compare_references

Emily Kibbler

2025-08-03

This script compares two publicly-available reference data sets to each other.

One is from the Satija lab.

Click here to download data

The second is a combined data set from Jin et al.

Jin, K., [. . .] Aronow, B. J. (2021). An interactive single cell web portal identifies gene and cell networks in COVID-19 host responses. iScience, 24(10), Article 103115. https://doi.org/10.1016/j.isci.2021.103115

Click here to download data

My hope is to see agreement on cell type calls between the references. If this is the case, it will provide evidence that either would be appropriate for independent analysis.

```
library(anndata)
library(Seurat)
library(MuDataSeurat)
library(BPCells)
library(reticulate)
library(Seurat)
library(Azimuth)
library(tidyverse)
library(ggpubr)
```

Define functions:

```
# This function takes a data set (Seurat object) and a list of desired features and returns a ggarrange
# Features must be something that can be retrieved by Seurat's FetchData function
# Heavy lifting performed by Seurat's VlnPlot function; formatting and tweaking by E. Kibbler

myVlnPlot <- function(dat, feats) {
  # Structure as a list even if only one feature is given
  if (length(feats) == 1) {
    feats <- c(feats)
  }
  # Initiate empty list to put the plot(s) in
  plots <- c()
  # Plot each feature separately
  for (i in 1:length(feats)) {
    # Use suppressWarnings because the VlnPlot() has a default argument which uses a deprecated argumen
    temp <- suppressWarnings(VlnPlot(dat,
                     features = feats[i])) +
      theme(axis.text.x = element_blank(),
            axis.title.x = element_blank(),
            legend.position = "none")
```

```
    plots <- c(plots, list(temp))
  }
  # Generate and return a plot of panel(s)
  res <- ggarrange(plotlist = plots, nrow = 1)
  return(res)
}
```

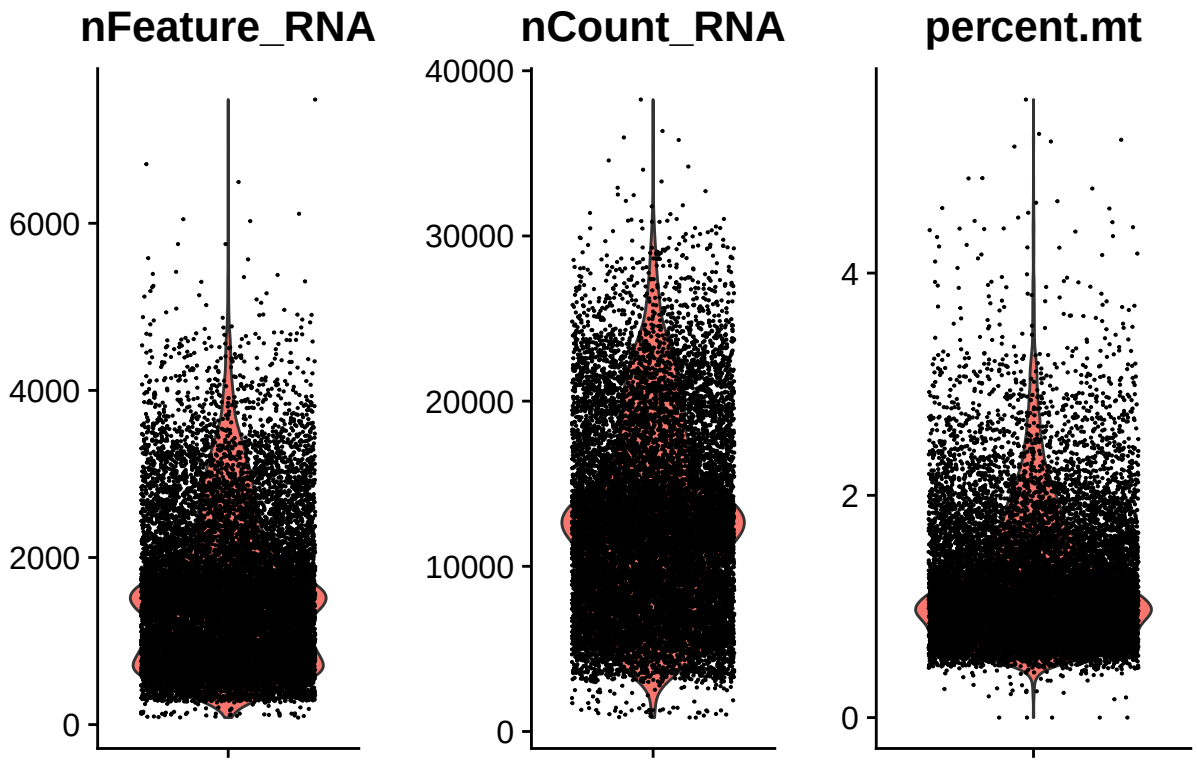
Read in Jin data:

```
path <- "./data/pbmc_reference/jin_pbmc.h5ad"

# h5ad file was formatted in Python; convert to a format available to R
numpy <- import('numpy', convert = FALSE)
anndata <- import('anndata', convert = FALSE)
scipy <- import('scipy', convert = FALSE)

adata <- py_to_r(read_h5ad(path))
mat <- as.sparse(adata$X)
mat <- t(mat)
```

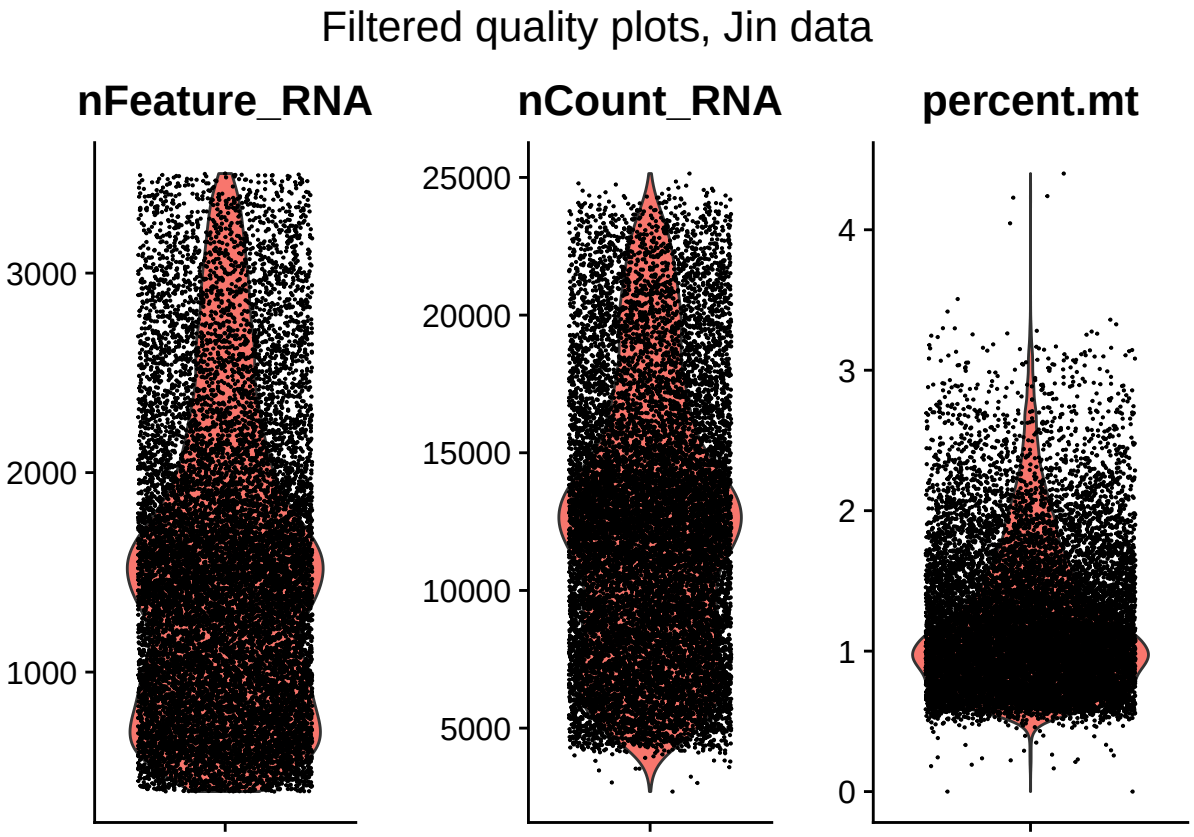
```
mat <- Azimuth:::ConvertEnsembleToSymbol(mat = mat, species = "human")
jin <- CreateSeuratObject(counts = mat,
                          meta.data = adata$obs)
```

Visualize quality of Jin data and filter:

```
# Down sample data to make it possible to process locally
jin <- subset(jin, downsample = 15000)

# Calculate mitochondrial percentage
jin[["percent.mt"]] <- PercentageFeatureSet(jin,
                                            pattern = "(?i)^MT-")
# Plot quality data
myVlnPlot(dat = jin,
        feats = c("nFeature_RNA",
                  "nCount_RNA",
                  "percent.mt")) %>%
  annotate_figure(top = text_grob("Unfiltered quality plots, Jin data",
                                  size = 16))
```



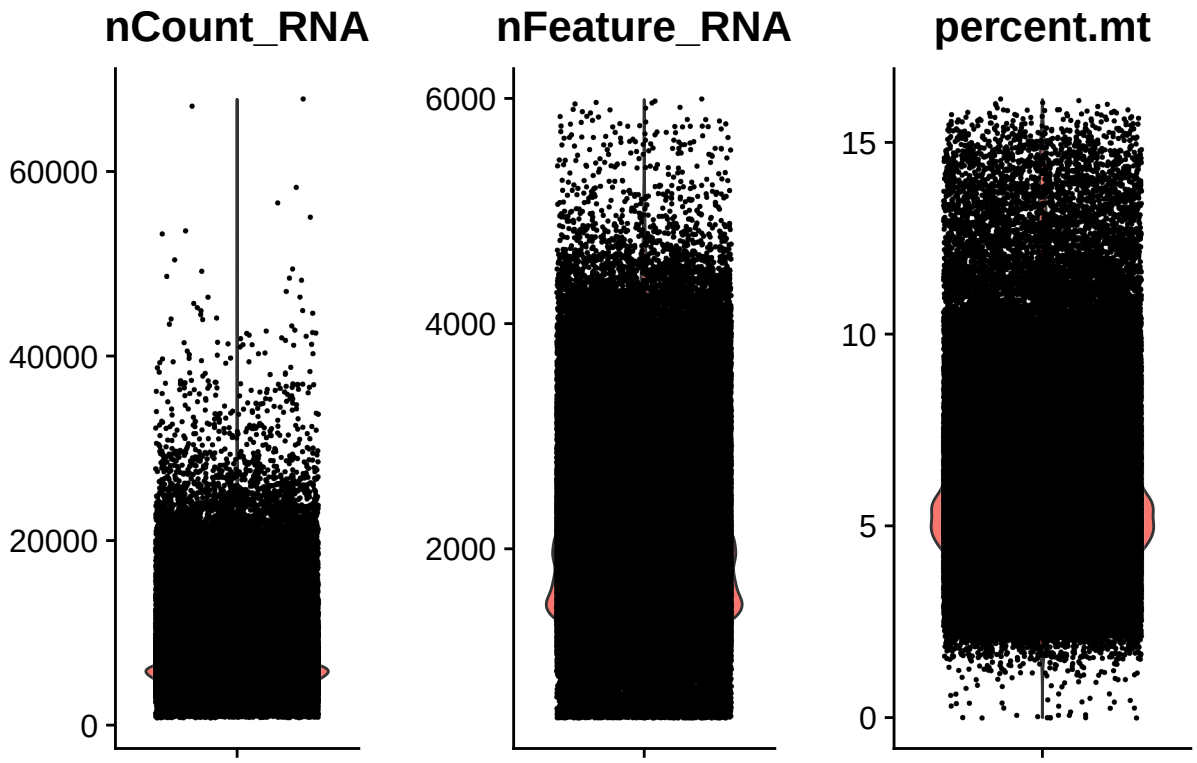

```
# Remove cells that do not meet criteria
jin <- subset(jin,
              subset = nFeature_RNA > 400 &
                nFeature_RNA < 3500 &
                percent.mt < 5)
# Plot filtered data
myVlnPlot(dat = jin,
        feats = c("nFeature_RNA",
                  "nCount_RNA",
                  "percent.mt")) %>%
  annotate_figure(top = text_grob("Filtered quality plots, Jin data",
                                  size = 16))
```



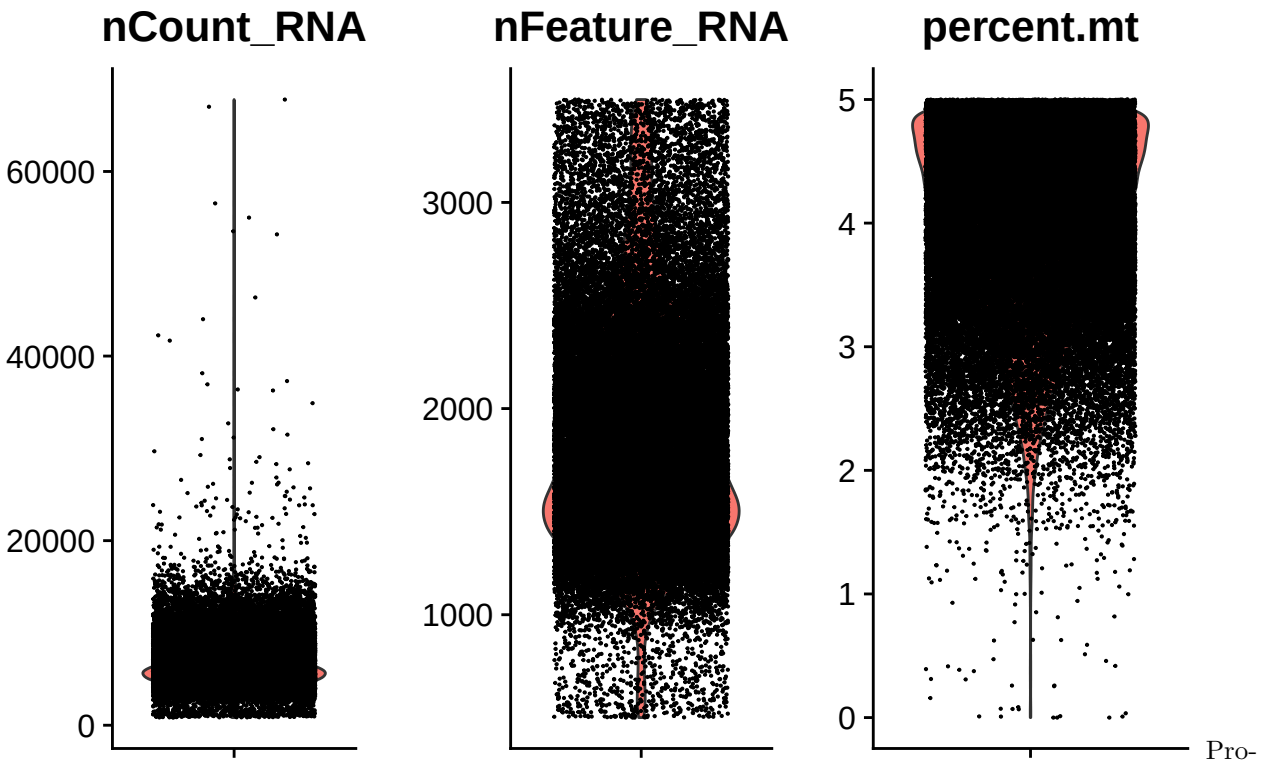
Read in and filter Satija data:

```
reference <- readRDS("./data/pbmc_reference/pbmc_multimodal_2023.rds")
# Down sample
reference <- subset(reference, downsample = 15000)
# Calculate mitochondrial DNA percentage
reference[["percent.mt"]] <- PercentageFeatureSet(reference,
                                                  pattern = "(?i)^MT-")
# Clear cluster identities before plotting to visualize data holistically
Idents(reference) <- "all"
reference %>% myVlnPlot(c("nCount_RNA",
                          "nFeature_RNA",
                          "percent.mt")) %>%
  annotate_figure(top = text_grob("Unfiltered quality plots, Satija data",
                                  size = 16))
```

```
## Rasterizing points since number of points exceeds 100,000.
## To disable this behavior set `raster=FALSE`
## Rasterizing points since number of points exceeds 100,000.
## To disable this behavior set `raster=FALSE`
## Rasterizing points since number of points exceeds 100,000.
## To disable this behavior set `raster=FALSE`
```




```
reference <- subset(reference,
                    nFeature_RNA > 400 &
                      nFeature_RNA < 3500 &
                      percent.mt < 5)

reference %>% myVlnPlot(c("nCount_RNA",
                          "nFeature_RNA",
                          "percent.mt")) %>%
  annotate_figure(top = text_grob("Filtered quality plots, Satija data",
                                  size = 16))
```



Process Jin data for comparison:

```
jin <- NormalizeData(jin, verbose = FALSE)
all.genes <- rownames(jin)
jin <- ScaleData(jin, features = all.genes, verbose = FALSE)
jin <- FindVariableFeatures(jin, verbose = FALSE)

jin <- RunPCA(jin,
              features = VariableFeatures(object = jin),
              verbose = FALSE)
jin <- SCTransform(jin, verbose = FALSE)
```

```
## Warning: The `slot` argument of `GetAssayData()` is deprecated as of SeuratObject 5.0.0.
## i Please use the `layer` argument instead.
## i The deprecated feature was likely used in the Seurat package.
##   Please report the issue at <https://github.com/satijalab/seurat/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: The `slot` argument of `SetAssayData()` is deprecated as of SeuratObject 5.0.0.
## i Please use the `layer` argument instead.
## i The deprecated feature was likely used in the Seurat package.
##   Please report the issue at <https://github.com/satijalab/seurat/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
jin <- RunUMAP(jin, dims = 1:30, return.model = TRUE, verbose = FALSE)
```

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

Map Jin data onto Satija data:

```
anchors <- FindTransferAnchors(
  reference = reference,
  query = jin,
  reference.reduction = "pca",
  normalization.method = "SCT",
  dims = 1:50,
  verbose = FALSE
)
```

```
## Only one SCT model detected; no need to select.
```

```
jin <- MapQuery(
  anchorset = anchors,
  query = jin,
  reference = reference,
  refdata = list(
    celltype.l1 = "celltype.l1",
    celltype.l2 = "celltype.l2",
    celltype.l3 = "celltype.l3"
  ),
  reduction.model = "wnn.umap",
  verbose = FALSE
)
```

```
## Warning: Layer counts isn't present in the assay object; returning NULL
```

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

```
## Warning: Layer counts isn't present in the assay object; returning NULL
```

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

```
## Warning: Layer counts isn't present in the assay object; returning NULL
## Warning: Layer counts isn't present in the assay object; returning NULL
## Warning: Layer counts isn't present in the assay object; returning NULL
```

```
## Computing nearest neighbors
```

```
## Running UMAP projection
```

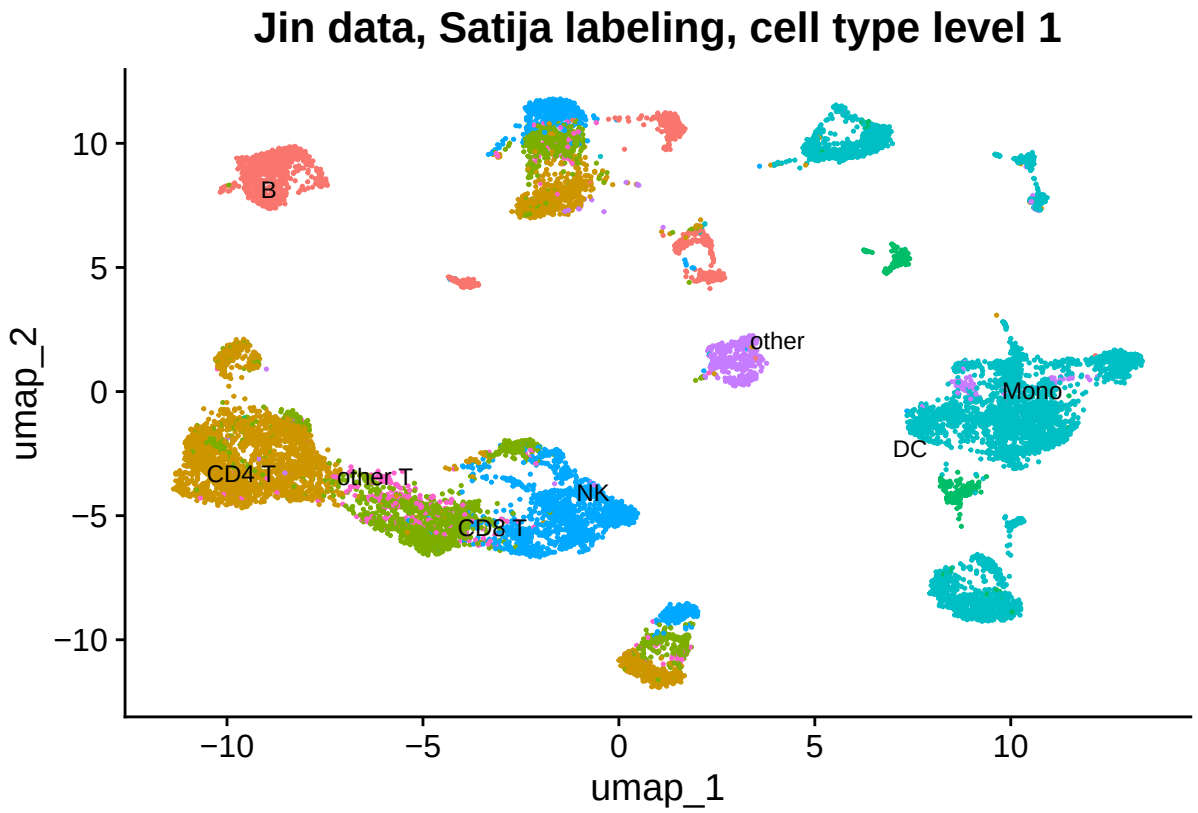
```
## 17:45:30 Read 14005 rows
```

```
## 17:45:30 Processing block 1 of 1
```

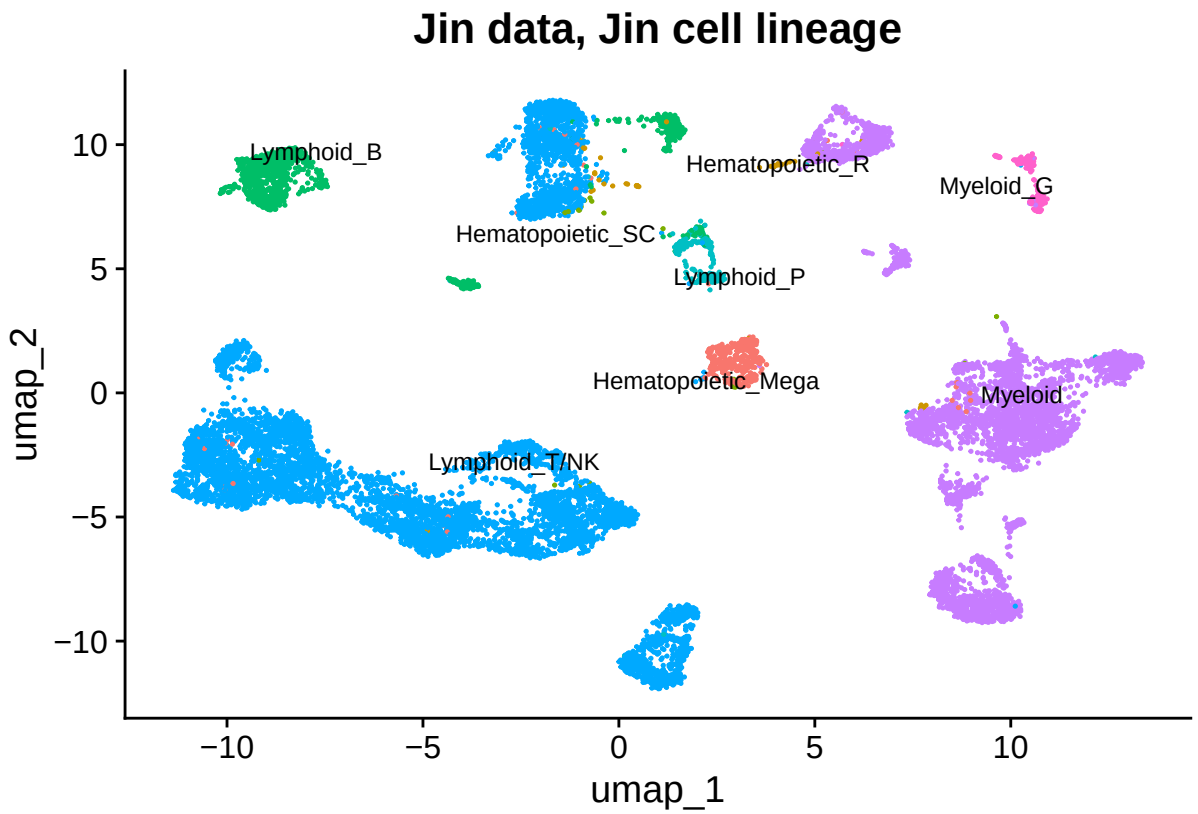
```
## 17:45:30 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 15
## 17:45:30 Initializing by weighted average of neighbor coordinates using 1 thread
## 17:45:30 Commencing optimization for 67 epochs, with 210075 positive edges
## 17:45:31 Finished
```

Plot outcome:

```
DimPlot(jin,
        reduction = "umap",
        group.by = "predicted.celltype.l1",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Jin data, Satija labeling, cell type level 1")
```

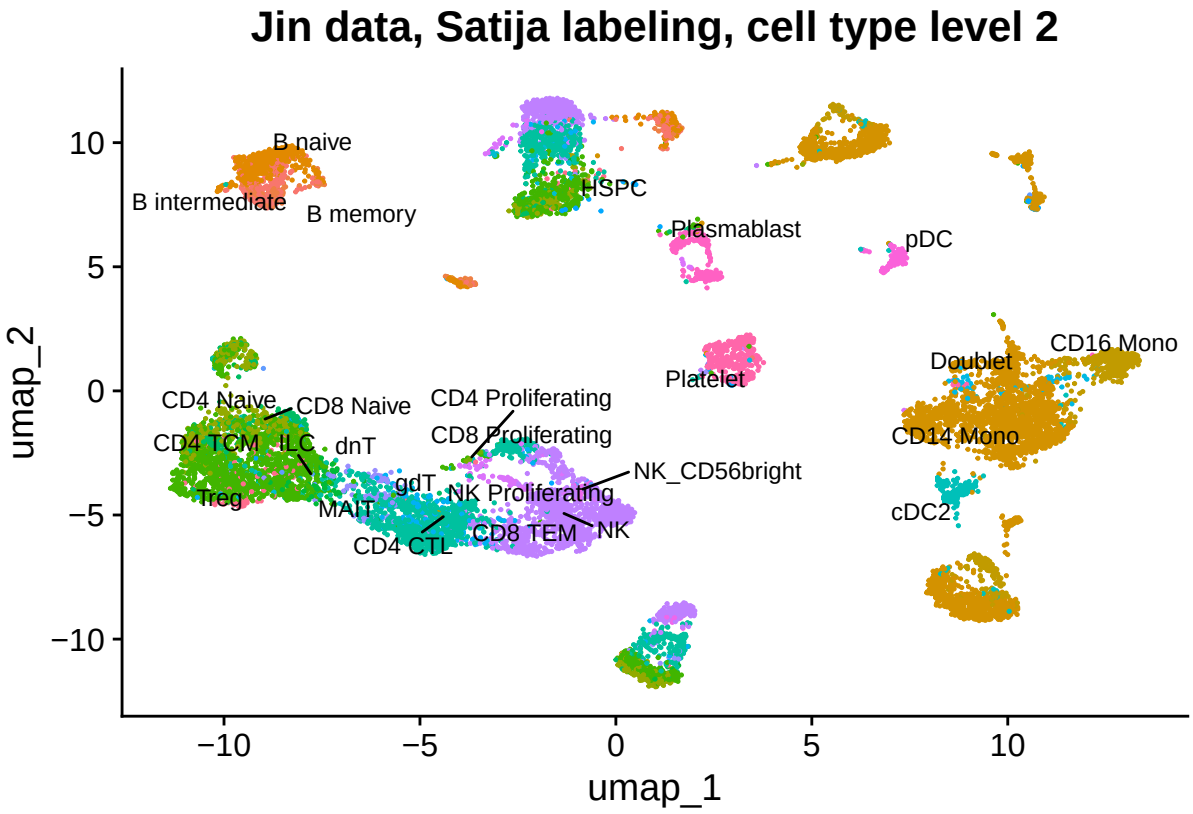



```
DimPlot(jin,
        reduction = "umap",
        group.by = "Lineage",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Jin data, Jin cell lineage")
```



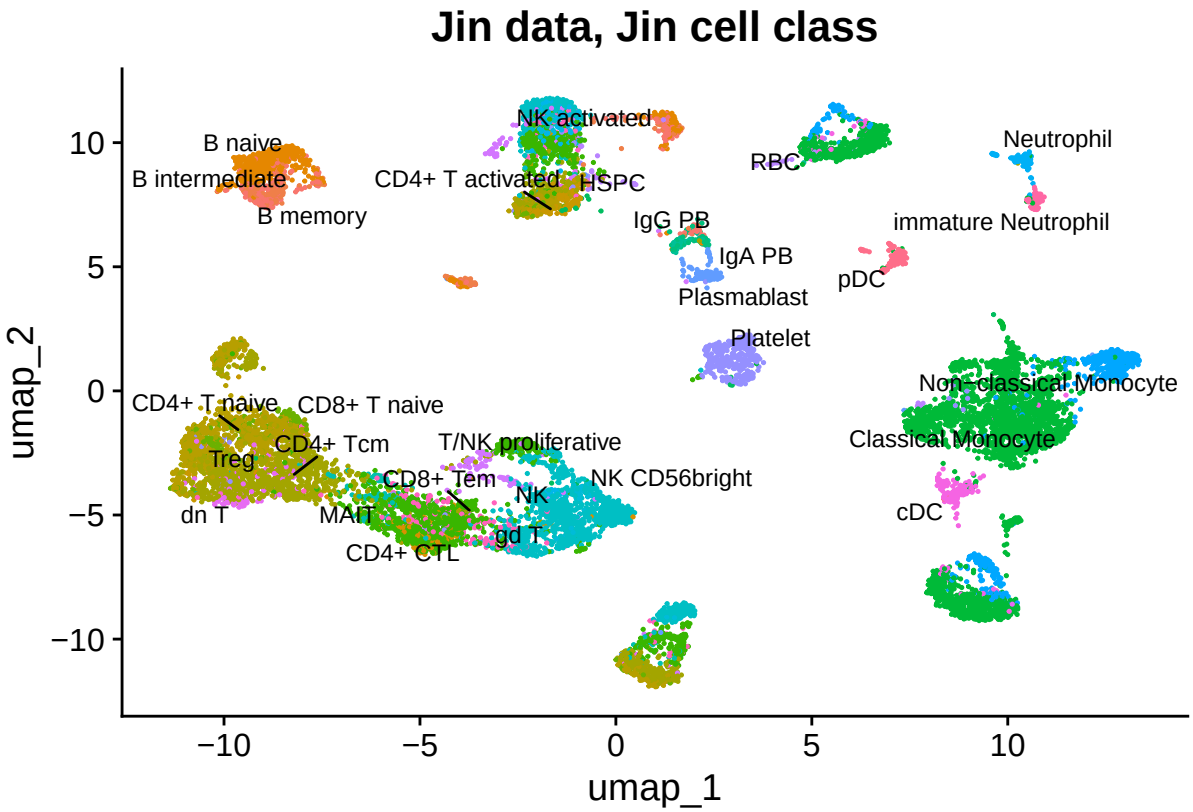

```
DimPlot(jin,
        reduction = "umap",
        group.by = "predicted.celltype.l2",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Jin data, Satija labeling, cell type level 2")
```

```
## Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



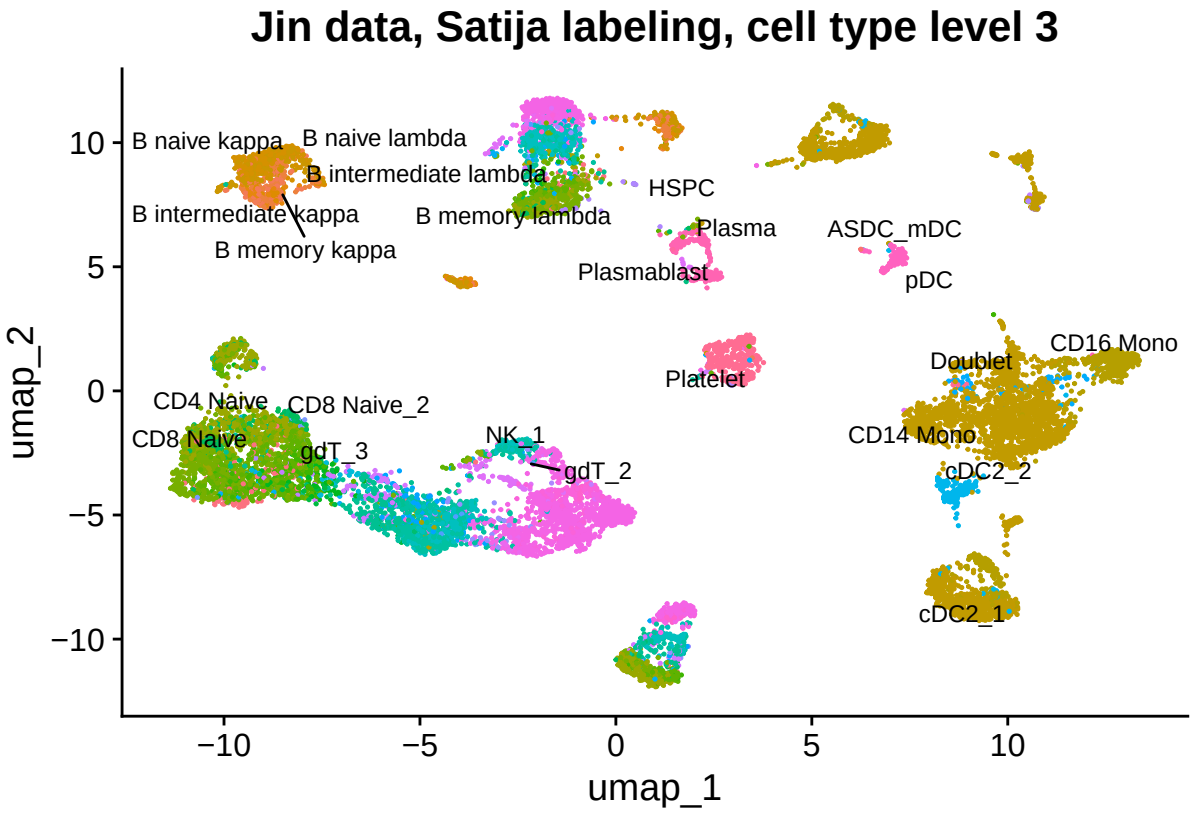

```
DimPlot(jin,
        reduction = "umap",
        group.by = "Cell.class",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Jin data, Jin cell class")
```

```
## Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```

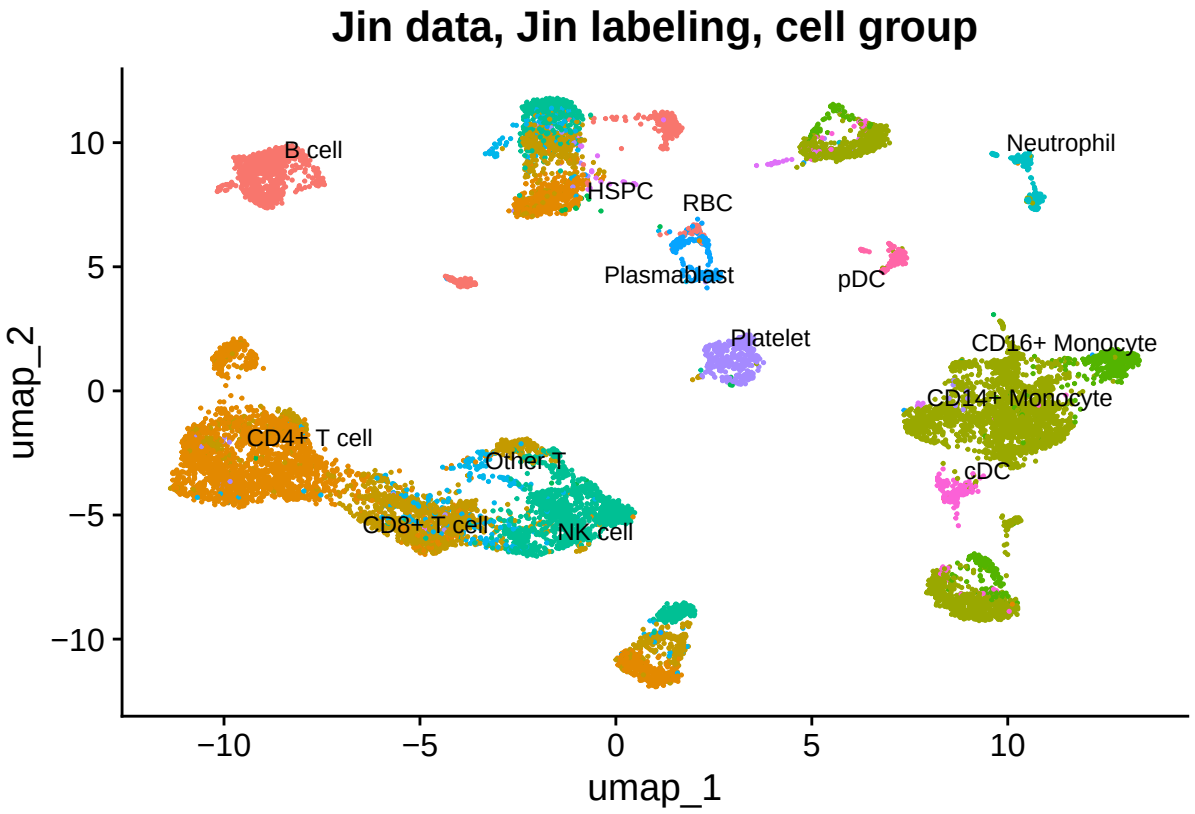



```
DimPlot(jin,
        reduction = "umap",
        group.by = "predicted.celltype.l3",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Jin data, Satija labeling, cell type level 3")
```

```
## Warning: ggrepel: 24 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```




```
DimPlot(jin,
        reduction = "umap",
        group.by = "Cell.group",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Jin data, Jin labeling, cell group")
```



We are not seeing large discrepancies in cell type classification. This lends support to the hypothesis that either reference would be appropriate to use for further analysis. It also adds validation to this pipeline as a method for cell type classification.

Satija data processing:

```
reference <- NormalizeData(reference)
reference <- FindVariableFeatures(reference)
reference <- ScaleData(reference)
```

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

```
anchors <- FindTransferAnchors(
  reference = jin,
  query = reference,
  reference.reduction = "pca",
  normalization.method = "SCT",
  dims = 1:50,
  recompute.residuals = FALSE,
  verbose = FALSE
)
```

```
## Only one SCT model detected; no need to select.
```

```
reference <- MapQuery(
  anchorset = anchors,
  query = reference,
  reference = jin,
  refdata = list(
    Lineage = "Lineage",
    Cell.group = "Cell.group",
```

```
    Cell.class = "Cell.class"
  ),
  reduction.model = "umap",
  verbose = FALSE
)
```

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

```
## Warning: Layer counts isn't present in the assay object; returning NULL
## Warning: Layer counts isn't present in the assay object; returning NULL
## Warning: Layer counts isn't present in the assay object; returning NULL
## Warning: Layer counts isn't present in the assay object; returning NULL
## Warning: Layer counts isn't present in the assay object; returning NULL
```

```
## Computing nearest neighbors
```

```
## Running UMAP projection
```

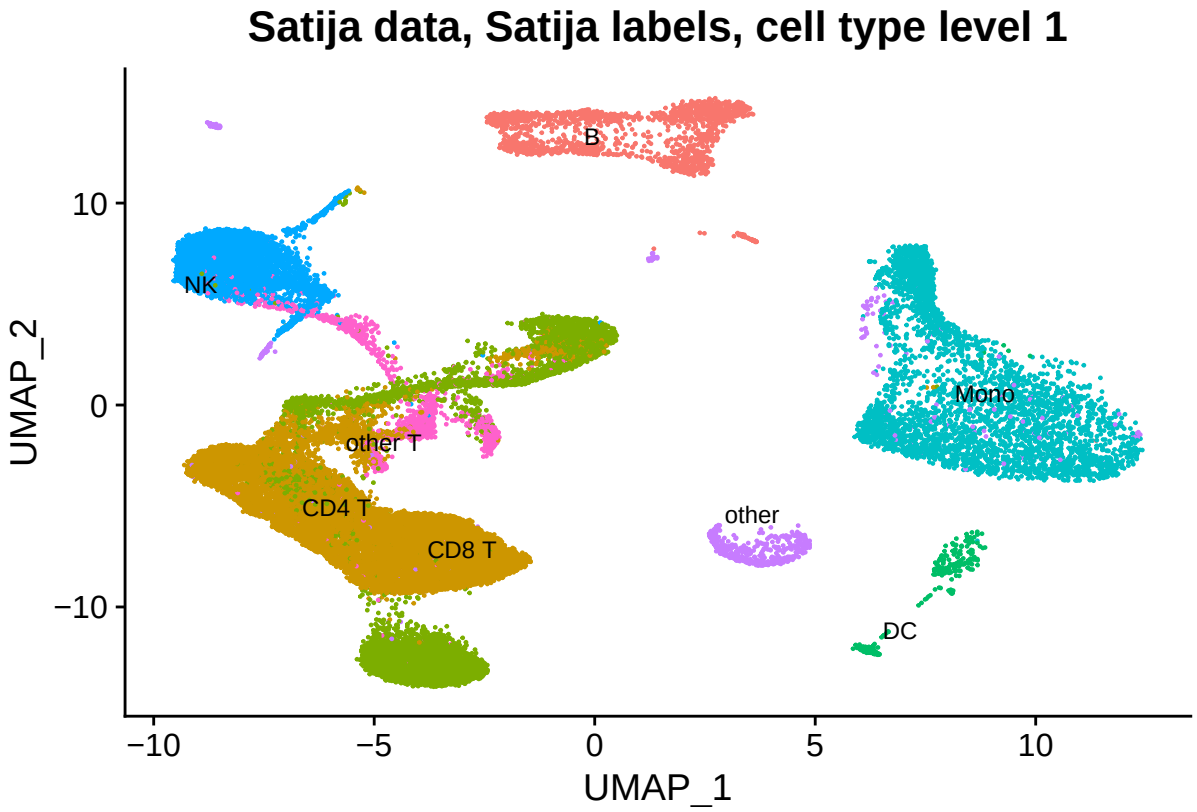
```
## 17:47:52 Read 45947 rows
```

```
## 17:47:52 Processing block 1 of 1
```

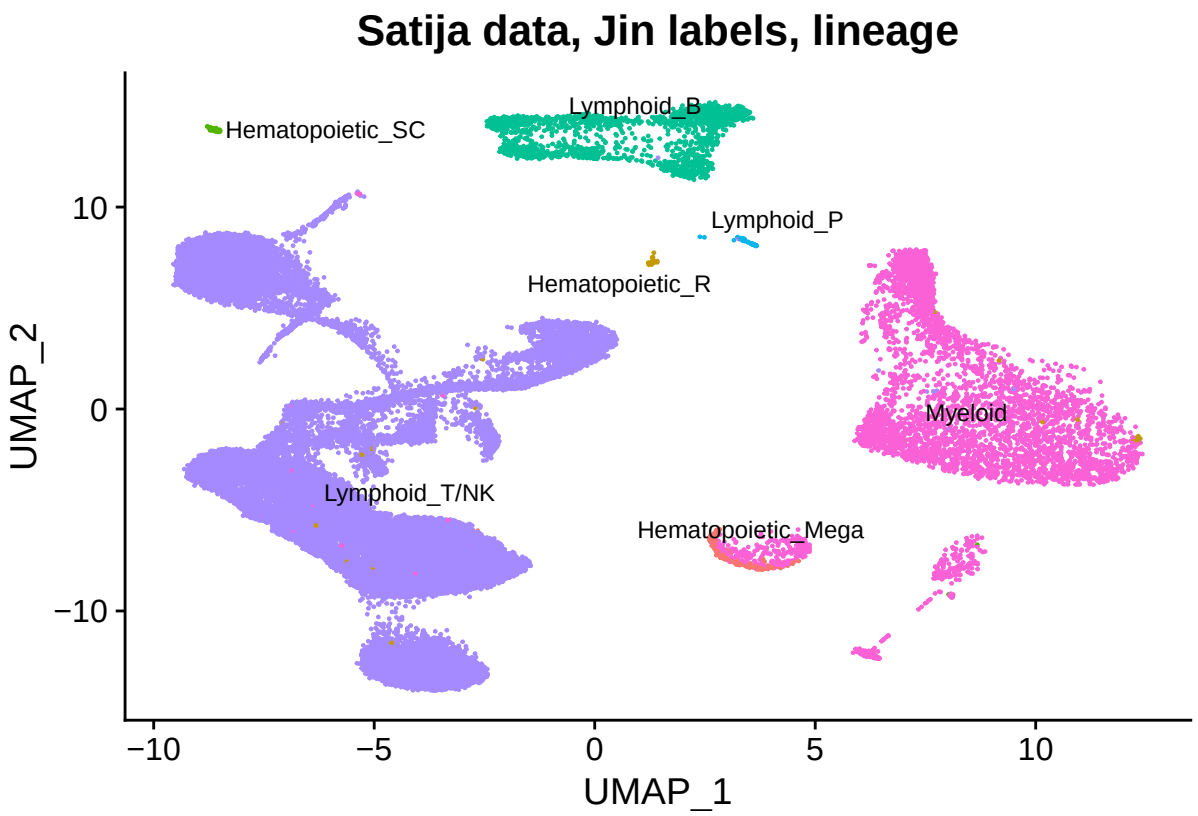
```
## 17:47:52 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 30
## 17:47:52 Initializing by weighted average of neighbor coordinates using 1 thread
## 17:47:52 Commencing optimization for 67 epochs, with 1378410 positive edges
## 17:47:56 Finished
```

Plot outcomes:

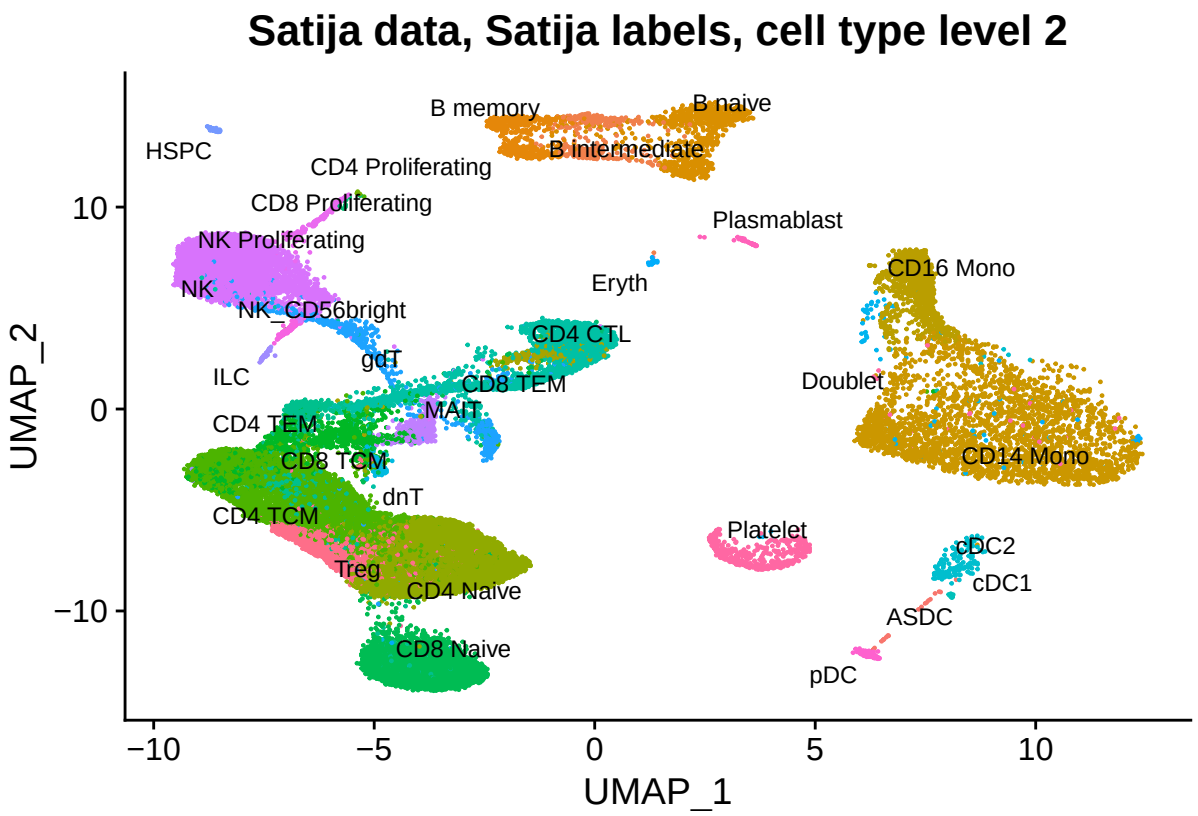
```
DimPlot(reference,
        reduction = "umap",
        group.by = "celltype.l1",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Satija data, Satija labels, cell type level 1")
```



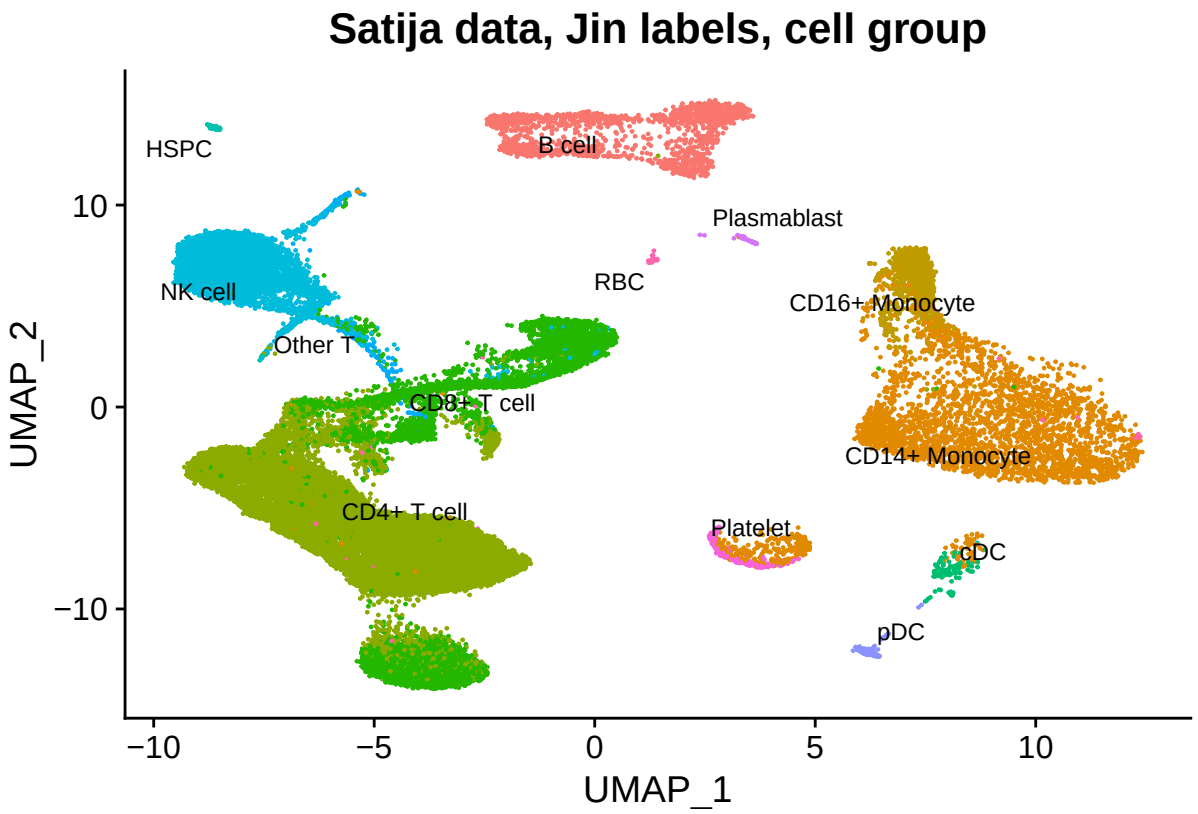

```
DimPlot(reference,
        reduction = "umap",
        group.by = "predicted.Lineage",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Satija data, Jin labels, lineage")
```




```
DimPlot(reference,
        reduction = "umap",
        group.by = "celltype.l2",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Satija data, Satija labels, cell type level 2")
```

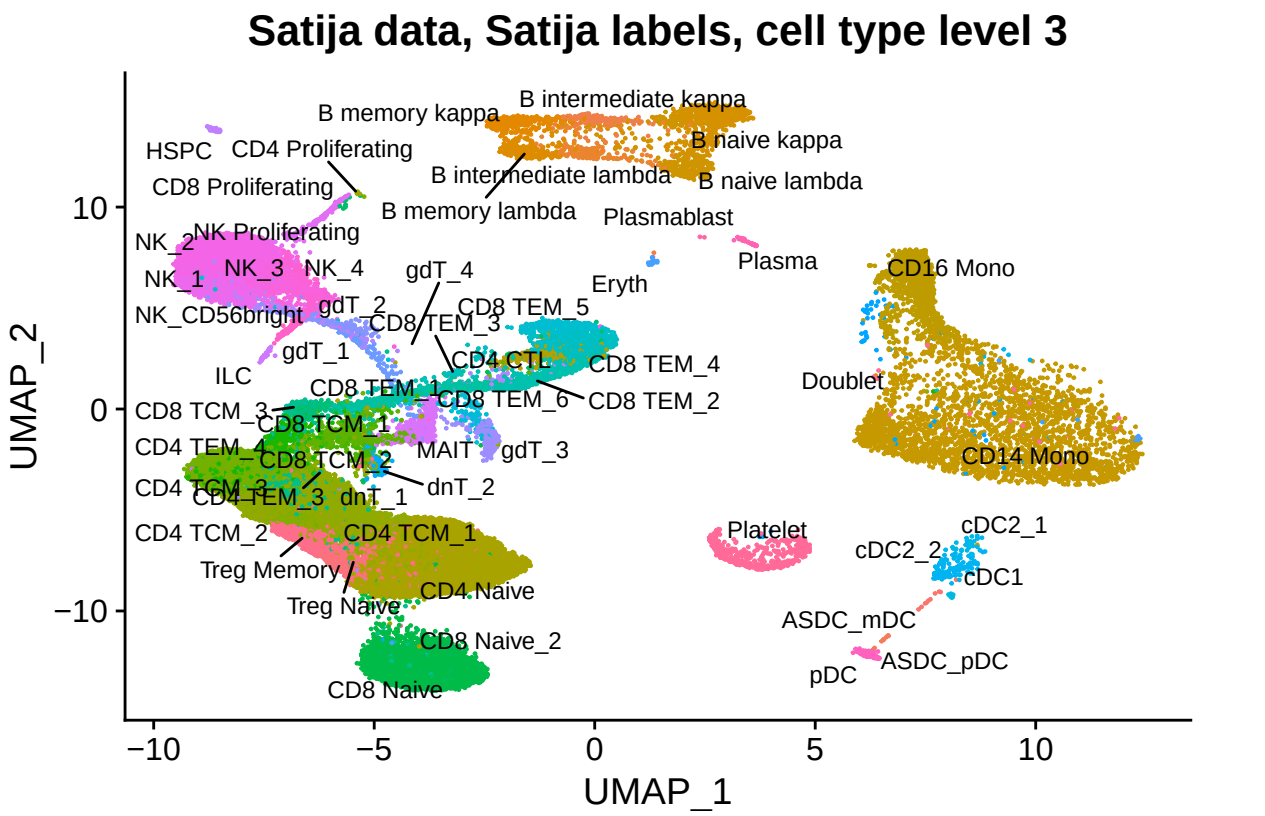



```
DimPlot(reference,
        reduction = "umap",
        group.by = "predicted.Cell.group",
        label = TRUE,
        label.size = 3,
        repel = TRUE) +
  NoLegend() +
  ggtitle("Satija data, Jin labels, cell group")
```

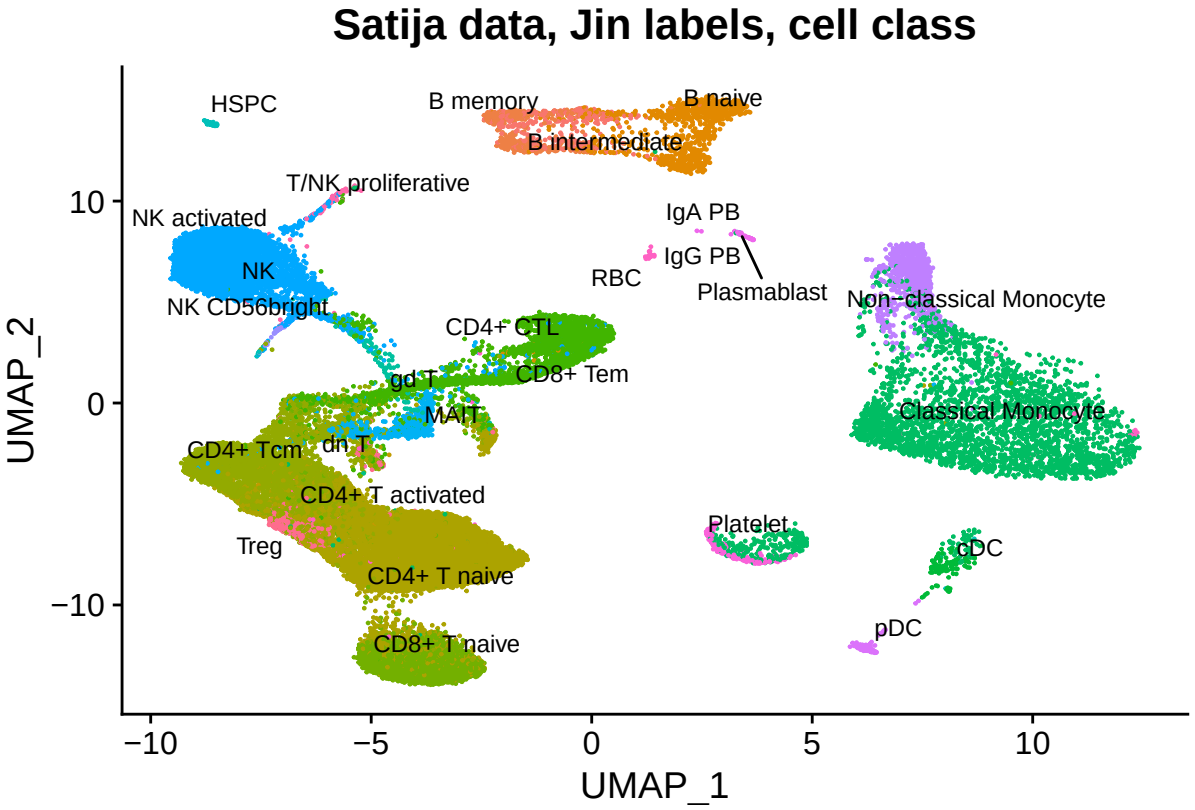



```
DimPlot(reference,
        reduction = "umap",
        group.by = "celltype.l3",
        label = TRUE,
        label.size = 3,
        repel = TRUE) + NoLegend() +
  ggtitle("Satija data, Satija labels, cell type level 3")
```

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




```
DimPlot(reference,
        reduction = "umap",
        group.by = "predicted.Cell.class",
        label = TRUE,
        label.size = 3,
        repel = TRUE) + NoLegend() +
  ggtitle("Satija data, Jin labels, cell class")
```



Again, we see good agreement.

Discrepancies arise in "leveling" these classifications. PBMCs are complicated cells with a branching tree of differentiation, so determining parent/child/subclass designations is not simple. However, when evaluating the classification as whole, a high-level consensus is reached.

Generally, we can say that the pipeline and reference data sources have been sufficiently validated for my use at this time.