```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(rpart)          # For Decision Tree
```

```
## Warning: package 'rpart' was built under R version 4.3.3
```

```
library(MASS)           # For LDA & generalized analysis
library(car)
```

```
## Loading required package: carData
```

```
library(e1071)          # For Naive Bayes
library(class)          # For k-NN
library(glmnet)         # For Logistic Regression
```

```
## Warning: package 'glmnet' was built under R version 4.3.3
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
library(tidymodels)     # For modeling and evaluation
```

```
## Warning: package 'tidymodels' was built under R version 4.3.3
```

```
## ── Attaching packages ────────────────────────────────────── tidymodels 1.1.1 ──
```

```
## ✓ broom        1.0.5     ✓ rsample      1.2.0
## ✓ dials        1.2.1     ✓ tibble       3.2.1
## ✓ dplyr        1.1.4     ✓ tidyr        1.3.0
## ✓ infer        1.0.6     ✓ tune         1.1.2
## ✓ modeldata    1.3.0     ✓ workflows    1.1.4
## ✓ parsnip      1.2.0     ✓ workflowsets 1.0.1
## ✓ purrr        1.0.2     ✓ yardstick    1.3.0
## ✓ recipes      1.0.9
```

```
## Warning: package 'dials' was built under R version 4.3.3
```

```
## Warning: package 'infer' was built under R version 4.3.3
```

```
## Warning: package 'modeldata' was built under R version 4.3.3
```

```
## Warning: package 'parsnip' was built under R version 4.3.3
```

```
## Warning: package 'rsample' was built under R version 4.3.3
```

```
## Warning: package 'tune' was built under R version 4.3.3
```

```
## Warning: package 'workflows' was built under R version 4.3.3
```

```
## Warning: package 'workflowsets' was built under R version 4.3.3
```

```
## Warning: package 'yardstick' was built under R version 4.3.3
```

```
## ── Conflicts ───────────────────────────────────────── tidymodels_conflicts() ──
## ✖ dplyr::combine()         masks randomForest::combine()
## ✖ purrr::discard()         masks scales::discard()
## ✖ tidyr::expand()          masks Matrix::expand()
## ✖ dplyr::filter()          masks stats::filter()
## ✖ dplyr::lag()             masks stats::lag()
## ✖ purrr::lift()            masks caret::lift()
## ✖ randomForest::margin()   masks ggplot2::margin()
## ✖ tidyr::pack()            masks Matrix::pack()
## ✖ rsample::permutations()  masks e1071::permutations()
## ✖ yardstick::precision()   masks caret::precision()
## ✖ dials::prune()           masks rpart::prune()
## ✖ yardstick::recall()      masks caret::recall()
## ✖ dplyr::recode()          masks car::recode()
## ✖ dplyr::select()          masks MASS::select()
## ✖ yardstick::sensitivity() masks caret::sensitivity()
## ✖ purrr::some()            masks car::some()
## ✖ yardstick::specificity() masks caret::specificity()
## ✖ recipes::step()          masks stats::step()
## ✖ tune::tune()             masks parsnip::tune(), e1071::tune()
## ✖ tidyr::unpack()          masks Matrix::unpack()
## ✖ recipes::update()        masks Matrix::update(), stats::update()
## • Use suppressPackageStartupMessages() to eliminate package startup messages
```

```r
library(pROC)           # For ROC curve analysis
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(xgboost)        # For XGBoost analysis
```

```
## Warning: package 'xgboost' was built under R version 4.3.3
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:purrr':
##
##     cross
```

```
## The following object is masked from 'package:scales':
##
##     alpha
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(pROC)
library(Rtsne)
```

```
## Warning: package 'Rtsne' was built under R version 4.3.3
```

```
library(ggplot2)
```

Data Exploration and Analysis:

Loading and Understanding the Raw Dataset:

```
# Setting up the working directory:
setwd("C://Users//Maisam//Downloads//4. Assignments//MA 321-7  ; Applied Statistics ; Team Proje
ct")
getwd()
```

```
## [1] "C:/Users/Maisam/Downloads/4. Assignments/MA 321-7  ; Applied Statistics ; Team Project"
```

```
# Loading the raw dataset
initial_data <- read.csv(file="gene-expression-invasive-vs-noninvasive-cancer.csv")

# Understanding the structure of raw dataset
str(initial_data[,1:10])
```

```
## 'data.frame':    78 obs. of  10 variables:
##  $ J00129       : num  -0.448 -0.48 -0.568 -0.819 -0.112 -0.391 -0.624 -0.528 -0.811 -0.839
...
##  $ Contig29982_RC: num  -0.296 -0.512 -0.411 -0.267 -0.67 -0.31 -0.12 -0.447 -0.536 2 ...
##  $ Contig42854   : num  -0.1 -0.031 -0.398 0.023 0.421 -0.06 -0.236 -0.254 -0.211 0.147 ...
##  $ Contig42014_RC: num  -0.177 -0.075 0.116 -0.23 -0.19 -0.164 -0.175 0.017 -0.201 -0.325 ...
##  $ Contig27915_RC: num  -0.107 -0.104 -0.092 0.198 0.032 -0.173 0.253 0.654 0.287 -0.303 ...
##  $ Contig20156_RC: num  -0.11 -0.234 -0.166 -0.51 0.281 -0.034 -0.125 0.364 -0.08 -0.061 ...
##  $ Contig50634_RC: num  -0.095 -0.225 0.036 0.529 0.31 -0.091 -0.127 0.068 -0.15 0.097 ...
##  $ Contig42615_RC: num  -0.076 -0.094 0.397 0.354 0.056 0.036 -0.02 0.181 0.045 0.006 ...
##  $ Contig56678_RC: num  -0.134 0.115 -0.194 -0.261 0.116 0.346 0.047 -1.14 -0.11 0.176 ...
##  $ Contig48659_RC: num  -0.14 0.019 -0.128 0.012 0.074 0.007 -0.15 -0.111 -0.072 -0.084 ...
```

```
# Verifying the class label column name
dimnames(initial_data)[[2]][4947:4949]
```

```
## [1] "NM_000898" "AF067420"  "Class"
```

```
# Tabulating the frequencies of class label column
table(initial_data[4949])
```

```
## Class
##  1  2
## 34 44
```

Removing Nulls, NA Values and Detecting Outliers:

```
# Check for missing values
na_count <- sum(is.na(initial_data))
na_count
```

```
## [1] 177
```

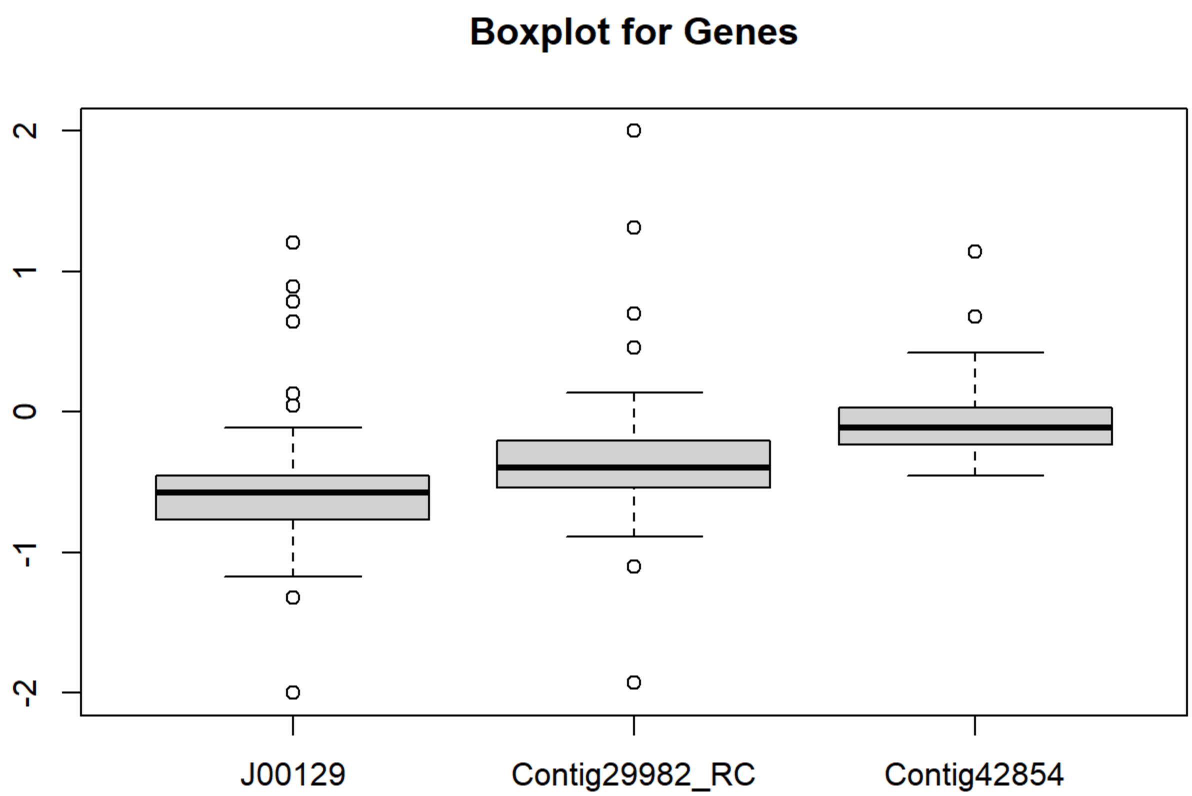
```r
# Check for infinite values
inf_count <- sum(sapply(initial_data, function(x) sum(is.infinite(x))))
inf_count
```

```
## [1] 0
```

```r
# Handling missing values
# install.packages('zoo')
library(zoo)

# replace missing values with mean
processed_data <- na.aggregate(initial_data, FUN = mean)
na_count <- sum(is.na(processed_data))
na_count
```

```
## [1] 0
```

```r
# Identify outliers
# Example: Z-score method for identifying outliers in the first gene column
z_scores <- scale(processed_data[,1])
outliers <- which(abs(z_scores) > 3)

# Visualize outliers (for the first gene as an example)
boxplot(processed_data[,1:3], main="Boxplot for Genes")
```

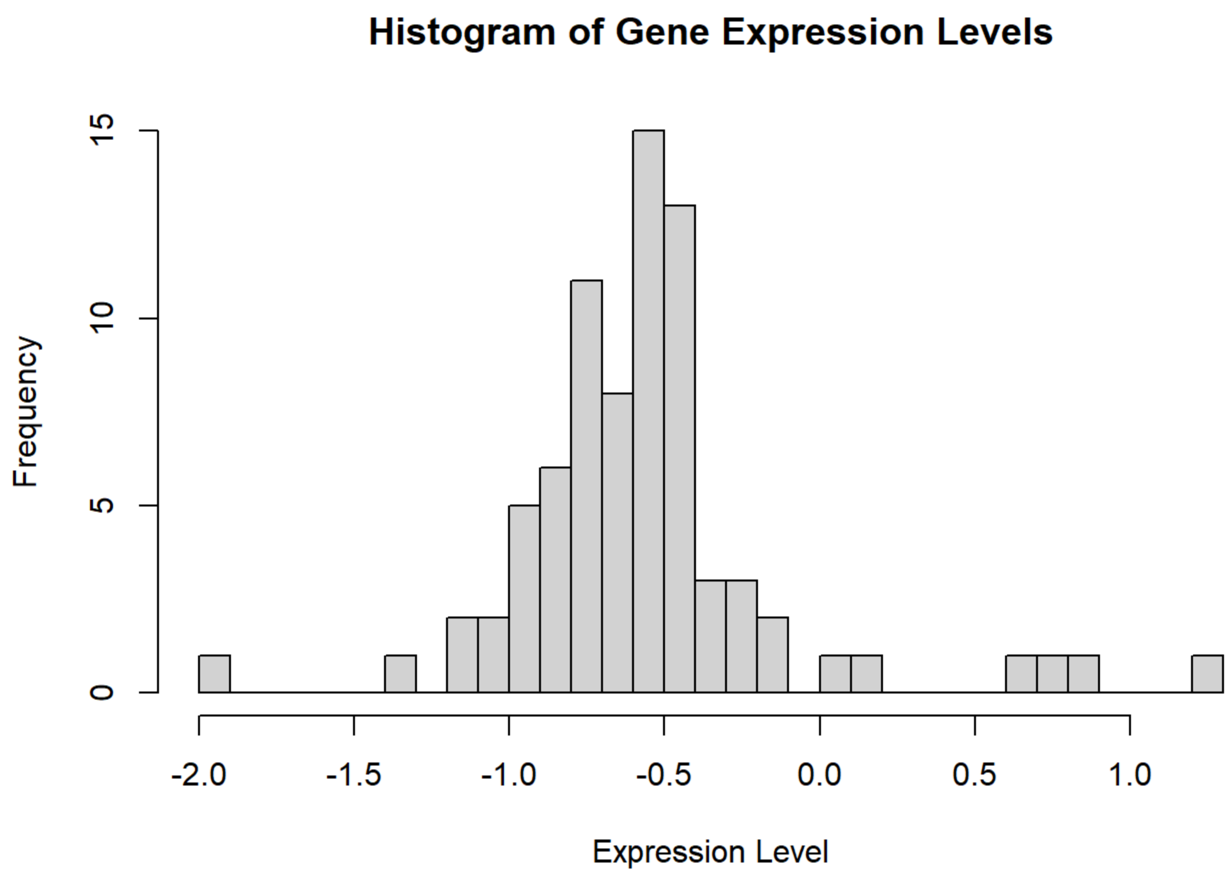


Statistical Measures & Analysis:

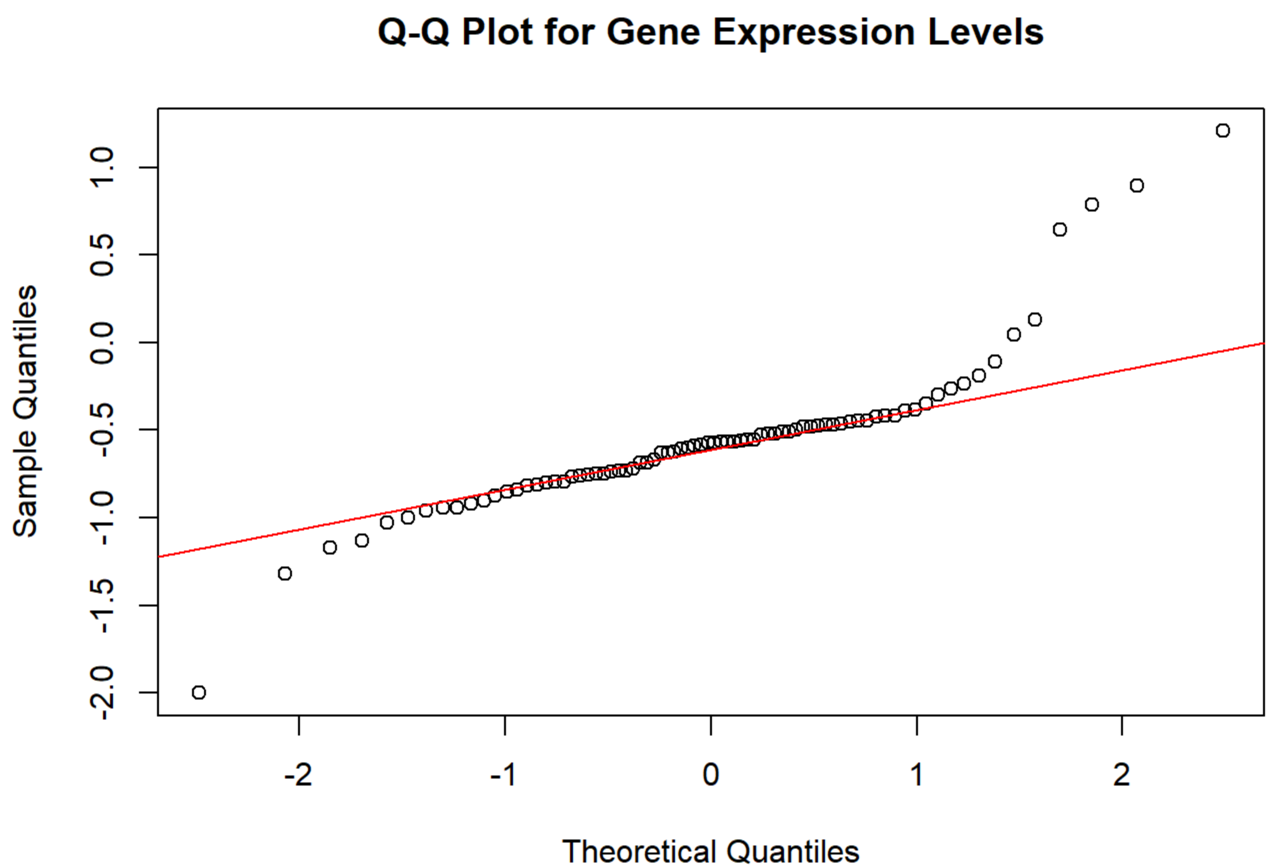
```r
# Exclude the column with labels for invasive vs. noninvasive cancer if present
genes_data <- processed_data[, !names(processed_data) %in% c('class')]

# Calculate statistical measures
mean_values <- apply(genes_data, 2, mean)

median_values <- apply(genes_data, 2, median)

std_dev_values <- apply(genes_data, 2, sd)

variance_values <- apply(genes_data, 2, var)

range_values <- apply(genes_data, 2, function(x) max(x) - min(x))

statistical_measures <- data.frame(
  Mean = mean_values,
  Median = median_values,
  StandardDeviation = std_dev_values,
  Variance = variance_values,
  Range = range_values
)

# Display the statistical measures for the first gene column
# Assuming the first gene corresponds to the first column in the original data
first_gene_measures <- statistical_measures[1:5,]
print(first_gene_measures)
```

```
##                        Mean Median StandardDeviation   Variance Range
## J00129         -0.55766667 -0.574         0.4499578 0.20246199 3.206
## Contig29982_RC -0.33938462 -0.396         0.4673664 0.21843138 3.927
## Contig42854    -0.07625641 -0.110         0.2508432 0.06292232 1.593
## Contig42014_RC -0.03465385 -0.058         0.2094581 0.04387270 1.292
## Contig27915_RC -0.04126923 -0.090         0.2436339 0.05935750 1.127
```

```r
# Distribution of gene expression levels - Histogram for a single gene as an example
hist(genes_data[,1], breaks = 30, main = "Histogram of Gene Expression Levels", xlab = "Expressi
on Level", ylab = "Frequency")
```




```
# Assessing normality - Q-Q plot for the same gene as an example
qqnorm(genes_data[,1], main = "Q-Q Plot for Gene Expression Levels")
qqline(genes_data[,1], col = "red")
```




```
# Assuming 'genes_data' is your cleaned and preprocessed dataset, excluding the label column

# Calculate variance for each gene
gene_variances <- apply(genes_data, 2, var)

# You can set a threshold for variance to filter genes
# For example, selecting genes with variance in the top 25%
variance_threshold <- quantile(gene_variances, 0.75)
high_variance_genes <- names(gene_variances[gene_variances > variance_threshold])

# Calculate the Coefficient of Variation (CV) for each gene
gene_means <- apply(genes_data, 2, mean)
gene_sd <- apply(genes_data, 2, sd)
gene_cv <- gene_sd / gene_means

# Optionally, filter genes based on CV
# For example, selecting genes with CV in the top 25%
cv_threshold <- quantile(gene_cv, 0.75)
high_cv_genes <- names(gene_cv[gene_cv > cv_threshold])

# Summary of results
cat("Number of genes with high variance:", length(high_variance_genes), "\n")
```

```
## Number of genes with high variance: 1237
```

```r
cat("Number of genes with high CV:", length(high_cv_genes), "\n")
```

```
## Number of genes with high CV: 1237
```

Generating a Random Subset:

```r
# Setting up random seed for reproducibility
registration_number <- 2315740 #Mantosh
set.seed(registration_number)

# Generating random subset of 2000 features
subset_indices <- sample(1:(ncol(initial_data) - 1), 2000)
subset_indices[1:20]
```

```
##  [1] 2692 2091 3641 2769 4909 1672 3200 1142 3892 2575 2307 1138  855 3066  334
## [16] 3848 1950 2667 3749 2881
```

```r
subset_df <- initial_data[ , subset_indices]

# Validating and understanding the random subset
dim(subset_df)
```

```
## [1]   78 2000
```

Preprocessing the Randomly Generated Subset:

```r
# Checking null values
na_count <- sum(is.na(subset_df))
na_count
```

```
## [1] 65
```

```r
# Replacing null values with column means
subset_df <- na.aggregate(subset_df, FUN = mean)

# Validating null values
na_count <- sum(is.na(subset_df))
na_count
```

```
## [1] 0
```

```r
# Checking for infinte values / error values
inf_count <- sum(sapply(subset_df, function(x) sum(is.infinite(x))))
inf_count
```

```
## [1] 0
```

Understanding Correlation Between Variables:

```
# Craeting a correlation matrix
correlation <- cor(subset_df)

# Creating highly correlated pairs
highly_correlated_pairs <- which(correlation > 0.7 & correlation < 1, arr.ind = TRUE)

# Removing one feature for each pair
features_to_remove <- character(0)
for (i in 1:nrow(highly_correlated_pairs)) {
  feature1 <- colnames(subset_df)[highly_correlated_pairs[i, 1]]
  feature2 <- colnames(subset_df)[highly_correlated_pairs[i, 2]]
  if (!(feature1 %in% features_to_remove)) {
    features_to_remove <- c(features_to_remove, feature2)
  }
}

# Creating a filtered df with removed features
filtered_df <- subset_df[, !colnames(subset_df) %in% features_to_remove]
dim(filtered_df)
```

```
## [1]   78 1429
```

PART 1: Dimensionality Reduction:

I) Consider unsupervised and supervised dimension reduction of the 2000 observed gene expression values in your data set.

Applying Principal Component Analysis for Dimensionality Reduction:

```
# Performing PCA while scaling the dataset
pca <- prcomp(filtered_df, scale. = TRUE , center = TRUE)
summary(pca)
```

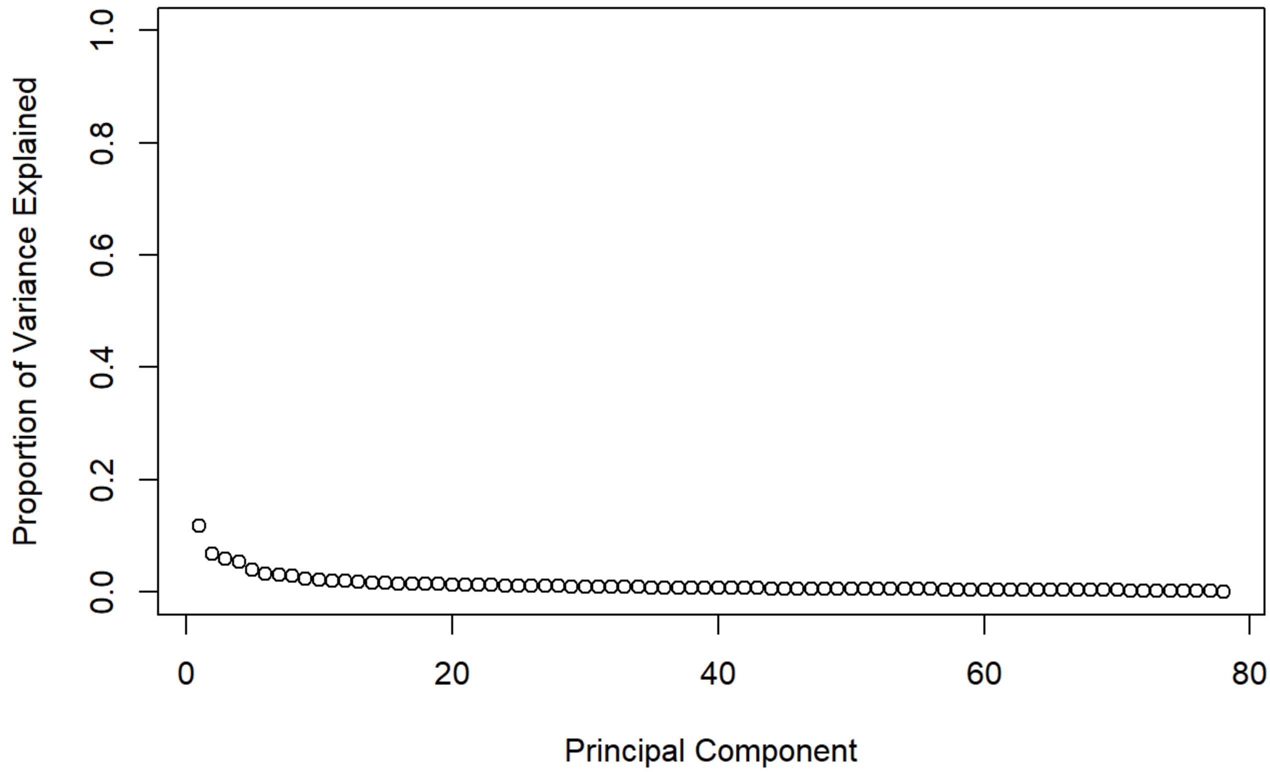
```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     13.0223 9.8284 9.25056 8.76649 7.59301 6.76422 6.56015
## Proportion of Variance  0.1187 0.0676 0.05988 0.05378 0.04035 0.03202 0.03012
## Cumulative Proportion   0.1187 0.1863 0.24615 0.29993 0.34028 0.37230 0.40241
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     6.43962 5.88419 5.55992 5.34519 5.21624 5.05868 4.80252
## Proportion of Variance 0.02902 0.02423 0.02163 0.01999 0.01904 0.01791 0.01614
## Cumulative Proportion  0.43143 0.45566 0.47729 0.49729 0.51633 0.53424 0.55038
##                           PC15    PC16    PC17    PC18    PC19    PC20   PC21
## Standard deviation     4.74818 4.66365 4.62757 4.53056 4.44574 4.36092 4.3106
## Proportion of Variance 0.01578 0.01522 0.01499 0.01436 0.01383 0.01331 0.0130
## Cumulative Proportion  0.56615 0.58137 0.59636 0.61072 0.62455 0.63786 0.6509
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     4.21539 4.13073 4.07541 3.98923 3.96891 3.90878 3.84235
## Proportion of Variance 0.01243 0.01194 0.01162 0.01114 0.01102 0.01069 0.01033
## Cumulative Proportion  0.66330 0.67524 0.68686 0.69800 0.70902 0.71971 0.73005
##                           PC29    PC30    PC31    PC32    PC33    PC34    PC35
## Standard deviation     3.72961 3.70281 3.62993 3.61303 3.51768 3.47629 3.43040
## Proportion of Variance 0.00973 0.00959 0.00922 0.00914 0.00866 0.00846 0.00823
## Cumulative Proportion  0.73978 0.74937 0.75859 0.76773 0.77639 0.78485 0.79308
##                           PC36    PC37    PC38    PC39    PC40    PC41    PC42
## Standard deviation     3.37112 3.33194 3.31362 3.29383 3.22428 3.17435 3.15009
## Proportion of Variance 0.00795 0.00777 0.00768 0.00759 0.00727 0.00705 0.00694
## Cumulative Proportion  0.80103 0.80880 0.81649 0.82408 0.83135 0.83840 0.84535
##                           PC43    PC44    PC45    PC46    PC47    PC48    PC49
## Standard deviation     3.10326 3.05101 3.00554 2.98243 2.94585 2.92674 2.88405
## Proportion of Variance 0.00674 0.00651 0.00632 0.00622 0.00607 0.00599 0.00582
## Cumulative Proportion  0.85209 0.85860 0.86492 0.87115 0.87722 0.88322 0.88904
##                           PC50    PC51    PC52   PC53    PC54    PC55    PC56
## Standard deviation     2.85896 2.81952 2.78296 2.7791 2.76539 2.70510 2.67489
## Proportion of Variance 0.00572 0.00556 0.00542 0.0054 0.00535 0.00512 0.00501
## Cumulative Proportion  0.89476 0.90032 0.90574 0.9111 0.91649 0.92162 0.92662
##                           PC57   PC58    PC59    PC60    PC61    PC62    PC63
## Standard deviation     2.61270 2.5917 2.53919 2.49525 2.45853 2.42539 2.40288
## Proportion of Variance 0.00478 0.0047 0.00451 0.00436 0.00423 0.00412 0.00404
## Cumulative Proportion  0.93140 0.9361 0.94061 0.94497 0.94920 0.95332 0.95736
##                           PC64    PC65    PC66    PC67    PC68    PC69    PC70
## Standard deviation     2.35260 2.29672 2.28423 2.23989 2.19872 2.14353 2.09731
## Proportion of Variance 0.00387 0.00369 0.00365 0.00351 0.00338 0.00322 0.00308
## Cumulative Proportion  0.96123 0.96492 0.96857 0.97208 0.97547 0.97868 0.98176
##                           PC71    PC72   PC73    PC74    PC75    PC76    PC77
## Standard deviation     2.06713 2.02591 2.0012 1.94920 1.89281 1.80437 1.74541
## Proportion of Variance 0.00299 0.00287 0.0028 0.00266 0.00251 0.00228 0.00213
## Cumulative Proportion  0.98475 0.98762 0.9904 0.99308 0.99559 0.99787 1.00000
##                             PC78
## Standard deviation     5.783e-15
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

```r
# Analyzing the structure of PCA
str(pca)
```

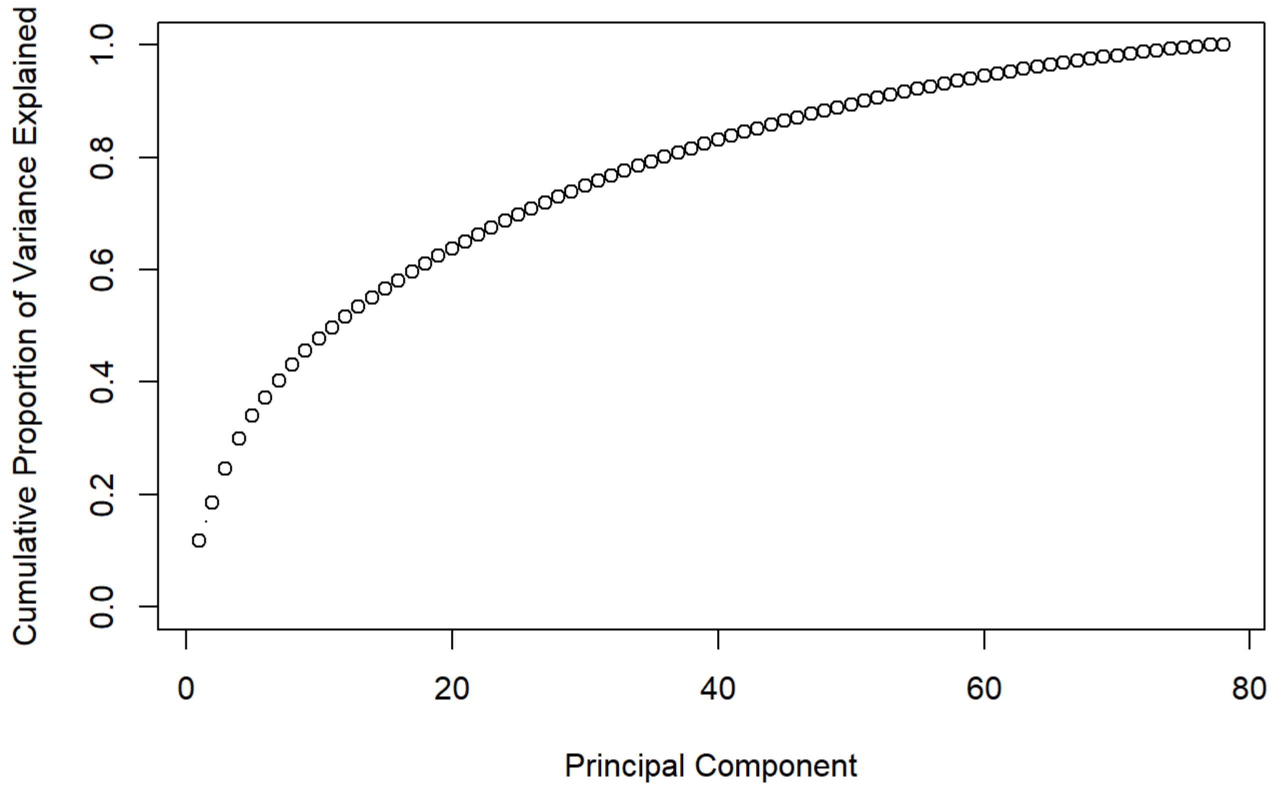
```
## List of 5
##  $ sdev    : num [1:78] 13.02 9.83 9.25 8.77 7.59 ...
##  $ rotation: num [1:1429, 1:78] -0.0222 -0.0257 -0.036 -0.0214 0.0295 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:1429] "NM_016073" "NM_004864" "NM_006006" "NM_016371" ...
##   .. ..$ : chr [1:78] "PC1" "PC2" "PC3" "PC4" ...
##  $ center  : Named num [1:1429] -0.031 -0.322 -0.1299 -0.0994 -0.0215 ...
##   ..- attr(*, "names")= chr [1:1429] "NM_016073" "NM_004864" "NM_006006" "NM_016371" ...
##  $ scale   : Named num [1:1429] 0.157 0.445 0.383 0.208 0.137 ...
##   ..- attr(*, "names")= chr [1:1429] "NM_016073" "NM_004864" "NM_006006" "NM_016371" ...
##  $ x       : num [1:78, 1:78] -13.14 -9.99 -1.32 1.01 -5.24 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:78] "PC1" "PC2" "PC3" "PC4" ...
##  - attr(*, "class")= chr "prcomp"
```

Plotting Explained and Cumulative Variance for Each Principal Component:

```r
pca.var <- pca$sdev^2
pve <- pca.var / sum(pca.var)

# Plotting variance explained for each principal component
plot(pve,
     xlab = "Principal Component",
     ylab = "Proportion of Variance Explained",
     ylim = c(0,1),
     type = "b")
```




```
plot(cumsum(pve), xlab = "Principal Component",
     ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1), type = "b")
```



Appending and Visualizing Results from PCA:

```
# Sub-setting PC's at 80% explained variation threshold:
pca_df <- as.data.frame(pca$x[, 1:36])

# Plotting interaction between top 2 principal components:
pca_x <- pca_df[,1]
pca_y <- pca_df[,2]
class <- initial_data$Class

pca_xyc <- as.data.frame( cbind(pca_x,pca_y,class) )

pca_plot <- ggplot(pca_xyc, aes(x = pca_x, y = pca_y, color = factor(class))) +
            geom_point() +
            ggtitle("PCA with Classes") +
            xlab("PC1") + ylab("PC2") +
            coord_fixed(ratio = 1) +
            theme_bw() +
            theme(aspect.ratio = 1) +
            theme(panel.grid = element_blank())

pca_plot
```

Appending and Storing Results in Dataframe:

```
# Creating dataframe for future usability
Class <- initial_data$Class

final_df <- cbind(filtered_df,pca_df,Class) # Dataframe containing all feature columns, top 50 p
rincipal component columns and class label column

pca_df_w_class <- cbind(pca_df,Class) # Dataframe containing top 36 principal component columns
and class label column

filtered_df_w_class <- cbind(filtered_df,Class) # Dataframe containing all feature columns and c
lass label column
```

```
# Breakup of class label column in the PCA reduced dataframe
table(pca_df_w_class[37])
```

```
## Class
##  1  2
## 34 44
```

PCA - for Dimensionality Reduction vs Supervised Algorithms:

Given the nature of our dataset, with a limited number of observations and large dimensions PCA over LDA or any supervised learning technique for dimensionality reduction suits us for following reasons:

1. Compatibility with Clustering
2. Compatibility with Classification
3. Interpretability
4. Efficiency
5. Feature Engineering

Overall, PCA offers a versatile and effective approach to dimensionality reduction that is well-suited for both clustering and classification tasks. It provides a balance between preserving important information in the data and reducing its dimensionality, making it a valuable technique in various data analysis scenarios such as our given the nature of our dataset.

PART 2: Unsupervised Learning:

II) Use unsupervised learning models/clustering to investigate clusters/groups of genes and clusters/groups of patients. Apply Principal Component Analysis, k-means clustering and hierarchical clustering. You may add one further method.

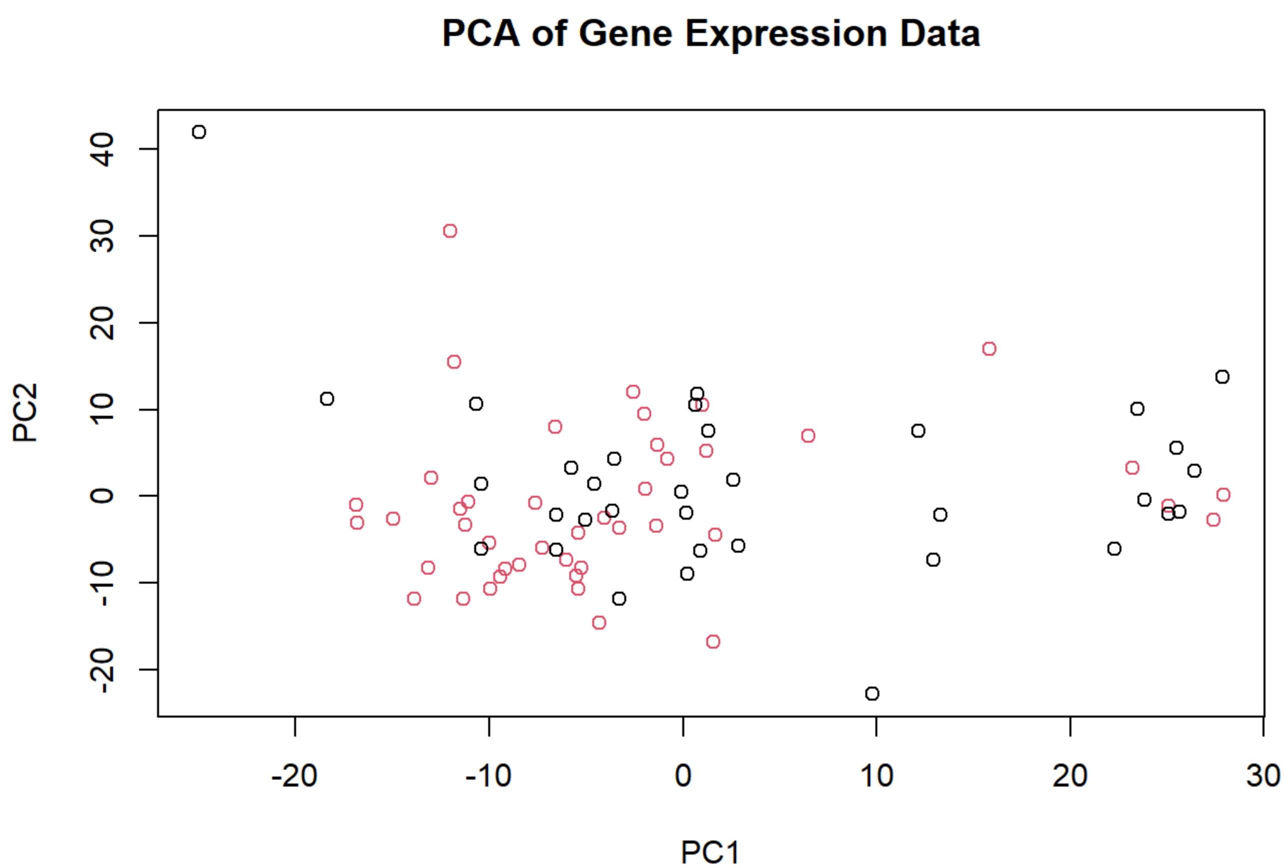
Principal Component Analysis - Creating a Model:

```
# Performing PCA on the cleaned and imputed dataset, excluding the last column
pcaResult <- prcomp(filtered_df_w_class[,-ncol(filtered_df_w_class)], center = TRUE, scale. = TR
UE)
summary(pcaResult)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     13.0223 9.8284 9.25056 8.76649 7.59301 6.76422 6.56015
## Proportion of Variance  0.1187 0.0676 0.05988 0.05378 0.04035 0.03202 0.03012
## Cumulative Proportion   0.1187 0.1863 0.24615 0.29993 0.34028 0.37230 0.40241
##                            PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     6.43962 5.88419 5.55992 5.34519 5.21624 5.05868 4.80252
## Proportion of Variance 0.02902 0.02423 0.02163 0.01999 0.01904 0.01791 0.01614
## Cumulative Proportion  0.43143 0.45566 0.47729 0.49729 0.51633 0.53424 0.55038
##                           PC15    PC16    PC17    PC18    PC19    PC20   PC21
## Standard deviation     4.74818 4.66365 4.62757 4.53056 4.44574 4.36092 4.3106
## Proportion of Variance 0.01578 0.01522 0.01499 0.01436 0.01383 0.01331 0.0130
## Cumulative Proportion  0.56615 0.58137 0.59636 0.61072 0.62455 0.63786 0.6509
##                           PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     4.21539 4.13073 4.07541 3.98923 3.96891 3.90878 3.84235
## Proportion of Variance 0.01243 0.01194 0.01162 0.01114 0.01102 0.01069 0.01033
## Cumulative Proportion  0.66330 0.67524 0.68686 0.69800 0.70902 0.71971 0.73005
##                           PC29    PC30    PC31    PC32    PC33    PC34    PC35
## Standard deviation     3.72961 3.70281 3.62993 3.61303 3.51768 3.47629 3.43040
## Proportion of Variance 0.00973 0.00959 0.00922 0.00914 0.00866 0.00846 0.00823
## Cumulative Proportion  0.73978 0.74937 0.75859 0.76773 0.77639 0.78485 0.79308
##                           PC36    PC37    PC38    PC39    PC40    PC41    PC42
## Standard deviation     3.37112 3.33194 3.31362 3.29383 3.22428 3.17435 3.15009
## Proportion of Variance 0.00795 0.00777 0.00768 0.00759 0.00727 0.00705 0.00694
## Cumulative Proportion  0.80103 0.80880 0.81649 0.82408 0.83135 0.83840 0.84535
##                           PC43    PC44    PC45    PC46    PC47    PC48    PC49
## Standard deviation     3.10326 3.05101 3.00554 2.98243 2.94585 2.92674 2.88405
## Proportion of Variance 0.00674 0.00651 0.00632 0.00622 0.00607 0.00599 0.00582
## Cumulative Proportion  0.85209 0.85860 0.86492 0.87115 0.87722 0.88322 0.88904
##                           PC50    PC51    PC52   PC53    PC54    PC55    PC56
## Standard deviation     2.85896 2.81952 2.78296 2.7791 2.76539 2.70510 2.67489
## Proportion of Variance 0.00572 0.00556 0.00542 0.0054 0.00535 0.00512 0.00501
## Cumulative Proportion  0.89476 0.90032 0.90574 0.9111 0.91649 0.92162 0.92662
##                           PC57   PC58    PC59    PC60    PC61    PC62    PC63
## Standard deviation     2.61270 2.5917 2.53919 2.49525 2.45853 2.42539 2.40288
## Proportion of Variance 0.00478 0.0047 0.00451 0.00436 0.00423 0.00412 0.00404
## Cumulative Proportion  0.93140 0.9361 0.94061 0.94497 0.94920 0.95332 0.95736
##                           PC64    PC65    PC66    PC67    PC68    PC69    PC70
## Standard deviation     2.35260 2.29672 2.28423 2.23989 2.19872 2.14353 2.09731
## Proportion of Variance 0.00387 0.00369 0.00365 0.00351 0.00338 0.00322 0.00308
## Cumulative Proportion  0.96123 0.96492 0.96857 0.97208 0.97547 0.97868 0.98176
##                           PC71    PC72   PC73    PC74    PC75    PC76    PC77
## Standard deviation     2.06713 2.02591 2.0012 1.94920 1.89281 1.80437 1.74541
## Proportion of Variance 0.00299 0.00287 0.0028 0.00266 0.00251 0.00228 0.00213
## Cumulative Proportion  0.98475 0.98762 0.9904 0.99308 0.99559 0.99787 1.00000
##                             PC78
## Standard deviation     5.783e-15
## Proportion of Variance 0.000e+00
## Cumulative Proportion  1.000e+00
```

Visualizing Points Across Top 2 Principal Components:

```
# Plot PCA, coloring by the class variable that shows clusters
plot(pcaResult$x[,1:2], col = as.factor(filtered_df_w_class[,ncol(filtered_df_w_class)]))
title("PCA of Gene Expression Data")
```



Features Before and After Performing PCA:

```
# Before PCA: Count the number of original variables (excluding the class variable)
num_original_variables <- ncol(filtered_df_w_class) - 1
print(paste("Number of original variables:", num_original_variables))
```

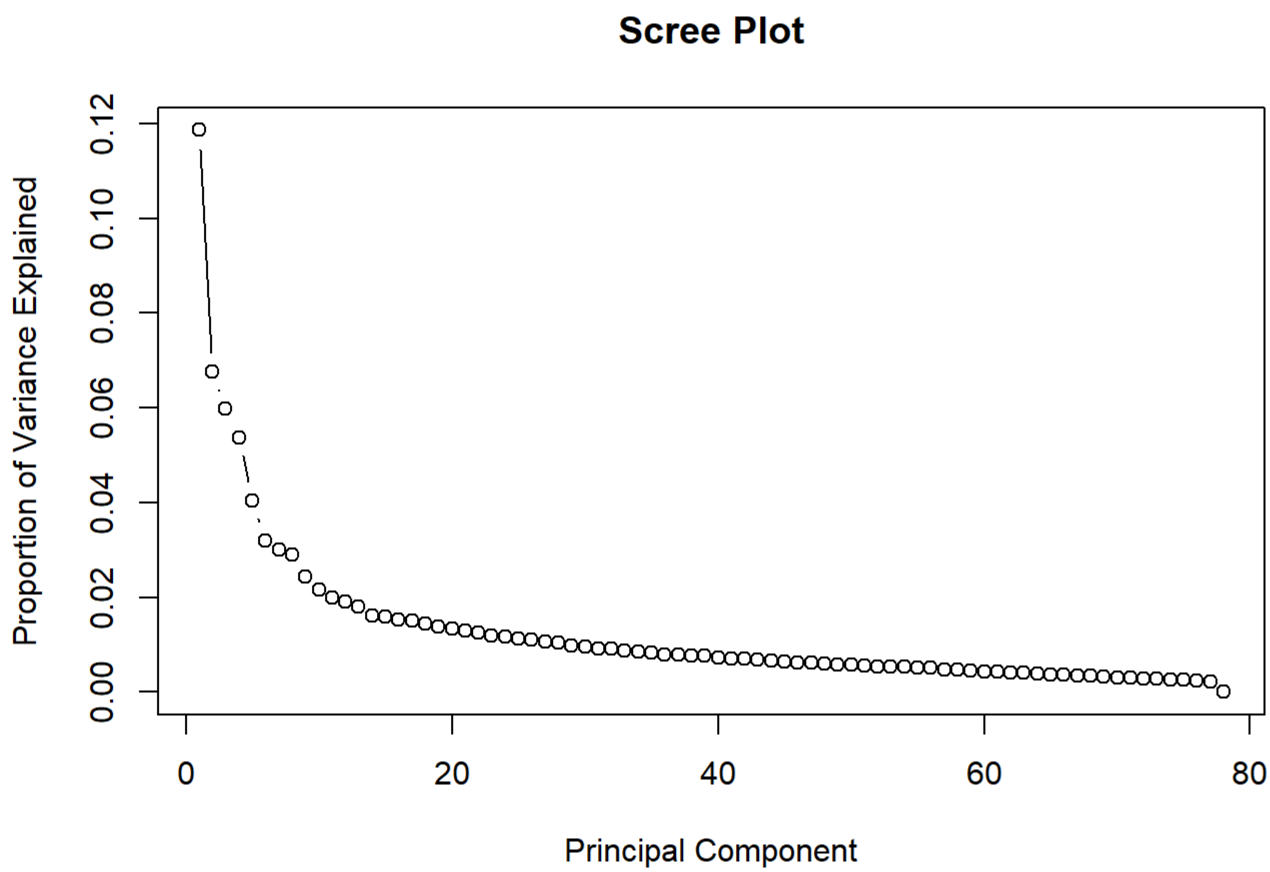
```
## [1] "Number of original variables: 1429"
```

```
# After PCA: Determine the number of principal components based on a variance threshold
explained_variance <- summary(pcaResult)$importance[2,]
cumulative_explained_variance <- cumsum(explained_variance)
variance_threshold <- 0.80 # for example, 80% of the variance
num_components_needed <- which(cumulative_explained_variance >= variance_threshold)[1]
print(paste("Number of components needed to explain at least", variance_threshold * 100, "% of t
he variance:", num_components_needed))
```

```
## [1] "Number of components needed to explain at least 80 % of the variance: 36"
```

Plotting Explained Variance by Each Component:

```
# Optionally, plot a scree plot to visually inspect the variance explained by each component
plot(explained_variance, type = "b", xlab = "Principal Component", ylab = "Proportion of Varianc
e Explained", main = "Scree Plot")
abline(h = variance_threshold, col = "red", lty = 2) # Add a horizontal line at the variance thr
eshold
```

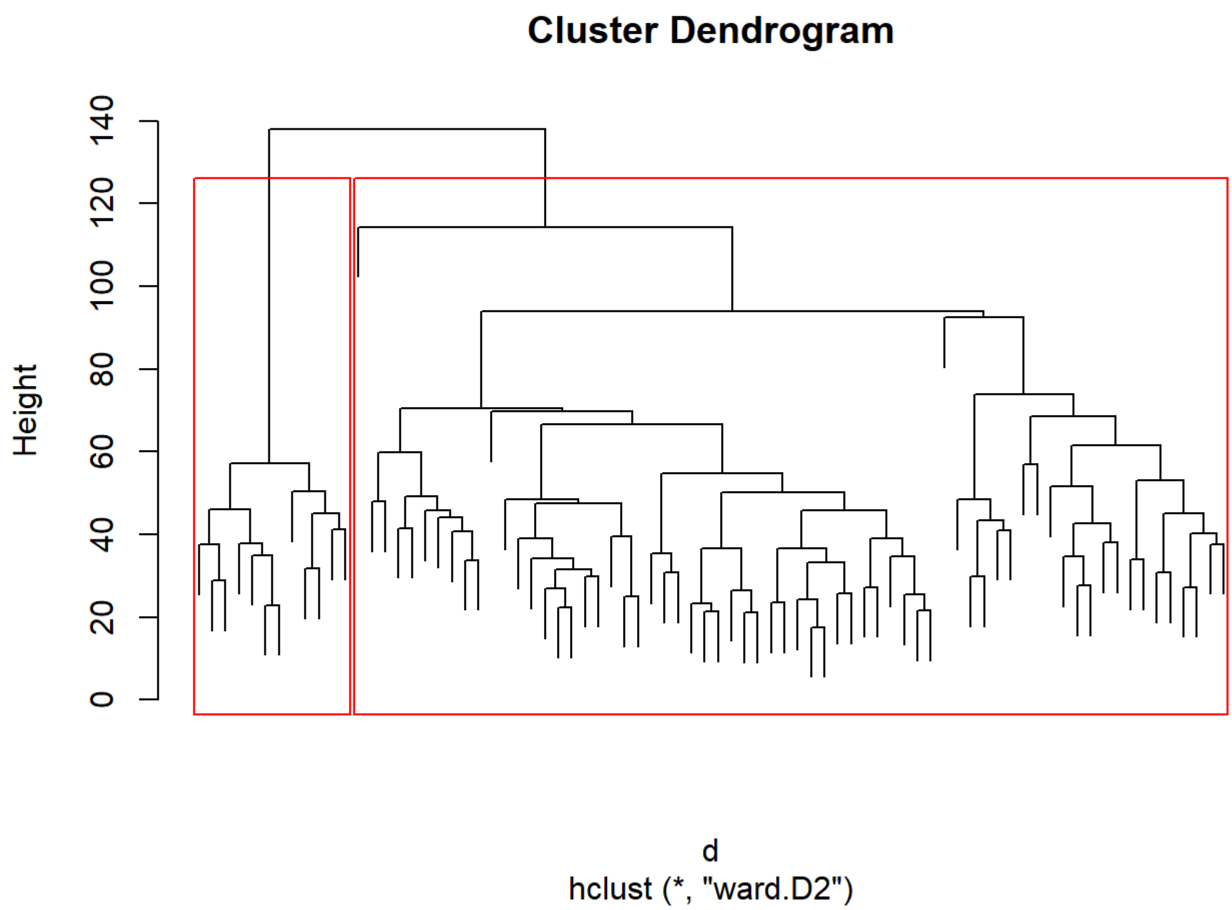


Agglomerative Clustering - Creating the Model:

```
# Exclude the class/label column from the clustering input
DataForClustering <- pca_df_w_class[, !names(pca_df_w_class) %in% c('Class')]

# Calculate distance matrix using Euclidean distance
d <- dist(DataForClustering, method = "euclidean")

# Perform agglomerative clustering using Ward's method
hc <- hclust(d, method = "ward.D2")

# Increasing plot margins to ensure clarity
par(mar=c(5,4,4,8) + 0.1)  # Adjust the margins (bottom, left, top, right)
```

Plotting the Clusters:

```
# Plot the dendrogram with enhanced clarity
plot(hc, labels=FALSE, cex=0.6)  # Adjust cex for label size if labels are used

# Choose the number of clusters k based on your analysis or requirement
k <- 2
rect.hclust(hc, k=k, border="red")  # Add colored rectangles around clusters
```



Exporting Dendogram as PDF:

```
# Exporting the dendrogram to a PDF for high-quality output
pdf("dendrogram.pdf", width=10, height=8)
plot(hc, labels=FALSE)
rect.hclust(hc, k=k, border="red")  # Optionally, add colored rectangles again for the PDF outpu
t
dev.off()  # Close the PDF device
```

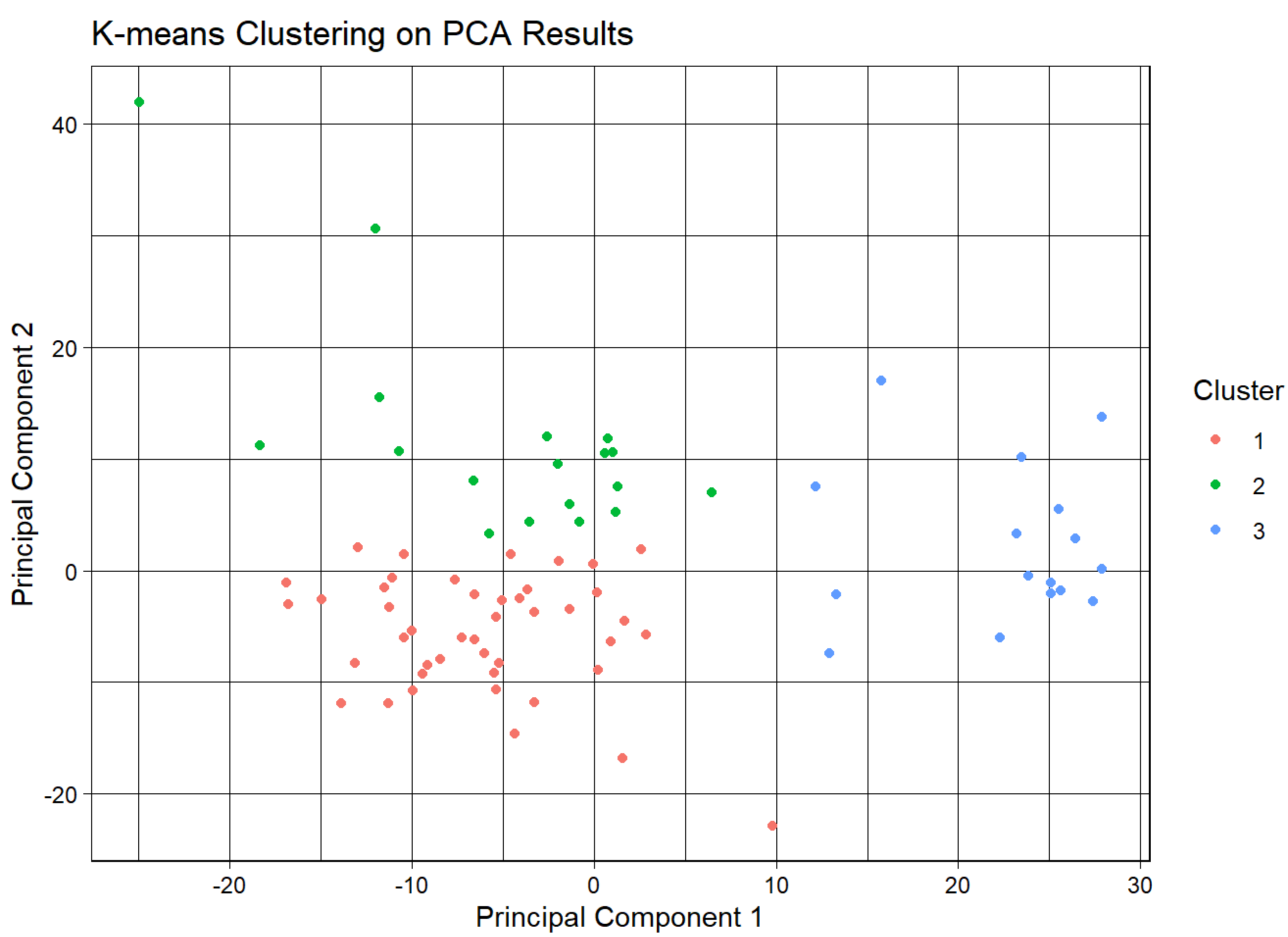
```
## png
##   2
```

K-Means Clustering - Creating the Model:

```
# Specify the number of clusters
k <- 3

# Perform k-means clustering
kmeans_result <- kmeans(pcaResult$x[, 1:2], centers = k)
```

```
# Convert the first 2 principal components to a data frame
kmeans_df_pca <- as.data.frame(pcaResult$x[, 1:2])

# Add k-means as a new factor column in the PCA data frame
kmeans_df_pca$cluster <- as.factor(kmeans_result$cluster)
```

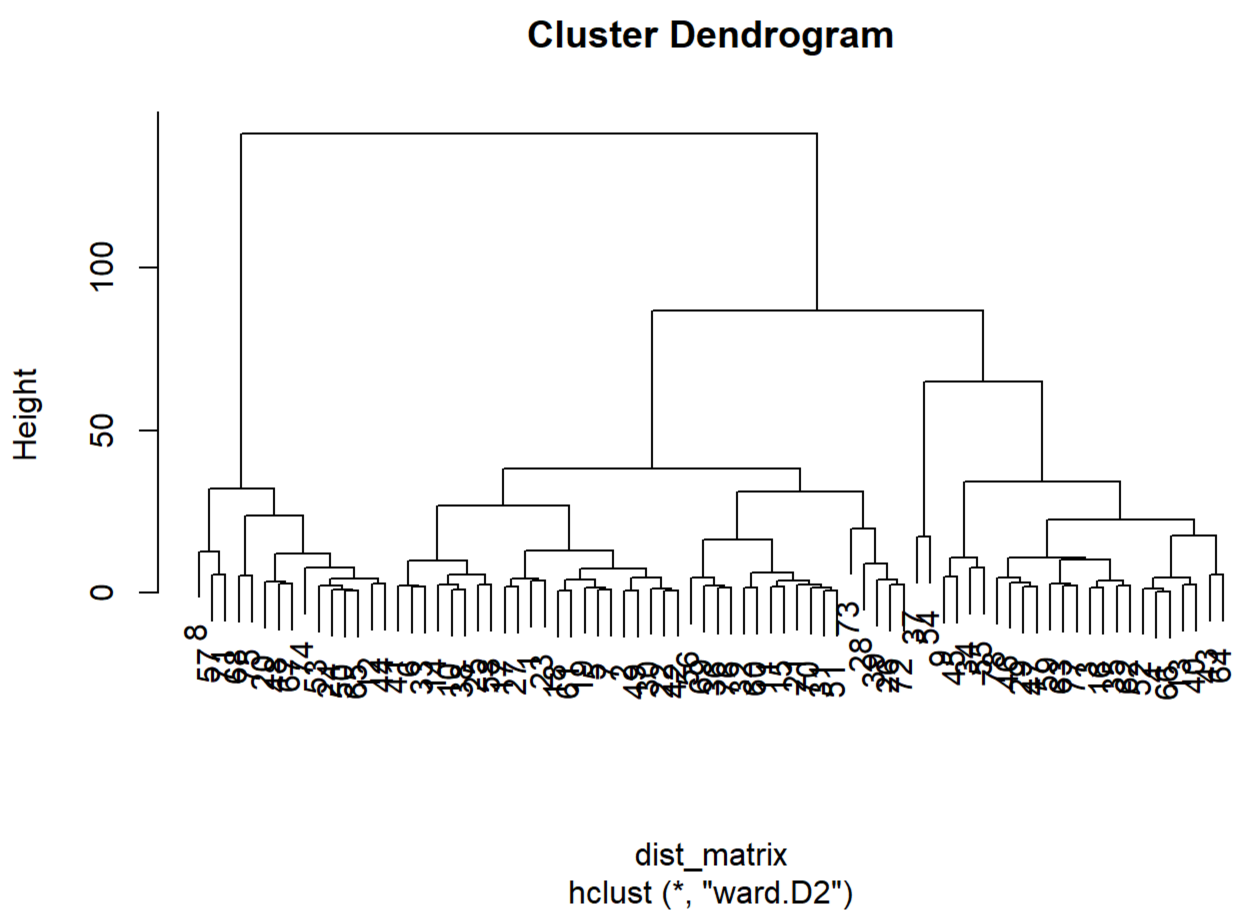
```
# Plot using ggplot2
kmeans_plot <- ggplot(kmeans_df_pca, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point(alpha = 1) +  theme_linedraw() + labs(title = "K-means Clustering on PCA Results",
      x = "Principal Component 1",
       y = "Principal Component 2") +
  scale_color_discrete(name = "Cluster")

kmeans_plot
```




```
# Save plot
ggsave("k-means.png", plot = kmeans_plot, width = 10, height = 8, units = "in")
```

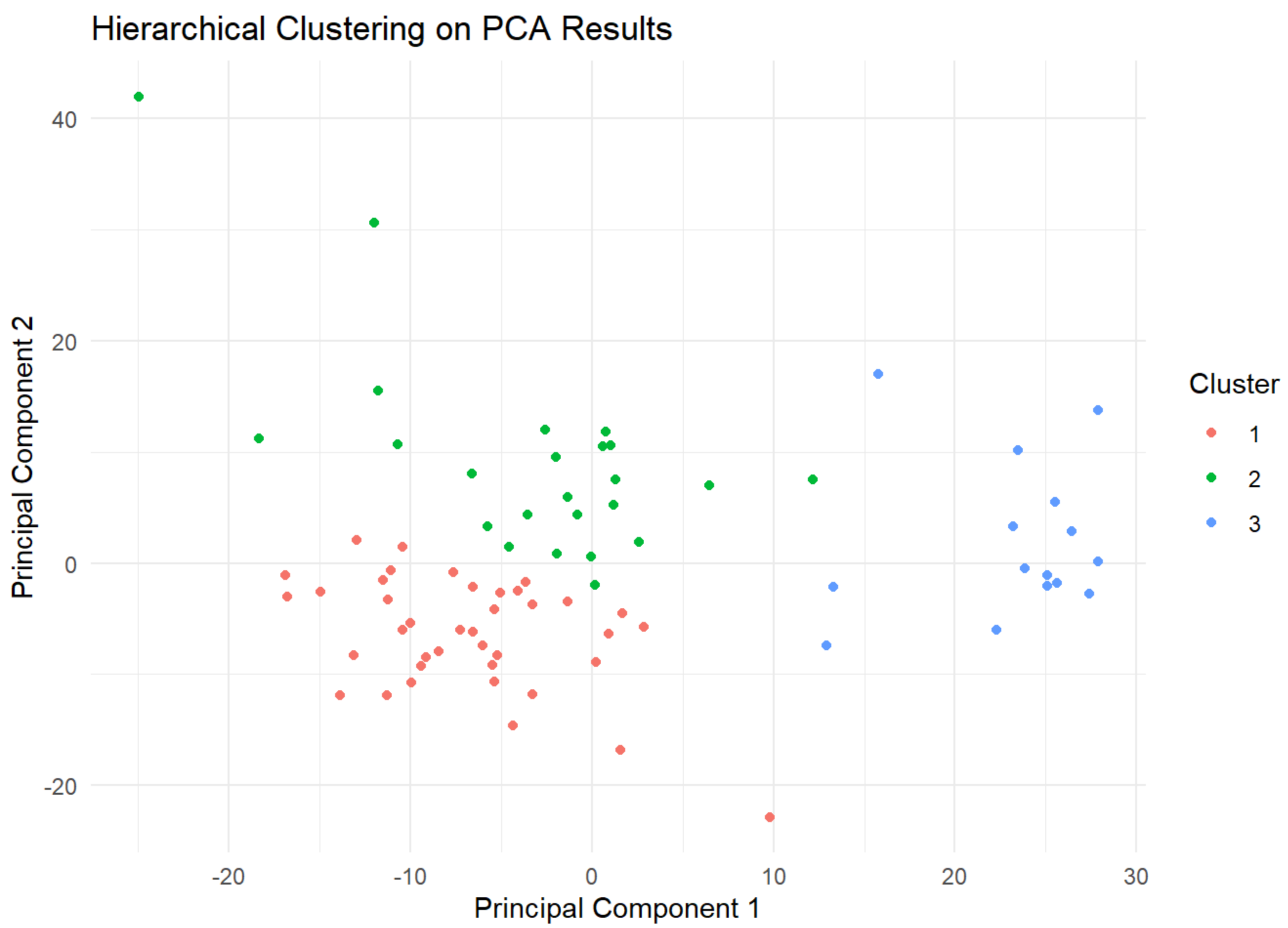
Hierarchical Clustering - Creating the Model:

```
# Create another data frame with first 2 principal components
df_hclust <- data.frame(PC1 = pcaResult$x[, 1], PC2 = pcaResult$x[, 2])

# Compute the distance matrix
dist_matrix <- dist(df_hclust, method = "euclidean")

# Perform hierarchical clustering
hc_result <- hclust(dist_matrix, method = "ward.D2")

# Plot the dendrogram
plot(hc_result)
```




```
# Specify the number of clusters
k <- 3

# Cut the dendrogram tree into 'k' clusters
clusters <- cutree(hc_result, k = k)

# Add the cluster assignments as a new factor column
df_hclust$cluster <- as.factor(clusters)
```

```
# Plot using ggplot2
ggplot(df_hclust, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point() +
  theme_minimal() +
  labs(title = "Hierarchical Clustering on PCA Results",
       x = "Principal Component 1",
       y = "Principal Component 2") +
  scale_color_discrete(name = "Cluster")
```

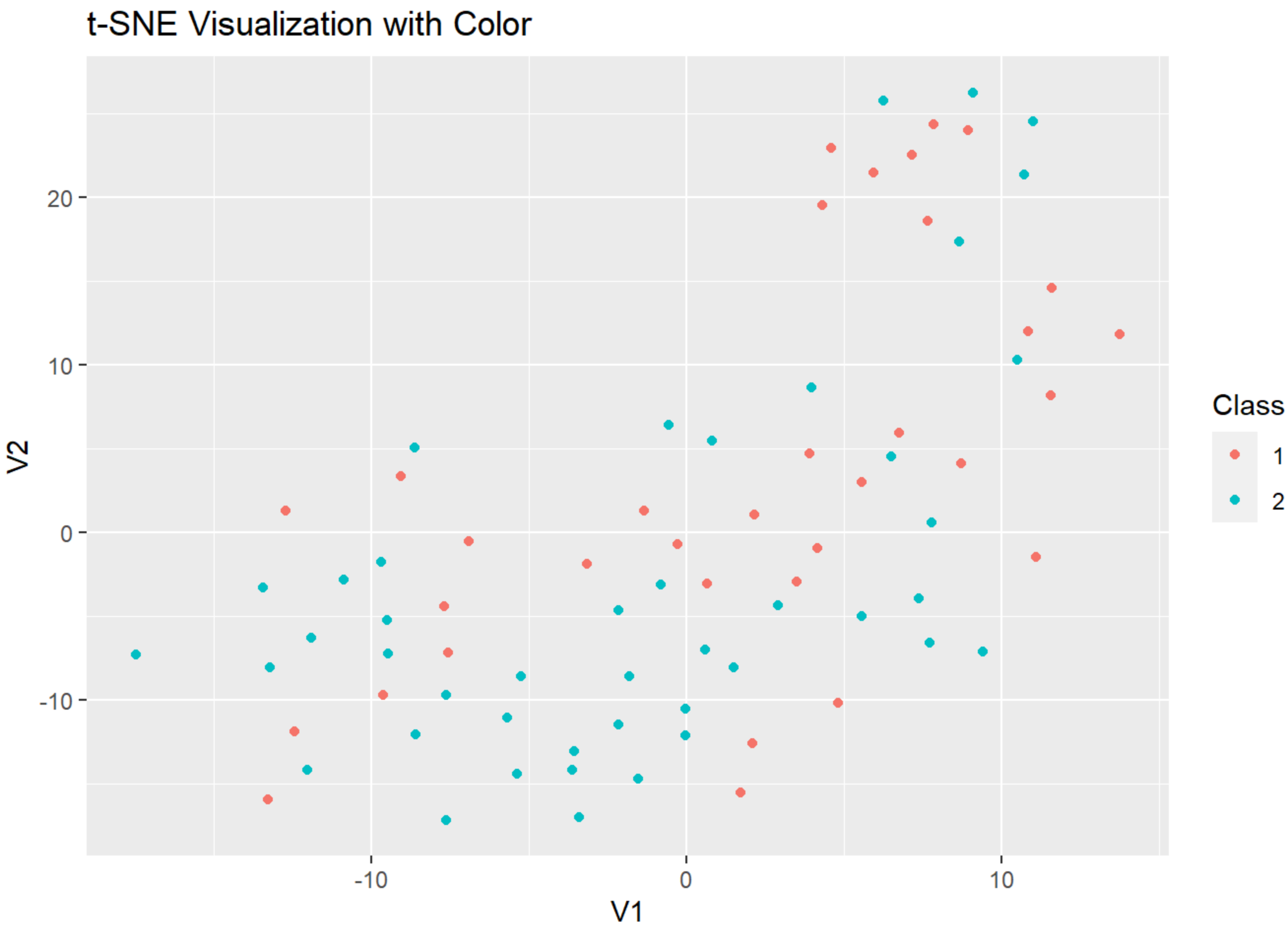


T-SNE Clustering - Creating the Model:

```
tsne_result <- Rtsne(pca_df_w_class, dims = 2, perplexity = 10, verbose = TRUE)
```

```
## Performing PCA
## Read the 78 x 37 data matrix successfully!
## OpenMP is working. 1 threads.
## Using no_dims = 2, perplexity = 10.000000, and theta = 0.500000
## Computing input similarities...
## Building tree...
## Done in 0.00 seconds (sparsity = 0.550625)!
## Learning embedding...
## Iteration 50: error is 68.400485 (50 iterations in 0.01 seconds)
## Iteration 100: error is 77.085138 (50 iterations in 0.01 seconds)
## Iteration 150: error is 74.987794 (50 iterations in 0.01 seconds)
## Iteration 200: error is 71.066682 (50 iterations in 0.01 seconds)
## Iteration 250: error is 71.874469 (50 iterations in 0.01 seconds)
## Iteration 300: error is 2.776082 (50 iterations in 0.00 seconds)
## Iteration 350: error is 2.112234 (50 iterations in 0.00 seconds)
## Iteration 400: error is 1.761509 (50 iterations in 0.00 seconds)
## Iteration 450: error is 1.578410 (50 iterations in 0.00 seconds)
## Iteration 500: error is 1.440502 (50 iterations in 0.00 seconds)
## Iteration 550: error is 1.264310 (50 iterations in 0.00 seconds)
## Iteration 600: error is 1.145408 (50 iterations in 0.00 seconds)
## Iteration 650: error is 1.051270 (50 iterations in 0.00 seconds)
## Iteration 700: error is 0.969101 (50 iterations in 0.00 seconds)
## Iteration 750: error is 0.916154 (50 iterations in 0.00 seconds)
## Iteration 800: error is 0.889137 (50 iterations in 0.00 seconds)
## Iteration 850: error is 0.841490 (50 iterations in 0.00 seconds)
## Iteration 900: error is 0.801222 (50 iterations in 0.00 seconds)
## Iteration 950: error is 0.780065 (50 iterations in 0.00 seconds)
## Iteration 1000: error is 0.748513 (50 iterations in 0.00 seconds)
## Fitting performed in 0.09 seconds.
```

```
tsne_df <- as.data.frame(tsne_result$Y)

ggplot(tsne_df, aes(x = V1, y = V2, color = as.factor(pca_df_w_class$Class))) +
  geom_point() +
  labs(title = "t-SNE Visualization with Color") +
  scale_color_discrete(name = "Class")
```



PART 3: Supervised Learning:

III) Use supervised learning models/classification to predict the class (invasive or non invasive) of future patients. Apply Logistic Regression, LDA, QDA, k-NN, Random Forest and SVM. Discuss why you choose specific hyper parameters of a supervised learning model. You may add one or two further methods to the investigation. Use resampling techniques to compare the machine learning models applied. Suggest and justify your 'best' machine learning model.

Preparing Dataset:

```
# Loading dataset
df <- pca_df_w_class

# Converting class label to factor with two levels
df$Class <- factor(df$Class)

str(df)
```

```
## 'data.frame':    78 obs. of  37 variables:
##  $ PC1  : num  -13.14 -9.99 -1.32 1.01 -5.24 ...
##  $ PC2  : num  -8.28 -5.43 5.92 10.56 -8.32 ...
##  $ PC3  : num  2.105 -1.513 1.64 0.462 -3.095 ...
##  $ PC4  : num  -2.63 4.61 -2.29 7.23 2.05 ...
##  $ PC5  : num  1.21 3.74 -14.16 2.21 8.71 ...
##  $ PC6  : num  -0.655 -5.21 -3.009 6.988 9.965 ...
##  $ PC7  : num  9.59 7.18 5.64 13.62 -11.09 ...
##  $ PC8  : num  -3.25 -5.09 -6.56 -0.65 2.97 ...
##  $ PC9  : num  -1.045 0.375 -7.463 -11.066 -2.377 ...
##  $ PC10 : num  0.676 4.419 -2.173 -7.456 3.187 ...
##  $ PC11 : num  -2.76 3.38 3.69 -12.05 -4.37 ...
##  $ PC12 : num  4.227 -3.51 -6.277 0.171 -0.255 ...
##  $ PC13 : num  -1.05 3.14 -3.66 2.77 1.14 ...
##  $ PC14 : num  0.233 -1.721 -0.508 0.446 -0.989 ...
##  $ PC15 : num  -0.8557 -0.0824 2.7536 11.4472 -3.9222 ...
##  $ PC16 : num  0.686 1.267 -1.025 -1.736 3.561 ...
##  $ PC17 : num  -3.236 6.898 1.396 1.715 0.467 ...
##  $ PC18 : num  0.4816 -0.0373 -2.6605 -0.8236 3.1296 ...
##  $ PC19 : num  -6.695 0.849 -2.249 2.093 1.44 ...
##  $ PC20 : num  -1.8562 -1.4178 -3.6952 -12.0431 0.0755 ...
##  $ PC21 : num  6.63 0.48 -1.5 1.7 -4.09 ...
##  $ PC22 : num  -2.59 -1.623 6.278 -0.473 3.436 ...
##  $ PC23 : num  -1.149 -0.41 -0.927 -10.003 -2.601 ...
##  $ PC24 : num  7.5 -1.11 5.48 -8.14 2.72 ...
##  $ PC25 : num  -1.41 1.977 -6.814 -3.255 -0.694 ...
##  $ PC26 : num  -1.435 -0.197 0.836 7.202 -0.321 ...
##  $ PC27 : num  -0.916 2.061 1.978 -0.4 -3.583 ...
##  $ PC28 : num  1.4852 -0.0887 -4.3069 4.0448 1.471 ...
##  $ PC29 : num  1.567 -1.927 -4.36 1.017 0.856 ...
##  $ PC30 : num  3.481 -9.818 -1.936 -0.435 -0.826 ...
##  $ PC31 : num  -5.09 1.21 -4.32 2.13 3.95 ...
##  $ PC32 : num  -2.104 0.728 3.676 2.088 -0.351 ...
##  $ PC33 : num  5.219 7.775 -1.177 -5.216 -0.841 ...
##  $ PC34 : num  7.84 3.98 -8.2 0.92 0.54 ...
##  $ PC35 : num  1.149 1.323 1.735 0.812 2.284 ...
##  $ PC36 : num  -9.206 -0.675 -5.135 3.552 -3.641 ...
##  $ Class: Factor w/ 2 levels "1","2": 2 2 2 2 2 2 2 2 2 2 ...
```

```
# Checking class distribution
class_distribution <- table(df$Class)
class_distribution
```

```
##
##  1  2
## 34 44
```

```
# Setting Seed for reproducibility
registration_number <- 2315740 #Mantosh
set.seed(registration_number)

# Splitting into train & test sets
train_index <- sample(1:nrow(df), 0.80 * nrow(df))
train_data <- df[train_index, ]
test_data <- df[-train_index, ]

class_distribution_train <- table(train_data$Class)
class_distribution_train
```

```
##
##  1  2
## 31 31
```

```
class_distribution_test <- table(test_data$Class)
class_distribution_test
```

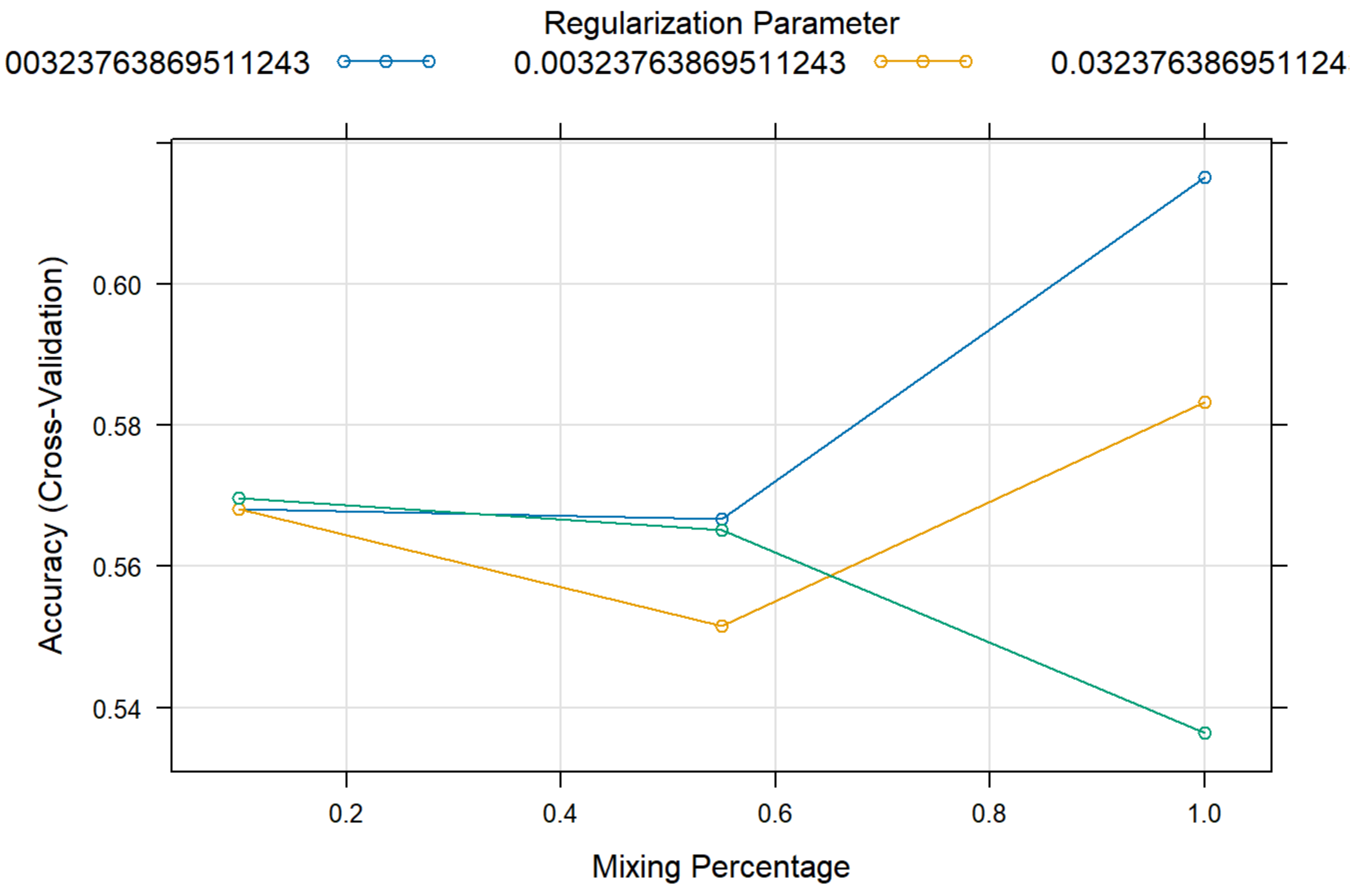
```
##
##  1  2
##  3 13
```

Creating and Tuning Logistic Regression Model:

```
# Define control parameters for model
lr_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross-validation

# Creating Logistic Regression model
lr <- train(Class ~ ., data = train_data, method = "glmnet", trControl = lr_ctrl)
lr_best_model <- lr

# Print cross-validation results
print(lr_best_model$results)
```

```
##   alpha       lambda  Accuracy      Kappa AccuracySD   KappaSD
## 1  0.10 0.0003237639 0.5681818 0.13636364 0.12373325 0.2474665
## 2  0.10 0.0032376387 0.5681818 0.13636364 0.12373325 0.2474665
## 3  0.10 0.0323763870 0.5696970 0.13939394 0.14780301 0.2956060
## 4  0.55 0.0003237639 0.5666667 0.13333333 0.07637626 0.1527525
## 5  0.55 0.0032376387 0.5515152 0.10303030 0.09773608 0.1954722
## 6  0.55 0.0323763870 0.5651515 0.13030303 0.07691539 0.1538308
## 7  1.00 0.0003237639 0.6151515 0.23030303 0.07837889 0.1567578
## 8  1.00 0.0032376387 0.5833333 0.16666667 0.10408330 0.2081666
## 9  1.00 0.0323763870 0.5363636 0.07272727 0.12103206 0.2420641
```

```
# Plotting accuracy vs. hyperparameters
plot(lr_best_model)
```




```
# Selecting the best model (not required for logistic regression)
# Logistic regression does not require selecting the best model

# Making predictions of Logistic Regression model
lr_predictions <- predict(lr_best_model, newdata = test_data)

# Evaluating the Logistic Regression model
lr_confusion_matrix <- table(Actual = test_data$Class, Predicted = lr_predictions)
lr_confusion_matrix
```

```
##        Predicted
## Actual 1 2
##      1 2 1
##      2 5 8
```

```
# Accuracy calculation of Logistic Regression model
lr_accuracy <- sum(diag(lr_confusion_matrix)) / sum(lr_confusion_matrix)
lr_accuracy
```

```
## [1] 0.625
```

Creating and Tuning Linear Discriminant (LDA) Model:

```r
# Defining control parameters for model
lda_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training LDA model
lda <- train(Class ~ ., data = train_data, method = "lda" ,
             trControl = lda_ctrl)
lda_best_model <- lda


# Printing cross-validation results
print(lda$results)
```

```
##   parameter  Accuracy     Kappa AccuracySD  KappaSD
## 1      none 0.5793651 0.1644245  0.0836163 0.170718
```

```r
#Making predictions of LDA model
lda_predictions <- predict(lda_best_model, newdata = test_data)

# Evaluating the model
lda_confusion_matrix <- table(Actual = test_data$Class, Predicted = lda_predictions)
lda_confusion_matrix
```

```
##       Predicted
## Actual 1 2
##      1 2 1
##      2 4 9
```

```r
# Accuracy calculation
lda_accuracy <- sum(diag(lda_confusion_matrix)) / sum(lda_confusion_matrix)
lda_accuracy
```

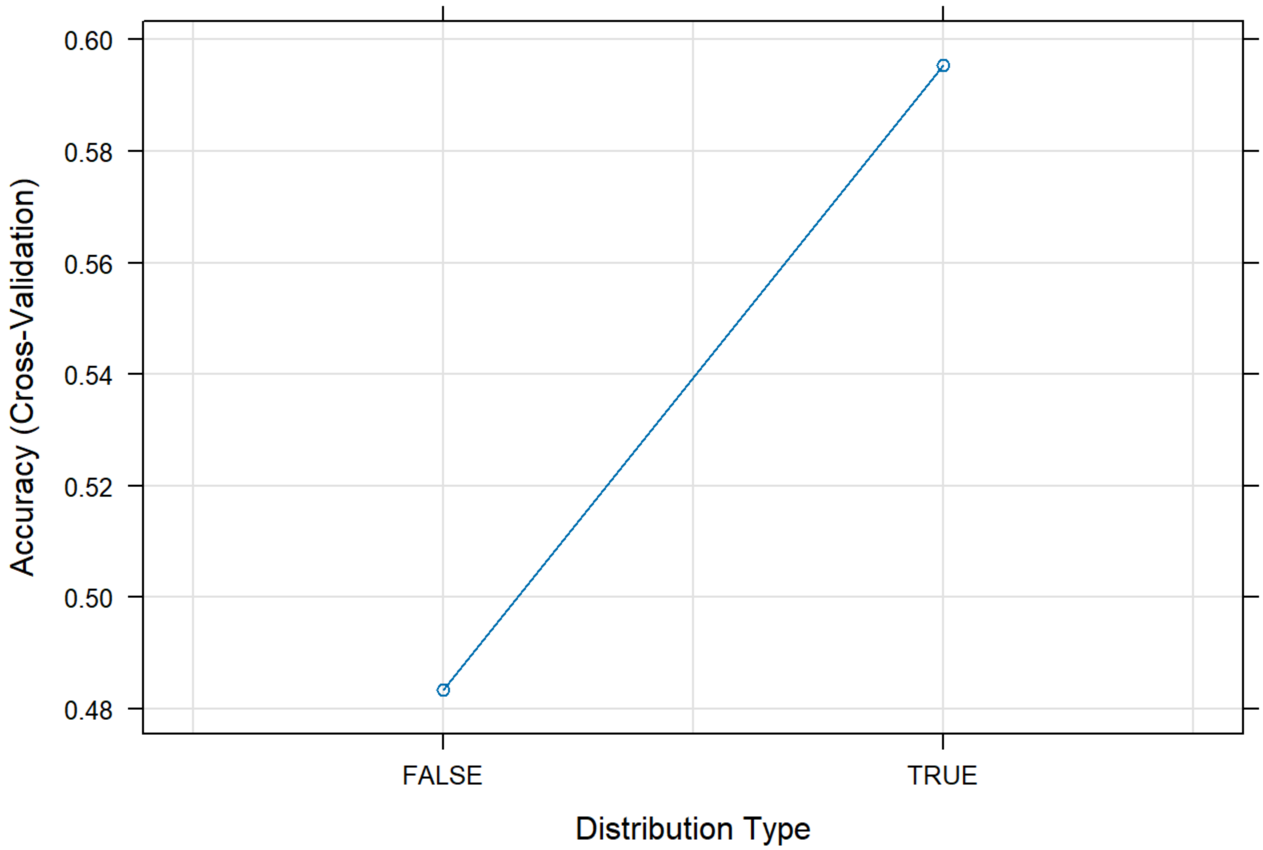
```
## [1] 0.6875
```

CREATING AND TUNING NAIVE BAYES MODEL FOR BEST RESULT:

```r
# Defining control parameters for model
nb_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training Naive Bayes model
nb <- train(Class ~ ., data = train_data, method = "naive_bayes",
            trControl = nb_ctrl)
nb_best_model <- nb

# Printing cross-validation results
print(nb_best_model$results)
```

```
##   usekernel laplace adjust  Accuracy       Kappa AccuracySD   KappaSD
## 1     FALSE       0      1 0.4833333 -0.03333333 0.07637626 0.1527525
## 2      TRUE       0      1 0.5954545  0.19090909 0.08294676 0.1658935
```

```
# Plotting accuracy
plot(nb_best_model)
```




```
# Making predictions of Naive Bayes model
nb_predictions <- predict(nb_best_model, newdata = test_data)

# Evaluating the model
nb_confusion_matrix <- table(Actual = test_data$Class, Predicted = nb_predictions)
nb_confusion_matrix
```

```
##         Predicted
## Actual 1 2
##      1 1 2
##      2 4 9
```

```
# Accuracy calculation
nb_accuracy <- sum(diag(nb_confusion_matrix)) / sum(nb_confusion_matrix)
nb_accuracy
```

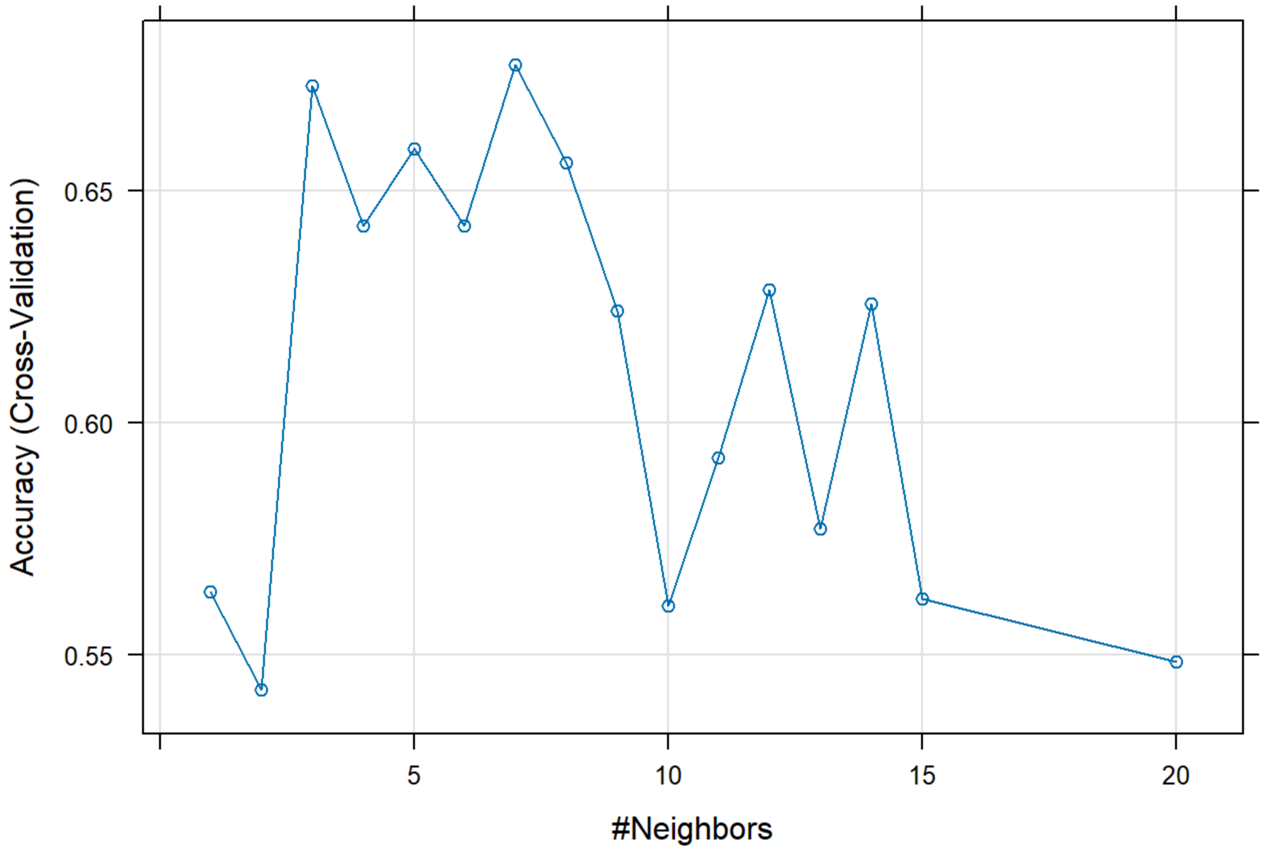
```
## [1] 0.625
```

CREATING AND TUNING K NEAREST NEIGHBORS MODEL FOR BEST RESULT:

```
# Defining control parameters for model
knn_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training KNN model with grid search for hyper-parameter tuning
k_values <- c(1, 2, 3, 4, 5 , 6, 7, 8, 9 , 10, 11, 12, 13 , 14 , 15, 20)
knn <- train(Class ~ ., data = train_data, method = "knn",
             trControl = knn_ctrl,
             tuneGrid = expand.grid(k = k_values))

# Printing cross-validation results
print(knn$results)
```

```
##     k  Accuracy      Kappa AccuracySD    KappaSD
## 1   1 0.5636364 0.12727273 0.05529784 0.11059568
## 2   2 0.5424242 0.08484848 0.16771023 0.33542046
## 3   3 0.6727273 0.34545455 0.13552774 0.27105548
## 4   4 0.6424242 0.28484848 0.07348094 0.14696189
## 5   5 0.6590909 0.31818182 0.06412153 0.12824305
## 6   6 0.6424242 0.28484848 0.08887884 0.17775769
## 7   7 0.6772727 0.35454545 0.02530802 0.05061604
## 8   8 0.6560606 0.31212121 0.15917747 0.31835493
## 9   9 0.6242424 0.24848485 0.12859165 0.25718330
## 10 10 0.5606061 0.12121212 0.11627245 0.23254491
## 11 11 0.5924242 0.18484848 0.11942816 0.23885632
## 12 12 0.6287879 0.25757576 0.02584655 0.05169310
## 13 13 0.5772727 0.15454545 0.09392717 0.18785435
## 14 14 0.6257576 0.25151515 0.11555948 0.23111897
## 15 15 0.5621212 0.12424242 0.06898517 0.13797033
## 16 20 0.5484848 0.09696970 0.05006882 0.10013765
```

```
# Plotting accuracy vs. k
plot(knn)
```




```r
# Selecting the best model
best_k <- knn$bestTune$k
knn_best_model <- train(Class ~ ., data = train_data, method = "knn",
                  trControl = knn_ctrl,
                  tuneGrid = data.frame(k = best_k))
knn_best_model
```

```
## k-Nearest Neighbors
##
## 62 samples
## 36 predictors
##  2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 41, 41, 42
## Resampling results:
##
##   Accuracy   Kappa
##   0.5325397  0.07312172
##
## Tuning parameter 'k' was held constant at a value of 7
```

```r
#Making predictions of knn Model
knn_predictions <- predict(knn_best_model, newdata = test_data)

# Evaluating the model
knn_confusion_matrix <- table(Actual = test_data$Class, Predicted = knn_predictions)
knn_confusion_matrix
```

```
##       Predicted
## Actual  1  2
##      1  3  0
##      2  2 11
```

```r
# Accuracy calculation
knn_accuracy <- sum(diag(knn_confusion_matrix)) / sum(knn_confusion_matrix)
knn_accuracy
```

```
## [1] 0.875
```

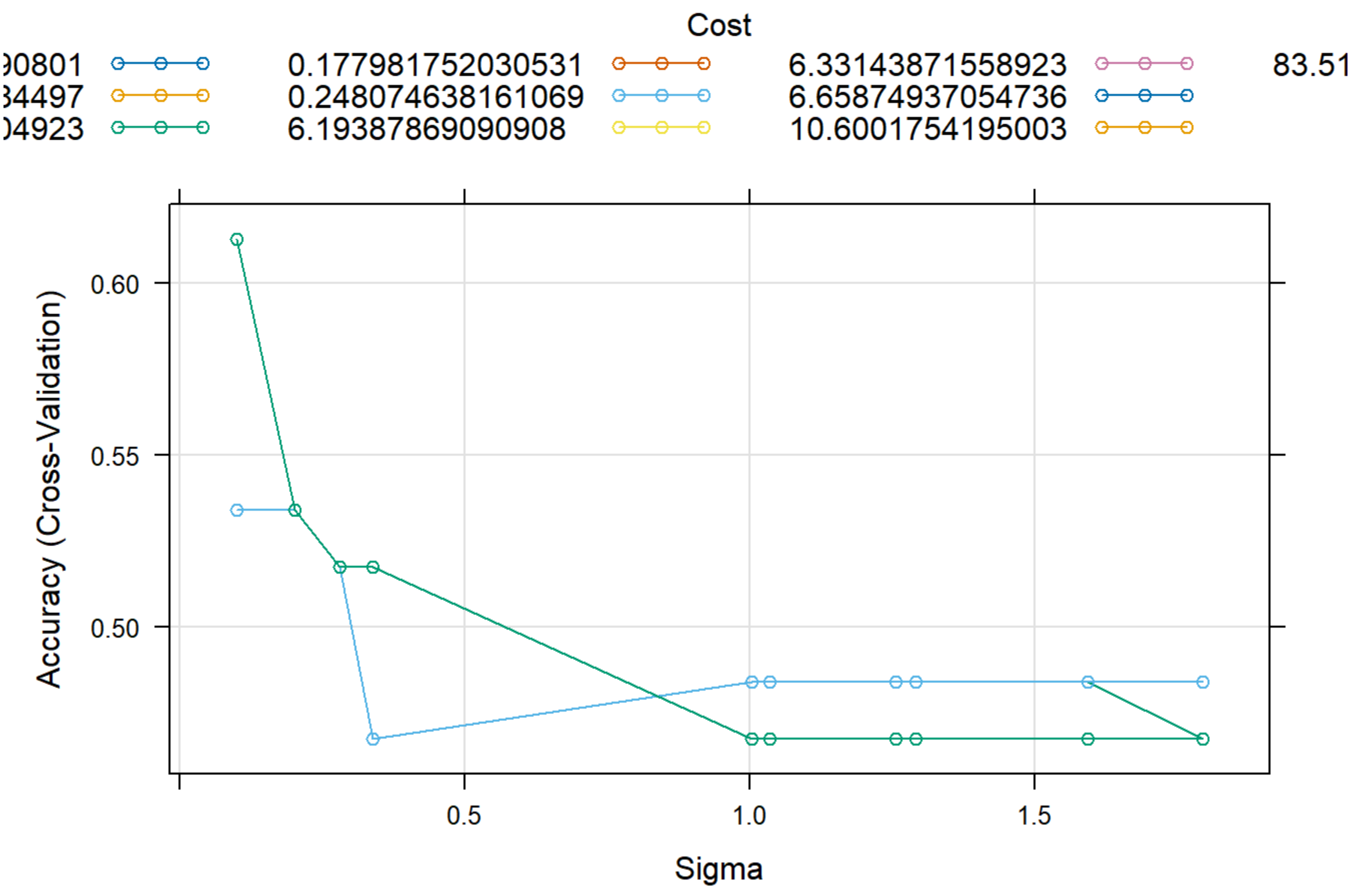
CREATING AND TUNING SVM MODEL FOR BEST RESULT:

```r
# Defining control parameters for model
svm_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training SVM model
svm_grid <- expand.grid(sigma = runif(10, 0.1, 2), C = 10^runif(10, -2, 2))
svm <- train(Class ~ .,data = train_data, method = "svmRadial",
                  trControl = svm_ctrl,
                  tuneGrid = svm_grid)

# Printing cross-validation results
print(svm$results)
```

```
##        sigma           C  Accuracy       Kappa AccuracySD    KappaSD
## 1   0.1007622  0.02505485 0.5341270  0.10000000 0.10034898 0.17320508
## 2   0.1007622  0.02703130 0.5341270  0.10000000 0.10034898 0.17320508
## 3   0.1007622  0.04728802 0.5341270  0.10000000 0.10034898 0.17320508
## 4   0.1007622  0.17798175 0.5341270  0.10000000 0.10034898 0.17320508
## 5   0.1007622  0.24807464 0.5341270  0.10000000 0.10034898 0.17320508
## 6   0.1007622  6.19387869 0.6126984  0.23554572 0.01099715 0.03096186
## 7   0.1007622  6.33143872 0.6126984  0.23554572 0.01099715 0.03096186
## 8   0.1007622  6.65874937 0.6126984  0.23554572 0.01099715 0.03096186
## 9   0.1007622 10.60017542 0.6126984  0.23554572 0.01099715 0.03096186
## 10  0.1007622 83.51410509 0.6126984  0.23554572 0.01099715 0.03096186
## 11  0.2009877  0.02505485 0.5341270  0.10000000 0.10034898 0.17320508
## 12  0.2009877  0.02703130 0.5341270  0.10000000 0.10034898 0.17320508
## 13  0.2009877  0.04728802 0.5341270  0.10000000 0.10034898 0.17320508
## 14  0.2009877  0.17798175 0.5341270  0.10000000 0.10034898 0.17320508
## 15  0.2009877  0.24807464 0.5341270  0.10000000 0.10034898 0.17320508
## 16  0.2009877  6.19387869 0.5341270  0.10000000 0.10034898 0.17320508
## 17  0.2009877  6.33143872 0.5341270  0.10000000 0.10034898 0.17320508
## 18  0.2009877  6.65874937 0.5341270  0.10000000 0.10034898 0.17320508
## 19  0.2009877 10.60017542 0.5341270  0.10000000 0.10034898 0.17320508
## 20  0.2009877 83.51410509 0.5341270  0.10000000 0.10034898 0.17320508
## 21  0.2814957  0.02505485 0.5174603  0.06666667 0.07148146 0.11547005
## 22  0.2814957  0.02703130 0.5174603  0.06666667 0.07148146 0.11547005
## 23  0.2814957  0.04728802 0.5174603  0.06666667 0.07148146 0.11547005
## 24  0.2814957  0.17798175 0.5174603  0.06666667 0.07148146 0.11547005
## 25  0.2814957  0.24807464 0.5174603  0.06666667 0.07148146 0.11547005
## 26  0.2814957  6.19387869 0.5174603  0.06666667 0.07148146 0.11547005
## 27  0.2814957  6.33143872 0.5174603  0.06666667 0.07148146 0.11547005
## 28  0.2814957  6.65874937 0.5174603  0.06666667 0.07148146 0.11547005
## 29  0.2814957 10.60017542 0.5174603  0.06666667 0.07148146 0.11547005
## 30  0.2814957 83.51410509 0.5174603  0.06666667 0.07148146 0.11547005
## 31  0.3385042  0.02505485 0.4674603 -0.03333333 0.01512108 0.05773503
## 32  0.3385042  0.02703130 0.4674603 -0.03333333 0.01512108 0.05773503
## 33  0.3385042  0.04728802 0.4674603 -0.03333333 0.01512108 0.05773503
## 34  0.3385042  0.17798175 0.4674603 -0.03333333 0.01512108 0.05773503
## 35  0.3385042  0.24807464 0.4674603 -0.03333333 0.01512108 0.05773503
## 36  0.3385042  6.19387869 0.5174603  0.06666667 0.07148146 0.11547005
## 37  0.3385042  6.33143872 0.5174603  0.06666667 0.07148146 0.11547005
## 38  0.3385042  6.65874937 0.5174603  0.06666667 0.07148146 0.11547005
## 39  0.3385042 10.60017542 0.5174603  0.06666667 0.07148146 0.11547005
## 40  0.3385042 83.51410509 0.5174603  0.06666667 0.07148146 0.11547005
## 41  1.0027563  0.02505485 0.4841270  0.00000000 0.01374643 0.00000000
## 42  1.0027563  0.02703130 0.4841270  0.00000000 0.01374643 0.00000000
## 43  1.0027563  0.04728802 0.4841270  0.00000000 0.01374643 0.00000000
## 44  1.0027563  0.17798175 0.4841270  0.00000000 0.01374643 0.00000000
## 45  1.0027563  0.24807464 0.4841270  0.00000000 0.01374643 0.00000000
## 46  1.0027563  6.19387869 0.4674603 -0.03333333 0.01512108 0.05773503
## 47  1.0027563  6.33143872 0.4674603 -0.03333333 0.01512108 0.05773503
## 48  1.0027563  6.65874937 0.4674603 -0.03333333 0.01512108 0.05773503
## 49  1.0027563 10.60017542 0.4674603 -0.03333333 0.01512108 0.05773503
## 50  1.0027563 83.51410509 0.4674603 -0.03333333 0.01512108 0.05773503
## 51  1.0349055  0.02505485 0.4841270  0.00000000 0.01374643 0.00000000
```

```
## 52  1.0349055  0.02703130 0.4841270  0.00000000 0.01374643 0.00000000
## 53  1.0349055  0.04728802 0.4841270  0.00000000 0.01374643 0.00000000
## 54  1.0349055  0.17798175 0.4841270  0.00000000 0.01374643 0.00000000
## 55  1.0349055  0.24807464 0.4841270  0.00000000 0.01374643 0.00000000
## 56  1.0349055  6.19387869 0.4674603 -0.03333333 0.01512108 0.05773503
## 57  1.0349055  6.33143872 0.4674603 -0.03333333 0.01512108 0.05773503
## 58  1.0349055  6.65874937 0.4674603 -0.03333333 0.01512108 0.05773503
## 59  1.0349055 10.60017542 0.4674603 -0.03333333 0.01512108 0.05773503
## 60  1.0349055 83.51410509 0.4674603 -0.03333333 0.01512108 0.05773503
## 61  1.2560276  0.02505485 0.4841270  0.00000000 0.01374643 0.00000000
## 62  1.2560276  0.02703130 0.4841270  0.00000000 0.01374643 0.00000000
## 63  1.2560276  0.04728802 0.4841270  0.00000000 0.01374643 0.00000000
## 64  1.2560276  0.17798175 0.4841270  0.00000000 0.01374643 0.00000000
## 65  1.2560276  0.24807464 0.4841270  0.00000000 0.01374643 0.00000000
## 66  1.2560276  6.19387869 0.4674603 -0.03333333 0.01512108 0.05773503
## 67  1.2560276  6.33143872 0.4674603 -0.03333333 0.01512108 0.05773503
## 68  1.2560276  6.65874937 0.4674603 -0.03333333 0.01512108 0.05773503
## 69  1.2560276 10.60017542 0.4674603 -0.03333333 0.01512108 0.05773503
## 70  1.2560276 83.51410509 0.4674603 -0.03333333 0.01512108 0.05773503
## 71  1.2903930  0.02505485 0.4841270  0.00000000 0.01374643 0.00000000
## 72  1.2903930  0.02703130 0.4841270  0.00000000 0.01374643 0.00000000
## 73  1.2903930  0.04728802 0.4841270  0.00000000 0.01374643 0.00000000
## 74  1.2903930  0.17798175 0.4841270  0.00000000 0.01374643 0.00000000
## 75  1.2903930  0.24807464 0.4841270  0.00000000 0.01374643 0.00000000
## 76  1.2903930  6.19387869 0.4674603 -0.03333333 0.01512108 0.05773503
## 77  1.2903930  6.33143872 0.4674603 -0.03333333 0.01512108 0.05773503
## 78  1.2903930  6.65874937 0.4674603 -0.03333333 0.01512108 0.05773503
## 79  1.2903930 10.60017542 0.4674603 -0.03333333 0.01512108 0.05773503
## 80  1.2903930 83.51410509 0.4674603 -0.03333333 0.01512108 0.05773503
## 81  1.5929506  0.02505485 0.4841270  0.00000000 0.01374643 0.00000000
## 82  1.5929506  0.02703130 0.4841270  0.00000000 0.01374643 0.00000000
## 83  1.5929506  0.04728802 0.4841270  0.00000000 0.01374643 0.00000000
## 84  1.5929506  0.17798175 0.4841270  0.00000000 0.01374643 0.00000000
## 85  1.5929506  0.24807464 0.4841270  0.00000000 0.01374643 0.00000000
## 86  1.5929506  6.19387869 0.4674603 -0.03333333 0.01512108 0.05773503
## 87  1.5929506  6.33143872 0.4674603 -0.03333333 0.01512108 0.05773503
## 88  1.5929506  6.65874937 0.4674603 -0.03333333 0.01512108 0.05773503
## 89  1.5929506 10.60017542 0.4674603 -0.03333333 0.01512108 0.05773503
## 90  1.5929506 83.51410509 0.4674603 -0.03333333 0.01512108 0.05773503
## 91  1.7937243  0.02505485 0.4674603 -0.03333333 0.01512108 0.05773503
## 92  1.7937243  0.02703130 0.4674603 -0.03333333 0.01512108 0.05773503
## 93  1.7937243  0.04728802 0.4674603 -0.03333333 0.01512108 0.05773503
## 94  1.7937243  0.17798175 0.4841270  0.00000000 0.01374643 0.00000000
## 95  1.7937243  0.24807464 0.4841270  0.00000000 0.01374643 0.00000000
## 96  1.7937243  6.19387869 0.4674603 -0.03333333 0.01512108 0.05773503
## 97  1.7937243  6.33143872 0.4674603 -0.03333333 0.01512108 0.05773503
## 98  1.7937243  6.65874937 0.4674603 -0.03333333 0.01512108 0.05773503
## 99  1.7937243 10.60017542 0.4674603 -0.03333333 0.01512108 0.05773503
## 100 1.7937243 83.51410509 0.4674603 -0.03333333 0.01512108 0.05773503
```

```r
# Plotting accuracy vs cost/sigma
plot(svm)
```




```r
# Extracting the best model
best_sigma <- svm$bestTune$sigma
best_C <- svm$bestTune$C

svm_best_model <- train(Class ~ .,  data = train_data, method = "svmRadial",
                        trControl = svm_ctrl,
                        tuneGrid = data.frame(sigma = best_sigma, C = best_C))

#Making predictions of SVM MODEL
svm_predictions <- predict(svm_best_model, newdata = test_data)

# Evaluating the SVM MODEL
svm_confusion_matrix <- table(Actual = test_data$Class, Predicted = svm_predictions)
svm_confusion_matrix
```

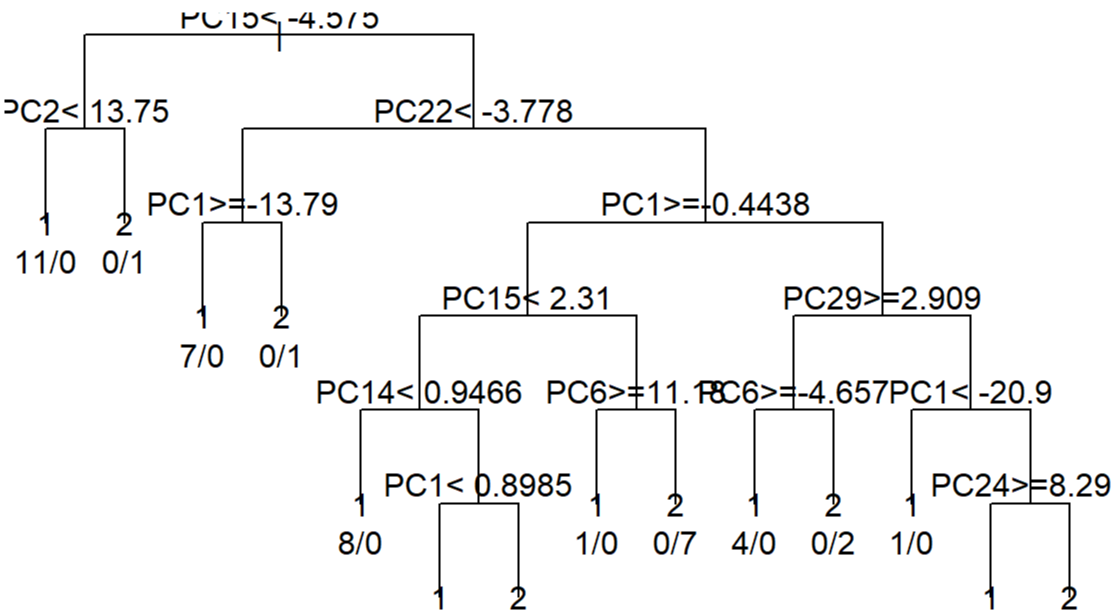
```
##       Predicted
## Actual 1 2
##      1 3 0
##      2 7 6
```

```
# Accuracy calculation
svm_accuracy <- sum(diag(svm_confusion_matrix)) / sum(svm_confusion_matrix)
svm_accuracy
```

```
## [1] 0.5625
```

CREATING AND TUNING RANDOM FOREST MODEL FOR BEST RESULT:

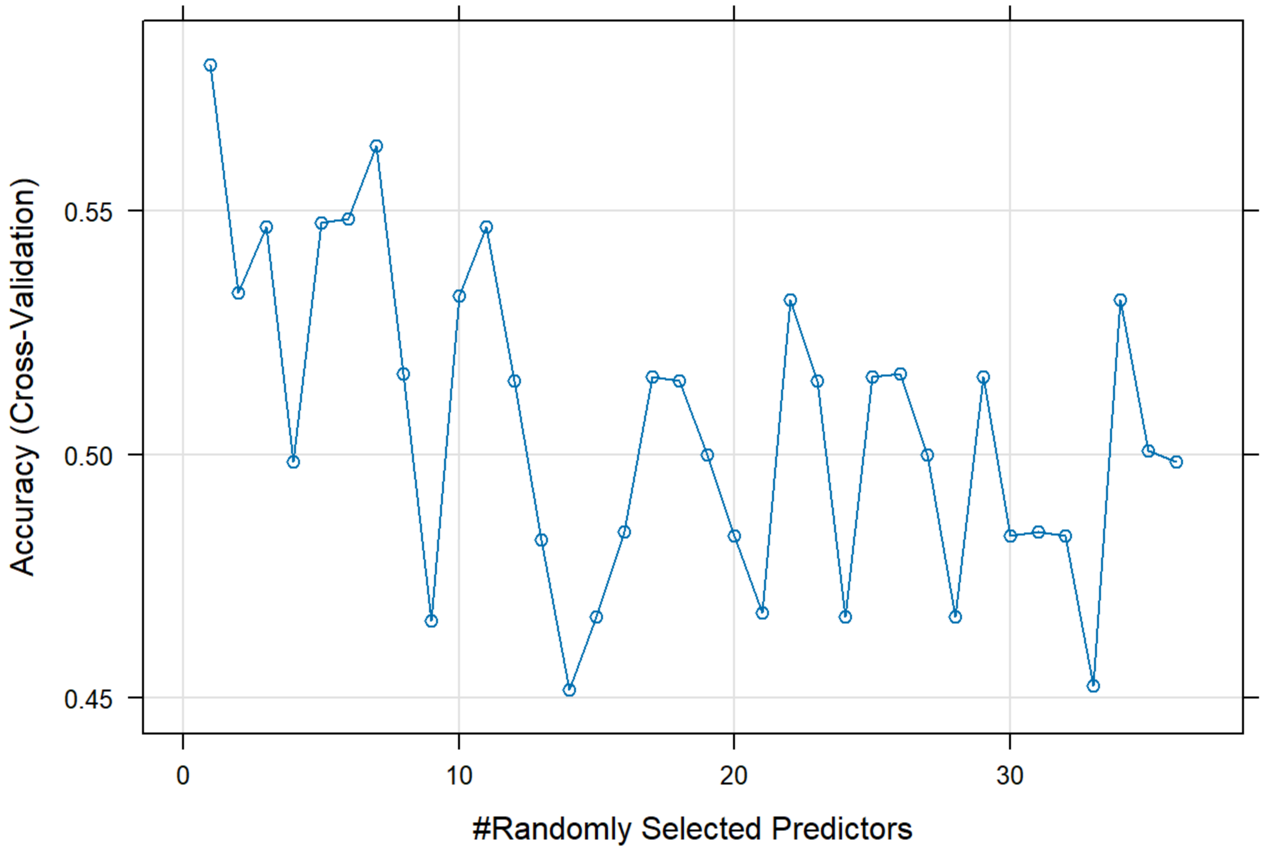
```
# Creating a sample decision tree
tree <- rpart(Class ~. , method = "class" , control = rpart.control(cp = 0 , minsplit = 1) , dat
a = df)
plot(tree, uniform = TRUE)
text(tree, use.n = TRUE)
```



```
# Defining control parameters for model

# Using bootstrapping for cross validation by taking 500 samples
# mtry = number of features to include in creating random forests
# ntree = number of trees

rf_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training Random Forest model with grid search for hyper-parameter tuning
rf_tune_grid <- expand.grid(mtry = seq(1, ncol(train_data) - 1))

rf <- train(Class ~ ., data = train_data, method = "rf", ntree = 500,
            trControl = rf_ctrl,
            tuneGrid = rf_tune_grid)

# Printing cross-validation results
print(rf$results)
```

```
##    mtry  Accuracy        Kappa AccuracySD    KappaSD
## 1     1 0.5801587  0.167747748 0.03534197 0.07193523
## 2     2 0.5333333  0.075541854 0.06245180 0.11847681
## 3     3 0.5468254  0.098569870 0.08716801 0.17785557
## 4     4 0.4984127  0.002432127 0.08849117 0.18007567
## 5     5 0.5476190  0.101598793 0.04123930 0.08831491
## 6     6 0.5484127  0.102173474 0.02384917 0.04924215
## 7     7 0.5634921  0.132121212 0.05991932 0.12062281
## 8     8 0.5166667  0.035901826 0.03741960 0.08075229
## 9     9 0.4658730 -0.064015152 0.10034898 0.20454651
## 10   10 0.5325397  0.066502867 0.01512108 0.02932629
## 11   11 0.5468254  0.095662100 0.08716801 0.17862186
## 12   12 0.5150794  0.035765460 0.06118322 0.12456873
## 13   13 0.4825397 -0.028416290 0.08589047 0.18058932
## 14   14 0.4515873 -0.097189085 0.11905556 0.24059232
## 15   15 0.4666667 -0.063473298 0.06245180 0.12503302
## 16   16 0.4841270 -0.030165913 0.09622504 0.19183244
## 17   17 0.5158730  0.030579144 0.04956348 0.09488797
## 18   18 0.5150794  0.030346785 0.06118322 0.12025869
## 19   19 0.5000000  0.006059872 0.02380952 0.04526947
## 20   20 0.4833333 -0.030398272 0.03741960 0.07809333
## 21   21 0.4674603 -0.060360360 0.04996219 0.10063107
## 22   22 0.5317460  0.071351351 0.03636965 0.08139040
## 23   23 0.5150794  0.030371589 0.09098645 0.18048259
## 24   24 0.4666667 -0.063473298 0.06245180 0.12503302
## 25   25 0.5158730  0.033169533 0.01374643 0.02904568
## 26   26 0.5166667  0.039393205 0.03741960 0.06911347
## 27   27 0.5000000  0.006059872 0.02380952 0.04526947
## 28   28 0.4666667 -0.063446252 0.09184429 0.18323648
## 29   29 0.5158730  0.035984848 0.01374643 0.03230819
## 30   30 0.4833333 -0.030398272 0.03741960 0.07809333
## 31   31 0.4841270 -0.027027027 0.04956348 0.09744733
## 32   32 0.4833333 -0.030398272 0.03741960 0.07809333
## 33   33 0.4523810 -0.093514329 0.06299408 0.12485190
## 34   34 0.5317460  0.069898990 0.07653691 0.14477789
## 35   35 0.5007937  0.006860378 0.10433716 0.20136692
## 36   36 0.4984127  0.002432127 0.08849117 0.18007567
```

```
# Plotting accuracy vs. mtry
plot(rf)
```




```
# Selecting the best model
best_mtry <- rf$bestTune$mtry
rf_best_model <- train(Class ~ ., data = train_data, method = "rf",
                       trControl = rf_ctrl,
                       tuneGrid = data.frame(mtry = best_mtry))
rf_best_model
```

```
## Random Forest
##
## 62 samples
## 36 predictors
##  2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 41, 42, 41
## Resampling results:
##
##   Accuracy   Kappa
##   0.6126984  0.2285345
##
## Tuning parameter 'mtry' was held constant at a value of 1
```

```
#Making predictions
rf_predictions <- predict(rf_best_model, newdata = test_data)

# Evaluating the model
rf_confusion_matrix <- table(Actual = test_data$Class, Predicted = rf_predictions)
rf_confusion_matrix
```

```
##       Predicted
## Actual 1 2
##      1 1 2
##      2 5 8
```

```
# Accuracy calculation
rf_accuracy <- sum(diag(rf_confusion_matrix)) / sum(rf_confusion_matrix)
rf_accuracy
```

```
## [1] 0.5625
```

CREATING AND TUNING XG BOOST MODEL FOR BEST RESULT:

```
# Defining control parameters for model
xgb_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training XGBoost model with hyperparameters
xgb <- train(Class ~ ., data = train_data, method = "xgbTree",
             trControl = xgb_ctrl,
             tuneLength = 3)  # Tune over a limited number of parameter combinations
```

```
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:18] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:19] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:20] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:21] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:22] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:23] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:24] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:25] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:26] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:27] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
## [23:13:28] WARNING: src/c_api/c_api.cc:935: `ntree_limit` is deprecated, use `iteration_range
` instead.
```

```
# Printing cross-validation results
print(xgb$results)
```

```
##     eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1   0.3         1     0              0.6                1      0.50      50
## 4   0.3         1     0              0.6                1      0.75      50
## 7   0.3         1     0              0.6                1      1.00      50
## 10  0.3         1     0              0.8                1      0.50      50
## 13  0.3         1     0              0.8                1      0.75      50
## 16  0.3         1     0              0.8                1      1.00      50
## 55  0.4         1     0              0.6                1      0.50      50
## 58  0.4         1     0              0.6                1      0.75      50
## 61  0.4         1     0              0.6                1      1.00      50
## 64  0.4         1     0              0.8                1      0.50      50
## 67  0.4         1     0              0.8                1      0.75      50
## 70  0.4         1     0              0.8                1      1.00      50
## 19  0.3         2     0              0.6                1      0.50      50
## 22  0.3         2     0              0.6                1      0.75      50
## 25  0.3         2     0              0.6                1      1.00      50
## 28  0.3         2     0              0.8                1      0.50      50
## 31  0.3         2     0              0.8                1      0.75      50
## 34  0.3         2     0              0.8                1      1.00      50
## 73  0.4         2     0              0.6                1      0.50      50
## 76  0.4         2     0              0.6                1      0.75      50
## 79  0.4         2     0              0.6                1      1.00      50
## 82  0.4         2     0              0.8                1      0.50      50
## 85  0.4         2     0              0.8                1      0.75      50
## 88  0.4         2     0              0.8                1      1.00      50
## 37  0.3         3     0              0.6                1      0.50      50
## 40  0.3         3     0              0.6                1      0.75      50
## 43  0.3         3     0              0.6                1      1.00      50
## 46  0.3         3     0              0.8                1      0.50      50
## 49  0.3         3     0              0.8                1      0.75      50
## 52  0.3         3     0              0.8                1      1.00      50
## 91  0.4         3     0              0.6                1      0.50      50
## 94  0.4         3     0              0.6                1      0.75      50
## 97  0.4         3     0              0.6                1      1.00      50
## 100 0.4         3     0              0.8                1      0.50      50
## 103 0.4         3     0              0.8                1      0.75      50
## 106 0.4         3     0              0.8                1      1.00      50
## 2   0.3         1     0              0.6                1      0.50     100
## 5   0.3         1     0              0.6                1      0.75     100
## 8   0.3         1     0              0.6                1      1.00     100
## 11  0.3         1     0              0.8                1      0.50     100
## 14  0.3         1     0              0.8                1      0.75     100
## 17  0.3         1     0              0.8                1      1.00     100
## 56  0.4         1     0              0.6                1      0.50     100
## 59  0.4         1     0              0.6                1      0.75     100
## 62  0.4         1     0              0.6                1      1.00     100
## 65  0.4         1     0              0.8                1      0.50     100
## 68  0.4         1     0              0.8                1      0.75     100
## 71  0.4         1     0              0.8                1      1.00     100
## 20  0.3         2     0              0.6                1      0.50     100
## 23  0.3         2     0              0.6                1      0.75     100
## 26  0.3         2     0              0.6                1      1.00     100
```

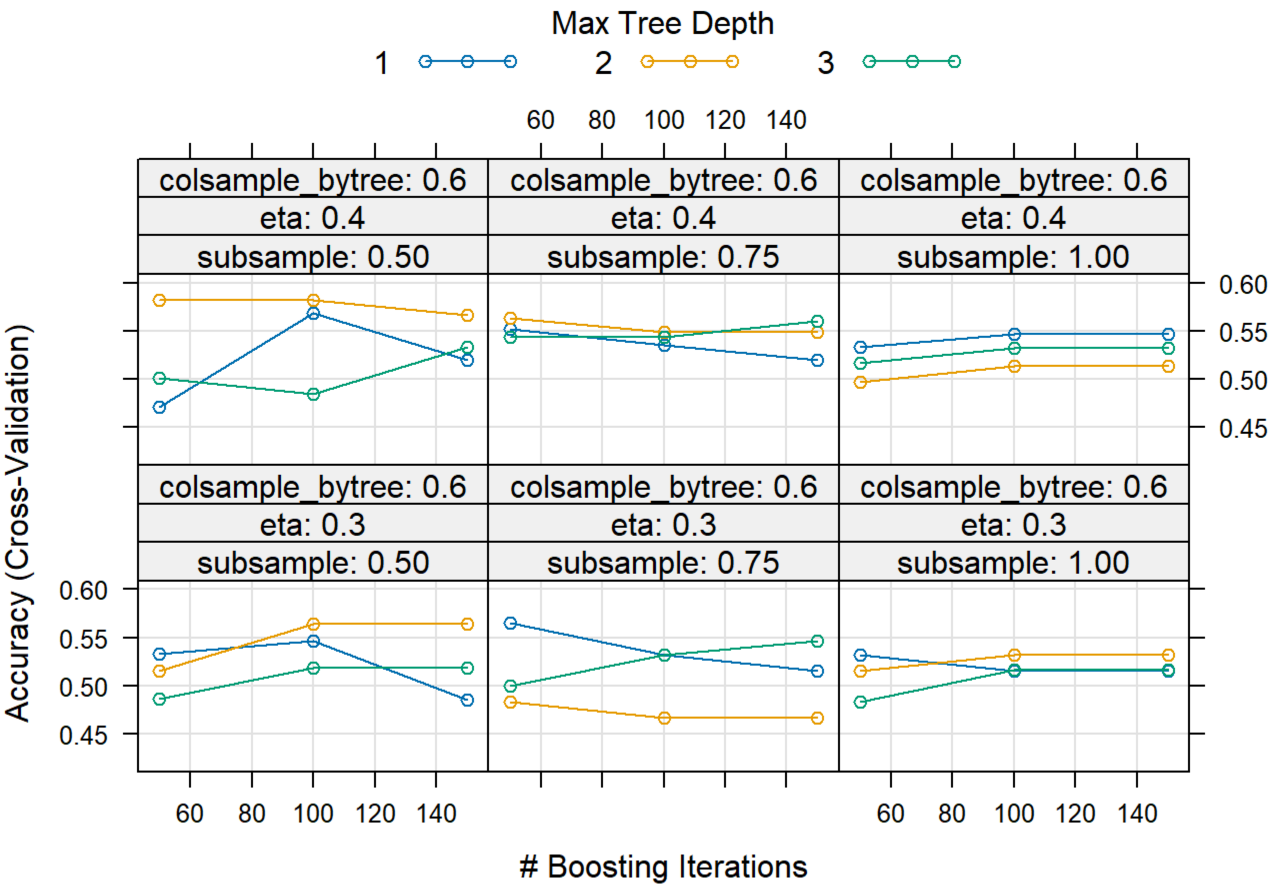
```
## 29  0.3          2     0              0.8                1      0.50     100
## 32  0.3          2     0              0.8                1      0.75     100
## 35  0.3          2     0              0.8                1      1.00     100
## 74  0.4          2     0              0.6                1      0.50     100
## 77  0.4          2     0              0.6                1      0.75     100
## 80  0.4          2     0              0.6                1      1.00     100
## 83  0.4          2     0              0.8                1      0.50     100
## 86  0.4          2     0              0.8                1      0.75     100
## 89  0.4          2     0              0.8                1      1.00     100
## 38  0.3          3     0              0.6                1      0.50     100
## 41  0.3          3     0              0.6                1      0.75     100
## 44  0.3          3     0              0.6                1      1.00     100
## 47  0.3          3     0              0.8                1      0.50     100
## 50  0.3          3     0              0.8                1      0.75     100
## 53  0.3          3     0              0.8                1      1.00     100
## 92  0.4          3     0              0.6                1      0.50     100
## 95  0.4          3     0              0.6                1      0.75     100
## 98  0.4          3     0              0.6                1      1.00     100
## 101 0.4          3     0              0.8                1      0.50     100
## 104 0.4          3     0              0.8                1      0.75     100
## 107 0.4          3     0              0.8                1      1.00     100
## 3   0.3          1     0              0.6                1      0.50     150
## 6   0.3          1     0              0.6                1      0.75     150
## 9   0.3          1     0              0.6                1      1.00     150
## 12  0.3          1     0              0.8                1      0.50     150
## 15  0.3          1     0              0.8                1      0.75     150
## 18  0.3          1     0              0.8                1      1.00     150
## 57  0.4          1     0              0.6                1      0.50     150
## 60  0.4          1     0              0.6                1      0.75     150
## 63  0.4          1     0              0.6                1      1.00     150
## 66  0.4          1     0              0.8                1      0.50     150
## 69  0.4          1     0              0.8                1      0.75     150
## 72  0.4          1     0              0.8                1      1.00     150
## 21  0.3          2     0              0.6                1      0.50     150
## 24  0.3          2     0              0.6                1      0.75     150
## 27  0.3          2     0              0.6                1      1.00     150
## 30  0.3          2     0              0.8                1      0.50     150
## 33  0.3          2     0              0.8                1      0.75     150
## 36  0.3          2     0              0.8                1      1.00     150
## 75  0.4          2     0              0.6                1      0.50     150
## 78  0.4          2     0              0.6                1      0.75     150
## 81  0.4          2     0              0.6                1      1.00     150
## 84  0.4          2     0              0.8                1      0.50     150
## 87  0.4          2     0              0.8                1      0.75     150
## 90  0.4          2     0              0.8                1      1.00     150
## 39  0.3          3     0              0.6                1      0.50     150
## 42  0.3          3     0              0.6                1      0.75     150
## 45  0.3          3     0              0.6                1      1.00     150
## 48  0.3          3     0              0.8                1      0.50     150
## 51  0.3          3     0              0.8                1      0.75     150
## 54  0.3          3     0              0.8                1      1.00     150
## 93  0.4          3     0              0.6                1      0.50     150
```

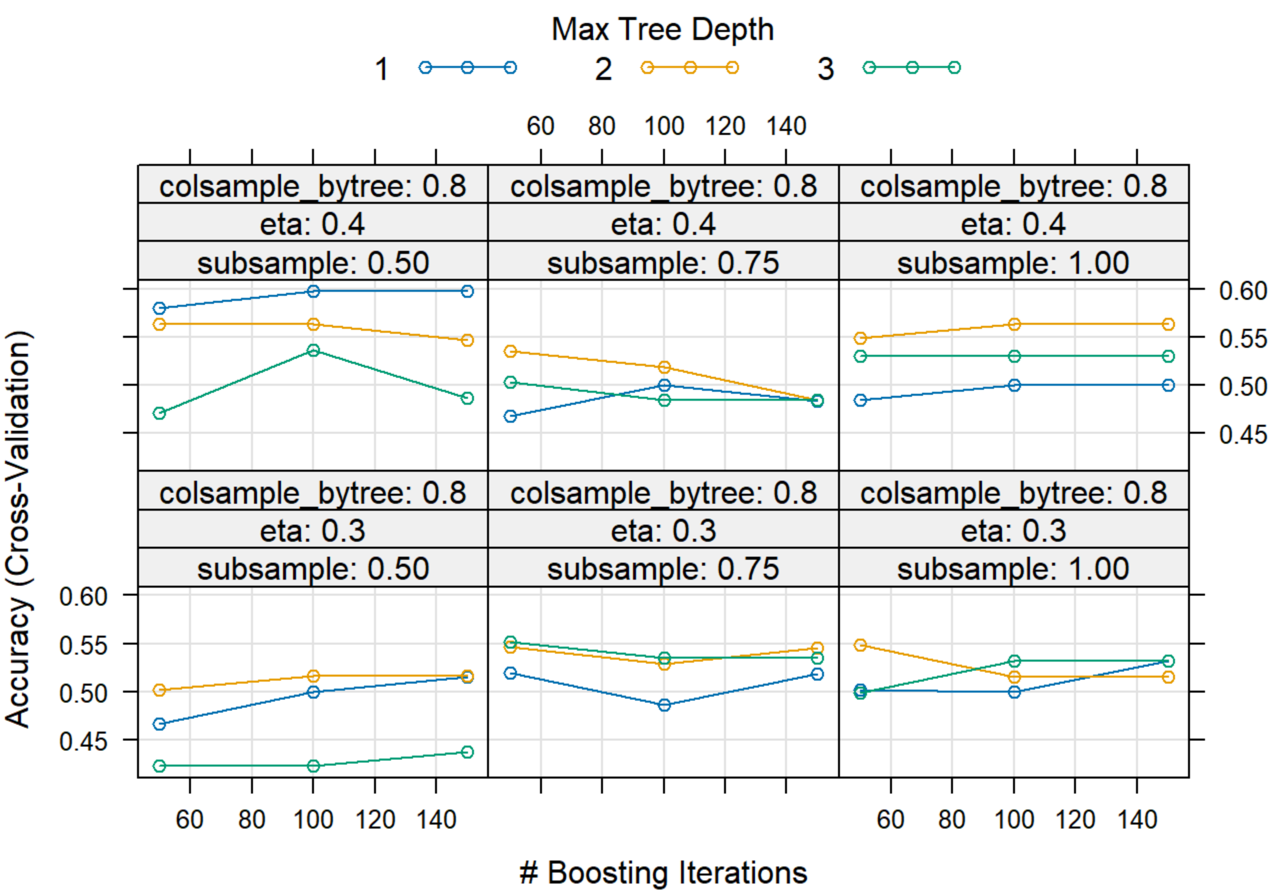
```
## 96  0.4         3     0             0.6               1      0.75     150
## 99  0.4         3     0             0.6               1      1.00     150
## 102 0.4         3     0             0.8               1      0.50     150
## 105 0.4         3     0             0.8               1      0.75     150
## 108 0.4         3     0             0.8               1      1.00     150
##      Accuracy         Kappa  AccuracySD    KappaSD
## 1   0.5333333  6.666667e-02 0.104083300 0.20816660
## 4   0.5651515  1.303030e-01 0.030265128 0.06053026
## 7   0.5318182  6.363636e-02 0.027648921 0.05529784
## 10  0.4666667 -6.666667e-02 0.057735027 0.11547005
## 13  0.5196970  3.939394e-02 0.121656383 0.24331277
## 16  0.5015152  3.030303e-03 0.085320656 0.17064131
## 55  0.4712121 -5.757576e-02 0.155898710 0.31179742
## 58  0.5515152  1.030303e-01 0.130584615 0.26116923
## 61  0.5333333  6.666667e-02 0.057735027 0.11547005
## 64  0.5803030  1.606061e-01 0.026633933 0.05326787
## 67  0.4681818 -6.363636e-02 0.027648921 0.05529784
## 70  0.4848485 -3.030303e-02 0.056468622 0.11293724
## 19  0.5151515  3.030303e-02 0.026243194 0.05248639
## 22  0.4833333 -3.333333e-02 0.028867513 0.05773503
## 25  0.5151515  3.030303e-02 0.026243194 0.05248639
## 28  0.5015152  3.030303e-03 0.085320656 0.17064131
## 31  0.5469697  9.393939e-02 0.045530240 0.09106048
## 34  0.5484848  9.696970e-02 0.050068823 0.10013765
## 73  0.5818182  1.636364e-01 0.059090909 0.11818182
## 76  0.5636364  1.272727e-01 0.102751405 0.20550281
## 79  0.4969697 -6.060606e-03 0.081353902 0.16270780
## 82  0.5636364  1.272727e-01 0.055297841 0.11059568
## 85  0.5348485  6.969697e-02 0.143043464 0.28608693
## 88  0.5484848  9.696970e-02 0.050068823 0.10013765
## 37  0.4863636 -2.727273e-02 0.071437426 0.14287485
## 40  0.5000000  3.700743e-17 0.050000000 0.10000000
## 43  0.4833333 -3.333333e-02 0.028867513 0.05773503
## 46  0.4227273 -1.545455e-01 0.093927174 0.18785435
## 49  0.5515152  1.030303e-01 0.097736081 0.19547216
## 52  0.4984848 -3.030303e-03 0.047745307 0.09549061
## 91  0.5015152  3.030303e-03 0.047745307 0.09549061
## 94  0.5439394  8.787879e-02 0.121995574 0.24399115
## 97  0.5166667  3.333333e-02 0.028867513 0.05773503
## 100 0.4712121 -5.757576e-02 0.119601036 0.23920207
## 103 0.5030303  6.060606e-03 0.081353902 0.16270780
## 106 0.5303030  6.060606e-02 0.052486388 0.10497278
## 2   0.5469697  9.393939e-02 0.084101146 0.16820229
## 5   0.5318182  6.363636e-02 0.027648921 0.05529784
## 8   0.5151515  3.030303e-02 0.056468622 0.11293724
## 11  0.5000000  3.700743e-17 0.050000000 0.10000000
## 14  0.4863636 -2.727273e-02 0.071437426 0.14287485
## 17  0.5000000  3.700743e-17 0.050000000 0.10000000
## 56  0.5681818  1.363636e-01 0.101537763 0.20307553
## 59  0.5348485  6.969697e-02 0.102281144 0.20456229
## 62  0.5469697  9.393939e-02 0.045530240 0.09106048
## 65  0.5969697  1.939394e-01 0.050274727 0.10054945
```

```
## 68  0.5000000  3.700743e-17 0.050000000 0.10000000
## 71  0.5000000  3.700743e-17 0.050000000 0.10000000
## 20  0.5636364  1.272727e-01 0.055297841 0.11059568
## 23  0.4666667 -6.666667e-02 0.028867513 0.05773503
## 26  0.5318182  6.363636e-02 0.027648921 0.05529784
## 29  0.5166667  3.333333e-02 0.076376262 0.15275252
## 32  0.5287879  5.757576e-02 0.096459358 0.19291872
## 35  0.5151515  3.030303e-02 0.056468622 0.11293724
## 74  0.5818182  1.636364e-01 0.059090909 0.11818182
## 77  0.5484848  9.696970e-02 0.100034429 0.20006886
## 80  0.5136364  2.727273e-02 0.071437426 0.14287485
## 83  0.5636364  1.272727e-01 0.055297841 0.11059568
## 86  0.5181818  3.636364e-02 0.055110708 0.11022142
## 89  0.5636364  1.272727e-01 0.055297841 0.11059568
## 38  0.5181818  3.636364e-02 0.055110708 0.11022142
## 41  0.5318182  6.363636e-02 0.075924059 0.15184812
## 44  0.5166667  3.333333e-02 0.076376262 0.15275252
## 47  0.4227273 -1.545455e-01 0.093927174 0.18785435
## 50  0.5348485  6.969697e-02 0.073901505 0.14780301
## 53  0.5318182  6.363636e-02 0.027648921 0.05529784
## 92  0.4848485 -3.030303e-02 0.026243194 0.05248639
## 95  0.5439394  8.787879e-02 0.121995574 0.24399115
## 98  0.5318182  6.363636e-02 0.027648921 0.05529784
## 101 0.5363636  7.272727e-02 0.121032063 0.24206413
## 104 0.4848485 -3.030303e-02 0.026243194 0.05248639
## 107 0.5303030  6.060606e-02 0.052486388 0.10497278
## 3   0.4848485 -3.030303e-02 0.056468622 0.11293724
## 6   0.5151515  3.030303e-02 0.026243194 0.05248639
## 9   0.5151515  3.030303e-02 0.056468622 0.11293724
## 12  0.5151515  3.030303e-02 0.056468622 0.11293724
## 15  0.5181818  3.636364e-02 0.074412298 0.14882460
## 18  0.5318182  6.363636e-02 0.075924059 0.15184812
## 57  0.5196970  3.939394e-02 0.098996341 0.19799268
## 60  0.5196970  3.939394e-02 0.121656383 0.24331277
## 63  0.5469697  9.393939e-02 0.045530240 0.09106048
## 66  0.5969697  1.939394e-01 0.005248639 0.01049728
## 69  0.4833333 -3.333333e-02 0.028867513 0.05773503
## 72  0.5000000  3.700743e-17 0.050000000 0.10000000
## 21  0.5636364  1.272727e-01 0.055297841 0.11059568
## 24  0.4666667 -6.666667e-02 0.028867513 0.05773503
## 27  0.5318182  6.363636e-02 0.027648921 0.05529784
## 30  0.5166667  3.333333e-02 0.076376262 0.15275252
## 33  0.5454545  9.090909e-02 0.078729582 0.15745916
## 36  0.5151515  3.030303e-02 0.056468622 0.11293724
## 75  0.5666667  1.333333e-01 0.076376262 0.15275252
## 78  0.5484848  9.696970e-02 0.100034429 0.20006886
## 81  0.5136364  2.727273e-02 0.071437426 0.14287485
## 84  0.5469697  9.393939e-02 0.084101146 0.16820229
## 87  0.4848485 -3.030303e-02 0.056468622 0.11293724
## 90  0.5636364  1.272727e-01 0.055297841 0.11059568
## 39  0.5181818  3.636364e-02 0.055110708 0.11022142
## 42  0.5469697  9.393939e-02 0.084101146 0.16820229
```

```
## 45  0.5166667  3.333333e-02 0.076376262 0.15275252
## 48  0.4378788 -1.242424e-01 0.068985166 0.13797033
## 51  0.5348485  6.969697e-02 0.073901505 0.14780301
## 54  0.5318182  6.363636e-02 0.027648921 0.05529784
## 93  0.5333333  6.666667e-02 0.057735027 0.11547005
## 96  0.5606061  1.212121e-01 0.104972776 0.20994555
## 99  0.5318182  6.363636e-02 0.027648921 0.05529784
## 102 0.4863636 -2.727273e-02 0.100515202 0.20103040
## 105 0.4848485 -3.030303e-02 0.026243194 0.05248639
## 108 0.5303030  6.060606e-02 0.052486388 0.10497278
```

```
# Plotting accuracy vs. hyperparameters
plot(xgb)
```


Max Tree Depth
1
2
3
colsample_bytree: 0.6
eta: 0.4
subsample: 0.50
subsample: 0.75
subsample: 1.00
eta: 0.3
Accuracy (Cross-Validation)
Boosting Iterations
0.45
0.50
0.55
0.60
60
80
100
120
140


Max Tree Depth
1
2
3
colsample_bytree: 0.8
eta: 0.4
subsample: 0.50
subsample: 0.75
subsample: 1.00
eta: 0.3
Accuracy (Cross-Validation)
Boosting Iterations
0.45
0.50
0.55
0.60
60
80
100
120
140

```
# Making predictions
xgb_predictions <- predict(xgb, newdata = test_data)

# Evaluating the model
xgb_confusion_matrix <- table(Actual = test_data$Class, Predicted = xgb_predictions)
xgb_confusion_matrix
```

```
##       Predicted
## Actual 1 2
##      1 3 0
##      2 5 8
```

```
# Accuracy calculation
xgb_accuracy <- sum(diag(xgb_confusion_matrix)) / sum(xgb_confusion_matrix)
xgb_accuracy
```

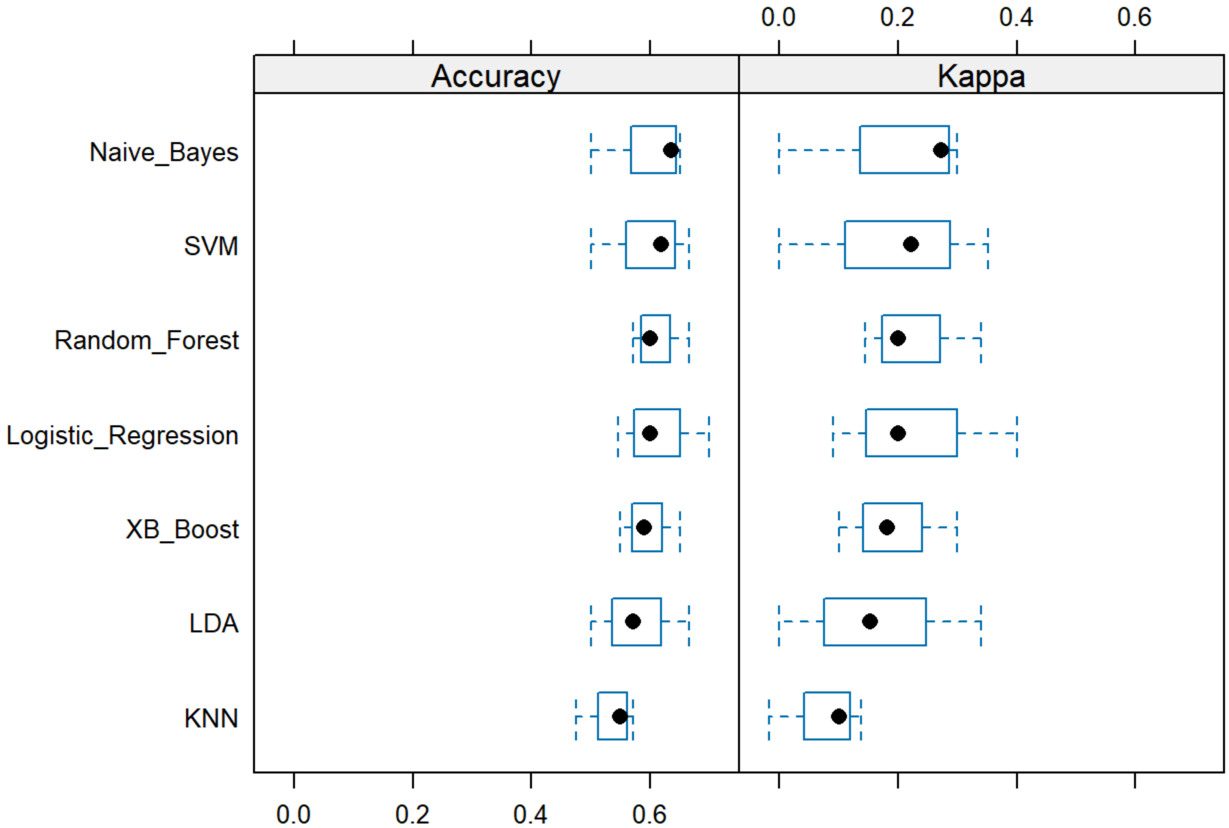
```
## [1] 0.6875
```

Resampling & Evaluating All Models Agaisnt Accuracy:

```
# Create a list of trained models
models <- list(
  Logistic_Regression = lr_best_model,
  LDA = lda_best_model,
  Naive_Bayes = nb_best_model,
  KNN = knn_best_model,
  SVM = svm_best_model,
  Random_Forest = rf_best_model,
  XB_Boost = xgb
  )

# Evaluate models on multiple datasets
model_eval <- resamples(models, data = datasets, method = "accuracy")

# Summarize results
summary(model_eval)
```

```
##
## Call:
## summary.resamples(object = model_eval)
##
## Models: Logistic_Regression, LDA, Naive_Bayes, KNN, SVM, Random_Forest, XB_Boost
## Number of resamples: 3
##
## Accuracy
##                          Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## Logistic_Regression 0.5454545 0.5727273 0.6000000 0.6151515 0.6500000 0.7000000
## LDA                 0.5000000 0.5357143 0.5714286 0.5793651 0.6190476 0.6666667
## Naive_Bayes         0.5000000 0.5681818 0.6363636 0.5954545 0.6431818 0.6500000
## KNN                 0.4761905 0.5130952 0.5500000 0.5325397 0.5607143 0.5714286
## SVM                 0.5000000 0.5595238 0.6190476 0.5952381 0.6428571 0.6666667
## Random_Forest       0.5714286 0.5857143 0.6000000 0.6126984 0.6333333 0.6666667
## XB_Boost            0.5500000 0.5704545 0.5909091 0.5969697 0.6204545 0.6500000
##                     NA's
## Logistic_Regression    0
## LDA                    0
## Naive_Bayes            0
## KNN                    0
## SVM                    0
## Random_Forest          0
## XB_Boost               0
##
## Kappa
##                            Min.    1st Qu.    Median       Mean   3rd Qu.
## Logistic_Regression  0.09090909 0.14545455 0.2000000 0.23030303 0.3000000
## LDA                  0.00000000 0.07623318 0.1524664 0.16442451 0.2466368
## Naive_Bayes          0.00000000 0.13636364 0.2727273 0.19090909 0.2863636
## KNN                 -0.01762115 0.04118943 0.1000000 0.07312172 0.1184932
## SVM                  0.00000000 0.11111111 0.2222222 0.19154838 0.2873226
## Random_Forest        0.14479638 0.17239819 0.2000000 0.22853452 0.2704036
## XB_Boost             0.10000000 0.14090909 0.1818182 0.19393939 0.2409091
##                          Max. NA's
## Logistic_Regression 0.4000000    0
## LDA                 0.3408072    0
## Naive_Bayes         0.3000000    0
## KNN                 0.1369863    0
## SVM                 0.3524229    0
## Random_Forest       0.3408072    0
## XB_Boost            0.3000000    0
```

```
# Visualize results
bwplot(model_eval)
```



Evaluating All Models:

```
model_predictions <- lapply(models, function(model) {
  predict(model, newdata = test_data)
})

# Evaluate performance metrics (e.g., accuracy, precision, recall) on the testing dataset for ea
ch model
model_metrics <- lapply(model_predictions, function(predictions) {
  confusionMatrix(predictions, test_data$Class)
})

# Model metrics
model_metrics
```

```
## $Logistic_Regression
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2
##          1 2 5
##          2 1 8
##
##                Accuracy : 0.625
##                  95% CI : (0.3543, 0.848)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.9810
##
##                   Kappa : 0.1864
##
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.6667
##             Specificity : 0.6154
##          Pos Pred Value : 0.2857
##          Neg Pred Value : 0.8889
##              Prevalence : 0.1875
##          Detection Rate : 0.1250
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.6410
##
##        'Positive' Class : 1
##
##
## $LDA
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2
##          1 2 4
##          2 1 9
##
##                Accuracy : 0.6875
##                  95% CI : (0.4134, 0.8898)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.9373
##
##                   Kappa : 0.2593
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.6667
##             Specificity : 0.6923
##          Pos Pred Value : 0.3333
##          Neg Pred Value : 0.9000
##              Prevalence : 0.1875
##          Detection Rate : 0.1250
```

```
##    Detection Prevalence : 0.3750
##       Balanced Accuracy : 0.6795
## 
##        'Positive' Class : 1
## 
## 
## $Naive_Bayes
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction 1 2
##          1 1 4
##          2 2 9
## 
##                Accuracy : 0.625
##                  95% CI : (0.3543, 0.848)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.9810
## 
##                   Kappa : 0.0204
## 
##  Mcnemar's Test P-Value : 0.6831
## 
##             Sensitivity : 0.3333
##             Specificity : 0.6923
##          Pos Pred Value : 0.2000
##          Neg Pred Value : 0.8182
##              Prevalence : 0.1875
##          Detection Rate : 0.0625
##    Detection Prevalence : 0.3125
##       Balanced Accuracy : 0.5128
## 
##        'Positive' Class : 1
## 
## 
## $KNN
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction  1  2
##          1  3  2
##          2  0 11
## 
##                Accuracy : 0.875
##                  95% CI : (0.6165, 0.9845)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.3998
## 
##                   Kappa : 0.6735
## 
##  Mcnemar's Test P-Value : 0.4795
## 
```

```
##             Sensitivity : 1.0000
##             Specificity : 0.8462
##          Pos Pred Value : 0.6000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1875
##          Detection Rate : 0.1875
##    Detection Prevalence : 0.3125
##       Balanced Accuracy : 0.9231
##
##        'Positive' Class : 1
##
##
## $SVM
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2
##          1 3 7
##          2 0 6
##
##                Accuracy : 0.5625
##                  95% CI : (0.2988, 0.8025)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.99536
##
##                   Kappa : 0.2432
##
##  Mcnemar's Test P-Value : 0.02334
##
##             Sensitivity : 1.0000
##             Specificity : 0.4615
##          Pos Pred Value : 0.3000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1875
##          Detection Rate : 0.1875
##    Detection Prevalence : 0.6250
##       Balanced Accuracy : 0.7308
##
##        'Positive' Class : 1
##
##
## $Random_Forest
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2
##          1 1 5
##          2 2 8
##
##                Accuracy : 0.5625
##                  95% CI : (0.2988, 0.8025)
##     No Information Rate : 0.8125
```

```
##     P-Value [Acc > NIR] : 0.9954
##
##                   Kappa : -0.037
##
##  Mcnemar's Test P-Value : 0.4497
##
##             Sensitivity : 0.3333
##             Specificity : 0.6154
##          Pos Pred Value : 0.1667
##          Neg Pred Value : 0.8000
##              Prevalence : 0.1875
##          Detection Rate : 0.0625
##    Detection Prevalence : 0.3750
##       Balanced Accuracy : 0.4744
##
##        'Positive' Class : 1
##
##
## $XB_Boost
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 1 2
##          1 3 5
##          2 0 8
##
##                Accuracy : 0.6875
##                  95% CI : (0.4134, 0.8898)
##     No Information Rate : 0.8125
##     P-Value [Acc > NIR] : 0.93735
##
##                   Kappa : 0.375
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 1.0000
##             Specificity : 0.6154
##          Pos Pred Value : 0.3750
##          Neg Pred Value : 1.0000
##              Prevalence : 0.1875
##          Detection Rate : 0.1875
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8077
##
##        'Positive' Class : 1
##
```

```
# Model Accuracy on test dataset
model_accuracy_test_dataset <- sapply(model_metrics, function(metrics) {
  metrics$overall["Accuracy"]
})

# Compare performance metrics of models
model_accuracy_test_dataset
```

```
## Logistic_Regression.Accuracy                  LDA.Accuracy
##                       0.6250                        0.6875
##         Naive_Bayes.Accuracy                  KNN.Accuracy
##                       0.6250                        0.8750
##                 SVM.Accuracy        Random_Forest.Accuracy
##                       0.5625                        0.5625
##            XB_Boost.Accuracy
##                       0.6875
```

```
# Get the name of the best model based on accuracy
best_model_name <- names(model_accuracy_test_dataset[model_accuracy_test_dataset == max(model_ac
curacy_test_dataset)])

# Print the best model based on accuracy
print("Best model based on accuracy:")
```

```
## [1] "Best model based on accuracy:"
```

```
print(best_model_name)
```

```
## [1] "KNN.Accuracy"
```

PART 4: Investigating Best Machine Learning Model:

IV) Investigate if clusters established under II) improve your ‘best’ machine learning model.

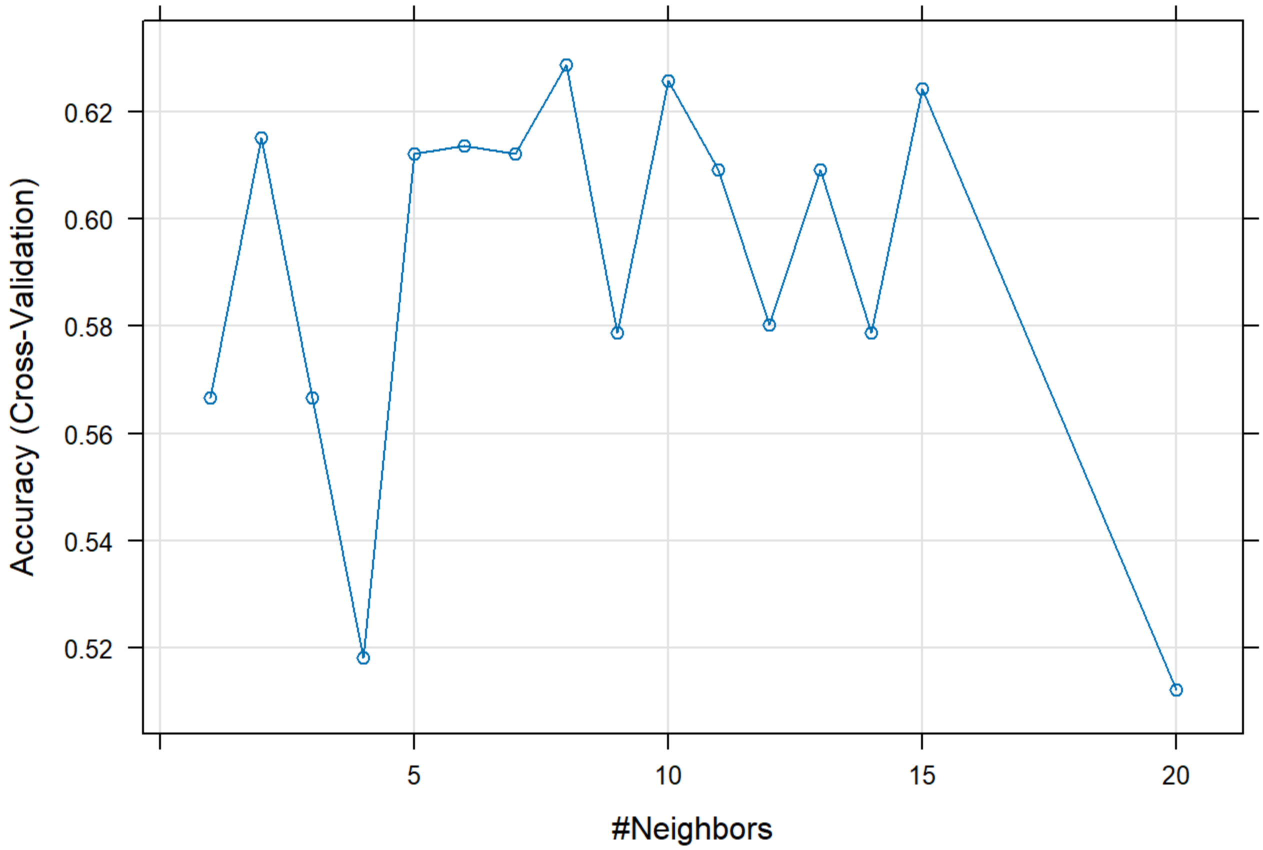
```
# Setting seed for reproducibility
set.seed(2315740)

# Converting class label to factor with two levels
df2 <- filtered_df_w_class
df2$Class <- factor(df2$Class)

# Splitting dataset into test and train
raw_train_index <- sample(1:nrow(df2), 0.80 * nrow(df2))
raw_train_data <- df2[raw_train_index, ]
raw_test_data <- df2[-raw_train_index, ]
```

Testing KNN Model on Raw Dataset (Non-Dimensionally Reduced):

```r
# Defining control parameters for model
raw_knn_ctrl <- trainControl(method = "cv", number = 3)  # 3-fold cross validation

# Training KNN model with grid search for hyper-parameter tuning
raw_k_values <- c(1, 2, 3, 4, 5 , 6, 7, 8, 9 , 10, 11, 12, 13 , 14 , 15, 20)
raw_knn <- train(Class ~ ., data = raw_train_data, method = "knn",
             trControl = raw_knn_ctrl,
             tuneGrid = expand.grid(k = raw_k_values))

# Printing cross-validation results
print(raw_knn$results)
```

```
##     k  Accuracy      Kappa AccuracySD    KappaSD
## 1   1 0.5666667 0.13333333 0.05773503 0.11547005
## 2   2 0.6151515 0.23030303 0.06035935 0.12071869
## 3   3 0.5666667 0.13333333 0.07637626 0.15275252
## 4   4 0.5181818 0.03636364 0.05511071 0.11022142
## 5   5 0.6121212 0.22424242 0.02099456 0.04198911
## 6   6 0.6136364 0.22727273 0.03181818 0.06363636
## 7   7 0.6121212 0.22424242 0.05422888 0.10845776
## 8   8 0.6287879 0.25757576 0.02584655 0.05169310
## 9   9 0.5787879 0.15757576 0.07061321 0.14122643
## 10 10 0.6257576 0.25151515 0.15282014 0.30564028
## 11 11 0.6090909 0.21818182 0.11390876 0.22781753
## 12 12 0.5803030 0.16060606 0.02663393 0.05326787
## 13 13 0.6090909 0.21818182 0.11390876 0.22781753
## 14 14 0.5787879 0.15757576 0.07061321 0.14122643
## 15 15 0.6242424 0.24848485 0.12859165 0.25718330
## 16 20 0.5121212 0.02424242 0.10759710 0.21519419
```

```r
# Plotting accuracy vs. k
plot(raw_knn)
```




```
# Selecting the best model
raw_best_k <- raw_knn$bestTune$k
raw_knn_best_model <- train(Class ~ ., data = raw_train_data, method = "knn",
                  trControl = raw_knn_ctrl,
                  tuneGrid = data.frame(k = raw_best_k))
raw_knn_best_model
```

```
## k-Nearest Neighbors
##
##   62 samples
## 1429 predictors
##    2 classes: '1', '2'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold)
## Summary of sample sizes: 42, 40, 42
## Resampling results:
##
##   Accuracy   Kappa
##   0.6151515  0.230303
##
## Tuning parameter 'k' was held constant at a value of 8
```

```
#Making predictions of knn Model
raw_knn_predictions <- predict(raw_knn_best_model, newdata = raw_test_data)

# Evaluating the model
raw_knn_confusion_matrix <- table(Actual = raw_test_data$Class, Predicted = raw_knn_predictions)
raw_knn_confusion_matrix
```

```
##       Predicted
## Actual  1  2
##      1  2  1
##      2  3 10
```

```
# Accuracy calculation
raw_knn_accuracy <- sum(diag(raw_knn_confusion_matrix)) / sum(raw_knn_confusion_matrix)
raw_knn_accuracy
```

```
## [1] 0.75
```

Comparing Results of Best KNN Model on Dimensioanlly Reduced vs Raw Dataset:

```
# Calculating precision, recall, and F1-score for KNN model trained on the raw dataset
raw_knn_conf_mat <- raw_knn_confusion_matrix
raw_knn_precision <- raw_knn_conf_mat[2, 2] / sum(raw_knn_conf_mat[, 2])
raw_knn_recall <- raw_knn_conf_mat[2, 2] / sum(raw_knn_conf_mat[2, ])
raw_knn_f1_score <- 2 * (raw_knn_precision * raw_knn_recall) / (raw_knn_precision + raw_knn_reca
ll)

cat("KNN Model on Raw Dataset:\n")
```

```
## KNN Model on Raw Dataset:
```

```
cat("Accuracy:", raw_knn_accuracy, "\n")
```

```
## Accuracy: 0.75
```

```
cat("Precision:", raw_knn_precision, "\n")
```

```
## Precision: 0.9090909
```

```
cat("Recall:", raw_knn_recall, "\n")
```

```
## Recall: 0.7692308
```

```
cat("F1-score:", raw_knn_f1_score, "\n")
```

```
## F1-score: 0.8333333
```

```
# Calculating precision, recall, and F1-score for KNN model trained on the dimensionally reduced dataset
knn_conf_mat <- knn_confusion_matrix
knn_precision <- knn_conf_mat[2, 2] / sum(knn_conf_mat[, 2])
knn_recall <- knn_conf_mat[2, 2] / sum(knn_conf_mat[2, ])
knn_f1_score <- 2 * (knn_precision * knn_recall) / (knn_precision + knn_recall)

cat("KNN Model on Dimensioanlly Reduced Dataset:\n")
```

```
## KNN Model on Dimensioanlly Reduced Dataset:
```

```
cat("Accuracy:", knn_accuracy, "\n")
```

```
## Accuracy: 0.875
```

```
cat("Precision:", knn_precision, "\n")
```

```
## Precision: 1
```

```
cat("Recall:", knn_recall, "\n")
```

```
## Recall: 0.8461538
```

```
cat("F1-score:", knn_f1_score, "\n")
```

```
## F1-score: 0.9166667
```

Plotting and Checking ROC/AUC for Both Models:

```
# Plotting ROC curves for both models

# Calculating predicted probabilities for both models
raw_knn_pred_probs <- predict(raw_knn_best_model, newdata = raw_test_data, type = "prob")[, "1"]
knn_pred_probs <- predict(knn_best_model, newdata = test_data, type = "prob")[, "1"]

# Plotting ROC curves
raw_roc_knn <- roc(raw_test_data$Class, raw_knn_pred_probs)
```

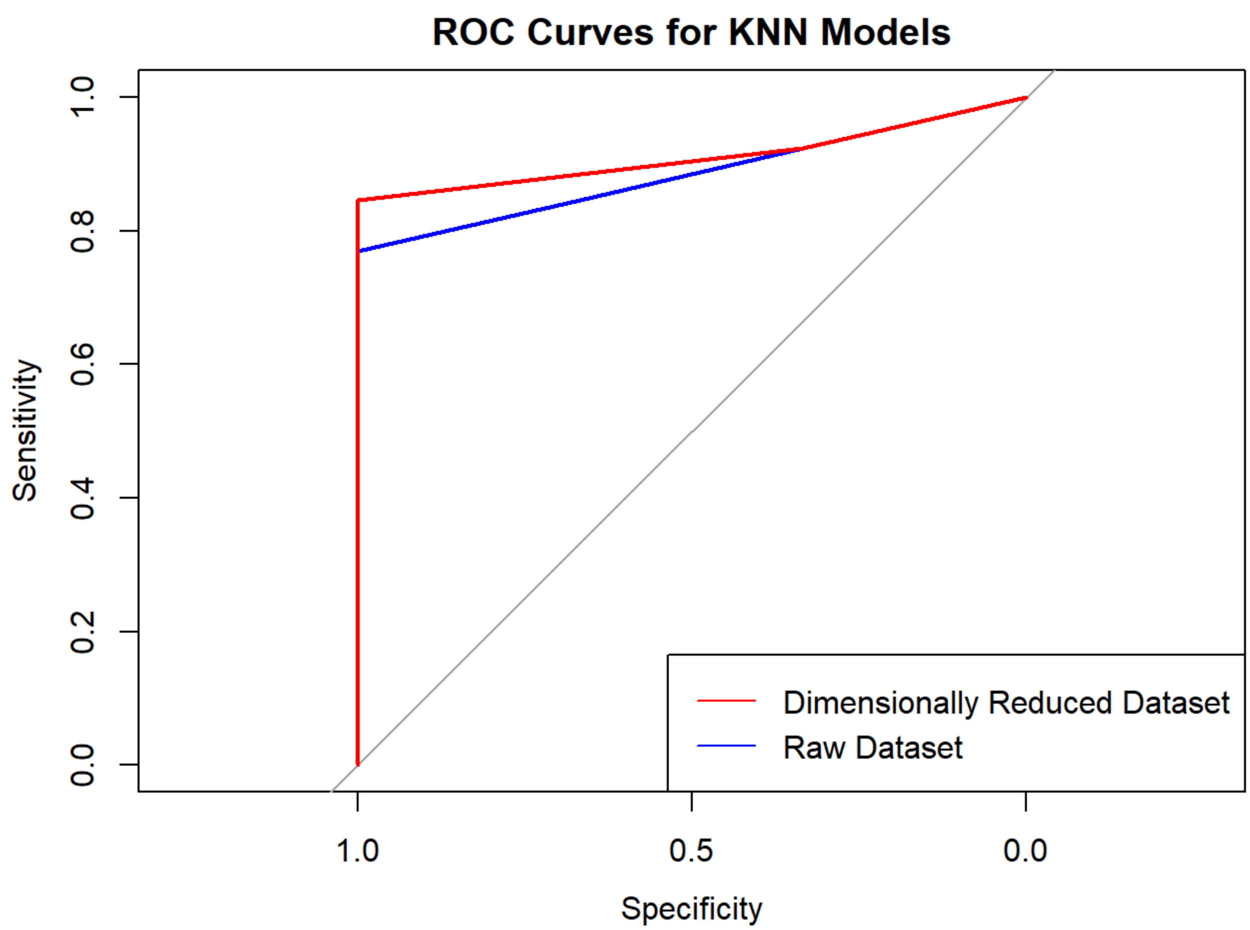
```
## Setting levels: control = 1, case = 2
```

```
## Setting direction: controls > cases
```

```
roc_knn <- roc(test_data$Class, knn_pred_probs)
```

```
## Setting levels: control = 1, case = 2
## Setting direction: controls > cases
```

```
plot(raw_roc_knn, col = "blue", main = "ROC Curves for KNN Models")
plot(roc_knn, col = "red", add = TRUE)
legend("bottomright", legend = c("Dimensionally Reduced Dataset", "Raw Dataset"), col = c("red",
"blue"), lty = 1)
```




```
# Calculating AUC for both models
raw_auc_knn <- auc(raw_roc_knn)
auc_knn <- auc(roc_knn)

cat("\nKNN Model AUC on Raw Dataset:", raw_auc_knn, "\n")
```

```
##
## KNN Model AUC on Raw Dataset: 0.8846154
```

```
cat("KNN Model AUC on Dimensionally Reduced Dataset:", auc_knn, "\n")
```

```
## KNN Model AUC on Dimensionally Reduced Dataset: 0.9102564
```