Code for Zhu et al., “A comprehensive temporal patterning gene network in Drosophila medulla”

Dave Zhao

01 January, 2022

1 Introduction

The following analyses are presented roughly in the order in which they appear in the manuscript.

This analysis uses Seurat version 3.2.3. To downgrade Seurat if necessary:

```
## devtools::install_version("Seurat", version = "3.2.3", repos = "http://cran.us.r-p
roject.org")
```

2 Load libraries

```
rm(list = ls())
library(dplyr)
library(Seurat)
library(cowplot)
library(monocle3)
library(ggplot2)
library(mgcv)
library(parallel)
library(future)
library(pheatmap)
library(data.table)
library(readxl)
set.seed(1)
```

3 Load data

Load and preprocess data. Filter out cells with no `dpn` expression as well as cells with more than 10% mitochondrial content.

```
## load datasets separately
counts = list(Read10X("~/data/drosophila_nb/july_2019_run/MNB_1/outs/filtered_feature
_bc_matrix"),
              Read10X("~/data/drosophila_nb/dec_2019_run/MNB_2/outs/filtered_feature_
bc_matrix"))

counts = lapply(counts, function(x) {

  ## filter cells by dpn expression level and mitochondrial mRNA content
  dpn = x[which(rownames(x) == "dpn"),]
  feat.mito = grep("^mt:", rownames(x), value = TRUE)
  percent.mito = Matrix::colSums(x[feat.mito, ]) /
    Matrix::colSums(x)

  x = x[, dpn > 0 & percent.mito < 0.1]
  return(x)

})
```

4 Integrate data

Integrate data following vignette from here (https://satijalab.org/seurat/v3.1/immune_alignment.html).

```
## prepare for integration
mnbs = list(CreateSeuratObject(counts[[1]], project = "jul"),
            CreateSeuratObject(counts[[2]], project = "dec"))

mnbs_prep = lapply(mnbs, function(x) {
  x = NormalizeData(x)
  x = FindVariableFeatures(x, selection.method = "vst")
  x = PercentageFeatureSet(x, pattern = "^mt:", col.name= "percent.mt")
  return(x)
})

## integrate, make sure integrated features contain genes of interest
mnb_anchors = FindIntegrationAnchors(mnbs_prep)
```

```
## Computing 2000 integration features
```

```
## Scaling features for provided objects
```

```
## Finding all pairwise anchors
```

```
## Running CCA
```

```
## Merging objects
## Finding neighborhoods
```

```
## Finding anchors
```

```
##  Found 3030 anchors
```

```
## Filtering anchors
```

```
##  Retained 1954 anchors
```

```
feat_int = rownames(mnbs[[1]]) ## all genes
mnb = IntegrateData(anchorset = mnb_anchors, features.to.integrate = feat_int)
```

```
## Merging dataset 1 into 2
```

```
## Extracting anchors for merged samples
```

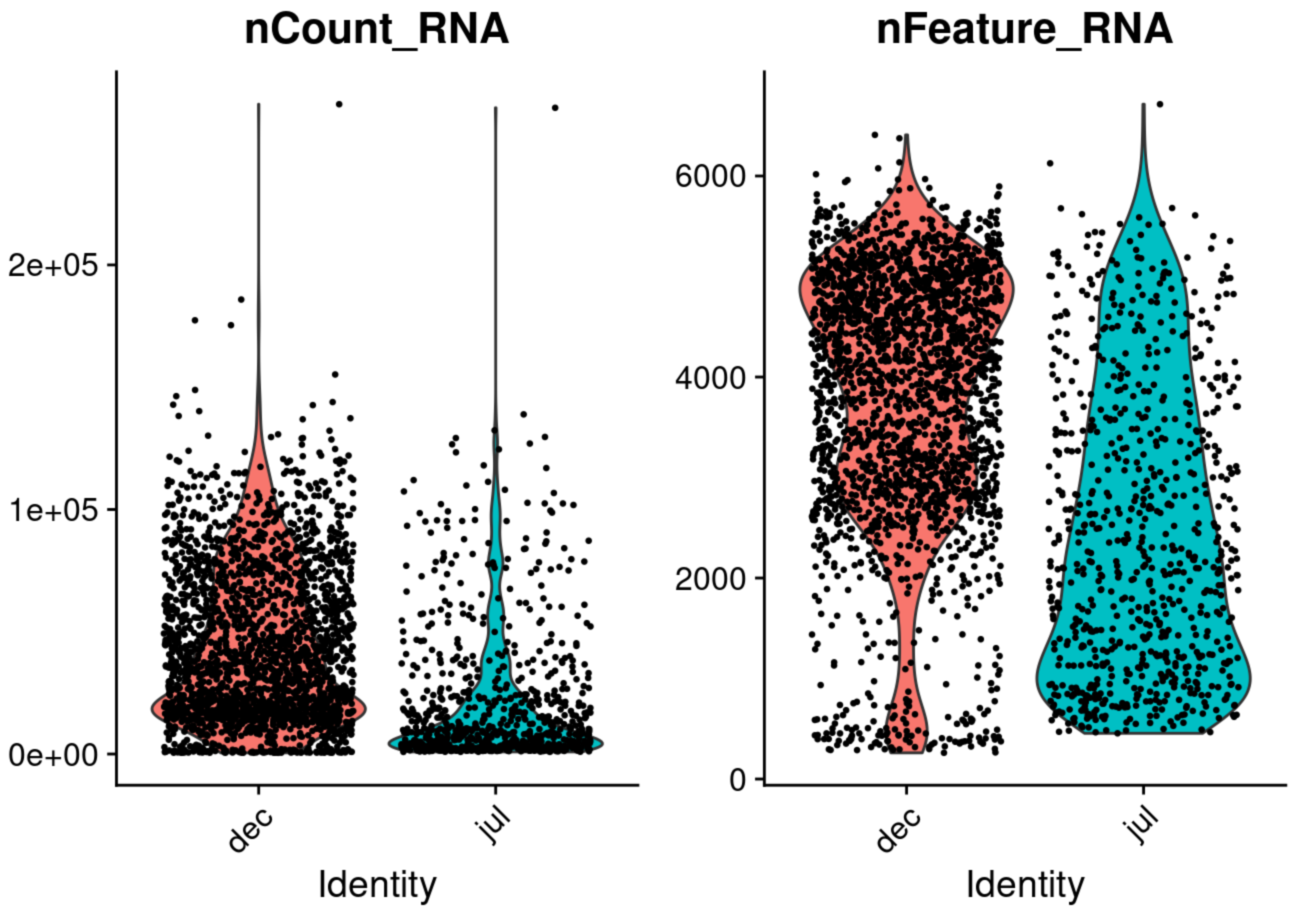
```
## Finding integration vectors
```

```
## Finding integration vector weights
```

```
## Integrating data
```

```
## Warning: Adding a command log without an assay associated with it
```

```
DefaultAssay(mnb) = "integrated"

## quality control
VlnPlot(mnb, features = c("nCount_RNA", "nFeature_RNA"), pt.size = 0.5)
```



```
keep = WhichCells(mnb, expression = nCount_RNA <= 1.7e5)
mnb = subset(mnb, cells = keep)

## scale data
plan("multiprocess", workers = 2)
```

```
## Warning in supportsMulticoreAndRStudio(...): [ONE-TIME WARNING] Forked
## processing ('multicore') is not supported when running R from RStudio
## because it is considered unstable. For more details, how to control forked
## processing or not, and how to silence this warning in future R sessions, see ?
## parallelly::supportsMulticore
```

```
mnb = ScaleData(mnb)
```

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

```
## summarize data
dim(mnb)
```

```
## [1] 17753  3074
```

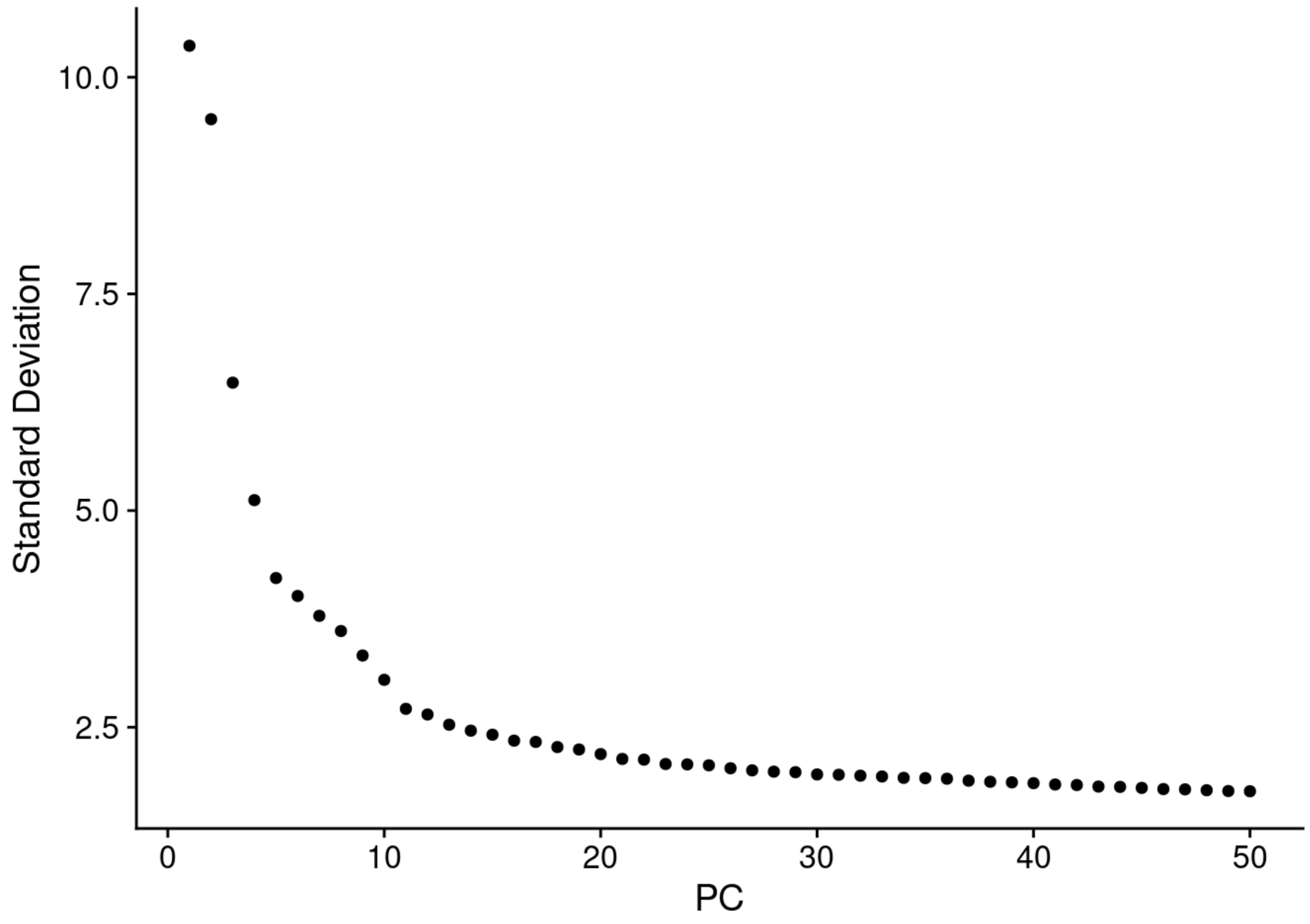
```
summary(mnb[["nFeature_RNA"]])
```

```
##   nFeature_RNA
##  Min.   : 261
##  1st Qu.:2556
##  Median :3682
##  Mean   :3467
##  3rd Qu.:4707
##  Max.   :6409
```

5 Cluster and visualize

Find clusters. Picked resolution to find about 15 clusters. This analysis corresponds to Figure 1C.

```
DefaultAssay(mnb) = "integrated"

## dimension reduction
npcs = 50
mnb = RunPCA(mnb, npcs = npcs, verbose = FALSE)
ElbowPlot(mnb, ndims = npcs)
```



```
## find clusters
mnb = FindNeighbors(mnb, reduction = "pca")
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
res = 0.9
mnb = FindClusters(mnb, resolution = res)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 3074
## Number of edges: 93204
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8060
## Number of communities: 15
## Elapsed time: 0 seconds
```

```
## Warning: UNRELIABLE VALUE: Future ('future_lapply-1') unexpectedly generated
## random numbers without specifying argument 'future.seed'. There is a risk that
## those random numbers are not statistically sound and the overall results might
## be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper,
## parallel-safe random numbers are produced via the L'Ecuyer-CMRG method. To
## disable this check, use 'future.seed=NULL', or set option 'future.rng.onMisuse'
## to "ignore".
```

```
## plot umap
dims_umap = 10
mnb = RunUMAP(mnb, reduction = "pca", dims = 1:dims_umap)
```

```
## Warning: The default method for RunUMAP has changed from calling Python UMAP via r
eticulate 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
```

```
## 17:27:51 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 17:27:51 Read 3074 rows and found 10 numeric columns
```

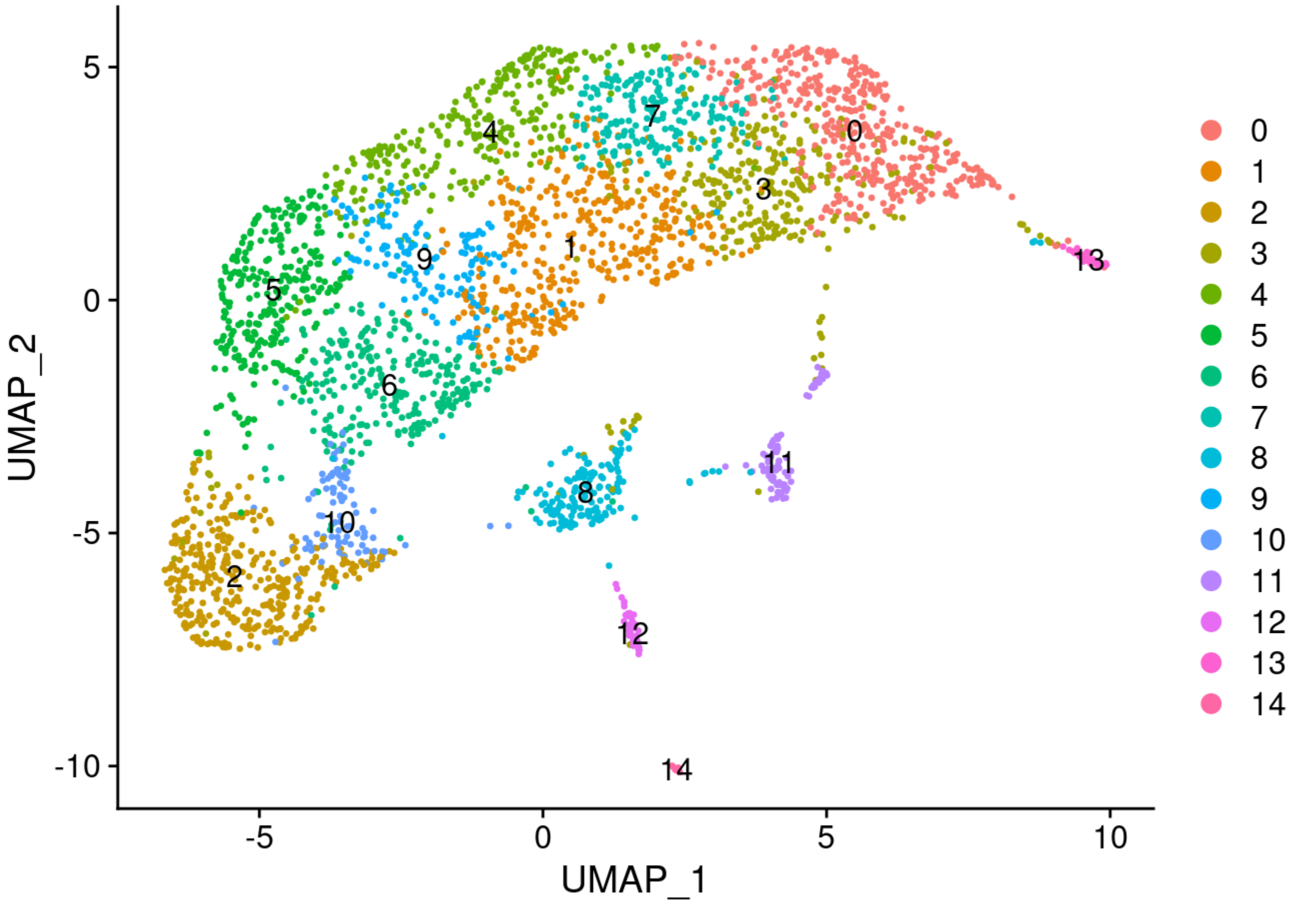
```
## 17:27:51 Using Annoy for neighbor search, n_neighbors = 30
## 17:27:51 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 17:27:51 Writing NN index file to temp file /tmp/Rtmp3YuEdx/file998d77b298ed3
## 17:27:51 Searching Annoy index using 2 threads, search_k = 3000
## 17:27:52 Annoy recall = 100%
## 17:27:52 Commencing smooth kNN distance calibration using 2 threads
## 17:27:53 Initializing from normalized Laplacian + noise
## 17:27:53 Commencing optimization for 500 epochs, with 119634 positive edges
## 17:27:57 Optimization finished
```

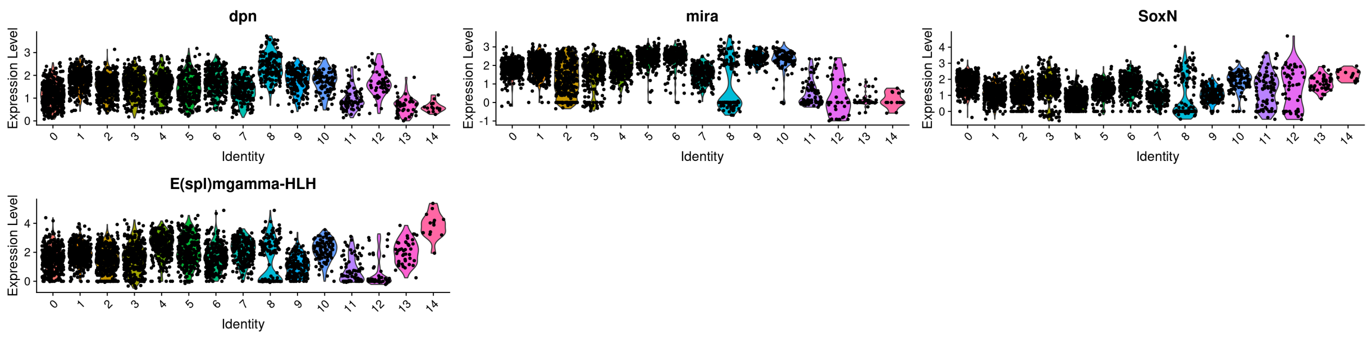
```
DimPlot(mnb, reduction = "umap", label = TRUE)
```



6 Characterize outlier clusters

Clusters 8, 11, 12, and 14 appear to be outliers. Check their levels of other neuroblast markers. This analysis corresponds to Supplementary Figure 1B.

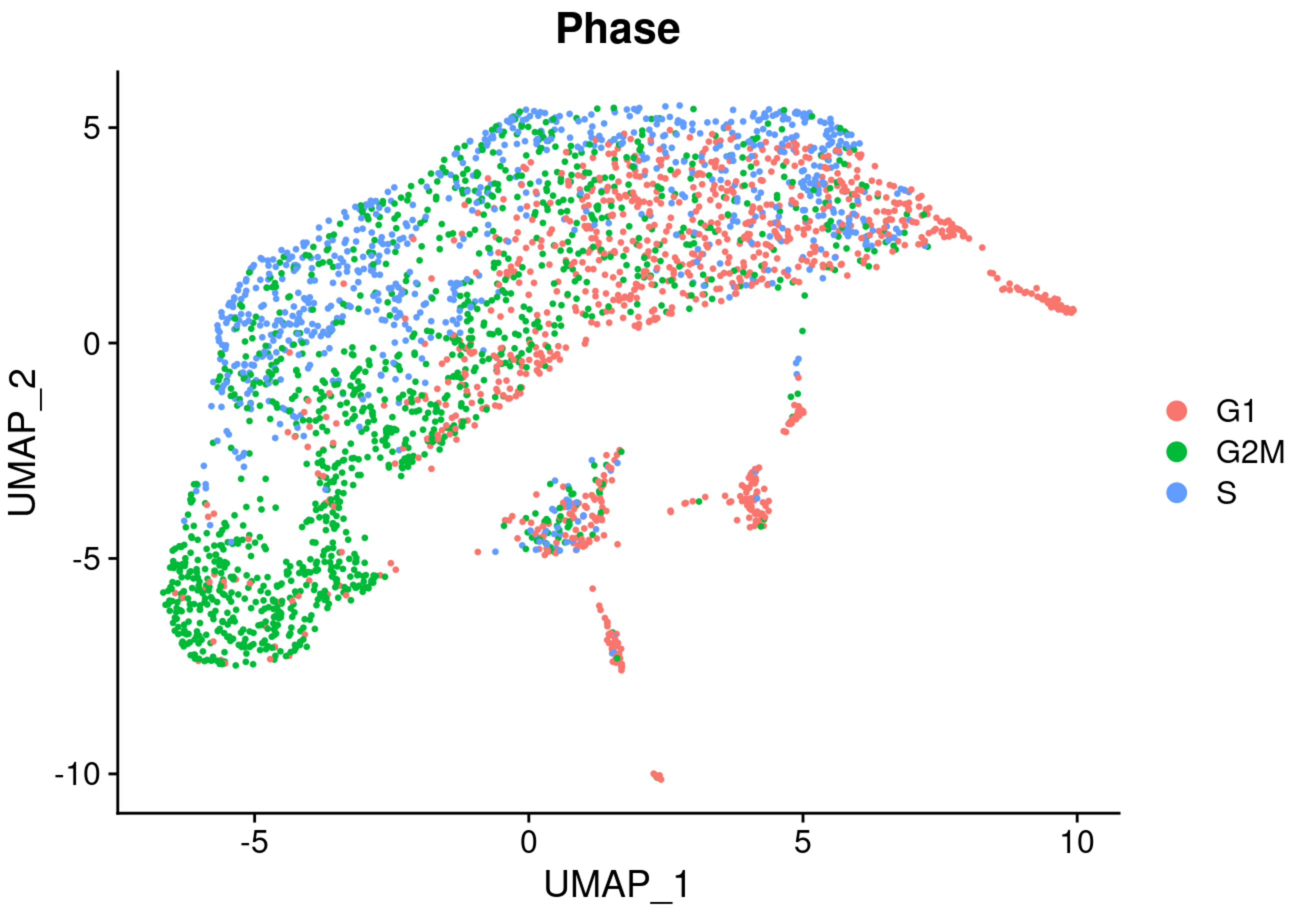
```
VlnPlot(mnb,
        features = c("dpn", "mira", "SoxN", "E(spl)mgamma-HLH"),
        group.by = "seurat_clusters")
```



7 Estimate cell cycle phase

This analysis corresponds to Figure 1D.

```
cc_genes = read.csv("List-Drosophila-cell-cycle-markers.csv",
                    header = TRUE, colClasses = "character")
s.genes = intersect(cc_genes$gene.name[cc_genes$phase == "S"],
                    rownames(mnb))
g2m.genes = intersect(cc_genes$gene.name[cc_genes$phase == "G2/M"],
                    rownames(mnb))
mnb = CellCycleScoring(mnb,
                       s.features = s.genes,
                       g2m.features = g2m.genes,
                       set.ident = TRUE)

DimPlot(mnb, reduction = "umap", group.by = "Phase")
```



8 Infer pseudotime trajectories

This analysis corresponds to Figure 1E. Remove outlier clusters before inferring pseudotime.

```
outlier_clusters = c(8, 11, 12, 14)
Idents(mnb) = "seurat_clusters"
keep = setdiff(Idents(mnb), outlier_clusters)
mnb_subset = subset(mnb, idents = keep)
```

Load subsetted Seurat data into Monocle 3. Load expression data as raw count data from RNA assay but embedding data from integrated assay.

```
mnb_int_data = as.matrix(GetAssayData(mnb_subset, assay = "RNA", slot = "counts"))
cdata = mnb_subset[[]]
gdata = data.frame(gene_short_name = rownames(mnb_int_data), row.names = rownames(mnb
_int_data))

cds = new_cell_data_set(as(mnb_int_data, "sparseMatrix"),
                        cell_metadata = cdata,
                        gene_metadata = gdata)

## process as if using monocle 3
cds = preprocess_cds(cds, num_dim = dims_umap)
cds = reduce_dimension(cds)
```

```
## No preprocess_method specified, using preprocess_method = 'PCA'
```

```
## replace things with seurat quantities
cds@int_colData@listData[["reducedDims"]][["PCA"]] = mnb_subset@reductions[["pca"]]@c
ell.embeddings
cds@int_colData@listData[["reducedDims"]][["UMAP"]] = mnb_subset@reductions[["umap"]]
@cell.embeddings

## use monocle clustering here
cds = cluster_cells(cds)
```

Use Monocle 3 to infer developmental trajectories. Place the root (beginning) of the trajectory at the vertex closest to cells with highest median `hth` expression.

```
cds = learn_graph(cds,
                  use_partition = FALSE,
                  learn_graph_control = list(minimal_branch_len = 5))
```

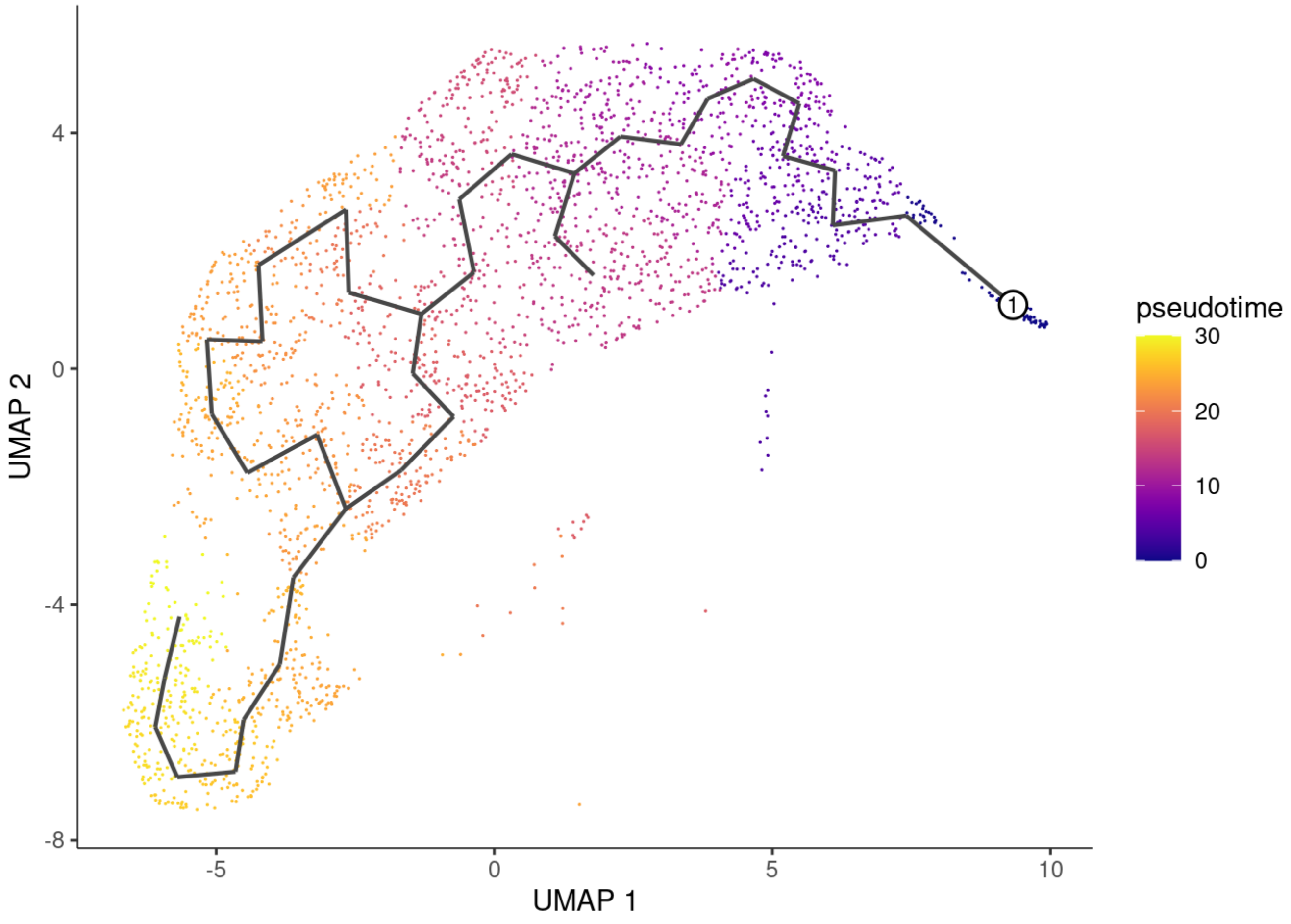
```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
## set root node
cv = cds@principal_graph_aux[["UMAP"]]$pr_graph_cell_proj_closest_vertex
part = cds@clusters@listData[["UMAP"]][["partitions"]]
roots = sapply(unique(part), function(p) {
  tmp = data.frame(hth = mnb_int_data["hth",],
                   cv = cv)[part == p,] %>%
    group_by(cv) %>%
    summarize(hth_expr = median(hth))
  tmp$cv[which.max(tmp$hth_expr)]
})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
root_pr_nodes = igraph::V(principal_graph(cds)[["UMAP"]])$name[roots]

cds = order_cells(cds, root_pr_nodes = root_pr_nodes)
plot_cells(cds,
           color_cells_by = "pseudotime",
           label_cell_groups = FALSE,
           label_leaves = FALSE,
           label_branch_points = FALSE,
           graph_label_size = 3)
```

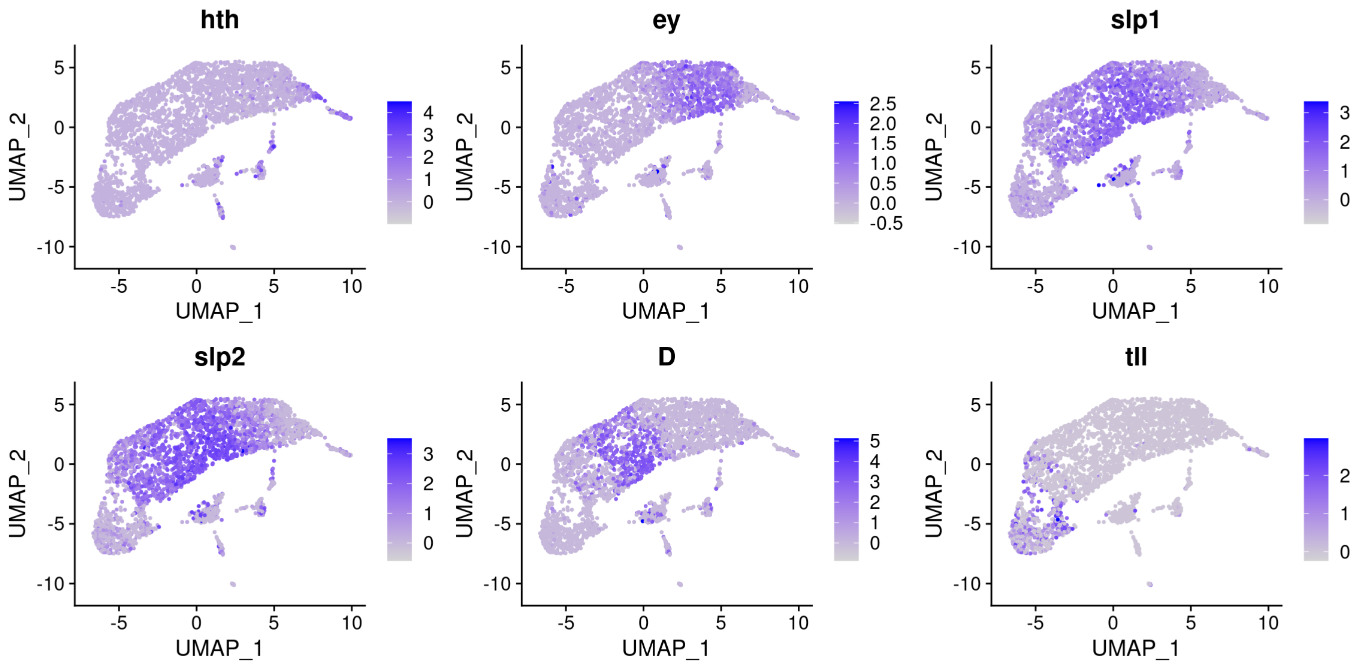
```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



9 Visualize expression patterns of known TTFs

This analysis corresponds to Figure 1F.

```
knownttfs = c("hth", "ey", "slp1", "slp2", "D", "tll")
FeaturePlot(mnb,
            features = knownttfs,
            combine = TRUE, ncol = 3)
```



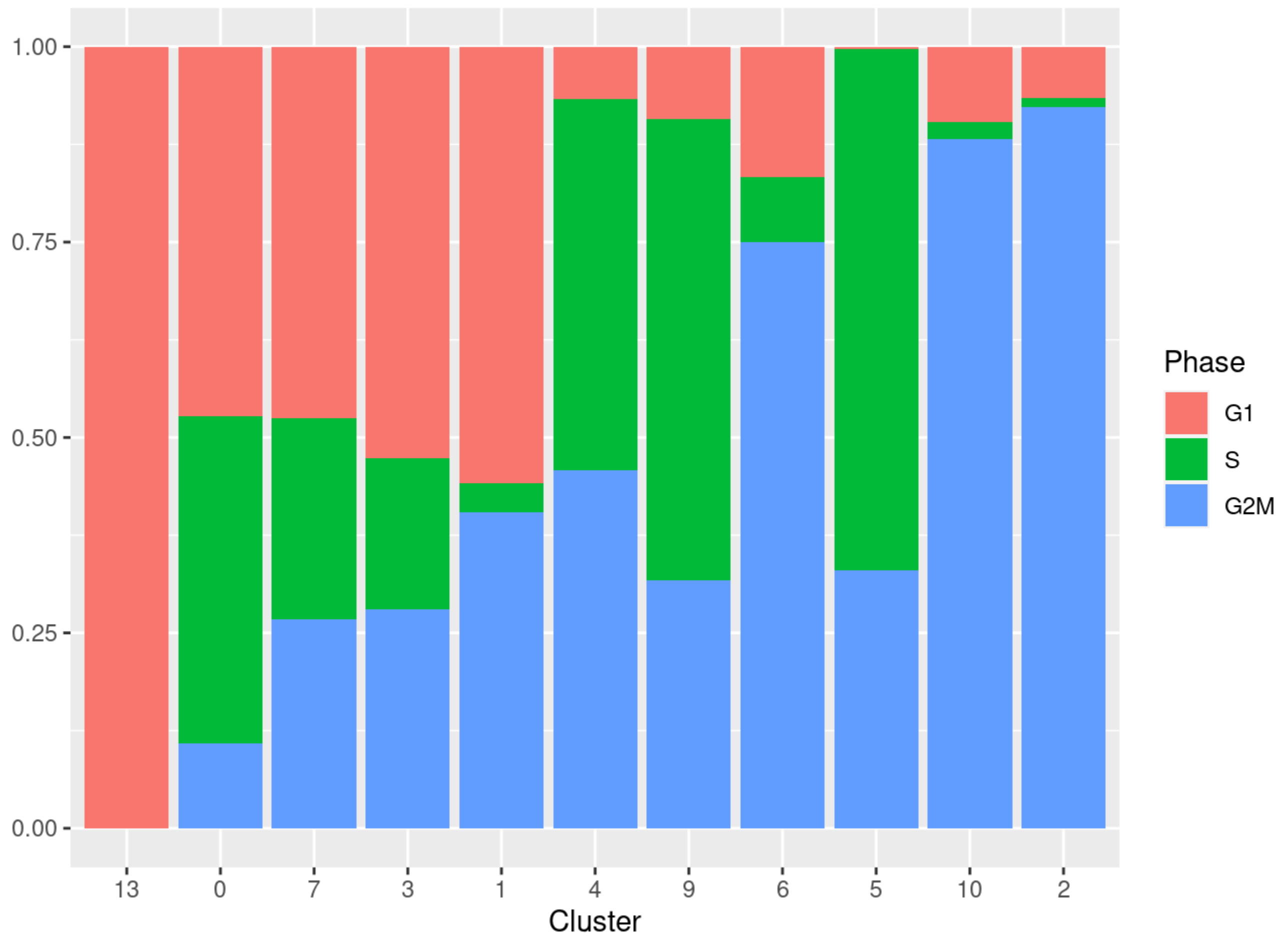
10 Visualize cell cycle proportions across clusters

Visualize proportion of phases in each cluster; order the clusters by median pseudotime of cells in the cluster. Remove outlier clusters. This analysis corresponds to Supplementary Figure 1D.

```
pst = pseudotime(cds)
dt = data.table(pst = pst, cluster = mnb_subset$seurat_clusters, phase = mnb_subset$P
hase)
clust_pst = dt[, list(med_pst = median(pst)), by = cluster]
print(clust_pst[order(clust_pst[, med_pst]), cluster])
```

```
##  [1] 13 0  7  3  1  4  9  6  5  10 2
## Levels: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
clust_v_phase = table(mnb$Phase, mnb$seurat_clusters)
## take out outliers
clust_v_phase = clust_v_phase[, !colnames(clust_v_phase) %in% outlier_clusters]
df = data.frame(t(clust_v_phase))
colnames(df) = c("Cluster", "Phase", "Count")

df$Cluster = factor(df$Cluster, levels = clust_pst[order(clust_pst[, med_pst]), clust
er])
df$Phase = factor(df$Phase, levels = c("G1", "S", "G2M"))
ggplot(data = df) +
  geom_bar(mapping = aes(x = Cluster, y = Count, fill = Phase),
           position = "fill", stat = "identity") +
  ylab("")
```



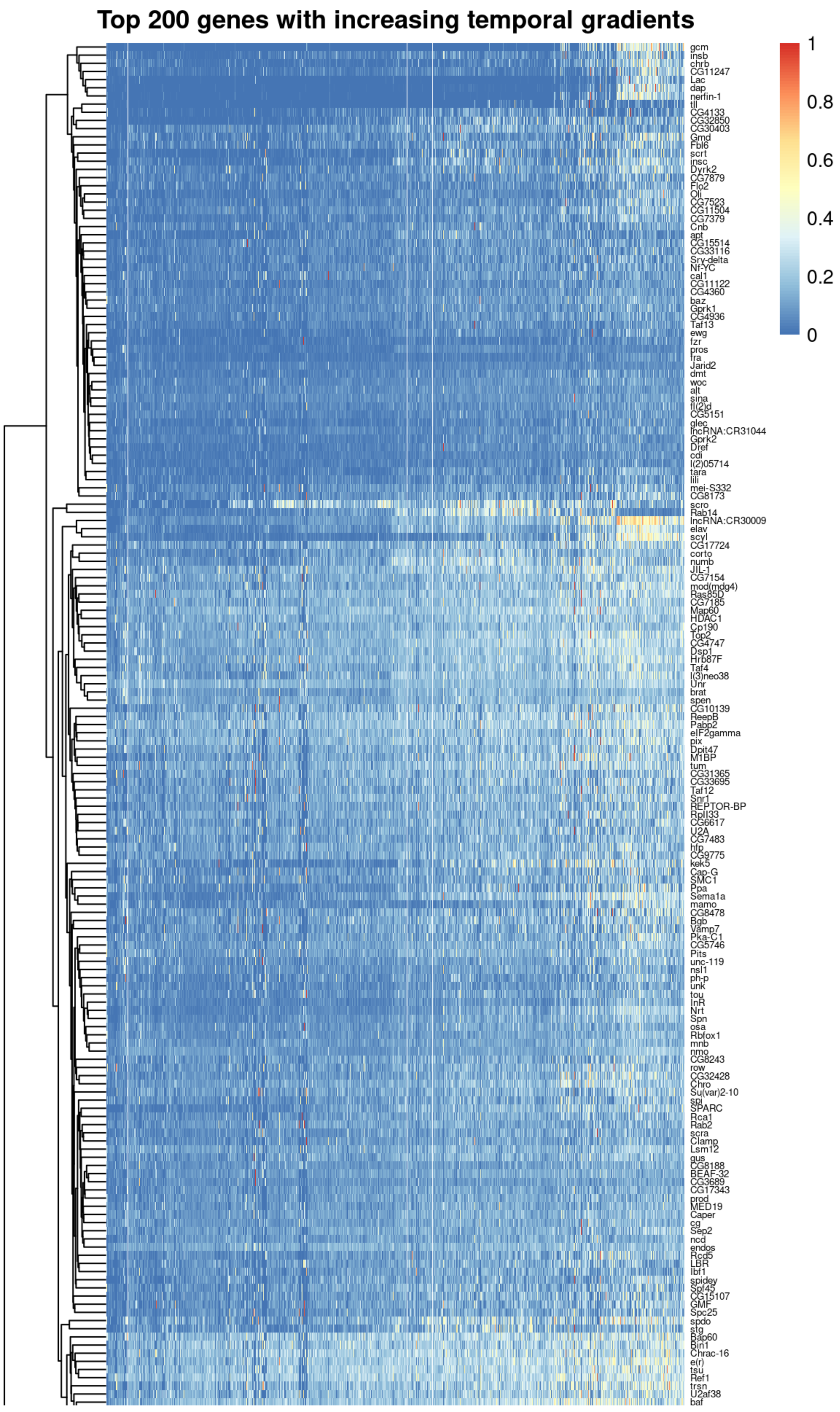
11 Visualize expression gradients with respect to pseudotime

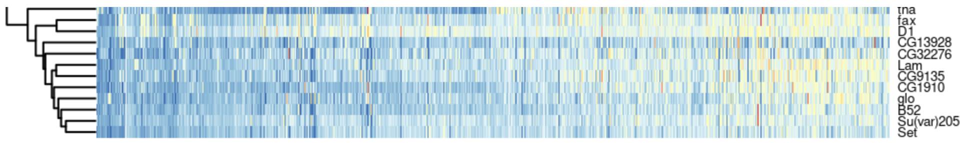
These analyses correspond to Supplementary Figures 2A through 2C. Find genes whose expressions are either increasing or decreasing with time. Only consider those with FDR less than 0.05.

```
spearman_corr_with_time = t(apply(normalized_counts(cds, norm_method = "size_only"),
1, function(gene) {
  if (sd(gene) == 0) {
    return(c(0, 1))
  } else {
    fit = cor.test(gene, pst, method = "spearman", exact = FALSE)
    return(c(fit$estimate, fit$p.value))
  }
}))

spearman_corr_with_time = data.frame(spearman_corr_with_time, fdr = p.adjust(spearman
_corr_with_time[, 2], method = "fdr"))
colnames(spearman_corr_with_time) = c("corr", "pval", "fdr")
cutoff = 0.05
spearman_corr_with_time = spearman_corr_with_time[spearman_corr_with_time$fdr <= cuto
ff,]
```

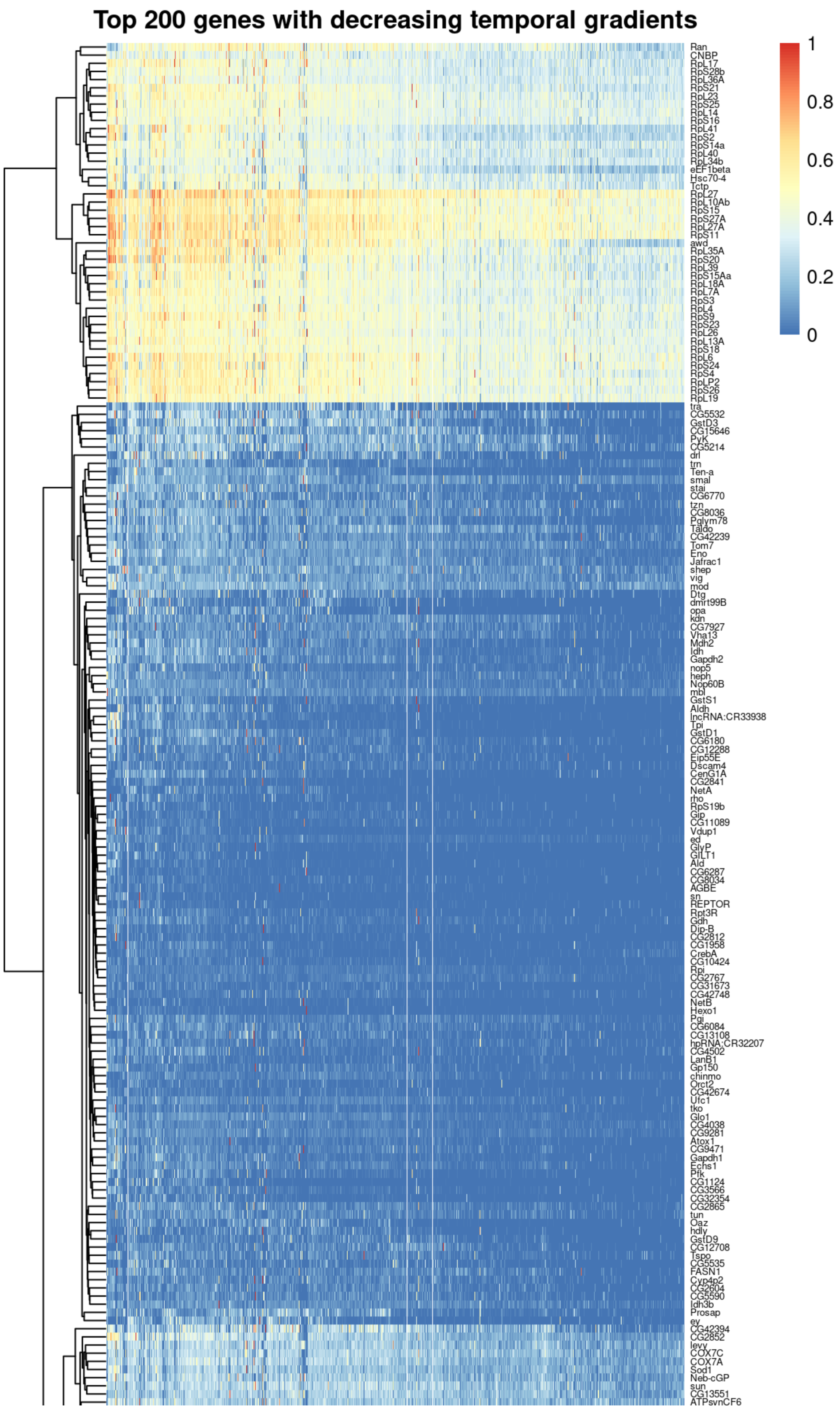
Heatmap for top 200 genes most and least linearly correlated with time.

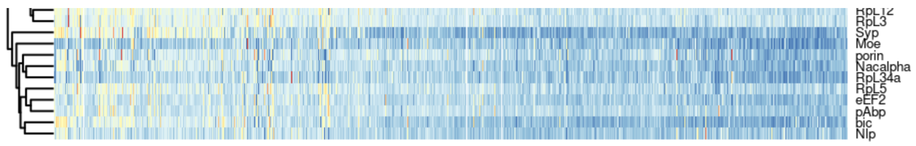
```
pos = rownames(spearman_corr_with_time)[order(-spearman_corr_with_time$corr)[1:200]]
neg = rownames(spearman_corr_with_time)[order(spearman_corr_with_time$corr)[1:200]]

## sort cells by pseudotime
pst = pseudotime(cds)
ordered_exprs = normalized_counts(cds, norm_method = "size_only")[, order(pst)]
colnames(ordered_exprs) = sort(pst)
## visualize percentages of maximum expression across pseudotime
ordered_perc_max_expr = ordered_exprs / apply(ordered_exprs, 1, max)

pheatmap(ordered_perc_max_expr[pos,],
         main = "Top 200 genes with increasing temporal gradients",
         cluster_rows = TRUE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 5)
```


Top 200 genes with increasing temporal gradients
1
0.8
0.6
0.4
0.2
0
gcm
insb
chrb
CG11247
Lac
dap
nerfin-1
tll
CG4133
CG32850
CG30403
Gmd
Fbl6
scrt
insc
Dyrk2
CG7879
Flo2
Oli
CG7523
CG11504
CG7379
Cnb
apt
CG15514
CG33116
Sry-delta
Nf-YC
cal1
CG11122
CG4360
baz
Gprk1
CG4936
Taf13
ewg
fzr
pros
fra
Jarid2
dmt
woc
alt
sina
fl(2)d
CG5151
glec
lncRNA:CR31044
Gprk2
Dref
cdi
l(2)05714
tara
lili
mei-S332
CG8173
scro
Rab14
lncRNA:CR30009
elav
scyl
CG17724
corto
numb
JIL-1
CG7154
mod(mdg4)
Ras85D
CG7185
Map60
HDAC1
Cp190
Top2
CG4747
Dsp1
Hrb87F
Taf4
l(3)neo38
Unr
brat
spen
CG10139
ReepB
Pabp2
eIF2gamma
pix
Dpit47
M1BP
tum
CG31365
CG33695
Taf12
Snr1
REPTOR-BP
RpII33
CG6617
U2A
CG7483
hfp
CG9775
kek5
Cap-G
SMC1
Ppa
Sema1a
mamo
CG8478
Bgb
Vamp7
Pka-C1
CG5746
Pits
unc-119
nsl1
ph-p
unk
tou
InR
Nrt
Spn
osa
Rbfox1
mnb
nmo
CG8243
row
CG32428
Chro
Su(var)2-10
spi
SPARC
Rca1
Rab2
scra
Clamp
Lsm12
gus
CG8188
BEAF-32
CG3689
CG17343
prod
MED19
Caper
cg
Sep2
ncd
endos
Rcd5
LBR
Ibf1
spidey
Spf45
CG15107
GMF
Spc25
spdo
stg
Bap60
Bin1
Chrac-16
e(r)
tsu
Ref1
trsn
U2af38
baf



```
pheatmap(ordered_perc_max_expr[neg,],
         main = "Top 200 genes with decreasing temporal gradients",
         cluster_rows = TRUE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 5)
```


Top 200 genes with decreasing temporal gradients
1
0.8
0.6
0.4
0.2
0
Ran
CNBP
RpL17
RpS28b
RpL36A
RpS21
RpL23
RpS25
RpL14
RpS16
RpL41
RpS2
RpS14a
RpL40
RpL34b
eEF1beta
Hsc70-4
Tctp
RpL27
RpL10Ab
RpS15
RpS27A
RpL27A
RpS11
awd
RpL35A
RpS20
RpL39
RpS15Aa
RpL18A
RpL7A
RpS3
RpL4
RpS9
RpS23
RpL26
RpL13A
RpS18
RpL6
RpS24
RpS4
RpLP2
RpS26
RpL19
tra
CG5532
GstD3
CG15646
PyK
CG5214
drl
trn
Ten-a
smal
stai
CG6770
tzn
CG8036
Pglym78
Taldo
CG42239
Tom7
Eno
Jafrac1
shep
vig
mod
Dtg
dmrt99B
opa
kdn
CG7927
Vha13
Mdh2
Idh
Gapdh2
nop5
heph
Nop60B
mbl
GstS1
Aldh
lncRNA:CR33938
Tpi
GstD1
CG6180
CG12288
Eip55E
Dscam4
CenG1A
CG2841
NetA
rho
RpS19b
Gip
CG11089
Vdup1
ed
GlyP
GILT1
Ald
CG6287
CG8034
AGBE
sn
REPTOR
Rpt3R
Gdh
Dip-B
CG2812
CG1958
CrebA
CG10424
Rpi
CG2767
CG31673
CG42748
NetB
Hexo1
Pgi
CG6084
CG13108
hpRNA:CR32207
CG4502
LanB1
Gp150
chinmo
Orct2
CG42674
Ufc1
tko
Glo1
CG4038
CG9281
Atox1
CG9471
Gapdh1
Echs1
Pfk
CG1124
CG3566
CG32354
CG2865
tun
Oaz
hdly
GstD9
CG12708
Tspo
CG5535
FASN1
Cyp4p2
CG2604
CG5590
Idh3b
Prosap
ey
CG42394
CG2852
levy
COX7C
COX7A
Sod1
Neb-cGP
sun
CG13551
ATPsynCF6



Heatmaps for top 10 differentially expressed TFs in cluster. Differentially expressed genes are first called using an FDR threshold of 0.05, then the top 10 genes with largest magnitude of average log-fold change are extracted. List of TFs comes from this article: https://www.nature.com/articles/nmeth.1763 (https://www.nature.com/articles/nmeth.1763).

```
## identify differentially expressed genes
DefaultAssay(mnb) = "RNA"
plan("multiprocess", workers = 2)
mnb_de = FindAllMarkers(mnb)
```

```
## Calculating cluster 0
```

```
## Calculating cluster 1
```

```
## Calculating cluster 2
```

```
## Calculating cluster 3
```

```
## Calculating cluster 4
```

```
## Calculating cluster 5
```

```
## Calculating cluster 6
```

```
## Calculating cluster 7
```

```
## Calculating cluster 8
```

```
## Calculating cluster 9
```

```
## Calculating cluster 10
## Calculating cluster 11
```

```
## Calculating cluster 12
```

```
## Calculating cluster 13
```

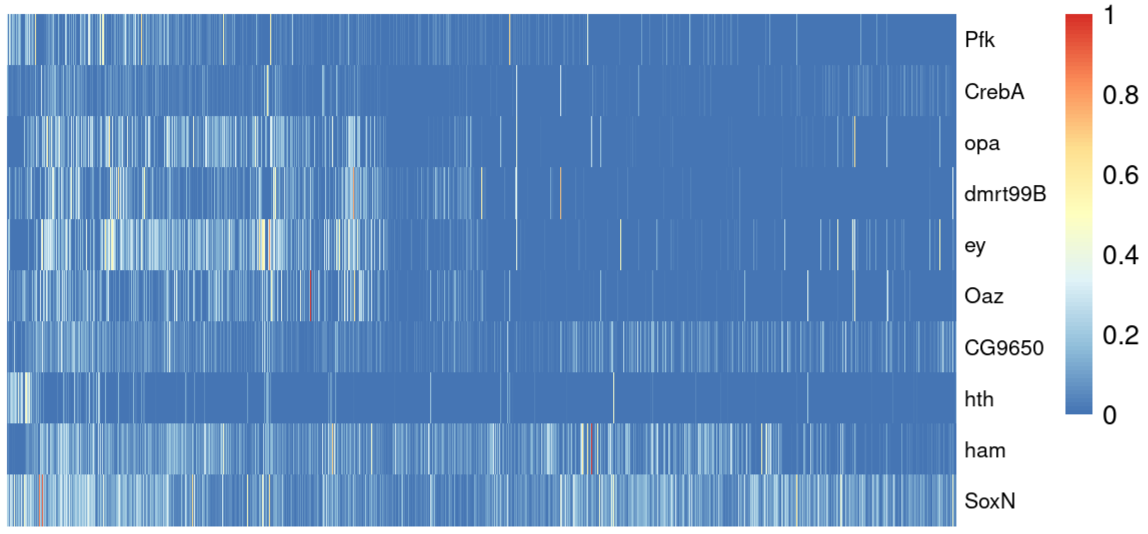
```
## Calculating cluster 14
```

```
## load TFs
dmtfs = read.csv("dmtfs.csv", header = TRUE)
tfs = rownames(mnb_subset)[which(tolower(rownames(mnb_subset)) %in% tolower(dmtfs$Sym
bol))]
print(length(tfs))
```

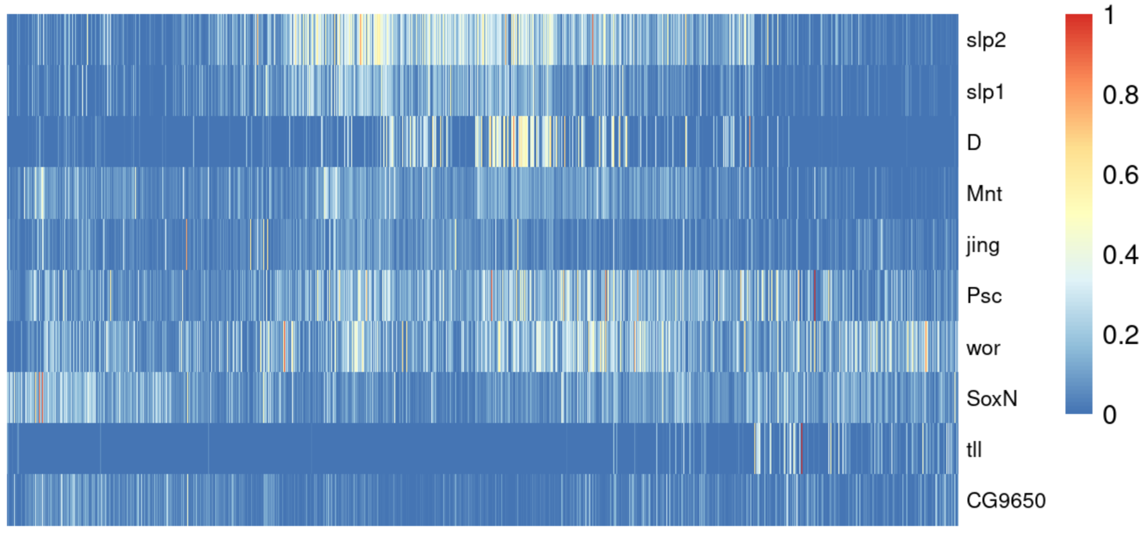
```
## [1] 635
```

```
## for each cluster
for (cl in setdiff(unique(mnb_de$cluster), outlier_clusters)) {

  ## extract DE results for TFs
  ## want these to be more highly expressed in the cluster than in other clusters
  mnb_de_tf = mnb_de %>% subset(cluster == cl & gene %in% tfs & p_val_adj <= 0.05) %
>% top_n(n = 10, wt = avg_logFC)

  ## visualize
  pheatmap(ordered_perc_max_expr[intersect(mnb_de_tf$gene, rownames(ordered_perc_max_
expr)),],
           main = paste("Cluster", cl),
           cluster_rows = FALSE, cluster_cols = FALSE,
           show_colnames = FALSE,
           show_rownames = TRUE, fontsize_row = 8)

}
```

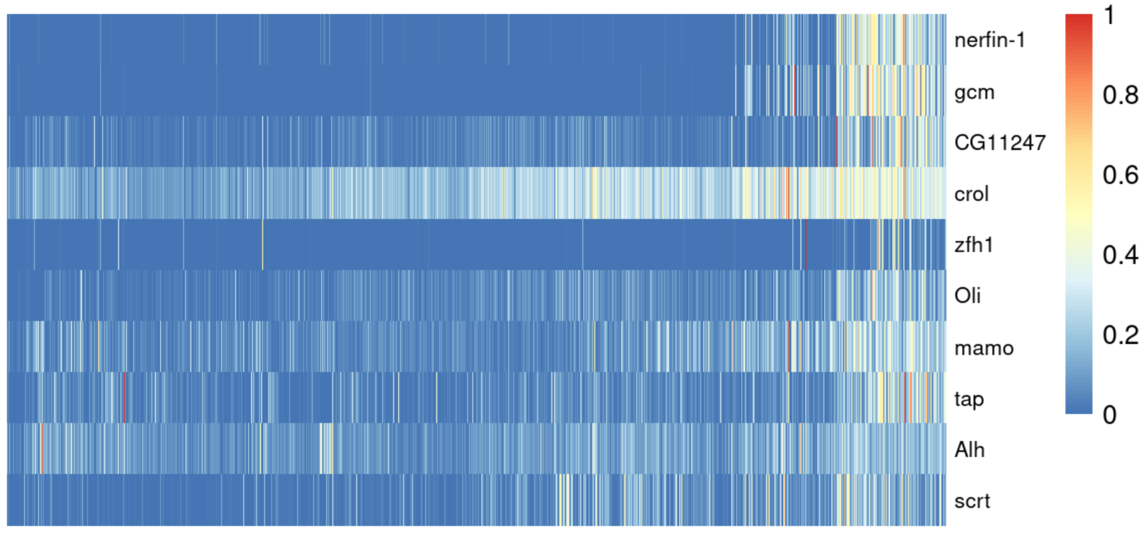
Cluster 0

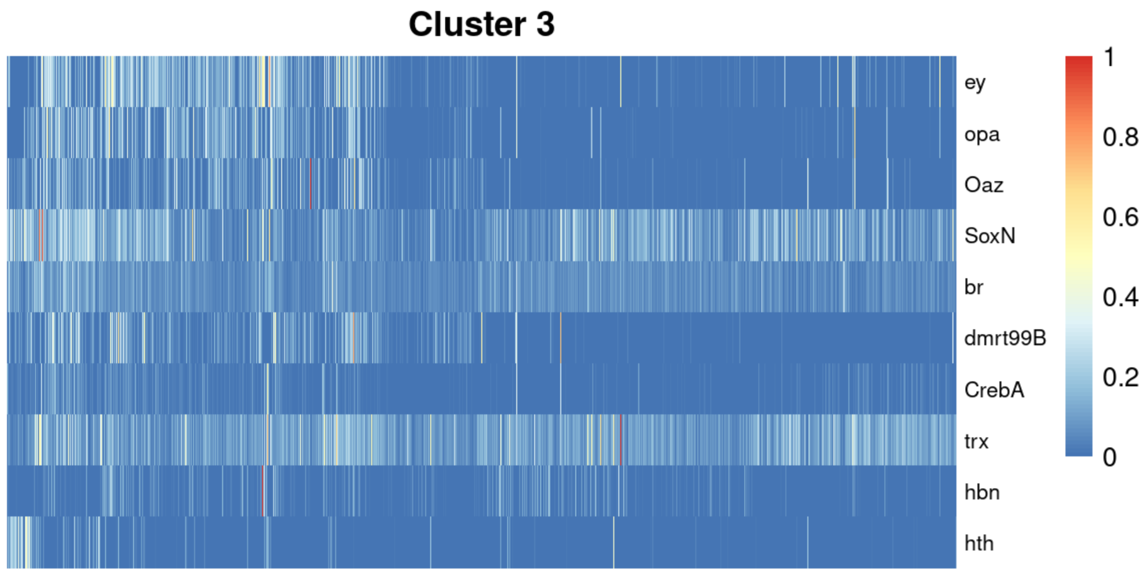


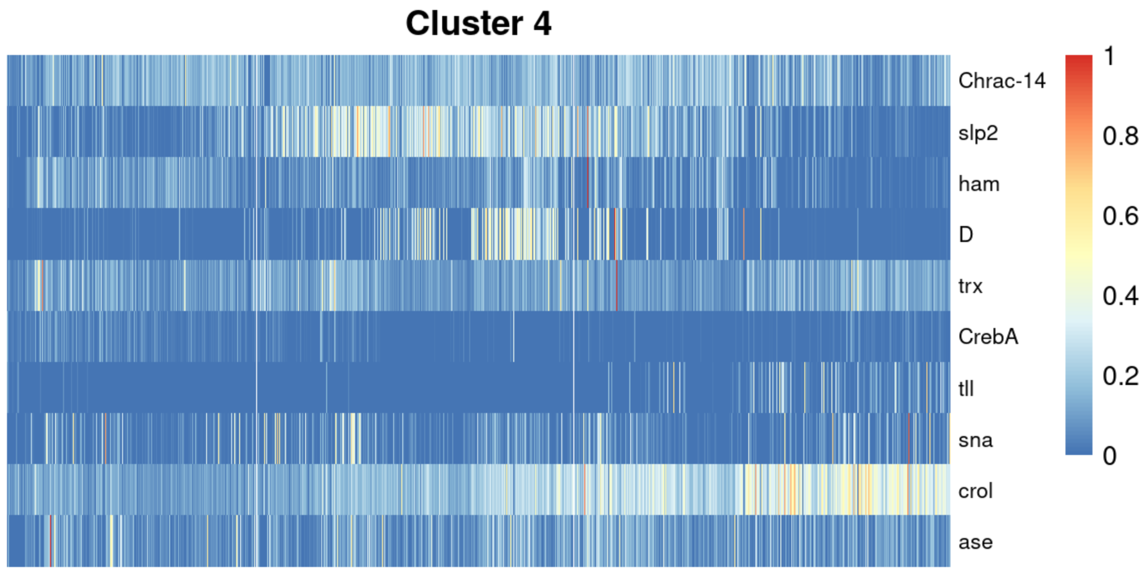
Cluster 1

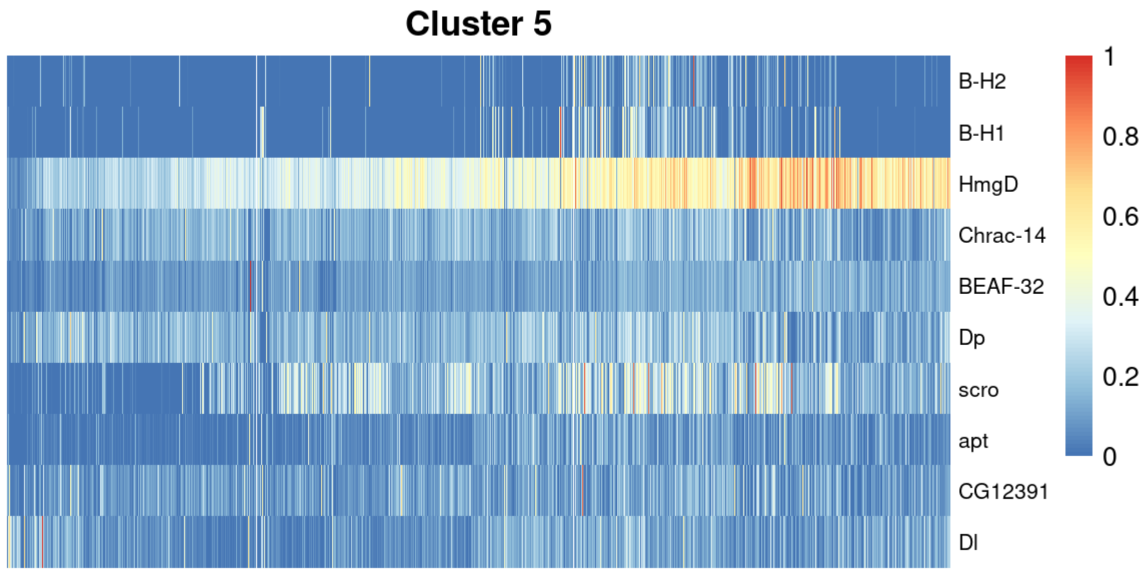


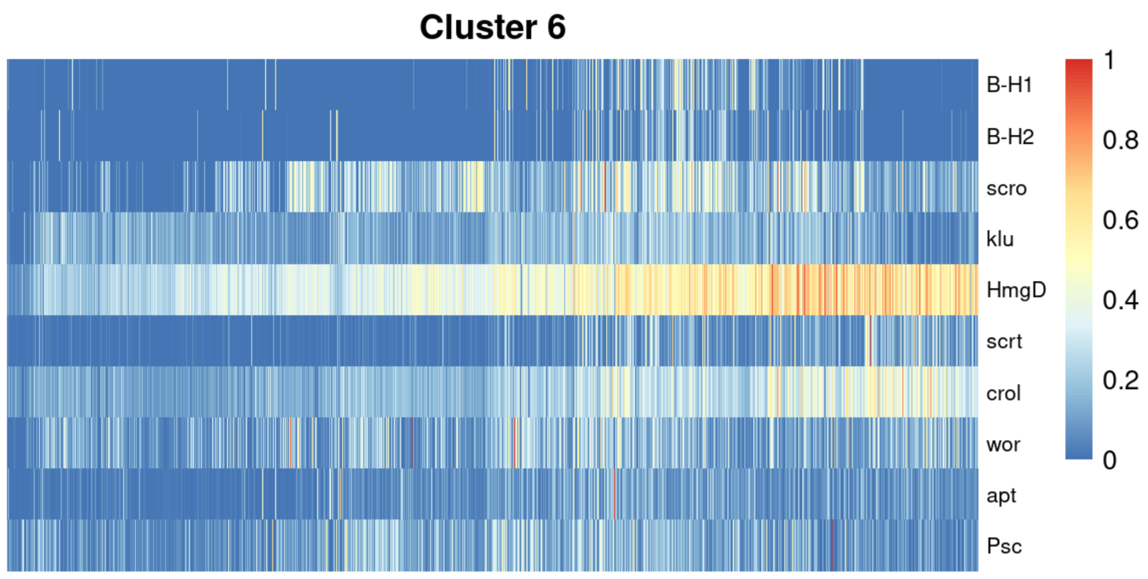
Cluster 2

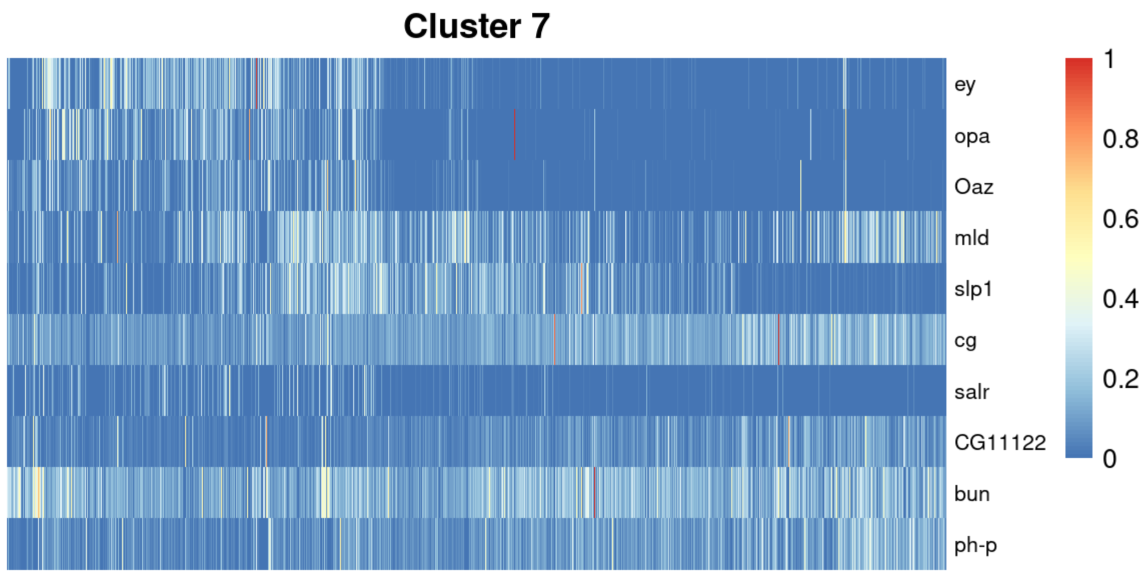


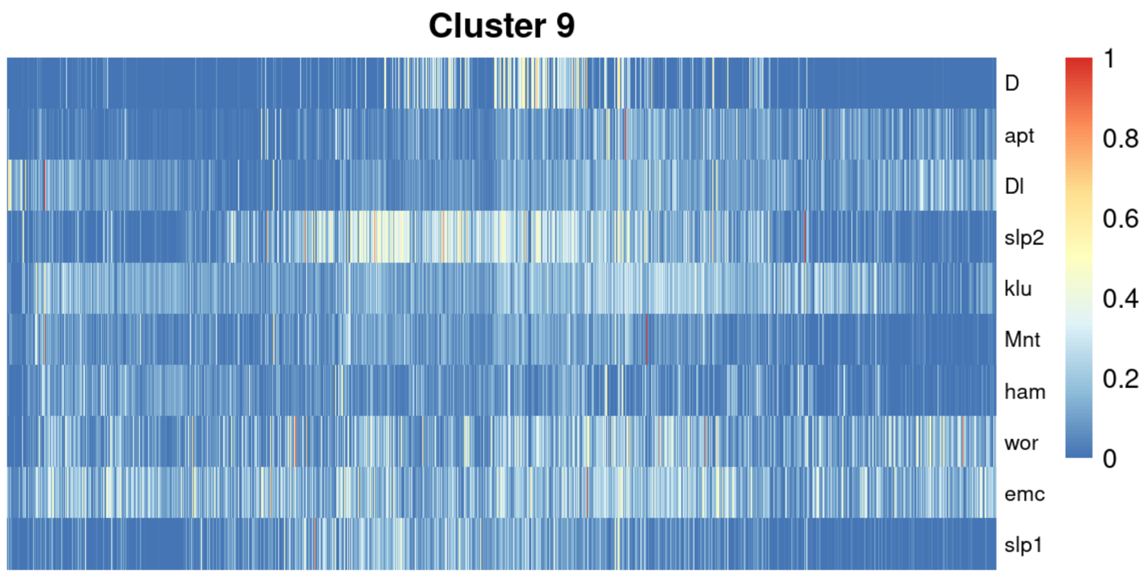

Cluster 3
ey
opa
Oaz
SoxN
br
dmrt99B
CrebA
trx
hbn
hth
1
0.8
0.6
0.4
0.2
0

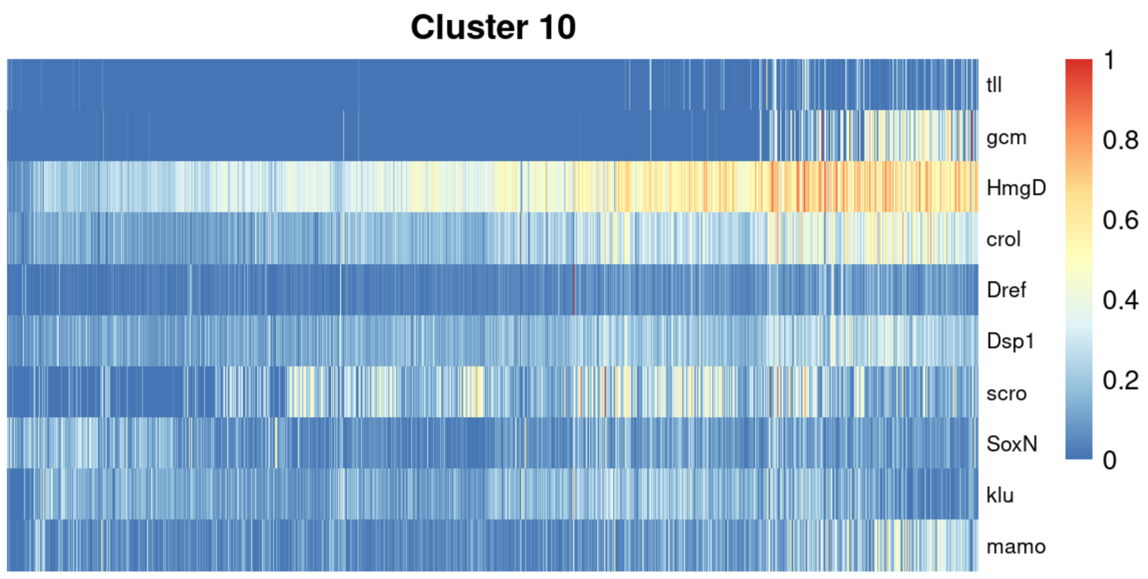

Cluster 4
Chrac-14
slp2
ham
D
trx
CrebA
tll
sna
crol
ase
1
0.8
0.6
0.4
0.2
0

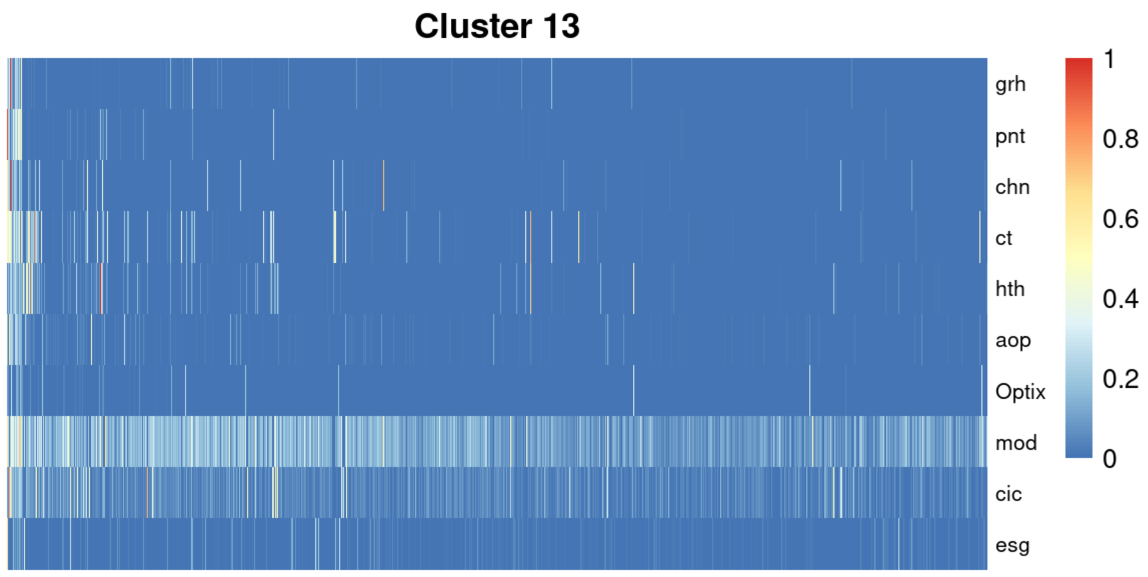

Cluster 5
B-H2
B-H1
HmgD
Chrac-14
BEAF-32
Dp
scro
apt
CG12391
Dl
1
0.8
0.6
0.4
0.2
0


Cluster 6
B-H1
B-H2
scro
klu
HmgD
scrt
crol
wor
apt
Psc
1
0.8
0.6
0.4
0.2
0


Cluster 7
ey
opa
Oaz
mld
slp1
cg
salr
CG11122
bun
ph-p
1
0.8
0.6
0.4
0.2
0


Cluster 9
D
apt
Dl
slp2
klu
Mnt
ham
wor
emc
slp1
1
0.8
0.6
0.4
0.2
0

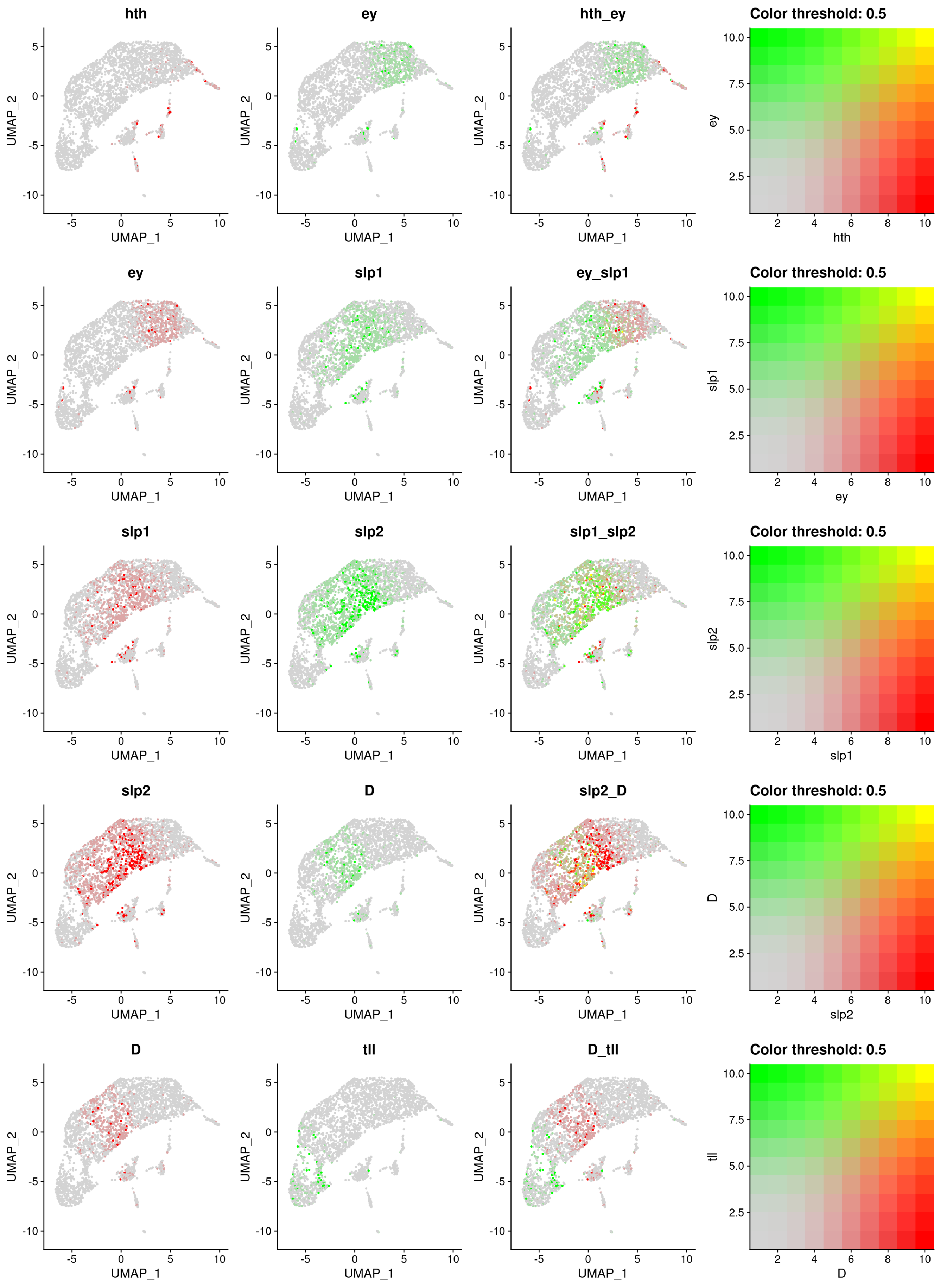




12 Visualize overlapping expression patterns of known TTFs

This analysis corresponds to Supplementary Figures 4A through 4E.

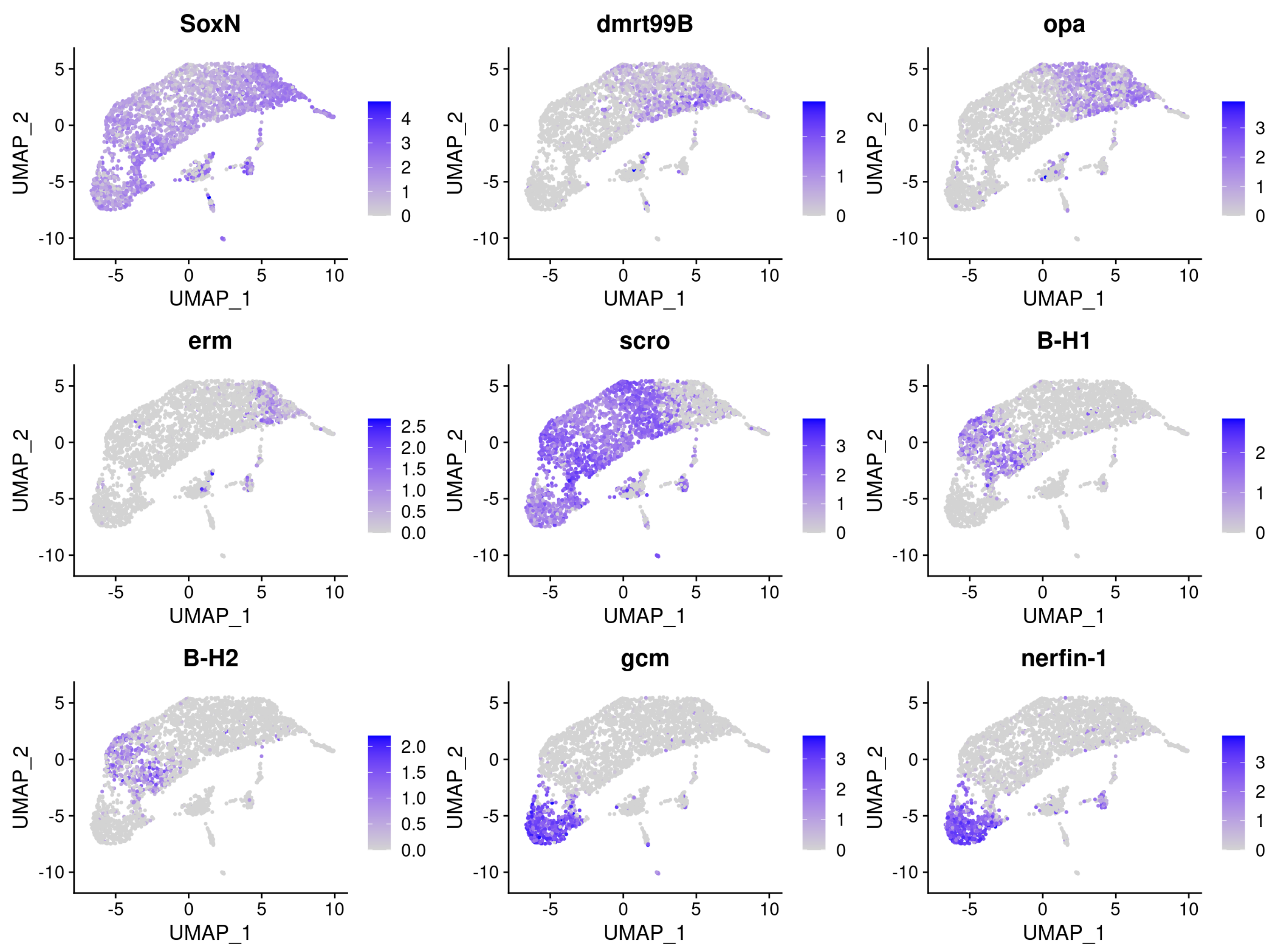
```
knowntttfs = c("hth", "ey", "slp1", "slp2", "D", "tll")

for (i in 1:(length(knowntttfs) - 1)) {

  p = FeaturePlot(mnb,
                  features = knowntttfs[i:(i + 1)],
                  blend = TRUE)
  print(p)

}
```


hth
ey
hth_ey
Color threshold: 0.5
ey
slp1
ey_slp1
Color threshold: 0.5
slp1
slp2
slp1_slp2
Color threshold: 0.5
slp2
D
slp2_D
Color threshold: 0.5
D
tll
D_tll
Color threshold: 0.5
UMAP_1
UMAP_2

13 Visualize expression patterns of novel TTFs

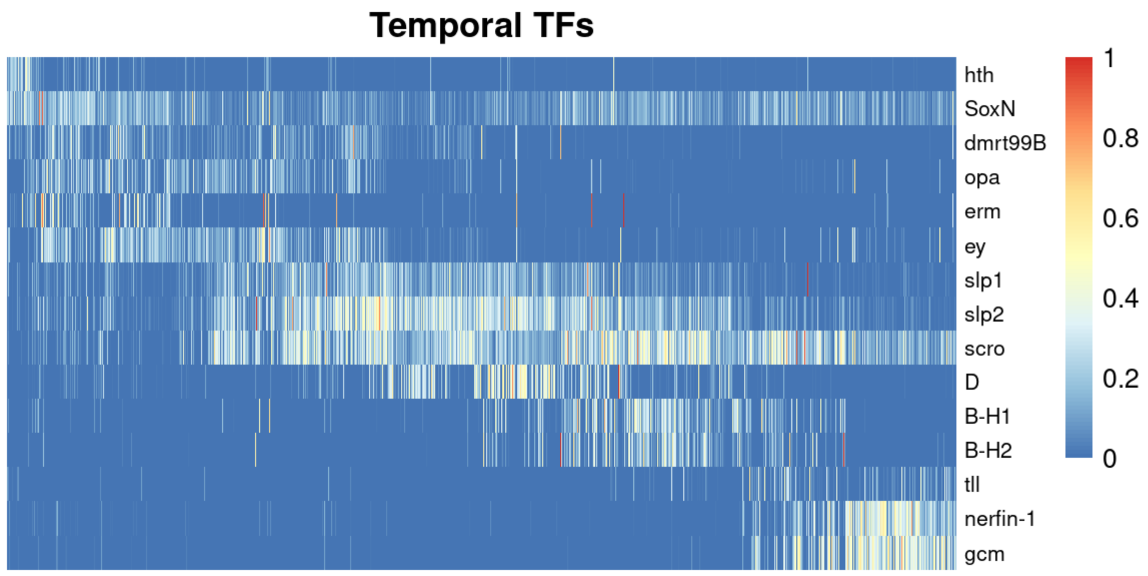
This analysis corresponds to Figures 1G and 1H. Visualize expression on UMAP.

```
## temporal tfs
ttfs = c("SoxN", "dmrt99B", "opa", "erm", "scro", "B-H1", "B-H2", "gcm", "nerfin-1")
FeaturePlot(mnb,
            features = ttfs,
            combine = TRUE, ncol = 3)
```

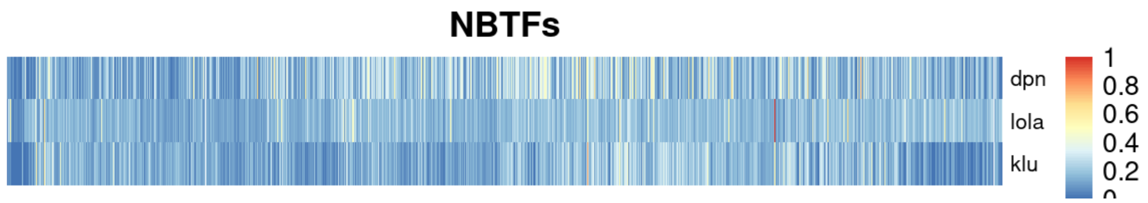


Visualize expression over time.

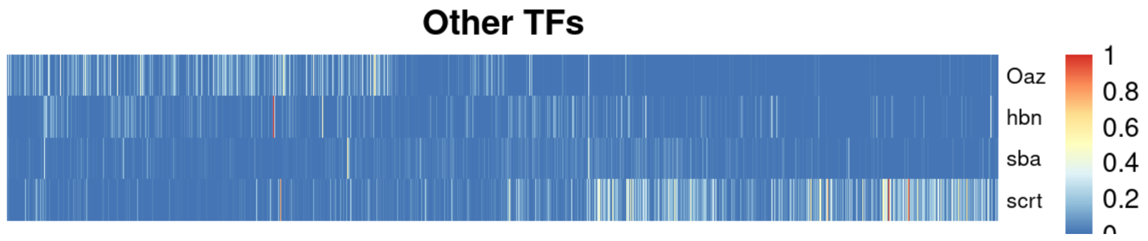
```
## temporal tfs
ttfs = c("hth", "SoxN", "dmrt99B", "opa", "erm", "ey", "slp1", "slp2", "scro", "D", "
B-H1", "B-H2", "tll", "nerfin-1", "gcm")

pheatmap(ordered_perc_max_expr[intersect(ttfs, rownames(ordered_perc_max_expr)),],
         main = "Temporal TFs",
         cluster_rows = FALSE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 8)
```



```
NBTFs = c("dpn", "lola", "klu")
pheatmap(ordered_perc_max_expr[intersect(NBTFs, rownames(ordered_perc_max_expr)),],
         main = "NBTFs",
         cluster_rows = FALSE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 8)
```



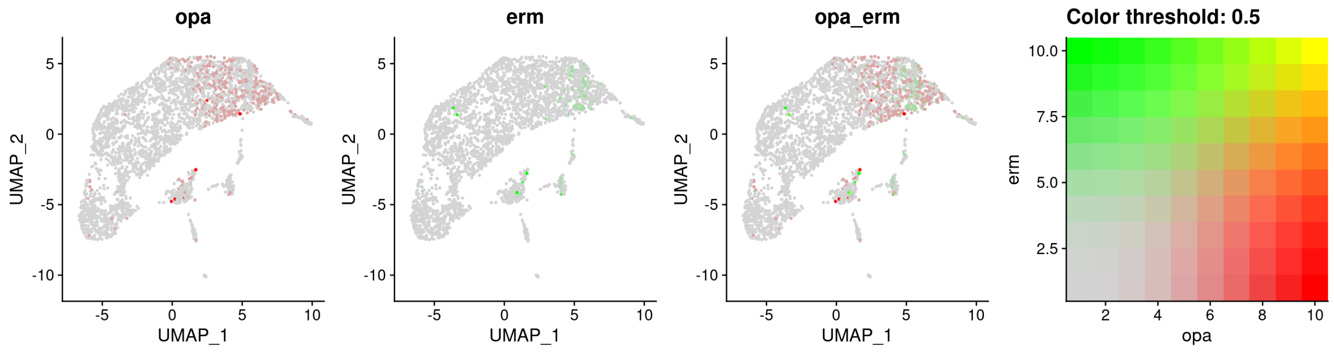
```
othertfs = c("Oaz", "hbn", "sba", "scrt")
pheatmap(ordered_perc_max_expr[intersect(othertfs, rownames(ordered_perc_max_exp
r)),],
         main = "Other TFs",
         cluster_rows = FALSE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 8)
```



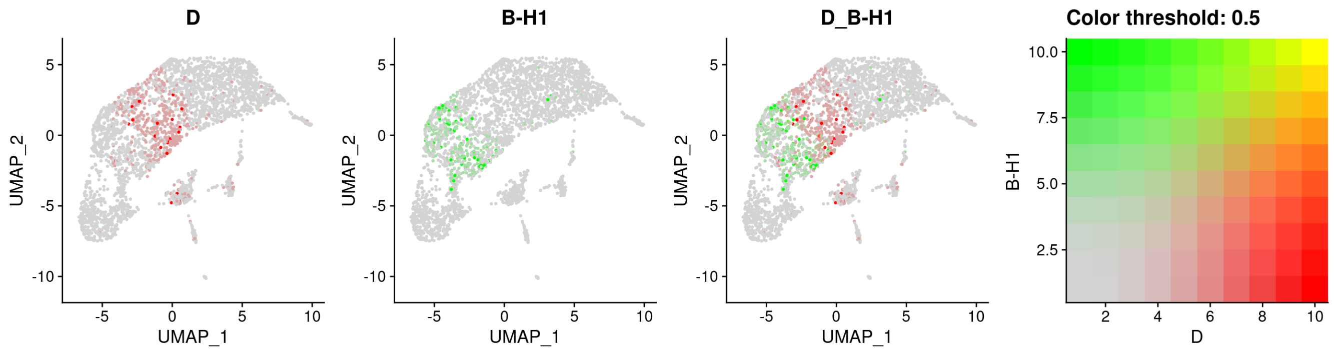
14 Visualize overlapping expression patterns of novel and known TTFs

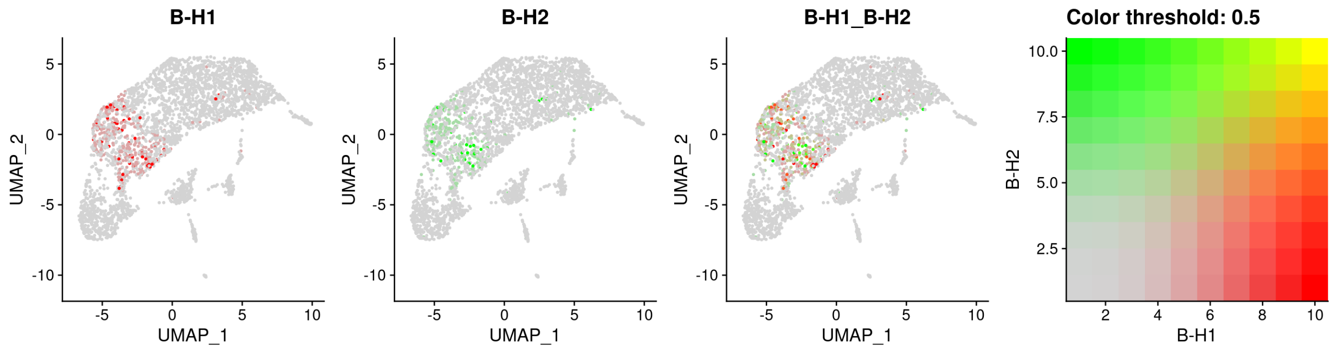
This analysis corresponds to Supplementary Figures 5A through 5D.

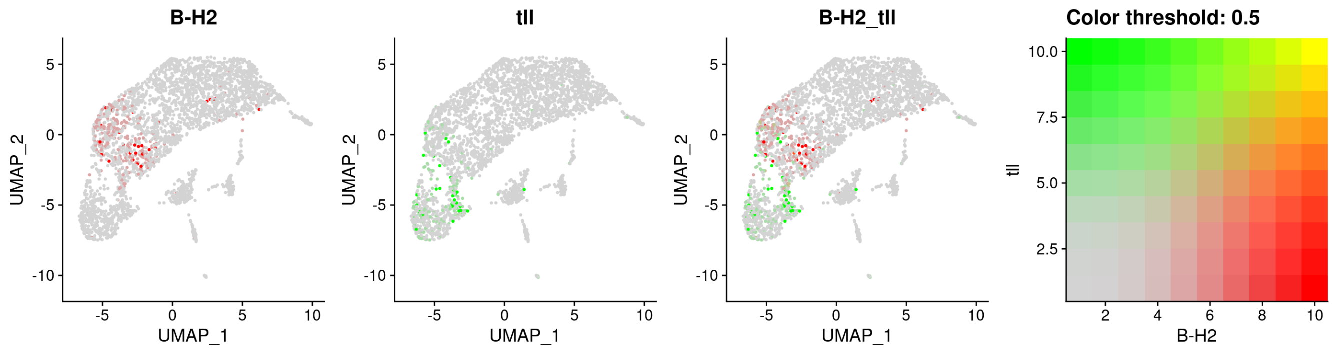
```
FeaturePlot(mnb, features = c("opa", "erm"), blend = TRUE)
```



```
ttfs = c("D", "B-H1", "B-H2", "tll")

for (i in 1:(length(ttfs) - 1)) {

  p = FeaturePlot(mnb,
                  features = ttfs[i:(i + 1)],
                  blend = TRUE)
  print(p)

}
```



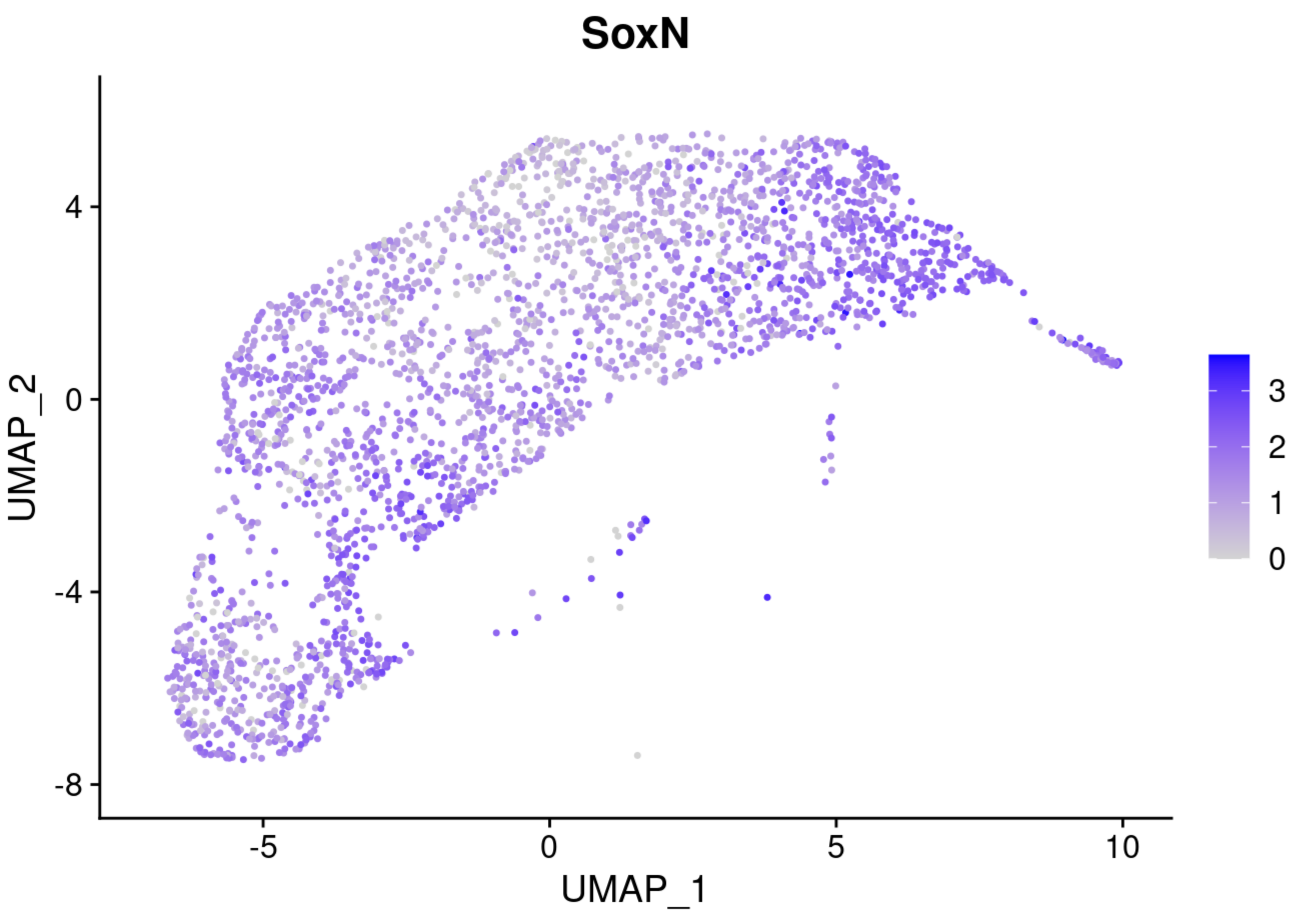




15 SoxN temporal expression pattern

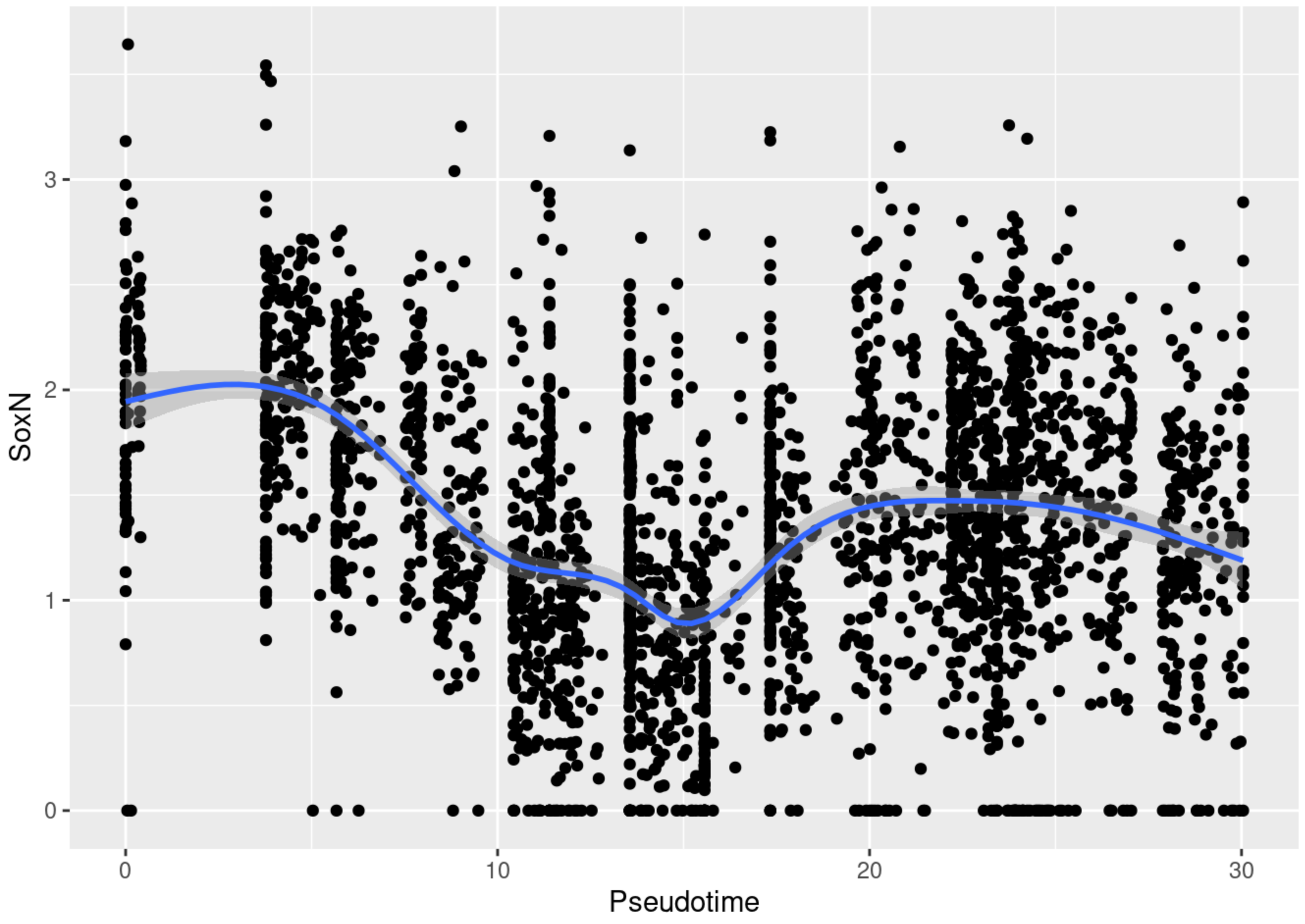
This analysis corresponds to Supplementary Figure 7A.

```
DefaultAssay(mnb_subset) = "RNA"
FeaturePlot(mnb_subset, "SoxN")
```



```
df = data.frame(Pseudotime = pst,
                SoxN = GetAssayData(mnb_subset, slot = "data")["SoxN",])
ggplot(data = df) +
  geom_point(aes(x = Pseudotime, y = SoxN)) +
  geom_smooth(aes(x = Pseudotime, y = SoxN))
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



```
cor.test(df$Pseudotime, df$SoxN, method = "spearman")
```

```
## Warning in cor.test.default(df$Pseudotime, df$SoxN, method = "spearman"): Cannot
## compute exact p-value with ties
```

```
##
##  Spearman's rank correlation rho
##
## data:  df$Pseudotime and df$SoxN
## S = 3.93e+09, p-value = 2.474e-09
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##        rho
## -0.1130504
```

16 RNA-binding protein temporal expression patterns

These analyses correspond to Supplementary Figures 6D and 6E. First read in RNA binding proteins. Some genes are not in the database.

```
rbp = read_xlsx("RNA-binding-chromatin-gene lists.xlsx",
                col_names = FALSE)[[1]]
```

```
## New names:
## * `` -> ...1
```

```
## rename some gene names so that they are in the dataset
rbp[rbp == "Liprin-γ"] = "Liprin-gamma"
rbp[rbp == "Liprin-α"] = "Liprin-alpha"
rbp[rbp == "eIF2β"] = "eIF2beta"

## which genes are not in the data
rbp[which(!(rbp %in% rownames(mnb_subset)))]
```

```
## [1] "Secp43" "Cpsf6"  "Marf1"
```

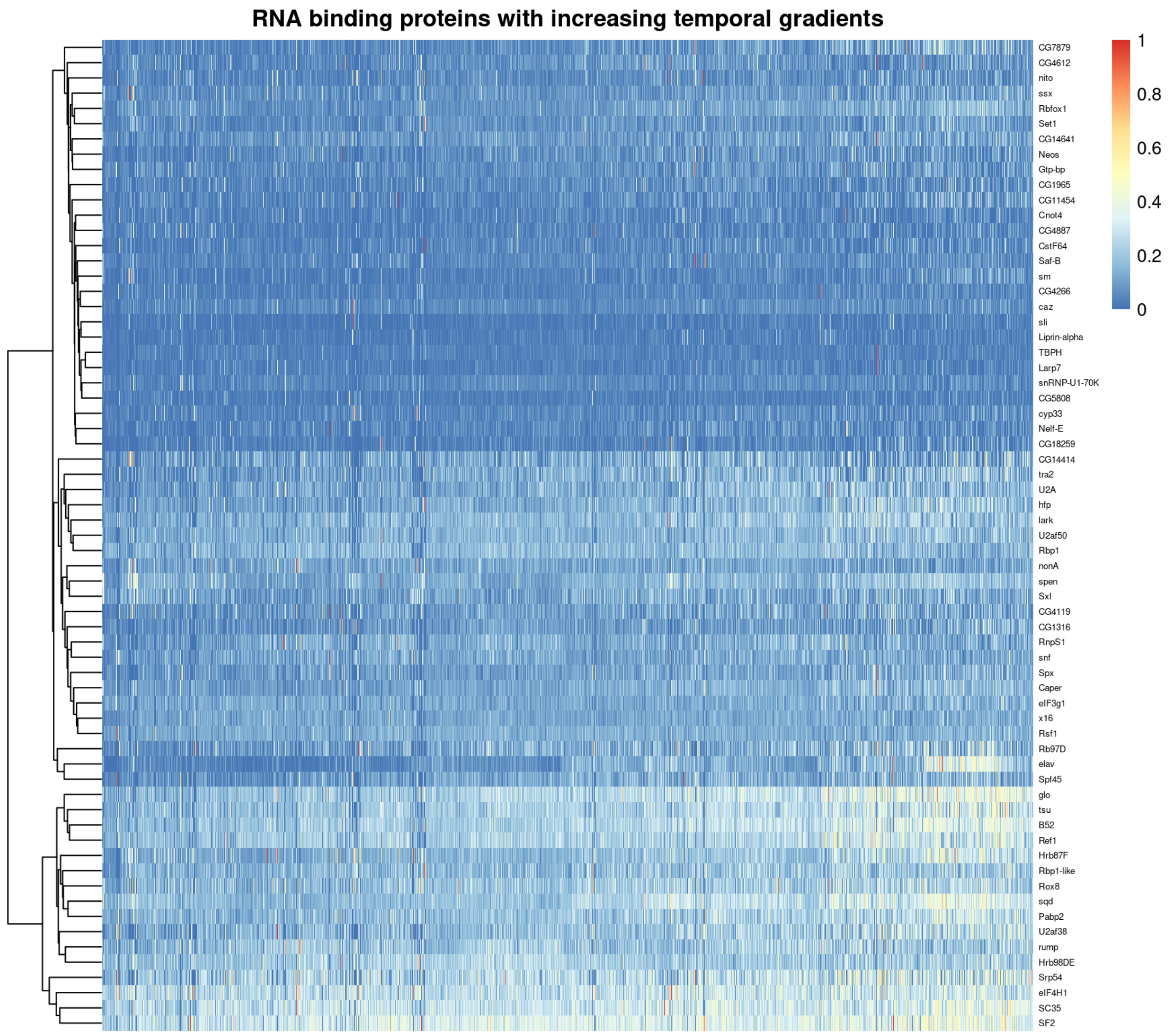
```
## only take genes in the dataset
rbp = intersect(rbp, rownames(mnb_subset))

## only take genes that are expressed in at least one cell
keep = rowSums(GetAssayData(mnb_subset[rbp,], assay = "RNA", slot = "counts")) > 0
rbp = rbp[keep]
```

Find RNA binding proteins that are highly correlated with pseudotime.

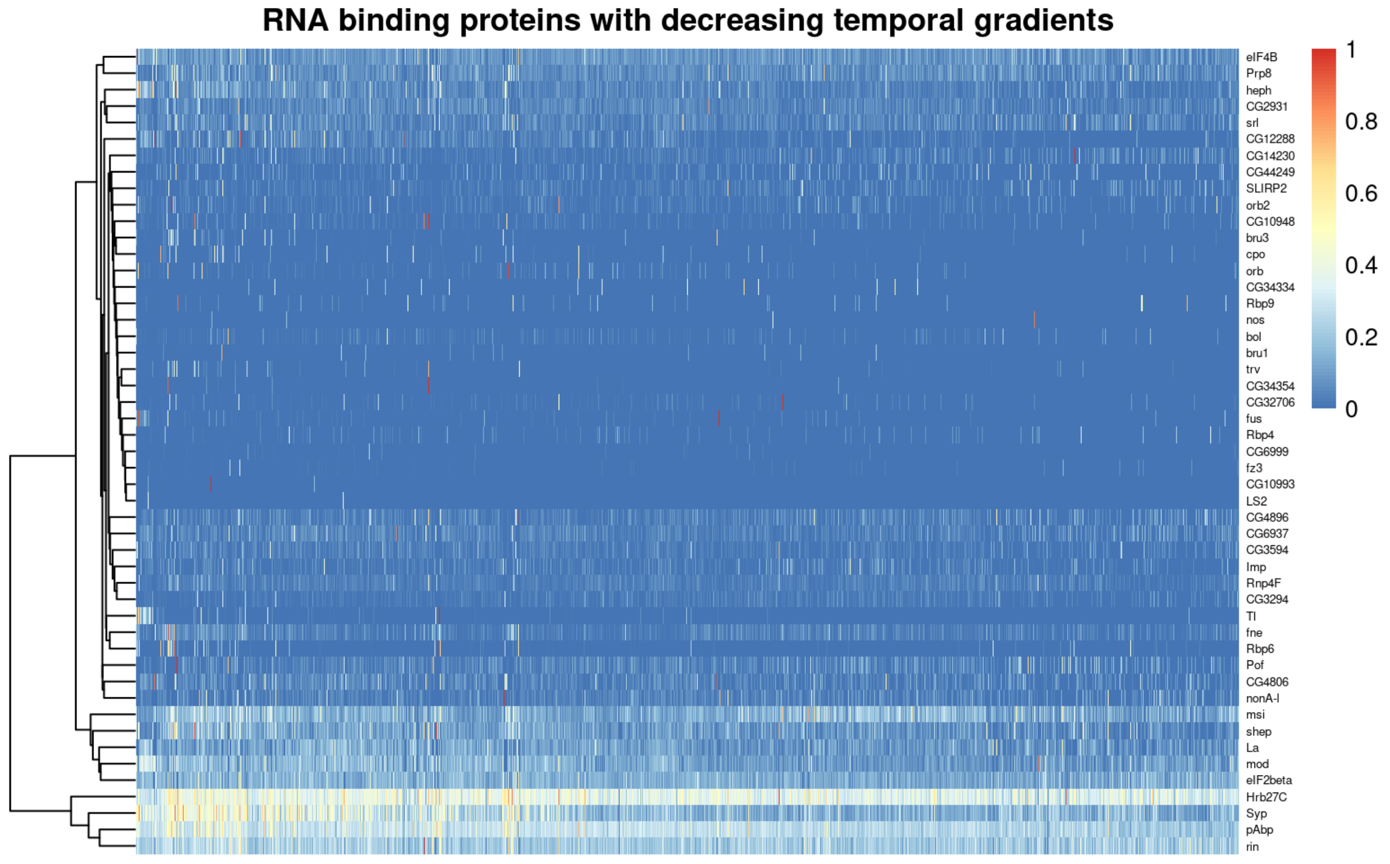
```
pst = pseudotime(cds)

spearman_corr_with_time = t(apply(normalized_counts(cds[rbp,], norm_method = "size_on
ly"),
                                1, function(gene) {
  if (sd(gene) == 0) {
      return(c(0, 1))
    } else {
      fit = cor.test(gene, pst, method = "spearman", exact = FALSE)
      return(c(fit$estimate, fit$p.value))
    }
}))

spearman_corr_with_time = data.frame(spearman_corr_with_time,
                                     fdr = p.adjust(spearman_corr_with_time[, 2],
                                                    method = "fdr"))

colnames(spearman_corr_with_time) = c("corr", "pval", "fdr")
cutoff = 0.05
spearman_corr_with_time = spearman_corr_with_time[spearman_corr_with_time$fdr <= cuto
ff,]
```

Heatmap for all RNA binding proteins significantly associated with pseudotime.

```r
pst = pseudotime(cds)
ordered_exprs = normalized_counts(cds[rownames(spearman_corr_with_time),],
                                  norm_method = "size_only")[, order(pst)]
colnames(ordered_exprs) = sort(pst)
## visualize percentages of maximum expression across pseudotime
ordered_perc_max_expr = ordered_exprs / apply(ordered_exprs, 1, max)

pos_corr = spearman_corr_with_time[spearman_corr_with_time$corr >= 0,]
neg_corr = spearman_corr_with_time[spearman_corr_with_time$corr < 0,]
pos = rownames(pos_corr)[order(-pos_corr$corr)]
neg = rownames(neg_corr)[order(neg_corr$corr)]

pheatmap(ordered_perc_max_expr[pos,],
         main = "RNA binding proteins with increasing temporal gradients",
         cluster_rows = TRUE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 5)
```



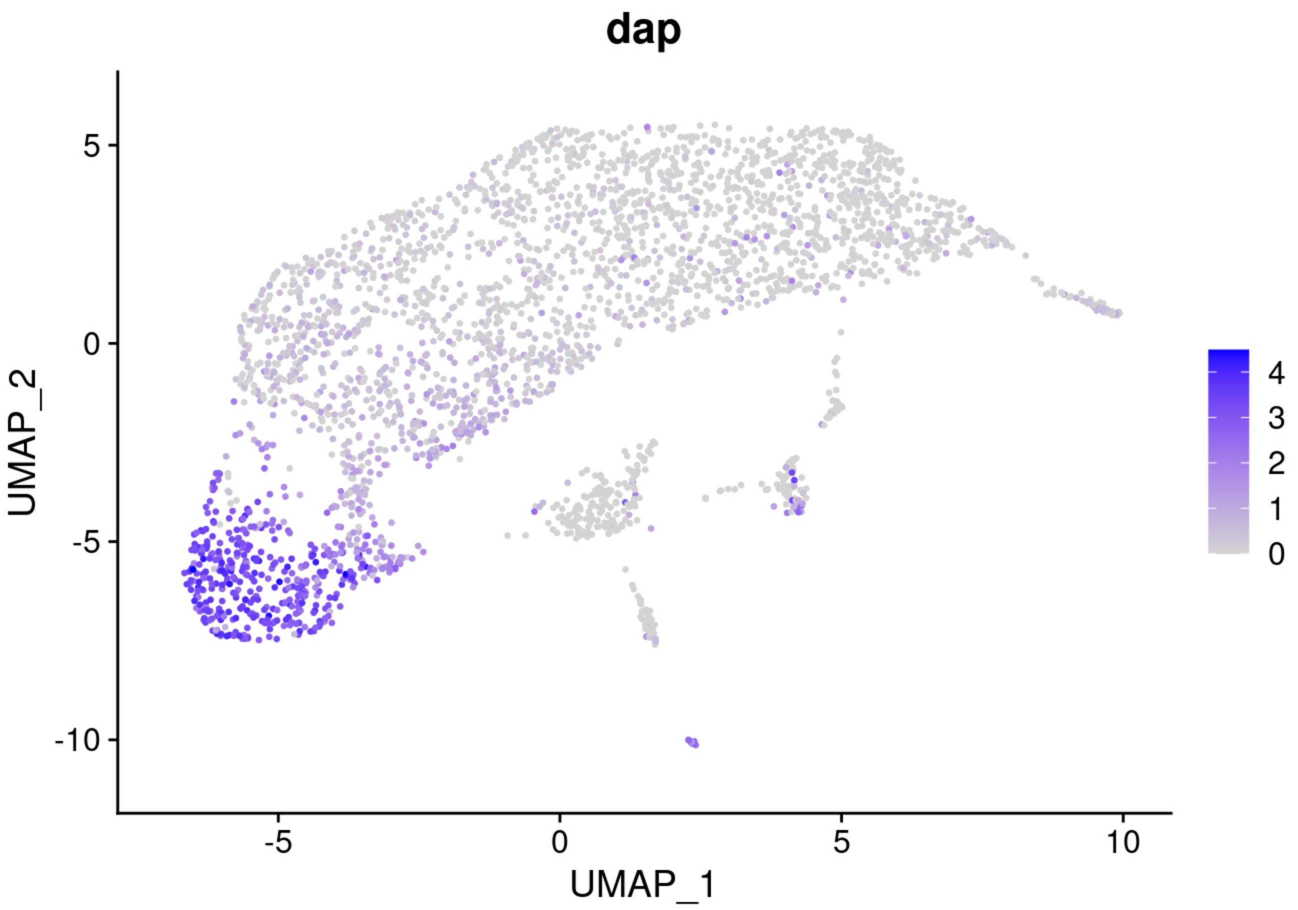
```
pheatmap(ordered_perc_max_expr[neg,],
         main = "RNA binding proteins with decreasing temporal gradients",
         cluster_rows = TRUE, cluster_cols = FALSE,
         show_colnames = FALSE,
         show_rownames = TRUE, fontsize_row = 5)
```



17 Dap temporal expression pattern

This analysis corresponds to Supplementary Figure 10A.

```
FeaturePlot(mnb, features = "dap")
```



18 Regress out cell cycle and repeat analysis

We repeated our basic analysis after regressing out cell cycle phase when normalizing data. These analyses correspond to Supplementary Figures 1E through 1I.

Start fresh.

```
rm(list = ls())
set.seed(1)

## load datasets separately
counts = list(Read10X("~/data/drosophila_nb/july_2019_run/MNB_1/outs/filtered_feature
_bc_matrix"),
              Read10X("~/data/drosophila_nb/dec_2019_run/MNB_2/outs/filtered_feature_
bc_matrix"))

## preprocess
counts = lapply(counts, function(x) {

  ## filter cells by dpn expression level and mitochondrial mRNA content
  dpn = x[which(rownames(x) == "dpn"),]
  feat.mito = grep("^mt:", rownames(x), value = TRUE)
  percent.mito = Matrix::colSums(x[feat.mito, ]) /
    Matrix::colSums(x)

  x = x[, dpn > 0 & percent.mito < 0.1]
  print(dim(x))
  return(x)

})
```

```
## [1] 17753   777
## [1] 17753  2302
```

```
## prepare for integration
mnbs = list(CreateSeuratObject(counts[[1]], project = "jul"),
            CreateSeuratObject(counts[[2]], project = "dec"))

mnbs_prep = lapply(mnbs, function(x) {
  x = NormalizeData(x)
  x = FindVariableFeatures(x, selection.method = "vst")
  x = PercentageFeatureSet(x, pattern = "^mt:", col.name= "percent.mt")
  return(x)
})

## integrate, make sure integrated features contain genes of interest
mnb_anchors = FindIntegrationAnchors(mnbs_prep)
```

```
## Computing 2000 integration features
```

```
## Scaling features for provided objects
```

```
## Finding all pairwise anchors
## Running CCA
```

```
## Merging objects
```

```
## Finding neighborhoods
```

```
## Finding anchors
```

```
##  Found 3030 anchors
```

```
## Filtering anchors
```

```
##  Retained 1954 anchors
```

```
## Warning: UNRELIABLE VALUE: Future ('future_lapply-1') unexpectedly generated
## random numbers without specifying argument 'future.seed'. There is a risk that
## those random numbers are not statistically sound and the overall results might
## be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper,
## parallel-safe random numbers are produced via the L'Ecuyer-CMRG method. To
## disable this check, use 'future.seed=NULL', or set option 'future.rng.onMisuse'
## to "ignore".
```

```
feat_int = rownames(mnbs[[1]]) ## all genes
mnb = IntegrateData(anchorset = mnb_anchors, features.to.integrate = feat_int)
```

```
## Merging dataset 1 into 2
```

```
## Extracting anchors for merged samples
```

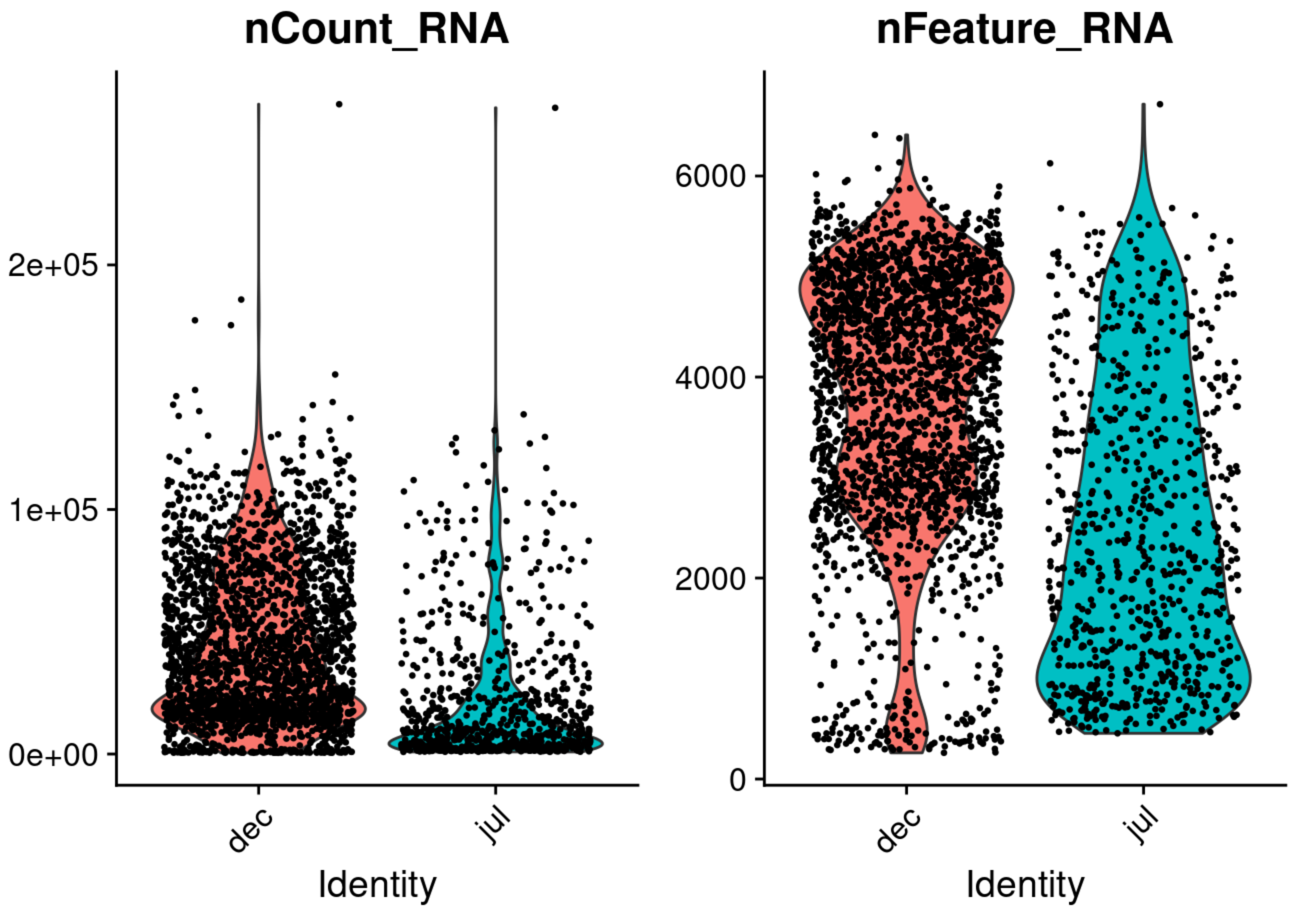
```
## Finding integration vectors
```

```
## Finding integration vector weights
```

```
## Integrating data
```

```
## Warning: Adding a command log without an assay associated with it
```

```
DefaultAssay(mnb) = "integrated"

## quality control
VlnPlot(mnb, features = c("nCount_RNA", "nFeature_RNA"), pt.size = 0.5)
```



```
keep = WhichCells(mnb, expression = nCount_RNA <= 1.7e5)
mnb = subset(mnb, cells = keep)

## scale data
mnb = ScaleData(mnb)
```

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

```
## cell cycle prediction
cc_genes = read.csv("List-Drosophila-cell-cycle-markers.csv",
                    header = TRUE, colClasses = "character")
s.genes = intersect(cc_genes$gene.name[cc_genes$phase == "S"],
                    rownames(mnb))
g2m.genes = intersect(cc_genes$gene.name[cc_genes$phase == "G2/M"],
                    rownames(mnb))
mnb = CellCycleScoring(mnb,
                       s.features = s.genes,
                       g2m.features = g2m.genes,
                       set.ident = TRUE)
```

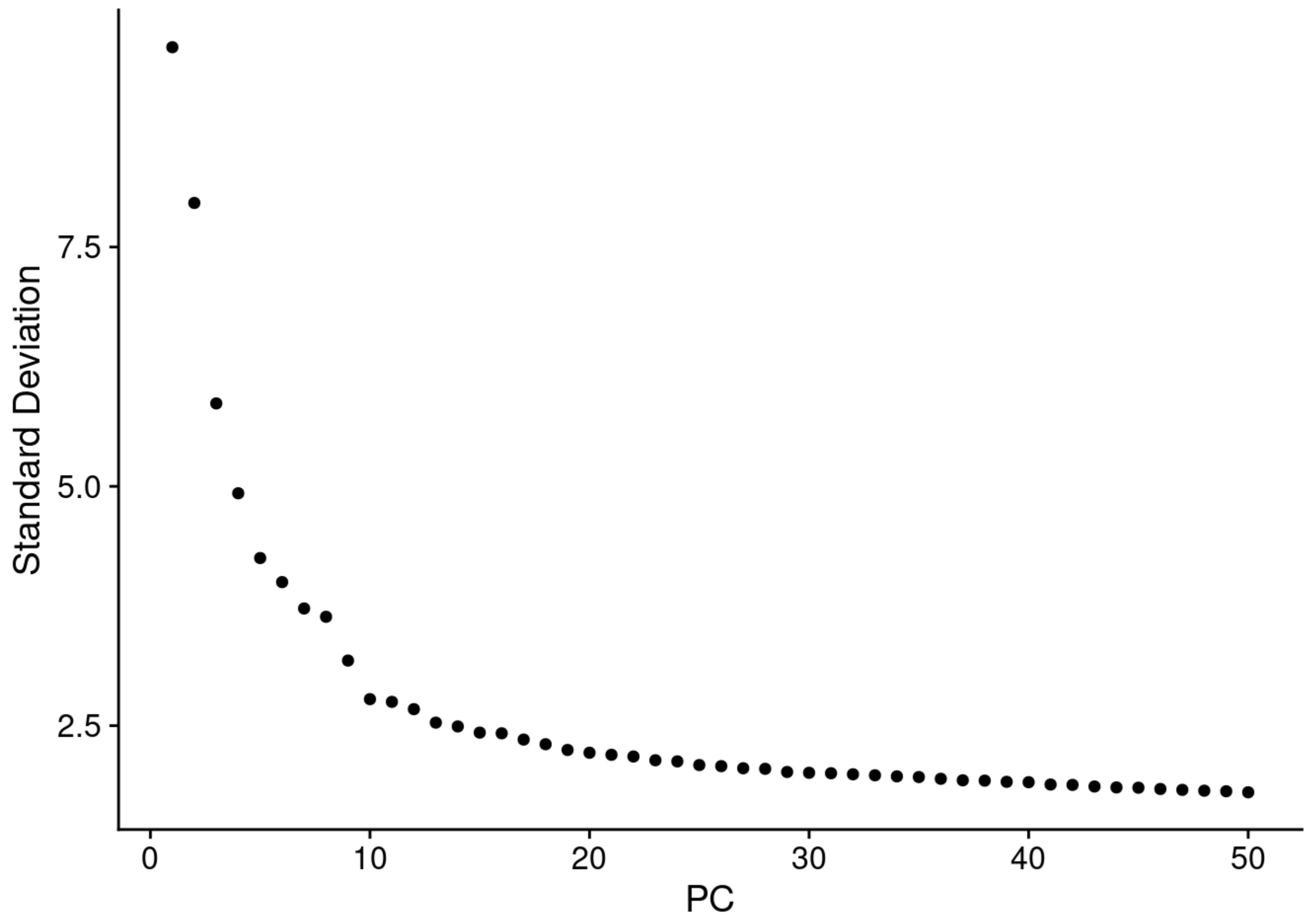
Regress out cell cycle and then re-cluster the cells.

```
## regress out cell cycle
mnb = ScaleData(mnb, vars.to.regress = c("S.Score", "G2M.Score"))
```

```
## Regressing out S.Score, G2M.Score
```

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

```
## pca and umap
npcs = 50
mnb = RunPCA(mnb, npcs = npcs, verbose = FALSE)
ElbowPlot(mnb, ndims = npcs)
```



```
dims_umap = 10
mnb = RunUMAP(mnb, reduction = "pca", dims = 1:dims_umap)
```

```
## 17:32:31 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 17:32:31 Read 3074 rows and found 10 numeric columns
## 17:32:31 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 17:32:31 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 17:32:32 Writing NN index file to temp file /tmp/Rtmp3YuEdx/file998d722f3d8b4
## 17:32:32 Searching Annoy index using 2 threads, search_k = 3000
## 17:32:32 Annoy recall = 100%
## 17:32:33 Commencing smooth kNN distance calibration using 2 threads
## 17:32:33 Initializing from normalized Laplacian + noise
## 17:32:33 Commencing optimization for 500 epochs, with 117792 positive edges
## 17:32:37 Optimization finished
```

```
## cluster
mnb = FindNeighbors(mnb, reduction = "pca")
```

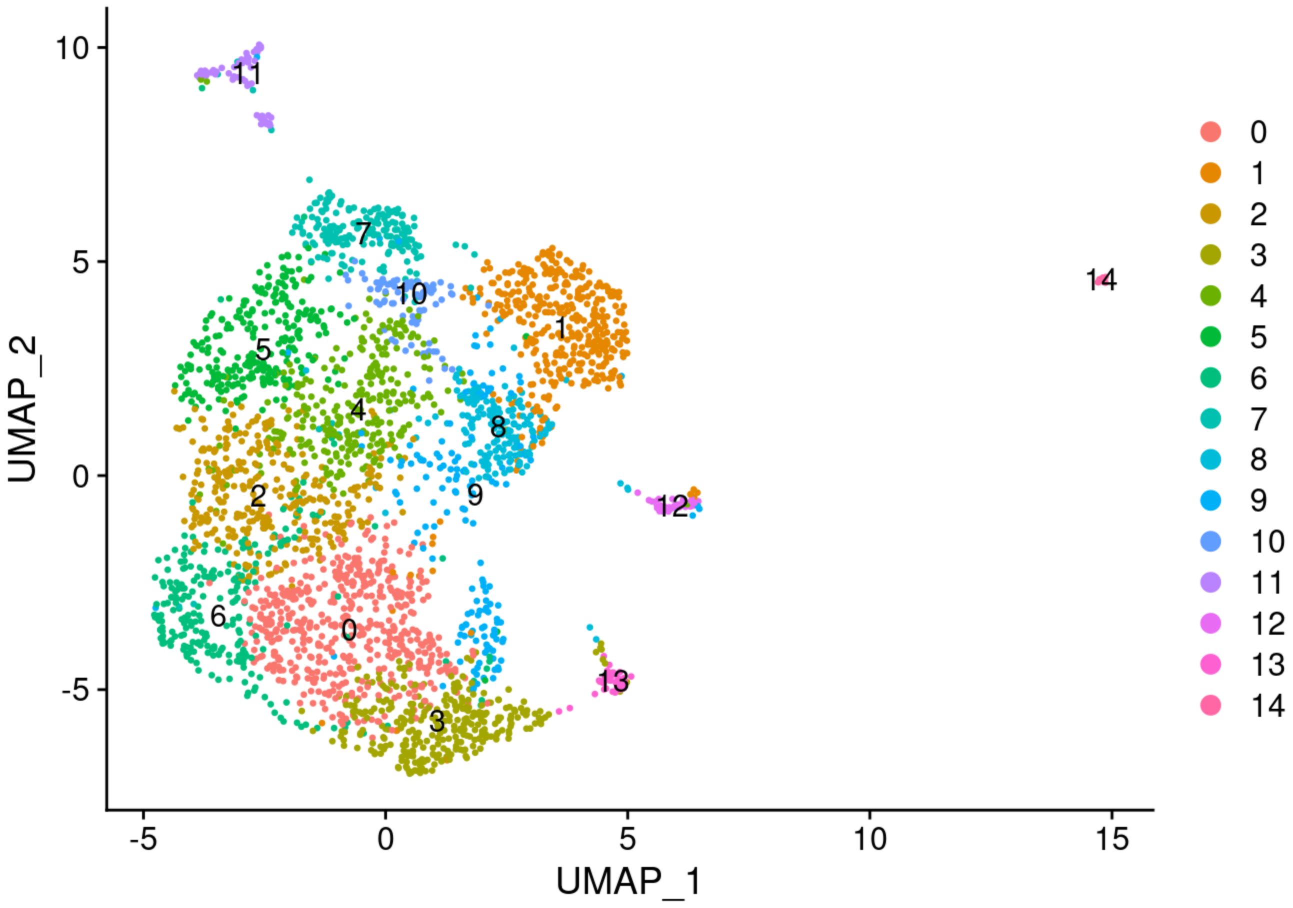
```
## Computing nearest neighbor graph
## Computing SNN
```

```
res = 0.9
mnb = FindClusters(mnb, resolution = res)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 3074
## Number of edges: 93800
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7898
## Number of communities: 15
## Elapsed time: 0 seconds
```

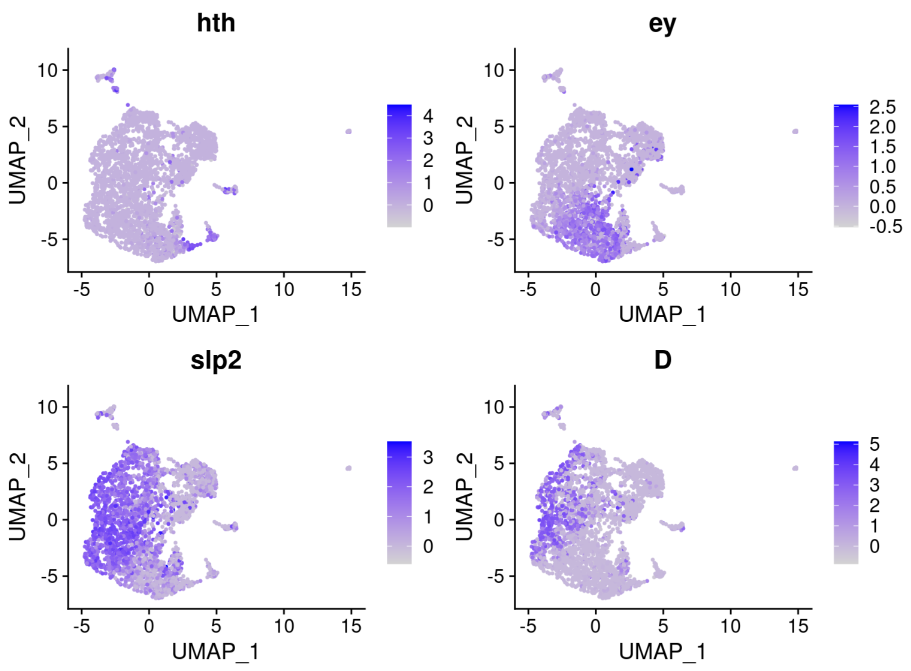
```
## Warning: UNRELIABLE VALUE: Future ('future_lapply-1') unexpectedly generated
## random numbers without specifying argument 'future.seed'. There is a risk that
## those random numbers are not statistically sound and the overall results might
## be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper,
## parallel-safe random numbers are produced via the L'Ecuyer-CMRG method. To
## disable this check, use 'future.seed=NULL', or set option 'future.rng.onMisuse'
## to "ignore".
```

```
DimPlot(mnb, reduction = "umap", label = TRUE)
```



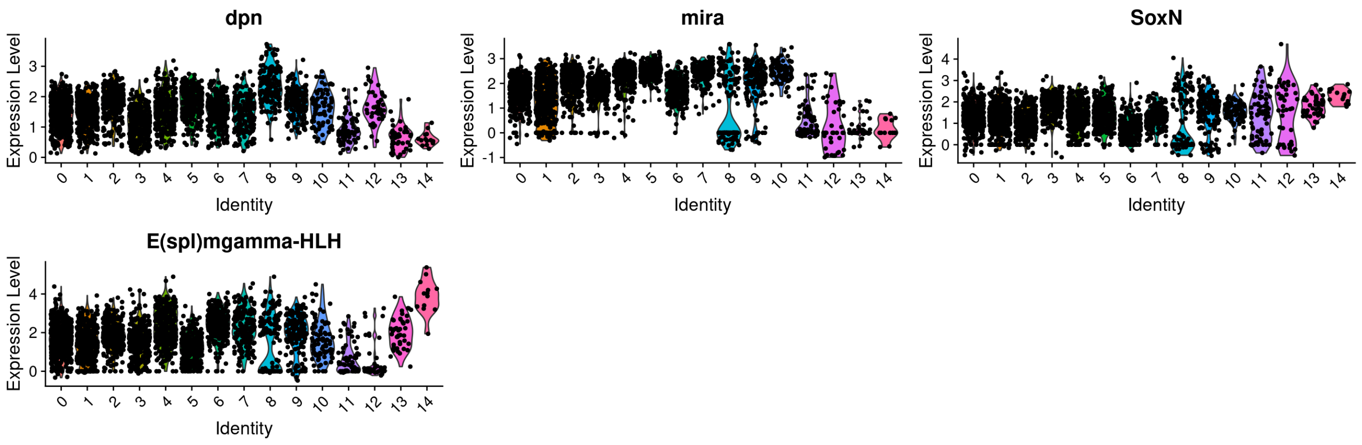
Visualize known temporal TF ordering.

```
knownttfs = c("hth", "ey", "slp1", "slp2", "D", "tll")
FeaturePlot(mnb,
            features = knownttfs,
            combine = TRUE, ncol = 3)
```



Identify outlier clusters.

```
VlnPlot(mnb, features = c("dpn", "mira", "SoxN", "E(spl)mgamma-HLH"), group.by = "seu
rat_clusters")
```



```
## outlier clusters look to be the same
outlier_clusters = c(8, 11, 12, 14)
```

Remove outlier cells and calculate pseudotime.

```
## remove cells
keep = setdiff(Idents(mnb), outlier_clusters)
mnb_subset = subset(mnb, idents = keep)

mnb_int_data = as.matrix(GetAssayData(mnb_subset, assay = "RNA", slot = "counts"))
cdata = mnb_subset[[]]
gdata = data.frame(gene_short_name = rownames(mnb_int_data), row.names = rownames(mnb
_int_data))

cds = new_cell_data_set(as(mnb_int_data, "sparseMatrix"),
                        cell_metadata = cdata,
                        gene_metadata = gdata)

## process as if using monocle 3
cds = preprocess_cds(cds, num_dim = dims_umap)
cds = reduce_dimension(cds)
```

```
## No preprocess_method specified, using preprocess_method = 'PCA'
```

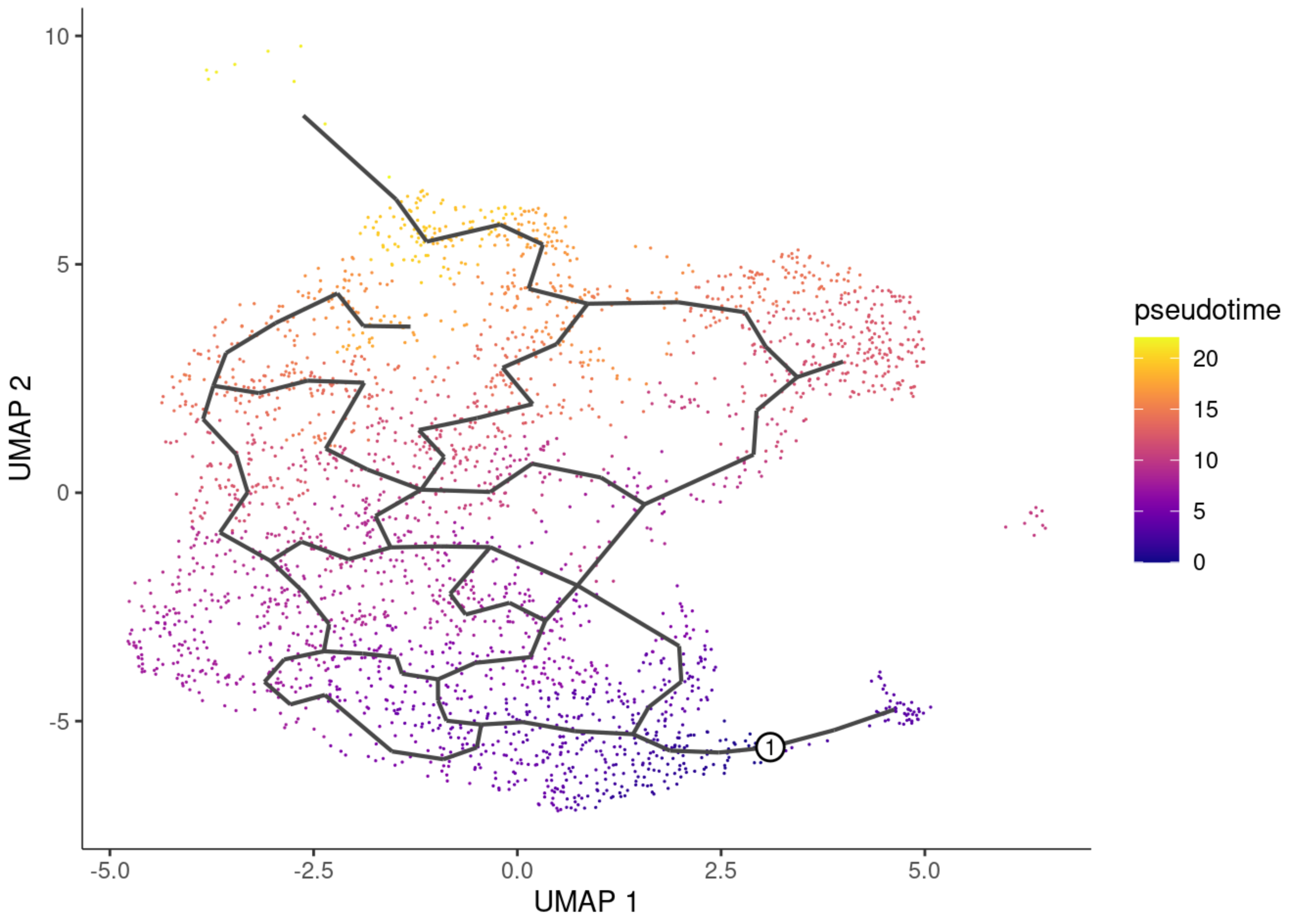
```
## replace things with seurat quantities
cds@int_colData@listData[["reducedDims"]][["PCA"]] = mnb_subset@reductions[["pca"]]@c
ell.embeddings
cds@int_colData@listData[["reducedDims"]][["UMAP"]] = mnb_subset@reductions[["umap"]]
@cell.embeddings

## use monocle clustering here
cds = cluster_cells(cds)

cds = learn_graph(cds,
                  use_partition = FALSE,
                  learn_graph_control = list(minimal_branch_len = 5))
```

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

```
## set root node
cv = cds@principal_graph_aux[["UMAP"]]$pr_graph_cell_proj_closest_vertex

part = cds@clusters@listData[["UMAP"]][["partitions"]]
roots = sapply(unique(part), function(p) {
  tmp = data.frame(hth = mnb_int_data["hth",],
                   cv = cv)[part == p,] %>%
    group_by(cv) %>%
    summarize(hth_expr = median(hth))
  tmp$cv[which.max(tmp$hth_expr)]
})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
root_pr_nodes = igraph::V(principal_graph(cds)[["UMAP"]])$name[roots]

cds = order_cells(cds, root_pr_nodes = root_pr_nodes)
plot_cells(cds,
           color_cells_by = "pseudotime",
           label_cell_groups = FALSE,
           label_leaves = FALSE,
           label_branch_points = FALSE,
           graph_label_size = 3)
```

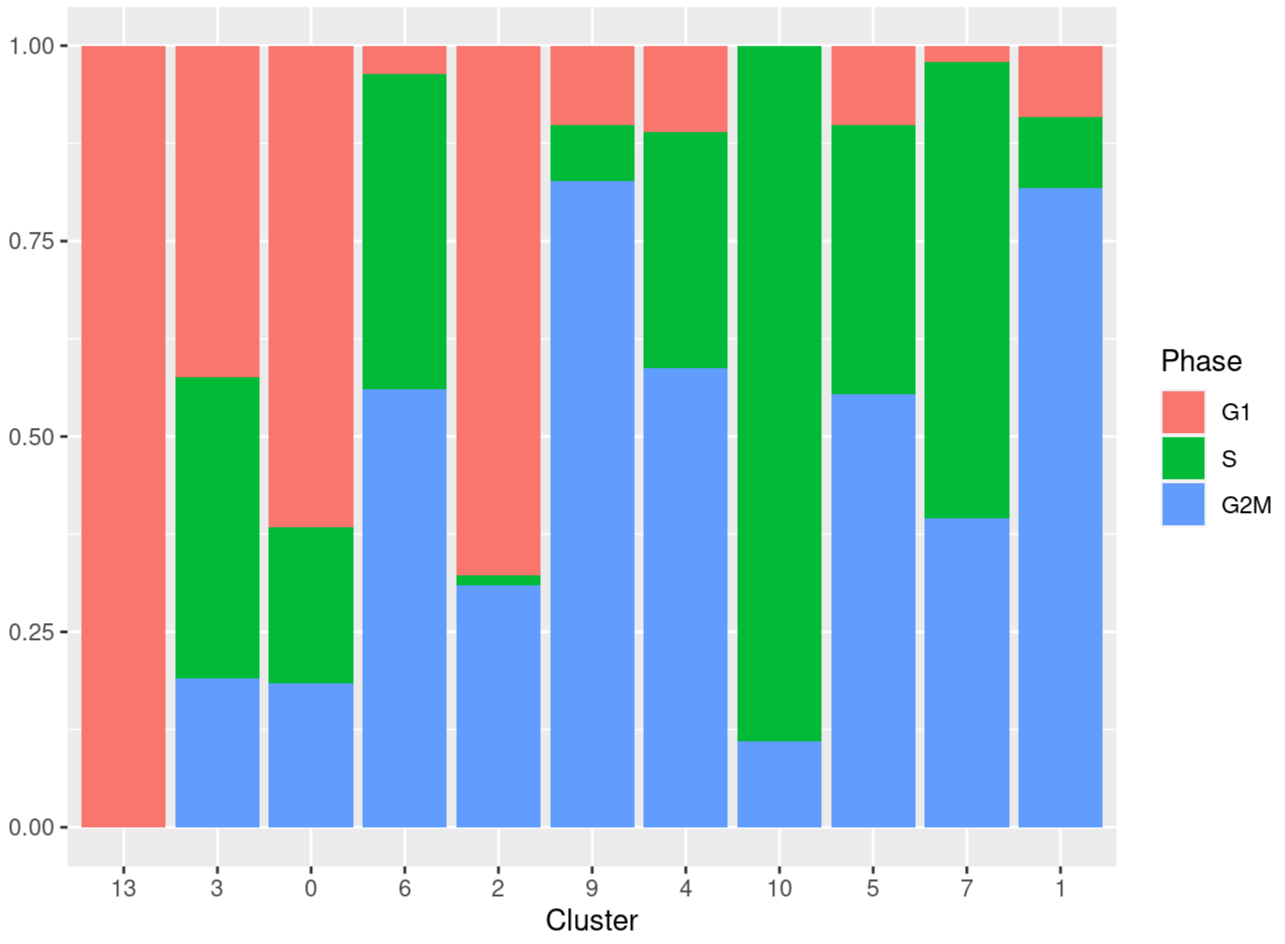


Pseudotime doesn’t line up with known temporal TF ordering. Visualize cell phase proportions using manually constructed cluster ordering that seems to be better correlate with known temporal TF ordering.

```
pst = pseudotime(cds)
dt = data.table(pst = pst, cluster = mnb_subset$seurat_clusters, phase = mnb_subset$P
hase)
clust_pst = dt[, list(med_pst = median(pst)), by = cluster]
print(clust_pst[order(clust_pst[, med_pst]), cluster])
```

```
##  [1] 3  13 0  6  9  2  4  1  5  10 7
## Levels: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
clust_v_phase = table(mnb$Phase, mnb$seurat_clusters)
## take out outliers
clust_v_phase = clust_v_phase[, !colnames(clust_v_phase) %in% outlier_clusters]
df = data.frame(t(clust_v_phase))
colnames(df) = c("Cluster", "Phase", "Count")

## manually constructed cluster order
df$Cluster = factor(df$Cluster, levels = as.character(c(13, 3, 0, 6, 2, 9, 4, 10, 5,
7, 1)))
df$Phase = factor(df$Phase, levels = c("G1", "S", "G2M"))
ggplot(data = df) +
  geom_bar(mapping = aes(x = Cluster, y = Count, fill = Phase),
           position = "fill", stat = "identity") +
  ylab("")
```



Regress out cell cycle using the alternate workflow see here (https://satijalab.org/seurat/articles/cell_cycle_vignette.html), then cluster and calculate pseudotime as in previous subsection.

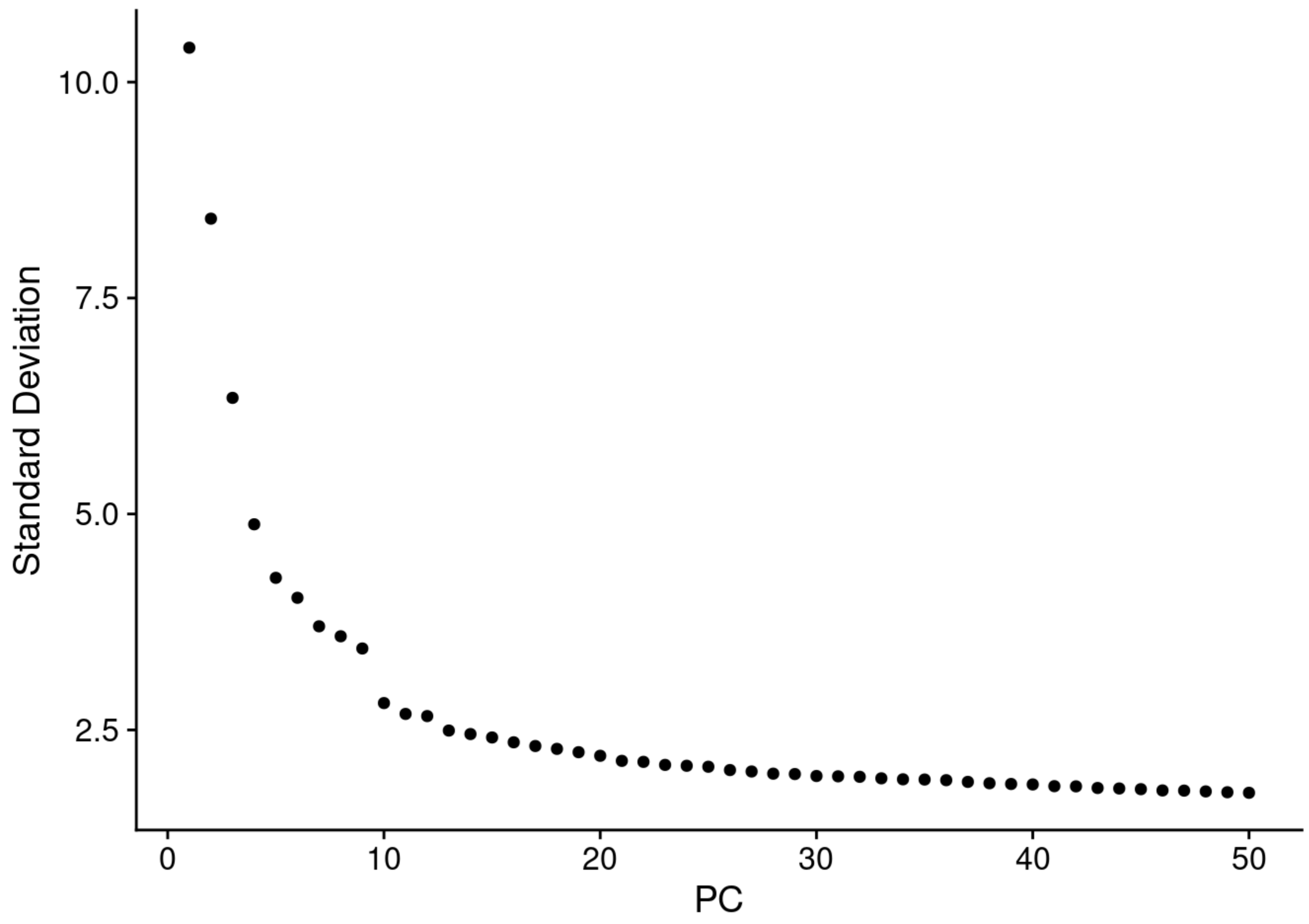
Cluster. Change resolution to get 15 clusters.

```
## regress out cell cycle
mnb$CC.Difference = mnb$S.Score - mnb$G2M.Score
mnb = ScaleData(mnb, vars.to.regress = c("CC.Difference"))
```

```
## Regressing out CC.Difference
```

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

```
## pca and umap
npcs = 50
mnb = RunPCA(mnb, npcs = npcs, verbose = FALSE)
ElbowPlot(mnb, ndims = npcs)
```



```
dims_umap = 10
mnb = RunUMAP(mnb, reduction = "pca", dims = 1:dims_umap)
```

```
## 17:33:24 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 17:33:24 Read 3074 rows and found 10 numeric columns
```

```
## 17:33:24 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 17:33:24 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
```

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

```
## **************************************************|
## 17:33:24 Writing NN index file to temp file /tmp/Rtmp3YuEdx/file998d76ad131de
## 17:33:24 Searching Annoy index using 2 threads, search_k = 3000
## 17:33:24 Annoy recall = 100%
## 17:33:25 Commencing smooth kNN distance calibration using 2 threads
## 17:33:26 Initializing from normalized Laplacian + noise
## 17:33:26 Commencing optimization for 500 epochs, with 120248 positive edges
## 17:33:30 Optimization finished
```

```
## cluster
mnb = FindNeighbors(mnb, reduction = "pca")
```

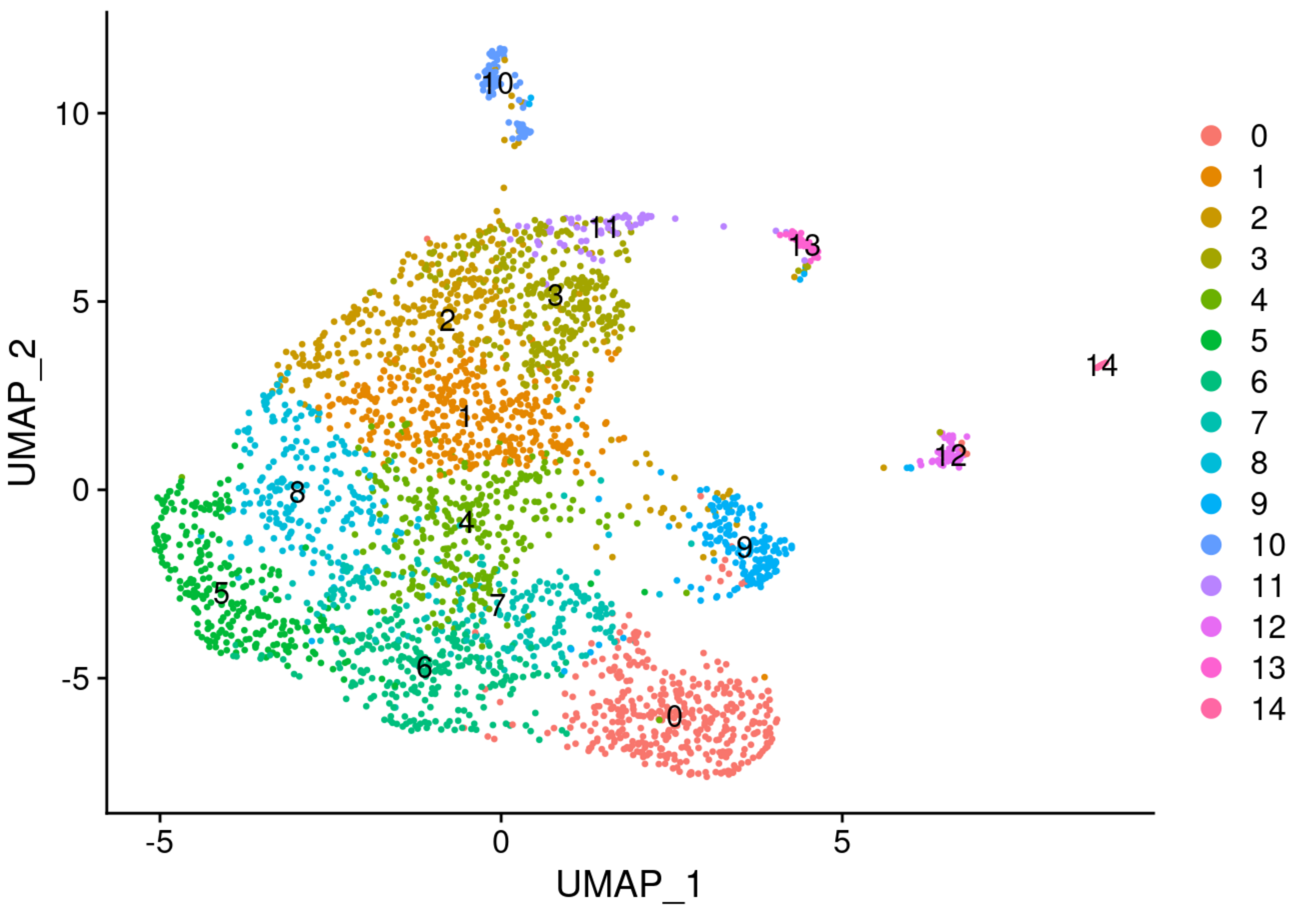
```
## Computing nearest neighbor graph
## Computing SNN
```

```
res = 1
mnb = FindClusters(mnb, resolution = res)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 3074
## Number of edges: 94737
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7855
## Number of communities: 15
## Elapsed time: 0 seconds
```

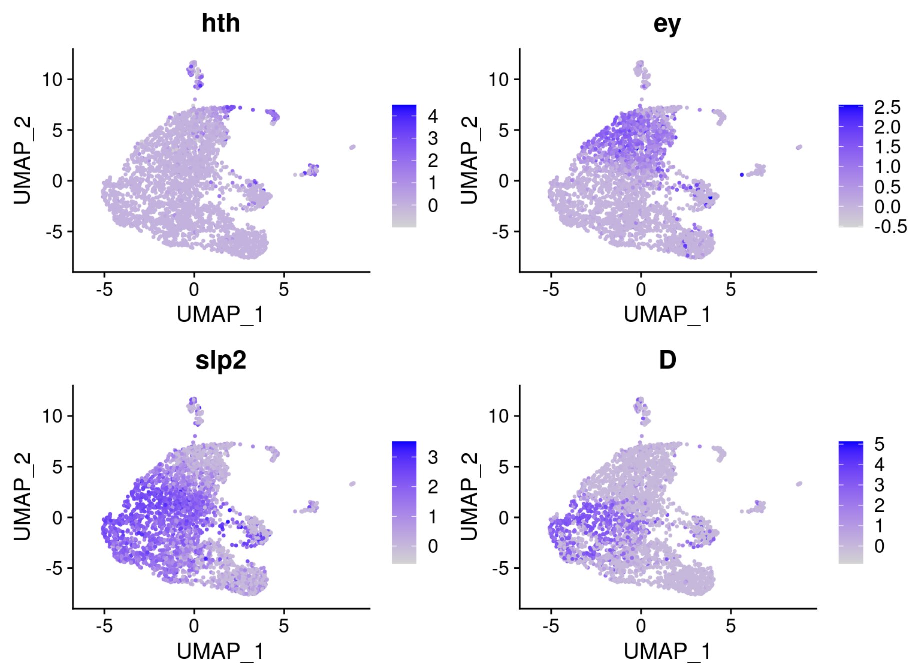
```
## Warning: UNRELIABLE VALUE: Future ('future_lapply-1') unexpectedly generated
## random numbers without specifying argument 'future.seed'. There is a risk that
## those random numbers are not statistically sound and the overall results might
## be invalid. To fix this, specify 'future.seed=TRUE'. This ensures that proper,
## parallel-safe random numbers are produced via the L'Ecuyer-CMRG method. To
## disable this check, use 'future.seed=NULL', or set option 'future.rng.onMisuse'
## to "ignore".
```

```
DimPlot(mnb, reduction = "umap", label = TRUE)
```



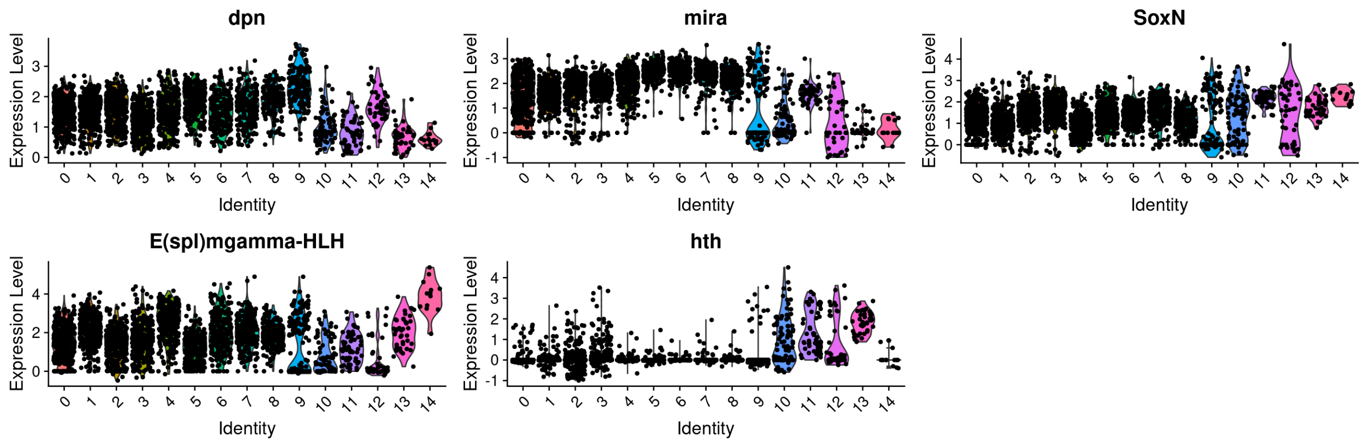
Visualize known temporal TF ordering.

```
knownttfs = c("hth", "ey", "slp1", "slp2", "D", "tll")
FeaturePlot(mnb,
            features = knownttfs,
            combine = TRUE, ncol = 3)
```



Identify outlier clusters.

```
VlnPlot(mnb, features = c("dpn", "mira", "SoxN", "E(spl)mgamma-HLH", "hth"), group.by
= "seurat_clusters")
```



```
outlier_clusters = c(9, 10, 12, 14)
```

Remove outlier cells and calculate pseudotime.

```
## remove cells
keep = setdiff(Idents(mnb), outlier_clusters)
mnb_subset = subset(mnb, idents = keep)

mnb_int_data = as.matrix(GetAssayData(mnb_subset, assay = "RNA", slot = "counts"))
cdata = mnb_subset[[]]
gdata = data.frame(gene_short_name = rownames(mnb_int_data), row.names = rownames(mnb
_int_data))

cds = new_cell_data_set(as(mnb_int_data, "sparseMatrix"),
                        cell_metadata = cdata,
                        gene_metadata = gdata)

## process as if using monocle 3
cds = preprocess_cds(cds, num_dim = dims_umap)
cds = reduce_dimension(cds)
```

```
## No preprocess_method specified, using preprocess_method = 'PCA'
```

```
## replace things with seurat quantities
cds@int_colData@listData[["reducedDims"]][["PCA"]] = mnb_subset@reductions[["pca"]]@c
ell.embeddings
cds@int_colData@listData[["reducedDims"]][["UMAP"]] = mnb_subset@reductions[["umap"]]
@cell.embeddings

## use monocle clustering here
cds = cluster_cells(cds)

cds = learn_graph(cds,
                  use_partition = FALSE,
                  learn_graph_control = list(minimal_branch_len = 5))
```

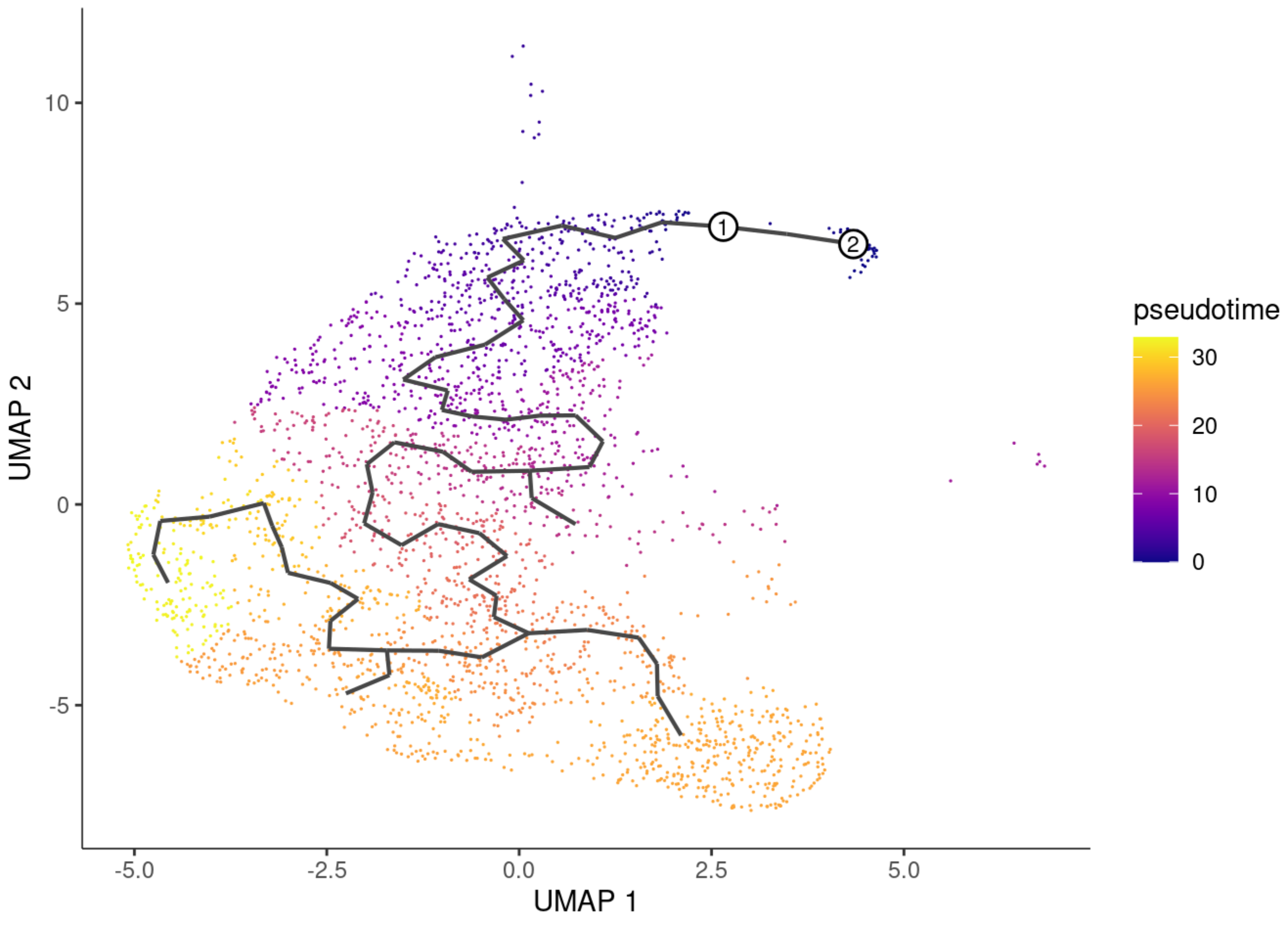
```
##
  |
  |                                                                      |   0%
  |
  |======================================================================| 100%
```

```
## set root node
cv = cds@principal_graph_aux[["UMAP"]]$pr_graph_cell_proj_closest_vertex
part = cds@clusters@listData[["UMAP"]][["partitions"]]
roots = sapply(unique(part), function(p) {
  tmp = data.frame(hth = mnb_int_data["hth",],
                   cv = cv)[part == p,] %>%
    group_by(cv) %>%
    summarize(hth_expr = median(hth))
  tmp$cv[which.max(tmp$hth_expr)]
})
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
root_pr_nodes = igraph::V(principal_graph(cds)[["UMAP"]])$name[roots]

cds = order_cells(cds, root_pr_nodes = root_pr_nodes)
plot_cells(cds,
           color_cells_by = "pseudotime",
           label_cell_groups = FALSE,
           label_leaves = FALSE,
           label_branch_points = FALSE,
           graph_label_size = 3)
```

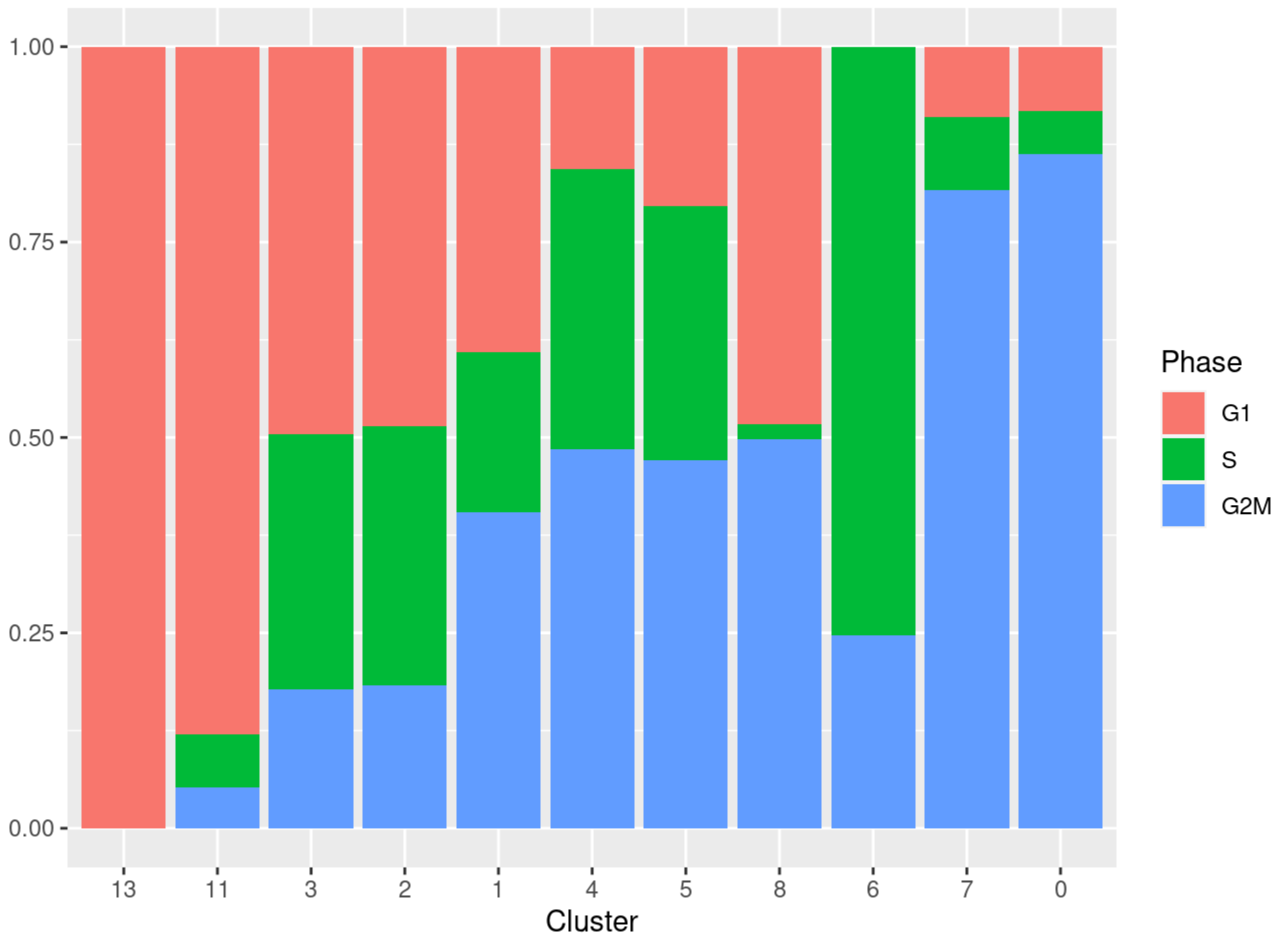


Pseudotime doesn’t line up with known temporal TF ordering. Visualize cell phase proportions using manually constructed cluster ordering that seems to be better correlate with known temporal TF ordering.

```
pst = pseudotime(cds)
dt = data.table(pst = pst, cluster = mnb_subset$seurat_clusters, phase = mnb_subset$P
hase)
clust_pst = dt[, list(med_pst = median(pst)), by = cluster]
print(clust_pst[order(clust_pst[, med_pst]), cluster])
```

```
##  [1] 13 11 3  2  1  4  7  6  0  8  5
## Levels: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

```
clust_v_phase = table(mnb$Phase, mnb$seurat_clusters)
## take out outliers
clust_v_phase = clust_v_phase[, !colnames(clust_v_phase) %in% outlier_clusters]
df = data.frame(t(clust_v_phase))
colnames(df) = c("Cluster", "Phase", "Count")

## manually constructed cluster order
df$Cluster = factor(df$Cluster, levels = as.character(c(13, 11, 3, 2, 1, 4, 5, 8, 6,
7, 0)))
df$Phase = factor(df$Phase, levels = c("G1", "S", "G2M"))
ggplot(data = df) +
  geom_bar(mapping = aes(x = Cluster, y = Count, fill = Phase),
           position = "fill", stat = "identity") +
  ylab("")
```



19 Session Information

```
sessionInfo()
```

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.9.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] readxl_1.3.1                data.table_1.13.6
##  [3] pheatmap_1.0.12             future_1.21.0
##  [5] mgcv_1.8-33                 nlme_3.1-152
##  [7] ggplot2_3.3.5               monocle3_0.2.3.0
##  [9] SingleCellExperiment_1.12.0 SummarizedExperiment_1.20.0
## [11] GenomicRanges_1.42.0        GenomeInfoDb_1.26.2
## [13] IRanges_2.24.1              S4Vectors_0.28.1
## [15] MatrixGenerics_1.2.0        matrixStats_0.61.0
## [17] Biobase_2.50.0              BiocGenerics_0.36.0
## [19] cowplot_1.1.1               Seurat_3.2.3
## [21] dplyr_1.0.2
##
## loaded via a namespace (and not attached):
##   [1] Rtsne_0.15                colorspace_2.0-0
##   [3] deldir_0.2-3              ellipsis_0.3.1
##   [5] ggridges_0.5.2            XVector_0.30.0
##   [7] proxy_0.4-24              spatstat.data_2.1-0
##   [9] farver_2.0.3              leiden_0.3.6
##  [11] listenv_0.8.0             ggrepel_0.9.0
##  [13] RSpectra_0.16-0           sparseMatrixStats_1.2.0
##  [15] codetools_0.2-18          splines_4.1.2
##  [17] knitr_1.33                polyclip_1.10-0
##  [19] jsonlite_1.7.2            ica_1.0-2
##  [21] cluster_2.1.2             png_0.1-7
##  [23] uwot_0.1.10               shiny_1.5.0
##  [25] sctransform_0.3.2         compiler_4.1.2
##  [27] httr_1.4.2                assertthat_0.2.1
##  [29] Matrix_1.4-0              fastmap_1.0.1
##  [31] lazyeval_0.2.2            limma_3.46.0
##  [33] later_1.1.0.1             htmltools_0.5.1.1
##  [35] tools_4.1.2                rsvd_1.0.3
##  [37] igraph_1.2.6               gtable_0.3.0
##  [39] glue_1.4.2                 GenomeInfoDbData_1.2.4
##  [41] RANN_2.6.1                 reshape2_1.4.4
##  [43] Rcpp_1.0.6                 spatstat_1.64-1
##  [45] scattermore_0.7            cellranger_1.1.0
##  [47] jquerylib_0.1.4            vctrs_0.3.6
##  [49] DelayedMatrixStats_1.12.1 lmtest_0.9-38
##  [51] xfun_0.24                  stringr_1.4.0
##  [53] globals_0.14.0             mime_0.9
##  [55] miniUI_0.1.1.1             lifecycle_0.2.0
##  [57] irlba_2.3.3                goftest_1.2-2
##  [59] zlibbioc_1.36.0            MASS_7.3-54
##  [61] zoo_1.8-8                  scales_1.1.1
##  [63] promises_1.1.1             spatstat.utils_2.1-0
##  [65] RColorBrewer_1.1-2         yaml_2.2.1
##  [67] leidenbase_0.1.2           reticulate_1.18
##  [69] pbapply_1.4-3              gridExtra_2.3
##  [71] sass_0.4.0                 rpart_4.1-15
##  [73] stringi_1.5.3              highr_0.8
##  [75] rlang_0.4.10               pkgconfig_2.0.3
##  [77] bitops_1.0-6               evaluate_0.14
##  [79] lattice_0.20-45            ROCR_1.0-11
##  [81] purrr_0.3.4                tensor_1.5
##  [83] labeling_0.4.2             patchwork_1.1.1
##  [85] htmlwidgets_1.5.3          tidyselect_1.1.0
##  [87] parallelly_1.23.0          RcppAnnoy_0.0.18
##  [89] plyr_1.8.6                 magrittr_2.0.1
##  [91] R6_2.5.0                   generics_0.1.0
##  [93] DelayedArray_0.16.0        withr_2.3.0
##  [95] pillar_1.4.7               fitdistrplus_1.1-3
##  [97] survival_3.2-13            abind_1.4-5
##  [99] RCurl_1.98-1.2             tibble_3.0.4
## [101] future.apply_1.7.0         crayon_1.3.4
## [103] KernSmooth_2.23-20         plotly_4.9.3
## [105] rmarkdown_2.11             viridis_0.5.1
## [107] grid_4.1.2                 digest_0.6.27
## [109] xtable_1.8-4               tidyr_1.1.2
## [111] httpuv_1.5.4               munsell_0.5.0
## [113] viridisLite_0.3.0          bslib_0.2.5.1
```