EAS 509

Statistical Leaning II

Project-1

Team Members:

Sai Teja Dondeti (saitejad)

Sai Teja Sankarneni (saitejas)

Nishant Varma Indukuri (nindukur)

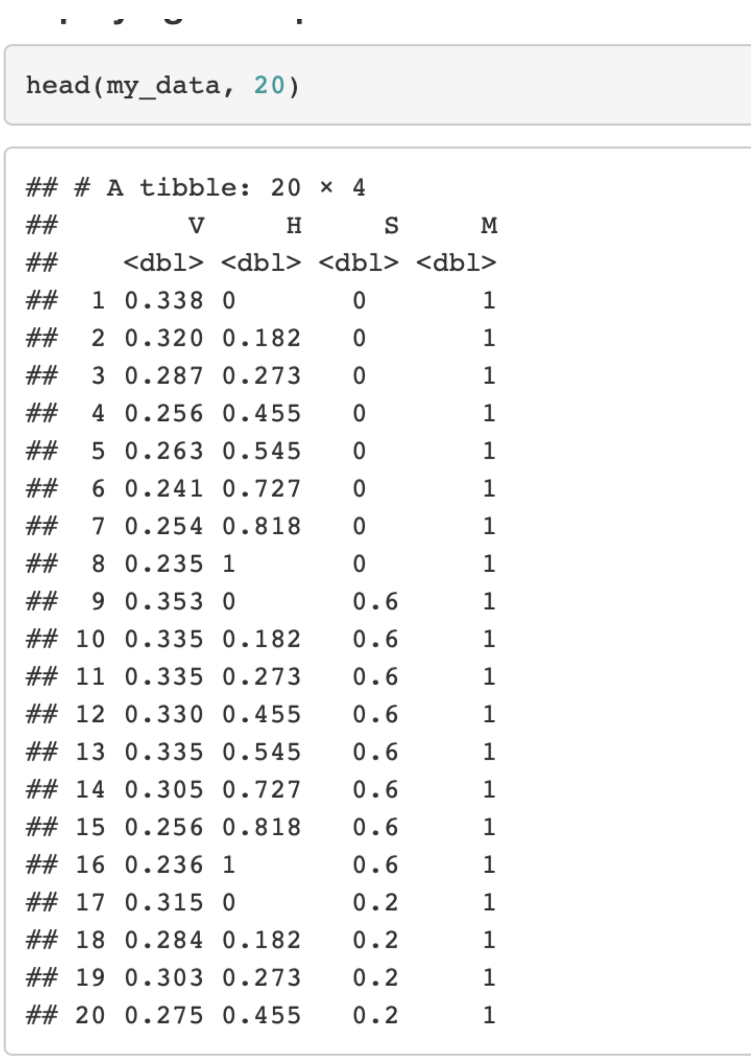
Shanawaz Pathan (spathan)

DATA LOADING

We have chosen the Land Mines dataset for this project, cause detection of mines which are hidden under the ground are crucial when it comes to the life of the people and their property.

In order to procced with this data set we need to know few statistics about it. We are loading the data from the second sheet of the `Mine_Dataset.xls` into a data frame named ‘my_data’.

This data set contains 4 features they are ‘V’, ‘H’, ‘S’ and ‘M’ for our reference now printing the top 20 rows of our data set.



```r
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
```

Now printing the structure of our data set including number of entries or rows and number of features or columns and the data type of each column.

```
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 ...
```

Our dataset contains '338' rows and ' 4' columns and the datatype of each column is numeric.

Getting more details and basic statistics of our data set like mean, median, min and max etc.

```
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  
```

PREPROCESSING

After knowing the basic knowledge of our dataset, we now perform data preprocessing in R by extracting, normalizing, and combining feature variables with a target variable into a new data frame.

Once we are done with the preprocessing, now we print the dimensions of our normalized dataset.

```
dim(normalized_data)
```

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

From this dataset we have 4 features they are 'V', 'H', 'S' and 'M' were 'V' is voltage , 'H' is high, 'S' is soil type and 'M' is mine type.

```
colnames(normalized_data)
```

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

From the above column we have chosen 'M' mine type as our predictor column and now fetching the unique elements in 'M'.

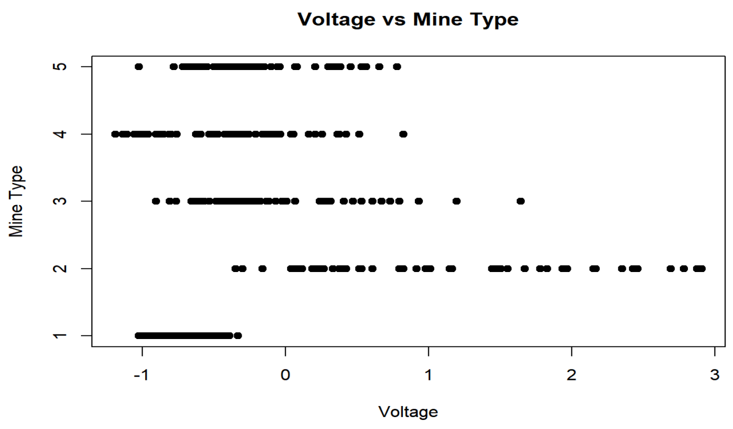
```
unique(normalized_data$M)
```

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

From the above results we came to know that we have '1','2','3','4' and '5' as unique elements in 'M'. (5 output labels).

VISUALIZATION

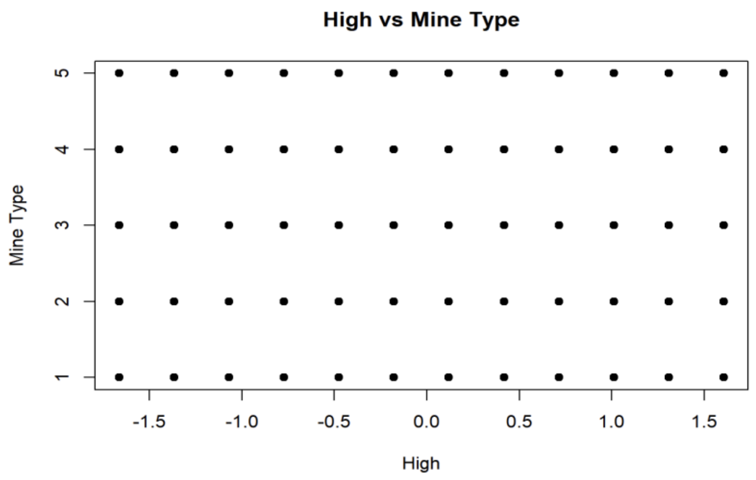
Finding the relation between the target column and predictor features. First we find the relation between voltage and mine type from the normalized dataset.



Voltage vs Mine Type

From the above graph the result is gives the scatter plot between the 'V' and 'M'.

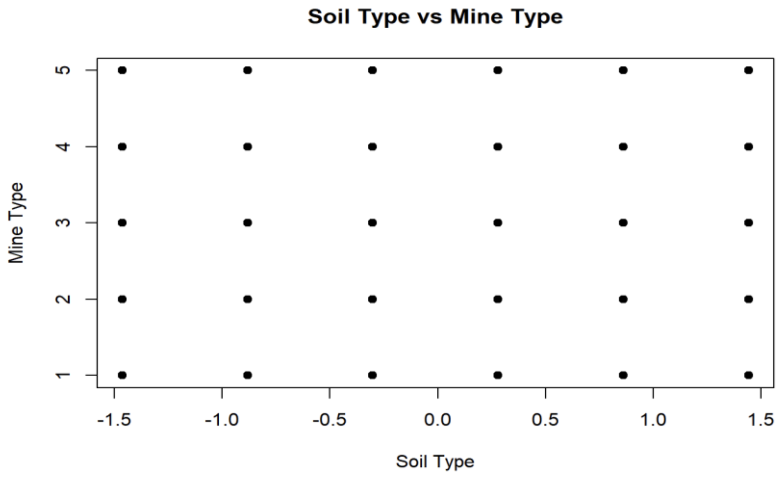
Now we find the relation between high and mine type from the normalized dataset.



High vs Mine Type

From the above graph the result gives the scatter plot between the 'H' and 'M'.

Now we find the relation between soil type and mine type from the normalized dataset.



Soil Type vs Mine Type

From the above graph result is gives the scatter plot between the 'S' and 'M'. from all the above results it looks like the relation between soil type and high with mine type is quite similar.

CORRELATION MATRIX

Now we will calculate the correlation matrix of the columns in normalized_data.

This means that for each pair of columns in normalized_data, the function will calculate the correlation coefficient between them, which measures the strength of the linear relationship between the two variables. The resulting correlation matrix will be a square matrix where each entry (i,j) represents the correlation coefficient between the i-th and j-th columns of normalized_data.

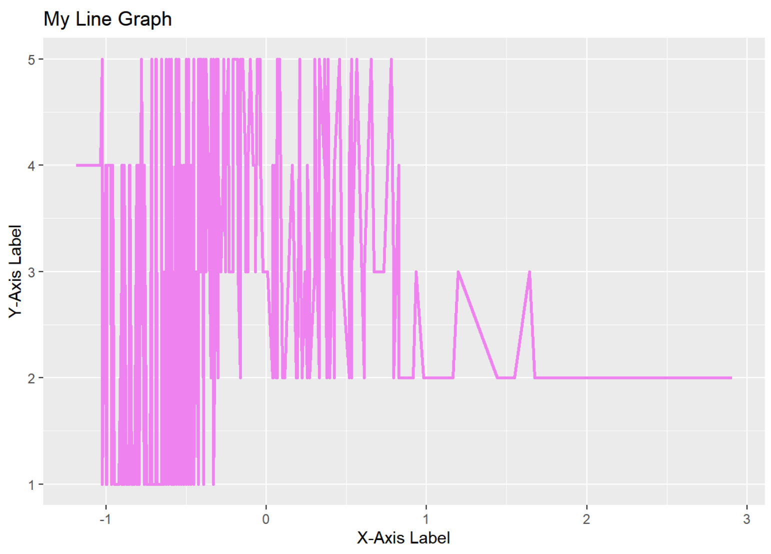
The values in the correlation matrix will range between -1 and 1, where a value of 1 indicates a perfect positive correlation between two variables, 0 indicates no correlation, and -1 indicates a perfect negative correlation.

```
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

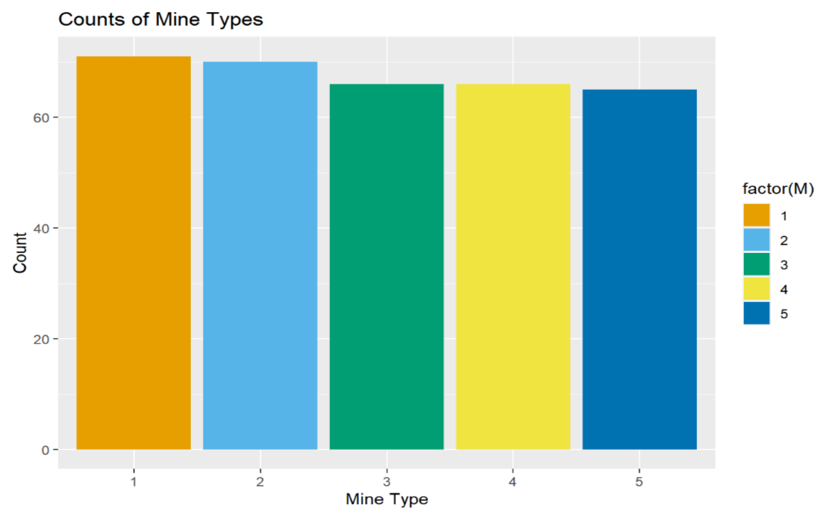
Now we plot the distribution of voltage over the mine type column which gives the line graph of the V and M variables in my_data, with a violet line connecting the data points.



Voltage Type distribution

BAR CHART TO VISUALIZE DIFFERENT MINE TYPES

In this part we are going to count the number of mine types with each of their unique value of 'M'.

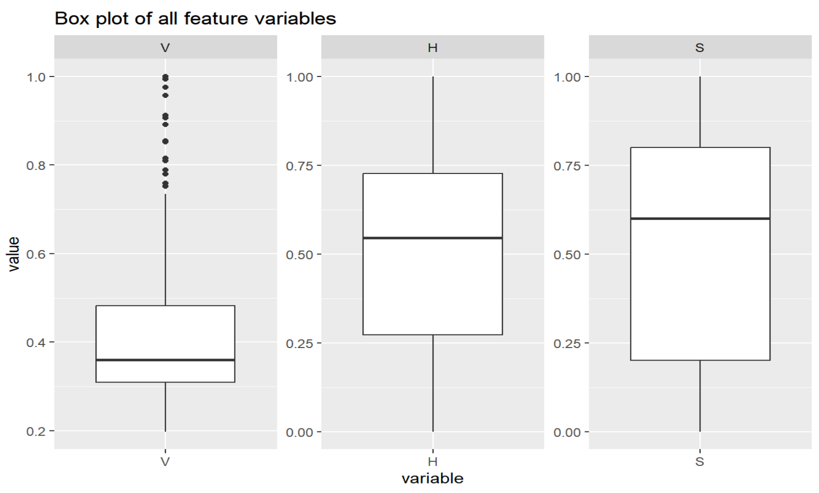


Count of each mine type

From the above graph '1' mine type have around 70 and mine type '2' have the count nearly 68 whereas the mine type '3' ,'4' and '5' have the similar count of 65.

BOXPLOT TO FIND THE OUTLIERS

Now from the normalized dataset we find the outliers of each column.



Boxplot of each variable

From the above graphical results, it is clear that voltage has very minimal outliers whereas the rest of the columns are clear from the outliers.

ALGORITHMS PERFORMED:

1.) K – MEANS CLUSTERING
2.) HIREARCHIAL CLUSTERING
3.) DECISION TREE ALGORITHM
4.) NEURAL NETWORKS
5.) K – NEAREST NEIGHBOURS (KNN)
6.) MULTINOMIAL LOGISTIC REGRESSION
7.) NAÏVE BAYES

1.)K- MEANS CLUSTERING

It is a popular unsupervised machine learning algorithm which works on the basic idea of grouping similar data points together and discovering underlying patterns for which we choose a fixed number of clusters in a dataset.

We first choose a target number of centroids needed in a dataset. Then every data point is allocated to each of the clusters through reducing in the cluster sum of squares. It then comes to a halt when there is no change in their values or the defined number of iterations have been achieved. After final reassignment we name the cluster as the final cluster.

```
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
```

- After splitting the data into test and train data we, perform K-Means clustering on it.

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

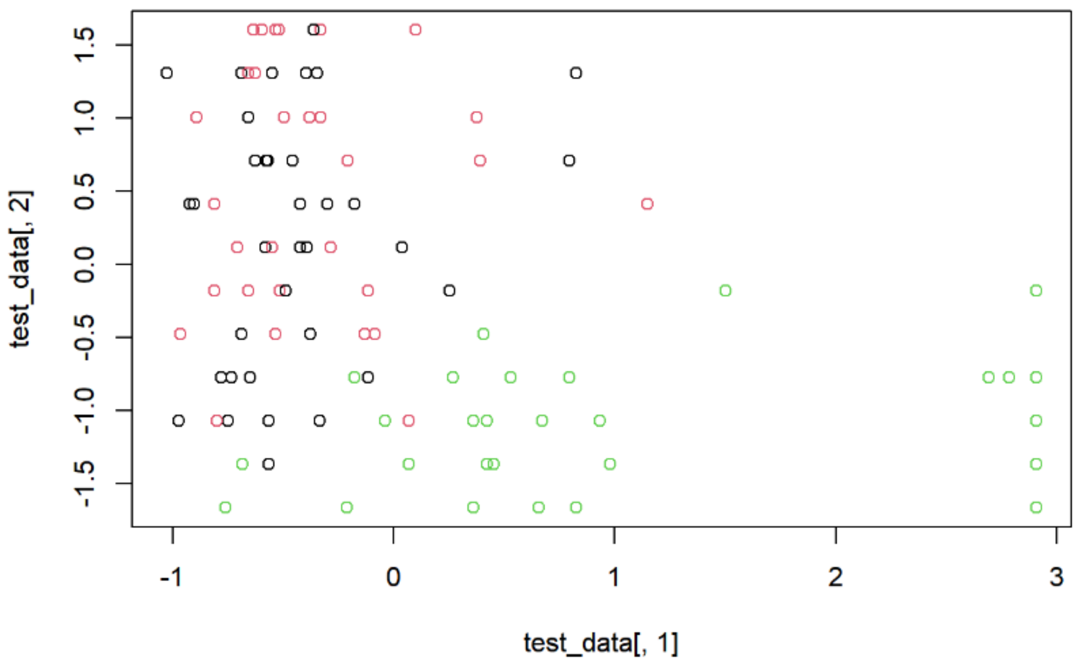
```
# Use the cluster centers to predict the clusters of the test data
test_clusters <- apply(test_data[, 1:3], 1, function(x) {
  # Calculate the distance from each point to each center
  distances <- apply(centers, 1, function(c) sqrt(sum((x - c)^2)))

  # 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
```

- After the data has been divided into clusters of similar types, we, print the clusters from the test data as the final clusters.

- We then plot a graph showing the division of data into clusters of test data 1 and test data 2

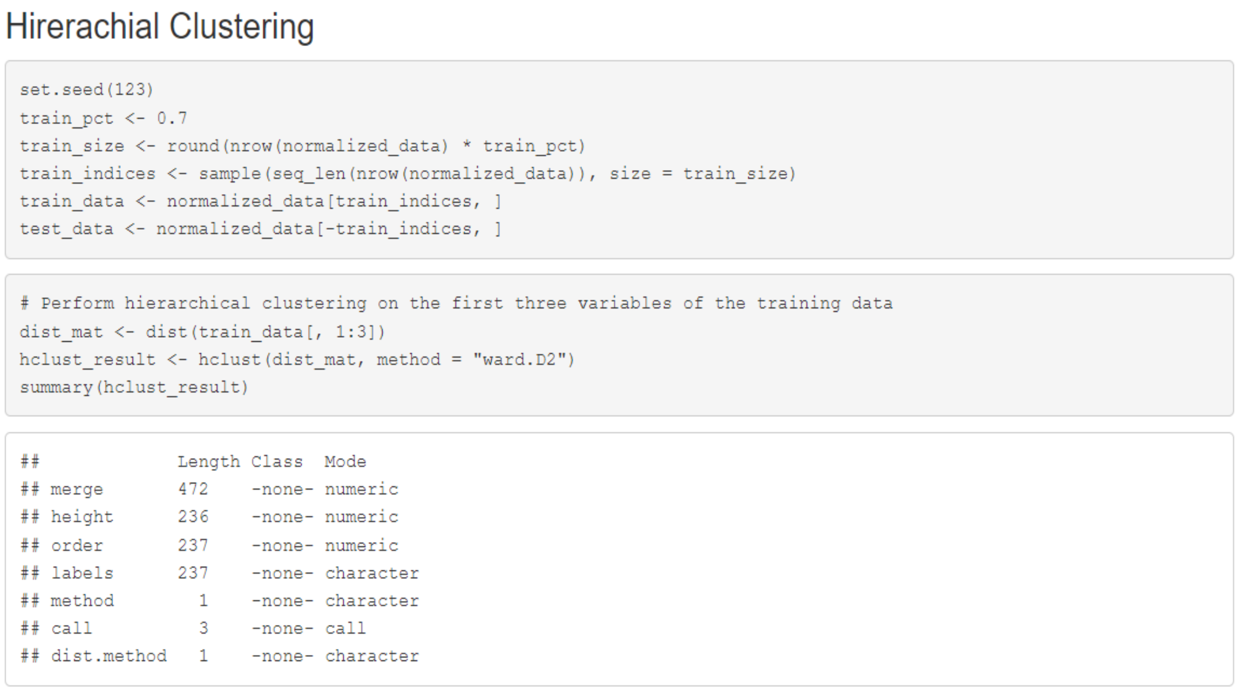
```
plot(test_data[,1], test_data[,2], col = test_clusters)
```



2.)HIREARCHIAL CLUSTERING

A clustering algorithm known as "hierarchical clustering" creates a hierarchy of clusters by repeatedly breaking up a collection of objects into smaller subsets until distinct clusters are produced. It can be either agglomerative, in which case each object starts forming its own cluster and then all of the clusters are combined, or divisive, in which case all of the objects begin in the same cluster and are then separated repeatedly. Dendrograms, which show the hierarchical links between the clusters, are one way to visualize the results of hierarchical clustering.

On the other hand, K-means clustering is a partitioning technique that seeks to group a set of objects into K clusters while minimizing the sum of squares within each cluster. It operates by updating the cluster centers after iteratively allocating each object to its nearest cluster center.



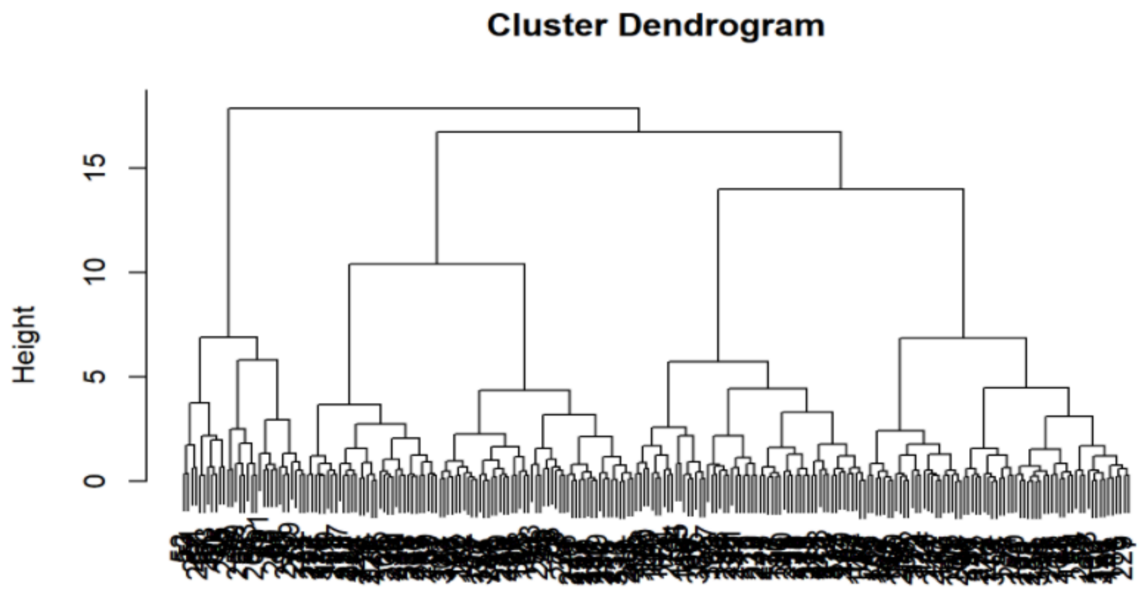
Hirerachial Clustering

```r
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, ]
```

```r
# 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
```

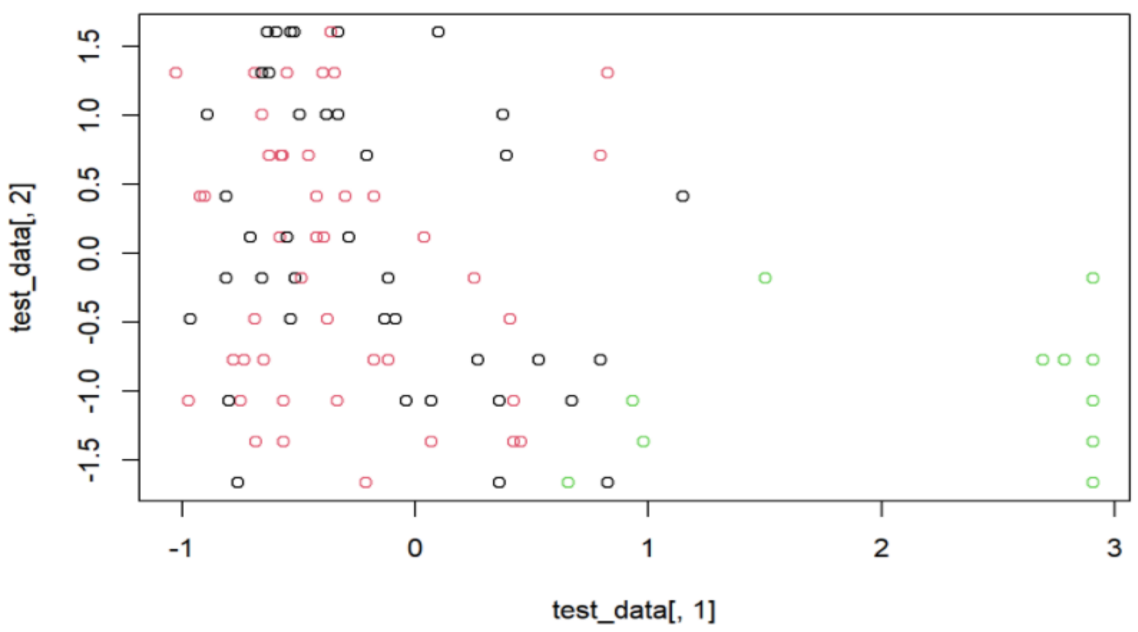
- First, we have randomly split the data into training and testing datasets.
- Then we performed hierarchical clustering on the first three variables of the training data using **Ward’s** method.



- We plotted the resulting dendrogram to visualize the cluster structure.
- Then we cut the dendrogram into 3 clusters and extract the cluster labels for the training data.

```
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)
```

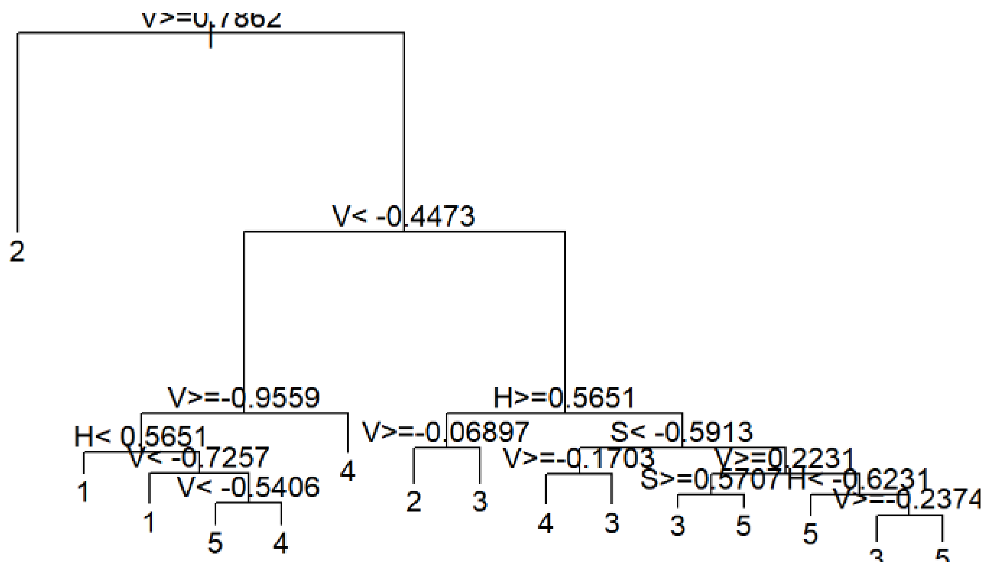
```
##   2   3  12  15  18  19  28  38  44  45  47  49  50  56  58  62  65  68  71  73
##   2   2   1   1   2   2   1   2   1   1   1   3   3   3   3   1   3   2   3   3
##  80  82  87  92  95  96  99 100 103 112 119 120 122 123 124 126 128 130 132 133
##   3   3   3   1   2   2   2   2   3   2   1   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
```



3.)DECISION TREE ALGORITHM

Decision trees have the ability to perform classification tasks. They are characterized by nodes and branches where the tests on each attribute are represented at the nodes and the outcome is represented at the branches and class labels are represented at the leaf nodes. Hence it uses a tree-like model based on various decisions that are used to compute their probable outcomes.

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

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



```r
# 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
```

4.)NEURAL NETWORKS

A Neural network is an information processing machine which is similar to human nervous system containing interconnected information processing units. They are used due to their power to parallel process the data.

- The data is first split into training and testing data after normalizing it

```
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
```

- After making predictions we have taken the accuracy of the predictions made by this model on the testing data and have got the following

```
# 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
```

- We have achieved an accuracy of 0.15686
- Through this we have deduced that neural network model isn't optimal for this data set.

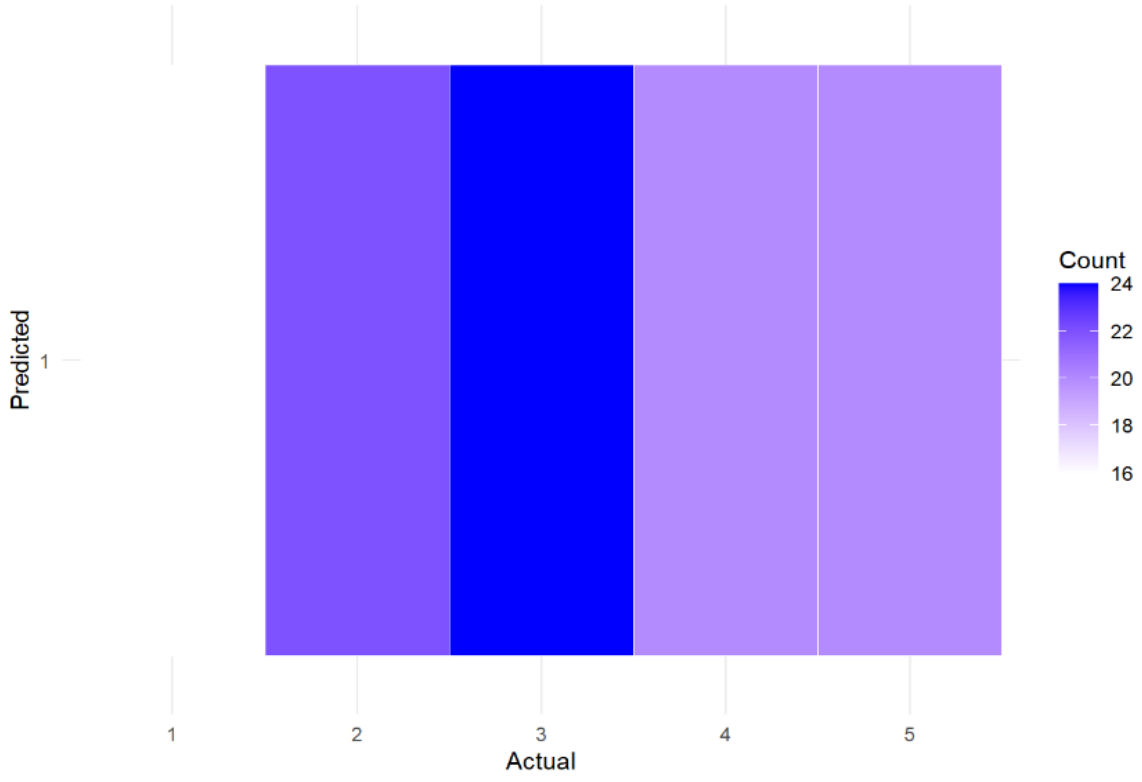
- Drawing a confusion matrix to analyze the outcomes.

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")
```



5.)K-NEAREST NEIGHBOUR [KNN]

It is a supervised machine learning algorithm which classifies a new data point into a target class based on the features of the neighboring data points. Which means that KNN checks how similar a data point is to neighbor or any other classes of data points, identifies which it is most similar to and classifies it into it.

- First, we split the data into training and testing data and select the value of k which is number of neighbors to consider

```r
# 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
```

- Then after we have trained the model, we use the testing data to calculate the accuracy of the model

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

```
## Accuracy: 0.44
```

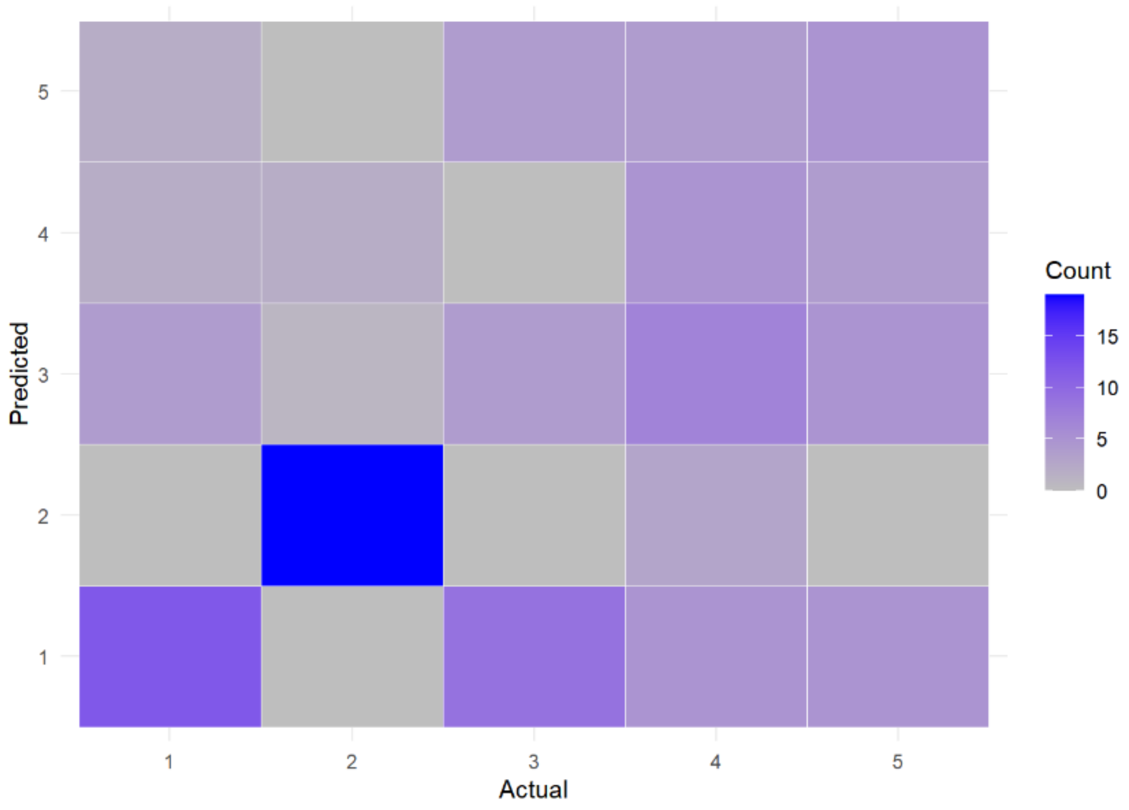
- We got an accuracy of 44% using KNN Model.
- Drawing a confusion matrix to analyze the outcomes.

Confusion Matrix

```r
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
```

```r
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")
```



6.)MULTINOMIAL LOGISTIC REGRESSION

It is used to predict nominal target variable. Generally, this algorithm is used when the dependent variable is nominal of more than two levels. Used to describe data and explain relation of one nominal variable and a minimum of one continuous independent variable.

```
# 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
```

- Here is the summary of the model

```
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
```

- Evaluating the performance of the model on the test data and taking the accuracy.

```
# 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
```

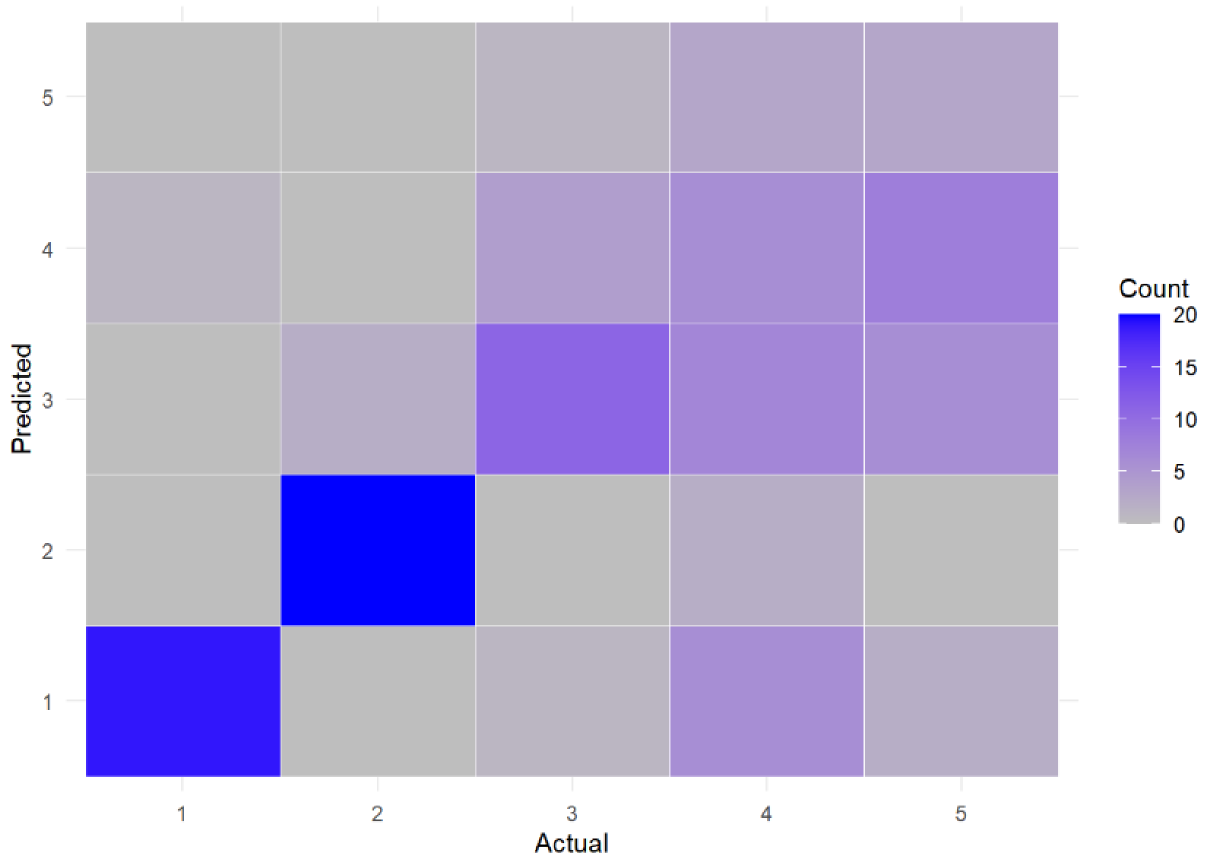
- As we can see the accuracy, we got is 57.8%
- Drawing a confusion matrix to analyze the outcomes.

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")
```



7.)NAÏVE BAYES

Naive bayes is a supervised classification algorithm based on Bayes theorem which gives us the probability of an event A given that an event B occurred.

It is called Naïve as it Assumes that the occurrence of a certain feature is independent of the occurrence of other features.

```
# 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
```

- Evaluating the performance of the model on the test data and taking the accuracy.

```
# 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
```

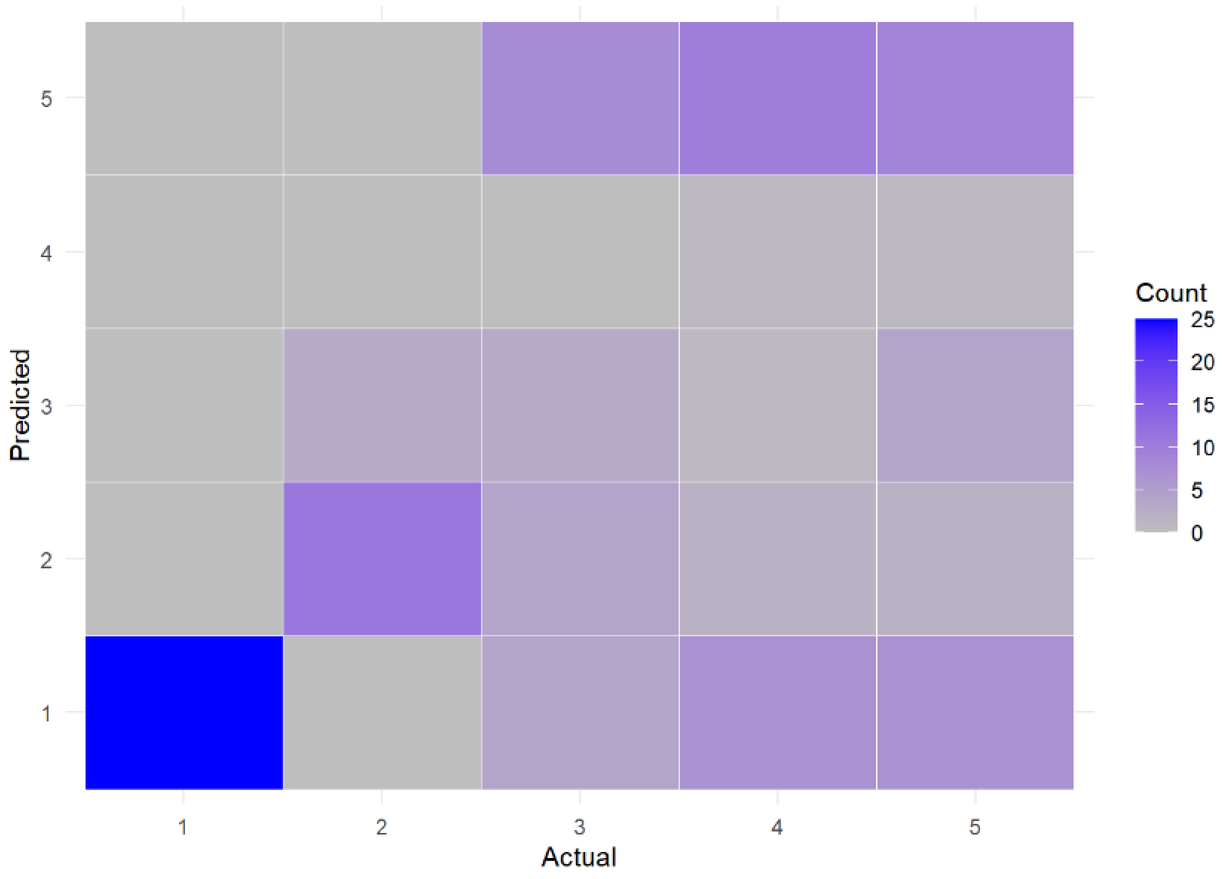
- The accuracy we got is 48.03%

- Drawing a confusion matrix to analyze the outcomes.

```r
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
```

```r
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")
```



Conclusion:

From the correlation matrix of the dataset, we can clearly observe that that output variable is not very much dependent on all the input features therefore, we are unable to attain great accuracies using the given “mine_dataset”. We obtained best accuracy while performing Multinomial Logistic Regression.