A dopamine-induced gene expression signature regulates neuronal function and cocaine response - Adult NAc Analysis

## Load Libraries

To begin, all packages/libraries that will be needed for the analysis will be loaded

suppressPackageStartupMessages(library("dplyr"))  
suppressPackageStartupMessages(library("Seurat"))  
suppressPackageStartupMessages(library("svMisc"))  
suppressPackageStartupMessages(library("cowplot"))  
suppressPackageStartupMessages(library("ggplot2"))  
suppressPackageStartupMessages(library("pheatmap"))  
suppressPackageStartupMessages(library("reshape2"))  
suppressPackageStartupMessages(library("gridExtra"))  
suppressPackageStartupMessages(library("RColorBrewer"))

## Data Import and Quality Control

To begin the analysis, CellRanger output will be loaded into the local environment using the Read10X function within the Seurat package. For this dataset, 4 GEM wells were utlized, one well for each Sex and treatment group.

Fem\_Sal\_data <- Read10X(data.dir = "~/Bioinformatics/JD0037/Female\_Saline/")  
Fem\_Coc\_data <- Read10X(data.dir = "~/Bioinformatics/JD0037/Female\_Cocaine/")  
Male\_Sal\_data <- Read10X(data.dir = "~/Bioinformatics/JD0037/Male\_Saline/")  
Male\_Coc\_data <- Read10X(data.dir = "~/Bioinformatics/JD0037/Male\_Cocaine/")

A Seurat object is essential to the following analysis. This object will contain raw data, log-normalized data, cellular identities, and other essential information. To create this object, the CreateSeuratObject function within the Seurat package will be used. This function provides two important options when creating the object, min.cells and min.features. These options provide some intital quality control in which genes that are not within a specified number of cells, or cells that do not have a specified number of features are dropped from further analysis. Here, these options will not be utilized as cells will be dropped from the analysis at a later step.

Fem\_Sal <- CreateSeuratObject(counts = Fem\_Sal\_data,min.cells = 1,min.features = 1)

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

Fem\_Coc <- CreateSeuratObject(counts = Fem\_Coc\_data,min.cells = 1,min.features = 1)

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

Male\_Sal <- CreateSeuratObject(counts = Male\_Sal\_data,min.cells = 1,min.features = 1)

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

Male\_Coc <- CreateSeuratObject(counts = Male\_Coc\_data,min.cells = 1,min.features = 1)

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

An important step within single cell RNA-sequencing analysis is removing cells with a high percentage of reads mapping to the mitochondrial genome. A high percentage of cells mapping to the mitochondrial genome is indicative of poor cell health. Here, the PercentageFeatureSet command will be used to calculate the percentage of total reads mapping to the mitochondrial genome. This function will create a feature called percent\_mito within each Seurat object.

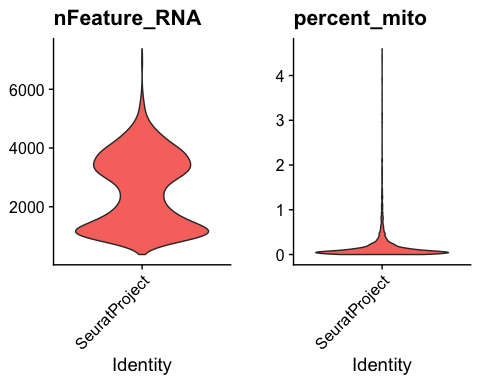
Fem\_Sal <- PercentageFeatureSet(Fem\_Sal, pattern = "^Mt-", col.name = "percent\_mito")  
Fem\_Coc <- PercentageFeatureSet(Fem\_Coc, pattern = "^Mt-", col.name = "percent\_mito")  
Male\_Sal <- PercentageFeatureSet(Male\_Sal, pattern = "^Mt-", col.name = "percent\_mito")  
Male\_Coc <- PercentageFeatureSet(Male\_Coc, pattern = "^Mt-", col.name = "percent\_mito")

After calculating the percentage of total reads mapping to the mitochondrial genome, quality control measures can be taken to remove cells from further analysis that do not meet speciic requirement. For the following analysis, only those cells with more than 200 genes and less than 5% of total reads mapping to the mitochondrial genome will be kept.

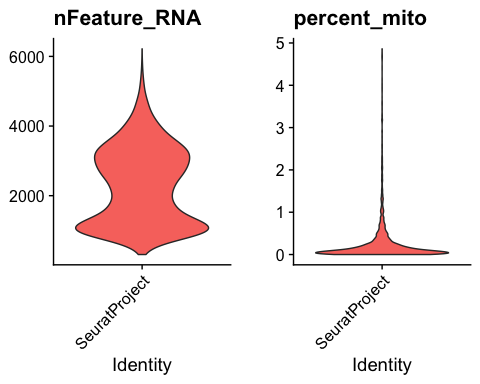
Fem\_Sal <- subset(x = Fem\_Sal, subset = nFeature\_RNA > 200 & percent\_mito < 5)  
Fem\_Coc <- subset(x = Fem\_Coc, subset = nFeature\_RNA > 200 & percent\_mito < 5)   
Male\_Sal <- subset(x = Male\_Sal, subset = nFeature\_RNA > 200 & percent\_mito < 5)   
Male\_Coc <- subset(x = Male\_Coc, subset = nFeature\_RNA > 200 & percent\_mito < 5)

Now that quality control measures have been taken, the distribution of number of genes and percentage of total reads mapping to the mitochondrial genome can be visualized with a violin plot.

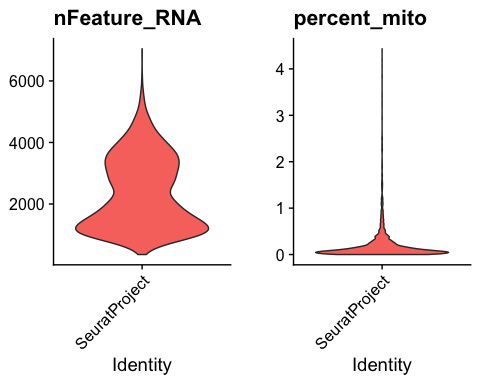
VlnPlot(Fem\_Sal, features = c("nFeature\_RNA","percent\_mito"), ncol = 2,pt.size = 0)



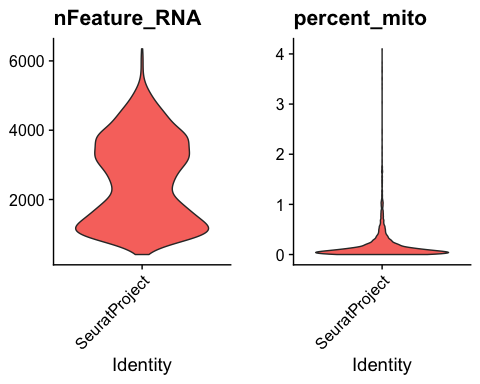
VlnPlot(Fem\_Coc, features = c("nFeature\_RNA","percent\_mito"), ncol = 2,pt.size = 0)



VlnPlot(Male\_Sal, features = c("nFeature\_RNA","percent\_mito"), ncol = 2,pt.size = 0)



VlnPlot(Male\_Coc, features = c("nFeature\_RNA","percent\_mito"), ncol = 2,pt.size = 0)



## Data Normalization

Now that the proper quality control measures have been taken, the raw data can be normalized using the Log-normalization method with a scaling factor of 10000.

Fem\_Sal <- NormalizeData(Fem\_Sal, normalization.method = "LogNormalize", scale.factor = 10000)  
Fem\_Coc <- NormalizeData(Fem\_Coc, normalization.method = "LogNormalize", scale.factor = 10000)  
Male\_Sal <- NormalizeData(Male\_Sal, normalization.method = "LogNormalize", scale.factor = 10000)  
Male\_Coc <- NormalizeData(Male\_Coc, normalization.method = "LogNormalize", scale.factor = 10000)

Following data normalization, 2000 of the most variable features within each dataset are identified. According to the Seurat documentation, identifying variable features helps highlight biological variation later.

Fem\_Sal <- FindVariableFeatures(Fem\_Sal, selection.method = "vst", nfeatures = 2000)  
Fem\_Coc <- FindVariableFeatures(Fem\_Coc, selection.method = "vst", nfeatures = 2000)  
Male\_Sal <- FindVariableFeatures(Male\_Sal, selection.method = "vst", nfeatures = 2000)  
Male\_Coc <- FindVariableFeatures(Male\_Coc, selection.method = "vst", nfeatures = 2000)

## Data Integration

As stated above, each GEM well for this data set contains one sex and treatment group. Therefore, before integrating all data, additional metadata attributes for sex and treatment will be added.

#Add treatment information  
Fem\_Sal$Stim <- "Saline"  
Fem\_Coc$Stim <- "Cocaine"  
Male\_Sal$Stim <- "Saline"  
Male\_Coc$Stim <- "Cocaine"  
#Add sex information  
Fem\_Sal$Sex <- "Female"  
Fem\_Coc$Sex <- "Female"  
Male\_Sal$Sex <- "Male"  
Male\_Coc$Sex <- "Male"

Here, integration anchors are identified using the FindIntegrationAnchors function. These anchors are used to integrate the data. Following the identification of anchors, the data is integrated using the IntegrateData function.

All\_Groups\_log <- FindIntegrationAnchors(object.list = list(Fem\_Sal,Fem\_Coc,Male\_Sal,Male\_Coc), dims = 1:17)

## Warning in CheckDuplicateCellNames(object.list = object.list): Some cell names  
## are duplicated across objects provided. Renaming to enforce unique cell names.

## Computing 2000 integration features

## Scaling features for provided objects

## Finding all pairwise anchors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

## Found 9939 anchors

## Filtering anchors

## Retained 7111 anchors

## Extracting within-dataset neighbors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

## Found 8634 anchors

## Filtering anchors

## Retained 6405 anchors

## Extracting within-dataset neighbors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

## Found 10051 anchors

## Filtering anchors

## Retained 7647 anchors

## Extracting within-dataset neighbors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

## Found 8089 anchors

## Filtering anchors

## Retained 6557 anchors

## Extracting within-dataset neighbors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

## Found 9523 anchors

## Filtering anchors

## Retained 7951 anchors

## Extracting within-dataset neighbors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

## Found 7926 anchors

## Filtering anchors

## Retained 6519 anchors

## Extracting within-dataset neighbors

All\_Groups\_log <- IntegrateData(anchorset = All\_Groups\_log,dims = 1:17)

## Merging dataset 4 into 2

## Extracting anchors for merged samples

## Finding integration vectors

## Finding integration vector weights

## Integrating data

## Merging dataset 3 into 2 4

## Extracting anchors for merged samples

## Finding integration vectors

## Finding integration vector weights

## Integrating data

## Merging dataset 1 into 2 4 3

## Extracting anchors for merged samples

## Finding integration vectors

## Finding integration vector weights

## Integrating data

Following integration of the data, a standard workflow will be used to build a UMAP containing all cells. This UMAP will be essential to identifying cell types within the adult rat Nucleus Accumbens. First, data is scaled and then run through a series of dimensionality reduction techniques before identifying clusters for building the UMAP.

All\_Groups\_log <- ScaleData(All\_Groups\_log,verbose = FALSE)  
All\_Groups\_log <- RunPCA(All\_Groups\_log,npcs = 17 ,verbose = FALSE)  
All\_Groups\_log <- RunUMAP(All\_Groups\_log, reduction = "pca", dims = 1:17)  
All\_Groups\_log <- FindNeighbors(All\_Groups\_log, reduction = "pca", dims = 1:17)

## Computing nearest neighbor graph

## Computing SNN

All\_Groups\_log <- FindClusters(All\_Groups\_log, resolution = 0.2)

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 15631  
## Number of edges: 680995  
##   
## Running Louvain algorithm...  
## Maximum modularity in 10 random starts: 0.9671  
## Number of communities: 16  
## Elapsed time: 2 seconds

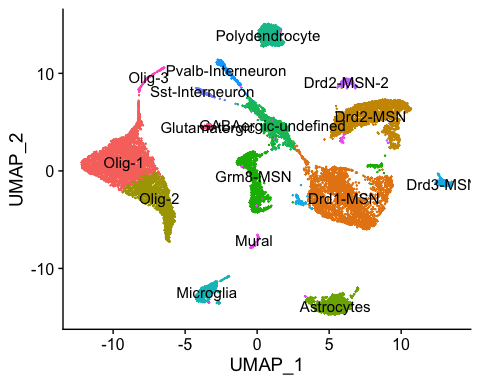
Now that clusters have been identified, the arbitrary numbers can be replaced with cell type identities. The cell type identities can then be added to the metadata and set as the identity for each cell.

All\_Groups\_log <- RenameIdents(object = All\_Groups\_log,  
 "0" = "Olig-1",  
 "1" = "Drd1-MSN",  
 "2" = "Drd2-MSN",  
 "3" = "Olig-2",  
 "4" = "Astrocytes",  
 "5" = "Grm8-MSN",  
 "6" = "GABAergic-undefined",  
 "7" = "Polydendrocyte",  
 "8" = "Microglia",  
 "9" = "Drd3-MSN",  
 "10" = "Pvalb-Interneuron",  
 "11" = "Sst-Interneuron",  
 "12" = "Drd2-MSN-2",  
 "13" = "Mural",  
 "14" = "Olig-3",  
 "15" = "Glutamatergic")  
All\_Groups\_log$CellType <- Idents(All\_Groups\_log)  
Idents(All\_Groups\_log) <- All\_Groups\_log$CellType

Finally, the UMAP can be visualized.

DimPlot(object = All\_Groups\_log,reduction = "umap",label = TRUE) + NoLegend()

## Warning: Using `as.character()` on a quosure is deprecated as of rlang 0.3.0.  
## Please use `as\_label()` or `as\_name()` instead.  
## This warning is displayed once per session.



## Identification of Differentially Expressed Genes

To control input, Seurat source code was modified to enable the testing of every gene within our dataset. To begin, cellular identities are modified to show both celltype and treatment condition. To begin, a list is generated that will contain DEGs for that specific cell type. Then, every element of that list is given a name that corresponds to the specific cell type. This section of the analysis takes ~1.5-2 hours to complete. Thus, when compiling this word document the code was not run.

Cluster\_Lists <- vector(mode = "list",length = length(unique(All\_Groups\_log$CellType)))  
names(Cluster\_Lists) <- as.character(unique(All\_Groups\_log$CellType))

Using modified source code, the log2 fold change can be calculated for each gene between cocaine and saline treated cells.

for(i in names(Cluster\_Lists)){  
 #Print the element so we can actually see the cell type as we loop through  
 print(i)  
 #Create a dataframe in each element of the list where FC will be the log2FC and gene will be the name  
 Cluster\_Lists[[i]] <- data.frame(log2FC = ( log2(rowMeans(expm1(x = as.matrix(GetAssayData(object = All\_Groups\_log,slot = "data",assay = "RNA")[,WhichCells(object = All\_Groups\_log,idents = paste(i,"Cocaine",sep = "\_"))])))) -  
 log2(rowMeans(expm1(x = as.matrix(GetAssayData(object = All\_Groups\_log,slot = "data",assay = "RNA")[,WhichCells(object = All\_Groups\_log,idents = paste(i,"Saline",sep = "\_"))]))))),  
 gene = row.names(as.matrix(GetAssayData(object = All\_Groups\_log,slot = "data",assay= "RNA"))[,WhichCells(object = All\_Groups\_log,idents = paste0(i,"\_Cocaine"))]))  
 #Remove infinite values  
 Cluster\_Lists[[i]] <- Cluster\_Lists[[i]][!is.infinite(Cluster\_Lists[[i]]$FC),]  
 #remove NaN values  
 Cluster\_Lists[[i]] <- Cluster\_Lists[[i]][!is.nan(Cluster\_Lists[[i]]$FC),]  
}

After calculating log2 fold change of each gene, testing for significant differentially expressed genes can be completed. Here a wilcoxon ranked sum test is used.

for(i in names(Cluster\_Lists)){  
 #Create a data.frame where the rownames are the cells within the clusters   
 group.info <- data.frame(row.names = c(WhichCells(object = All\_Groups\_log,idents = paste(i,"Cocaine",sep = "\_")),  
 WhichCells(object = All\_Groups\_log,idents = paste(i,"Saline",sep = "\_"))))  
 #Create a group column where cocaine is Group1 and saline is group2. The way to do that is to search the rows names for Cocaine and Saline cell identities  
 group.info[WhichCells(object = All\_Groups\_log,idents = paste(i,"Cocaine",sep = "\_")), "group"] <- "Group1"  
 group.info[WhichCells(object = All\_Groups\_log,idents = paste(i,"Saline",sep = "\_")), "group"] <- "Group2"  
 #Make the group column a factor so it can be tested with the wilcox   
 group.info[, "group"] <- factor(x = group.info[, "group"])  
 #Create an expression matrix where the row.names are the genes found within that cluster - do this by running row.names(Cluster\_Lists[[i]])  
 #the columns should be row.names(group.info) which are the cells within that cluster   
 #Drop maintains the structure of the matrix when subsetting, which is needed when pulling the expression matrix  
 #Pull counts from log-normalized matrix  
 data.use <- GetAssayData(object = All\_Groups\_log,slot = "data",assay = "RNA")[row.names(Cluster\_Lists[[i]]), rownames(x = group.info), drop = FALSE]  
 #Loop through the rows in the expression matrix which are the genes  
 for(l in 1:length(row.names(data.use))){  
 #Print the progress of the loop  
 progress(l,max.value = length(row.names(data.use)))  
 #Calculate p-values for every gene   
 #Do this by entering the element of the list for the specific cluster that we are running. i will always be the cell type  
 #row.names(data.use)[l] will be the gene name  
 #wilcox.test will test gene by group to see if there are any differences. Then by adding $p.value we can pull only the p-value from the test   
 Cluster\_Lists[[i]][row.names(data.use)[l],"p.val"] <- wilcox.test(data.use[row.names(data.use)[l], ] ~ group.info[,"group"])$p.value  
 }  
}

Following statistical testing, p-values are adjusted using a bonferroni correction for the total number of genes within the dataset.

for(i in 1:length(Cluster\_Lists)){  
 #Adjust the p-value with a bonferroni correction  
 Cluster\_Lists[[i]]$adj.p.val <- p.adjust(p = Cluster\_Lists[[i]]$p.val,method = "bonferroni",n = nrow(GetAssayData(object = All\_Groups\_log,slot = "data",assay = "RNA")))  
}

## Subclustering of Drd1-MSNs

To begin subclustering, the Seurat object is subset for only Drd1-MSNs.

# Subset Drd1-MSNs into new Seurat object.   
Drd1 <- subset(All\_Groups\_log, idents = c("Drd1-MSN"))

Following the subset, a new UMAP is generated using the same dimensionality reduction techniques discussed above.

# Generate new UMAP with only Drd1-MSN subcluster. This identifies 4 subclusters.   
Drd1 <- RunUMAP(Drd1, reduction = "pca", dims = 1:17)  
Drd1 <- FindNeighbors(Drd1, reduction = "pca", dims = 1:17)

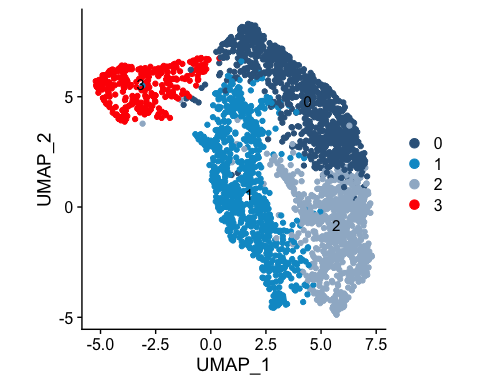
## Computing nearest neighbor graph

## Computing SNN

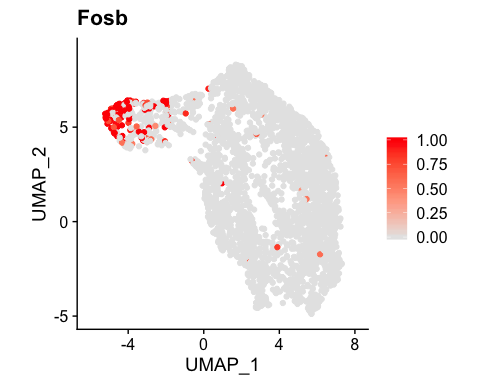
Drd1 <- FindClusters(Drd1, resolution = 0.2)

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck  
##   
## Number of nodes: 2748  
## Number of edges: 107141  
##   
## Running Louvain algorithm...  
## Maximum modularity in 10 random starts: 0.8949  
## Number of communities: 4  
## Elapsed time: 0 seconds

DimPlot(Drd1, reduction = "umap", pt.size =1.5, label = TRUE,   
 cols = c("steelblue4", "deepskyblue3", "slategray3", "red")) + coord\_fixed(ratio = 1)

 With single nuclei RNA sequencing technology, we now have the ability to interrogate the expression of specific genes within single cells. Here the expression of FosB, a canonical immediate early genes, is plotted on top of the UMAP created in the last step.

# Plot specific features  
DefaultAssay(Drd1) <- "RNA"  
FeaturePlot(Drd1, features = c("Fosb"), pt.size =1.5, max.cutoff = 1, cols = c("gray90", "red")) +  
 coord\_fixed(ratio = 1)

 To identify genes that predict activated cluster membership, ROC analysis is used. Following ROC testing, the top ten genes marking active cluster membership are written to a csv.

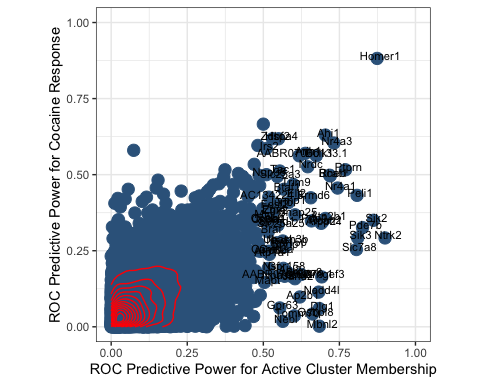
# Use ROC analysis to identify genes that predict activated cluster membership. Write file with predictive genes.   
Drd1\_Markers <- FindMarkers(object = Drd1, ident.1 = "3", ident.2 = NULL, test.use = "roc",  
 assay = "RNA", slot = "data", logfc.threshold = 0, min.pct = 0.1)  
Drd1\_Markers\_up <- subset(Drd1\_Markers, subset=(myAUC >0.5))  
write.csv(Drd1\_Markers, "Marker\_ROC.csv")

Now, genes that are cocaine responsive will be identified using ROC analysis. These genes will then be written out to a CSV.

# Make Celltype\_Stim identity for within-cluster testing  
Drd1$Celltype\_Stim <- paste(Idents(Drd1), Drd1$Stim, sep = "\_")  
# Find cocaine-responsive genes in activated subcluster. Write file with results.   
Idents(Drd1) <- Drd1$Celltype\_Stim  
Three\_Cocaine <- FindMarkers(object = Drd1, ident.1 = "3\_Cocaine",   
 ident.2 = "3\_Saline", test.use = "roc",   
 assay = "RNA", slot = "data", logfc.threshold = 0, min.pct = 0.1)  
write.csv(Three\_Cocaine, "Responder\_ROC.csv")

After testing for both active and responsive genes, a dataframe containing genes from both tests are created. Finally, the predictive power of each gene for both responsiveness and activity are plotted.

# Make matrix with Drd1-MSN ROC values only for activated cluster  
Master <- merge(Drd1\_Markers,   
 Three\_Cocaine,   
 by.x = "row.names",   
 by.y = "row.names")  
  
# Make plot of genes that predict activated cluster membership and genes that predict cocaine or saline treatment.   
ggplot(Master, aes(power.x, power.y)) +  
 geom\_point(color = "steelblue4", size = 4) +  
 geom\_density2d(color = "red") +  
 geom\_text(aes(label=ifelse(power.x>0.5, as.character(Row.names),'')), nudge\_x = 0.01, nudge\_y = 0.01, size = 3) +  
 theme\_bw() +  
 xlim(0,1) +  
 ylim(0,1) +  
 ylab("ROC Predictive Power for Cocaine Response") +  
 xlab("ROC Predictive Power for Active Cluster Membership")+  
 coord\_equal()



sessionInfo()

## R version 3.6.0 (2019-04-26)  
## Platform: x86\_64-apple-darwin15.6.0 (64-bit)  
## Running under: macOS Mojave 10.14.6  
##   
## Matrix products: default  
## BLAS: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib  
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib  
##   
## locale:  
## [1] en\_US.UTF-8/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8  
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## other attached packages:  
## [1] RColorBrewer\_1.1-2 gridExtra\_2.3 reshape2\_1.4.3 pheatmap\_1.0.12   
## [5] ggplot2\_3.2.1 cowplot\_1.0.0 svMisc\_1.1.0 Seurat\_3.0.2   
## [9] dplyr\_0.8.3   
##   
## loaded via a namespace (and not attached):  
## [1] tsne\_0.1-3 nlme\_3.1-142 bitops\_1.0-6   
## [4] httr\_1.4.1 sctransform\_0.2.0 tools\_3.6.0   
## [7] backports\_1.1.5 R6\_2.4.1 irlba\_2.3.3   
## [10] KernSmooth\_2.23-16 lazyeval\_0.2.2 colorspace\_1.4-1   
## [13] withr\_2.1.2 npsurv\_0.4-0 tidyselect\_0.2.5   
## [16] compiler\_3.6.0 plotly\_4.9.1 labeling\_0.3   
## [19] caTools\_1.17.1.3 scales\_1.0.0 lmtest\_0.9-37   
## [22] ggridges\_0.5.1 pbapply\_1.4-2 stringr\_1.4.0   
## [25] digest\_0.6.22 rmarkdown\_1.17 R.utils\_2.9.0   
## [28] pkgconfig\_2.0.3 htmltools\_0.4.0 bibtex\_0.4.2   
## [31] htmlwidgets\_1.5.1 rlang\_0.4.1 zoo\_1.8-6   
## [34] jsonlite\_1.6 ica\_1.0-2 gtools\_3.8.1   
## [37] R.oo\_1.23.0 magrittr\_1.5 Matrix\_1.2-17   
## [40] Rcpp\_1.0.3 munsell\_0.5.0 ape\_5.3   
## [43] reticulate\_1.13 lifecycle\_0.1.0 R.methodsS3\_1.7.1   
## [46] stringi\_1.4.3 yaml\_2.2.0 gbRd\_0.4-11   
## [49] MASS\_7.3-51.4 gplots\_3.0.1.1 Rtsne\_0.15   
## [52] plyr\_1.8.4 grid\_3.6.0 parallel\_3.6.0   
## [55] gdata\_2.18.0 listenv\_0.7.0 ggrepel\_0.8.1   
## [58] crayon\_1.3.4 lattice\_0.20-38 splines\_3.6.0   
## [61] SDMTools\_1.1-221.1 zeallot\_0.1.0 knitr\_1.26   
## [64] pillar\_1.4.2 igraph\_1.2.4.1 future.apply\_1.3.0   
## [67] codetools\_0.2-16 glue\_1.3.1 evaluate\_0.14   
## [70] lsei\_1.2-0 metap\_1.1 data.table\_1.12.8   
## [73] vctrs\_0.2.0 png\_0.1-7 Rdpack\_0.11-0   
## [76] gtable\_0.3.0 RANN\_2.6.1 purrr\_0.3.3   
## [79] tidyr\_1.0.0 future\_1.15.0 assertthat\_0.2.1   
## [82] xfun\_0.11 rsvd\_1.0.2 survival\_3.1-7   
## [85] viridisLite\_0.3.0 tibble\_2.1.3 cluster\_2.1.0   
## [88] globals\_0.12.4 fitdistrplus\_1.0-14 ROCR\_1.0-7