Demonstration of the guided CellRef construction using human lung endothelial data

Minzhe Guo

Compiled: 2022-06-16

In this vignette, we demonstrate the guided CellRef construction pipeline using a subset of our collected human lung scRNA-seq data, which contains 21,971 human lung endothelial cells predicted in three studies (Adams et al., Sci. Adv. 2020, Habermann et al., Sci. Adv. 2020, Travaglini et al., Nature 2020). The runtime of this demo is about 30 minutes.

```
library(CellRef)
library(Seurat)
library(monocle3)
library(SingleR)
library(SingleCellExperiment)
library(harmony)
library(RobustRankAggreg)
library(dplyr)
library(reshape2)
library(ggplot2)
library(pheatmap)

options(future.globals.maxSize = 1280000 * 1024^2)

start = Sys.time()
```

Currently, we utilize Seurat (v4) object as main data structure for storage and computation. First, let's create a Seurat object using the UMI count matrix (lung_endo.counts.rds) and cell information (lung_endo.cells.rds) of the demo data, which can be downloaded from the data folder of this CellRef github.

```
counts = readRDS(file = "./lung_endo.counts.rds")
cells = readRDS(file = "./lung_endo.cells.rds")

obj <- CreateSeuratObject(counts = counts, project = "lung_endo_demo", meta.data = cells)
obj <- NormalizeData(obj)

print(obj)
```

```
## An object of class Seurat
## 32284 features across 21971 samples within 1 assay
## Active assay: RNA (32284 features, 0 variable features)
```

Download the CellRef cell type dictionary for lung endothelial cells (Lung_endothelial_ctd.rds) from the data folder of this CellRef github and load the dictionary into R. We assume that the dictionary is a

data.frame that contains at least three columns: CellType, Marker, and MarkType (p - positive marker, n - negative marker).

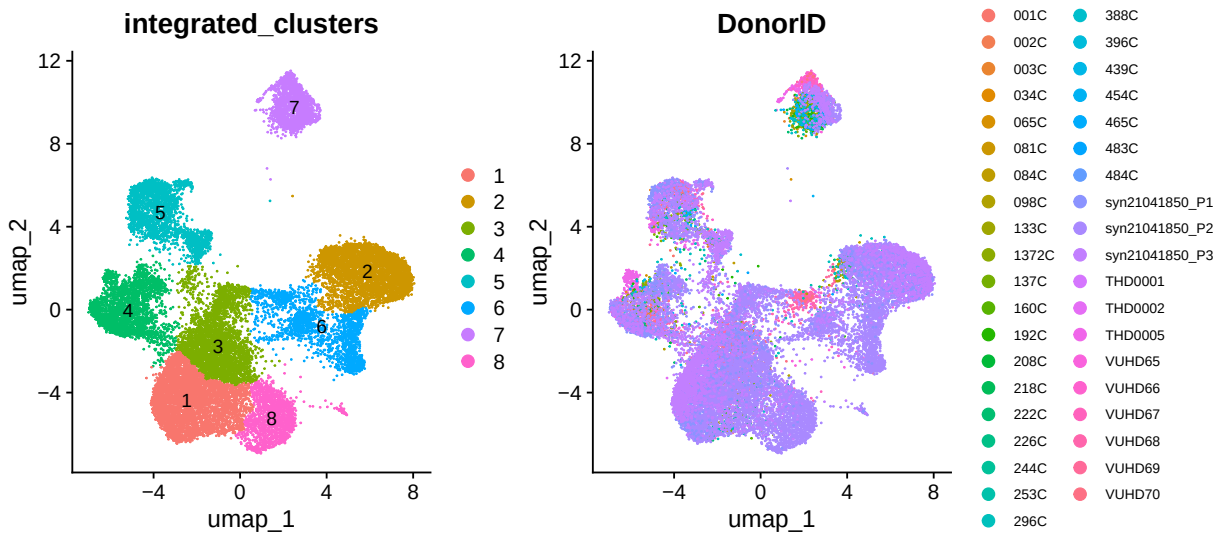
```
ctd = readRDS(file = "./lung_endothelial_ctd.rds")

print(ctd)
```

```
##    CellType  Marker MarkerType
## 1      CAP1    IL7R          p
## 2      CAP1    FCN3          p
## 3      CAP1 GPIHBP1          p
## 4      CAP2    APLN          p
## 5      CAP2   EDNRB          p
## 6      CAP2    HPGD          p
## 7      CAP2     CA4          p
## 8       AEC    DKK2          p
## 9       AEC    GJA5          p
## 10      AEC     BMX          p
## 11      VEC   ACKR1          p
## 12      VEC   EPHB4          p
## 13      VEC   HDAC9          p
## 14      LEC   PROX1          p
## 15      LEC   MMRN1          p
## 16      LEC   CCL21          p
## 17      LEC   LYVE1          p
## 18     SVEC   ACKR1          p
## 19     SVEC COL15A1          p
## 20     SVEC   ABCB1          p
## 21     SVEC    VWA1          p
```

Step 1: batch correction. By default, we used the mutual nearest matching (MNN, Haghverdi et al., 2018) in Monocle 3 (Cao et al., Nature 2019). Alternative options include: 'Seurat4-rpca' - Seurat4's reciprocal PCA based integration (Hao and Hao et al., 2021) and 'Harmony' (Korsunsky et al., Nature Methods 2019).

By setting "do.clustering=T", we also identified cell clusters after batch correction. By default, the clusters will be stored in a meta.data column named "integrated_clusters"

```
# batch correction of data from different donors
obj = doDataIntegration(obj, integration.batch = "DonorID", method = "Monocle3-mnn", npcs = 200,
    do.clustering = T)

g1 = DimPlot(obj, reduction = "umap", group.by = "integrated_clusters", label = T)
g2 = DimPlot(obj, reduction = "umap", group.by = "DonorID", label = F)
g2 = g2 + theme(legend.text = element_text(size = 8))
g1 + g2
```



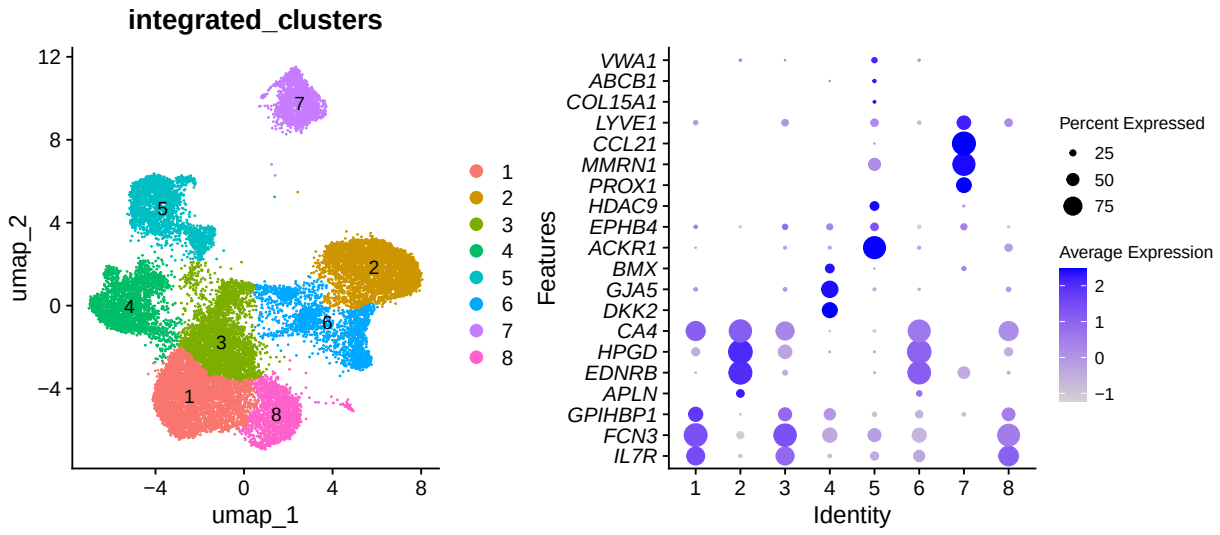
Step2: candidate cell clusters. Using the integrated data and cell clusters, we identify the candidate cell clusters for each cell type in the dictionary.

```
candidate_clusters = findCandidateClusters(obj, ctd, verbose = F)

candidate_clusters
```

```
##   Cluster CellType
## 1       4      AEC
## 2       1     CAP1
## 3       3     CAP1
## 4       2     CAP2
## 5       7      LEC
## 6       5     SVEC
## 7       5      VEC
```

We suggest double checking and curating the identified candidate cell clusters using expression of marker genes. For example, using dotplot, we can validate that SVEC and VEC markers both selectively expressed in cluster 5, consistent with candidate cell cluster results. Based on cell clusters in the UMAP and expression of CAP1 and CAP2 markers, we can add cluster 8 for CAP1 and cluster 6 for CAP2.

```
g1 = DimPlot(obj, reduction = "umap", group.by = "integrated_clusters", label = T)

g = DotPlot(obj, assay = "RNA", features = unique(as.character(ctd$Marker)), dot.min = 0.05, group.by =
g = g + coord_flip()
g = g + theme(axis.text.y = element_text(face = "italic"), legend.title = element_text(size = 10),
    legend.text = element_text(size = 10))

g1 + g
```



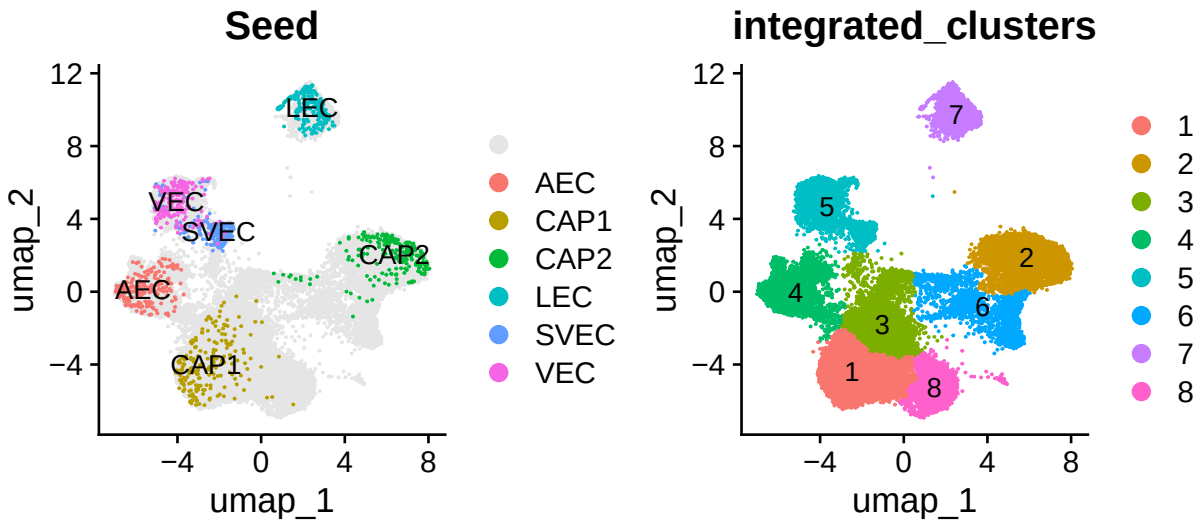

```
# Based marker exprssion and umap of cell clusters, we can curate the candidate cell clusters,
# e.g., adding cluster 8 for CAP1 and cluster 6 for CAP2.
candidate_clusters <- rbind(candidate_clusters, c(8, "CAP1"), c(6, "CAP2"))
```

Step 3: using the cell type dictionary and the candidate cell clusters, we used marker based single cell ranking to identify seed cells for each cell type.

```
obj_seed = findSeedCells(obj, ctd, candidate_clusters, score.thresh = Inf, seed.n.min = 5, seed.n.max =
    verbose = F)
```

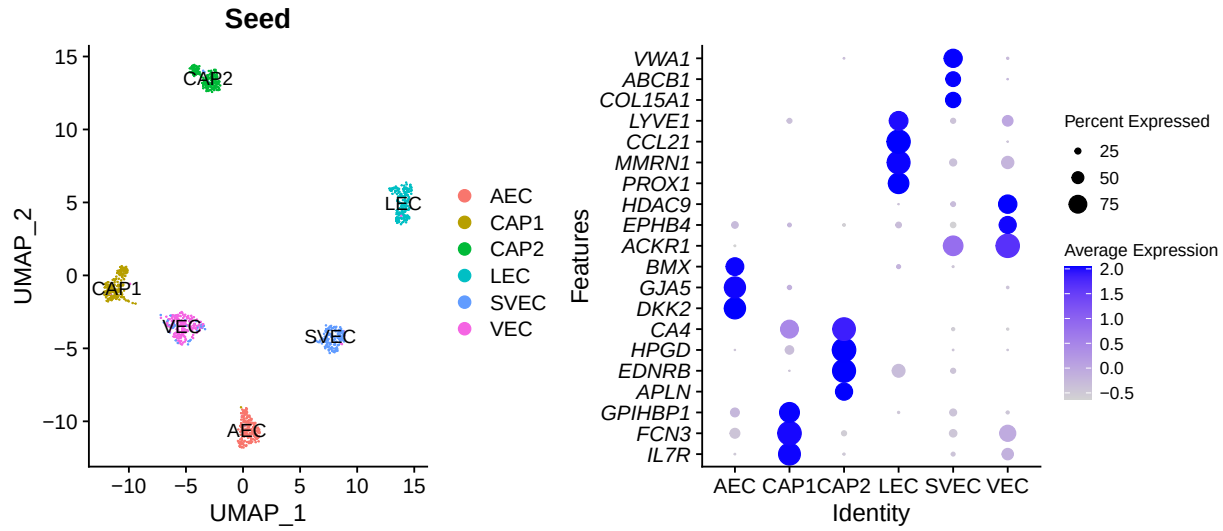
Let's visualize the identified seed cells in the original UMAP. The seed cells of the same cell type are close to each other and separate from seed cells of other cell types. We can see the clear separation of the seed cells of SVEC and VEC within cluster 5.

```
obj1 = obj
obj1@meta.data$Seed = " "
obj1@meta.data[rownames(obj_seed@meta.data), "Seed"] = as.character(obj_seed@meta.data$Seed)

g = DimPlot(obj1, reduction = "umap", group.by = "Seed", label = T, order = T, label.size = 4, cols = c
    (scales::hue_pal())(6)))
g1 = DimPlot(obj, reduction = "umap", group.by = "integrated_clusters", label = T)

g + g1
```



The seed cells will be used for automated cell type identification. To prepare for this, we integrated seed cells from different studies using Seurat's RPCA pipeline. UMAP visualization shows the separation of seed cells of different cell types.

```
obj_seed_rpca = doDataIntegration(obj_seed, integration.batch = "Dataset", method = "Seurat4-rpca",
    npcs = 200, do.clustering = F, rpca.prune_anchors = F, verbose = F)

g1 = DimPlot(obj_seed_rpca, reduction = "umap", pt.size = 0.001, group.by = "Seed", label = T)

g = DotPlot(obj_seed_rpca, assay = "RNA", features = unique(as.character(ctd$Marker)), dot.min = 0.05,
    group.by = "Seed")
g = g + coord_flip()
g = g + theme(axis.text.y = element_text(face = "italic"), legend.title = element_text(size = 10),
    legend.text = element_text(size = 10))

g1 + g
```

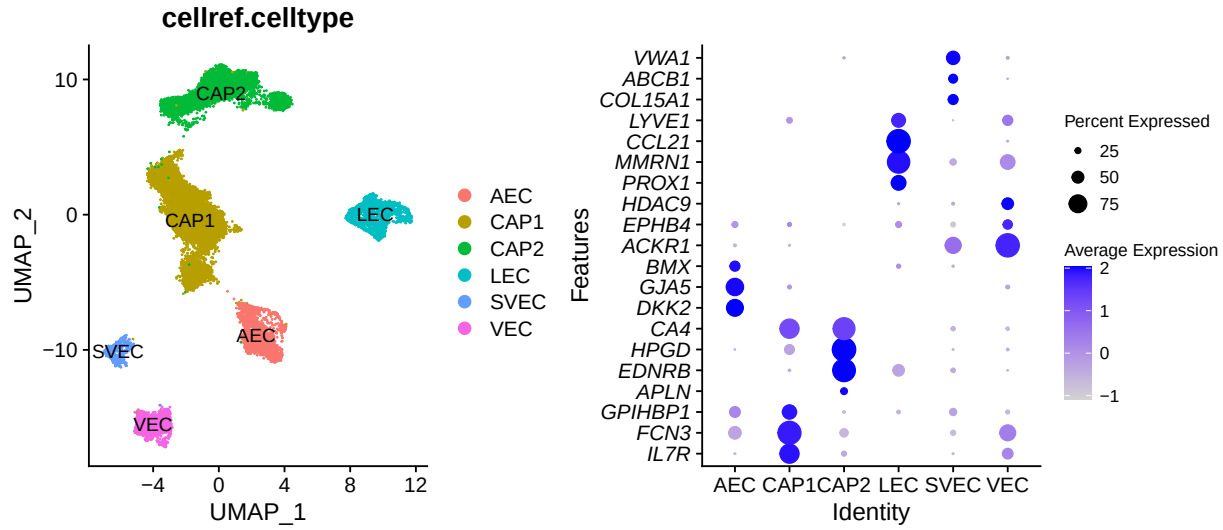


Step 4. Now, we are ready to construct the CellRef. We mapped all other cells to the seed cells using Seurat's label transfer and SingleR. The CellRef will be comprised of the seed cells and cells with consistent cell type predictions by both methods and with kNN purity socre >=0.6 (means at least 60% of the nearest neighbors of a cell have the same cell type prediction)

```
obj.cellref = buildCellRef(obj, obj_seed_rpca, verbose = F)
```

Let's visualize the CellRef in UMAP and evaluate cell identities in the CellRef using expression of marker genes.

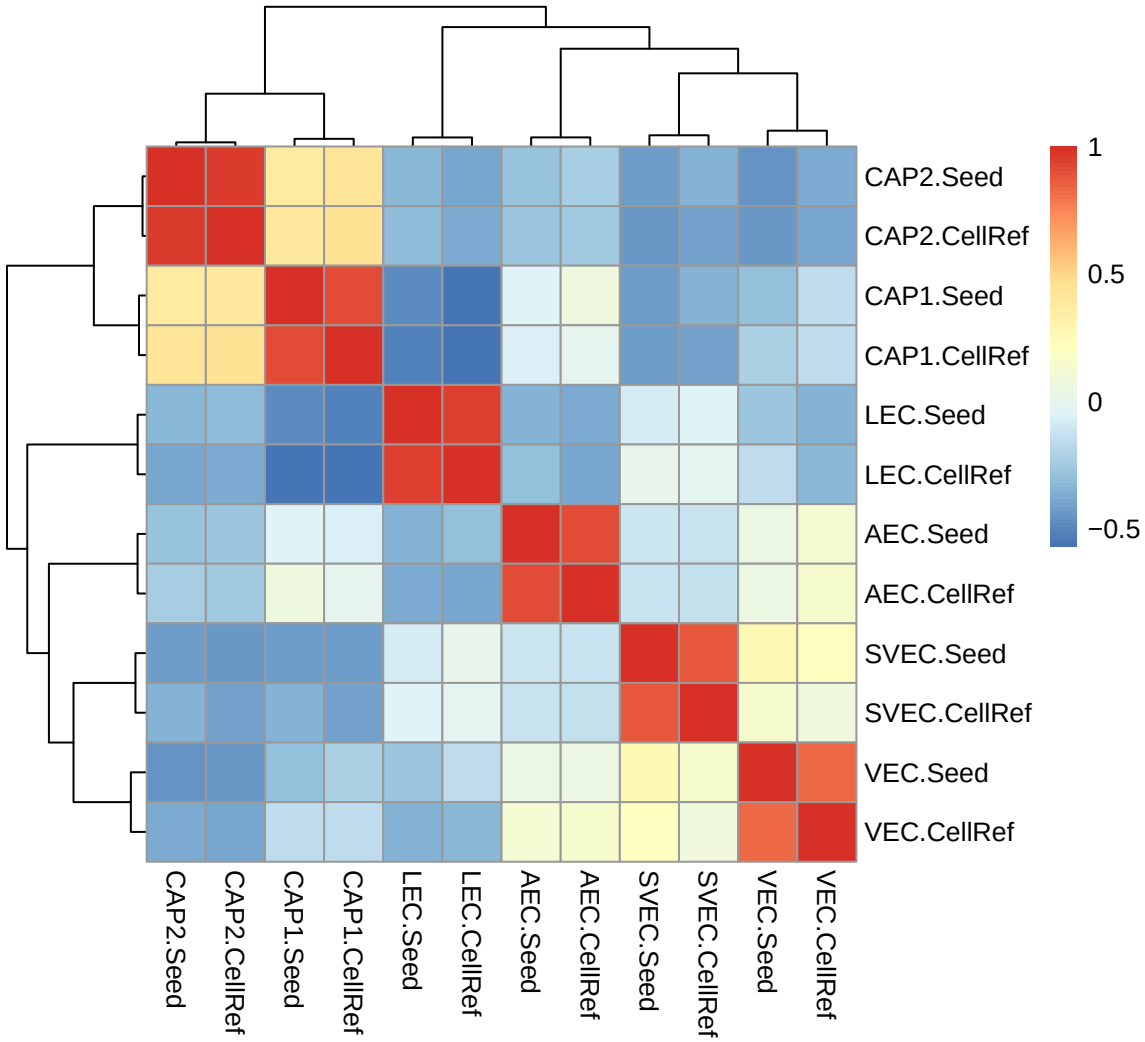
```
g1 = DimPlot(obj.cellref, reduction = "umap", group.by = "cellref.celltype", label = T)

g = DotPlot(obj.cellref, assay = "RNA", features = unique(as.character(ctd$Marker)), dot.min = 0.05,
    group.by = "cellref.celltype")
g = g + coord_flip()
g = g + theme(axis.text.y = element_text(face = "italic"), legend.title = element_text(size = 10),
    legend.text = element_text(size = 10))

g1 + g
```



Create pseudo-bulk profiles for each cell type using gene expression in the seed and the cellref cells and perform hierarchical clustering analysis.

```
seed.avg = AverageExpression(obj_seed, assay = "RNA", group.by = "Seed", return.seurat = T)
seed.avg = FindVariableFeatures(seed.avg, nfeatures = 2000)

cellref.avg = AverageExpression(obj.cellref, assay = "RNA", group.by = "cellref.celltype", return.seura
cellref.avg = FindVariableFeatures(cellref.avg, nfeatures = 2000)

hvg.common = union(seed.avg@assays$RNA@var.features, cellref.avg@assays$RNA@var.features)

seed.avg = ScaleData(seed.avg, features = hvg.common)
cellref.avg = ScaleData(cellref.avg, features = hvg.common)

seed.data = seed.avg@assays$RNA@scale.data
```

```
cellref.data = cellref.avg@assays$RNA@scale.data

colnames(seed.data) = paste0(colnames(seed.data), ".Seed")
colnames(cellref.data) = paste0(colnames(cellref.data), ".CellRef")

tmp = cbind(seed.data, cellref.data[rownames(seed.data), ])
tmp = cor(tmp)

pheatmap::pheatmap(tmp, clustering_method = "complete")
```



Execution time of this demo:

```
end = Sys.time()
print(end - start)
```

```
## Time difference of 36.75686 mins
```

Finally, save the CellRef seed and CellRef objects.

```
# NOT RUN
save(obj_seed_rpca, file = "lung_endo.cellref_seed.rds")
save(obj.cellref, file = "lung_endo.cellref.rds")
```

Session Info

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] pheatmap_1.0.12             ggplot2_3.3.6
##  [3] reshape2_1.4.4              dplyr_1.0.9
##  [5] RobustRankAggreg_1.1        harmony_0.1.0
##  [7] Rcpp_1.0.8.3                SingleR_1.10.0
##  [9] monocle3_1.2.9              SingleCellExperiment_1.18.0
## [11] SummarizedExperiment_1.26.1 GenomicRanges_1.48.0
## [13] GenomeInfoDb_1.32.2         IRanges_2.30.0
## [15] S4Vectors_0.34.0            MatrixGenerics_1.8.0
## [17] matrixStats_0.62.0          Biobase_2.56.0
## [19] BiocGenerics_0.42.0         sp_1.5-0
## [21] SeuratObject_4.1.0          Seurat_4.1.1
## [23] CellRef_0.1.0
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.2                reticulate_1.25
##   [3] tidyselect_1.1.2          lme4_1.1-29
##   [5] htmlwidgets_1.5.4         grid_4.2.0
##   [7] BiocParallel_1.30.3       Rtsne_0.16
##   [9] munsell_0.5.0             ScaledMatrix_1.4.0
##  [11] codetools_0.2-18          ica_1.0-2
##  [13] statmod_1.4.36            scran_1.24.0
##  [15] future_1.26.1             miniUI_0.1.1.1
##  [17] withr_2.5.0               batchelor_1.12.1
##  [19] spatstat.random_2.2-0     colorspace_2.0-3
##  [21] progressr_0.10.1          highr_0.9
##  [23] knitr_1.39                rstudioapi_0.13
##  [25] ROCR_1.0-11               tensor_1.5
```

```
##  [27] listenv_0.8.0             labeling_0.4.2
##  [29] GenomeInfoDbData_1.2.8    polyclip_1.10-0
##  [31] farver_2.1.0              parallelly_1.32.0
##  [33] vctrs_0.4.1               generics_0.1.2
##  [35] xfun_0.31                 R6_2.5.1
##  [37] rsvd_1.0.5                locfit_1.5-9.5
##  [39] bitops_1.0-7              spatstat.utils_2.3-1
##  [41] DelayedArray_0.22.0       assertthat_0.2.1
##  [43] promises_1.2.0.1          scales_1.2.0
##  [45] rgeos_0.5-9               gtable_0.3.0
##  [47] beachmat_2.12.0           globals_0.15.0
##  [49] goftest_1.2-3             rlang_1.0.2
##  [51] splines_4.2.0             lazyeval_0.2.2
##  [53] spatstat.geom_2.4-0       yaml_2.3.5
##  [55] abind_1.4-5               httpuv_1.6.5
##  [57] tools_4.2.0               ellipsis_0.3.2
##  [59] spatstat.core_2.4-4       RColorBrewer_1.1-3
##  [61] ggridges_0.5.3            plyr_1.8.7
##  [63] sparseMatrixStats_1.8.0   zlibbioc_1.42.0
##  [65] purrr_0.3.4               RCurl_1.98-1.6
##  [67] rpart_4.1.16              deldir_1.0-6
##  [69] pbapply_1.5-0             cowplot_1.1.1
##  [71] zoo_1.8-10                ggrepel_0.9.1
##  [73] cluster_2.1.3             magrittr_2.0.3
##  [75] data.table_1.14.2         RSpectra_0.16-1
##  [77] scattermore_0.8           ResidualMatrix_1.6.0
##  [79] lmtest_0.9-40             RANN_2.6.1
##  [81] fitdistrplus_1.1-8        patchwork_1.1.1
##  [83] mime_0.12                 evaluate_0.15
##  [85] xtable_1.8-4              gridExtra_2.3
##  [87] compiler_4.2.0            tibble_3.1.7
##  [89] KernSmooth_2.23-20        crayon_1.5.1
##  [91] minqa_1.2.4               htmltools_0.5.2
##  [93] mgcv_1.8-40               later_1.3.0
##  [95] tidyr_1.2.0               DBI_1.1.2
##  [97] formatR_1.12              MASS_7.3-57
##  [99] boot_1.3-28               leidenbase_0.1.11
## [101] Matrix_1.4-1              cli_3.3.0
## [103] parallel_4.2.0            metapod_1.4.0
## [105] igraph_1.3.1              pkgconfig_2.0.3
## [107] terra_1.5-21              plotly_4.10.0
## [109] scuttle_1.6.2             spatstat.sparse_2.1-1
## [111] dqrng_0.3.0               XVector_0.36.0
## [113] stringr_1.4.0             digest_0.6.29
## [115] sctransform_0.3.3         RcppAnnoy_0.0.19
## [117] spatstat.data_2.2-0       rmarkdown_2.14
## [119] leiden_0.4.2              uwot_0.1.11
## [121] edgeR_3.38.1              DelayedMatrixStats_1.18.0
## [123] shiny_1.7.1               nloptr_2.0.3
## [125] lifecycle_1.0.1           nlme_3.1-157
## [127] jsonlite_1.8.0            BiocNeighbors_1.14.0
## [129] viridisLite_0.4.0         limma_3.52.1
## [131] fansi_1.0.3               pillar_1.7.0
## [133] lattice_0.20-45           fastmap_1.1.0
```

```
## [135] httr_1.4.3                survival_3.3-1
## [137] glue_1.6.2                png_0.1-7
## [139] bluster_1.6.0             stringi_1.7.6
## [141] BiocSingular_1.12.0       irlba_2.3.5
## [143] future.apply_1.9.0
```