Data visualization methods in Seurat

Compiled: 2023-10-31

Source: `vignettes/visualization_vignette.Rmd` (https://github.com/satijalab/seurat/blob/HEAD/vignettes/visualization_vignette.Rmd)

We'll demonstrate visualization techniques in Seurat using our previously computed Seurat object from the 2,700 PBMC tutorial. You can download this dataset from SeuratData (https://github.com/satijalab/seurat-data)

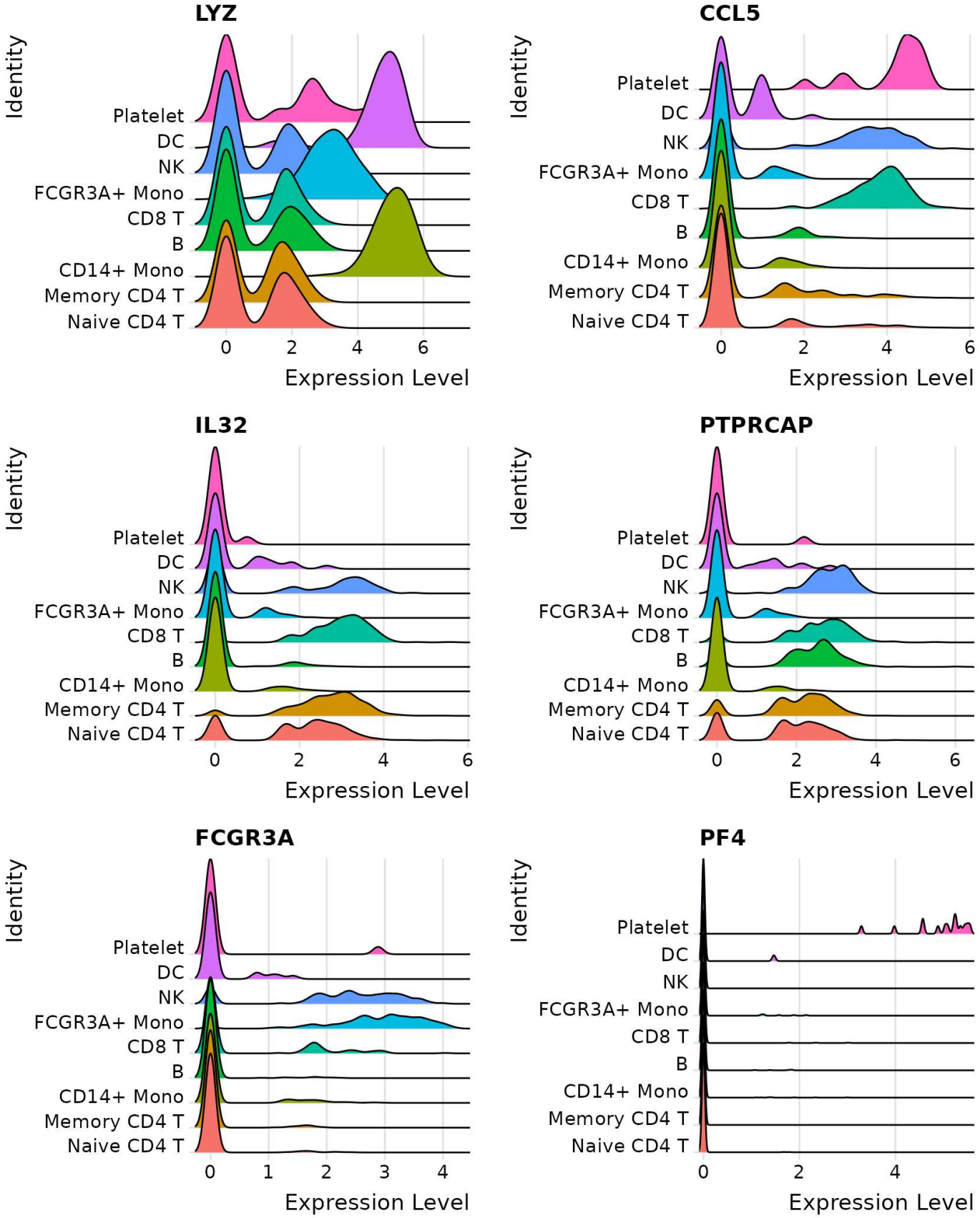
```
SeuratData::InstallData("pbmc3k")
```

```
library(Seurat)
library(SeuratData)
library(ggplot2)
library(patchwork)
pbmc3k.final <- LoadData("pbmc3k", type = "pbmc3k.final")
pbmc3k.final$groups <- sample(c("group1", "group2"), size = ncol(pbmc3k.final), replace = TRUE)
features <- c("LYZ", "CCL5", "IL32", "PTPRCAP", "FCGR3A", "PF4")
pbmc3k.final
```

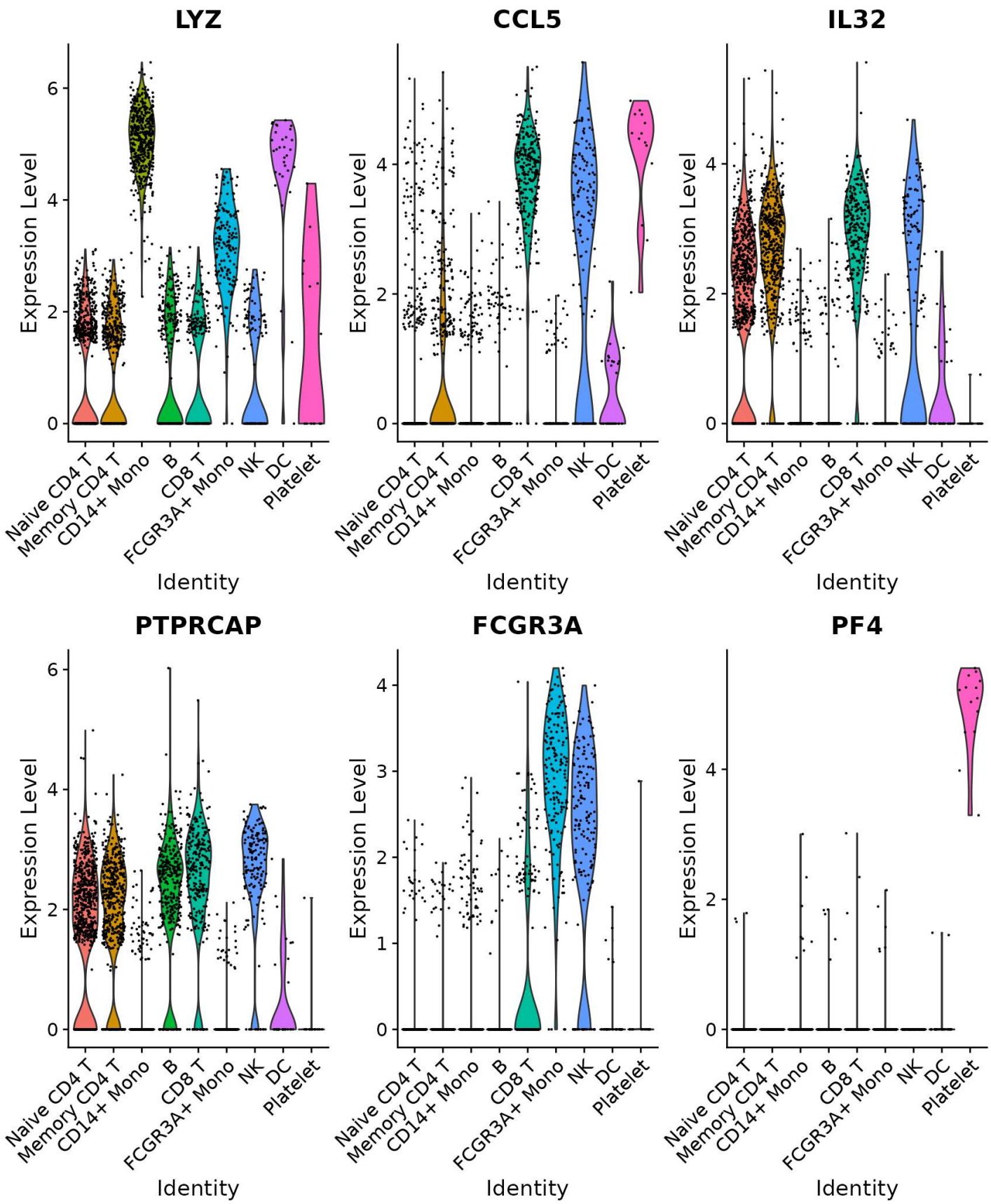
```
## An object of class Seurat
## 13714 features across 2638 samples within 1 assay
## Active assay: RNA (13714 features, 2000 variable features)
##  3 layers present: data, counts, scale.data
##  2 dimensional reductions calculated: pca, umap
```

Five visualizations of marker feature expression

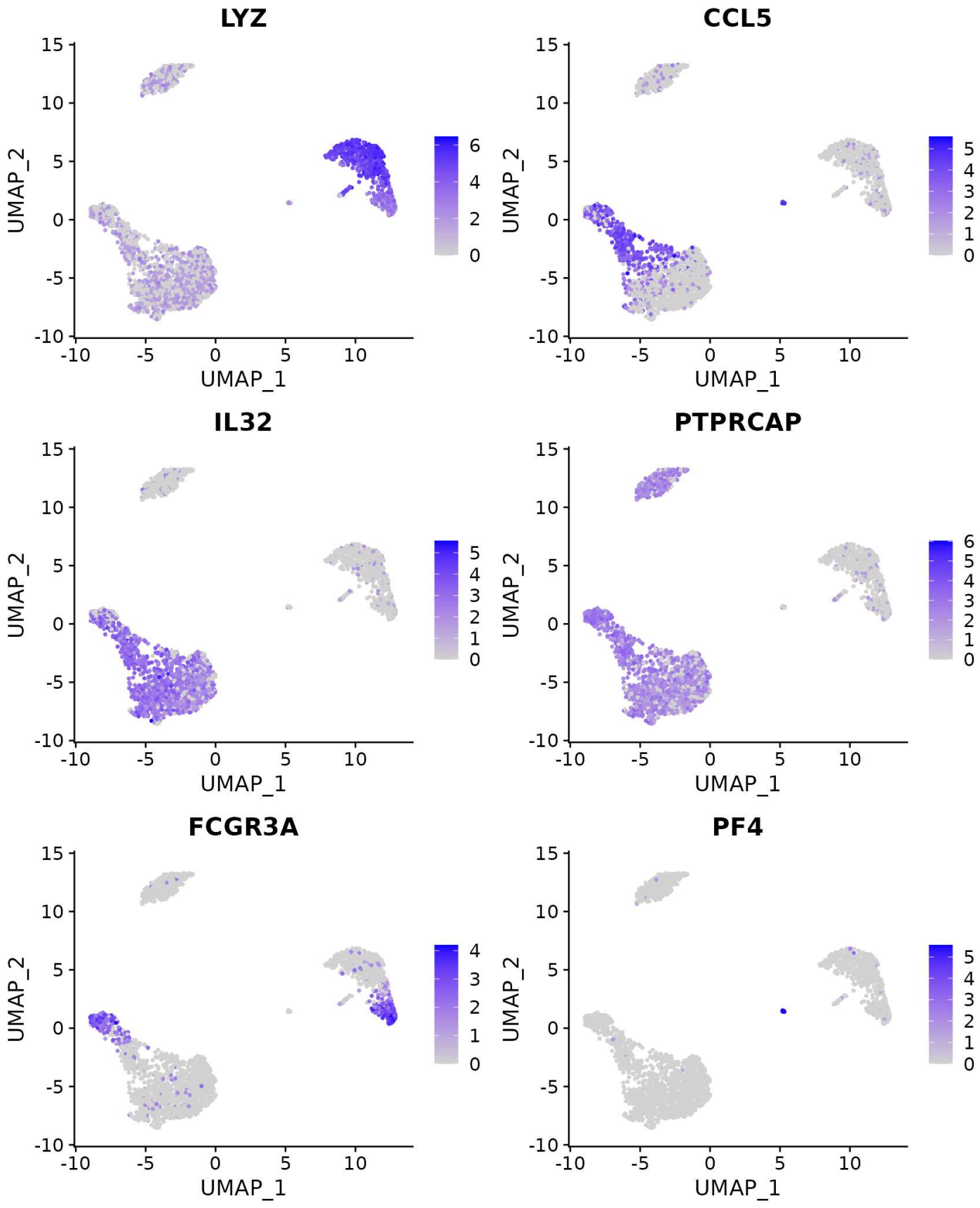
```
# Ridge plots - from ggridges. Visualize single cell expression distributions in each cluster
RidgePlot(pbmc3k.final, features = features, ncol = 2)
```



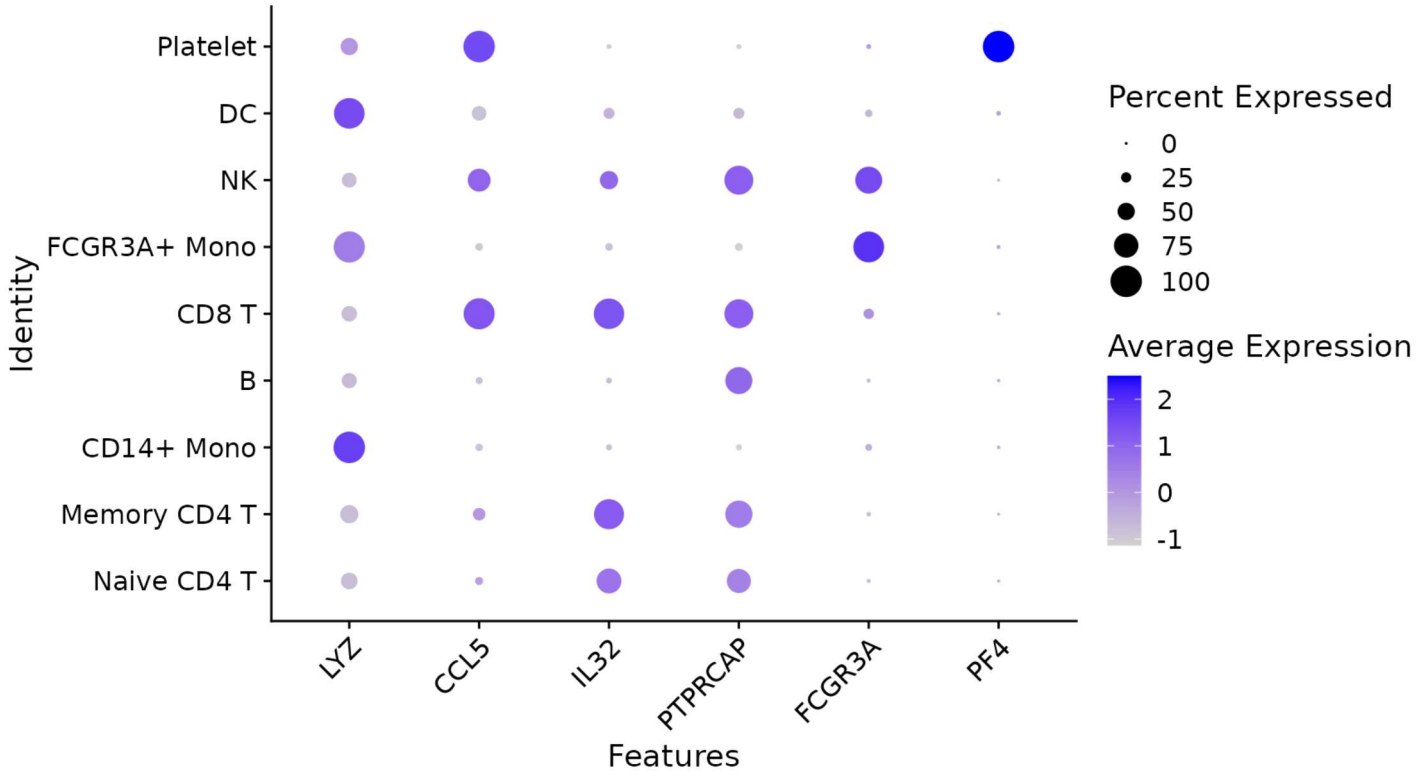

```
# Violin plot - Visualize single cell expression distributions in each cluster
VlnPlot(pbmc3k.final, features = features)
```



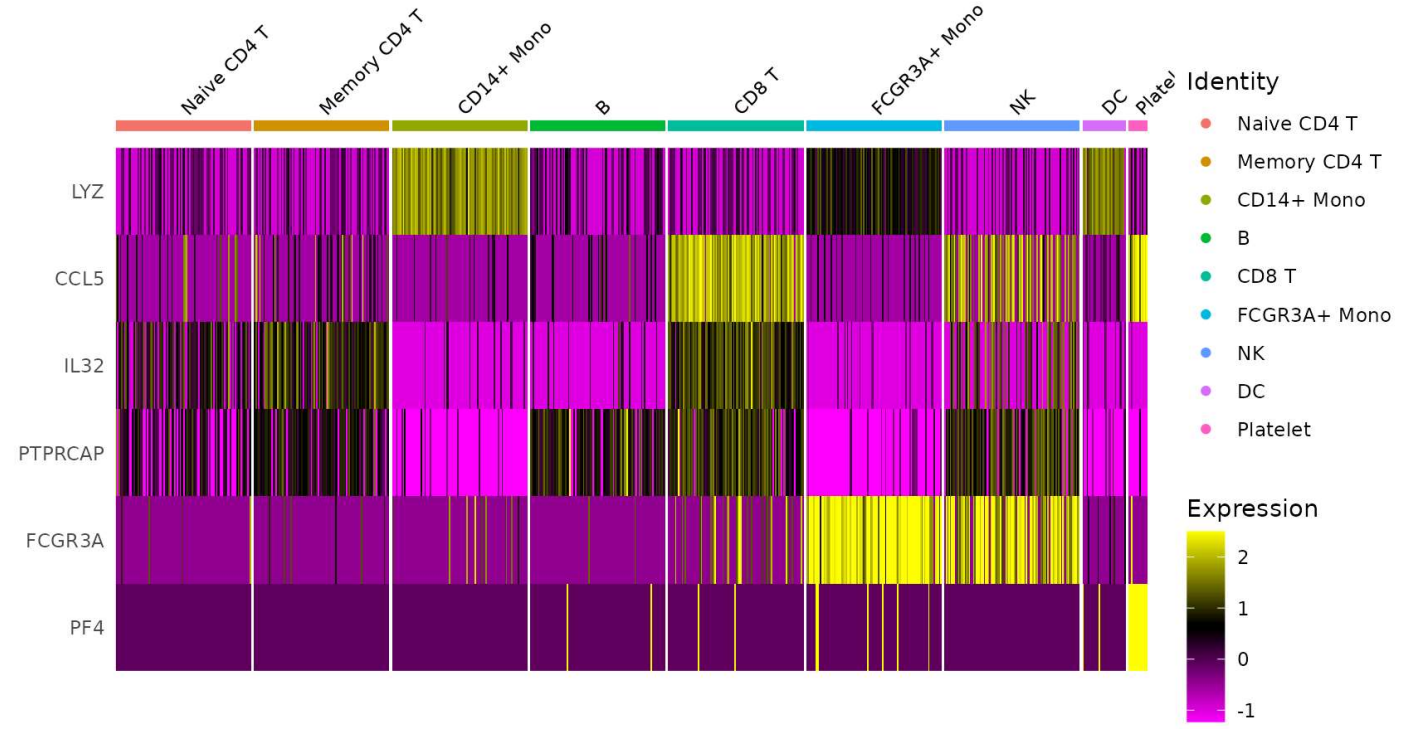

```
# Feature plot - visualize feature expression in low-dimensional space
FeaturePlot(pbmc3k.final, features = features)
```




```
# Dot plots - the size of the dot corresponds to the percentage of cells expressing the
# feature in each cluster. The color represents the average expression level
DotPlot(pbmc3k.final, features = features) + RotatedAxis()
```

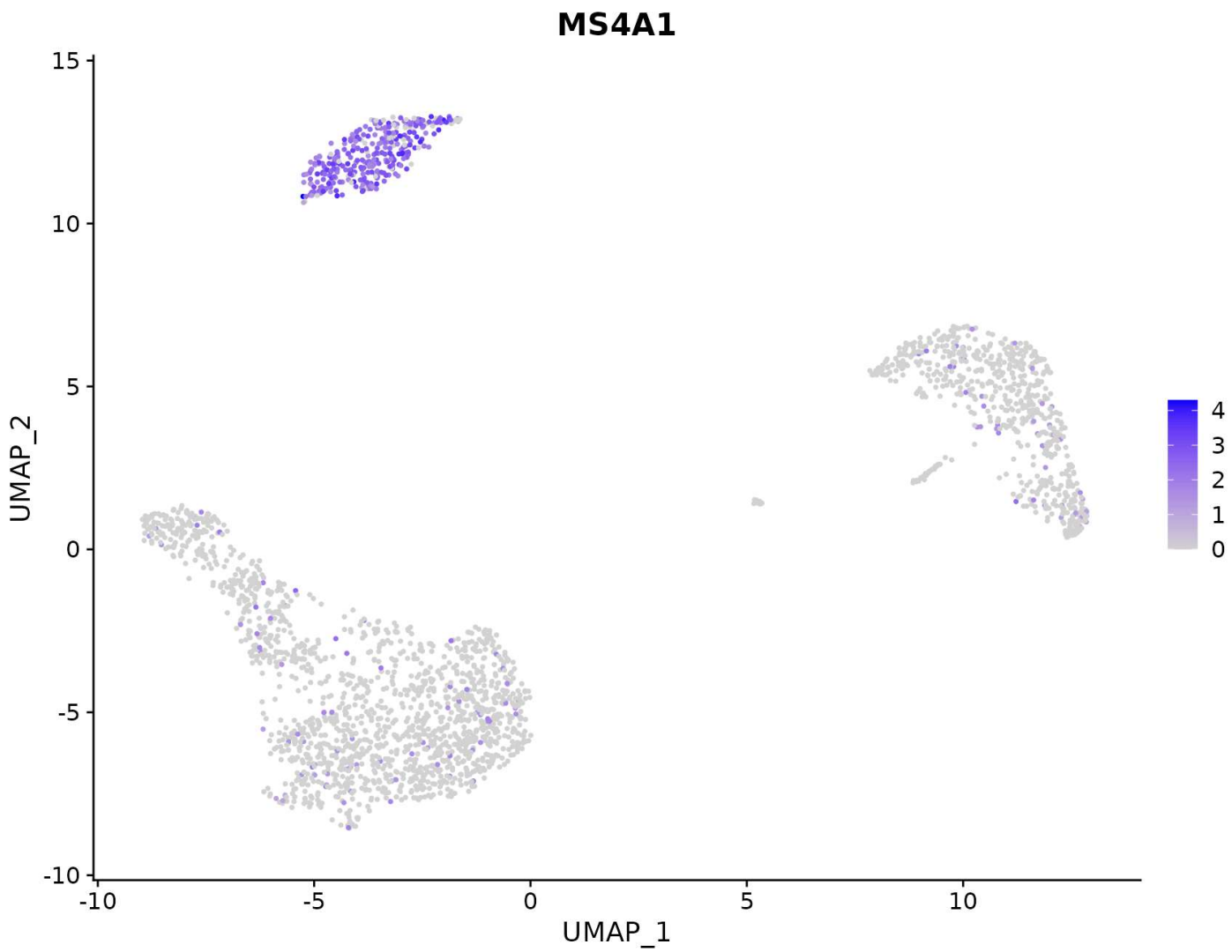



```
# Single cell heatmap of feature expression
DoHeatmap(subset(pbmc3k.final, downsample = 100), features = features, size = 3)
```

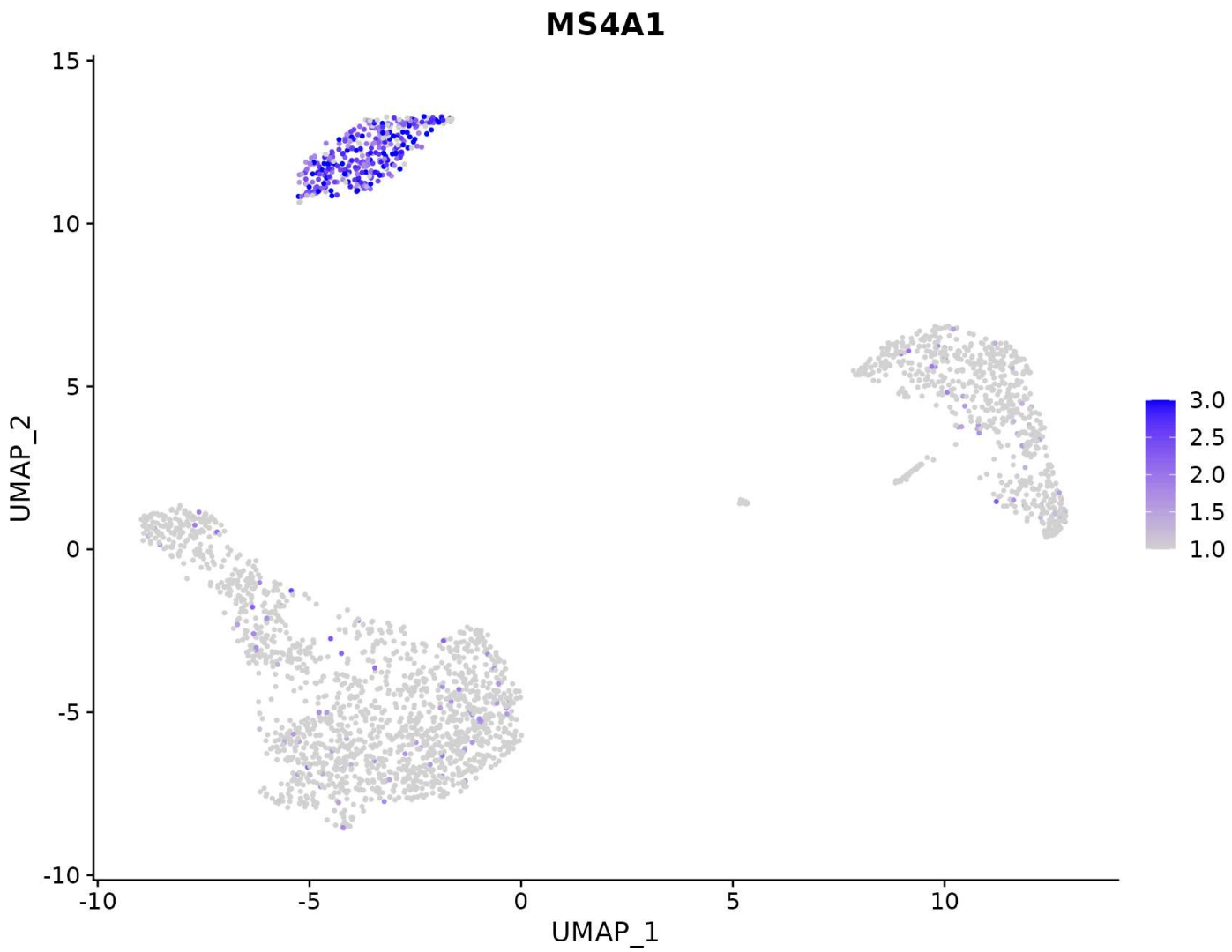


New additions to `FeaturePlot`

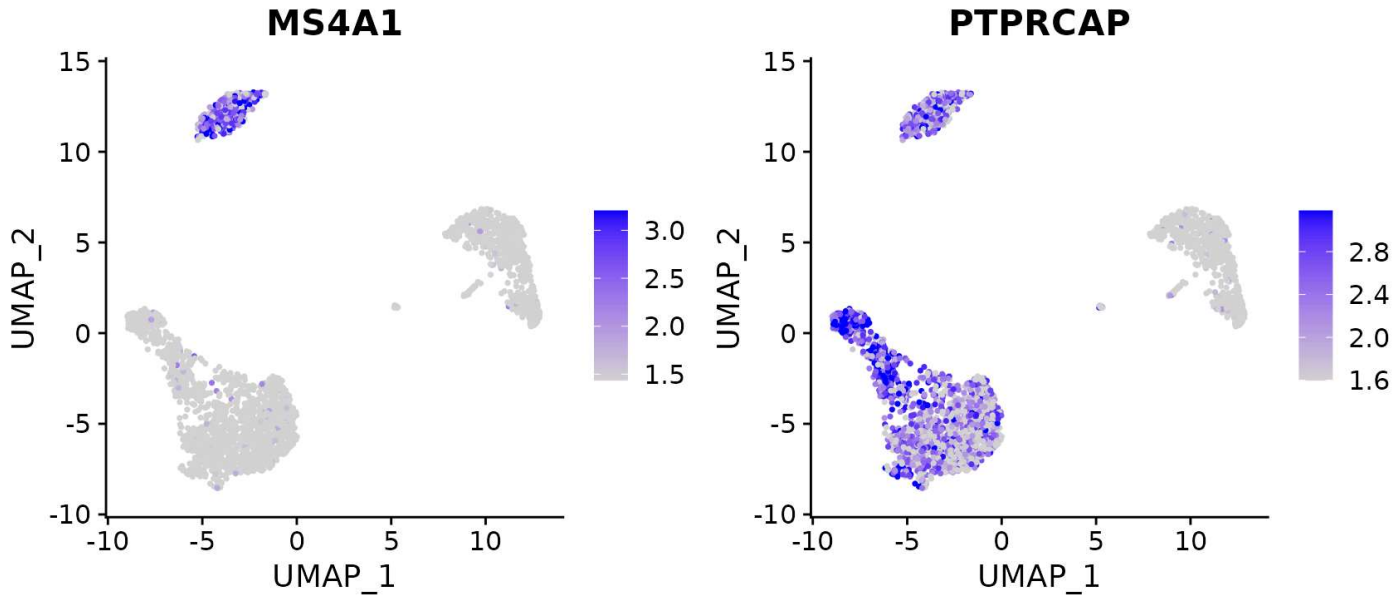
```
# Plot a legend to map colors to expression levels
FeaturePlot(pbmc3k.final, features = "MS4A1")
```



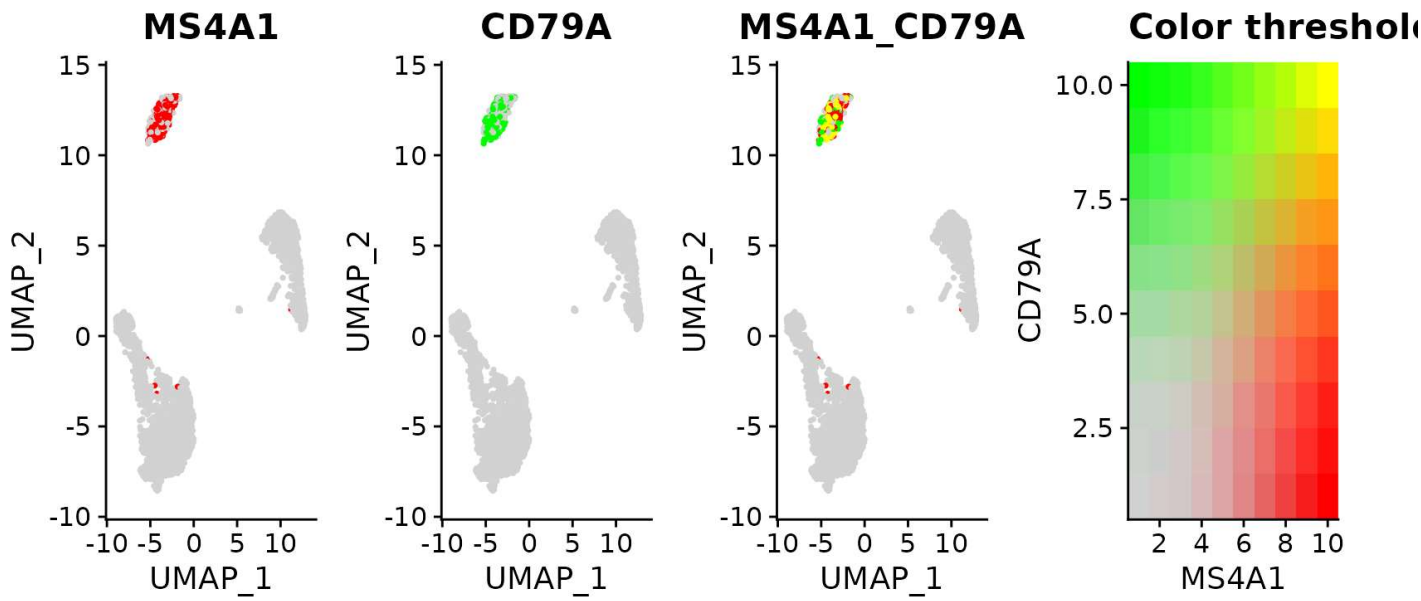

```
# Adjust the contrast in the plot
FeaturePlot(pbmc3k.final, features = "MS4A1", min.cutoff = 1, max.cutoff = 3)
```



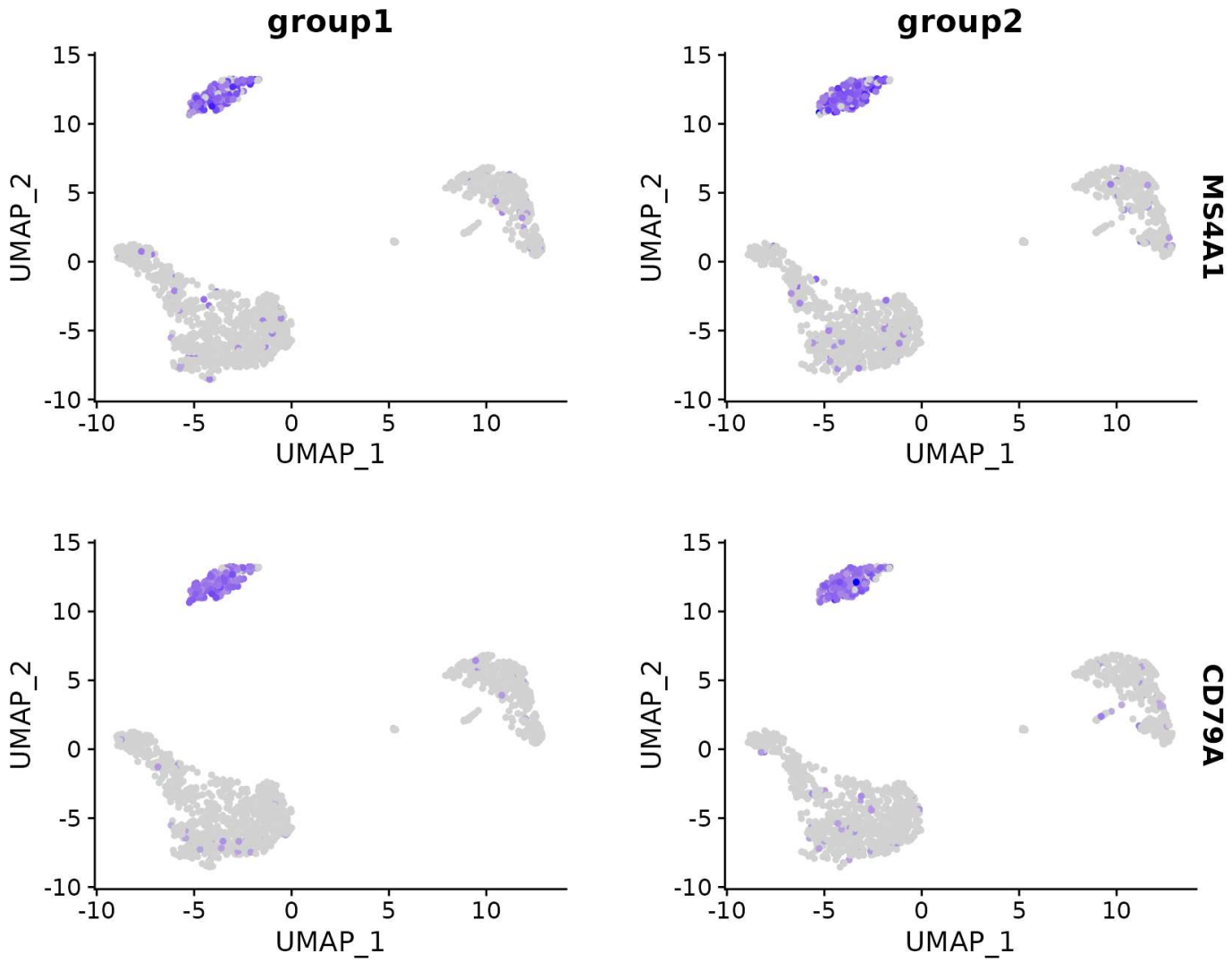

```
# Calculate feature-specific contrast levels based on quantiles of non-zero expression.
# Particularly useful when plotting multiple markers
FeaturePlot(pbmc3k.final, features = c("MS4A1", "PTPRCAP"), min.cutoff = "q10", max.cutoff = "q90")
```



```
# Visualize co-expression of two features simultaneously
FeaturePlot(pbmc3k.final, features = c("MS4A1", "CD79A"), blend = TRUE)
```



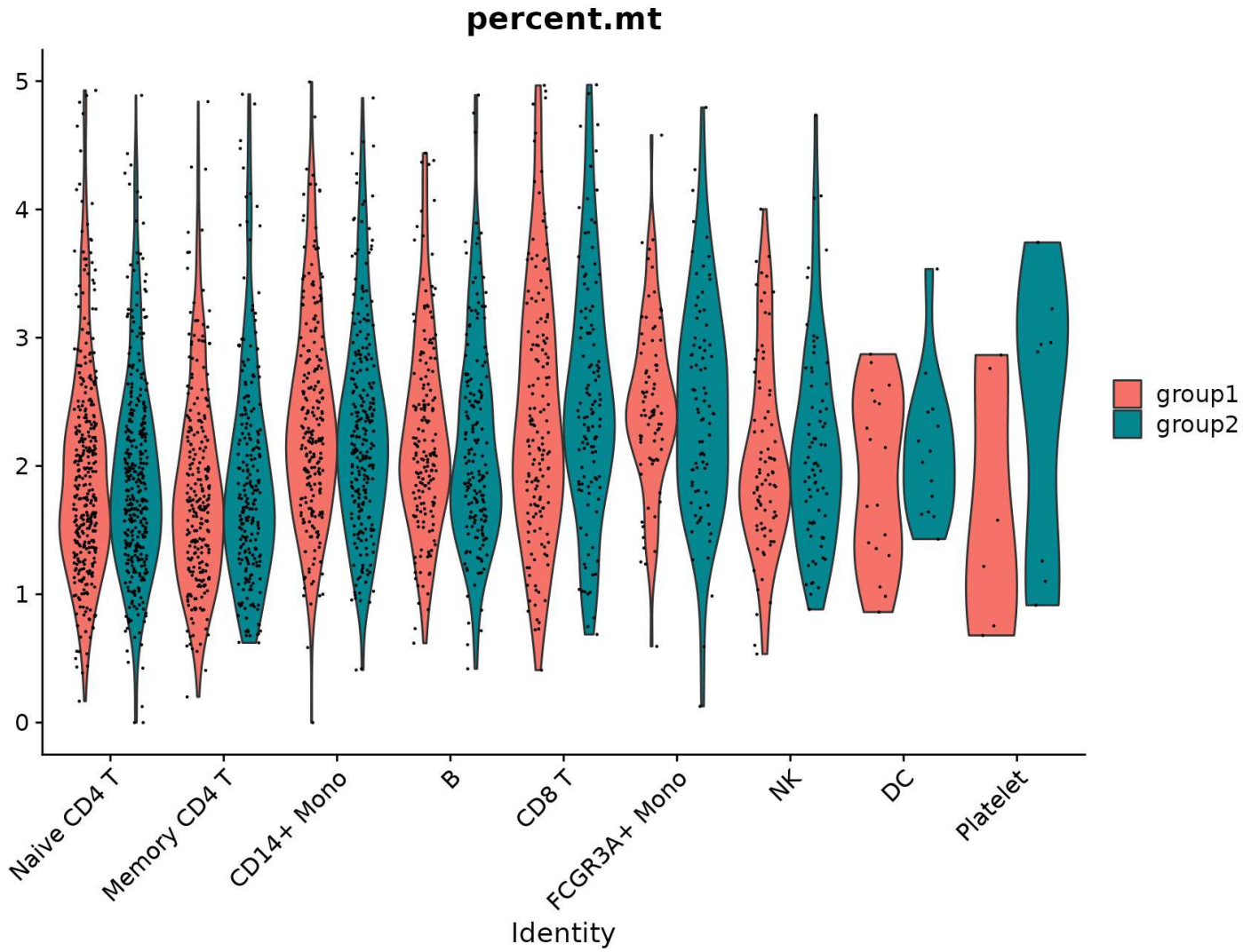

```
# Split visualization to view expression by groups (replaces FeatureHeatmap)
FeaturePlot(pbmc3k.final, features = c("MS4A1", "CD79A"), split.by = "groups")
```



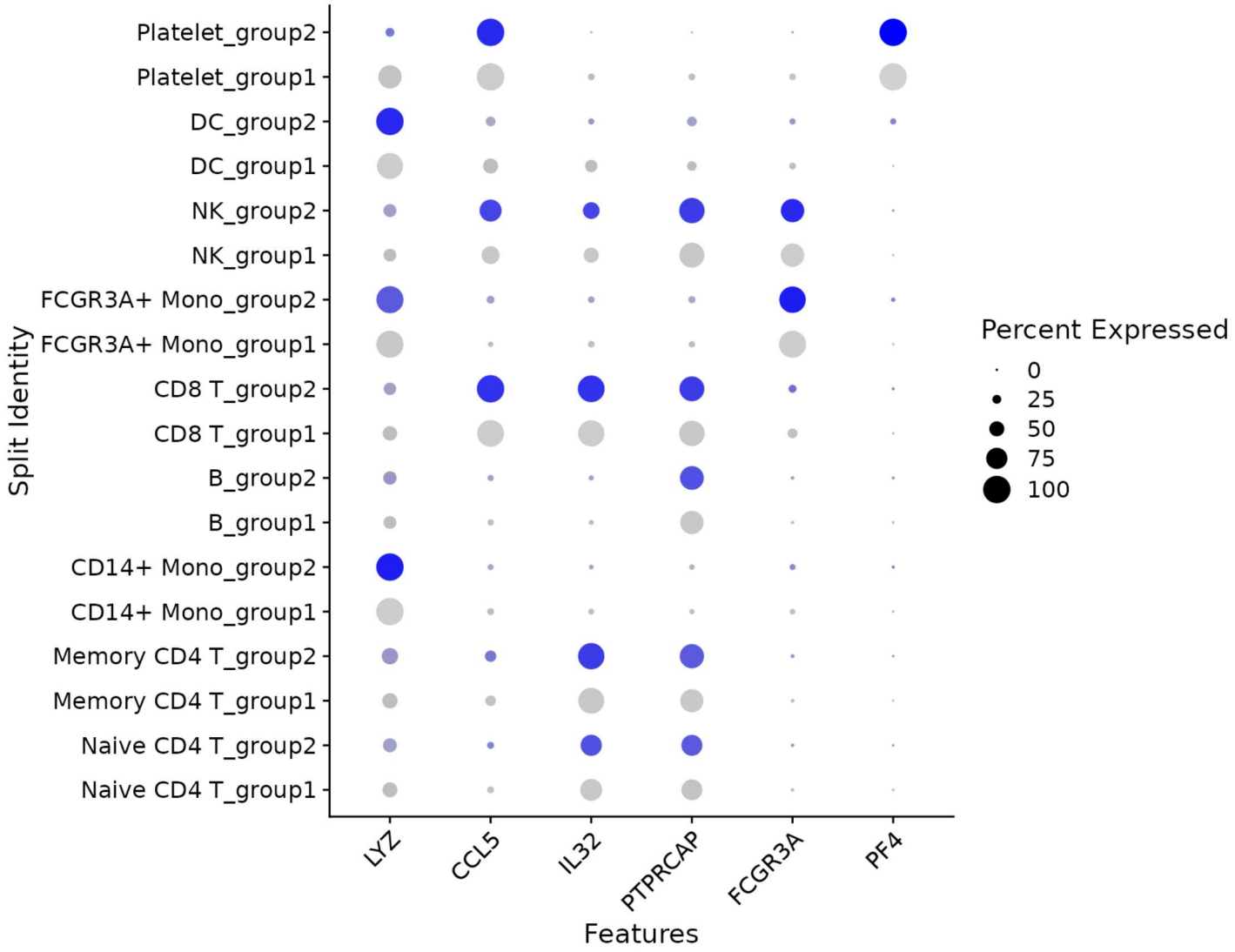
Updated and expanded visualization functions

In addition to changes to `FeaturePlot()`, several other plotting functions have been updated and expanded with new features and taking over the role of now-deprecated functions

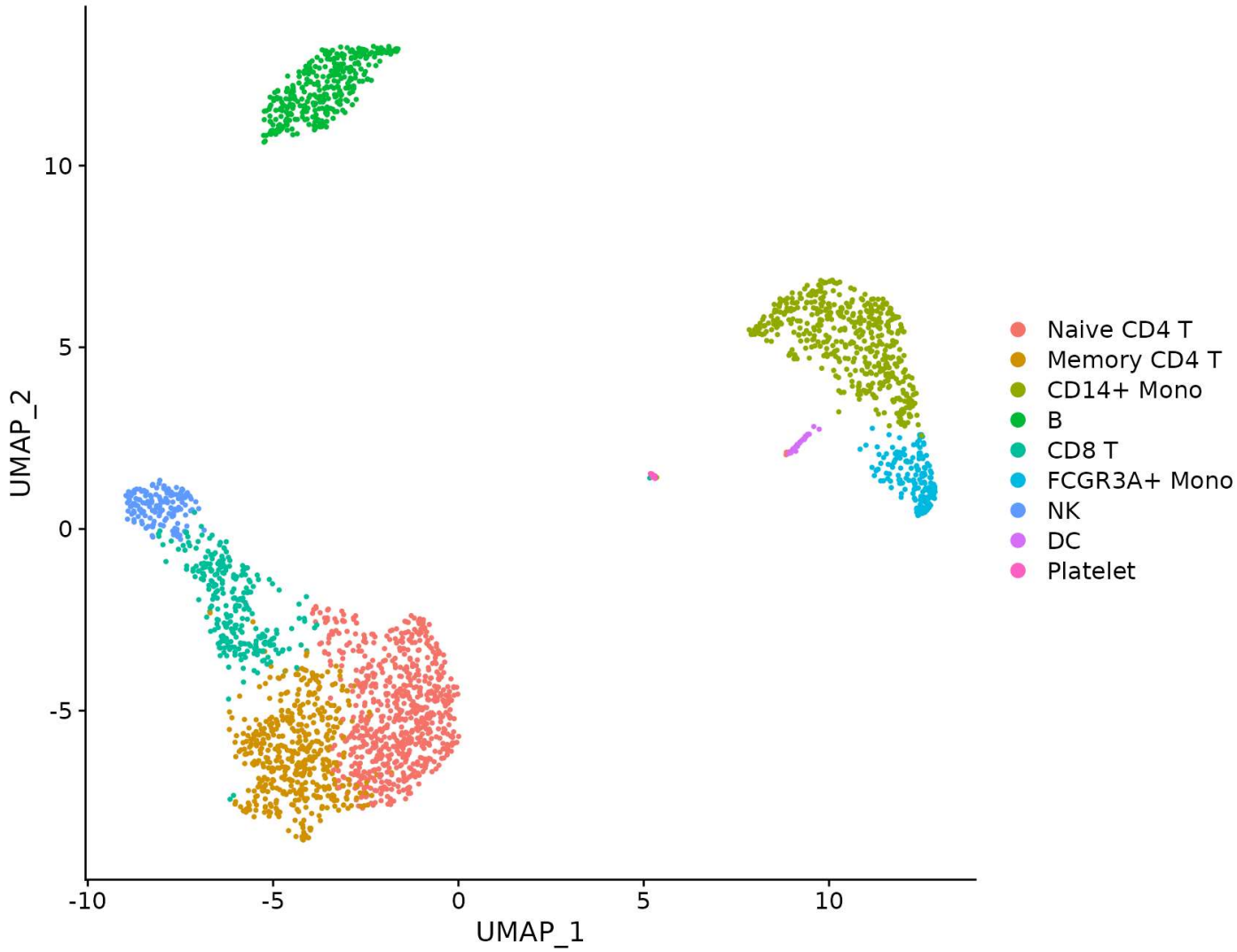
```
# Violin plots can also be split on some variable. Simply add the splitting variable to object
# metadata and pass it to the split.by argument
VlnPlot(pbmc3k.final, features = "percent.mt", split.by = "groups")
```



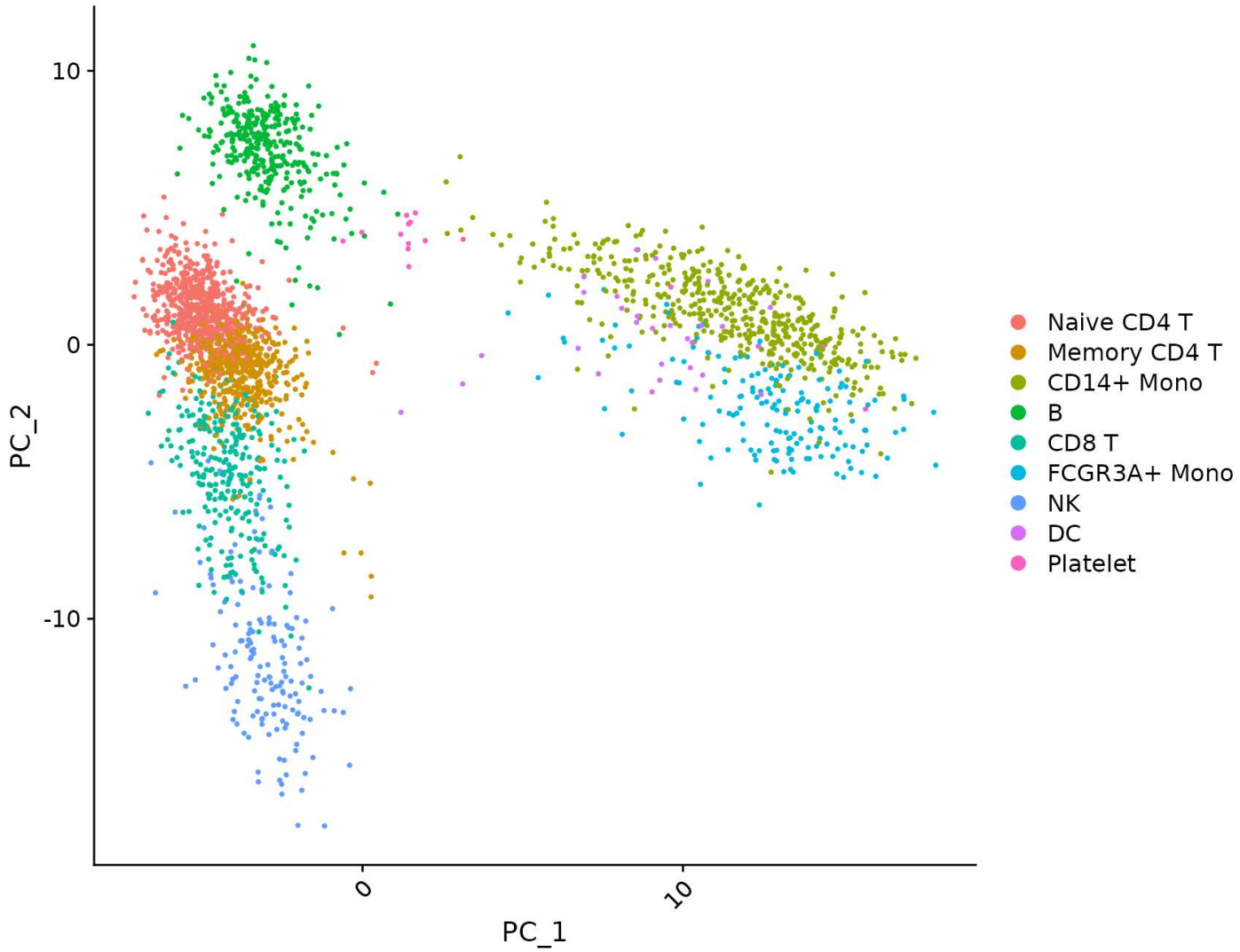

```
# SplitDotPlotGG has been replaced with the `split.by` parameter for DotPlot
DotPlot(pbmc3k.final, features = features, split.by = "groups") + RotatedAxis()
```



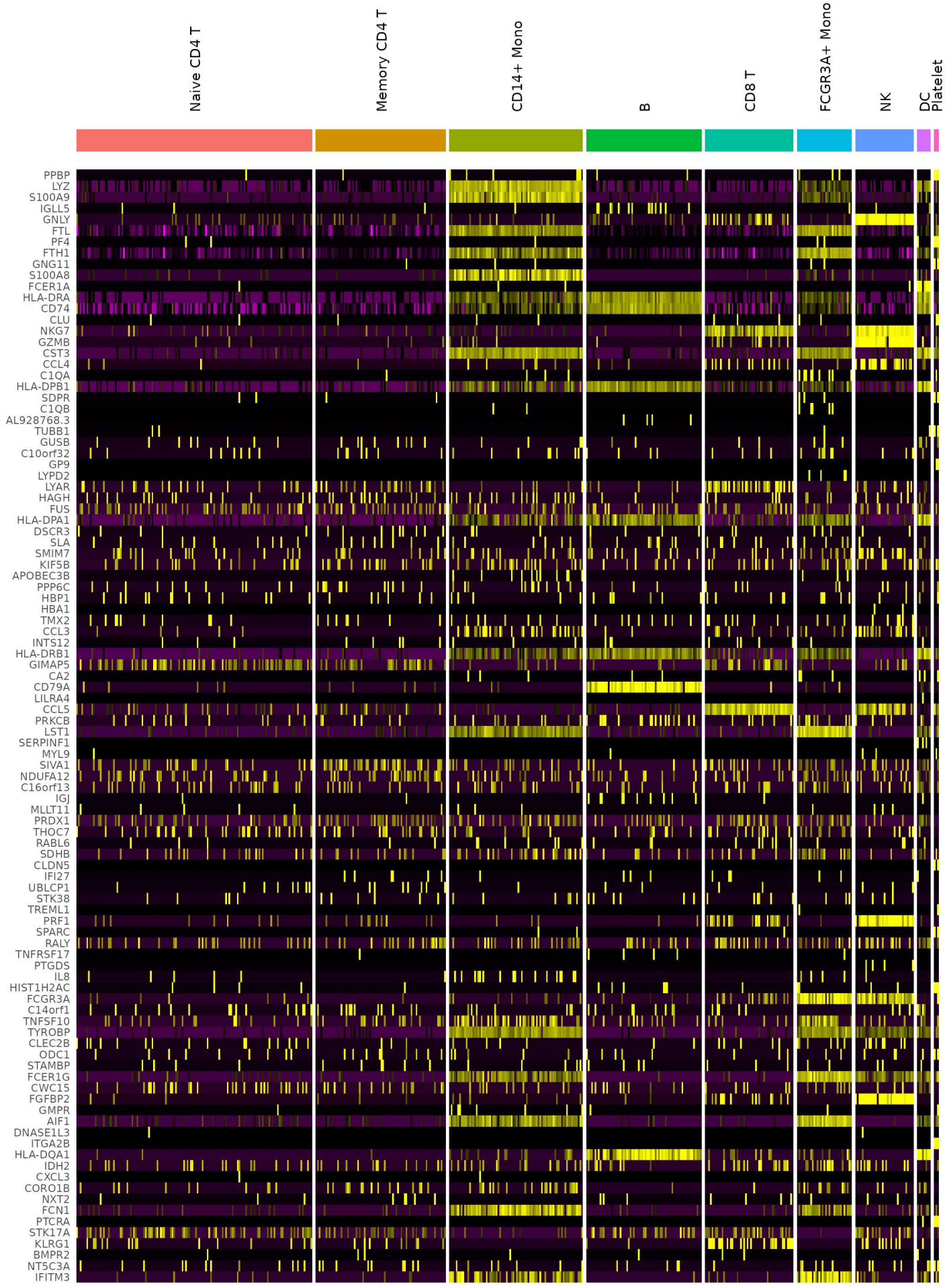

```
# DimPlot replaces TSNEPlot, PCAPlot, etc. In addition, it will plot either 'umap', 'tsne', or
# 'pca' by default, in that order
DimPlot(pbmc3k.final)
```




```
pbmc3k.final.no.umap <- pbmc3k.final
pbmc3k.final.no.umap[["umap"]] <- NULL
DimPlot(pbmc3k.final.no.umap) + RotatedAxis()
```



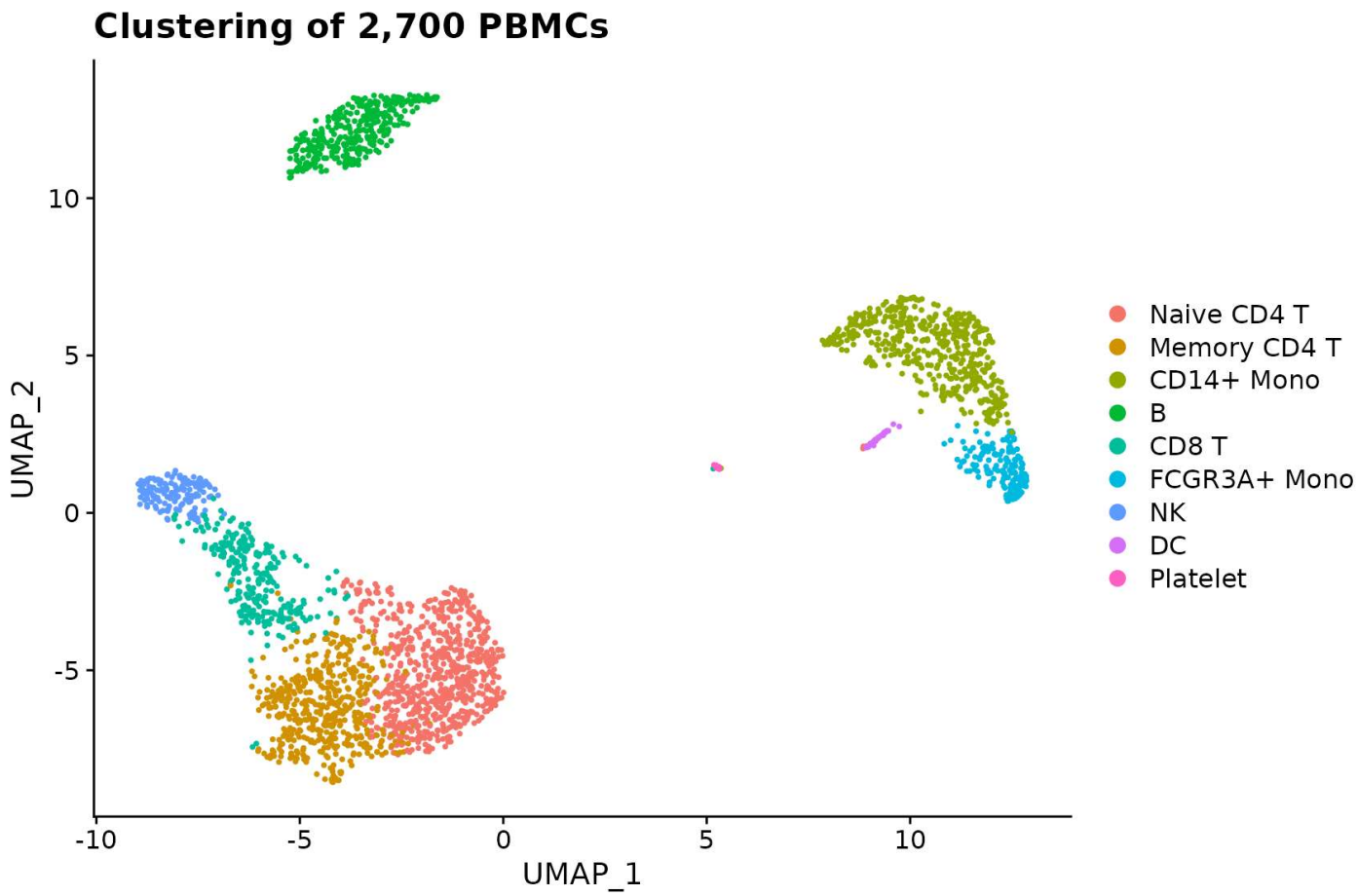

```
# DoHeatmap now shows a grouping bar, splitting the heatmap into groups or clusters. This can
# be changed with the `group.by` parameter
DoHeatmap(pbmc3k.final, features = VariableFeatures(pbmc3k.final)[1:100], cells = 1:500, size = 4,
    angle = 90) + NoLegend()
```


Naive CD4 T
Memory CD4 T
CD14+ Mono
B
CD8 T
FCGR3A+ Mono
NK
DC
Platelet
PPBP
LYZ
S100A9
IGLL5
GNLY
FTL
PF4
FTH1
GNG11
S100A8
FCER1A
HLA-DRA
CD74
CLU
NKG7
GZMB
CST3
CCL4
C1QA
HLA-DPB1
SDPR
C1QB
AL928768.3
TUBB1
GUSB
C10orf32
GP9
LYPD2
LYAR
HAGH
FUS
HLA-DPA1
DSCR3
SLA
SMIM7
KIF5B
APOBEC3B
PPP6C
HBP1
HBA1
TMX2
CCL3
INTS12
HLA-DRB1
GIMAP5
CA2
CD79A
LILRA4
CCL5
PRKCB
LST1
SERPINF1
MYL9
SIVA1
NDUFA12
C16orf13
IGJ
MLLT11
PRDX1
THOC7
RABL6
SDHB
CLDN5
IFI27
UBLCP1
STK38
TREML1
PRF1
SPARC
RALY
TNFRSF17
PTGDS
IL8
HIST1H2AC
FCGR3A
C14orf1
TNFSF10
TYROBP
CLEC2B
ODC1
STAMBP
FCER1G
CWC15
FGFBP2
GMPR
AIF1
DNASE1L3
ITGA2B
HLA-DQA1
IDH2
CXCL3
CORO1B
NXT2
FCN1
PTCRA
STK17A
KLRG1
BMPR2
NT5C3A
IFITM3

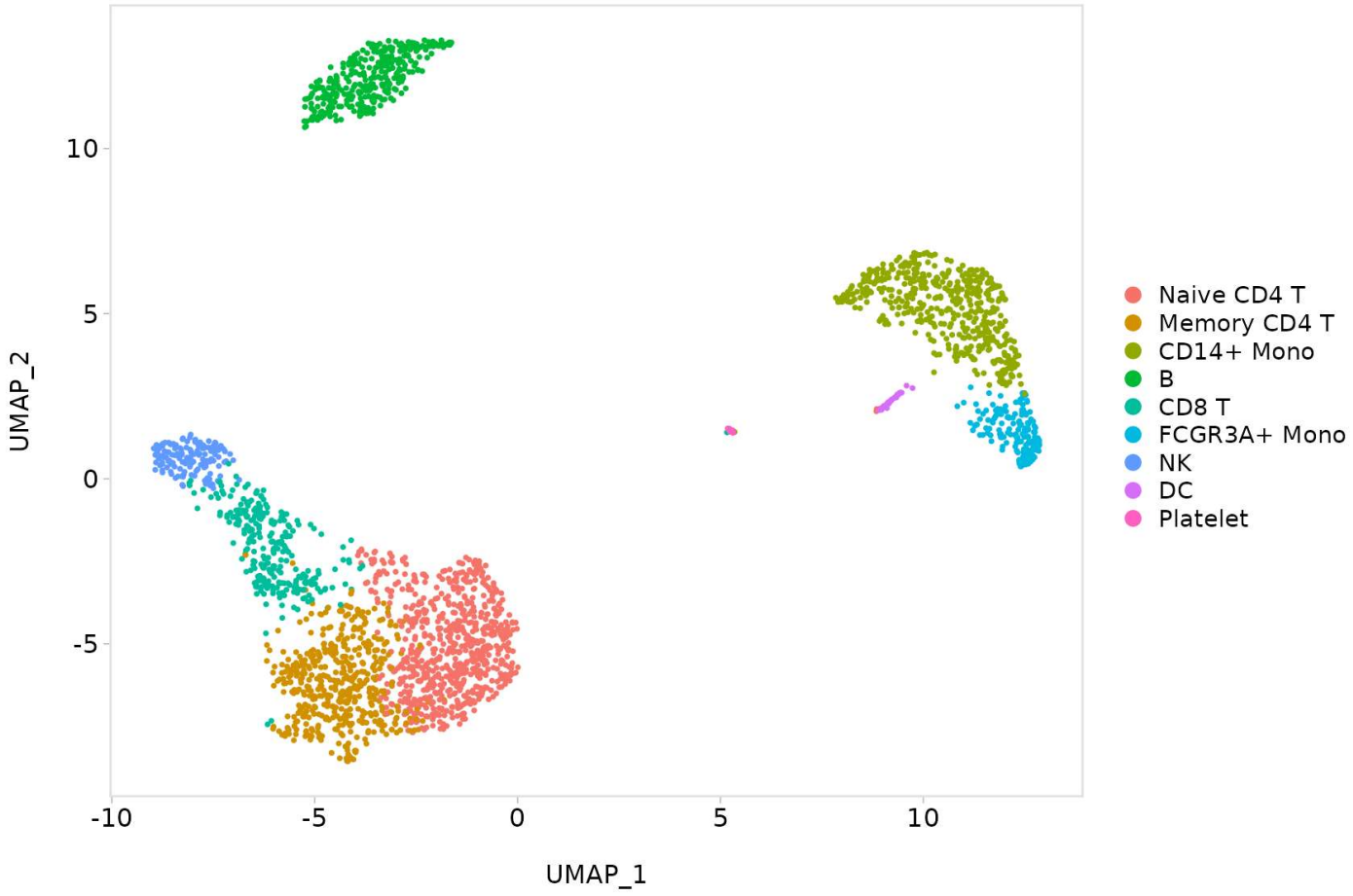
Applying themes to plots

With Seurat, all plotting functions return ggplot2-based plots by default, allowing one to easily capture and manipulate plots just like any other ggplot2-based plot.

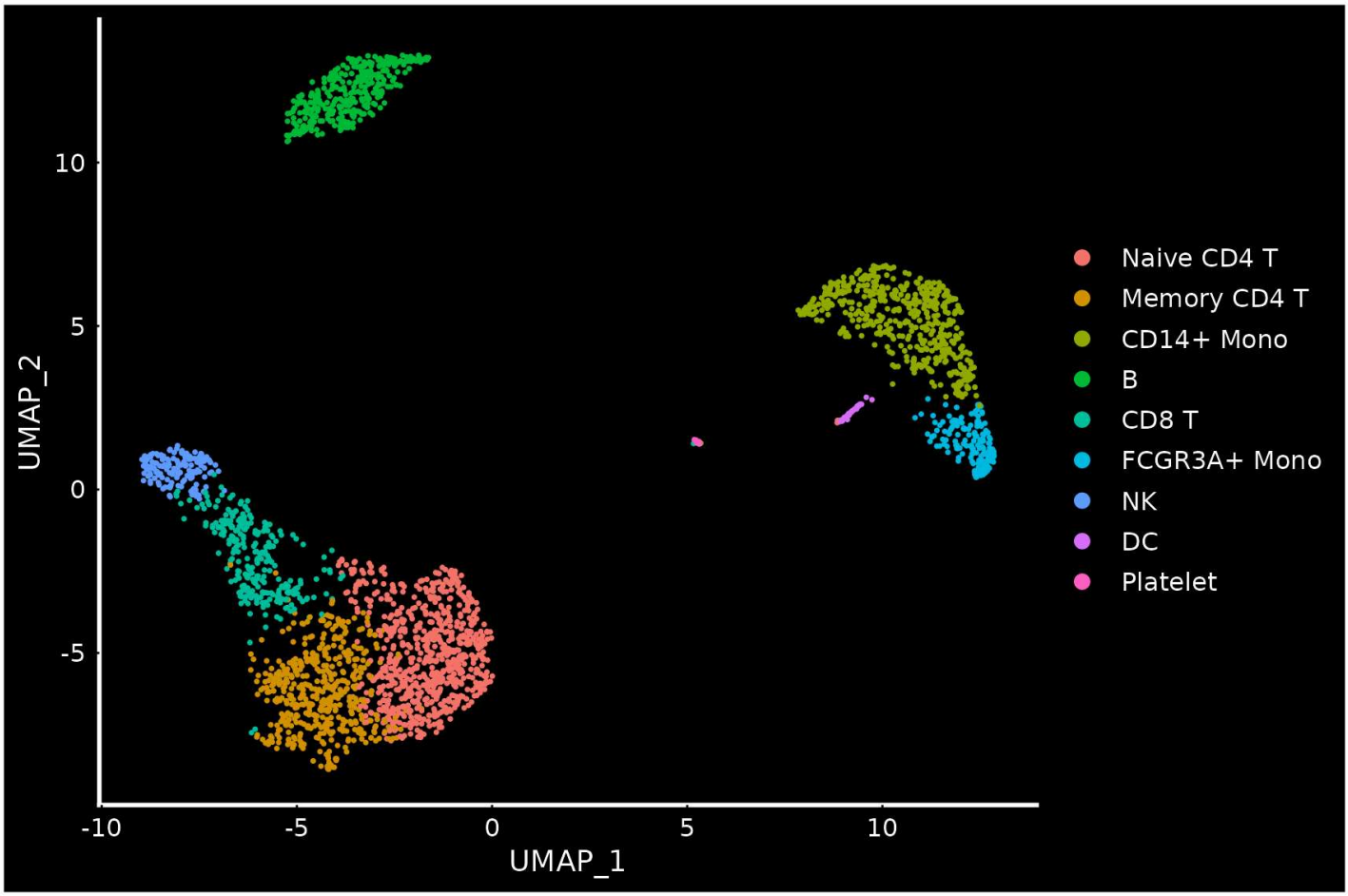
```
baseplot <- DimPlot(pbmc3k.final, reduction = "umap")
# Add custom labels and titles
baseplot + labs(title = "Clustering of 2,700 PBMCs")
```



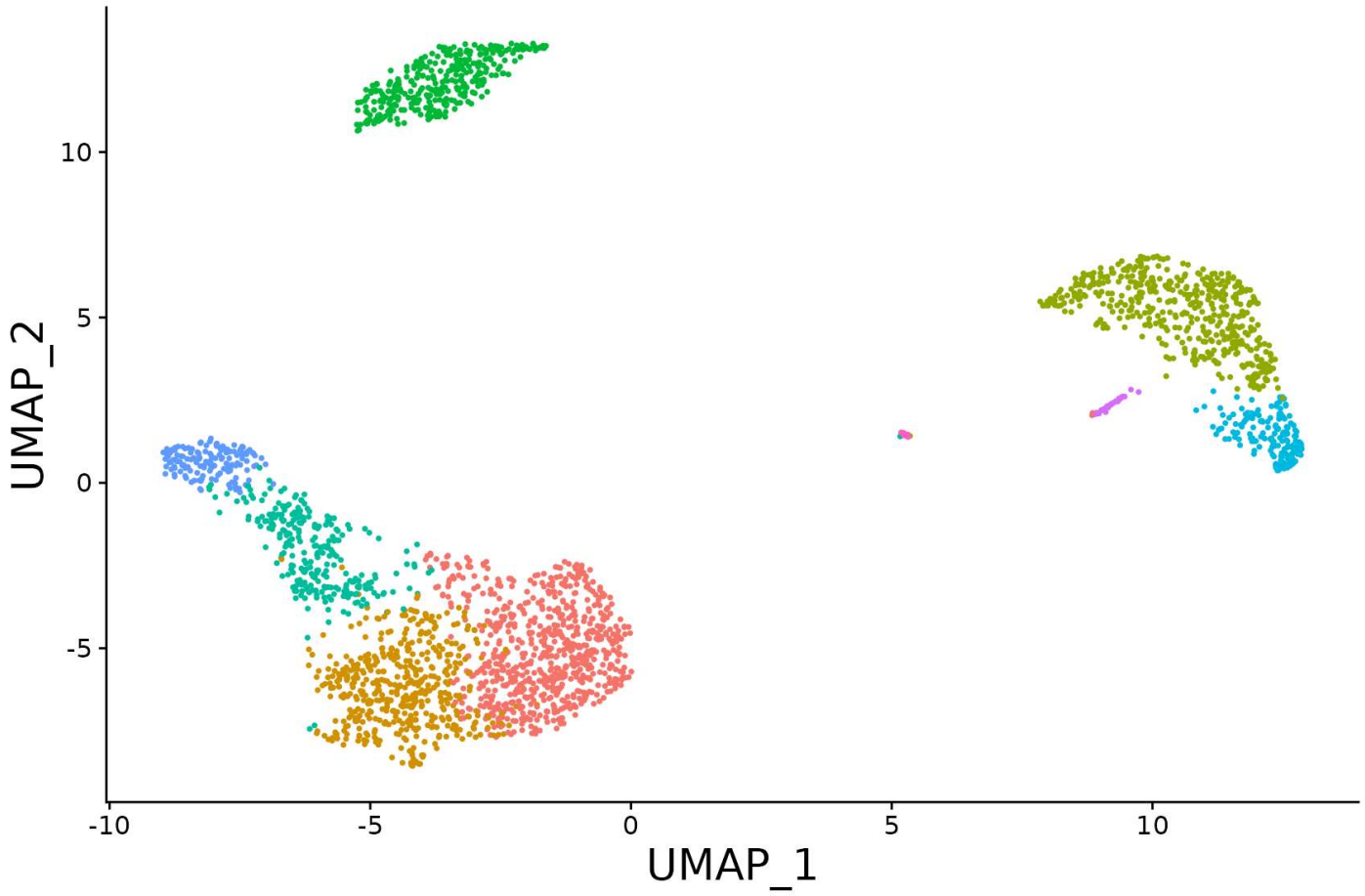

```
# Use community-created themes, overwriting the default Seurat-applied theme Install ggmin
# with remotes::install_github('sjessa/ggmin')
baseplot + ggmin::theme_powerpoint()
```




```
# Seurat also provides several built-in themes, such as DarkTheme; for more details see
# ?SeuratTheme
baseplot + DarkTheme()
```



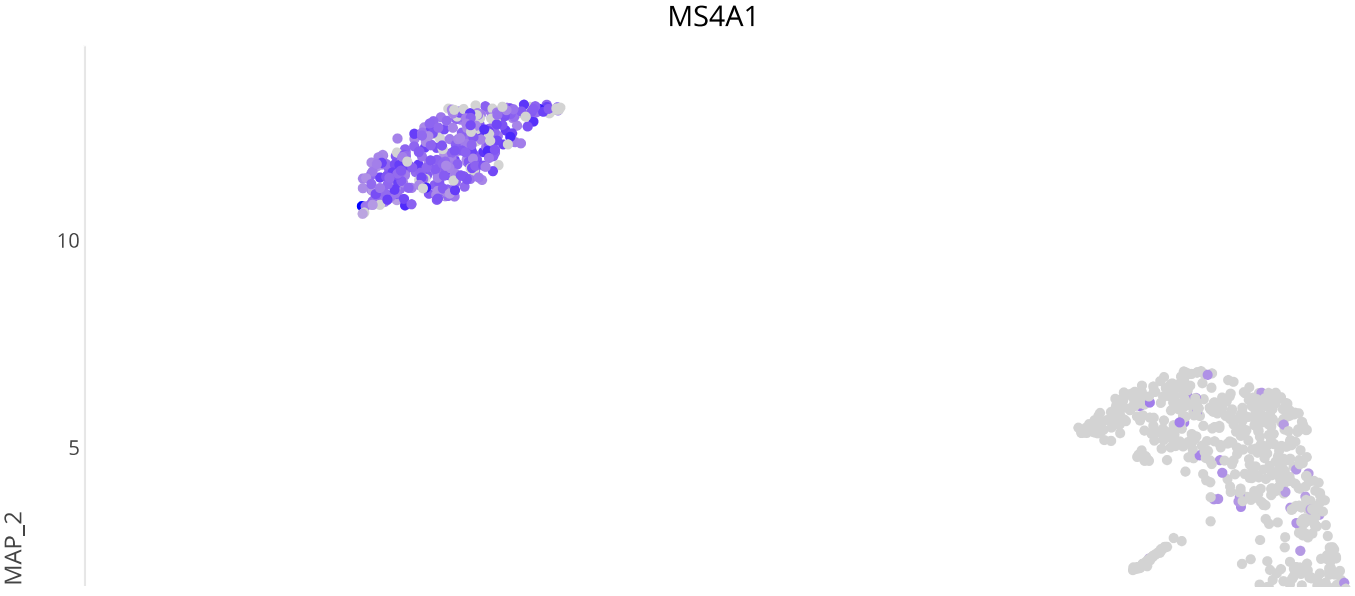

```
# Chain themes together
baseplot + FontSize(x.title = 20, y.title = 20) + NoLegend()
```

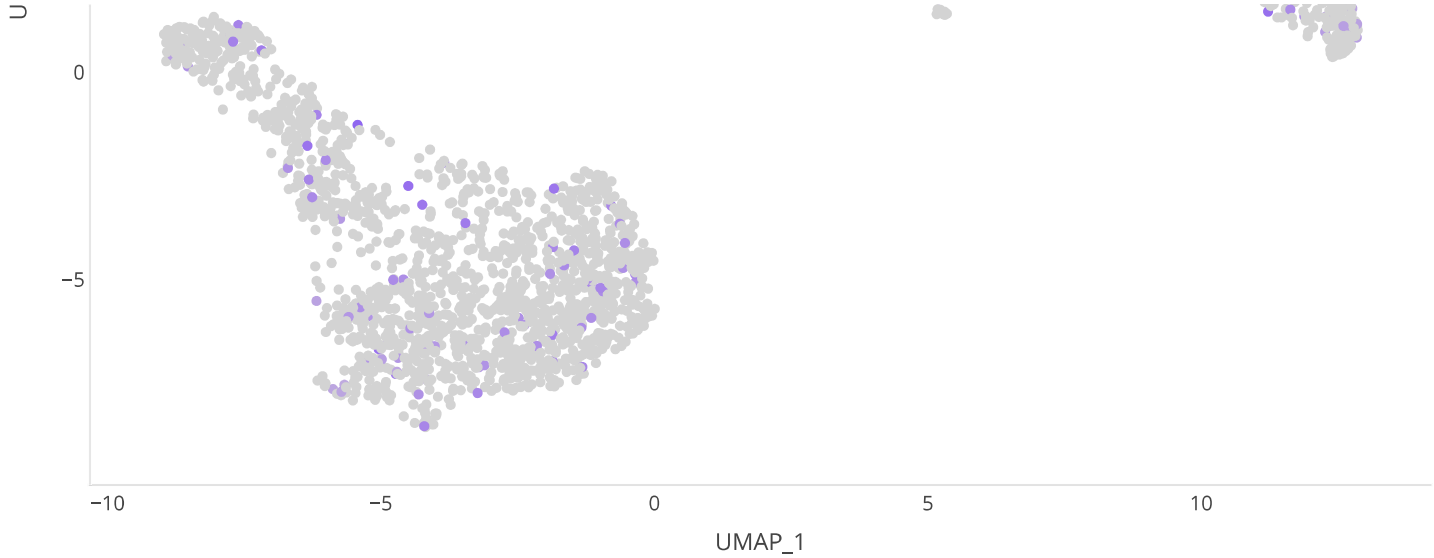


Interactive plotting features

Seurat utilizes R's plotly graphing library to create interactive plots. This interactive plotting feature works with any ggplot2-based scatter plots (requires a `geom_point` layer). To use, simply make a ggplot2-based scatter plot (such as `DimPlot()` or `FeaturePlot()`) and pass the resulting plot to `HoverLocator()`

```
# Include additional data to display alongside cell names by passing in a data frame of
# information.  Works well when using FetchData
plot <- FeaturePlot(pbmc3k.final, features = "MS4A1")
HoverLocator(plot = plot, information = FetchData(pbmc3k.final, vars = c("ident", "PC_1", "nFeature_RNA")))
```

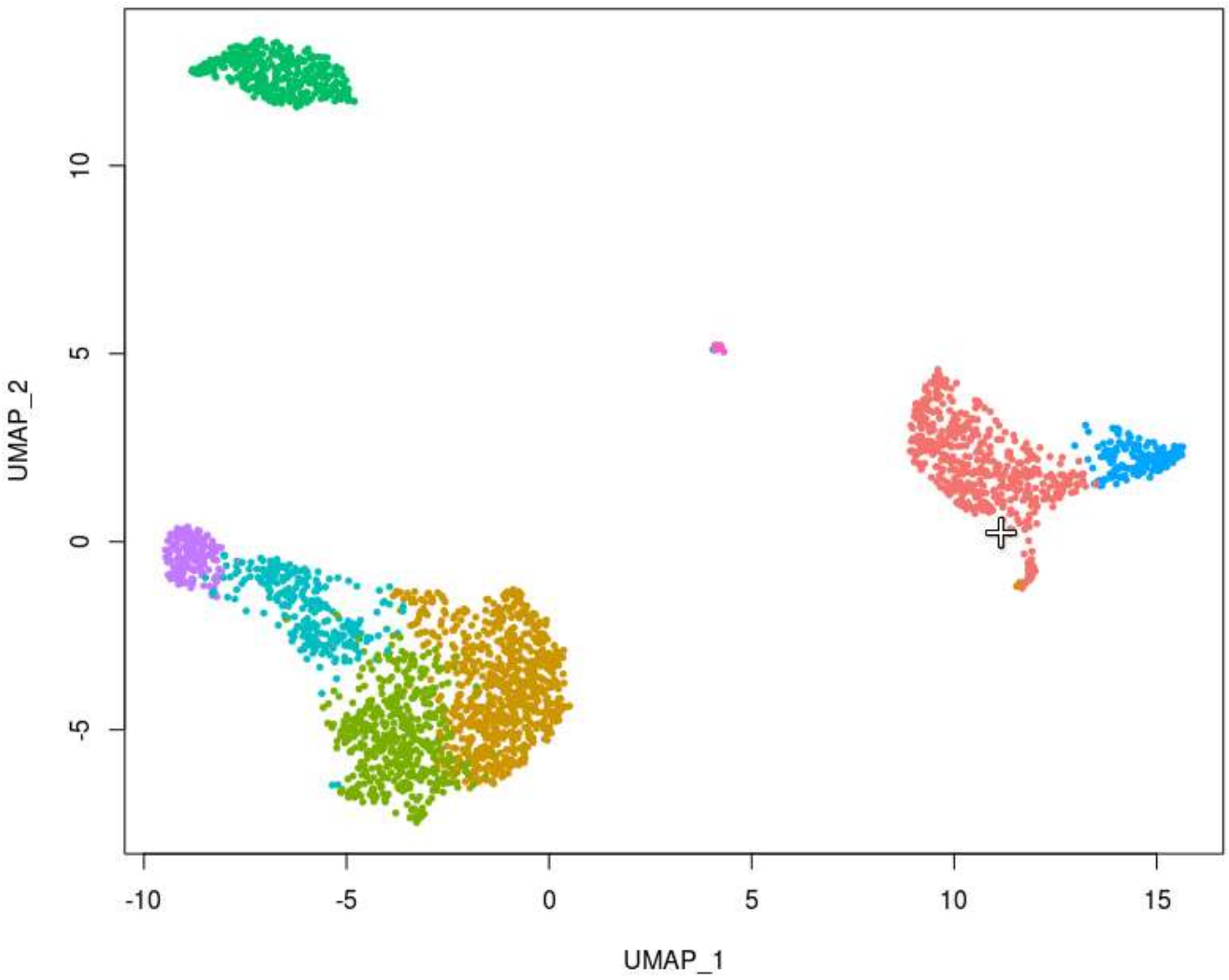




Another interactive feature provided by Seurat is being able to manually select cells for further investigation. We have found this particularly useful for small clusters that do not always separate using unbiased clustering, but which look tantalizingly distinct. You can now select these cells by creating a ggplot2-based scatter plot (such as with `DimPlot()` or `FeaturePlot()`, and passing the returned plot to `CellSelector()`. `CellSelector()` will return a vector with the names of the points selected, so that you can then set them to a new identity class and perform differential expression.

For example, let's pretend that DCs had merged with monocytes in the clustering, but we wanted to see what was unique about them based on their position in the tSNE plot.

```
pbmc3k.final <- RenameIdents(pbmc3k.final, DC = "CD14+ Mono")
plot <- DimPlot(pbmc3k.final, reduction = "umap")
select.cells <- CellSelector(plot = plot)
```



We can then change the identity of these cells to turn them into their own mini-cluster.

```
head(select.cells)
```

```
## [1] "AAGATTACCGCCTT" "AAGCCATGAACTGC" "AATTACGAATTCCT" "ACCCGTTGCTTCTA"
## [5] "ACGAGGGACAGGAG" "ACGTGATGCCATGA"
```

```
Idents(pbmc3k.final, cells = select.cells) <- "NewCells"

# Now, we find markers that are specific to the new cells, and find clear DC markers
newcells.markers <- FindMarkers(pbmc3k.final, ident.1 = "NewCells", ident.2 = "CD14+ Mono", min.diff.pct =
0.3,
    only.pos = TRUE)
head(newcells.markers)
```

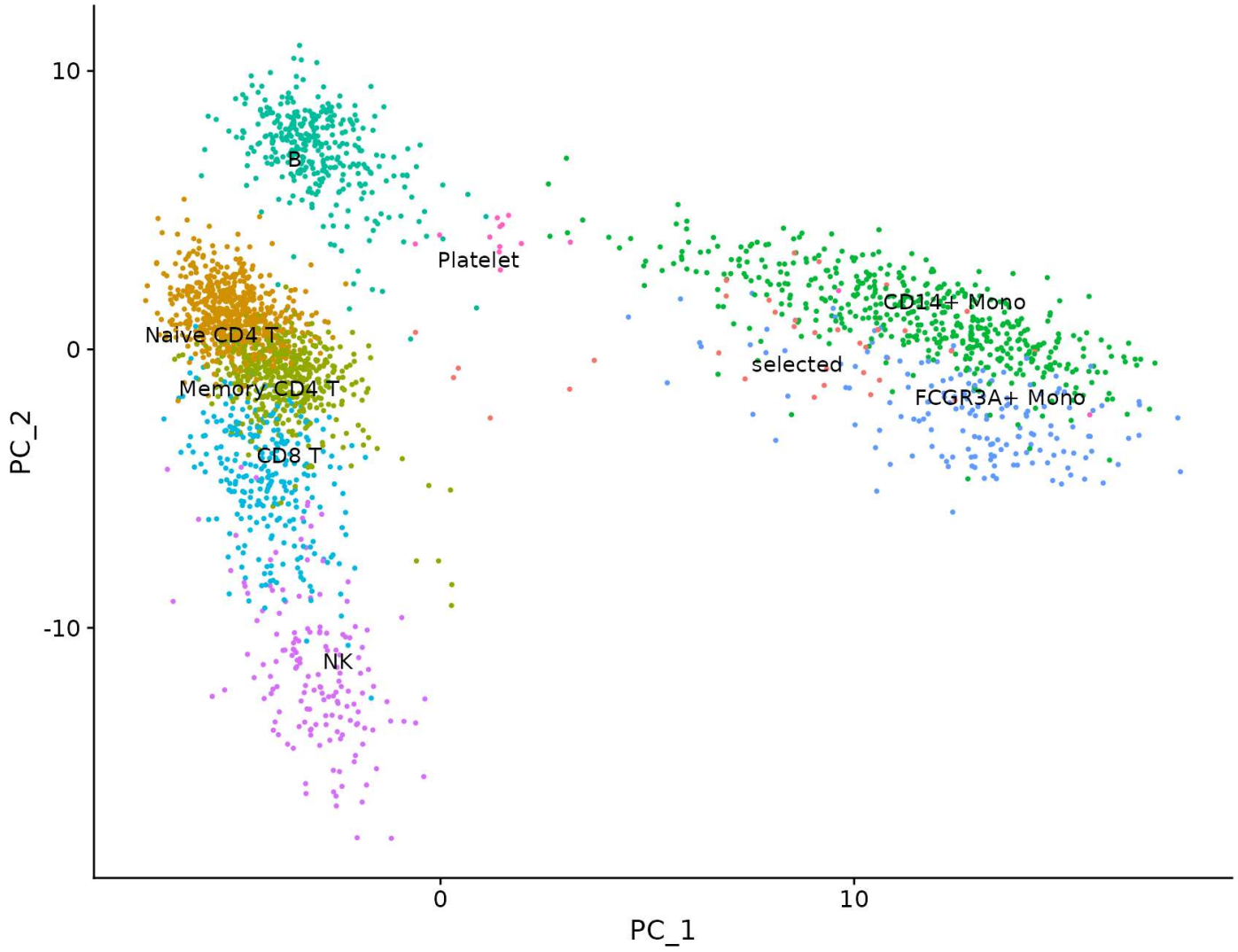
```
##                 p_val avg_log2FC pct.1 pct.2    p_val_adj
## FCER1A   3.239004e-69   6.504163 0.800 0.017 4.441970e-65
## SERPINF1 7.761413e-36   5.456560 0.457 0.013 1.064400e-31
## HLA-DQB2 1.721094e-34   4.752397 0.429 0.010 2.360309e-30
## CD1C     2.304106e-33   4.929607 0.514 0.025 3.159851e-29
## ENHO     5.099765e-32   5.076634 0.400 0.010 6.993818e-28
## ITM2C    4.299994e-29   5.660234 0.371 0.010 5.897012e-25
```

▶ Using `CellSelector` to Automatically Assign Cell Identities

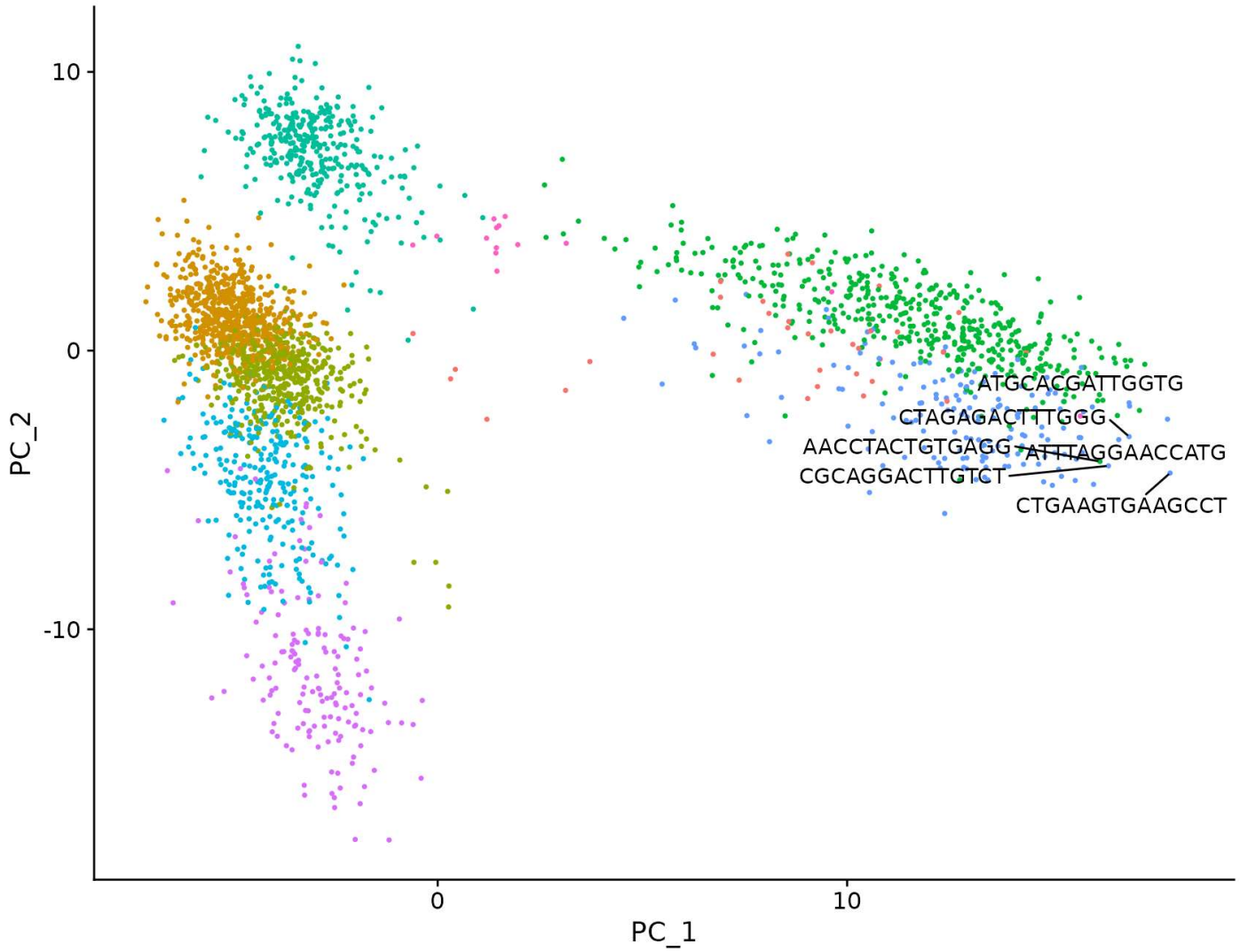
Plotting Accessories

Along with new functions add interactive functionality to plots, Seurat provides new accessory functions for manipulating and combining plots.

```
# LabelClusters and LabelPoints will label clusters (a coloring variable) or individual points
# on a ggplot2-based scatter plot
plot <- DimPlot(pbmc3k.final, reduction = "pca") + NoLegend()
LabelClusters(plot = plot, id = "ident")
```

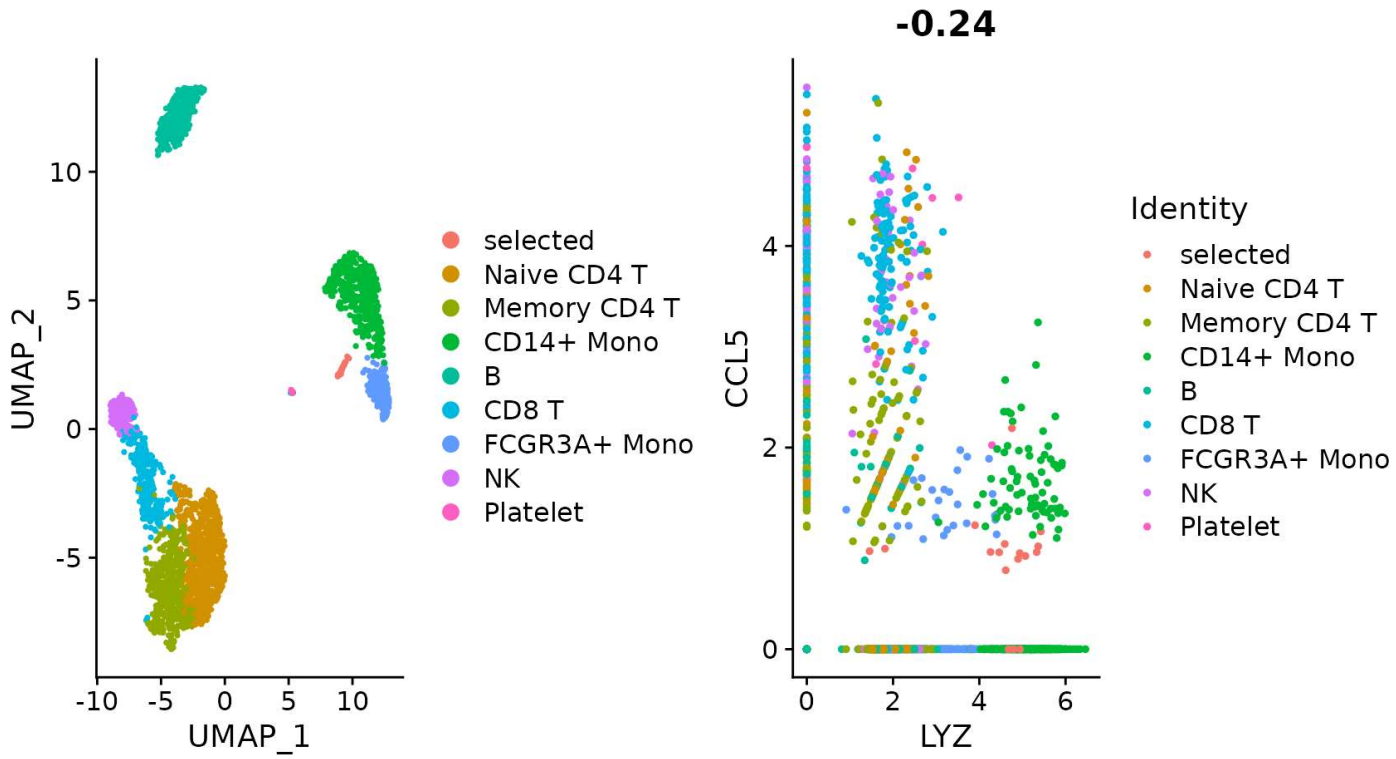



```
# Both functions support `repel`, which will intelligently stagger labels and draw connecting
# lines from the labels to the points or clusters
LabelPoints(plot = plot, points = TopCells(object = pbmc3k.final[["pca"]]), repel = TRUE)
```

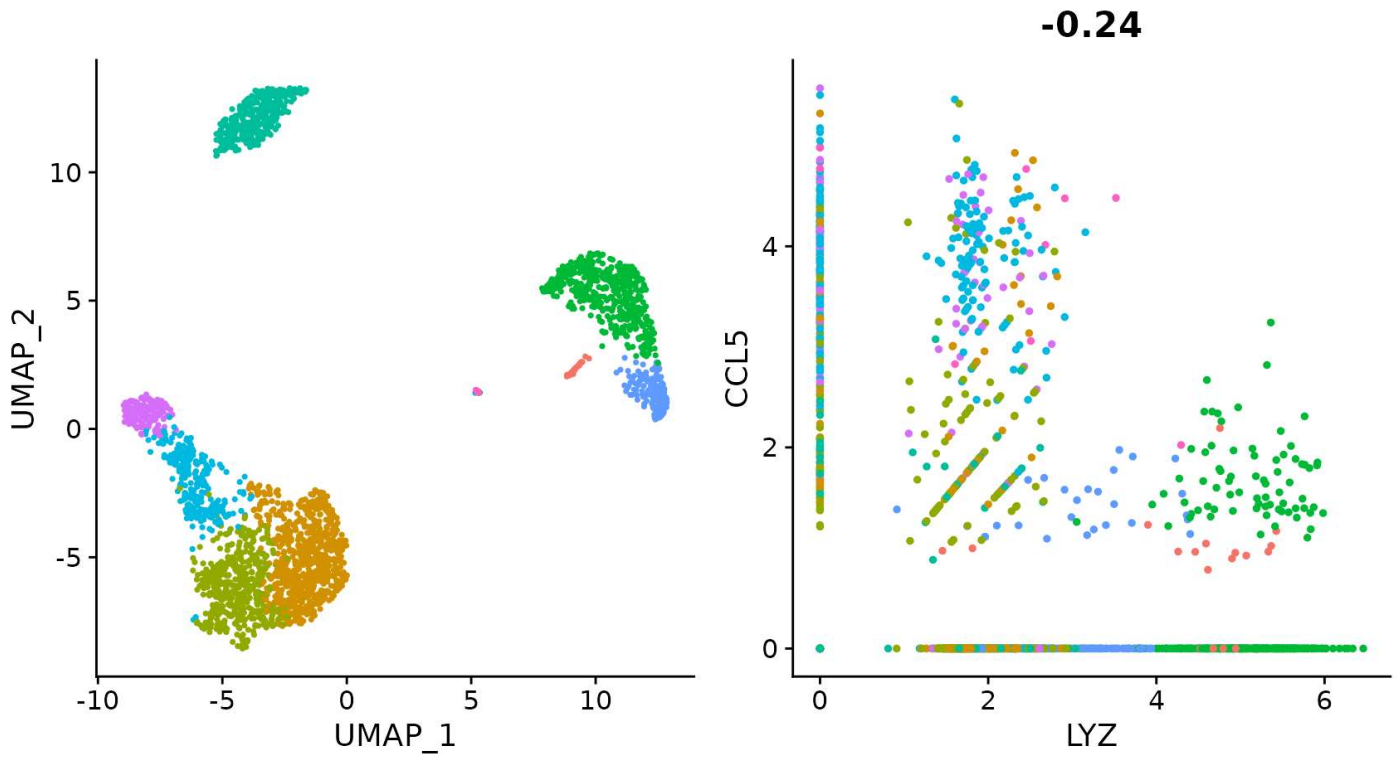


Plotting multiple plots was previously achieved with the `CombinePlot()` function. We are deprecating this functionality in favor of the patchwork (https://patchwork.data-imaginist.com/) system. Below is a brief demonstration but please see the patchwork package website here (https://patchwork.data-imaginist.com/) for more details and examples.

```
plot1 <- DimPlot(pbmc3k.final)
# Create scatter plot with the Pearson correlation value as the title
plot2 <- FeatureScatter(pbmc3k.final, feature1 = "LYZ", feature2 = "CCL5")
# Combine two plots
plot1 + plot2
```




```
# Remove the legend from all plots
(plot1 + plot2) & NoLegend()
```



▶ **Session Info**