##Install R

##Install R studio

##Install Rtools; Close R Studio and reopen after installing Rtools to allow initialization

##Installing the needed packages

```
install.packages(c("dplyr", "ggplot2", "Rtsne", "data.table", "targets",
"DT", "tidyverse"))
```

##Load installed packages so Far:

```
library(dplyr)

library(ggplot2)

library(Rtsne)

library(data.table)

library(targets)

library(DT)

library(tidyverse)

library(utils)

library(gridExtra)
```

##The packages "utils" and "gridExtra" are inbuilt and do not need to be installed; only loaded

##Confirm directory before continuing coding:

```
getwd()
```

##Since I already have the two datasets I want to work with in the directory on my local

computer, they are visible in the files window on the bottom right

##Now, to read the two datasets (improve and MDS grouping) into the R environment:

##Reading the improve genetic matrix dataset: saved as improve

```
improve=read.table("improve_sczcmd_genetics.txt", header = TRUE, na.strings
= ".")
```

##Reading the MDS predetermined classification: saved as pregroup

```
pregroup = read.table("20200921_groups.txt", header = TRUE)
```

##After loading into the environment, by clicking on the datasets, we are able to load them in the source window

##Note: the goal is to run a t-SNE with PCA Initialization, which is a machine learning algorithm to visualize high dimensional data - it's advantage over other is that it helps with scalability and stability

#However, it does not allow missing data and also only works on numeric data type

##Visual exploration of the improve dataset shows that the genotypes in improve are coded as 11,12, and 22.

##Transforming genotypes to allele frequencies in the improve dataset

```
improve[improve==11] = 0
improve[improve==12] = 1
improve[improve==22] = 2
```

##Data Preparation Steps.

#1. Confirm that the data is in numerical data type. Change to numerical if not.

```
str(improve)
str(pregroup)
```

##For improve: it shows that the ID columns are integers and the SNPs are numeric which is Okay

##For pregroup: it shows that the ID columns are integers and the grouping are character data types

##In the improve data set, it shows NA (meaning some missing data points). Now we consider methods for handling missing data:

#I will proceed with three methods:

#1. Mean imputation

#2. K-Nearest Neighbour/multiple/advanced imputation

#3. Complete Case Analysis

##I will create three data frames to use for each by copying the improve dataframe

##Creating a copy of my merged dataframe to ensure that I do not lose access to it

```
original_improve <- improve
```

#1. Mean imputation method:

#Computing a function to impute missing values with the mean of the column

```
imputeMean <- function(vec) {
 m <- mean(vec, na.rm = TRUE)
 vec[is.na(vec)] <- m
 return(vec)
}
```

#Creating a copy of the original dataset for mean imputation

```
meanimprove <- improve
```

#Imputing the mean to missing data points for the meanimprove dataset:

```
meanimprove[,-c(1,2)] <- as.data.frame(apply(meanimprove[,-c(1,2)], 2,
imputeMean))
print(meanimprove)
str(meanimprove)
```

#2. K Nearest Neighbour (KNN) imputation method:

#Creating a copy of the original dataset for KNN imputation

```
knnimprove <- improve
```

Install and load the VIM package for KNN: Can also be done in DMwR package with kNNImputation code instead of kNN

```
install.packages("VIM")
library(VIM)
```

#Now performing the multiple imputation in knnimprove dataset

Exclude ID columns for imputation

```
numeric_data <- knnimprove[, -c(1, 2)] # Exclude columns 1 and 2
```

Perform KNN imputation on the numeric data

```
imputed_numeric_data <- kNN(numeric_data, k = 5, imp_var = FALSE)
```

Recombine with ID columns

```
knnimprove_imputed <- cbind(knnimprove[, c(1, 2)], imputed_numeric_data)
```

Update the original dataset

```
knnimprove <- knnimprove_imputed
```

View the updated dataset

```
print(knnimprove)
str(knnimprove)
```

#3. Complete case analysis method

#Creating a copy of the original dataset for complete case analysis

```
ccaimprove <- improve
```

#Dropping samples with missing values

```
ccaimprove <- na.omit(ccaimprove)
str(ccaimprove)
```

##Result:

#Mean Imputation: 3468 obs of 149 variables (147 SNPs)

#KNN Imputation: 3468 obs of 149 variables (147 SNPs)

#Complete Case Analysis: 2874 obs of 149 variables (147 SNPs)

#Next step is to merge my pregroup dataset with the genomic matrices so that I can use it in classification for the tSNE

#I will merge using the individual ID columns.

#First rename IID to iid in each three datasets

Rename column 'IID' to 'iid'

```
names(meanimprove)[names(meanimprove) == "IID"] <- "iid"
names(knnimprove)[names(knnimprove) == "IID"] <- "iid"
names(ccaimprove)[names(ccaimprove) == "IID"] <- "iid"
```

#Now merging using iid as the unique identifier in each of the three datasets

Merge 'pregroup' with 'meanimprove' by column 'iid'

```
meanimprove <- merge(meanimprove, pregroup, by = "iid")
```

View the merged dataset

```
head(meanimprove)
```

Merge 'pregroup' with 'knnimprove' by column 'iid'

```
knnimprove <- merge(knnimprove, pregroup, by = "iid")
```

View the merged dataset

```
head(knnimprove)
```

Merge 'pregroup' with 'ccaimprove' by column 'iid'

```
ccaimprove <- merge(ccaimprove, pregroup, by = "iid")
```

View the merged dataset

```
head(ccaimprove)
```

##From the result fid is duplicated (with FID) in each three datasets and groups_maf10 is at the last column

#To drop fid duplicate from each data

```
meanimprove <- subset(meanimprove, select = -fid)
knnimprove <- subset(knnimprove, select = -fid)
ccaimprove <- subset(ccaimprove, select = -fid)
```

#Now to move groups_maf10 to the third column

#First: create a function to reoder the columns to place groups_maf10 in column 3

```
# Function to move a column to a specific position
move_column_to_position <- function(df, column_name, position) {

# Get the current column order
column_order <- names(df)

# Remove the column to move from its current position
column_to_move <- df[[column_name]]
column_order <- column_order[column_order != column_name]

# Insert the column to the desired position
new_column_order <- c(column_order[1:(position-1)], column_name,
column_order[position:length(column_order)])

# Reorder the columns based on new_column_order
df <- df[, new_column_order]

# Return the reordered data frame
return(df)
}

#Applying this function to each dataset
meanimprove <- move_column_to_position(meanimprove, "groups_maf10", 3)
knnimprove <- move_column_to_position(knnimprove, "groups_maf10", 3)
ccaimprove <- move_column_to_position(ccaimprove, "groups_maf10", 3)

##To save my cleaned data sets so far:

# Save each dataset as a CSV file
write.csv(meanimprove, file = "meanimprove.csv", row.names = FALSE)
write.csv(knnimprove, file = "knnimprove.csv", row.names = FALSE)
write.csv(ccaimprove, file = "ccaimprove.csv", row.names = FALSE)

##Code to read the datasets later:
```

Read the datasets from CSV files

```
meanimprove <- read.csv("meanimprove.csv")
knnimprove <- read.csv("knnimprove.csv")
ccaimprove <- read.csv("ccaimprove.csv")
```

Check the data type of 'groups_maf10' in meanimprove

```
class(meanimprove$groups_maf10)
```

Check the data type of 'groups_maf10' in knnimprove

```
class(knnimprove$groups_maf10)
```

Check the data type of 'groups_maf10' in ccaimprove

```
class(ccaimprove$groups_maf10)
```

##I read on something called standardising/normalising the genomic data set during preprocessing to ensure that all features contribute equally to the distance calculations

##I will now proceed with two analysis method (unnormalised datasets and normalised data Sets so as to compare the results)

#I will use the prefix "u_" for datasets for the unnormalised genomic data and "n_" for the normalised genomic data

#creating pairs of new data sets below for each

```
u_meanimprove <- meanimprove
n_meanimprove <- meanimprove
u_knnimprove <- knnimprove
n_knnimprove <- knnimprove
u_ccaimprove <- ccaimprove
n_ccaimprove <- ccaimprove
```

##For the n_ dataframes, I will normalise the genomic data in these to ensure that each data points contribute to the distance calculations:

#For the n_meanimprove data set

Extract genomic data (excluding columns 1, 2, and 3)

```
genomic_data <- n_meanimprove[, -c(1, 2, 3)]
```

Normalize the genomic data

```
normalized_genomic_data <- scale(genomic_data)
```

Combine normalized data with ID and grouping columns

```
n_meanimprove <- cbind(n_meanimprove[, 1:3], normalized_genomic_data)
```

#For the n_knnimprove data set

Extract genomic data (excluding columns 1, 2, and 3)

```
genomic_data <- n_knnimprove[, -c(1, 2, 3)]
```

Normalize the genomic data

```
normalized_genomic_data <- scale(genomic_data)
```

Combine normalized data with ID and grouping columns

```
n_knnimprove <- cbind(n_knnimprove[, 1:3], normalized_genomic_data)
```

#For the n_ccaimprove data set

Extract genomic data (excluding columns 1, 2, and 3)

```
genomic_data <- n_ccaimprove[, -c(1, 2, 3)]
```

Normalize the genomic data

```
normalized_genomic_data <- scale(genomic_data)
```

Combine normalized data with ID and grouping columns

```
n_ccaimprove <- cbind(n_ccaimprove[, 1:3], normalized_genomic_data)
```

##I have created these two variations (normalised and unnormalised) so that I can compare the final results of both methods

##I will now move in to PCA and tSNE

##Note: the groups_maf10 variable is of the character data type. PCA cannot use character data

type so I will need to convert it to numeric. I will use one-hot encoding technique which allows me to convert it to numeric data type without losing its feature as a categorical variable

##I will also encode the missing group to allow for their visualisation also.

##Install and load the dummies package

```
install.packages("fastDummies")
```

Load the necessary package

```
library(fastDummies)
```

Handling the groups_maf10 variable in all 6 data frames so that I can plot PCA and group by all 4 categories (groups 1 – 3, and NA)

##For the u_mean

Convert 'groups_maf10' to a factor, treating missing values as a separate level in the u_meanimprove data frame:

```
u_meanimprove$groups_maf10 <- factor(u_meanimprove$groups_maf10, levels = c("1", "2", "3", NA))
```

Perform one-hot encoding

```
u_meanimprove_encoded <- dummy_cols(u_meanimprove, select_columns = "groups_maf10", remove_first_dummy = FALSE)
```

Remove the original 'groups_maf10' column from the dataset

```
u_meanimprove <- u_meanimprove_encoded[, !names(u_meanimprove_encoded) %in%
"groups_maf10"]
```

##For the n_mean

Convert 'groups_maf10' to a factor, treating missing values as a separate level

```
n_meanimprove$groups_maf10 <- factor(n_meanimprove$groups_maf10, levels = c("1", "2", "3", NA))
```

Perform one-hot encoding

```
n_meanimprove_encoded <- dummy_cols(n_meanimprove, select_columns = "groups_maf10",
remove_first_dummy = FALSE)
```

Remove the original 'groups_maf10' column and keep the encoded columns

```
n_meanimprove <- n_meanimprove_encoded[, !names(n_meanimprove_encoded) %in%
"groups_maf10"]
```

##For the u_knn

Convert 'groups_maf10' to a factor, treating missing values as a separate level

```
u_knnimprove$groups_maf10 <- factor(u_knnimprove$groups_maf10, levels =
c("1", "2", "3", NA))
```

Perform one-hot encoding

```
u_knnimprove_encoded <- dummy_cols(u_knnimprove, select_columns =
"groups_maf10",

remove_first_dummy = FALSE)
```

Remove the original 'groups_maf10' column from the dataset and keep the encoded columns

```
u_knnimprove <- u_knnimprove_encoded[, !names(u_knnimprove_encoded) %in%

"groups_maf10"]
```

##For the n_knn

Convert 'groups_maf10' to a factor, treating missing values as a separate level

```
n_knnimprove$groups_maf10 <- factor(n_knnimprove$groups_maf10, levels =
c("1", "2", "3", NA))
```

Perform one-hot encoding

```
n_knnimprove_encoded <- dummy_cols(n_knnimprove, select_columns =
"groups_maf10",

remove_first_dummy = FALSE)
```

Remove the original 'groups_maf10' column from the dataset and keep the encoded columns

```
n_knnimprove <- n_knnimprove_encoded[, !names(n_knnimprove_encoded) %in%

"groups_maf10"]
```

##For the u_cca

Convert 'groups_maf10' to a factor, treating missing values as a separate level

```
u_ccaimprove$groups_maf10 <- factor(u_ccaimprove$groups_maf10, levels =
c("1", "2", "3", NA))
```

Perform one-hot encoding

```
u_ccaimprove_encoded <- dummy_cols(u_ccaimprove, select_columns =
"groups_maf10",

remove_first_dummy = FALSE)
```

Remove the original 'groups_maf10' column from the dataset and keep the encoded columns

```
u_ccaimprove <- u_ccaimprove_encoded[, !names(u_ccaimprove_encoded) %in%

"groups_maf10"]
```

##For the n_cca

Convert 'groups_maf10' to a factor, treating missing values as a separate level

```
n_ccaimprove$groups_maf10 <- factor(n_ccaimprove$groups_maf10, levels =
c("1", "2", "3", NA))
```

Perform one-hot encoding

```
n_ccaimprove_encoded <- dummy_cols(n_ccaimprove, select_columns =
"groups_maf10",

remove_first_dummy = FALSE)
```

Remove the original 'groups_maf10' column from the dataset and keep the encoded columns

```
n_ccaimprove <- n_ccaimprove_encoded[, !names(n_ccaimprove_encoded) %in%

"groups_maf10"]
```

Load necessary library

```
library(ggplot2)

library(fastDummies)
```

Function to perform PCA and plot results, then save the plot

```
perform_pca_and_save_plot <- function(df, filename) {
```

Extract numeric data excluding the ID columns and the one-hot encoded columns

```
numeric_data <- df[, !(names(df) %in% c("iid", "FID", "groups_maf10_1",
"groups_maf10_2", "groups_maf10_3", "groups_maf10_NA"))]
```

Check if numeric_data has the expected structure

```
print("Numeric Data:")

print(head(numeric_data))
```

Perform PCA

```
pca_result <- prcomp(numeric_data, scale. = TRUE)
```

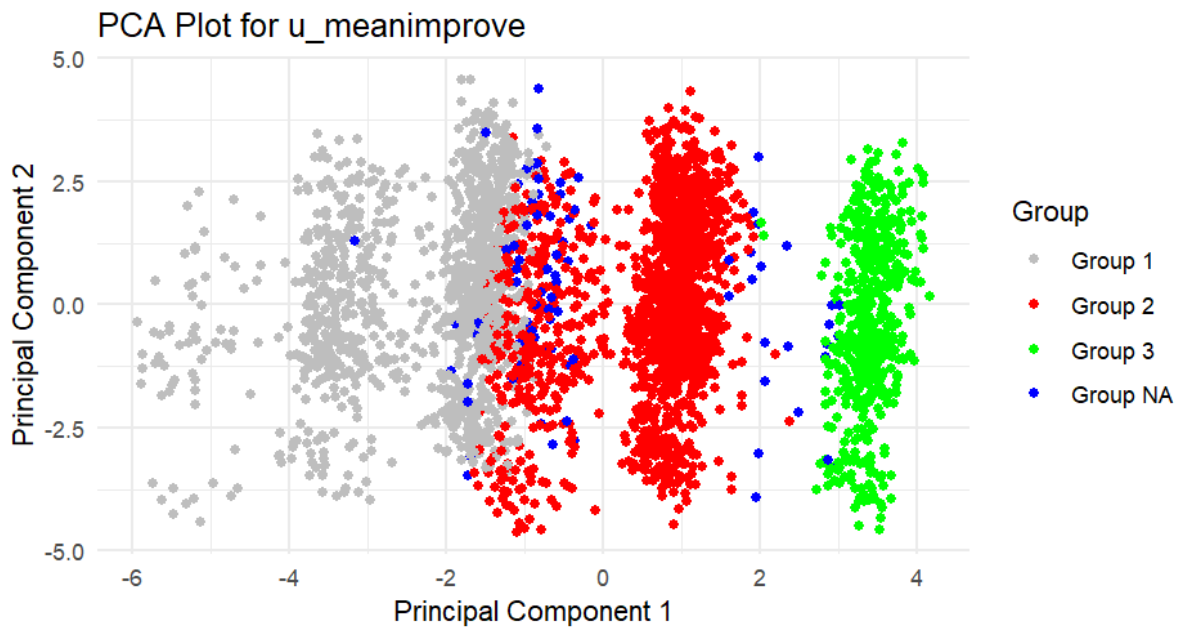
Check PCA result

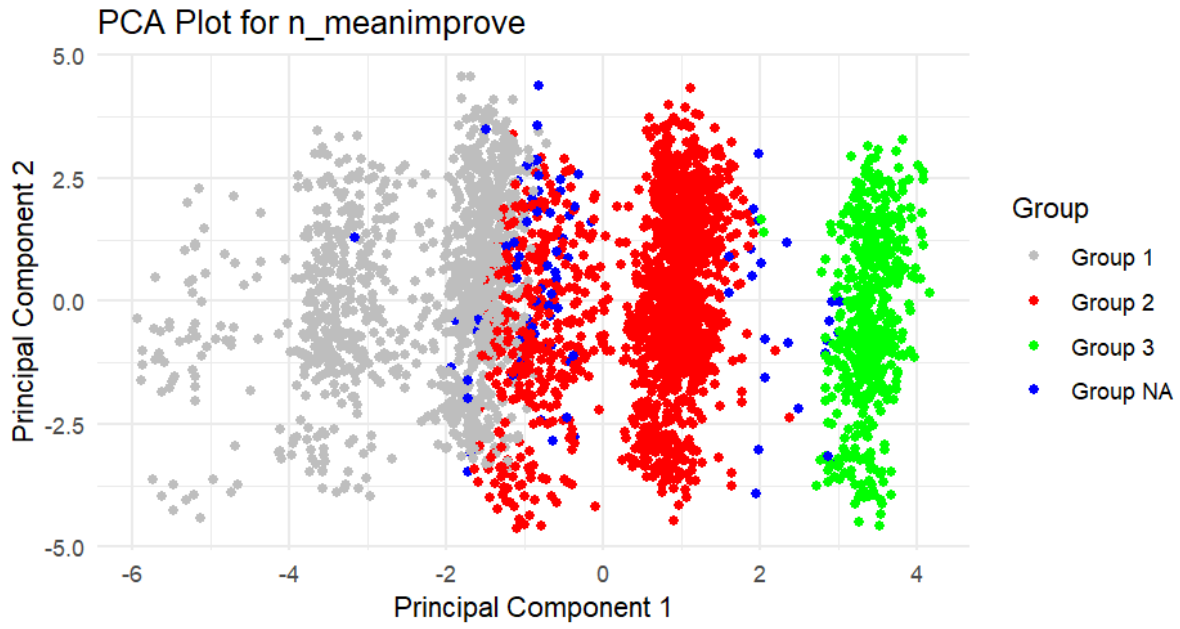
```
 print("PCA Summary:")

 print(summary(pca_result))
```

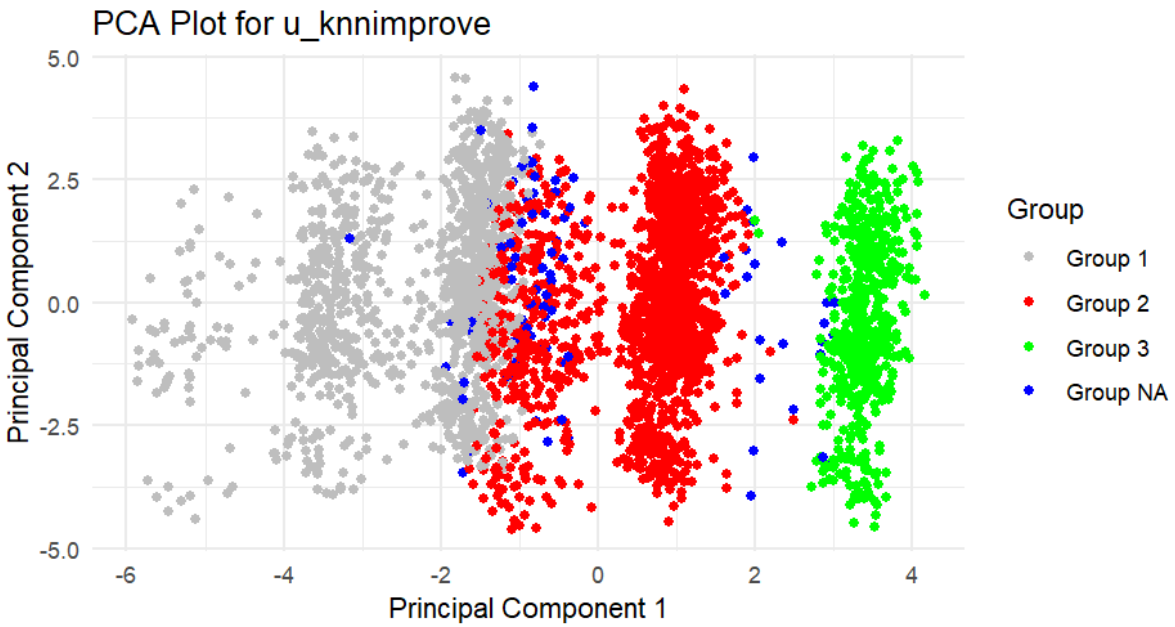
```
# Create a data frame with PCA results
 pca_df <- data.frame(
 PCA1 = pca_result$x[, 1],
 PCA2 = pca_result$x[, 2],
 Group1 = df$groups_maf10_1,
 Group2 = df$groups_maf10_2,
 Group3 = df$groups_maf10_3,
 GroupNA = df$groups_maf10_NA
 )

# Check PCA Data Frame
 print("PCA Data Frame:")
 print(head(pca_df))

# Create a grouping variable for colouring
 pca_df$Group <- factor(
 apply(pca_df[, c("Group1", "Group2", "Group3", "GroupNA")], 1, function(x)
{
 if (!is.na(x["Group1"]) && x["Group1"] == 1) return("Group 1")
 if (!is.na(x["Group2"]) && x["Group2"] == 1) return("Group 2")
 if (!is.na(x["Group3"]) && x["Group3"] == 1) return("Group 3")
 if (!is.na(x["GroupNA"]) && x["GroupNA"] == 1) return("Group NA")
 return(NA) # Handle any unexpected cases
 })
 )

# Check Group assignment
print("Group Assignment:")
print(table(pca_df$Group))

# Define colors
 colors <- c("Group 1" = "gray", "Group 2" = "red", "Group 3" = "green",
"Group NA" = "blue")
```

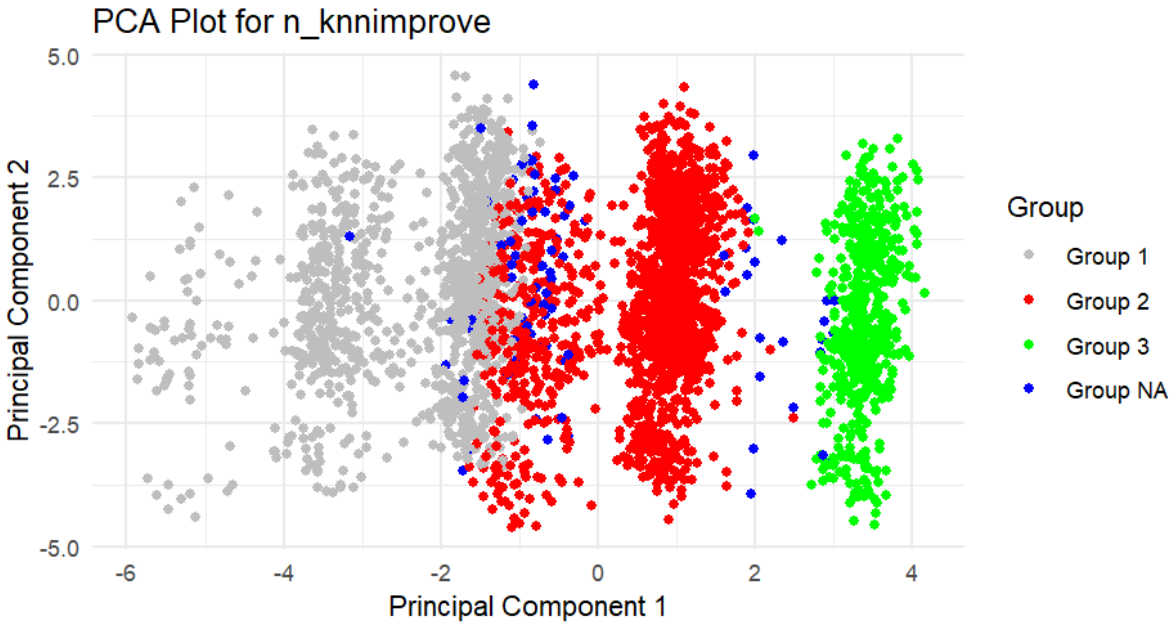
```
# Plot PCA results
pca_plot <- ggplot(pca_df, aes(x = PCA1, y = PCA2, color = Group)) +
 geom_point() +
 scale_color_manual(values = colors) +
 labs(title = paste("PCA Plot for", deparse(substitute(df))),
 x = "Principal Component 1",
 y = "Principal Component 2") +
 theme_minimal()

# Print the plot to R console
 print(pca_plot)

# Save the plot
 ggsave(filename = filename, plot = pca_plot, width = 8, height = 6)
}

# Apply the function to each dataframe and save the plots
perform_pca_and_save_plot(u_meanimprove, "u_meanimprove_pca.png")
perform_pca_and_save_plot(u_knnimprove, "u_knnimprove_pca.png")
perform_pca_and_save_plot(u_ccaimprove, "u_ccaimprove_pca.png")
perform_pca_and_save_plot(n_meanimprove, "n_meanimprove_pca.png")
perform_pca_and_save_plot(n_knnimprove, "n_knnimprove_pca.png")
perform_pca_and_save_plot(n_ccaimprove, "n_ccaimprove_pca.png")
```

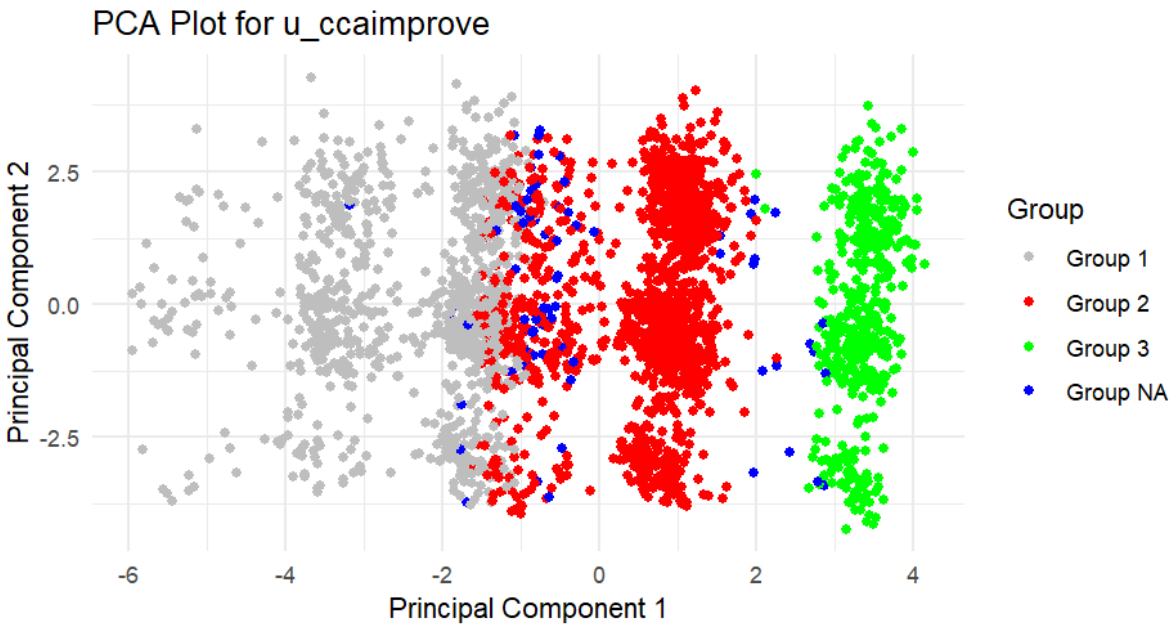
PCA Results:

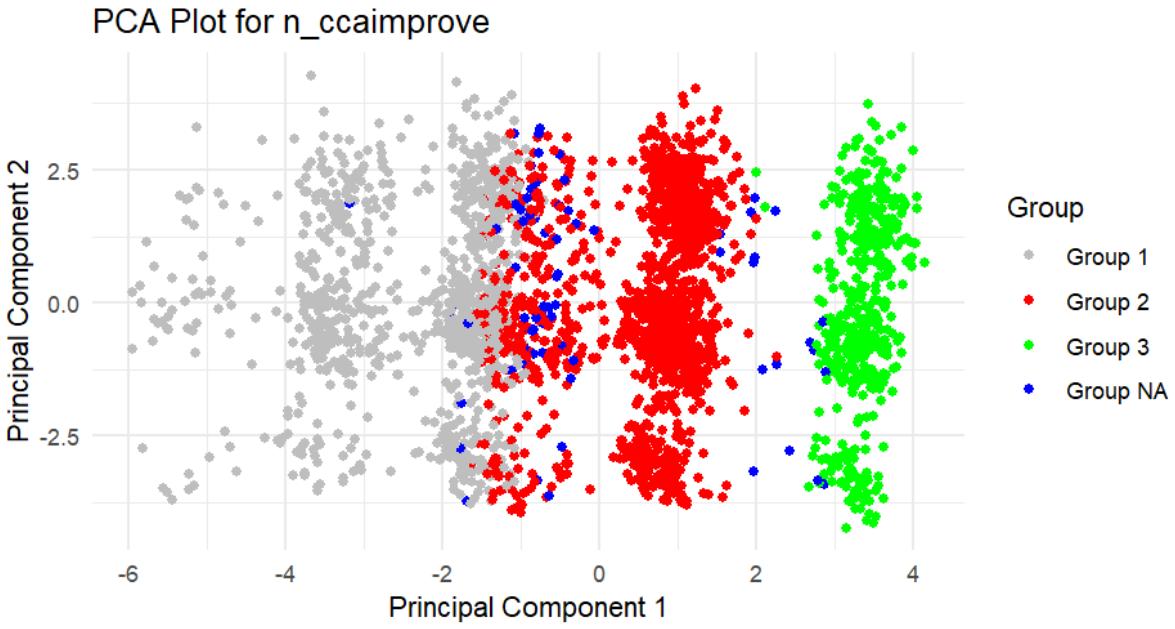

PCA Plot for u_meanimprove
Principal Component 2
5.0
2.5
0.0
-2.5
-5.0
-6
-4
-2
0
2
4
Principal Component 1
Group
Group 1
Group 2
Group 3
Group NA


PCA Plot for n_meanimprove
Principal Component 2
5.0
2.5
0.0
-2.5
-5.0
-6
-4
-2
0
2
4
Principal Component 1
Group
Group 1
Group 2
Group 3
Group NA


PCA Plot for u_knnimprove
Principal Component 2
5.0
2.5
0.0
-2.5
-5.0
-6
-4
-2
0
2
4
Principal Component 1
Group
Group 1
Group 2
Group 3
Group NA


PCA Plot for n_knnimprove
Principal Component 2
5.0
2.5
0.0
-2.5
-5.0
-6
-4
-2
0
2
4
Principal Component 1
Group
Group 1
Group 2
Group 3
Group NA


PCA Plot for u_ccaimprove
Principal Component 2
2.5
0.0
-2.5
-6
-4
-2
0
2
4
Principal Component 1
Group
Group 1
Group 2
Group 3
Group NA


PCA Plot for n_ccaimprove
Principal Component 2
2.5
0.0
-2.5
-6
-4
-2
0
2
4
Principal Component 1
Group
Group 1
Group 2
Group 3
Group NA

Running tSNE on all datasets:

Load necessary libraries

```
library(Rtsne)

library(ggplot2)

library(gridExtra) # For arranging plots in a grid
```

Define parameters

```
max_iter <- 1000

theta <- 0.1

perplexities <- c(25, 50, 100)

pcaDims <- c(2, 5, 10)

mycolors <- c("Group 1" = "gray", "Group 2" = "red", "Group 3" = "green",
"Group NA" = "blue")

figWidth <- 2000

pointSize <- 0.5

legendSize <- 5

textSize <- 5

num_threads <- 0
```

Function to perform t-SNE with PCA initialization and create plots

```
doRtsne <- function(perplexity, pcaDim) {

  tsne <- Rtsne(dat[ , !(names(dat) %in% c("iid", "FID", "groups_maf10_1",
"groups_maf10_2", "groups_maf10_3", "groups_maf10_NA"))],
                initial_dims = pcaDim,

                dims = 2,

                perplexity = perplexity,

                verbose = TRUE,

                max_iter = max_iter,

                theta = theta,

                num_threads = num_threads)


  tsne_plot <- data.frame(x = tsne$Y[, 1], y = tsne$Y[, 2], Groups =
factor(

    apply(dat[, c("groups_maf10_1", "groups_maf10_2", "groups_maf10_3",
"groups_maf10_NA")], 1, function(x) {
```

```
      if (!is.na(x["groups_maf10_1"]) && x["groups_maf10_1"] == 1)
return("Group 1")
      if (!is.na(x["groups_maf10_2"]) && x["groups_maf10_2"] == 1)
return("Group 2")
      if (!is.na(x["groups_maf10_3"]) && x["groups_maf10_3"] == 1)
return("Group 3")
      if (!is.na(x["groups_maf10_NA"]) && x["groups_maf10_NA"] == 1)
return("Group NA")
      return(NA)
    })
  ))

  ggplot(tsne_plot) +
    geom_point(aes(x = x, y = y, color = Groups), size = pointSize) +
    scale_color_manual(values = mycolors) +
    ggtitle(paste0("Perplexity=", perplexity, ", PCA_dimension=", pcaDim))
+
    xlab("Dimension 1") +
    ylab("Dimension 2") +
    theme_bw() +
    theme(text = element_text(size = textSize), legend.key.size =
unit(legendSize, "point"))
}
```

Define your datasets

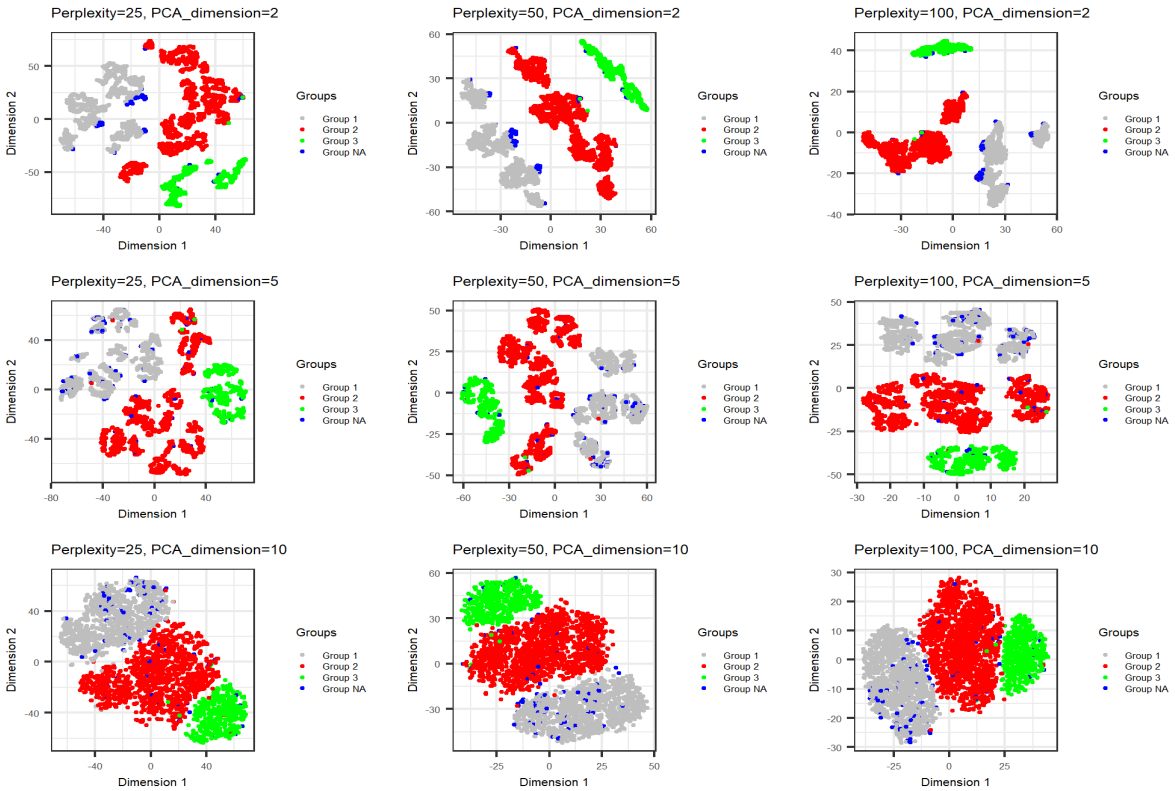
```
datasets <- list(
  u_meanimprove = u_meanimprove,
  n_meanimprove = n_meanimprove,
  u_knnimprove = u_knnimprove,
  n_knnimprove = n_knnimprove,
  u_ccaimprove = u_ccaimprove,
  n_ccaimprove = n_ccaimprove
)
```

Iterate over datasets

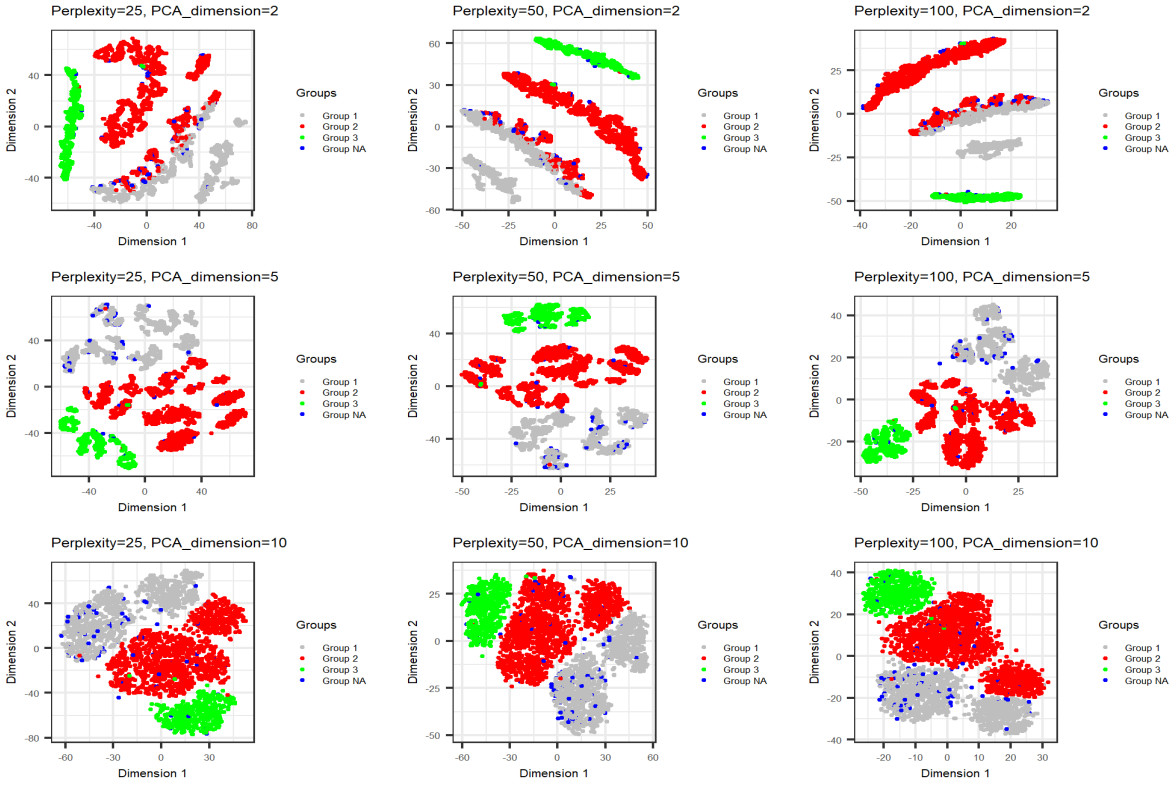
```
for (datname in names(datasets)) {
  dat <- datasets[[datname]]
```

```
  # Check for required grouping columns
  if (all(c("groups_maf10_1", "groups_maf10_2", "groups_maf10_3",
"groups_maf10_NA") %in% names(dat))) {


    # Initialize a list to store plots
    pls <- list()


    # Perform t-SNE and plot for each PCA dimension and perplexity combination
    for (pcaDim in pcaDims) {
      plots <- lapply(perplexities, function(perplexity)
doRtsne(perplexity, pcaDim))
      pls <- c(pls, plots)
    }


  # Arrange plots in a grid
  grid_plot_filename <- paste0("tsne_2d_grid_", datname, ".png")
    png(grid_plot_filename, width = figWidth, height = figWidth * 0.75,
units = "px", res = 300)
    grid.arrange(grobs = pls, nrow = length(pcaDims), ncol =
length(perplexities))
    dev.off()


  } else {
    warning(paste("Some one-hot encoded columns are missing in", datname))
  }
}
```

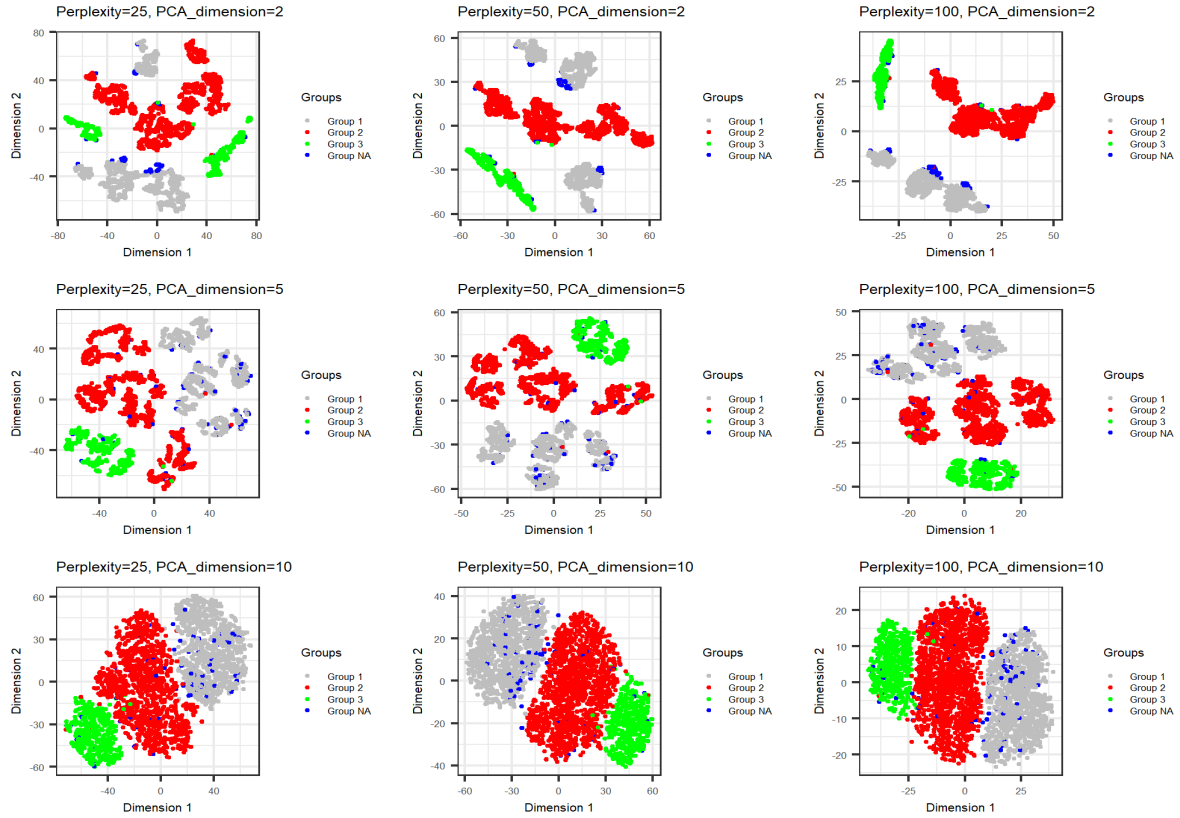
u_mean tSNE Results:



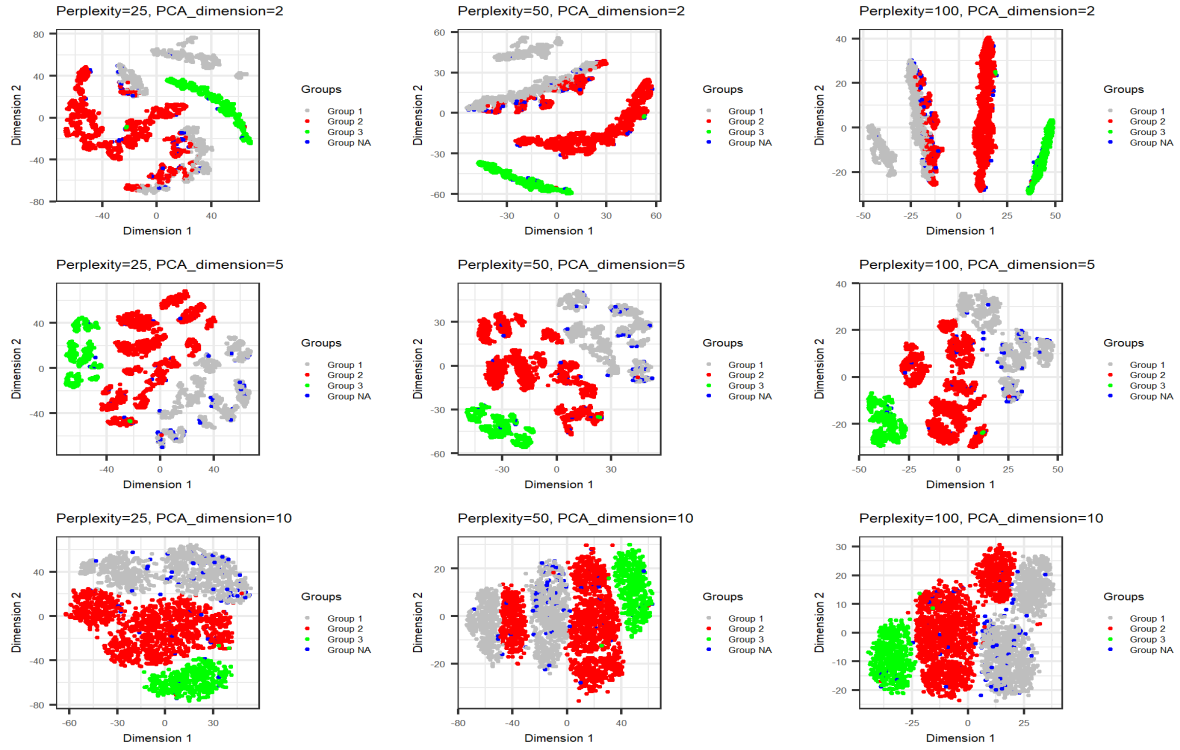
n_mean tSNE Results:



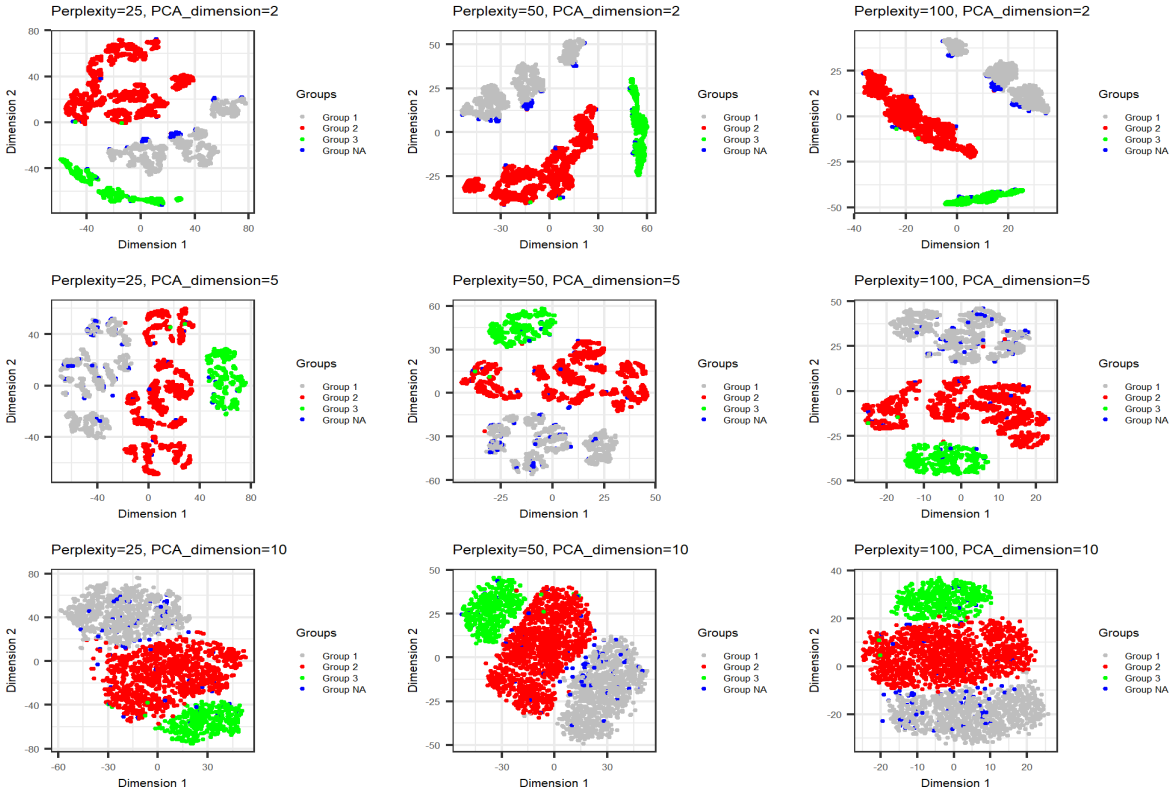
u_knn tSNE Results:



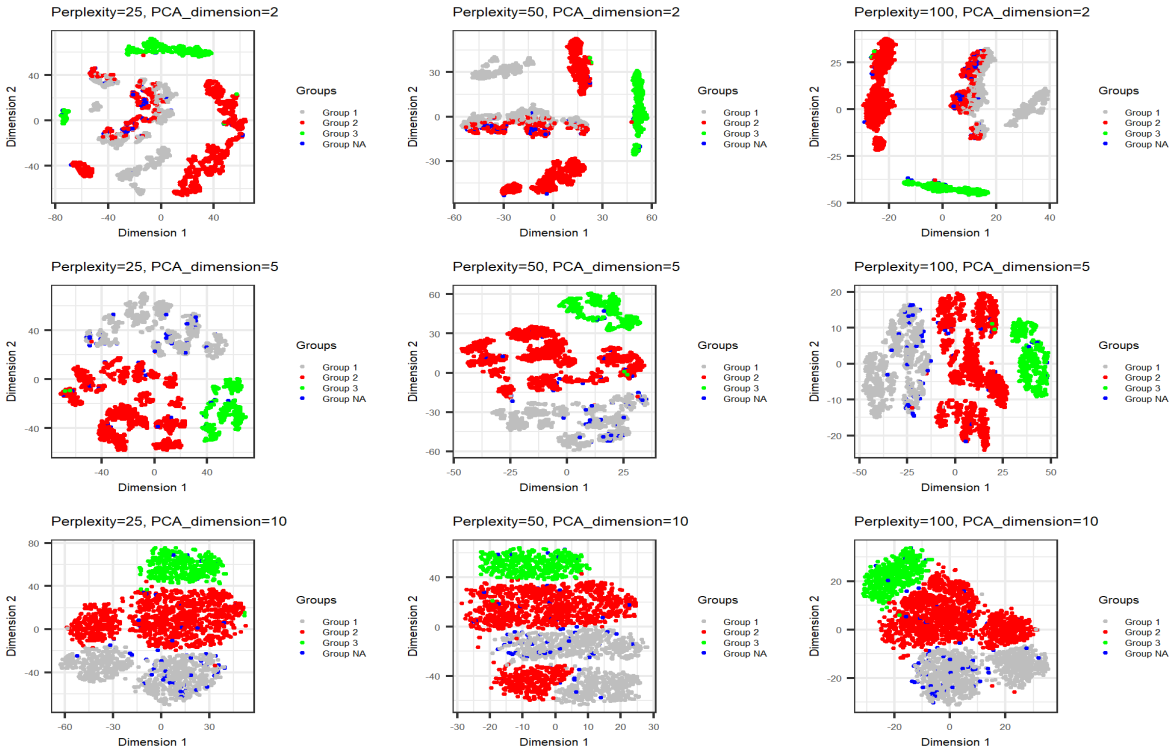
n_knn tSNE Results:



u_cca tSNE Results:



n_cca tSNE Results:



Since we know that perplexity between 5 – 50 are more robust and give better results, I will redo the tSNE with lower perplexities, I will increase the maximum number of iterations to 1500 just to make sure that the iterations to give tSNE separations are complete, and I will set seed for reproducibility.

#Key Changes

#1. Set Seed: Added set.seed(42) at the beginning for reproducibility.

#2. Increase Iterations: Updated max_iter to 1500.

#3. Update Perplexity: Changed perplexities to c(10, 20, 30).

Load necessary libraries

```
library(Rtsne)

library(ggplot2)

library(gridExtra)  # For arranging plots in a grid
```

Define parameters

```
set.seed(42)  # Set seed for reproducibility

max_iter <- 1500  # Increase maximum number of iterations

theta <- 0.1

perplexities <- c(10, 20, 30)  # Updated perplexity values

pcaDims <- c(2, 5, 10)

mycolors <- c("Group 1" = "gray", "Group 2" = "red", "Group 3" = "green",
"Group NA" = "blue")

figWidth <- 2000

pointSize <- 0.5

legendSize <- 5

textSize <- 5

num_threads <- 0
```

Function to perform t-SNE with PCA initialization and create plots

```
doRtsne <- function(perplexity, pcaDim) {

  tsne <- Rtsne(dat[ , !(names(dat) %in% c("iid", "FID", "groups_maf10_1",
"groups_maf10_2", "groups_maf10_3", "groups_maf10_NA"))],

                initial_dims = pcaDim,

                dims = 2,

                perplexity = perplexity,
```

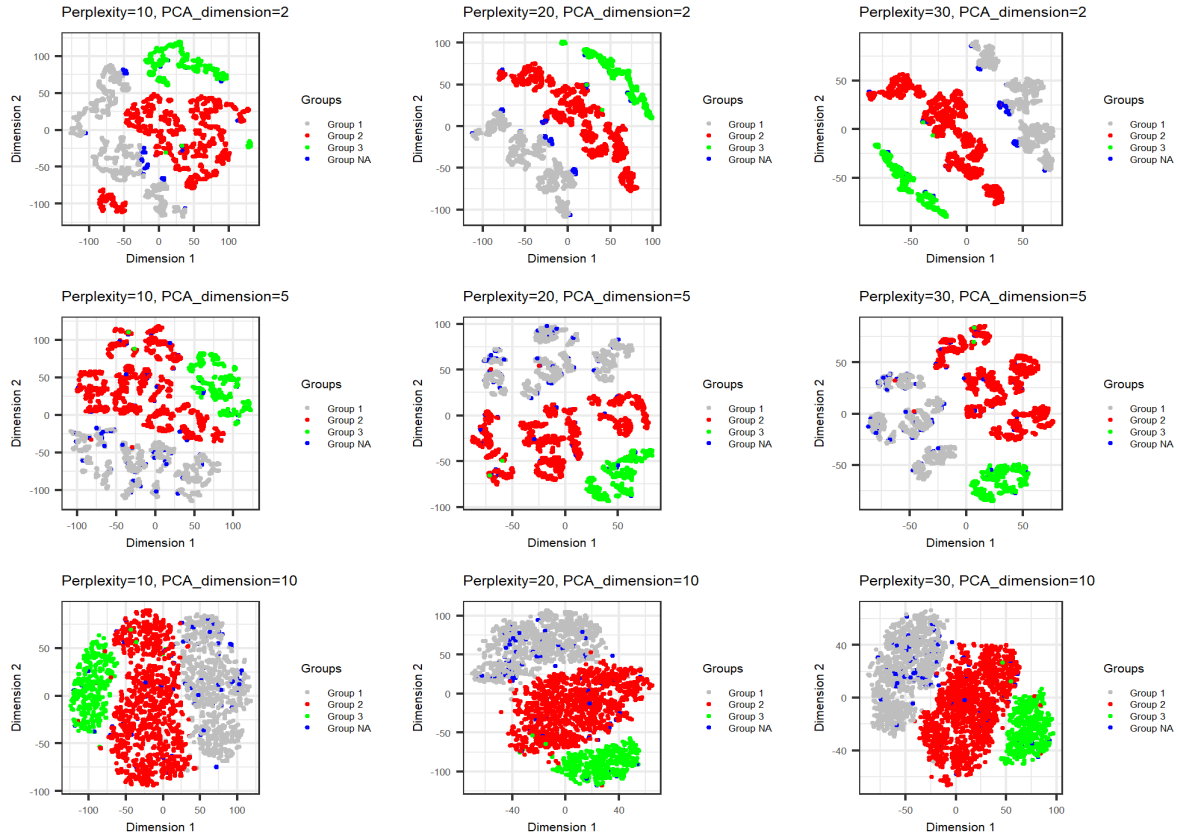
```
                verbose = TRUE,
                max_iter = max_iter,
                theta = theta,
                num_threads = num_threads)


  tsne_plot <- data.frame(x = tsne$Y[, 1], y = tsne$Y[, 2], Groups =
factor(
    apply(dat[, c("groups_maf10_1", "groups_maf10_2", "groups_maf10_3",
"groups_maf10_NA")], 1, function(x) {
      if (!is.na(x["groups_maf10_1"]) && x["groups_maf10_1"] == 1)
return("Group 1")
      if (!is.na(x["groups_maf10_2"]) && x["groups_maf10_2"] == 1)
return("Group 2")
      if (!is.na(x["groups_maf10_3"]) && x["groups_maf10_3"] == 1)
return("Group 3")
      if (!is.na(x["groups_maf10_NA"]) && x["groups_maf10_NA"] == 1)
return("Group NA")
      return(NA)
    })
  ))


  ggplot(tsne_plot) +
    geom_point(aes(x = x, y = y, color = Groups), size = pointSize) +
    scale_color_manual(values = mycolors) +
    ggtitle(paste0("Perplexity=", perplexity, ", PCA_dimension=", pcaDim))
+
    xlab("Dimension 1") +
    ylab("Dimension 2") +
    theme_bw() +
    theme(text = element_text(size = textSize), legend.key.size =
unit(legendSize, "point"))
}
```

Define your datasets

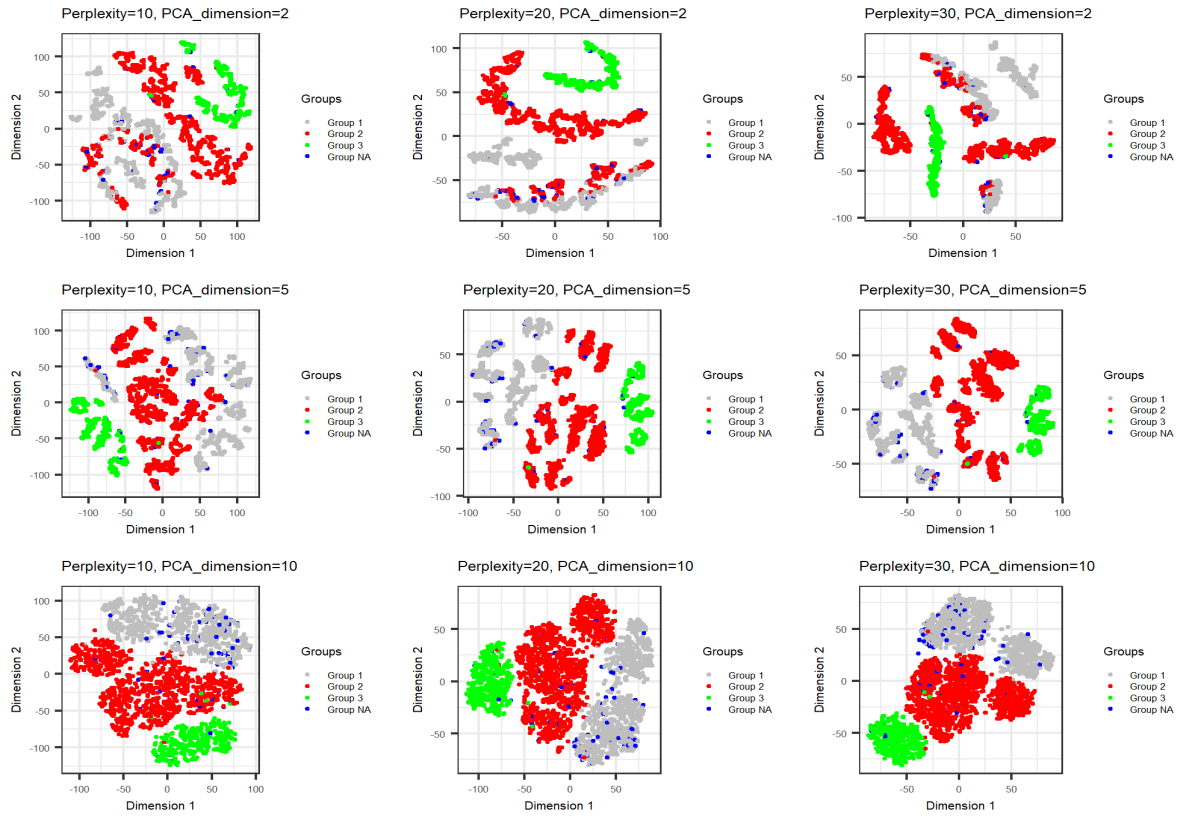
```
datasets <- list(
  u_meanimprove = u_meanimprove,
  n_meanimprove = n_meanimprove,
  u_knnimprove = u_knnimprove,
```

```
  n_knnimprove = n_knnimprove,
  u_ccaimprove = u_ccaimprove,
  n_ccaimprove = n_ccaimprove
)

# Iterate over datasets
for (datname in names(datasets)) {
  dat <- datasets[[datname]]

  if (all(c("groups_maf10_1", "groups_maf10_2", "groups_maf10_3",
"groups_maf10_NA") %in% names(dat))) {

    pls <- list()

    for (pcaDim in pcaDims) {
      plots <- lapply(perplexities, function(perplexity)
doRtsne(perplexity, pcaDim))
      pls <- c(pls, plots)
    }

    grid_plot_filename <- paste0("tsne_2d_grid_", datname, ".png")
    png(grid_plot_filename, width = figWidth, height = figWidth * 0.75,
units = "px", res = 300)
    grid.arrange(grobs = pls, nrow = length(pcaDims), ncol =
length(perplexities))
    dev.off()

  } else {
    warning(paste("Some one-hot encoded columns are missing in", datname))
  }
}
```

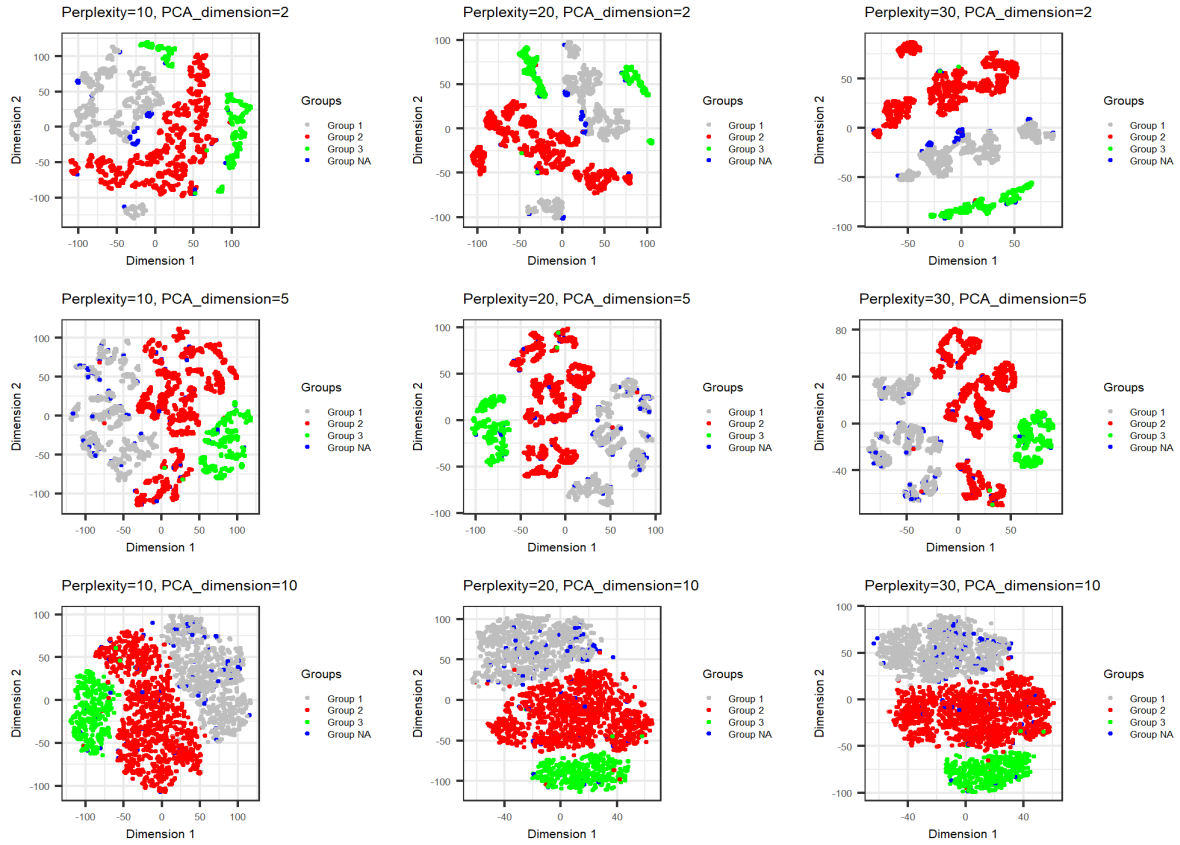
u_mean tSNE result:



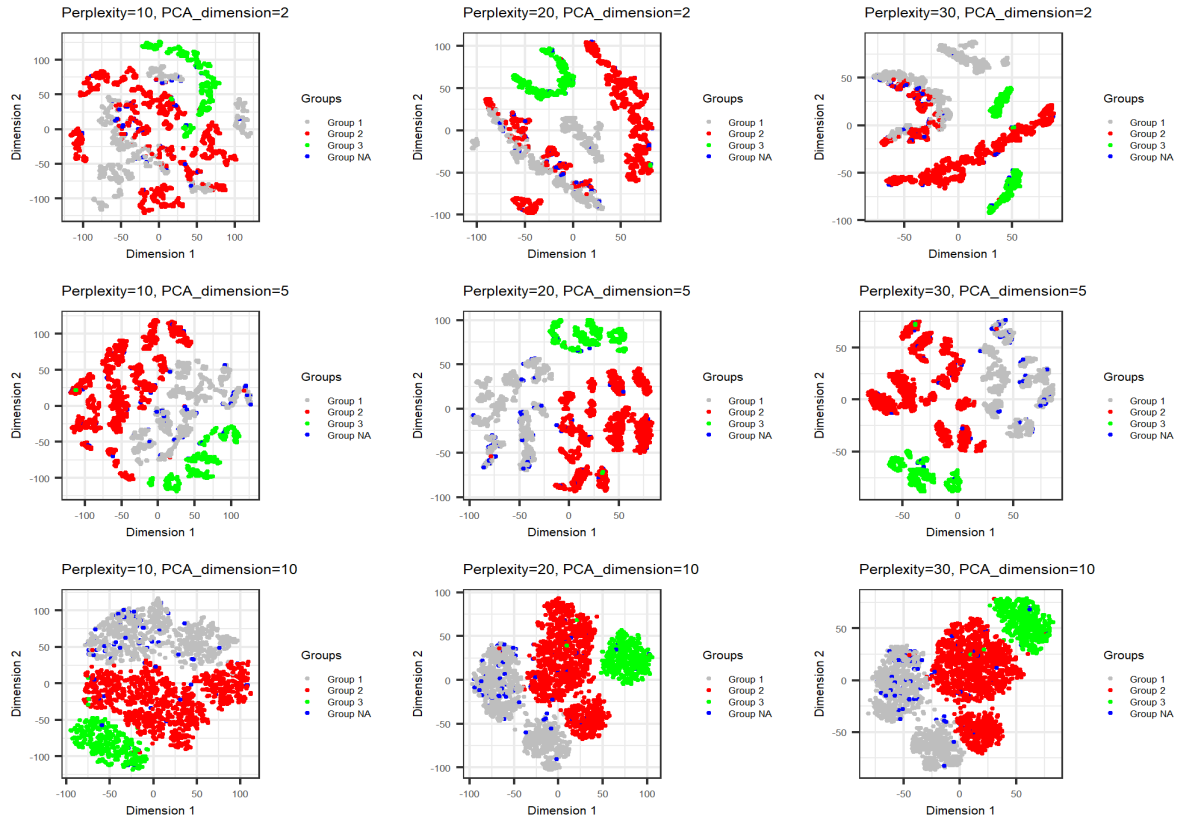
n_mean tSNE result:



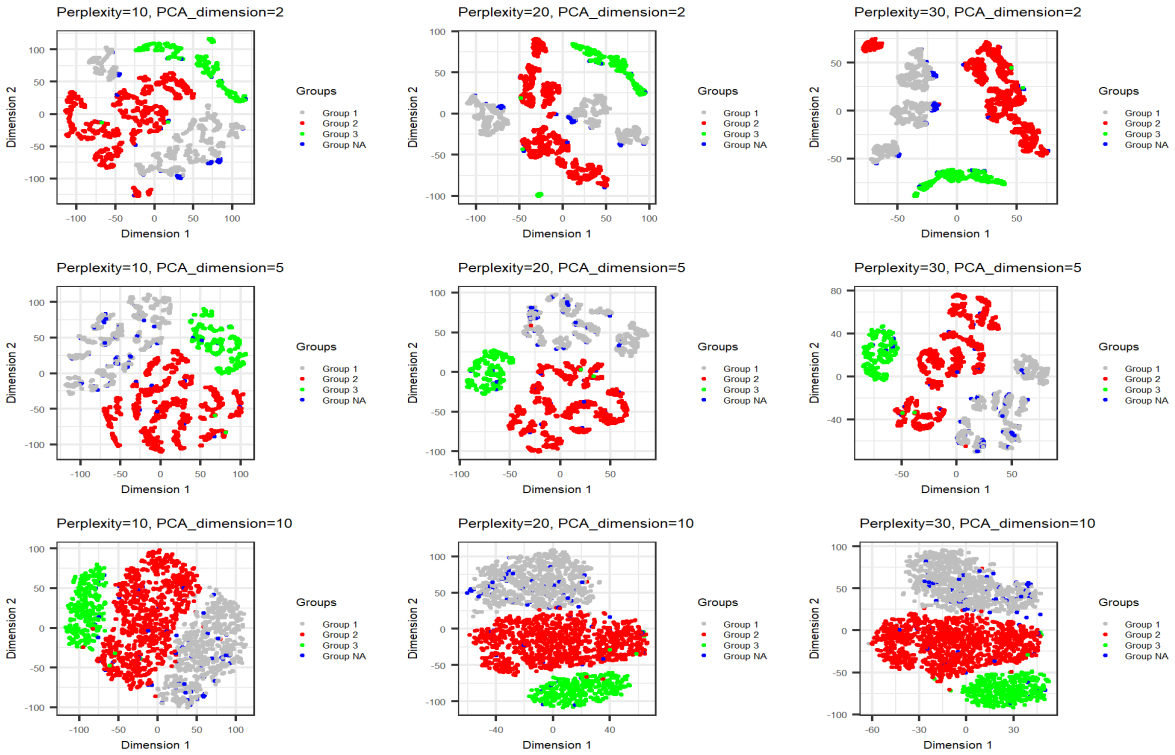
u_knn tSNE result:



n_knn tSNE result:



u_cca tSNE result:



n_cca tSNE result: