Project-1

2023-04-20

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

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

```
## Warning: package 'lattice' was built under R version 4.2.3
```

```
library(class)
library(readxl)
```

```
## Warning: package 'readxl' was built under R version 4.2.3
```

```
library(rpart)
library(stats)
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.2.3
```

```
library(nnet)
library(ggplot2)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.2.3
```

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 4.2.3
```

We are using land mines data for our project

```
my_data <- read_excel("C:/Users/saite/OneDrive - University at Buffalo/Documents/2nd Sem Coursew
ork/SDM2/Project1/Mine_Dataset.xls",sheet=2)
```

Displaying the top 20 rows of our dataset

```
head(my_data, 20)
```

```
## # A tibble: 20 × 4
##        V     H     S     M
##    <dbl> <dbl> <dbl> <dbl>
##  1 0.338 0       0       1
##  2 0.320 0.182   0       1
##  3 0.287 0.273   0       1
##  4 0.256 0.455   0       1
##  5 0.263 0.545   0       1
##  6 0.241 0.727   0       1
##  7 0.254 0.818   0       1
##  8 0.235 1       0       1
##  9 0.353 0       0.6     1
## 10 0.335 0.182   0.6     1
## 11 0.335 0.273   0.6     1
## 12 0.330 0.455   0.6     1
## 13 0.335 0.545   0.6     1
## 14 0.305 0.727   0.6     1
## 15 0.256 0.818   0.6     1
## 16 0.236 1       0.6     1
## 17 0.315 0       0.2     1
## 18 0.284 0.182   0.2     1
## 19 0.303 0.273   0.2     1
## 20 0.275 0.455   0.2     1
```

Displaying Structure of our landmines dataset including the number of rows and variables, and the type of data in each feature.

```
str(my_data)
```

```
## tibble [338 × 4] (S3: tbl_df/tbl/data.frame)
##  $ V: num [1:338] 0.338 0.32 0.287 0.256 0.263 ...
##  $ H: num [1:338] 0 0.182 0.273 0.455 0.545 ...
##  $ S: num [1:338] 0 0 0 0 0 0 0 0 0.6 0.6 ...
##  $ M: num [1:338] 1 1 1 1 1 1 1 1 1 1 ...
```

Statistics of every feature of our dataset

```
summary(my_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
```

Normalizing the data

```
# Extract the feature variables from the data
features <- my_data[, c("V", "H", "S")]

# Normalize the feature variables
normalized_features <- scale(features)

# Combine the normalized features with the target variable
normalized_data <- data.frame(normalized_features, M = my_data$M)
```

The dimensions of the data are obtained by dim function

```
dim(normalized_data)
```

```
## [1] 338   4
```

Names of the columns present in data

```
colnames(normalized_data)
```

```
## [1] "V" "H" "S" "M"
```

V - Voltage

H - High

S - Soil Type

M - Mine Type

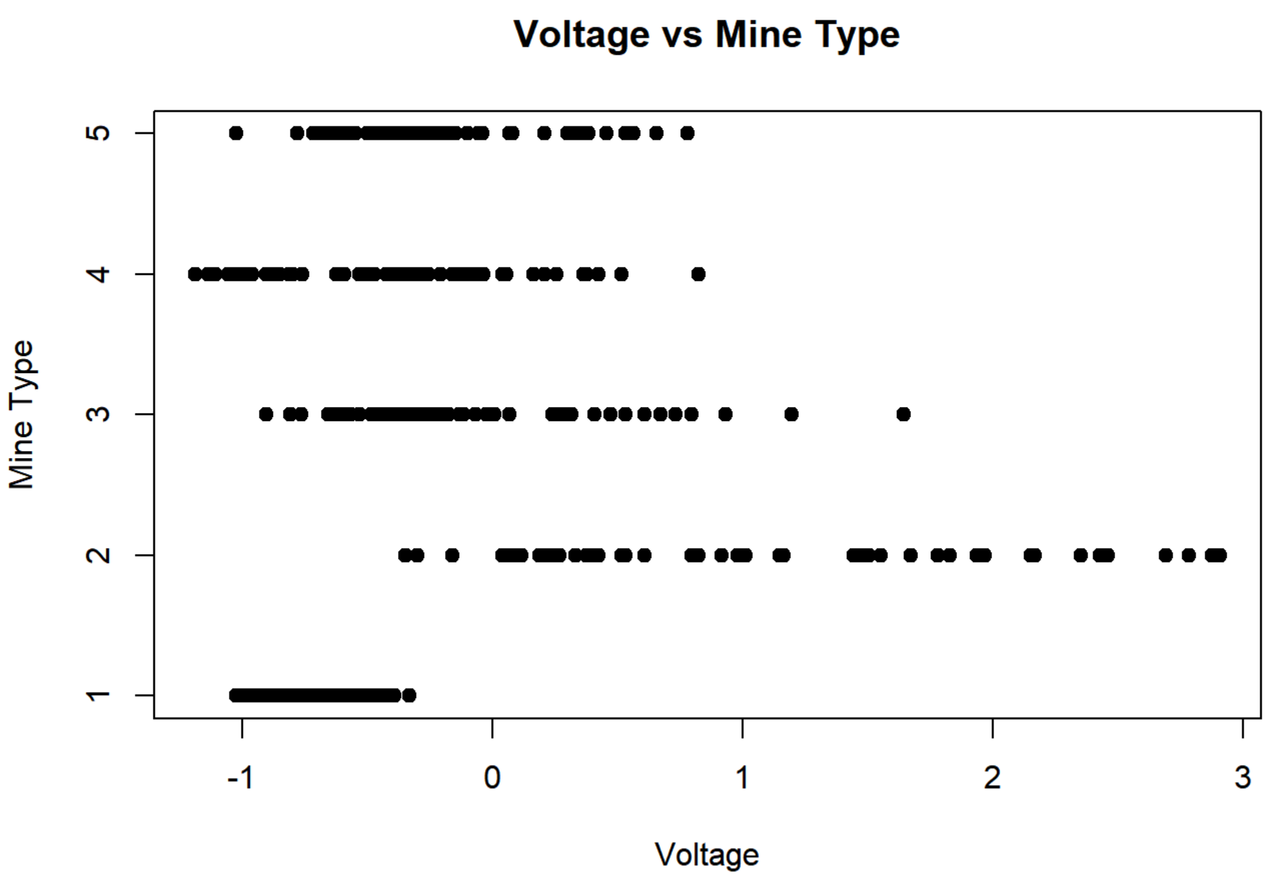
Finding unique elements in the predictor column

```
unique(normalized_data$M)
```

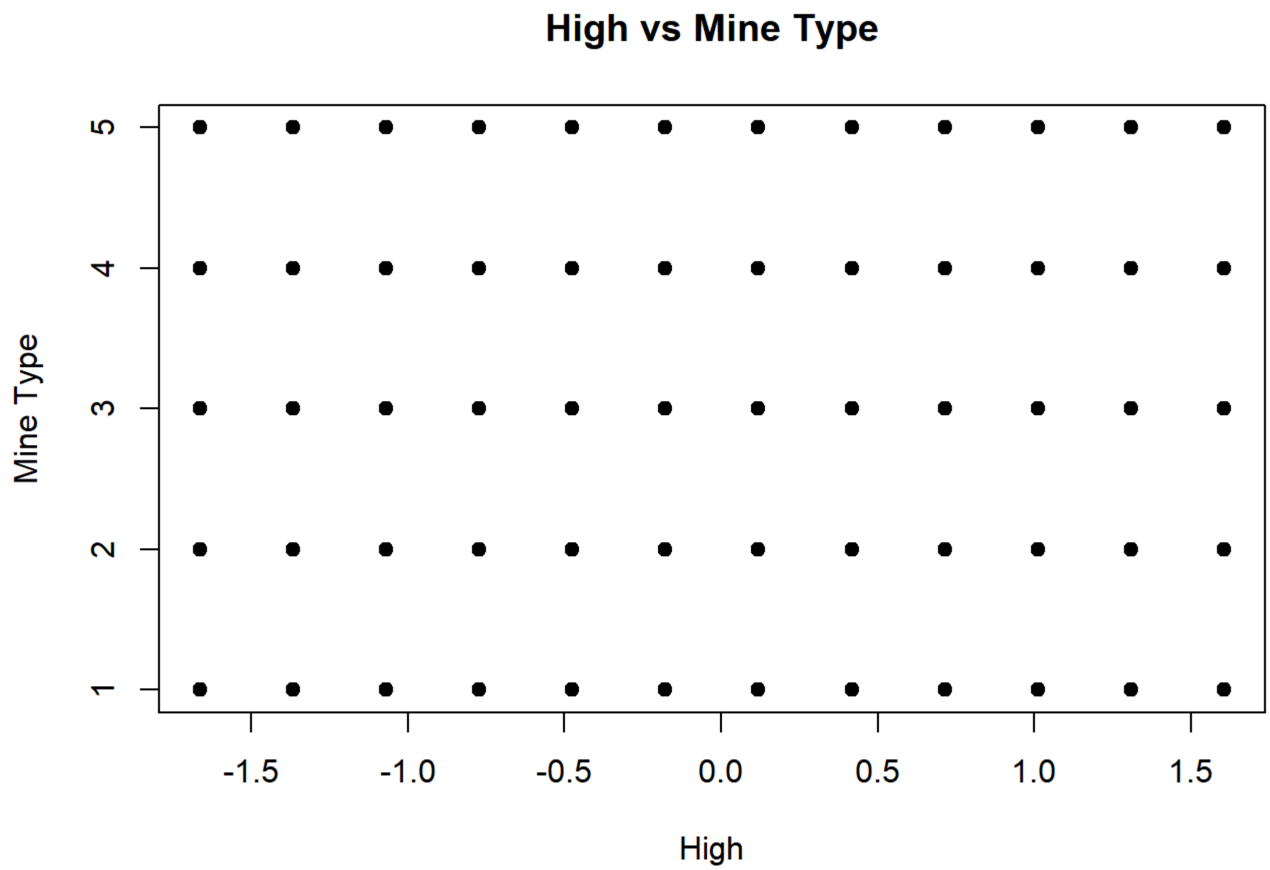
```
## [1] 1 2 3 4 5
```

Finding the relation between Feature and target variables.

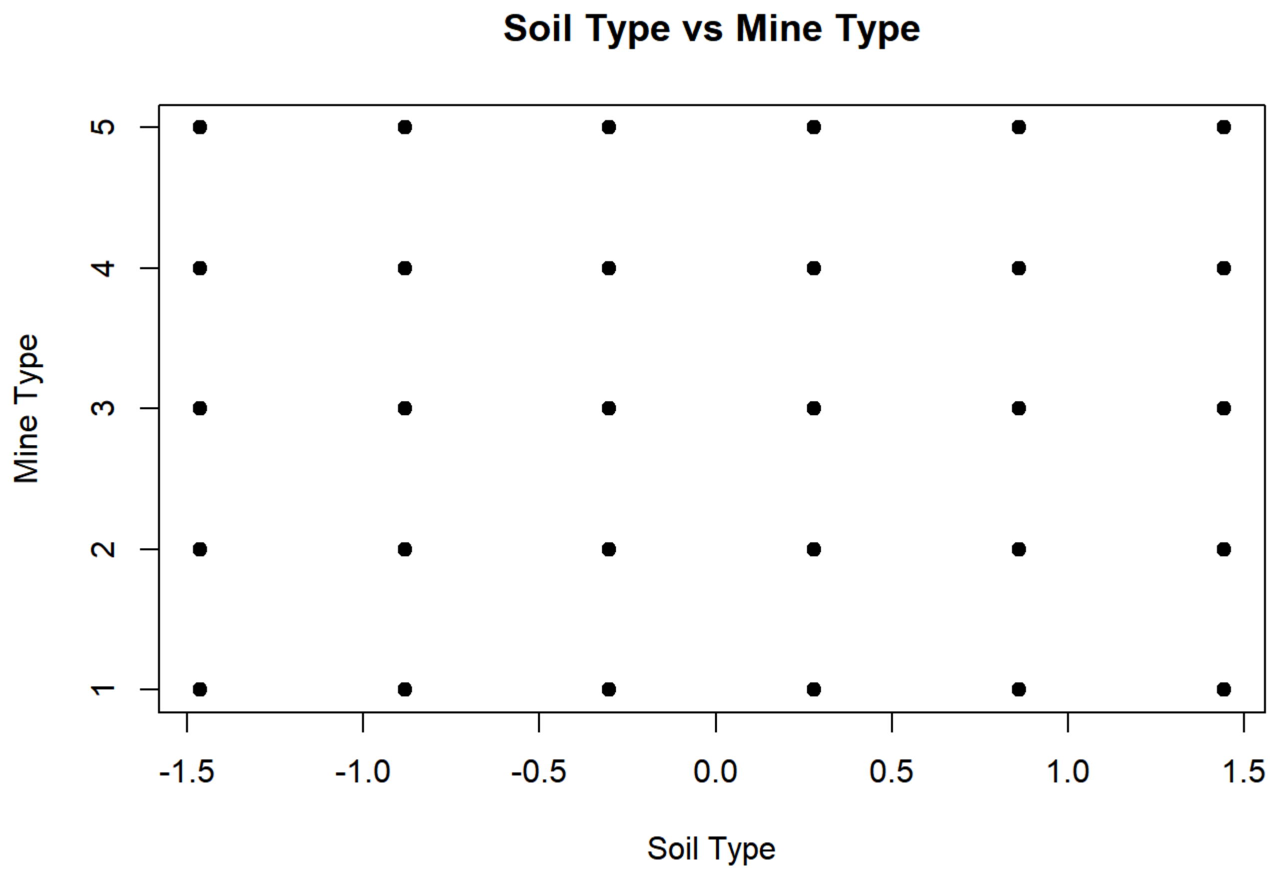
```
plot(normalized_data$V, normalized_data$M, main="Voltage vs Mine Type",
   xlab="Voltage", ylab="Mine Type", pch=19)
```




```
plot(normalized_data$H, normalized_data$M, main="High vs Mine Type",
   xlab="High", ylab="Mine Type", pch=19)
```




```
plot(normalized_data$S, normalized_data$M, main="Soil Type vs Mine Type",
   xlab="Soil Type", ylab="Mine Type", pch=19)
```



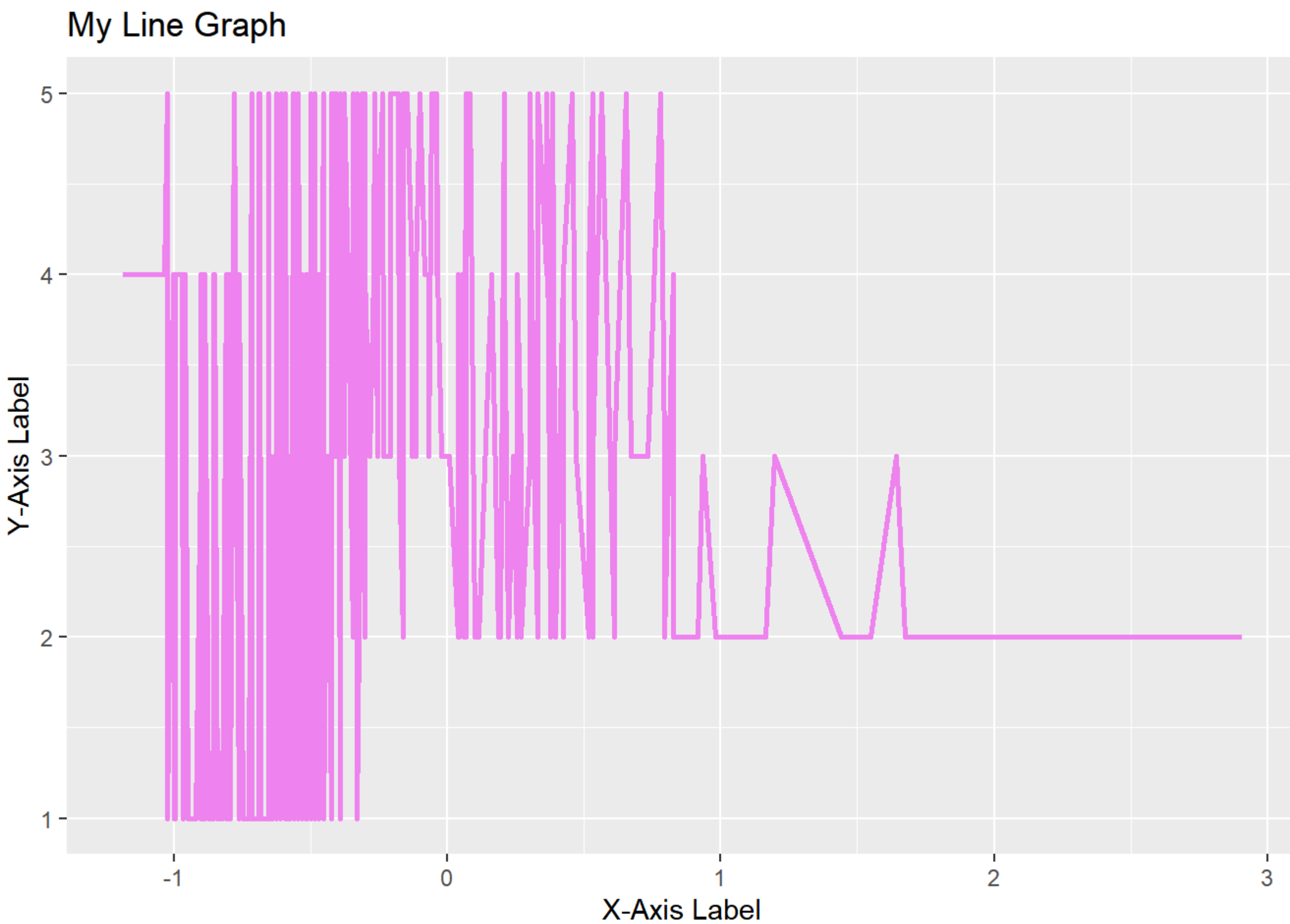

```
cor(normalized_data)
```

```
##             V            H            S           M
## V  1.00000000 -0.377523411  0.070673464 -0.14456945
## H -0.37752341  1.000000000 -0.006957347  0.04132607
## S  0.07067346 -0.006957347  1.000000000  0.01734552
## M -0.14456945  0.041326070  0.017345516  1.00000000
```

Voltage type distribution

```
ggplot(my_data, aes(x=normalized_data$V, y=normalized_data$M)) +
  geom_line(color="violet", size=1) +
  labs(title="My Line Graph", x="X-Axis Label", y="Y-Axis Label")
```

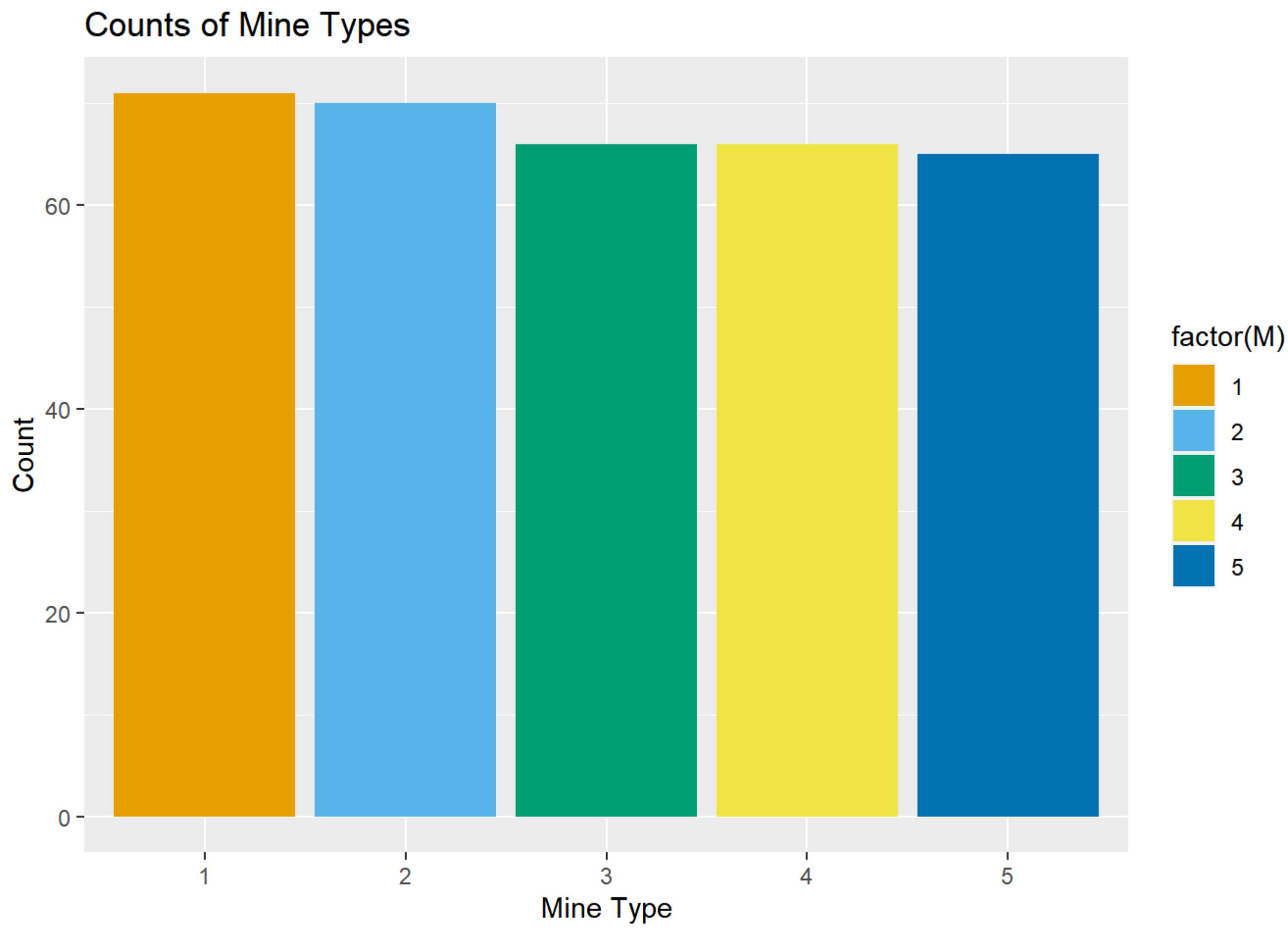
```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Barchart to visualize different mine types

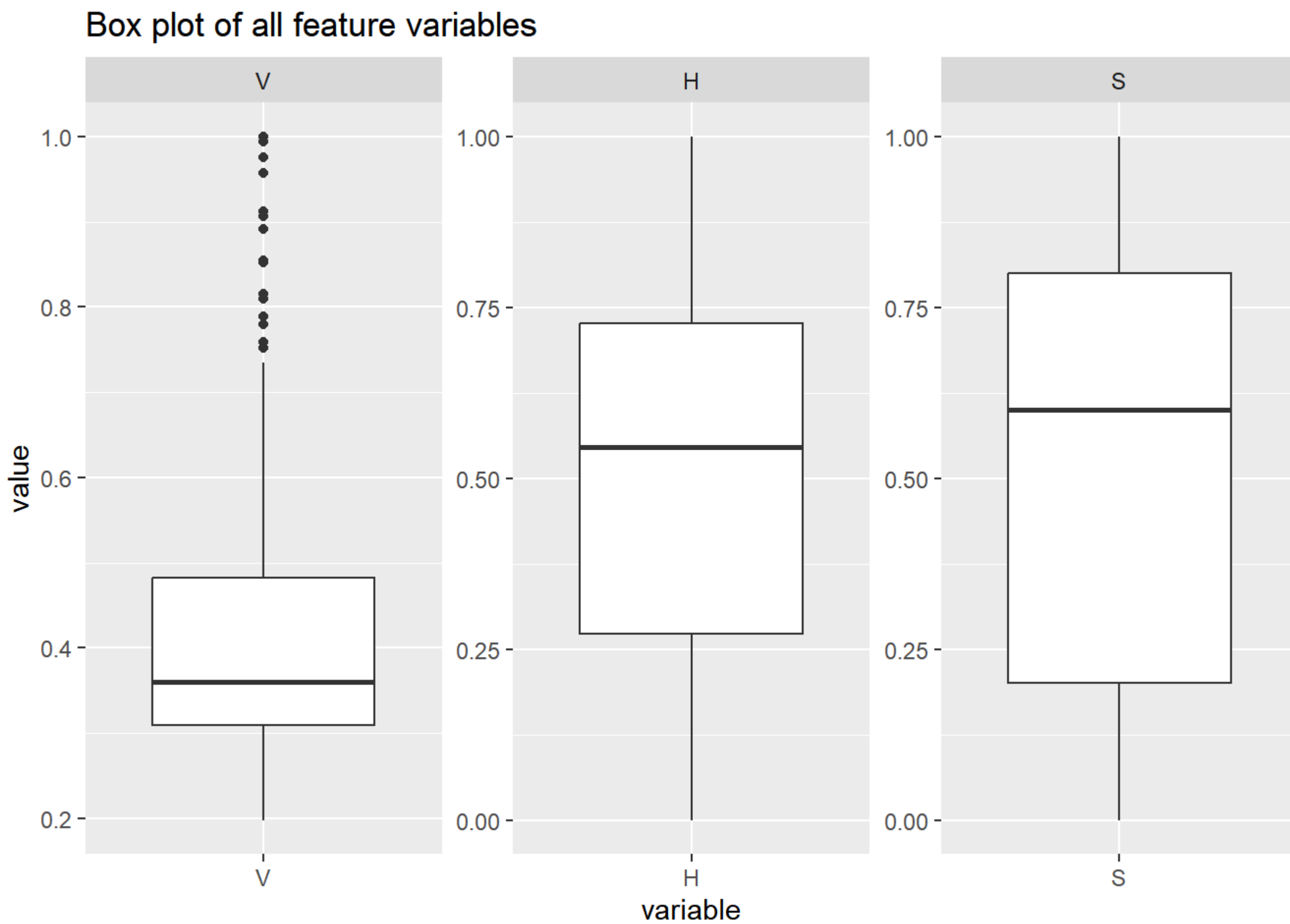
```
mine_colors <- c("#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2")

ggplot(normalized_data, aes(x = factor(M), fill = factor(M))) +
  geom_bar() +
  scale_fill_manual(values = mine_colors) +
  labs(title = "Counts of Mine Types", x = "Mine Type", y = "Count")
```



Boxplot to find the outliers

```
ggplot(melt(my_data, id.vars="M"), aes(x=variable, y=value)) +
  geom_boxplot() +
  facet_wrap(~ variable, scales="free") +
  labs(title="Box plot of all feature variables")
```



Algorithms to be Performed

K-Means Clusteting

```
set.seed(123)
train_pct <- 0.7
train_size <- round(nrow(normalized_data) * train_pct)
train_indices <- sample(seq_len(nrow(normalized_data)), size = train_size)
train_data <- normalized_data[train_indices, ]
test_data <- normalized_data[-train_indices, ]
```

```
kmeans_result <- kmeans(train_data[, 1:3], centers = 3)
summary(kmeans_result)
```

```
##              Length Class  Mode
## cluster      237    -none- numeric
## centers        9    -none- numeric
## totss          1    -none- numeric
## withinss       3    -none- numeric
## tot.withinss   1    -none- numeric
## betweenss      1    -none- numeric
## size           3    -none- numeric
## iter           1    -none- numeric
## ifault         1    -none- numeric
```

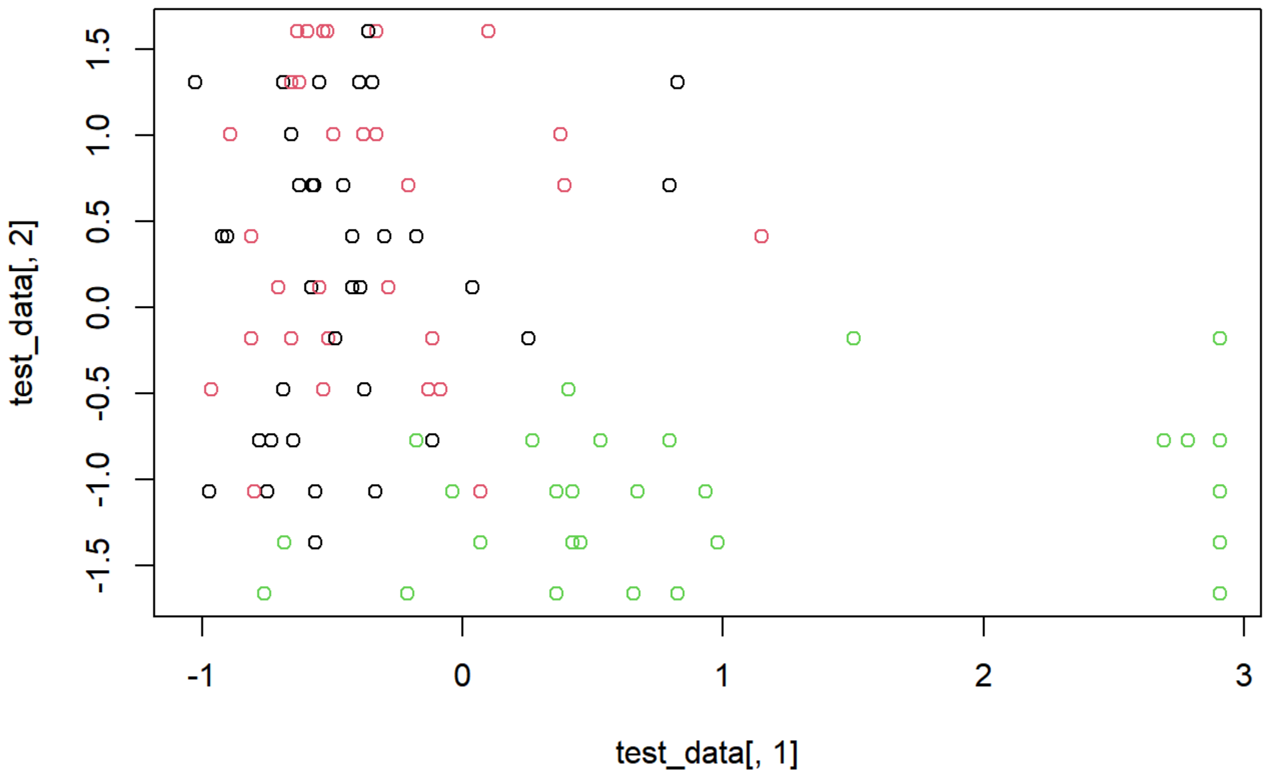
```
centers <- kmeans_result$centers
```

```
# Use the cluster centers to predict the clusters of the test data
test_clusters <- apply(test_data[, 1:3], 1, function(x) {
  # Calculate the distance from each point to each center
  distances <- apply(centers, 1, function(c) sqrt(sum((x - c)^2)))

  # Return the index of the closest center
  which.min(distances)
})
print(test_clusters)
```

```
##   2   3  12  15  18  19  28  38  44  45  47  49  50  56  58  62  65  68  71  73
##   1   1   2   2   1   1   2   1   2   2   2   3   3   3   3   2   3   1   3   3
##  80  82  87  92  95  96  99 100 103 112 119 120 122 123 124 126 128 130 132 133
##   3   3   3   2   3   1   1   1   3   1   3   2   2   2   2   3   1   1   3   3
## 138 140 145 146 148 150 161 162 169 175 181 182 185 186 189 191 192 193 198 200
##   2   1   1   3   2   2   3   2   1   3   2   2   1   1   1   3   3   3   2   1
## 202 205 216 222 225 226 228 231 234 237 252 255 257 258 259 261 265 268 271 278
##   1   3   1   2   2   1   1   2   3   1   1   3   2   3   1   3   3   1   2   2
## 279 281 282 286 287 297 298 301 302 304 307 311 314 315 317 318 331 333 334 335
##   2   3   1   2   2   1   1   2   3   1   2   1   2   2   3   1   3   1   1   2
## 338
##   2
```

```
plot(test_data[,1], test_data[,2], col = test_clusters)
```



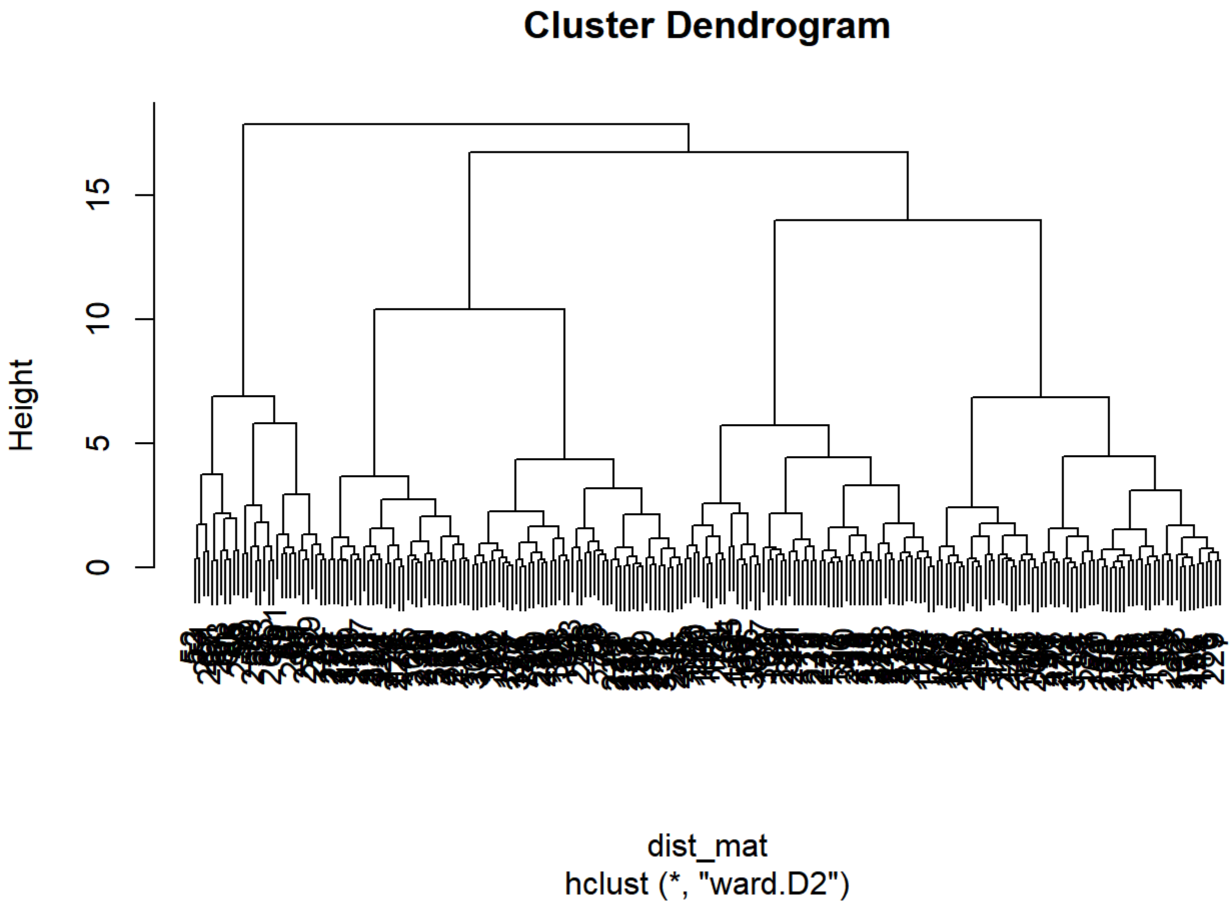
Hirerachial Clustering

```
set.seed(123)
train_pct <- 0.7
train_size <- round(nrow(normalized_data) * train_pct)
train_indices <- sample(seq_len(nrow(normalized_data)), size = train_size)
train_data <- normalized_data[train_indices, ]
test_data <- normalized_data[-train_indices, ]
```

```
# Perform hierarchical clustering on the first three variables of the training data
dist_mat <- dist(train_data[, 1:3])
hclust_result <- hclust(dist_mat, method = "ward.D2")
summary(hclust_result)
```

```
##              Length Class  Mode
## merge        472    -none- numeric
## height       236    -none- numeric
## order        237    -none- numeric
## labels       237    -none- character
## method         1    -none- character
## call           3    -none- call
## dist.method    1    -none- character
```

```
# Plot the dendrogram
plot(hclust_result)
```



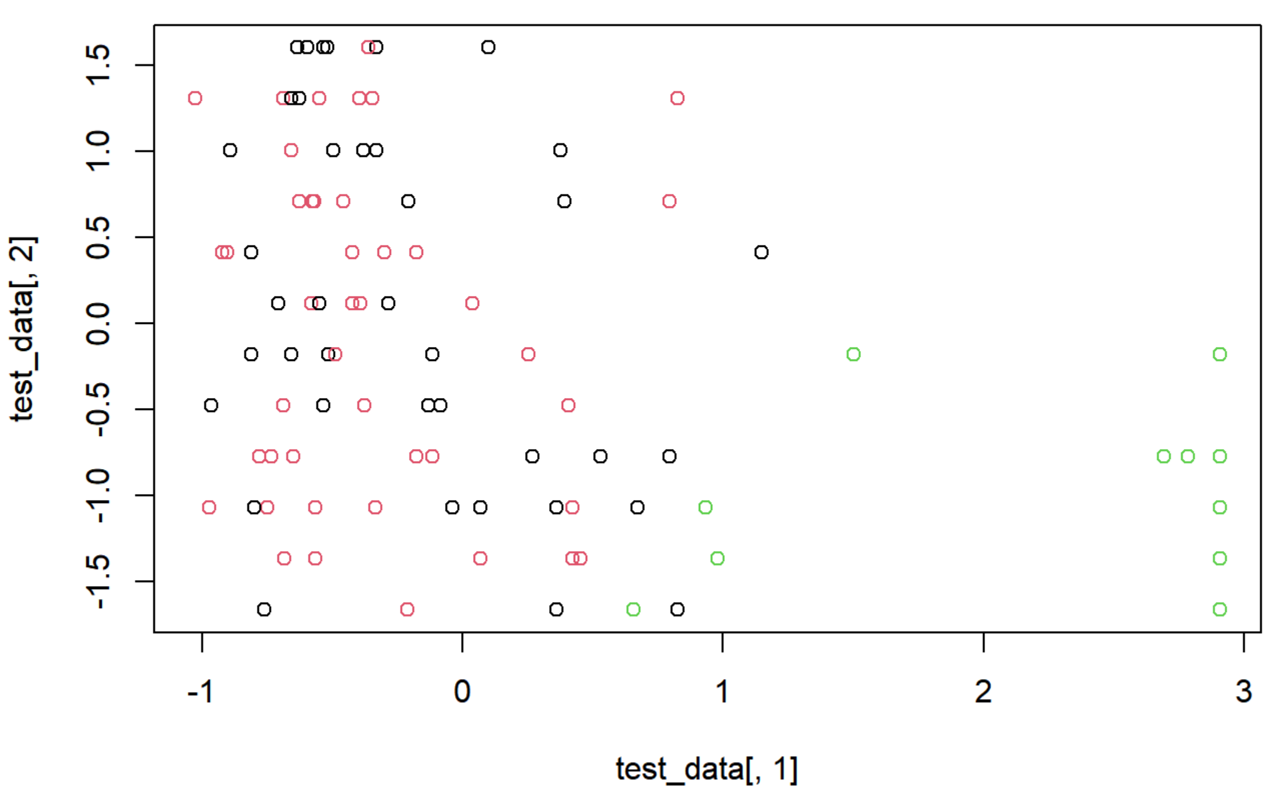

```
train_clusters <- cutree(hclust_result, k = 3)

# Use the cluster labels to predict the clusters of the test data
test_clusters <- apply(test_data[, 1:3], 1, function(x) {
  # Calculate the distance from each point to each cluster center
  cluster_centers <- aggregate(train_data[, 1:3], list(train_clusters), mean)
  distances <- apply(cluster_centers[, -1], 1, function(c) sqrt(sum((x - c)^2)))

  # Return the index of the closest cluster
  which.min(distances)
})
print(test_clusters)
```

```
##   2   3  12  15  18  19  28  38  44  45  47  49  50  56  58  62  65  68  71  73
##   2   2   1   1   2   2   1   2   1   1   1   3   3   3   3   1   3   2   3   3
##  80  82  87  92  95  96  99 100 103 112 119 120 122 123 124 126 128 130 132 133
##   3   3   3   1   2   2   2   2   3   2   1   1   1   1   1   2   2   2   1   1
## 138 140 145 146 148 150 161 162 169 175 181 182 185 186 189 191 192 193 198 200
##   1   2   2   1   1   1   1   1   2   1   1   1   2   2   2   3   1   1   1   2
## 202 205 216 222 225 226 228 231 234 237 252 255 257 258 259 261 265 268 271 278
##   2   1   2   1   1   2   2   1   2   2   2   3   1   2   2   3   3   2   1   1
## 279 281 282 286 287 297 298 301 302 304 307 311 314 315 317 318 331 333 334 335
##   1   2   2   1   1   2   2   1   2   2   1   2   1   1   2   2   2   2   2   1
## 338
##   1
```

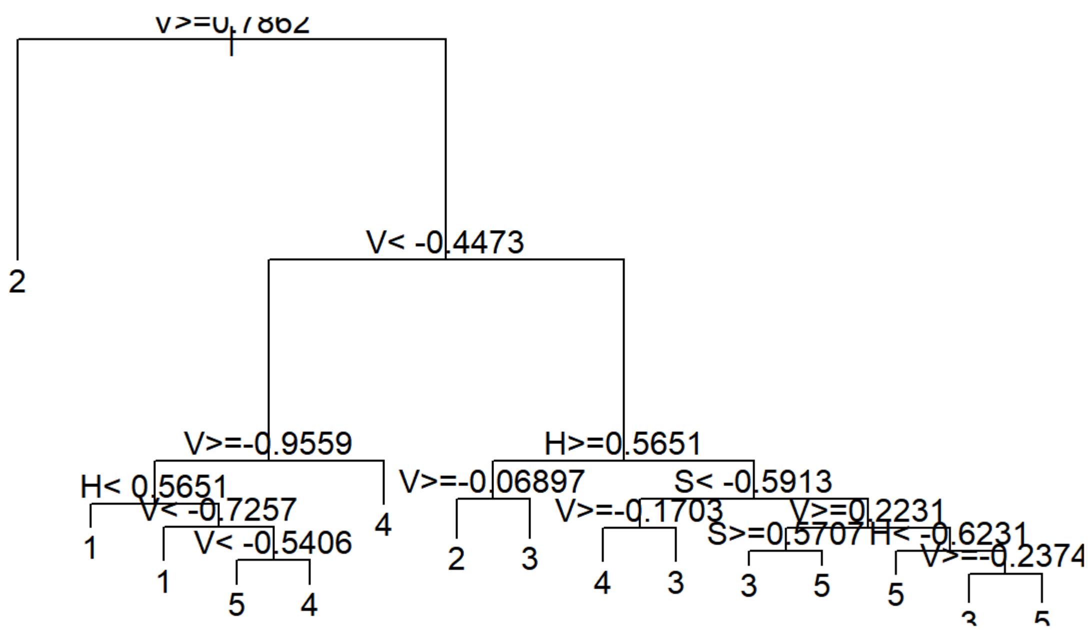
```
# Plot the test data with different colors representing the predicted clusters
plot(test_data[,1], test_data[,2], col = test_clusters)
```



Decision tree algorithm

```
# Create the decision tree
tree <- rpart(M ~ V + H + S, data = normalized_data, method = "class")

# Plot the decision tree
plot(tree)
text(tree)
```




```
# Create a new dataset to predict on
new_data <- data.frame(V = c(1, 2, 3),
                       H = c(4, 5, 6),
                       S = c(7, 8, 9))

# Make predictions using the decision tree
predictions <- predict(tree, new_data, type = "class")
predictions
```

```
## 1 2 3
## 2 2 2
## Levels: 1 2 3 4 5
```

Neural Networks

```
set.seed(123)
train_idx <- sample(nrow(normalized_data), nrow(normalized_data)*0.7)
train_data <- normalized_data[train_idx, ]
test_data <- normalized_data[-train_idx, ]

# Fit a neural network model
nnet_model <- nnet(M ~ V + H + S, data = train_data, size = 3)
```

```
## # weights:  16
## initial  value 1819.492187
## final  value 1350.000000
## converged
```

```
# Make predictions on the test set
nnet_predictions <- predict(nnet_model, newdata = test_data)

# Evaluate the performance of the model
nnet_accuracy <- mean(nnet_predictions == test_data$M)
nnet_accuracy
```

```
## [1] 0.1568627
```

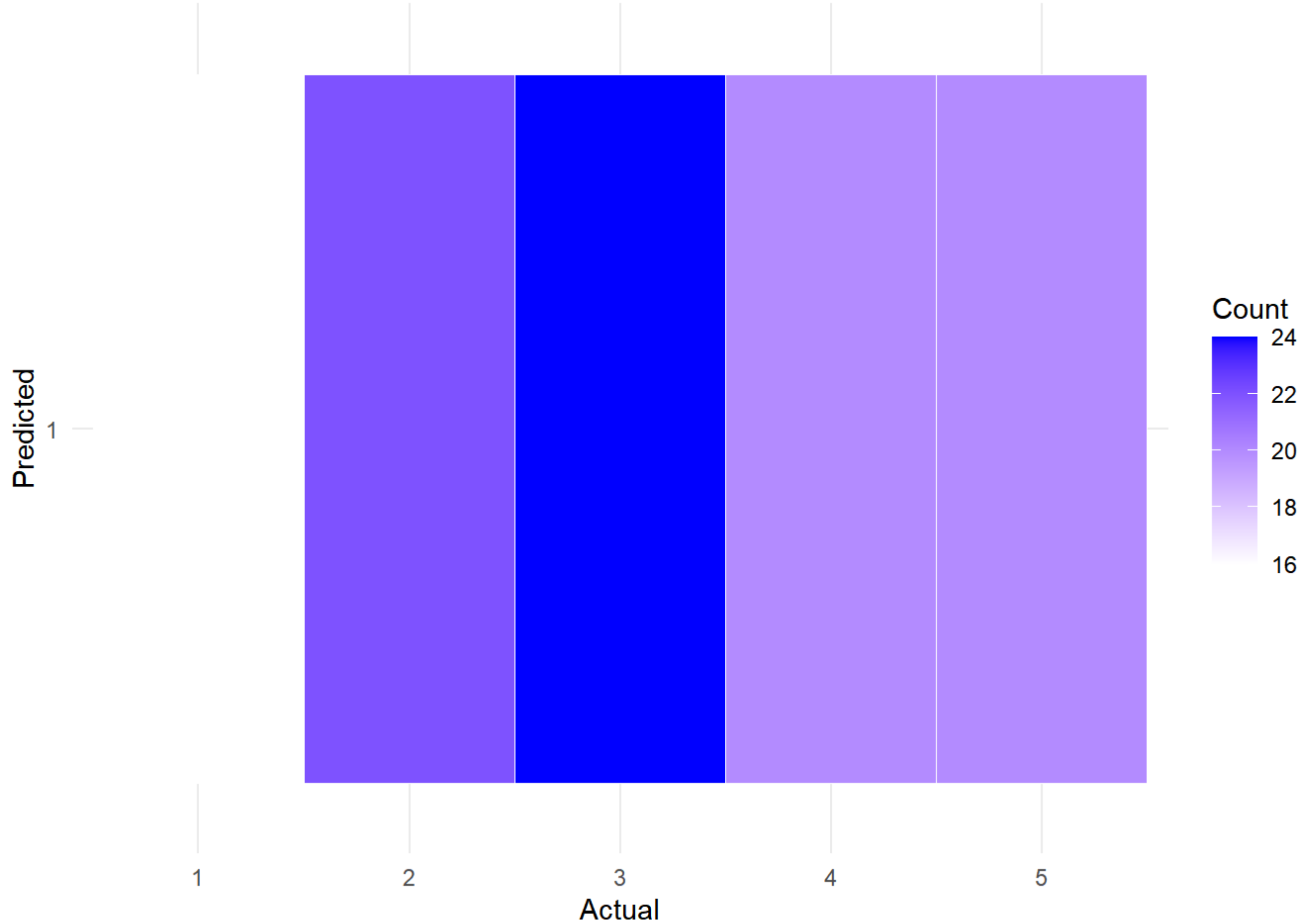
Confusion Matrix

```
nnet_confusion <- table(nnet_predictions, test_data$M)
nnet_confusion
```

```
##
## nnet_predictions  1  2  3  4  5
##                1 16 22 24 20 20
```

```
nnet_confusion_df <- as.data.frame.matrix(nnet_confusion)
nnet_confusion_df$predicted <- rownames(nnet_confusion_df)
nnet_confusion_df <- tidyr::gather(nnet_confusion_df, actual, value, -predicted)

# Create confusion matrix plot
ggplot(nnet_confusion_df, aes(x = actual, y = predicted, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "white", high = "blue") +
  theme_minimal() +
  labs(x = "Actual", y = "Predicted", fill = "Count")
```



KNN

```
# Split the data into training and testing sets
train_idx <- sample(nrow(normalized_data), nrow(normalized_data)*0.7)
train <- normalized_data[train_idx, ]
test <- normalized_data[-train_idx, ]

# Create the k-Nearest Neighbors model
k <- 5  # set the number of neighbors to consider
predicted <- knn(train[, c("V", "H", "S")], test[, c("V", "H", "S")], train$M, k)
summary(predicted)
```

```
##  1  2  3  4  5
## 31 22 21 13 15
```

```
# Evaluate the model's accuracy
actual <- test$M
accuracy <- mean(predicted == actual)
cat("Accuracy:", round(accuracy, 2))
```

```
## Accuracy: 0.44
```

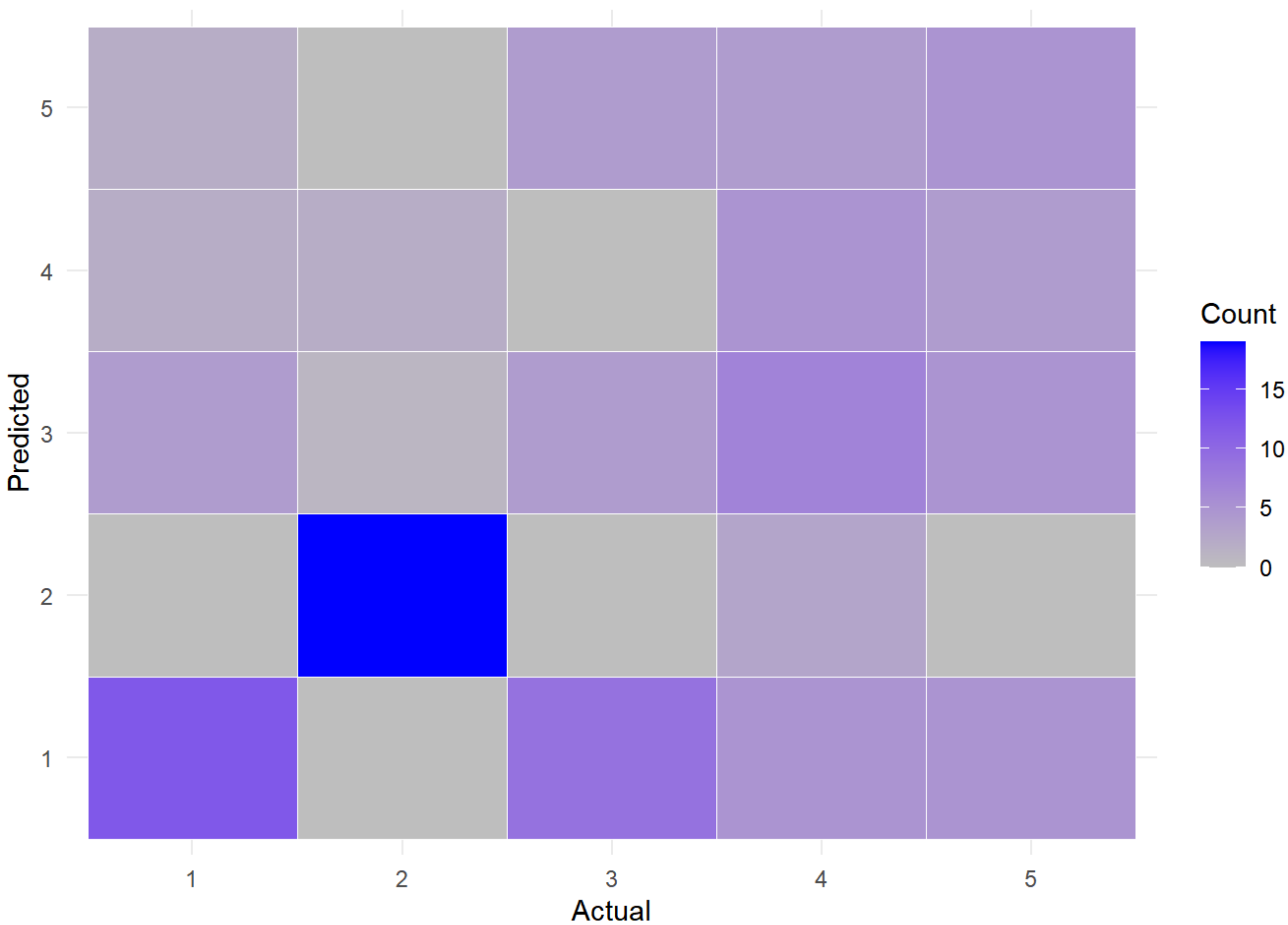
Confusion Matrix

```
knn_confusion <- table(predicted, actual)
knn_confusion
```

```
##          actual
## predicted  1  2  3  4  5
##         1 12  0  9  5  5
##         2  0 19  0  3  0
##         3  4  1  4  7  5
##         4  2  2  0  5  4
##         5  2  0  4  4  5
```

```
knn_confusion_df <- as.data.frame.matrix(knn_confusion)
knn_confusion_df$predicted <- rownames(knn_confusion_df)
knn_confusion_df <- tidyr::gather(knn_confusion_df, actual, value, -predicted)

# Create confusion matrix plot
ggplot(knn_confusion_df, aes(x = actual, y = predicted, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "grey", high = "blue") +
  theme_minimal() +
  labs(x = "Actual", y = "Predicted", fill = "Count")
```



Multinomial Logistic Regression

```
# Fit a multinomial logistic regression model
model <- multinom(M ~ V + H + S, data = normalized_data)
```

```
## # weights:  25 (16 variable)
## initial  value 543.990014
## iter  10 value 365.300110
## iter  20 value 354.464893
## final  value 353.427237
## converged
```

```
summary(model)
```

```
## Call:
## multinom(formula = M ~ V + H + S, data = normalized_data)
##
## Coefficients:
##   (Intercept)         V         H           S
## 2  0.01669617 12.658799 4.1336199 -1.30995804
## 3  2.52008537  6.081081 1.1650168 -0.25325736
## 4  1.67621726  3.060963 0.3512396 -0.06063898
## 5  2.28893611  4.910304 0.8255910 -0.12018539
##
## Std. Errors:
##   (Intercept)         V         H         S
## 2   0.6665148 1.4909007 0.6445234 0.4459910
## 3   0.4316487 0.8024127 0.2605537 0.2060755
## 4   0.4355513 0.6764230 0.1990000 0.1798208
## 5   0.4332435 0.7489207 0.2329597 0.1955670
##
## Residual Deviance: 706.8545
## AIC: 738.8545
```

```
# Extract the test set from my_data using the test logical vector
test_data <- normalized_data[-train_idx, ]

# Make predictions on the test set
predictions <- predict(model, newdata = test_data, type = "class")

# Evaluate the performance of the model
accuracy <- mean(predictions == test_data$M)
accuracy
```

```
## [1] 0.5784314
```

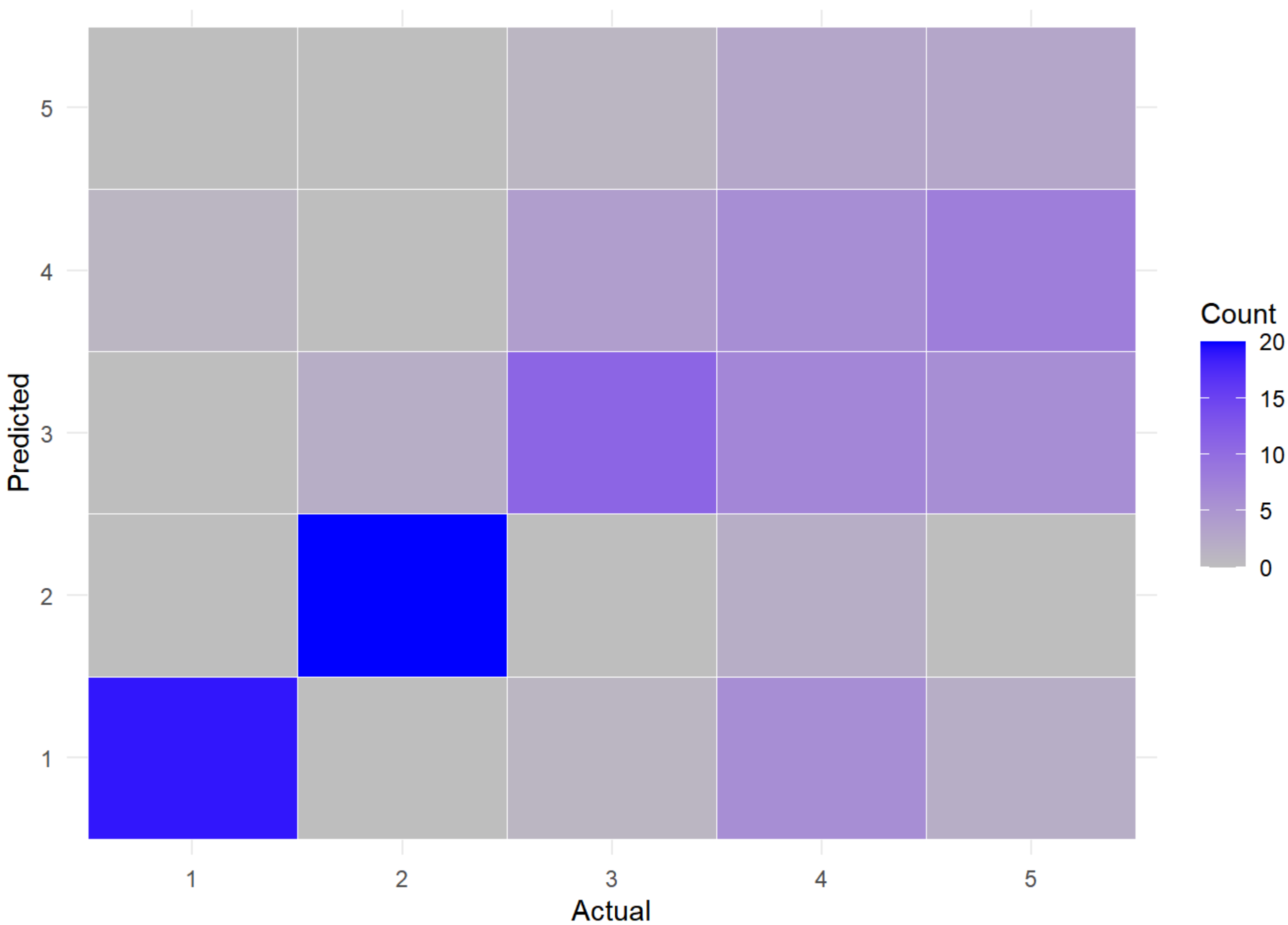
Confusion Matrix

```
mlr_confusion <- table(predictions, test_data$M)
mlr_confusion
```

```
##
## predictions  1  2  3  4  5
##           1 19  0  1  6  2
##           2  0 20  0  2  0
##           3  0  2 11  7  6
##           4  1  0  4  6  8
##           5  0  0  1  3  3
```

```
mlr_confusion_df <- as.data.frame.matrix(mlr_confusion)
mlr_confusion_df$predicted <- rownames(mlr_confusion_df)
mlr_confusion_df <- tidyr::gather(mlr_confusion_df, actual, value, -predicted)

# Create confusion matrix plot
ggplot(mlr_confusion_df, aes(x = actual, y = predicted, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "grey", high = "blue") +
  theme_minimal() +
  labs(x = "Actual", y = "Predicted", fill = "Count")
```



Naive Bayes

```
# Split the data into training and testing sets
train_indices <- sample(nrow(normalized_data), 0.7 * nrow(normalized_data))
train_data <- normalized_data[train_indices, ]
test_data <- normalized_data[-train_indices, ]

# Create the Naive Bayes classifier
nb_classifier <- naiveBayes(M ~ V + H + S, data = train_data)
summary(nb_classifier)
```

```
##           Length Class  Mode
## apriori   5      table  numeric
## tables    3      -none- list
## levels    5      -none- character
## isnumeric 3      -none- logical
## call      4      -none- call
```

```
# Make predictions on the testing set
predictions <- predict(nb_classifier, test_data)
table(predictions, test_data$M)
```

```
##
## predictions  1  2  3  4  5
##           1 25  0  4  7  7
##           2  0 11  4  2  2
##           3  0  3  3  1  4
##           4  0  0  0  1  1
##           5  0  0  8 10  9
```

```
# Calculate the accuracy of the classifier
accuracy <- sum(predictions == test_data$M) / length(predictions)
accuracy
```

```
## [1] 0.4803922
```

Confusion Matrix

```
nb_confusion <- table(predictions, test_data$M)
nb_confusion
```

```
##
## predictions  1  2  3  4  5
##           1 25  0  4  7  7
##           2  0 11  4  2  2
##           3  0  3  3  1  4
##           4  0  0  0  1  1
##           5  0  0  8 10  9
```

```
nb_confusion_df <- as.data.frame.matrix(nb_confusion)
nb_confusion_df$predicted <- rownames(nb_confusion_df)
nb_confusion_df <- tidyr::gather(nb_confusion_df, actual, value, -predicted)

# Create confusion matrix plot
ggplot(nb_confusion_df, aes(x = actual, y = predicted, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "grey", high = "blue") +
  theme_minimal() +
  labs(x = "Actual", y = "Predicted", fill = "Count")
```