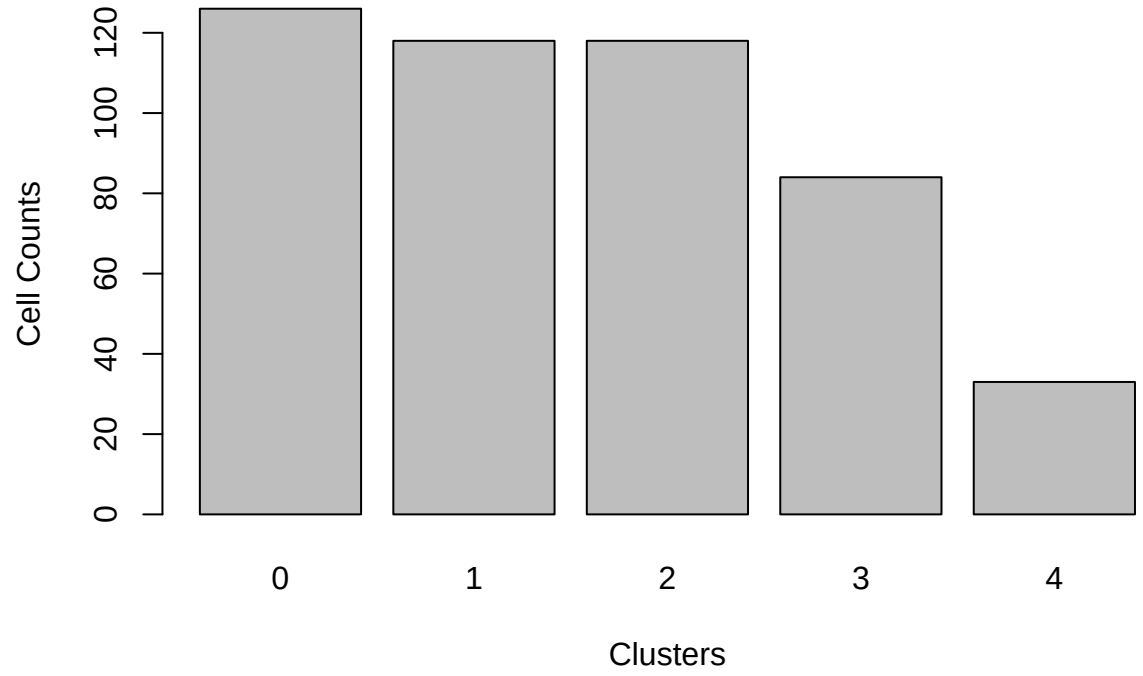
```
PCElbowPlot(object = pbmc)
```



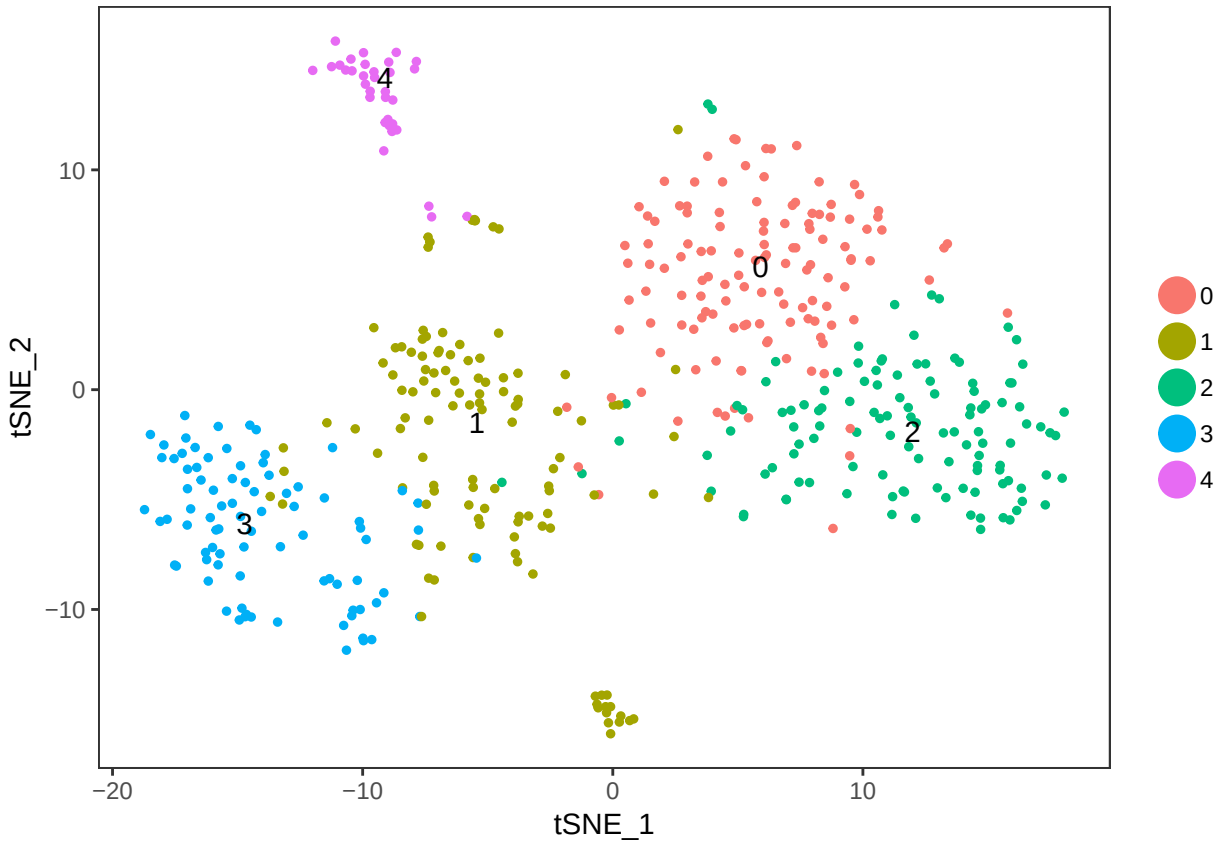

```
paste("Dimension: 9, resolution 1 was chosen before t-SNE")
```

```
## [1] "Dimension: 9, resolution 1 was chosen before t-SNE"
```

```
## rerun to choose clusters setting
pbmc <- FindClusters(object = pbmc, reduction.type = "pca", dims.use = 1:9,
    resolution = 1, print.output = 0, save.SNN = TRUE)
barplot(table(pbmc@ident), xlab = "Clusters", ylab = "Cell Counts")
```



```
pbmc <- RunTSNE(object = pbmc, dims.use = 1:9, do.fast = TRUE)

# note that you can set do.label=T to help label individual
# clusters
plot1 <- TSNEPlot(object = pbmc, do.label = T)
```

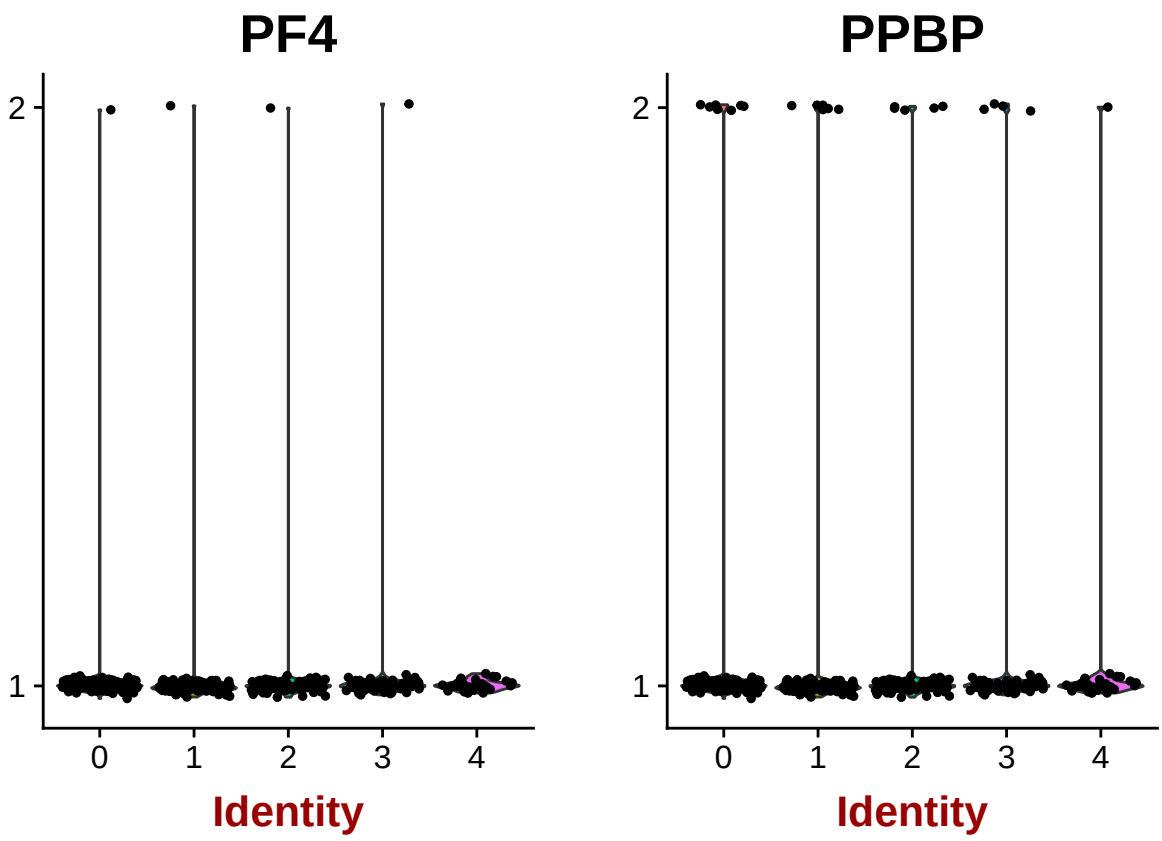



```
## You can save the object at this point so that it can easily
## be loaded back in without having to rerun the
## computationally intensive steps performed above, or easily
## shared with collaborators.

saveRDS(pbmc, file = "~/Documents/Stats/7659Genetics/Proj/pbmc_GSM3374614.rds")

# find all markers of cluster 1
for (i in 0:4) {
    cluster1.markers <- FindMarkers(object = pbmc, ident.1 = i,
        min.pct = 0.25)
    print(x = head(x = cluster1.markers, n = 5))
    if ("PF4" %in% rownames(cluster1.markers)) {
        print(which(rownames(cluster1.markers %in% "PF4")))
    } else {
        print("No")
    }
}
```

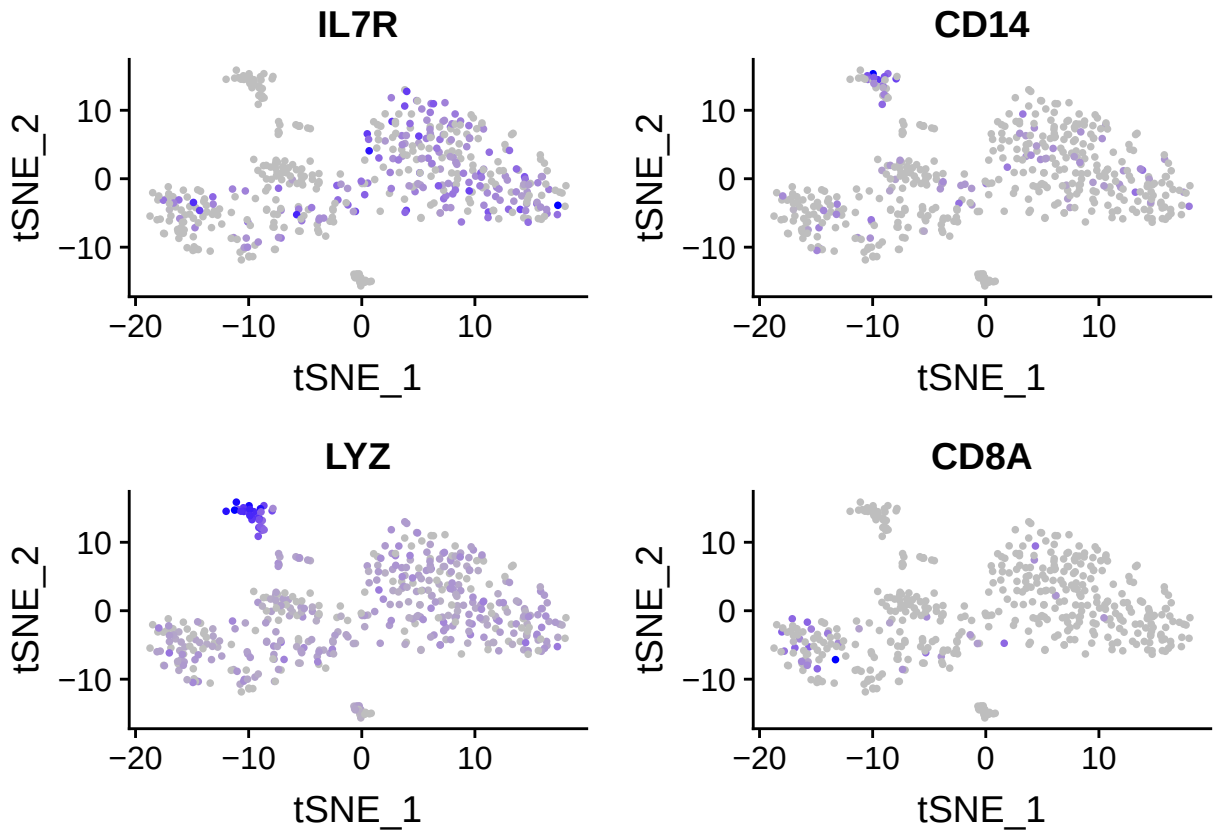
```
##              p_val avg_logFC pct.1 pct.2    p_val_adj
## RPL13 9.225103e-26 0.3971148     1 0.994 1.563563e-21
```

```
## RPS6  5.102441e-24 0.4045051     1 0.963 8.648127e-20
## RPL32 2.449273e-23 0.3864024     1 0.975 4.151273e-19
## RPL34 1.036862e-21 0.3951440     1 0.977 1.757377e-17
## RPS12 2.229004e-20 0.3657651     1 0.992 3.777940e-16
## [1] "No"
##               p_val  avg_logFC pct.1 pct.2    p_val_adj
## RPL21  8.706512e-28 -0.6484239 0.949 1.000 1.475667e-23
## RPL3   8.989501e-27 -0.6258101 0.890 0.997 1.523631e-22
## RPS3A  1.784517e-26 -0.6214791 0.924 0.994 3.024579e-22
## RPL35A 2.166248e-26 -0.6051539 0.839 0.997 3.671574e-22
## RPS13  2.218985e-25 -0.6071093 0.822 0.992 3.760957e-21
## [1] "No"
##               p_val avg_logFC pct.1 pct.2    p_val_adj
## RPS8   5.882943e-23 0.4156083 1.000 0.961 9.971001e-19
## RPS3A  1.682725e-20 0.3813985 1.000 0.970 2.852050e-16
## RPL21  6.041562e-20 0.3668703 1.000 0.983 1.023984e-15
## RPL11  1.144579e-19 0.3598457 0.992 0.986 1.939947e-15
## RPS27A 2.832557e-19 0.2965453 1.000 0.994 4.800901e-15
## [1] "No"
##               p_val avg_logFC pct.1 pct.2    p_val_adj
## NKG7   1.728184e-42 1.3310337 0.952 0.410 2.929098e-38
## FGFBP2 1.130273e-39 0.5603550 0.595 0.053 1.915700e-35
## GZMA   1.900047e-39 0.6199464 0.786 0.149 3.220389e-35
## GNLY   3.911322e-36 1.6417059 0.976 0.668 6.629300e-32
## GZMH   1.876956e-34 0.7880573 0.750 0.175 3.181253e-30
## [1] "No"
##                      p_val avg_logFC pct.1 pct.2    p_val_adj
## HLA-DMA       3.430487e-23 0.5174735 0.697 0.112 5.814333e-19
## MNDA          7.823857e-23 0.4873723 0.636 0.083 1.326066e-18
## CST3          3.216934e-20 1.2120897 1.000 0.632 5.452381e-16
## RP11-1143G9.4 3.553439e-20 1.5530546 1.000 0.594 6.022724e-16
## HLA-DRB1      7.166284e-20 0.8831972 0.909 0.330 1.214614e-15
## [1] "No"
```

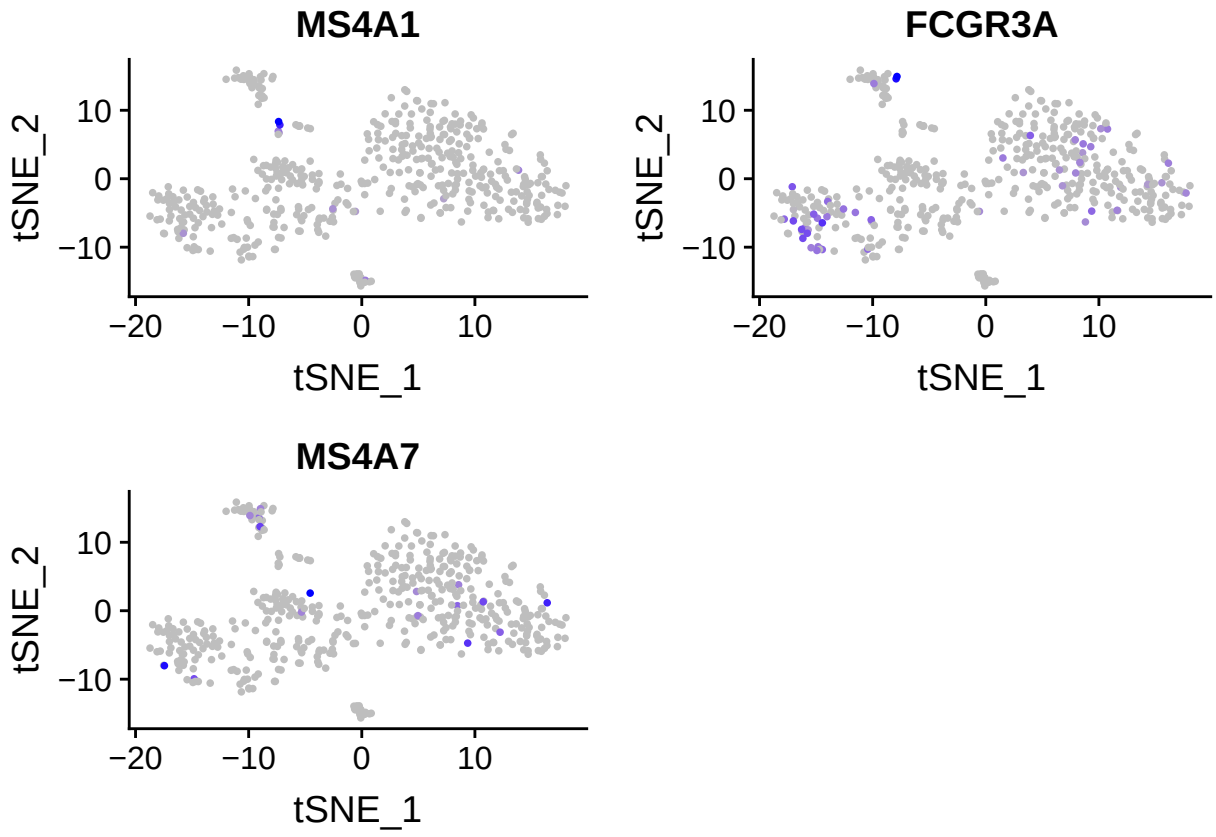
```
# you can plot raw UMI counts as well
VlnPlot(object = pbmc, features.plot = c("PF4", "PPBP"), use.raw = TRUE,
    y.log = TRUE)
```



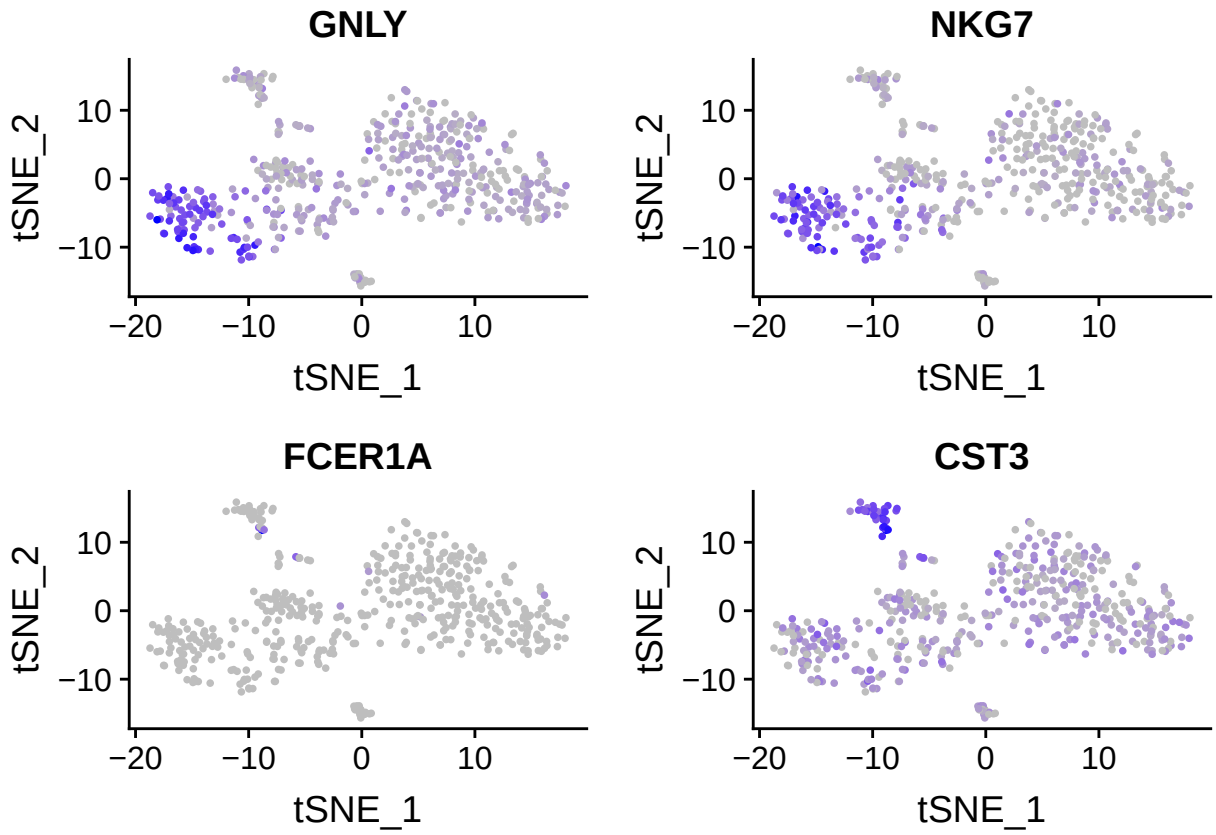

```
FeaturePlot(object = pbmc, features.plot = c("IL7R", "CD14",
    "LYZ", "CD8A"), cols.use = c("grey", "blue"), reduction.use = "tsne")
```



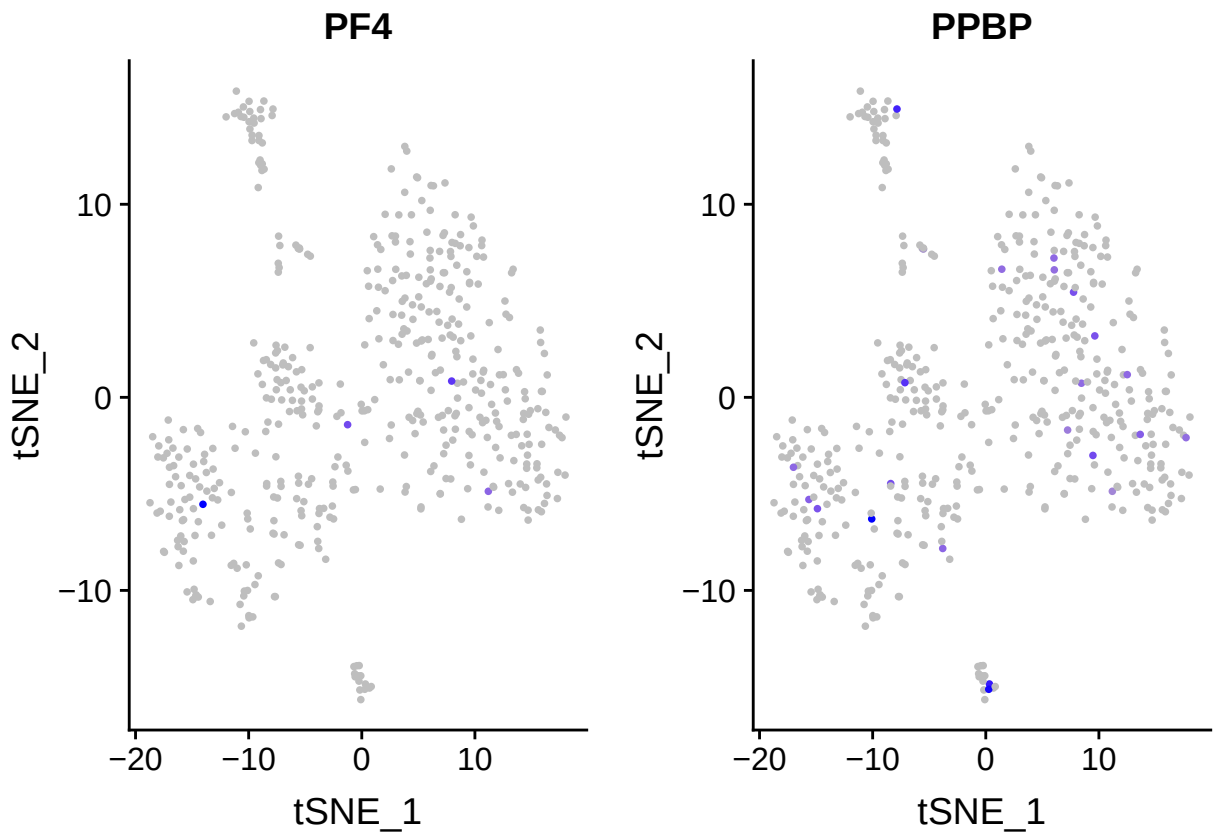

```
FeaturePlot(object = pbmc, features.plot = c("MS4A1", "FCGR3A",
    "MS4A7"), cols.use = c("grey", "blue"), reduction.use = "tsne")
```



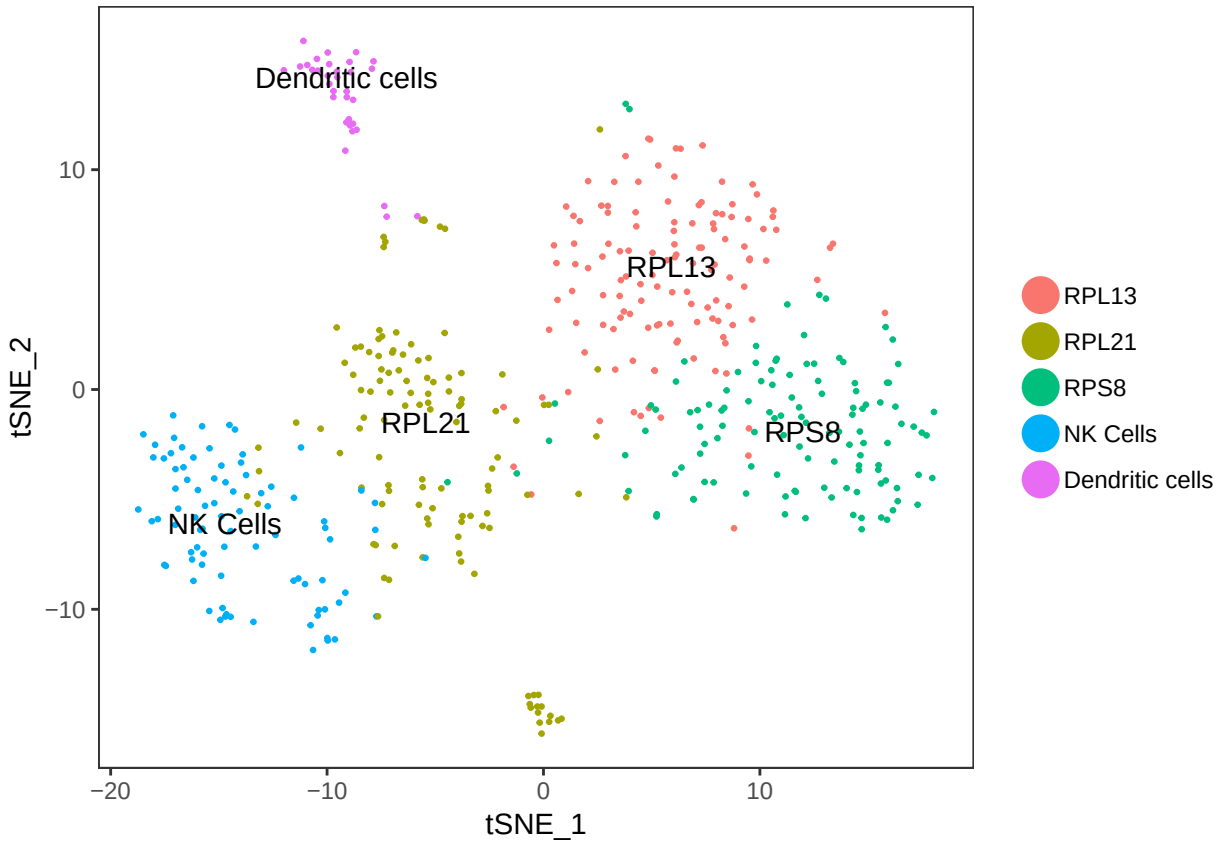

```
FeaturePlot(object = pbmc, features.plot = c("GNLY", "NKG7",
    "FCER1A", "CST3"), cols.use = c("grey", "blue"), reduction.use = "tsne")
```



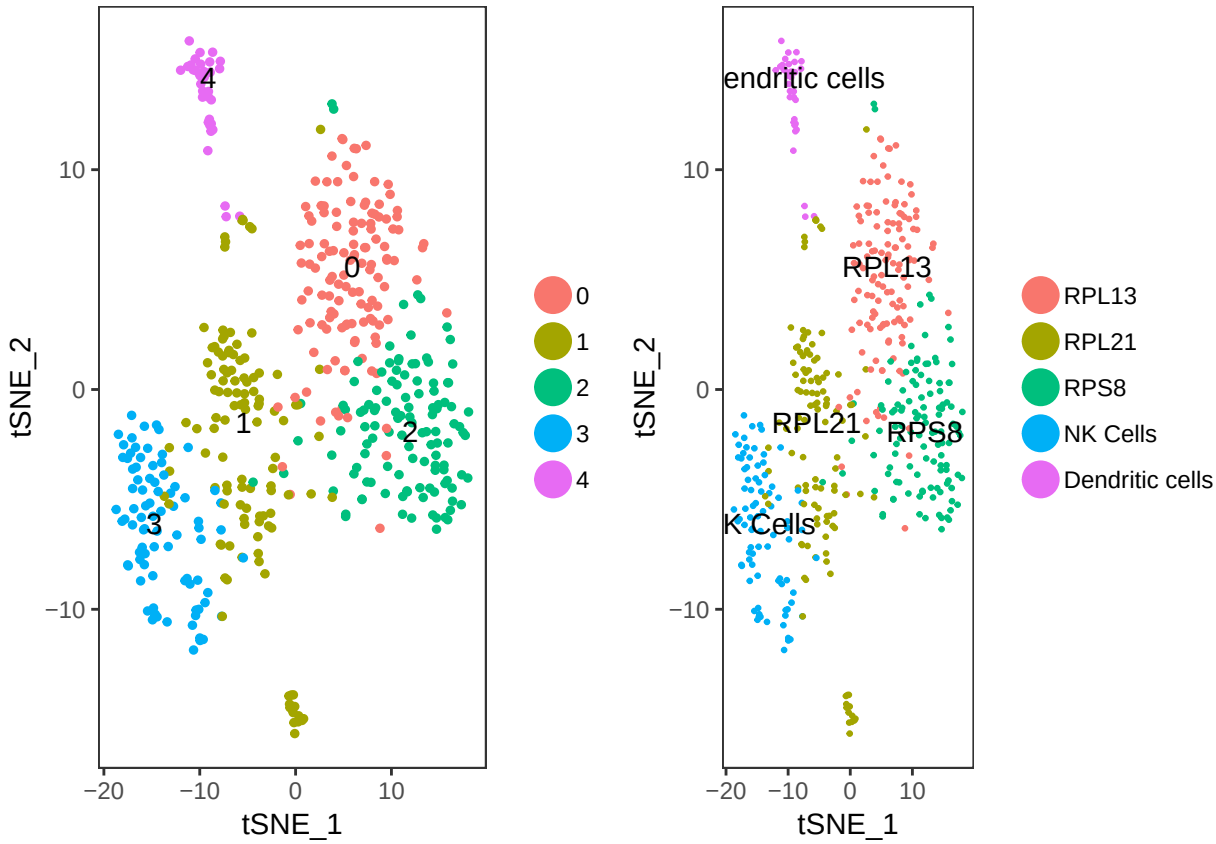

```
FeaturePlot(object = pbmc, features.plot = c("PF4", "PPBP"),
    cols.use = c("grey", "blue"), reduction.use = "tsne")
```




```
current.cluster.ids <- c(0, 1, 2, 3, 4)

new.cluster.ids <- c("RPL13", "RPL21", "RPS8", "NK Cells", "Dendritic cells")
pbmc@ident <- plyr::mapvalues(x = pbmc@ident, from = current.cluster.ids,
    to = new.cluster.ids)
plot2 <- TSNEPlot(object = pbmc, do.label = TRUE, pt.size = 0.5)
```




```
## save name First lets stash our identities for later
pbmc <- StashIdent(object = pbmc, save.name = "ClusterNames_1_9")

plot_grid(plot1, plot2)
```




```
saveRDS(pbmc, file = "~/Documents/Stats/7659Genetics/Proj/pbmc_GSM3374614_final.rds")
```