CENG 574 Final Paper

Raheem Hashmani

27/01/2021

Contents

1 Introduction

In this report, we analyze the U.S. Census Bureau Income Dataset taken from Data Science Dojo [1].

This dataset was chosen because it has a large number of datapoints, 14 different variables, 2 labels with the possibility of having more than 2 clusters, and because of a personal interest in how various factors might affect the average income of an American citizen.

We first conduct a preliminery analysis where we try to find more information about the raw data, such as the data types of the various variables and the number of unique values they contain. Then, we perform

both Principal Component Analysis (PCA) and multiple versions of Multidimensional scaling (MDS) to try and visualize our dataset on 2 dimensions. Following this, we perform multiple hierarchical clusterings and k-means clusterings to try and find the possible clusters within the dataset. While the labels tell us there are 2, we feel this might be a generalization given the 48,842 datapoints, and strive to see if more good clusters can be formed. Finally, we perform evaluations on the the formed clusters to test their stability and internal and external validity.

Towards the end of the report, we discuss and analyze our results and give a brief conclusion of our findings.

2 Data

In this dataset, the US Census Bureau surveyed 48,842 people and recorded 15 attributes including their ages, working class, years of education, and income, among other things. This dataset in its current form was compiled by Ronny Kohavi [2] and is designed to be used for training and testing various classifiers, with 14 attributes as features and the last attribute, income level, as the class, being either above or below $50,000 USD (50K).

This dataset has been previously used for various statistical analysis purposes, such as clustering aggregation [3], anomaly detection [4], and mining emerging patterns [5].

This dataset contains files for both training and testing datasets. We will perform preliminary analysis on both, perform PCA on the combined dataset, and perform MDS on only 5000 samples, taken as such that their class division matches the combined dataset population's division. This last one is due to the limitations of the computer being used and is subject to change for the final report.

2.1 Preliminary Analysis

```
# All the Libraries we need
library("knitr")
library(plyr)
library(dplyr)
library(ggbiplot)
library(smacof)
library(MASS)
library(cluster)
library(purrr)
library(dendextend)
library(factoextra)
library(cluster)  # For k-medoid
library(kohonen)  # SOM
# For Validation
library(fpc)
library(clValid)
library(mclust)
library(NbClust)
# For nonlinear projection
library(umap)
library(Rtsne)
```

2.1.1 Reading the Data

We first load the datasets. Table 1 and Table 2 show us the a peek inside the train dataset (the test dataset is of the same form).

```
census_train <- read.csv("../Data/census/adult.data.csv", header=TRUE, sep = ",")
census_test <- read.csv("../Data/census/adult.test.csv", header=TRUE, sep = ",")


kable(census_train[0:8,0:7], align = "l", row.names = TRUE,
      caption = "The First 7 Columns.", booktabs = T)
kable(census_train[0:8,8:15], align = "l", row.names = TRUE,
      caption = "The Remaining 8 Columns.", booktabs = T)
```

Table 1: The First 7 Columns.

	age	workclass	fnlwgt	education	education.num	marital.status	occupation
1	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical
2	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial
3	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners
4	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners
5	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty
6	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial
7	49	Private	160187	9th	5	Married-spouse-absent	Other-service
8	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial

Table 2: The Remaining 8 Columns.

	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
1	Not-in-family	White	Male	2174	0	40	United-States	<=50K
2	Husband	White	Male	0	0	13	United-States	<=50K
3	Not-in-family	White	Male	0	0	40	United-States	<=50K
4	Husband	Black	Male	0	0	40	United-States	<=50K
5	Wife	Black	Female	0	0	40	Cuba	<=50K
6	Wife	White	Female	0	0	40	United-States	<=50K
7	Not-in-family	Black	Female	0	0	16	Jamaica	<=50K
8	Husband	White	Male	0	0	45	United-States	>50K

2.1.2 Determining Basic Properties

Now let us find the number of samples for the test and train datasets and the number of features for each sample.

```
samples_train <- nrow(census_train)
features_train <- ncol(census_train)

samples_test <- nrow(census_test)
features_test <- ncol(census_test)

sprintf(paste0("The total number of samples in the training set is %s and the number of ",
    "features is %s."), samples_train, features_train)
sprintf(paste0("The total number of samples in the test set is %s and the number of ",
    "features is %s."), samples_test, features_test)
```

```
## [1] "The total number of samples in the training set is 32561 and the number of features is 15."
## [1] "The total number of samples in the test set is 16281 and the number of features is 15."
```

However, this dataset contains one column for its classes, the "income" column. Thus, in essence, there are **14 features and 2 classes**.

2.1.3 Determining the Number of Unique Values

Training Set

First, we will find the number of unique values in each column for the Training set.

```
census_train %>% summarise_all(n_distinct)
```

```
##   age workclass fnlwgt education education.num marital.status occupation
## 1  73         9  21648        16            16              7         15
##   relationship race sex capital.gain capital.loss hours.per.week native.country
## 1            6    5   2          119           92             94             42
##   income
## 1      2
```

Let us see if our dataset is imbalanced.

```
census_train %>% group_by(income) %>% summarize(count = n())
```

```
## # A tibble: 2 x 2
##   income   count
##   <chr>    <int>
## 1 " <=50K" 24720
## 2 " >50K"   7841
```

As we can see, there are almost **3 times more "<=50K" classes then there are ">50K" classes** (3.15 times, to be precise).

Test Set

Now we will find the number of unique values in each column for the Test set.

```
census_test %>% summarise_all(n_distinct)
```

```
##   age workclass fnlwgt education education.num marital.status occupation
## 1  73         9  12787        16            16              7         15
##   relationship race sex capital.gain capital.loss hours.per.week native.country
## 1            6    5   2          113           82             89             41
##   income
## 1      2
```

Let's see if the test set is imbalanced as well.

```
census_test %>% group_by(income) %>% summarize(count = n())
```

```
## # A tibble: 2 x 2
##   income   count
##   <chr>    <int>
## 1 " <=50K" 12435
## 2 " >50K"   3846
```

Once again, there are almost **3 times more "<=50K" classes then there are ">50K" classes** (3.23 times, to be precise).

2.1.4 Column Labels and Various Properties

Now let us summarize by **combining** both the Train and Test datasets. Table 3 shows the range of all the values and Table 4 summarizes the learned properties so far.

```r
census_total <- rbind(census_train, census_test)

#As a Sanity Check:

samples_total <- nrow(census_total)
features_total <- ncol(census_total)

sprintf(paste0("The number of samples in the total set is %s and the number of ",
               "features is %s."), samples_total, features_total)

census_total %>% summarise_all(n_distinct)

range <- data.frame(min=sapply(census_total,min),max=sapply(census_total,max))
kable(range, align = "l", row.names = TRUE,
      caption = "The Range of Values for all Features", booktabs = T)

# To see the unique classes in the final feature, "income".
vec <- as.vector(census_total['income'])
unique(vec)
```

```
## [1] "The number of samples in the total set is 48842 and the number of features is 15."
##   age workclass fnlwgt education education.num marital.status occupation
## 1  74         9  28523        16            16              7         15
##   relationship race sex capital.gain capital.loss hours.per.week native.country
## 1            6    5   2          123           99             96             42
##   income
## 1      2
```

Table 3: The Range of Values for all Features

	min	max
age	17	90
workclass	?	Without-pay
fnlwgt	12285	1490400
education	10th	Some-college
education.num	1	16
marital.status	Divorced	Widowed
occupation	?	Transport-moving
relationship	Husband	Wife
race	Amer-Indian-Eskimo	White
sex	Female	Male
capital.gain	0	99999
capital.loss	0	4356
hours.per.week	1	99
native.country	?	Yugoslavia
income	<=50K	>50K

```
##   income
## 1  <=50K
## 8   >50K
```

Table 4: Column labels, their type, number of unique values, range (if applicable), and representation type for the combined dataset.

Column Label	Type	Unique Values (Range if Applicable)	Representation
age	Discrete	73 (17 - 90)	Ratio
workclass	Qualitative	9	Nominal
fnlwgt	Discrete	21648 (12285 - 1490400)	Ratio
education	Qualitative	16	Ordinal
education-num	Discrete	16 (1-16)	Ratio
marital-status	Qualitative	7	Nominal
occupation	Qualitative	15	Nominal
relationship	Qualitative	6	Nominal
race	Qualitative	5	Nominal
sex	Qualitative	2	Nominal
capital-gain	Discrete	119 (0 - 99999)	Ratio
capital-loss	Discrete	92 (0 - 4356)	Ratio
hours-per-week	Discrete	94 (1 - 99)	Ratio
native-country	Qualitative	42	Nominal
income	Qualitative (Classes)	2	Ordinal

2.2 Data Preparation

We will perform three different data preparations, one for PCA, one for MDS, and one for clustering.

PCA

First, we will turn our Data Frame into a numerical matrix. This process will apply a label encoding to our qualitative data. Table 5 and Table 6 shows a sample of this matrix.

```
data_matrix_orig <- data.matrix(census_total)

kable(data_matrix_orig[0:8, 0:7], align = "l", caption = "The First 7 Columns.",
    booktabs = T)
kable(data_matrix_orig[0:8, 8:15], align = "l", caption = "The Remaining 8 Columns.",
    booktabs = T)
```

Table 5: The First 7 Columns.

age	workclass	fnlwgt	education	education.num	marital.status	occupation
39	8	77516	10	13	5	2
50	7	83311	10	13	3	5
38	5	215646	12	9	1	7
53	5	234721	2	7	3	7
28	5	338409	10	13	3	11
37	5	284582	13	14	3	5
49	5	160187	7	5	4	9
52	7	209642	12	9	3	5

Table 6: The Remaining 8 Columns.

relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
2	5	2	2174	0	40	40	1
1	5	2	0	0	13	40	1
2	5	2	0	0	40	40	1
1	3	2	0	0	40	40	1
6	3	1	0	0	40	6	1
6	5	1	0	0	40	40	1
2	3	1	0	0	16	24	1
1	5	2	0	0	45	40	2

Finally, we will remove the last column as it is our "classes" column, i.e. income.

```
data_matrix <- data_matrix_orig[, -c(15)]
```

Data Cleaning

```
# Removing Missing Values
df_no_missing <- census_total[!(census_total$workclass == " ?" | census_total$occupation ==
    " ?" | census_total$native.country == " ?"), ]
# Removing duplicate rows
data_frame_orig <- df_no_missing[!duplicated(df_no_missing), ]

# Removing Final Weight and Categorical Education
head(data_frame_orig, 5)

data_frame <- data_frame_orig

# Removing 2 superfluous variables
data_frame <- data_frame[, -c(3, 4)]
head(data_frame, 5)

frame_no_outlier <- data_frame
```

```
##   age         workclass fnlwgt  education education.num      marital.status
## 1  39         State-gov  77516  Bachelors            13       Never-married
## 2  50  Self-emp-not-inc  83311  Bachelors            13  Married-civ-spouse
## 3  38           Private 215646    HS-grad             9            Divorced
## 4  53           Private 234721       11th             7  Married-civ-spouse
## 5  28           Private 338409  Bachelors            13  Married-civ-spouse
##           occupation   relationship   race     sex capital.gain capital.loss
## 1       Adm-clerical  Not-in-family  White    Male         2174            0
## 2    Exec-managerial        Husband  White    Male            0            0
## 3  Handlers-cleaners  Not-in-family  White    Male            0            0
## 4  Handlers-cleaners        Husband  Black    Male            0            0
## 5     Prof-specialty           Wife  Black  Female            0            0
##   hours.per.week native.country income
## 1             40  United-States  <=50K
## 2             13  United-States  <=50K
## 3             40  United-States  <=50K
## 4             40  United-States  <=50K
## 5             40           Cuba  <=50K
##   age         workclass education.num      marital.status         occupation
```

```
## 1  39         State-gov            13       Never-married       Adm-clerical
## 2  50  Self-emp-not-inc            13  Married-civ-spouse    Exec-managerial
## 3  38           Private             9            Divorced  Handlers-cleaners
## 4  53           Private             7  Married-civ-spouse  Handlers-cleaners
## 5  28           Private            13  Married-civ-spouse     Prof-specialty
##     relationship   race     sex capital.gain capital.loss hours.per.week
## 1  Not-in-family  White    Male         2174            0             40
## 2        Husband  White    Male            0            0             13
## 3  Not-in-family  White    Male            0            0             40
## 4        Husband  Black    Male            0            0             40
## 5           Wife  Black  Female            0            0             40
##   native.country income
## 1  United-States  <=50K
## 2  United-States  <=50K
## 3  United-States  <=50K
## 4  United-States  <=50K
## 5           Cuba  <=50K
```

```r
# Interquartile Method for Outlier Removal For Age

# find Q1, Q3, and interquartile range for values in column A
Q1 <- quantile(frame_no_outlier$age, 0.25)
Q3 <- quantile(frame_no_outlier$age, 0.75)
IQR <- IQR(frame_no_outlier$age)

# only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
frame_no_outlier <- subset(frame_no_outlier, frame_no_outlier$age >= (Q1 - 1.5 *
    IQR) & frame_no_outlier$age <= (Q3 + 1.5 * IQR))

############## For Education

# find Q1, Q3, and interquartile range for values in column A
Q1 <- quantile(frame_no_outlier$education.num, 0.25)
Q3 <- quantile(frame_no_outlier$education.num, 0.75)
IQR <- IQR(frame_no_outlier$education.num)

# only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
frame_no_outlier <- subset(frame_no_outlier, frame_no_outlier$education.num >= (Q1 -
    1.5 * IQR) & frame_no_outlier$education.num <= (Q3 + 1.5 * IQR))

############## For capital gain

# find Q1, Q3, and interquartile range for values in column A
Q1 <- quantile(frame_no_outlier$capital.gain, 0.25)
Q3 <- quantile(frame_no_outlier$capital.gain, 0.75)
IQR <- IQR(frame_no_outlier$capital.gain)

# only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
frame_no_outlier <- subset(frame_no_outlier, frame_no_outlier$capital.gain >= (Q1 -
    1.5 * IQR) & frame_no_outlier$capital.gain <= (Q3 + 1.5 * IQR))

############## For capital loss

# find Q1, Q3, and interquartile range for values in column A
```

```r
Q1 <- quantile(frame_no_outlier$capital.loss, 0.25)
Q3 <- quantile(frame_no_outlier$capital.loss, 0.75)
IQR <- IQR(frame_no_outlier$capital.loss)

# only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
frame_no_outlier <- subset(frame_no_outlier, frame_no_outlier$capital.loss >= (Q1 -
    1.5 * IQR) & frame_no_outlier$capital.loss <= (Q3 + 1.5 * IQR))

############## For hours per week

# find Q1, Q3, and interquartile range for values in column A
Q1 <- quantile(frame_no_outlier$hours.per.week, 0.25)
Q3 <- quantile(frame_no_outlier$hours.per.week, 0.75)
IQR <- IQR(frame_no_outlier$hours.per.week)

# only keep rows in dataframe that have values within 1.5*IQR of Q1 and Q3
frame_no_outlier <- subset(frame_no_outlier, frame_no_outlier$hours.per.week >= (Q1 -
    1.5 * IQR) & frame_no_outlier$hours.per.week <= (Q3 + 1.5 * IQR))

# Remove the zeroed columns of income gain/loss

data_full_clean <- frame_no_outlier[, -c(9, 10)]

# view row and column count of new data frame
dim(data_full_clean)
head(data_full_clean, 2)
```

```
## [1] 28568    11
##   age workclass education.num      marital.status         occupation
## 3  38   Private             9           Divorced  Handlers-cleaners
## 4  53   Private             7  Married-civ-spouse  Handlers-cleaners
##     relationship   race   sex hours.per.week native.country income
## 3  Not-in-family  White  Male             40  United-States  <=50K
## 4        Husband  Black  Male             40  United-States  <=50K
```

```r
#### SAMPLING

clean_sample <- data_full_clean %>% group_by(income) %>% sample_n(2500)

clean_sample %>% group_by(income) %>% summarize(count = n())  # Checking if above was done correctly.
```

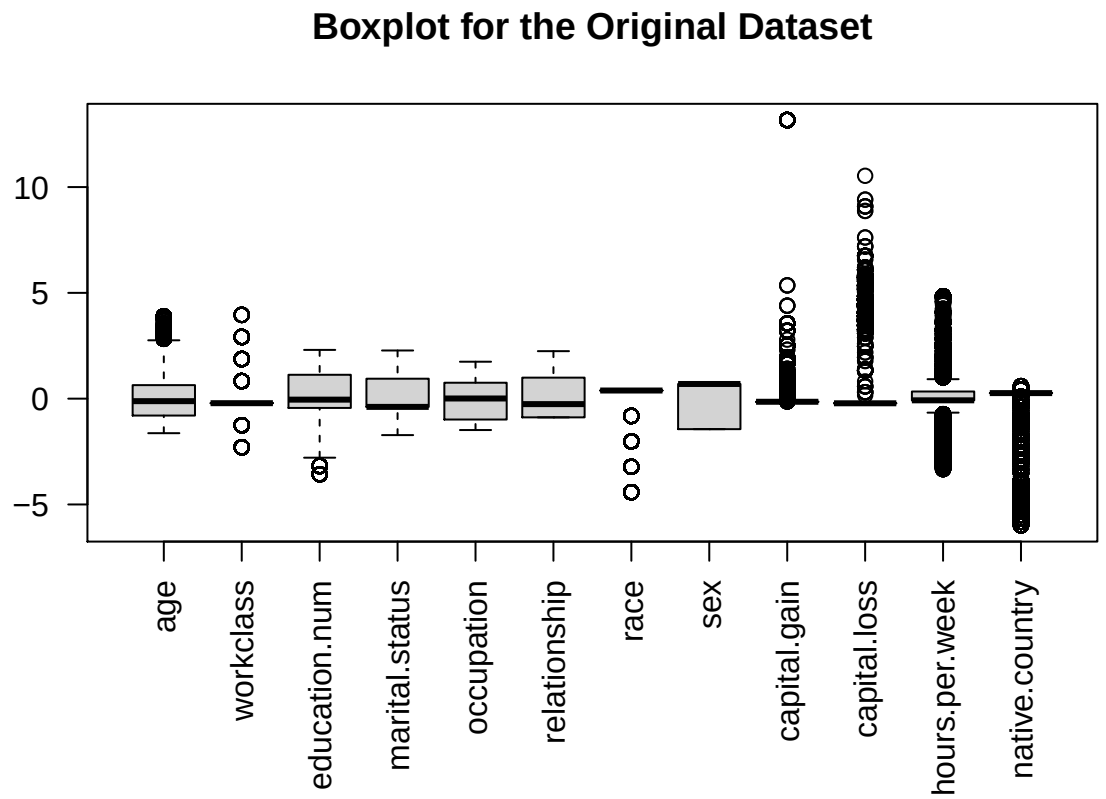
```
## # A tibble: 2 x 2
##   income   count
##   <chr>    <int>
## 1 " <=50K"  2500
## 2 " >50K"   2500
```

```r
# To save the selected sample dataset, for reproducibility.  write.csv(temp
# ,'5000framev1.csv', row.names = FALSE, col.names = TRUE)

# To read the sampled data

clean_sample <- read.csv("../Data/5000framev1.csv")

head(clean_sample, 5)
```

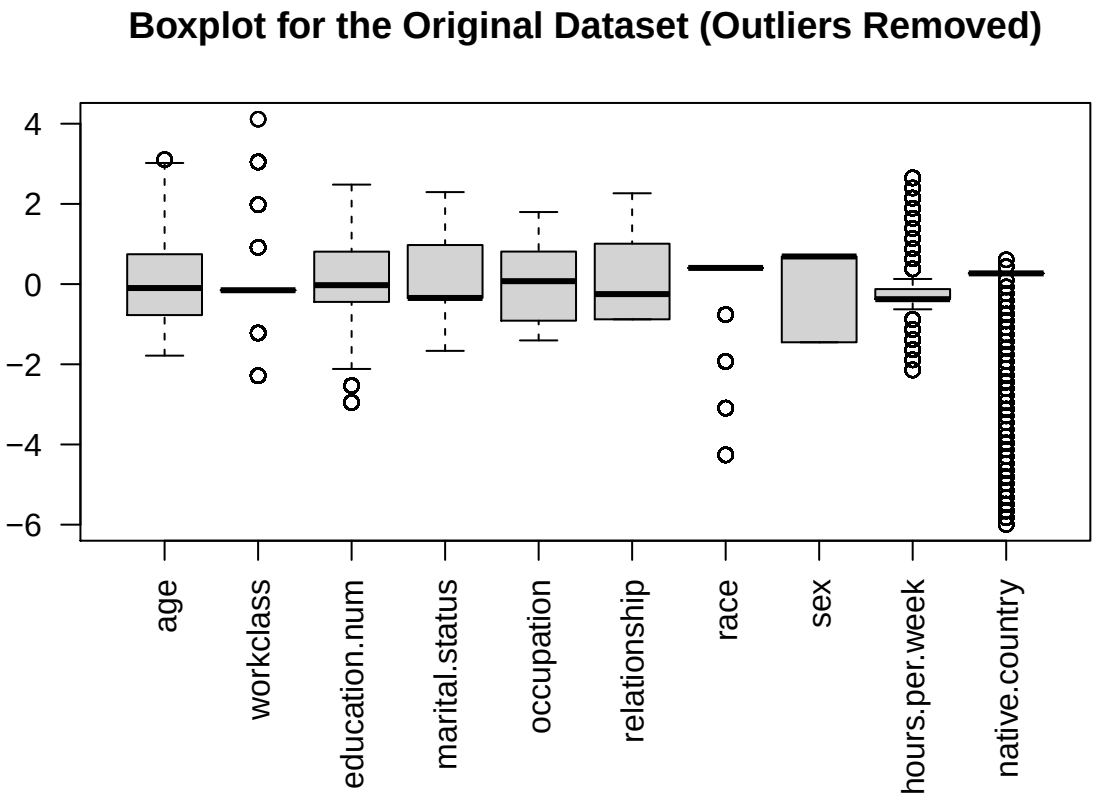
```r
sample <- scale(data.matrix(clean_sample))
```

```
##   age         workclass education.num      marital.status         occupation
## 1  30           Private             9       Never-married    Exec-managerial
## 2  20           Private             9  Married-civ-spouse  Handlers-cleaners
## 3  31  Self-emp-not-inc             9       Never-married       Craft-repair
## 4  38      Self-emp-inc            11            Divorced    Farming-fishing
## 5  31           Private            14  Married-civ-spouse     Prof-specialty
##     relationship   race     sex hours.per.week native.country income
## 1  Not-in-family  White    Male             40    Puerto-Rico  <=50K
## 2        Husband  White    Male             50  United-States  <=50K
## 3      Unmarried  White    Male             50  United-States  <=50K
## 4  Not-in-family  White    Male             50  United-States  <=50K
## 5           Wife  White  Female             50  United-States  <=50K
```

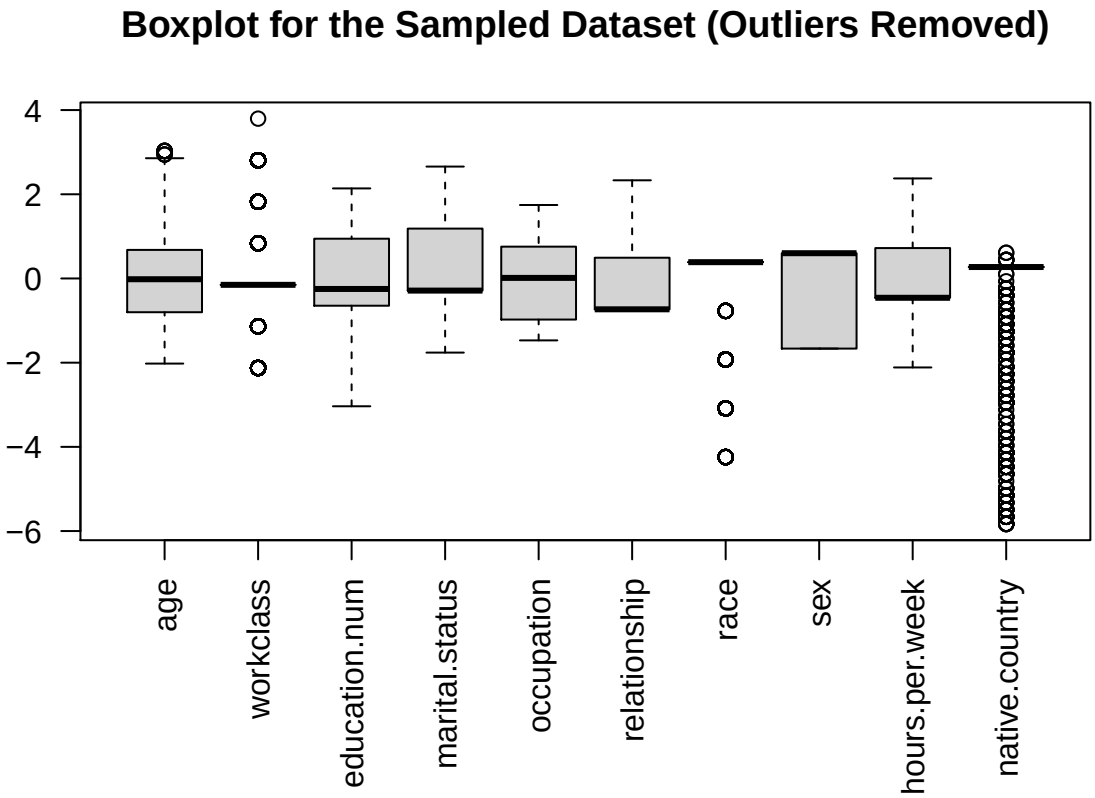
```r
# SCALED BOXPLOT
data_frame_box <- data_frame[, -c(13)]
data_full_clean_box <- data_full_clean[, -c(11)]
sample_box <- sample[, -c(11)]

par(mar = c(7, 4.1, 4.1, 2.1))

boxplot(scale(data.matrix(data_frame_box)), main = "Boxplot for the Original Dataset",
    las = 2)
```



```r
boxplot(scale(data.matrix(data_full_clean_box)), las = 2, main = "Boxplot for the Original Dataset (Outl
```




```r
boxplot(sample_box, las = 2, main = "Boxplot for the Sampled Dataset (Outliers Removed)")
```




```
par(mar = c(5.1, 4.1, 4.1, 2.1))
```

MDS

Because many MDS algorithms are either $O(n^3)$ or $O(n^2)$, we will sample them according to their proportions for the "income" class. Using the combined dataset, we will remove rows with missing data (since we have an ample number of rows, we can afford to delete them) and duplicate data (certain MDS algorithms, such as Sammon's Non-Linear Mapping, do not work if there are 0s present in the distance matrix, so duplicate rows must be removed) and sample 5000 datapoints, making sure that the ratio of "<=50K" to ">50K" is 3.2:1, as averaged in Section 2.1.3. Finally, we will convert it into a data matrix and remove the "income" column, similar to what we did for PCA.

```
####### ALREADY DONE ABOVE IN DATA CLEANING ########



# # Removing rows with missing values df_no_missing <-
# census_total[!(census_total$workclass==' ?' | census_total$occupation==' ?' |
# census_total$native.country ==' ?'),] # Removing duplicate rows df_no_dup =
# df_no_missing[!duplicated(df_no_missing), ] # Choosing Equal Number of
# datapoints from both classes.  temp_df <- df_no_dup %>% group_by(income) %>%
# sample_n(100) # Removing the excess >50K class datapoints in order to make the
# sample # representative of the original dataset.  sample_frame
# <-temp_df[1:(nrow(temp_df) - 70),] data_matrix_orig_mds <-
# data.matrix(sample_frame) data_matrix_mds <- data_matrix_orig_mds[,-c(15)]
# sample_number <- nrow(data_matrix_mds) print(paste0('The number of samples
# selected is: ', sample_number))
```

```r
# To Remove Class Labels

data_matrix_orig_mds <- sample

data_matrix_mds <- data_matrix_orig_mds[, -c(11)]
sample_number <- nrow(data_matrix_mds)
print(paste0("The number of samples selected is: ", sample_number))
```

```
## [1] "The number of samples selected is: 5000"
```

Clustering

For clustering, the data preparation is similar to MDS's data preparation. In addition, because some of the clustering algorithms use pseudorandom numbers, we will set a seed for reproducibility. Finally, we scale the dataset as an additional step in order to get a more accurate clustering.

```r
# Same pre-processing

data_matrix_clus <- data_matrix_orig_mds[, -c(11)]

sample_number <- nrow(data_matrix_clus)
print(paste0("The number of samples selected is: ", sample_number))

# Setting Seed to allow reproducible results
set.seed(786)  # Setting Seed

data_matrix_scaled <- data_matrix_clus  # It's already scaled.
```

```
## [1] "The number of samples selected is: 5000"
```

3 Methods

PCA and MDS

We first apply Principal Component Analysis (PCA) and 4 different Multidimensional Scaling (MDS) algorithms to better visualize our dataset. For PCA, two online tutorials were used to get experience with properly coding PCA in R [6], [7]. In addition, Scree Plots will be plotted in order to to get a better understanding of the PCA. For MDS, Classical multidimensional scaling of a data matrix, Sammon's Non-Linear Mapping, Kruskal's Non-metric Multidimensional Scaling, and Symmetric Smacof was used, with another online tutorial being used as a general practical guide [8].

For the criterion function, Sammon's Non-Linear Mapping uses:

$$J_{\text{samm}} = \frac{1}{\sum_{i<j} \delta_{ij}} \sum_{i<j} \frac{(d_{ij} - \delta_{ij})^2}{\delta_{ij}}$$

which provides a good compromise between emphasizing large errors and large fractional errors. In comparison,

Kruskal's Non-metric Multidimensional Scaling criterion function is:

$$J_{\text{kruskal}} = \sqrt{\frac{\sum_{i<j}(d_{ij} - \delta_{ij})^2}{\sum_{i<j} \delta_{ij}^2}}$$

which emphasizes larger errors.

Clustering

For clustering, hierarchical and k-means clustering is performed.

For hierarchical clustering, we perform Agglomerative Nesting (AGNES) hierarchical clustering with 6 different linkages and Divisive Analysis (DIANA) clustering. Two online tutorials were used to get a better understanding of the functions [9] [10].

For AGNES, we use the agnes() function over the hclust() function because it also outputs an agglomerative coefficient, which measures how strong the clustering structure is, with values closer to 1 being better. In addition, it uses fewer shortcuts when updating the distance matrix, such as calculating the distance between observations and clusters as averages of the distances between all the observations in 2 clusters, rather than just using two observations that were recently merged (as in hclust) [11].

In addition, we use 6 different linkages: Group Average, single-link (MIN), complete-link (MAX), Ward's method, weighted (Unweighted Pair Group Method with Arithmetic Mean, UPGMA), and Generalized Average (Flexible UPGMA). All of these behave slightly differently and will not only allow us to see which method helps us better cluster our dataset, but will also help us learn more about our dataset by understanding why a particular method worked better. We will compare all of their agglomerative coefficients and choose the best 2 as a basis for our k-means clustering, to use as initialization. We will also use DIANA as well, which instead has a divisive coefficient, and consider it as a basis for k-means clustering too. In all 7 of these cases, we will tentatively make the cutoff at 5 clusters, based on our PCA clustering suggestion, and 2 clusters, in order to compare with the original dataset's 2 classes. Finally, dendrogram will be drawn for all the cases in order to get a batter visualization of the hierarchical clustering.

For k-means clustering, we first try to estimate the number of clusters, k, to select and additionally use $k = 2$ (original dataset clustering) and $k = 5$ (estimated clustering from PCA). An online tutorial was used to get a better understanding of how to implement related functions [12] and the documentation was used to understand the various options [13].

To estimate the number of clusters, we use two methods. First, the elbow method, which plots the Total Within Sum of Squares (WSS) against different number of clusters to see if an "elbow" forms, where the error drastically falls. And second, we use the silhouette method, which plots the average silhouette width vs. different number of clusters, with a silhouette being the value of how well each object lies within its cluster.

For each of the estimated clusters and for $k = 2$ and $k = 5$, we will randomly initialize k-means clustering 100 times each and find the clustering with the best results. The multiple random initialization is automatically done using the kmeans() function. Then, using the results of the hierarchical clustering, we will find 6 more k-means clusterings ($k = 2$ and $k = 5$ for the top 2 linkages and DIANA) using the hierarchical clustering centers as initialization points. Finally, we perform another PCA and project all the clusterings onto the first 2 principal components.

Validation of Clusterings

To validate our clusterings, we use various stability, internal, and external validation tests. For stability and internal, an online tutorial [14] was used to better learn the functions. Similarly, the function for external validation was learned from another online tutorial [15].

For stability tests, we perform the nonparametric bootstrap, Average Proportion of Non-overlap (APN), Average Distance (AD), Average Distance between Means (ADM), and Figure Of Merit (FOM) stability tests.

The nonparametric boostrap method uses the model we have already fitted to our dataset (the cluster model) to generate more samples with the same distribution as the real data, with the "nonparametric" version making no assumption about how the data is distributed. It then uses this simulated data to calculate standard errors, construct confidence intervals, and perform hypothesis testing in what is known as "bootstrapping" [16]. Using the clusterboot() function, we receive a clusterwise Jaccard bootstrap mean, with larger values being better, the number of dissolved clusters, with smaller values being better, and the number of recovered clusters, with values close to the number of bootstrap repetitions chosen being better.

APN values are within the interval $[0, 1]$, with values closer to 0 indicating consistent clustering results. It

measures the average proportion of datapoints that are not placed in the same cluster. AD, ADM, and FOM all have ranges from $[0, \infty]$, with smaller values being preferred. AD measures the average distance between datapoints within the same cluster. ADM measures the average distance between cluster centers for datapoints within the same cluster, with our implementation using Euclidean distance as the distance measure. Finally, FOM measures the average intra-cluster variance of datapoints in a deleted row, where the clustering is based on the remaining datapoints [17].

For internal validation, we perform the Connectivity, Silhouette Width, and Dunn Index validation tests. Connectivity measures the extent to which datapoints are placed within the same cluster as their nearest neighbor. Its values range from $[0, \infty]$, with smaller values being preferred. The Silhouette Width is a non-inear combination measurement of the compactness and separation of the clusters. It averages each datapoint's Silhouette value, which measures the degree of confidence of that datapoints assignment to its particular cluster, and has values within the range of $[-1, 1]$, with values closer to 1 being better. Finally, the Dunn Index is also a non-linear combination measurement of the compactness and separation of the clusters, but it takes the ratio of the smallest distance between datapoints in different clusters to the largest intra-cluster distance. It has values in the range of $[0, \infty]$, with larger values being better.

Finally, for external validation, we choose size 2 clusters, k = 2, to match the 2 classes in the original dataset and compute the Rand Index. The Rand Index measures the agreement between two clusterings which, in our case, is the original cluster labels and our k=2 clusters. We use a corrected version which measures the same similarity, adjusted for chance, with a range from -1 (no agreement between the clusters) to +1 (a perfect agreement) [15].

4 Code and Results

4.1 Principal Component Analysis

First, let us calculate the covariance matrix. Table 7 shows a subsample of this matrix.

```
cov_matrix<- cov(data_matrix)

kable(cov_matrix[0:5,0:5], align = "l", row.names = TRUE,
      caption = "Covariance Matrix for the First 5 Features",
      booktabs = T)
```

Table 7: Covariance Matrix for the First 5 Features

	age	workclass	fnlwgt	education	education.num
age	1.879781e+02	0.3518480	-1.109486e+05	-0.7998851	1.090628e+00
workclass	3.518480e-01	2.1439803	-2.558475e+03	0.0975025	2.105023e-01
fnlwgt	-1.109486e+05	-2558.4754667	1.115221e+10	-9234.9738145	-1.052372e+04
education	-7.998851e-01	0.0975025	-9.234974e+03	15.0116916	3.582727e+00
education.num	1.090628e+00	0.2105023	-1.052372e+04	3.5827269	6.609901e+00

Now, we will find the Eigenvectors and corresponding Eigenvalues of the Covariance Matrix

```
eigen_info <- eigen(cov_matrix)
eigen_info
```

```
## eigen() decomposition
## $values
##  [1] 1.115221e+10 5.553198e+07 1.622496e+05 1.885317e+02 1.494532e+02
##  [6] 6.042144e+01 1.800758e+01 1.604838e+01 5.024804e+00 2.436235e+00
## [11] 1.960878e+00 1.900012e+00 6.843967e-01 1.387255e-01
##
```

```
## $vectors
##                [,1]          [,2]          [,3]          [,4]          [,5]
##  [1,] -9.948765e-06 -1.415560e-04  2.012919e-03  9.585703e-01  2.817385e-01
##  [2,] -2.294235e-07 -7.070009e-06  4.346951e-05  5.161265e-03 -1.616087e-02
##  [3,]  1.000000e+00 -2.628254e-04  1.713208e-05  9.901998e-06  1.263682e-06
##  [4,] -8.281043e-07 -1.499574e-05  1.775991e-04 -1.541676e-03 -2.138351e-02
##  [5,] -9.437011e-07 -4.312401e-05  5.412637e-04  8.821056e-03 -2.729687e-02
##  [6,]  4.261951e-07  8.872918e-06 -1.316461e-04 -3.240466e-02  1.187519e-02
##  [7,]  3.445177e-08 -1.371813e-05  1.886357e-04  1.310114e-03 -3.197655e-02
##  [8,]  1.379549e-07  1.214118e-05 -2.346921e-04 -3.621778e-02  2.088501e-02
##  [9,] -2.167956e-07 -1.303060e-06  3.959480e-05  2.050653e-03 -2.058228e-03
## [10,]  1.236515e-07 -2.981349e-06  5.508519e-05  4.788551e-03 -7.402616e-03
## [11,] -2.628525e-04 -9.999985e-01  1.705629e-03 -1.793014e-04  9.527385e-05
## [12,] -1.666005e-05  1.706198e-03  9.999948e-01 -2.439387e-03  1.147301e-03
## [13,] -1.586448e-06 -1.365232e-04  1.756145e-03  2.803799e-01 -9.578578e-01
## [14,] -3.584156e-06  2.084120e-06  6.131233e-05 -5.155506e-03 -2.869874e-03
##                [,6]          [,7]          [,8]          [,9]         [,10]
##  [1,] -5.781995e-03  6.351707e-03 -6.176035e-03  6.057595e-03 -3.773081e-02
##  [2,]  2.048509e-03  9.571750e-02 -8.280569e-03  1.130561e-02  2.435225e-01
##  [3,] -3.690121e-06  6.349305e-08 -9.405147e-07 -5.024300e-07 -1.259749e-07
##  [4,] -4.135118e-02 -5.110236e-02 -9.412988e-01  3.299962e-01  7.376251e-03
##  [5,] -1.982587e-02  7.616137e-02 -3.319497e-01 -9.377268e-01 -5.611471e-02
##  [6,]  4.620152e-03 -3.318720e-03  1.428041e-02  2.389129e-02 -4.654501e-01
##  [7,]  1.056518e-02  9.900983e-01 -2.155053e-02  9.006761e-02 -4.497341e-02
##  [8,]  1.209575e-03 -2.671232e-02  4.415957e-03  4.959760e-02 -8.268851e-01
##  [9,] -1.512110e-02  1.691824e-03 -4.525643e-04 -7.181309e-03  9.192641e-02
## [10,]  5.564924e-04  6.910168e-03  4.789204e-03  5.623245e-05  1.542704e-01
## [11,] -6.799821e-08 -1.362602e-05  2.640871e-05  3.219904e-05 -3.266120e-06
## [12,]  8.592948e-05 -1.830099e-04  3.115273e-04  3.999881e-04 -9.857881e-05
## [13,]  2.504507e-03 -3.454317e-02  2.961756e-02  1.931493e-02 -3.744994e-02
## [14,] -9.987441e-01  1.108111e-02  4.550826e-02  6.220442e-03 -3.503640e-03
##               [,11]         [,12]         [,13]         [,14]
##  [1,] -3.869933e-03 -1.279300e-02 -1.867559e-03  2.818500e-03
##  [2,]  5.369638e-01 -8.008115e-01  3.700811e-02 -6.915319e-03
##  [3,]  2.896574e-07 -9.400935e-08 -1.468950e-07 -1.163793e-07
##  [4,] -1.471167e-02  9.221368e-04 -7.680630e-04  2.399107e-03
##  [5,]  7.680067e-03 -1.203937e-02  2.347475e-03  1.013341e-02
##  [6,] -6.602481e-01 -5.866069e-01 -3.567190e-02  7.087235e-03
##  [7,] -4.109423e-02  7.909446e-02 -4.564761e-03 -2.198032e-03
##  [8,]  5.164244e-01  8.647413e-02 -7.878218e-02  1.758980e-01
##  [9,]  2.625441e-03 -1.607563e-02 -9.954313e-01 -1.044052e-02
## [10,] -8.387619e-02 -1.668387e-02  3.997594e-03  9.842369e-01
## [11,] -1.392690e-06  2.488080e-06  2.105840e-07 -1.135669e-06
## [12,]  3.831349e-05 -7.096580e-06  1.902591e-05 -1.724827e-05
## [13,] -4.995150e-03  2.215502e-03 -1.167428e-03 -2.985985e-03
## [14,] -1.382147e-03 -2.900208e-03  1.483345e-02  5.905655e-04
```

We can see that the Eigenvalues are decreasing and that the first 3 Eigenvalues are considerably larger than the rest. We can predict that the PCA will give us decent results and some structure to visualize.

Finally, we apply PCA,

```
pca_data <- prcomp(data_matrix, center = TRUE, scale. = TRUE, retx = TRUE)
```

show its summary,

```
summary(pca_data)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.4551 1.1892 1.11738 1.06042 1.04282 1.01388 0.97489
## Proportion of Variance 0.1512 0.1010 0.08918 0.08032 0.07768 0.07343 0.06789
## Cumulative Proportion  0.1512 0.2522 0.34143 0.42175 0.49943 0.57285 0.64074
##                           PC8     PC9   PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.96297 0.92427 0.9196 0.86081 0.82801 0.76782 0.62150
## Proportion of Variance 0.06624 0.06102 0.0604 0.05293 0.04897 0.04211 0.02759
## Cumulative Proportion  0.70698 0.76800 0.8284 0.88133 0.93030 0.97241 1.00000
```

and give general output of the PCA.

```
pca_data
```

```
## Standard deviations (1, .., p=14):
##  [1] 1.4551272 1.1891553 1.1173838 1.0604199 1.0428245 1.0138824 0.9748929
##  [8] 0.9629711 0.9242705 0.9195780 0.8608134 0.8280071 0.7678178 0.6215048
##
## Rotation (n x k) = (14 x 14):
##                         PC1         PC2         PC3          PC4         PC5
## age            -0.28736932  0.13471452 -0.36195332  0.178667602 -0.43501720
## workclass      -0.21740371 -0.10045420  0.53437843 -0.171143153 -0.30101999
## fnlwgt          0.04459721  0.10757261  0.19411770  0.183993403  0.48930857
## education      -0.09937714 -0.60193440 -0.15322572  0.105794932  0.23241150
## education.num  -0.21354042 -0.61392412 -0.03223075  0.139751109  0.11587047
## marital.status  0.32328113 -0.02980465  0.25535330 -0.093052130  0.35585172
## occupation     -0.16974776 -0.12561392  0.59976541 -0.115744981 -0.23454498
## relationship    0.52084503 -0.23213217  0.05323358 -0.006700423 -0.23428155
## race           -0.15771576 -0.01654796 -0.14055230 -0.629233128  0.07206385
## sex            -0.45651839  0.29056554  0.06663493 -0.024848196  0.37583647
## capital.gain   -0.14166822 -0.15878346 -0.02163546  0.228119414 -0.13659902
## capital.loss   -0.10872484 -0.06830459 -0.06683563  0.042589422  0.07896805
## hours.per.week -0.37629635 -0.06388623  0.09991970  0.095991741  0.06302004
## native.country -0.03622851 -0.17817800 -0.23315015 -0.626492138  0.03795243
##                         PC6         PC7         PC8         PC9         PC10
## age            -0.05817413  0.15442830 -0.04829907 -0.35550552  0.173879272
## workclass      -0.02872956  0.07764689 -0.10851178  0.02662147 -0.094570450
## fnlwgt          0.09683157  0.74908231 -0.24708585 -0.09730343  0.127903840
## education      -0.01939552 -0.12597369 -0.26191808 -0.17401084 -0.050262824
## education.num  -0.02573246 -0.03300521  0.01835546 -0.14552259  0.002592872
## marital.status  0.04180255 -0.30713908  0.40427823 -0.17977118  0.064558291
## occupation     -0.07814078  0.03530868  0.02364464 -0.26314551  0.238803170
## relationship   -0.02216139  0.19149900 -0.03109455  0.20355952 -0.107638211
## race            0.09606772  0.23585404  0.06765942 -0.21500598 -0.634015749
## sex             0.05191752 -0.23749522  0.06977161 -0.08342584  0.092120401
## capital.gain    0.60245637  0.23902523  0.63844828  0.05449773 -0.005369932
## capital.loss   -0.77055432  0.27239944  0.51631673  0.08735577 -0.013301860
## hours.per.week  0.01661270 -0.02820613 -0.08213339  0.74228329 -0.156033341
## native.country  0.09012812  0.12500261  0.02334961  0.23283019  0.658889166
##                        PC11          PC12        PC13          PC14
## age            -0.12682866  0.5340562831  0.09632467 -0.2243551158
## workclass      -0.68879015 -0.0330779373 -0.18263150  0.0044119079
## fnlwgt         -0.04187886  0.1264027174 -0.01835856  0.0057568596
```

```
## education      -0.25093725 -0.1025952815  0.58981688  0.0090195640
## education.num   0.20833889  0.1072203055 -0.67530814 -0.0963750849
## marital.status -0.19192697  0.5977230331  0.05734662 -0.0264748713
## occupation      0.54844444  0.0005845398  0.30381578  0.0003124419
## relationship    0.04372538 -0.0077625044  0.06372748 -0.7210290353
## race            0.13956126  0.1146464022  0.05045158 -0.0341386960
## sex            -0.05331689 -0.2485397679  0.00245098 -0.6456141944
## capital.gain   -0.06827579 -0.1725126066  0.13181746  0.0324386433
## capital.loss   -0.08308544 -0.1183069664  0.08642914  0.0167486082
## hours.per.week  0.14925198  0.4467778450  0.16054718 -0.0184223492
## native.country -0.07519332  0.0085607888 -0.01669589 -0.0018646755
```

Let us generate a Scree Plot to better visualize the proportion of variance for each principal component. Figure 1 shows this scree plot.

```
plot(pca_data$sdev^2, xlab = "Principal Component", ylab = "Proportion of Variance",
    type = "b", las = 1, main = "Scree Plot")
```

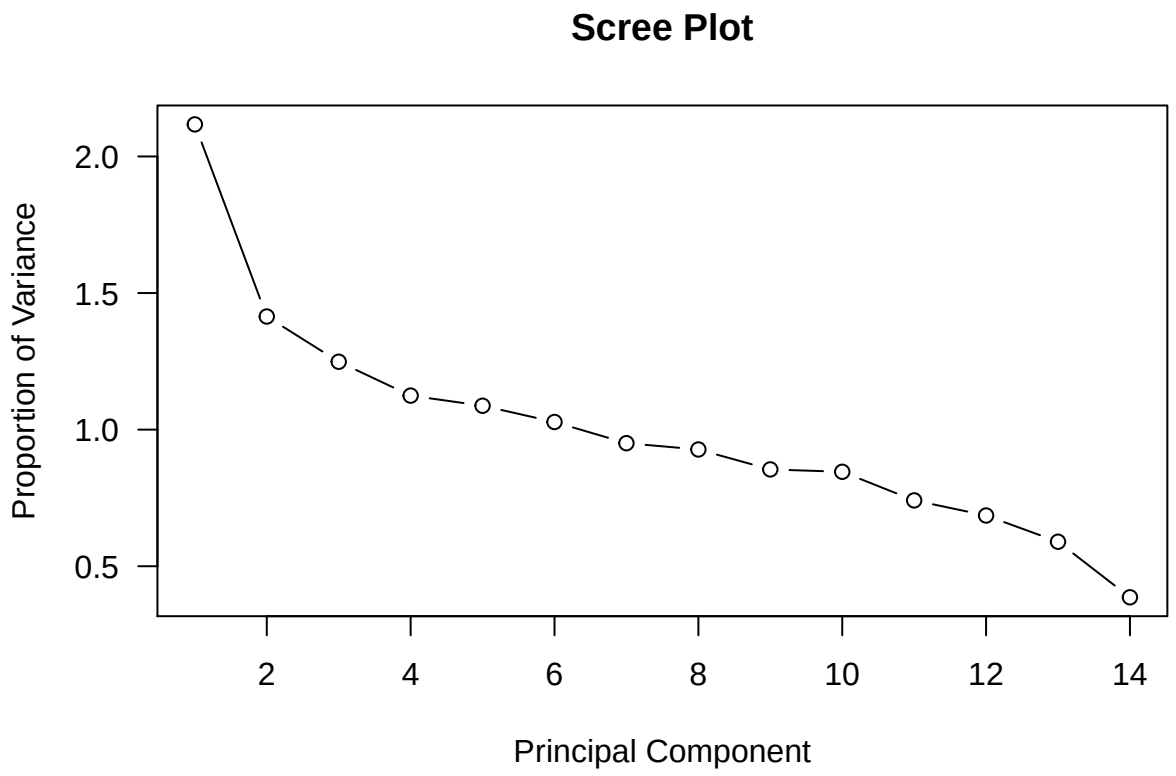


Figure 1: Scree Plot of the Principal Components.

While that gives us a sense of how much greater the first principal component's proportion of variance is, it might be helpful to see it normalized to get a better perspective. Figure 2 shows the normalized scree plot.

```
pca_data_sdev2 <- pca_data$sdev^2  #Compute Variance
pca_data_normalized <- pca_data_sdev2/sum(pca_data_sdev2)  #Compute Proportion of Variance
plot(pca_data_normalized, xlab = "Principal Component", ylab = "Proportion of Variance",
    ylim = c(0, 1), type = "b", las = 1, main = "Normalized Scree Plot")
```

```r
axis(2, at = c(0.1, 0.3, 0.5, 0.7, 0.9), las = 1)  # To add additional ticks
```

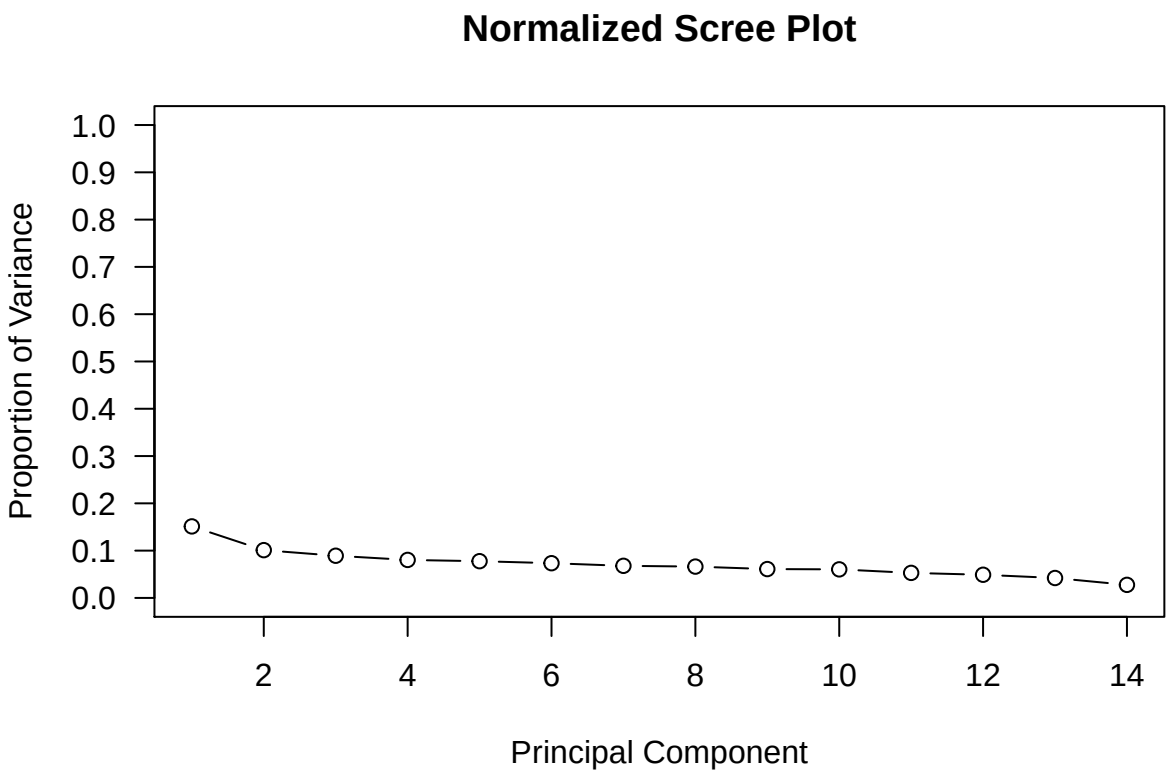


Figure 2: Scree Plot of the Principal Components, normalized.

Finally, we will use a modified ggbiplot to plot the first 2 principal components, shown in Figure 3. The modified function components are listed in the appendix.

```r
#I created a function, my_ggbiplot, to edit the color scheme of the ggbiplot:
source("../my_functions.R")

pca_plot<-my_ggbiplot(pca_data, var.axes=TRUE, groups = data_matrix_orig[,c(15)])

# To make the arrows more visible

# pca_plot$layers
pca_plot$layers <- c(pca_plot$layers, pca_plot$layers[[1]])
# pca_plot$layers
pca_plot +
  labs(color="Income", # Labeling the Legend
       title = "Plotting Datapoints on the First 2 Principal Components") +
  theme(plot.title = element_text(hjust = 0.5),
                    plot.caption=element_text(hjust = 0))
```

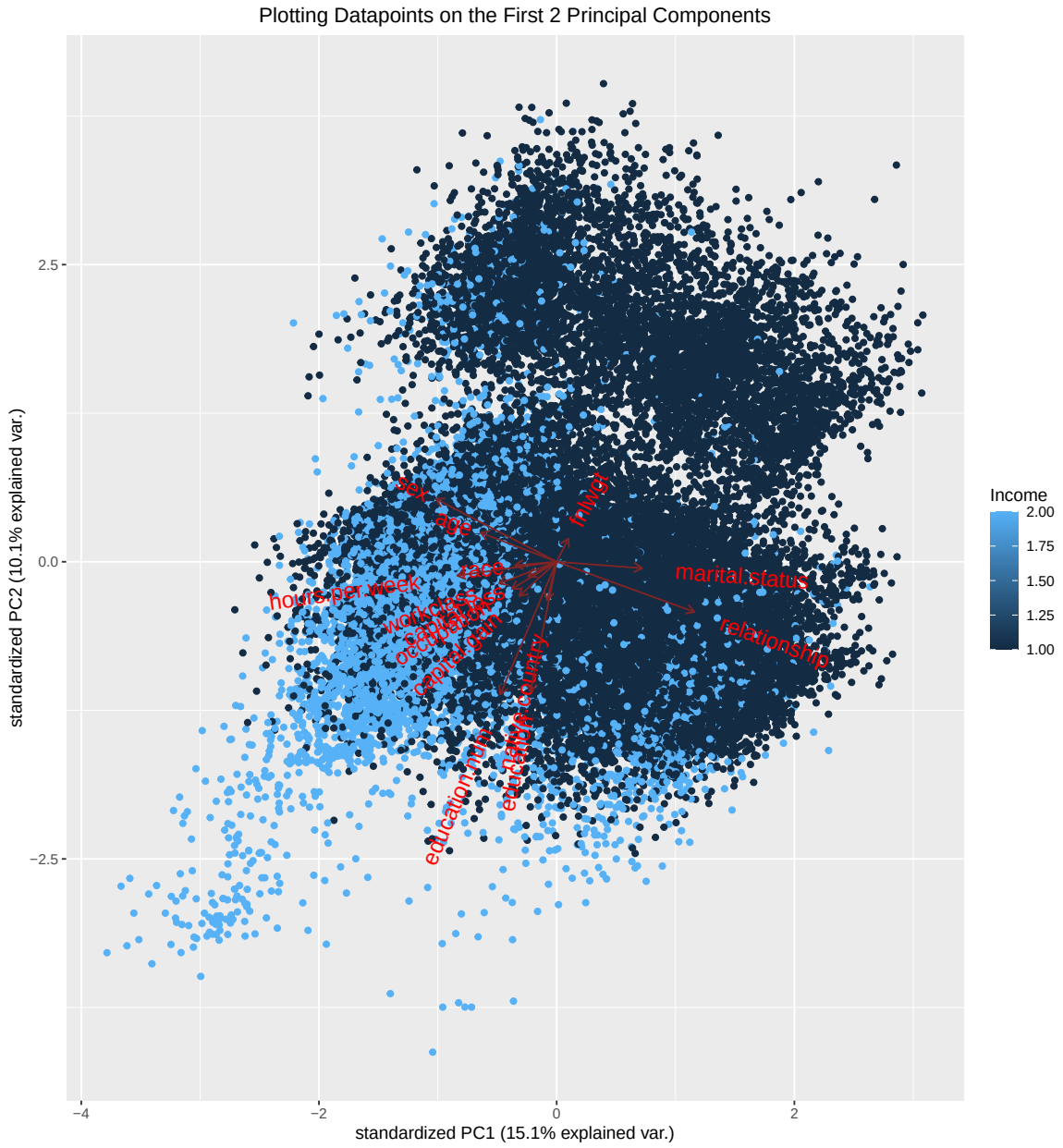


Figure 3: Plotting all the datapoints on the first 2 Principal Components.

Before we continue, let us redo the above steps for our smaller, sampled dataset which is used for MDS and clustering, to get a better perspective on it. The PCA summary is shown below, the normalized Scree Plot is shown in Figure 4, and plot of the first 2 principal components is shown in Figure 5.

```
pca_data <- prcomp(data_matrix_mds, center = TRUE, scale. = TRUE, retx = TRUE)

summary(pca_data)

# Plotting Normalized Scree Plot
pca_data_sdev2 <- pca_data$sdev ^ 2 #Compute Variance
pca_data_normalized <- pca_data_sdev2/sum(pca_data_sdev2) #Compute Proportion of Variance
plot(pca_data_normalized,xlab="Principal Component", ylab="Proportion of Variance",
```

```
ylim=c(0,1), type='b', las=1, main = "Normalized Scree Plot for the Sampled Data")
axis(2,at=c(0.1, 0.3, 0.5, 0.7, 0.9), las=1) # To add additional ticks
```

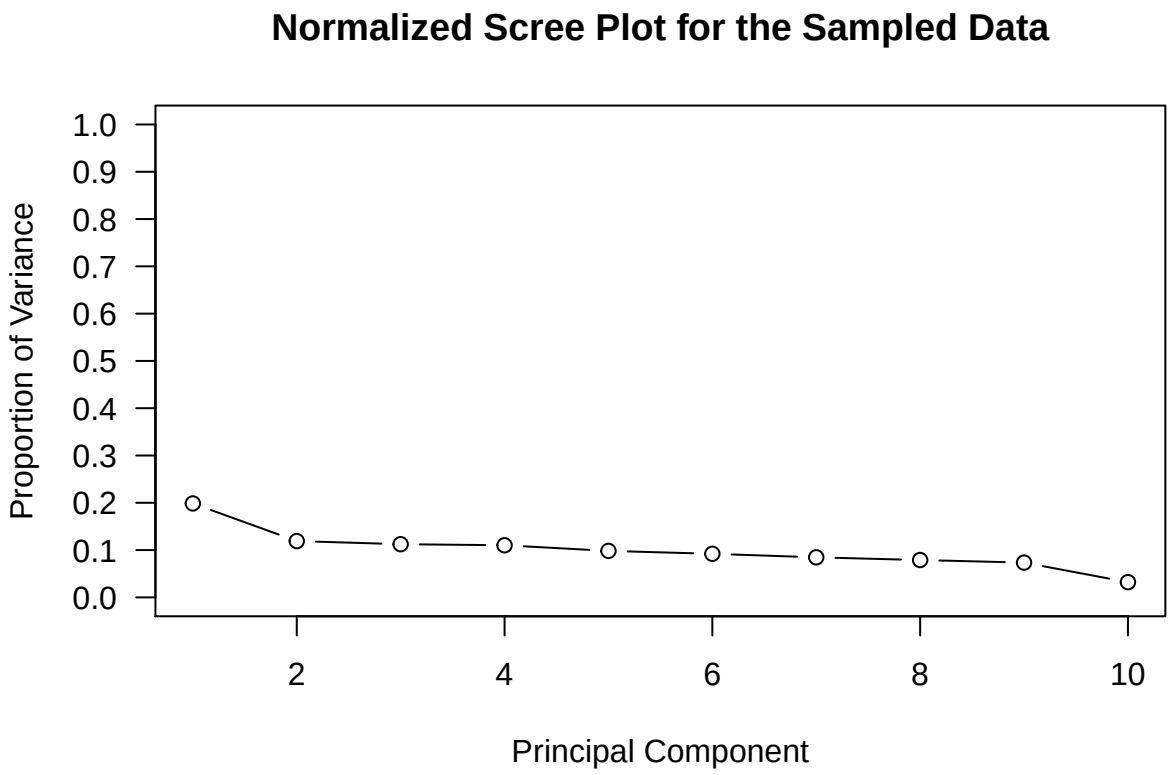


Figure 4: Scree Plot of the Principal Components using the sampled data, normalized.

```
## Importance of components:
##                            PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     1.4093 1.0910 1.0604 1.0496 0.99131 0.95972 0.92015
## Proportion of Variance 0.1986 0.1190 0.1124 0.1102 0.09827 0.09211 0.08467
## Cumulative Proportion  0.1986 0.3176 0.4301 0.5403 0.63854 0.73064 0.81531
##                            PC8     PC9   PC10
## Standard deviation     0.88863 0.85747 0.5674
## Proportion of Variance 0.07897 0.07353 0.0322
## Cumulative Proportion  0.89427 0.96780 1.0000
```

```
# Plotting the first 2 principal components
pca_plot<-my_ggbiplot(pca_data, var.axes=TRUE, groups = clean_sample[,c(11)])

# To make the arrows more visible
pca_plot$layers <- c(pca_plot$layers, pca_plot$layers[[1]])

pca_plot +
  labs(color="Income", # Labeling the Legend
  title = "Plotting Samples on the First 2 Principal Components for the Sampled Data") +
  theme(plot.title = element_text(hjust = 0.5), plot.caption=element_text(hjust = 0))
```

Figure 5: Plot of the first 2 principal components using our sampled data.

4.2 Multidimensional Scaling

We will perform our 4 MDS processes on the distance matrix, generated from the sampled dataset. Table 8 shows a sample of the distance matrix.

```
dist_matrix <- dist(data_matrix_mds, method = "euclidean")
```

```
dist_matrix_display <- as.matrix(dist_matrix)

kable(dist_matrix_display[0:8,0:8], align = "l", row.names = TRUE,
      caption = "Distance Matrix for the First 8 Features",
      booktabs = T)
```

Table 8: Distance Matrix for the First 8 Features

	1	2	3	4	5	6	7	8
1	0.000000	3.188555	3.736873	4.177427	5.109391	3.284725	3.702740	4.903122
2	3.188555	0.000000	3.680381	2.583108	4.515030	4.562907	5.393300	6.231991
3	3.736873	3.680381	0.000000	3.779341	4.305479	4.223639	6.103042	5.643576
4	4.177427	2.583108	3.779341	0.000000	4.197326	4.923961	5.525308	6.314496
5	5.109391	4.515030	4.305479	4.197326	0.000000	3.799942	5.318658	5.262991
6	3.284725	4.562907	4.223639	4.923961	3.799942	0.000000	4.237567	3.877535
7	3.702740	5.393300	6.103042	5.525308	5.318658	4.237567	0.000000	5.106621
8	4.903122	6.231991	5.643576	6.314496	5.262991	3.877535	5.106621	0.000000

4.2.1 Classical Multidimensional Scaling

Figure 6 shows the classic MDS applied on the distance matrix.

```
classic_mds <- cmdscale(dist_matrix, k = 2)
```

```
plot(classic_mds, main = "Classic Torgerson's MDS")
```

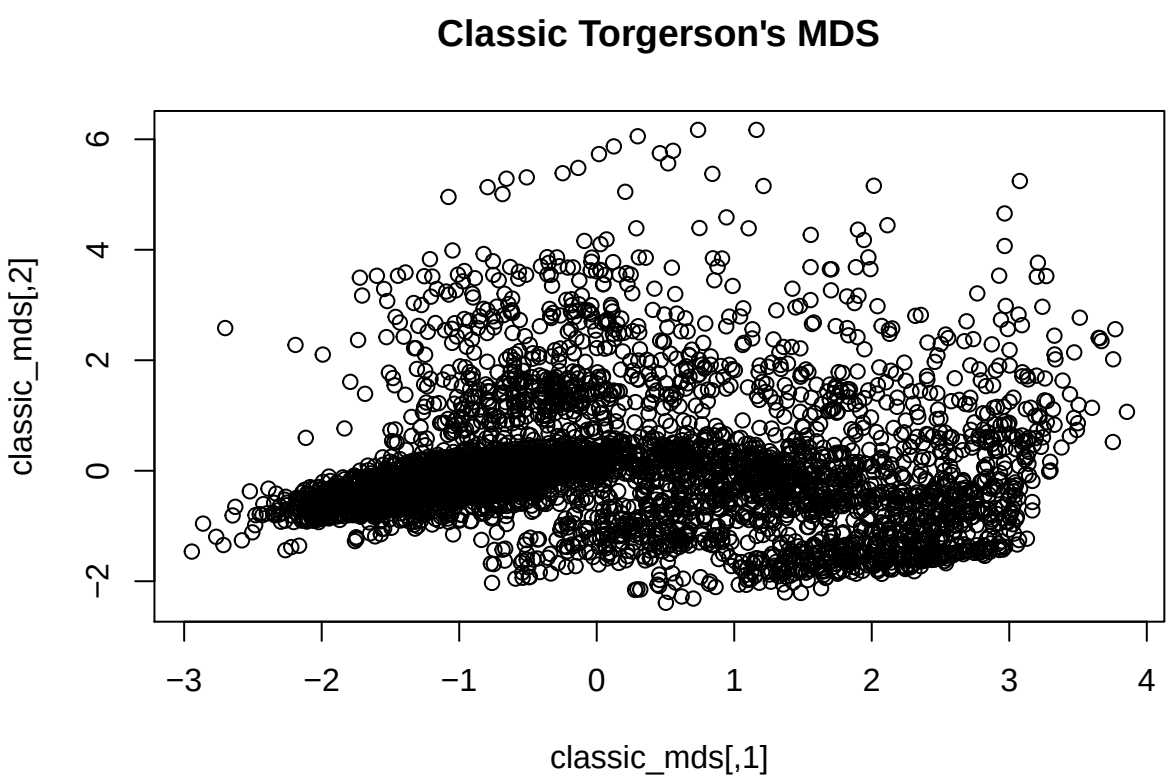


Figure 6: Plot of Classic Torgerson's Metric MDS applied on the distance matrix.

4.2.2 Sammon's Non-Linear Mapping

We apply Sammon's Non-Linear Mapping using 11 different initializations: using the Classical MDS as an initialization and 10 random initializations. For the random initializations, we show the stress and plot of the best initialization.

Figure 7 shows the Sammon's Non-Linear Mapping with Classic MDS as the initialization.

```
samm_mds_classic <- sammon(dist_matrix, y = classic_mds, k = 2, niter = 50)
```

```
## Initial stress        : 0.29921
## stress after   1 iters: 0.29423
```

```
print(paste0("Stress when using Classic_mds as initialization: ",
             samm_mds_classic$stress))
plot(samm_mds_classic$points,
     xlab="Samm_mds[,1]",
     ylab="Samm_mds[,2]",
     main = paste0("Sammon's MDS Mapping (Classic MDS as Initialization)")
     )
```

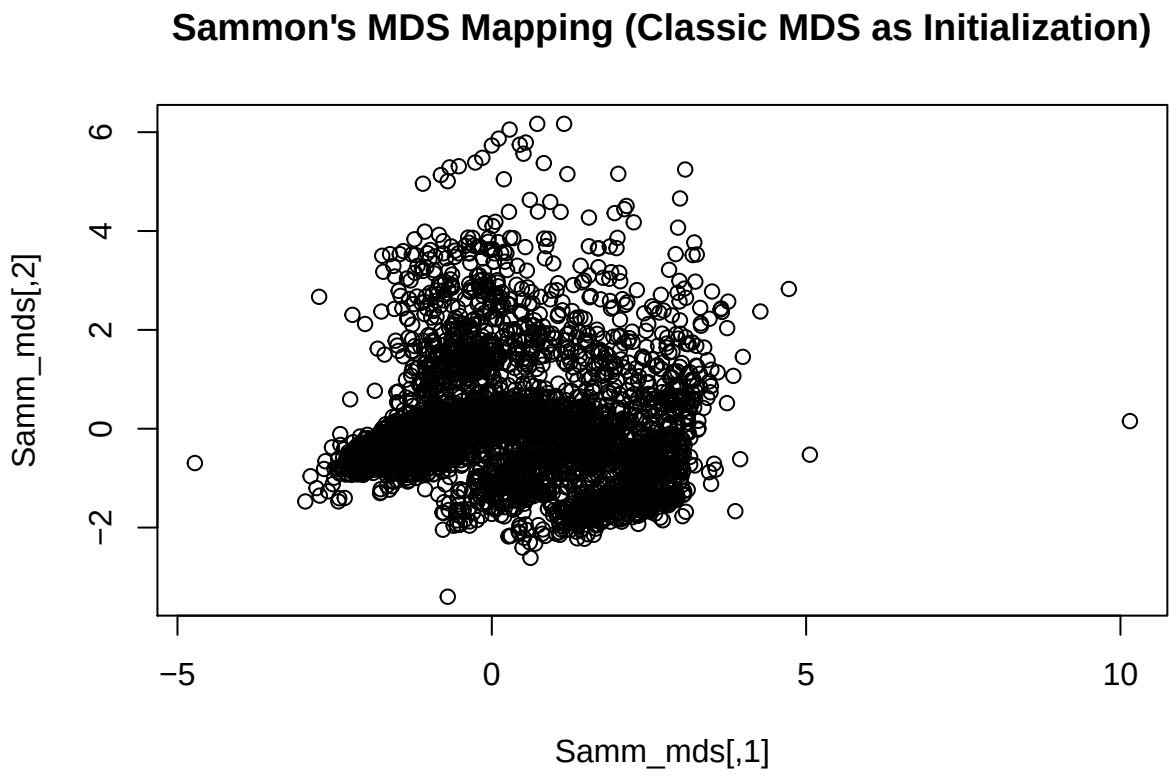


Figure 7: Plot of Sammon's non-linear mapping using classic MDS as the initialization.

```
## [1] "Stress when using Classic_mds as initialization: 0.294230053872011"
```

We now find Sammon's Non-Linear Mapping with 10 random initializations.

```
min_stress_samm <- .Machine$double.xmax # Largest double value the computer can store
```

```r
for(i in 1:10) {
  print(paste0("Iteration Number ", i,":"), i)
  rand_init <- matrix(rnorm(sample_number*2, 0, 100000), nrow = sample_number, ncol = 2)
  samm_mds <- sammon(dist_matrix, rand_init, k=2, niter = 50 )

  if (samm_mds$stress < min_stress_samm){
    best_samm_mds <- samm_mds
    min_stress_samm <- samm_mds$stress
    best_samm_i <- i
  }
}
```

```
## [1] "Iteration Number 1:"
## Initial stress        : 2480953868.50148
## stress after  10 iters: 14264.88139, magic = 0.500
## stress after  20 iters: 0.55388, magic = 0.014
## stress after  24 iters: 0.49957
## [1] "Iteration Number 2:"
## Initial stress        : 2459654178.99860
## stress after  10 iters: 14193.84314, magic = 0.500
## stress after  17 iters: 0.56640
## [1] "Iteration Number 3:"
## Initial stress        : 2515644300.03559
## stress after  10 iters: 14537.80356, magic = 0.500
## stress after  17 iters: 0.58661
## [1] "Iteration Number 4:"
## Initial stress        : 2457724410.84358
## stress after  10 iters: 14167.57712, magic = 0.500
## stress after  17 iters: 0.56512
## [1] "Iteration Number 5:"
## Initial stress        : 2454147993.29770
## stress after  10 iters: 14146.76715, magic = 0.500
## stress after  20 iters: 1.25833, magic = 0.068
## stress after  21 iters: 1.13138
## [1] "Iteration Number 6:"
## Initial stress        : 2421174753.47889
## stress after  10 iters: 14018.65235, magic = 0.500
## stress after  20 iters: 0.55107, magic = 0.014
## stress after  21 iters: 0.53939
## [1] "Iteration Number 7:"
## Initial stress        : 2506976420.10964
## stress after  10 iters: 14433.07426, magic = 0.500
## stress after  17 iters: 0.57303
## [1] "Iteration Number 8:"
## Initial stress        : 2497679921.13759
## stress after  10 iters: 14376.66928, magic = 0.500
## stress after  19 iters: 0.54261
## [1] "Iteration Number 9:"
## Initial stress        : 2452747317.52247
## stress after  10 iters: 14090.83697, magic = 0.500
## stress after  20 iters: 0.49808, magic = 0.068
## stress after  26 iters: 0.42495
## [1] "Iteration Number 10:"
## Initial stress        : 2502806933.53711
```

```
## stress after  10 iters: 14433.64894, magic = 0.500
## stress after  20 iters: 0.51060, magic = 0.068
## stress after  30 iters: 0.45316, magic = 0.031
## stress after  33 iters: 0.38619
```

The best random initialization's stress and iteration number are shown below, while its plot is shown in Figure 8.

```
print(paste0("Best Initialization's iteration number : ", best_samm_i))
print(paste0("Best Initialization's Stress: ", best_samm_mds$stress))

plot(best_samm_mds$points,
         xlab="Best Random initialization's Samm_mds[,1]",
         ylab="Best Random Initialization's Samm_mds[,2]",
         main = paste0("Sammon's MDS Mapping (Best Random Initialization)"))
```

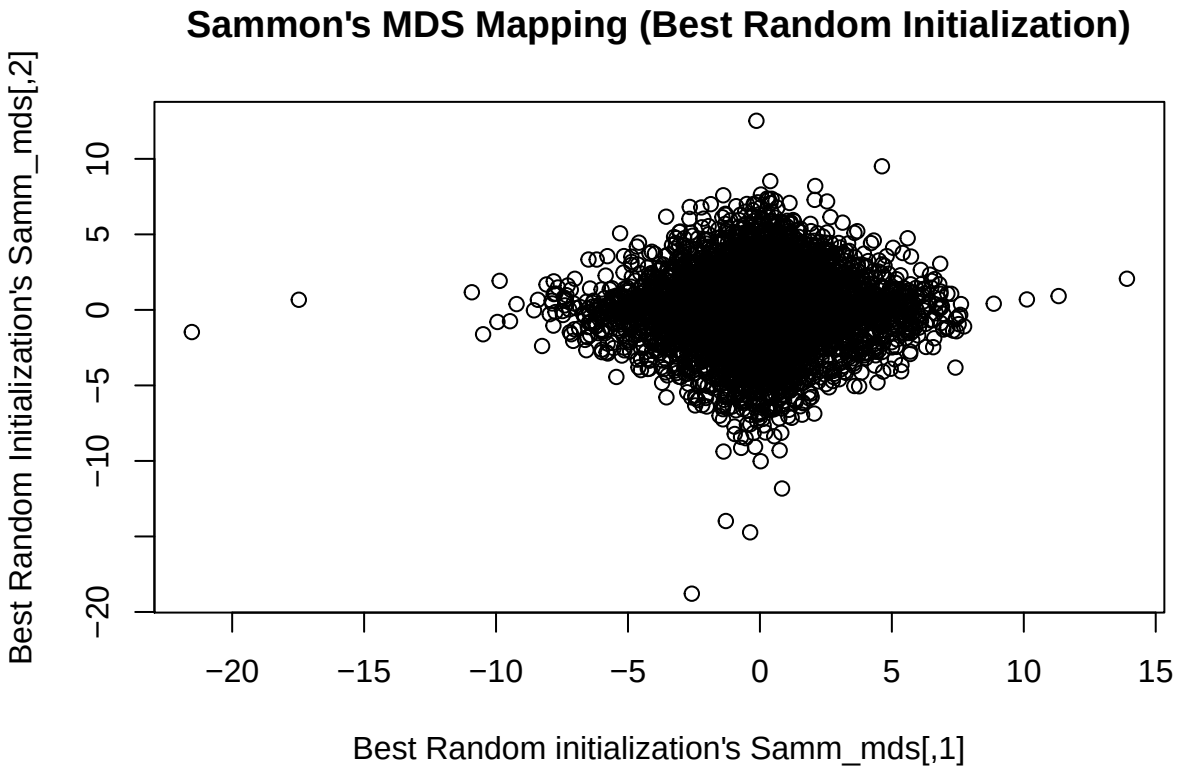


Figure 8: Plot of Sammon's non-linear mapping using the best found random initialization.

```
## [1] "Best Initialization's iteration number : 10"
## [1] "Best Initialization's Stress: 0.386185353688259"
```

4.2.3 Kruskal's Non-metric Multidimensional Scaling

We apply Kruskal's Non-metric Multidimensional Scaling using 11 different initializations: using the Classical MDS as an initialization and 10 random initializations. For the random initializations, we show the stress and plot of the best initialization.

Figure 9 shows the Kruskal's Non-metric MDS with Classic MDS as the initialization.

```
kruskal_mds_classic <- isoMDS(dist_matrix, y = classic_mds, k = 2, maxit = 10)
```

```
## initial  value 35.622289
## iter   5 value 27.931513
## iter  10 value 26.551124
## final  value 26.551124
## stopped after 10 iterations
```

```
print(paste0("Stress when using classic_mds as initialization: ", kruskal_mds_classic$stress))
plot(kruskal_mds_classic$points, xlab = "kruskal_mds[,1]", ylab = "kruskal_mds[,2]",
    main = paste0("Kruskal's MDS Mapping (Classic MDS as Initialization)"))
```

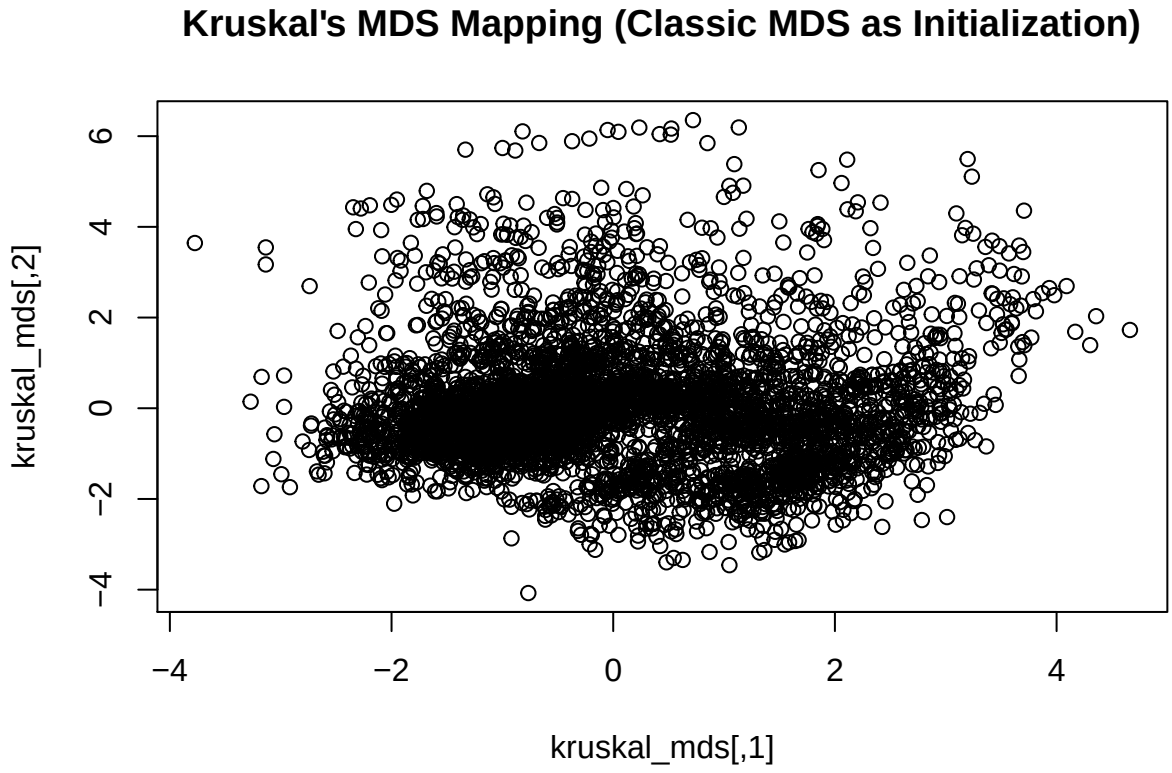


Figure 9: Plot of Kruskal's non-linear mapping using classic MDS as initialization.

```
## [1] "Stress when using classic_mds as initialization: 26.5511238380599"
```

We now find Kruskal's Non-metric MDS Mapping with 10 random initializations.

```
min_stress_kruskal <- .Machine$double.xmax # Largest double value the computer can store

for(i in 1:10) {
  print(paste0("Iteration Number ", i,":"), i)
  rand_init <- matrix(rnorm(sample_number*2, 0, 100000), nrow = sample_number, ncol = 2)
  kruskal_mds <- isoMDS(dist_matrix, rand_init, k=2, maxit = 10 )

  if (kruskal_mds$stress < min_stress_kruskal){
    best_kruskal_mds <- kruskal_mds
```

```
    min_stress_kruskal <- kruskal_mds$stress
    best_kruskal_i <- i
  }
}
```

```
## [1] "Iteration Number 1:"
## initial  value 46.266270
## final  value 46.266270
## converged
## [1] "Iteration Number 2:"
## initial  value 46.319993
## final  value 46.319993
## converged
## [1] "Iteration Number 3:"
## initial  value 46.161005
## final  value 46.161005
## converged
## [1] "Iteration Number 4:"
## initial  value 46.311481
## final  value 46.311481
## converged
## [1] "Iteration Number 5:"
## initial  value 46.171638
## final  value 46.171638
## converged
## [1] "Iteration Number 6:"
## initial  value 46.249081
## final  value 46.249081
## converged
## [1] "Iteration Number 7:"
## initial  value 46.298491
## final  value 46.298491
## converged
## [1] "Iteration Number 8:"
## initial  value 46.346385
## final  value 46.346385
## converged
## [1] "Iteration Number 9:"
## initial  value 46.209023
## final  value 46.209023
## converged
## [1] "Iteration Number 10:"
## initial  value 46.164926
## final  value 46.164926
## converged
```

The best random initialization's stress and iteration number are shown below, while its plot is shown in Figure 10.

```
print(paste0("Best Initialization's iteration number : ", best_kruskal_i))
print(paste0("Best Initialization's Stress: ", best_kruskal_mds$stress))

plot(best_kruskal_mds$points,
         xlab="Best Random initialization's kruskal_mds[,1]",
         ylab="Best Random Initialization's kruskal_mds[,2]",
```

```
        main = paste0("Kruskal's MDS Mapping (Best Random Initialization)"))
```

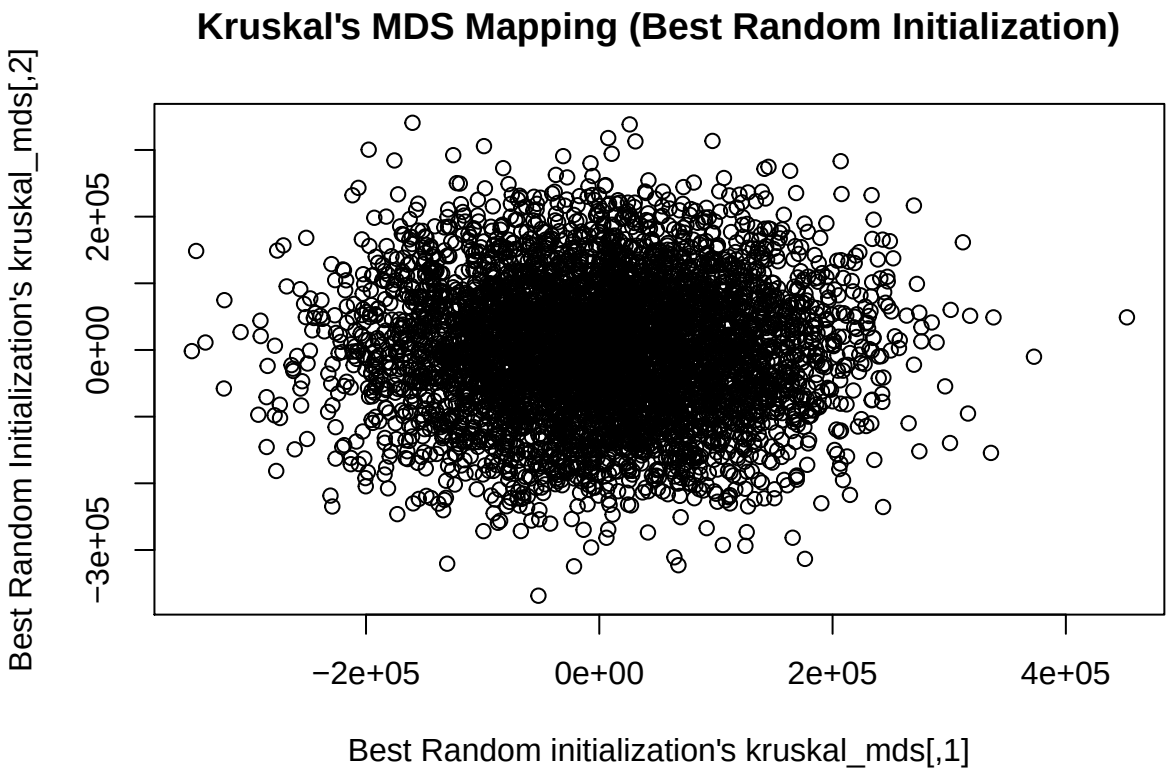


Figure 10: Plot of Kruskal's Non-metric MDS mapping using the best found random initialization.

```
## [1] "Best Initialization's iteration number : 3"
## [1] "Best Initialization's Stress: 46.1610051079746"
```

4.2.4 Symmetric Smacof

We apply MDS on a symmetric dissimilarity (distance in our case) matrix using SMACOF with 11 different initializations: using the Classical MDS as an initialization and 10 random initializations. For the random initializations, we show the stress and plot of the best initialization.

Figure 11 shows the Symmetric Smacof MDS with Classic MDS ("torgerson") as the initialization.

```
smacof_mds_classic <- smacofSym(dist_matrix, ndim = 2, init = "torgerson", itmax = 1000)
```

```
print(paste0("Stress when using torgerson's classic MDS as initialization: ",
             smacof_mds_classic$stress))

plot(smacof_mds_classic$conf,
     xlab="smacofSym_mds[,1]",
     ylab="smacofSym_mds[,2]",
     main = paste0("Symmetric Smacof MDS (Classic MDS as Initialization)")
     )
```

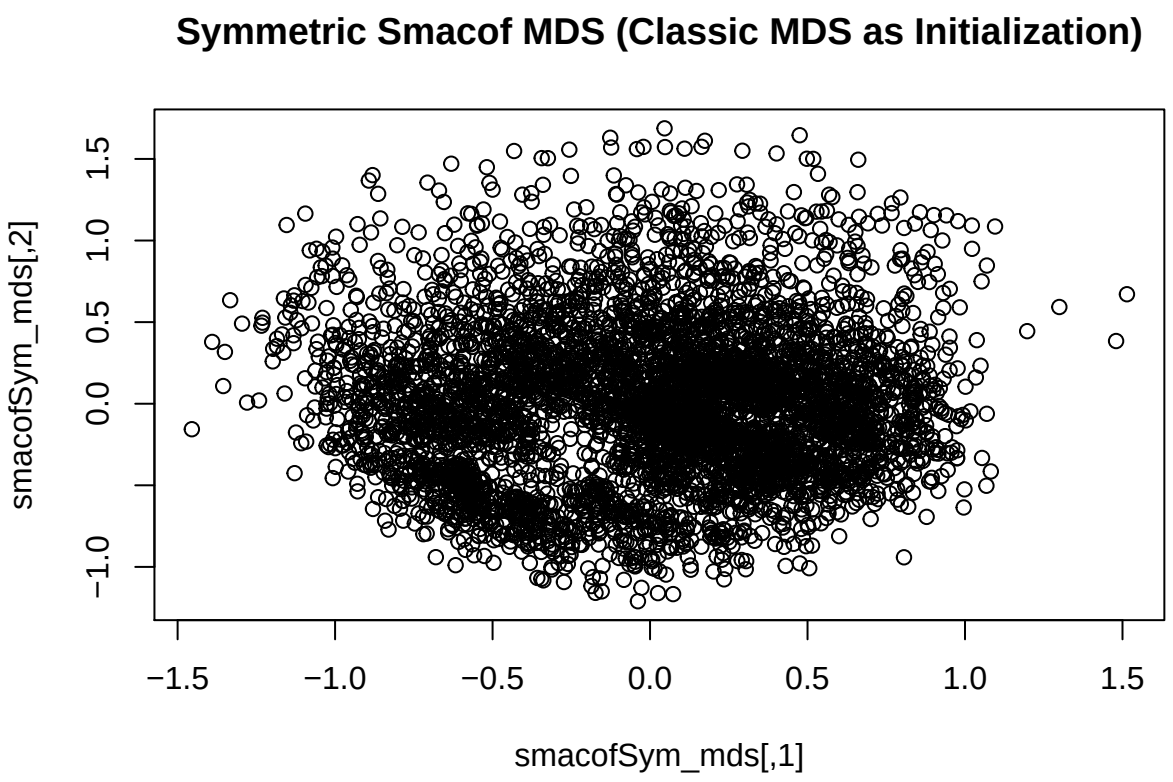


Figure 11: Plot of Symmetric Smacof non-linear mapping using classic MDS as initialization.

```
## [1] "Stress when using torgerson's classic MDS as initialization: 0.289220141587415"
```

We now find the Symmetric Smacof MDS Mapping with 10 random initializations.

```
min_stress_smacof <- .Machine$double.xmax # Largest double value the computer can store

for(i in 1:10) {
  print(paste0("Iteration Number ", i,":"), i)
  smacof_mds <- smacofSym(dist_matrix, ndim = 2, init = "random", itmax = 1000 )

  if (smacof_mds$stress < min_stress_smacof){
    best_smacof_mds <- smacof_mds
    min_stress_smacof <- smacof_mds$stress
    best_smacof_i <- i
  }
}
```

```
## [1] "Iteration Number 1:"
## [1] "Iteration Number 2:"
## [1] "Iteration Number 3:"
## [1] "Iteration Number 4:"
## [1] "Iteration Number 5:"
## [1] "Iteration Number 6:"
## [1] "Iteration Number 7:"
## [1] "Iteration Number 8:"
```

```
## [1] "Iteration Number 9:"
## [1] "Iteration Number 10:"
```

The best random initialization's stress and iteration number are shown below, while its plot is shown in Figure 12.

```
print(paste0("Best Initialization's iteration number : ", best_smacof_i))
print(paste0("Best Initialization's Stress: ", best_smacof_mds$stress))

plot(best_smacof_mds$conf,
         xlab="Best Random initialization's smacofSym_mds[,1]",
         ylab="Best Random Initialization's smacofSym_mds[,2]",
         main = paste0("Symmetric Smacof MDS Mapping (Best Random Initialization)"))
```

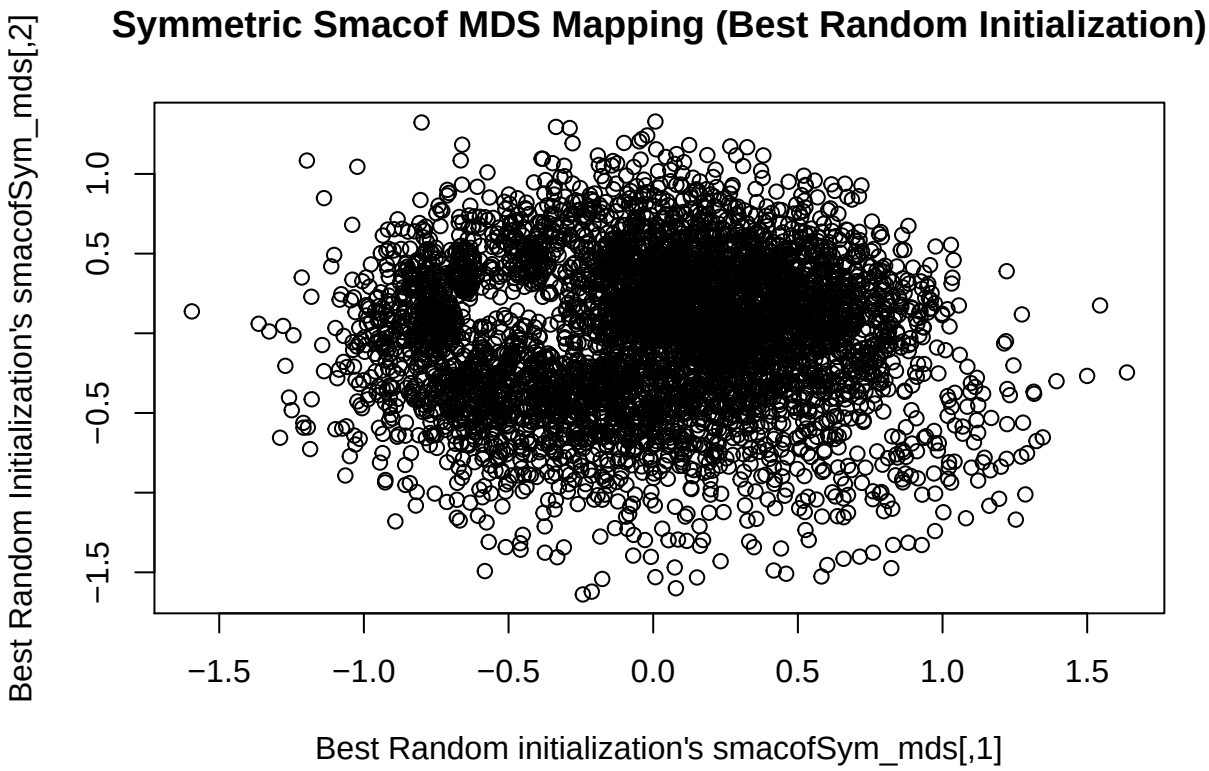


Figure 12: Plot of Symmetric Smacof MDS mapping using the best found random initialization.

```
## [1] "Best Initialization's iteration number : 4"
## [1] "Best Initialization's Stress: 0.291379692436169"
```

4.3 UMAP

Note: Details on UMAP were directly written on the Final Paper.

```
head(data_matrix_mds, 3)
```

```
##             age  workclass education.num marital.status occupation relationship
## [1,] -0.8891898 -0.1515961    -0.6474759      1.1848224 -0.7287627   -0.1200077
## [2,] -1.7602272 -0.1515961    -0.6474759     -0.2881045 -0.2339819   -0.7329173
```

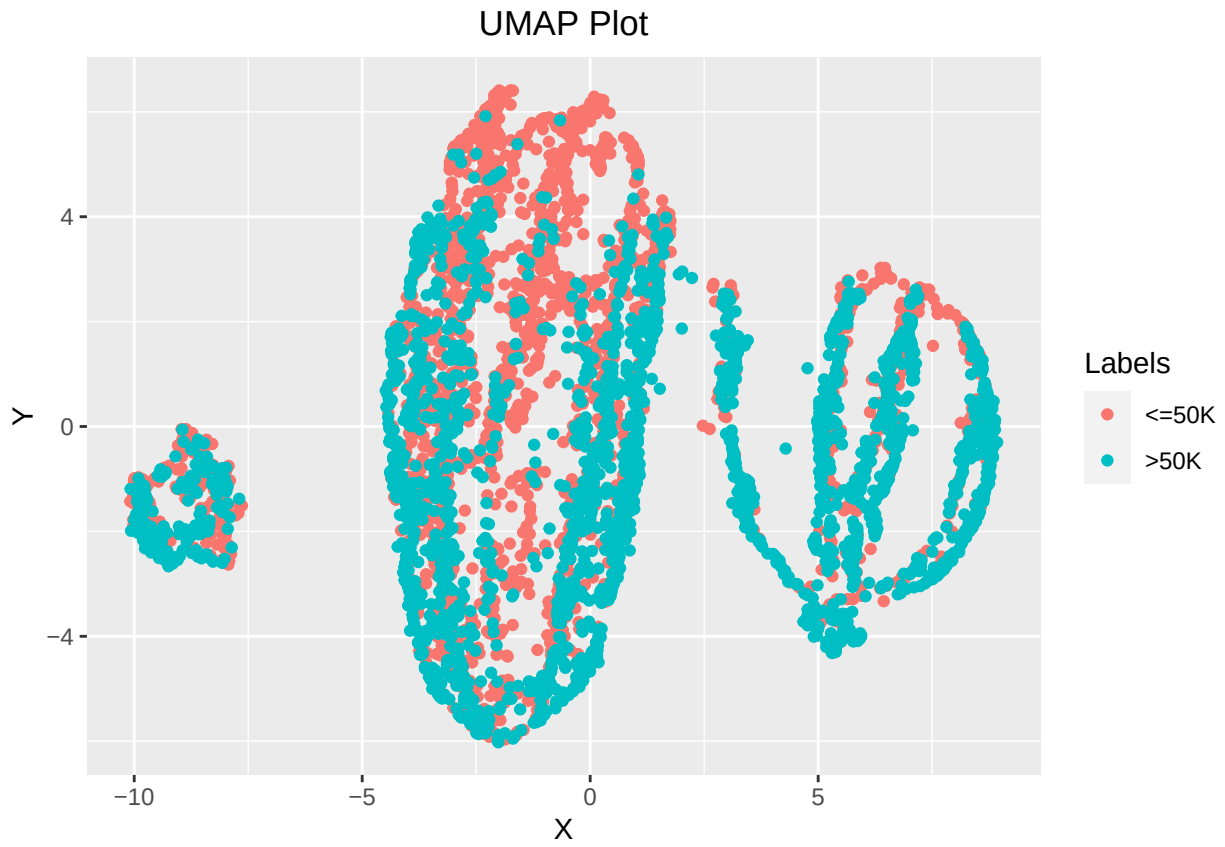
```
## [3,] -0.8020861  1.8223112    -0.6474759      1.1848224 -0.9761531    1.7187210
##           race       sex hours.per.week native.country
## [1,] 0.3872955 0.6000851     -0.4594432     -0.7454363
## [2,] 0.3872955 0.6000851      1.9034617      0.2718538
## [3,] 0.3872955 0.6000851      1.9034617      0.2718538
```

```
# Hyperparameter tuning. Not included in the RMarkdown pdf, hence eval=FALSE. We
# did tuning durectly in RMarkdown.

min_dist_1 = seq(0.001, 0.2, 0.05)
min_dist_2 = seq(0.1, 0.5, 0.05)
min_dist = c(min_dist_1, min_dist_2)
c = 0
for (n_neig in seq(5, 50, 5)) {
    # 10 Iterations 13 Iterations
    for (min_distance in min_dist) {
        UMAP_Data = umap(data_matrix, n_neighbors = n_neig, min_dist = min_distance)
        df = data.frame(X = UMAP_Data[, 1], Y = UMAP_Data[, 2], Labels = dataLabels)
        plot = ggplot(data = df, aes(x = X, y = Y, col = Labels)) + geom_point()
        c = c + 1
        fileName = paste("ID", c, "N_N", n_neig, "M_D", min_distance, ".png")
        png(fileName)
        print(plot)
        dev.off()
    }
}
```

```
dataLabels = clean_sample[, 11] %>% unlist %>% factor(labels = c("<=50K", ">50K"))

sample_UMAP <- data.matrix(clean_sample)
sample_UMAP <- sample_UMAP[, -c(11)]
```

```
UMAP_5000 = umap(sample_UMAP, n_neighbors = 30, min_dist = 0.15)
df = data.frame(X = UMAP_5000$layout[, 1], Y = UMAP_5000$layout[, 2], Labels = dataLabels)
```

```
plot = ggplot(data = df, aes(x = X, y = Y, col = Labels)) + geom_point() + ggtitle("UMAP Plot") +
    theme(plot.title = element_text(hjust = 0.5))

print(plot)
```



4.4 TSNE

Note: Details on t-SNE were directly written on the Final Paper.

```
# Hyperparameter tuning. Not included in the RMarkdown pdf, hence eval=FALSE. We
# did tuning durectly in RMarkdown

c = 0
for (per in seq(1, 400, 5)) {
    # 50 Iterations

    tSNE_Data = Rtsne(data_matrix, perplexity = per)
    df = data.frame(X = tSNE_Data$Y[, 1], Y = tSNE_Data$Y[, 2], Labels = dataLabels)
    plot = ggplot(data = df, aes(x = X, y = Y, col = Labels)) + geom_point()
    c = c + 1
    fileName = paste("ID", c, "Perplexity", per, ".png")
    png(fileName)
    print(plot)
    dev.off()

}
```

```
tSNE_Data = Rtsne(sample_UMAP, perplexity = 81)
df_tsne = data.frame(X = tSNE_Data$Y[, 1], Y = tSNE_Data$Y[, 2], Labels = dataLabels)
```

```
plot_tsne = ggplot(data = df_tsne, aes(x = X, y = Y, col = Labels)) + geom_point() +
    ggtitle("t-SNE Plot") + theme(plot.title = element_text(hjust = 0.5))
```

```
print(plot_tsne)
```

4.5 Clustering

4.5.1 Hierarchical Clustering

We first perform Agglomerative Nesting (AGNES) hierarchical clustering. We load all the necessary libraries, create our scaled distance matrix, and find hierarchical clusters for the 6 linkages, noting their agglomerative coefficients below. Figure 13 shows the dendrograms for all the linkages. Then, we perform the Divisive Analysis (DIANA) clustering, show its divisive coefficient, and plot its dendrogram in Figure 14.

```
dist_matrix_scaled <- dist(data_matrix_scaled, method = "euclidean")
```

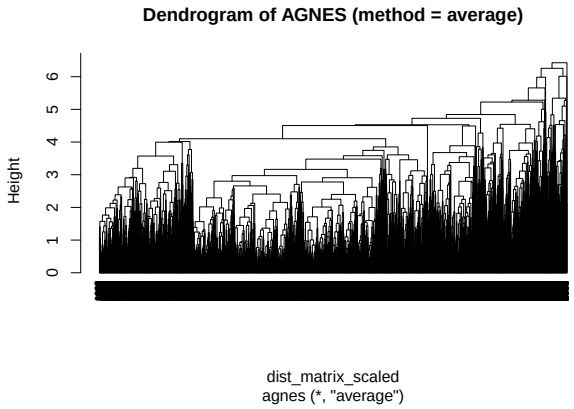
```
hier_aver <- agnes(dist_matrix_scaled, method = "average")
hier_sing <- agnes(dist_matrix_scaled, method = "single")
hier_comp <- agnes(dist_matrix_scaled, method = "complete")
hier_ward <- agnes(dist_matrix_scaled, method = "ward")
hier_weig <- agnes(dist_matrix_scaled, method = "weighted")
hier_gave <- agnes(dist_matrix_scaled, method = "gaverage")
```

```
m <- list(hier_aver, hier_sing, hier_comp, hier_ward, hier_weig, hier_gave)
names(m) <- c("average", "single", "complete", "ward", "weighted", "gaverage")

ac <- function(x) {
    x$ac
}
```

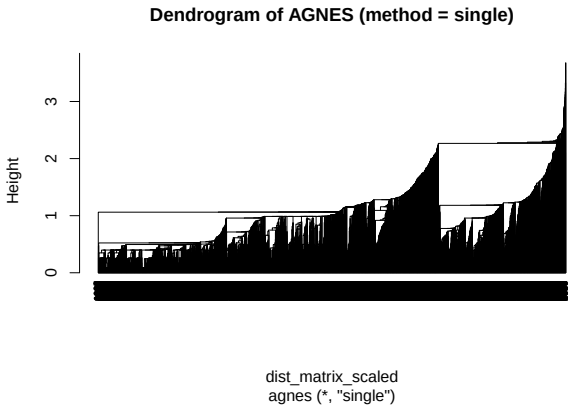
```r
print("The agglomerative coefficients for each of the 6 linkages are shown below:")
map_dbl(m, ac)
```

```
## [1] "The agglomerative coefficients for each of the 6 linkages are shown below:"
##   average    single  complete      ward  weighted  gaverage
## 0.8950687 0.8367716 0.9351114 0.9934071 0.9073330 0.9761316
```

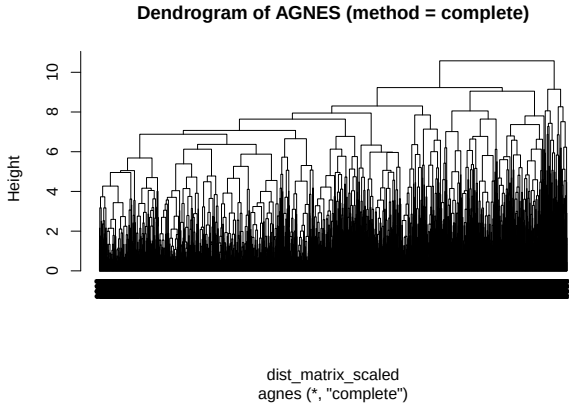
```r
pltree(hier_aver, cex = 0.6, hang = -1, main = paste0("Dendrogram of AGNES (method = average)"))
pltree(hier_sing, cex = 0.6, hang = -1, main = paste0("Dendrogram of AGNES (method = single)"))
pltree(hier_comp, cex = 0.6, hang = -1, main = paste0("Dendrogram of AGNES (method = complete)"))
pltree(hier_ward, cex = 0.6, hang = -1, main = paste0("Dendrogram of AGNES (method = ward)"))
pltree(hier_weig, cex = 0.6, hang = -1, main = paste0("Dendrogram of AGNES (method = weighted)"))
pltree(hier_gave, cex = 0.6, hang = -1, main = paste0("Dendrogram of AGNES (method = gaverage)"))
```



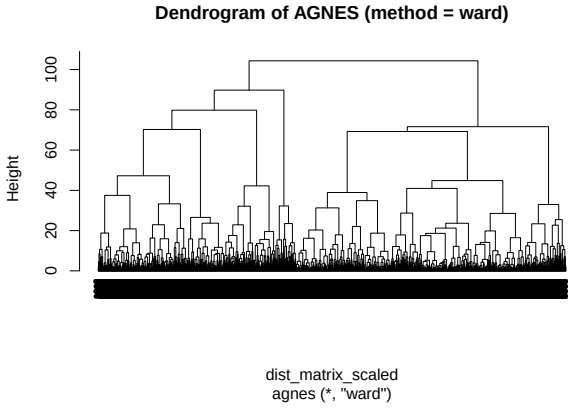
(a) Cluster Similarity: Group Average



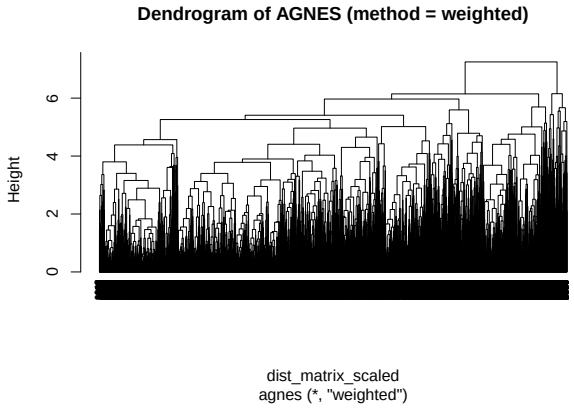
(b) Cluster Similarity: Single Link



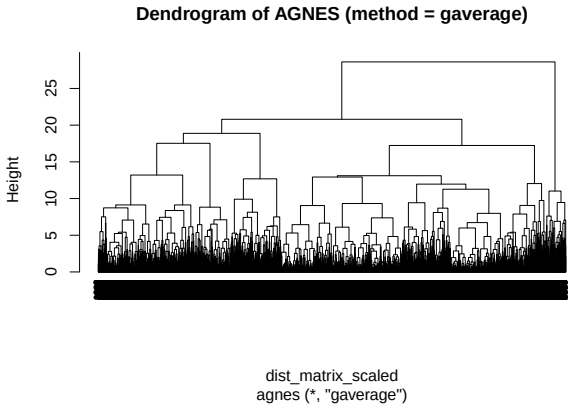
(c) Cluster Similarity: Complete Link



(d) Cluster Similarity: Ward's Method



(e) Cluster Similarity: Weighted (UPGMA)



(f) Cluster Similarity: Generalized Average (Flexible UPGMA)

Figure 13: Dendrogram of the 6 hierarchical clustering methods.

```
# For the divisive hierarchical clustering
hier_diana <- diana(dist_matrix_scaled)
```

```r
# Divise coefficient
print(paste0("The divisive coefficient is: ", hier_diana$dc))

# Plotting
pltree(hier_diana, cex = 0.6, hang = -1, main = "Dendrogram of DIANA")
```

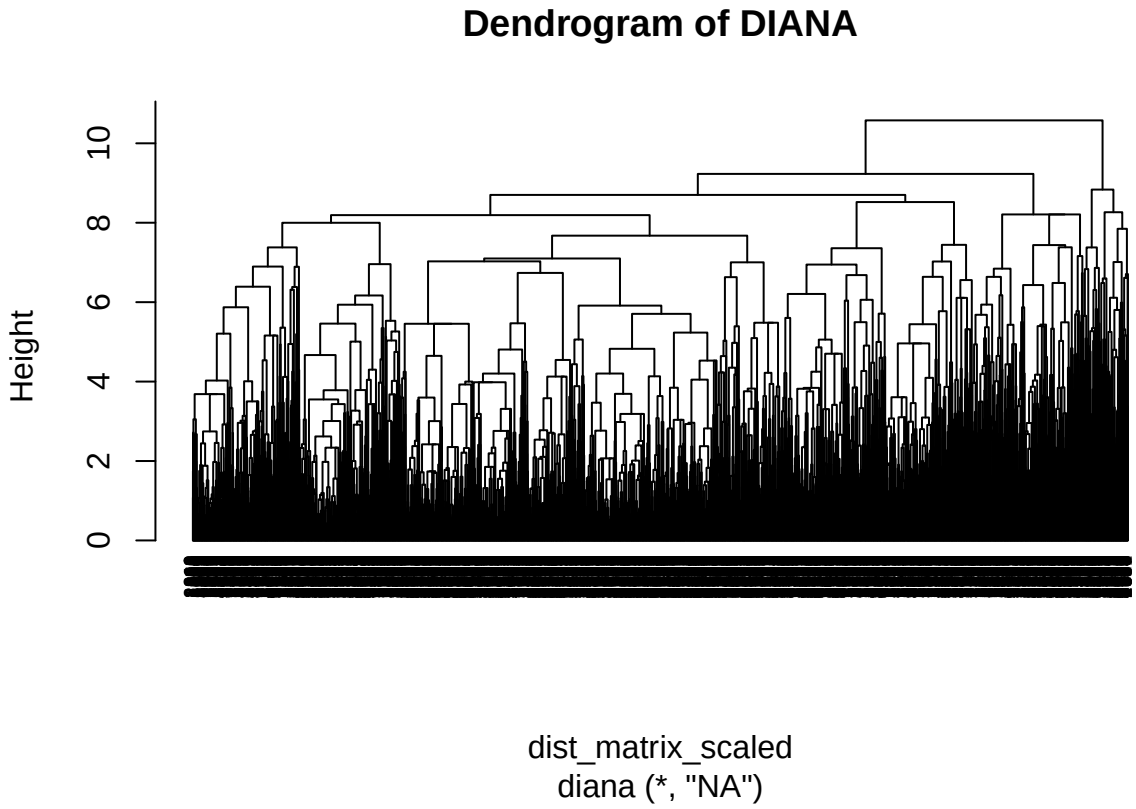


Figure 14: Dendrogram of the Divisive Analysis clustering.

```
## [1] "The divisive coefficient is: 0.924897043630849"
```

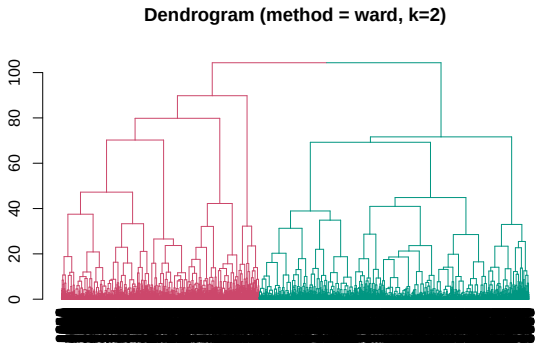
Given that the highest agglomerative coefficients are for the Ward's method and Generalized Average linkage, we will select their and DIANA's dendrogram and cut the tree to make 2 and 5 clusters, which we will use as initialization for k-means clustering. In addition, Figure 15 shows their colored dendrograms depicting these cluster choices.

```r
# Ward
cut_ward_2 <- cutree(hier_ward, k = 2)
cut_ward_5 <- cutree(hier_ward, k = 5)
# GAverage
cut_gave_2 <- cutree(hier_gave, k = 2)
cut_gave_5 <- cutree(hier_gave, k = 5)
# DIANA
cut_diana_2 <- cutree(hier_diana, k = 2)
cut_diana_5 <- cutree(hier_diana, k = 5)
```

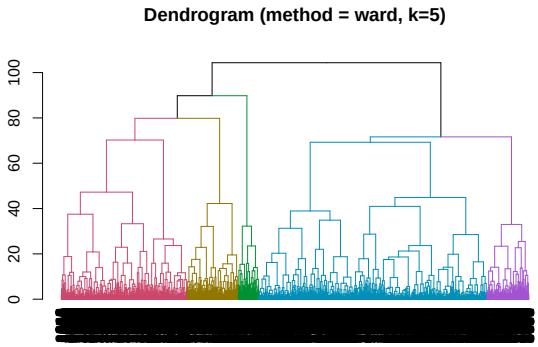
```r
# Ward
cluster_ward_2 <- aggregate(data_matrix_scaled, list(cluster = cut_ward_2), mean)
```

```r
cluster_ward_2 <- cluster_ward_2[, -c(1)]

cluster_ward_5 <- aggregate(data_matrix_scaled, list(cluster = cut_ward_5), mean)
cluster_ward_5 <- cluster_ward_5[, -c(1)]

# GAverage
cluster_gave_2 <- aggregate(data_matrix_scaled, list(cluster = cut_gave_2), mean)
cluster_gave_2 <- cluster_gave_2[, -c(1)]

cluster_gave_5 <- aggregate(data_matrix_scaled, list(cluster = cut_gave_5), mean)
cluster_gave_5 <- cluster_gave_5[, -c(1)]

# DIANA
cluster_diana_2 <- aggregate(data_matrix_scaled, list(cluster = cut_diana_2), mean)
cluster_diana_2 <- cluster_diana_2[, -c(1)]

cluster_diana_5 <- aggregate(data_matrix_scaled, list(cluster = cut_diana_5), mean)
cluster_diana_5 <- cluster_diana_5[, -c(1)]
```

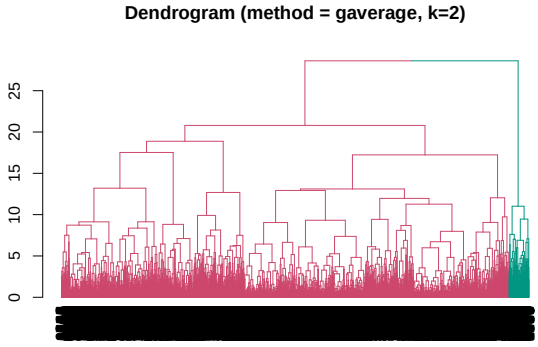
```r
dend_obj_ward <- as.dendrogram(hier_ward)
dend_col_ward_2 <- color_branches(dend_obj_ward, k = 2)
dend_col_ward_5 <- color_branches(dend_obj_ward, k = 5)
plot(dend_col_ward_2, main = "Dendrogram (method = ward, k=2)")
plot(dend_col_ward_5, main = "Dendrogram (method = ward, k=5)")

dend_obj_gave <- as.dendrogram(hier_gave)
dend_col_gave_2 <- color_branches(dend_obj_gave, k = 2)
dend_col_gave_5 <- color_branches(dend_obj_gave, k = 5)
plot(dend_col_gave_2, main = "Dendrogram (method = gaverage, k=2)")
plot(dend_col_gave_5, main = "Dendrogram (method = gaverage, k=5)")

dend_obj_diana <- as.dendrogram(hier_diana)
dend_col_diana_2 <- color_branches(dend_obj_diana, k = 2)
dend_col_diana_5 <- color_branches(dend_obj_diana, k = 5)
plot(dend_col_diana_2, main = "Dendrogram (method = DIANA, k=2)")
plot(dend_col_diana_5, main = "Dendrogram (method = DIANA, k=5)")
```



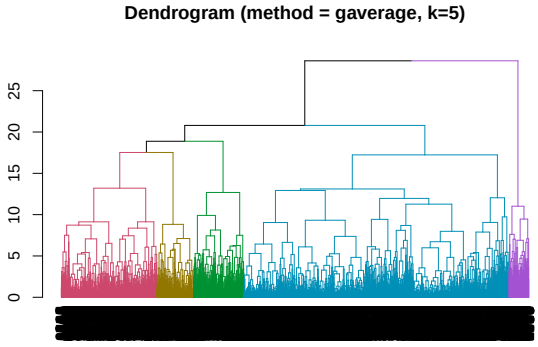
(a) Ward's Method, Clustering = 2.



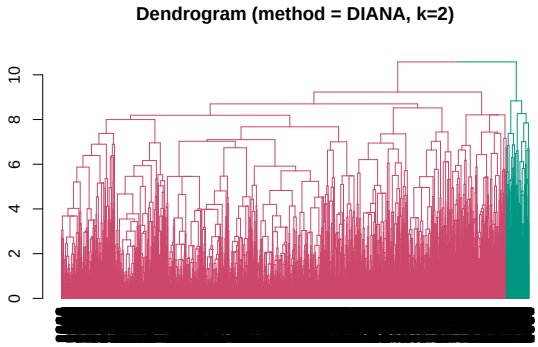
(b) Ward's Method, Clustering = 5.



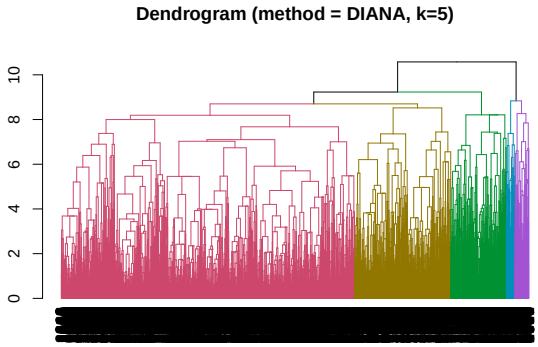
(c) Generalized Average, Clustering = 2.



(d) Generalized Average, Clustering = 5.



(e) DIANA, Clustering = 2.



(f) DIANA, Clustering = 5.

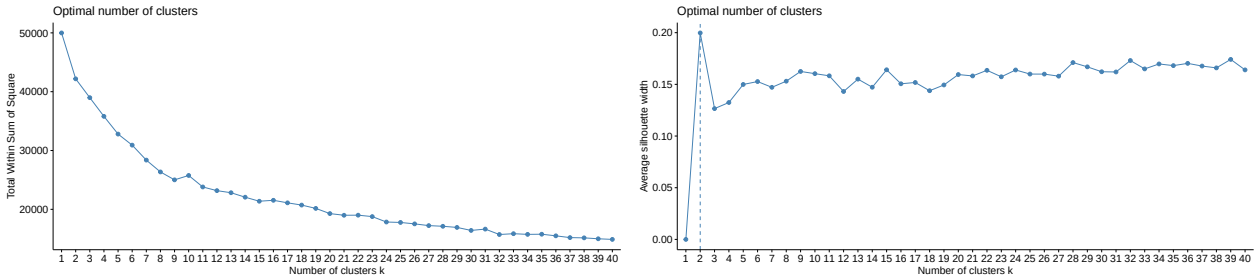
Figure 15: Dendrograms of the hierarchical clusterings using Ward's method, Generalized Average method, and Divisive Analysis (DIANA) clustering, colored to represent clusters of 2 (the original clustering) and 5 (predicted clustering from PCA).

4.5.2 K-Means Clustering

First, we try to estimate the best number of clusters, k. Figure 16 shows the results of the elbow and silhouette methods. For the elbow method, there seems to be a small bend at k = 12. From the silhouette method, k = 2 seems to be the best choice.

```
elbow_wss <- fviz_nbclust(data_matrix_scaled, kmeans, method = "wss", k.max = 40)
elbow_silho <- fviz_nbclust(data_matrix_scaled, kmeans, method = "silhouette", k.max = 40)
```

```
# To plot the k-estimating plots.

elbow_wss

elbow_silho
```



(a) Total Within Sum of Squares vs. different number of clusters. (b) Average silhouette width vs. different number of clusters.

Figure 16: Plots to help estimate the number of clusters, k, to select for k-means clustering, using the elbow method (a) and the silhouette method (b).

Now, we perform k-means clustering with random initializations for our estimated k = 12 and k = 5, as well as k = 2 (the original clustering) and k = 4 (out of curiosity). In addition, we perform 6 additional k-means clustering using the clusters found from our hierarchical clustering results. Figure 17 shows the best randomly initialized k-means clustering projected onto the first 2 principal components of a PCA, while Figure 18 shows similar plots for the hierarchical clustering-initialized k-means clusters. Below, we also give the size of each of the clusters within each k-means clustering.

```
k_clus_2 <- kmeans(data_matrix_scaled, centers = 2, nstart = 100)
k_clus_4 <- kmeans(data_matrix_scaled, centers = 4, nstart = 100)
k_clus_5 <- kmeans(data_matrix_scaled, centers = 5, nstart = 100)
k_clus_12 <- kmeans(data_matrix_scaled, centers = 12, nstart = 100)
```

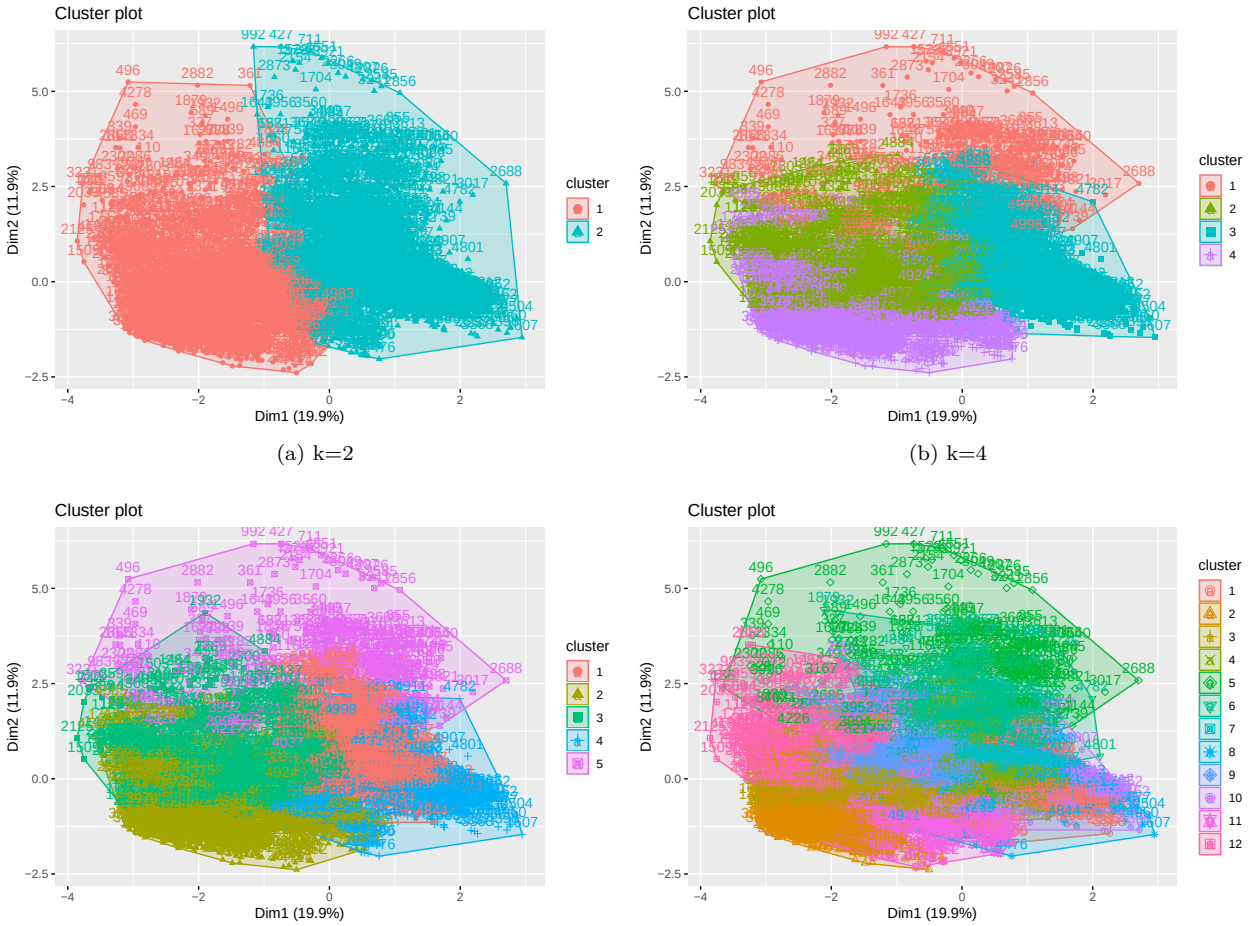
```
k_clus_ward_2 <- kmeans(data_matrix_scaled, centers = cluster_ward_2)
k_clus_ward_5 <- kmeans(data_matrix_scaled, centers = cluster_ward_5)
k_clus_gave_2 <- kmeans(data_matrix_scaled, centers = cluster_gave_2)
k_clus_gave_5 <- kmeans(data_matrix_scaled, centers = cluster_gave_5)
k_clus_diana_2 <- kmeans(data_matrix_scaled, centers = cluster_diana_2)
k_clus_diana_5 <- kmeans(data_matrix_scaled, centers = cluster_diana_5)
```

```
noquote("The size of the clusters for each of the k-means clusters are shown below:")
noquote("Randomly Initialized, k = 2:")
print(k_clus_2$size)
noquote("Randomly Initialized, k = 4:")
print(k_clus_4$size)
noquote("Randomly Initialized, k = 5:")
print(k_clus_5$size)
```

```r
noquote("Randomly Initialized, k = 12:")
print(k_clus_12$size)

noquote("Ward's Method Initialized, k = 2:")
print(k_clus_ward_2$size)
noquote("Ward's Method Initialized, k = 5:")
print(k_clus_ward_5$size)
noquote("Generalized Averaged Method Initialized, k = 2:")
print(k_clus_gave_2$size)
noquote("Generalized Averaged Method Initialized, k = 5:")
print(k_clus_gave_5$size)
noquote("DIANA Method Initialized, k = 2:")
print(k_clus_diana_2$size)
noquote("DIANA Method Initialized, k = 5:")
print(k_clus_diana_5$size)
```

```
## [1] The size of the clusters for each of the k-means clusters are shown below:
## [1] Randomly Initialized, k = 2:
## [1] 1597 3403
## [1] Randomly Initialized, k = 4:
## [1]  236 1245 2788  731
## [1] Randomly Initialized, k = 5:
## [1] 1841  702 1161 1063  233
## [1] Randomly Initialized, k = 12:
##  [1] 501 284 447 807 179 321 618 376 476 409 356 226
## [1] Ward's Method Initialized, k = 2:
## [1] 1597 3403
## [1] Ward's Method Initialized, k = 5:
## [1] 1199 2418  585  587  211
## [1] Generalized Averaged Method Initialized, k = 2:
## [1] 4711  289
## [1] Generalized Averaged Method Initialized, k = 5:
## [1] 1033 2520  687  546  214
## [1] DIANA Method Initialized, k = 2:
## [1] 4711  289
## [1] DIANA Method Initialized, k = 5:
## [1] 2895 1284  581   93  147
```

```r
fviz_cluster(k_clus_2, data = data_matrix_scaled)
fviz_cluster(k_clus_4, data = data_matrix_scaled)
fviz_cluster(k_clus_5, data = data_matrix_scaled)
fviz_cluster(k_clus_12, data = data_matrix_scaled)
```

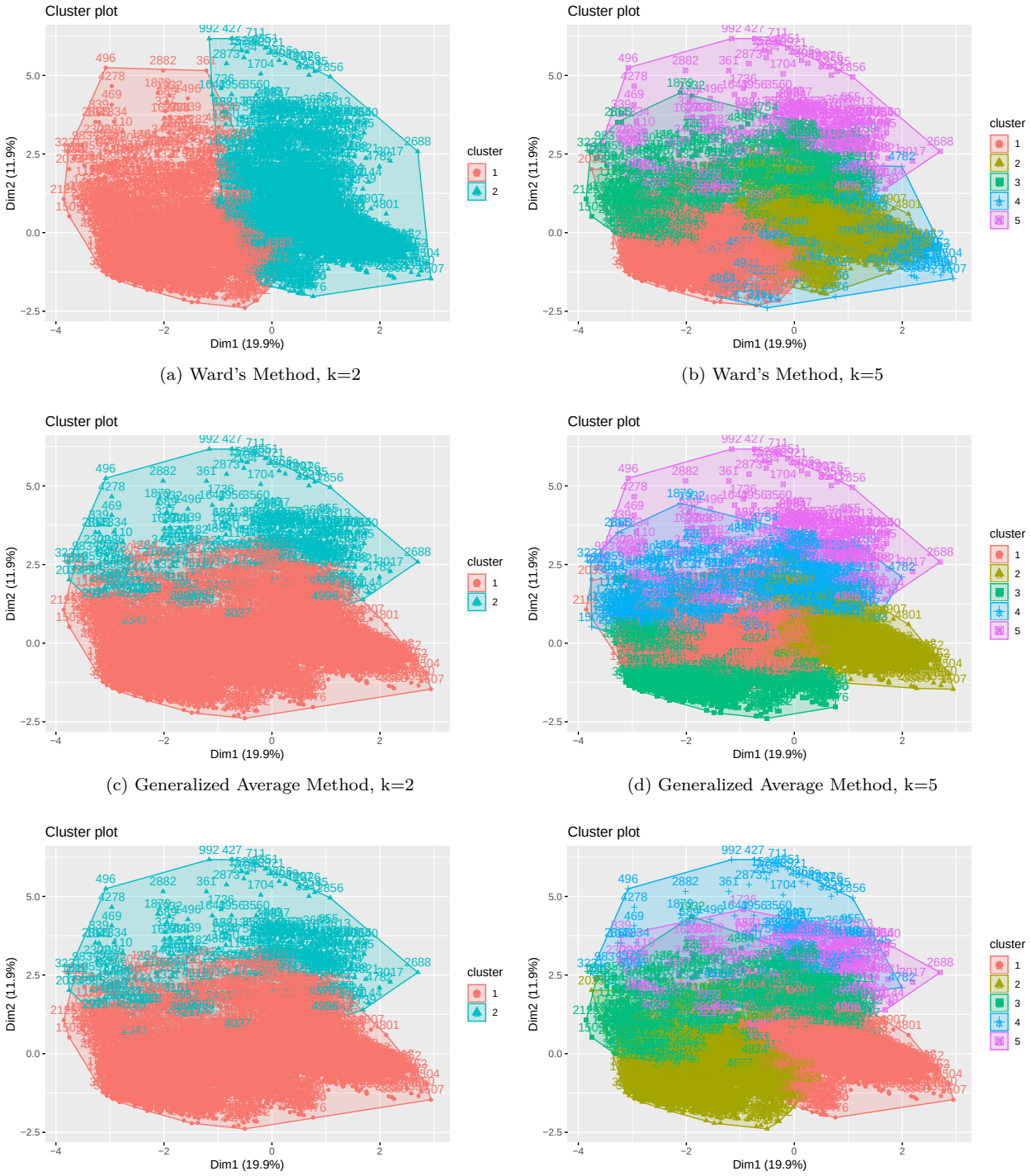


(a) k=2 (b) k=4

(c) k=5 (d) k=12

Figure 17: Results of the k-means clustering for all 4 randomly initialized cluster options. A PCA was produced and the clusterings were projected onto the first 2 principal components in order to visualize them.

```
fviz_cluster(k_clus_ward_2, data = data_matrix_scaled)
fviz_cluster(k_clus_ward_5, data = data_matrix_scaled)
fviz_cluster(k_clus_gave_2, data = data_matrix_scaled)
fviz_cluster(k_clus_gave_5, data = data_matrix_scaled)
fviz_cluster(k_clus_diana_2, data = data_matrix_scaled)
fviz_cluster(k_clus_diana_5, data = data_matrix_scaled)
```



(a) Ward's Method, k=2

(b) Ward's Method, k=5

(c) Generalized Average Method, k=2

(d) Generalized Average Method, k=5

(e) DIANA Method, k=2

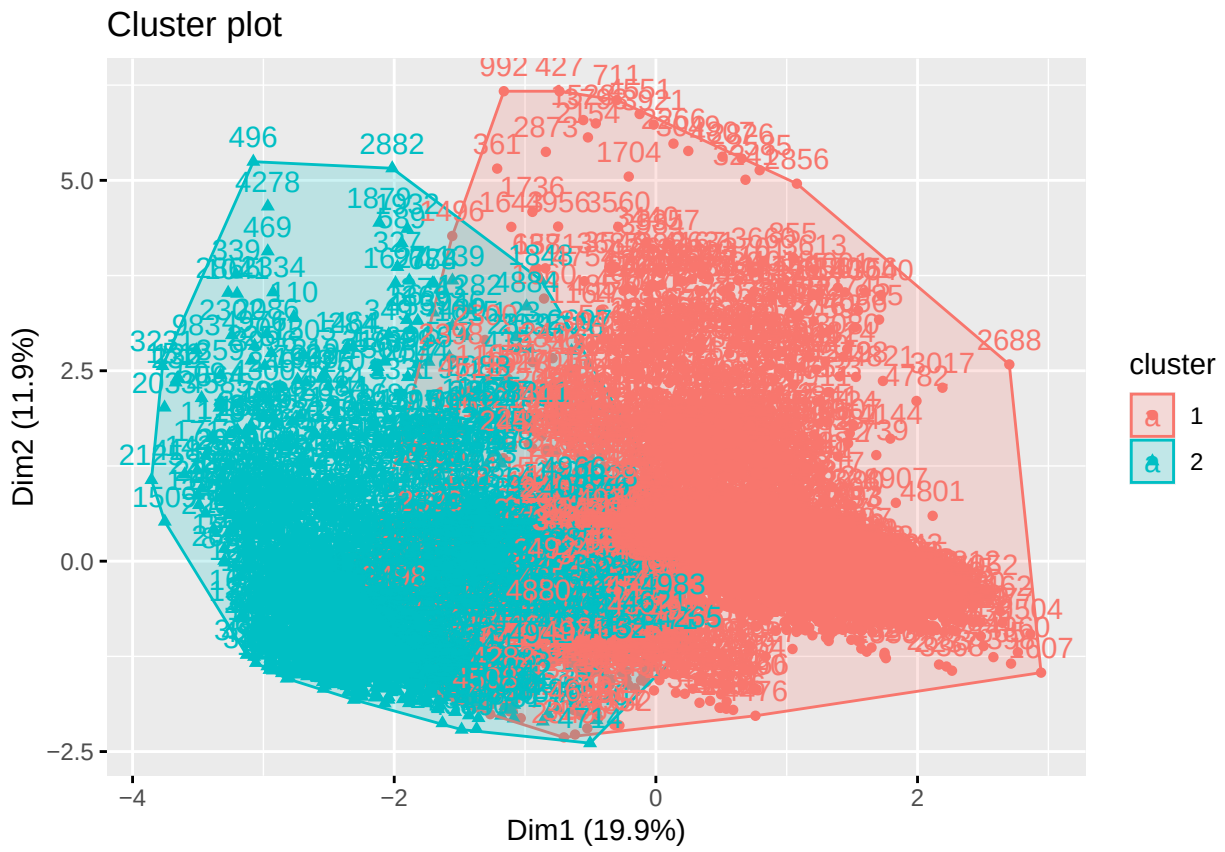
(f) DIANA Method, k=5

Figure 18: Results of the k-means clustering for all 6 hierarchical clustering-determined cluster options. A PCA was produced and the clusterings were projected onto the first 2 principal components in order to visualize them.

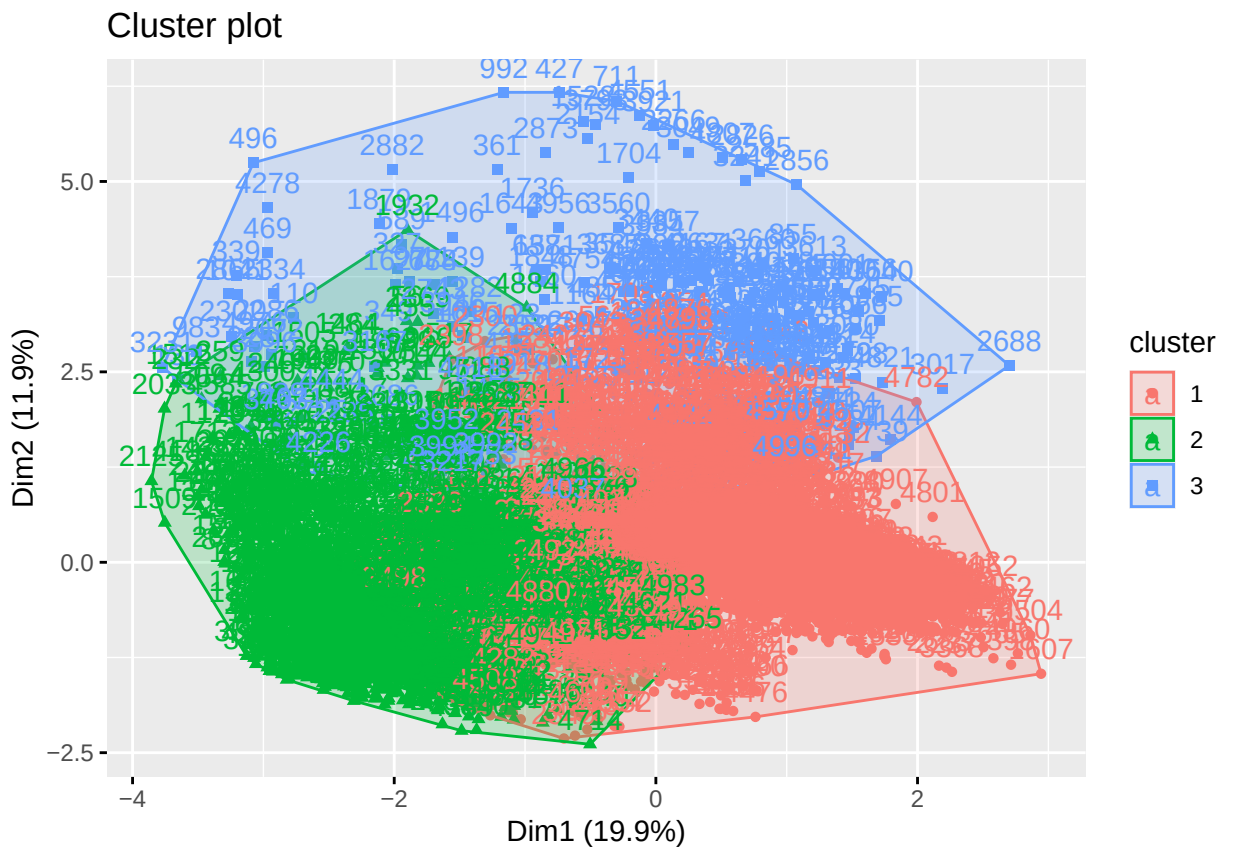
4.6 k-Medoids

```
clara_2 <- clara(data_matrix_clus, k = 2, metric = "euclidean", stand = TRUE, samples = 1000,
    pamLike = TRUE)
clara_3 <- clara(data_matrix_clus, k = 3, metric = "euclidean", stand = TRUE, samples = 1000,
    pamLike = TRUE)
clara_4 <- clara(data_matrix_clus, k = 4, metric = "euclidean", stand = TRUE, samples = 1000,
    pamLike = TRUE)
clara_5 <- clara(data_matrix_clus, k = 5, metric = "euclidean", stand = TRUE, samples = 1000,
    pamLike = TRUE)
clara_8 <- clara(data_matrix_clus, k = 8, metric = "euclidean", stand = TRUE, samples = 1000,
    pamLike = TRUE)
clara_12 <- clara(data_matrix_clus, k = 12, metric = "euclidean", stand = TRUE, samples = 1000,
    pamLike = TRUE)
```

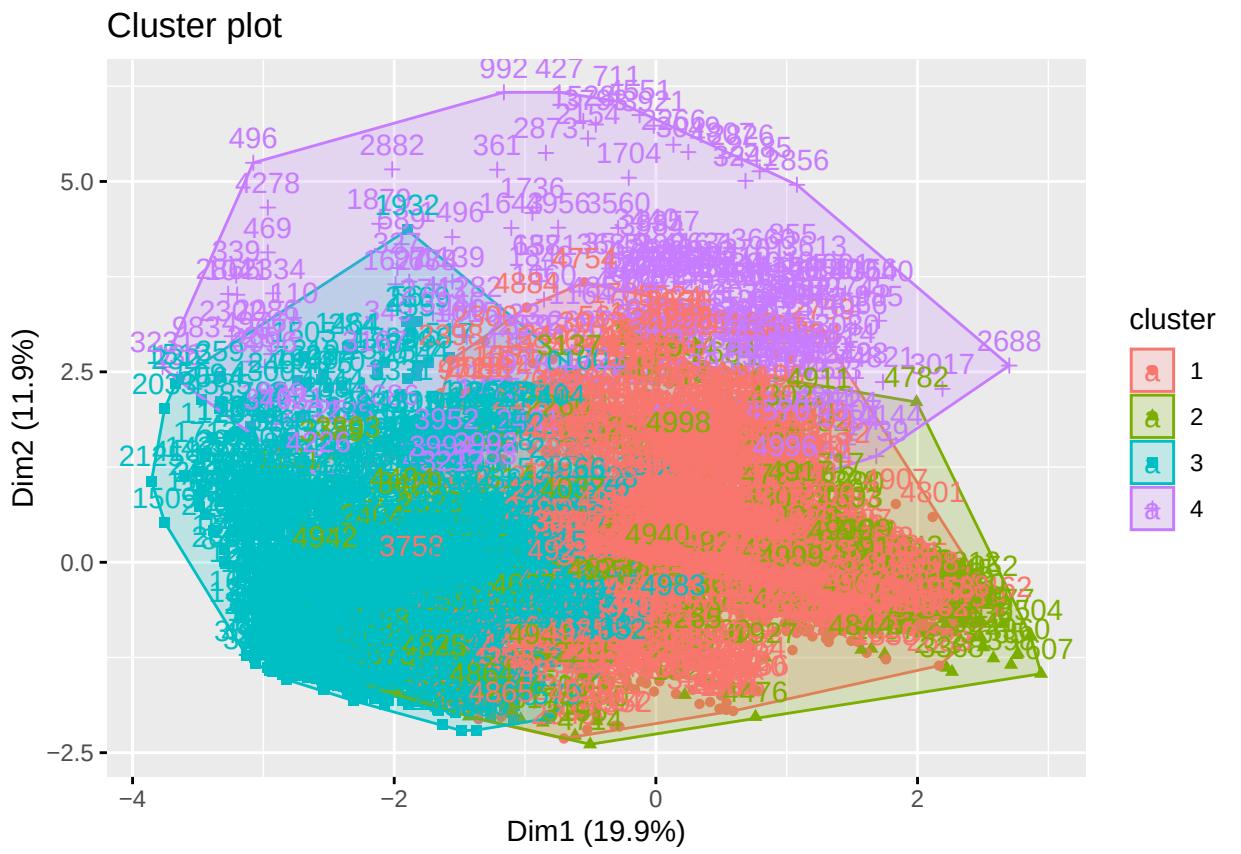
```
print(clara_2)
fviz_cluster(clara_2, data = data_matrix_clus)
```



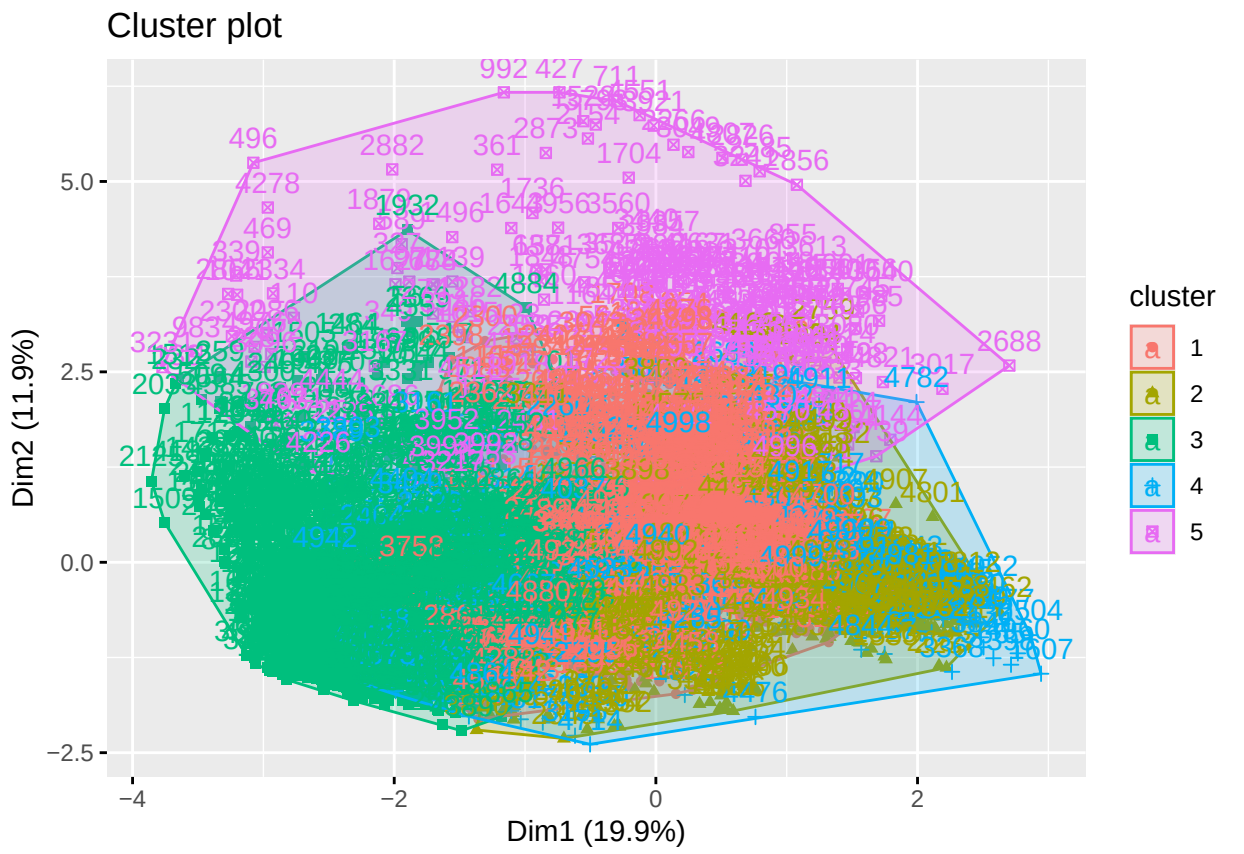

```
fviz_cluster(clara_3, data = data_matrix_clus)
```



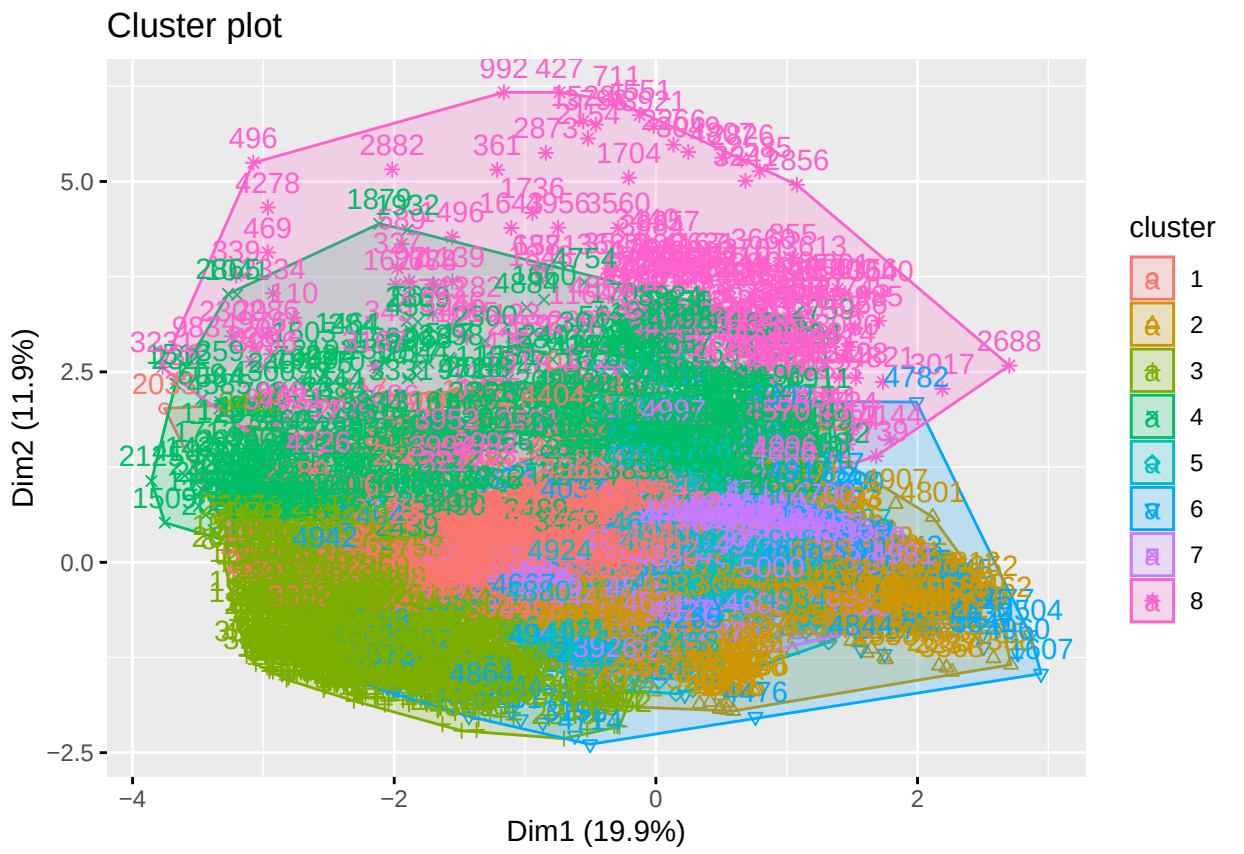

```
fviz_cluster(clara_4, data = data_matrix_clus)
```



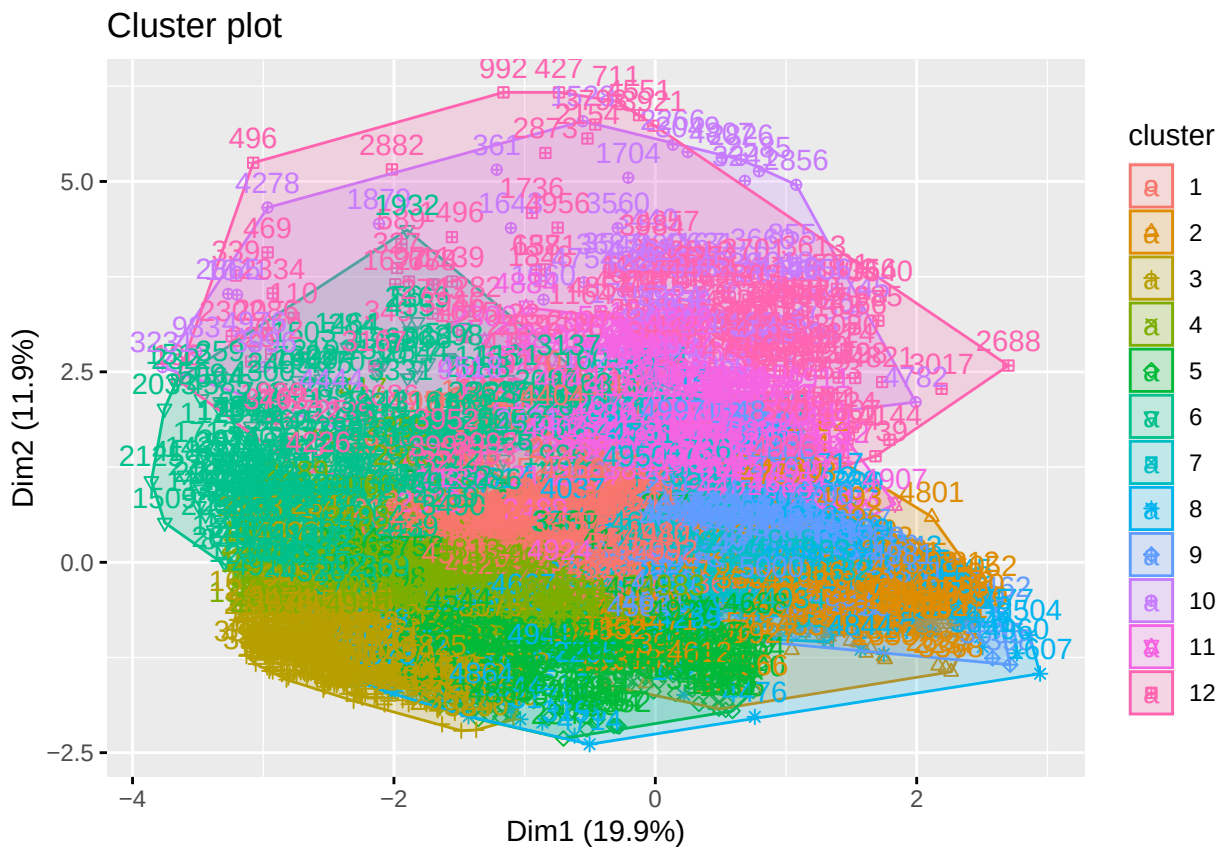

```
fviz_cluster(clara_5, data = data_matrix_clus)
```




```
fviz_cluster(clara_8, data = data_matrix_clus)
```




```
fviz_cluster(clara_12, data = data_matrix_clus)
```



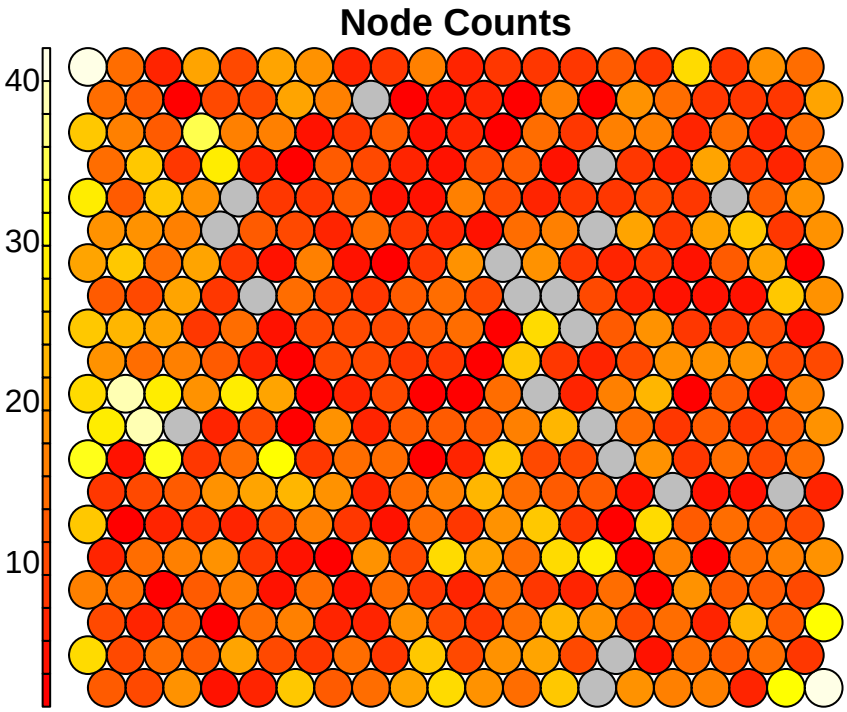

```
## Call:     clara(x = data_matrix_clus, k = 2, metric = "euclidean", stand = TRUE,      samples = 1000
## Medoids:
##              age  workclass education.num marital.status occupation
## [1,]  0.06895132 -0.1515961    -0.2494211     -0.2881045 0.01340856
## [2,] -1.15050105 -0.1515961    -0.2494211      1.1848224 0.26079898
##      relationship      race        sex hours.per.week native.country
## [1,]   -0.7329173 0.3872955  0.6000851     0.01313775      0.2718538
## [2,]    1.1058114 0.3872955 -1.6660972    -0.45944324      0.2718538
## Objective function:   3.810104
## Clustering vector:    int [1:5000] 1 1 2 1 2 2 2 2 2 2 1 2 2 2 1 1 1 1 ...
## Cluster sizes:            3570 1430
## Best sample:
##  [1]  174  827  885  977 1065 1108 1150 1242 1247 1326 1445 1595 1848 1860 1891
## [16] 1938 2123 2209 2212 2256 2464 2483 2566 2580 2906 3120 3191 3245 3395 3676
## [31] 3866 3979 4122 4149 4263 4324 4543 4600 4658 4679 4765 4775 4902 4967
##
## Available components:
##  [1] "sample"     "medoids"    "i.med"      "clustering" "objective"
##  [6] "clusinfo"   "diss"       "call"       "silinfo"    "data"
```

4.7 Self-Organizing maps (SOM)

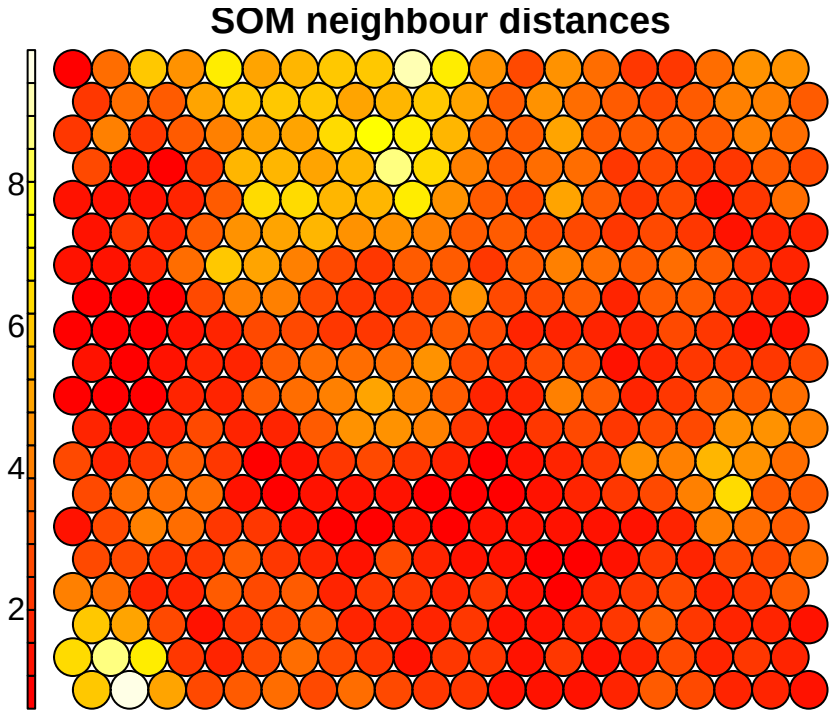
```
data_matrix_s <- data_matrix_clus
som_grid <- somgrid(xdim = 20, ydim = 20, topo = "hexagonal")
som_model <- som(data_matrix_s, grid = som_grid, rlen = 200, keep.data = TRUE)
som_codebook = getCodes(som_model)
```

```r
# Training progress for SOM
plot(som_model, type = "changes")
```


```r
# Node count plot
plot(som_model, type = "count", main = "Node Counts")
```



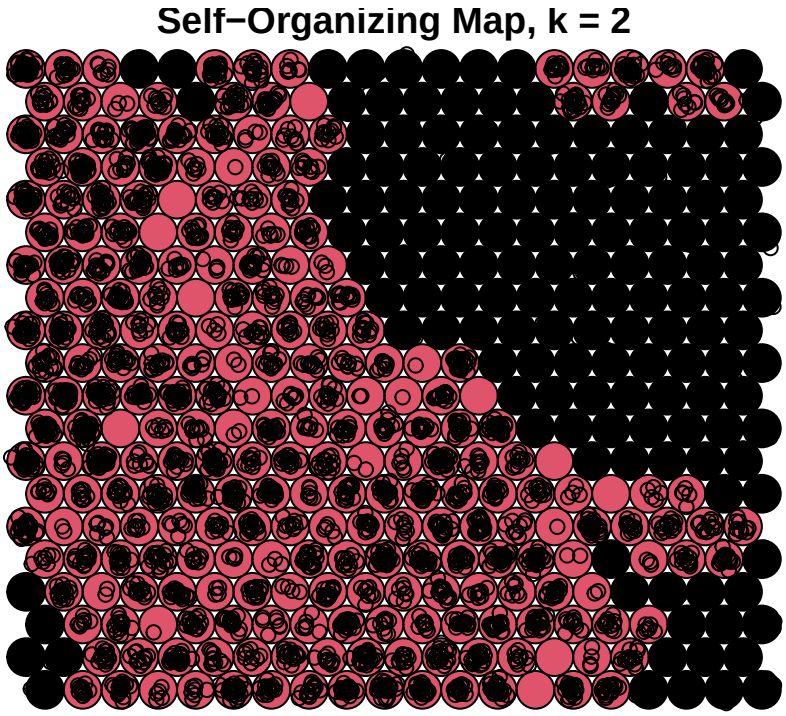

```
# U-matrix visualisation
plot(som_model, type = "dist.neighbours", main = "SOM neighbour distances")
```



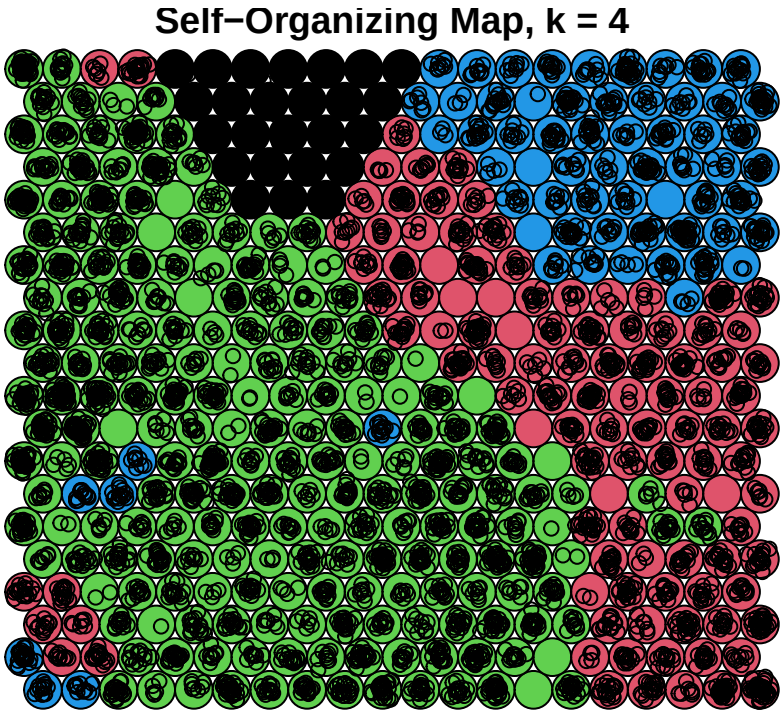

```
k_clus_som_2 <- kmeans(scale(som_codebook), centers = 2, nstart = 500)
k_clus_som_4 <- kmeans(scale(som_codebook), centers = 4, nstart = 500)
k_clus_som_5 <- kmeans(scale(som_codebook), centers = 5, nstart = 500)
k_clus_som_12 <- kmeans(scale(som_codebook), centers = 12, nstart = 500)
```

```
## Warning: did not converge in 10 iterations
```

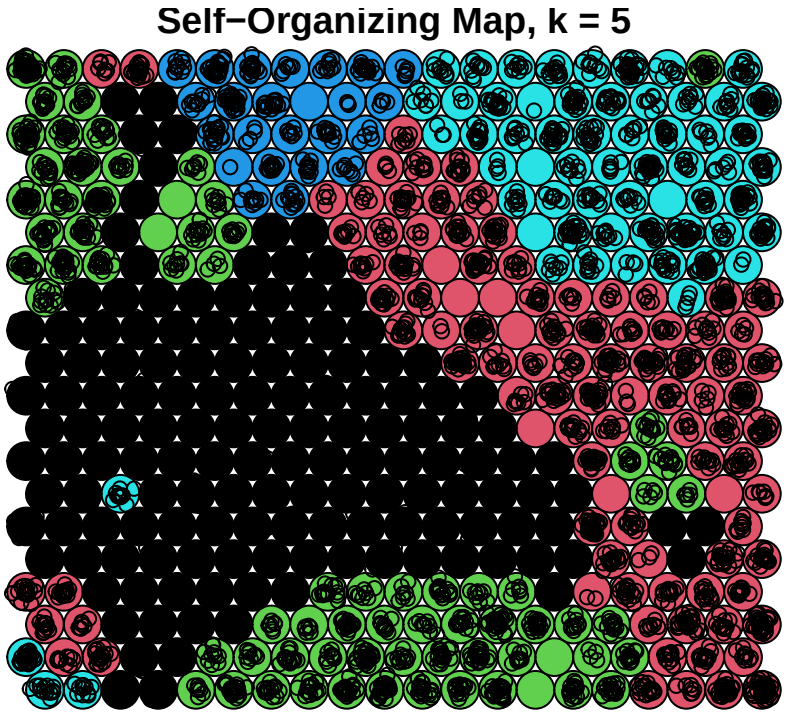
```
plot(som_model, type = "mapping", bgcol = k_clus_som_2$cluster, main = "Self-Organizing Map, k = 2")
```



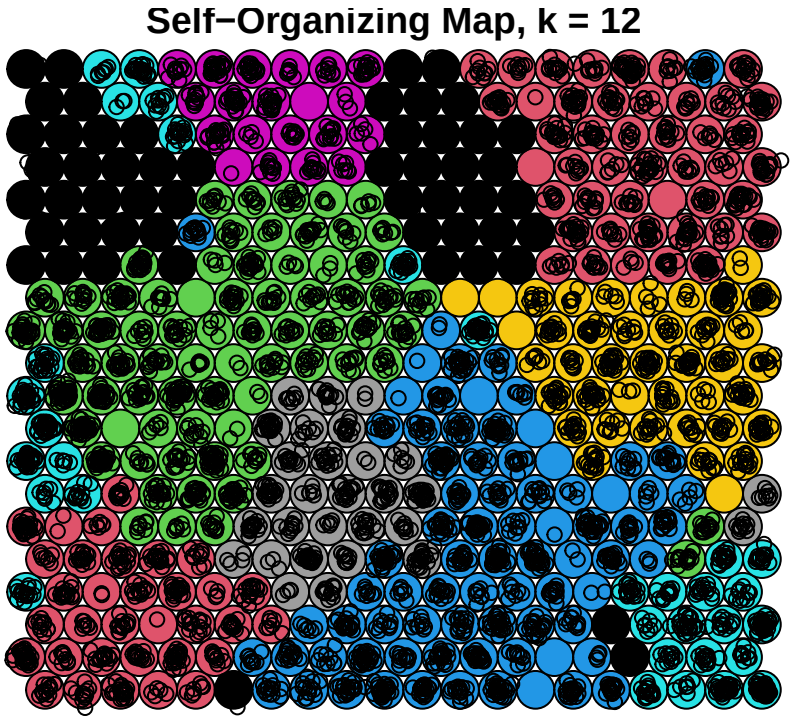

```
plot(som_model, type = "mapping", bgcol = k_clus_som_4$cluster, main = "Self-Organizing Map, k = 4")
```




```
plot(som_model, type = "mapping", bgcol = k_clus_som_5$cluster, main = "Self-Organizing Map, k = 5")
```




```
plot(som_model, type = "mapping", bgcol = k_clus_som_12$cluster, main = "Self-Organizing Map, k = 12")
```



4.8 Validation of Clusterings

4.8.1 Stability Testing

We first perform the nonparametric bootstrap test for our k-means clustering result for 100 bootstraps and k = 2, 4, 5, and 12 (the best options from the previous results). The results for each option are shown below.

```
k_clus_2_boot <- clusterboot(data_matrix_scaled, B=100, bootmethod="boot",
                         clustermethod=kmeansCBI,
                         krange=2, seed=786)

k_clus_4_boot <- clusterboot(data_matrix_scaled, B=100, bootmethod="boot",
                         clustermethod=kmeansCBI,
                         krange=4, seed=786)

k_clus_5_boot <- clusterboot(data_matrix_scaled, B=100, bootmethod="boot",
                         clustermethod=kmeansCBI,
                         krange=5, seed=786)

k_clus_12_boot <- clusterboot(data_matrix_scaled, B=100, bootmethod="boot",
                         clustermethod=kmeansCBI,
                         krange=12, seed=786)
```

4.8.2 K-medoid

```
clara_2_boot <- clusterboot(data_matrix_scaled, B = 100, bootmethod = "boot", clustermethod = claraCBI,
    k = 2, seed = 786)
clara_3_boot <- clusterboot(data_matrix_scaled, B = 100, bootmethod = "boot", clustermethod = claraCBI,
    k = 3, seed = 786)
clara_4_boot <- clusterboot(data_matrix_scaled, B = 100, bootmethod = "boot", clustermethod = claraCBI,
    k = 4, seed = 786)
clara_5_boot <- clusterboot(data_matrix_scaled, B = 100, bootmethod = "boot", clustermethod = claraCBI,
    k = 5, seed = 786)
clara_8_boot <- clusterboot(data_matrix_scaled, B = 100, bootmethod = "boot", clustermethod = claraCBI,
    k = 8, seed = 786)
clara_12_boot <- clusterboot(data_matrix_scaled, B = 100, bootmethod = "boot", clustermethod = claraCBI
    k = 12, seed = 786)
```

For k = 2,

```
print(k_clus_2_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  kmeans
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  2
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
## [1] 0.8837559 0.9369528
## dissolved:
## [1] 11  1
## recovered:
## [1] 89 89
```

For k = 4,

```
print(k_clus_4_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  kmeans
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  4
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
## [1] 0.6790478 0.4835833 0.2922057 0.5392781
## dissolved:
## [1] 31 70 79 61
## recovered:
## [1] 38 25 20 25
```

For k = 5,

```r
print(k_clus_5_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  kmeans
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  5
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
## [1] 0.6053753 0.5731961 0.5071577 0.3451179 0.5267891
## dissolved:
## [1] 50 43 53 68 52
## recovered:
## [1] 25 40 46 20  7
```

For k = 12,

```r
print(k_clus_12_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  kmeans
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  12
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
##  [1] 0.4350039 0.6284001 0.8123564 0.8348095 0.5325379 0.8279439 0.5173669
##  [8] 0.4777071 0.7506903 0.5116385 0.4967474 0.7262115
## dissolved:
##  [1] 74 46 12  9 52  5 60 62  8 62 71 27
## recovered:
##  [1]  4 43 82 78 17 88 15  2 58 20 11 56
```

BOOTSTRAP FOR K-MEDOID

For k = 2,

```r
print(clara_2_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  clara/pam
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  2
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
## [1] 0.9749886 0.9360996
## dissolved:
```

```
## [1] 0 0
## recovered:
## [1] 100 100
```

For k = 4,

```
print(clara_4_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  clara/pam
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  4
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
## [1] 0.6647225 0.7236842 0.7292943 0.7227594
## dissolved:
## [1] 40 28 19 13
## recovered:
## [1] 55 53 55 42
```

For k = 5,

```
print(clara_5_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  clara/pam
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  5
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
## [1] 0.6472312 0.6863078 0.6295850 0.7018961 0.3733538
## dissolved:
## [1] 11 33 33 16 67
## recovered:
## [1] 27 56 33 48 33
```

For k = 12,

```
print(clara_12_boot)
```

```
## * Cluster stability assessment *
## Cluster method:  clara/pam
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
## of the resampling results.
## Number of resampling runs:  100
##
## Number of clusters found in data:  12
##
##  Clusterwise Jaccard bootstrap (omitting multiple points) mean:
```

```
##  [1] 0.9239595 0.7929994 0.8224148 0.8770152 0.7542200 0.7057239 0.8368959
##  [8] 0.7223946 0.9555502 0.8566016 0.9335859 0.7843670
## dissolved:
##  [1]  0  6  1  7 18 12  4  9  0  0  0 21
## recovered:
##  [1] 100  70  52  93  51  29  77  39 100  79 100  70
```

Then, for APN, AD, ADM, and FOM, we use clValid(), which also conveniently does our internal validity tests as well. Below, we show the combined summary for APN, AD, ADM, FOM, and the internal validation measurements, Connectivity, Silhouette Width, and the Dunn Index. Figure 19 graphically shows the results of the APN, AD, ADM, and FOM measurements.

```
multi_valid_test <- clValid(data_matrix_scaled, c(2, 3, 4, 5, 8, 12),
                      clMethods = c("agnes", "diana", "kmeans", "clara", "som" ),
                      validation = c("internal", "stability"),
                      maxitems = 50000,
                      method = "ward"
)
```

```
summary(multi_valid_test)
```

```
##
## Clustering Methods:
##  agnes diana kmeans clara som
##
## Cluster sizes:
##  2 3 4 5 8 12
##
## Validation Measures:
##                              2         3         4         5         8        12
##
## agnes  APN              0.2340    0.3048    0.2887    0.3488    0.3681    0.3415
##        AD               4.1085    4.0066    3.8721    3.7875    3.4647    3.2281
##        ADM              0.7146    0.8781    0.8832    1.0761    1.0622    1.0194
##        FOM              0.9922    0.9600    0.9589    0.9561    0.9490    0.9387
##        Connectivity   271.4889  329.0238  354.1786  447.9853  637.0702  780.1127
##        Dunn             0.0376    0.0436    0.0436    0.0451    0.0296    0.0099
##        Silhouette       0.1457    0.1558    0.1623    0.1601    0.1291    0.1168
## diana  APN              0.0236    0.0718    0.1394    0.1072    0.2052    0.2905
##        AD               4.1263    3.9520    3.9359    3.7079    3.5417    3.3725
##        ADM              0.0855    0.2416    0.4576    0.5247    0.6841    0.9466
##        FOM              0.9975    0.9943    0.9937    0.9646    0.9541    0.9439
##        Connectivity   113.8706  176.9694  188.0202  234.7647  339.2877 1049.6052
##        Dunn             0.0655    0.0684    0.0694    0.0709    0.0425    0.0113
##        Silhouette       0.3346    0.2086    0.2063    0.2170    0.2011    0.1249
## kmeans APN              0.0638    0.2098    0.3723    0.2508    0.2028    0.2682
##        AD               3.9368    3.8787    3.8670    3.6127    3.2332    3.0501
##        ADM              0.2202    0.7418    1.2625    0.7810    0.6838    0.7956
##        FOM              0.9555    0.9578    0.9508    0.9527    0.9526    0.9410
##        Connectivity   388.5226  491.4885  653.8663  653.8488 1091.8607 1147.7940
##        Dunn             0.0346    0.0247    0.0163    0.0197    0.0281    0.0104
##        Silhouette       0.1998    0.2012    0.1632    0.1772    0.1531    0.1669
## clara  APN              0.1564    0.2223    0.3185    0.4646    0.5162    0.3914
##        AD               4.0246    3.8753    3.8201    3.8867    3.6383    3.2665
##        ADM              0.4935    0.5868    0.9282    1.4024    1.4751    1.0889
```

```
##        FOM              0.9806    0.9652    0.9602    0.9527    0.9543    0.9478
##        Connectivity   520.6246 1087.5349 1050.2004 1914.2984 1843.1417 1729.3571
##        Dunn             0.0167    0.0087    0.0089    0.0090    0.0097    0.0104
##        Silhouette       0.2006    0.1146    0.1294    0.1065    0.1073    0.1490
## som    APN              0.1004    0.1851    0.2554    0.2393    0.2396    0.3362
##        AD               3.9598    3.9145    3.7557    3.5686    3.3073    3.1130
##        ADM              0.3535    0.8174    0.8097    0.6890    0.8184    0.9517
##        FOM              0.9567    0.9610    0.9635    0.9545    0.9435    0.9458
##        Connectivity   428.5484  460.8968  600.5063  829.0861 1033.4306 1294.4687
##        Dunn             0.0251    0.0175    0.0263    0.0197    0.0104    0.0211
##        Silhouette       0.1985    0.1878    0.1388    0.1503    0.1588    0.1665
##
## Optimal Scores:
##
##              Score    Method Clusters
## APN            0.0236 diana  2
## AD             3.0501 kmeans 12
## ADM            0.0855 diana  2
## FOM            0.9387 agnes  12
## Connectivity 113.8706 diana  2
## Dunn           0.0709 diana  5
## Silhouette     0.3346 diana  2
```

```r
plot(multi_valid_test, measure=c("APN"), legend=FALSE, lwd = 3.5)
legend("bottomright", clusterMethods(multi_valid_test),col=1:5,
       lty=1:5, pch=paste(1:5), lwd = 3.5)

plot(multi_valid_test, measure=c("AD"), legend=FALSE, lwd = 3.5)
legend("bottomleft", clusterMethods(multi_valid_test), col=1:5,
       lty=1:5, pch=paste(1:5), lwd = 3.5)

plot(multi_valid_test, measure=c("ADM"), legend=FALSE, lwd = 3.5)
legend("bottomright", clusterMethods(multi_valid_test), col=1:5,
       lty=1:5, pch=paste(1:5), lwd = 3.5)

plot(multi_valid_test, measure=c("FOM"), legend=FALSE, lwd = 3.5)
legend("bottomleft", clusterMethods(multi_valid_test), col=1:5,
       lty=1:5, pch=paste(1:5), lwd = 3.5)
```

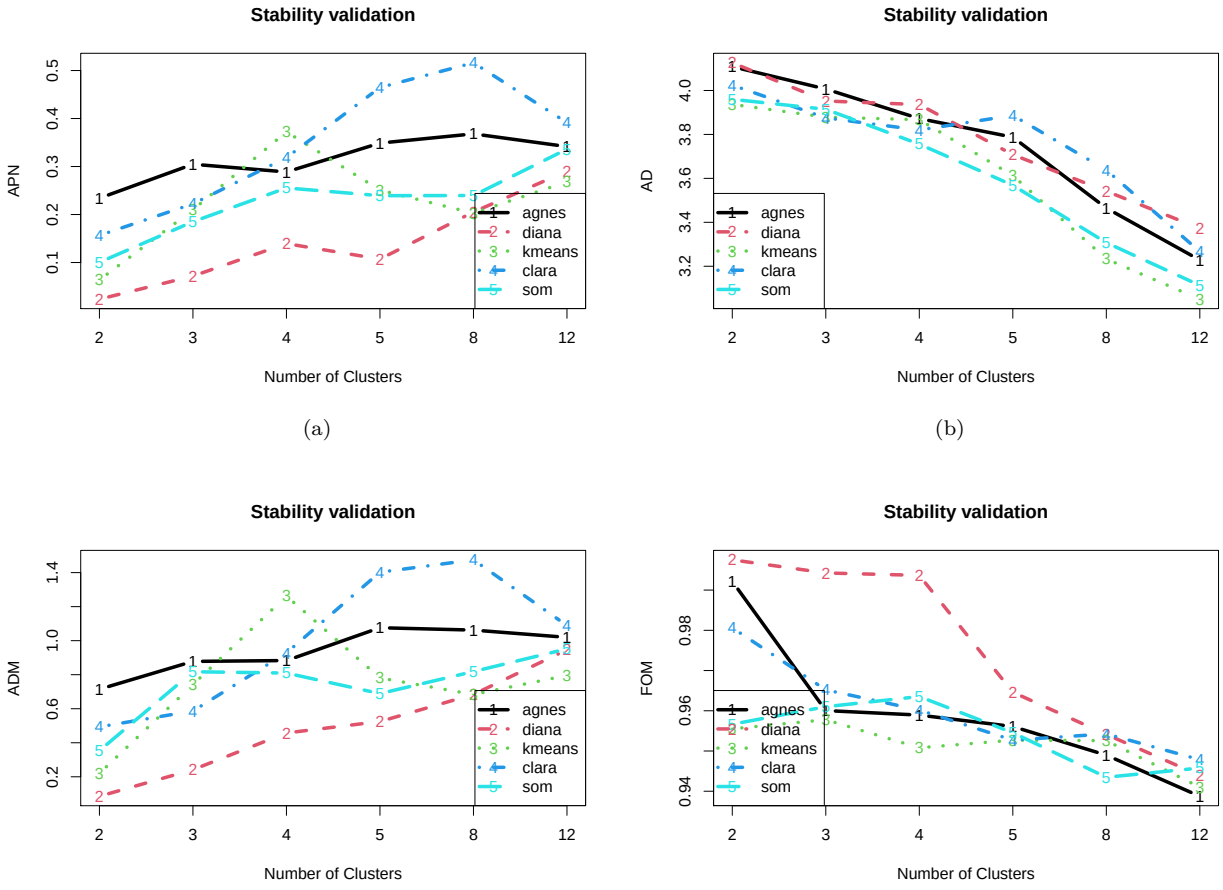
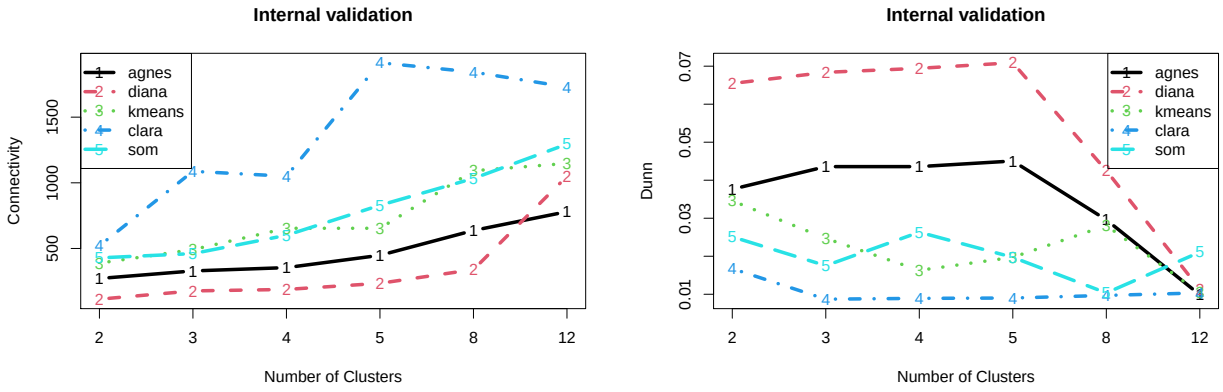


Figure 19: Results of the stability validation tests for all the specified clustering methods and k values.

4.8.3 Internal Validation

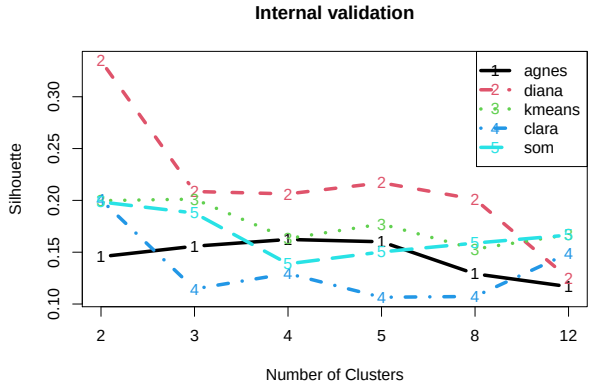
The results of the Connectivity, Silhouette Width, and the Dunn Index measurements were given above. Figure 20 graphically shows their results. In addition, since k = 2 seems to be best clustering option (which is also the original number of labels), we plot the Silhouette figures for all of our k = 2 clusterings, both hierarchical, Figure 21, and k-means, Figure 22. In addition, since k-means clustering with k = 12 was the best result for the AD and FOM measurements, we plot its Silhouette plot as well, in Figure 23.

```
plot(multi_valid_test, measure=c("Connectivity"), legend=FALSE, lwd = 3.5)
legend("topleft", clusterMethods(multi_valid_test), col=1:5, lty=1:5,
       pch=paste(1:5), lwd = 3.5)

plot(multi_valid_test, measure=c("Dunn"), legend=FALSE, lwd = 3.5)
legend("topright", clusterMethods(multi_valid_test), col=1:5, lty=1:5,
       pch=paste(1:5), lwd = 3.5)

plot(multi_valid_test, measure=c("Silhouette"), legend=FALSE, lwd = 3.5)
legend("topright", clusterMethods(multi_valid_test), col=1:5, lty=1:5,
       pch=paste(1:5), lwd = 3.5)
```



(a)

(b)



(c)

Figure 20: Results of the Internal validation tests for all the specified clustering methods and k values.

```
# hierarchical

sil_heir_diana_2 <- silhouette(cut_diana_2, dist_matrix_scaled)
sil_heir_ward_2 <- silhouette(cut_ward_2, dist_matrix_scaled)
sil_heir_gave_2 <- silhouette(cut_gave_2, dist_matrix_scaled)

noquote("Cluster Sizes and Silhouette values for DIANA, Ward's Method, and ")
noquote("Generalized Average hierarchical clusterings, respectively:")

fviz_silhouette(sil_heir_diana_2, print.summary = TRUE,
                main ="Silhouette Plot (Hierarchical Clustering, DIANA, k = 2)")
fviz_silhouette(sil_heir_ward_2, print.summary = TRUE,
                main ="Silhouette Plot (Hierarchical Clustering, Ward's Method, k = 2)")
fviz_silhouette(sil_heir_gave_2, print.summary = TRUE,
                main ="Silhouette Plot (Hierarchical Clustering, Generalized Avg., k = 2)")
```

```
## [1] Cluster Sizes and Silhouette values for DIANA, Ward's Method, and
## [1] Generalized Average hierarchical clusterings, respectively:
##   cluster size ave.sil.width
## 1       1 4759          0.34
```

```
## 2       2  241          0.24
##   cluster size ave.sil.width
## 1       1 2114          0.00
## 2       2 2886          0.25
##   cluster size ave.sil.width
## 1       1 4788          0.35
## 2       2  212          0.26
```

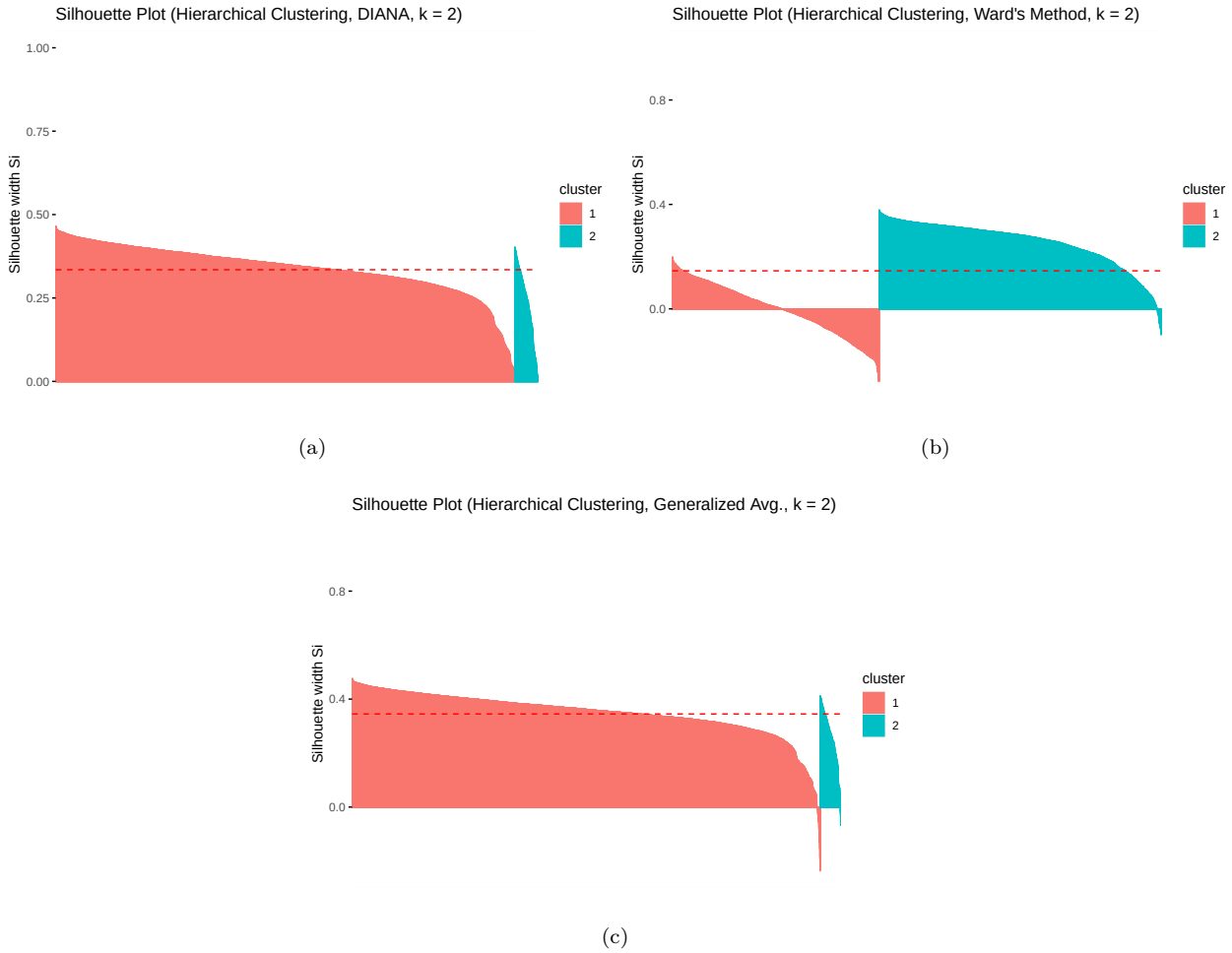
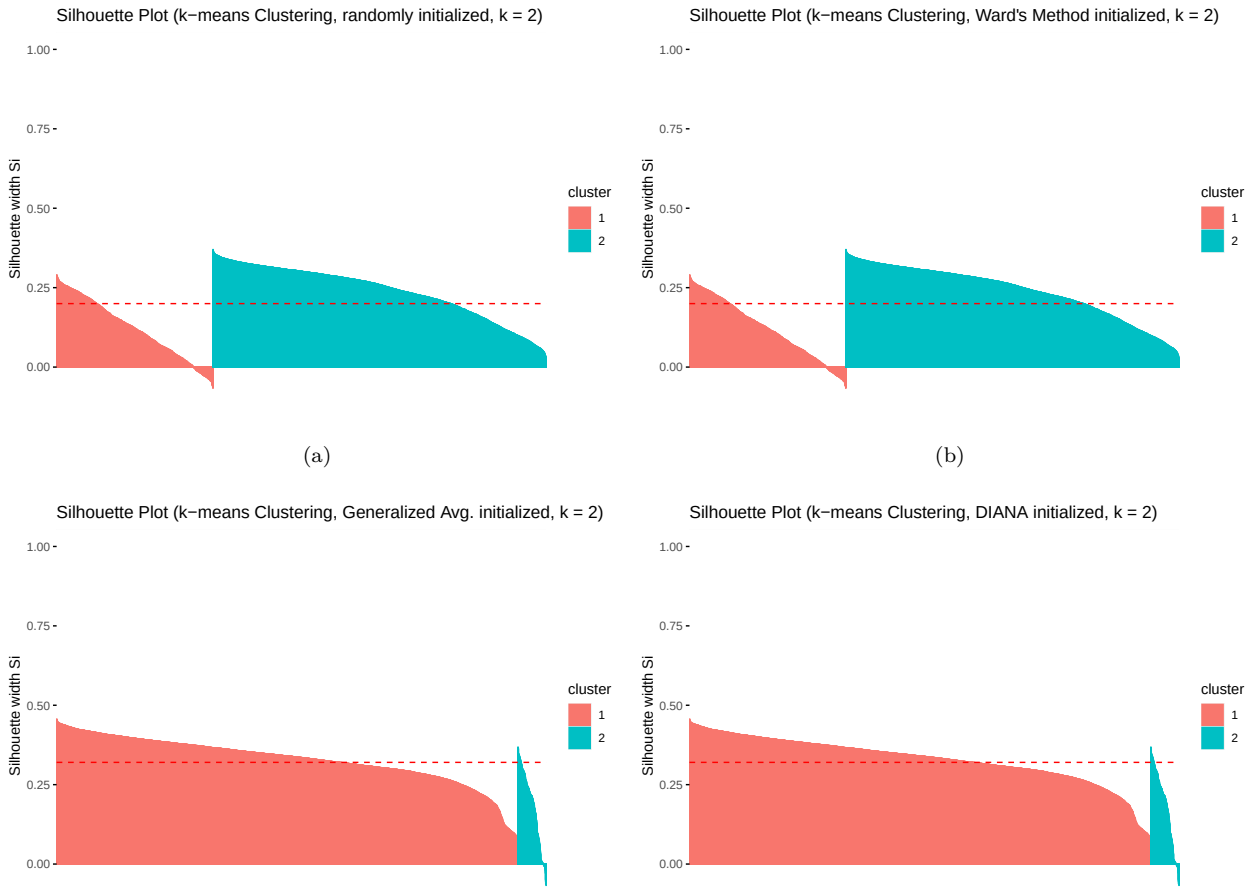


Figure 21: Silhouette plots of all the k = 2 hierarchical clusters we have generated.

```
# k-means

sil_k_rand_2 <- silhouette(k_clus_2$cluster, dist_matrix_scaled)
sil_k_ward_2 <- silhouette(k_clus_ward_2$cluster, dist_matrix_scaled)
sil_k_gave_2 <- silhouette(k_clus_gave_2$cluster, dist_matrix_scaled)
sil_k_diana_2 <- silhouette(k_clus_diana_2$cluster, dist_matrix_scaled)

noquote("Cluster Sizes and Silhouette values for the randomly initialized and Ward's")
noquote("Method, Generalized Average, and DIANA hierarchical clustering-initialized")
noquote("k-means clusters, respectively:")

fviz_silhouette(sil_k_rand_2, print.summary = TRUE,
```

```
                main ="Silhouette Plot (k-means Clustering, randomly initialized, k = 2)")
fviz_silhouette(sil_k_ward_2, print.summary = TRUE,
          main ="Silhouette Plot (k-means Clustering, Ward's Method initialized, k = 2)")
fviz_silhouette(sil_k_gave_2, print.summary = TRUE,
        main ="Silhouette Plot (k-means Clustering, Generalized Avg. initialized, k = 2)")
fviz_silhouette(sil_k_diana_2, print.summary = TRUE,
                main ="Silhouette Plot (k-means Clustering, DIANA initialized, k = 2)")
```

```
## [1] Cluster Sizes and Silhouette values for the randomly initialized and Ward's
## [1] Method, Generalized Average, and DIANA hierarchical clustering-initialized
## [1] k-means clusters, respectively:
##   cluster size ave.sil.width
## 1       1 1597          0.12
## 2       2 3403          0.24
##   cluster size ave.sil.width
## 1       1 1597          0.12
## 2       2 3403          0.24
##   cluster size ave.sil.width
## 1       1 4711          0.33
## 2       2  289          0.18
##   cluster size ave.sil.width
## 1       1 4711          0.33
## 2       2  289          0.18
```



(a) (b) (c) (d)

Figure 22: Silhouette plots of all the k = 2 k-means clusters we have generated.

```
sil_k_rand_12 <- silhouette(k_clus_12$cluster, dist_matrix_scaled)

noquote("Cluster Size and Silhouette value for the randomly initialized k-means cluster, k=12.")
fviz_silhouette(sil_k_rand_12, print.summary = TRUE,
                main ="Silhouette Plot (k-means Clustering, randomly initialized, k = 12)")
```

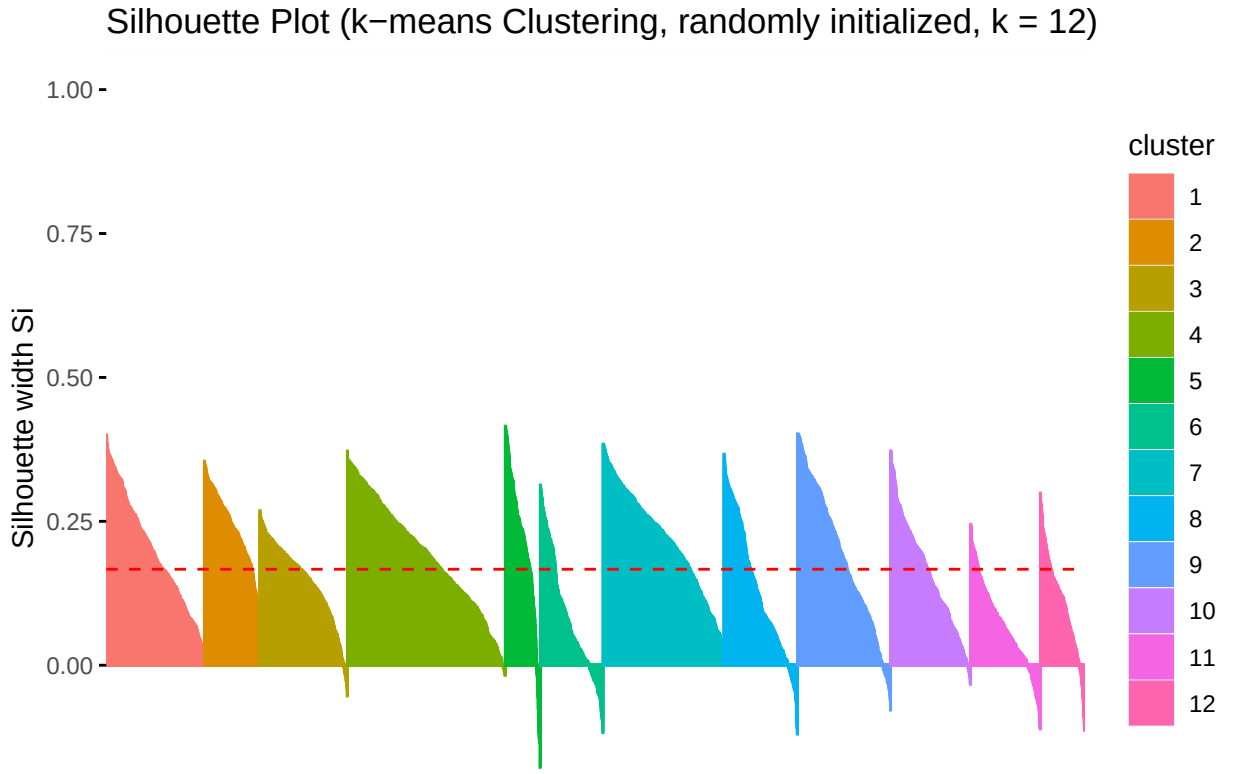


Figure 23: Silhouette plot for the randomly initialized k = 12 k-means cluster.

```
## [1] Cluster Size and Silhouette value for the randomly initialized k-means cluster, k=12.
##    cluster size ave.sil.width
## 1        1  501          0.20
## 2        2  284          0.24
## 3        3  447          0.15
## 4        4  807          0.19
## 5        5  179          0.21
## 6        6  321          0.08
## 7        7  618          0.22
## 8        8  376          0.13
## 9        9  476          0.18
## 10      10  409          0.16
## 11      11  356          0.07
## 12      12  226          0.11
```

4.8.4 External

Finally, we perform external validation and determine the Rand Index for all of our k = 2 clusterings. First, we setup our original label clustering:

```
orig_cluster <- data.matrix(clean_sample)
orig_cluster <- orig_cluster[, c(11)]

sample_frame <- data.frame(sample)
```

Then we show the clustering size comparison with the original class size and the Rand Index for each k = 2 clustering, with a summary shown in Table 9:

DIANA hierarchical clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, cut_diana_2)
ext_stats_hier_diana_2 <- cluster.stats(d = dist_matrix_scaled, orig_cluster, cut_diana_2)
randInd_hier_diana_2 <- ext_stats_hier_diana_2$corrected.rand
print(paste0("Rand Index for DIANA Hierarchical Clustering: ", randInd_hier_diana_2))
```

```
## [1] Cluster Size Comparison:
##                    cut_diana_2
##                        1    2
##   -0.9998999949995 2403   97
##   0.9998999949995  2356  144
## [1] "Rand Index for DIANA Hierarchical Clustering: 0.000316742877485066"
```

Ward's method hierarchical clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, cut_ward_2)
ext_stats_hier_ward_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster, cut_ward_2)
randInd_hier_ward_2 <- ext_stats_hier_ward_2$corrected.rand
print(paste0("Rand Index for Ward's Method Hierarchical Clustering: ",
             randInd_hier_ward_2))
```

```
## [1] Cluster Size Comparison:
##                    cut_ward_2
##                        1    2
##   -0.9998999949995 1423 1077
##   0.9998999949995   691 1809
## [1] "Rand Index for Ward's Method Hierarchical Clustering: 0.0855532749068989"
```

Generalized Average method hierarchical clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, cut_gave_2)
ext_stats_hier_gave_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster, cut_gave_2)
randInd_hier_gave_2 <- ext_stats_hier_gave_2$corrected.rand
print(paste0("Rand Index for GAveraged Method Hierarchical Clustering: ",
             randInd_hier_gave_2))
```

```
## [1] Cluster Size Comparison:
##                    cut_gave_2
##                        1    2
##   -0.9998999949995 2414   86
##   0.9998999949995  2374  126
## [1] "Rand Index for GAveraged Method Hierarchical Clustering: 0.000223518972091278"
```

k = 2, randomly initialized k-means clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, k_clus_2$cluster)
ext_stats_k_rand_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster, k_clus_2$cluster)
randInd_k_rand_2 <- ext_stats_k_rand_2$corrected.rand
print(paste0("Rand Index for the randomly initialized k-means clustering: ",
             randInd_k_rand_2))
```

```
## [1] Cluster Size Comparison:
##
##                        1    2
##   -0.9998999949995 1196 1304
##   0.9998999949995   401 2099
## [1] "Rand Index for the randomly initialized k-means clustering: 0.100967620779298"
```

k = 2, Ward's Method hierarchical clustering-initialized k-means clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, k_clus_ward_2$cluster)
ext_stats_k_ward_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster,
                                    k_clus_ward_2$cluster)
randInd_k_ward_2 <- ext_stats_k_ward_2$corrected.rand
print(paste0("Rand Index the Ward's Method-initialized k-means clustering: ",
             randInd_k_ward_2))
```

```
## [1] Cluster Size Comparison:
##
##                        1    2
##   -0.9998999949995 1196 1304
##   0.9998999949995   401 2099
## [1] "Rand Index the Ward's Method-initialized k-means clustering: 0.100967620779298"
```

k = 2, Generalized Average hierarchical clustering-initialized k-means clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, k_clus_gave_2$cluster)
ext_stats_k_gave_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster,
                                    k_clus_gave_2$cluster)
randInd_k_gave_2 <- ext_stats_k_gave_2$corrected.rand
print(paste0("Rand Index the GAveraged-initialized k-means clustering: ", randInd_k_gave_2))
```

```
## [1] Cluster Size Comparison:
##
##                        1    2
##   -0.9998999949995 2379  121
##   0.9998999949995  2332  168
## [1] "Rand Index the GAveraged-initialized k-means clustering: 0.000309877460025048"
```

k = 2, DIANA hierarchical clustering-initialized k-means clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, k_clus_diana_2$cluster)
ext_stats_k_diana_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster,
                                     k_clus_diana_2$cluster)
randInd_k_diana_2 <- ext_stats_k_diana_2$corrected.rand
print(paste0("Rand Index the DIANA-initialized k-means clustering: ", randInd_k_diana_2))
```

```
## [1] Cluster Size Comparison:
##
##                        1    2
##   -0.9998999949995 2379  121
##   0.9998999949995  2332  168
## [1] "Rand Index the DIANA-initialized k-means clustering: 0.000309877460025048"
```

k = 2, k-medoid clustering:

```
noquote("Cluster Size Comparison:")
table(sample_frame$income, clara_2$cluster)
ext_stats_k_medoid_2 <- cluster.stats(d=dist_matrix_scaled, orig_cluster,
                                     clara_2$cluster)
randInd_k_medoid_2 <- ext_stats_k_medoid_2$corrected.rand
print(paste0("Rand Index for k-medoid clustering: ", randInd_k_medoid_2))
```

```
## [1] Cluster Size Comparison:
##
##                         1    2
##   -0.9998999949995 1452 1048
##    0.9998999949995  2118  382
## [1] "Rand Index for k-medoid clustering: 0.0708171353489152"
```

Table 9: Summary of all of the k = 2 clustering methods and their Rand Index when compared to the original labeling.

	Clustering	Rand Index
Hierarchical	DIANA	0.0003
	Ward's Method	0.0855
	GAveraged	0.0002
k-means	Random-Init	0.1009
	Ward-Init	0.1009
	GAveraged-Init	0.0003
	DIANA-Init	0.0003
k-medoids		0.0708

5 Analysis and Evaluation of Results / Discussion

NOTE: This section is outdated. Please refer to the final paper for an up-to-date version that includes information gained from the nonlinear projections, k-medoid clustering, and k-means on SOM clustering.

5.1 PCA

From the Eigenvalues, as we noted before, because they are decreasing and the first 3 eigenvalues are considerably larger than the rest, we predicted that the PCA will give us decent results and some structure to visualize.

Upon applying PCA, we saw from the Scree plots in Figures 1 and 2 (as well as the summary(pca_data)) that it would take the first 8 principal components to encompass 70% of the cumulative variance. However, since our goal is to project them, we chose to focus on the first 2 components, which contain about 25% of the variance.

From Figure 3, we can see the shape that the samples take when plotted across the first 2 principal components. From the arrows, we can understand that higher values in that variable causes a datapoint to go in the direction of the arrow on the PCA plot. This seemingly also works for categorical data such as "relationship" where a label of 6 (Wife) would place the datapoint away from a label of 1 (Husband).

Interestingly, while the dataset only contains 2 official classes, "<=50K" (Income 1) and ">50K" (Income 2), we can see more clusters being formed. Figure 24 shows the possible clusters circled. Clusters 1, 2, and 3 are primarily "<=50", but they seem to form 3 sub-clusters on their own, possibly proving that "<=50" can be further broken down into 3 clusters. Clusters 4 and 5 seems to be primarily ">50", but Cluster 5 is sparser. While this could indicate that the datapoints within are anomalies, they are still relatively dense compared

to some of the other visible outliers. As such, for now, cluster 5 can be considered an additional clustering, possibly proving that ">50" can be further broken down into 2 sub-clusters.

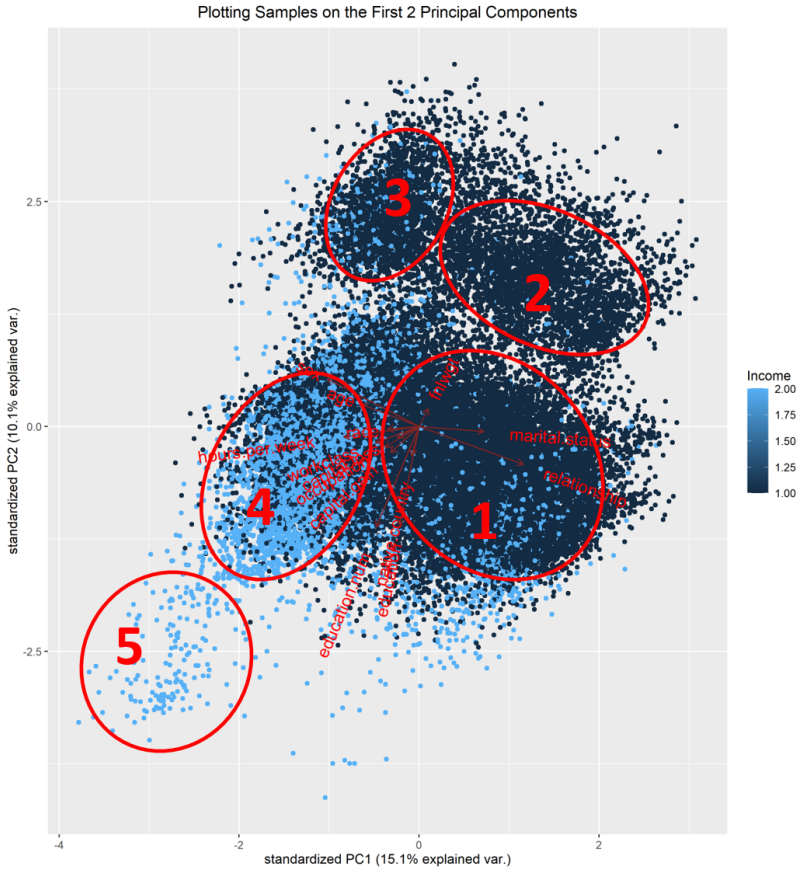


Figure 24: PCA Plot, but with possible regions highlighted.

5.2 MDS

For the Classical MDS, from Figure 6, we can roughly see 4 to 5 clusters in the shape of lines. This is highlighted in Figure 25, which is a repeat of the classical MDS Figure 6, but with the vertical axis expanded. We can see the general clusters start to form, such as in the case of the 4th arrow's region. Since our sample size was 5000, more samples might flesh this out so that the clusters are seen more clearly.

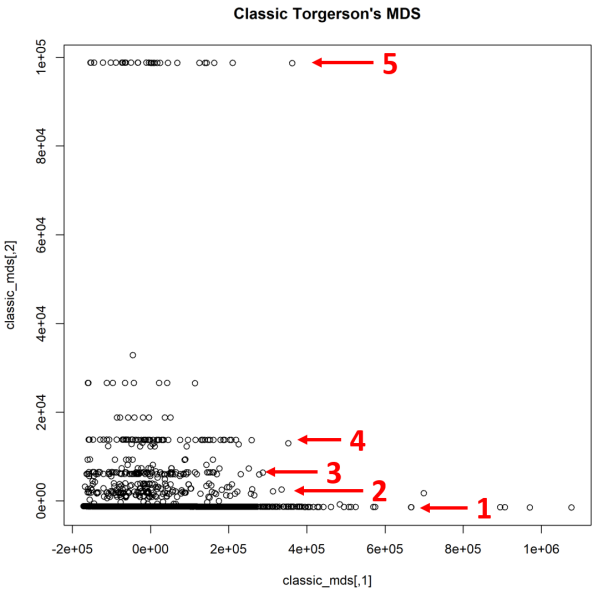


Figure 25: Classic MDS Plot, but with possible regions highlighted.

When using the classical MDS as the initialization, all of the other MDS algorithms, Sammon's, Kruskal's, and Symmetric Smacof, have plots identical to the Classical MDS (Figures 7, 9, 11, and 6, respectively). Their final stress levels differ however, with Sammon's MDS Stess being the lowest.

```
print(paste0("Sammon's MDS Stress (Classic Initialized): ", samm_mds_classic$stress))
print(paste0("Kruskal's MDS Stress (Classic Initialized): ", kruskal_mds_classic$stress))
print(paste0("Symmetric Smacof MDS Stress (Classic Initialized): ", smacof_mds_classic$stress))
```

```
## [1] "Sammon's MDS Stress (Classic Initialized): 0.294230053872011"
## [1] "Kruskal's MDS Stress (Classic Initialized): 26.5511238380599"
## [1] "Symmetric Smacof MDS Stress (Classic Initialized): 0.289220141587415"
```

When we look at the randomly initialized versions of these algorithms, we cannot easily make out any distinct clusters. In addition, the lowest stress value from the 10 randomly initialized iterations for each of these algorithms is shown below.

```
print(paste0("Best Sammon's MDS Stress (Randomly Initialized): ", best_samm_mds$stress))
print(paste0("Best Kruskal's MDS Stress (Randomly Initialized): ", best_kruskal_mds$stress))
print(paste0("Best Symmetric Smacof MDS Stress (Randomly Initialized): ", best_smacof_mds$stress))
```

```
## [1] "Best Sammon's MDS Stress (Randomly Initialized): 0.386185353688259"
## [1] "Best Kruskal's MDS Stress (Randomly Initialized): 46.1610051079746"
## [1] "Best Symmetric Smacof MDS Stress (Randomly Initialized): 0.291379692436169"
```

We can see that they are all larger than the stress values achieved when their respective algorithms use Classic MDS as the initialization.

Unlike PCA, MDS did not give us very useful visual information about our datasets. This could partly be because, while PCA was able to use all 48,842 datapoints, MDS was only able to use 5000 samples due to computation time limitations. In addition, Kruskal's MDS might not have been very effective in any case due to it being a non-metric MDS algorithm, which specializes in ranked dataset (since our dataset did not contain many ranked variables).

5.3 Clustering

For the AGNES hierarchical clustering, Ward's method and the Generalized Average linkage had agglomerative coefficients closest to 1. DIANA clustering's divisive coefficient was also relatively close to 1, suggesting that these 3 methods had good cluster distinction.

```
print(paste0("Ward's Method agglomerative coefficient: ", hier_ward$ac))
print(paste0("Generalized Average agglomerative coefficient: ", hier_gave$ac))
print(paste0("DIANA clustering  divisive coefficient: ", hier_diana$dc))
```

```
## [1] "Ward's Method agglomerative coefficient: 0.99340706745761"
## [1] "Generalized Average agglomerative coefficient: 0.976131616054977"
## [1] "DIANA clustering  divisive coefficient: 0.924897043630849"
```

From Ward's method, we can understand that using the squared error when two clusters are merged to judge similarity is a good fit for our dataset. This method is less susceptible to noise and outliers, so the fact that it has one of the highest coefficients could be an indication that our data is noisy/contains outliers. The Generalized Average (flexible UPGMA) linkage uses the modified, i.e. "flexible", Lance-Williams algorithm where the α_1 and α_2 coefficients are not constant, but proportional to the size of the 2 compared clusters. This method is better at recovering cluster structure over different error conditions and is also not susceptible to outliers [18], further indicating that our dataset contains outliers.

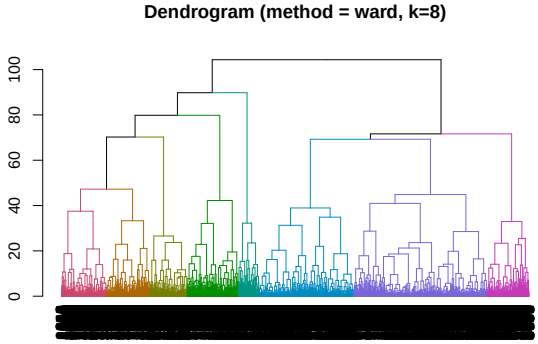
Comparing their dendrograms, (d) and (f) in Figure 13, respectively, we can see that Generalized Average's dendrogram clustering seems to be skewed towards the right side while Ward's method's dendrogram clustering seems a bit more balanced. In comparison, DIANA clustering's dendrogram, Figure 14, seems to be in between the 2 in terms of skewness.

Looking at their colored, clustered dendrograms, we can see that for k = 2, Generalized Average's and DIANA's hierarchical clusters, Figures 15 (c) and 15 (e) respectively, are clearly imbalanced while Ward's method's hierarchical clusters, Figure 15 (a), seems a bit more proportional. However, it should be noted that while the original dataset's class imbalance is 3.2:1, the imbalance shown here is not the same. Finally, a similar observation can be made for Generalized Average's and DIANA's k = 5 clusters, Figures 15 (d) and 15 (f) respectively, as one cluster seems to dominate in both, while there seems to be more of a balance in Ward's method's k = 5 clusters, Figure 15 (b), except for the blue cluster, which is barely visible.

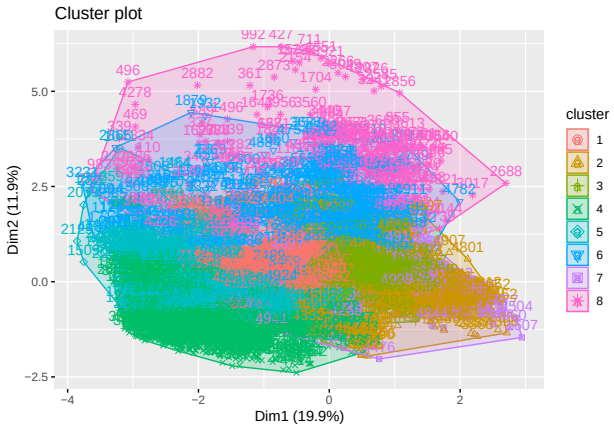
Based on visual inspection, it seems k = 8 gives a certain combination of clusters that can give us a ratio similar to the original 3.2:1 ratio if we combined some of the clusters. Indeed, Figure 26 (a) depicts such a clustering, where if we combine the 3 clusters on the left into 1 and the rest into another cluster, we get close to the original ratio. This could imply that the ">50K" class can be split into 3 different clusters and the "<=50K" class can be split into 5 clusters. However, if we use these clusters to initialize a K-means clustering, Figure 26 (b), it is difficult discern the clusters.

```
dend_col_ward_8 <- color_branches(dend_obj_ward, k = 8)
plot(dend_col_ward_8, main = "Dendrogram (method = ward, k=8)")

cut_ward_8 <- cutree(hier_ward, k = 8)
cluster_ward_8 = aggregate(data_matrix_scaled, list(cluster = cut_ward_8), mean)
cluster_ward_8 <- cluster_ward_8[, -c(1)]
k_clus_ward_8 <- kmeans(data_matrix_scaled, centers = cluster_ward_8)
noquote("Cluster Sizes for Ward's Method-Initialized k-means clustering, k = 8:")
print(k_clus_ward_8$size)
fviz_cluster(k_clus_ward_8, data = data_matrix_scaled)

k_clus_3 <- kmeans(data_matrix_scaled, centers = 3, nstart = 100)
fviz_cluster(k_clus_3, data = data_matrix_scaled)
```

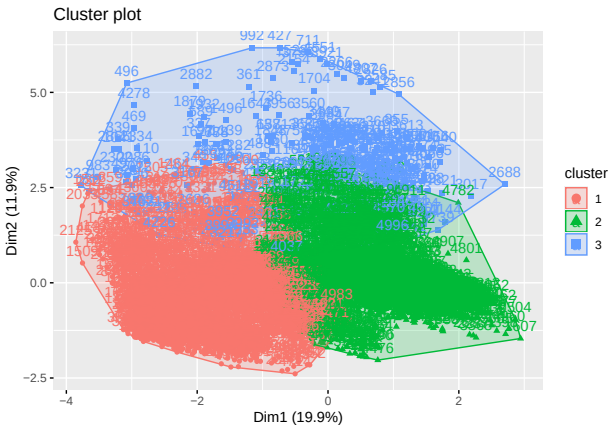
```
## [1] Cluster Sizes for Ward's Method-Initialized k-means clustering, k = 8:
## [1]  523  895 1374  632  558  409  414  195
```



(a) Hierarchical Clustering, k=8



(b) k-means Clustering (Hierarchical Clustering-Initialized), k=8



(c) k-means Clustering (Randomly Initialized), k=3

Figure 26: (a) Hierarchical Clustering using Ward's method and (b) its correspondingly initialized k-means clustering if we take k = 8. (c) k-means clustering for k=3, randomly initialized.

For the k-means cluster estimation, as discussed previously, the elbow method's plot, Figure 16, does not show any apparent bend. Nonetheless, given the shape of the plot, we can estimate a slight bend around k = 10. For the silhouette method, while k = 5 seems to be the optimal choice, it should be noted that other values of k also came close, such as k = 3.

Looking at the plots for the randomly initialized k-means clustering, Figure 17, we can see that there is a lot of overlap between the clusters, especially for k values 4, 5, and 10. For k = 4, three clusters can easily be seen, with a 4th one hidden behind. This seems to imply that k = 3 might be a better choice. k = 5 has 4 apparent cluster shapes with one hidden and k = 10 has no discernible cluster shapes at all.

Looking at the plots for the hierarchical clusters-initialized k-means clustering, Figure 18, we once again see a lot of overlap in the k = 5 clustering. Ward's method-initialized k = 5 clustering, Figure 18 (b), shows 3 major clusters which seem distinct enough, overlapped by a 4th, blue cluster. There is also an additional purple cluster on the top right that could be attributed to outliers (however, since our sampled dataset is only 5000 out of 48,842 datapoints, this could very well be another cluster). Taking this into consideration, there is a strong possibility of k = 3 being the correct number of clusters. We can see a randomly initialized k-means clustering with k = 3 plotted in Figure 26 (c), however there is still a cluster hidden beneath two others. For the Generalized Average-initialized and DIANA-initialized k-means clusterings, Figures 18 (d) and 18 (f) respectively, 2 overall clusters can easily be seen but the others overlap too much for any discernible guess at

the correct number of k.

Interestingly, we can see that for k = 2, the Generalized Average-initialized and DIANA-initialized k-means clusterings, Figures 18 (c) and 18 (e) respectively, seem to focus on the presumed outliers in the top-right corner. This can explain why, for the hierarchical clusterings, their plots were severely imbalanced as the clusterings were essentially "outlier or not outlier". The Ward's method-initialized k = 2 clustering, Figure 18 (a), seems to have some overlap at the border, but is otherwise split with the blue cluster taking into account the presumed anomalies.

5.4 Validation of Clusterings

From the stability tests, for the nonparametric bootstrap method, we can see that k = 2 had the highest clusterwise Jaccard bootstrap mean for its clusters, the lowest number of dissolved clusters, on average, and its number of recovered clusters closest to the number of bootstraps chosen (100). This indicates that k = 2 clustering is the most stable one given this test. For APN and ADM tests, the DIANA hierarchical clustering with cutoff at k = 2 was the most stable clustering, with values very close to 0 (0.0068 and 0.0323 respectively). However, for the AD and FOM tests, the randomly initialized, k = 10 k-means clustering was more stable, but with slightly larger values (3.4642 and 0.9514, respectively). This indicates that k = 2 clusters and k = 10 clusters seem to be at a tie with respect to the last 4 stability tests.

For all of the internal validation tests, Connectivity, Dunn Index, and Silhouette Width, the DIANA hierarchical clustering with cutoff at k = 2 had the best scores (0.0000, 0.6309, 0.6581, respectively). Thus, 2 clusters seem to be the best with respect to internal validation measures.

Since we can visualize Silhouette plots, and we know k = 2 seems to be the ideal clustering size, Figures 21 and 22 show the Silhouettes of all of our k = 2 clusterings, both hierarchical and k-means. The DIANA and GAveraged hierarchical plots, Figures 21 (a) and (c) seem to be identical, while the Ward's Method hierarchical plot, Figure 21 (b) seems to have some negative values, indicating that some samples might have been assigned to the wrong cluster. The DIANA and GAveraged initialized k-means silhouettes, Figures 22 (c) and (d), also seem to be identical, while (a) and (b) have some negative values again. The k=10 k-means clustering's Silhouette was also plotted, Figure 23, because it had the best AD and FOM test scores. However, we can see numerous negative values, indicating that this cluster numbering might not be ideal.

Finally, for external validation tests, Table 9 shows that almost all of the clustering options have poor scores that are very close to 0, with some (Ward's Method hierarchical and its initialized k-means clustering) being negative values. This tells us that there is a poor agreement between these k = 2 clusterings and the original labeling of our dataset. The largest value is only 0.0219, shared by the DIANA and GAveraged hierarchical clusterings as well as their initialized k-means clustering.

Once again, upon inspection of their colored dendograms, Figures 15 (c) and (e), and k-means plots, Figures 18 (c) and (e), it is clear to see that the clusters seem to be divided between the outlier and non-outliers. Previously, these outliers were kept with the assumption that, due to only sampling 5000 datapoints, these "outliers" might actually be a budding cluster. Regardless of whether this is correct or not, it seems that these 4 clusterings have taken advantage of this and formed an "outliers versus non-outliers" clustering that scored the highest validity among the other clustering number options.

Thus, while k = 2 is the original dataset's labeling, our k = 2 has focused more on the separation between outliers and non-outliers.

5.5 Possible Additional Processes

Certain additional things can be tried in order to get better results for the MDS segment. For starters, further processing time can be allotted so that more than 5000 samples can be processed. For this assignment, the main goal was proof of concept: testing whether our code and algorithms give decent results. For the final report, a larger sample can be utilized and more processing time can be devoted. Another additional thing that can be done is applying our MDS algorithms on distance matrices that are not based on Euclidean distance, but on other distance measurements such as the Mahalanobis or Manhattan distance. They could

be more beneficial to our goal of visualizing our dataset. Finally, other MDS algorithms can also be tried, such as metaMDS and wcmdscale (Weighted Classical Multidimensional Scaling).

For hierarchical and k-means clustering, more distances other than the Euclidean distance can be used, as the Euclidean distance favors spherical clusters. In addition, outliers can be removed before clustering is done. Previously, they were not removed on the off chance they were budding clusters (due to us sampling a smaller portion of our dataset). However, it might be beneficial to remove them.

The results of our clustering validation also affirm the benefits of removing the outliers. Whether they are budding clusters or not, because of our chosen 5000 samples, they effectively act as outliers that disrupt the clustering process by making it easy to cluster outlier versus non-outlier. As such, given our small data sample, it might be beneficial to identify and remove these outliers before creating the clusters.

These things will be taken into consideration for the final report.

6 Conclusion

In conclusion, this report performed a preliminary analysis, sampled the dataset with respect to the original "income" population distribution, performed PCA on the entire dataset, performed PCA and multiple MDS processes on the sampled dataset, created multiple hierarchical and k-means clusterings and, lastly, measured their stability and validity, both internal and external.

While MDS did not give us a very good visual description of our dataset with the given sample size, PCA on the entire dataset was very valuable because we were able to make out 4 or 5 potential sub-clusters within the 2 assigned clusters.

Hierarchical clustering indicated that our data might have too many outliers and k = 8 was inferred to be the ideal number of clustering. While the elbow method was not fruitful, the silhouette method indicated k = 5 as ideal, which is in line with our PCA clustering suggestion for the whole dataset. However, actually plotting the k-means clusters showed that k = 3 might be the better choice.

Validation of our various clusters reinforced the idea that k = 4, 5, and 10 might not be good fits for our sampled dataset. While our k = 2 clusters seem to have the best scores, particularly the DIANA hierarchical clustering, we can see that this is mostly likely due to the clustering of outliers versus non-outliers, which is not fruitful for our case.

For the final report, outliers can be removed before clustering is applied. In addition to the k = 2 clustering, clusterings based on the previous hierarchical, elbow method, and k-means results, namely, k = 8, 5, and 3, can be redone and validated as well in order to get an better idea of how many possible clusters our dataset contains.

7 References

[1] Rebecca Merrett, "Census Income." Jan. 2019, Accessed: Nov. 04, 2020. [Online]. Available: https://code.datasciencedojo.com/datasciencedojo/datasets/tree/master/Census%20Income.

[2] R. Kohavi, "Scaling Up the Accuracy of Naive-Bayes Classifiers: a Decision-Tree Hybrid," in *KDD'96: Proceedings of the second international conference on knowledge discovery and data mining*, Aug. 1996, pp. 202–207, [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3001460.3001502.

[3] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation," *ACM Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, p. 4, Mar. 2007, doi: 10.1145/1217299.1217303.

[4] D. Pelleg and A. Moore, "Scalable and Practical Probability Density Estimators for Scientific Anomaly Detection," PhD thesis, Carnegie Mellon University, 2004.

[5] J. Bailey, T. Manoukian, and K. Ramamohanarao, "Fast algorithms for mining emerging patterns," in *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, 2002, vol. 2431 LNAI, pp. 39–50, doi: 10.1007/3-540-45681-3_4.

[6] Luke Hayden, "PCA Analysis in R - DataCamp." Aug. 2018, Accessed: Nov. 26, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/pca-analysis-r.

[7] Aaron Schlegel, "Principal Component Analysis with R Example." Jan. 2017, Accessed: Nov. 26, 2020. [Online]. Available: https://aaronschlegel.me/principal-component-analysis-r-example.html.

[8] Gaston Sanchez, "7 Functions to do Metric Multidimensional Scaling in R | Visually Enforced." Jan. 2013, Accessed: Nov. 26, 2020. [Online]. Available: http://www.gastonsanchez.com/visually-enforced/how-to/2013/01/23/MDS-in-R/.

[9] Perceptive Analytics, "How to Perform Hierarchical Clustering using R." Dec. 2017, Accessed: Dec. 06, 2020. [Online]. Available: https://www.r-bloggers.com/2017/12/how-to-perform-hierarchical-clustering-using-r/.

[10] Manish Pathak, "Hierarchical Clustering in R." Jul. 2018, Accessed: Dec. 06, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/hierarchical-clustering-R.

[11] Berkeley, "Cluster Analysis." Mar. 2011, Accessed: Dec. 06, 2020. [Online]. Available: https://www.stat.berkeley.edu/%7B~%7Ds133/Cluster2a.html.

[12] Bradley Boehmke, "K-means Cluster Analysis." Mar. 2017, Accessed: Dec. 06, 2020. [Online]. Available: https://uc-r.github.io/kmeans%7B/_%7Dclustering.

[13] R-core, "Kmeans function." Nov. 2016, Accessed: Dec. 07, 2020. [Online]. Available: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/kmeans.

[14] Kasia Kulma, "Cluster Validation In Unsupervised Machine Learning." May 2017, Accessed: Dec. 24, 2020. [Online]. Available: https://kkulma.github.io/2017-05-10-cluster-validation-in-unsupervised-machine-learning/.

[15] STHDA, "Clustering Validation Statistics: 4 Vital Things Everyone Should Know." Oct. 2015, Accessed: Dec. 24, 2020. [Online]. Available: http://www.sthda.com/english/wiki/wiki.php?id%7B/_%7Dcontents=7952%7B/#%7Dexternal-clustering-validation.

[16] Cosma Shalizi, "The Bootstrap." Feb. 2011, Accessed: Dec. 24, 2020. [Online]. Available: https://www.stat.cmu.edu/%7B~%7Dcshalizi/402/lectures/08-bootstrap/lecture-08.pdf.

[17] G. Brock, V. Pihur, S. Datta, and S. Datta, "clValid , an R package for cluster validation," Department of Bioinformatics; Biostatistics, University of Louisville, Jul. 2020.

[18] L. Belbin, D. P. Faith, and G. W. Milligan, "A Comparison of Two Approaches to Beta-Flexible Clustering," *Multivariate Behavioral Research*, vol. 27, no. 3, pp. 417–433, Jul. 1992, doi: 10.1207/s15327906mbr2703_6.

8 Appendix

8.0.1 Modified ggbiplot Function from my_functions.R:

Note: Only 2 small color changes were made, so only those parts will be listed for brevity.

```
my_ggbiplot <- function (pcobj, choices = 1:2, scale = 1, pc.biplot = TRUE,
    obs.scale = 1 - scale, var.scale = scale, groups = NULL,
    ellipse = FALSE, ellipse.prob = 0.68, labels = NULL, labels.size = 3,
    alpha = 1, var.axes = TRUE, circle = FALSE, circle.prob = 0.69,
    varname.size = 3, varname.adjust = 1.5, varname.abbrev = FALSE,
    ...)
{
    ...

    if (var.axes) {
        if (circle) {
            theta <- c(seq(-pi, pi, length = 50), seq(pi, -pi,
                length = 50))
            circle <- data.frame(xvar = r * cos(theta), yvar = r *
                sin(theta))
            g <- g + geom_path(data = circle, color = muted("white"),
                size = 1/2, alpha = 1/3)
        }
        g <- g + geom_segment(data = df.v, aes(x = 0, y = 0,
            xend = xvar, yend = yvar), arrow = arrow(length = unit(1/2,
            "picas")), color = muted("red"))
    }

    ...

    if (var.axes) {
        g <- g + geom_text(data = df.v, aes(label = varname,
            x = xvar, y = yvar, angle = angle, hjust = hjust),
            color = "red", size = 5)
    }
    return(g)
}
```