QC Filtering

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

#load libraries  
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(Seurat)

## Attaching SeuratObject

library(patchwork)  
library(data.table)

##   
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

library(stringr)  
library(ggplot2)  
  
#set working directory   
setwd("~/OneDrive - Queen Mary, University of London/QMUL/Lab/Coding/data/R/Seurat/SandLKerato")  
  
#load the dataset from the raw data downloaded  
kerato.data <- Read10X(data.dir = "~/OneDrive - Queen Mary, University of London/QMUL/Lab/Coding/data/R/Seurat/SandLKerato/rawdata/")  
  
#initialise the seurat object with the raw (non-normalised) data.  
kerato <- CreateSeuratObject(counts = kerato.data, project = "Kerato", min.cells = 3, min.features = 100)

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

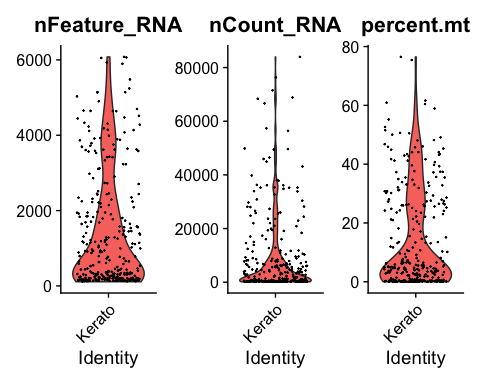
kerato

## An object of class Seurat   
## 10026 features across 299 samples within 1 assay   
## Active assay: RNA (10026 features, 0 variable features)

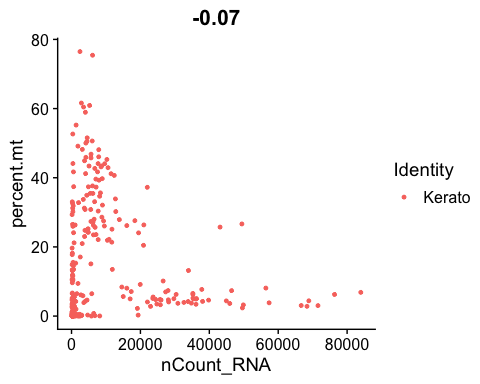
# The [[ operator can add columns to object metadata. This is a great place to stash QC stats. Added percentage of mitochondrial RNA per barcode to 'percent.mt'.   
kerato[["percent.mt"]] <- PercentageFeatureSet(kerato, pattern = "^Mt-")  
  
# show example metadata present.   
head(kerato@meta.data, 5)

## orig.ident nCount\_RNA nFeature\_RNA percent.mt  
## AAACGGGCAGGGTTAG-1 Kerato 2540 889 17.086614  
## AAAGATGCAGTCGTGC-1 Kerato 19948 3642 9.153800  
## AAAGATGGTCTAACGT-1 Kerato 522 364 1.915709  
## AAATGCCCAGGAATGC-1 Kerato 6345 1834 26.256895  
## AACACGTGTTAAGATG-1 Kerato 170 137 0.000000

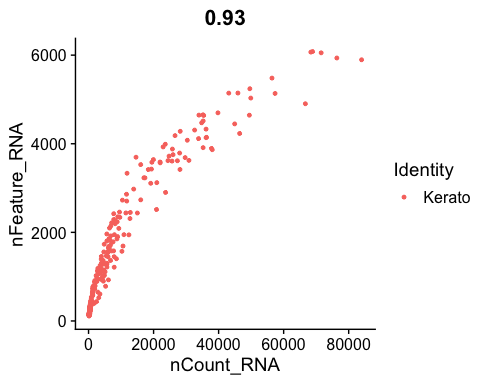
# Visualize QC metrics as a violin plot  
VlnPlot(kerato, features = c("nFeature\_RNA", "nCount\_RNA", "percent.mt"), ncol = 3)



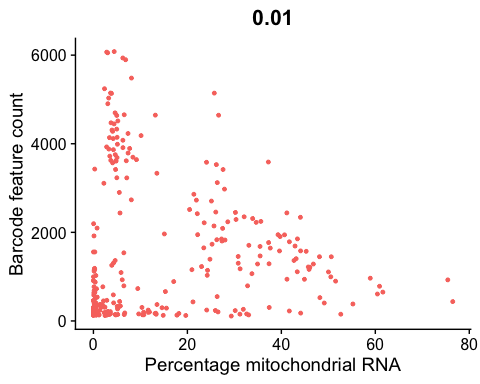
# FeatureScatter is typically used to visualize feature-feature relationships, but can be used  
# for anything calculated by the object, i.e. columns in object metadata, PC scores etc.  
  
plot1 <- FeatureScatter(kerato, feature1 = "nCount\_RNA", feature2 = "percent.mt")  
plot2 <- FeatureScatter(kerato, feature1 = "nCount\_RNA", feature2 = "nFeature\_RNA")  
plot3 <- FeatureScatter(kerato, feature1 = "percent.mt" , feature2 = "nFeature\_RNA") + xlab('Percentage mitochondrial RNA') + ylab('Barcode feature count') + theme(legend.position = "None")  
plot1



plot2



plot3



summary(kerato@meta.data)

## orig.ident nCount\_RNA nFeature\_RNA percent.mt   
## Kerato:299 Min. : 167 Min. : 110 Min. : 0.0000   
## 1st Qu.: 361 1st Qu.: 229 1st Qu.: 0.5935   
## Median : 2449 Median : 793 Median : 5.1345   
## Mean : 9164 Mean :1462 Mean :14.3097   
## 3rd Qu.: 9142 3rd Qu.:2232 3rd Qu.:26.2106   
## Max. :83967 Max. :6080 Max. :76.4802

# here is where we filter with QC metrics, look at violin plots to see number of cells excluded. Will need high mt% and low feature no. to process majority of cells  
kerato <- subset(kerato, subset = nFeature\_RNA > 200 & nFeature\_RNA < 6000 & percent.mt < 50)  
  
kerato\_info <- kerato@meta.data %>% as.data.frame()  
## extract meta data  
# the resulting object has one "row" per cell  
cat('Number of cells in analysis:', nrow(kerato\_info))

## Number of cells in analysis: 227

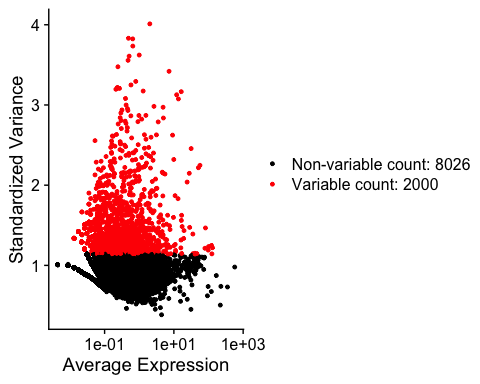
#number of cells pulled through using the filters above is printed to the terminal.

#log normalisation of data  
kerato <- NormalizeData(kerato, normalization.method = "LogNormalize", scale.factor = 10000)

#finding HVGs  
# vst: First, fits a line to the relationship of log(variance) and log(mean) using local polynomial regression (loess). Then standardizes the feature values using the observed mean and expected variance (given by the fitted line). Feature variance is then calculated on the standardized values after clipping to a maximum (see clip.max parameter).  
kerato <- FindVariableFeatures(kerato, selection.method = "vst", nfeatures = 2000)  
  
# Identify the 10 most highly variable genes  
top10 <- head(VariableFeatures(kerato), 10)  
  
# plot variable features with and without labels  
plot1 <- VariableFeaturePlot(kerato)  
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)

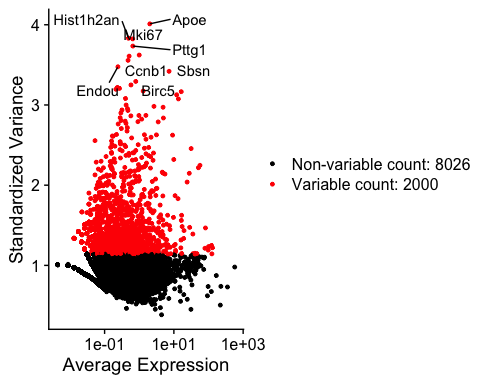
## When using repel, set xnudge and ynudge to 0 for optimal results

plot1



plot2

## Warning: ggrepel: 2 unlabeled data points (too many overlaps). Consider  
## increasing max.overlaps



#Next, we apply a linear transformation ('scaling') that is a standard pre-processing step prior to dimensional reduction techniques like PCA. The ScaleData function:  
  
#Shifts the expression of each gene, so that the mean expression across cells is 0  
#Scales the expression of each gene, so that the variance across cells is 1  
#This step gives equal weight in downstream analyses, so that highly-expressed genes do not dominate  
#The results of this are stored in pbmc[["RNA"]]@scale.data  
all.genes <- rownames(kerato)  
kerato <- ScaleData(kerato, features = all.genes)

## Centering and scaling data matrix

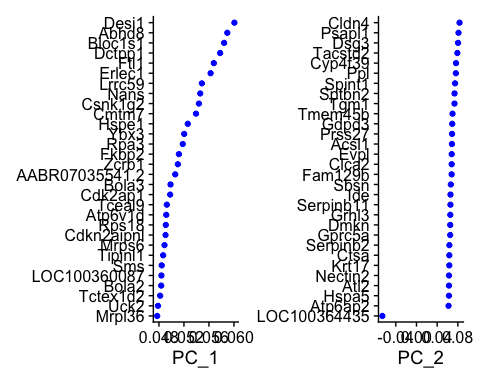
#run the PCA analysis of the dataset  
kerato <- RunPCA(kerato, features = VariableFeatures(object = kerato))

## PC\_ 1   
## Positive: Desi1, Abhd8, Bloc1s1, Dctpp1, Ftl1, Erlec1, Lrrc59, Nans, Csnk1g2, Cmtm7   
## Hspe1, Ybx3, Rpa3, Fkbp2, Zcrb1, AABR07035541.2, Bola3, Cdk2ap1, Tceal9, Atp6v1d   
## Rps18, Cdkn2aipnl, Mrps6, Tipinl1, Sms, LOC100360087, Bola2, Tctex1d2, Uck2, Mrpl36   
## Negative: Krt17, Cnfn, Krtdap, Krt10, Hspb1, Fabp5, Sprr1a, Klk6, Krt16, Dmkn   
## Mfap5, Ly6d, Elovl4, Cryab, Tmem251, AC139608.2, Lrba, Sdr16c6, Atp6v0a4, Nfkbie   
## Wdr41, Endou, Sbsn, Slc34a3, Nr5a2, Tgm3, Klk7, Cenpf, Cdkn1c, Lipn   
## PC\_ 2   
## Positive: Cldn4, Psapl1, Dsg3, Tacstd2, Cyp4f39, Ppl, Spint1, Sptbn2, Tgm1, Tmem45b   
## Gdpd3, Prss27, Acsl1, Evpl, Clca2, Fam129b, Sbsn, Ide, Serpinb11, Grhl3   
## Dmkn, Gprc5a, Serpinb2, Ctsa, Krt17, Nectin2, Atl2, Hspa5, Atp6ap2, Hk2   
## Negative: LOC100364435, Rps23, Rpl21.3, Rps2, Rplp0, Pdgfa, Rps12, Lmo1, Mt2A, Rps27a.1   
## Ftl1, Rps15a, Cavin3, Rps14, Slc25a4, Cav1, Rgs10, Hmgn2, Mt1, H2afv   
## Tmsb4x, Wnt6, AABR07026311.1, Stmn1, LOC100360087, Fth1, Emp3, Cda, Wnt10a, Ifitm3   
## PC\_ 3   
## Positive: Fabp5, Gltp, Rps23, Rpl21.3, Tmsb4x, Rplp0, Rps12, Rps15a, Rps2, Rps14   
## Rps27a.1, Hopx, Gpx2, Perp, Cysrt1, Gng5, Ly6d, Rab25, Dbi, Arpc1a   
## Dmkn, LOC100364435, Sult2b1, Sprr1a, Krtdap, S100a16, Rbp2, Fth1, Msrb1, Gng12   
## Negative: Snhg11, Klf8, Tnc, Slc2a1, Col3a1, Ctsl, C1s, C1r, Hspa5, Thbs2   
## Ptgs2, Dpp7, Jag2, Fn1, Slc1a3, Clu, Fst, Cers4, Igfbp2, Ccdc80   
## Lamb1, Apoe, Wnt6, Gigyf1, Hsp90b1, Slc1a5, Pdgfa, Phpt1, Igfbp7, Dsg3   
## PC\_ 4   
## Positive: Fstl1, C1s, Igfbp2, Kitlg, Igfbp5, Jag2, Tmem205, Akr1b1, Ifi27, Slc1a3   
## Arl4a, Lef1, Traf1, Tspan17, Slc25a4, Tp53i11, Tmem136, Apoe, Rerg, Akap12   
## Fam117a, S100a4, Tnfrsf21, Lztr1, Spon2, Mrps6, Cers4, Cnpy4, Igfbp7, Zfp358   
## Negative: Mki67, Racgap1, Cenpe, Cdca3, Hmgb2l1, Cenpa, Psrc1, Smc2, Ckap2, Kif20b   
## Tpx2, Cks2, Arhgap11a, Ndc80, Kif2c, Cenpm, Cdca2, LOC100359539, Cenpf, Aurka   
## Ect2, Kifc1, Top2a, Spag5.1, Fam83d, Sgo2, AABR07069282.1, Prc1, Kif11, Spc24   
## PC\_ 5   
## Positive: Fam92a, Ahcyl1, Ptpn12, Clca2, AABR07044001.4, Chn2, Prkag2, Jup, Dsg3, Serinc5   
## Hist1h1b, Hist1h2an, Ldah, Pacrgl, Bspry, Bfsp2, Gatad1, Ptpn21, Aktip, RGD1560394   
## Epha2, Fam131c, Map2, Pkib, AABR07049695.3, Dnm1l, RGD1309036, Dph3, Ocel1, Snhg11   
## Negative: Hspb1, Glod4, Krt17, Anxa1, Sprr1a, Cnfn, Krt15, Ldha, Psmb6, Krt10   
## Fdps, Anxa2, Ttk, Tuba1c, Gstp1, Dstn, Tuba1b, Dbi, Anxa8, Prdx1   
## Anp32e, Ttc5, Actb, Tubb6, Mdh1, Prkar1a, Kifc1, Klk6, Kif20a, Sec13

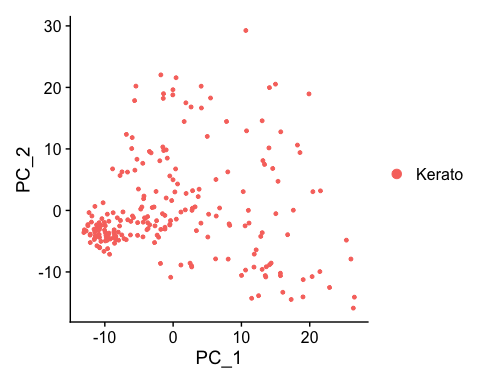
# Examine and visualize PCA results a few different ways  
print(kerato[["pca"]], dims = 1:5, nfeatures = 5)

## PC\_ 1   
## Positive: Desi1, Abhd8, Bloc1s1, Dctpp1, Ftl1   
## Negative: Krt17, Cnfn, Krtdap, Krt10, Hspb1   
## PC\_ 2   
## Positive: Cldn4, Psapl1, Dsg3, Tacstd2, Cyp4f39   
## Negative: LOC100364435, Rps23, Rpl21.3, Rps2, Rplp0   
## PC\_ 3   
## Positive: Fabp5, Gltp, Rps23, Rpl21.3, Tmsb4x   
## Negative: Snhg11, Klf8, Tnc, Slc2a1, Col3a1   
## PC\_ 4   
## Positive: Fstl1, C1s, Igfbp2, Kitlg, Igfbp5   
## Negative: Mki67, Racgap1, Cenpe, Cdca3, Hmgb2l1   
## PC\_ 5   
## Positive: Fam92a, Ahcyl1, Ptpn12, Clca2, AABR07044001.4   
## Negative: Hspb1, Glod4, Krt17, Anxa1, Sprr1a

#visualise the PCA coordinates of genes  
VizDimLoadings(kerato, dims = 1:2, reduction = "pca")

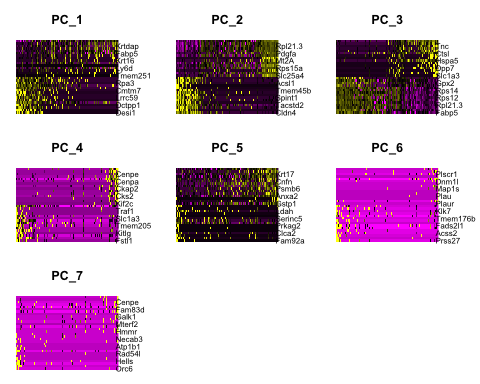


# plot cells using two PCAs as axis.   
DimPlot(kerato, reduction = "pca")



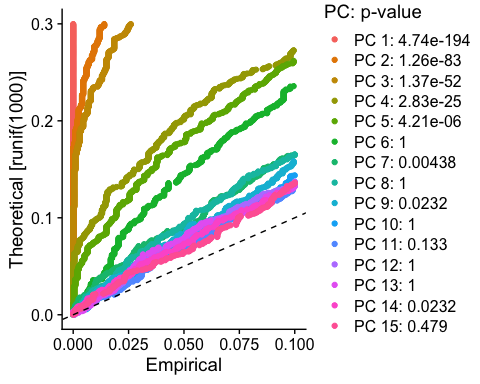
#In particular DimHeatmap allows for easy exploration of the primary sources of heterogeneity in a dataset, and can be useful when trying to decide which PCs to include for further downstream analyses. Both cells and features are ordered according to their PCA scores. Setting cells to a number plots the 'extreme' cells on both ends of the spectrum, which dramatically speeds plotting for large datasets. Though clearly a supervised analysis, we find this to be a valuable tool for exploring correlated feature sets.  
DimHeatmap(kerato, dims = 1:7, cells = 500, balanced = TRUE)

## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.  
  
## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.  
  
## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.  
  
## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.  
  
## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.  
  
## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.  
  
## Warning: Requested number is larger than the number of available items (227).  
## Setting to 227.

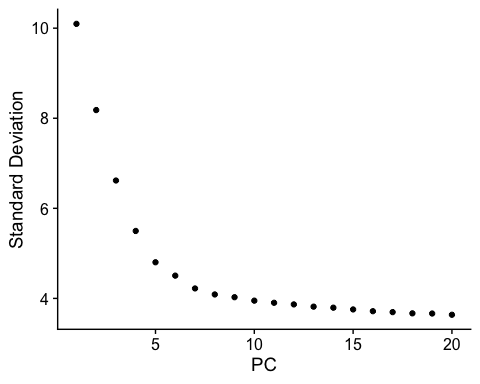


# NOTE: This process can take a long time for big datasets, comment out for expediency. More approximate techniques such as those implemented in ElbowPlot() can be used to reduce computation time  
kerato <- JackStraw(kerato, num.replicate = 100)  
kerato <- ScoreJackStraw(kerato, dims = 1:20)  
  
JackStrawPlot(kerato, dims = 1:15)

## Warning: Removed 24054 rows containing missing values (geom\_point).



#elbow plot , shows SD of PCs, elbow is where significance should begin to be negligible.  
ElbowPlot(kerato)



#Here is where we optimise the number of PCs used to cluster the cells, and the resolution of the clustering algorithm.  
kerato <- FindNeighbors(kerato, dims = 1:7)

## Computing nearest neighbor graph

## Computing SNN

kerato <- FindClusters(kerato, resolution = 0.4)

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 227  
## Number of edges: 5790  
##   
## Running Louvain algorithm...  
## Maximum modularity in 10 random starts: 0.8002  
## Number of communities: 4  
## Elapsed time: 0 seconds

head(Idents(kerato), 5)

## AAACGGGCAGGGTTAG-1 AAAGATGCAGTCGTGC-1 AAAGATGGTCTAACGT-1 AAATGCCCAGGAATGC-1   
## 0 2 0 1   
## AACCATGCAACCGCCA-1   
## 0   
## Levels: 0 1 2 3

#Plot UMAP  
kerato <- RunUMAP(kerato, dims = 1:7)

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

## 15:53:10 UMAP embedding parameters a = 0.9922 b = 1.112

## 15:53:10 Read 227 rows and found 7 numeric columns

## 15:53:10 Using Annoy for neighbor search, n\_neighbors = 30

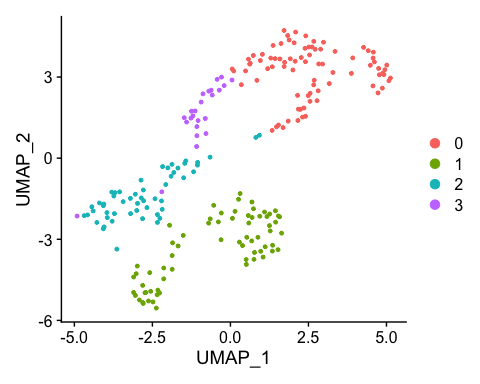
## 15:53:10 Building Annoy index with metric = cosine, n\_trees = 50

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

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

## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|  
## 15:53:10 Writing NN index file to temp file /var/folders/17/t98thp9n3zb7xfm2pnqtq84w0000gn/T//RtmpSQu0zc/file1a225fe5e2e1  
## 15:53:10 Searching Annoy index using 1 thread, search\_k = 3000  
## 15:53:10 Annoy recall = 100%  
## 15:53:11 Commencing smooth kNN distance calibration using 1 thread  
## 15:53:11 Initializing from normalized Laplacian + noise  
## 15:53:11 Commencing optimization for 500 epochs, with 7930 positive edges  
## 15:53:12 Optimization finished

DimPlot(kerato, reduction = "umap")



#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(kerato, file = "smalllargekera.rds")