Seurat analysis explanation

I am new to analyzing single-cell data in Seurat (R). The script I have below works for cell-type transfer, but I'm not entirely sure what it does, and why it works. I'm also not sure which lines of code are "fixed in stone" (i.e., I shouldn't change it when I analyze my own data), or what I can change when I'm analyzing a different dataset. Can you explain step-by-step what different chunks of this code does?

=====

```
> rm(list=ls())
>
> library(Seurat)
> library(DESeq2)
> library(clusterProfiler)
> library(org.Hs.eg.db)
>
> Sys.setenv(R_MAX_VSIZE = 16e9)
>
> load("/Users/kevinlin/Library/CloudStorage/Dropbox/Collaboration-and-People/sumie-katie/out/ADRC_workshop_2025/microglia_pfc/seaad_microglia.RData")
> seaad <- Seurat::FindVariableFeatures(seaad, nfeatures = 2000)
Calculating gene variances
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Calculating feature variances of standardized and clipped values
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
> seaad <- Seurat::ScaleData(seaad)
Centering and scaling data matrix
  |=========================================| 100%
> seaad <- Seurat::RunPCA(seaad,
+                         features = Seurat::VariableFeatures(seaad),
+                         verbose = FALSE)
> seaad <- Seurat::RunUMAP(seaad,
+                          dims = 1:30,
+                          verbose = FALSE)
> scCustomize::DimPlot_scCustom(seaad,
+                               reduction = "umap",
+                               group.by = "Supertype")
>
> # from https://compbio.mit.edu/microglia_states/
> # Files:
> # 1)
https://personal.broadinstitute.org/cboix/sun_victor_et_al_data/ROSMAP.ImmuneCells.6regions.snR
NAseq.counts.rds?dl=0
> # 2)
https://personal.broadinstitute.org/cboix/sun_victor_et_al_data/ROSMAP.ImmuneCells.6regions.snR
NAseq.meta.rds?dl=0
> # 3) https://cells.ucsc.edu/rosmap-ad-aging-brain/microglia-states/meta.tsv
> # construct the ROSMAP object
> rosmap_count <- readRDS("/Users/kevinlin/Library/CloudStorage/Dropbox/Collaboration-and-
People/sumie-
katie/out/ADRC_workshop_2025/microglia_pfc/ROSMAP.ImmuneCells.6regions.snRNAseq.counts.rd
s")
> rosmap_meta <- readRDS("/Users/kevinlin/Library/CloudStorage/Dropbox/Collaboration-and-
People/sumie-
katie/out/ADRC_workshop_2025/microglia_pfc/ROSMAP.ImmuneCells.6regions.snRNAseq.meta.rds")
> rosmap_meta2 <- read.csv("/Users/kevinlin/Library/CloudStorage/Dropbox/Collaboration-and-
People/sumie-katie/out/ADRC_workshop_2025/microglia_pfc/ROSMAP.meta.tsv",
+                          sep = "\t")
> rownames(rosmap_meta2) <- rosmap_meta2[,"cellName"]
>
> rosmap <- Seurat::CreateSeuratObject(counts = rosmap_count,
+                                      meta.data = rosmap_meta,
+                                      min.cells = 3,
+                                      min.features = 200)

> microglia_state <- rosmap_meta2[Seurat::Cells(rosmap), "State"]
> rosmap$State <- microglia_state
> rosmap <- subset(rosmap, !is.na(rosmap$State))
Warning message:
Removing 21914 cells missing data for vars requested
> rm(list = "rosmap_count"); gc(TRUE)
Garbage collection 606 = 293+88+225 (level 2) ...
690.8 Mbytes of cons cells used (47%)
6632.3 Mbytes of vectors used (28%)
            used   (Mb) gc trigger    (Mb)
Ncells  12934091  690.8   27482948  1467.8
Vcells 869306866 6632.3 3096037698 23620.9
       limit (Mb)   max used    (Mb)
Ncells         NA   27482948  1467.8
Vcells      32768 4284818774 32690.6
>
> # Do a simple processing of rosmap
> rosmap <- Seurat::NormalizeData(rosmap)
Normalizing layer: counts
Performing log-normalization
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
> rosmap <- Seurat::FindVariableFeatures(rosmap,
+                                        selection.method = "vst",
+                                        nfeatures = 2000)
Finding variable features for layer counts
```

```
Calculating gene variances
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Calculating feature variances of standardized and clipped values
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
> rosmap <- Seurat::ScaleData(rosmap)
Centering and scaling data matrix
  |=========================================| 100%
> rosmap <- Seurat::RunPCA(rosmap,
+                          features = Seurat::VariableFeatures(rosmap),
+                          verbose = FALSE)
> rosmap <- Seurat::RunUMAP(rosmap,
+                           dims = 1:30,
+                           verbose = FALSE)
> scCustomize::DimPlot_scCustom(rosmap,
+                               reduction = "umap",
+                               group.by = "State")
> # Add dataset-specific prefixes so barcodes don't collide
> seaad  <- Seurat::RenameCells(seaad,  add.cell.id = "seaad")
> rosmap <- Seurat::RenameCells(rosmap, add.cell.id = "rosmap")
> common_genes <- intersect(rownames(seaad), rownames(rosmap))
> seaad  <- subset(seaad,  features = common_genes)[common_genes, ]
> rosmap <- subset(rosmap,  features = common_genes)[common_genes, ]
> anchors <- Seurat::FindTransferAnchors(
+   reference          = seaad,
+   query              = rosmap,
+   normalization.method = "LogNormalize",
+   reference.reduction  = "pca",
+   reduction            = "pcaproject",
+   dims                 = 1:30
+ )
Projecting cell embeddings
Finding neighborhoods
Finding anchors
	Found 31008 anchors
> rosmap <- Seurat::IntegrateEmbeddings(
+   anchorset          = anchors,
+   reference          = seaad,
+   query              = rosmap,
+   reductions         = "pcaproject",
+   new.reduction.name  = "integrated.pca",
+   dims.to.integrate   = 1:30
+ )
Requested to reuse weights matrix, but no weights found. Computing new weights.
Warning: Layer counts isn't present in the assay object; returning NULL
Warning: Layer counts isn't present in the assay object; returning NULL
  |                                                  | 0 % ~calculating
Integrating dataset 2 with reference dataset
```

```
Finding integration vectors
Finding integration vector weights
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Integrating data
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=47s
> # Now visualize
> # copy the PCA reduction so both objects agree on the name
> seaad[["integrated.pca"]] <- seaad[["pca"]]
Warning message:
Key ‘PC_’ taken, using ‘integratedpca_’ instead
> seaad <- Seurat::DietSeurat(
+   object      = seaad,
+   features    = common_genes,
+   assays      = "RNA",
+   counts      = TRUE,   data = FALSE,  scale.data = FALSE,
+   dimreducs   = c("integrated.pca", "umap")
+ )
> rosmap <- Seurat::DietSeurat(
+   object      = rosmap,
+   features    = common_genes,
+   assays      = "RNA",
+   counts      = TRUE,   data = FALSE,  scale.data = FALSE,
+   dimreducs   = c("integrated.pca", "umap")
+ )
> combo <- merge(seaad,
+                rosmap)
> dataset_vec <- rep(NA, length(Seurat::Cells(combo)))
> names(dataset_vec) <- Seurat::Cells(combo)
> dataset_vec[Seurat::Cells(seaad)] <- "SEAAD"
> dataset_vec[Seurat::Cells(rosmap)] <- "ROSMAP"
> combo$dataset <- dataset_vec
> # Manually construct the integrated.pca assay
> dimred_mat <- matrix(NA,
+                      nrow = length(Seurat::Cells(combo)),
+                      ncol = ncol(rosmap[["integrated.pca"]]@cell.embeddings),
+                      dimnames = list(
+                        Seurat::Cells(combo),
+                        colnames(rosmap[["integrated.pca"]]@cell.embeddings)
+                      ))
> dimred_mat[Seurat::Cells(seaad),] <-
seaad[["integrated.pca"]]@cell.embeddings[,colnames(dimred_mat)]
> dimred_mat[Seurat::Cells(rosmap),] <-
rosmap[["integrated.pca"]]@cell.embeddings[,colnames(dimred_mat)]
> combo[["integrated.pca"]] <- Seurat::CreateDimReducObject(dimred_mat)
Warning message:
No assay specified, setting assay as RNA by default.
> combo <- Seurat::RunUMAP(
+   object     = combo,
+   reduction  = "integrated.pca",
+   dims       = 1:30,
+   reduction.name = "umap.integrated"
+ )
20:28:50 UMAP embedding parameters a = 0.9922 b = 1.112
20:28:50 Read 175493 rows and found 30 numeric columns
20:28:50 Using Annoy for neighbor search, n_neighbors = 30
20:28:50 Building Annoy index with metric = cosine, n_trees = 50
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
20:29:00 Writing NN index file to temp file
/var/folders/hs/22vv3mms6xbbkl2xk5frsnhc0000gp/T//RtmpG4hrIa/file11d14b2a6c34
20:29:00 Searching Annoy index using 1 thread, search_k = 3000
20:29:46 Annoy recall = 100%
20:29:47 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 30
20:29:49 Initializing from normalized Laplacian + noise (using RSpectra)
20:29:53 Commencing optimization for 200 epochs, with 7879712 positive edges
20:29:53 Using rng type: pcg
Using method 'umap'
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
20:30:47 Optimization finished
>
> Seurat::Idents(combo) <- "dataset"
> scCustomize::DimPlot_scCustom(combo,
+                              reduction = "umap.integrated",
+                              group.by = "dataset")
> # https://satijalab.org/seurat/articles/integration_mapping.html
> predictions <- Seurat::TransferData(anchorset = anchors,
+                                    refdata = seaad$Supertype,
+                                    dims = 1:30)
Finding integration vectors
Finding integration vector weights
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Predicting cell labels
> rosmap <- Seurat::AddMetaData(rosmap,
+                              metadata = predictions)
> table(rosmap$predicted.id)

      Micro-PVM_1 Micro-PVM_1_1-SEAAD        Micro-PVM_2 
              211                  11             130203 
Micro-PVM_2_1-SEAAD Micro-PVM_2_2-SEAAD   Micro-PVM_3-SEAAD 
              488                   1               2093 
> scCustomize::DimPlot_scCustom(rosmap,
+                              reduction = "umap",
+                              group.by = "Supertype")
Error in Meta_Present():
! No meta data columns found.
```

```
i The following meta data columns were not found: Supertype
Run rlang::last_trace() to see where the error occurred.
> scCustomize::DimPlot_scCustom(rosmap,
+                     reduction = "umap",
+                     group.by = "predicted.id")
> scCustomize::DimPlot_scCustom(rosmap,
+                     reduction = "umap",
+                     split.by = "predicted.id",
+                     split_seurat = TRUE,
+                     num_columns = 3)
> scCustomize::DimPlot_scCustom(seaad,
+                     reduction = "umap",
+                     split.by = "Supertype",
+                     split_seurat = TRUE,
+                     num_columns = 3)
> scCustomize::DimPlot_scCustom(combo,
+                     reduction = "umap.integrated",
+                     split.by = "dataset",
+                     split_seurat = TRUE)
=====
```

Below I've broken the script into logical chunks, explained **what each chunk does**, and flagged the lines that are essentially boiler-plate versus those you will likely tweak when you analyze a different dataset.

---

1 Environment set-up

```r
rm(list = ls())                          # clear the workspace
library(Seurat); library(DESeq2)
library(clusterProfiler); library(org.Hs.eg.db)
Sys.setenv(R_MAX_VSIZE = 16e9)          # raise R's memory cap
```

- **Fixed:** loading `Seurat` (core package) and a clean workspace.
- **Optional:** the other libraries aren't used later; include them only if you need them.
- **Optional:** adjust `R_MAX_VSIZE` to suit your machine.

---

2 Load & preprocess the *reference* dataset (SEA-AD microglia)

```r
load(".../seaad_microglia.RData")        # brings in a ready-made Seurat object
seaad <- FindVariableFeatures(seaad, nfeatures = 2000)   # pick HVGs
seaad <- ScaleData(seaad)                               # z-score per gene
seaad <- RunPCA(seaad, features = VariableFeatures(seaad))
seaad <- RunUMAP(seaad, dims = 1:30)
```

step	why it matters / tunable pieces
FindVariableFeatures – picks genes that drive most biological signal; 2 000 is a common default but you can raise/lower it.	
ScaleData – centers/scales each gene. Leave default unless you need regressions (e.g. `vars.to.regress`).	
RunPCA – computes PCs from HVGs. You can change `ndims.print` or the feature set.	
RunUMAP – visual; the `dims` argument should match how many PCs you trust (often based on an elbow plot).	

Nothing in this block is hard-wired; you will re-run exactly these steps on every new reference object, just with parameters tuned to the data size and quality.

---

3 Load & preprocess the *query* dataset (ROSMAP microglia)

```r
r

rosmap_count <- readRDS("counts.rds")  # raw counts matrix
rosmap_meta  <- readRDS("meta.rds")    # cell-level metadata
...
rosmap <- CreateSeuratObject(counts = rosmap_count,
                             meta.data = rosmap_meta,
                             min.cells    = 3,
                             min.features = 200)
# keep only microglia with state labels
rosmap$State <- rosmap_meta2[Cells(rosmap), "State"]
rosmap <- subset(rosmap, !is.na(State))
```

- `CreateSeuratObject(...)` parameters (`min.cells`, `min.features`) depend on your dataset's depth and filtering strategy.
- Adding the `State` column is **dataset-specific**; in your own data you'll use whatever column contains a ground-truth label (if any).
- The remainder (NormalizeData, FindVariableFeatures, ScaleData, RunPCA, RunUMAP) mirrors the reference preprocessing and is routine.

---

4 Make gene and barcode names compatible

```r
r

seaad  <- RenameCells(seaad,  add.cell.id = "seaad")
rosmap <- RenameCells(rosmap, add.cell.id = "rosmap")  # avoids barcode collisions
common_genes <- intersect(rownames(seaad), rownames(rosmap))
seaad  <- subset(seaad , features = common_genes)[common_genes, ]
rosmap <- subset(rosmap, features = common_genes)[common_genes, ]
```

- **Fixed idea:** every integration workflow requires unique cell IDs and the same gene set.
- **Tunable:** you might instead keep all genes and set `features = NULL` when running `FindTransferAnchors`, but restricting to an intersection is safer when gene nomenclature differs.

---

5 Discover “anchors” between reference and query

```
r

anchors <- FindTransferAnchors(
  reference            = seaad,
  query                = rosmap,
  normalization.method = "LogNormalize",
  reference.reduction  = "pca",
  reduction            = "pcaproject",
  dims                 = 1:30
)
```

- **What it does:** computes pairs of biologically similar cells (“anchors”) in PCA space; the result is an `AnchorSet` used downstream. satijalab.org
- **Must-keep:** reference/query objects, `reference.reduction` should match the reduction you computed for the reference (`pca` here).
- **Tunable:** `dims` (# of PCs), `normalization.method`, and `k.anchor` (not shown) are data-dependent.

---

6 Project the query into the reference PCA space

```
r

rosmap <- IntegrateEmbeddings(
  anchorset          = anchors,
  reference          = seaad,
  query              = rosmap,
  reductions         = "pcaproject",
  new.reduction.name = "integrated.pca",
  dims.to.integrate  = 1:30
)
```

- Produces **embedded coordinates for query cells** that are directly comparable to reference PCs – no expression matrix is changed, only the dimensional reduction. satijalab.org
- Parameters should mirror those in `FindTransferAnchors`.

---

7 (Manual) build a combined object for visualization

Most of the subsequent code (copying the PCA into `seaad`, calling `DietSeurat`, merging, then re-running UMAP) is **only for joint plotting**.
If that’s all you need, you can swap the manual steps for the convenience wrapper:

```
r

combo <- MapQuery(
  anchorset = anchors,
  reference = seaad,
  query     = rosmap,
  refdata   = list(Supertype = seaad$Supertype),  # optional label transfer
  new.reduction = "integrated.pca"
)
```

`MapQuery` automatically calls `TransferData`, `IntegrateEmbeddings`, and `ProjectUMAP`. satijalab.org

---

8 Transfer cell-type labels

```
r

predictions <- TransferData(anchorset = anchors,
                            refdata   = seaad$Supertype,
                            dims      = 1:30)
rosmap <- AddMetaData(rosmap, metadata = predictions)
```

- **What it does:** for every query cell, computes a predicted label (`predicted.id`) and a prediction score by weighting its anchors. satijalab.org
- **Fixed:** you always pass the same `anchorset`.
- **Change:** `refdata` can be any vector (categorical) or matrix (continuous) you want to propagate from reference to query – e.g., cell types, disease scores, gene signatures.

---

9 Plotting

All `scCustomize::DimPlot_scCustom(...)` calls are purely for visualization.
Swap them for `DimPlot` or `SpatialDimPlot` etc. as you like.

Summary of “fixed” vs. “flexible”

Part of the script	Treat as boiler-plate	Commonly customized
Clearing workspace, loading Seurat	✓	–
Normalization → HVG → Scale → PCA	✓ workflow	HVG number, regression variables, #PCs
Renaming cells, intersecting genes	✓ necessity	Gene filter strategy
`FindTransferAnchors` / `IntegrateEmbeddings` / `TransferData` chain	✓ functions & order	dims, reduction type
Visualization code	–	colors, reductions, facets
Memory tricks (`DietSeurat`, `gc()`)	optional	object size–dependent

Practical tips when adapting to a new dataset

1. **Pick the right reference.** It should already contain the cell types you expect in the query and be processed with the same chemistry (e.g. 10x v3 vs v2).
2. **Match preprocessing choices.** If you SCTransform the reference you should SCTransform the query before finding anchors.
3. **Mind gene naming conventions.** Use Ensembl IDs or gene symbols consistently, or map before intersecting.

4. **Evaluate prediction scores.** `predicted.id` comes with `prediction.score.max`; filter or threshold low-confidence cells.
5. **Use wrappers in Seurat v5.** `MapQuery` + `IntegrateLayers` give the same result with far fewer manual steps.

With those adjustments, the anchor-based label transfer pipeline will generalize to virtually any pair of single-cell datasets.