BMEG 591E Project: Reproduction of clustering Analysis of Single-Cell RNA Sequencing Data from Liver Samples

Erik Lamoureux (21282141) and Joyce Teodoro (44994085)

Contents

Introduction

In this project we assess the reproducibility of "Identification of distinct tumour cell populations and key genetic mechanisms through single cell sequencing in hepatoblastoma" published in Communications Biology (2021)4:1049, https://doi.org/10.1038/s42003-021-02562-8. Its Genome Expression Omnibus (GEO) entry can be found at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE180666. Hepatoblastoma (HB) is the most common liver malignacy in childhood. Molecular investigations of HB are limited and there is a need for effective treatment options for chemoresistant disease. Current standards of care are multimodal, including neoadjuvant chemotherapy in addition to surgical resection or liver transplantation. For understanding the pathogenesis of HB, researchers have assessed its molecular characteristics using whole genome, exome, and RNA sequencing. Although some signalling pathways have been indicated for the development of HB, specific mechanisms remain elusive. To assess this, Bondoc et al. (2021) used single cell ribonucleic acid sequencing (scRNAseq) to analyze human tumour (HB17, HB30 and HB53), background liver (HB17 and HB53), and patient derived xenograft (PDX) samples (HB17 and HB30) to determine intratumour cell subtype heterogeneity. They sought to define different roles in pathogenesis based on intracelluar signalling in paediatric HB. They found a driver cell cluster in HB by genetic expression that can be used to help define disease mechanism and treatment. By identifying both mechanistic pathways and unique cell populations, they provide indicators for the direction of future research for investigating novel HB treatment strategies.

Scope

Initially, we intended to recreate the entirety of the results from this paper. However, this was too ambitious, so we focused on Figure 3 (see below). Figure 3 provides single-cell RNA sequencing characterization of background liver, tumour, and PDX samples. Figure 3A presents a UMAP clustering of cell classes and sample groups of the combined data from background, tumour and PDX. Labels for endothelial cell, hepatic stellate cell, erythrocyte, 774 T cell, NK cell, cholangiocyte, and hproliferative hepatocyte are included. Figure 3B depicts the UMAP clustering of marker genes for for common cell populations in the liver and tumours. Included marker genes are GPC3 (HB tumour cells), CYP3A4 (hepatocyte cells), COL6A3 (hepatic stellate cells), CD163 (Kupffer cells), FLT1 (epithelial cells), PTPRC (immune cells). The original paper includes Figure 3C: a heatmap for 200 most upregulated genes for cell types in background liver, tumour, and PDX samples. We were not able to recreate this heatmap because the authors provided scant detail how they assigned the labelled cell type. Given the additional challenges of single-cell RNA sequencing compared to DNA sequencing, we determined that recreating the UMAP clusters of cell types, sample groups, and cell population marker genes would be sufficiently challenging.

Quality Control

We assessed the quality of our data and analyses in a number of ways. First, after applying fastqc to the fastq RNA sequencing files, we assessed the resulting fastqc .html files. These files provide a FastQC report describing the quality of the sequencing data in terms of basic statistics, per base sequence quality, per tile sequence quality, per sequence quality scores, per base sequence content, per sequence GC content, per base N content, sequence length distribution, sequence duplication levels, overrepresented sequences, and adapter content. After aligning the RNA reads using Cell Ranger, further quality control measures were taken. The cells of each sample were filtered so only viable and high quality cells remain. In the paper, the authors specify thresholds for the number of expressed genes, UMIs and mitochondrial reads, so only high quality cells are used for downstream analyses. Further, we obtained sample meta data to assess the number of UMIs, number of unique genes detected per cell, and percentage of reads mapping to mitochondrial genes. This provided additional information about detemining thresholds for filtering poor quality cells. Then, we assessed the principal component overlap after data integration of the tumour, PDX, and background samples. We expected tumour and PDX samples to overlap, whicle the background samples should overlap with other background samples. Additionally, quality assessment of the clustering assessment was undertaken in a similar manner: identify tumour and PDX overlap, and background sample overlap with other background samples.

Quality control measures are covered in more detail below when discussed in the experimental chronology.

Methods and Analysis

Our methods and analysis are thoroughly covered below chronologically, but here we will provide an outline. We start by downloading the fastq files from the European Nucleotide Archive, assess their quality using fastqc, and perform read alignment using Cell Ranger. After the gene-barcode counts matrix has been generated, we pre-process the data and perform additional quality control measures. We do this by creating a Seurat object for each sample so their meta data can be evaluated. The meta data evaluates sample quality, such as number of UMIs, number of unique genes detected per cell, and percentage of reads mapping

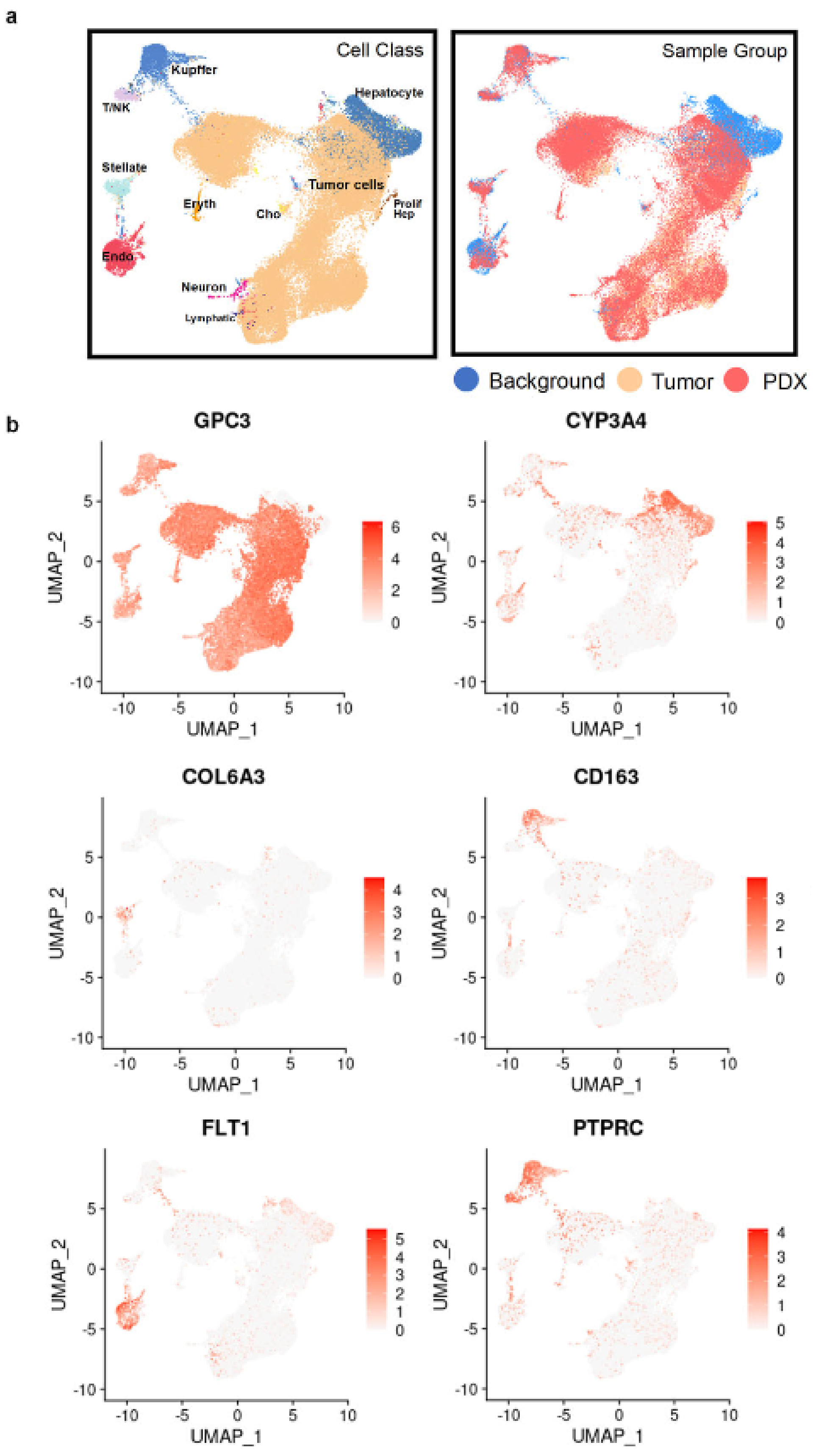


Figure 1: Figure 3 (Bondoc et al., 2021)

to mitochondrial genes. These data are presented as violin plots. This was used to inform us about the thresholds for filtering poor quality cells. Next, the data were normalized, integrated, and dimensionality reduced to reduce batch effects and allow for comparable downstream assessments. Then, the principal components of the data were plotted to assess the separation between the background liver samples, the tumour samples, and the PDX samples. Data integration was performed to correct against batch effects. Generally, we find that the tumour and PDX samples overlap, while the background samples from different donors overlap with each other. After integration, we determined that using 30 principal components for clustering is sufficient to capture most of the variation in the model, and for easy comparison to the original paper. We performed nearest neighbour clustering visualized using Uniform Manifold Approximation and Projection (UMAP). We visualized the clusters to confirm tumour and PDX samples were similar and that different background samples were similar. Finally, we qualitatively explored the clusters for different marker genes that are associated with particular cell types.

More details and discussion surrounding the methods and analysis can be found in the parts below.

Summary

Overall, we successfully validated the cluster analysis of cell types, sample groups, and cell population marker genes from the background liver, tumour, and PDX sample for hepatoblastoma single cell RNA sequencing. Although some of our methods differed from the paper, including the use of an updated reference genome with ~3,000 additional genes, we do not see any marked changes resulting from the difference. Further, during the use of Cell Ranger we included intronic reads to increase the sensitivity of the assay. 10X genomics recommends including intronic reads, and we think its inclusion is the cause of some variation between our results an the paper's, albeit these effects are minor. We demonstrate that some of the early results from this paper are reproducible, providing support that their results are valid, and that with enough time and resources the rest of the paper's results could be reproduced. Additional concluding remarks can be found at the end of the document.

Part 0: Loading Libraries

```r
## Install required libraries

library(Seurat)
library(patchwork)
library(ggplot2)
library(dplyr)
library(harmony)
library(Matrix)
library(reshape2)
```

Part 1: Raw Data Download and Gene-Barcode Counts Matrix Generation

The Genome Expression Omnibus (GEO) code for this project, GSE180666, contains two types of reads, bulk (GSE180664) and single-cell (GSE180665). We downloaded all the bulk and single-cell read files for this project from the European Nucleotide Archive (ENA). The corresponding ENA project name from this GEO entry is PRJNA749045, comprised of PRJNA749048 (bulk) and PRJNA749050 (single-cell RNA).

The bulk RNA data consists of 22 samples (numbered 029-050), of which 3 are sequenced twice. Using a Compute Canada high performance computing cluser, we downloaded these samples using a bash script.

The script uses `fasterq-dump` to download the bulk and single-cell RNA data SRR15217914. In total, there are 42.06 Gb of Bulk RNA data, with an average of 1.68 Gb per sample. The single-cell RNA data consists of 14 samples (numbered 001-014) for a total of 117.3 Gb, averaging 8.4 Gb a sample.

Script downloadData.sh:

```
#!/bin/bash
#SBATCH --time=02:00:00
#SBATCH --account=def-hongma
#SBATCH --mem=4G
#SBATCH --cpus-per-task=6
#SBATCH --job-name=SRR15217914
#SBATCH --output=SRR15217914.slurm.log
# Set up the slurm requests and load required modules

module load StdEnv/2020

fasterq-dump SRR15217914 --split-files --include-technical
                      --outdir /scratch/user/singleCell_RNA_RawData
```

Then, `fastqc` function was applied on the fastq.gz files to assess the read quality.

Script fastqc_raw.sh:

```
#!/bin/bash
#SBATCH --time=20:00:00
#SBATCH --account=def-hongma
#SBATCH --mem-per-cpu=500M
# Set up the slurm requests and load FastQC

module load StdEnv/2020
module load fastqc/0.11.9

set -e # Script exits when an error occurs

mkdir -p /scratch/user/MyLogDirectory
logDir=/scratch/user/MyLogDirectory

# Change the working directory to the location of the raw RNA reads
cd /scratch/user/singleCell_RNA_Raw
# For every file in the directory, run fastqc
for file in $(pwd | ls)
do
    #run this code only if $logDir/$file.fastqc.done is missing
    if [ ! -e $logDir/$file.fastqc.done ]
    then
            echo Performing fastqc of sample $file

            # Commands to run fastqc
              fastqc $file --outdir=/scratch/user/fastqc_Raw

            # create the file that we were looking for at the beginning of this
            # if statement so that this same code is not run next time
              touch $logDir/$file.fastqc.done
    else # $logDir/$file.fastqc.done was not missing
            echo Already performed fastqc of $file
    fi
done
```

Applying fastqc to the .fastq files produces a .zip and .html output assessing the read quality. The fastqc .html output was extracted from the server and downloaded to our local machines using the `pscp` command in our local Windows PowerShell.

```
# Example pscp command in local Windows PowerShell
pscp user@cedar.computecanada.ca:projects/def-hongma/user/bmeg_591e/
                                                scRNAseq-project/bulk_data/
SRR15217829_1_fastqc.html SRR15217829_1_fastqc.html
```

After the .html fastqc output files were downloaded to our local machines, we inspected them and accumulated summary statistics. Overall, for both bulk and single-cell RNA reads, the basic statistics, per base sequencing quality, per tile sequencing quality, per sequence quality scores, per sequence GC content, per base N content, sequence length distribution, and adapter content were nearly all of good quality, with a few exceptions. However, per base sequence content and sequence duplication levels were of poor quality for both bulk and single-cell reads. The only main difference between the bulk and single-cell data is that the bulk data had good quality with respect to overrepresented sequences, while the single-cell data had mid or inconclusive quality readings. We generally find that the RNA sequencing reads are of good quality, but have poorer quality aspects compared to DNA sequences assessed during the assignments. Reduced quality of the RNA sequencing data compared to DNA sencing is common and can be attributed to a number of factors. First, adapter or primer sequences may not have been trimmed, resulting in high overrepresented sequences. Additionally, fluctuations seen in the per base sequence content below read position 30 is due to the restriction enzyme, transposase, having a preference for cutting the read at lower position numbers. We think these data quality issues are minor, and should not overtly affect our analyses.

Below are example figures generally representative of the single-cell data quality.

After the RNA sequences were determined to be of sufficient quality, we used a bash script to determine the number of gene counts from the fastq files. We used Cell Ranger to determine the gene counts, https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/what-is-cell-ranger. Cell Ranger is a set of analysis pipelines that process single-cell RNA data to align reads, generate feature-barcode matrices, and perform clustering. Cell Ranger command `count` takes the fastq files and performs alignment, filtering, barcode counting, and UMI coutning. Details of the implementation of `count` is found below. Read alignment was performed using the GRCh38 referene genome.

The sequencing samples used in this project came from a 10X Genomics https://www.10xgenomics.com/ platform: Chromium Single Cell Gene Expression with Feature Barcoding Analysis https://gccri.uthscsa.edu/wp-content/uploads/sites/128/2020/04/Chromium-Single-Cell-software-training.pdf. This technology provides unbiased single cell transcriptome 3' gene expression and can simultaneously assess perturbation phenotypes, protein abundance, and gene expression from the same cell. It is ready-to-use, has a robust workflow, and is compatible with whole cells and nuclei. Recent improvements include increased sensitivity for detecting more unique transcripts per cell, reducing sequencing requirements. Further, this approach is compatible with the easy-to-use Cell Ranger Analysis pipeline for gene expression analysis. Procedurally, a single cell 3' 3x barcoded gel bead is combined with reagents and single cells in suspension in microfludic crossflows to produce oil encapsulated emusions of barcoded single cells. This procedure produces a digital gene expression and cell surface protein expression or CRISPR perturbation profiles from each partitioned cell. This sequencing is done using Illumina sequencers https://www.illumina.com/.

Here, we use Cell Ranger to conduct the read count and gene expression analysis. However, Cell Ranger is only compatible with 10X Genomics sequencing protocols. Therefore, if the sample data did not come from 10X Genomics, or if we wanted to try a different approach, there are a few ways we could do

Per base sequence quality

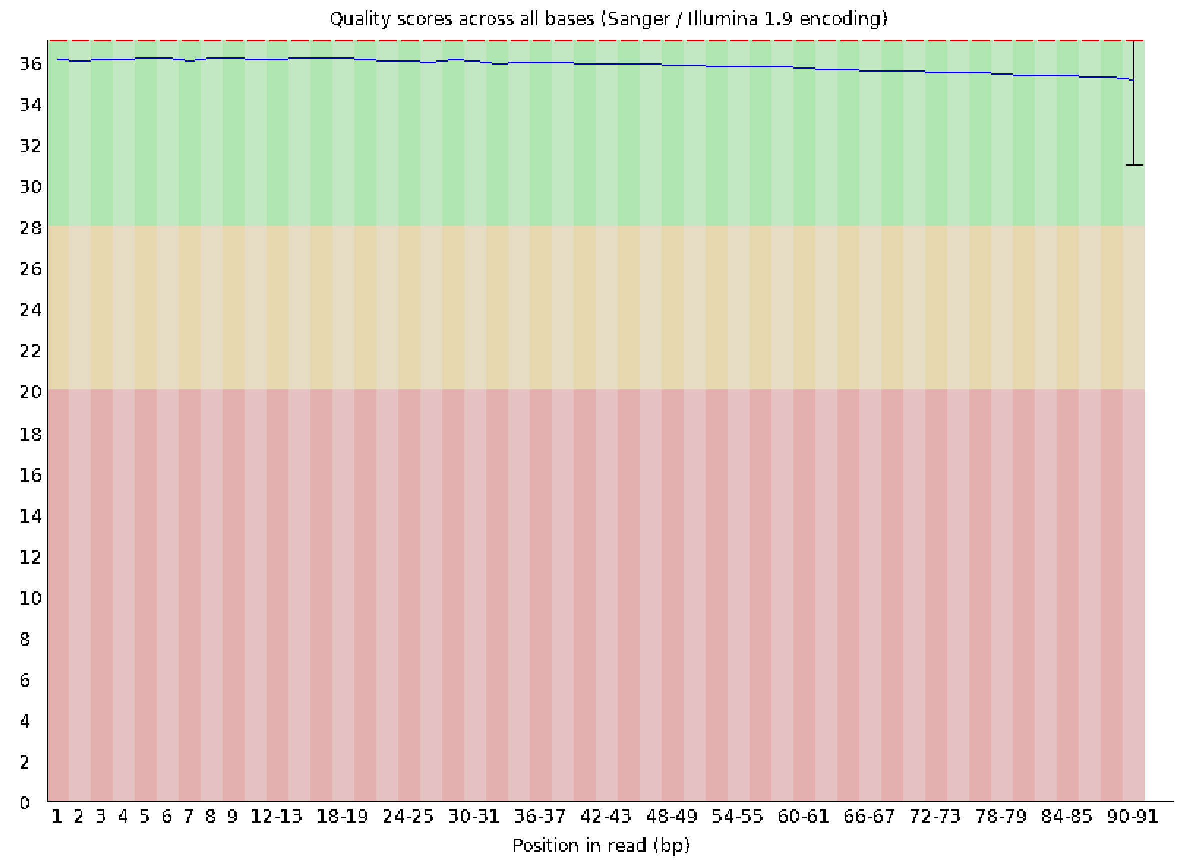


Figure 2: Per Base Sequence Quality

✓ Per sequence GC content

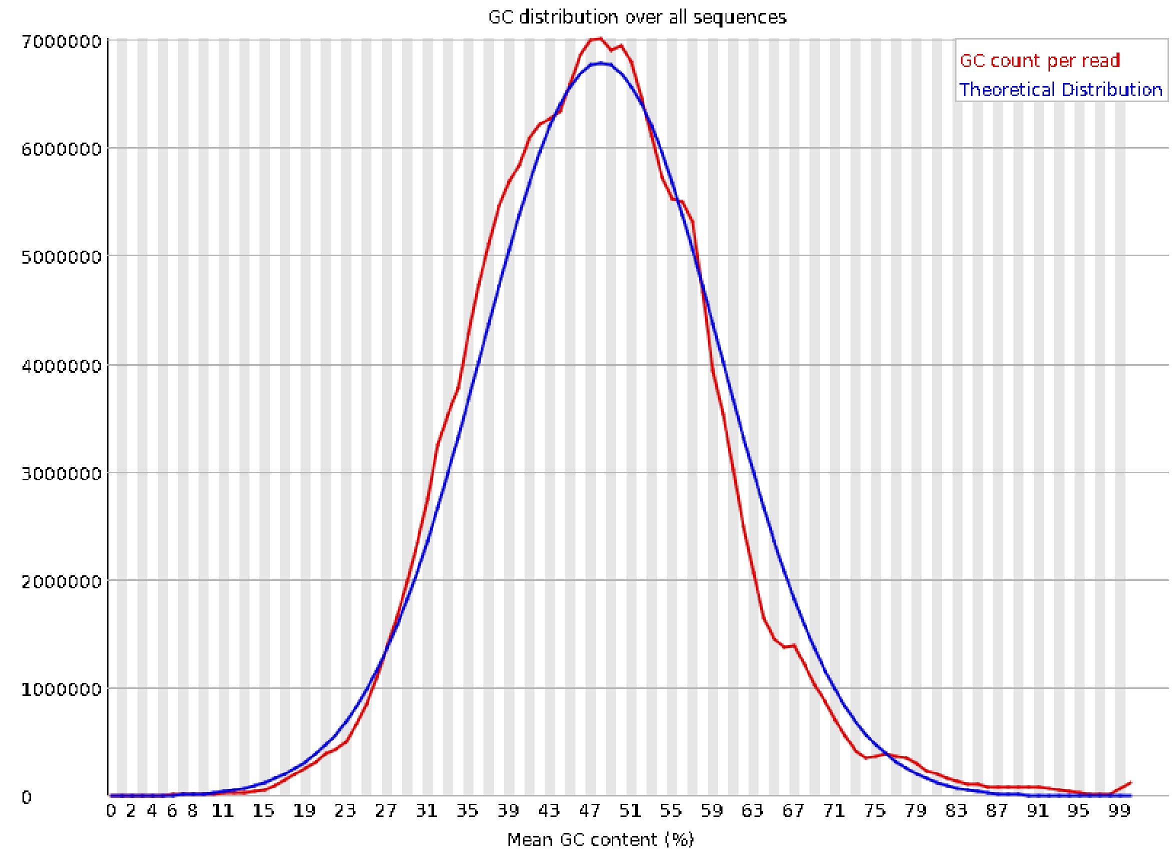


Figure 3: Per Sequence GC Count

Per sequence quality scores

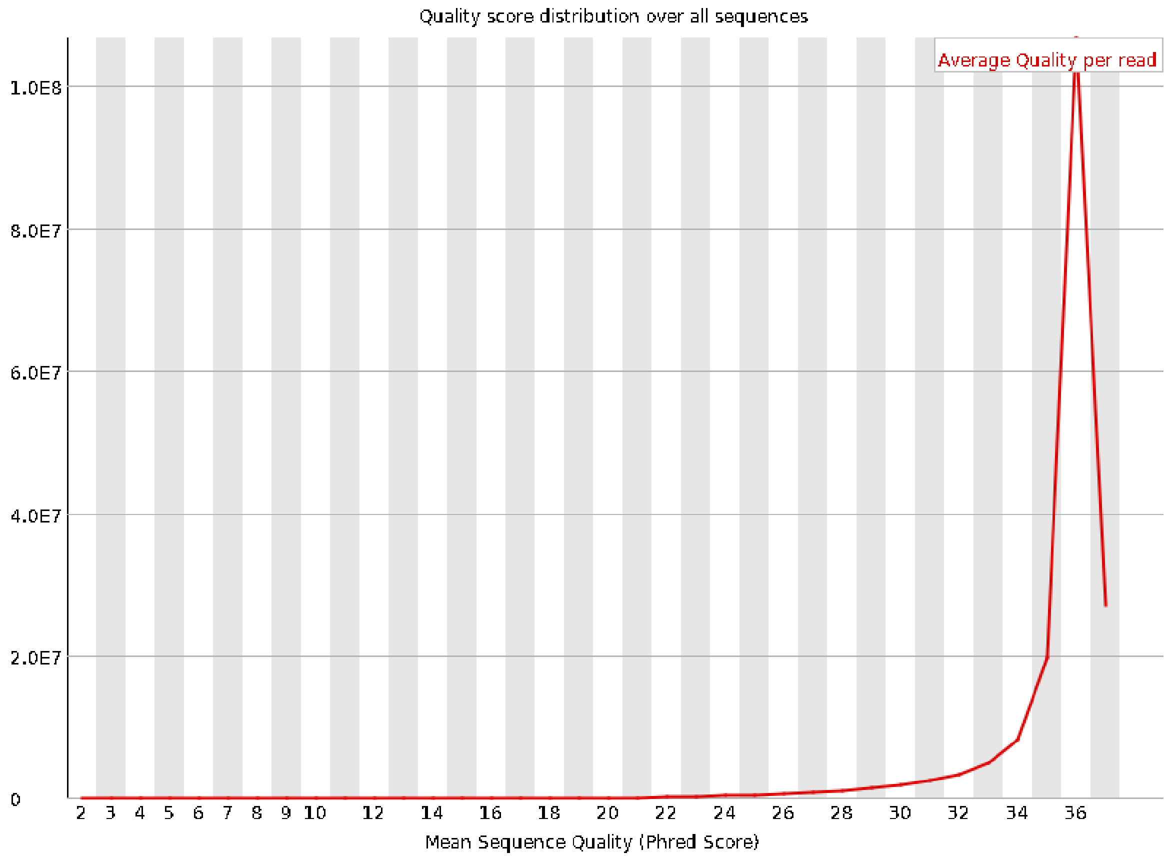


Figure 4: Per Sequence Quality

Per base sequence content

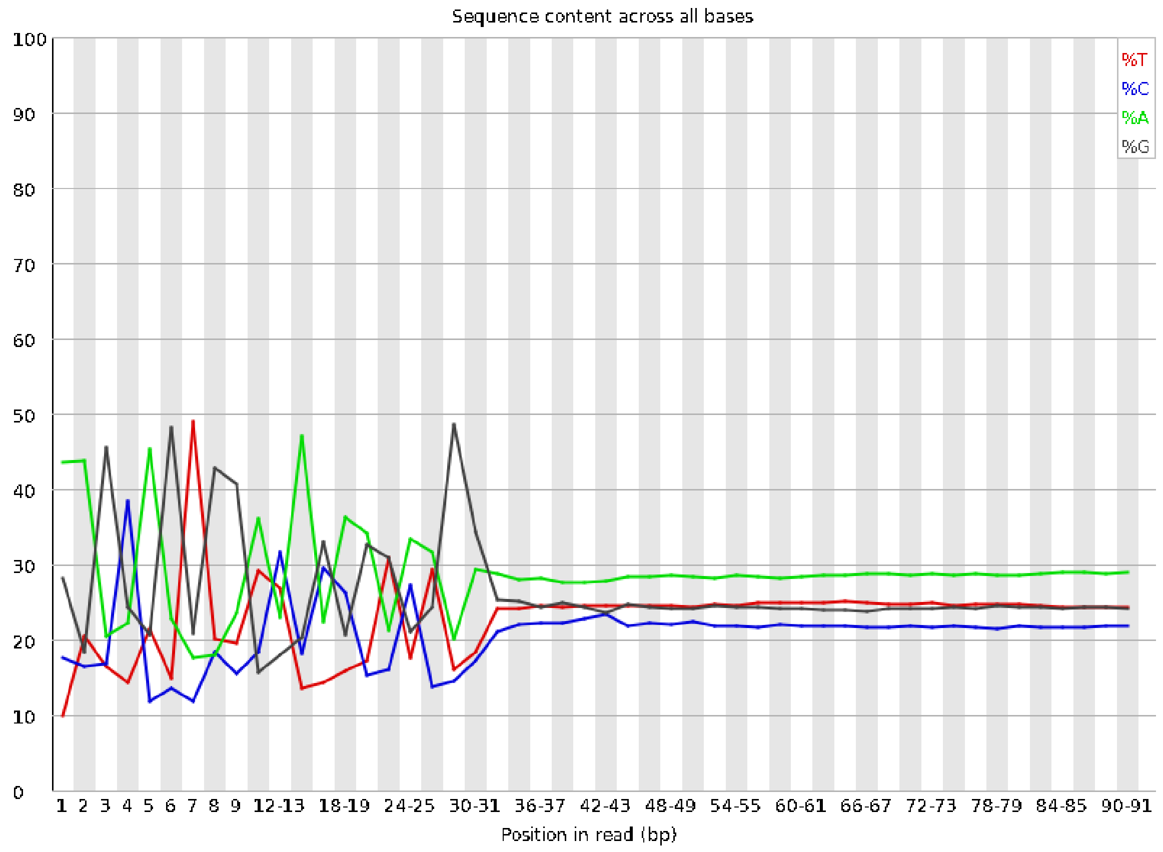
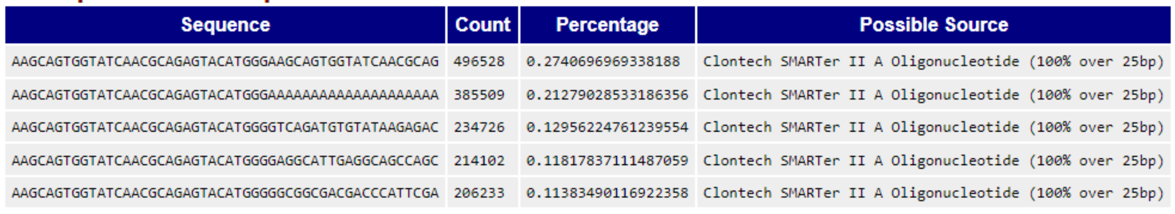


Figure 5: Per Base Sequence Content

Overrepresented sequences



Sequence	Count	Percentage	Possible Source
AAGCAGTGGTATCAACGCAGAGTACATGGGAAGCAGTGGTATCAACGCAG	496528	0.2740696969338188	Clontech SMARTer II A Oligonucleotide (100% over 25bp)
AAGCAGTGGTATCAACGCAGAGTACATGGGAAAAAAAAAAAAAAAAAAAA	385509	0.21279028533186356	Clontech SMARTer II A Oligonucleotide (100% over 25bp)
AAGCAGTGGTATCAACGCAGAGTACATGGGGTCAGATGTGTATAAGAGAC	234726	0.12956224761239554	Clontech SMARTer II A Oligonucleotide (100% over 25bp)
AAGCAGTGGTATCAACGCAGAGTACATGGGGAGGCATTGAGGCAGCCAGC	214102	0.11817837111487059	Clontech SMARTer II A Oligonucleotide (100% over 25bp)
AAGCAGTGGTATCAACGCAGAGTACATGGGGGCGGCGACGACCCATTCGA	206233	0.11383490116922358	Clontech SMARTer II A Oligonucleotide (100% over 25bp)

Figure 6: Overrepresented Sequences

this. A common method for aligning scRNAseq data is using the STAR aligner (https://github.com/alexdobin/STAR, https://hbctraining.github.io/Intro-to-rnaseq-hpc-O2/lessons/03_alignment.html). The Spliced Transcripts Alignment to a Reference (STAR) aligner is designed to address many of the challenges of RNA seq alignment using a strategy to account for spliced alignments. The STAR aligner benefits from high accuracy and outperforms other aligners by over 50x in mapping speed, but is computationally expensive. This alignment is highly efficient due to (1) seed searching and (2) clustering, stitching, and scoring. Seed searching invloves identifying the longest sequence that matches one or more locations on the reference genome. These longest matching sequences are called the Maximal Mappable Prefixes (MMPs). The different parts of the reads that are mapped separately are called seeds. After identifying a seed, STAR will then search again but only for the unmapped portions to find the next longest sequence, and so on. This sequential searching of only the unmapped portions of the reads demonstrates the efficiency of the STAR algorithm. After seed searching, clustering, stitching, and scoring occurs. The separate seeds are stitched together to create a complete read by first clustering the seeds together based on proximity. And finally, the seeds are stictched togeher based on the best scored alignment for the read. A recent improvement to STAR is STARsolo (https://doi.org/10.7490/f1000research.1117634.1). STARsolo is more computationally efficient than Cell Ranger because cell barcode demultiplexing, UMI collapsing, mapping, and quantification are integrated into a sinlge code and are executed simultaneously. A major advantage STAR solo has over Cell Ranger is that it is compatible with many established and emerging scRNAseq protocols, including Drop-seq, inDrop, and Microwell-seq.

Script fastqToGeneCounts.sh

```
#!/bin/bash
#SBATCH --time=20:00:00
#SBATCH --account=def-hongma
#SBATCH --mem=128G
#SBATCH --cpus-per-task=20
#SBATCH --job-name=HB17-Background_geneCounts
#SBATCH --output=HB17-Background_geneCounts.slurm.log
# Set up the slurm requests and load required modules

module load StdEnv/2020

cellranger count --id=HB17-Background \
                 --transcriptome=/home/user/refdata-gex-GRCh38-2020-A \
                 --fastqs=/scratch/user/singleCell_RNA_RawData/HB17-Background \
                 --sample=HB17-Background \
                 --include-introns \
                 --expect-cells=3000 \
                 --localcores=20 \
                 --localmem=115
```

Part 2: Pre-processing and Quality Control (QC) of scRNA-seq data

The CellRanger **6.1.2** pipeline for generating a gene-barcode counts matrix yields raw and filtered counts matrices. In the subsequent steps, we decided to propagate the `filtered` counts matrices because these have already been subjected to a first pass removal of barcodes that do not correspond to cells. The `raw` counts matrices include all of the valid cell barcodes found in the sample, as compared to a cell barcodes whitelist maintained by 10X Genomics, that contain at least one read. Since not all of the droplets contain cells, there is a need to remove the barcodes that may be associated with background noise and not include these in subsequent processing. According to CellRanger documentation https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/algorithms/overview, they implemented the EmptyDrops method (Lun et al., 2019) for determining which barcodes are associated with cells and which are likely associated with noise. Although not performed in this work, this algorithm can be applied to the `raw` counts matrix to filter out barcodes that are not associated with cells. This tool is part of the `DropletUtils` Bioconductor package.

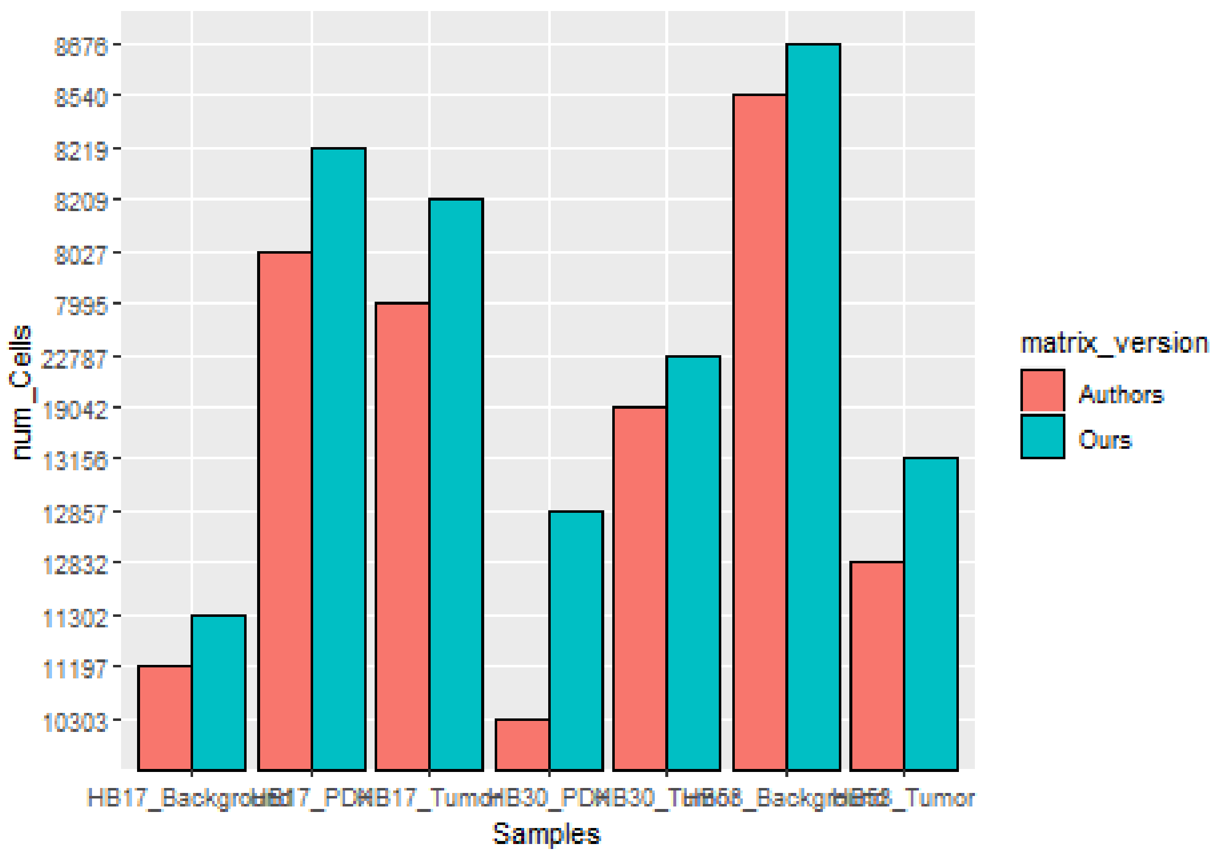
Prior to downstream analysis of the single cell RNA sequencing data, quality control of such data must be performed. Specifically, there is a need to filter the cells in each sample, such that only the viable and high quality cells are retained. The authors performed quality control of the data by specifying certain thresholds for the number of expressed genes, UMIs and mitochondrial reads, such that high quality cells can be isolated and propagated for downstream analysis. In the paper, the authors have specified the following thresholds for identifying cells of low quality or non-viable cells: number of expressed genes < 500 AND number of UMIs < 800 AND mitochondrial counts $> 10\%$. We implemented the same thresholds as the authors for our filtering for ease of comparison. Additionally, it would not serve us well to further restrict the filtering outcomes because it is better under-filter as to not remove informative reads.

```r
## Loading the gene-barcode matrix and creating a Seurat object for
## each sample.

## Store sample data information (ie. Sample name, number of genes
## detected, number of cells detected) in a data frame. Data frame is
## initialized to be empty.
Samples_info.df <- data.frame(Samples = character(), num_Genes = integer(),
    num_Cells = integer())

## The filtered gene-barcode matrices are loaded as follows:
i = 1  # Count variable
for (file in c("HB17_Background", "HB17_PDX", "HB17_Tumor", "HB30_PDX",
    "HB30_Tumor", "HB53_Background", "HB53_Tumor")) {
    counts_matrix_h5.data <- Read10X_h5(filename = paste0(getwd(), "/data_OurRun/",
        file, "/filtered_feature_bc_matrix.h5"))

    # Add the sample name, number of genes detected and number of
    # cells detected to the data frame.
    Samples_info.df[i, ] <- c(file, counts_matrix_h5.data@Dim[1], counts_matrix_h5.data@Dim[2])

    # Create a Seurat object using the data loaded above. The
    # `min.cells` parameter includes genes (ie. features) that have
    # been detected in at least 'x' cells. The `min.features`
    # parameter includes cells that have at least 'y' detected genes.
    seurat_object <- CreateSeuratObject(counts = counts_matrix_h5.data,
        project = file, min.cells = 0, min.features = 0)

    # Calculate the percentage of UMI counts or transcripts mapping
    # to mitochondrial genes in each cell. These gene names typically
    # start with 'MT-'. Append this column to the Seurat object meta
    # data.
    seurat_object$percent_mitochondria <- PercentageFeatureSet(seurat_object,
        pattern = "^MT-")

    # Assign the value of `seurat_object` to a different variable
    assign(paste0(file, "_seurat"), seurat_object)

    i = i + 1
}


# Remove the temporary storage objects (ie. `counts_matrix_h5.data`,
# `seurat_object`, loop count variable `i` and `file') from the
# workspace
rm(counts_matrix_h5.data, seurat_object, i, file)

gc()
```

```
##             used   (Mb) gc trigger (Mb)  max used   (Mb)
## Ncells   2857990  152.7    5261398  281   4959694  264.9
## Vcells 337504256 2575.0  513012132 3914 510485003 3894.7
```

Since we processed the raw fastq files and fed them into the CellRanger pipeline, we would like to compare the number of cells present in the resulting output counts matrix. This is shown below. It can be seen that the number of detected cells across all samples from our version of CellRanger processing is greater than the authors'. This could be due to the inclusion of intronic reads when the CellRanger pipeline was implemented for generating gene-cell count matrices. Including intronic reads can lead to increased number of valid UMI counts, which can lead to more barcodes being associated to cells.

```
## Comparing CellRanger output matrices for each sample between two
## different CellRanger runs: Author's run: CellRanger 3.1.0 Our run:
## CellRanger 6.1.2

# Read in authors' filtered gene count matrix information
Samples_info_orig <- read.csv(file = paste0(getwd(), "/data_Authors/",
    "Samples_info_orig.csv"))

Samples_info.df$matrix_version <- ("Ours")

Samples_info_orig$X <- NULL
Samples_info_orig$matrix_version <- ("Authors")

Samples_info_orig[1, 1] <- "HB17_Background"
Samples_info_orig[3, 1] <- "HB17_Tumor"
Samples_info_orig[4, 1] <- "HB30_PDX"
Samples_info_orig[5, 1] <- "HB30_Tumor"
Samples_info_orig[6, 1] <- "HB53_Background"
Samples_info_orig[7, 1] <- "HB53_Tumor"

Samples_info_combined.df <- Samples_info.df

Samples_info_combined.df <- rbind.data.frame(Samples_info_combined.df,
    Samples_info_orig)

Samples_info_combined_melted <- melt(Samples_info_combined.df, measure.vars = c("matrix_version"))

ggplot(data = Samples_info_combined.df, mapping = aes(x = Samples, y = num_Cells,
    fill = matrix_version)) + geom_col(colour = "black", position = "dodge")
```



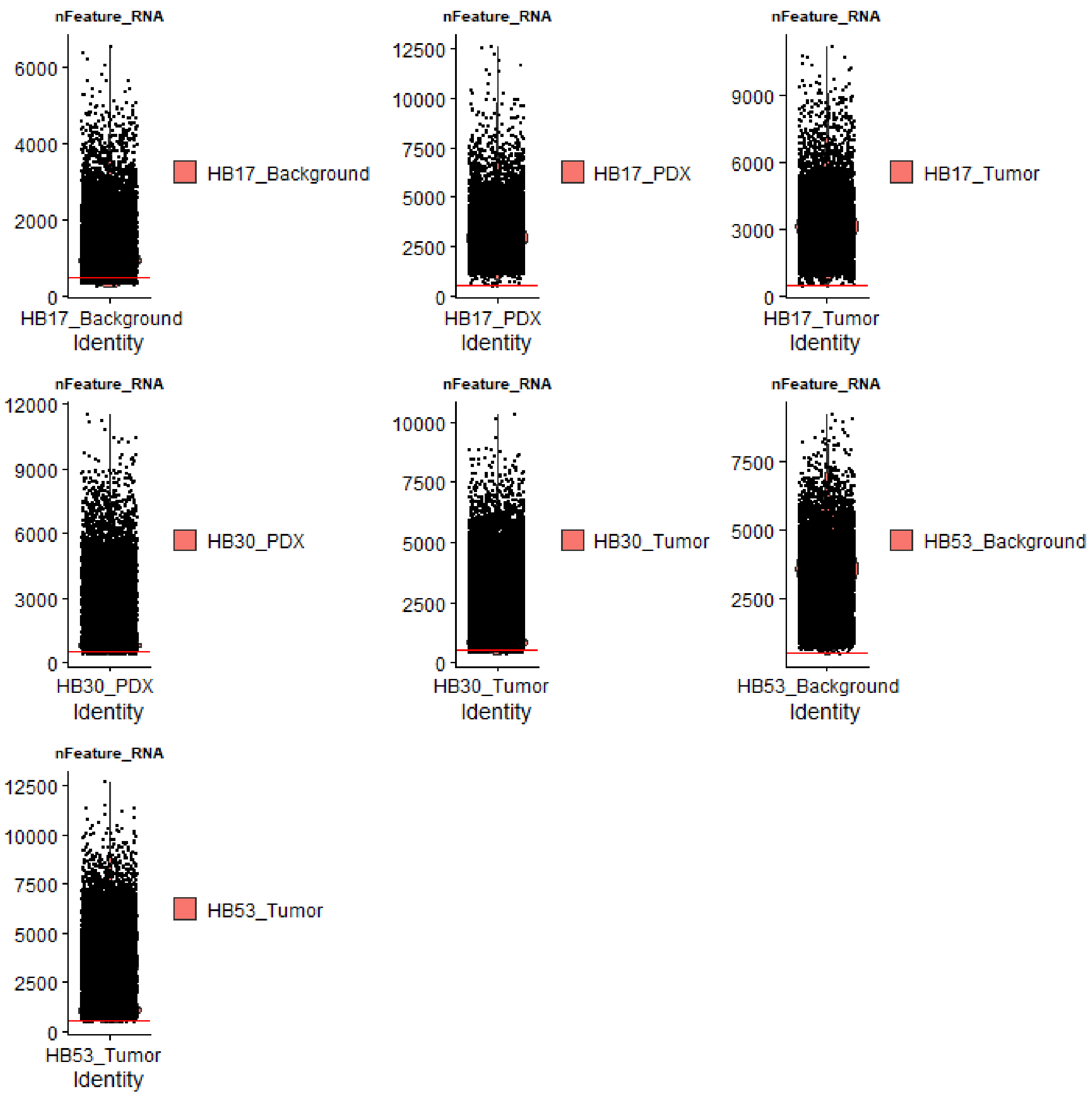
We decided to use the Seurat (4.1.0) package for data preprocessing and clustering of single-cell data mainly because this is the same tool that was used by the authors in the paper. The authors used Seurat 3.9.9.9010. Once the Seurat object for each sample is created, their respective meta data can be accessed. The meta data contains the information about the total number of RNA molecules (UMIs) detected per cell, total number of unique genes detected per cell and percentage of reads mapping to mitochondrial genes. These are indicated in the last three columns of a sample's meta data, `nCount_RNA`, `nFeature_RNA` and `percent_mitochondria`. The first column, which has a 16-nucleotide sequence, contains the barcodes associated with each cell. The following shows the first six rows of the HB17-Background sample's meta data.

```
# Display the first few rows of the HB17_Background_seurat object's
# meta data
head(HB17_Background_seurat@meta.data)
```

```
##                          orig.ident nCount_RNA nFeature_RNA percent_mitochondria
## AAACCCAAGCACGGAT-1 HB17_Background       2470         1080            10.323887
## AAACCCACAACTGAAA-1 HB17_Background      11601         2508            17.645031
## AAACCCACATTCACCC-1 HB17_Background       1066          716             7.223265
## AAACCCAGTACGGTTT-1 HB17_Background       8152         2567             7.249755
## AAACCCAGTATCGAGG-1 HB17_Background       3239         1051            14.109293
## AAACCCAGTATGAGCG-1 HB17_Background       2422         1305             8.835673
```

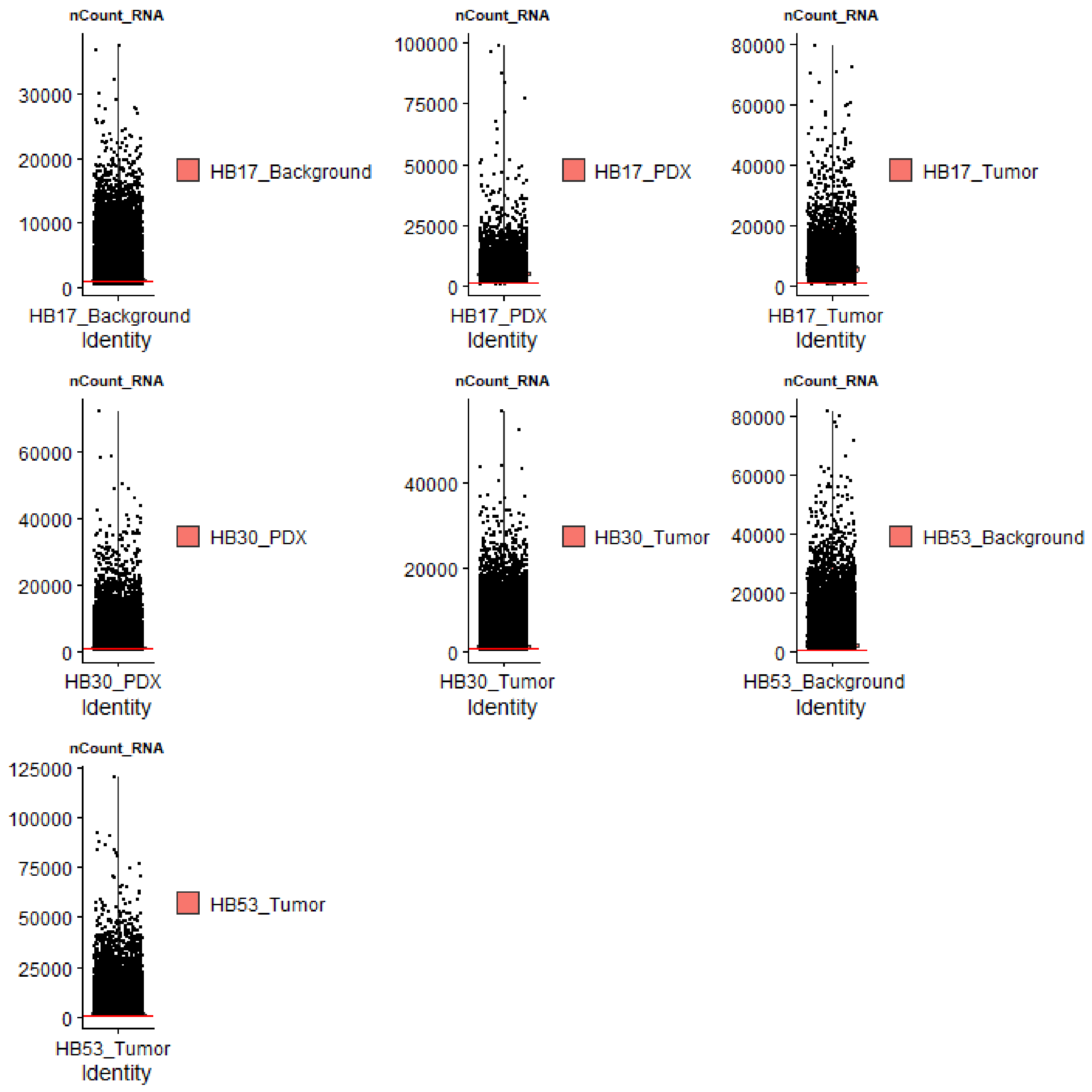
In order to get an idea of the distribution of the quality control indicators (ie. total number of RNA molecules (UMIs) detected per cell, total number of unique genes detected per cell, percentage of reads mapping to mitochondrial genes) across the different samples, it would be helpful to generate plots that can be analyzed for potential outliers. These plots would assist in setting the thresholds that would determine which cells to eliminate from the sample, as well as comparing such thresholds against those selected by the authors.

```r
# Create a vector of Seurat objects, which represent each sample
samples <- c(HB17_Background_seurat, HB17_PDX_seurat, HB17_Tumor_seurat,
    HB30_PDX_seurat, HB30_Tumor_seurat, HB53_Background_seurat, HB53_Tumor_seurat)

sample_names <- c("HB17_Background", "HB17_PDX", "HB17_Tumor", "HB30_PDX",
    "HB30_Tumor", "HB53_Background", "HB53_Tumor")


## Create a violin plot of number of unique genes detected in each
## cell (nFeature_RNA) for each sample.

for (i in 1:length(samples)) {
    Violin_plot <- VlnPlot(samples[[i]], features = c("nFeature_RNA"),
        pt.size = 0.1) & geom_hline(yintercept = 500, color = "red") &
        theme(plot.title = element_text(size = 10), axis.text.x = element_text(angle = 0,
            hjust = 0.55))

    assign(paste0("Vplot_", i), Violin_plot)

    rm(Violin_plot)
}

# Combine the violin plots for each sample on the same graph
wrap_plots(Vplot_1, Vplot_2, Vplot_3, Vplot_4, Vplot_5, Vplot_6, Vplot_7)
```

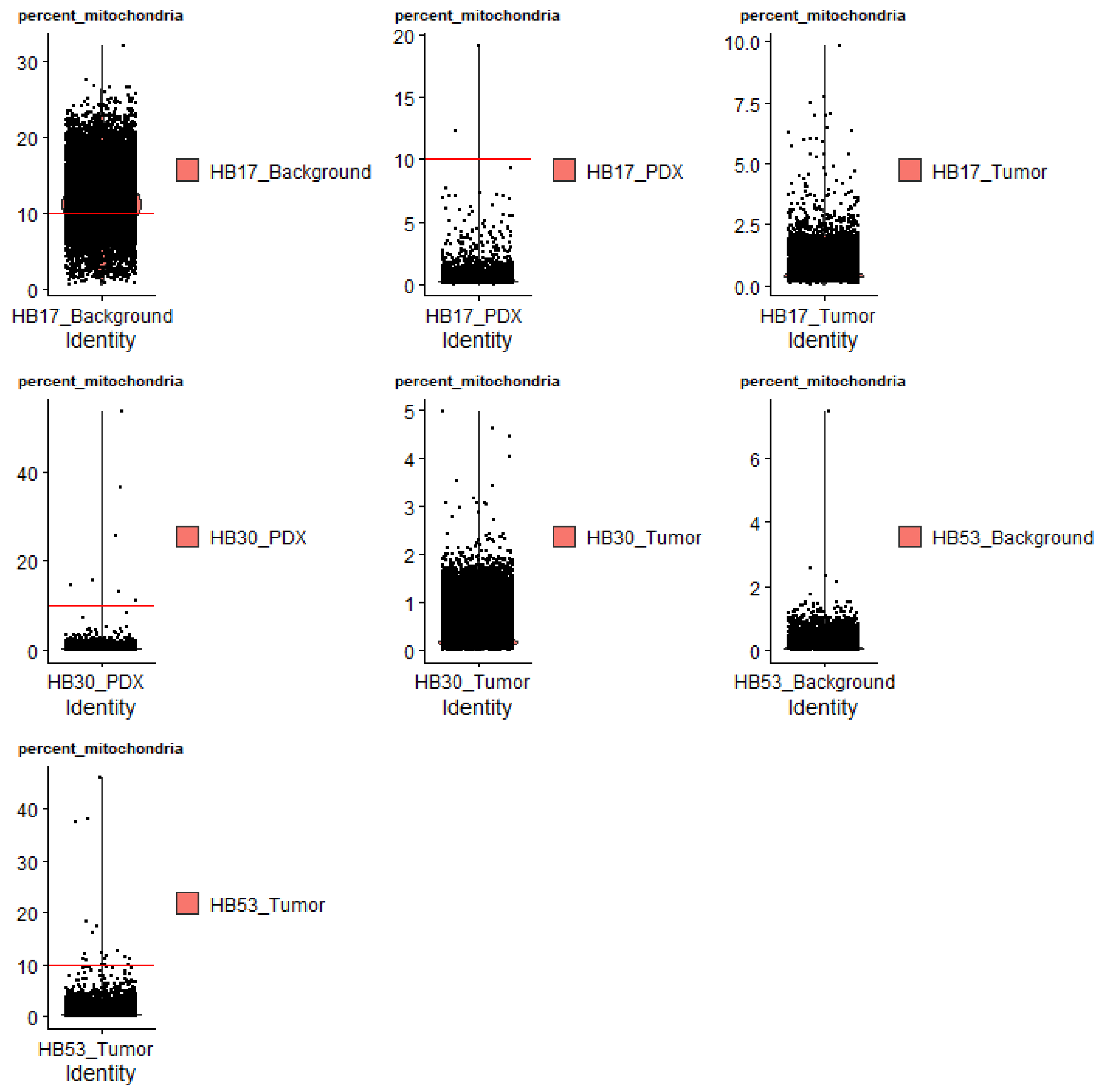


```
rm(Vplot_1, Vplot_2, Vplot_3, Vplot_4, Vplot_5, Vplot_6, Vplot_7)


## Create a violin plot of total number of RNA molecules or UMIs
## detected in each cell (nCount_RNA) for each sample.

for (i in 1:length(samples)) {
    Violin_plot <- VlnPlot(samples[[i]], features = c("nCount_RNA"), pt.size = 0.1) &
        geom_hline(yintercept = 800, color = "red") & theme(plot.title = element_text(size = 10),
        axis.text.x = element_text(angle = 0, hjust = 0.55))

    assign(paste0("Vplot_", i), Violin_plot)

    rm(Violin_plot)
```

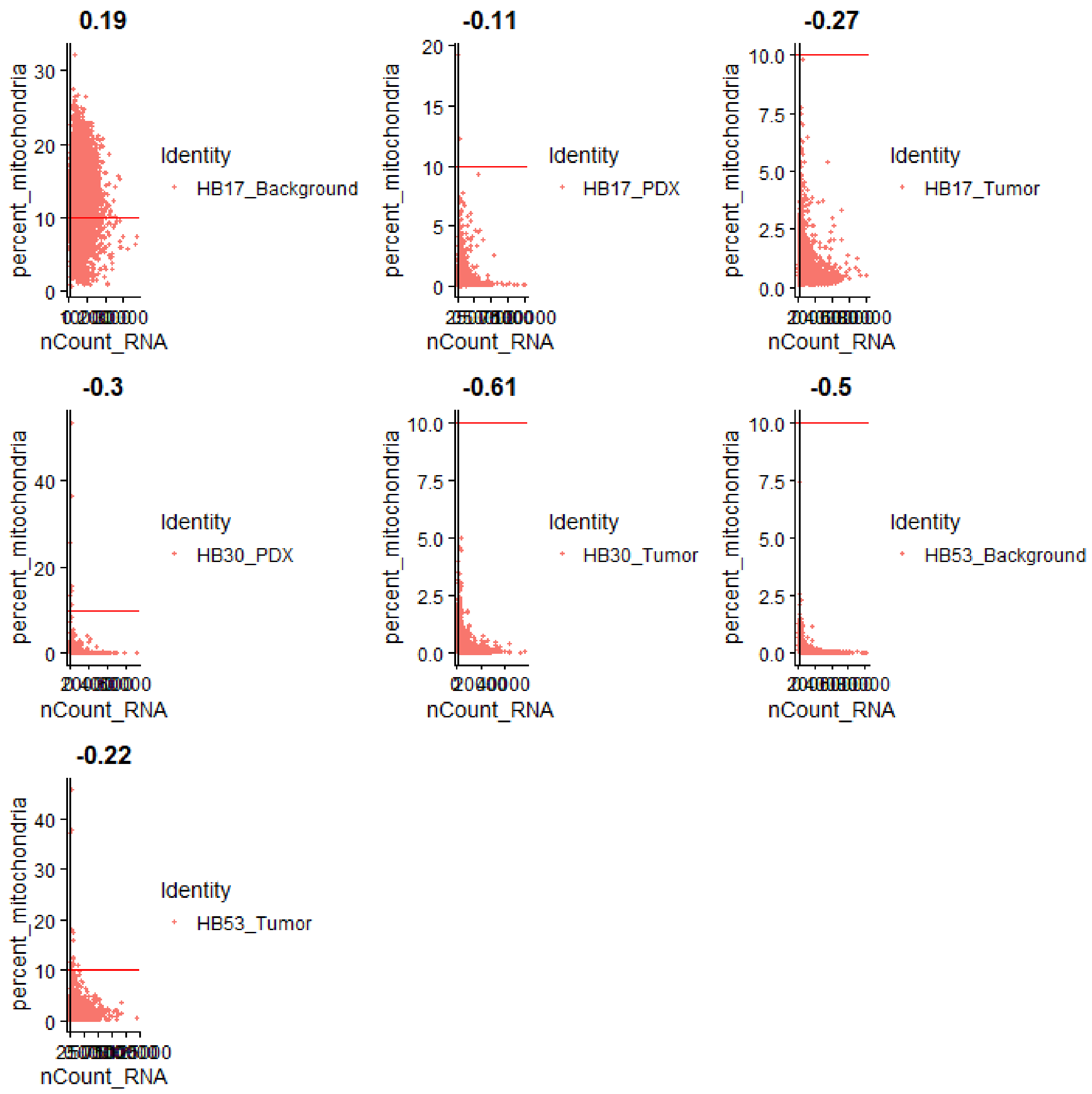
```r
}

# Combine the violin plots for each sample on the same graph
wrap_plots(Vplot_1, Vplot_2, Vplot_3, Vplot_4, Vplot_5, Vplot_6, Vplot_7)
```



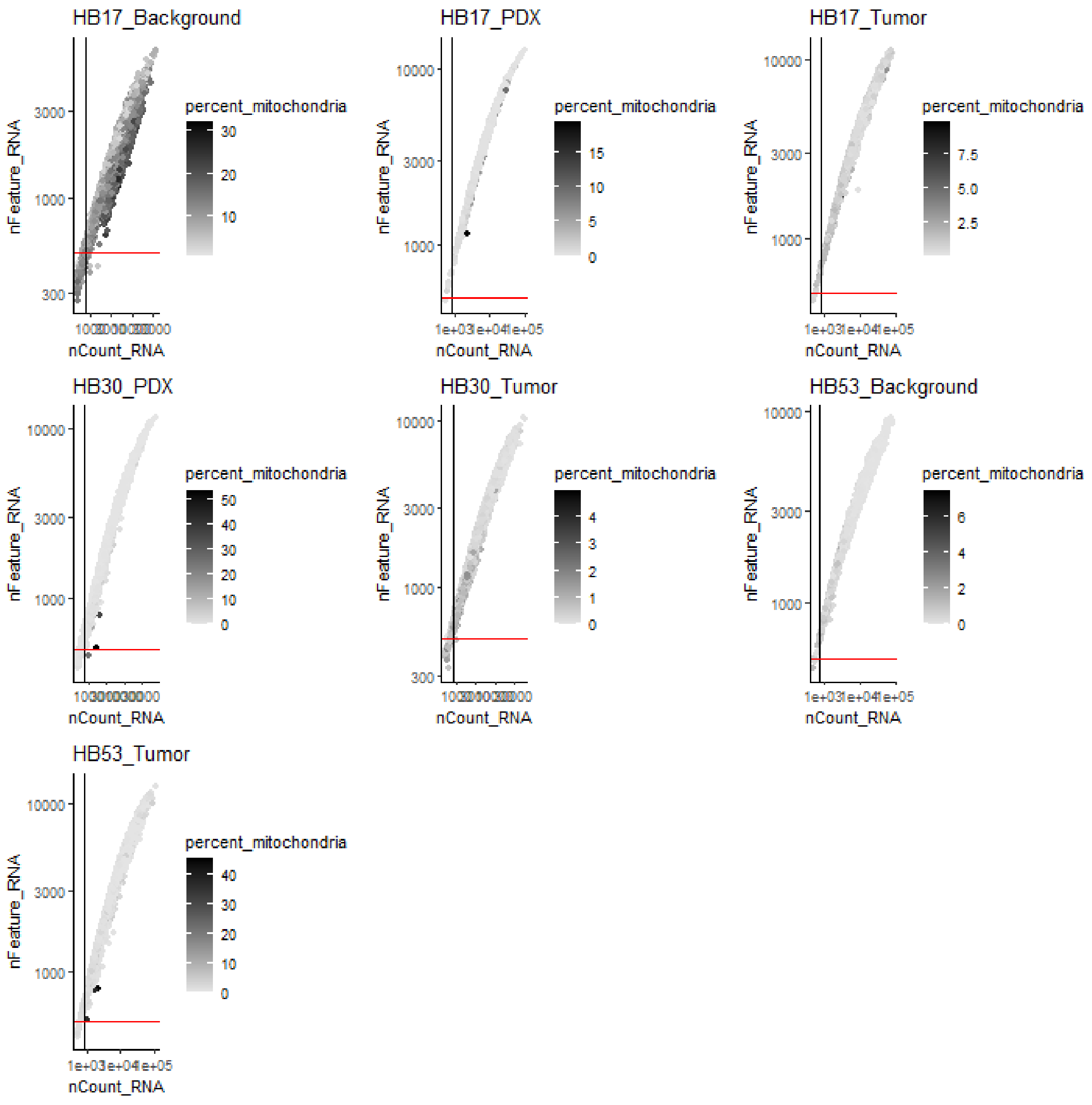
```r
rm(Vplot_1, Vplot_2, Vplot_3, Vplot_4, Vplot_5, Vplot_6, Vplot_7)


## Create a violin plot representing the fraction of total UMIs
## mapping to mitochondrial genes detected in each cell
## (percent_mitochondria) for each sample.

for (i in 1:length(samples)) {
    Violin_plot <- VlnPlot(samples[[i]], features = c("percent_mitochondria"),
        pt.size = 0.1) & geom_hline(yintercept = 10, color = "red") & theme(plot.title = element_text(s
        axis.text.x = element_text(angle = 0, hjust = 0.55))

    assign(paste0("Vplot_", i), Violin_plot)

    rm(Violin_plot)
}

# Combine the violin plots for each sample on the same graph
wrap_plots(Vplot_1, Vplot_2, Vplot_3, Vplot_4, Vplot_5, Vplot_6, Vplot_7)
```



```
rm(Vplot_1, Vplot_2, Vplot_3, Vplot_4, Vplot_5, Vplot_6, Vplot_7)


## Create a scatter plot of fraction of total UMIs mapping to
## mitochondrial genes detected in each cell (percent_mitochondria)
## against the total number of RNA molecules or UMIs detected in each
## cell (nCount_RNA) for each sample.

for (i in 1:length(samples)) {
    scatter_plot <- FeatureScatter(samples[[i]], feature1 = "nCount_RNA",
        feature2 = "percent_mitochondria") & geom_hline(yintercept = 10,
        color = "red") & geom_vline(xintercept = 800, color = "black")

    assign(paste0("Splot_", i), scatter_plot)

    rm(scatter_plot)
}

wrap_plots(Splot_1, Splot_2, Splot_3, Splot_4, Splot_5, Splot_6, Splot_7)
```



```
rm(Splot_1, Splot_2, Splot_3, Splot_4, Splot_5, Splot_6, Splot_7)


## Create a scatter plot of number of unique genes (nFeature_RNA)
## against the total number of RNA molecules or UMIs detected in each
## cell (nCount_RNA) for each sample. In this scatter plot, the
## fraction of total UMIs mapping to mitochondrial genes detected in
## each cell (percent_mitochondria) is also superimposed onto the
## plot, as indicated by the colour of the dots.  The x and y axes
## are transformed to a log10 scale.

for (i in 1:length(samples)) {

    # The %>% operator is a piping operator, which directs the sample
    # meta data to ggplot.
    scatter_plot <- samples[[i]]@meta.data %>%
        ggplot(aes(x = nCount_RNA, y = nFeature_RNA, color = percent_mitochondria)) +
        geom_point() + scale_colour_gradient(low = "gray90", high = "black") +
        scale_x_log10() + scale_y_log10() + theme_classic() + ggtitle(sample_names[i]) +
        geom_vline(xintercept = 800) + geom_hline(yintercept = 500, color = "red")

    assign(paste0("Splot_", i), scatter_plot)

    rm(scatter_plot)
}

wrap_plots(Splot_1, Splot_2, Splot_3, Splot_4, Splot_5, Splot_6, Splot_7)
```



```r
rm(Splot_1, Splot_2, Splot_3, Splot_4, Splot_5, Splot_6, Splot_7)

gc()
```

```
##              used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells    3713920  198.4    6991402  373.4   5261398  281.0
## Vcells 345852549 2638.7  513012132 3914.0 510485003 3894.7
```

In the above plots of the cell quality indicators, it is important to set thresholds for filtering out cells in a way that is more permissive, such that the risk of eliminating potentially viable cells is reduced. In general, we would want to eliminate cells that have too few RNA transcripts and expressed genes, as well as high mitochondrial gene expression. This is because these can indicate a dead or dying cell with a ruptured cell membrane, which enables leakage of mRNA located outside of the nucleus (Luecken et al., 2019). At this point, most of the mRNA left in the cell would be the ones located in the mitochondria. Although plots of the number of unique genes, total number of UMIs and the percentage of the total UMIs mapping to mitochondrial genes in each cell were generated across all the samples, the respective thresholds should not be set by only evaluating these plots separately. The quality indicators should be evaluated together to visualize how the indicators are related to each other, which would better inform thresholding decisions. The scatter plot of the nFeature_RNA against nCount_RNA above shows that using the thresholds set by the authors (indicated by the lines x = 800 (red line) and y = 500(black line)) did not remove a substantial number of cells for almost all of the samples, except for the HB17_background sample, which has a number of cells with more than 10% of the reads mapping to mitochondrial genes. Since, the thresholds set by the authors appear to be reasonable and sufficient for our data, we decided to use the same thresholds as specified in the paper for eliminating low quality cells.

The same scatter plot was created for the filtered counts matrices generated by the authors, as shown below. It can be seen that the distribution of the cells in each sample is very similar to the corresponding plots above. One observable difference is that our counts matrices appear to have generated more cells that have less gene counts and UMI counts than the threshold values.

The following code chunk demonstrates how the low quality cells were filtered out of the dataset. However, because of the amount of memory this step and the prior steps take, this code chunk will not be run during knitting of this document. Instead, the Seurat object containing the merged dataset, which is the output of this code chunk, has been saved as an RDS file that is loaded in the subsequent step.

```r
## Filter out the samples, such that only cells with nFeature_RNA >
## 500 & nCount_RNA > 800 & percent_mitochondria < 10 remain. Also,
## since the data from the different samples will be integrated
## together before clustering step, need to modify the cell names to
## indicate the sample it came from. This is done to differentiate
## cell barcodes that belong to one sample from those belonging to
## other samples. The individual sample data sets are then merged to
## form a single data set for downstream analysis.

for (i in 1:length(samples)) {

    # This method of applying the thresholds to each column of a
    # sample's meta data came from
    # https://satijalab.org/seurat/articles/pbmc3k_tutorial.html.
    samples[[i]] <- subset(samples[[i]], subset = nFeature_RNA > 500 &
        nCount_RNA > 800 & percent_mitochondria < 10)

    # This method modifies the cell barcode names by prepending the
    # sample name to the barcode
```

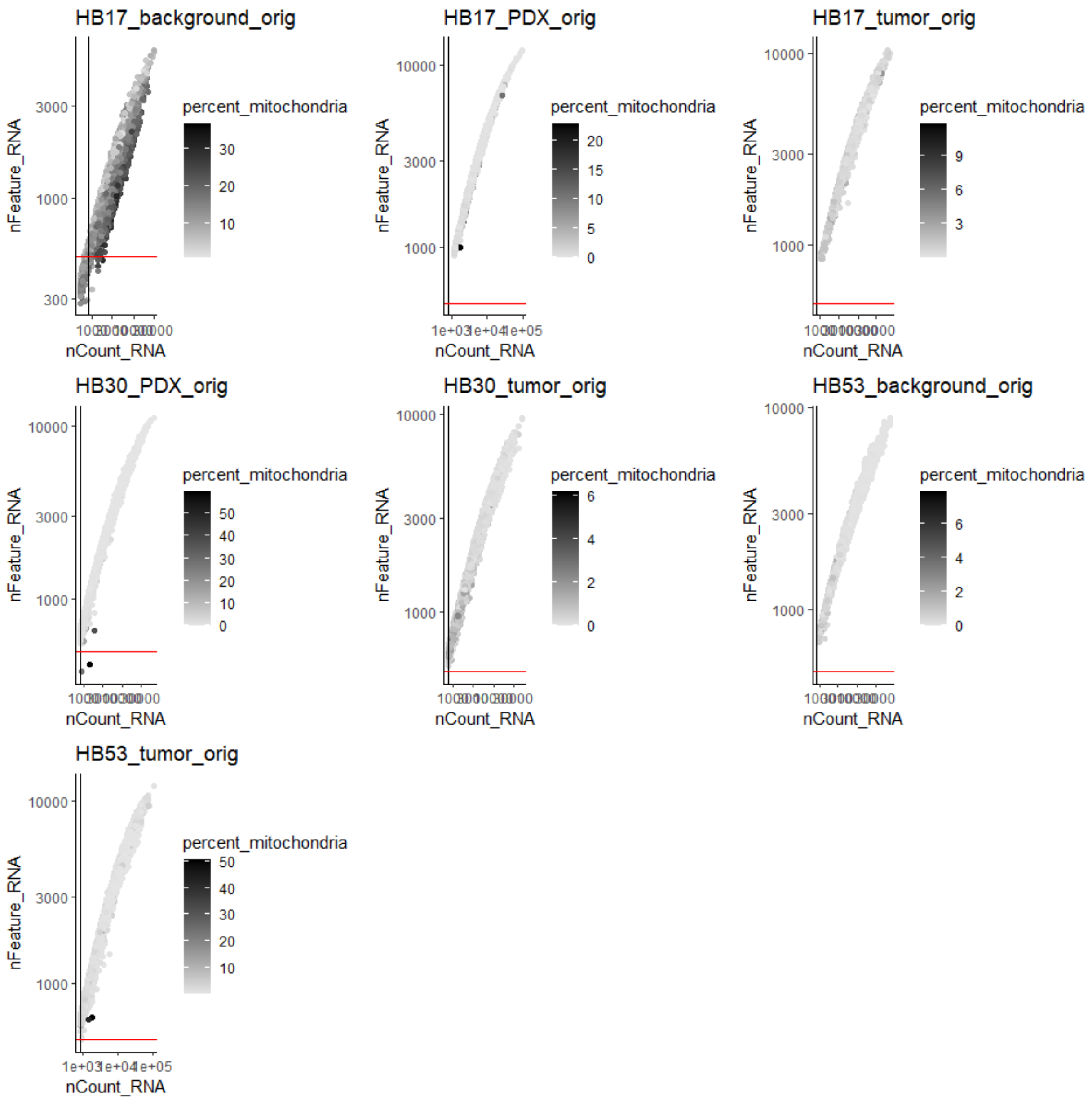


Figure 7: Quality control metrics for authors' CellRanger output counts matrices

```
    samples[[i]] <- RenameCells(object = samples[[i]], add.cell.id = sample_names[i])

    # Add a `sample_group` column to the meta data for each sample.
    samples[[i]]$sample_group <- gsub(".*_", "", sample_names[i])

    # Add a `sample` column to the meta data for each sample.
    samples[[i]]$sample <- sample_names[i]

}

## Merge the sample data sets to a single dataset for downstream
## analysis.

merged_samples <- merge(samples[[1]], y = c(samples[[2]], samples[[3]],
    samples[[4]], samples[[5]], samples[[6]], samples[[7]]))
```

After removing the low quality cells across all samples, we were left with a total of 76,310 cells. On the other hand, the authors reported that they retained 67,111 cells after the filtering step, which is 9,200 cells less than what we got. One of the reasons for this discrepancy could be the inclusion of intronic reads when the CellRanger `count` pipeline was executed. However, when we ran the authors' filtered counts matrix through our preprocessing steps above, we still did not get the same number of retained cells they reported. We retained 67,851 cells, which is 740 cells more than what they reported. It is possible that the authors performed additional filtering steps that were not specified in the paper.

To free up memory for the subsequent steps, remove the Seurat objects that are no longer needed in the workspace.

```
rm(samples, HB17_Background_seurat, HB17_PDX_seurat, HB17_Tumor_seurat,
    HB30_PDX_seurat, HB30_Tumor_seurat, HB53_Background_seurat, HB53_Tumor_seurat)

gc()
```

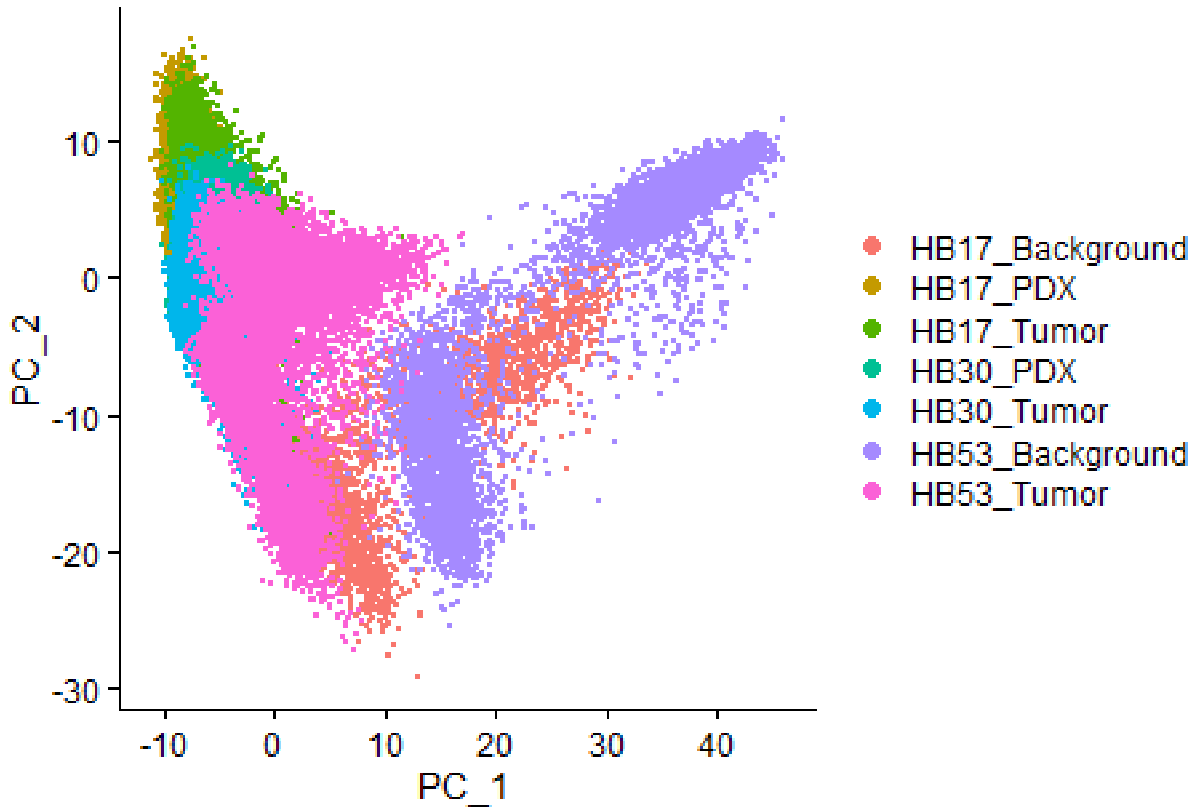
```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3125468  167.0    6991402   373.4    6991402   373.4
## Vcells 637031168 4860.2 2614359463 19946.0 2062388543 15734.8
```

Part 3: Normalization and Integration

Just as with any other genomics analysis, the data, in this case, the UMI counts need to be normalized, such that they are comparable between cells. In this study, the authors normalized the total UMI counts in each cell to 10,000 followed by a natural log transformation. The log transformation in genomics analysis is important especially when the values being compared are orders of magnitude different from each other. Although there are other ways to normalize the total UMI counts, we will follow the method implemented by the authors. The following illustrates the steps implemented by the authors, as described in the paper, for integration and dimensionality reduction of the single cell RNA sequencing data.

```
# The `LogNormalize` method divides the gene counts for each cell the
# total UMI count for that cell and multiplied by the scale.factor.
# Then, a natural-log transformation is performed using log(x+1).
merged_samples_norm <- NormalizeData(merged_samples, normalization.method = "LogNormalize",
    scale.factor = 10000)
```

```
## In the feature selection step, the authors identified the top 2000
## genes with the highest cell-to-cell variability.

merged_samples_norm <- FindVariableFeatures(merged_samples_norm, selection.method = "vst",
    nfeatures = 2000)

merged_samples_norm <- ScaleData(merged_samples_norm)  # Scale the data so that

merged_samples_norm <- RunPCA(merged_samples_norm)

# Plot the top two principal components to see how the samples are
# grouped together
DimPlot(merged_samples_norm, reduction = "pca")
```

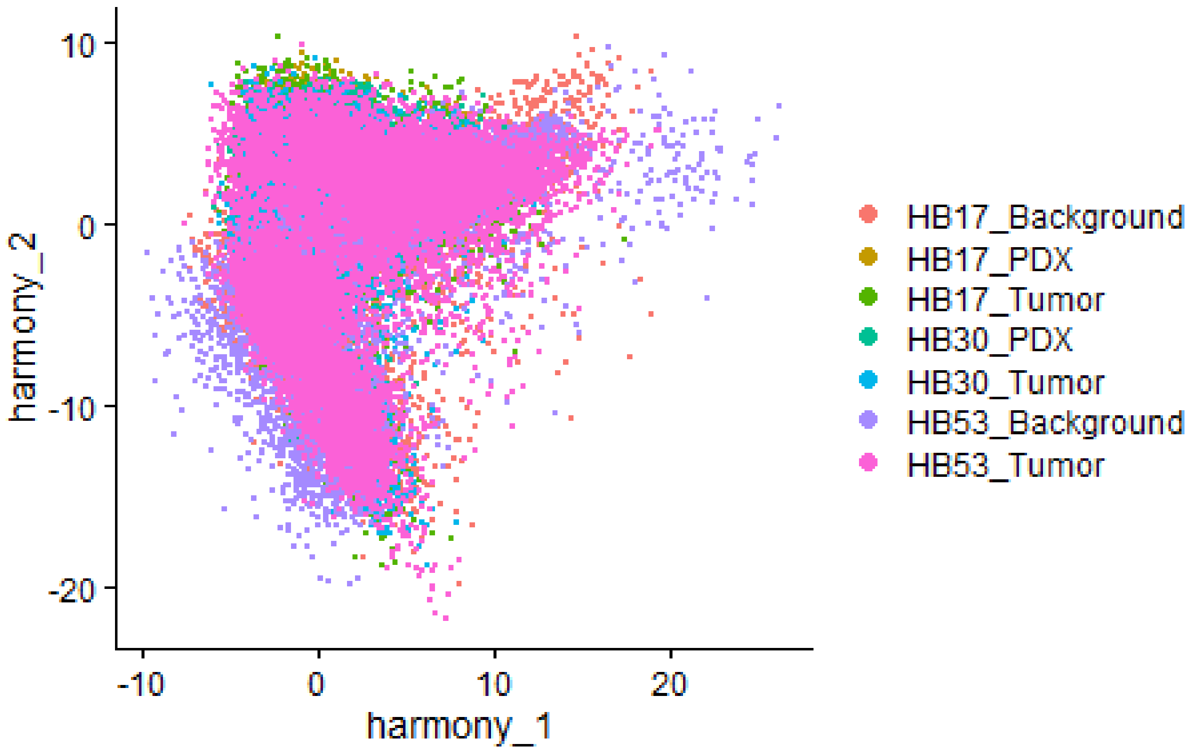


```
gc()
```

```
##               used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells     3089961  165.1    6991402   373.4    6991402   373.4
## Vcells 1109843690 8467.5 2614359463 19946.0 2062388543 15734.8
```

The plot of the top two principal components (PC_1 and PC_2) shows that there is a difference or a separation between the different tumor (HB17, HB30 and HB53) and PDX samples (HB17 and HB30), as well as between the background samples. We would expect that the cells from the tumor and PDX samples, and the background samples, would have more overlap, regardless of the patient sample. This could be due to the presence of batch effects, which are technical (non-biological) variations that can mask the true biological differences across the samples. In order to correct for these batch effects and perform data integration, we used a function called `RunHarmony` from the `Harmony` R package, which is the same function used in the paper.

```
## Integration of data across different samples (Background, PDX and
## Tumor)

# `RunHarmony` performs batch correction on the samples to eliminate
# technical variation between samples. It corrects the principal
# component (PC) scores assigned to each cell. `group.by.vars`
# specifies which variable to eliminate. It will use all PC
# dimensions (50 PCs) that resulted from running the `RunPCA`
# function.
merged_samples_norm <- RunHarmony(merged_samples_norm, group.by.vars = "sample")

DimPlot(merged_samples_norm, reduction = "harmony", group.by = "sample") +
    ggtitle("")
```



After the data integration step, the above plot of the corrected principal components (PCs) shows that the tumor and PDX samples overlap, and the background samples overlap with other background samples. This indicates that the batch correction was able to filter out technical variations that may have been present in the data. Following integration, the data can now be used for clustering analysis. Before proceeding to the clustering step, we decided to continue processing the authors' filtered counts matrices to compare against our results thus far. The authors' counts matrices were subjected to the exact same preprocessing pipeline above, followed by normalization and integration. The resulting plots of the top two PCs before and after integration are shown below.

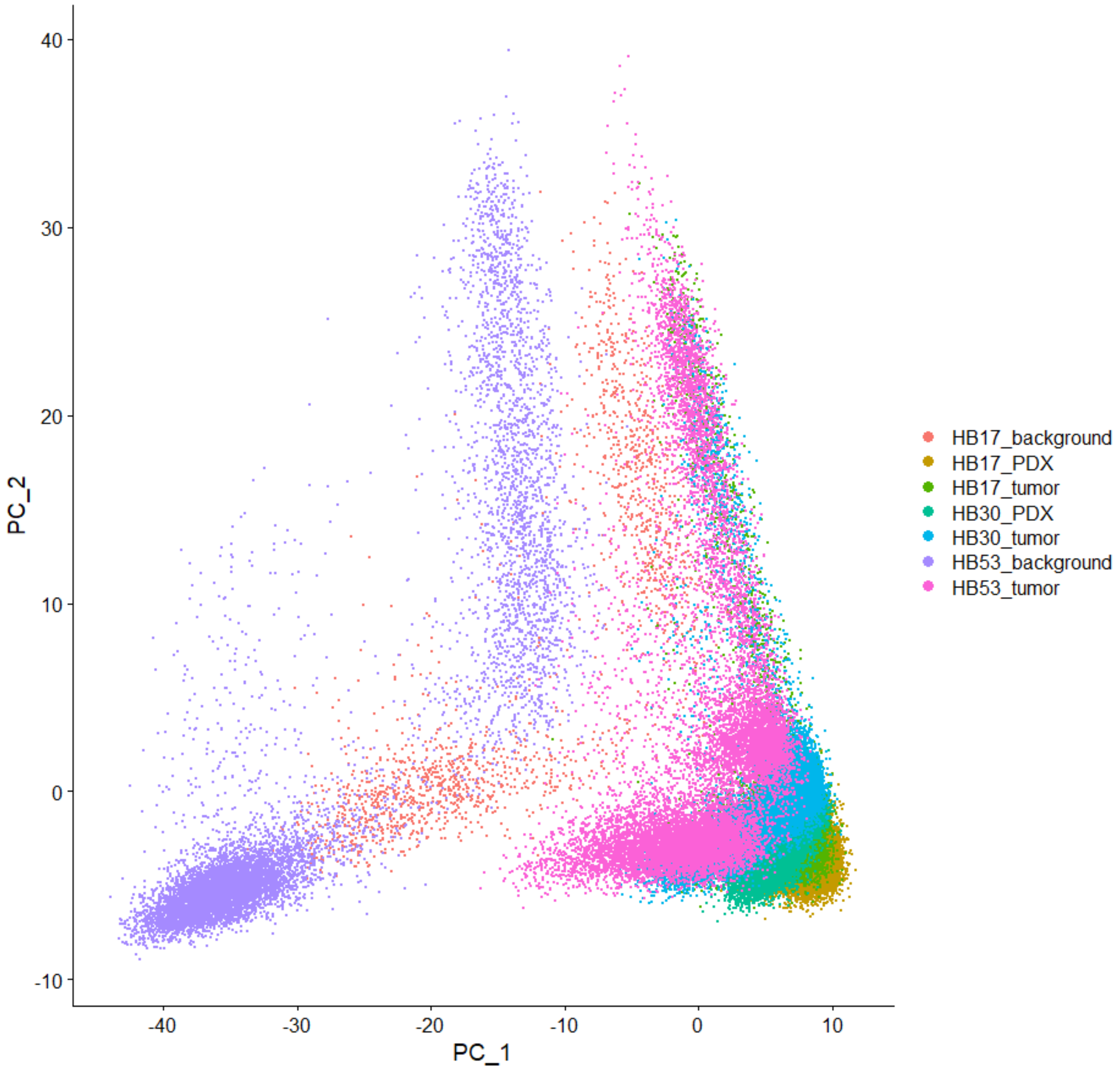


Figure 8: Authors' filtered counts data before batch correction and integration

Comparing these plots to the corresponding plots above, they appear to be very similar, except that when using the authors' filtered counts data, it appears that the plot is rotated 180 degrees. Upon checking the genes driving the PCs, the genes that are positively associated with PC1 in the authors' filtered counts are negatively associated with PC1 in our processed version of the filtered counts, and vice versa. This is an interesting observation, however, it may not be a problem as long as the different cell types are able to form their respective clusters.

In order to perform clustering, there is a need to determine how many corrected PCs (Harmony) to include in the clustering step. The number of corrected PCs to include must capture the majority of the variation present in the dataset. One way to check this amount of variation captured at each corrected PC is by plotting the standard deviation against the PC.

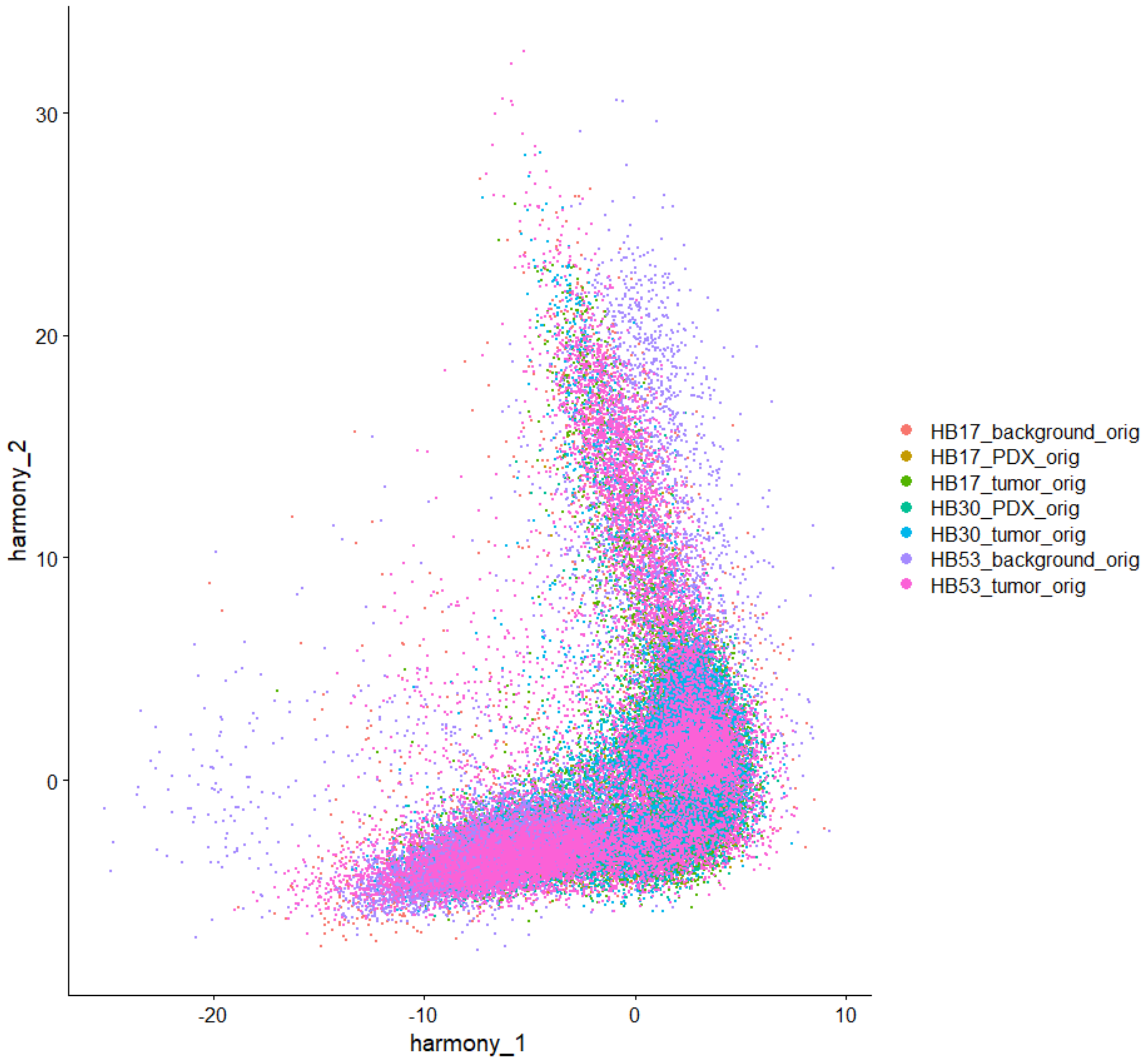
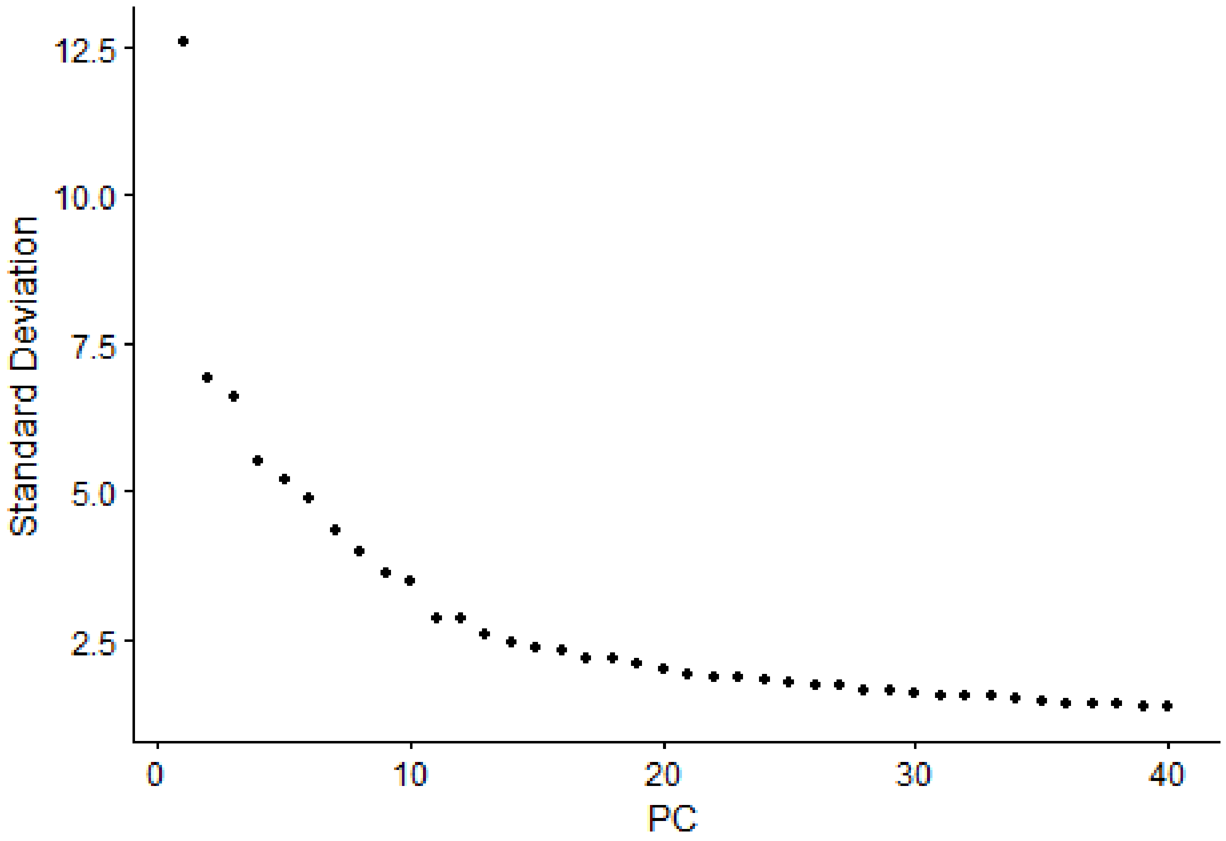


Figure 9: Authors' filtered counts data after batch correction and integration

```
gc()
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3095593  165.4    6991402   373.4    6991402   373.4
## Vcells 1112758378 8489.7 2614359463 19946.0 2062388543 15734.8
```

```
# This idea for plotting the standard deviation against the PC came
# from:
# https://github.com/hbctraining/scRNA-seq/blob/master/lessons/07_SC_clustering_cells_SCT.md

ElbowPlot(object = merged_samples_norm, ndims = 40)
```



The elbow plot above shows that most of the variation present in the dataset is captured in the first 10 PCs. However, to capture as much of the variation as possible and in turn, include as many different cell types as possible, we chose 30 PCs for clustering analysis. This is the same number of PCs used by the authors in the study. We chose 30 PCs for clustering because based on the above elbow plot, the amount of variation explained by PCs greater than 30 appears to plateau and including more PCs may not lead to significantly improved results.

Part 4: Clustering Analysis

To cluster the cells, we used the `FindNeighbors` and `FindClusters` functions in Seurat, just as described in the paper. The `FindClusters` function in Seurat has a resolution parameter, which determines the number of resulting clusters. The higher the resolution, the higher the number of clusters. The authors tested three different resolutions: 0.5, 1 and 2, which we replicated below. We used Uniform Manifold Approximation and Projection (UMAP), which is a dimensionality reduction technique for visualizing the clusters.

```
## Clustering

merged_samples_norm <- RunUMAP(merged_samples_norm, reduction = "harmony",
    dims = 1:30)
merged_samples_norm <- FindNeighbors(merged_samples_norm, reduction = "harmony",
    dims = 1:30)

# Test out different resolutions for determining granularity of
# clusters (ie. number of clusters )
merged_samples_norm <- FindClusters(merged_samples_norm, resolution = c(0.5,
    1, 2))
```

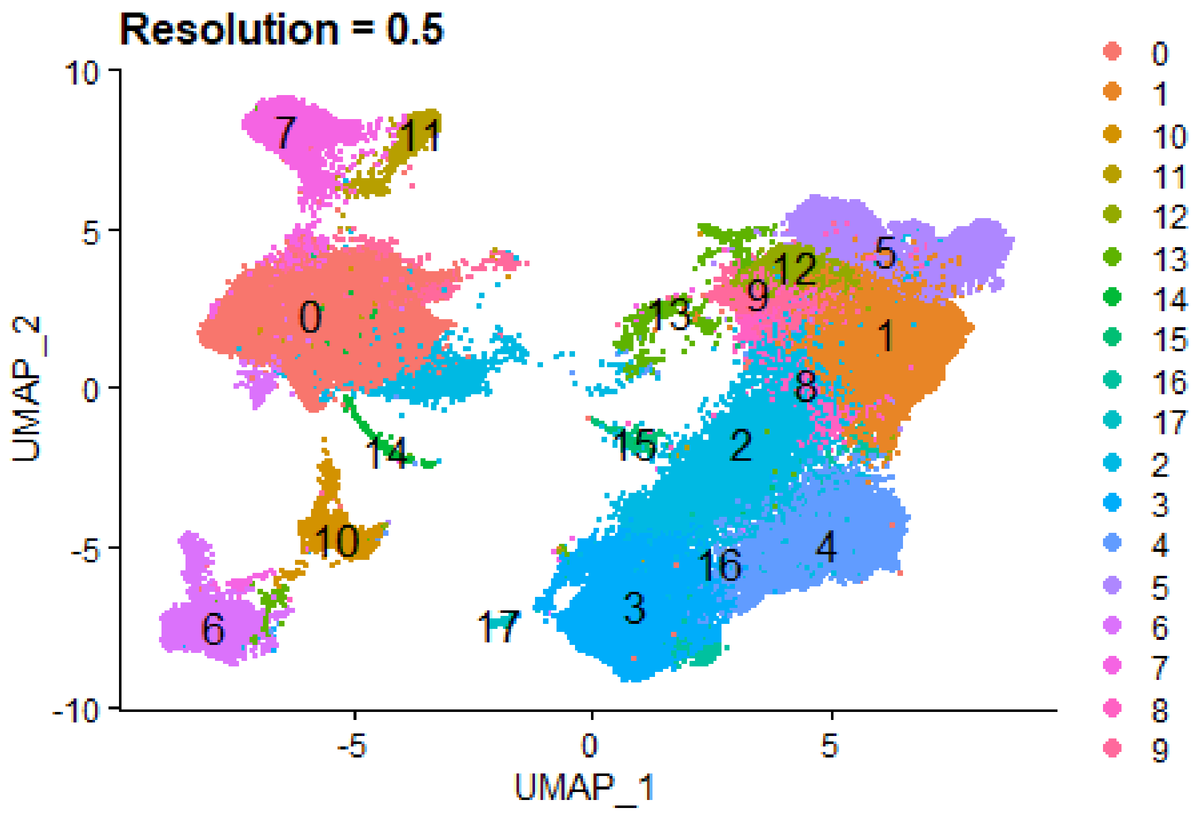
```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 76310
## Number of edges: 2264544
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9109
## Number of communities: 35
## Elapsed time: 25 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 76310
## Number of edges: 2264544
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8721
## Number of communities: 49
## Elapsed time: 29 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 76310
## Number of edges: 2264544
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8241
## Number of communities: 63
## Elapsed time: 28 seconds
```

```
gc()
```

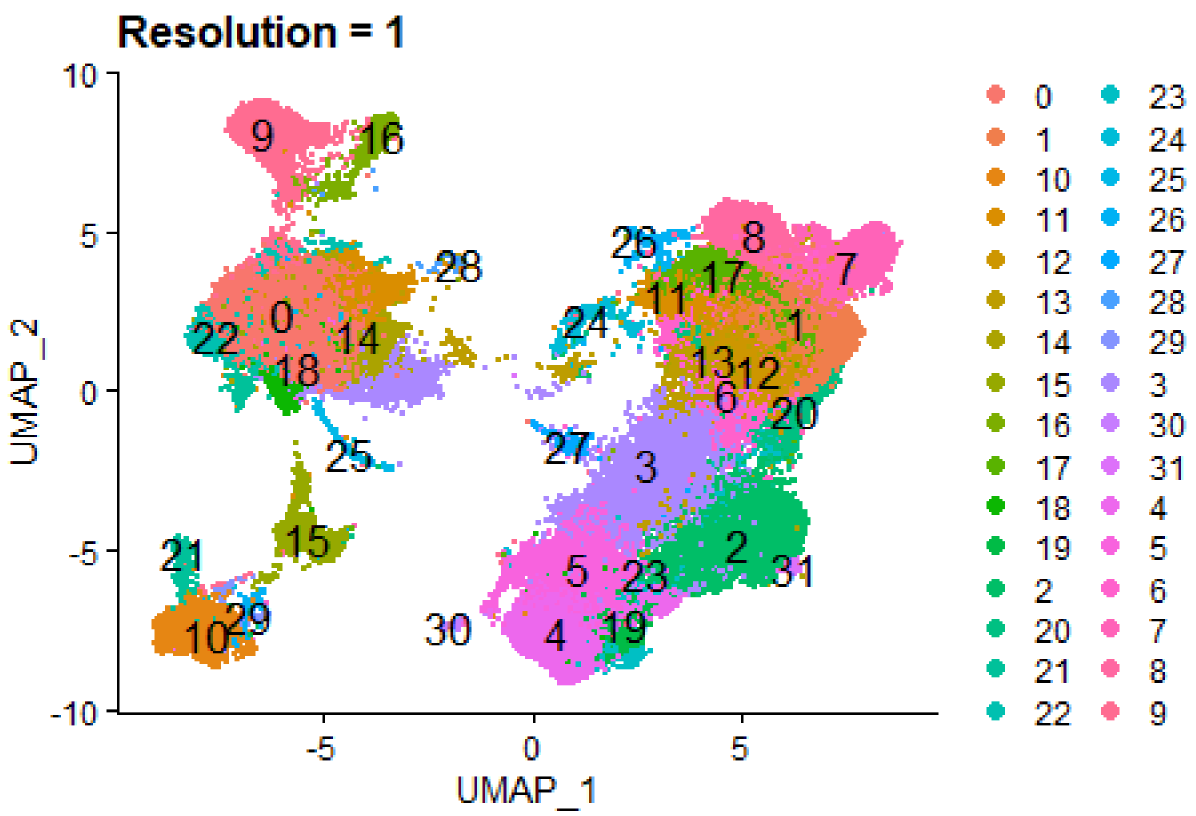
```
##               used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3167806  169.2    6991402   373.4    6991402   373.4
## Vcells 1122384309 8563.2 2614359463 19946.0 2062388543 15734.8
```

```
# Plot the UMAP at different resolutions to assist in deciding how
# many clusters to have.

Idents(merged_samples_norm) <- "RNA_snn_res.0.5"
```

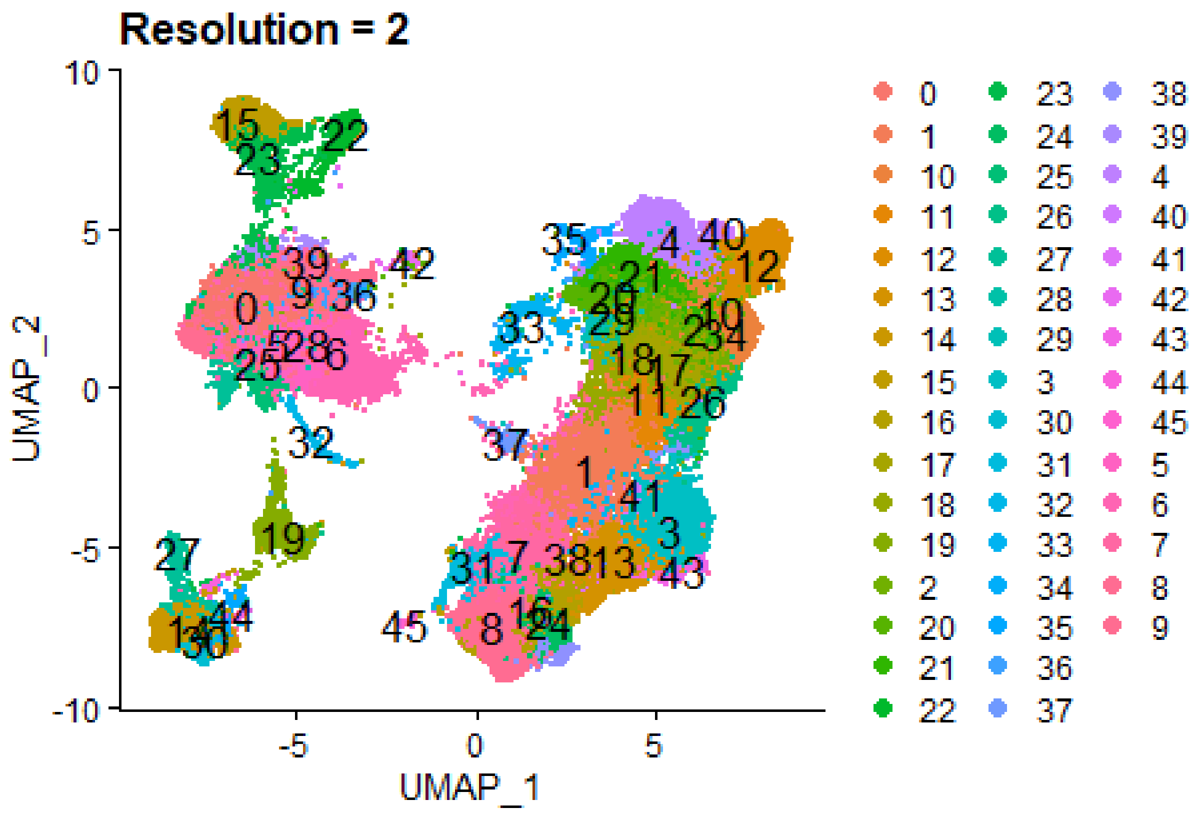
```
DimPlot(merged_samples_norm, reduction = "umap", label = TRUE, label.size = 6) +
    ggtitle("Resolution = 0.5")
```



```
Idents(merged_samples_norm) <- "RNA_snn_res.1"
DimPlot(merged_samples_norm, reduction = "umap", label = TRUE, label.size = 6) +
    ggtitle("Resolution = 1")
```



```
Idents(merged_samples_norm) <- "RNA_snn_res.2"
DimPlot(merged_samples_norm, reduction = "umap", label = TRUE, label.size = 6) +
    ggtitle("Resolution = 2")
```

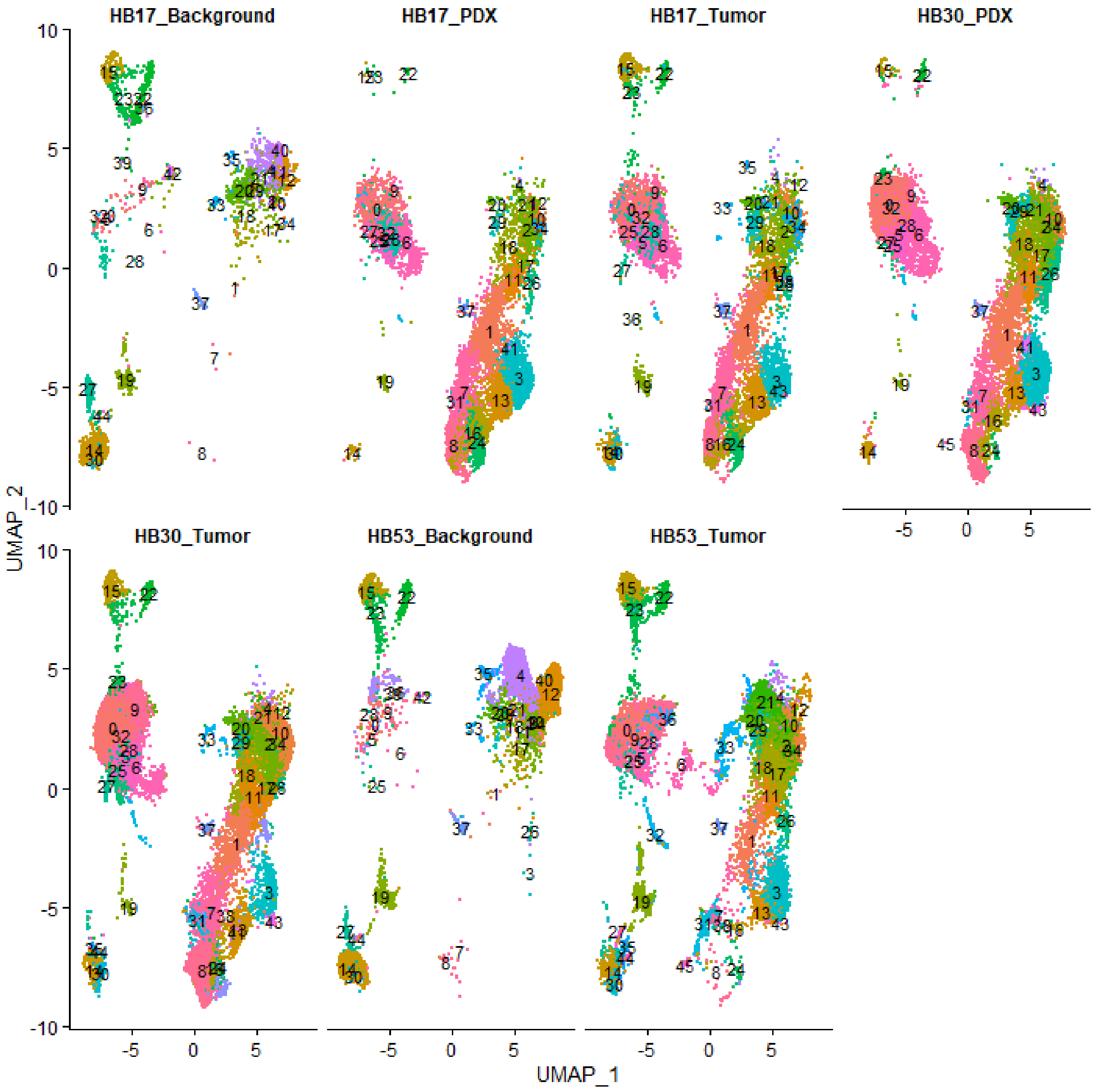


```
gc()
```

```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3599029  192.3    6991402   373.4    6991402   373.4
## Vcells 1127935977 8605.5 2614359463 19946.0 2062388543 15734.8
```

Since this is a large dataset with 76,310 cells, we decided to use a resolution of 2. As shown in the plot above, a resolution equal to 2 produced 46 clusters, while a resolution of 0.5 and 1 yielded 18 and 32 clusters, respectively. Furthermore, having more clusters (ie. higher resolution) allows for easier combination of clusters if they are identified to represent the same cell types. Otherwise, if a cluster is identified to contain two or more different cell types, then that cluster will have to be split by performing clustering again at a higher resolution. Before we proceed with annotating the clusters according to cell type and sample (Background, tumor, PDX), it is a good idea to perform a quality control step on the clusters to ensure that the clusters are formed due to differences in cell type and not due to other confounding factors. One way to implement this is by visualizing the clusters across the different samples. We would expect the clusters associated with tumor and PDX samples to be very similar, while the clusters that correspond to the background samples should exhibit high similarity. Furthermore, the distribution of the aforementioned features should also be fairly uniform across the clusters. This plot is shown below.

```
## Quality control of clustering results

# Check the clustering results for each sample by UMAP
DimPlot(merged_samples_norm, reduction = "umap", label = TRUE, ncol = 4,
    split.by = "sample") + NoLegend()
```

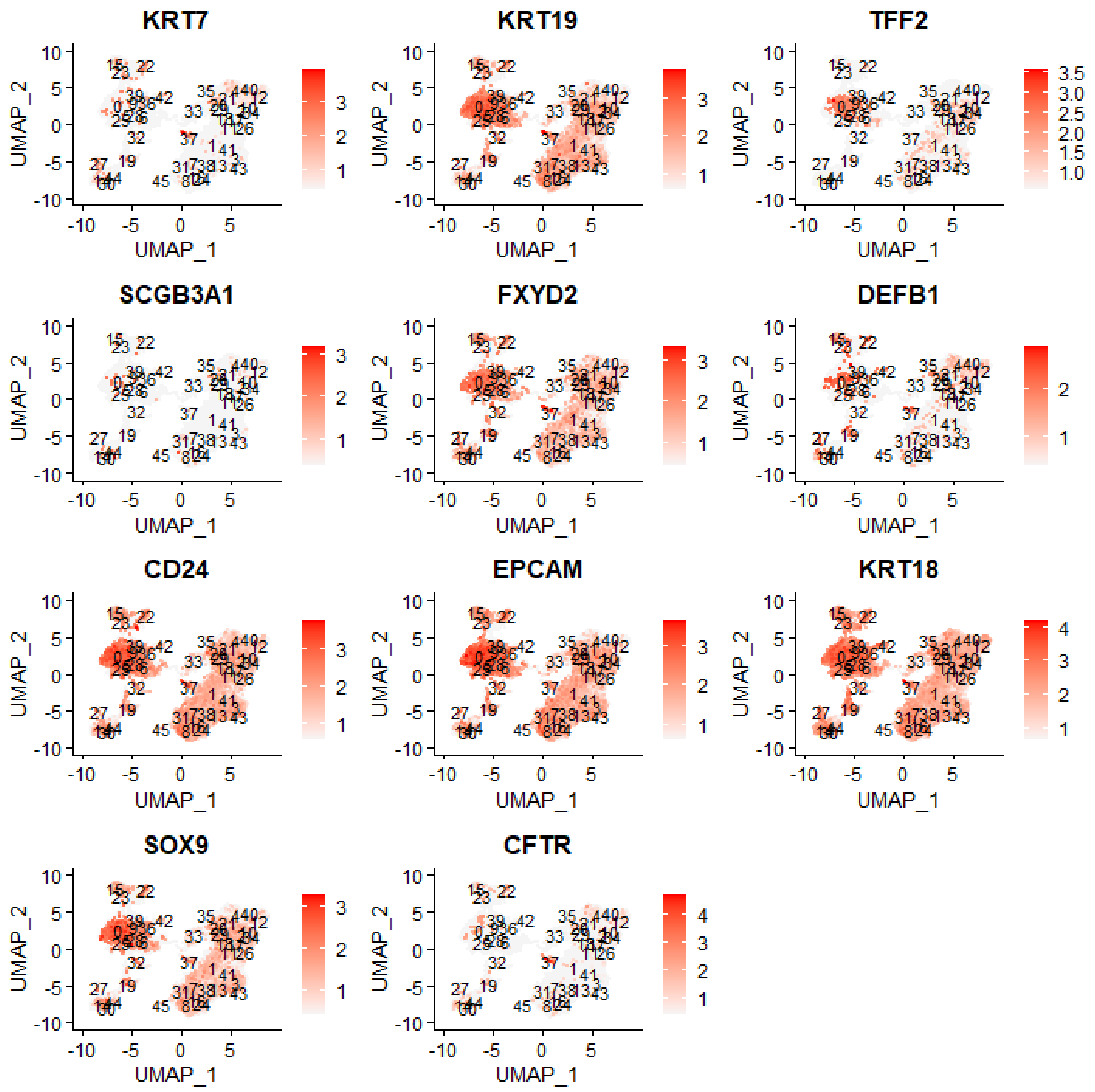


```r
gc()
```

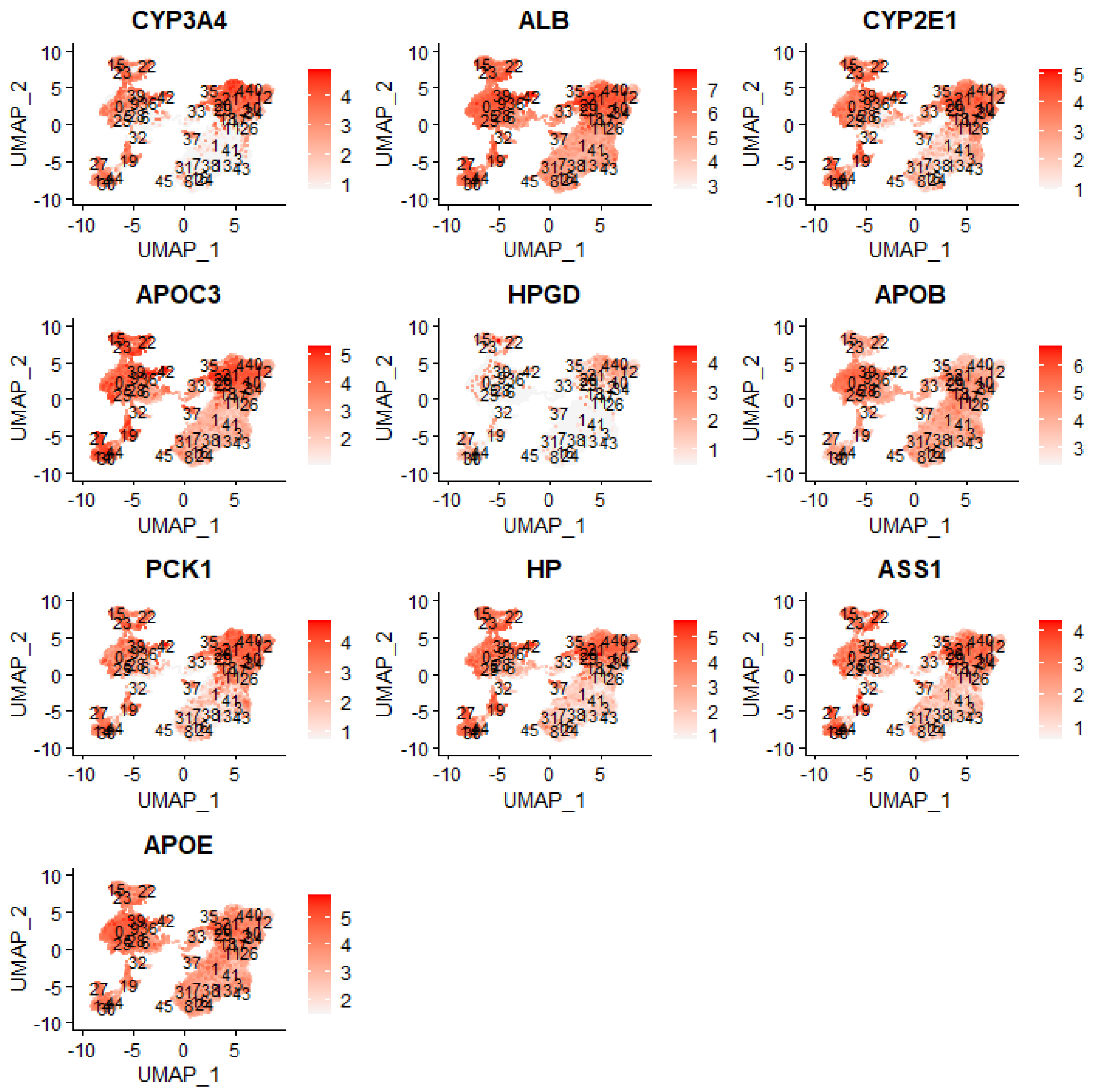
```
##              used   (Mb) gc trigger    (Mb)   max used    (Mb)
## Ncells    3237758  173.0    6991402   373.4    6991402   373.4
## Vcells 1124689874 8580.8 2614359463 19946.0 2062388543 15734.8
```

Based on the plot of the clusters across different samples, it can be seen that the generated clusters are very similar across tumor and PDX samples. The clusters are also very similar across the background samples. They are not exactly the same, however there is sufficient similarity to proceed to the cluster annotation according to cell type step. The authors did not specify in the paper how cell classes were assigned to each cluster. However, they indicated that cluster annotation was performed based on gene expression levels of known (canonical) markers and those specified in "A human liver cell atlas reveals heterogeneity and epithelial progenitors" (Aizarani et al., 2019). As a first pass attempt to assign cell classes to clusters according to the list of marker genes provided by the authors (as part of the supplementary data for the paper), we decided to qualitatively explore these marker genes by comparing plots of the clusters across the different marker genes for a particular cell class. To assign a cell class to a particular cluster, that cluster must exhibit sustained high expression of the marker genes associated with a cell class. This process is shown below.

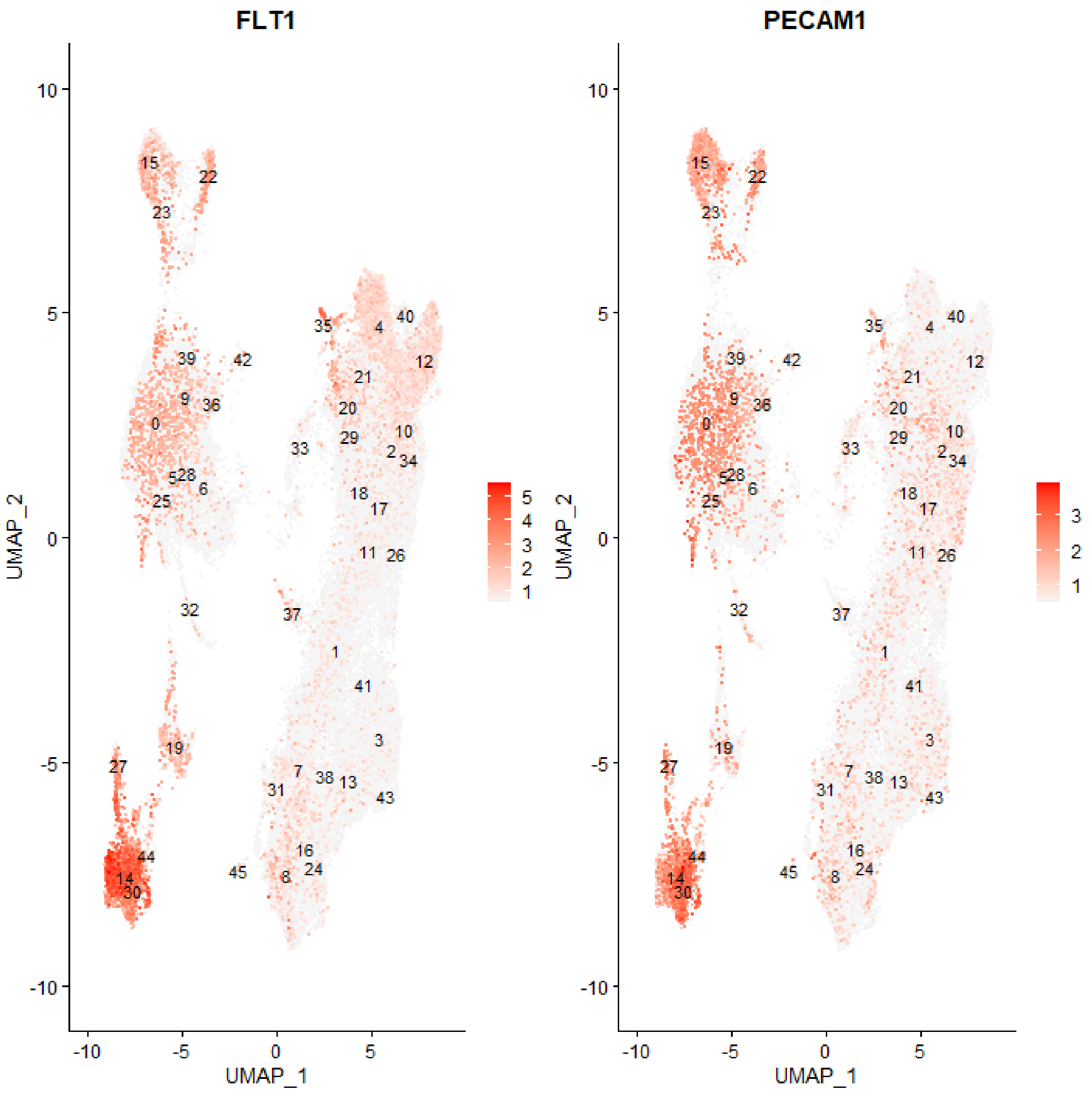
```r
## Visualizing expression of marker genes for cluster annotation
## Authors annotated clusters based on gene expression levels of
## known markers and those specified in 'A human liver cell atlas
## reveals heterogeneity and epithelial progenitors' (Aizarani et
## al., 2019).


## Marker genes for Cholangiocytes
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("KRT7",
    "KRT19", "TFF2", "SCGB3A1", "FXYD2", "DEFB1", "CD24", "EPCAM", "KRT18",
    "SOX9", "CFTR"), cols = c("#F5F5F5", "red"), ncol = 3, sort.cell = TRUE,
    min.cutoff = "q10", label = TRUE)  # CLuster 37
```



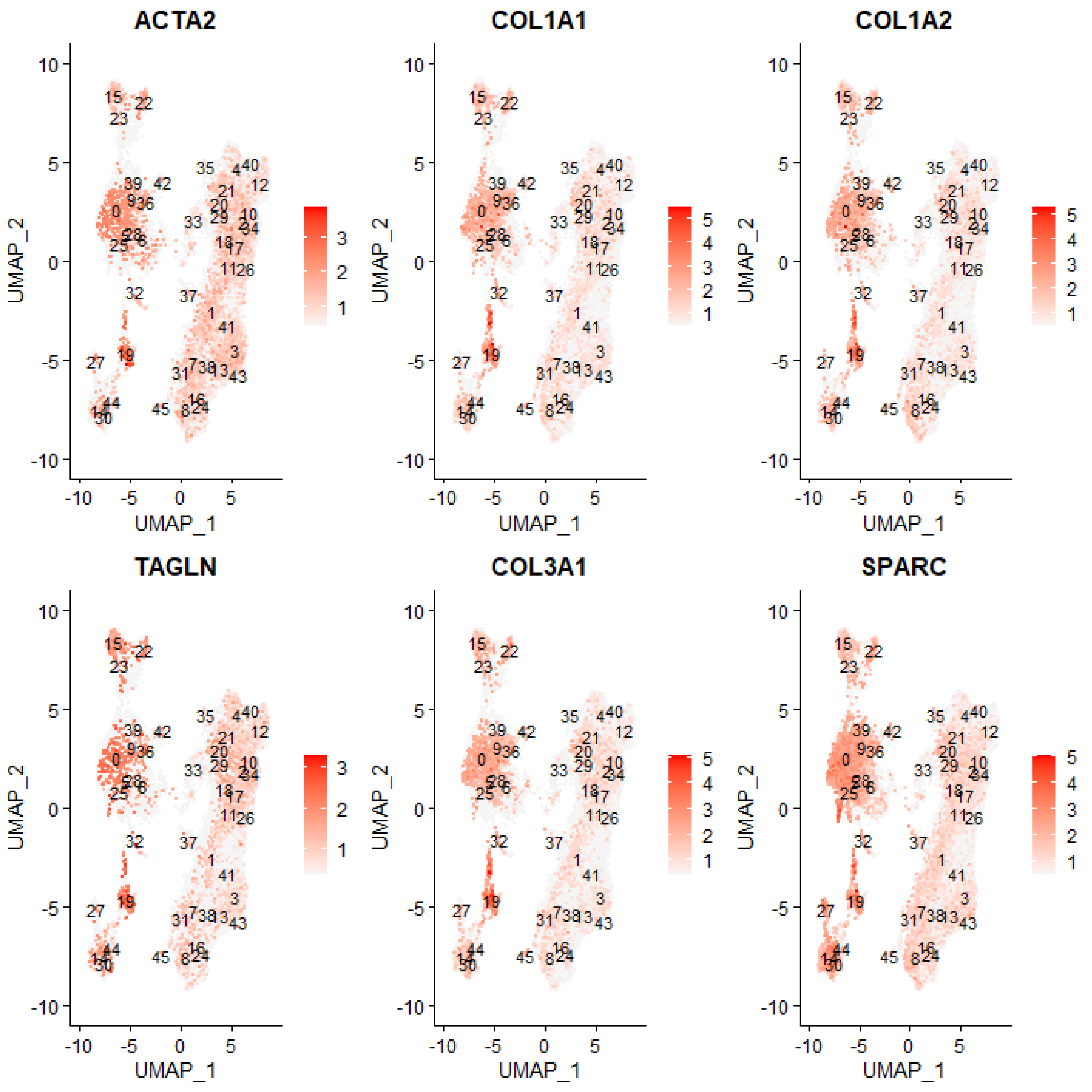
```
## Marker genes for Hepatocytes
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("CYP3A4",
    "ALB", "CYP2E1", "APOC3", "HPGD", "APOB", "PCK1", "HP", "ASS1", "APOE"),
    cols = c("#F5F5F5", "red"), ncol = 3, sort.cell = TRUE, min.cutoff = "q10",
    label = TRUE)  # Clusters 4, 12, 21, 40
```



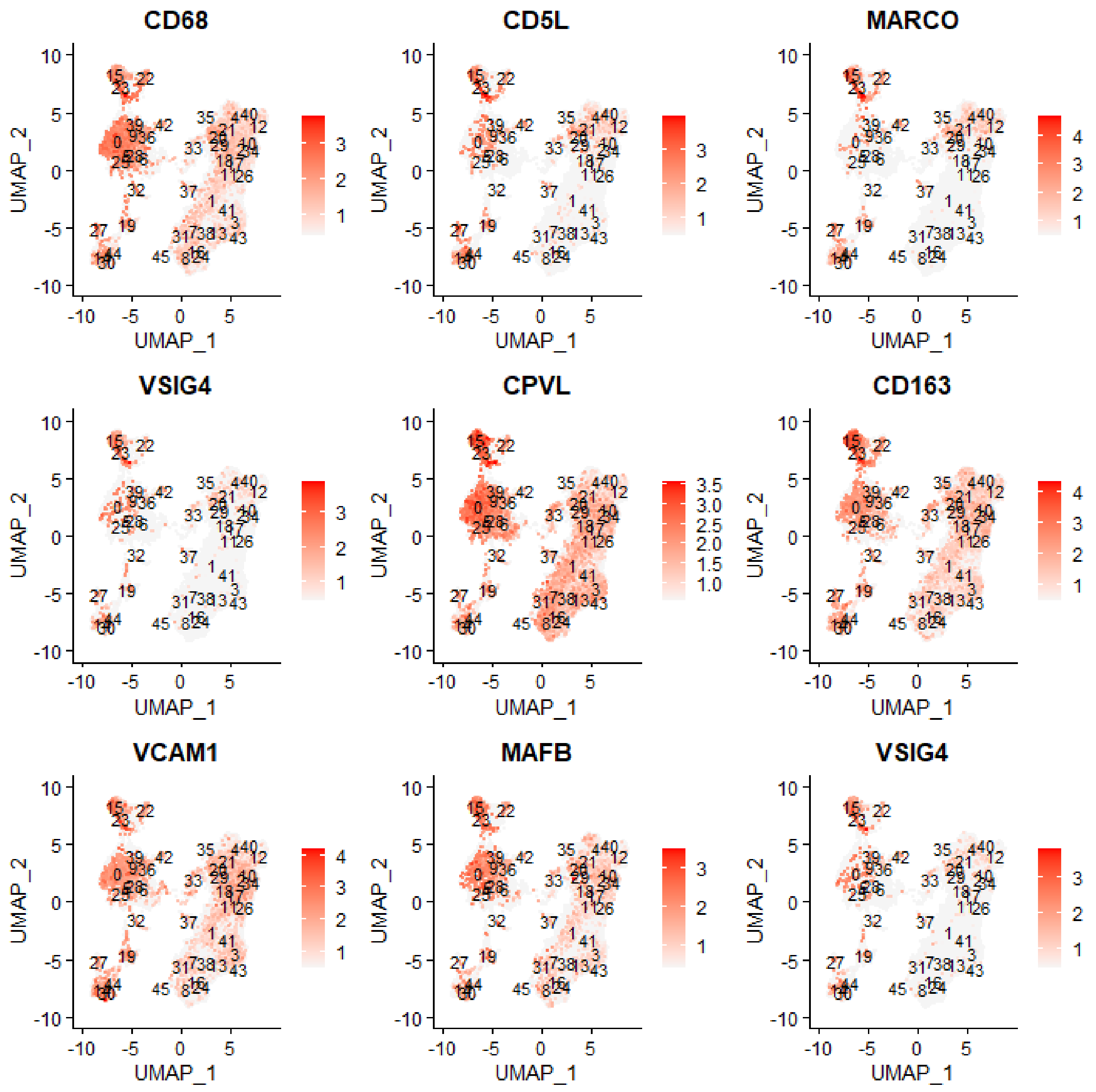
```r
## Marker genes for Endothelial cells
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("FLT1",
    "PECAM1"), cols = c("#F5F5F5", "red"), sort.cell = TRUE, min.cutoff = "q10",
    label = TRUE)  # Clusters 14, 27, 30, 44
```



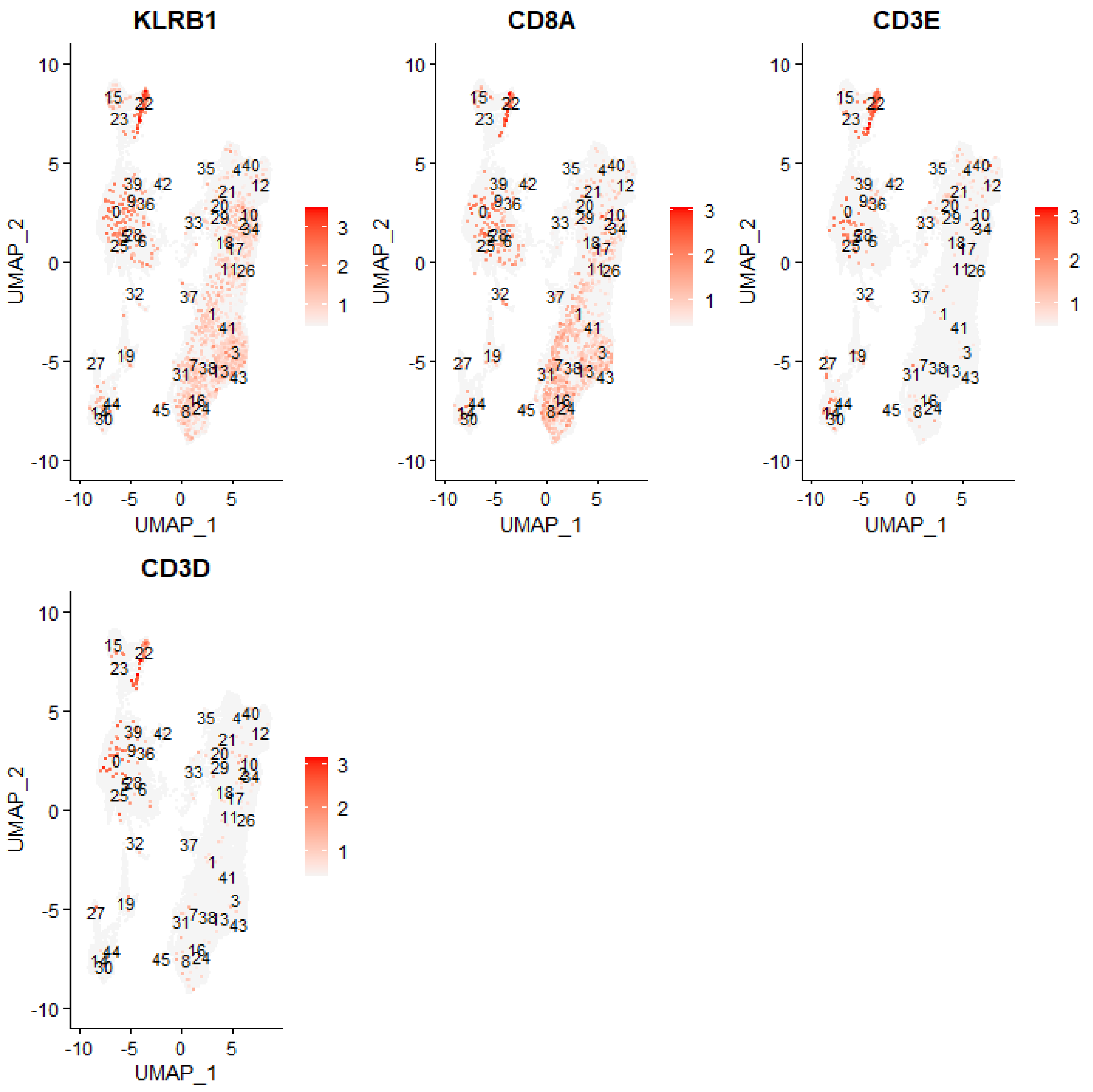
```
## Marker genes for Stellate cells
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("ACTA2",
    "COL1A1", "COL1A2", "TAGLN", "COL3A1", "SPARC"), cols = c("#F5F5F5",
    "red"), ncol = 3, sort.cell = TRUE, min.cutoff = "q10", label = TRUE)  # Cluster 19
```



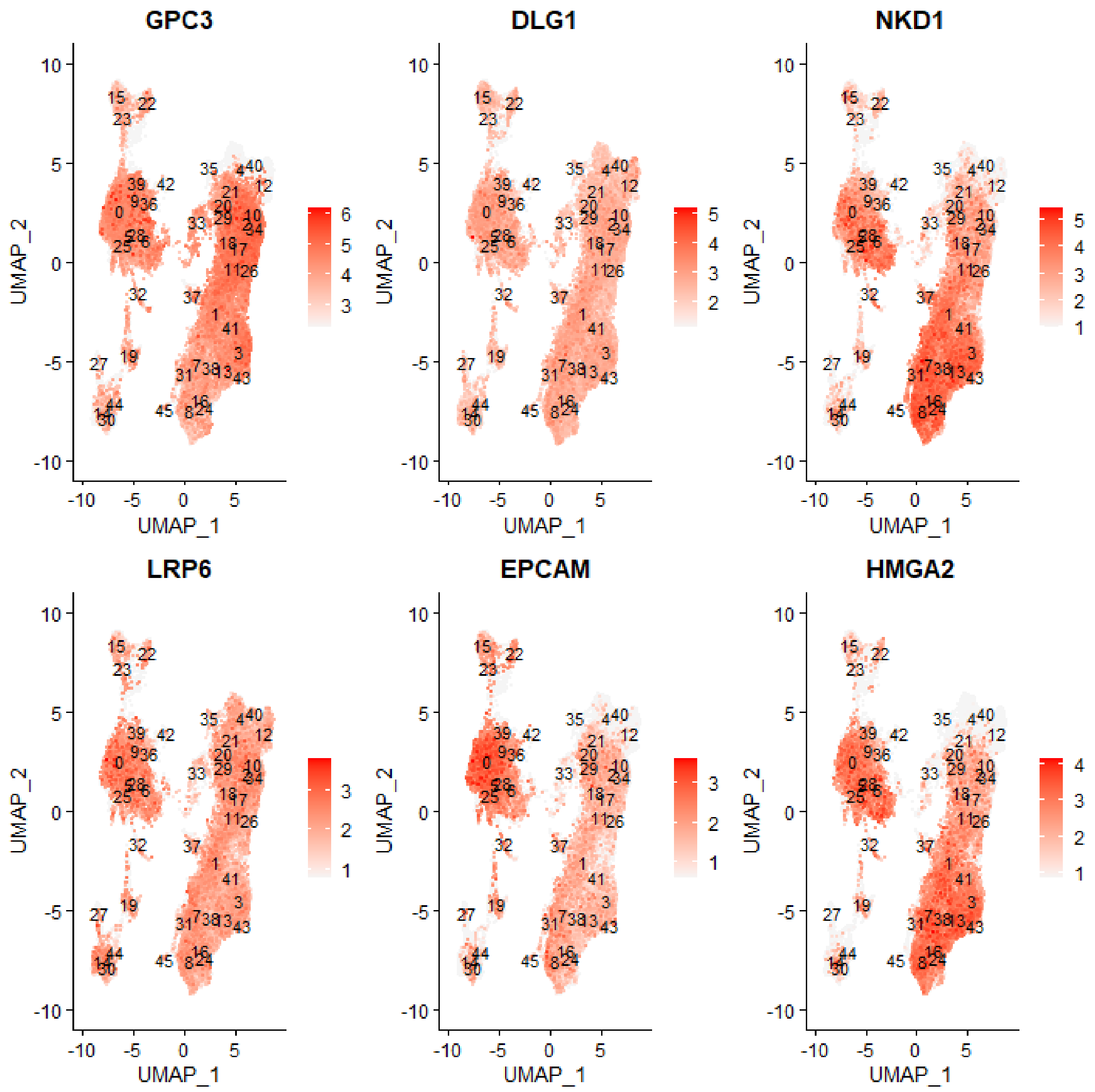
```
## Marker genes for Kupffer cells
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("CD68",
    "CD5L", "MARCO", "VSIG4", "CPVL", "CD163", "VCAM1", "MAFB", "VSIG4"),
    cols = c("#F5F5F5", "red"), ncol = 3, sort.cell = TRUE, min.cutoff = "q10",
    label = TRUE)  # Cluster 15, 23
```



```r
## Marker genes for T/NK cells
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("KLRB1",
    "CD8A", "CD3E", "CD3D"), cols = c("#F5F5F5", "red"), ncol = 3, sort.cell = TRUE,
    min.cutoff = "q10", label = TRUE)   # Cluster 22
```

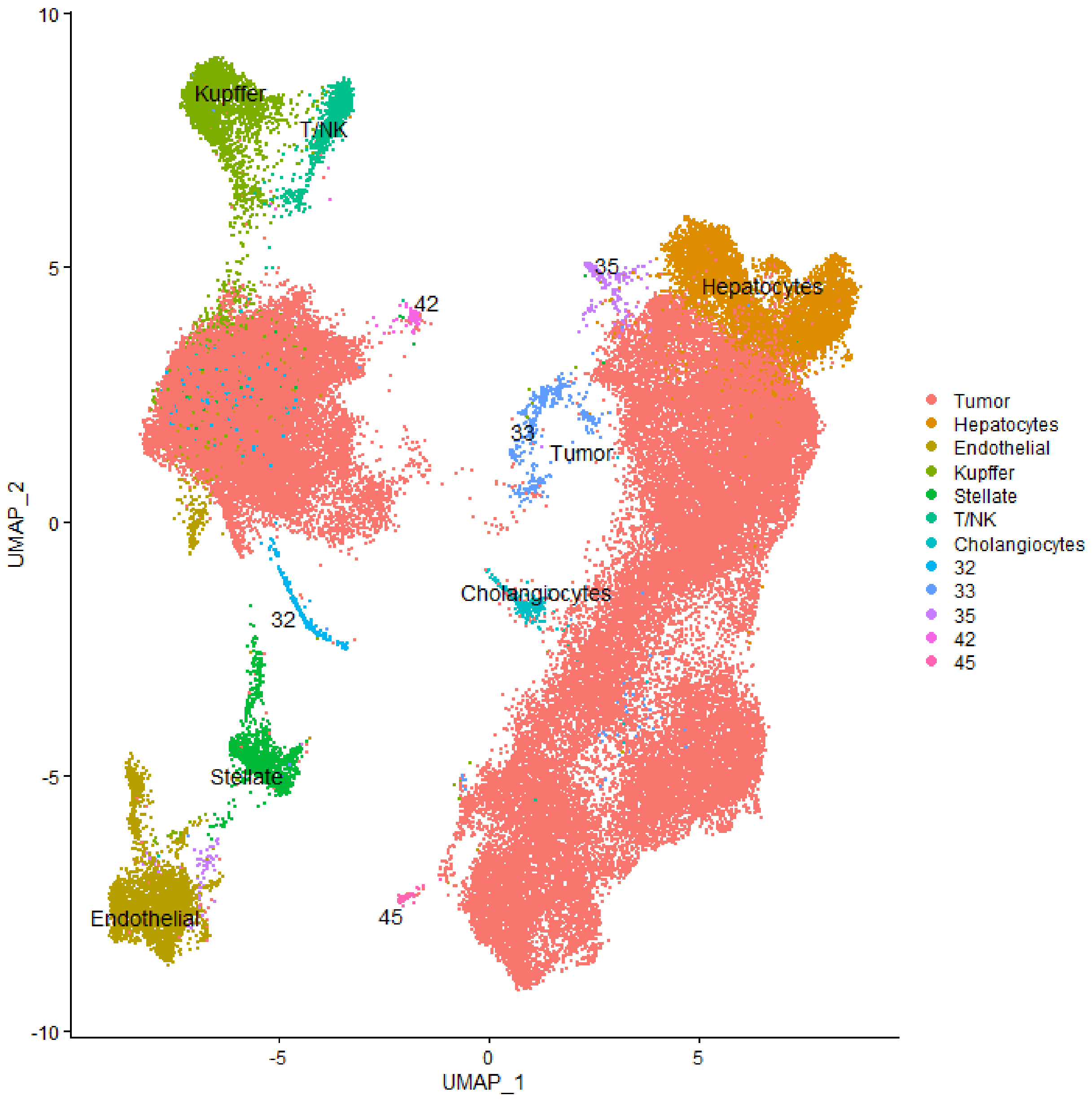


```r
## Marker genes for Tumor cells (Based on paper)
FeaturePlot(merged_samples_norm, reduction = "umap", features = c("GPC3",
    "DLG1", "NKD1", "LRP6", "EPCAM", "HMGA2"), cols = c("#F5F5F5", "red"),
    ncol = 3, sort.cell = TRUE, min.cutoff = "q10", label = TRUE)  # Cluster
```

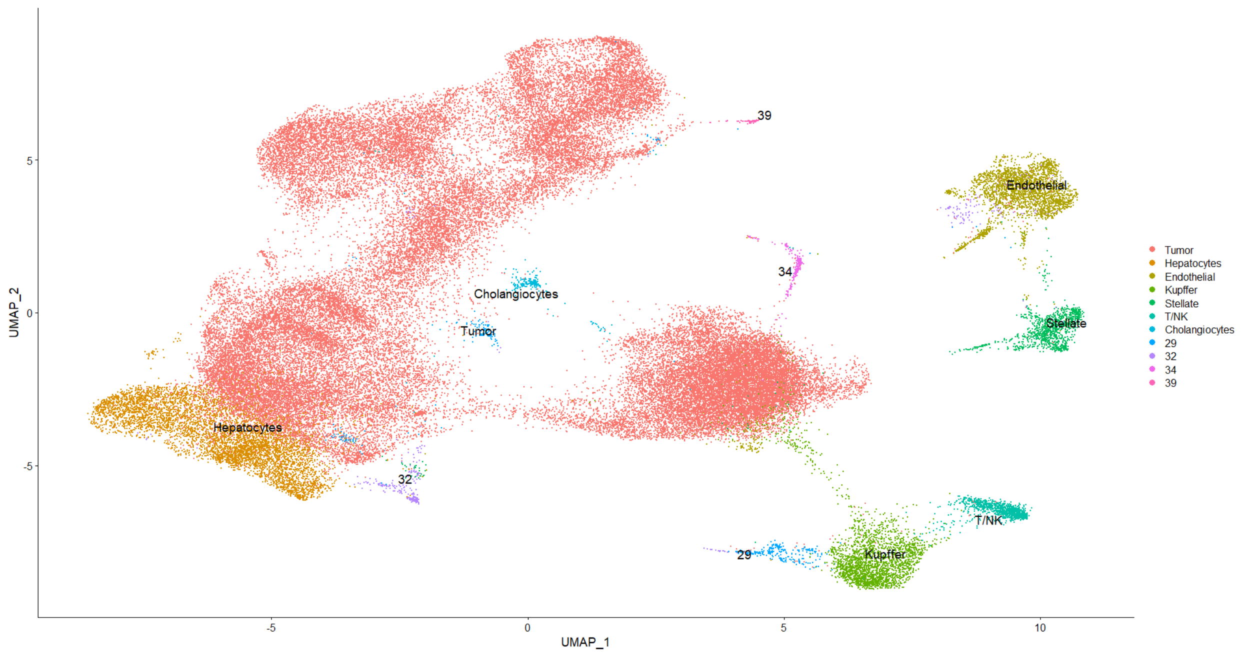


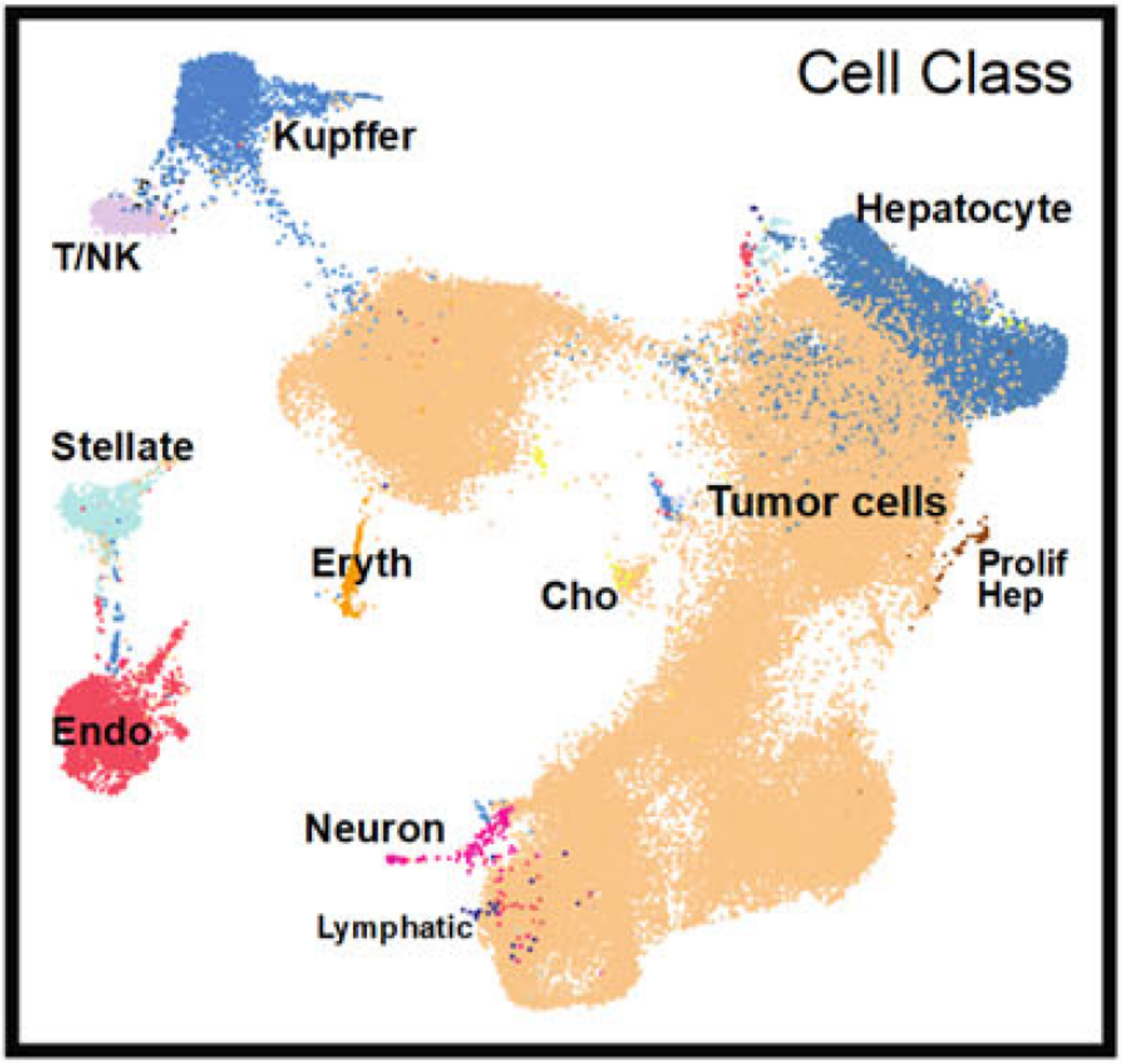
```
## Assign cell classes to clusters
merged_samples_norm <- RenameIdents(object = merged_samples_norm, `0` = "Tumor",
    `1` = "Tumor", `2` = "Tumor", `3` = "Tumor", `4` = "Hepatocytes", `5` = "Tumor",
    `6` = "Tumor", `7` = "Tumor", `8` = "Tumor", `9` = "Tumor", `10` = "Tumor",
    `11` = "Tumor", `12` = "Hepatocytes", `13` = "Tumor", `14` = "Endothelial",
    `15` = "Kupffer", `16` = "Tumor", `17` = "Tumor", `18` = "Tumor", `19` = "Stellate",
    `20` = "Tumor", `21` = "Tumor", `22` = "T/NK", `23` = "Kupffer", `24` = "Tumor",
    `25` = "Tumor", `26` = "Tumor", `27` = "Endothelial", `28` = "Tumor",
    `29` = "Tumor", `30` = "Endothelial", `31` = "Tumor", `34` = "Tumor",
    `36` = "Tumor", `37` = "Cholangiocytes", `38` = "Tumor", `39` = "Tumor",
    `40` = "Hepatocytes", `41` = "Tumor", `43` = "Tumor", `44` = "Endothelial")


## Plot clusters with cell classes
```

```
DimPlot(merged_samples_norm, reduction = "umap", label = TRUE, label.size = 5,
    repel = TRUE)
```



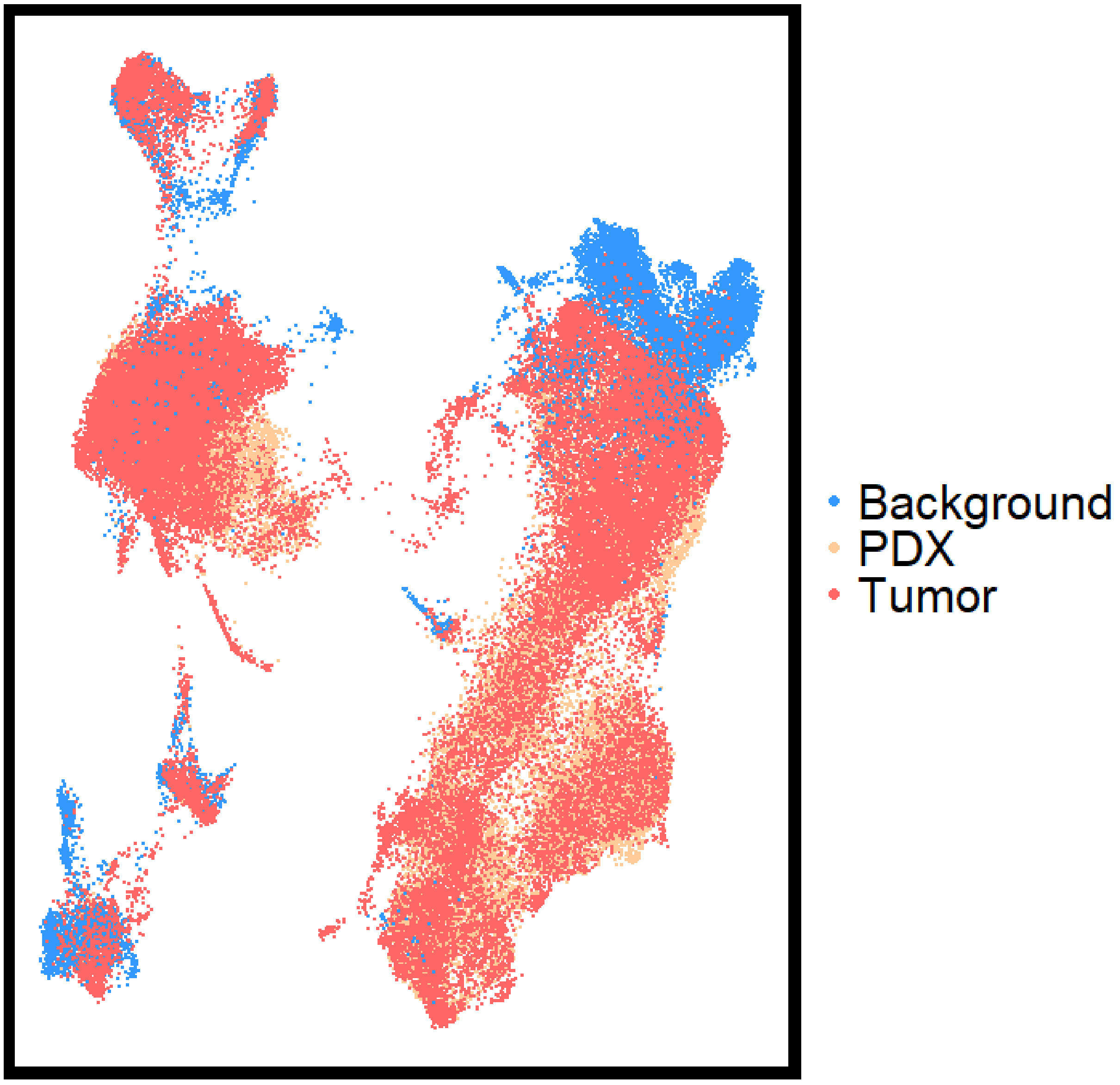
This cluster annotation according to cell type was also performed on the authors' filtered counts data and is plotted below. This is very similar to the corresponding plot included in the paper, which can also be found below. The only difference is, they are 180 degrees out of phase from each other.


39
Endothelial
34
Cholangiocytes
Tumor
Stellate
Hepatocytes
32
T/NK
29
Kupffer
UMAP_2
UMAP_1
5
0
-5
-5
0
5
10
Tumor
Hepatocytes
Endothelial
Kupffer
Stellate
T/NK
Cholangiocytes
29
32
34
39



The following plots demonstrate that we are able to closely reproduce the plots that were included in the paper, as shown below.

```
## Plot clusters according to sample group
DimPlot(merged_samples_norm, reduction = "umap", group.by = "sample_group",
    cols = c("#3399FF", "#FFCC99", "#FF6666")) + theme(legend.position = "right",
    legend.text = element_text(family = "Helvetica", size = 30), panel.background = element_blank(),
    axis.line = element_blank(), axis.title = element_blank(), axis.ticks = element_blank(),
    axis.text = element_blank(), panel.border = element_rect(colour = "black",
        fill = NA, size = 5)) + ggtitle("")
```



```r
## Plot marker genes specific to certain cell classes
DefaultAssay(merged_samples_norm) = "RNA"
FeaturePlot(merged_samples_norm, features = c("GPC3", "CYP3A4", "COL6A3",
    "CD163", "FLT1", "PTPRC"), cols = c("#F5F5F5", "red"), ncol = 3)
```

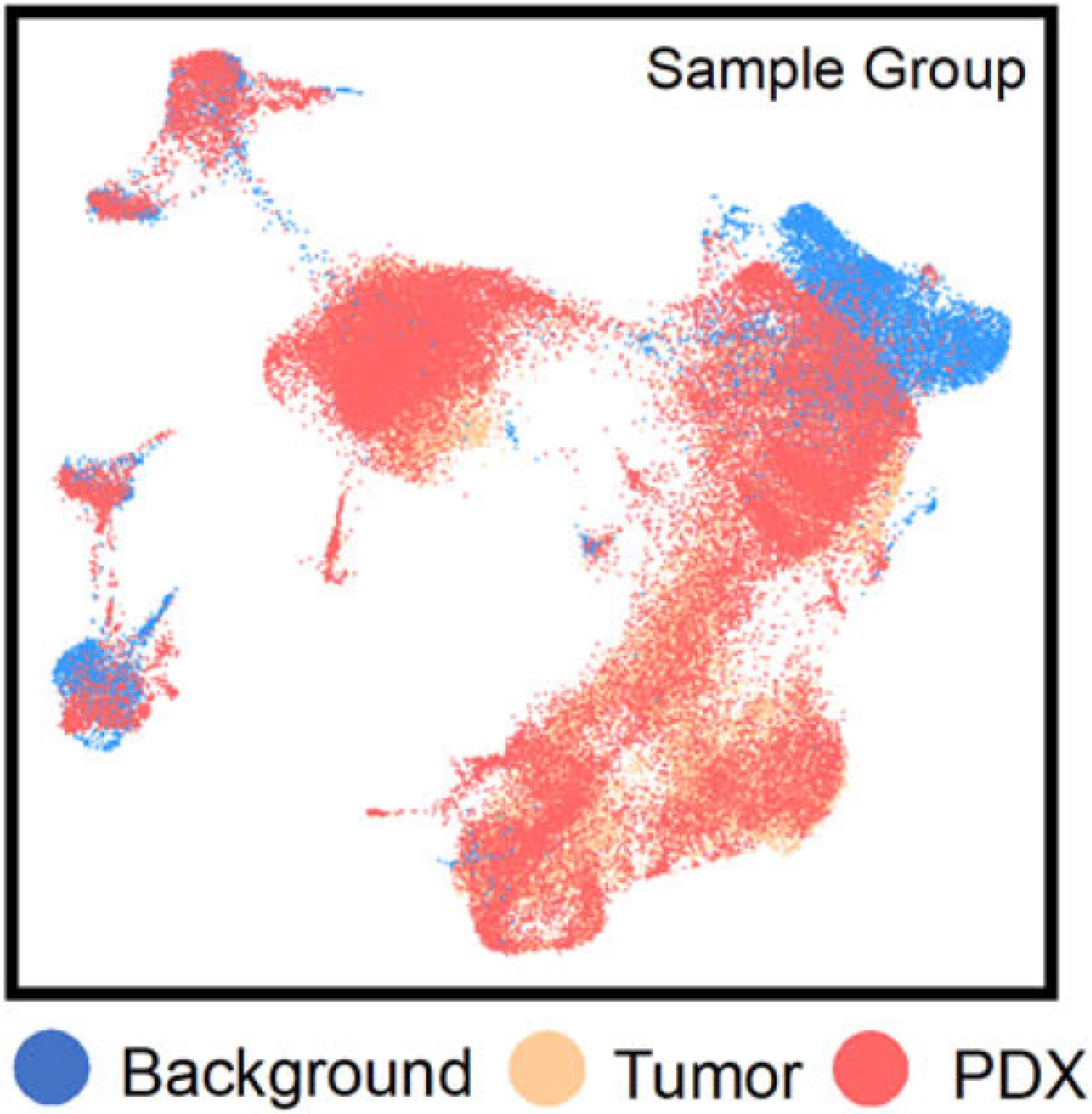
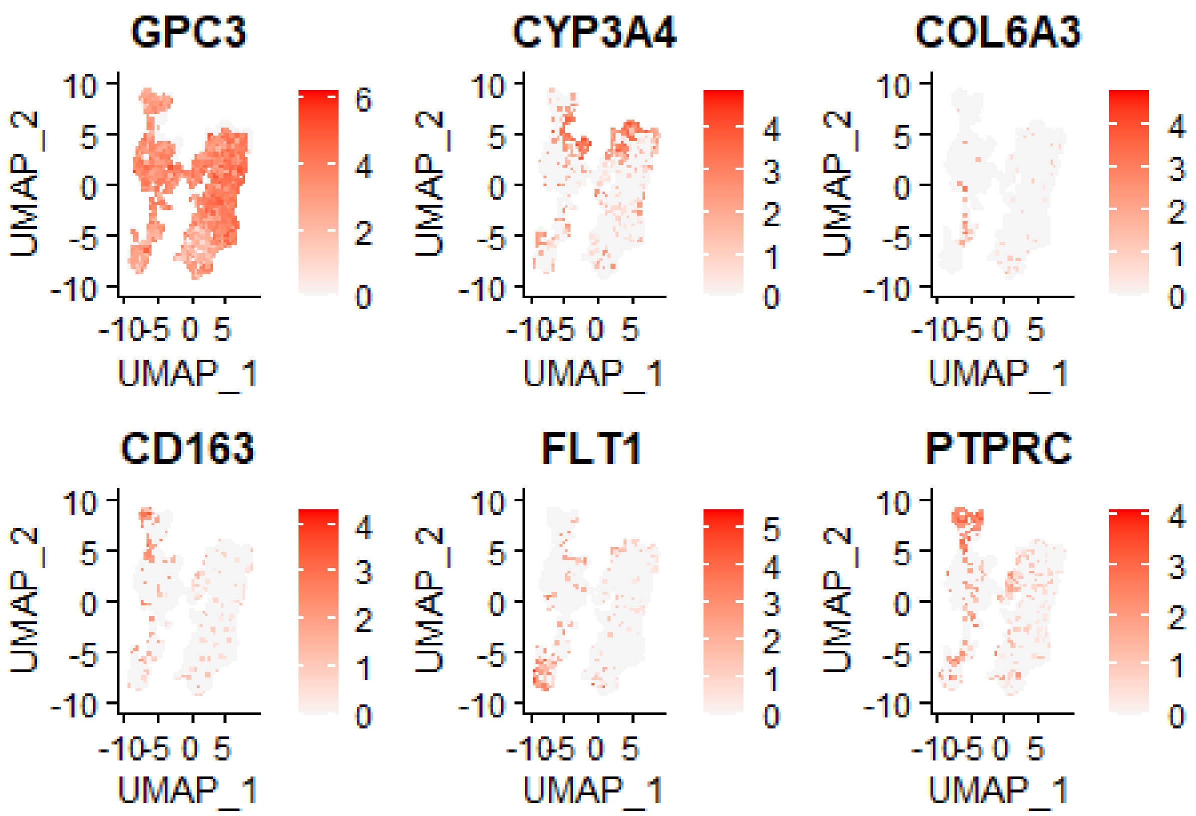


Figure 10: Cluster annotations according to sample group as published in paper (Bondoc et al., 2021)



Conclusion

In this project we assessed the reproducibility of single cell RNA sequencing data for hepatoblastoma. Data was processed from two background liver samples, three tumour samples, and two patient derived xenograft samples. Overall, the reads were of high quality and the gene expression clustering analyses were comparable to the original work by Bondoc et al. (2021). Some differences in our approach compared to the original paper include using a newer reference genome with ~3,000 additional genes that can be mapped and keeping intronic reads when executing Cell Ranger `count`. The count matrices have an additional ~3,000 features in our analysis, a 9% increase from the original paper. This could result in additional reads being aligned to the refernce genome. Further, the inclusion of intronic reads increases the sensitivity of the assessment because if there is no UMI associated with the exon, the gene can still be detected if it is present as an intronic read. This is likely why our approach results in higher gene counts than Bondoc et al. (2021). Despite our inclusion of additional mapped genes, our clustering analysis of cell types, sample groups, and cell population marker genes were very similar to the original work. This provides support for both the validity of our analysis and the validity of the original paper. We are confident that with enough time and reseources this entire paper could be reproduced.

References

Aizarani, N., Saviano, A., Mailly, L., Durand, S., Herman, J. S., Pessaux, P., … & Grün, D. (2019). A human liver cell atlas reveals heterogeneity and epithelial progenitors. Nature, 572(7768), 199-204.

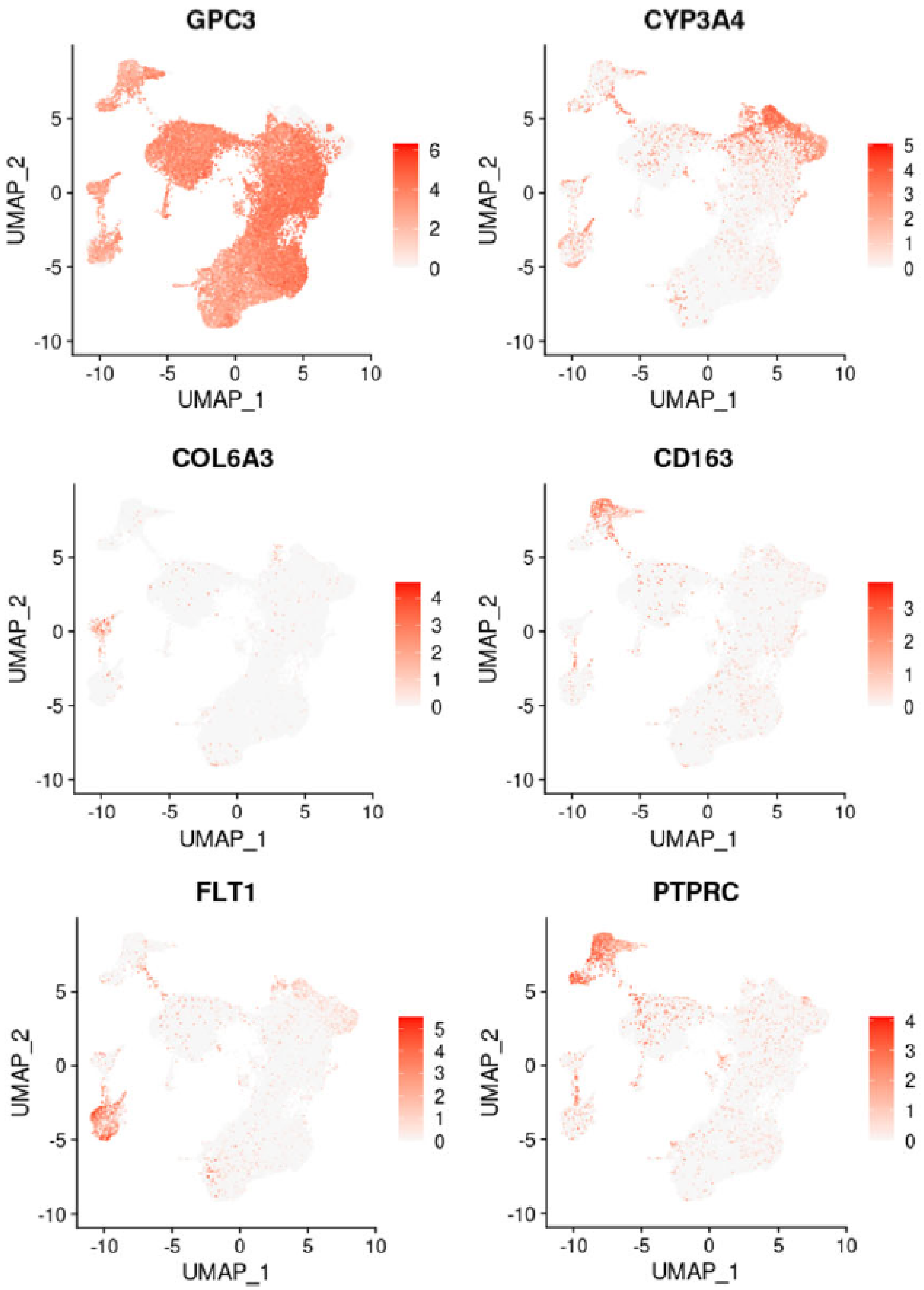


Figure 11: Expression of gene markers on clusters as published in paper (Bondoc et al., 2021)

Andrews, T. S., Kiselev, V. Y., McCarthy, D., & Hemberg, M. (2021). Tutorial: guidelines for the computational analysis of single-cell RNA sequencing data. Nature protocols, 16(1), 1-9.

Bondoc, A., Glaser, K., Jin, K., Lake, C., Cairo, S., Geller, J., … & Aronow, B. (2021). Identification of distinct tumor cell populations and key genetic mechanisms through single cell sequencing in hepatoblastoma. Communications biology, 4(1), 1-14.

Luecken, M. D., & Theis, F. J. (2019). Current best practices in single-cell RNA-seq analysis: a tutorial. Molecular systems biology, 15(6), e8746.

Lun, A. T. L., Riesenfeld, S., Andrews, T., Dao, T. P., Gomes, T., Marioni, J. C., & participants in the 1st Human Cell Atlas Jamboree. (2019). EmptyDrops: Distinguishing cells from empty droplets in droplet-based single-cell RNA sequencing data. Genome Biology, 20(1), 63.