scRNA-seq data analysis

David Chisanga and Wei Shi

24 June, 2022

Contents

Chapter 1

Preface

This tutorial describes the Bioinformatics protocol for analyzing single-cell RNA-seq (scRNA-seq) data using *cellCounts* function in (**Rsubread**) and **Seurat** pipeline. This guide provides a brief discussion and examples of the steps involved in the analysis of scRNA-seq data. These include;

- processing raw data,
- quality control,
- normalization,
- data correction,
- dimension reduction,
- cluster analysis,
- cell type annotation,
- trajectory analysis
- and other downstream analyses.

Chapter 2

Introduction

Single-cell RNA-seq (scRNA-seq) first published by (Tang et al., 2009) measures the distribution of expression levels in individual cells. This allows for the study of transcriptome changes at the cellular level to provide unprecedented resolution of gene expression changes in complex cellular systems. As in bulk RNA-seq, scRNA-seq involves a range of analyses including quantification, dimension reduction, cell clustering, cell type identification, inference of gene-regulatory networks, differential expression and cell hierarchy reconstruction. This scRNA-seq protocol provides a step-by-step overview of a standard scRNA-seq analysis. The steps can be categorized into 3 groups; quantification, quality control and downstream analysis

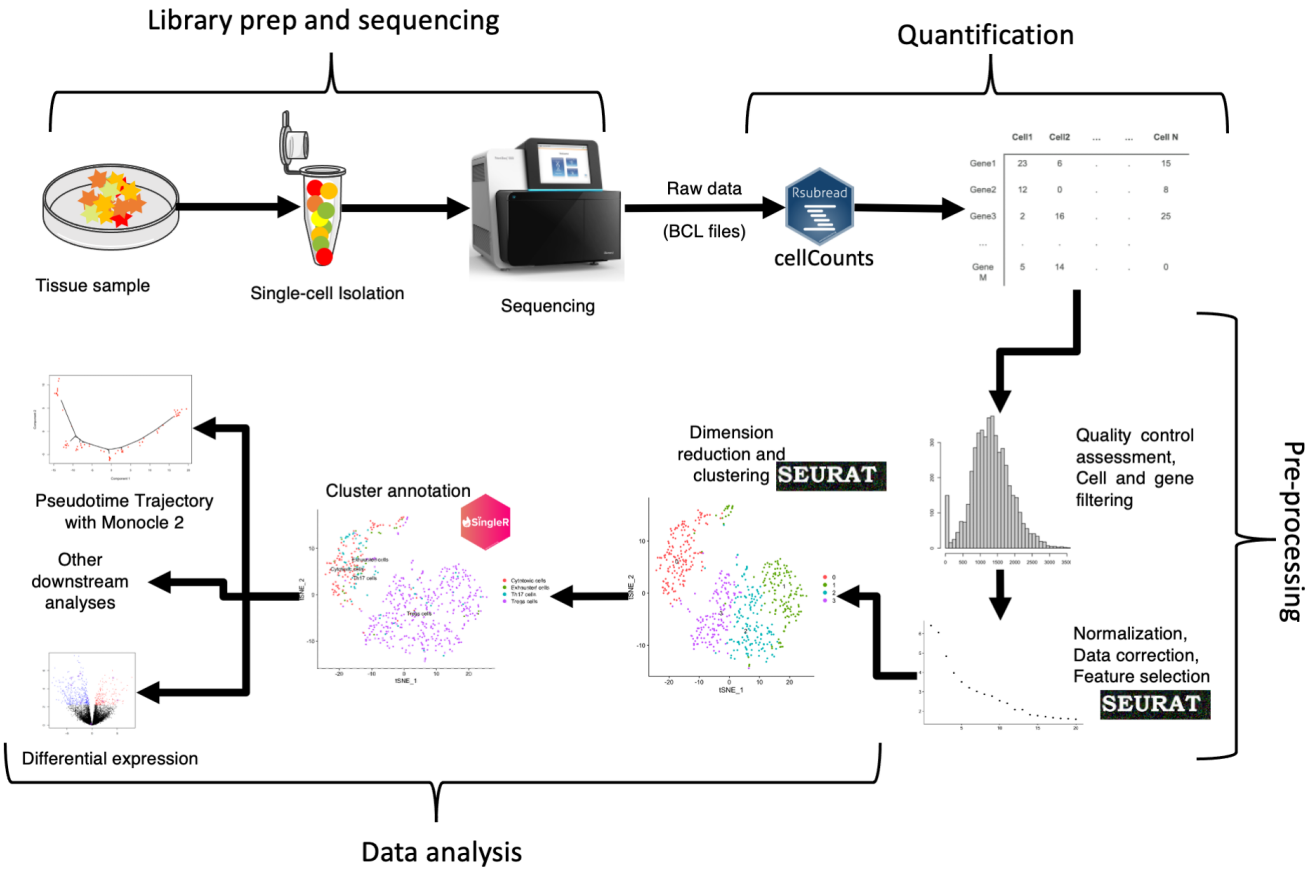


Figure 2.1: Overview of the scRNA-seq analysis protocol

2.1 Quantification

Converting the raw data into a matrix of counts is the first step in a scRNA-seq analysis. *cellCounts* a scRNA-seq quantification program within **Rsubread** is used in the quantification step. **cellCounts** is a program developed for quantifying single-cell RNA-seq (scRNA-seq) data generated from the 10X platform. cellCounts takes as input scRNA-seq reads, maps them to the reference genome and then produces UMI (Unique Molecular Identifier) counts for each gene in each cell. Its read mapping is based on the **align** program (Liao et al., 2019, 2013b) and its UMI counting is based on the *featureCounts* program (Liao et al., 2013a). Sample demultiplexing, cell barcode demultiplexing and read deduplication are carried out before generating the UMI counts.

The *cellCounts* function returns a R list object which contains the following components:

counts
A List object including UMI counts for each sample. Each component in this object is a matrix that contains UMI counts for a sample. Rows in the matrix are genes and columns are cells.

annotation
A data.frame object containing a gene annotation. This is the annotation that was used for the assignment of UMIs to genes during quantification. Rows in the annotation are genes. Columns of the annotation include GeneID, Chr, Start, End and Length.

sample.info
A data.frame object containing sample information and quantification statistics. It includes the following columns: SampleName, InputDirectory (if the input format is BCL), TotalCells,HighConfidenceCells (if umi.cutoff is NULL), RescuedCells (if umi.cutoff is NULL), TotalUMI, MinUMI, MedianUMI, MaxUMI, MeanUMI, TotalReads, MappedReads and AssignedReads. Each row in the data frame is a sample.

cell.confidence
A List object indicating if a cell is a high-confidence cell or a rescued cell (low confidence). Each component in the object is a logical vector indicating which cells in a sample are high-confidence cells. cell confidence is included in the output only if umi cutoff is NULL.

2.2 Quality control and Normalization

This step largely consists of quality control, normalization, data correction and dimension reduction. For this step, we will use visualization plots and utilize a number of functions within the **Seurat** (Hao et al., 2021) pipeline for scRNA-seq analysis.

2.2.1 Quality control

Cell filtering

Quality control is performed in several ways to remove;

- low quality cells
- contaminants

A number of metrics are used to assess the quality of the scRNA-seq data including the proportion of detected genes in each cell, and fraction of UMI accounts associated with Mitochondrial/Ribosomal genes.

Cells with a low or high number of detected genes are indicative of

- quiescent/damaged cells or
- doublets/multiplets (Nayak and Hasija, 2021)

A high fraction of mitochondrial genes in a cell is associated with (See 10X website)

- poor sample quality which leads to a high fraction of apoptotic or lysing cells.
- biology of the particular sample, for example, tumor biopsies, may have increased mitochondrial gene expression due to metabolic activity and/or necrosis.

A high fraction of ribosomal genes in each cell is indicative of RNA degradation which leads to more templating of rRNA-fragments (See 10X website)

To decide what cut-offs to use, we will visualize the distributions of each metric using plots such as histograms and box plots.

Gene filtering

Genes not meeting the following criteria are also filtered;

- Genes not expressed in at least a given number of cells
- Genes that may have very high expression
- Mitochondrial encoded genes
- Ribosomal encoded genes
- Genes deemed to induce technical bias

2.2.2 Normalisation and data correction

The filtered count matrix is normalized to account for gene expression variability between cells. There are several normalization methods, a number of which have been derived from Bulk RNA-seq and include;

- CPM (Counts Per Million)
- TPM (Transcripts Per Million)
- FPKM (Fragments Per Kilobase Million)
- TMM (Trimmed Mean of M-values)
- Quantile normalization

The sparsity and high heterogeneity in scRNA-seq adds to the complexity of normalizing scRNA-seq data. Here, we will utilize normalization methods included in the **Seurat** package. We will be using a global-scaling normalization method "LogNormalize" in **Seurat**, this method 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. In addition, for Differential Expression analysis, we utilize the the limma-voom pipeline (Ritchie et al., 2015, law2014voom) and here, we will be using the CPM normalization method.

2.2.3 Dimension reduction

The resulting count matrix from scRNA-seq is high dimensional in nature. Furthermore, not all the genes included in the matrix are needed for the classification of the cellular expression profiles. As such, it is important that we focus on the Biological signals that are informative. Popular techniques for dimension reduction are;

- PCA (Principal Component Analysis)
- t-distributed Stochastic Neighbor Embedding (t-SNE)
- Uniform Manifold Approximation, and Projection (UMAP)
- Self Organizing Maps (SOM)

There are two components involved in dimension reduction;

- Feature selection where a small subset of features is selected
- Feature extraction where higher dimension data is projected to a lower dimension

In this workshop, t-SNE and UMAP are used to demonstrate how to group cells into clusters.

2.3 Data analysis

After the data has been processed, there are several analyses that can be done depending on the research question. In this tutorial, we will look at the standard analysis in scRNA-seq such as Cluster analysis, cell type annotation, and Trajectory inference/pseudo-temporal ordering. We will also perform gene-level analysis including differential expression analysis, pathway and gene set enrichment analysis, and the generation of Gene Regulatory Networks

Chapter 3

Prerequisites

Before you get started with the rest of the analysis, it is important that you have the necessary data and software that will be used in this analysis.

3.1 Data

scRNA-seq data

A subset of the scRNA-seq dataset that was generated in a published study (Chen et al., 2020) is used as an example dataset in this protocol. The data consists of two well-characterized cellular reference samples (human breast cancer cell line (HCC1395, sample A) and the matched normal B lymphocyte line (HCC1395BL, sample B)) that were captured using the 10X platform. The data is available in the SRA repository under accession code no. PRJNA504037. However, for the convenience of this analysis, FASTQ files of the raw scRNA-seq data were also saved in the 'Workshop_scRNAseq' directory on the Z drive.

A text file called "targets.csv", which includes relevant sample information can also be found in this directory.

Reference genome data

A FASTA-format file including all chromosomal sequences of the GRCh38/hg38 genome was also saved in the 'Workshop_scRNAseq' directory on Z drive.

3.2 SOFTWARE

The following software tools should be installed on the UNIX server and on your laptop:

- R
- Rsubread
- limma
- edgeR
- org.Hs.eg.db
- statmod
- Seurat
- SingleR
- monocle

Consult the R Project website for the installation of R (https://www.r-project.org/) (R Core Team, 2022). Make sure the latest release version of R is downloaded and installed. After R is installed, launch R and type the following commands to install *Rsubread*, *limma*, *edgeR*, *org.Hs.eg.db*, *statmod*, *Seurat*, *SingleR* and *Monocle*:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(
  c(
    'Rsubread',
    "org.Hs.eg.db",
    "SingleR",
    'BiocGenerics',
    'DelayedArray',
    'DelayedMatrixStats',
    'S4Vectors',
    'SingleCellExperiment',
    'limma',
    'edgeR',
    'SummarizedExperiment',
    'batchelor',
    'Matrix.utils',
    'monocle',
    'celldex'
  ),
  update = T
)

if (!requireNamespace("statmod", quietly = TRUE))
```

```
  install.packages("statmod")
if (!requireNamespace("Seurat", quietly = TRUE))
  install.packages("Seurat")
```

Alternatively, you may use Rstudio to run this protocol.

Chapter 4

Running Environment

4.1 RStudio

This is the recommended approach for running this protocol. An RStudio application ('RStudio(2672)') has been created under the 'Analysis' tab in the Remote Access Facility. After you log into it (using your email account name and password), change to the directory '/data/RawPrimary/Public' (you can do this by choosing Session > Set Working Directory >Choose Directory and then using the '...' to specify the directory you want to change to) and then you will find a folder called 'Workshop_scRNAseq' which includes all the materials included in this Workshop. You can make a copy of this folder and then start to run the protocol in your own folder.

4.2 UNIX server + laptop

Run read mapping and counting on a UNIX server and then perform the rest of the analysis on your laptop. Refer to the document "How to Access Linux Bioinformatics Analysis Platform.pdf" for how to access the Bioinformatics UNIX server. Once you logged in to the server, you can issue the following commands to copy the Workshop data to a directory you create on the server and to launch R. You are then ready to run the protocol.

```
# connect AllStaffShare Drive to the server
cifscreds add svr-fs95
```

```
# change directory to the Public directory
cd /data/Processing/Public/
```

```
# create your own directory, eg 'my_directory'
mkdir my_directory
```

```
# copy the Workshop data to your directory
cp -r /mnt/AllStaffShare/Workshop_scRNAseq my_directory
```

```
# Change directory to the Workshop_scRNAseq materials
cd my_directory/Workshop_scRNAseq
```

R has already been installed for you, use the *module load R* command to activate **R**

```
# activate R environment
module load R
```

```
#Start R environment
R
```

Due to the complexity and resource requirements of scRNA-seq data analysis, the quantification step will be run on the server and the rest of the analysis can be run on your laptop.

Chapter 5

Quantification

5.1 Build index for a reference genome

Start an R session using the terminal on the server

```
module load R
R
```

First change to the working directory 'Workshop_scRNAseq' using the command below, remember to replace 'my_directory' with the name of the directory you created in 4.2.

```
setwd("/data/Processing/Public/my_directory/Workshop_scRNAseq/")
```

Build an index for the reference genome *GRCh38/hg38* using the *buildindex* function in **Rsubread**. The created index files will be saved to the current working directory. This index only needs to be built once and it can be reused in future RNA-seq (both bulk and single) data analyses.

```
library(Rsubread)
```

```
buildindex(basename = "hg38_reference",
           reference = "GRCh38.primary_assembly.genome.fa.gz")
```

5.2 Map and quantify single cell RNA-seq data

The raw scRNA-seq data is processed and UMI counts generated using *cellCounts*. It takes as input scRNA-seq reads generated by the 10X platform,

maps them to the reference genome and then produces UMI (Unique Molecular Identifier) counts for each gene in each cell. It uses the *align* read mapping function and the *featureCounts* quantification function, both part of the **Rsubread** package.

For more information on how to use *cellCounts*, type the command `?cellCounts` in the R console.

Create sample-related information

Before using cellCounts, we need to generate a data frame containing sample information which we then pass to cellCounts via the sample parameter.

Note that for this workshop, *the input format for our dataset is FASTQ* and as such we create a data frame that consists of 3 columns 'BarcodeUMIFile', 'ReadFile' and 'SampleName'. However, if the input format were BCL (ie. input.mode="BCL"), we would have created a dataframe that included the location where the read data are stored, flowcell lanes used for sequencing, sample names and names of index sets used for indexing samples.

```
umiread <- list.files("FQs",
                      pattern = "^.*_1.fastq.gz",
                      full.names = T)
readfile <- gsub("1.fastq.gz", "2.fastq.gz", umiread)
samples <- gsub("10X_|_1.fastq.gz", "", basename(umiread))
sample.sheet <-
  data.frame(
    BarcodeUMIFile = umiread,
    ReadFile = readfile,
    SampleName = samples,
    stringsAsFactors = F
  )
```

Align and count reads

Sequence reads are mapped to the human genome (GRCh38/hg38) and UMIs are counted to each gene using the inbuilt human (*hg38*) annotation. The *cellCounts* function returns a list object to R. It also outputs one BAM file for each sample which includes location-sorted read mapping results. The returned R list object is saved as an R object "counts.RData", this will then be loaded to perform further downstream analysis.

```
counts <-
  cellCounts(
    index = "hg38_reference",
    sample = sample.sheet,
    annot.inbuilt = "hg38",
    input.mode = "FASTQ",
```

```
    nthreads = 20
  )
save(counts, file = "counts.RData")
```

Chapter 6

Analysis protocol

6.1 Read mapping and UMI counts

As discussed in section 5.2, de-multiplexing, alignment to the GRCh38 transcriptome and unique molecular identifier (UMI)-collapsing were performed using *cellCounts*. Here we load the saved counts R object.

```
load("counts.RData")
names(counts)
```

```
## [1] "counts"          "cell.confidence" "annotation"      "sample.info"
```

The information below provides a summary of the quantification stats returned by *cellCounts*.

```
counts$sample.info
```

```
##    SampleName TotalCells HighConfidenceCells RescuedCells TotalUMI MinUMI
## 7  LLU_A_Seq1       2954                2691          263 59485648    506
## 12 LLU_B_Seq1       1451                1305          146 33887743    504
##    MedianUMI MaxUMI  MeanUMI TotalReads MappedReads AssignedReads
## 7      18718  91172 20137.32  132228861   106079174      83649022
## 12     22124  99487 23354.75  134624084   105735336      78858247
```

```
#rename sample name
```

In addition, the sample information discussed in 3.1 is loaded as shown by the details below.

```
sample.info <- read.csv("sample_info.csv",stringsAsFactors = F)
sample.info
```

```
##   Library.Name                                  cell_line Sample.Name    sex
## 1   LLU_A_Seq1         breast cancer cell line (HCC1395)     HCC1395 female
## 2   LLU_B_Seq1 normal B lymphocyte cell line (HCC1395BL)   HCC1395BL female
##   tissue
## 1 breast
## 2  blood
```

The counts are transformed to a **Seurat** object using the *CreateSeuratObject* function. Entrez gene IDs are also converted to their corresponding gene symbols.

```
#load Seurat package
library(Seurat)
#load annotation package
library(org.Hs.eg.db)
#Generate a list mapping gene Ids to symbols
ID2symbol.lst<-as.list(org.Hs.egSYMBOL)
head(rownames(counts$counts$LLU_A_Seq1))
```

```
## [1] "100287102" "653635"    "102466751" "100302278" "645520"    "79501"
```

```
#Get list of Mitochondrial genes to be used for calculating percentage
#of UMIs assigned to Mitochondrial genes in each cell
genes.MT <- unlist(ID2symbol.lst[as.list(revmap(org.Hs.egCHR))$MT],
                   use.names = F)
```

```
#Create list of Seurat objects
counts_st <- sapply(counts$sample.info$SampleName, function(x) {
  x.sample <- sample.info[sample.info$Library.Name == x,]
  #get counts from cellCounts
  x.counts <- counts$counts[[x]]
  #Change gene IDs to symbols
  x.genes <- ID2symbol.lst[rownames(x.counts)]
  rownames(x.counts)[rownames(x.counts) %in% names(ID2symbol.lst)] <-
    unlist(ID2symbol.lst[rownames(x.counts)], use.names = F)
  #create Seurat object
  x.counts <- CreateSeuratObject(x.counts,
                                 min.cells = 0)
  #Calculate percentage of UMIs assigned to Mitochondrial genes in each Cell
  x.counts[["percent.MT"]] <-
```

```
    PercentageFeatureSet(x.counts, features = genes.MT)

  #Calculate percentage of UMIs assigned to Ribosomal genes in each Cell
  x.counts[["percent.Ribo"]] <-
    PercentageFeatureSet(x.counts, pattern = "^RP[SL]")
  #Add Sample metadata
  x.counts[["cell.line"]] <- x.sample$cell_line
  x.counts[["tissue"]] <- x.sample$tissue
  x.counts[["Sample"]] <- x.sample$Sample.Name
  return(x.counts)
}, simplify = F)
#Merge Seurat objects
counts_st <- merge(
  counts_st$LLU_A_Seq1,
  counts_st$LLU_B_Seq1,
  add.cell.ids = counts$sample.info$SampleName,
  merge.data = TRUE
)
counts_st
```

```
## An object of class Seurat
## 28395 features across 4405 samples within 1 assay
## Active assay: RNA (28395 features, 0 variable features)
```

6.2 Quality control checks

6.2.1 Mitochondria content

We first look at the distribution of the Mitochondrial content to decide on the cut-off for filtering. The histogram below shows the distribution of the percentage of mitochondria content across all cells.

```
samples <- unique(counts_st@meta.data$Sample)
hist(
  counts_st@meta.data$percent.MT,
  main = "Mitochondria content across all samples",
  cex.main = 1,
  cex.lab = 0.8,
  xlab = "Percentage"
)
```

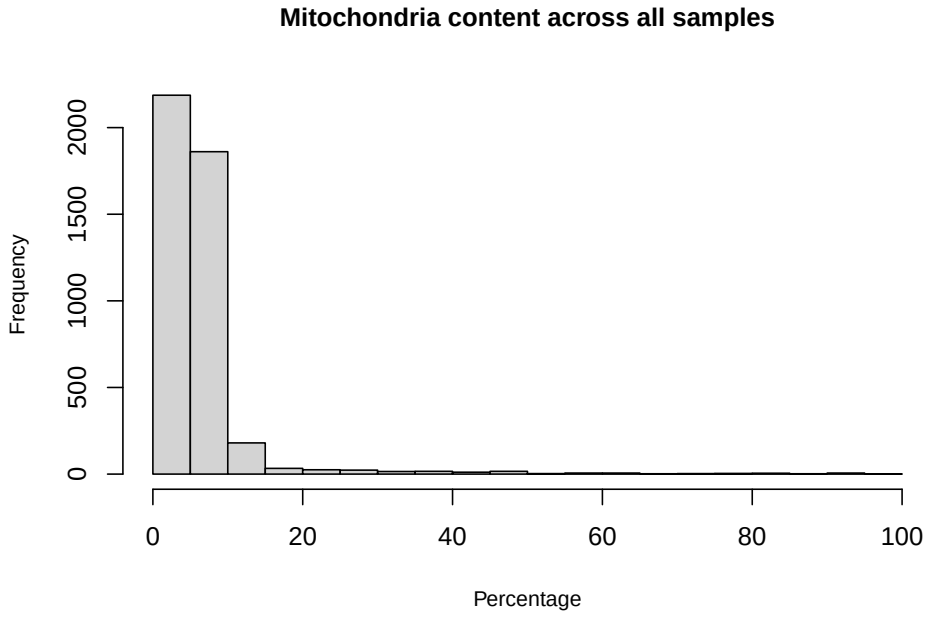
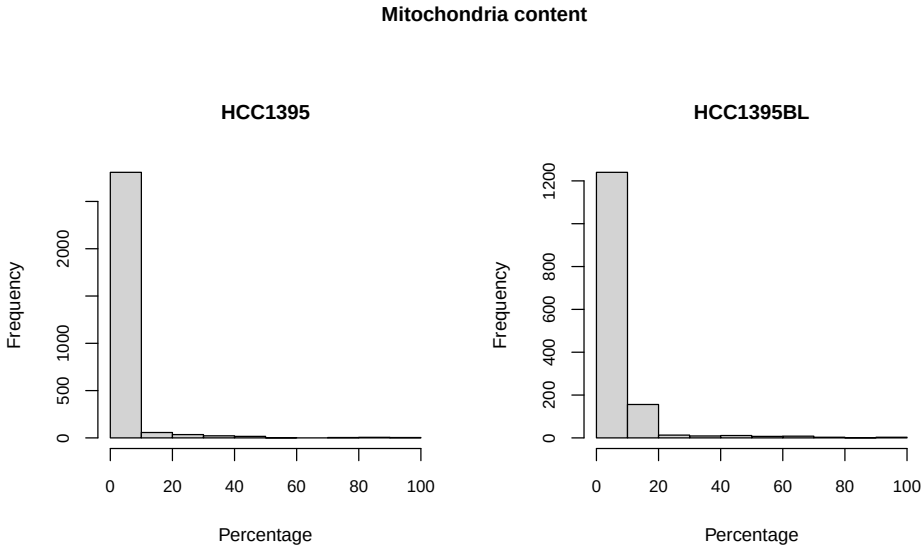


Figure 6.1: Percentage of mitochondria

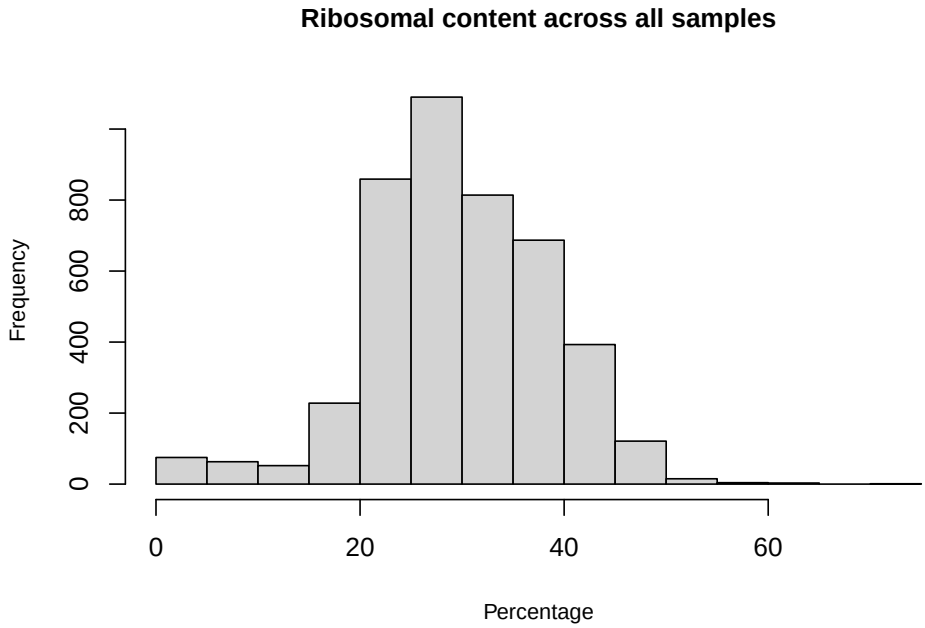
```
par(mfrow = c(1,2), oma = c(0, 0, 4, 0))
for (s in samples)
{
  hist(
    subset(counts_st@meta.data, subset = Sample == s)$percent.MT,
    main = paste0(s),
    xlab = "Percentage",
    cex.main = 1,
    cex.lab = 0.9,
    cex.axis = 0.8
  )
}
title(main = "Mitochondria content",
      cex.main = 1,
      outer = T)
```



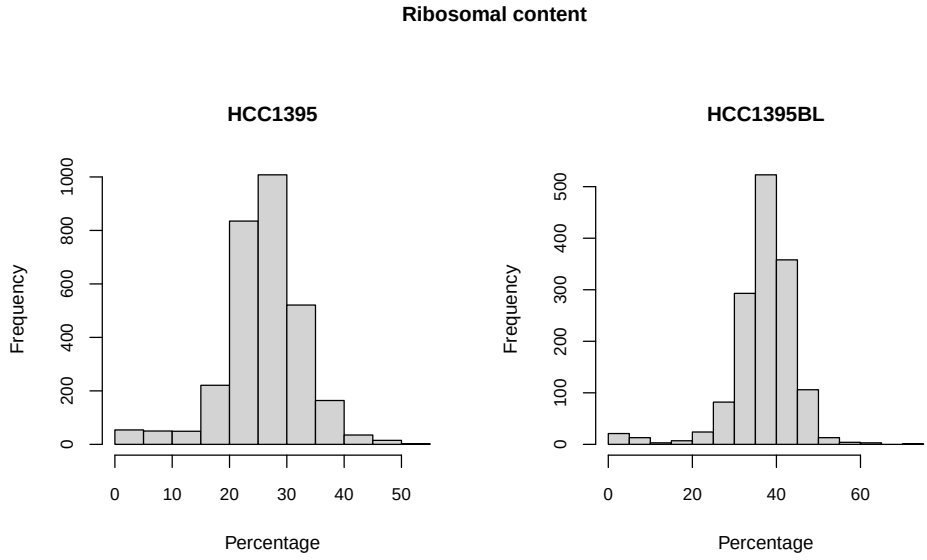
6.2.2 Ribosomal content

The histograms below show the distribution of the Ribosomal content across all samples.

```
hist(
  counts_st@meta.data$percent.Ribo,
  xlab = "Percentage",
  cex.main = 1,
  main = "Ribosomal content across all samples",
  cex.lab = 0.8
)
```



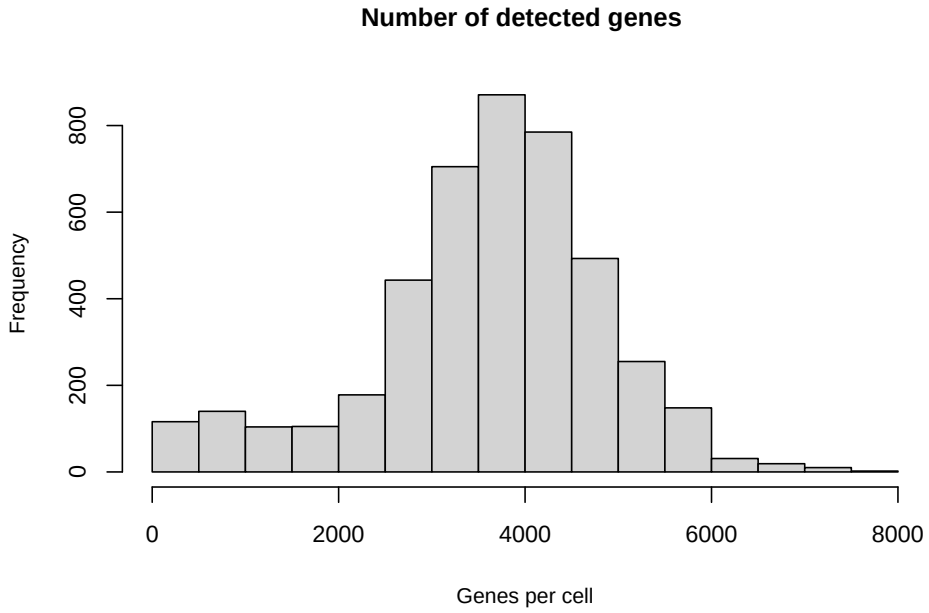

```
par(mfrow = c(1, 2), oma = c(0, 0, 4, 0))
for (s in samples)
{
  hist(
    subset(counts_st@meta.data, subset = Sample == s)$percent.Ribo,
    main = s,
    xlab = "Percentage",
    cex.main = 1,
    cex.lab = 0.9,
    cex.axis = 0.8
  )
}
title(main = "Ribosomal content",
      cex.main = 1,
      outer = T)
```



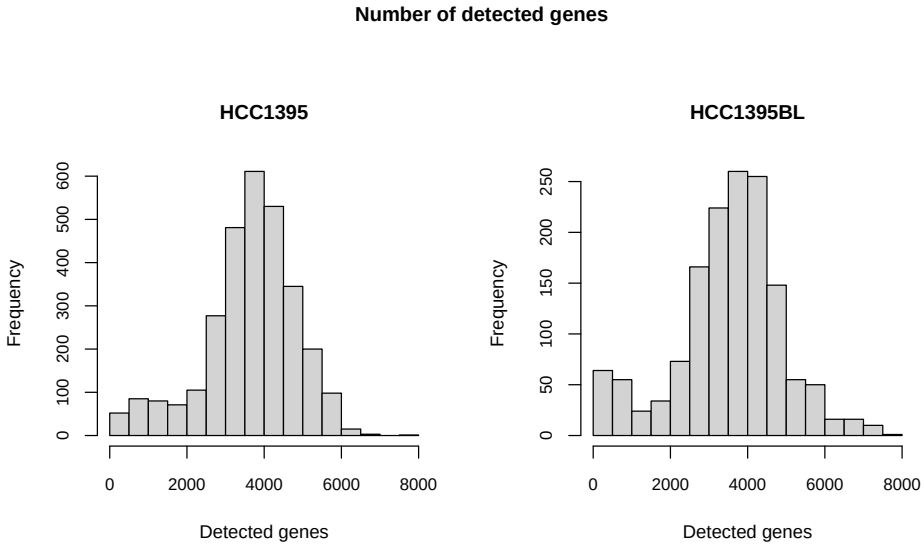
6.2.3 Number of detected genes

The histogram plot below shows the distribution of the number of detected genes across cells from all samples.

```
hist(
  counts_st@meta.data$nFeature_RNA,
  main = "Number of detected genes ",
  xlab = "Genes per cell",
  cex.main = 1,
  cex.axis = 0.9,
  cex.lab = 0.8
)
```




```
par(mfrow = c(1, 2), oma = c(0, 0, 4, 0))
for (s in samples)
{
  hist(
    subset(counts_st@meta.data, subset = Sample == s)$nFeature_RNA,
    main = s,
    xlab = "Detected genes",
    cex.main = 1,
    cex.lab = 0.9,
    cex.axis = 0.8
  )
}
title(main = "Number of detected genes",
      outer = T,
      cex.main = 1)
```



6.3 Filtering

6.3.1 Cell filtering

```
#Filters used
min.exp_genes<-200 #minimum number of detected genes in each cell
max.exp_genes<-6000 #Maximum number of detected genes in each cell
percent.mito <- 20 #Percentage of mitochondrial content
percent.ribos <- 40 # Max percentage of Ribosomal content
```

Based on the quality control checks above, the following filters are used to exclude cells that did not meet the criteria from the rest of the analysis

- cells with <200 detected genes
- cells with >6000 detected genes
- cells with >20 of Mitochondria content per cell
- cells with > 40 of Ribosomal content per cell

Number of cells before filtering for mitochondrial and ribosomal content: 4,405

```
#exclude cells with a high content of mitochondrial and ribosomal content
counts_st <-subset(counts_st, subset = percent.MT < percent.mito &
           percent.Ribo < percent.ribos)
```

Number of cells after filtering for mitochondria and ribosomal content: 3,731
Next we exclude cells that had less than 200 and greater than 6,000

```
#exclude cells with low or high number of detected features
counts_st <-subset(counts_st,subset = nFeature_RNA >= min.exp_genes & nFeature_RNA <= r
```

Number of cells remaining after filtering for cells with low number of detected genes: 3,678 Number of cells left per sample

HCC1395 HCC1395BL 2794 884

6.3.2 Gene filtering

Genes that failed to express (an expressed gene has at least 1 UMI count) in at least 3 cells in at least 1 sample together with Mitochondrial and Ribosomal genes are excluded from the rest of the analysis;

Number of genes before filter: 28,395

```
#Check genes expressed in at least 3 cells in at least one of the samples
keep.genes <- rowSums(do.call(cbind, lapply(samples, function(x)
  rowSums(subset(counts_st, Sample == x)@assays$RNA@counts > 0) >= 3))) >
  0
table(keep.genes)
```

```
## keep.genes
## FALSE  TRUE
## 10516 17879
```

```
counts_st <- counts_st[keep.genes, ]
```

Number of genes after filtering for genes that failed to express in at least 3 cells in at least 1 sample: 17,879

Mitochondrial and Ribosomal genes are also excluded from the rest of the analysis.

```
#filter ribosomal and mitochondrial genes
counts_st <-
  counts_st[!(rownames(counts_st) %in% genes.MT) &
       !grepl("^RP[SL]", rownames(counts_st)),]
```

Number of genes after filtering for mitochondrial and ribosomal genes: 17,711

6.4 Normalization

After removing unwanted cells from the dataset, the data was normalized using 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.

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

6.5 Dimension reduction and clustering

A subset of highly variable genes between cells were calculated using *FindVariableFeatures* function within Seurat.

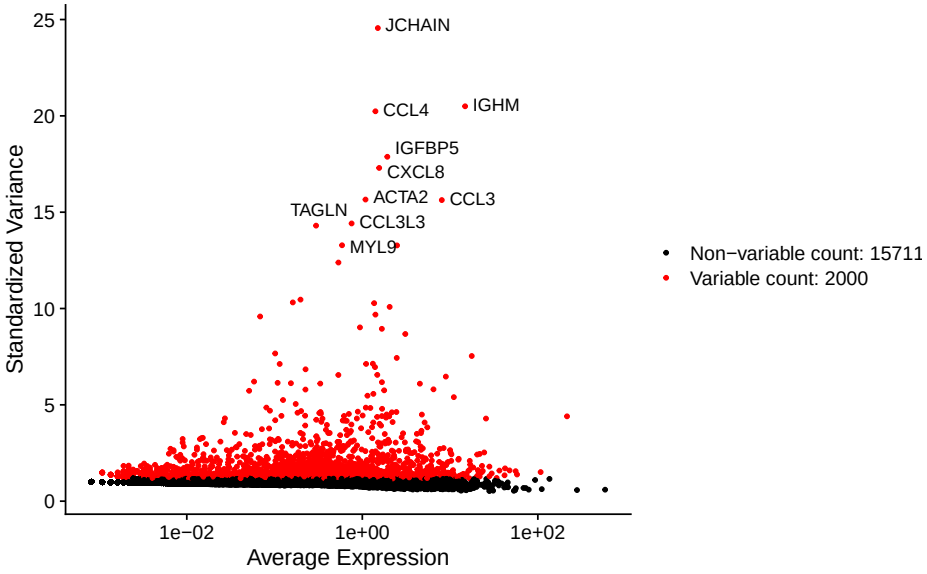
```
#Identification of highly variable features (feature selection)
counts_st <- FindVariableFeatures(object = counts_st )
```

The scatter plot below shows the top 10 identified variable genes and their average expression against variance

```
# Identify the 10 most highly variable genes
top10 <- head(VariableFeatures(counts_st), 10)
# plot variable features with labels
plot1 <- VariableFeaturePlot(counts_st)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
```

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

```
plot2
```



A linear transformation ('scaling') was applied to the normalized data prior to dimensional reduction techniques using the *ScaleData* function in **Seurat**.

```
counts_st <-  ScaleData(
    object = counts_st,
    features = rownames(counts_st),
    verbose = F)
```

Principal component analysis was performed using the variable genes identified above.

```
counts_st <-RunPCA(
    counts_st,
    features = VariableFeatures(object = counts_st))
```

```
## PC_ 1
## Positive:  S100A6, S100A10, TNFRSF12A, TSPO, TPM2, CAV1, TPM1, CALD1, ANXA2, RHEB
##     ITGB1, LARP6, NQO1, HSBP1, SERPINE2, MGST1, FTH1P3, HSPB1, CYP1B1, KRT81
##     DEF8, TSC22D1, LY6K, TNNT1, TGFBI, CAV2, CALU, DSTN, CCND1, TUBB3
## Negative:  CD74, HLA-DRA, MS4A1, HLA-DPA1, LAPTM5, ARHGDIB, HLA-DPB1, IGHM, LSP1, LI
##     MIR155HG, CD70, CCND2, PDLIM1, CORO1A, BST2, CD79A, SNCA, CCL3, CD48
##     RAC2, SRGN, TMSB4X, IGLC2, PLEK, CCR7, HCLS1, TNFRSF13B, CD53, IRF4
## PC_ 2
## Positive:  NME7, NDRG1, GADD45A, WIPI1, YPEL5, H2BC12, LURAP1L, GABARAPL1, GDF5, ZNF
##     ULK4, MYLK, KRT16, CCN1, FN1, LINC01315, NUPR1, BEX2, DNAJB9, PLK2
##     SGK1, KRT75, CCN2, FBLIM1, MYC, MAP1LC3B, SELENOM, LINC01133, DUSP10, IL1RL1
```

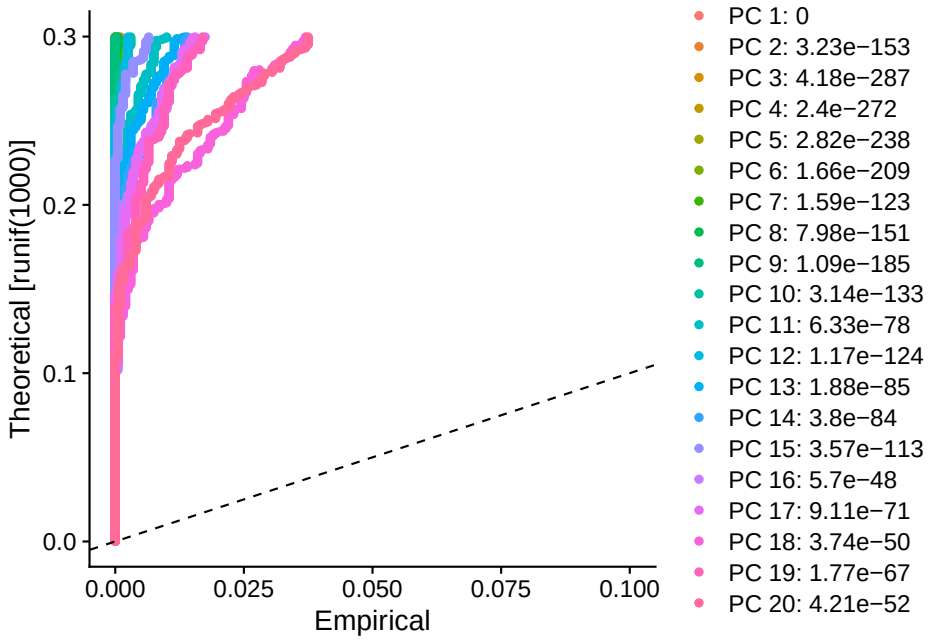
```
## Negative:  MKI67, CENPF, PLK1, HMGB2, CDC20, AURKA, CDCA3, UBE2C, CENPA, CCNA2
##     GTSE1, HMMR, PTTG1, TPX2, NUSAP1, SMC4, TOP2A, CKS2, CENPE, CCNB1
##     CDKN3, BIRC5, PRC1, CCNB2, ARL6IP1, DEPDC1, SGO2, ASPM, UBE2S, KPNA2
## PC_ 3
## Positive:  ULK4, ASPM, MKI67, C1orf56, CENPF, JCHAIN, TOP2A, GTSE1, NUSAP1, MXD3
##     HMGB2, CDKN3, KIF14, CENPA, IGLL1, HMMR, IGLC7, BIRC5, PHKG1, IGLC5
##     ITGA4, PRDM1, CENPE, CKAP2, TPX2, PRR11, DLGAP5, CDCA3, PIF1, IGLL5
## Negative:  GADD45A, MARCKSL1, LAPTM4A, FDPS, ATP6V0B, PSMD8, PSMC2, HLA-C, PRMT1, TAGLN2
##     CD44, TMBIM6, PAK1IP1, LGMN, DNAJA1, UBB, XRCC6, ENO1, TIMP1, RIPK2
##     HACD3, DCAF13, UBC, TXNL1, HCCS, ADH5, CCT4, RTCB, DNAJB6, PGK1
## PC_ 4
## Positive:  ZFP36L1, DEK, BASP1, H4C3, STAG3, C12orf75, CCL22, SCG5, SERTAD4-AS1, CD82
##     ATAD2, TNFSF4, CENPX, PRRX1, ARL4C, HCFC1R1, TNFAIP3, RAC3, FAM20C, DUSP4
##     GTSE1, FTH1P3, ANKRD33B, SOX4, MARCKSL1, TNNT1, ANKRD11, PRR11, SMIM14, RUNX3
## Negative:  UBB, PRDX1, PGK1, PSMA4, ENO1, PSMA3, EIF4A3, TMBIM6, ERP44, JCHAIN
##     DCAF13, NPM1, CCT8, XRCC6, PSME2, IGLL1, IGLC7, IGLC5, MDH1, HSPA5
##     HSPD1, HSP90AB1, CCT4, CCT5, HSP90B1, PSMC2, UBC, IGLL5, PRMT1, DNAJB11
## PC_ 5
## Positive:  C12orf75, JCHAIN, IGLC7, IGLL1, IGLC5, IGLL5, FN1, FTL, ITGA4, SPHK1
##     PRDM1, TNFRSF17, GLCCI1, SSR3, IGLL3P, RASSF6, GUSBP11, LINC01480, CLEC2B, ST6GAL1
##     GLIPR1, ISG15, XBP1, POU2AF1, FAM107B, SDF2L1, RRAGD, CCDC69, MEF2C, CYTOR
## Negative:  MMP7, LTA, AKR1C2, WDR91, BCL2A1, CCL22, AKR1C1, PRDX1, ENO1, SSTR2
##     HSP90AB1, LDHA, AKR1C3, TRAF1, TNFRSF8, STAG3, PKM, HSPD1, EBI3, AKR1B10
##     MUC12-AS1, LINC00158, ICAM1, HSP90AA1, ZBTB32, RHOV, AKR1C4, NFKBIA, CCT5, BCAT1
```

We use the *JackStraw* function in **Seurat** to establish the dimensionality of the dataset. This is done by randomly selecting a subset of genes and their significance in a computed PC.

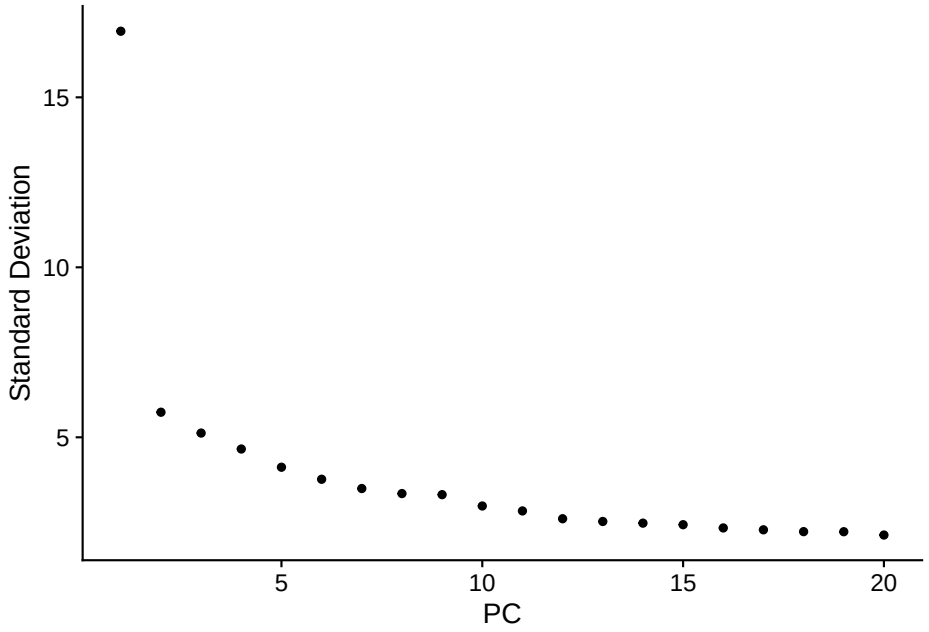
```
counts_st <- JackStraw(counts_st, verbose = F, dims = 20)
counts_st <-
  ScoreJackStraw(counts_st, reduction = "pca", dims = 1:20)
```

```
JackStrawPlot(counts_st, reduction = "pca", dims = 1:20)
```

```
## Warning: Removed 28000 rows containing missing values (geom_point).
```




```
ElbowPlot(counts_st)
```



```
res<-0.5
sel.dims<-10
```

10 dimensions are used to establish the number of clusters with a resolution of 0.5

```
counts_st <-FindNeighbors(counts_st,
                reduction = "pca",
                verbose = F,
                dims = 1:sel.dims)
```

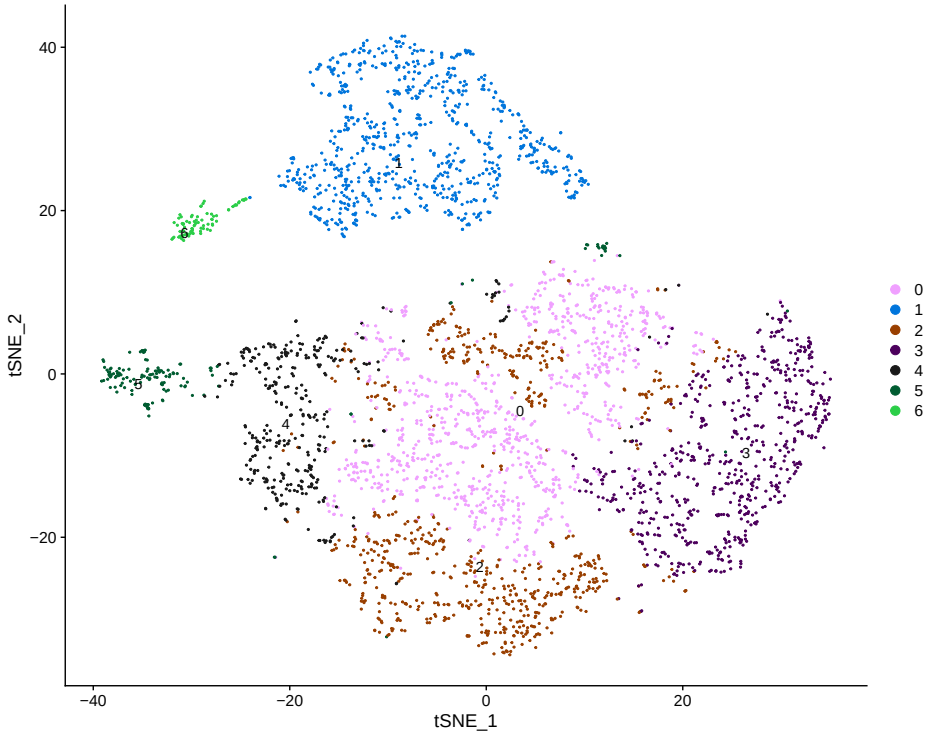
Next we run the **FindClusters** to identify clusters of cells using a resolution of 0.5.

```
counts_st <-
  FindClusters(
    object = counts_st,
    resolution = res,
    verbose = 0
  )
```

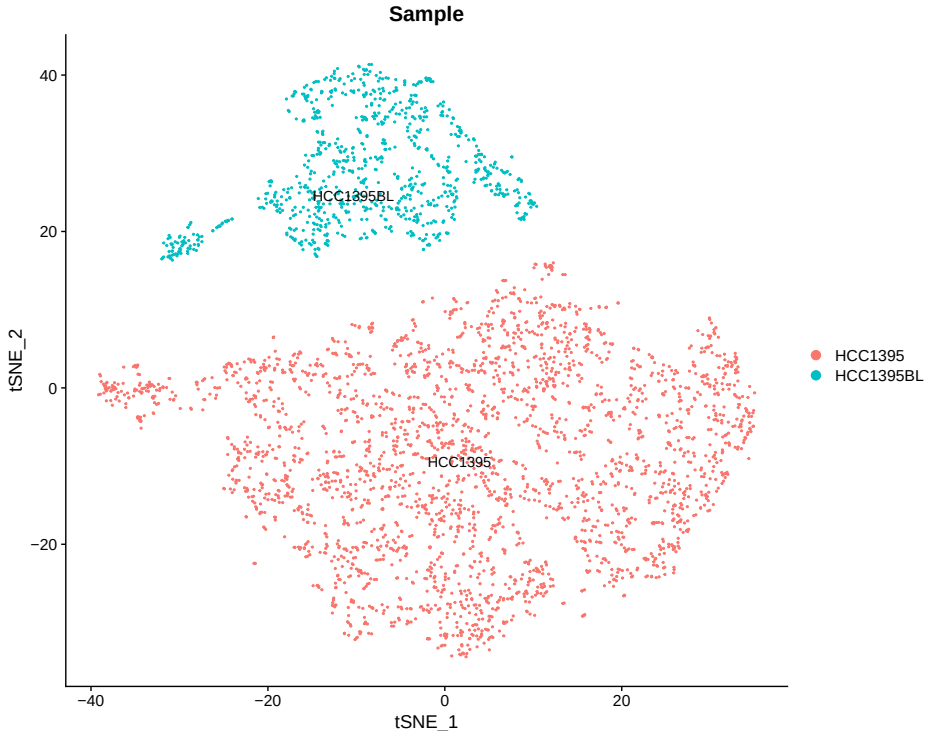
t-SNE dimensionality reduction on is run on the identified variable features.

```
counts_st <- RunTSNE(counts_st, dims = 1:sel.dims)
```

```
DimPlot(
  counts_st,
  reduction = "tsne",
  label = T,
  size = 0.5,
  repel = T,
  cols = DiscretePalette(length(levels(Idents(
    counts_st
  ))))
)
```




```
DimPlot(
  counts_st,
  reduction = "tsne",
  label = T,
  group.by = "Sample",
  size = 0.5,
  repel = T
)
```

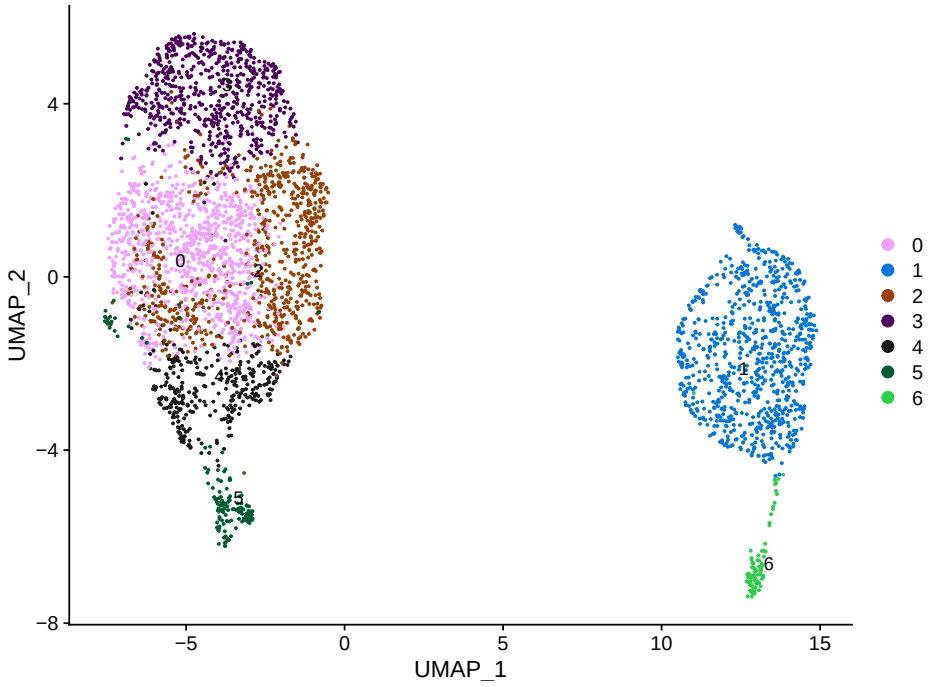


Optionally, we can also perform dimension reductionality reduction using UMAP

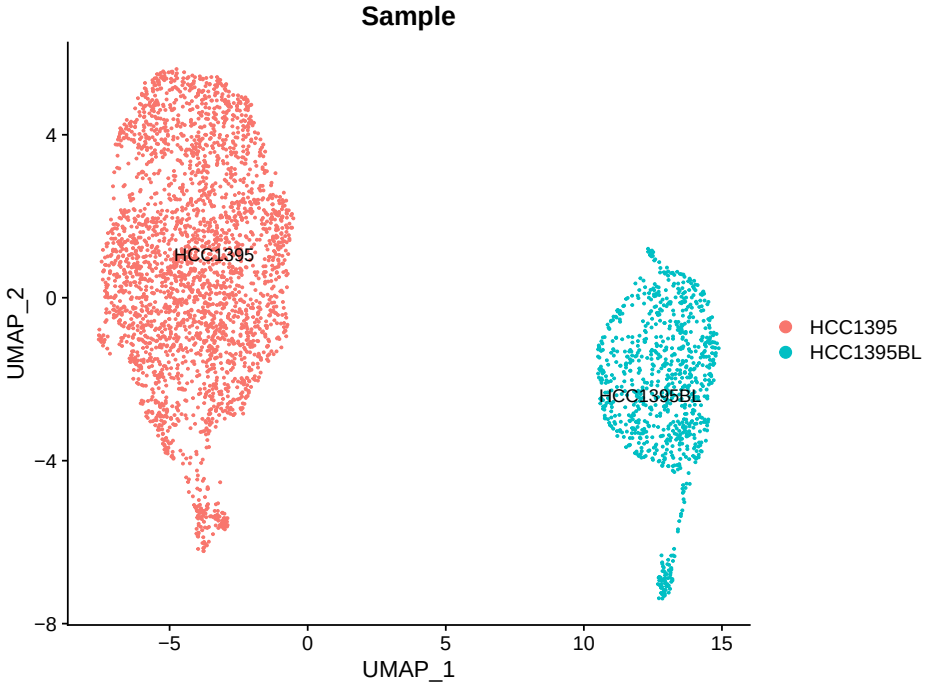
```
counts_st <- RunUMAP(counts_st, dims = 1:sel.dims,verbose = F)
```

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

```
DimPlot(
  counts_st,
  reduction = "umap",
  label = T,
  size = 0.5,
  repel = T,
  cols = DiscretePalette(length(levels(Idents(
    counts_st
  ))))
)
```




```
DimPlot(
  counts_st,
  reduction = "umap",
  label = T,
  group.by = "Sample",
  size = 0.5,
  repel = T
)
```



Differential expression analysis between clusters

To identify differentially expressed genes between clusters we can use the *FindMarker* function in Seurat. As an example, DE genes between clusters 0 and 1 are computed below.

```
DE.cluster0_1 <- FindMarkers(counts_st,
                             ident.1 = 0,
                             ident.2 = 1,
                             verbose = F)
head(DE.cluster0_1[order(abs(DE.cluster0_1$avg_log2FC), decreasing = T), ])
```

```
##                p_val avg_log2FC pct.1 pct.2     p_val_adj
## CD74    0.000000e+00  -5.891577 0.008 1.000  0.000000e+00
## IGHM    0.000000e+00  -5.729445 0.000 0.998  0.000000e+00
## S100A6  0.000000e+00   5.598281 1.000 0.047  0.000000e+00
## TMSB4X 1.657565e-278  -4.911549 0.970 1.000 2.935713e-274
## CCL3   6.580286e-308  -4.861987 0.000 0.945 1.165434e-303
## KRT81  4.273972e-296   4.593045 0.962 0.000 7.569632e-292
```

Marker genes which can be used to uniquely identify each of the clusters are identified using the *FindAllMarkers* function.

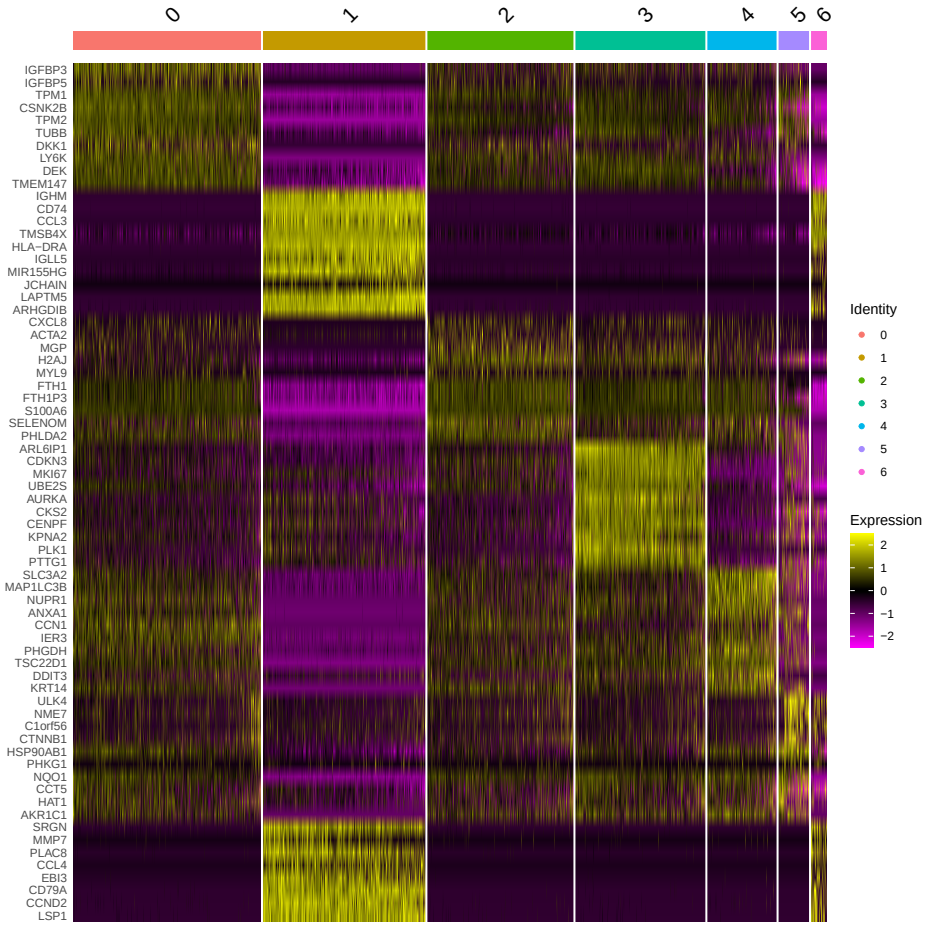
```
all.markers<-FindAllMarkers(counts_st,verbose = F)
```

```
library(dplyr)
all.markers %>%
    group_by(cluster) %>%
    slice_max(n = 5, order_by = avg_log2FC)
```

```
## # A tibble: 35 x 7
## # Groups:   cluster [7]
##        p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
##        <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 1.61e-155      1.23  0.94  0.525 2.86e-151 0       IGFBP3
##  2 4.07e- 65      1.17  0.417 0.149 7.21e- 61 0       IGFBP5
##  3 2.12e-147      1.06  0.998 0.714 3.75e-143 0       TPM1
##  4 2.31e-202      0.938 0.994 0.869 4.09e-198 0       CSNK2B
##  5 1.78e-131      0.901 0.998 0.665 3.15e-127 0       TPM2
##  6 0              4.79  0.998 0.021 0         1       IGHM
##  7 0              4.63  1     0.037 0         1       CD74
##  8 0              4.22  0.945 0.015 0         1       CCL3
##  9 0              4.08  1     0.959 0         1       TMSB4X
## 10 0              3.48  0.999 0.033 0         1       HLA-DRA
## # ... with 25 more rows
```

```
top10.markers<-all.markers %>%
    group_by(cluster) %>%
    slice_max(n = 10, order_by = avg_log2FC)
```

```
DoHeatmap(counts_st, features = top10.markers$gene)
```



6.6 Cell annotation

An unbiased cell type recognition is performed using **SingleR**. **celldex** has a range of annotations derived from Bulk RNA-seq data that can be used to annotate the identified clusters above. Here, we use the Human Primary Cell Atlas database as an example.

```
library(SingleR)
```

```
library(SingleCellExperiment)
cell.sce<- as.SingleCellExperiment(counts_st)
annot<-celldex::HumanPrimaryCellAtlasData()
cell.annots <- SingleR(
    test = cell.sce,
```

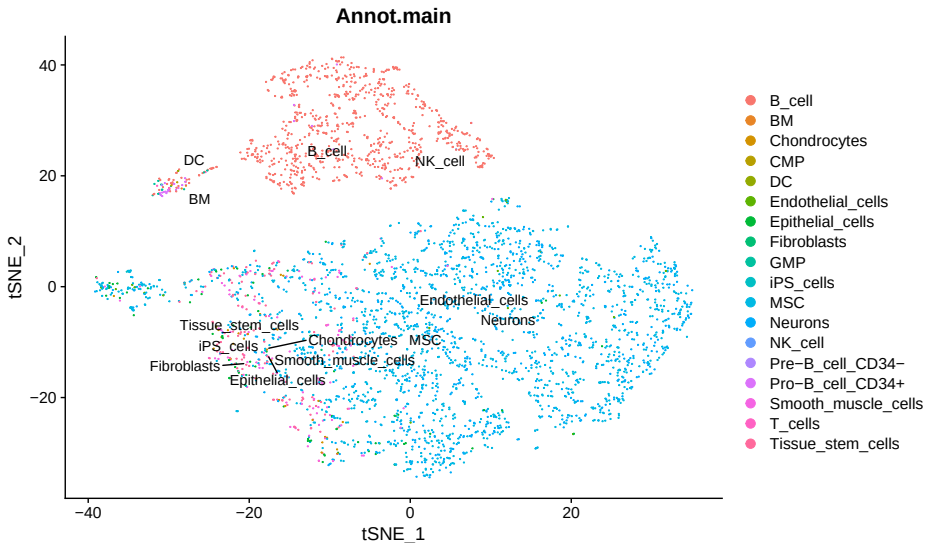
```
    ref = annot,
    labels = annot$label.main)

cell.annots.fine<-SingleR(
    test = cell.sce,
    ref = annot,
    labels = annot$label.fine)
```

```
counts_st <-
  AddMetaData(counts_st, cell.annots[rownames(counts_st@meta.data), "labels"], "Annot.m
```

```
DimPlot(
    counts_st,
    reduction = "tsne",
    group.by = "Annot.main",
    label = T,
    repel = T,
    pt.size = 0.1,
  )
```

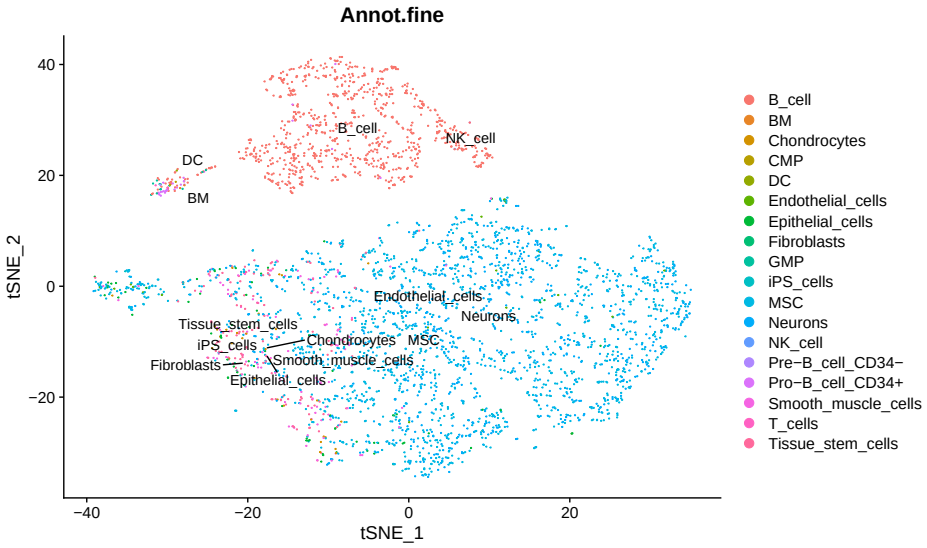
```
## Warning: ggrepel: 5 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```




```
counts_st <-AddMetaData(counts_st, cell.annots[
  rownames(counts_st@meta.data), "labels"], "Annot.fine")
```

```
DimPlot(
    counts_st,
    reduction = "tsne",
    group.by = "Annot.fine",
    label = T,
    repel = T,
    pt.size = 0.1)
```

```
## Warning: ggrepel: 5 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



6.7 Trajectory analysis

Pseudotime analysis of the tuft cells identified in the dataset was performed using **Monocle2**.

```
mono2.learn.traject <-
  function(X_counts,
           these.cell.types) {
    library(monocle) # It has to be Monocle 2.
    rds.fname <-"Trajectory-cds.rds"
    gsndf <- data.frame(gene_short_name = rownames(X_counts))
    csndf <- data.frame(cell.type = these.cell.types)
    rownames(gsndf) <- rownames(X_counts)
    rownames(csndf) <- colnames(X_counts)
```

```
    pd <- new("AnnotatedDataFrame", data = csndf)
    fd <- new("AnnotatedDataFrame", data = gsndf)
    cds <-
      newCellDataSet(
        X_counts,
        phenoData = pd,
        featureData = fd,
        expressionFamily = negbinomial.size()
      )
    cds <- estimateSizeFactors(cds)
    cds <- estimateDispersions(cds)
    cds <- detectGenes(cds, min_expr = 0.1)
    disp_table <- dispersionTable(cds)
    ordering_genes <- subset(disp_table, mean_expression >= 0.1)
    cds <- setOrderingFilter(cds, ordering_genes)
    cds <- reduceDimension(cds)
    saveRDS(cds , rds.fname)
}
```

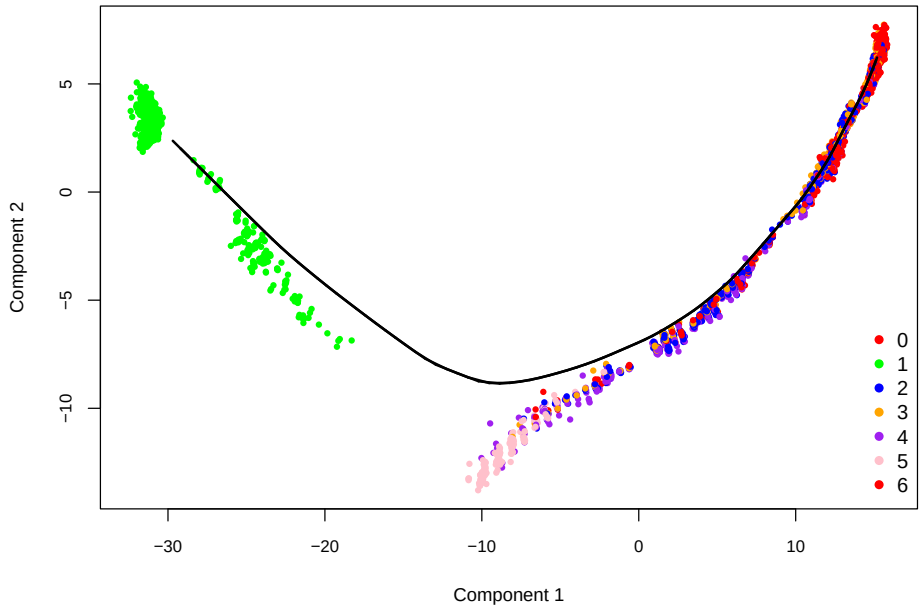
```
mono2.learn.traject(
  X_counts = as.matrix(counts_st@assays$RNA@counts),
  these.cell.types = counts_st[[]]$seurat_clusters)
```

```
mono.rds <-readRDS("Trajectory-cds.rds")
mono.Tree <- t(mono.rds@reducedDimS)
```

```
do.Traj.lines <- function(traj) {
  for (i in 1:length(traj@minSpanningTree[[]])) {
    p1no <- unlist(traj@minSpanningTree[[i]])[1]
    p2no <- unlist(traj@minSpanningTree[[i]])[2]
    lines(
      c(traj@reducedDimK[1, p1no],
        traj@reducedDimK[1, p2no]),
      c(traj@reducedDimK[2, p1no],
        traj@reducedDimK[2, p2no]),
      lwd = 2
    )
  }
}
```

```
clusters <- counts_st[[]]$seurat_clusters
cols.traj <- c("red", "green", "blue","orange","purple","pink")
traj.cols <-  cols.traj[clusters]
plot(
```

```
    mono.Tree,
    cex = 0.7,
    pch = 16,
    bg = traj.cols,
    col = traj.cols,
    xlab = "Component 1",
    ylab = "Component 2",
    cex.axis = 0.8,
    cex.lab = 0.9
)
do.Traj.lines(mono.rds)
legend(
  "bottomright",
  legend = levels(clusters),
  pch = 16,
  col = cols.traj,
  bty = "n"
)
```



Session info

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
```

```
## Running under: macOS Big Sur/Monterey 10.16
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] SingleR_1.10.0              SummarizedExperiment_1.26.1
##  [3] GenomicRanges_1.48.0        GenomeInfoDb_1.32.2
##  [5] MatrixGenerics_1.8.0        matrixStats_0.62.0
##  [7] dplyr_1.0.9                 org.Hs.eg.db_3.15.0
##  [9] AnnotationDbi_1.58.0        IRanges_2.30.0
## [11] S4Vectors_0.34.0            Biobase_2.56.0
## [13] BiocGenerics_0.42.0         sp_1.5-0
## [15] SeuratObject_4.1.0          Seurat_4.1.1
## [17] Rsubread_2.10.2
##
## loaded via a namespace (and not attached):
##   [1] plyr_1.8.7                igraph_1.3.2
##   [3] lazyeval_0.2.2            splines_4.2.0
##   [5] BiocParallel_1.30.3       listenv_0.8.0
##   [7] scattermore_0.8           ggplot2_3.3.6
##   [9] digest_0.6.29             htmltools_0.5.2
##  [11] fansi_1.0.3               magrittr_2.0.3
##  [13] memoise_2.0.1             ScaledMatrix_1.4.0
##  [15] tensor_1.5                cluster_2.1.3
##  [17] ROCR_1.0-11               limma_3.52.1
##  [19] globals_0.15.0            Biostrings_2.64.0
##  [21] spatstat.sparse_2.1-1     colorspace_2.0-3
##  [23] blob_1.2.3                ggrepel_0.9.1
##  [25] xfun_0.31                 crayon_1.5.1
##  [27] RCurl_1.98-1.7            jsonlite_1.8.0
##  [29] progressr_0.10.1          spatstat.data_2.2-0
##  [31] survival_3.3-1            zoo_1.8-10
##  [33] glue_1.6.2                polyclip_1.10-0
##  [35] gtable_0.3.0              zlibbioc_1.42.0
##  [37] XVector_0.36.0            leiden_0.4.2
##  [39] DelayedArray_0.22.0       BiocSingular_1.12.0
##  [41] future.apply_1.9.0        abind_1.4-5
```

```
##  [43] scales_1.2.0              DBI_1.1.2
##  [45] spatstat.random_2.2-0     miniUI_0.1.1.1
##  [47] Rcpp_1.0.8.3              viridisLite_0.4.0
##  [49] xtable_1.8-4              reticulate_1.25
##  [51] spatstat.core_2.4-4       rsvd_1.0.5
##  [53] bit_4.0.4                 htmlwidgets_1.5.4
##  [55] httr_1.4.3                RColorBrewer_1.1-3
##  [57] ellipsis_0.3.2            ica_1.0-2
##  [59] farver_2.1.0              pkgconfig_2.0.3
##  [61] uwot_0.1.11               deldir_1.0-6
##  [63] utf8_1.2.2                labeling_0.4.2
##  [65] tidyselect_1.1.2          rlang_1.0.2
##  [67] reshape2_1.4.4            later_1.3.0
##  [69] munsell_0.5.0             tools_4.2.0
##  [71] cachem_1.0.6              cli_3.3.0
##  [73] generics_0.1.2            RSQLite_2.2.14
##  [75] ggridges_0.5.3            evaluate_0.15
##  [77] stringr_1.4.0             fastmap_1.1.0
##  [79] yaml_2.3.5                goftest_1.2-3
##  [81] knitr_1.39                bit64_4.0.5
##  [83] fitdistrplus_1.1-8        purrr_0.3.4
##  [85] RANN_2.6.1                KEGGREST_1.36.2
##  [87] sparseMatrixStats_1.8.0   pbapply_1.5-0
##  [89] future_1.26.1             nlme_3.1-157
##  [91] mime_0.12                 compiler_4.2.0
##  [93] rstudioapi_0.13           plotly_4.10.0
##  [95] png_0.1-7                 spatstat.utils_2.3-1
##  [97] tibble_3.1.7              stringi_1.7.6
##  [99] highr_0.9                 RSpectra_0.16-1
## [101] rgeos_0.5-9               lattice_0.20-45
## [103] Matrix_1.4-1              vctrs_0.4.1
## [105] pillar_1.7.0              lifecycle_1.0.1
## [107] spatstat.geom_2.4-0       lmtest_0.9-40
## [109] BiocNeighbors_1.14.0      RcppAnnoy_0.0.19
## [111] data.table_1.14.2         cowplot_1.1.1
## [113] bitops_1.0-7              irlba_2.3.5
## [115] httpuv_1.6.5              patchwork_1.1.1
## [117] R6_2.5.1                  bookdown_0.27
## [119] promises_1.2.0.1          KernSmooth_2.23-20
## [121] gridExtra_2.3             parallelly_1.32.0
## [123] codetools_0.2-18          MASS_7.3-57
## [125] assertthat_0.2.1          withr_2.5.0
## [127] sctransform_0.3.3         GenomeInfoDbData_1.2.8
## [129] mgcv_1.8-40               parallel_4.2.0
## [131] beachmat_2.12.0           grid_4.2.0
## [133] rpart_4.1.16              tidyr_1.2.0
```

```
## [135] DelayedMatrixStats_1.18.0 rmarkdown_2.14
## [137] Rtsne_0.16                shiny_1.7.1
```

Bibliography

Chen, W., Zhao, Y., Chen, X., Yang, Z., Xu, X., Bi, Y., Chen, V., Li, J., Choi, H., Ernest, B., et al. (2020). A multicenter study benchmarking single-cell rna sequencing technologies using reference samples. *Nature Biotechnology*, pages 1–12.

Hao, Y., Hao, S., Andersen-Nissen, E., III, W. M. M., Zheng, S., Butler, A., Lee, M. J., Wilk, A. J., Darby, C., Zagar, M., Hoffman, P., Stoeckius, M., Papalexi, E., Mimitou, E. P., Jain, J., Srivastava, A., Stuart, T., Fleming, L. B., Yeung, B., Rogers, A. J., McElrath, J. M., Blish, C. A., Gottardo, R., Smibert, P., and Satija, R. (2021). Integrated analysis of multimodal single-cell data. *Cell*.

Liao, Y., Smyth, G. K., and Shi, W. (2013a). featurecounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923–930.

Liao, Y., Smyth, G. K., and Shi, W. (2013b). The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic acids research*, 41(10):e108–e108.

Liao, Y., Smyth, G. K., and Shi, W. (2019). The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. *Nucleic Acids Research*, 47:e47.

Nayak, R. and Hasija, Y. (2021). A hitchhiker's guide to single-cell transcriptomics and data analysis pipelines. *Genomics*.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., and Smyth, G. K. (2015). limma powers differential expression analyses for rna-sequencing and microarray studies. *Nucleic acids research*, 43(7):e47–e47.

Tang, F., Barbacioru, C., Wang, Y., Nordman, E., Lee, C., Xu, N., Wang, X., Bodeau, J., Tuch, B. B., Siddiqui, A., et al. (2009). mrna-seq whole-transcriptome analysis of a single cell. *Nature methods*, 6(5):377–382.