DAR F21 Project Status 3 Nonlinear Dimensionality Reduction & Density Based Clustering on Users' Average Transactions

IDEA-Blockchain (DeFi)

Duke Kwon

9/30/21

Contents

Weekly Work Summary 1
Personal Contribution 2
Discussion of Primary Findings 2
General Table of Contents: 2
Pre-Work Codebase 2

UMAP & HDBSCAN **8**
Problem Exploration 8
Solution Implementation, Results: 9
Further Analysis: 14

Weekly Work Summary

- RCS ID: kwond2
- Project Name: IDEA-Blockchain (DeFi)
- General Summary:
- This week, I worked on researching and implementing a nonlinear dimensionality reduction alternative to PCA (UMAP) of the AAVE data's users' averaged transactions, and also looked into other clustering methods (HDBSCAN) to improve over the simple k-means algorithm.
- I accomplished developing a high level understanding of how UMAP and HDBSCAN work, and extended the work done by Chris Cammilleri on his PCA for users' averaged transaction features.
- Github Commits:
- Branch: dar-kwond2 (However main code will be available to master/in this notebook)
- Directory: https://github.rpi.edu/DataINCITE/IDEA-Blockchain/tree/master/DefiResearch/StudentNotebooks/Assignment03
- References:
- The Following readthedocs provide an intuitive, high level insight to the algorithms used.
  - https://umap-learn.readthedocs.io/en/latest/
  - https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html
- Shared Code Base:

- Code was used from Chris Cammilleri's PCA & users' averaged transactions notebook, which is not available directly as a link. However, a clear distinction is made within the notebook.

Personal Contribution

- All code regarding nonlinear dimensionality reduction via UMAP and density based clustering via HDBSCAN was done exclusively by Duke Kwon. Comparisons were drawn to the k-means clusters generated from the original PCA notebook, along with comparing the quality of the visuals/projection (PCA vs UMAP).

Discussion of Primary Findings

- Problem: Since the principle components of PCA resulted in poor variance capture, the idea was to look into nonlinear dimensionality reduction techniques and other clustering methods to make the users' average transactions more interpretable, and also generate more distinguishable clusters that could be analyzed.
- Proposed Solution: We explore UMAP (Uniform Manifold Approximation Projection) for our choice of nonlinear dimensionality reduction, and HDBSCAN for our choice of clustering on the newly reduced data. A high level explanation is provided in the code below as we deploy them.
- Results: We produced multiple different embeddeding spaces of the users' average transaction features via UMAP (in 5D and 2D), and performed HDBSCAN clustering on the raw space, 5D space, and 2D space for comparisons, and was able to improve the visualizations of PCA and the clustering done by k-means.

General Table of Contents:

- Prework from PCA/generating features/Problem Exploration
- Problem Exploration
- Implementation & Results
  - UMAP in 2D & 5D
  - HDBSCAN in Various Dimensions
  - Comparing PCA & k-means vs. UMAP & HDBSCAN
- Further Analysis

Pre-Work Codebase

First, we introduce the codebase used to generate the user features from the standard AAVE dataset (assumed to be a CSV file).

```
library(ggplot2)
library(devtools)
library(ggbiplot)
library(gplots)
library(RColorBrewer)
library(beeswarm)
library(tidyverse)
```

```
## Warning in system("timedatectl", intern = TRUE): running command 'timedatectl'
## had status 1
```

```
library(ggbeeswarm)
library(foreach)
library(doParallel)
```

We begin by loading the csv file in to a dataframe.

```
#load in csv file to data frame
df<-read_csv(file='transactions.csv')
```

```
## Rows: 481519 Columns: 26
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): borrowRateMode, reserve, type, collateralReserve, principalReserve...
## dbl (19): amount, borrowRate, onBehalfOf, pool, timestamp, user, reservePric...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

We reformat the dataframe so that each row represents a user, and the columns represent a summarization of the user's transaction history.

```
#group by user and get time of user's first and last transaction, as well as number of transactions
df.users<- df%>%group_by(user)%>%
  summarise(timefirst=min(timestamp), timelast=max(timestamp), N=n())
#get the time the user has been active
df.users$timeactive<-df.users$timelast-df.users$timefirst
#get log amounts for columns
df$logUSD<-log(df$amountUSD)
df$logCollateralUSD<-log(df$amountUSDCollateral)
#get user's transaction information
for(Type in c(unique(df$type))){
  #filter for only transactions of certain type
  df.type <-filter(df%>%group_by(user)%>%
                     count(type),type==Type)

  #add means of each transaction type
  if(Type!="liquidation" || Type!="swap"){
    df.mean<-filter(df,type==Type)%>%
      group_by(user)%>%
      summarise(Mean=mean(logUSD))
    colnames(df.mean)[2]<-paste('mean_',Type,sep='')
    df.users<-merge(x=df.users,y=df.mean,by="user",all.x=TRUE)
  }

  #add counts of transaction types to df
  ntypes<-paste("n",Type,sep='')
  colnames(df.type)[3]<-ntypes
  df.users<-merge(x=df.users,y=select(df.type,user,ntypes),by="user",all.x=TRUE)

  #get proportion of transaction types and weekly number of transaction type
  df.users[paste("prop_",Type,sep='')]<-(df.users[ntypes]+.05)/((df.users$N)+.3)
  df.users[paste("weekly_",Type,sep='')]<-df.users[ntypes]/(((df.users$timeactive)+1)/(3600*24*7))
}
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(ntypes)` instead of `ntypes` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
#df.users
```

Next, we select only the columns we wish to cluster on.

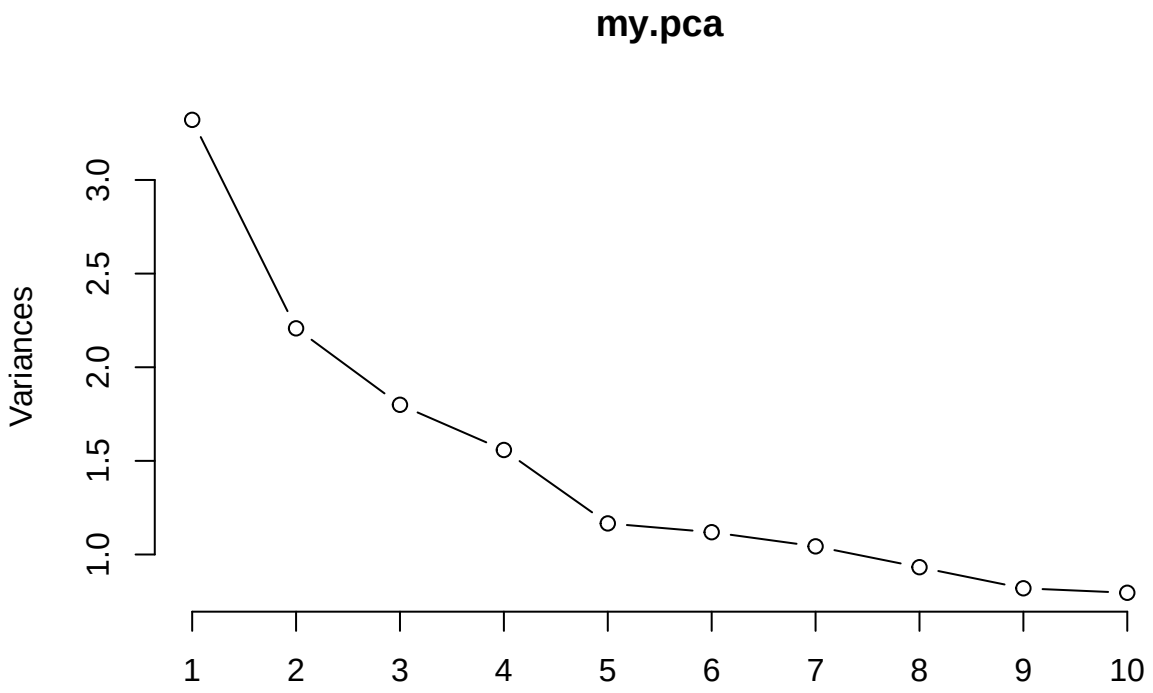
```
#subset only columns we wish to scale by removing columns that we will not cluster on
df.sub<-select(df.users,-c(user,timefirst,timelast,mean_liquidation,nborrow,nrepay,nswap,nliquidation,n
#repalce missing values as 0's
df.sub<-df.sub%>%replace(is.na(.),0)
```

We scale the data to prepare for PCA.

```
#scale data
df.scaled<-df.sub%>%mutate_all(scale)
#df.scaled
```

Now, we perform PCA on the scaled data.

```
df.scaled <- df.scaled[-which(names(df.scaled) == "mean_swap")]
#perform pca on data
my.pca<-prcomp(df.scaled,retx=TRUE,center=FALSE,scale=FALSE) # Run PCA and save to my.pca
#make scree plot
plot(my.pca, type="line")
```




```
#summary of pca
summary(my.pca)
```

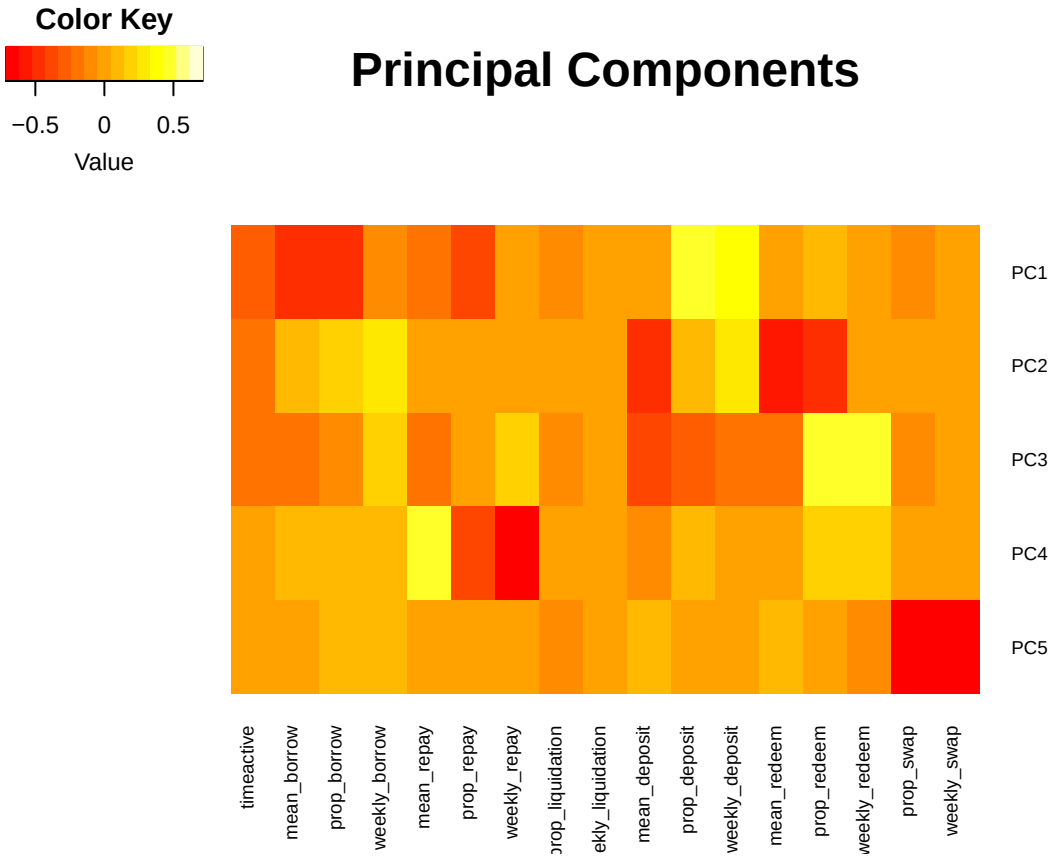
```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.8224 1.4858 1.3414 1.24821 1.07980 1.05787 1.02149
## Proportion of Variance 0.1953 0.1299 0.1058 0.09165 0.06859 0.06583 0.06138
## Cumulative Proportion  0.1953 0.3252 0.4311 0.52272 0.59130 0.65713 0.71851
##                           PC8     PC9   PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.96547 0.90530 0.8920 0.80927 0.72960 0.60864 0.52545
## Proportion of Variance 0.05483 0.04821 0.0468 0.03852 0.03131 0.02179 0.01624
## Cumulative Proportion  0.77334 0.82155 0.8683 0.90688 0.93819 0.95998 0.97622
```

```
##                            PC15    PC16    PC17
## Standard deviation     0.46980 0.42285 0.06862
## Proportion of Variance 0.01298 0.01052 0.00028
## Cumulative Proportion  0.98921 0.99972 1.00000
```

```
ncomps=5
```

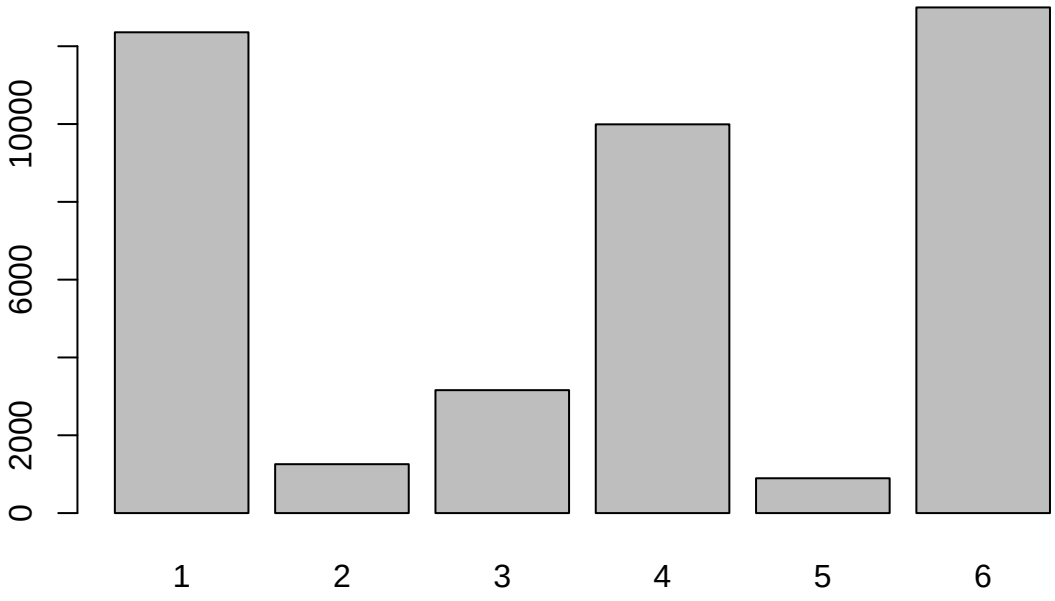
Next, we plot a heatmap to show the makeups of the principal components.

```
V <- t(my.pca$rotation[,1:ncomps]) # We transpose to make the principal components be rows
heatmap.2(V, main='Principal Components', cexRow=0.75, cexCol=0.75, scale="none", dendrogram="none",
Colv= FALSE, Rowv=FALSE, tracecol=NA,density.info='none')
```



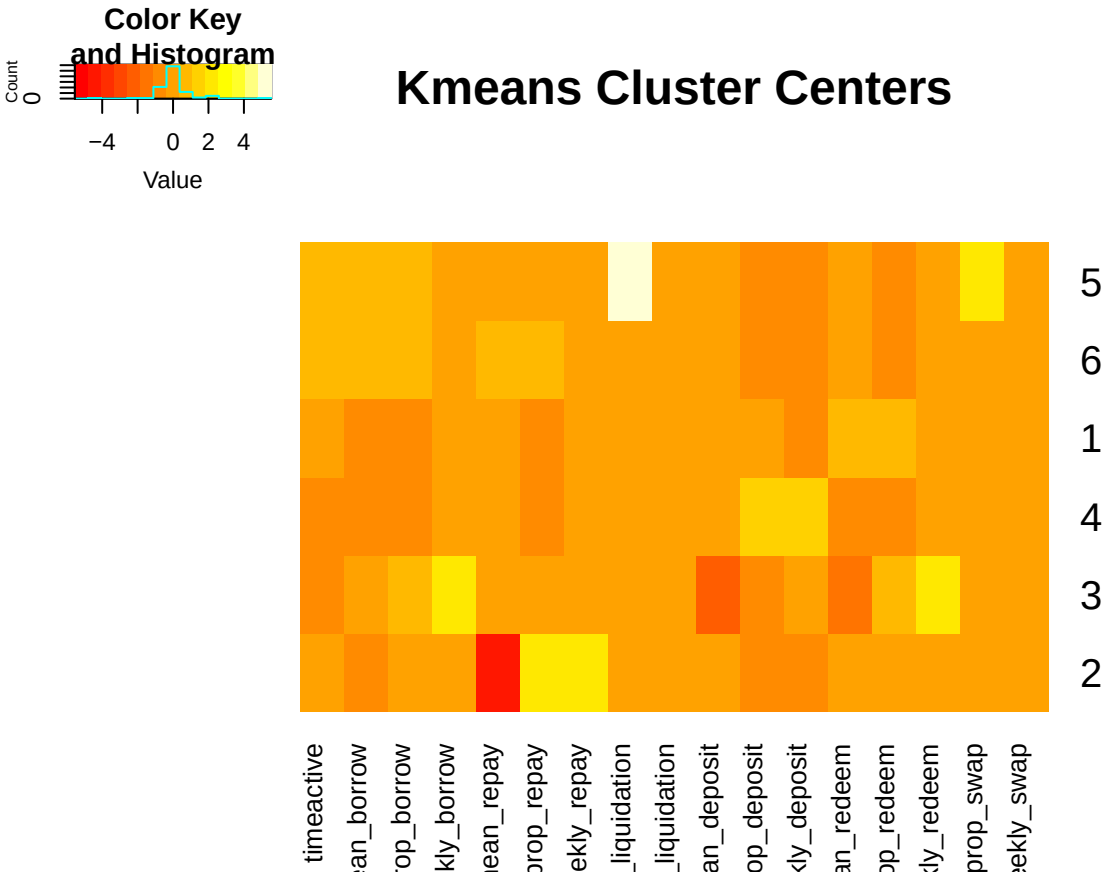
Finally, we run the kmeans clustering algorithm on the data. We select the number of clusters euqal to the number of selected components.

```
#run kmeans algorithm
pca.matrix<-my.pca$x[,1:ncomps]
set.seed(1)
km <-kmeans(df.scaled,6)
#plot frequencies of each cluster
barplot(table(km$cluster),main="Kmeans Cluster Size")
```



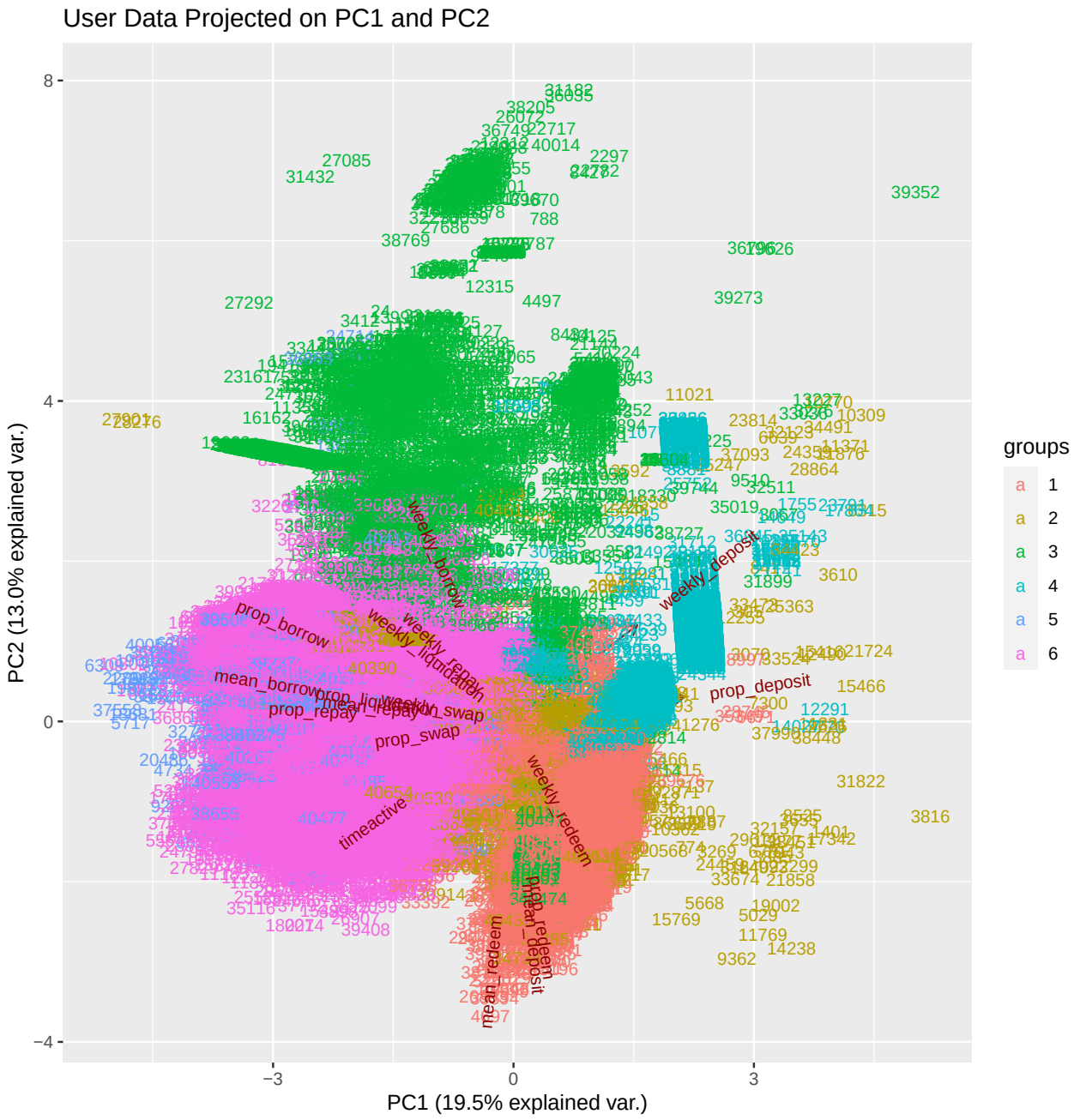
Finally, we view the centers of the kmeans clusters.

```
#make heatmap of cluster centers
heatmap.2(km$centers,
scale = "none",
dendrogram = "none",
Colv=FALSE,
cexCol=1.0,
main = "Kmeans Cluster Centers",
trace ="none")
```



Exploring the Influential Factors with a Biplot

```
plot1<-ggbiplot(my.pca,choices=c(1,2),
  labels=rownames(df.scaled), #show point labels
  var.axes=TRUE, # Display axes
  ellipse = FALSE, # Don't display ellipse
  obs.scale=1,
  groups=as.factor(km$cluster)) +
  ggtitle("User Data Projected on PC1 and PC2 ")
plot1
```



UMAP & HDBSCAN

Problem Exploration

There are 2 primary results we wish to extend about from the PCA analysis. First, we see that the first principal component only explains 19.5% of the variance, which is quite low. Generally, we can try new features, but we'll explore upon the idea that the data is nonlinear, and PCA will not be able to capture this nonlinearity. Secondly, notice how there exist many smaller clusters within the PCA visualization, and also points that are scattered about, and are always given a label. We look to address capturing clusters that may be smaller, but have something more specific in common. Also, we look to not include outliers in our clusters (which k-means automatically does).

Solution Implementation, Results:

We can further try some nonlinear dimensionality reduction and clustering techniques. Here we implement UMAP (uniform manifold approximation projection), which effectively learns a manifold approximation of the distribution our data lies in (in the simplest case, effectively generating a directed weighted graph via a style of knn), and maps the approximation to a lower dimensional space by minimizing a style of cross entropy error, iteratively minimizing the objective so that close points are weighted close together, and far points are pushed apart.

Under the assumption that blockchain data may have some special, nonlinear structures, we can also cluster via a density based approach, which we do through HDBSCAN (hierarchical density-based spatial clustering and applications). HDBSCAN is effective in mitigating the issues of some data not being close to the centers, but being locally close to other points that are locally close to the mean centers.

```
#install.packages("umap")
#install.packages("dbscan")
library(umap)
library(dbscan)
```

Here we compute two lower dimensional projections of the scaled data. "users.umap" contains a 2-D easily visualizable set of mapped vectors, whereas "users.umap_5" has embedded the original data into a 5-D space. 2-D is mainly for the visualization, and the 5-D embedding is what we'll be performing the clustering on, as less information is lost.

```
set.seed(42)
users.umap = umap(df.scaled)
```

```
## Warning: failed creating initial embedding; using random embedding instead
```

```
set.seed(43)
users.umap_5 = umap(df.scaled, n_components = 5)
```

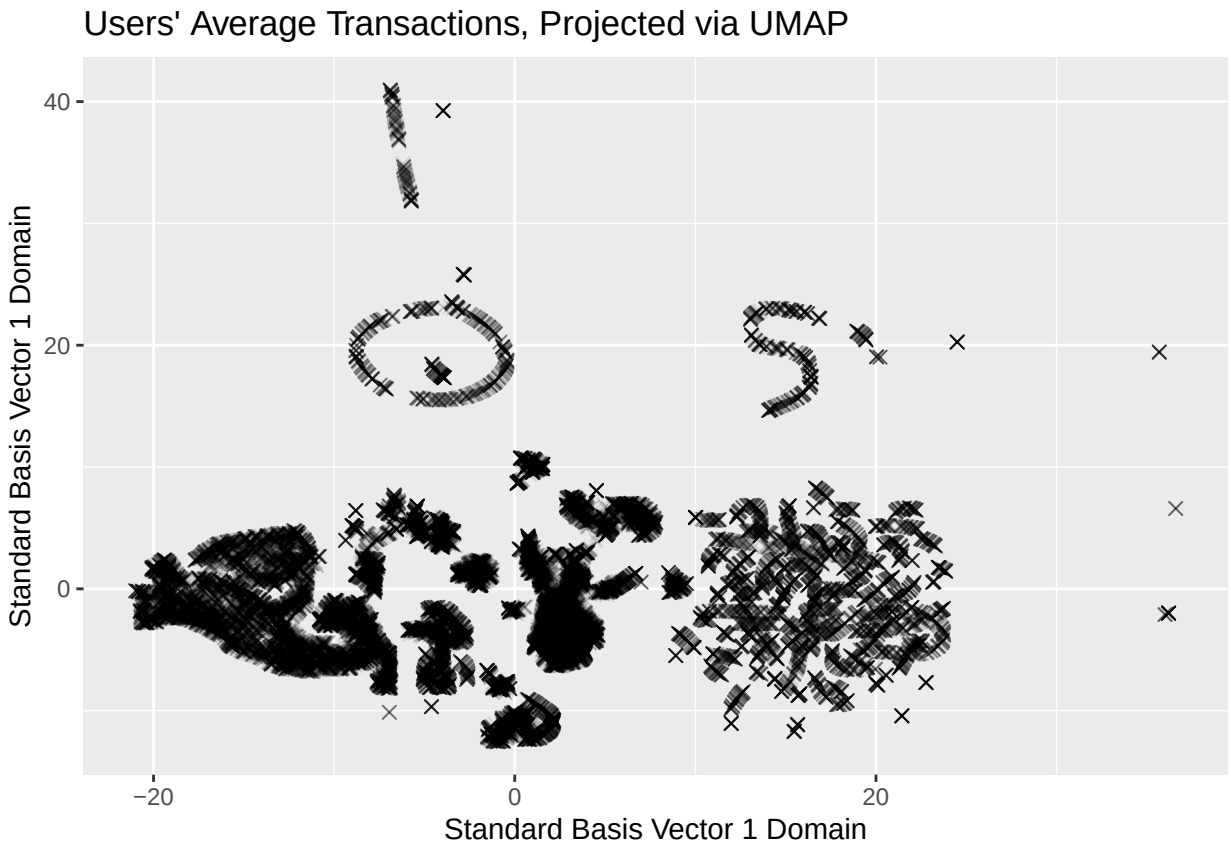
```
## Warning: failed creating initial embedding; using random embedding instead
```

The layout column of the umap object contains the list of users/vectors in the projected space. Here we plot the data in 2-D. Note that our axes aren't particularly meaningful, as it is an embedded space that may not have much interpretable meaning. However, we do know the distances between points have a certain degree of meaning.

```
head(users.umap$layout)
```

```
##            [,1]      [,2]
## [1,]  -6.398588  6.571819
## [2,]   3.896180 -5.184153
## [3,] -19.371077  2.371569
## [4,] -14.715164 -2.206104
## [5,]   3.018769  7.498836
## [6,]  -4.265725 -1.508643
```

```
df_umap <- data.frame(users.umap$layout)

ggplot(df_umap, aes(x = X1, y = X2))  + geom_point(alpha = 0.1, size= 2,shape = 4) + ggtitle("Users' Av
```



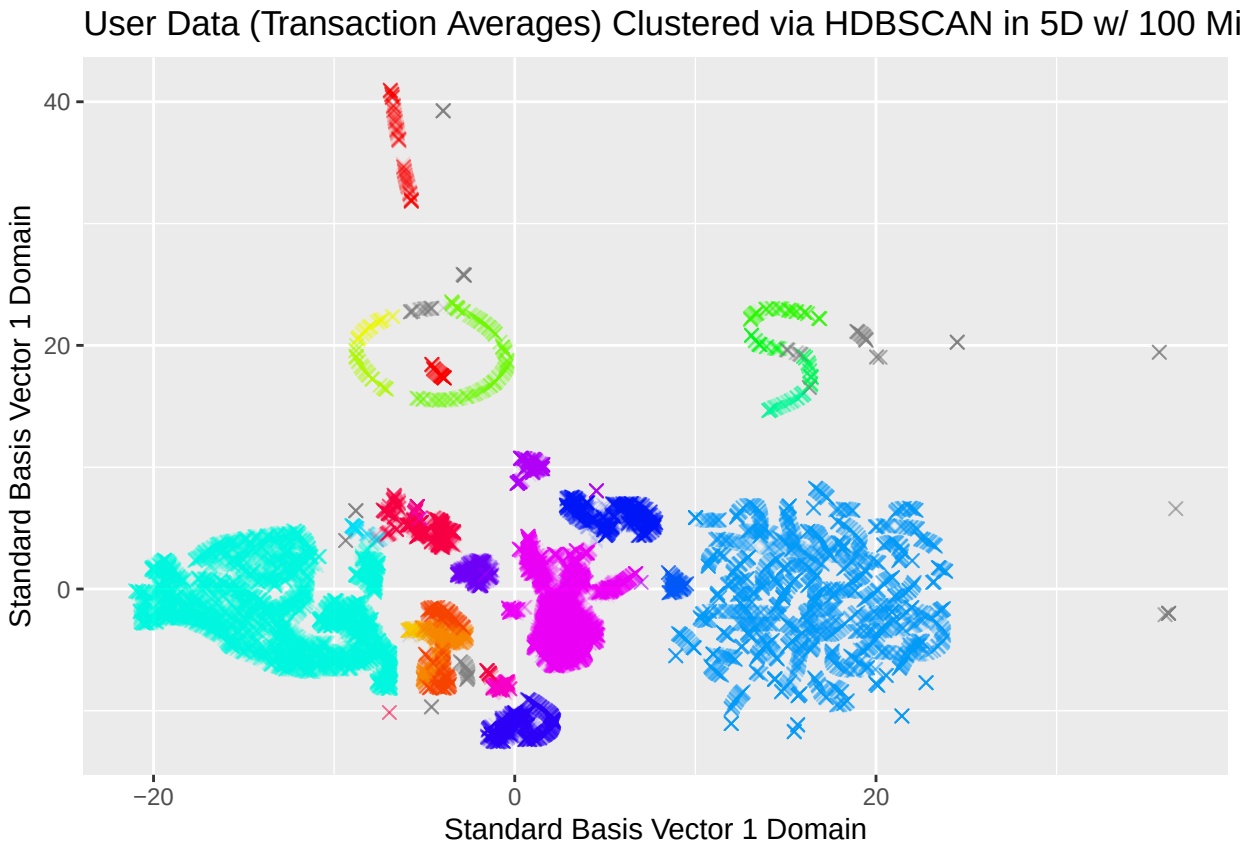
We now cluster the data via a density based approach, HDBSCAN (hierarchical density based spatial clustering of applications). We use a moderately high Minimum Points hyperparameter, which determines minimum points required to be a cluster. For the other parameters, we use the default which tend to be good enough for our exploratory analysis. Generally, HDBSCAN (especially density-based distance methods) do poorly on clustering in high dimensions, but we would still like more information than just 2-D (Consider the plot above - as a projection, we know certain information is lost/compressed.) That being said, recall we've computed an approximation in 5-D, which we do the clustering on. Which we'll then project down to 2D for visualizations.

In an attempt to showcase some of the issues with clustering with HDBSCAN in both the full and 2D space, we also run clustering on them (users.hdb_raw corresponds to full dimensions and users.hdb corresponds to clustering in just 2D).

```
## takes time!
set.seed(42)
users.hdb <- hdbscan(users.umap$layout, minPts = 100)
set.seed(43)
users.hdb_5 <- hdbscan(users.umap_5$layout, minPts = 100)
set.seed(44)
users.hdb_raw <- hdbscan(df.scaled, minPts = 100)
```

```
df_umap <- data.frame(users.umap$layout)
n_colors = max(users.hdb_5$cluster)
pre_colors <- rainbow(n_colors)
palette1 <- c("#808080", pre_colors) ## set 0 to a light grey as outliers
color_labels <- vector("character", length = length(users.hdb_5$cluster))
for (i in 0:n_colors) {
  color_labels[users.hdb_5$cluster == i] = palette1[i+1]
```

```
}
cluster_names <- c("Outliers", c(1:n_colors))

ggplot(df_umap, aes(x=X1, y=X2)) + geom_point(alpha = 0.1, size = 2, shape = 4, colour = color_labels) +
```

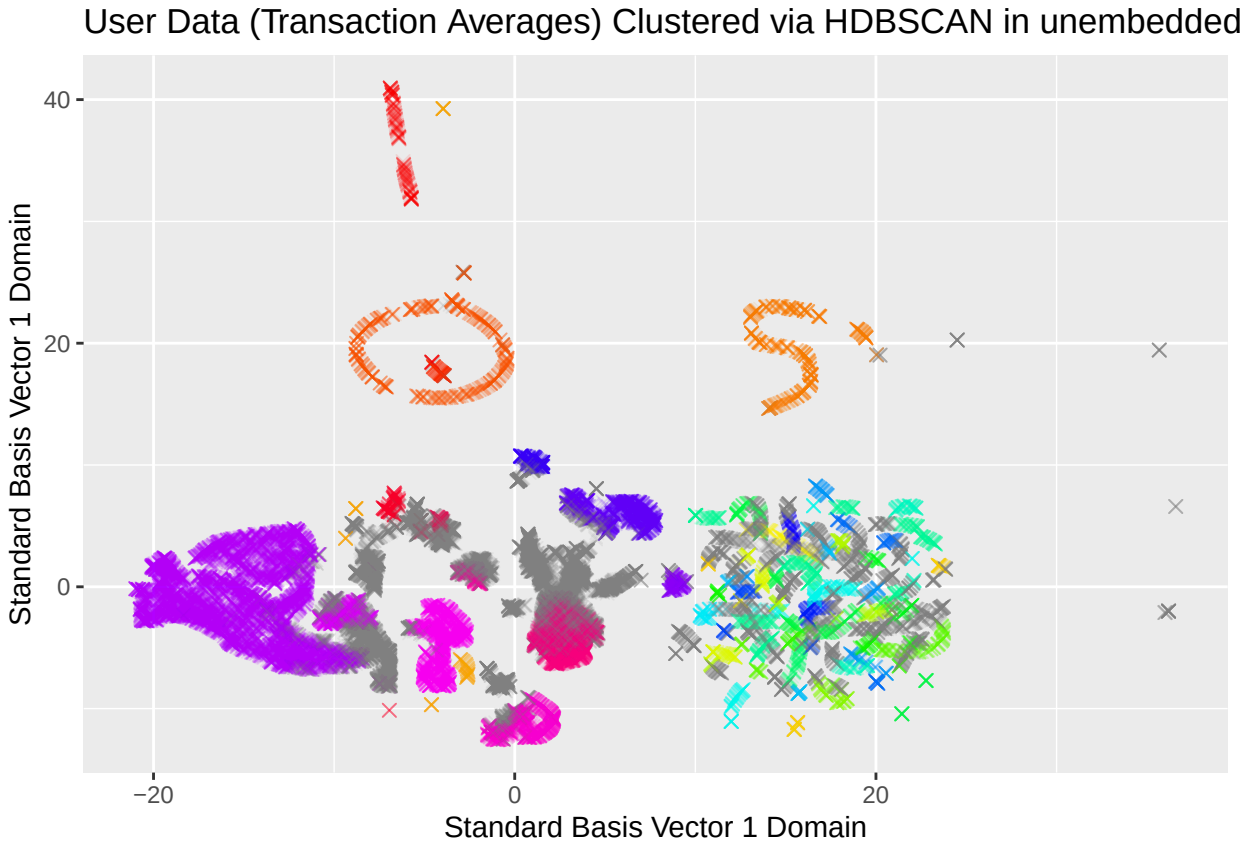


In grey we have outliers.

Now, we try our proposed alternative procedures - rather than clustering on the reduced embedding space (5-D), we cluster directly on the scaled data (17-D) with full dimensions.

One thing to be observant of is that if we see that the same clusters exist across different spaces, informally it could mean that those clusters are significant. That is, if HDBSCAN detected the same cluster in high dimensions as the same on in low dimensions, it might mean the cluster is meaningful.

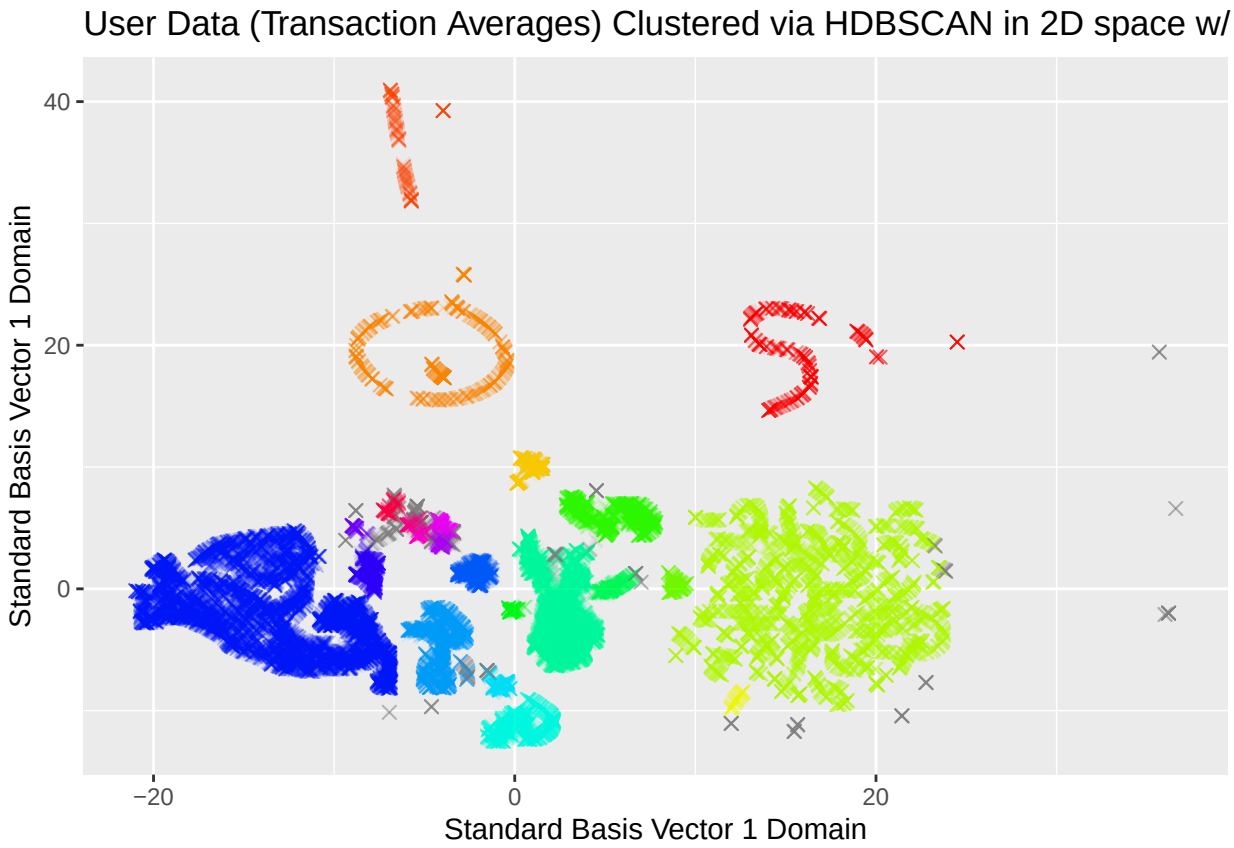
```
## Try clustering on full dim data.
n_colors_raw = max(users.hdb_raw$cluster)
pre_colors_raw <- rainbow(n_colors_raw)
palette2 <- c("#808080", pre_colors_raw) ## set 0 to a light grey as outliers
color_labels_raw <- vector("character", length = length(users.hdb_raw$cluster))
for (i in 0:n_colors_raw) {
  color_labels_raw[users.hdb_raw$cluster == i] = palette2[i+1]
}
cluster_names_raw <- c("Outliers", c(1:n_colors_raw))

ggplot(df_umap, aes(x=X1, y=X2)) + geom_point(alpha = 0.1, size = 2, shape = 4, colour = color_labels_ra
```



Generally, we see that a significan portion of the data in the full space was classified as outliers. Alternatively, we also see that some of the clusters (in the right) are overlapping, which means some structure was lost ("compressed" to these dimensions).

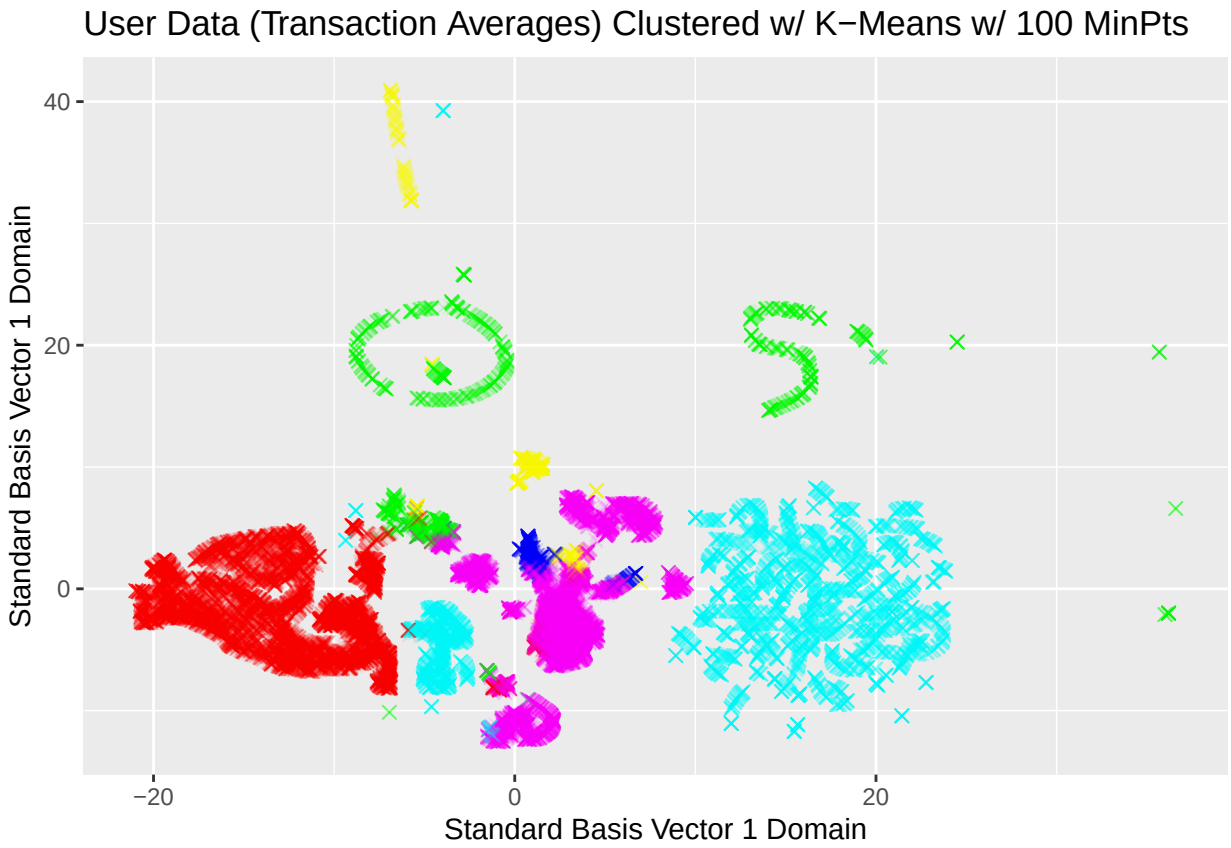
Finally, we demonstrate the results of clustering in 2D. It generates clusters that we expect. However, compared to the other two clusters, we do notice that some of the information is lost, and the algorithm just groups points visually.

```
## cluster in just 2D
n_colors_2d = max(users.hdb$cluster)
pre_colors_2d <- rainbow(n_colors_2d)
palette2d <- c("#808080", pre_colors_2d) ## set 0 to a light grey as outliers
color_labels_2d <- vector("character", length = length(users.hdb$cluster))
for (i in 0:n_colors_2d) {
  color_labels_2d[users.hdb$cluster == i] = palette2d[i+1]
}
cluster_names_2d <- c("Outliers", c(1:n_colors_2d))

ggplot(df_umap, aes(x=X1, y=X2)) + geom_point(alpha = 0.1, size = 2, shape = 4, colour = color_labels_2
```



Finally, we use the clusters generated by k-means in our UMAP visualization.

```
n_colors_km = max(km$cluster)
pre_colors_km <- rainbow(n_colors_km)
palette3 <- c("#808080", pre_colors_km) ## set 0 to a light grey as outliers
color_labels_km <- vector("character", length = length(km$cluster))
for (i in 0:n_colors_km) {
  color_labels_km[km$cluster == i] = palette3[i+1]
}
cluster_names_km <- c("Outliers", c(1:n_colors_km))

ggplot(df_umap, aes(x=X1, y=X2)) + geom_point(alpha = 0.1, size = 2, shape = 4, colour = color_labels_k
```



Surprisingly, it seems like k-means does a fairly decent job of clustering. However, as we expected, some of the finer clusters (compared to the UMAP in 5D or full space) is lost, and mainly resembles clustering in 2D. Also, we see that there are no outlier points (i.e. green points in the far right were outliers in UMAP).

Further Analysis:

More research can be done on evaluating internal clustering metrics and comparing them to PCA. Since we can't quantify how much variance captured by principal components like in PCA, using other metrics to evaluate performance would be beneficial. Alternatively, there is nothing stopping us from further projecting our nonlinearly dimensionality reduced data linearly to 2D via PCA, i.e., Take our 5D UMAP representation, and map it to 2D via PCA, and compute variance captured that way (or generally other combinations)

One could also modify the procedure in which we obtain the clusters and show them. We performed some alternative procedures to when HDBSCAN is run (e.g., running HDBSCAN on different dimensionality reduced data) but adjusting the procedures and tuning both the UMAP and HDBSCAN hyperparameters might be insightful.

One of the burning questions is what do the users clustered have in common (besides being close mathematically)? As clustering only provides the groupings, further analysis needs to be done as to what is common amongst these groups. We can take the clusters of these users and look through their densities, or run survival analysis on them.