A2780S

Suchakhree\_Amsaard

2024-04-10

## Set working directory to pull files

# Define the base directory where the new subfolder will be created  
base\_dir <- "/Users/fin/Desktop/A2780S/"

# STEP 1: Load all the data & Quality Control

## Load RDS files

A2780S\_S <- readRDS("/Users/fin/Dropbox (UFL)/research-share/Fin/data/A2780S-CT\_S\_UnprocessedCohort\_SeuratObj\_20191023.rds")  
A2780S\_R <- readRDS("/Users/fin/Dropbox (UFL)/research-share/Fin/data/A2780S-CT\_R\_UnprocessedCohort\_SeuratObj\_20191028.rds")

## Combine Sensitive and Resistance data

A2780S <- merge(x=A2780S\_S, y=A2780S\_R, #x and y are used to specify the two Seurat objects you want to merge.  
 add.cell.ids = c("A2780S\_S","A2780S\_R"),  
 merge.data=TRUE,  
 project = "A2780S\_Pair")  
dim(A2780S) # 33538 3870 rows then columns. Features across samples

## [1] 33538 3870

# Define the base directory and the new subfolder name  
new\_subfolder\_name <- "Unprocessed\_data"  
  
# Construct the full path for the new subfolder  
new\_subfolder\_path <- file.path(base\_dir, new\_subfolder\_name)  
  
# Create the new subfolder if it doesn't exist  
if (!dir.exists(new\_subfolder\_path)) {  
 dir.create(new\_subfolder\_path)  
}  
  
# Specify the filename for the RDS file, incorporating the current date  
rds\_filename <- paste0("A2780S-CT\_S+R\_UnprocessedCohort\_SeuratObj\_", ".rds")  
  
# Full path for the RDS file including the subfolder  
full\_rds\_path <- file.path(new\_subfolder\_path, rds\_filename)  
  
# Save the Seurat object as an RDS file in the specified location  
saveRDS(A2780S, file = full\_rds\_path)

# STEP 2: Quality control

# grep function searches for matches to argument patterns within each element of a character vector   
# pattern chosen is anything that contains ^MT-  
mito.genes <- grep(pattern = "^MT-", A2780S@assays$RNA@counts@Dimnames[[1]], value = TRUE)   
mito.genes

## [1] "MT-ND1" "MT-ND2" "MT-CO1" "MT-CO2" "MT-ATP8" "MT-ATP6" "MT-CO3"   
## [8] "MT-ND3" "MT-ND4L" "MT-ND4" "MT-ND5" "MT-ND6" "MT-CYB"

A2780S$percent.MT <- PercentageFeatureSet(A2780S,pattern="^MT-") #The percentage of reads that map to the mitochondrial genome  
percent.mito <- A2780S$percent.MT/100  
  
#Add a column to meta.data   
A2780S[['percent.mito']] <- percent.mito  
head(A2780S@meta.data,5) #Show QC metrics for the first 5 cells in the control group

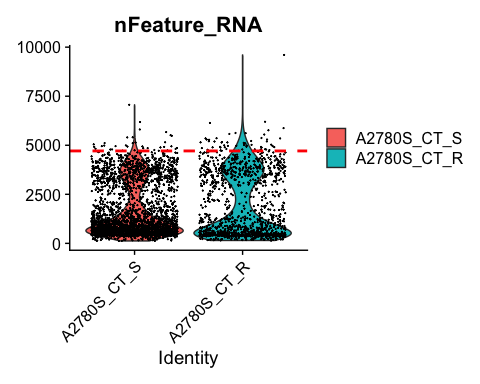
## orig.ident nCount\_RNA nFeature\_RNA  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGCAGGGTATG A2780S\_CT\_S 19540 3903  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTAGCGTCC A2780S\_CT\_S 19408 3499  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTATAGGGC A2780S\_CT\_S 1643 859  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTGAAGGCT A2780S\_CT\_S 970 661  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTGCCTTGG A2780S\_CT\_S 580 404  
## percent.MT percent.mito  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGCAGGGTATG 5.112590 0.05112590  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTAGCGTCC 3.127576 0.03127576  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTATAGGGC 1.825928 0.01825928  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTGAAGGCT 5.154639 0.05154639  
## A2780S\_S\_A2780S\_CT\_S\_AAACCTGGTGCCTTGG 3.965517 0.03965517

## Descriptive stats:  
summary(A2780S@meta.data$nFeature\_RNA)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 123 575 1095 1791 3236 9602

# Min. 1st Qu. Median Mean 3rd Qu. Max.   
# 123 575 1095 1791 3236 9602   
  
# Set up an upper and lower limit, na.rm=TRUE because the features column has some missing values   
## Upper limit is the mean of the feature values + 2 sd   
nFeatUpper <- mean(A2780S@meta.data$nFeature\_RNA, na.rm=TRUE) + 2\*sd(A2780S@meta.data$nFeature\_RNA, na.rm=TRUE) ## 1790.597 + 2919.648  
nFeatLower <- 200   
  
VlnPlot(object=A2780S, features = c("nFeature\_RNA"))+geom\_hline(yintercept=nFeatUpper, linetype="dashed", color="red", size=1)

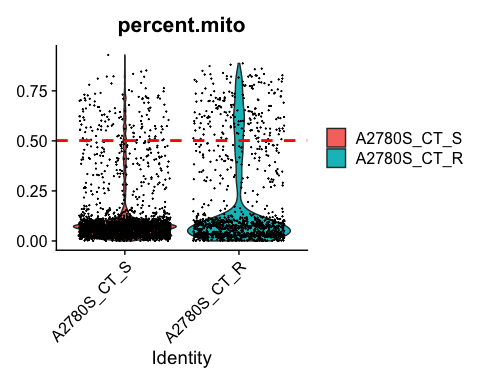
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.  
## ℹ Please use `linewidth` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.



summary(A2780S@meta.data$percent.mito)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.00000 0.03396 0.06772 0.13197 0.09882 0.92928

# Min. 1st Qu. Median Mean 3rd Qu. Max.   
# 0.00000 0.03396 0.06772 0.13197 0.09882 0.92928  
  
### assumption: cancer cells may have higher metabolic rate  
##Set an upper limit with the mean and 2 sd, not use the 5%, we need to cut off a higher percentage anyways because cancer cells have a high metabolic rate   
perMitoUpper <- mean(A2780S@meta.data$percent.mito, na.rm=TRUE) + 2\*sd(A2780S@meta.data$percent.mito, na.rm=TRUE) ## 0.1319656 + 0.3696054  
VlnPlot(object=A2780S, features = c("percent.mito"))+geom\_hline(yintercept=perMitoUpper, linetype="dashed", color="red", size=1)



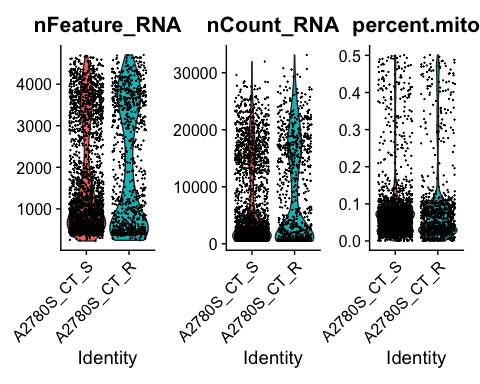
# Divide the data based on the limits established  
# subset function in Seurat to filter cells in A2780S  
# filter away cells that have > perMitoUpper mitochondrial counts  
A2780S <- subset(x=A2780S, subset=nFeature\_RNA > nFeatLower & nFeature\_RNA < nFeatUpper & percent.mito < perMitoUpper)  
  
# The subset argument filters cells based on the specified conditions, including  
# nFeature\_RNA > nFeatLower (number of features greater than a lower threshold),   
# nFeature\_RNA < nFeatUpper (number of features less than an upper threshold),  
# percent.mito < perMitoUpper (percentage of mitochondrial genes less than an upper threshold).  
  
dim(A2780S)

## [1] 33538 3449

# 33538 3449

# Define the name of the new folder  
new\_subfolder\_name <- "Process\_data"  
  
# Construct the full path for the 'Process\_data' folder  
process\_data\_folder\_path <- file.path(base\_dir, new\_subfolder\_name)  
  
# Create the 'Process\_data' folder if it doesn't exist  
if (!dir.exists(process\_data\_folder\_path)) {  
 dir.create(process\_data\_folder\_path)  
}  
  
# Specify the filename for the RDS file, including the automated date  
rds\_filename <- paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep2\_", ".rds")  
  
# Construct the full path for the RDS file within the 'Process\_data' folder  
full\_rds\_path <- file.path(process\_data\_folder\_path, rds\_filename)  
  
# Save the Seurat object as an RDS file in the specified location  
saveRDS(A2780S, file = full\_rds\_path)

# Visualize QC metrics as a violin plot  
VlnPlot(A2780S, features = c("nFeature\_RNA", "nCount\_RNA", "percent.mito"), ncol = 3)



# STEP 3: Normalize

Normalize count data per cell and transform to log scale. Make variable contribute equally to the analysis LogNormalize: feature counts for each cell are divided by the total counts for that cell and multiplied by the scale factor

A2780S <- NormalizeData(object=A2780S, normalization.method="LogNormalize", scale.factor=1e4)  
dim(A2780S)

## [1] 33538 3449

# 33538 3449  
  
# Specify the filename for the RDS file, including the automated date and step identifier  
rds\_filename <- paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep3\_", ".rds")  
  
# Construct the full path for the RDS file within the 'Process\_data' folder  
full\_rds\_path <- file.path(process\_data\_folder\_path, rds\_filename)  
  
# Save the Seurat object as an RDS file in the specified location  
saveRDS(A2780S, file = full\_rds\_path)

# STEP 4: Detection of variable features across the single cells #Start Here

Seurat can calculate high variable genes and focuses on these

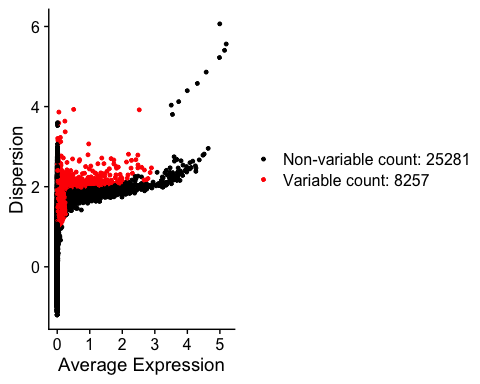
# FindVariableFeatures calculates the average expression and dispersion for each gene, places these genes into bins and calculates a z-score for dispersion within each bin   
A2780S <- FindVariableFeatures(object=A2780S, selection.method = 'mean.var.plot',  
 mean.cutoff = c(0.0125, 3), # x-axis, gene expression  
 dispersion.cutoff = c(0.5, Inf), #y-axis  
 do.plot=TRUE)

## Warning: The following arguments are not used: do.plot

length(x = VariableFeatures(object=A2780S))

## [1] 8257

## number of variable features detected after filtering, normalization: 8257  
  
VariableFeaturePlot(A2780S)



rds\_filename <- paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep4\_", ".rds")  
full\_rds\_path <- file.path(process\_data\_folder\_path, rds\_filename)  
saveRDS(A2780S, file = full\_rds\_path)

# STEP 5: Scaling data and removing unwanted sources of variation

Remove ‘uninteresting’ sources of variation such as technical noise, batch effects and biological sources of variation (cell cycle arrest). Improve analysis since it reduces dimensions and improves clustering

#Seurat scale data function already does all these things for use, we only want to scale certain variables   
A2780S <- ScaleData(object=A2780S, vars.to.regress = c("nCount\_RNA","percent.mito"))

## Regressing out nCount\_RNA, percent.mito

## Centering and scaling data matrix

rds\_filename <- paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep5\_", ".rds")  
full\_rds\_path <- file.path(process\_data\_folder\_path, rds\_filename)  
saveRDS(A2780S, file = full\_rds\_path)

# STEP 6: Perform linear dimension reduction/ Determine PCA

Do linear dimension reduction to group the features per principal components. The ones that are the most similar. Clustering starts here Look at different graphs but heatamps are most helpful when trying to decide which PCs to include for further downstream analyses. Cells and genes ordered according to their PCA scores. We want to select the PCA that have the most variable genes to select for further study ESTABLISHED: We randomly permute a subset of the data (1% by default) and rerun PCA, constructing a ‘null distribution’ of gene scores, and repeat this procedure. We identify ‘significant’ PCs as those who have a strong enrichment of low p-value genes.

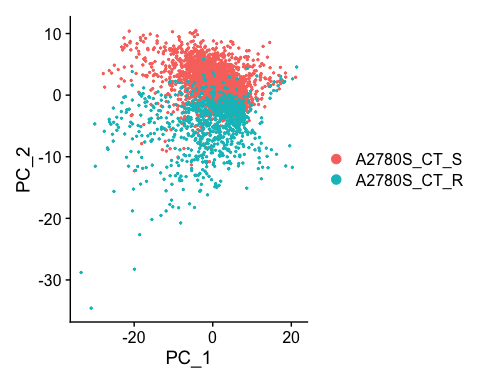
#Get the principal components, Seurat selects the most variable and orders them, we have to decide what PC we will be working with   
A2780S <- RunPCA(object=A2780S, features=VariableFeatures(object=A2780S), verbose = TRUE, npcs = 200)

## PC\_ 1   
## Positive: HIST1H4C, SNHG25, PNPLA4, ARHGDIG, BEX2, MRPL53, ZNF90, S100A2, AC008946.1, PHKG1   
## SMIM27, C3orf14, MATK, LRRC75A, TLE6, HBQ1, AC135048.3, HSD17B8, PIK3CD-AS2, ROM1   
## AGBL3, AL121601.1, FES, SPINT2, AL121899.1, WSB2, CGREF1, STARD6, STN1, AL359232.1   
## Negative: PCDH9, HSPA5, HSP90B1, XIST, SON, CFH, COL3A1, CALR, RPN2, NOP56   
## SERPINH1, SEC62, SCN9A, APLP2, TMBIM6, PLD3, SPRY1, PCOLCE, ITGB1, KPNA2   
## GJA1, HACD3, HSP90AB1, LAPTM4A, PDIA3, PDIA4, KCNQ1OT1, VCAN, FUS, CALD1   
## PC\_ 2   
## Positive: UCHL1, BST2, CNN3, TPM1, HMGB1, SOX4, H2AFZ, PSAT1, TAGLN, PTGES3   
## IGFBP2, MFAP4, TUBB, MYL9, STMN1, BEX2, MT1X, HNRNPH1, TUBA1B, DNAJA1   
## H2AFY2, SNRPB2, MGST1, FKBP4, TUBB6, SDC2, MCM7, CALM1, CDC42EP1, BRIX1   
## Negative: DCN, KCNMA1, TIMP1, NEAT1, DKK1, LUM, SLC10A4, PURPL, GDF15, ID2   
## PCDH9, CDKN1A, DSC2, RNF207, AC097478.1, SHC1, DNAJC1, CRYBG3, APLP1, HGF   
## CRABP2, NES, LGALS1, EMP1, FILIP1L, COL1A1, AL391807.1, HIST1H1C, POLR2J3.1, DDB2   
## PC\_ 3   
## Positive: UBE2C, BIRC5, HMGB2, TUBA1B, CCNA2, PLK1, PTTG1, TOP2A, CDC20, CDK1   
## CENPF, RRM2, PBK, MKI67, NUSAP1, CCNB1, TUBB4B, PCLAF, CDCA3, CENPA   
## NUF2, SLC10A4, KCNMA1, TK1, DEPDC1, UBE2S, MAD2L1, TUBB, CCNB2, AURKA   
## Negative: SOX4, MFAP4, NREP, KCNQ1OT1, BST2, UCHL1, TNNT1, TPM1, PGF, VCAN   
## XIST, DLG1, MAGED2, TLE4, N4BP2L2, TMSB10, PNISR, MT1X, SLC25A42, MEX3A   
## MAGI1, ZNF579, JARID2, YPEL5, SERINC1, MCOLN3, PCMTD1, CDH2, ANXA2, KIF1A   
## PC\_ 4   
## Positive: XIST, KCNQ1OT1, ARHGAP11B, POLR2J3.1, LTBP4, FANCA, ARID1B, KANSL1, DLG1, ASPM   
## GPATCH8, SLC16A1-AS1, HIST1H4C, NSD1, SMCR5, RAD9A, BRCA1, NUP210, MMS22L, DCLRE1C   
## NUP107, GPC6, AC016831.5, DNASE1, PIF1, KHDC4, SULF2, DDX11, CLN6, AC097376.2   
## Negative: LGALS1, C1D, MYL9, MYL12B, S100A1, DCN, RSU1, CRABP2, TNNT1, CHMP5   
## ATP5PB, ARMCX2, CCT8, DDX5, ARMCX3, LDHB, HIST1H2AC, UBC, LUM, CITED2   
## LMNA, CDKN1A, SNX2, ID3, RPS27L, HIST1H1C, BBC3, S100A4, MAGED2, ESD   
## PC\_ 5   
## Positive: TNFRSF12A, SPHK1, TIMP1, COTL1, HSP90B1, ATP1A1, SLC3A2, HSPA8, NPAS1, IL27RA   
## HSPA5, SIL1, TUFM, PPT1, PDIA6, ITGB1, DDX21, SLC7A5, OLFM2, DUSP2   
## MYC, TMEM158, LIPA, PLTP, DIMT1, CD3EAP, TPST2, SHC1, FJX1, LOXL2   
## Negative: CDCA3, PTTG1, HMGB2, BIRC5, CENPF, UBE2C, NUSAP1, TROAP, STMN1, CENPA   
## CCNB2, TOP2A, HIST1H4C, PLK1, PSRC1, NUF2, AURKA, SOX4, MKI67, UBE2S   
## NEK2, CDC20, ASPM, DLGAP5, CKS2, ARL6IP1, CCNA2, CCNB1, PIMREG, AURKB

A2780S <- ProjectDim(object=A2780S)

## PC\_ 1   
## Positive: HIST1H4C, SNHG25, PNPLA4, ARHGDIG, BEX2, MRPL53, ZNF90, S100A2, AC008946.1, PHKG1   
## SMIM27, C3orf14, MATK, LRRC75A, TLE6, HBQ1, AC135048.3, HSD17B8, PIK3CD-AS2, ROM1   
## Negative: PCDH9, HSPA5, HSP90B1, XIST, SON, CFH, COL3A1, CALR, RPN2, NOP56   
## SERPINH1, SEC62, SCN9A, APLP2, TMBIM6, PLD3, SPRY1, PCOLCE, ITGB1, KPNA2   
## PC\_ 2   
## Positive: UCHL1, BST2, CNN3, TPM1, HMGB1, SOX4, H2AFZ, PSAT1, TAGLN, PTGES3   
## IGFBP2, MFAP4, TUBB, MYL9, STMN1, BEX2, MT1X, HNRNPH1, TUBA1B, DNAJA1   
## Negative: DCN, KCNMA1, TIMP1, NEAT1, DKK1, LUM, SLC10A4, PURPL, GDF15, ID2   
## PCDH9, CDKN1A, DSC2, RNF207, AC097478.1, SHC1, DNAJC1, CRYBG3, APLP1, HGF   
## PC\_ 3   
## Positive: UBE2C, BIRC5, HMGB2, TUBA1B, CCNA2, PLK1, PTTG1, TOP2A, CDC20, CDK1   
## CENPF, RRM2, PBK, MKI67, NUSAP1, CCNB1, TUBB4B, PCLAF, CDCA3, CENPA   
## Negative: SOX4, MFAP4, NREP, KCNQ1OT1, BST2, UCHL1, TNNT1, TPM1, PGF, VCAN   
## XIST, DLG1, MAGED2, TLE4, N4BP2L2, TMSB10, PNISR, MT1X, SLC25A42, MEX3A   
## PC\_ 4   
## Positive: XIST, KCNQ1OT1, ARHGAP11B, POLR2J3.1, LTBP4, FANCA, ARID1B, KANSL1, DLG1, ASPM   
## GPATCH8, SLC16A1-AS1, HIST1H4C, NSD1, SMCR5, RAD9A, BRCA1, NUP210, MMS22L, DCLRE1C   
## Negative: LGALS1, C1D, MYL9, MYL12B, S100A1, DCN, RSU1, CRABP2, TNNT1, CHMP5   
## ATP5PB, ARMCX2, CCT8, DDX5, ARMCX3, LDHB, HIST1H2AC, UBC, LUM, CITED2   
## PC\_ 5   
## Positive: TNFRSF12A, SPHK1, TIMP1, COTL1, HSP90B1, ATP1A1, SLC3A2, HSPA8, NPAS1, IL27RA   
## HSPA5, SIL1, TUFM, PPT1, PDIA6, ITGB1, DDX21, SLC7A5, OLFM2, DUSP2   
## Negative: CDCA3, PTTG1, HMGB2, BIRC5, CENPF, UBE2C, NUSAP1, TROAP, STMN1, CENPA   
## CCNB2, TOP2A, HIST1H4C, PLK1, PSRC1, NUF2, AURKA, SOX4, MKI67, UBE2S

DimPlot(A2780S,reduction="pca")

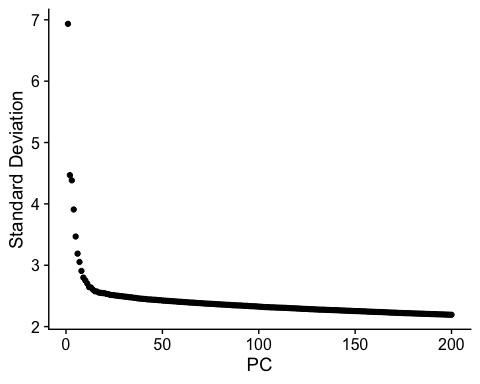


rds\_filename <- paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep6\_",".rds")  
full\_rds\_path <- file.path(process\_data\_folder\_path, rds\_filename)  
saveRDS(A2780S, file = full\_rds\_path)  
A2780S@reductions$pca

## A dimensional reduction object with key PC\_   
## Number of dimensions: 200   
## Number of cells: 3449   
## Projected dimensional reduction calculated: TRUE   
## Jackstraw run: FALSE   
## Computed using assay: RNA

## ElbowPlot

ElbowPlot(A2780S,ndims = 200)

 ## JackStraw Plot

# Load the Seurat object  
A2780S <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep6\_.rds")  
  
# Run JackStraw to determine statistically significant principal components  
A2780S <- JackStraw(A2780S, dims = 50, num.replicate = 100)  
  
# Score JackStraw to apply statistical testing to the PCS  
A2780S <- ScoreJackStraw(A2780S, dims = 1:50)  
  
# Generate the JackStraw plot for the first 50 dimensions  
jackStrawPlot <- JackStrawPlot(A2780S, dims = 1:50)  
  
# Define the path for saving the plot  
plot\_path <- "/Users/fin/Desktop/A2780S/Process\_data/jackStrawPlot.png"  
  
# Save the JackStraw plot  
ggsave(filename = plot\_path, plot = jackStrawPlot, width = 10, height = 8, dpi = 300)

## Warning: Removed 326380 rows containing missing values or values outside the scale range  
## (`geom\_point()`).

# STEP 7: Find Clusters

library(data.table)

##   
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

library(dplyr)  
library(clustree)

Create a k-nearest neighbor (KNN) graph, that will draw connections between cells with similar feature expression patterns, and then partition this graph into highly interconnected communities KNN graphs constructed are based in euclidean distance in PCA space, and refine the edge weights between two cells based on the shared overlap in their local neighborhoods, Do this using the FindNeighbors() function Cluster cells use an optimization technique such as Louvain algorithm (detect communities in large networks) or SLM to group cells together. FindCLuster() function does this, and contains a resolution parameter that sets the ‘granularity’ of the downstream clustering, the higher the values the more clusters. Set parameter between 0.4-1.2 for around 3000 cells.

A2780S <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep6\_.rds") # Ensure full\_rds\_path is correctly defined earlier in your script  
  
# Define the name of the new folder for clustree plots  
clustree\_folder\_name <- "clustree"  
  
# Construct the full path for the clustree folder  
clustree\_folder\_path <- file.path(base\_dir, clustree\_folder\_name)  
  
# Create the clustree folder if it doesn't exist  
if (!dir.exists(clustree\_folder\_path)) {  
 dir.create(clustree\_folder\_path, recursive = TRUE)  
}  
  
# Define the range of resolutions to scan  
range <- seq(0, 1, by = 0.05)  
  
# Loop through each resolution  
for (res in range) {  
 A2780S <- FindNeighbors(object = A2780S, dims = 1:50) # Use 50 principal components  
 A2780S <- FindClusters(object = A2780S, resolution = res, n.start = 100) # Louvain clustering  
}

## Computing nearest neighbor graph

## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 1.0000  
## Number of communities: 1  
## Elapsed time: 4 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.9503  
## Number of communities: 2  
## Elapsed time: 4 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.9197  
## Number of communities: 3  
## Elapsed time: 4 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8989  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8837  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8685  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8534  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8382  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8231  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8079  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7928  
## Number of communities: 5  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7785  
## Number of communities: 6  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7651  
## Number of communities: 7  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7531  
## Number of communities: 7  
## Elapsed time: 6 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7426  
## Number of communities: 7  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7323  
## Number of communities: 8  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7222  
## Number of communities: 8  
## Elapsed time: 6 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7121  
## Number of communities: 9  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.7027  
## Number of communities: 10  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.6951  
## Number of communities: 10  
## Elapsed time: 5 seconds

## Computing nearest neighbor graph  
## Computing SNN

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.6876  
## Number of communities: 10  
## Elapsed time: 5 seconds

# Generate and save the clustree plot  
# Construct the full path for the clustree plot, including the current date  
clustree\_plot\_path <- file.path(clustree\_folder\_path, paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_clustree\_ResScan\_",".pdf"))  
  
# Open a PDF device to save the clustree plot  
pdf(clustree\_plot\_path, width = 15, height = 24)  
# Generate the clustree plot  
tPlot <- clustree(A2780S, prefix = 'RNA\_snn\_res.')  
print(tPlot)  
# Close the PDF device  
dev.off()

## quartz\_off\_screen   
## 2

rds\_filename <- paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep7\_", ".rds")  
full\_rds\_path <- file.path(process\_data\_folder\_path, rds\_filename)  
saveRDS(A2780S, file = full\_rds\_path)

# STEP 8: UMAP

## Select res from clustree to construct UMAP

## Tell the path

# Base dir  
base\_dir <- "/Users/fin/Desktop/A2780S/"  
  
# Define the name of the new folder  
new\_subfolder\_name <- "Process\_data"  
  
# Construct the full path for the 'Process\_data' folder  
process\_data\_folder\_path <- file.path(base\_dir, new\_subfolder\_name)

umap\_folder\_name <- "UMAP"  
  
# Construct the full path for the UMAP folder  
umap\_folder\_path <- file.path(base\_dir, umap\_folder\_name)  
  
# Create the UMAP folder if it doesn't exist  
if (!dir.exists(umap\_folder\_path)) {  
 dir.create(umap\_folder\_path, recursive = TRUE)  
}  
  
# Read the Seurat object  
A2780S.F <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep7\_.rds") # Ensure full\_rds\_path is correctly defined earlier in your script  
  
# Find Clusters at a specific resolution  
clustree\_resolution = 0.15  
A2780S <- FindClusters(object = A2780S.F, resolution = clustree\_resolution, n.start = 100) #res select from clustree

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 100 random starts: 0.8989  
## Number of communities: 5  
## Elapsed time: 5 seconds

# Find all markers  
A2780S.markers <- FindAllMarkers(A2780S, min.pct = 0.25, logfc.threshold = 0.25)

## Calculating cluster 0

## For a (much!) faster implementation of the Wilcoxon Rank Sum Test,  
## (default method for FindMarkers) please install the presto package  
## --------------------------------------------  
## install.packages('devtools')  
## devtools::install\_github('immunogenomics/presto')  
## --------------------------------------------  
## After installation of presto, Seurat will automatically use the more   
## efficient implementation (no further action necessary).  
## This message will be shown once per session

## Calculating cluster 1

## Calculating cluster 2

## Calculating cluster 3

## Calculating cluster 4

# Run UMAP with PCA reduction and specified dimensions  
A2780S <- RunUMAP(object = A2780S, reduction = "pca", dims = 1:50)

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

## 23:27:08 UMAP embedding parameters a = 0.9922 b = 1.112

## 23:27:08 Read 3449 rows and found 50 numeric columns

## 23:27:08 Using Annoy for neighbor search, n\_neighbors = 30

## 23:27:08 Building Annoy index with metric = cosine, n\_trees = 50

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

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

## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|  
## 23:27:08 Writing NN index file to temp file /var/folders/ys/hhllw\_jj35q9vnv51l9bbgq00000gn/T//RtmpF2wotI/file1931d620c35  
## 23:27:08 Searching Annoy index using 1 thread, search\_k = 3000  
## 23:27:09 Annoy recall = 100%  
## 23:27:09 Commencing smooth kNN distance calibration using 1 thread with target n\_neighbors = 30  
## 23:27:10 Initializing from normalized Laplacian + noise (using RSpectra)  
## 23:27:10 Commencing optimization for 500 epochs, with 158342 positive edges  
## 23:27:15 Optimization finished

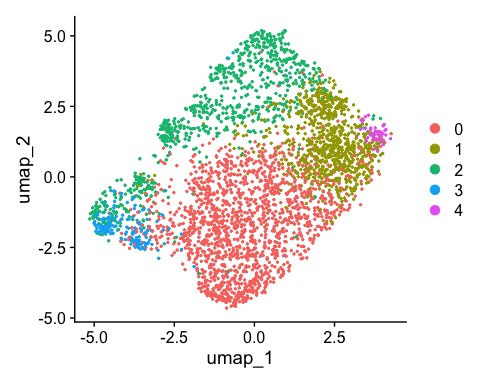
# Generate and save UMAP plot by cluster  
pdf\_path\_cluster <- file.path(umap\_folder\_path, paste0("A2780S\_S+R\_QC-Subset-Cohort\_UMAP\_byCluster\_",clustree\_resolution, ".pdf"))  
pdf(pdf\_path\_cluster, width=12, height=8)  
dPlot <- DimPlot(A2780S, reduction = "umap") + xlim(-15, 15) + ylim(-15, 15)  
print(dPlot)  
dev.off()

## quartz\_off\_screen   
## 2

# Generate and save UMAP plot by cell type  
pdf\_path\_celltype <- file.path(umap\_folder\_path, paste0("A2780S\_S+R\_QC-Subset-Cohort\_UMAP\_clustersByType\_", ".pdf"))  
pdf(pdf\_path\_celltype, width=12, height=8)  
dPlot <- DimPlot(A2780S, reduction = "umap", group.by = 'orig.ident') + xlim(-15, 15) + ylim(-15, 15)  
print(dPlot)  
dev.off()

## quartz\_off\_screen   
## 2

# Save the modified Seurat object  
rds\_path <- file.path(process\_data\_folder\_path, paste0("A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep8\_", ".rds"))  
saveRDS(A2780S, file = rds\_path)  
DimPlot(A2780S, reduction = "umap")



## grouped and count data cluster in sensitive and resistance

## Get Column Name

A2780S <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep8\_.rds")  
listNames <- A2780S@meta.data$orig.ident  
  
# Create a data frame without column name  
df <- data.frame(listNames = listNames, stringsAsFactors = FALSE)  
  
# Define the output file path  
output\_file\_path <- "/Users/fin/Desktop/A2780S/UMAP/A2780SOutputDataNames.csv"  
  
# Save the list to a CSV file without row names and column names  
write.csv(df, file = output\_file\_path, row.names = FALSE, col.names = FALSE, sep=",")

## Warning in write.csv(df, file = output\_file\_path, row.names = FALSE, col.names  
## = FALSE, : attempt to set 'col.names' ignored

## Warning in write.csv(df, file = output\_file\_path, row.names = FALSE, col.names  
## = FALSE, : attempt to set 'sep' ignored

## Get Cluster in for each cell

# Replace 'seurat\_clusters' with the actual column name if it's different  
cluster\_assignments <- A2780S@meta.data$seurat\_clusters  
  
# Create a data frame of cluster assignments  
df\_clusters <- data.frame(cluster = cluster\_assignments)  
  
# Define the output file path for the cluster assignments CSV  
output\_file\_path <- "/Users/fin/Desktop/A2780S/UMAP/A2780SClusterAssignments.csv"  
  
# Save the cluster assignments to a CSV file  
write.csv(df\_clusters, file = output\_file\_path, row.names = FALSE, quote = FALSE)

# Read the CSV files  
mydat1 <- fread("/Users/fin/Desktop/A2780S/UMAP/A2780SClusterAssignments.csv")  
mydat2 <- fread("/Users/fin/Desktop/A2780S/UMAP/A2780SOutputDataNames.csv")  
  
# Merge the two data tables by columns  
fulldat <- cbind(UMI = mydat2[[1]], cluster = mydat1[[1]])  
  
# Convert the merged data into a data table if not already  
fulltab <- as.data.table(fulldat, stringsAsFactors = FALSE)  
  
# name table columns  
names(fulltab)[1] <- paste("UMI")  
names(fulltab)[2] <- paste("cluster")  
print(fulltab)

## UMI cluster  
## <char> <char>  
## 1: A2780S\_CT\_S 0  
## 2: A2780S\_CT\_S 0  
## 3: A2780S\_CT\_S 0  
## 4: A2780S\_CT\_S 0  
## 5: A2780S\_CT\_S 1  
## ---   
## 3445: A2780S\_CT\_R 2  
## 3446: A2780S\_CT\_R 1  
## 3447: A2780S\_CT\_R 2  
## 3448: A2780S\_CT\_R 2  
## 3449: A2780S\_CT\_R 2

# From fulltab, group by cluster then count number in each cluster  
# create a table counting unique UMIs/cells per cluster  
tabPerClus <- fulltab %>%  
 group\_by(UMI, cluster) %>%  
 summarise(count = n(), .groups = 'drop')  
  
divout <- "/Users/fin/Desktop/A2780S/UMAP/CellBreakdown\_PerClusterPerType.csv"  
write.csv(tabPerClus, file=divout)  
print(tabPerClus)

## # A tibble: 10 × 3  
## UMI cluster count  
## <chr> <chr> <int>  
## 1 A2780S\_CT\_R 0 4  
## 2 A2780S\_CT\_R 1 237  
## 3 A2780S\_CT\_R 2 588  
## 4 A2780S\_CT\_R 3 104  
## 5 A2780S\_CT\_R 4 19  
## 6 A2780S\_CT\_S 0 1581  
## 7 A2780S\_CT\_S 1 603  
## 8 A2780S\_CT\_S 2 193  
## 9 A2780S\_CT\_S 3 86  
## 10 A2780S\_CT\_S 4 34

## Find markers for each of the identity classes in a dataset

# Store cell names  
cell\_names <- colnames(A2780S)  
  
# Define the GeneList folder path within the base directory  
gene\_list\_dir <- file.path(base\_dir, "GeneList")  
  
# Create the GeneList directory if it doesn't exist  
if (!dir.exists(gene\_list\_dir)) {  
 dir.create(gene\_list\_dir, recursive = TRUE)  
}  
  
# Save the complete marker list with the automated date  
write.csv(A2780S.markers, file.path(gene\_list\_dir, paste0("DiffExpGeneList\_A2780S\_ALL\_", ".csv")))  
  
# Iterate over each cluster and save the differentially expressed gene list in the GeneList folder  
unique\_clusters <- unique(A2780S.markers$cluster)  
  
for (cluster in unique\_clusters) {  
 # Filter markers for the current cluster  
 clus\_markers <- A2780S.markers %>%   
 filter(cluster == cluster) %>%   
 select(avg\_log2FC, gene)  
   
 # Define the file path for this cluster's gene list, including the GeneList directory  
 file\_path <- file.path(gene\_list\_dir, paste0("DiffExpGeneList\_A2780S\_Cluster-", cluster, "\_", ".csv"))  
   
 # Save the filtered gene list to the CSV file  
 write.csv(clus\_markers, file\_path, row.names = FALSE)  
}

#STEP DEG: DIFFERENTIALLY EXPRESSED GENES Identifies differentially expressed genes between two groups of cells using a Wilcoxon Rank Sum test (default) Reports only both positive and negatively differentially expressed genes Requires the feature to be detected at a min of 25% in either groups. Limit testing to genes which show, on average, at least X-fold difference (log-scale) between the two groups of cells ## DE

base\_plot\_dir = "/Users/fin/Desktop/A2780S/Feature\_Plots"  
  
# Check and create the base directory if it does not exist  
if (!dir.exists(base\_plot\_dir)) {  
 dir.create(base\_plot\_dir, recursive = TRUE)  
}  
  
# Find the top 15 markers based on avg\_log2FC for each cluster  
top15 <- A2780S.markers %>%  
 group\_by(cluster) %>%  
 top\_n(n = 5, wt = avg\_log2FC)  
# Write the data to a CSV file  
write.csv(top15, paste0(base\_plot\_dir, "/top15\_markers\_resolution\_", clustree\_resolution, ".csv"), row.names = FALSE)  
  
# Generate a DotPlot for the top genes  
plot <- DotPlot(object = A2780S, features = unique(top15$gene), assay = "RNA") + coord\_flip()  
  
# Create a specific folder for this resolution  
resolution\_folder <- file.path(base\_plot\_dir, paste0("Resolution\_", clustree\_resolution))  
  
# Check if the directory exists; if not, create it  
if (!dir.exists(resolution\_folder)) {  
 dir.create(resolution\_folder, recursive = TRUE)  
}  
  
# Save the DotPlot to the resolution-specific folder  
dotplot\_filename <- file.path(resolution\_folder, paste0("top\_genes\_dotplot\_res", clustree\_resolution, ".png"))  
ggsave(dotplot\_filename, plot = plot, width = 10, height = 8, dpi = 300)  
  
# Loop through each gene in top15 to create and save a feature plot  
for (gene in unique(top15$gene)) {  
 p <- FeaturePlot(object = A2780S, features = gene) + NoLegend() + ggtitle(gene)  
   
 # Construct the file path, including the resolution folder and file name  
 feature\_plot\_filename <- file.path(resolution\_folder, paste0("feature\_plot\_", gene, ".png"))  
   
 # Save the plot to the file in the resolution-specific folder  
 ggsave(feature\_plot\_filename, plot = p, width = 8, height = 6, dpi = 300)  
}

# GetCountMatrix

## Reading Seurat Object and Extracting Data:

# Load the Seurat object  
A2780S <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep8\_.rds")  
  
# assay name (verify this is correct)  
matrix <- GetAssayData(object = A2780S, slot = "scale.data")

## Warning: The `slot` argument of `GetAssayData()` is deprecated as of SeuratObject 5.0.0.  
## ℹ Please use the `layer` argument instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_lifecycle\_warnings()` to see where this warning was  
## generated.

View(matrix)  
#data: This slot usually contains the normalized data. Normalization is done to account for differences in sequencing depth across cells and other technical factors. Normalized data is important for accurate comparison between cells.  
matrix1 <- as.data.frame(matrix)  
  
# Define the directory for saving outputs  
metadir1 <- "/Users/fin/Desktop/A2780S/"  
  
# Define the new subfolder for trajectory analysis within the base directory  
trajectory\_folder <- file.path(metadir1, "Trajectory")  
  
# Check if the trajectory directory exists; if not, create it  
if (!dir.exists(trajectory\_folder)) {  
 dir.create(trajectory\_folder, recursive = TRUE)  
}

## Writing Data to CSV and RDS

# Paths for the files to be saved in the new "Trajectory" folder  
csv\_path <- file.path(trajectory\_folder, "CORRECTcountmatrix.csv")  
rds\_path <- file.path(trajectory\_folder, "CORRECTcountmatrix.rds")  
  
# Save the dataframe as CSV  
fwrite(matrix1, file = csv\_path, row.names = TRUE, col.names = TRUE)  
  
# Save the dataframe as RDS  
saveRDS(matrix1, file = rds\_path)

## Preparing Data for Other Analysis Tools

# Write data as TSV for compatibility with other tools  
fwrite(matrix1, file = file.path(trajectory\_folder, "CORRECTcountmatrix.tsv"), row.names = TRUE, col.names = TRUE)

## Extract count matrix only for A2780S

# Define the directory for saving specific outputs  
metadir2 <- '/Users/fin/Desktop/A2780S/Trajectory'  
  
# Assuming '3449' is the correct number of columns for A2780S specific data  
dim(A2780S) #[1] 33538 3449

## [1] 33538 3449

A2780S\_matrix <- matrix1[, 1:3449]  
fwrite(A2780S\_matrix, file = file.path(metadir2, "CORRECTcountmatrix.csv"), row.names = TRUE, col.names = TRUE)  
fwrite(A2780S\_matrix, file = file.path(metadir2, "CORRECTcountmatrix.tsv"), row.names = TRUE, col.names = TRUE)

## Transpose count matrix and add the cluster label

# Transpose the matrix and convert it to a data frame  
matrix1.t <- t(matrix1)  
matrix2.t <- as.data.frame(matrix1.t)  
  
# Extract the cluster labels from the Seurat object's metadata  
A2780S\_clusters <- A2780S@meta.data[["seurat\_clusters"]] # Adjust if using a different name for Seurat object or clusters  
  
# Combine the transposed data frame with cluster labels  
A2780Scountmatrix <- cbind(matrix2.t, seurat\_clusters = A2780S\_clusters)  
  
# Define the directory for saving outputs  
metadir1 <- "/Users/fin/Desktop/A2780S/Trajectory/" # Adjust as necessary, perhaps to include a subdirectory  
  
# Save the combined dataframe as CSV  
csv\_path <- file.path(metadir1, "countmatrix\_with\_clusters.csv")  
fwrite(A2780Scountmatrix, file = csv\_path, row.names = TRUE, col.names = TRUE)  
fwrite(matrix2.t, file =file.path(metadir1, "Normalize\_express.csv" ),row.names = TRUE, col.names = TRUE)  
View(matrix2.t)

# Extract the names and clusters from the metadata of the Seurat object  
A2780S\_names <- A2780S@meta.data[0]  
A2780S\_clusters <- A2780S@meta.data[["seurat\_clusters"]] # Ensure this is the correct metadata column for clusters  
  
# Combine the names and cluster labels  
A2780S\_names\_and\_clusters <- cbind(A2780S\_names, seurat\_clusters = A2780S\_clusters)  
  
# Convert to a data frame  
A2780S\_names\_and\_clusters\_df <- as.data.frame(A2780S\_names\_and\_clusters)  
  
# Define the directory for saving outputs  
metadir1 <- "/Users/fin/Desktop/A2780S/Trajectory/" # Assuming you want it saved in the Trajectory folder  
  
# Save the dataframe as CSV  
csv\_path <- file.path(metadir1, "names\_and\_clusters.csv")  
fwrite(A2780S\_names\_and\_clusters\_df, file = csv\_path, row.names = TRUE, col.names = TRUE)

# Trajectory

# Monocle3

library(monocle3)

## Loading required package: Biobase

## Loading required package: BiocGenerics

##   
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:dplyr':  
##   
## combine, intersect, setdiff, union

## The following object is masked from 'package:SeuratObject':  
##   
## intersect

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, aperm, append, as.data.frame, basename, cbind,  
## colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,  
## get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,  
## match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,  
## Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,  
## table, tapply, union, unique, unsplit, which.max, which.min

## Welcome to Bioconductor  
##   
## Vignettes contain introductory material; view with  
## 'browseVignettes()'. To cite Bioconductor, see  
## 'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: SingleCellExperiment

## Loading required package: SummarizedExperiment

## Loading required package: MatrixGenerics

## Loading required package: matrixStats

##   
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':  
##   
## anyMissing, rowMedians

## The following object is masked from 'package:dplyr':  
##   
## count

##   
## Attaching package: 'MatrixGenerics'

## The following objects are masked from 'package:matrixStats':  
##   
## colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,  
## colCounts, colCummaxs, colCummins, colCumprods, colCumsums,  
## colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,  
## colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,  
## colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,  
## colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,  
## colWeightedMeans, colWeightedMedians, colWeightedSds,  
## colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,  
## rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,  
## rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,  
## rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,  
## rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,  
## rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,  
## rowWeightedMads, rowWeightedMeans, rowWeightedMedians,  
## rowWeightedSds, rowWeightedVars

## The following object is masked from 'package:Biobase':  
##   
## rowMedians

## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: S4Vectors

##   
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:data.table':  
##   
## first, second

## The following objects are masked from 'package:dplyr':  
##   
## first, rename

## The following object is masked from 'package:utils':  
##   
## findMatches

## The following objects are masked from 'package:base':  
##   
## expand.grid, I, unname

## Loading required package: IRanges

##   
## Attaching package: 'IRanges'

## The following object is masked from 'package:data.table':  
##   
## shift

## The following objects are masked from 'package:dplyr':  
##   
## collapse, desc, slice

## The following object is masked from 'package:sp':  
##   
## %over%

## Loading required package: GenomeInfoDb

##   
## Attaching package: 'SummarizedExperiment'

## The following object is masked from 'package:Seurat':  
##   
## Assays

## The following object is masked from 'package:SeuratObject':  
##   
## Assays

##   
## Attaching package: 'monocle3'

## The following objects are masked from 'package:Biobase':  
##   
## exprs, fData, fData<-, pData, pData<-

library(SeuratWrappers)  
# convert seurant obj to cell\_data\_set (cds) object for monocle3

# Get Marker and save file  
A2780S <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep8\_.rds")  
  
A2780S.markers <- FindAllMarkers(A2780S, min.pct = 0.25, logfc.threshold = 0.25)

## Calculating cluster 0

## Calculating cluster 1

## Calculating cluster 2

## Calculating cluster 3

## Calculating cluster 4

saveRDS(A2780S.markers, "/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_marker.rds")  
  
# Get marker into data frame  
A2780S\_marker <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_marker.rds")  
marker\_matrix <- as.data.frame(A2780S\_marker)  
  
# Get cell meta data  
A2780S\_meta <- A2780S@meta.data  
  
# Unprocess data load  
Unprocess\_data <- readRDS("/Users/fin/Desktop/A2780S/Unprocessed\_data/A2780S-CT\_S+R\_UnprocessedCohort\_SeuratObj\_.rds")  
Unprocess\_data <- Unprocess\_data@meta.data  
  
# Get Cell expression data  
A2780S\_expression <- read.csv("/Users/fin/Desktop/A2780S/Trajectory/CORRECTcountmatrix.csv")  
  
# Create Seurant Object  
seu.obj <- CreateSeuratObject(counts = A2780S\_expression)

## Warning: Data is of class data.frame. Coercing to dgCMatrix.

## Warning in matrix(data = as.numeric(x = x), ncol = nc): NAs introduced by  
## coercion

# convert seurant obj to cell\_data\_set (cds) object for monocle3  
cds <- as.cell\_data\_set(A2780S)

## Warning: Monocle 3 trajectories require cluster partitions, which Seurat does  
## not calculate. Please run 'cluster\_cells' on your cell\_data\_set object

## Assign Partition

Clusters are particularly useful while trying to assign cells to a certain type, because they are based on the similarity in gene expression. The relationships between different clusters are analysed to identify possible trajectories.

Partitions, meanwhile, are larger groups of cells that usually contain several clusters. Trajectory inference is performed only within one partition, so it is essential that all the cells that we want to analyse in pseudotime belong to the same partition.

#https://training.galaxyproject.org/training-material/topics/single-cell/tutorials/scrna-case\_monocle3-rstudio/tutorial.html#trajectory-inference  
# Create a uniform partition label for all cells  
reacreate.partition <- rep(1, length(cds@colData@rownames))  
names(reacreate.partition) <- rownames(cds@colData)  
reacreate.partition <- as.factor(reacreate.partition)  
  
# Update the cds object with the new partition label  
# Check if the structure of your object supports this assignment  
cds@clusters$UMAP$partitions <- reacreate.partition

## Assign the cluster info

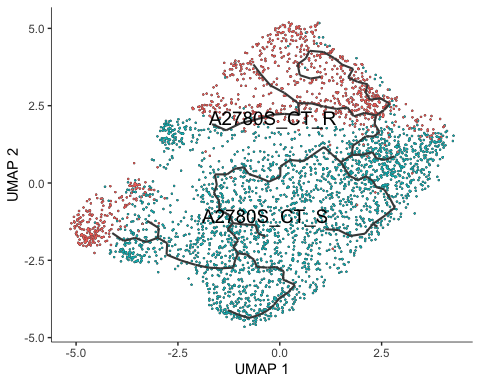
lsit\_cluster <- A2780S@active.ident # This retrieves the active identity classes (clusters)  
  
# Assign cluster labels from 'lsit\_cluster' to the UMAP cluster information slot within the 'cds' object.  
# This integrates clustering results directly with the UMAP dimensionality reduction in the 'cds' data structure,  
# allowing for enhanced visualization and analysis of clusters based on UMAP coordinates.  
cds@clusters$UMAP$clusters <- lsit\_cluster  
  
  
# Assign UMAP coordinate - cell embedding  
cds@int\_colData@listData$reducedDims$UMAP <- A2780S@reductions$umap@cell.embeddings

## Learn trajectory graph

cds <- learn\_graph(cds, use\_partition = FALSE)

## | | | 0% | |======================================================================| 100%

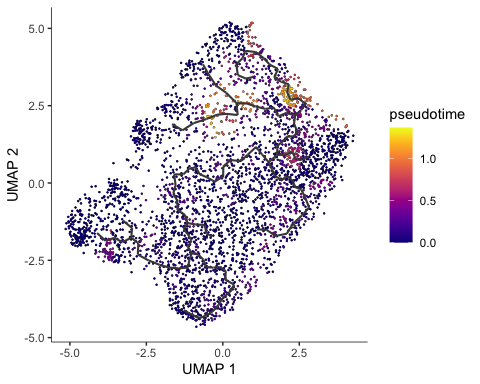
plot\_cells(cds,  
 color\_cells\_by = 'orig.ident',  
 label\_groups\_by\_cluster = FALSE,  
 label\_branch\_points = FALSE,  
 label\_roots =FALSE,  
 label\_leaves = FALSE,  
 group\_label\_size = 5)



## Order the cells in pseudotime

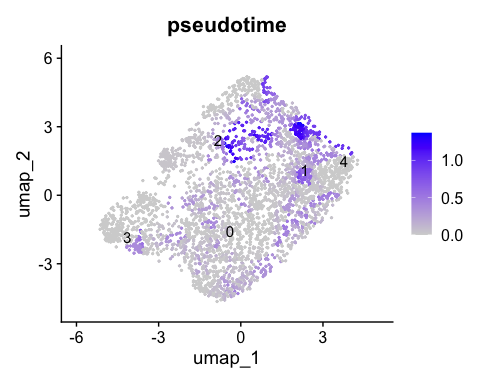
cds <- order\_cells(cds, reduction\_method = 'UMAP', root\_cells = colnames(cds[,clusters(cds) == 0])) #input start cluster from prior knowledge  
  
plot\_cells(cds,  
 color\_cells\_by = 'pseudotime',  
 label\_groups\_by\_cluster = FALSE,  
 label\_branch\_points = FALSE,  
 label\_roots = FALSE,  
 label\_leaves = FALSE)

## Cells aren't colored in a way that allows them to be grouped.

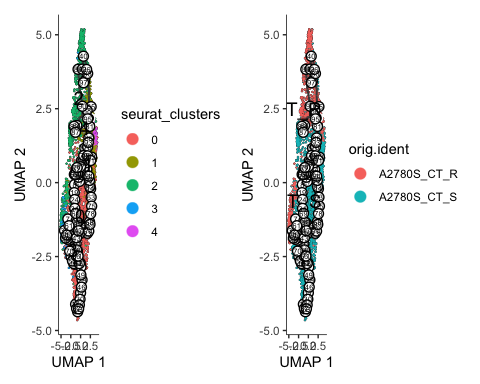


## Visualizing pseudotime in seurat

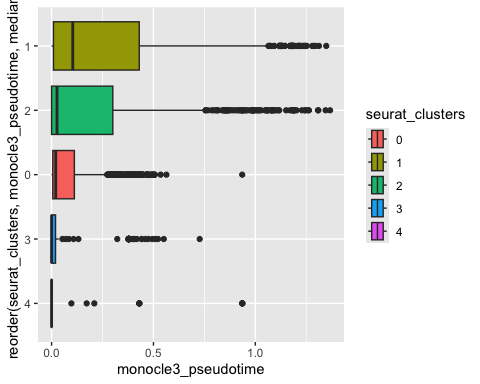
## Cells ordered by monocle3 pseudotime  
A2780S$pseudotime <- pseudotime(cds)  
Idents(A2780S) <- A2780S$seurat\_clusters  
FeaturePlot(A2780S, features = "pseudotime", label = T)



cluster.before.trajectory <- plot\_cells(cds,  
 color\_cells\_by = 'seurat\_clusters',  
 label\_groups\_by\_cluster = FALSE,  
 group\_label\_size = 5) + theme(legend.position = 'right')  
cluster.names <- plot\_cells(cds,  
 color\_cells\_by = 'orig.ident',  
 label\_groups\_by\_cluster = FALSE,  
 group\_label\_size = 5) + theme(legend.position = 'right')  
  
cluster.before.trajectory | cluster.names



cds$monocle3\_pseudotime <- pseudotime(cds)  
data.pseudo <- as.data.frame(colData(cds))  
  
ggplot(data.pseudo, aes(monocle3\_pseudotime, reorder(seurat\_clusters, monocle3\_pseudotime, median), fill = seurat\_clusters)) +  
 geom\_boxplot()



# Slingshot

#https://bustools.github.io/BUS\_notebooks\_R/slingshot.html  
  
library(slingshot, quietly = FALSE)

## Loading required package: princurve

## Loading required package: TrajectoryUtils

library(SingleCellExperiment)  
library(slingshot)

A2780S <- readRDS("/Users/fin/Desktop/A2780S/Process\_data/A2780S\_S+R\_QC-Subset-Cohort\_SeuratObj\_thruStep7\_.rds")  
A2780S <- FindNeighbors(object=A2780S, dims = 1:50)

## Computing nearest neighbor graph

## Computing SNN

A2780S <- FindClusters(object=A2780S, resolution = 0.15)

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 3449  
## Number of edges: 188287  
##   
## Running Louvain algorithm...  
## Maximum modularity in 10 random starts: 0.8989  
## Number of communities: 5  
## Elapsed time: 0 seconds

A2780S <- RunUMAP(object = A2780S, reduction = "pca", dims = 1:50)

## 23:48:20 UMAP embedding parameters a = 0.9922 b = 1.112

## Found more than one class "dist" in cache; using the first, from namespace 'spam'

## Also defined by 'BiocGenerics'

## 23:48:20 Read 3449 rows and found 50 numeric columns

## 23:48:20 Using Annoy for neighbor search, n\_neighbors = 30

## Found more than one class "dist" in cache; using the first, from namespace 'spam'

## Also defined by 'BiocGenerics'

## 23:48:20 Building Annoy index with metric = cosine, n\_trees = 50

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

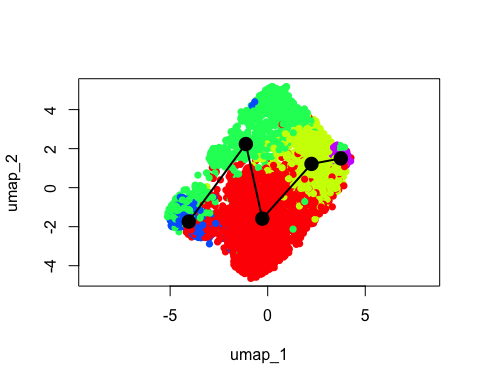
## [----|----|----|----|----|----|----|----|----|----|

## \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|  
## 23:48:20 Writing NN index file to temp file /var/folders/ys/hhllw\_jj35q9vnv51l9bbgq00000gn/T//RtmpF2wotI/file19316bc48e4b  
## 23:48:20 Searching Annoy index using 1 thread, search\_k = 3000  
## 23:48:21 Annoy recall = 100%  
## 23:48:29 Commencing smooth kNN distance calibration using 1 thread with target n\_neighbors = 30  
## 23:48:39 Initializing from normalized Laplacian + noise (using RSpectra)  
## 23:48:39 Commencing optimization for 500 epochs, with 158342 positive edges  
## 23:48:50 Optimization finished

A2780S.markers <- FindAllMarkers(A2780S, min.pct = 0.25, logfc.threshold = 0.25)

## Calculating cluster 0  
## Calculating cluster 1  
## Calculating cluster 2  
## Calculating cluster 3  
## Calculating cluster 4

# Assuming your Seurat object is named seuratObj  
sce <- as.SingleCellExperiment(A2780S)  
# Run Slingshot  
sce <- slingshot(sce, clusterLabels = 'seurat\_clusters', reducedDim = 'UMAP')  
  
plot(reducedDims(sce)$UMAP, col = rainbow(length(unique(sce$seurat\_clusters)))[sce$seurat\_clusters], pch = 16, asp = 1)  
lines(SlingshotDataSet(sce), lwd = 2, type = 'lineages')



# Extract Slingshot output  
slingshot\_output <- SlingshotDataSet(sce)  
  
# Pseudotime values  
pseudotime\_values <- slingPseudotime(slingshot\_output, na = FALSE)  
  
# Lineage information  
lineages <- slingCurves(slingshot\_output)  
plot(pseudotime\_values, col = rainbow(length(unique(sce$seurat\_clusters)))[sce$seurat\_clusters], pch = 16, xlab = "Pseudotime", ylab = "Cells")

