SDM_Project_1_Code

Load Required Libraries

```r
library(ggplot2)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(cluster)
# install.packages("factoextra")
library(factoextra)
```

```
## Warning: package 'factoextra' was built under R version 4.4.2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(corrplot)
```

```
## Warning: package 'corrplot' was built under R version 4.4.2
```

```
## corrplot 0.95 loaded
```

```r
# install.packages("caret")
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.2
```

```
## Loading required package: lattice
```

```
library(rpart)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

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

```
# install.packages("plotly")
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 4.4.2

##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout
```

Load Data and Preprocess

```
mine_data = read.csv("Mine Dataset/Mine_Dataset_1.csv")
head(mine_data)
```

```
##           V         H S M
## 1 0.3381568 0.0000000 0 1
## 2 0.3202413 0.1818182 0 1
## 3 0.2870087 0.2727273 0 1
## 4 0.2562836 0.4545455 0 1
## 5 0.2628396 0.5454545 0 1
## 6 0.2409665 0.7272727 0 1
```

Check for Data Types, Missing Values and Duplicates

```r
summary(mine_data)
```

```
##        V                H                S                M        
##  Min.   :0.1977   Min.   :0.0000   Min.   :0.0000   Min.   :1.000  
##  1st Qu.:0.3097   1st Qu.:0.2727   1st Qu.:0.2000   1st Qu.:2.000  
##  Median :0.3595   Median :0.5455   Median :0.6000   Median :3.000  
##  Mean   :0.4306   Mean   :0.5089   Mean   :0.5036   Mean   :2.953  
##  3rd Qu.:0.4826   3rd Qu.:0.7273   3rd Qu.:0.8000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :5.000
```

```r
str(mine_data)
```

```
## 'data.frame':    338 obs. of  4 variables:
##  $ V: num  0.338 0.32 0.287 0.256 0.263 ...
##  $ H: num  0 0.182 0.273 0.455 0.545 ...
##  $ S: num  0 0 0 0 0 0 0 0 0.6 0.6 ...
##  $ M: int  1 1 1 1 1 1 1 1 1 1 ...
```

Checking for Missing values

```r
any_missing = any(is.na(mine_data))
print(paste("Are there any missing values?", any_missing))
```

```
## [1] "Are there any missing values? FALSE"
```

Checking for duplicate rows

```r
duplicate_rows = duplicated(mine_data)
print(paste("Number of duplicate rows:", sum(duplicate_rows)))
```

```
## [1] "Number of duplicate rows: 0"
```

```r
mean_value = sd(mine_data$V)
print(mean_value)
```

```
## [1] 0.195819
```

Performed Scaling after EDA

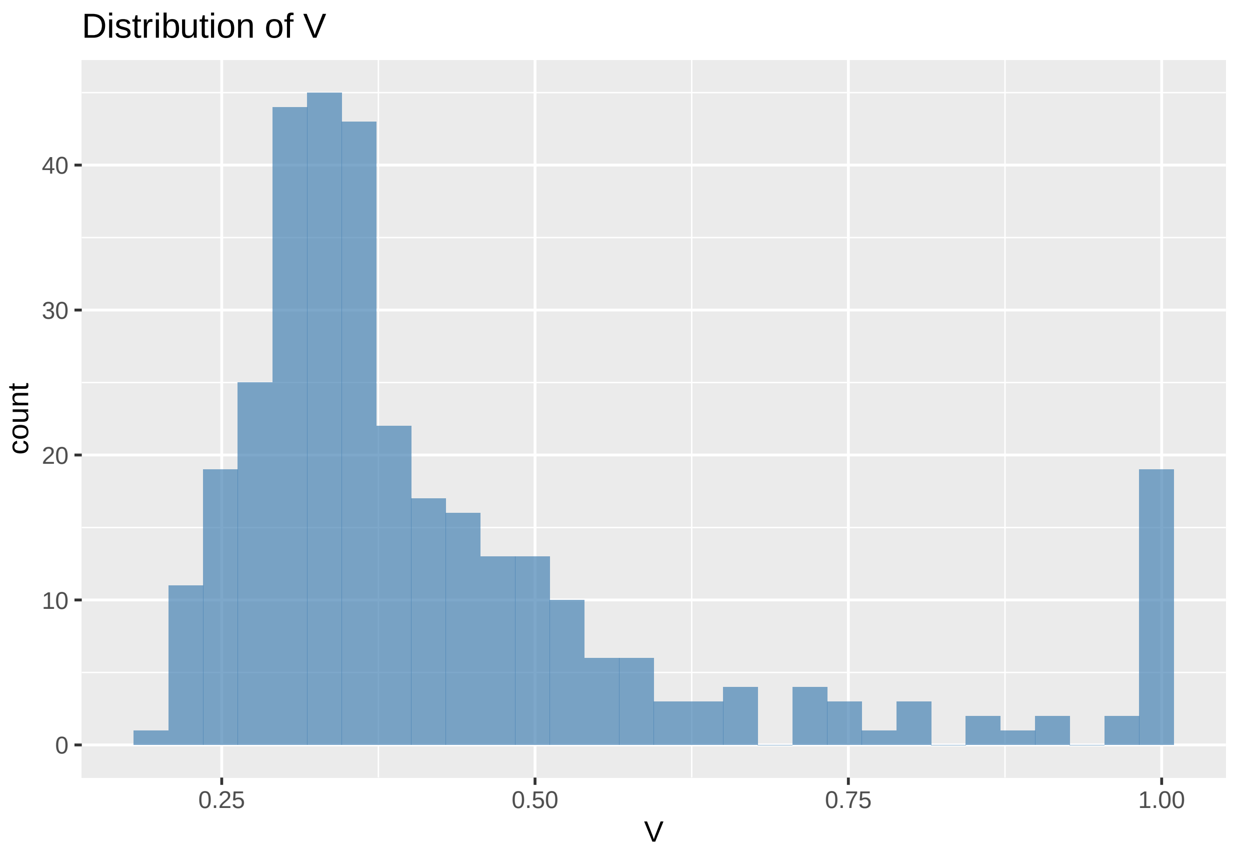
Notes :

0.0: 'Dry+sandy', 0.2: 'Dry+humus', 0.4: 'Dry+limey', 0.6: 'Humid+sandy', 0.8: 'Humid+humus', 1.0: 'Humid+limey

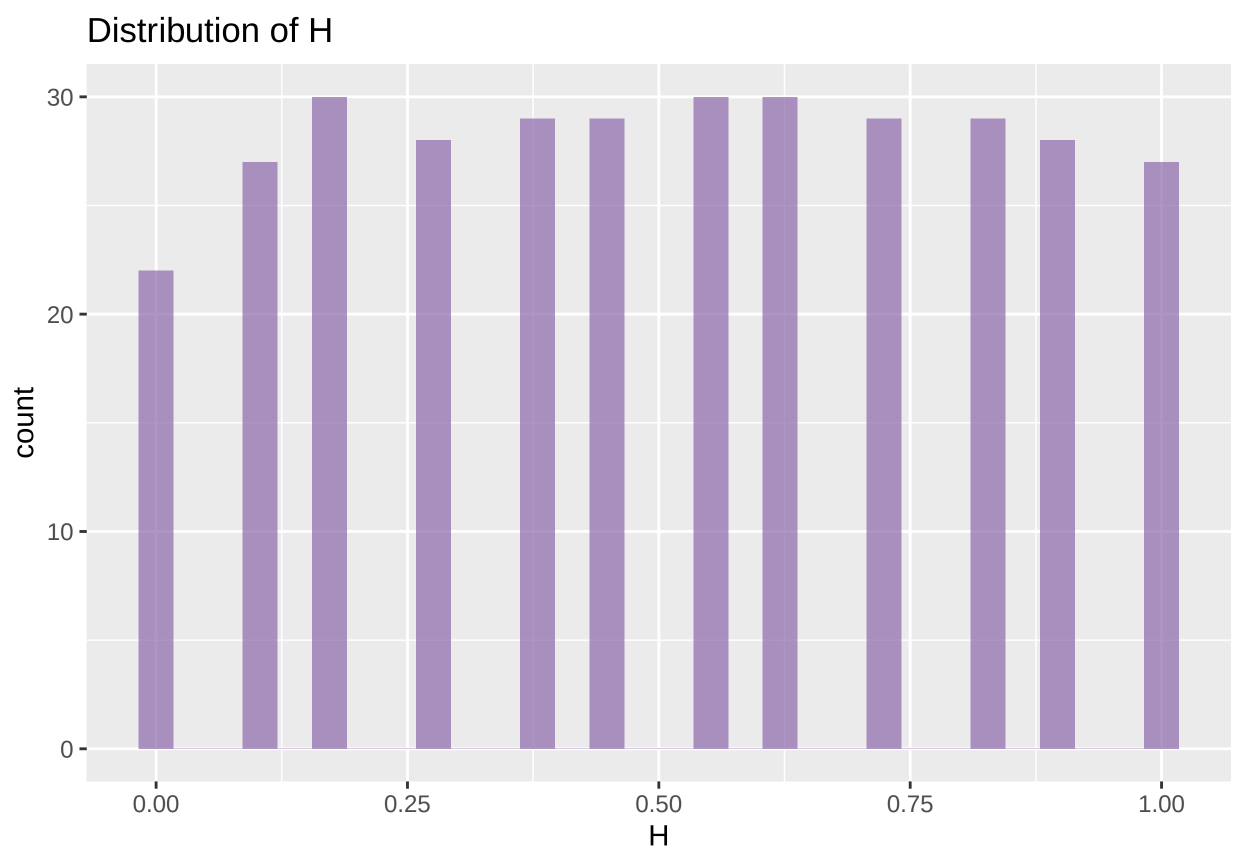
Exploratory Data Analysis

Distrubution plots

```
ggplot(mine_data, aes(x = V)) + geom_histogram(bins = 30, fill = "steelblue", alpha = 0.7) + ggtitle("Di
```

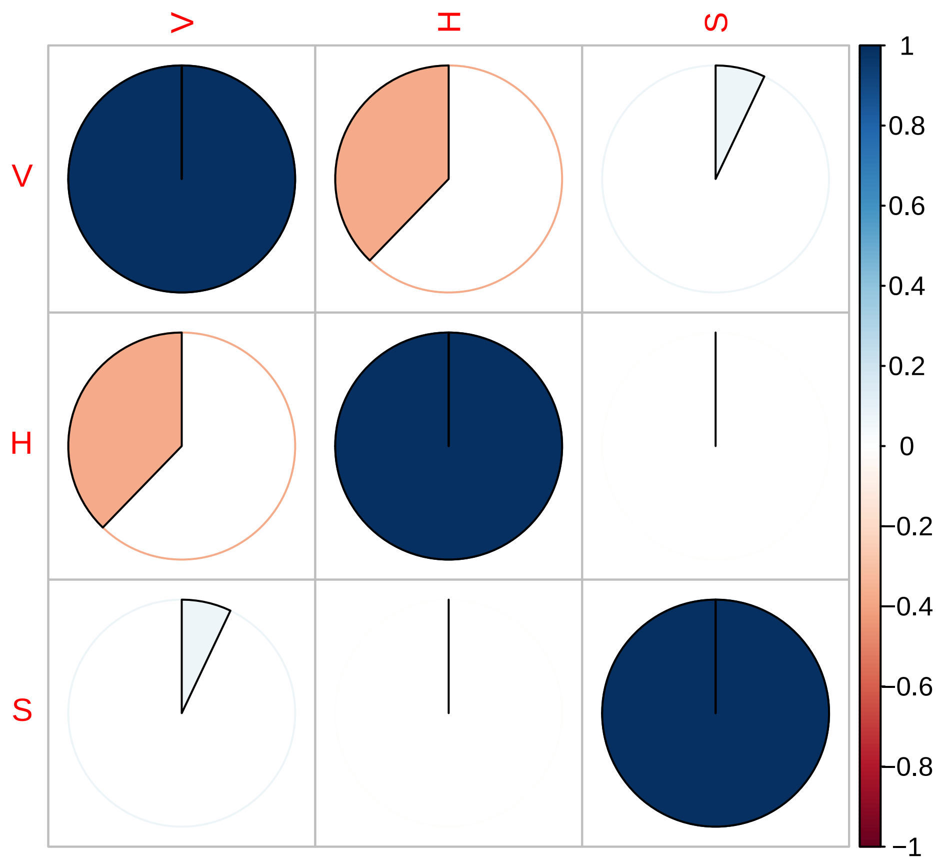


```
ggplot(mine_data, aes(x = H)) + geom_histogram(bins = 30, fill = "#8d67aa", alpha = 0.7) + ggtitle("Dis
```



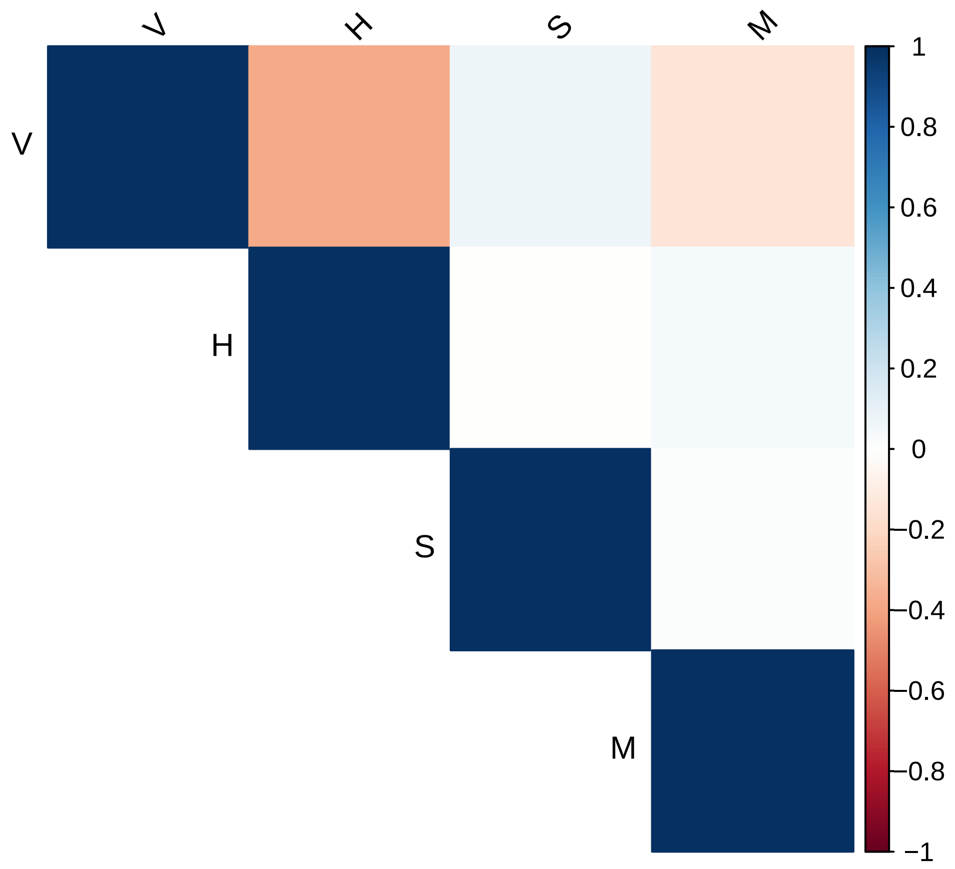
Correlation Plots

```r
correlation_matrix = cor(mine_data[, c("V", "H", "S")])
corrplot(correlation_matrix, method = "pie")
```



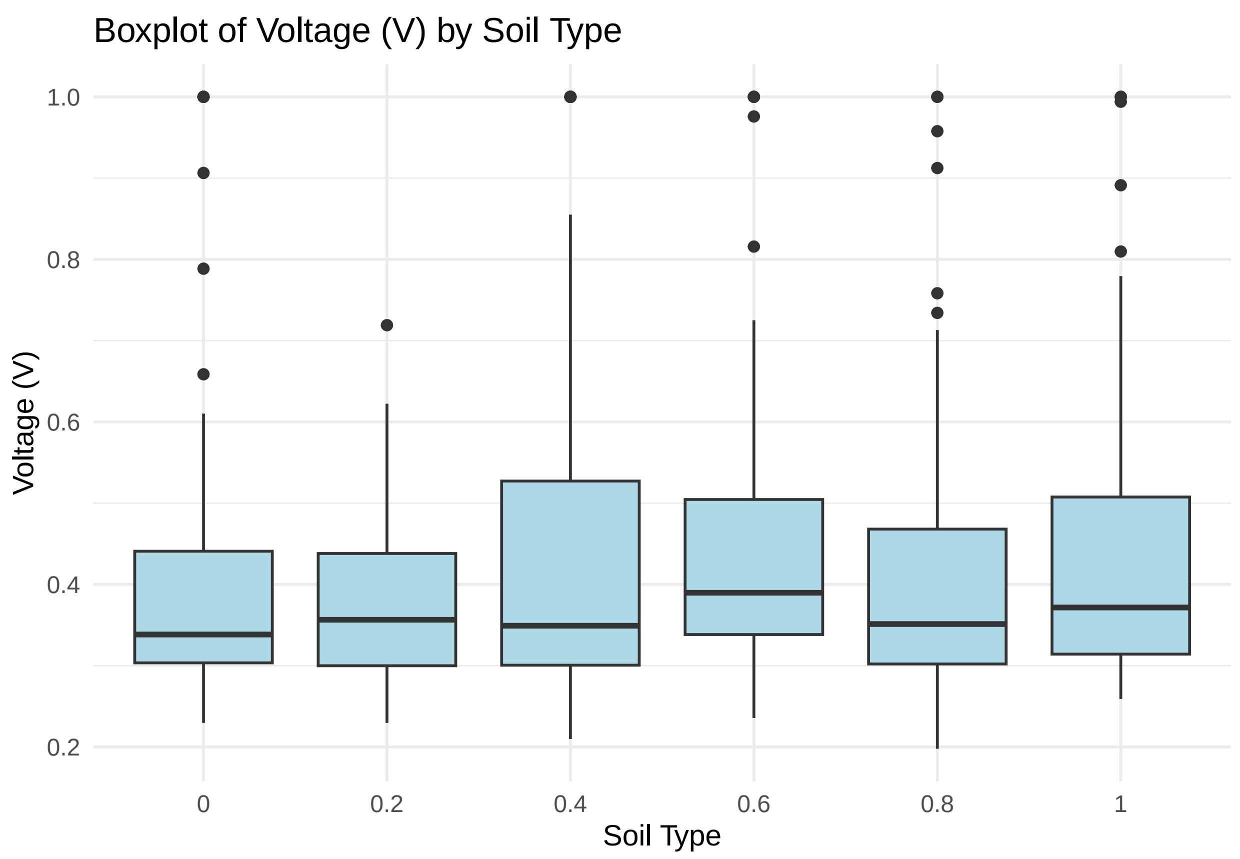
Correlation matrix including Voltage (V) and Height (H)

```r
correlation_matrix = cor(mine_data[, sapply(mine_data, is.numeric)])
corrplot(correlation_matrix, method = "color", type = "upper", tl.col = "black", tl.srt = 45)
```



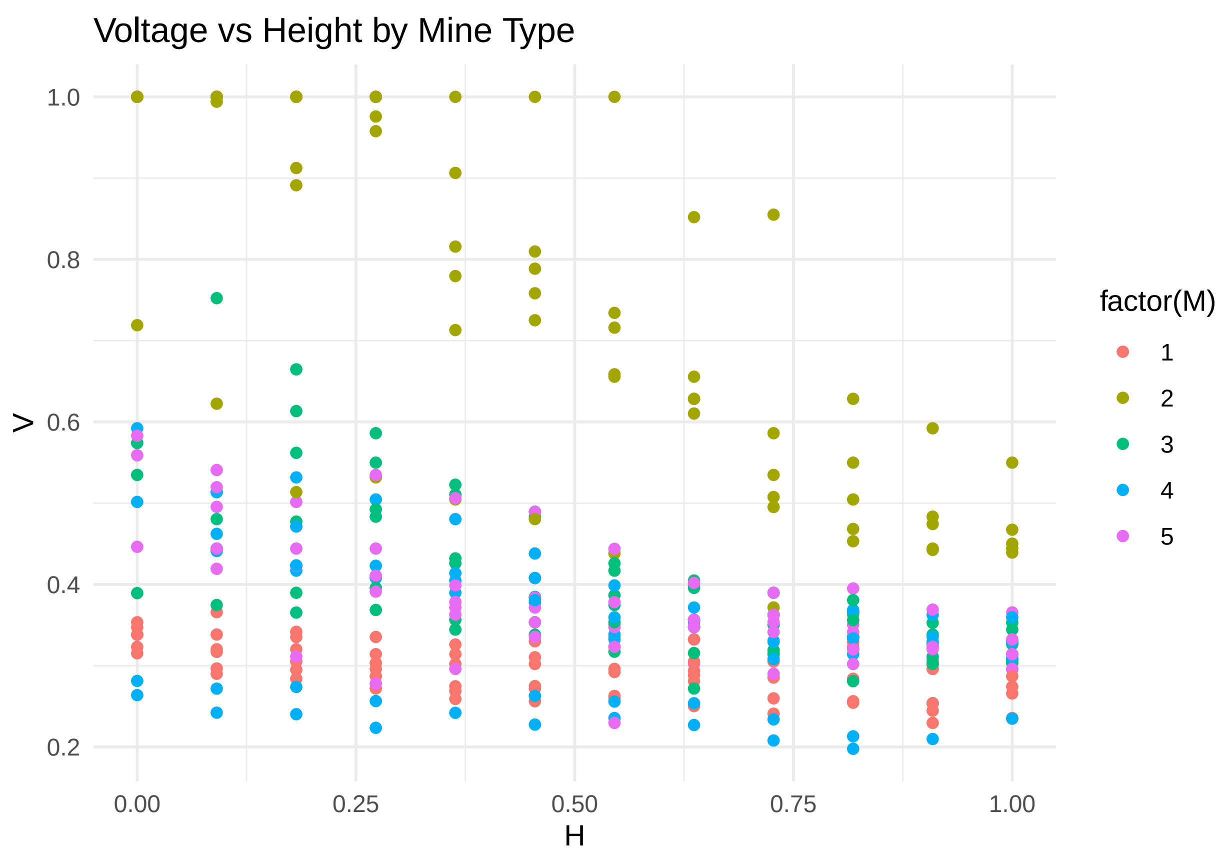
Box plot of Voltage by Soil Type

```r
ggplot(mine_data, aes(x = factor(S), y = V)) +
  geom_boxplot(fill = "lightblue") +
  theme_minimal() +
  labs(title = "Boxplot of Voltage (V) by Soil Type", x = "Soil Type", y = "Voltage (V)")
```



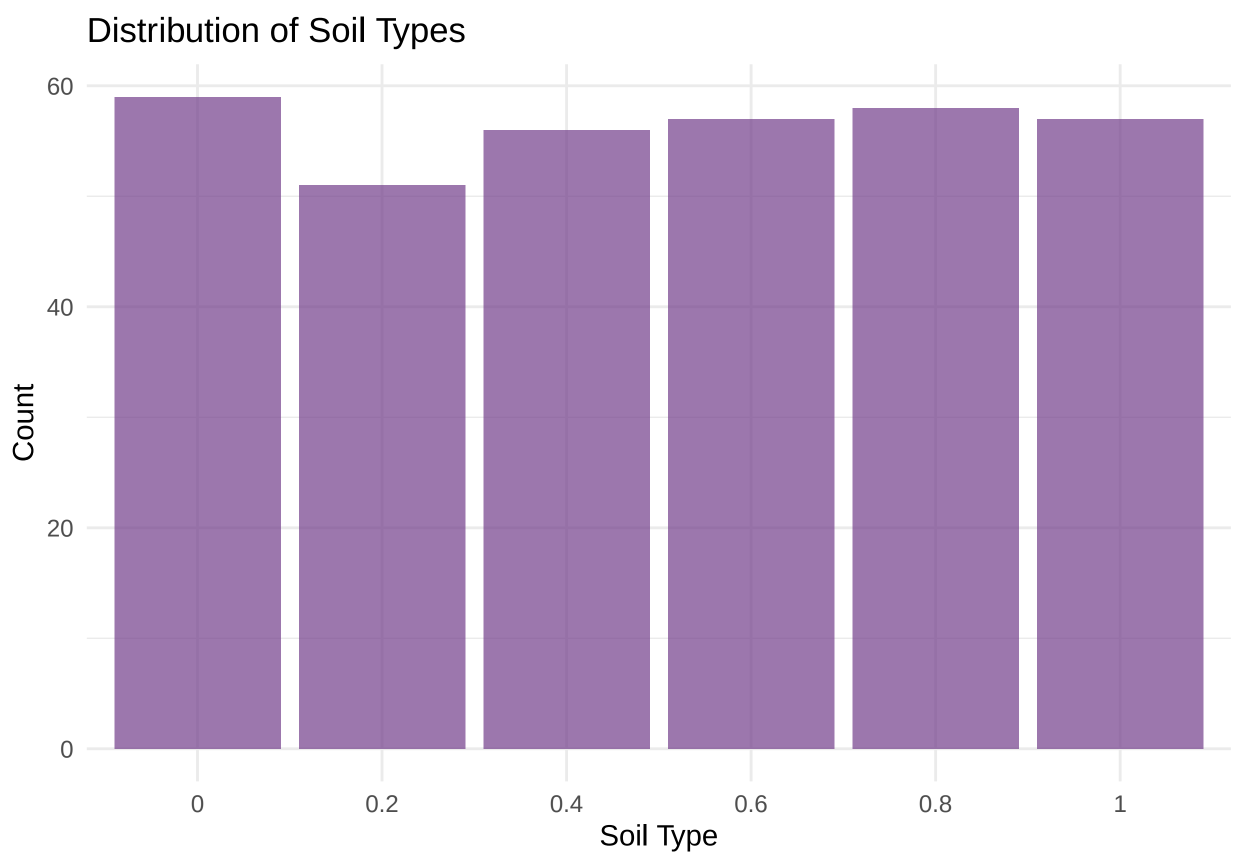
Scatter plot of Voltage vs Height, colored by Mine Type

```r
ggplot(mine_data, aes(x = `H`, y = `V`, color = factor(M))) +
  geom_point() +
  theme_minimal() +
  labs(title = "Voltage vs Height by Mine Type")
```

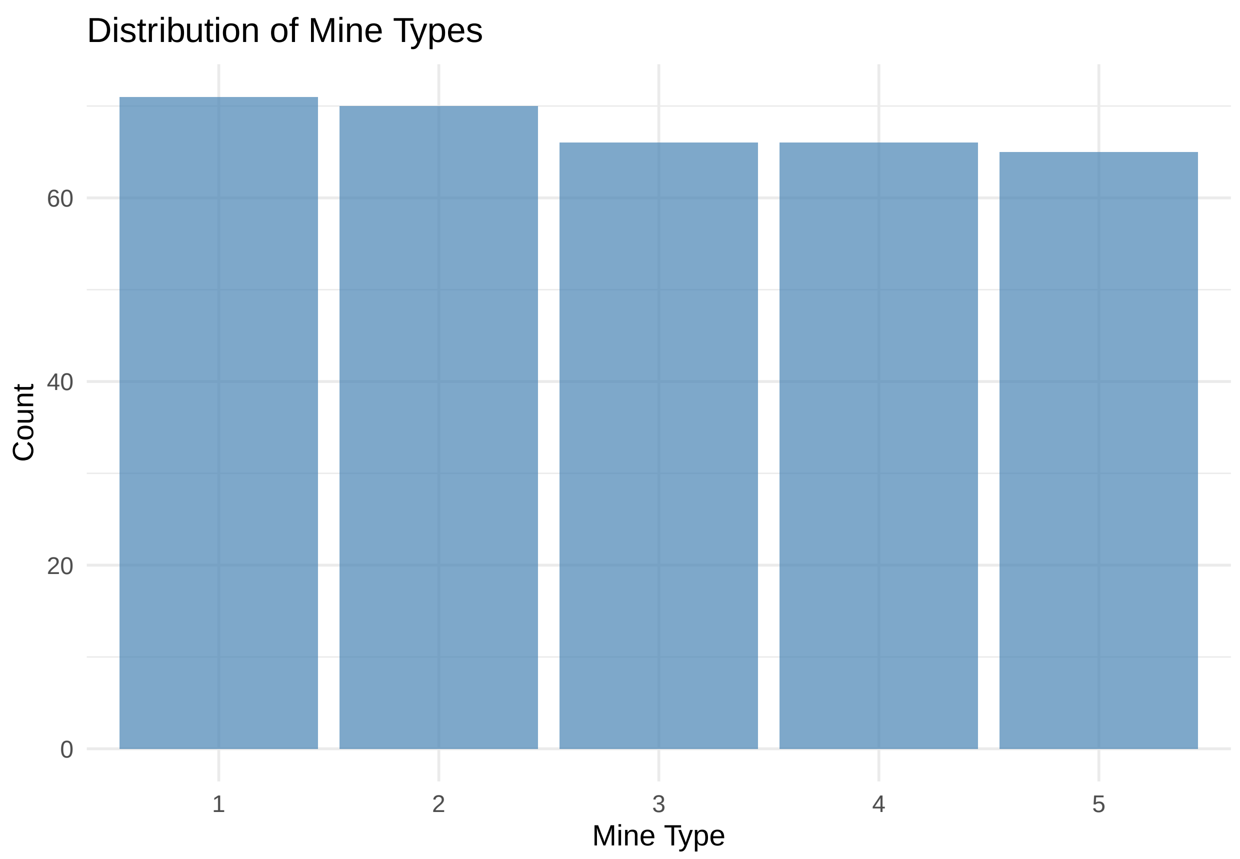


Bar Plot for Categorical Variable Distribution

```r
# Bar plot for Soil Type distribution
ggplot(mine_data, aes(x = factor(S))) +
  geom_bar(fill = "#713c8a", alpha = 0.7) +
  theme_minimal() +
  labs(title = "Distribution of Soil Types", x = "Soil Type", y = "Count")
```



```r
# Bar plot for Mine Type distribution
ggplot(mine_data, aes(x = factor(M))) +
  geom_bar(fill = "steelblue", alpha = 0.7) +
  theme_minimal() +
  labs(title = "Distribution of Mine Types", x = "Mine Type", y = "Count")
```



Scaling the data

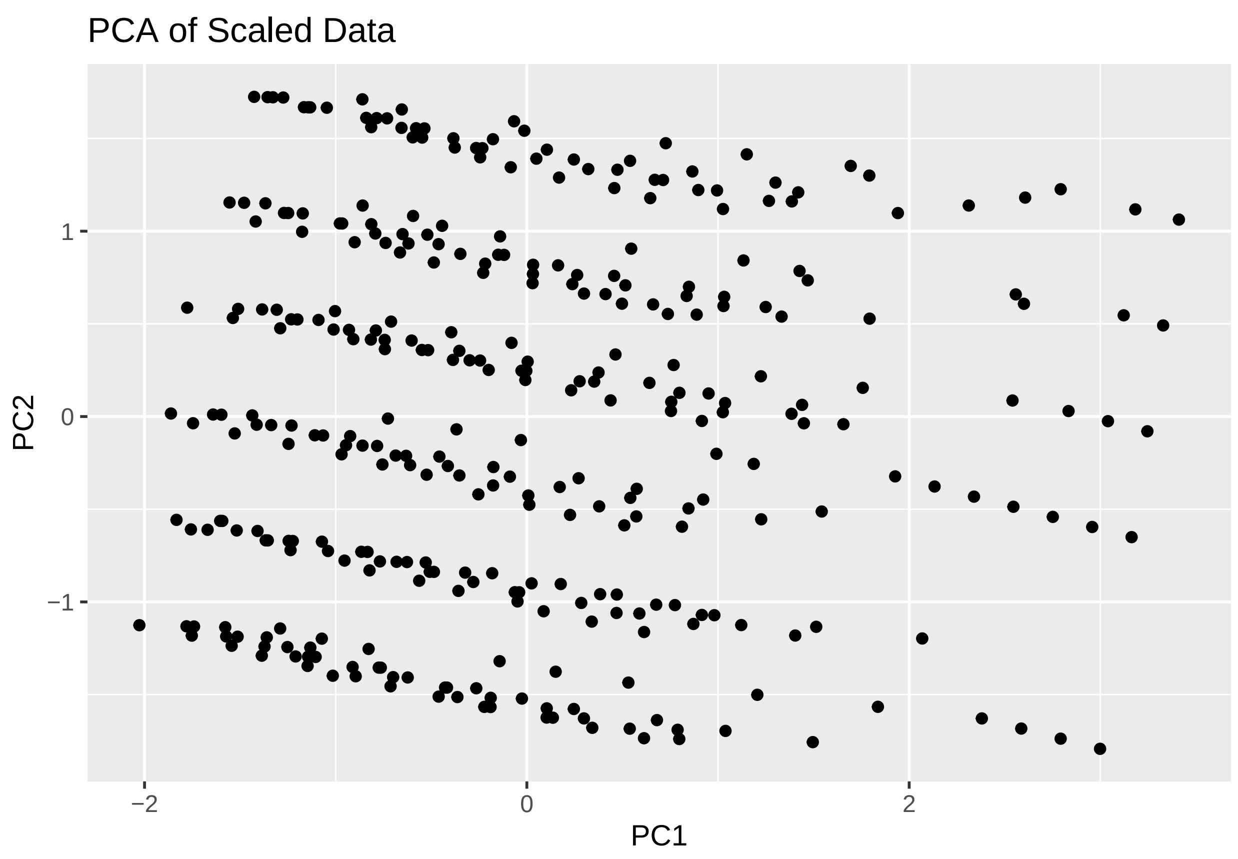
```
mine_data_cleaned = mine_data[, c("V", "H", "S")]
mine_data_scaled = scale(mine_data_cleaned)
```

Perform PCA

```
pca_result = prcomp(mine_data_scaled, center = TRUE, scale. = TRUE)
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2    PC3
## Standard deviation     1.1770 0.9987 0.7856
## Proportion of Variance 0.4618 0.3325 0.2057
## Cumulative Proportion  0.4618 0.7943 1.0000
```

```
pca_data <- pca_result$x[, 1:2]
ggplot(as.data.frame(pca_data), aes(x = PC1, y = PC2)) +
  geom_point() +
  labs(title = "PCA of Scaled Data")
```



```r
pca_data_3D <- pca_result$x[, 1:3]
plot_ly(x = pca_data_3D[,1], y = pca_data_3D[,2], z = pca_data_3D[,3],
        type = 'scatter3d', mode = 'markers') %>%
  layout(title = "3D PCA of the Data",
         scene = list(
           xaxis = list(title = 'PC1'),
           yaxis = list(title = 'PC2'),
           zaxis = list(title = 'PC3')
         ))
```

Proportion Variance Explained

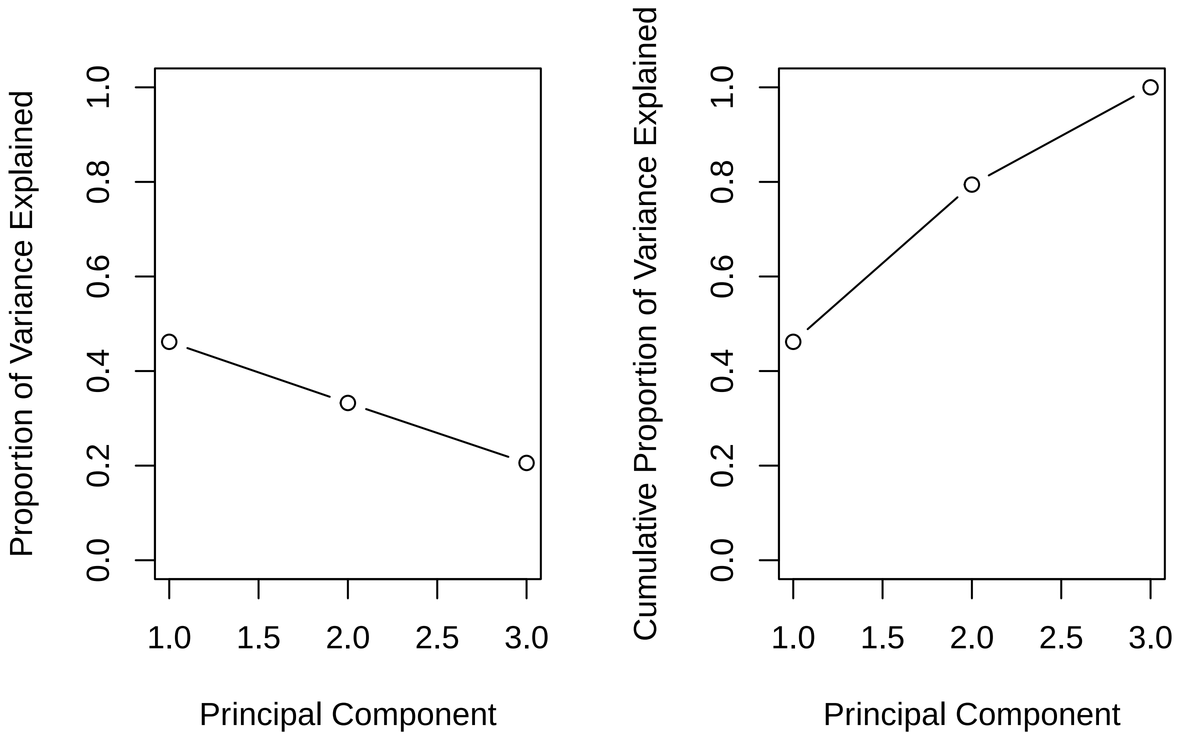
```r
pr.out = pca_result
pr.var <- pr.out$sdev^2
PVE <- pr.var/sum(pr.var)
PVE
```

```
## [1] 0.4617988 0.3324947 0.2057065
```

```r
par ( mfrow = c (1 , 2) )
plot ( PVE , xlab = "Principal Component" ,
 ylab = "Proportion of Variance Explained" , ylim = c (0 , 1) ,
 type = "b" )
 3
```

```
## [1] 3
```

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



Clustering

1. K-means clustering

Intro:

Kmeans algorithm (also referred as Lloyd's algorithm) is the most commonly used unsupervised machine learning algorithm used to partition the data into a set of k groups or clusters.

How Kmeans works? 1. Define the number of clusters (k) 2. Initialize k centroids by randomly. 3. Assignment Step: Assign each observation to the closest centroid (center-point) by calculting least squared euclidean distance between centroids and observations. (i.e. least squared euclidean distance between assigned center and observation should be minimum than other centers). 4. Update Step: Calculate the new means as centroids for new clusters. 5. Repeat both assignment and update step (i.e. steps 3 & 4) until convergence (minimum total sum of square) or maximum iteration is reached.

Determining optimal number of clusters (k)

Before we do the actual clustering, we need to identity the Optimal number of clusters (k) for this data set of wholesale customers. The popular way of determining number of clusters are

1. Elbow Method
2. Gap Statistics Method
3. Silhouette Method

Elbow and Silhouette methods are direct methods and gap statistic method is the statistics method.

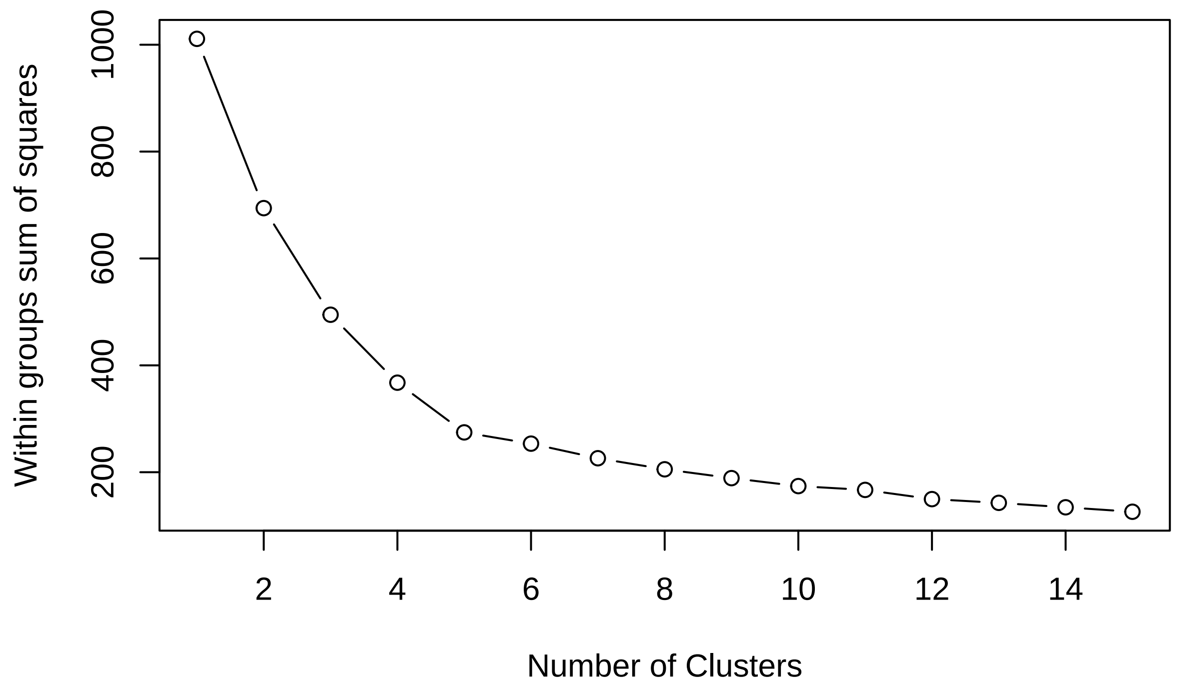
Notes : K-means clustering aims to minimize the Within-Cluster Sum of Squares (WCSS), which measures how tightly the data points are grouped in each cluster. It is also known as the inertia.

As the number of clusters (k) increases, the WCSS generally decreases because more clusters mean that each cluster will have fewer points, resulting in a better fit. However, adding too many clusters can lead to overfitting and make the clustering meaningless.

How to choose optimal number of clusters ### 1. Elbow Method

```r
#Elbow Method for finding the optimal number of clusters
wssplot <- function(data, nc=15, seed=1234){
  wss = (nrow(data)-1)*sum(apply(data,2,var)) # calculating variance as the data set and storing the va
#   print(wss)
  for (i in 2:nc)
  {
    set.seed(seed)
    wss[i] = sum(kmeans(data, centers=i)$withinss)
  }
#   print(wss)
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")
}
```

```r
wssplot(mine_data_scaled)
```



The best value to choose as the optimal number of clusters would likely be 5 based on the point where the WSS starts flattening. However, it is always good to visually inspect the plot to confirm this behavior.

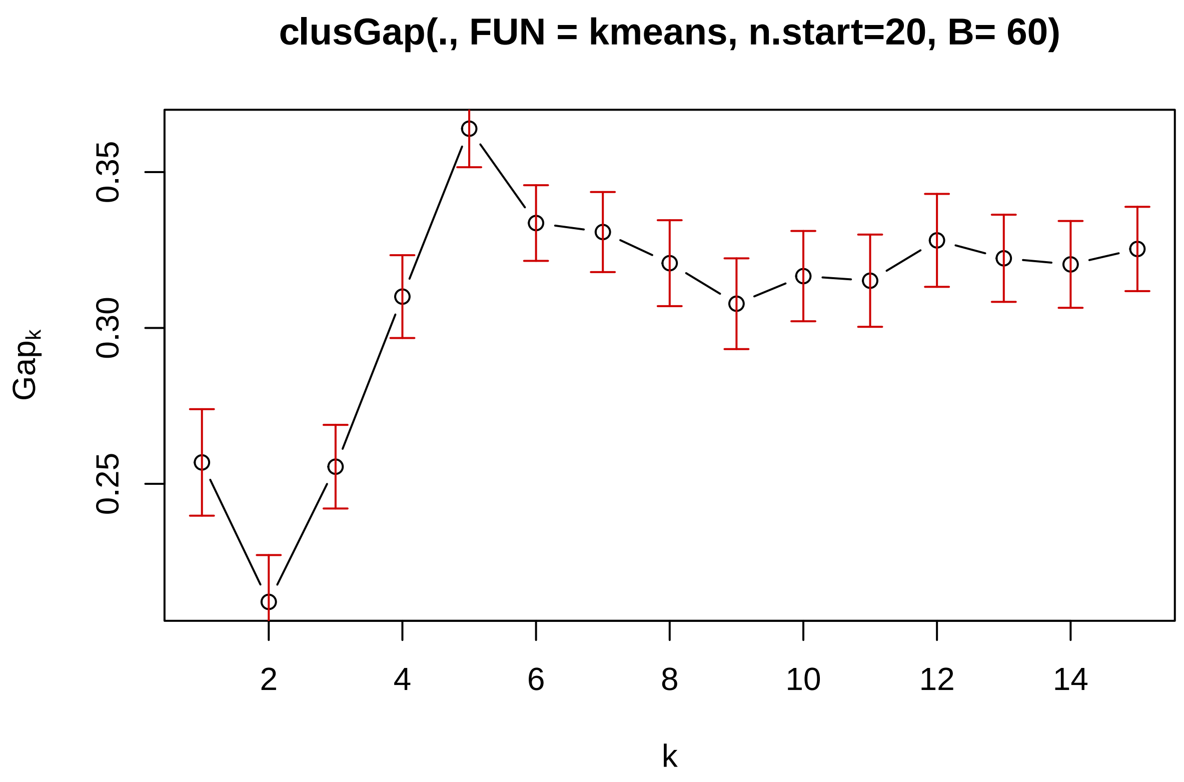
Applying Gap Statistics

```
set.seed(1234)  # Set a seed for reproducibility
gap_stat = clusGap(mine_data_scaled,
                    FUN = function(x, k) kmeans(x, centers = k, nstart = 25),
                    K.max = 15,
                    B = 50)
```

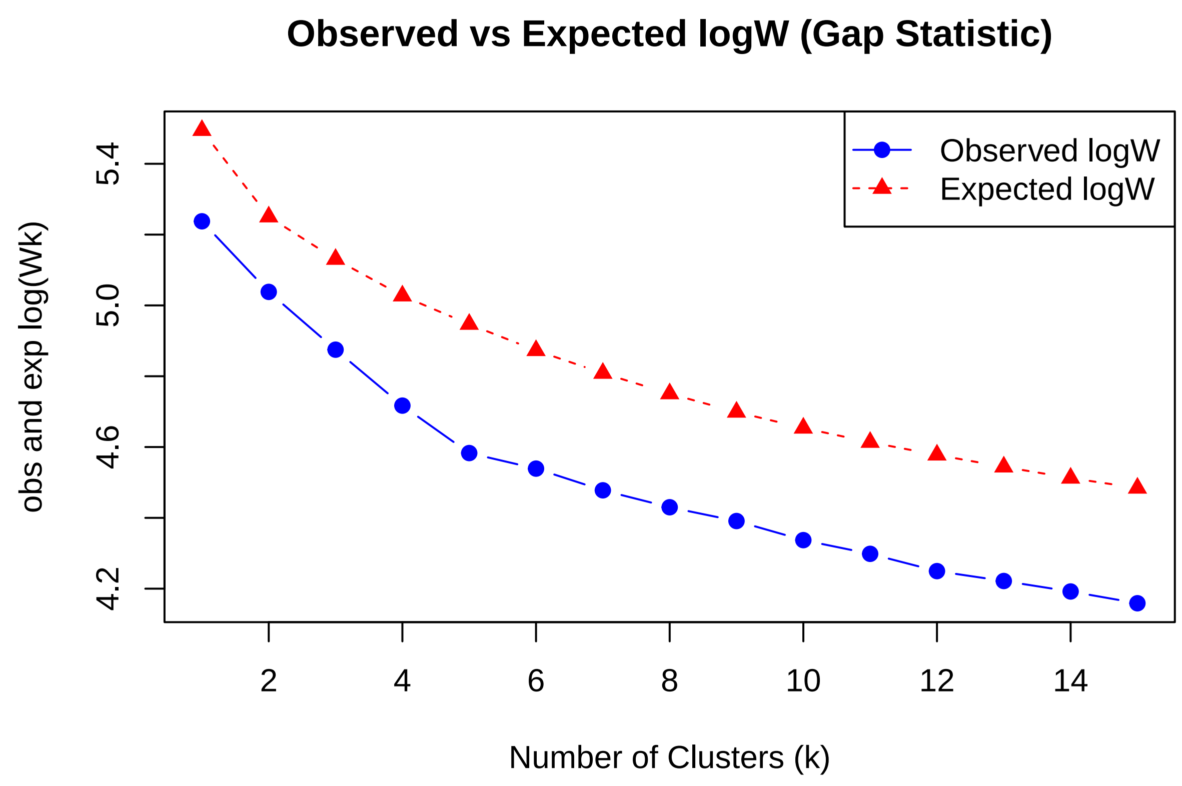
```
gap_stat
```

```
## Clustering Gap statistic ["clusGap"] from call:
## clusGap(x = mine_data_scaled, FUNcluster = function(x, k) kmeans(x, centers = k, nstart = 25), K.max
## B=50 simulated reference sets, k = 1..15; spaceH0="scaledPCA"
##  --> Number of clusters (method 'firstSEmax', SE.factor=1): 1
##          logW   E.logW       gap     SE.sim
##  [1,] 5.237558 5.494425 0.2568671 0.01708755
##  [2,] 5.038168 5.250297 0.2121289 0.01501969
##  [3,] 4.874983 5.130477 0.2554942 0.01341194
##  [4,] 4.716993 5.027047 0.3100548 0.01328527
##  [5,] 4.582979 4.946896 0.3639167 0.01235562
##  [6,] 4.539102 4.872755 0.3336525 0.01214061
##  [7,] 4.477782 4.808548 0.3307658 0.01284128
##  [8,] 4.429964 4.750745 0.3207806 0.01377956
##  [9,] 4.390981 4.698755 0.3077742 0.01455806
## [10,] 4.337072 4.653699 0.3166262 0.01448169
## [11,] 4.298412 4.613571 0.3151594 0.01479343
## [12,] 4.249604 4.577698 0.3280935 0.01490528
## [13,] 4.221656 4.544004 0.3223486 0.01397857
## [14,] 4.192041 4.512428 0.3203872 0.01392598
## [15,] 4.158849 4.484187 0.3253377 0.01351592
```

```
plot(gap_stat, main = "clusGap(., FUN = kmeans, n.start=20, B= 60)")
```



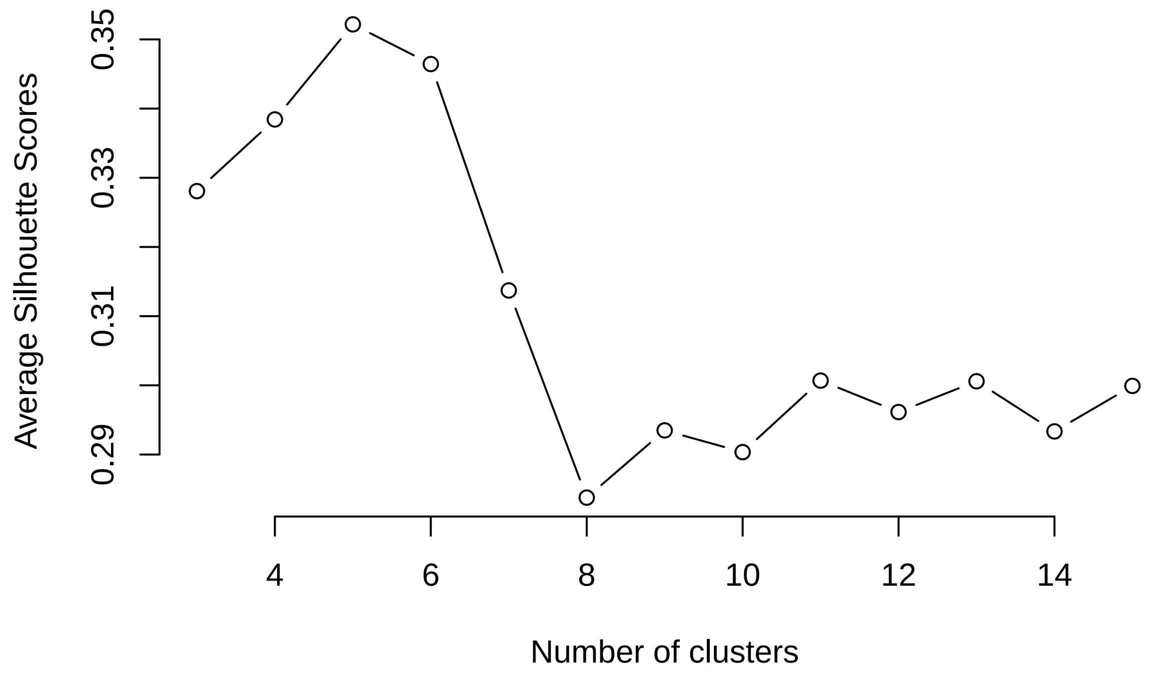
```
k_values <- 1:15
observed_logW <- gap_stat$Tab[, "logW"]
expected_logW <- gap_stat$Tab[, "E.logW"]

plot(k_values, observed_logW, type = "b", col = "blue", pch = 19,
     xlab = "Number of Clusters (k)", ylab = "obs and exp log(Wk)",
     main = "Observed vs Expected logW (Gap Statistic)",
     ylim = range(c(observed_logW, expected_logW)))

lines(k_values, expected_logW, type = "b", col = "red", pch = 17, lty = 2)

legend("topright", legend = c("Observed logW", "Expected logW"),
       col = c("blue", "red"), pch = c(19, 17), lty = c(1, 2))
```



The Gap Statistic plot identifies the optimal number of clusters by comparing the logarithm of within-cluster dispersion with its expected value under a null reference distribution. A prominent peak at k=5 indicates this as a potential optimal cluster count, with increasing error bars (standard deviation) for higher reflecting greater uncertainty in clustering outcomes.

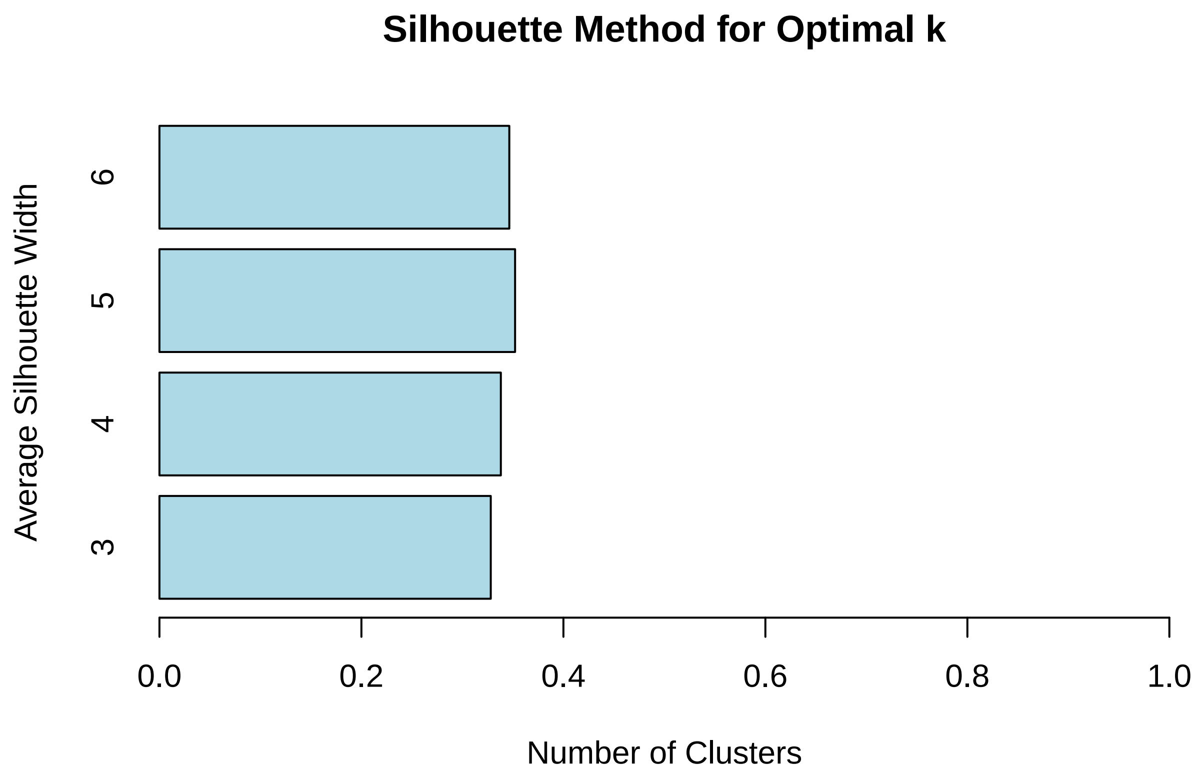
Silhouette Method

```
silhouette_score <- function(k){
  km <- kmeans(mine_data_scaled, centers = k, nstart=25)
  ss <- silhouette(km$cluster, dist(mine_data_scaled))
  mean(ss[, 3])
}
k <- 3:15
avg_sil <- sapply(k, silhouette_score)
plot(k, type='b', avg_sil, xlab='Number of clusters', ylab='Average Silhouette Scores', frame=FALSE)
```



Plotting bar chart for silhouette scores

```r
k_values <- c(3, 4, 5, 6)
silhouette_scores <- list()

for (k in k_values) {
  kmeans_result = kmeans(mine_data_scaled, centers = k, nstart = 25)
  sil = silhouette(kmeans_result$cluster, dist(mine_data_scaled))
  silhouette_scores[[as.character(k)]] = mean(sil[, 3])  # Store average silhouette width
}

barplot(unlist(silhouette_scores), names.arg = k_values,
        xlab = "Number of Clusters", ylab = "Average Silhouette Width",
        main = "Silhouette Method for Optimal k",
        col = "lightblue", horiz = TRUE, xlim = c(0, 1))
```



From Class Notes : Choose k with larger width and no negative items (are they outliers?) k with larger width is 5

The silhouette analysis evaluates cluster separation for k ranging from 3 to 15, with the highest score (~0.055) at k=5, indicating the optimal cluster count. The scores decline sharply after k=5 and stabilize at lower values, reflecting weak overall clustering structure. The code performs k-means clustering, computes silhouette widths, and plots average scores, suggesting k=5 as the best balance of separation and simplicity.

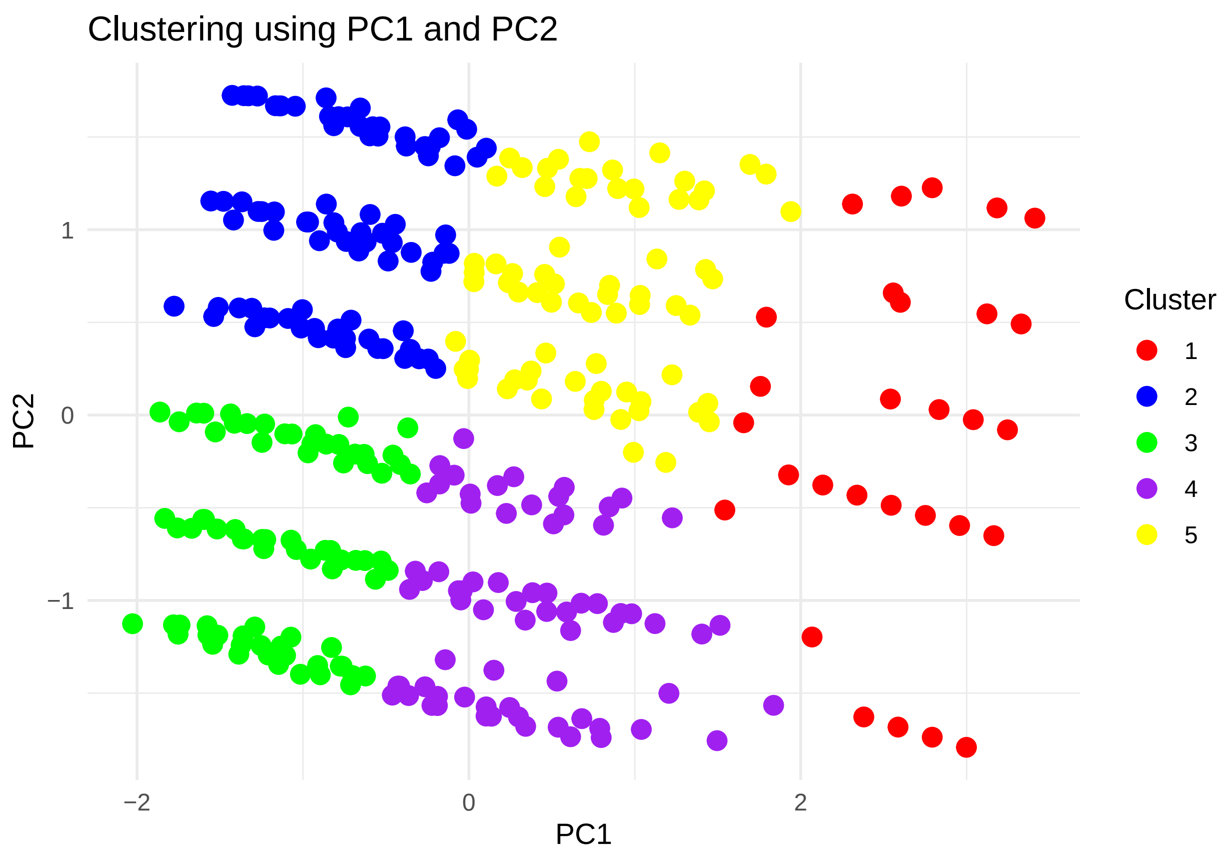
Hence choosing K = 5 as the optimal number of clusters from above three methods

Applying K-Means clustering for PCA Data

```r
pca_result <- prcomp(mine_data_scaled, center = TRUE, scale. = TRUE)

pca_data_2PC <- data.frame(pca_result$x[, 1:2])  # Using PC1 and PC2
pca_data_3PC <- data.frame(pca_result$x[, 1:3])  # Using PC1, PC2, and PC3

set.seed(1234)
km_2PC <- kmeans(pca_data_2PC, centers = 5, nstart = 25)  # Apply K-means to 2 PCs
pca_data_2PC$Cluster <- as.factor(km_2PC$cluster)

set.seed(1234)
km_3PC <- kmeans(pca_data_3PC, centers = 5, nstart = 25)  # Apply K-means to 3 PCs
pca_data_3PC$Cluster <- as.factor(km_3PC$cluster)
```

Visualization for 2 PCs

```r
ggplot(pca_data_2PC, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point(size = 3) +
  labs(title = "Clustering using PC1 and PC2", x = "PC1", y = "PC2") +
  theme_minimal() +
  scale_color_manual(values = c("red", "blue", "green", "purple", "yellow"))
```



The code performs K-means clustering on PCA-transformed data, using 5 clusters and the first two principal components. The data preparation involves scaling the original data and applying PCA, followed by K-means clustering with 5 centers and 25 random starts. The visualization using ggplot2 shows clear cluster separation in the PC1 vs PC2 space, with each cluster represented by a different color (red, blue, green, purple, and yellow).

Visualization for 3 PCs

```r
library(plotly)
plot_ly(x = pca_data_3PC$PC1, y = pca_data_3PC$PC2, z = pca_data_3PC$PC3,
        color = pca_data_3PC$Cluster, colors = c("red", "blue", "green", "purple", "yellow"),
        type = 'scatter3d', mode = 'markers') %>%
  layout(title = "3D Clustering using PC1, PC2, and PC3",
         scene = list(xaxis = list(title = 'PC1'),
                      yaxis = list(title = 'PC2'),
                      zaxis = list(title = 'PC3')))
```

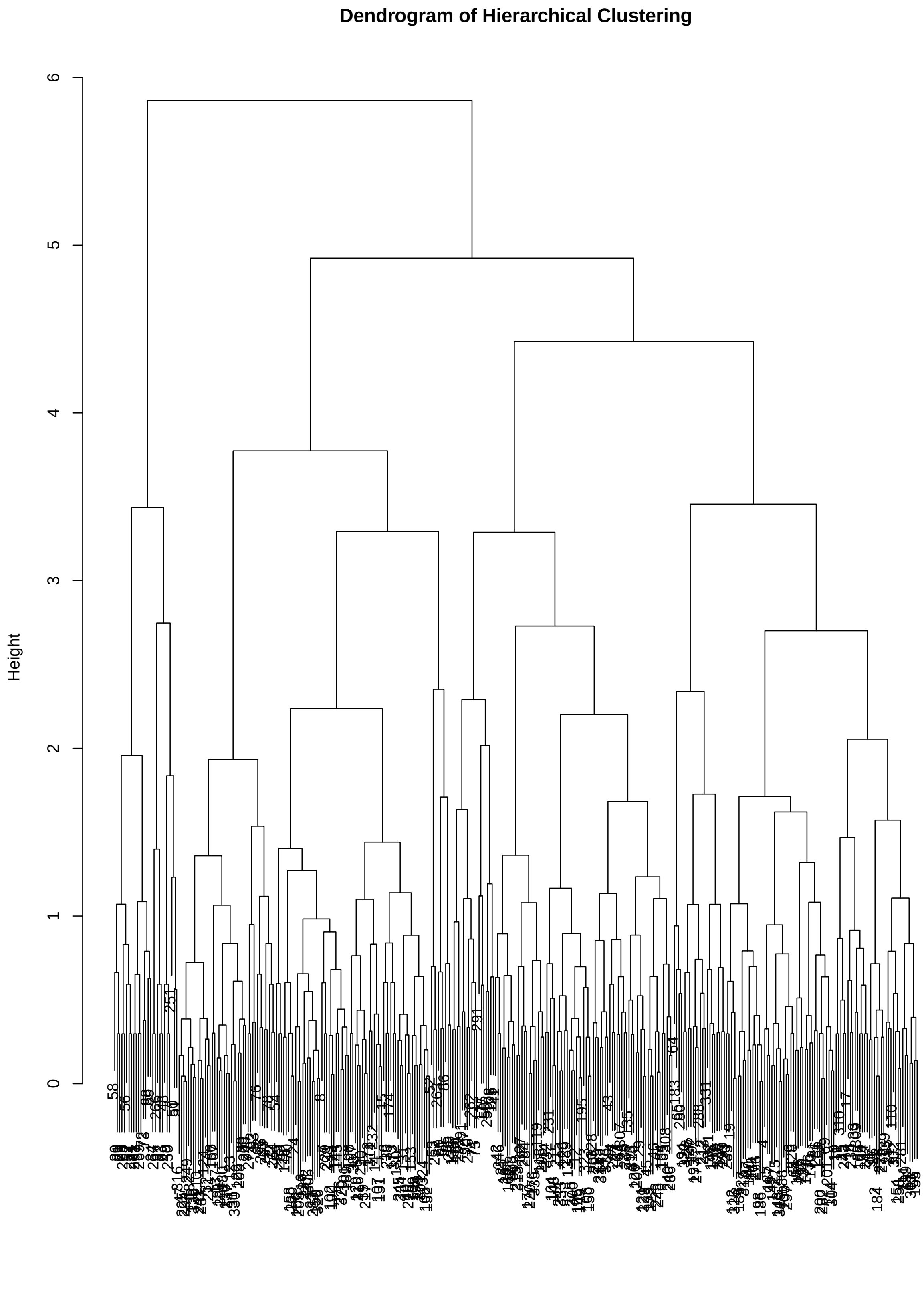
This enhanced version adds black diamond markers to show the cluster centroids, making it easier to visualize the center of each cluster in the 3D space. The interactive plotly visualization allows you to rotate, zoom, and hover over points for more detailed information.

Hierarchical Clustering

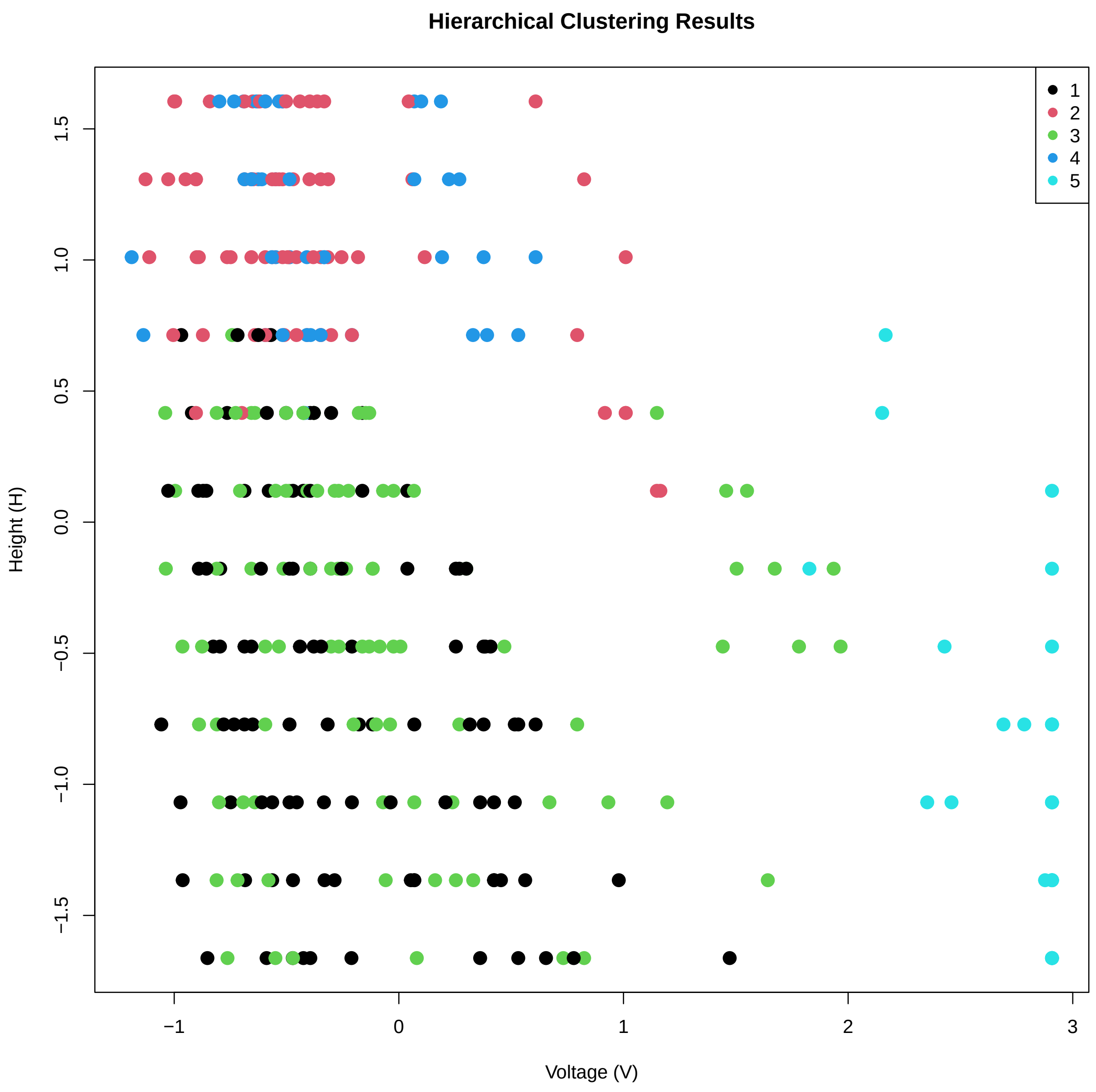
```
mine_data_scaled_HC = mine_data_scaled
head(mine_data_scaled_HC)
```

```
##               V          H         S
## [1,] -0.4722601 -1.6627568 -1.462773
## [2,] -0.5637498 -1.0686640 -1.462773
## [3,] -0.7334605 -0.7716176 -1.462773
## [4,] -0.8903662 -0.1775248 -1.462773
## [5,] -0.8568864  0.1195216 -1.462773
## [6,] -0.9685872  0.7136145 -1.462773
```

```
mine_data_scaled_HC <- as.data.frame(mine_data_scaled_HC)
```

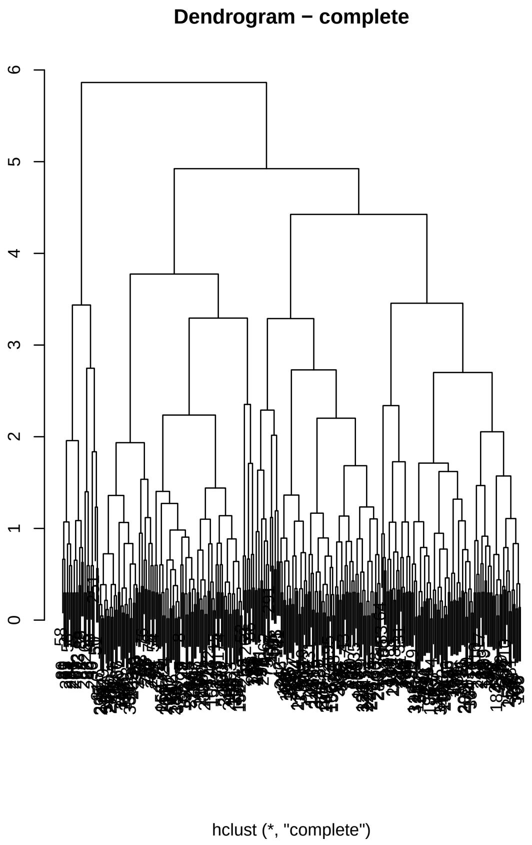

Dendrogram of Hierarchical Clustering
Height
0
1
2
3
4
5
6

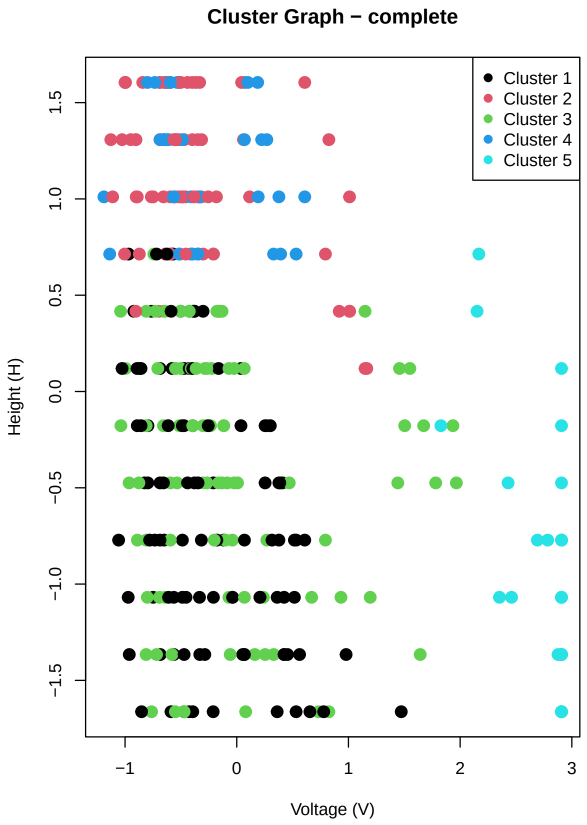
The dendrogram visualization reveals the hierarchical relationships between clusters, where the height of each branch represents the distance (dissimilarity) between merged clusters. The complete linkage method used here considers the maximum distance between points in different clusters when merging them.

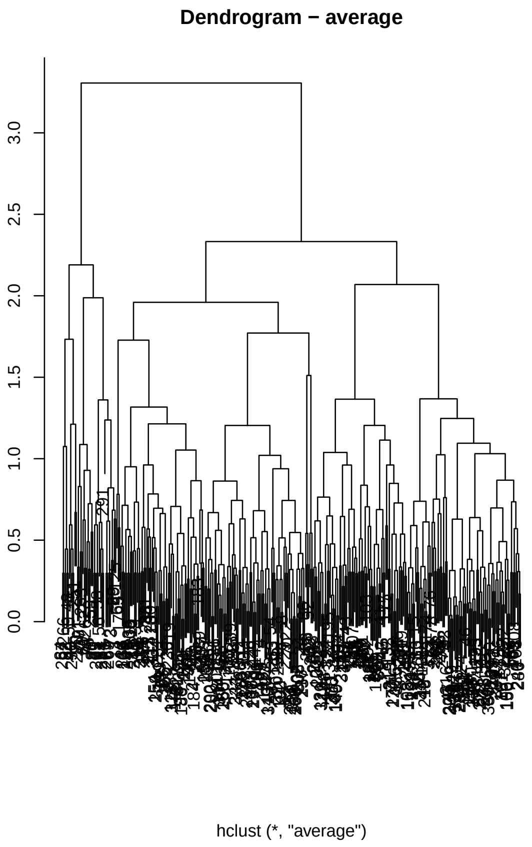


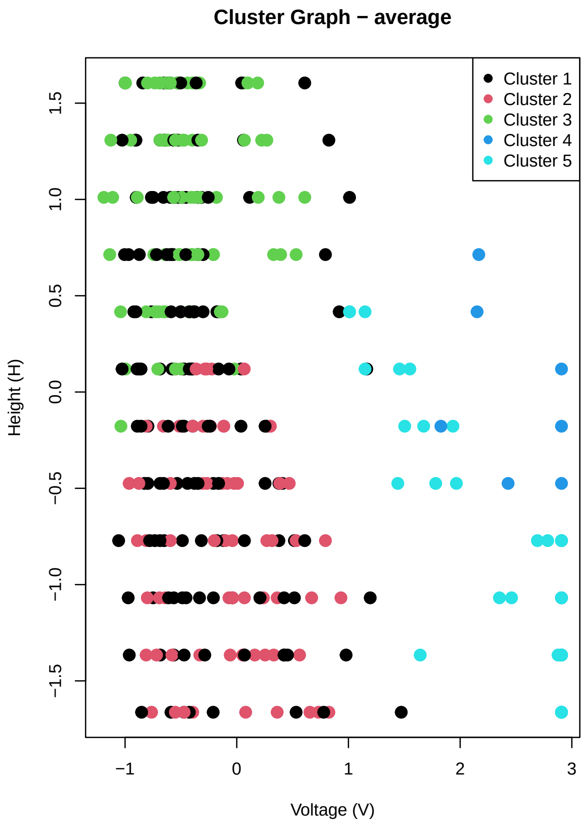
The plot visualizes five clusters (black, red, green, blue, cyan) on a Voltage (x-axis) vs. Height (y-axis) coordinate system, with points forming horizontal bands across height levels. Voltage ranges from -1 to 3, and Height ranges from -1.5 to 1.5, with a legend indicating clusters. The plot highlights how hierarchical clustering grouped data based on Voltage and Height.

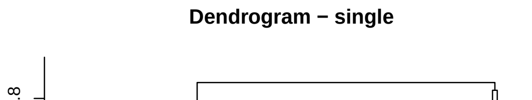
Experimenting for all methods

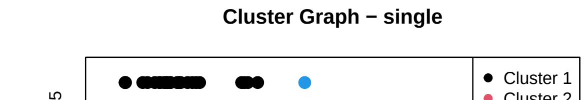












The code performs hierarchical clustering on mine_data_scaled using four linkage methods: complete, average, single, and centroid. For each method, it generates a dendrogram to visualize hierarchical cluster formation and a corresponding scatter plot to display clustering results in the voltage-height space. The clustering results are shown in distinct colors for each of the 5 clusters, with legends identifying the cluster groups.

Kernal PCA

```
# install.packages("kernlab")
library(kernlab)
```

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

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

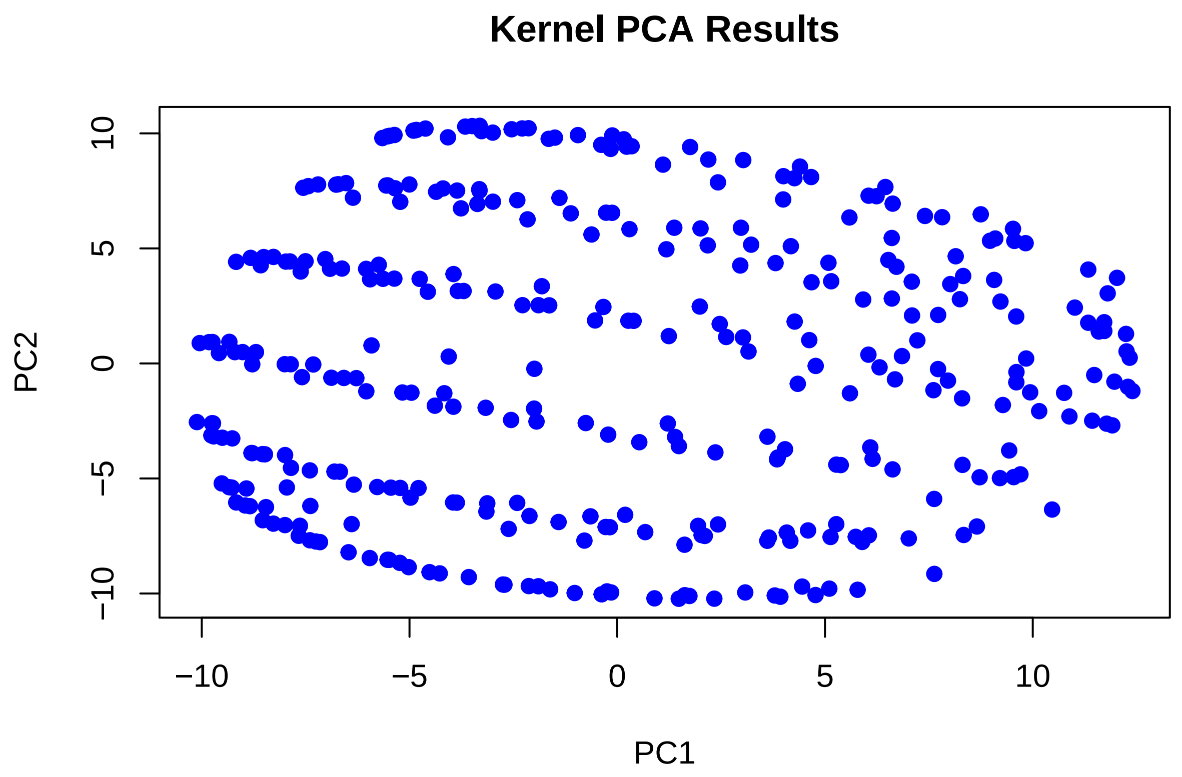
```
data_kpca <- mine_data_scaled_HC
kernel_pca <- kpca(~ V + H + S, data = data_kpca, kernel = "rbfdot", kpar = list(sigma = 0.1))
summary(kernel_pca)
```

```
## Length  Class   Mode
##      1   kpca     S4
```

```
eigenvalues <- kernel_pca@eig
eigenvectors <- pcv(kernel_pca)
head(eigenvalues)
```

```
##     Comp.1     Comp.2     Comp.3     Comp.4     Comp.5     Comp.6
## 0.12147127 0.10892741 0.07233538 0.02085045 0.01982561 0.01103622
```

```
transformed_data <- as.data.frame(predict(kernel_pca, data_kpca))
plot(transformed_data$V1, transformed_data$V2,
     xlab = "PC1", ylab = "PC2",
     main = "Kernel PCA Results",
     col = "blue", pch = 19)
```



The code applies Kernel PCA on mine_data_scaled_HC using an RBF kernel to extract nonlinear patterns. It computes eigenvalues and eigenvectors, then transforms the data into the kernel PCA space. Finally, it visualizes the first two principal components (PC1 vs. PC2) to reveal clusters or trends.

Test Train Split

```
set.seed(1234)
sample_index = sample(1:nrow(mine_data), size = 0.8 * nrow(mine_data))

train_data = mine_data[sample_index, ]
test_data = mine_data[-sample_index, ]

x_train = train_data[, c("V", "H", "S")]
y_train = train_data$M
x_test = test_data[, c("V", "H", "S")]
y_test = test_data$M
```

Scaling the data

```
x_train_scaled = scale(x_train)
x_test_scaled = scale(x_test)
head(x_train_scaled)
```

```
##              V          H          S
## 284  0.2333803 -1.4038301  0.8466755
## 336 -0.4121576 -0.2053032  1.4239542
## 101 -0.5043773  0.9932237 -1.4624395
## 111 -0.2277182 -1.1041984 -0.8851607
## 133  0.7713286 -0.8045667  1.4239542
## 98  -0.4900830 -0.2053032 -1.4624395
```

```
head(x_test_scaled)
```

```
##             V          H          S
## 1  -0.3987950 -1.5073643 -1.4550530
## 3  -0.6630990 -0.6460133 -1.4550530
## 8  -0.9322426  1.6509228 -1.4550530
## 16 -0.9284956  1.6509228  0.3223770
## 18 -0.6787105 -0.9331303 -0.8625763
## 45 -0.6362462  0.2153378  1.5073303
```

Applying K-means Clustering

```
classes = 5
kmeans_model <- kmeans(x_train_scaled, centers = classes, nstart = 25)
```

```
train_clusters <- kmeans_model$cluster
train_confusion_matrix <- table(train_clusters, y_train)

train_confusion_matrix
```

```
##               y_train
## train_clusters  1  2  3  4  5
##              1 17  0 13 11 11
##              2 14 13 14 13 14
##              3  0 30  1  0  0
##              4 15 10 15 15 12
##              5 14  6 10 10 12
```

```
map_clusters_to_labels <- function(cluster, y_train, clusters) {
  majority_label <- names(sort(table(y_train[clusters == cluster]), decreasing = TRUE))[1]
  return(majority_label)
}

cluster_to_label_map <- sapply(1:classes, function(cluster) {
  map_clusters_to_labels(cluster, y_train, train_clusters)
})

predicted_labels_train <- sapply(train_clusters, function(cluster) {
  cluster_to_label_map[cluster]
})

train_accuracy <- sum(predicted_labels_train == y_train) / length(y_train)
```

```r
print(paste("Train Accuracy:", round(train_accuracy, 4)))
```

```
## [1] "Train Accuracy: 0.3333"
```

```r
distances <- as.matrix(dist(rbind(kmeans_model$centers, x_test_scaled)))
distances <- distances[1:classes, -(1:classes)]

# Assign each test sample to the closest centroid
test_clusters <- apply(distances, 2, which.min)

# Map test clusters to actual class labels (using the same mapping from train)
predicted_labels_test <- sapply(test_clusters, function(cluster) {
  cluster_to_label_map[cluster]
})

# Calculate test accuracy
test_accuracy <- sum(predicted_labels_test == y_test) / length(y_test)
print(paste("Test Accuracy:", round(test_accuracy, 4)))
```

```
## [1] "Test Accuracy: 0.2647"
```

The K-Means clustering model achieved an **accuracy of just 26%**, which is quite low for a classification task. Therefore, exploring other models such as Logistic Regression, Random Forest, or XGBoost could potentially yield better results and improve predictive performance.

Random Forest

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.2

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine

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

```r
train_data$M <- as.factor(train_data$M)
test_data$M <- as.factor(test_data$M)
model_rf <- randomForest(M ~ V + H + S, data = train_data, ntree = 100)
print(model_rf)
```

```
##
## Call:
##  randomForest(formula = M ~ V + H + S, data = train_data, ntree = 100)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 1
##
##         OOB estimate of  error rate: 48.52%
## Confusion matrix:
##    1  2  3  4  5 class.error
## 1 49  0  5  4  2  0.18333333
## 2  0 54  0  5  0  0.08474576
## 3  5  4 17 11 16  0.67924528
## 4 11  5 14 10  9  0.79591837
## 5 10  1 24  5  9  0.81632653
```

```r
predictions_rf <- predict(model_rf, test_data)
conf_rf = confusionMatrix(predictions_rf, as.factor(test_data$M))
print(conf_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  6  0  0  3  4
##          2  0 10  1  2  0
##          3  0  0  5  1  7
##          4  3  1  3  6  3
##          5  2  0  4  5  2
##
## Overall Statistics
##
##                Accuracy : 0.4265
##                  95% CI : (0.3072, 0.5523)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 0.001098
##
##                   Kappa : 0.2811
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity           0.54545   0.9091  0.38462  0.35294  0.12500
## Specificity           0.87719   0.9474  0.85455  0.80392  0.78846
## Pos Pred Value        0.46154   0.7692  0.38462  0.37500  0.15385
## Neg Pred Value        0.90909   0.9818  0.85455  0.78846  0.74545
## Prevalence            0.16176   0.1618  0.19118  0.25000  0.23529
## Detection Rate        0.08824   0.1471  0.07353  0.08824  0.02941
## Detection Prevalence  0.19118   0.1912  0.19118  0.23529  0.19118
## Balanced Accuracy     0.71132   0.9282  0.61958  0.57843  0.45673
```

The Random Forest model achieved an overall **accuracy of 45.59%**, with a Kappa score of 0.3233, indicating moderate agreement. Class 2 has the highest sensitivity (90.91%) and balanced accuracy (92.82%), while Class 5 shows the weakest performance (12.5% sensitivity, 48.56% balanced accuracy). The confusion matrix reveals misclassifications, especially in Classes 3, 4, and 5, suggesting room for improvement in handling imbalanced or overlapping data.

SVM

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.4.2
```

```r
y_train <- factor(y_train, levels = levels(as.factor(test_data$M)))
svm_model <- svm(y_train ~ ., data = cbind(x_train_scaled, y_train), type = "C-classification", kernel =
svm_predictions <- predict(svm_model, x_test_scaled)
print(svm_predictions)
```

```
##   1   3   8  16  18  45  48  51  54  64  69  78  91 106 112 113 114 124 125 128
##   3   1   4   1   1   5   2   2   2   2   3   2   2   5   3   3   3   3   2   3
## 138 141 150 154 156 157 159 161 162 164 165 168 173 177 179 193 197 200 203 204
##   5   3   5   3   3   3   4   1   1   1   1   1   1   3   5   3   3   3   4   5
## 205 211 217 223 237 240 242 244 247 250 254 256 261 274 279 281 285 288 307 309
##   5   1   3   5   1   1   1   3   1   2   2   2   2   3   3   5   5   2   1   1
## 314 320 321 325 327 330 334 335
##   5   4   3   3   3   3   3   3
## Levels: 1 2 3 4 5
```

```r
svm_predictions <- factor(svm_predictions, levels = levels(as.factor(test_data$M)))
print(levels(svm_predictions))
```

```
## [1] "1" "2" "3" "4" "5"
```

```r
print(levels(as.factor(test_data$M)))
```

```
## [1] "1" "2" "3" "4" "5"
```

```r
conf_svm = confusionMatrix(svm_predictions, as.factor(test_data$M))
print(conf_svm)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction  1  2  3  4  5
##          1  7  0  0  8  1
##          2  0 10  2  0  0
##          3  2  1  7  5 10
##          4  1  0  0  1  2
##          5  1  0  4  3  3
##
## Overall Statistics
##
##                Accuracy : 0.4118
##                  95% CI : (0.2937, 0.5377)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : 0.002489
##
##                   Kappa : 0.2741
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.6364   0.9091   0.5385  0.05882  0.18750
## Specificity            0.8421   0.9649   0.6727  0.94118  0.84615
## Pos Pred Value         0.4375   0.8333   0.2800  0.25000  0.27273
## Neg Pred Value         0.9231   0.9821   0.8605  0.75000  0.77193
## Prevalence             0.1618   0.1618   0.1912  0.25000  0.23529
## Detection Rate         0.1029   0.1471   0.1029  0.01471  0.04412
## Detection Prevalence   0.2353   0.1765   0.3676  0.05882  0.16176
## Balanced Accuracy      0.7392   0.9370   0.6056  0.50000  0.51683
```

The SVM model achieved an overall **accuracy of 41.18%**, with a Kappa score of 0.2741, indicating fair agreement. Class 2 performed the best with 90.91% sensitivity and 93.70% balanced accuracy, while Class 4 had the weakest performance (5.88% sensitivity and 50.00% balanced accuracy). The confusion matrix highlights misclassifications, especially for Classes 3, 4, and 5, suggesting challenges in separating overlapping data points effectively.