Project-1(BRC)

Aastha Guragain

2025-01-24

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

```
#Setting the working directory
setwd("~/Desktop/seurat-learning")

#Loading the necessary Libraries
library(harmony)
```

```
## Loading required package: Rcpp
```

```
library(ggplot2)
library(Seurat)
```

```
## Loading required package: SeuratObject
```

```
## Loading required package: sp
```

```
## 'SeuratObject' was built under R 4.4.0 but the current version is
## 4.4.2; it is recomended that you reinstall 'SeuratObject' as the ABI
## for R may have changed
```

```
## 'SeuratObject' was built with package 'Matrix' 1.7.0 but the current
## version is 1.7.2; it is recomended that you reinstall 'SeuratObject' as
## the ABI for 'Matrix' may have changed
```

```
##
## Attaching package: 'SeuratObject'
```

```
## The following objects are masked from 'package:base':
##
##     intersect, t
```

```
library(SeuratObject)
library(patchwork)
```

```
# Reading file
brc_file <- readRDS("breast_carcinoma.rds")
```

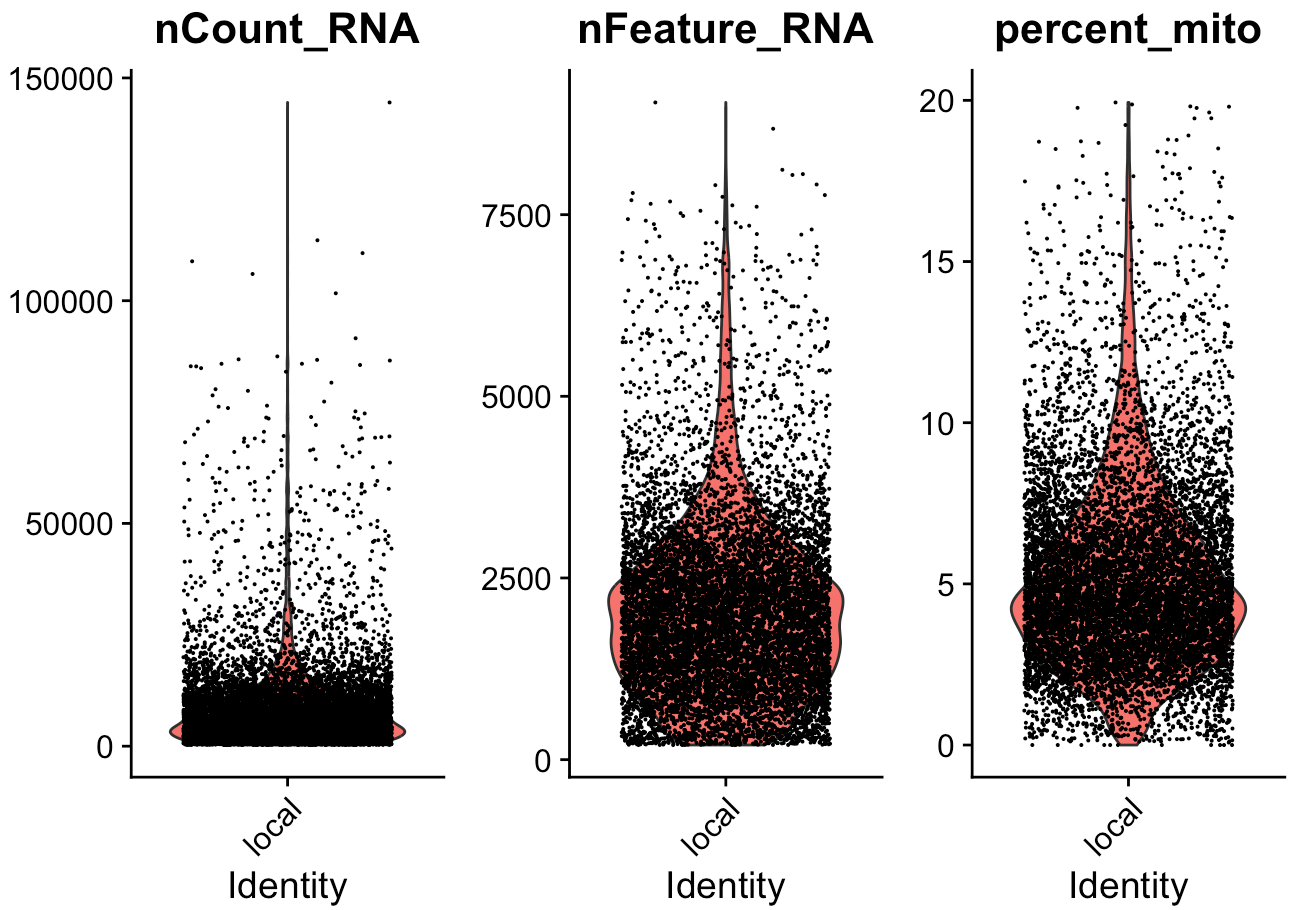
```
#Extracting metadata
metadata.brc <- brc_file@meta.data
```

```
#Calculating Mitochondrial percentage and adding it to metadata if not given
brc_file$mpercent <- PercentageFeatureSet(brc_file, pattern = "^MT")
metadata.brc$mpercent <- brc_file$mpercent
View(metadata.brc)
```

```
# If nCount_RNA and n_Feature RNA not given
#Extracting count matrix
count_matrix <- brc_file@assays$RNA@counts

#creating seurat object from count mmatrix
seurat_obj_cnt <- CreateSeuratObject(count_matrix)
new_seu_metadata <- seurat_obj_cnt@meta.data

#setting the variable
n_feature <- new_seu_metadata$nFeature_RNA
n_count <- new_seu_metadata$nCount_RNA
```

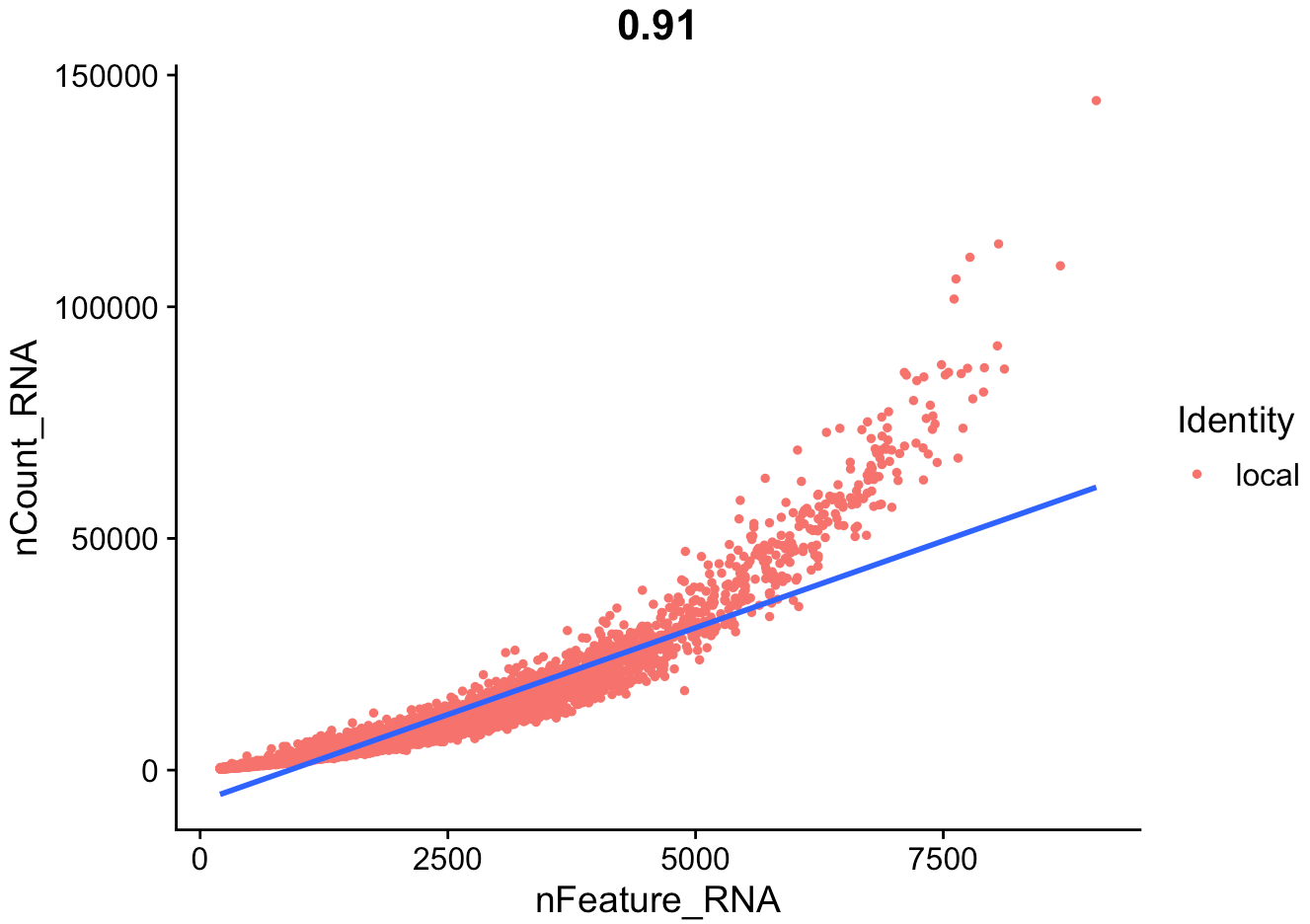
```
#Incorporating ncount and nfeature information into raw suerat file expecting it do not
have these features
brc_file$n_feature <- n_feature
brc_file$n_count <- n_count
brc_file$new_mito_percent <- PercentageFeatureSet(brc_file, pattern = "^MT-")

#Incorporating into metadata
metadata.brc$n_feature <- n_feature
metadata.brc$n_count <- n_count
metadata.brc$new_mito_percent <- PercentageFeatureSet(brc_file, pattern = "^MT-")
```

```
#Upstream Analysis#-----------------------------------------------------------------

# Visualize QC metrics as a violin plot
VlnPlot(brc_file, features = c("nCount_RNA", "nFeature_RNA", "percent_mito"), ncol = 3)
```



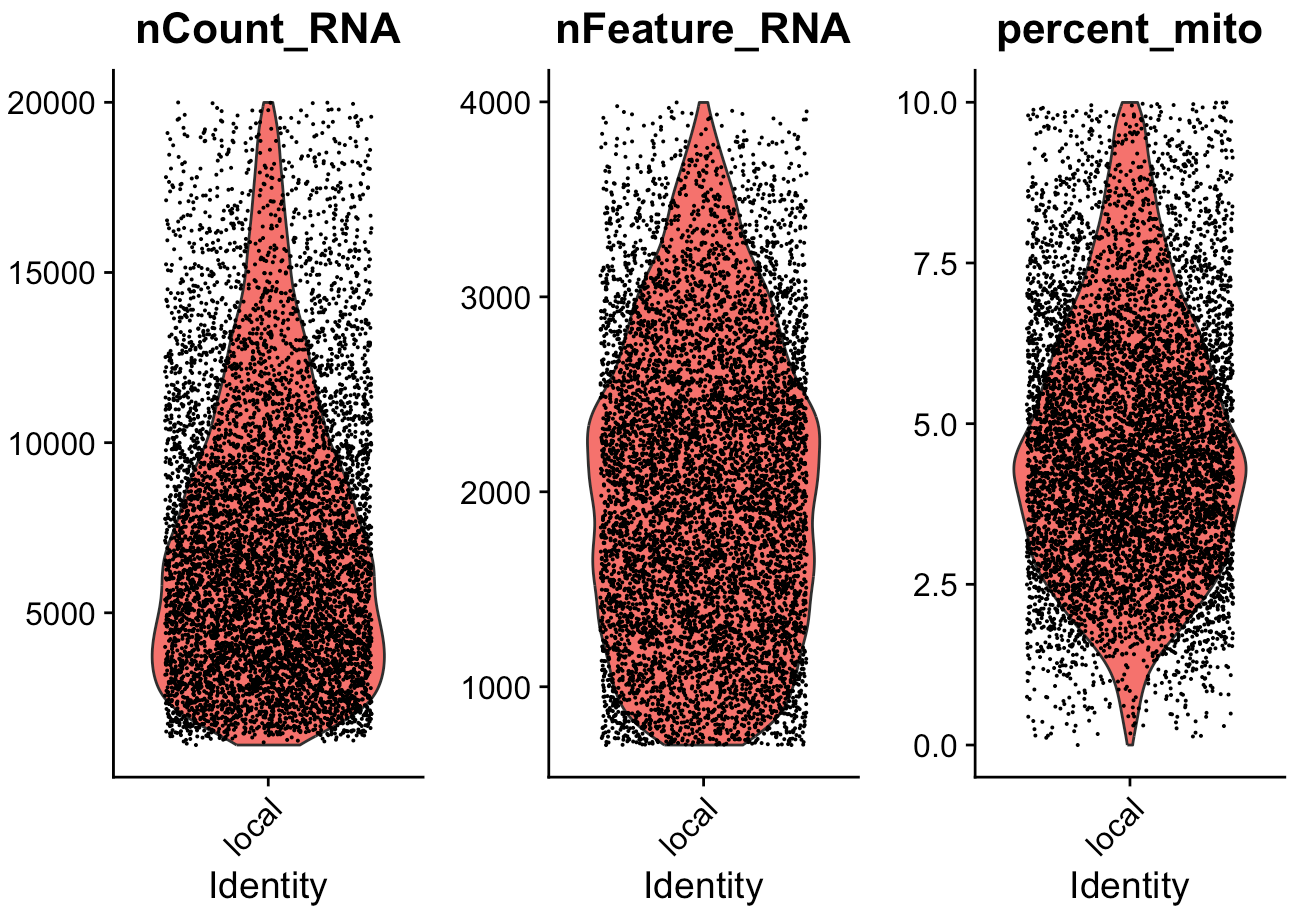

```
FeatureScatter(brc_file, feature1 = "nFeature_RNA", feature2 = "nCount_RNA") + geom_smoo
th(method = "lm")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



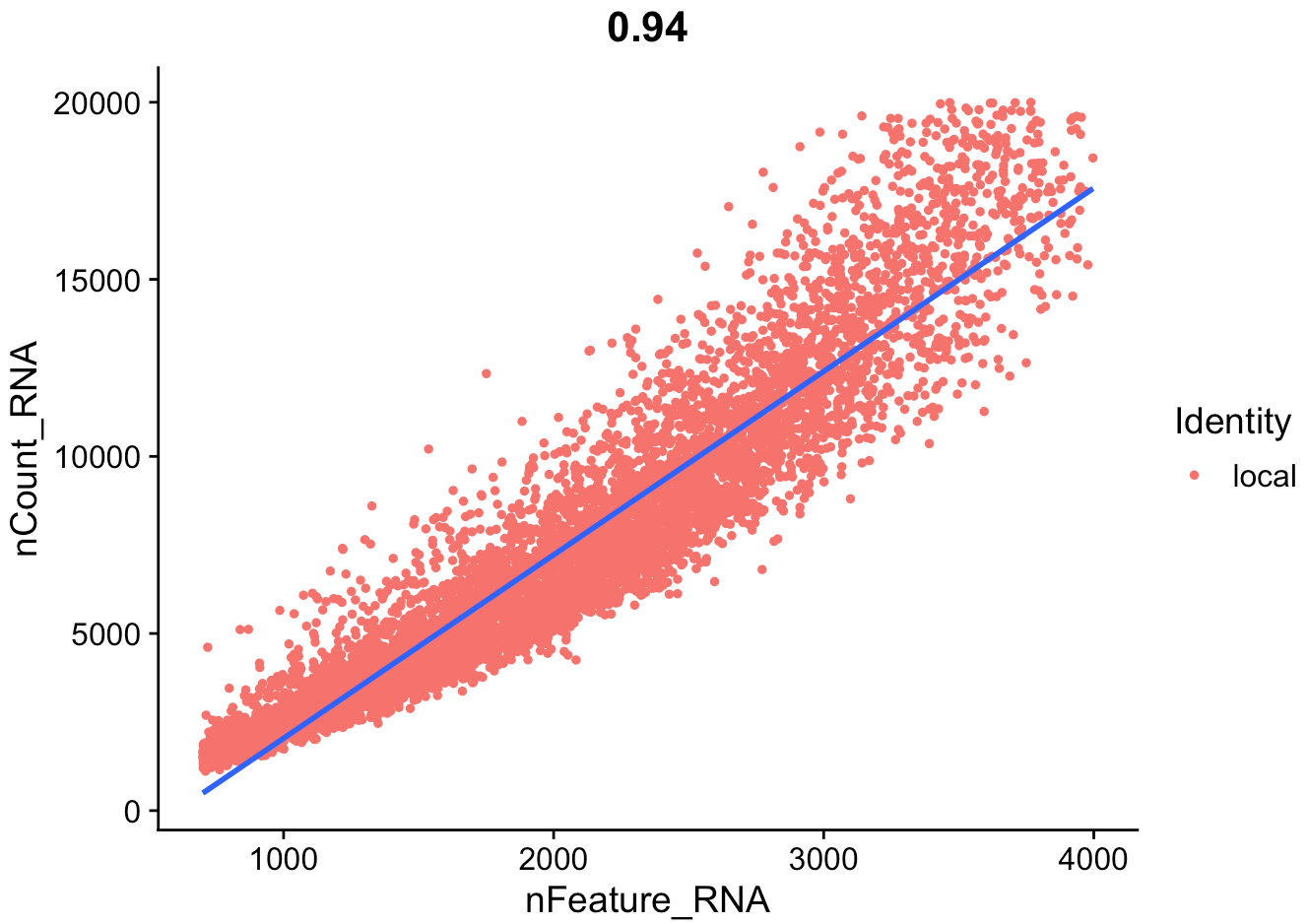

```
subset_brc <- subset(brc_file, subset = nFeature_RNA <4000 & nFeature_RNA > 700 & nCount
_RNA < 20000 & nCount_RNA >500 & percent_mito <10)
View(subset_brc@meta.data)
```

```
plot1 = VlnPlot(subset_brc, features = c("nCount_RNA", "nFeature_RNA", "percent_mito"),
ncol = 3)

plot2 = FeatureScatter(subset_brc, feature1 = "nFeature_RNA", feature2 = "nCount_RNA") +
geom_smooth(method = "lm")
plot1
```




```
plot2
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
#Normalization
subset_brc <- NormalizeData(subset_brc)
```

```
#Finding variable features
subset_brc <- FindVariableFeatures(subset_brc, selection.method = "vst", nfeatures = 200
0)
```

```
#Calculating the top10 variable genes
top10 <- head(VariableFeatures(subset_brc), 10)
top10
```

```
##  [1] "ENSG00000107317" "ENSG00000170323" "ENSG00000118785" "ENSG00000125144"
##  [5] "ENSG00000275385" "ENSG00000102970" "ENSG00000205358" "ENSG00000100453"
##  [9] "ENSG00000133048" "ENSG00000160307"
```

```
# Identify variable features
subset_brc <- FindVariableFeatures(subset_brc)
```

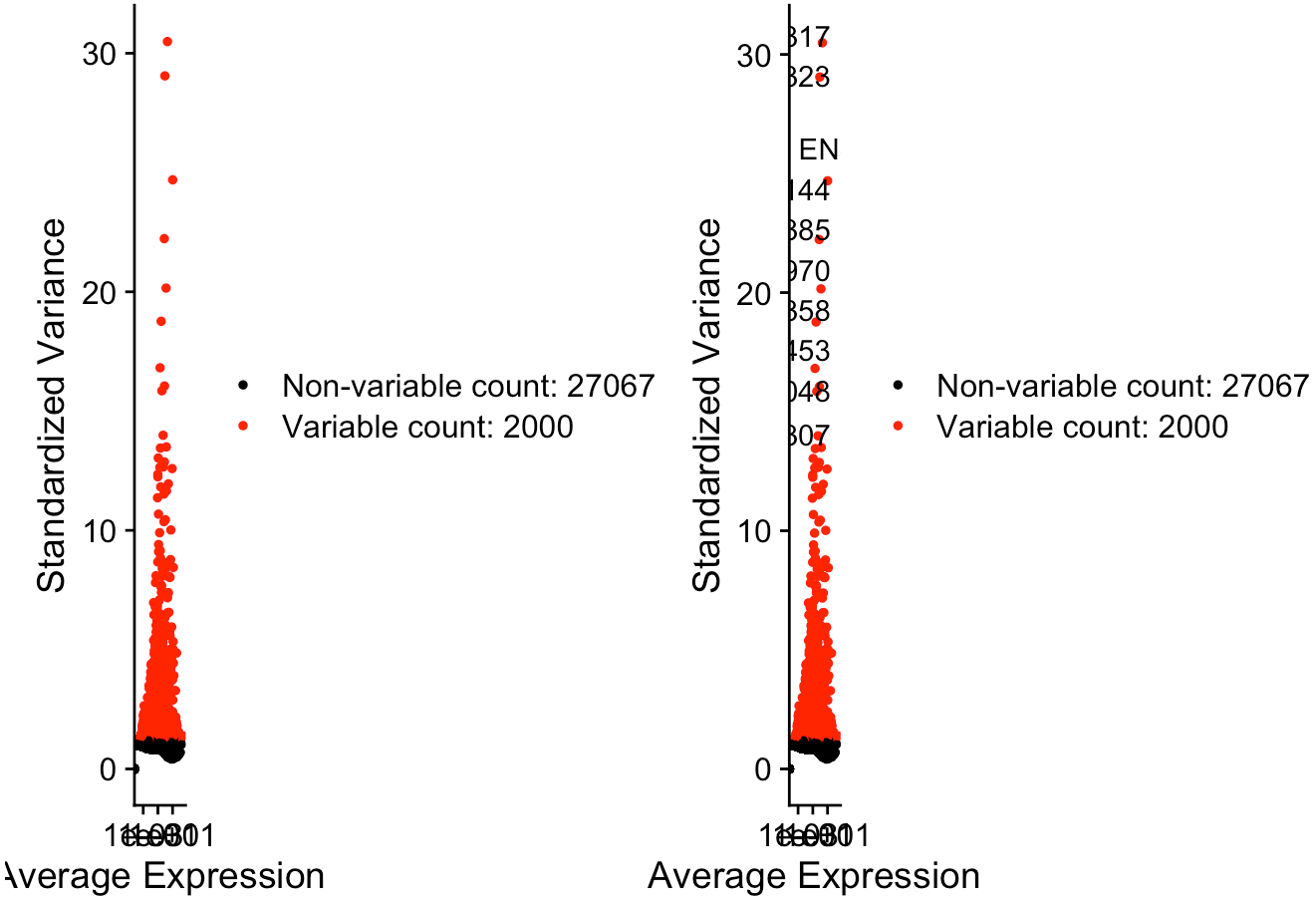
```
# Get top 10 variable features
top10 <- head(VariableFeatures(subset_brc), 10)
```

```
# Plot variable features without label and with label
plot1 <- VariableFeaturePlot(subset_brc)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
```

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

```
# Display the plot
plot <- plot1 + plot2
plot
```

```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
## log-10 transformation introduced infinite values.
```




```
#Scaling the data
all.genes <- rownames(subset_brc)

View(all.genes)

subset_brc <- ScaleData(subset_brc, features = all.genes)
```

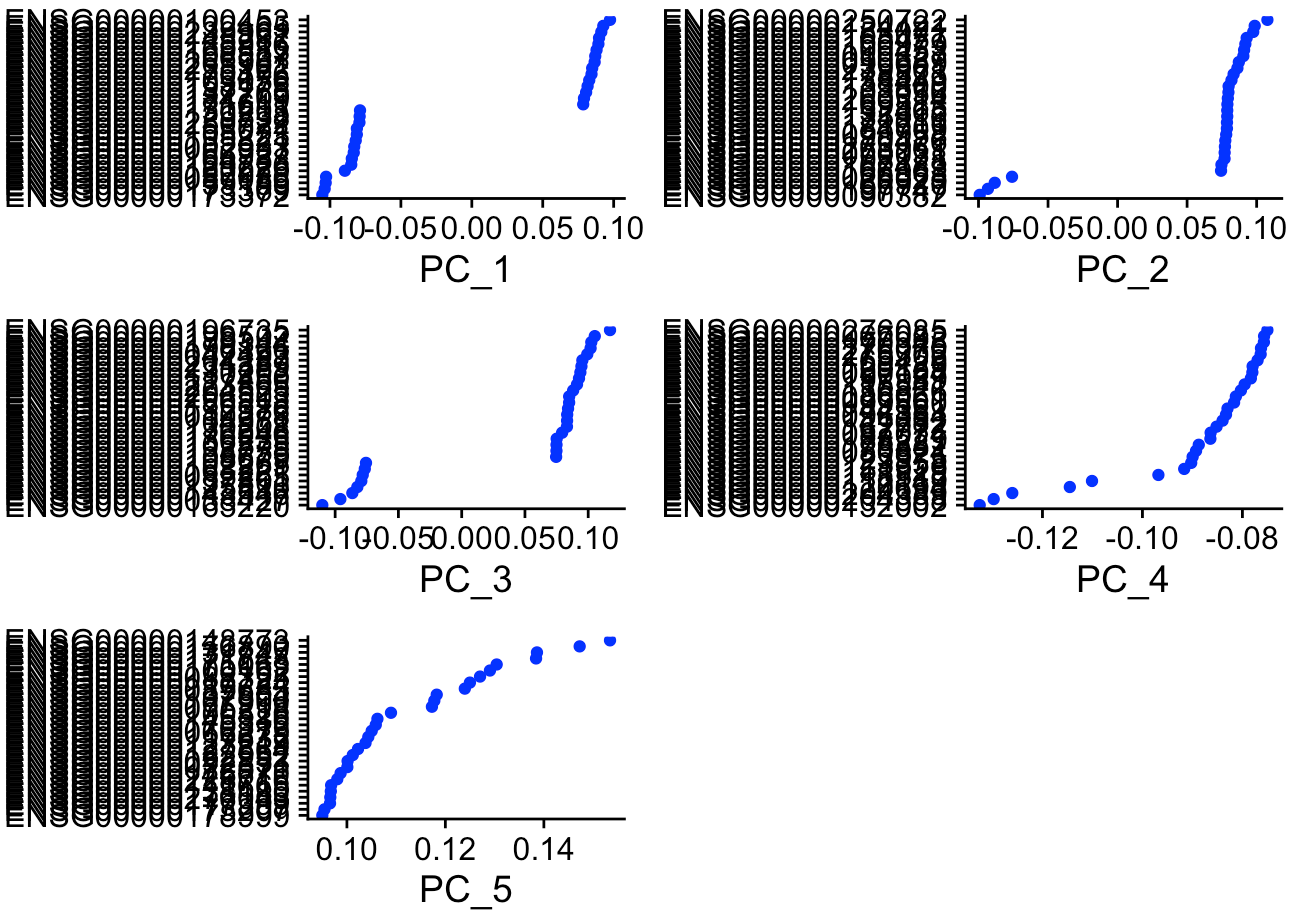
```
## Centering and scaling data matrix
```

```
#Perform linear dimensional reduction (PCA)
subset_brc <- RunPCA(subset_brc, features = VariableFeatures(object = subset_brc))
```

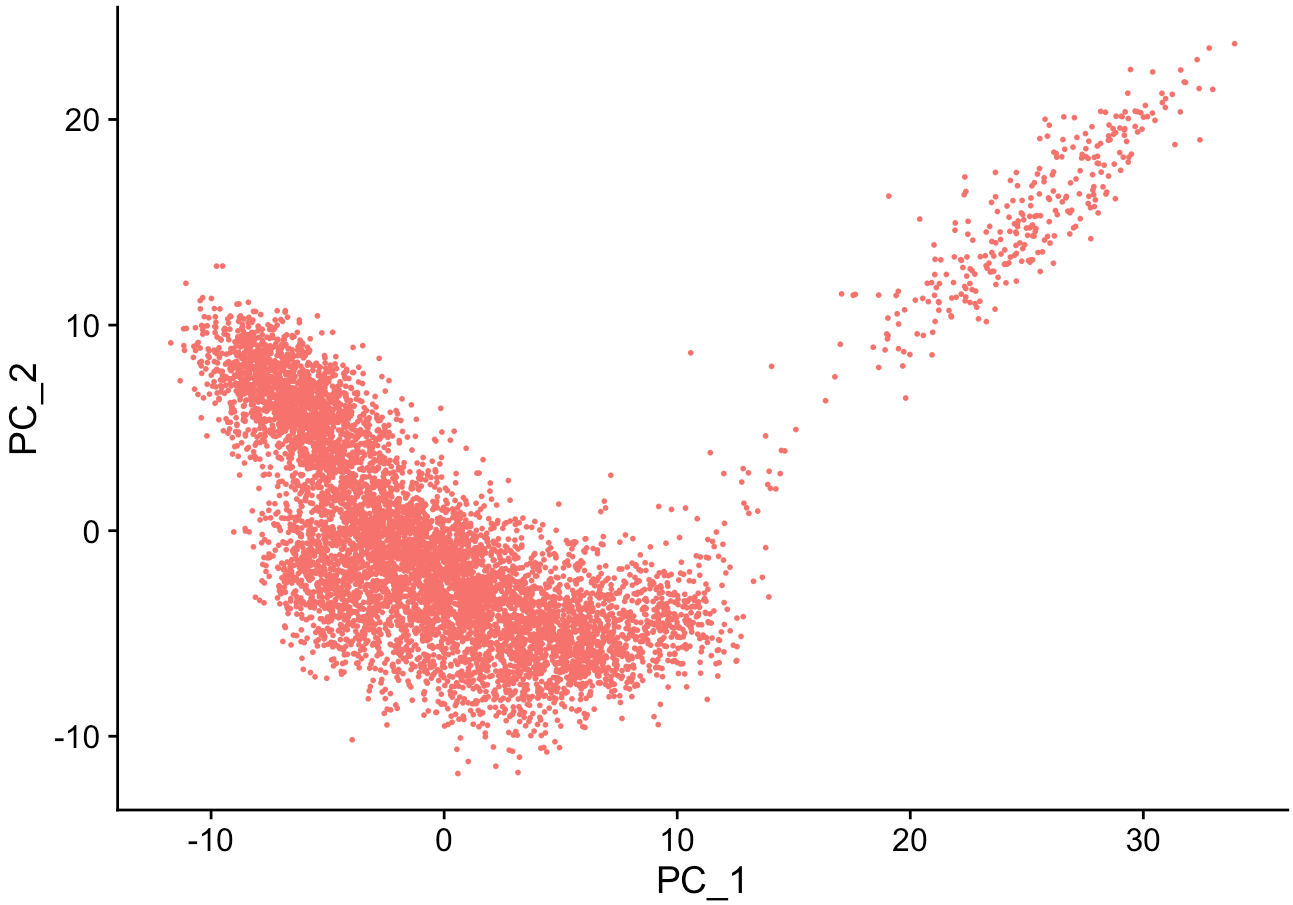
```
## PC_ 1
## Positive:  ENSG00000100453, ENSG00000132465, ENSG00000239961, ENSG00000145287, ENSG00
000135916, ENSG00000169583, ENSG00000106537, ENSG00000263961, ENSG00000235162, ENSG00000
170476
##     ENSG00000103056, ENSG00000198178, ENSG00000137265, ENSG00000184709, ENSG000001716
11, ENSG00000099958, ENSG00000227507, ENSG00000070031, ENSG00000186810, ENSG00000167483
##     ENSG00000185291, ENSG00000119866, ENSG00000105426, ENSG00000166963, ENSG000001636
87, ENSG00000076604, ENSG00000125869, ENSG00000204323, ENSG00000269404, ENSG00000161970
## Negative:  ENSG00000173372, ENSG00000173369, ENSG00000159189, ENSG00000087086, ENSG00
000100979, ENSG00000129226, ENSG00000164733, ENSG00000166927, ENSG00000002933, ENSG00000
135821
##     ENSG00000105223, ENSG00000153071, ENSG00000250722, ENSG00000129538, ENSG000001302
03, ENSG00000197746, ENSG00000106565, ENSG00000170458, ENSG00000010327, ENSG00000117984
##     ENSG00000165457, ENSG00000135404, ENSG00000143878, ENSG00000177575, ENSG000001100
79, ENSG00000165029, ENSG00000155659, ENSG00000130208, ENSG00000177606, ENSG00000260314
## PC_ 2
## Positive:  ENSG00000250722, ENSG00000124491, ENSG00000153071, ENSG00000165457, ENSG00
000100979, ENSG00000100453, ENSG00000010327, ENSG00000090659, ENSG00000239961, ENSG00000
178573
##     ENSG00000138449, ENSG00000133800, ENSG00000103056, ENSG00000260314, ENSG000001695
83, ENSG00000132465, ENSG00000135916, ENSG00000171611, ENSG00000184709, ENSG00000099958
##     ENSG00000170476, ENSG00000263961, ENSG00000070031, ENSG00000198178, ENSG000001674
83, ENSG00000166963, ENSG00000235162, ENSG00000196628, ENSG00000198467, ENSG00000100600
## Negative:  ENSG00000090382, ENSG00000197747, ENSG00000166920, ENSG00000026025, ENSG00
000100079, ENSG00000085265, ENSG00000197956, ENSG00000130592, ENSG00000169442, ENSG00000
143546
##     ENSG00000140105, ENSG00000135046, ENSG00000102265, ENSG00000140379, ENSG000000009
38, ENSG00000173391, ENSG00000196924, ENSG00000123689, ENSG00000182718, ENSG00000128383
##     ENSG00000115414, ENSG00000165140, ENSG00000204287, ENSG00000104951, ENSG000001808
17, ENSG00000117228, ENSG00000038427, ENSG00000216490, ENSG00000217555, ENSG00000010278
## PC_ 3
## Positive:  ENSG00000196735, ENSG00000198502, ENSG00000179344, ENSG00000196126, ENSG00
000042493, ENSG00000204287, ENSG00000231389, ENSG00000130203, ENSG00000117450, ENSG00000
223865
##     ENSG00000204525, ENSG00000206503, ENSG00000186818, ENSG00000132386, ENSG000000102
78, ENSG00000104951, ENSG00000130208, ENSG00000176046, ENSG00000136235, ENSG00000102575
##     ENSG00000138755, ENSG00000108679, ENSG00000115414, ENSG00000149131, ENSG000001098
61, ENSG00000117984, ENSG00000026751, ENSG00000109971, ENSG00000271503, ENSG00000008517
## Negative:  ENSG00000163220, ENSG00000038427, ENSG00000143546, ENSG00000125810, ENSG00
000137801, ENSG00000085265, ENSG00000163221, ENSG00000102265, ENSG00000245532, ENSG00000
113070
##     ENSG00000124882, ENSG00000059804, ENSG00000186407, ENSG00000018280, ENSG000001425
41, ENSG00000178726, ENSG00000196154, ENSG00000124491, ENSG00000112715, ENSG00000008988
##     ENSG00000172216, ENSG00000126759, ENSG00000182774, ENSG00000166441, ENSG000001474
54, ENSG00000125538, ENSG00000059728, ENSG00000183019, ENSG00000133800, ENSG00000224397
## PC_ 4
## Positive:  ENSG00000160932, ENSG00000119917, ENSG00000163220, ENSG00000168899, ENSG00
000117228, ENSG00000115415, ENSG00000243466, ENSG00000157601, ENSG00000126709, ENSG00000
121858
##     ENSG00000272398, ENSG00000149131, ENSG00000089127, ENSG00000185745, ENSG000001977
46, ENSG00000239951, ENSG00000241351, ENSG00000241755, ENSG00000206503, ENSG00000211659
##     ENSG00000134575, ENSG00000142089, ENSG00000154451, ENSG00000211653, ENSG000000254
```

```
34, ENSG00000109861, ENSG00000004468, ENSG00000068079, ENSG00000104951, ENSG00000165949
## Negative:  ENSG00000132002, ENSG00000204388, ENSG00000204389, ENSG00000120694, ENSG00
000112149, ENSG00000118515, ENSG00000151929, ENSG00000123358, ENSG00000123975, ENSG00000
080824
##     ENSG00000178381, ENSG00000188229, ENSG00000087074, ENSG00000162772, ENSG000001416
82, ENSG00000096384, ENSG00000144381, ENSG00000099860, ENSG00000086061, ENSG00000136826
##     ENSG00000115541, ENSG00000197989, ENSG00000090104, ENSG00000120129, ENSG000001694
29, ENSG00000275302, ENSG00000276070, ENSG00000170345, ENSG00000067082, ENSG00000276085
## PC_ 5
## Positive:  ENSG00000148773, ENSG00000176890, ENSG00000131747, ENSG00000171848, ENSG00
000175063, ENSG00000100162, ENSG00000088325, ENSG00000117724, ENSG00000089685, ENSG00000
137804
##     ENSG00000167900, ENSG00000075218, ENSG00000100526, ENSG00000145386, ENSG000001703
12, ENSG00000066279, ENSG00000117632, ENSG00000123219, ENSG00000127564, ENSG00000122952
##     ENSG00000092853, ENSG00000072571, ENSG00000146670, ENSG00000228716, ENSG000001116
65, ENSG00000138180, ENSG00000276043, ENSG00000117399, ENSG00000173207, ENSG00000178999
## Negative:  ENSG00000100906, ENSG00000291237, ENSG00000118503, ENSG00000144802, ENSG00
000137331, ENSG00000197766, ENSG00000119508, ENSG00000171174, ENSG00000125538, ENSG00000
081041
##     ENSG00000120129, ENSG00000128383, ENSG00000087074, ENSG00000162772, ENSG000001121
49, ENSG00000085265, ENSG00000090339, ENSG00000245532, ENSG00000130844, ENSG00000140379
##     ENSG00000116741, ENSG00000162711, ENSG00000158050, ENSG00000186818, ENSG000001855
07, ENSG00000124762, ENSG00000107372, ENSG00000255112, ENSG00000169508, ENSG00000173451
```

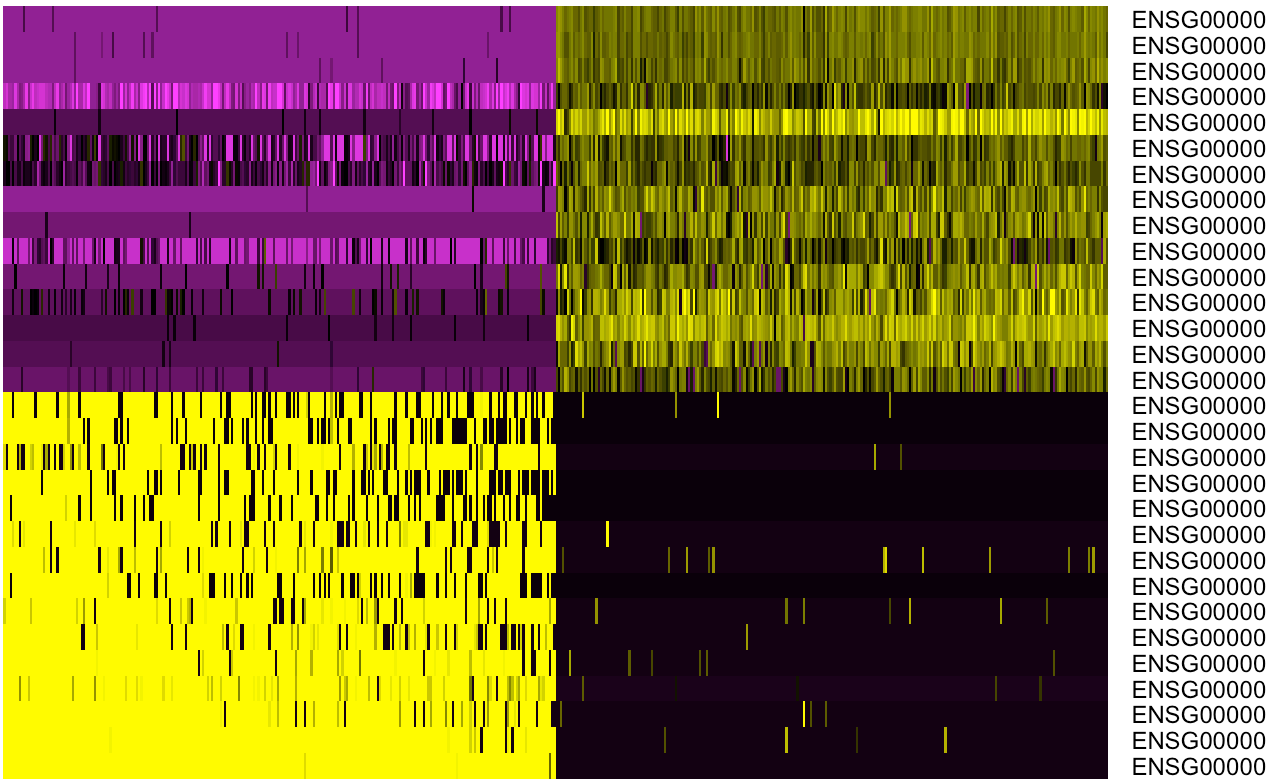
```
VizDimLoadings(subset_brc, dims = 1:5, reduction = "pca")
```



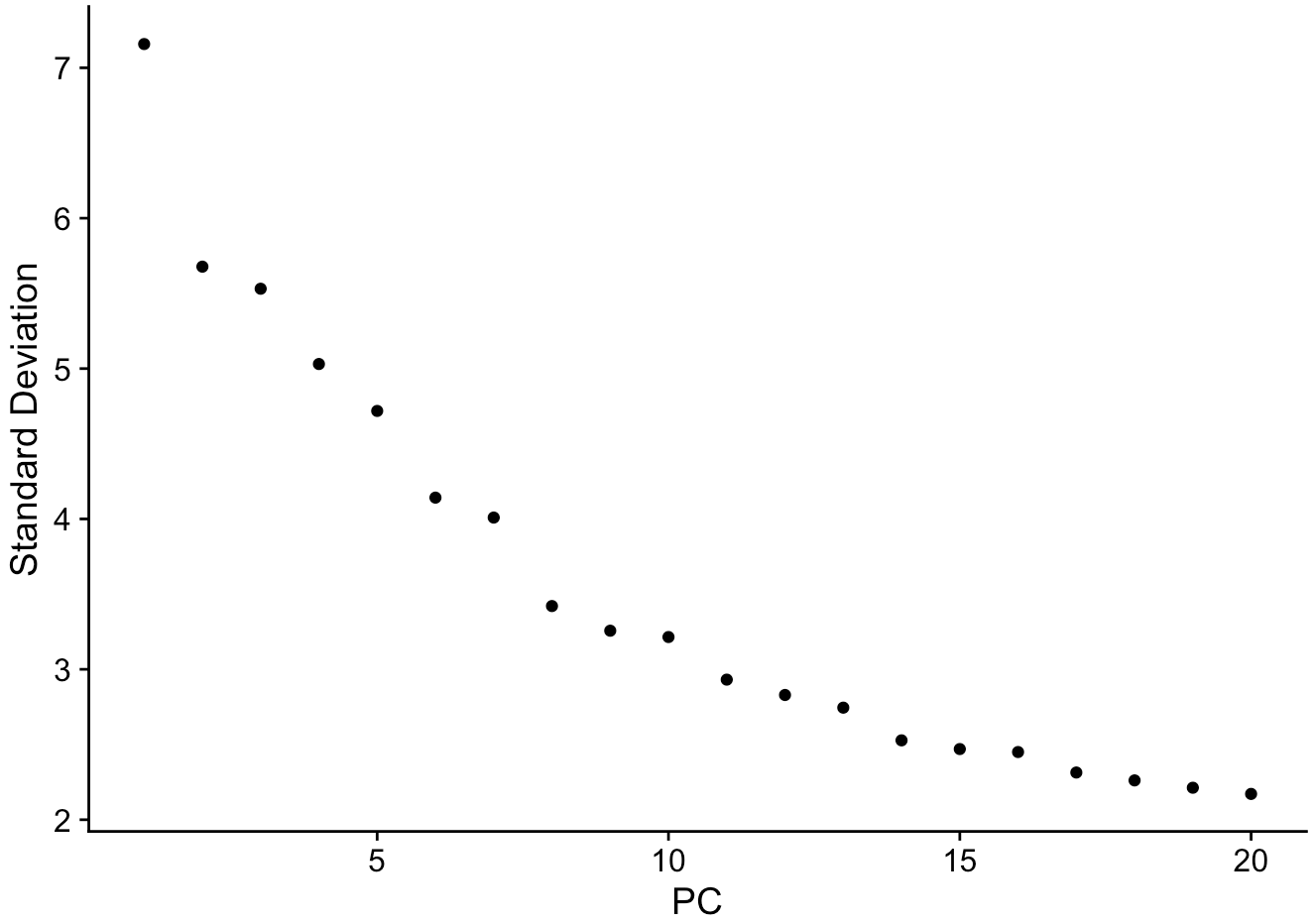
```
DimPlot(subset_brc, reduction = "pca") + NoLegend()
```




```
DimHeatmap(subset_brc, dim = 1, cells =500, balanced = TRUE)
```




```
#Determining Pcs
ElbowPlot(subset_brc)
```




```
subset_brc <- FindNeighbors(subset_brc, dims = 1:15) #dims = dimension,
```

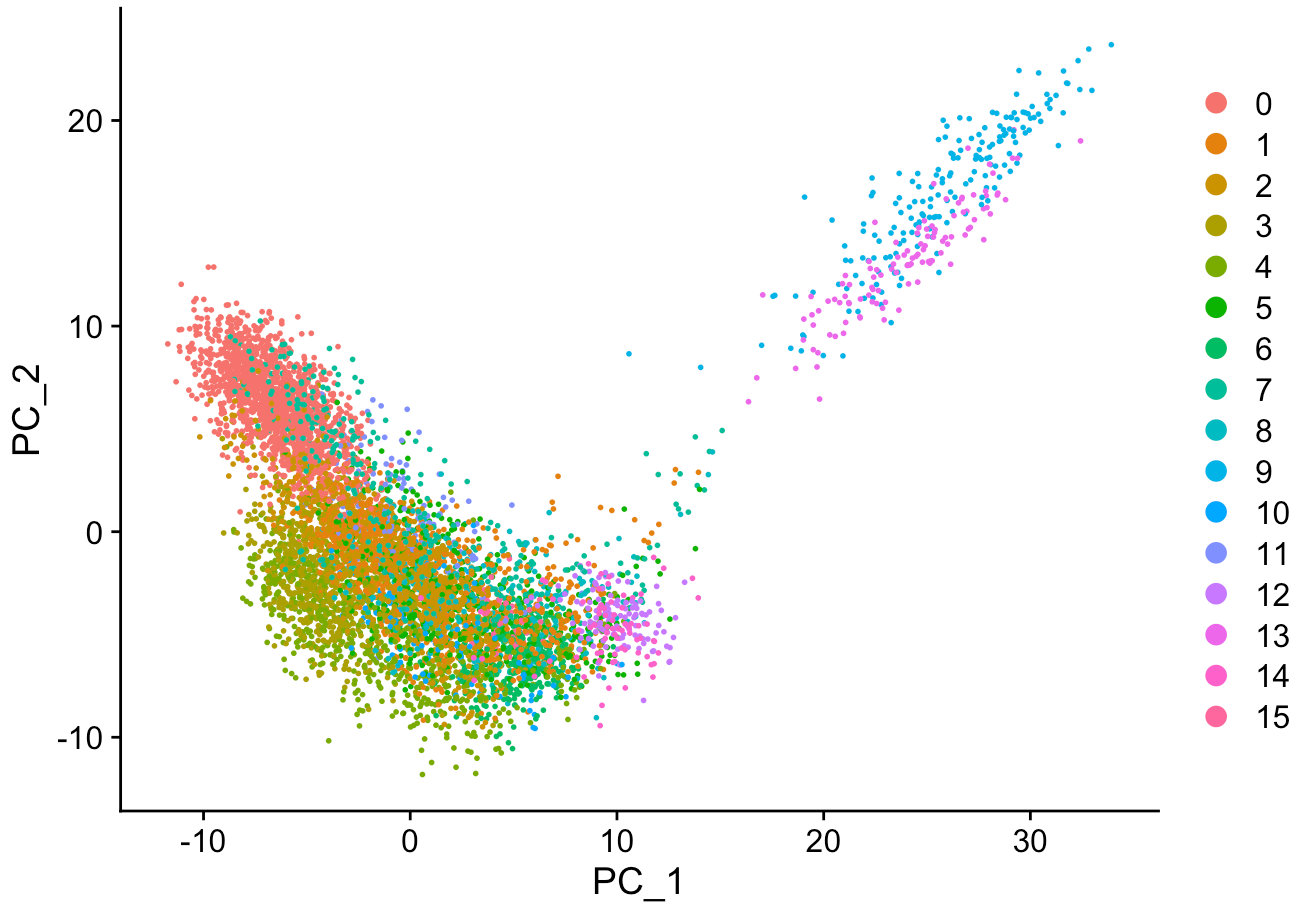
```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
subset_brc <- FindClusters(subset_brc, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 7488
## Number of edges: 245221
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9060
## Number of communities: 16
## Elapsed time: 0 seconds
```

```
DimPlot(subset_brc, reduction = "pca")
```




```
subset_brc <- RunUMAP(subset_brc, dims = 1:15, reduction = "pca")
```

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

```
## 20:52:31 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 20:52:31 Read 7488 rows and found 15 numeric columns
```

```
## 20:52:31 Using Annoy for neighbor search, n_neighbors = 30
```

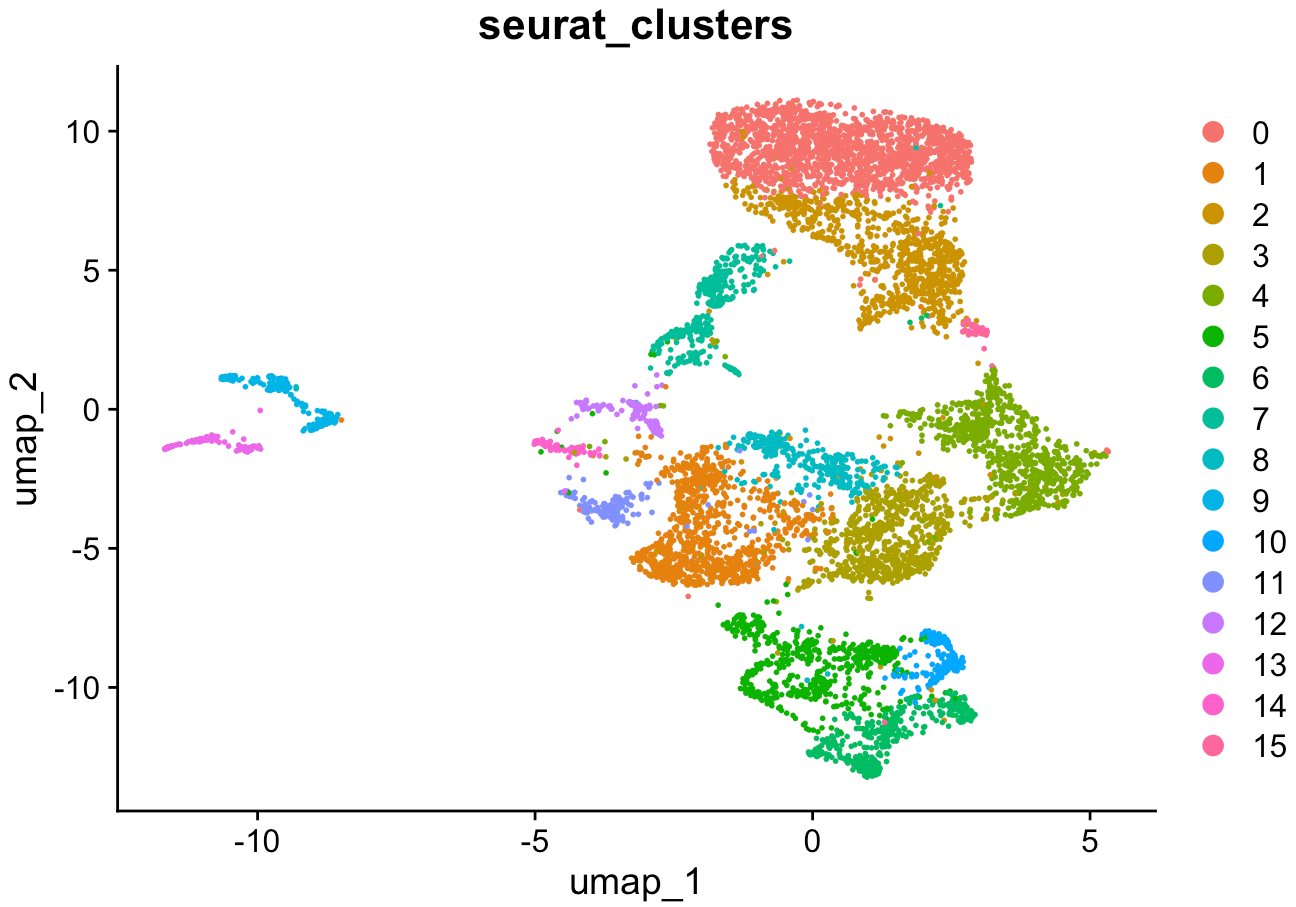
```
## 20:52:31 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

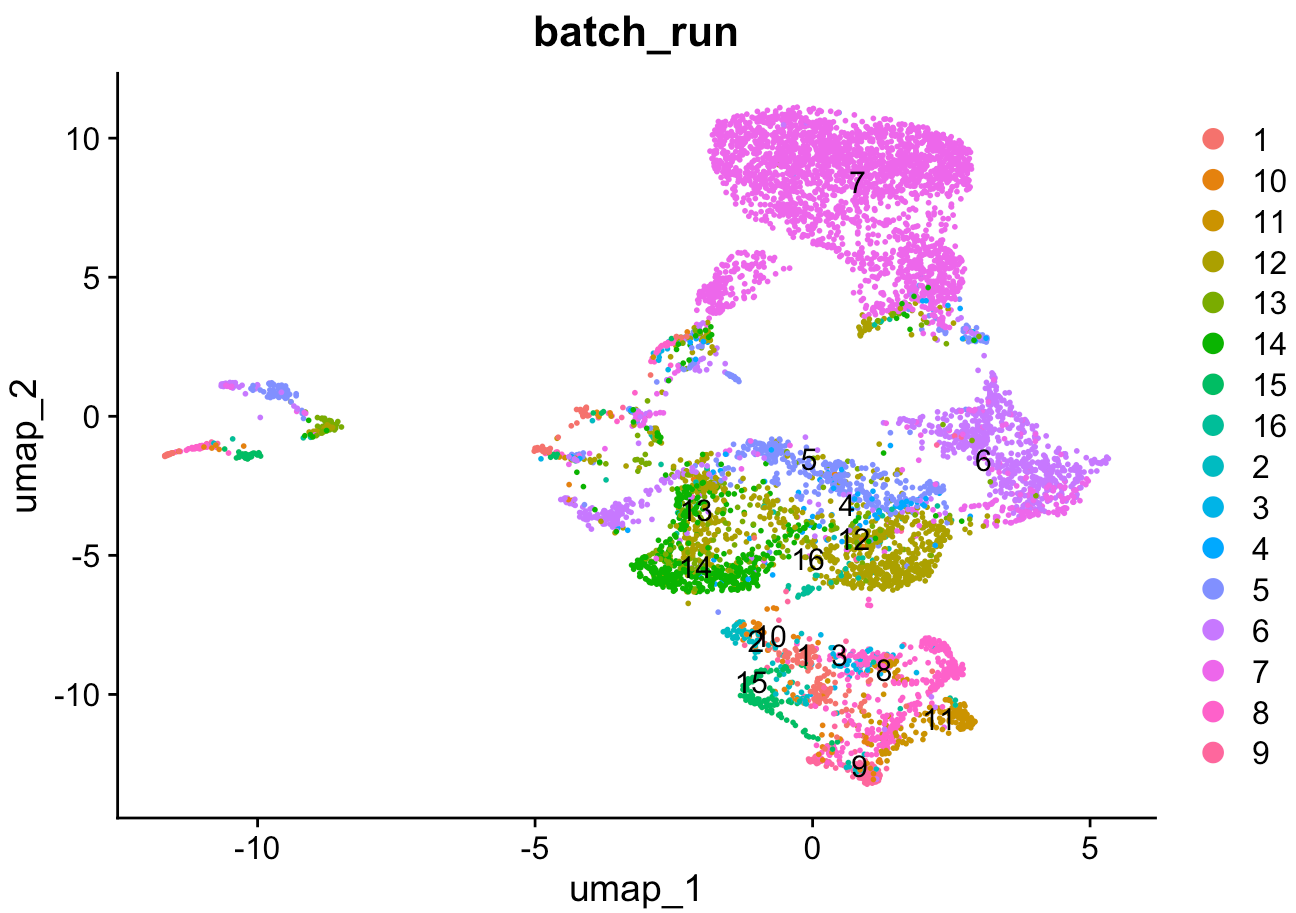
```
## ****************************************************|
## 20:52:31 Writing NN index file to temp file /var/folders/rq/8sjqg4hj3k99cfwf49v9_dlc0
000gn/T//Rtmp1kA5m1/file7e6b2b6a4a7e
## 20:52:31 Searching Annoy index using 1 thread, search_k = 3000
## 20:52:32 Annoy recall = 100%
## 20:52:33 Commencing smooth kNN distance calibration using 1 thread with target n_neig
hbors = 30
## 20:52:33 Initializing from normalized Laplacian + noise (using RSpectra)
## 20:52:33 Commencing optimization for 500 epochs, with 305360 positive edges
## 20:52:39 Optimization finished
```

```
DimPlot(subset_brc, reduction = "umap", group.by = "seurat_clusters")
```

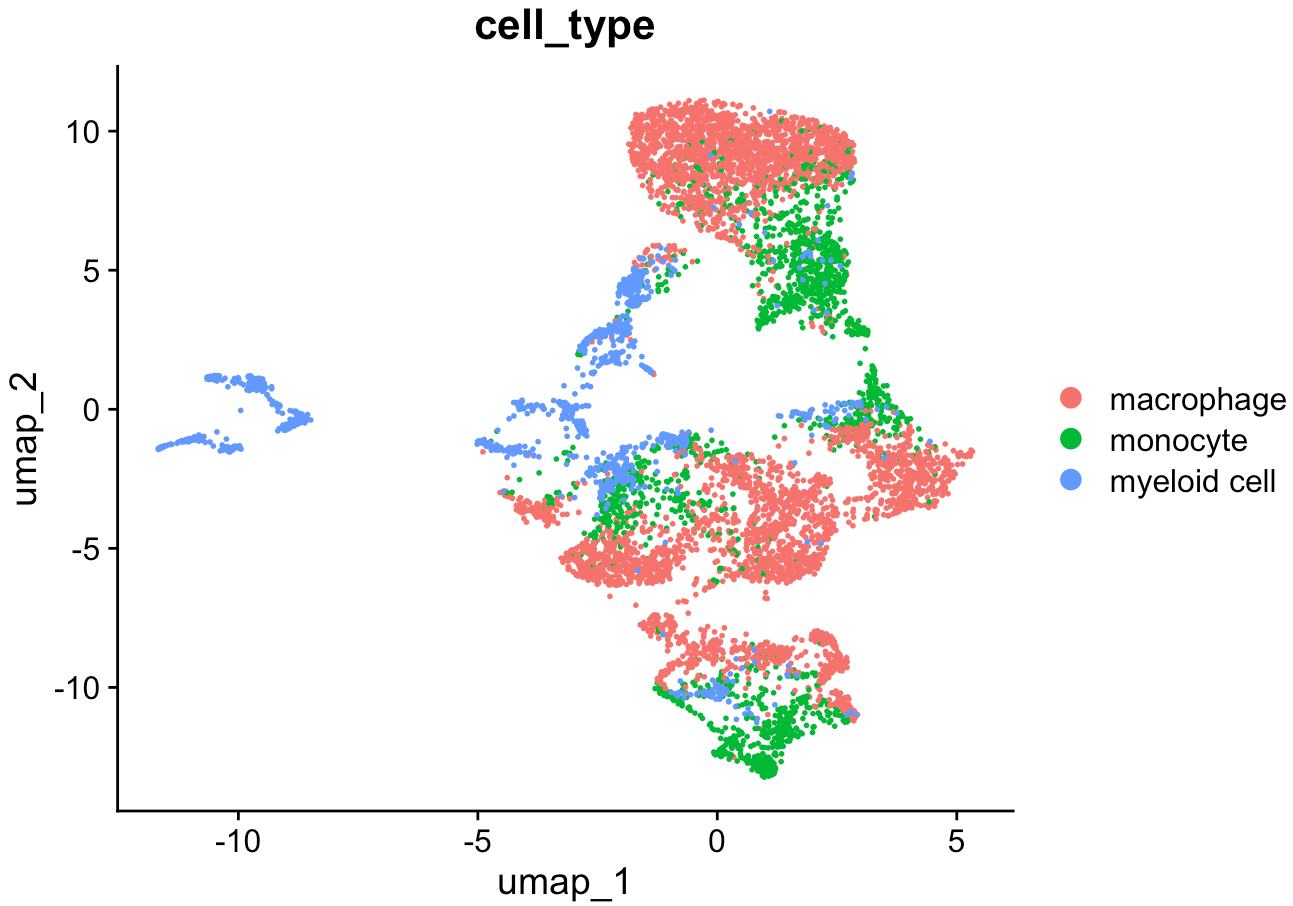



```
disease <- DimPlot(subset_brc, reduction = "umap", group.by = "disease" , label = "TRU
E")
```

```
batch <- DimPlot(subset_brc, reduction = "umap", group.by = "batch_run", label = "TRUE")
batch
```




```
cell_type<- DimPlot(subset_brc, reduction = "umap", group.by = "cell_type")
cell_type
```




```
#Batch effect removal using harmony
subset_brc <- RunHarmony(
  object = subset_brc,
  group.by.vars = "batch_run",
  dims.use = 1:15 # Use the same dimensions you selected in FindNeighbors
)
```

```
## Transposing data matrix
```

```
## Initializing state using k-means centroids initialization
```

```
## Harmony 1/10
```

```
## Harmony 2/10
```

```
## Harmony 3/10
```

```
## Harmony 4/10
```

```
## Harmony 5/10
```

```
## Harmony 6/10
```

```
## Harmony 7/10
```

```
## Harmony converged after 7 iterations
```

```
# Re- running clustering using harmony
subset_brc <- subset_brc %>%
  RunUMAP(reduction = 'harmony',dims = 1:15) %>%
  FindNeighbors(reduction = 'harmony' , dims = 1:15) %>%
  FindClusters(resolution = 0.5)
```

```
## 20:52:43 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 20:52:43 Read 7488 rows and found 15 numeric columns
```

```
## 20:52:43 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 20:52:43 Building Annoy index with metric = cosine, n_trees = 50
```

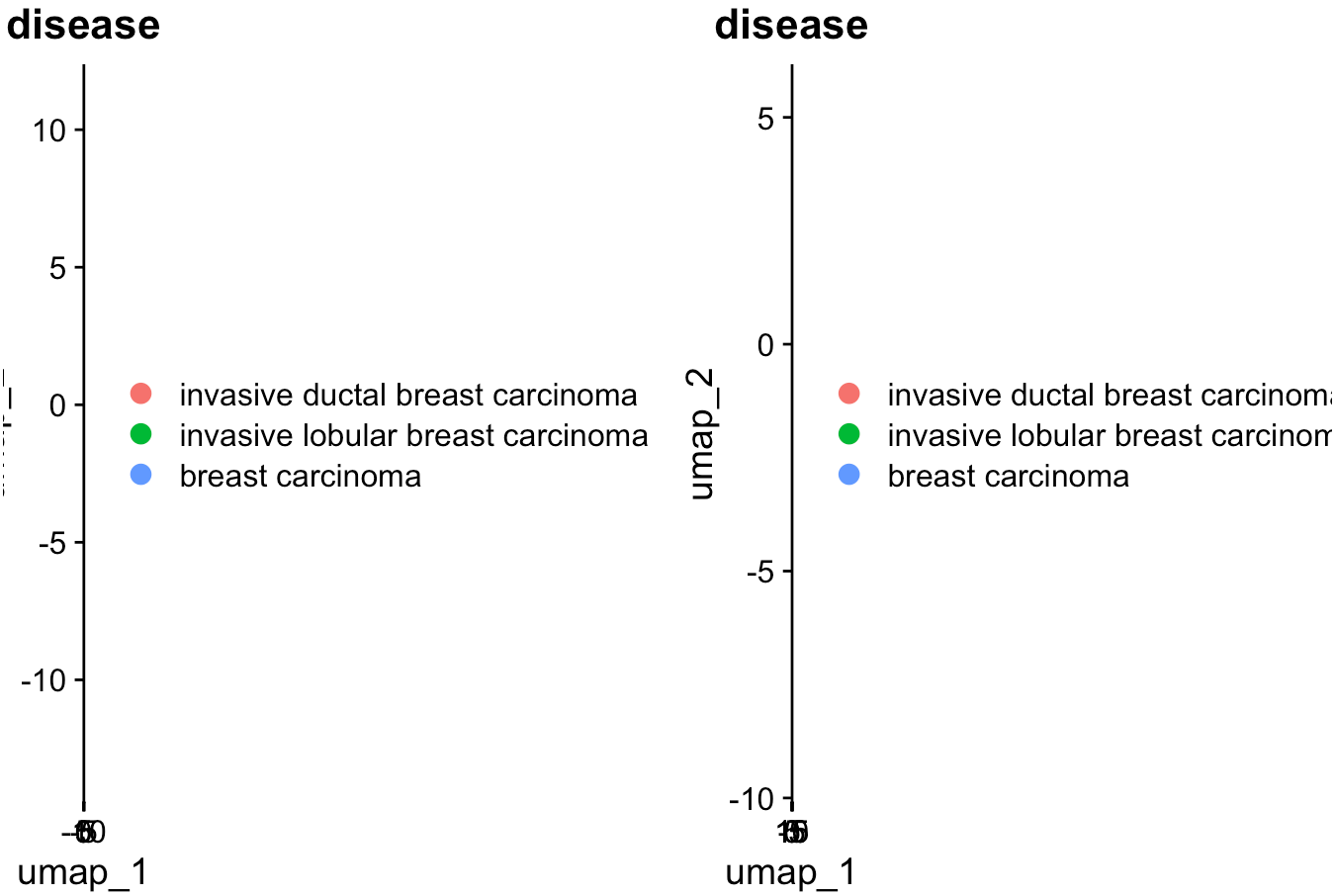
```
## 0%   10   20   30   40   50   60   70   80   90   100%
```

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

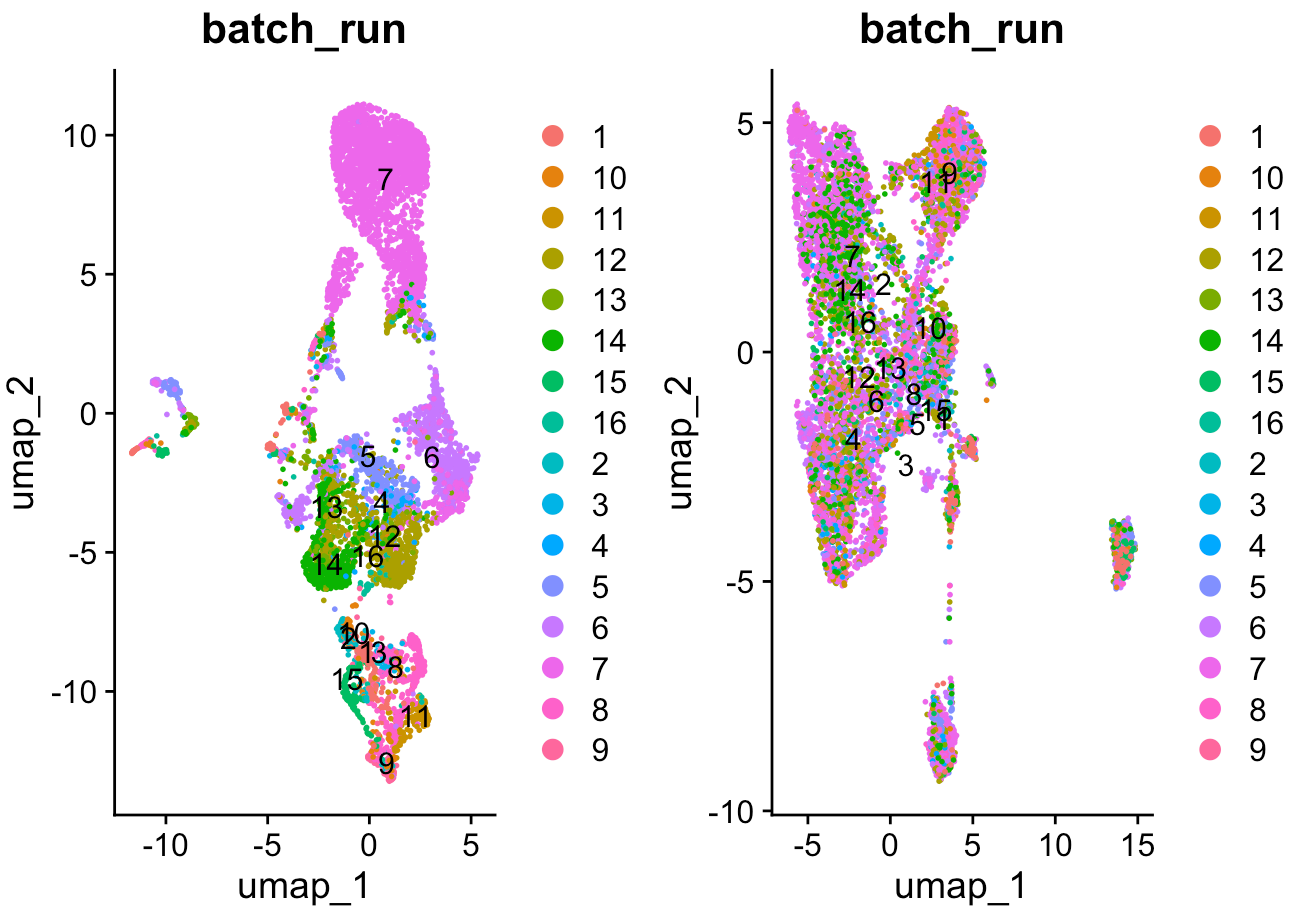
```
## **************************************************|
## 20:52:44 Writing NN index file to temp file /var/folders/rq/8sjqg4hj3k99cfwf49v9_dlc0
000gn/T//Rtmp1kA5m1/file7e6b79686d8
## 20:52:44 Searching Annoy index using 1 thread, search_k = 3000
## 20:52:45 Annoy recall = 100%
## 20:52:45 Commencing smooth kNN distance calibration using 1 thread with target n_neig
hbors = 30
## 20:52:46 Initializing from normalized Laplacian + noise (using RSpectra)
## 20:52:46 Commencing optimization for 500 epochs, with 311638 positive edges
## 20:52:51 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 7488
## Number of edges: 267661
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8731
## Number of communities: 13
## Elapsed time: 0 seconds
```

```
disease_after_harmony <- DimPlot(subset_brc, reduction = "umap", group.by = "disease" ,
label = "TRUE")
batch_after_harmony <- DimPlot(subset_brc, reduction = "umap", group.by = "batch_run", l
abel = "TRUE")
disease | disease_after_harmony
```




```
batch | batch_after_harmony
```


batch_run
umap_2
umap_1
10
5
0
-5
-10
-10
-5
0
5
7
5
6
13
4
12
14
16
10
2
13
8
15
11
9
1
10
11
12
13
14
15
16
2
3
4
5
6
7
8
9
batch_run
umap_2
umap_1
5
0
-5
-10
-5
0
5
10
15
9
11
7
14
2
16
10
12
13
6
8
15
5
4
3
1
10
11
12
13
14
15
16
2
3
4
5
6
7
8
9