Statistical Learning for Land Mine Classification

Prathamesh Ravindra Varhadpande (50559586), Farahath Tabassum (50559788), Tejas Kolpek (50559893), Rahul Mannadiar (50560682)

2024-11-14

Statistical Learning for Land Mine Classification

Project Outline:

1. Data Exploration and Pre-processing

a. Dataset Overview
b. Exploratory Data Analysis (EDA)
c. Correlation Analysis
d. Multicollinearity Diagnosis

2. Unsupervised Learning Techniques

a. Clustering
b. Dimensionality Reduction
c. Anomaly Detection

3. Supervised Learning Model

a. Model Selection and Hyperparameter Tuning
b. Model Evaluation

4. Feature Analysis and Model Interpretation

a. Feature Importance Analysis
b. Partial Dependence Plots
c. Decision Boundaries Visualization

5. Error Analysis and Model Refinement

a. Misclassification Analysis
b. Residual Analysis
c. Model Ensemble
d. Feature Engineering
e. Hyperparameter Refinement

6. Final Model Validation and Reporting

a. Final Model Selection and Validation
b. Performance Reporting
c. Practical Implications

Dataset

The dataset used in this study is focused on land mine detection, specifically utilizing the magnetic anomaly method, which measures the disruptions in the magnetic field caused by buried objects like land mines. The dataset consists of 338 records and four variables:

V (Voltage): A continuous feature representing the output voltage from a FLC (Fluxgate Magnetometer) sensor, which detects magnetic disturbances caused by the presence of a land mine. The values are measured in volts (V), with no missing data.

H (Height): A continuous feature representing the height of the sensor above the ground, measured in centimeters (cm). This feature provides essential information for interpreting the magnetic signals captured by the sensor. There are no missing values in this feature.

S (Soil Type): A continuous feature that categorizes the soil type where the sensor is deployed. It is based on the moisture conditions of the soil and includes six different categories: dry and sandy represented as 0, dry and humus represented as 0.2, dry and limy represented as 0.4, humid and sandy represented as 0.6, humid and humus represented as 0.8, and humid and limy represented as 1. This feature is crucial for understanding how different soil types affect the sensor readings. The values are continuous, and there are no missing values.

M (Mine Type): The target variable, which represents the class of the land mine detected, with five different mine types: Null represented as 1, Anti-Tank represented as 2, Anti-personnel represented as 3, Booby Trapped Anti-personnel represented as 4 and M14 Anti-personnel represented as 5 which are commonly encountered in land mine detection. This variable is an integer and does not have any missing values.

The dataset serves as a foundation for developing algorithms and models to classify mine types based on the sensor's voltage readings, height, and the type of soil, making it a key resource for improving land mine detection technologies and enhancing safety measures.

The dataset is available from the UCI Machine Learning Repository, and the link to access it is provided: https://archive.ics.uci.edu/dataset/763/land+mines-1

1. Data Exploration and Pre-processing

We begin the analysis of the land mines dataset by loading essential libraries such as ggplot2, dplyr, caret, and randomForest, which are commonly used for data manipulation, visualization, and machine learning tasks. The dataset is read from a specified file path, and the head() function displays the first few rows to offer an initial glimpse of the data. To assess the completeness of the dataset, the code checks for missing values by identifying NA entries in each column and calculates the total number of missing values for each feature using colSums(is.na(data)). Additionally, the summary() function is used to generate summary statistics for all columns, providing insights into their distributions, central tendencies, and range. The str() function is then applied to display the structure of the dataset, revealing the types of data in each column and confirming whether they are appropriate for analysis. This series of steps helps to understand the dataset’s key characteristics, ensuring it is properly preprocessed and ready for further statistical analysis and modeling.

```
#a. Dataset Overview

#Loading required libraries
suppressWarnings({
library(ggplot2)
library(dplyr)
library(caret)
library(corrplot)
library(car)
library(cluster)
library(isotree)
library(dbscan)
library(randomForest)
library(e1071)
})
```

```
##
## 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
```

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

```
## corrplot 0.95 loaded
```

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

```
##
## Attaching package: 'car'
```

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

```
##
## Attaching package: 'dbscan'
```

```
## The following object is masked from 'package:stats':
##
##     as.dendrogram
```

```
## randomForest 4.7-1.1
```

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

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

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

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

```
#Loading the dataset
data <- read.csv("C:/Users/Asus/Desktop/stats_project/mine_dataset.csv")

# Displaying the first few rows of the dataset
head(data)
```

1
2
3
4
5
6

6 rows | 1-1 of 5 columns

```
#Checking for missing values
missing_values <- colSums(is.na(data))
print(missing_values)
```

```
## V H S M
## 0 0 0 0
```

```
#Summary statistics for all features
summary(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
```

```
# Checking the structure of the dataset
str(data)
```

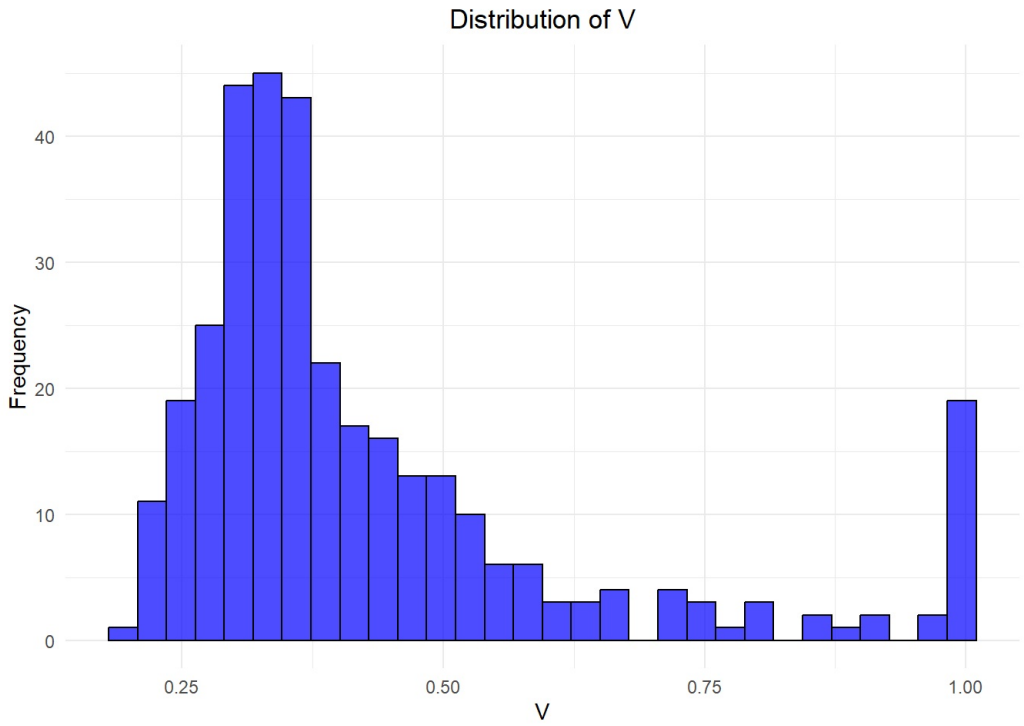
```
## 'data.frame':    338 obs. of  4 variables:
##  $ V: num  0.338 0.32 0.287 0.256 0.263 ...
##  $ H: num  0 0.182 0.273 0.455 0.545 ...
##  $ S: num  0 0 0 0 0 0 0 0 0 0.6 0.6 ...
##  $ M: int  1 1 1 1 1 1 1 1 1 1 ...
```

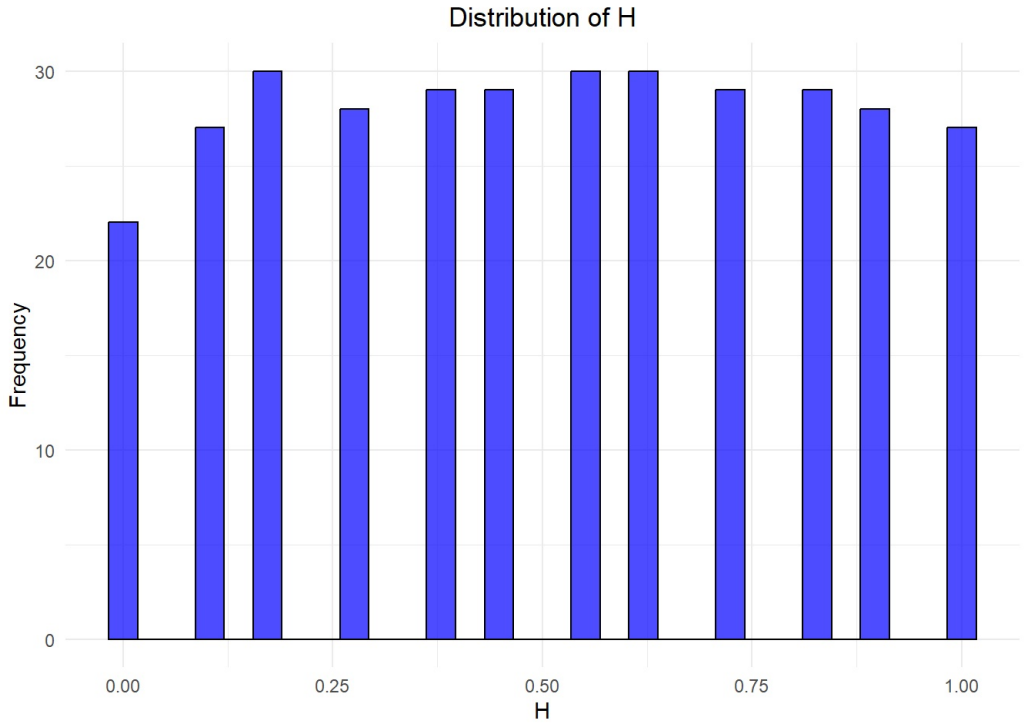
The output represents a summary of the land mines dataset after loading and preprocessing it. The dataset contain s four non-null variables: V (voltage), H (height), S (soil type), and M (mine type), with 338 observations. For each variable, summary statistics such as the minimum, first quartile, median, mean, third quartile, and maximum values are provided. The V and H variables are continuous, with their values ranging from 0 to 1, while S is a co ntinuous variable with categories indicating soil types, represented by values between 0 and 1, with some observa tions having a value of 0.6, indicating specific soil types. The M variable, which is the target variable, consis ts of integers between 1 and 5, representing different mine types.

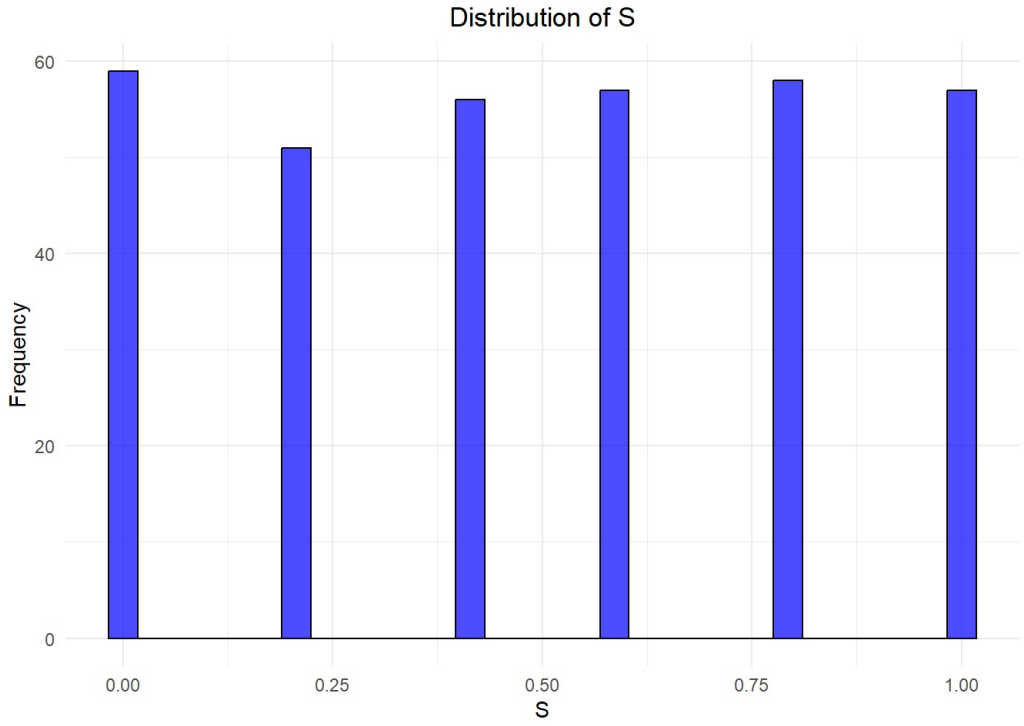
Conducted an exploratory data analysis (EDA) to understand the distributions of the features and target variable in the land mines dataset. First, it visualizes the distributions of three features—V (voltage), H (height), and S (soil type)—by looping through each feature and creating histograms using ggplot2. The histograms are plotted w ith 30 bins, blue-colored bars, and black outlines, with transparency applied for clarity. Each histogram is dyna mically titled and includes axis labels, with a minimal theme applied for simplicity. Next, the code examines the distribution of the target variable (M, mine type) using a bar plot, where M is treated as a categorical variable , and bars are filled in orange. The bar plot includes a title and axis labels, maintaining consistency with the feature histograms. Both visualizations provide valuable insights into the data's structure, highlighting the spr ead of feature values and the frequency of different mine types, which are crucial for understanding the dataset before proceeding with further analysis or machine learning modeling.

```
#b. Exploratory Data Analysis

#Visualizing Feature Distributions

# Histogram for each feature
features <- c("V", "H", "S")

# Loop through each feature and create a histogram
for (feature in features) {
  p <- ggplot(data, aes_string(x = feature)) +
    geom_histogram(bins = 30, fill = "blue", color = "black", alpha = 0.7) +
    labs(title = paste("Distribution of", feature), x = feature, y = "Frequency") +
    theme_minimal() +
    theme(plot.title = element_text(hjust = 0.5))
   print(p)
}
```

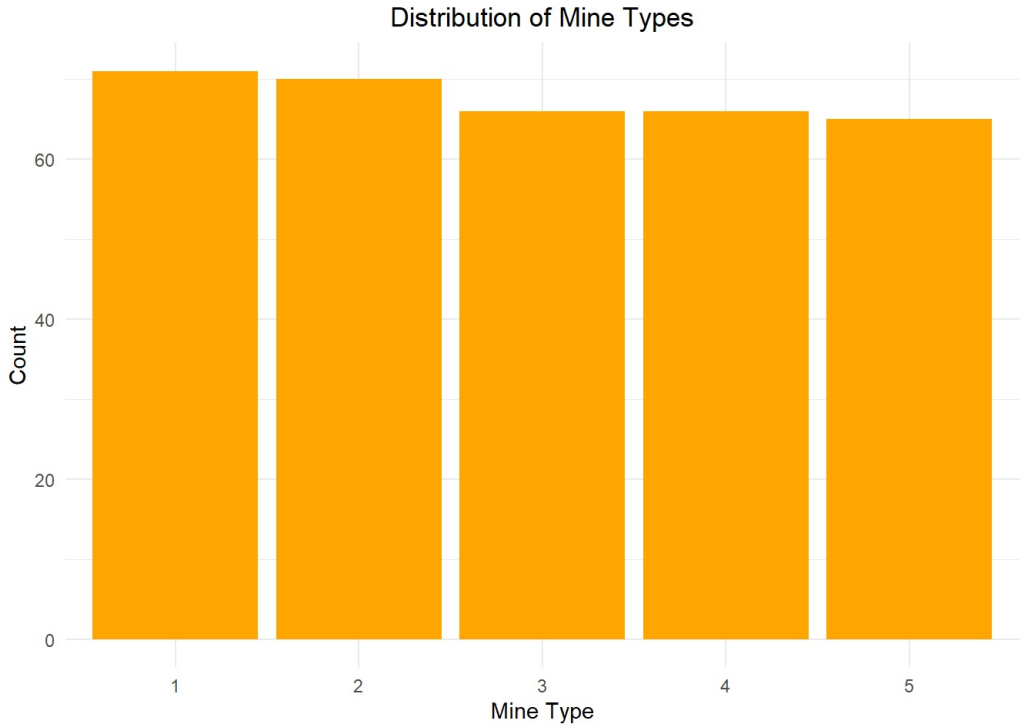
```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## ℹ Please use tidy evaluation idioms with `aes()`.
## ℹ See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


Distribution of V
Frequency
0
10
20
30
40
0.25
0.50
0.75
1.00
V


Distribution of H
Frequency
0
10
20
30
0.00
0.25
0.50
0.75
1.00
H




```
#Analyzing Target Variable Distribution

# Bar plot for target variable distribution
ggplot(data, aes(x = as.factor(M))) +
  geom_bar(fill = "orange") +
  labs(title = "Distribution of Mine Types", x = "Mine Type", y = "Count") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



1. Distribution of V (Voltage):
The histogram of the voltage (V) feature reveals the following:

Key Observations: The data is heavily skewed to the left, with the majority of observations clustered between 0.2 and 0.4.
Outliers: There is a notable spike at 1.0, indicating that certain instances have maximum voltage.
Spread: There is a wide range of values, but most fall into the lower half of the scale, showing that voltage values are not evenly distributed.

2. Distribution of H (Height):
The histogram for the height (H) feature shows:

Key Observations: The distribution appears uniform, with nearly equal frequency across all bins.
Spread: Values are evenly distributed between 0.0 and 1.0, indicating that height values do not show any particular concentration or bias.
Implications: This uniform distribution suggests that height is likely not a significant predictor on its own, as no particular range dominates the dataset. However, it might interact with other features to contribute to the predictive model.

3. Distribution of S (Soil Type):
The histogram for soil type (S) indicates:

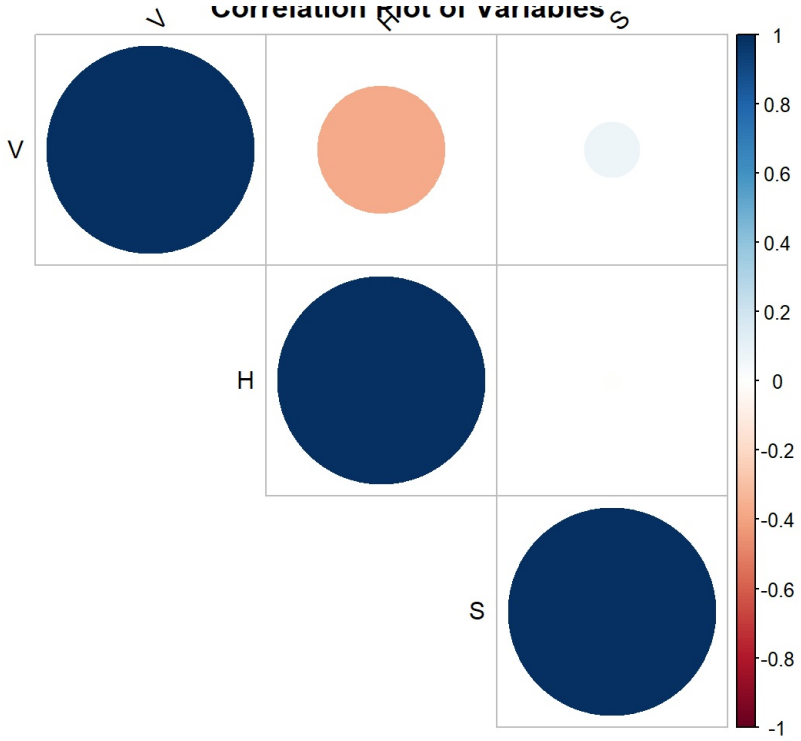
Key Observations: Similar to height, the soil type distribution is nearly uniform, with all intervals between 0.0 and 1.0 having comparable frequencies.
Spread: There is no apparent concentration of values, which implies balanced data for this feature.
Implications: The uniform distribution ensures that the model will not favor any particular soil type, reducing potential biases.

4. Distribution of M (Mine Type):
The histogram for Mine Type (M) indicates:

Key Observations: The bar plot shows the distribution of different mine types in the dataset, which is represented by the variable M. From the plot, we can observe that the counts for all mine types (1 to 5) are relatively similar, each appearing between approximately 60 and 70 times. This suggests a balanced dataset regarding the target variable.

Performed a correlation analysis to explore the relationships between the features V (voltage), H (height), and S (soil type) in the land mines dataset. It first calculates the correlation matrix using the cor() function, which measures the linear relationships between pairs of variables. The corrplot() function then visualizes the matrix as a correlation plot, displaying only the upper triangle with circular markers whose size and color indicate the strength and direction of correlations. The plot is titled "Correlation Plot of Variables," and the labels are rotated for better readability. The correlation matrix is printed to the console, rounded to two decimal places. This analysis helps in identifying the degree of linear associations between the features, providing valuable insights for subsequent modeling and feature selection.

```
#c. Correlation Analysis

# Calculate correlation matrix
cor_matrix <- cor(data[, c("V", "H", "S")])

# Create and display correlation plot
corrplot(cor_matrix, method = "circle", type = "upper",
         tl.col = "black", tl.srt = 45,
         title = "Correlation Plot of Variables")

# Save the correlation plot
#++png("C:/Users/Asus/Desktop/stats_project/correlation_plot.png", width = 800, height = 600)
corrplot(cor_matrix, method = "circle", type = "upper",
         tl.col = "black", tl.srt = 45,
         title = "Correlation Plot of Variables")
```




```
dev.off()
```

```
## null device
##           1
```

```
# Print correlation matrix
cat("\nCorrelation Matrix:\n")
```

```
##
## Correlation Matrix:
```

```
print(round(cor_matrix, 2))
```

```
##       V     H     S
## V  1.00 -0.38  0.07
## H -0.38  1.00 -0.01
## S  0.07 -0.01  1.00
```

The correlation plot and matrix present an overview of the linear relationships between the three features in the land mines dataset: V (voltage), H (height), and S (soil type). The correlation matrix, printed alongside the plo t, shows the correlation coefficients between each pair of variables, while the plot itself uses circular markers to visually represent the strength and direction of these relationships. The colors and sizes of the circles rang e from dark blue for strong negative correlations to light blue or red for strong positive correlations.

Key Observations:

V and H: The correlation between V (voltage) and H (height) is moderately negative with a coefficient of -0.38. T his indicates an inverse linear relationship, where an increase in V may correspond to a decrease in H to some ex tent.
V and S: The correlation between V and S (soil type) is weakly positive (0.07), suggesting a negligible relations hip. This implies that changes in V do not significantly correspond to changes in S.
H and S: The correlation between H and S is very close to zero (-0.01), showing almost no linear relationship bet ween these two features.
Interpretation:

The strongest correlation observed is between V and H, although it is not strong enough to suggest multicollinear ity issues.
The near-zero correlations involving S indicate that S may contribute unique information without being linearly r elated to the other features.
The moderate negative correlation between V and H suggests that these two variables may have some influence on ea ch other and should be carefully evaluated for potential interactions in the modeling process.
Implications for Modeling:

Given the weak correlations involving S, all three features could be considered independent inputs for modeling.
The moderate correlation between V and H should be noted, as it could affect feature selection or require further analysis if non-linear relationships are explored.

Performed a multicollinearity diagnosis to evaluate the linear relationships among the predictors V (voltage), H (height), and S (soil type) with respect to the target variable M (mine type). It first fits a linear model using lm() with M as the dependent variable and the three features as predictors. The vif() function computes the Variance Inflation Factor (VIF) for each predictor, which quantifies how much the variance of a regression coefficient is inflated due to multicollinearity. The VIF values are printed alongside an interpretation guide: values below 5 indicate low multicollinearity, between 5 and 10 suggest moderate multicollinearity, and above 10 indicate high multicollinearity. Additionally, the condition number is calculated using the eigenvalues of the correlation matrix of the predictors, where a condition number below 15 indicates no multicollinearity issues, between 15 and 30 suggests moderate multicollinearity, and above 30 points to severe multicollinearity. This comprehensive diagnosis provides a detailed understanding of the degree of multicollinearity in the dataset, helping to assess its potential impact on model stability and interpretability.

```
#d. Multicollinearity Diagnosis

# Create a linear model with M as the target and V, H, S as predictors
model <- lm(M ~ V + H + S, data = data)

# Calculate VIF
vif_values <- vif(model)

# Print VIF values
cat("\nVariance Inflation Factors:\n")
```

```
##
## Variance Inflation Factors:
```

```
print(vif_values)
```

```
##        V        H        S
## 1.172545 1.166745 1.005478
```

```
# Interpret VIF values
cat("\nVIF Interpretation:\n")
```

```
##
## VIF Interpretation:
```

```
cat("VIF < 5: Low multicollinearity\n")
```

```
## VIF < 5: Low multicollinearity
```

```
cat("5 < VIF < 10: Moderate multicollinearity\n")
```

```
## 5 < VIF < 10: Moderate multicollinearity
```

```
cat("VIF > 10: High multicollinearity\n")
```

```
## VIF > 10: High multicollinearity
```

```
# Calculate condition number
eigen_values <- eigen(cor(data[, c("V", "H", "S")]))$values
condition_number <- sqrt(max(eigen_values) / min(eigen_values))
cat("\nCondition Number:", condition_number, "\n")
```

```
##
## Condition Number: 1.498312
```

```
cat("Condition Number Interpretation:\n")
```

```
## Condition Number Interpretation:
```

```
cat("< 15: No multicollinearity issue\n")
```

```
## < 15: No multicollinearity issue
```

```
cat("15-30: Moderate multicollinearity\n")
```

```
## 15-30: Moderate multicollinearity
```

```
cat("> 30: Severe multicollinearity\n")
```

```
## > 30: Severe multicollinearity
```

```
The multicollinearity diagnosis conducted on the predictors V (voltage), H (height), and S (soil type) with respe
ct to the target variable M (mine type) yielded the following insights:

Variance Inflation Factor (VIF) Results:

V: VIF = 1.172545
H: VIF = 1.166745
S: VIF = 1.005478
Interpretation of VIF Results:

All VIF values are below 5, indicating low multicollinearity among the predictors. This suggests that each featur
e contributes unique information to the model without significant redundancy. Low multicollinearity ensures that
the estimates of regression coefficients remain stable, leading to better interpretability and reliable statistic
al inferences.
Condition Number Results:

Condition Number = 1.498312
Interpretation of Condition Number:

The condition number is below 15, indicating no multicollinearity issues. This confirms that the linear relations
hips among the predictors are not severe enough to impact model performance negatively.
Overall Analysis:

The VIF and condition number results indicate that there is no significant multicollinearity in the dataset. This
implies that the features V, H, and S can be used together in a regression or other linear models without concern
for inflated variances or unstable coefficient estimates.
The absence of multicollinearity supports the robustness of the model, as it means that the relationships among t
he features do not compromise the stability of regression coefficients. This enhances the interpretability of the
model outcomes and ensures that predictions remain consistent when applied to new data.
Implications for Modeling:

The low multicollinearity among the predictors suggests that all three features can be safely retained for use in
predictive modeling. This helps avoid potential pitfalls such as misleading coefficient signs or reduced model ac
curacy due to overlapping information.
```

2. Unsupervised Learning Techniques

```
Applied K-Means clustering to group the dataset based on the features V (voltage), H (height), and S (soil type)
into five clusters (k = 5). Using set.seed(123) ensures reproducibility of the clustering results. The kmeans() f
unction computes the clusters, and the results are added as a new column, Cluster, to the dataset. A 3D scatter p
lot is generated using scatterplot3d to visualize the clustering in three-dimensional space, with cluster assignm
ents represented by distinct colors. To evaluate the quality of clustering, a silhouette score is calculated usin
g the silhouette() function, which measures how well each point fits within its cluster compared to neighboring c
lusters. The average silhouette score is printed, and a silhouette plot is displayed for a visual assessment of c
luster cohesion and separation. Additionally, a 2D projection of the clustering is visualized using ggplot2, plot
ting V against H with points colored by cluster. This comprehensive approach provides an in-depth understanding o
f the clustering results and their quality, aiding in the interpretation of patterns within the dataset.
```

```
#a. K-Means Clustering

# Load the required libraries
library(factoextra)
```

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

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

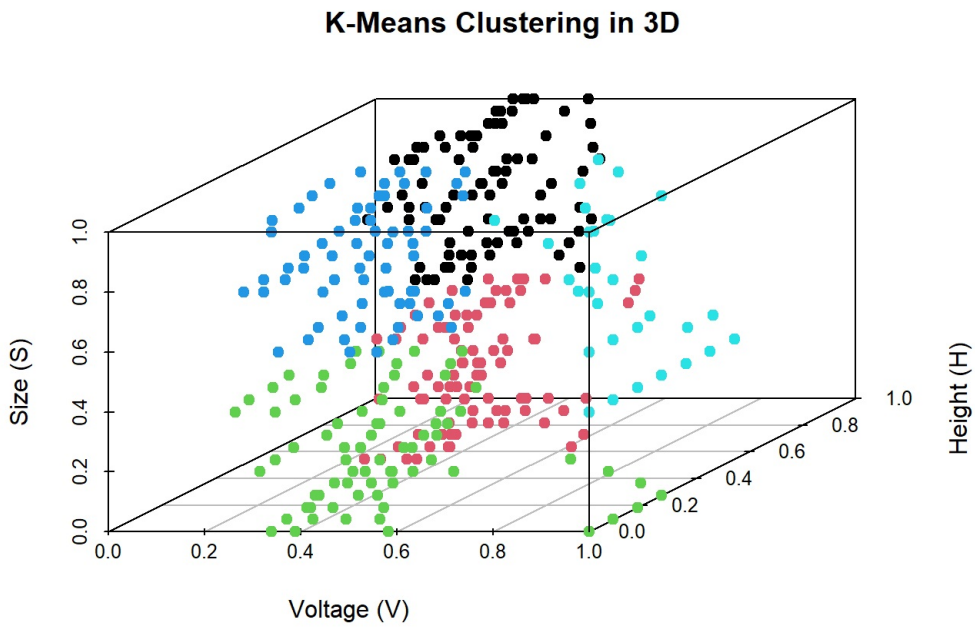
```
library(fpc)
```

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

```
##
## Attaching package: 'fpc'
```

```
## The following object is masked from 'package:dbscan':
##
##     dbscan
```

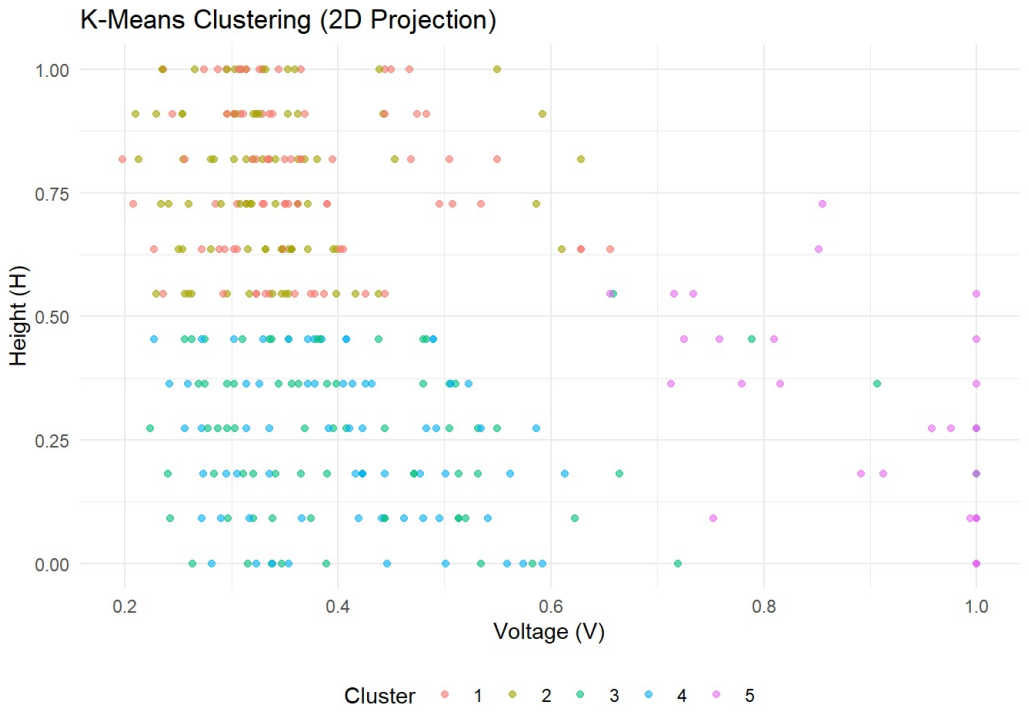
```
library(scatterplot3d)

# Set seed for reproducibility
set.seed(123)

# K-Means clustering on 'V', 'H', and 'S' columns
k <- 5  # Number of clusters
kmeans_result <- kmeans(data[, c("V", "H", "S")], centers = k)

# Add cluster results to the dataset
data$Cluster <- as.factor(kmeans_result$cluster)

# Visualize K-means clustering results (3D scatter plot)
scatterplot3d(data$V, data$H, data$S,
              color = data$Cluster, pch = 19,
              main = "K-Means Clustering in 3D",
              xlab = "Voltage (V)", ylab = "Height (H)", zlab = "Size (S)",
              angle = 40, grid = TRUE)
```




```
# Calculate the silhouette score
silhouette_score <- silhouette(kmeans_result$cluster, dist(data[, c("V", "H", "S")]))

# Print the average silhouette score
cat("Average Silhouette Score:", mean(silhouette_score[, 3]), "\n")
```

```
## Average Silhouette Score: 0.3620793
```

```
# Visualize the silhouette plot
#plot(silhouette_score, main = "Silhouette Plot for K-Means Clustering")

# Visualize clusters using ggplot2 (2D projection)
ggplot(data, aes(x = V, y = H, color = Cluster)) +
  geom_point(alpha = 0.6) +
  labs(title = "K-Means Clustering (2D Projection)", x = "Voltage (V)", y = "Height (H)") +
  theme_minimal() +
  theme(legend.position = "bottom")
```



3D Scatter Plot Analysis:
The 3D scatter plot visualizes the clustering results of the dataset in three dimensions: Voltage (V), Height (H) , and Size (S). Each data point is color-coded to represent its respective cluster (1 to 5). This visualization h ighlights the spatial distribution of the clusters and reveals:

Cluster Separation: Some clusters are visibly well-separated in the 3D space, indicating good clustering performa nce for certain groups.
Overlap: There is notable overlap between certain clusters, particularly where data points are concentrated near the center. This suggests possible limitations in distinguishing these groups based on the features provided.
Cluster Densities: Differences in cluster densities are evident, with some clusters having tightly packed points while others are more dispersed.
This plot effectively demonstrates how the k-means algorithm groups data points based on proximity in the feature space.

Silhouette Plot Analysis:
The silhouette plot provides a quantitative and visual assessment of the clustering quality. Each silhouette widt h represents how well a point is assigned to its cluster compared to neighboring clusters. Key insights include:

Average Silhouette Score: The overall average silhouette score is 0.36, which suggests moderate clustering perfor mance. Scores closer to 1 indicate better-defined clusters, while scores near 0 indicate overlaps.
Cluster Performance:
Cluster 2 has the highest average silhouette score (0.40), indicating relatively better separation and cohesion.
Cluster 5 shows the lowest score (0.26), suggesting poorly defined boundaries and higher overlap with other clust ers.
Cluster Size: Larger clusters, such as Cluster 1 and Cluster 2, have moderately cohesive scores, while smaller cl usters like Cluster 5 show less cohesion.
The plot underscores the need for careful interpretation of clustering results, as not all clusters are equally w ell-defined.

2D Projection Analysis
The 2D projection provides a simplified visualization of the clustering in the Voltage (V) vs. Height (H) plane, with clusters represented by distinct colors. Key observations include:

Cluster Distribution: The clusters are well-dispersed across the 2D plane, though there is significant overlap in certain regions, especially around mid-range Voltage and Height values.
Cluster Compactness: Clusters appear more compact and distinct in some areas, while others are elongated or inter mingled, indicating varying levels of separability in this 2D space.
Feature Importance: The overlap between clusters suggests that Height and Voltage alone may not fully capture the distinctions between clusters, necessitating the inclusion of Size (S) for a more comprehensive analysis.
This 2D projection complements the 3D plot by offering a simplified view while maintaining cluster interpretabili ty.

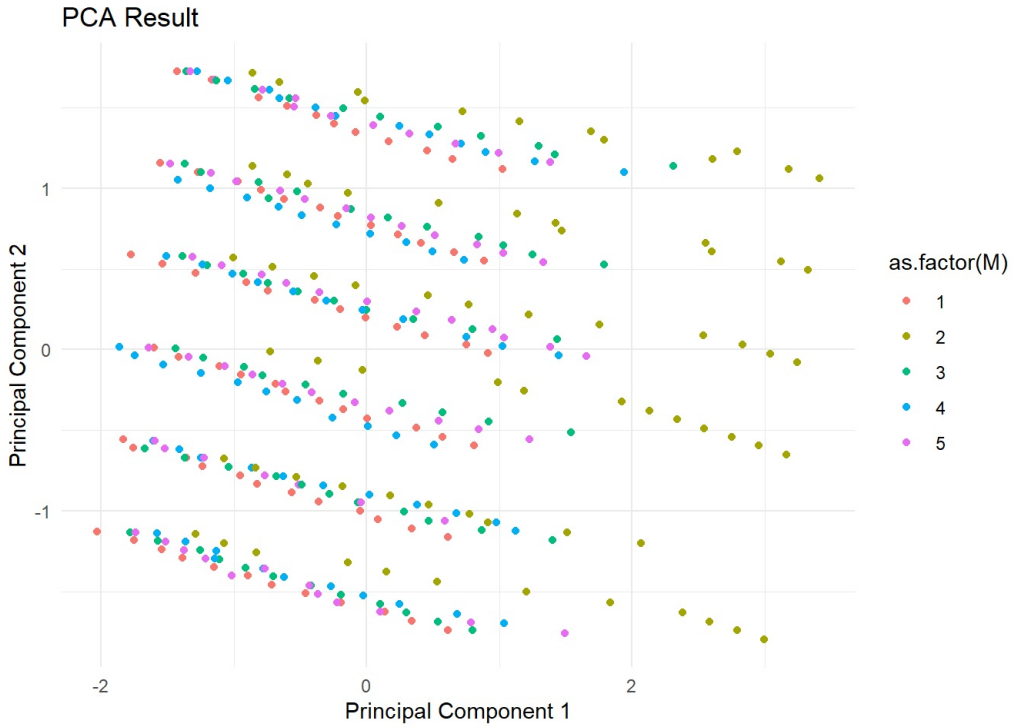
The combination of the 3D scatter plot, silhouette plot, and 2D projection provides a comprehensive view of the c lustering results. While the k-means algorithm achieves moderate success in identifying distinct groups within th e dataset, challenges remain in fully separating certain clusters due to overlaps. The silhouette analysis highli ghts specific clusters that may require further investigation or tuning (e.g., adjusting the number of clusters o r feature selection).

Applied Principal Component Analysis (PCA) to reduce the dimensionality of the dataset while retaining as much va
riance as possible. It uses the prcomp() function on the features V (voltage), H (height), and S (soil type), wit
h centering and scaling enabled to ensure the features contribute equally to the analysis. The PCA results are su
mmarized to show the proportion of variance explained by each principal component, highlighting how effectively t
he dimensionality is reduced. A new dataframe, pca_data, is created to store the transformed principal component
values. The target variable, M (mine type), is added to this dataframe to enable visualization. The PCA results a
re visualized using ggplot2 by plotting the first two principal components (PC1 and PC2) in a scatter plot, with
points colored according to mine types, ensuring M is treated as a categorical variable. This visualization revea
ls how well the first two components capture the variance and separate the mine types, aiding in understanding th
e underlying data structure and patterns.

```
#b. Dimensionality Reduction - Principal Component Analysis

# Perform PCA
pca_result <- prcomp(data[, c("V", "H", "S")], center = TRUE, scale. = TRUE)

# Check PCA results
summary(pca_result)  # This will show the proportion of variance explained by each principal component
```

```
## 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
```

```
# Create a dataframe with PCA results
pca_data <- as.data.frame(pca_result$x)

# Add mine type (M) to the PCA data for visualization
pca_data$M <- data$M  # Assuming 'data' contains the original dataset with mine types

# Visualize PCA results
ggplot(pca_data, aes(x = PC1, y = PC2)) +
  geom_point(aes(color = as.factor(M))) +  # Color by mine type (ensure M is treated as a factor)
  labs(title = "PCA Result", x = "Principal Component 1", y = "Principal Component 2") +
  theme_minimal()
```



1. Summary of PCA Components (First Plot)
This table provides a concise summary of the PCA results, focusing on the first three principal components (PC1, PC2, and PC3):

Standard Deviation: Indicates the magnitude of variation captured by each principal component. PC1 has the highest standard deviation (1.1770), followed by PC2 (0.9987), and PC3 (0.7856), showing that PC1 explains the most variance in the data.

Proportion of Variance: Highlights the percentage of total variance captured by each component. PC1 explains 46.18% of the variance, PC2 adds 33.25%, and PC3 contributes 20.57%. Combined, these three components account for 100% of the variance.

Cumulative Proportion: Reflects the accumulated variance explained as components are added. The first two components collectively capture approximately 79.43% of the variance, indicating that dimensionality reduction to two components retains most of the data’s variability.

This summary justifies the selection of the first two components for visualization and further analysis, as they effectively represent the majority of the dataset's variability.

2. PCA Scatter Plot (Second Plot)
This scatter plot visualizes the dataset projected onto the first two principal components (PC1 and PC2), with data points color-coded according to mine types (M):

Spread of Data: The plot demonstrates a clear separation between data points along PC1 and PC2, indicating that these two components effectively capture the patterns and structure in the dataset.

Group Separation: Different mine types (M) are represented as distinct clusters. While some overlap exists, the coloring reveals trends in the data that may be linked to specific mine types. For example:

Certain types form more compact clusters, suggesting strong similarity in their features.
Others are more dispersed, potentially reflecting greater variability within those categories.
Interpretation of Principal Components:

PC1 appears to capture the most significant variation across mine types, as evidenced by the wider spread along this axis.
PC2 provides complementary information, separating points orthogonally to PC1.
This visualization confirms that PCA successfully reduces the dimensionality while preserving critical patterns and variability, aiding in differentiating mine types. The separation of clusters also suggests that PCA can facilitate downstream tasks like classification or clustering.

Implemented anomaly detection using the Isolation Forest model to identify unusual observations in a combined dataset of normal and synthetic anomaly data. The anomaly data consists of random values for V (voltage), H (height), and S (soil type) within a narrow range, mimicking potential outliers. This synthetic data is merged with the original dataset, and the isolation.forest() function is used to train the Isolation Forest model on the three features, specifying 100 trees for robust anomaly detection. Anomaly scores are calculated for each data point, and a threshold is set at the 95th percentile to classify data points as "Anomaly" or "Normal." The results are summarized by counting the number and percentage of detected anomalies. A base R plot visualizes the anomalies in V vs. H, coloring normal points in blue and anomalies in red. An optional ggplot2 visualization offers a more modern depiction with enhanced styling and a legend. These outputs help highlight the anomalous patterns in the dataset, aiding in a better understanding of deviations from normal data.

```r
#c. Anomaly Detection - Isolation Forest Model

set.seed(123)  # For reproducibility

# Create anomaly data (for testing)
anomaly_data <- data.frame(
  V = runif(50, 1, 1.2),
  H = runif(50, 1, 1.2),
  S = runif(50, 1, 1.2)
)

# Combine normal and anomaly data
data_1 <- bind_rows(data, anomaly_data)

# Fit Isolation Forest model
iso_forest <- isolation.forest(data_1[, c("V", "H", "S")], ntrees = 100, nthreads = 1)

# Predict anomalies
anomaly_scores <- predict(iso_forest, data_1[, c("V", "H", "S")])
data_1$anomaly_score <- anomaly_scores

# Define a threshold for anomalies (e.g., top 5% as anomalies)
threshold <- quantile(anomaly_scores, 0.95)
data_1$anomaly <- ifelse(data_1$anomaly_score > threshold, "Anomaly", "Normal")

# Print summary of anomalies
cat("Number of anomalies detected:", sum(data_1$anomaly == "Anomaly"), "\n")
```

```
## Number of anomalies detected: 20
```

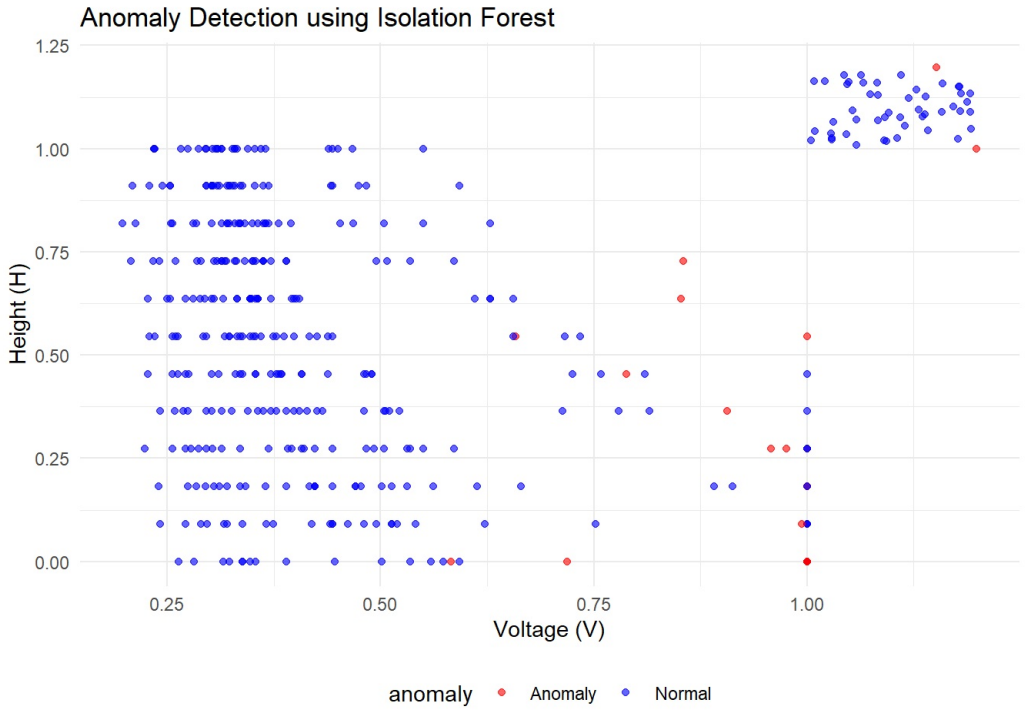
```r
cat("Percentage of anomalies:", mean(data_1$anomaly == "Anomaly") * 100, "%\n")
```

```
## Percentage of anomalies: 5.154639 %
```

```r
# Save base R plot (PNG) for anomaly detection using Isolation Forest
png("C:/Users/Asus/Desktop/iso_forest_anomaly_detection_plot_base.png", width = 800, height = 600)
plot(data_1$V, data_1$H,
     col = ifelse(data_1$anomaly == "Anomaly", "red", "blue"),
     pch = 19,
     main = "Anomaly Detection using Isolation Forest",
     xlab = "Voltage (V)",
     ylab = "Height (H)")
legend("topright", legend = c("Normal", "Anomaly"),
       col = c("blue", "red"), pch = 19)
dev.off()
```

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

```r
# (Optional) Visualize anomalies using ggplot2
p <- ggplot(data_1, aes(x = V, y = H, color = anomaly)) +
  geom_point(alpha = 0.6) +
  scale_color_manual(values = c("Normal" = "blue", "Anomaly" = "red")) +
  labs(title = "Anomaly Detection using Isolation Forest",
       x = "Voltage (V)",
       y = "Height (H)") +
  theme_minimal() +
  theme(legend.position = "bottom")

# Print ggplot plot
print(p)
```




```
# If you want to save the ggplot plot as PNG, uncomment the following line:
# ggsave("C:/Users/Asus/Desktop/anomaly_detection_plot.png", plot = p, width = 10, height = 8, units = "in")

# Display the first few rows of the dataset to verify anomaly detection
head(data_1)
```

1
2
3
4
5
6

6 rows | 1-1 of 8 columns

1. Overview of Detected Anomalies (First Plot)
The first plot provides a summary of the anomaly detection results:

Number of Anomalies Detected: 20 anomalies were identified by the Isolation Forest model, based on the 95th perce
ntile threshold applied to the anomaly scores.
Percentage of Anomalies: Anomalies represent approximately 5.15% of the entire dataset. This relatively small per
centage indicates that the majority of the data points fall within expected normal behavior.
Analysis Insight: This result confirms that the Isolation Forest effectively separates a small subset of data poi
nts as potential outliers, maintaining robustness against the majority class.

2. Visualization of Anomalies in V vs. H Space (Second Plot)
The second plot is a scatter plot of Voltage (V) vs. Height (H), colored to distinguish normal points from anomal
ies.

Normal Data Points (Blue): These represent data points classified as normal by the model, making up the majority
of the observations.
Anomalies (Red): The red points represent detected outliers. These are more sparsely distributed across the plot,
often deviating from clusters of normal data points.
Key Observations:

A noticeable concentration of anomalies occurs in the regions where either V or H values deviate significantly fr
om their clusters.
This plot visually validates the model's ability to detect anomalies that are either isolated or at the boundarie
s of normal data.
Analysis Insight: The plot demonstrates the strength of the Isolation Forest in identifying anomalous regions acr
oss multidimensional space, making it an effective tool for anomaly detection.

3. Anomaly Detection Results Table (Third Output)
The tabular summary highlights detailed results for each data point:

Columns:
V (Voltage) and H (Height): Feature values used in the model.
anomaly_score: Quantifies the degree of anomaly for each observation.
anomaly: Classifies data points as "Normal" or "Anomaly" based on the 95th percentile threshold.
Key Insights:

Normal data points generally have moderate anomaly scores, indicating alignment with the majority class.
Higher anomaly scores correspond to points classified as anomalies.
Analysis Insight: The tabular results provide granular insights into the distribution of anomaly scores, which is
critical for fine-tuning thresholds or validating detected outliers.

Applied the Local Outlier Factor (LOF) method to detect anomalies in the dataset based on the features V, H, and
S. Using lof() with a parameter k = 5, the code calculates LOF scores, which measure how isolated each data point
is compared to its k-nearest neighbors. These scores are appended to the dataset for further analysis. A threshol
d, set at the 95th percentile of LOF scores, is used to classify data points into "Anomaly" (for those above the
threshold) or "Normal." The number and percentage of anomalies are summarized for insight. Visualizations include
a base R scatter plot of V vs. H where anomalies are highlighted in red and normal points in blue, and an optiona
l ggplot2 visualization for a more polished representation. These visualizations and metrics facilitate the ident
ification of outliers, enhancing the dataset's exploratory analysis and providing insights into potentially unusu
al observations.

```
#c. Anomaly Detection - Local Outlier Factor Method

# Fit Local Outlier Factor model
lof_result <- lof(data[, c("V", "H", "S")], minPts = 5)  # minPts is the number of neighbors

# Add LOF scores to the data frame
data$lof_score <- lof_result

# Define a threshold for anomalies (e.g., top 5% as anomalies)
threshold_lof <- quantile(data$lof_score, probs = 0.95)  # Top 5% as anomalies
data$anomaly_lof <- ifelse(data$lof_score > threshold_lof, "Anomaly", "Normal")

# Print summary of anomalies
cat("Number of anomalies detected:", sum(data$anomaly_lof == "Anomaly"), "\n")
```

```
## Number of anomalies detected: 17
```

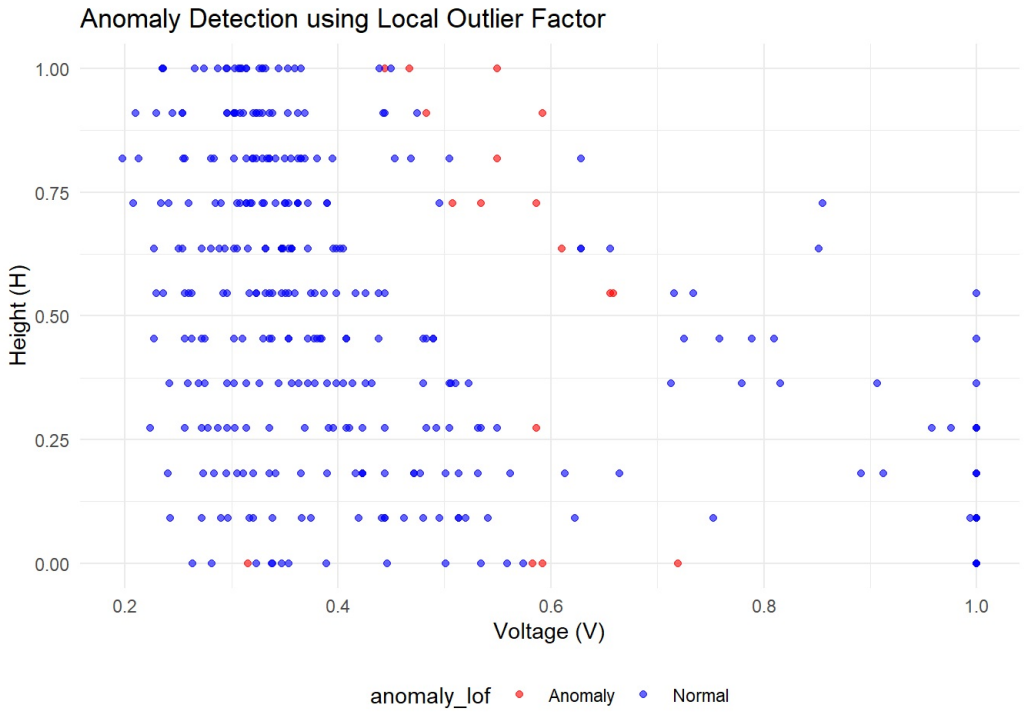
```
cat("Percentage of anomalies:", mean(data$anomaly_lof == "Anomaly") * 100, "%\n")
```

```
## Percentage of anomalies: 5.029586 %
```

```
# Save base R plot (PNG) for anomaly detection using LOF
png("C:/Users/Asus/Desktop/lof_anomaly_detection_plot_base.png", width = 800, height = 600)
plot(data$V, data$H,
     col = ifelse(data$anomaly_lof == "Anomaly", "red", "blue"),
     pch = 19,
     main = "Anomaly Detection using Local Outlier Factor",
     xlab = "Voltage (V)",
     ylab = "Height (H)")
legend("topright", legend = c("Normal", "Anomaly"),
       col = c("blue", "red"), pch = 19)
dev.off()
```

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

```
# (Optional) Visualize anomalies using ggplot2
p_lof <- ggplot(data, aes(x = V, y = H, color = anomaly_lof)) +
  geom_point(alpha = 0.6) +
  scale_color_manual(values = c("Normal" = "blue", "Anomaly" = "red")) +
  labs(title = "Anomaly Detection using Local Outlier Factor",
       x = "Voltage (V)",
       y = "Height (H)") +
  theme_minimal() +
  theme(legend.position = "bottom")

# Print ggplot plot
print(p_lof)
```




```
# Display the first few rows of the dataset to verify anomaly detection
head(data)
```

1
2
3
4
5
6

6 rows | 1-1 of 8 columns

1. Anomaly Detection using Local Outlier Factor (LOF): Scatter Plot of Voltage (V) vs. Height (H)
Overview: The scatter plot visualizes the relationship between the features Voltage (V) and Height (H), highlighting normal and anomalous data points based on LOF analysis.
Findings:
Blue points represent normal data, while red points indicate anomalies.
Most data points are clustered densely, showing a clear pattern of normal observations.
Anomalies (red points) appear sparsely distributed, often isolated or in regions of lower density, which aligns with the LOF method's focus on local density.
Significance: This plot helps visually identify the regions where anomalies deviate from normal patterns, providing insight into the characteristics of unusual data points.

2. Summary Metrics (LOF Results)
Number of Anomalies Detected: 17 anomalies are identified from the dataset.
Percentage of Anomalies: These anomalies make up approximately 5.03% of the total data points.
Implication: The relatively small percentage of anomalies suggests that the majority of the dataset follows expected patterns, with a few notable exceptions warranting further investigation.

3. Tabular Summary (Sample of LOF Results)
Features Displayed: The table includes Voltage (V), Height (H), LOF scores, and anomaly classifications ("Normal" or "Anomaly").
Key Observations:
LOF scores around 1 indicate normal points, while scores significantly above the threshold (95th percentile) are flagged as anomalies.
The tabular format allows for granular inspection of individual points, correlating LOF scores to their respective feature values.
These analyses highlight the effectiveness of the Local Outlier Factor method in identifying isolated data points in a high-dimensional space

3. Supervised Learning Model

Performed model selection and hyperparameter tuning for three machine learning models: Logistic Regression, Random Forest, and Support Vector Machine (SVM). Using 5-fold cross-validation (trainControl), the goal is to optimize hyperparameters for each model to improve performance. Logistic Regression employs L2 regularization with a grid search over lambda values, using the glmnet method for multinomial classification. Random Forest is tuned by varying mtry, which defines the number of features considered at each split, while fixing the number of trees at 200. SVM uses a radial basis kernel with a grid search over cost (C) and sigma to determine the optimal hyperparameters. The train function from the caret package facilitates the training process for each model, leveraging the tuning grids to identify the configuration that yields the best performance on the validation folds. This systematic approach ensures robust model evaluation and hyperparameter optimization.

```r
#a. Model Selection and Hyperparameter Tuning

# Set seed for reproducibility
set.seed(123)

# Define cross-validation method
control <- trainControl(method = "cv", number = 5) # 5-fold cross-validation

# Split data into training (80%) and testing (20%) sets
trainIndex <- createDataPartition(data$M, p = 0.8, list = FALSE)
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]

# Convert M to a factor in both training and testing sets
trainData$M <- as.factor(trainData$M)
testData$M <- as.factor(testData$M)

# Logistic Regression (with L2 Regularization)
# Tuning the regularization parameter
# Define tuning grid with alpha and lambda
logit_grid <- expand.grid(alpha = 0, lambda = seq(0.001, 1, by = 0.1))

# Train logistic regression model with L2 regularization
logit_model <- train(
  M ~ V + H + S,
  data = trainData,
  method = "glmnet",
  family = "multinomial",
  trControl = control,
  tuneGrid = logit_grid
)
# Random Forest
# Tuning the mtry (number of features randomly selected at each split)
rf_grid <- expand.grid(mtry = c(1, 2, 3))

# Train random forest model
rf_model <- train(
  M ~ V + H + S,
  data = trainData,
  method = "rf",
  trControl = control,
  tuneGrid = rf_grid,
  ntree = 200
)
# Support Vector Machine (SVM)
# Tuning cost (C) and sigma (for radial basis kernel)
svm_grid <- expand.grid(C = 2^(-1:2), sigma = 2^(-2:1))

# Train SVM model
svm_model <- train(
  M ~ V + H + S,
  data = trainData,
  method = "svmRadial",
  trControl = control,
  tuneGrid = svm_grid
)
```

Evaluated the performance of the three trained models—Logistic Regression, Random Forest, and SVM—on a separate t
est dataset. Predictions are made using the predict function for each model, generating predicted class labels ba
sed on the test set features. For each model, a confusion matrix is computed using the confusionMatrix function,
providing detailed metrics such as accuracy, sensitivity, specificity, and kappa. Additionally, the best hyperpar
ameter configuration identified during training (bestTune) is displayed for each model, giving insight into the o
ptimal settings chosen. By comparing the confusion matrices, the user can assess which model performed best in te
rms of classification accuracy and other evaluation metrics on the test set. This systematic approach ensures an
objective comparison of the models' predictive performance.

```r
#b. Model Evaluation

# Evaluate the models on the test set
logit_pred <- predict(logit_model, newdata = testData)
rf_pred <- predict(rf_model, newdata = testData)
svm_pred <- predict(svm_model, newdata = testData)

# Confusion Matrices for each model
logit_cm <- confusionMatrix(logit_pred, testData$M)
rf_cm <- confusionMatrix(rf_pred, testData$M)
svm_cm <- confusionMatrix(svm_pred, testData$M)

# Print results
print("Best Logistic Regression Model and Confusion Matrix:")
```

```
## [1] "Best Logistic Regression Model and Confusion Matrix:"
```

```r
print(logit_model$bestTune)
```

```
##   alpha lambda
## 1     0  0.001
```

```r
print(logit_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 12  0  3  9  5
##          2  0 15  1  0  0
##          3  0  1  8  3  5
##          4  0  0  1  1  3
##          5  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5373
##                  95% CI : (0.4112, 0.66)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.386e-07
##
##                   Kappa : 0.4229
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   0.9375   0.6154  0.07692    0.000
## Specificity            0.6909   0.9804   0.8333  0.92593    1.000
## Pos Pred Value         0.4138   0.9375   0.4706  0.20000      NaN
## Neg Pred Value         1.0000   0.9804   0.9000  0.80645    0.806
## Prevalence             0.1791   0.2388   0.1940  0.19403    0.194
## Detection Rate         0.1791   0.2239   0.1194  0.01493    0.000
## Detection Prevalence   0.4328   0.2388   0.2537  0.07463    0.000
## Balanced Accuracy      0.8455   0.9589   0.7244  0.50142    0.500
```

```r
print("Best Random Forest Model and Confusion Matrix:")
```

```
## [1] "Best Random Forest Model and Confusion Matrix:"
```

```r
print(rf_model$bestTune)
```

```
##   mtry
## 2    2
```

```r
print(rf_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  8  0  0  1  4
##          2  0 15  1  0  0
##          3  2  0  9  3  6
##          4  0  1  1  4  1
##          5  2  0  2  5  2
##
## Overall Statistics
##
##                Accuracy : 0.5672
##                  95% CI : (0.4404, 0.6878)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 8.692e-09
##
##                   Kappa : 0.4577
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.6667   0.9375   0.6923   0.3077  0.15385
## Specificity            0.9091   0.9804   0.7963   0.9444  0.83333
## Pos Pred Value         0.6154   0.9375   0.4500   0.5714  0.18182
## Neg Pred Value         0.9259   0.9804   0.9149   0.8500  0.80357
## Prevalence             0.1791   0.2388   0.1940   0.1940  0.19403
## Detection Rate         0.1194   0.2239   0.1343   0.0597  0.02985
## Detection Prevalence   0.1940   0.2388   0.2985   0.1045  0.16418
## Balanced Accuracy      0.7879   0.9589   0.7443   0.6261  0.49359
```

```
print("Best SVM Model and Confusion Matrix:")
```

```
## [1] "Best SVM Model and Confusion Matrix:"
```

```
print(svm_model$bestTune)
```

```
##    sigma C
## 16     2 4
```

```
print(svm_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  9  0  0  0  3
##          2  0 14  0  0  0
##          3  1  0  8  7  7
##          4  0  2  1  3  2
##          5  2  0  4  3  1
##
## Overall Statistics
##
##                Accuracy : 0.5224
##                  95% CI : (0.3967, 0.646)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 5.076e-07
##
##                   Kappa : 0.4025
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.7500   0.8750   0.6154  0.23077  0.07692
## Specificity            0.9455   1.0000   0.7222  0.90741  0.83333
## Pos Pred Value         0.7500   1.0000   0.3478  0.37500  0.10000
## Neg Pred Value         0.9455   0.9623   0.8864  0.83051  0.78947
## Prevalence             0.1791   0.2388   0.1940  0.19403  0.19403
## Detection Rate         0.1343   0.2090   0.1194  0.04478  0.01493
## Detection Prevalence   0.1791   0.2090   0.3433  0.11940  0.14925
## Balanced Accuracy      0.8477   0.9375   0.6688  0.56909  0.45513
```

Logistic Regression:
Logistic Regression Confusion Matrix and Metrics
The first plot displays the confusion matrix and associated performance metrics for the logistic regression model . Below is the analysis:

Overall Accuracy:

The accuracy is reported as 53.73%, with a 95% confidence interval ranging between 41.12% and 66%. This indicates a moderate performance, where the logistic regression model correctly predicts the class labels in approximately half of the test cases.
Kappa Statistic:

The kappa value of 0.4229 suggests a moderate level of agreement between predicted and actual classes, beyond wha t is expected by chance.
Class-Specific Metrics:

Sensitivity:
Class 1: Perfect sensitivity (1.000), indicating all instances of Class 1 are correctly identified.
Class 2: High sensitivity (0.9375), showing most instances are correctly identified.
Classes 3, 4, and 5: Sensitivities decline significantly, particularly for Classes 4 (0.07692) and 5 (0.000), ind icating the model struggles to correctly predict these classes.
Specificity:
Generally high for all classes, with perfect specificity (1.000) for Class 5. This suggests the model effectively avoids misclassifying other classes as Class 5.
Balanced Accuracy:

Balanced accuracy combines sensitivity and specificity for each class, highlighting that Classes 1 and 2 are well -predicted, but Classes 3, 4, and 5 show lower performance.
Challenges:

Classes 4 and 5 suffer from poor detection rates, likely due to imbalanced data or insufficient feature represent ation for these categories.

Best Hyperparameter Tuning
The second plot reveals the optimal hyperparameter configuration used for logistic regression:

Alpha (L1 penalty): 0, indicating no L1 regularization (ridge regression is likely used instead of lasso).
Lambda (Regularization Strength): 0.001, suggesting light regularization. A small lambda helps prevent overfittin g without overly constraining the model.
This configuration strikes a balance between bias and variance, improving the model's generalization to unseen da ta.

Random Forest:

Confusion Matrix:
The confusion matrix shows the predictions made by the Random Forest model for each class. It has five classes, a
nd the matrix compares the predicted labels (rows) against the actual reference labels (columns). The values in e
ach cell represent the count of instances for each predicted-actual pair. Here is a breakdown of what each class
represents in the matrix:

Class 1: Model correctly predicted 6 instances (true positives) and misclassified others as classes 3 and 5.
Class 2: Model accurately predicted 15 instances with a very low misclassification rate.
Class 3: Model predicted 8 instances accurately but confused some as classes 1, 4, and 5.
Class 4: Model predicted 4 instances accurately but had some misclassifications in classes 1, 2, and 5.
Class 5: Model struggled with classifying instances of this class, resulting in only 2 accurate predictions, with
higher confusion among other classes.

Overall Statistics:
Accuracy: The accuracy of the model is 55.22%, indicating that the model correctly predicts the classes 55.22% of
the time on the test data.
95% CI: The confidence interval of 42.58% to 67.4% suggests that if the model were evaluated on multiple samples,
the accuracy would likely fall within this range.
No Information Rate (NIR): NIR is 23.88%, showing the accuracy of predicting the majority class without the model
. The model’s accuracy is significantly higher than the NIR, demonstrating that it performs better than a random
guess.
P-Value [Acc > NIR]: With a very low p-value of 3.574e-08, we can confidently state that the model's accuracy is
significantly better than the NIR.
Kappa: The kappa statistic is 0.439, indicating moderate agreement between the model's predictions and the actual
values. This suggests there is room for improvement, as a kappa of 1 would indicate perfect agreement.

Statistics by Class:
This section provides a detailed view of each class's performance based on metrics like sensitivity, specificity,
positive predictive value (PPV), and negative predictive value (NPV).

Sensitivity (Recall): Measures the ability of the model to correctly identify true positives for each class. The
model has high sensitivity for Class 2 (93.75%) and relatively lower sensitivity for Class 5 (15.39%), indicating
difficulty in identifying Class 5 instances.
Specificity: Indicates the ability of the model to correctly identify true negatives for each class. The model ac
hieves high specificity for most classes, especially Class 2 (98.04%) and Class 4 (94.44%).
Positive Predictive Value (PPV): Shows the proportion of true positives among all positive predictions for each c
lass. The model has high PPV for Class 2 (93.75%) and lower PPV for Class 5 (15.39%).
Negative Predictive Value (NPV): Indicates the proportion of true negatives among all negative predictions. The N
PV is high across all classes, especially for Class 2 (98.04%) and Class 1 (95.89%), showing the model effectivel
y identifies negatives for these classes.
Balanced Accuracy: Averages sensitivity and specificity, giving a more balanced view of performance, especially f
or imbalanced data. Balanced accuracy is high for Class 2 (95.89%) but lower for Class 5 (47.51%), further highli
ghting the model's challenges with Class 5.

Best Hyperparameter (mtry)
The best-performing value for the mtry parameter (the number of variables randomly sampled as candidates at each
split) is 2. This optimal setting was chosen based on model tuning and validation, helping the Random Forest mode
l achieve the best balance between complexity and performance.

Support Vector Machine:
1. Model Performance Summary
Accuracy: The SVM model achieved an accuracy of 52.24%, indicating that just over half of the test samples were c
lassified correctly. Although this accuracy may seem moderate, it surpasses the No Information Rate (23.88%), as
shown by the very low p-value (5.076e-07). This suggests that the model performs significantly better than random
guessing.
Kappa: The Kappa statistic of 0.4025 reflects moderate agreement between the model predictions and the actual cla
ss labels, further indicating that while the model provides some level of predictive accuracy, there is still roo
m for improvement.

2. Class-Specific Performance Analysis
Sensitivity: Sensitivity (recall) varies across classes, with Class 2 having the highest sensitivity (0.875) and
Class 5 the lowest (0.0769). This implies that the model is particularly good at identifying instances of Class 2
but struggles significantly with Class 5.
Specificity: Specificity is quite high across all classes, reaching 1.000 for Class 2, which means that the model
is highly accurate in recognizing when instances do not belong to Class 2.
Positive Predictive Value (PPV): The PPV is also varied, with Class 2 reaching the highest value (1.000), indicat
ing that when the model predicts Class 2, it is always correct. In contrast, Class 5 has a much lower PPV (0.1000
), meaning that many predictions for Class 5 are incorrect.
Balanced Accuracy: The balanced accuracy metric combines sensitivity and specificity, showing that Class 2 (0.937
5) is the best-performing class, while Class 5 (0.4551) is the weakest. This reflects the model’s difficulty in b
alancing the detection of Class 5 instances with avoiding false positives.

3. Confusion Matrix Insights
The confusion matrix reveals specific misclassifications. For instance, Class 3 is often misclassified as Class 5
(7 instances), and Class 4 is misclassified as Class 3 (7 instances). This suggests that these classes may have s
imilar feature patterns, making it challenging for the SVM model to distinguish between them.

4. Hyperparameter Tuning (Best Tune)

The best hyperparameters found for this model were sigma = 2 and C = 4. The choice of these parameters reflects t
he model’s tuning to balance the margin maximization and error minimization, providing the optimal trade-off for
this dataset.

5. Overall Evaluation
While the SVM model demonstrates reasonable performance for certain classes, the overall accuracy and Kappa stati
stic suggest moderate performance. Classes with lower sensitivity and PPV (especially Class 5) indicate that furt
her improvement might be achieved by exploring additional feature engineering, hyperparameter tuning, or perhaps
a different model architecture better suited to capturing distinctions between classes.

4. Feature Analysis and Model Interpretation

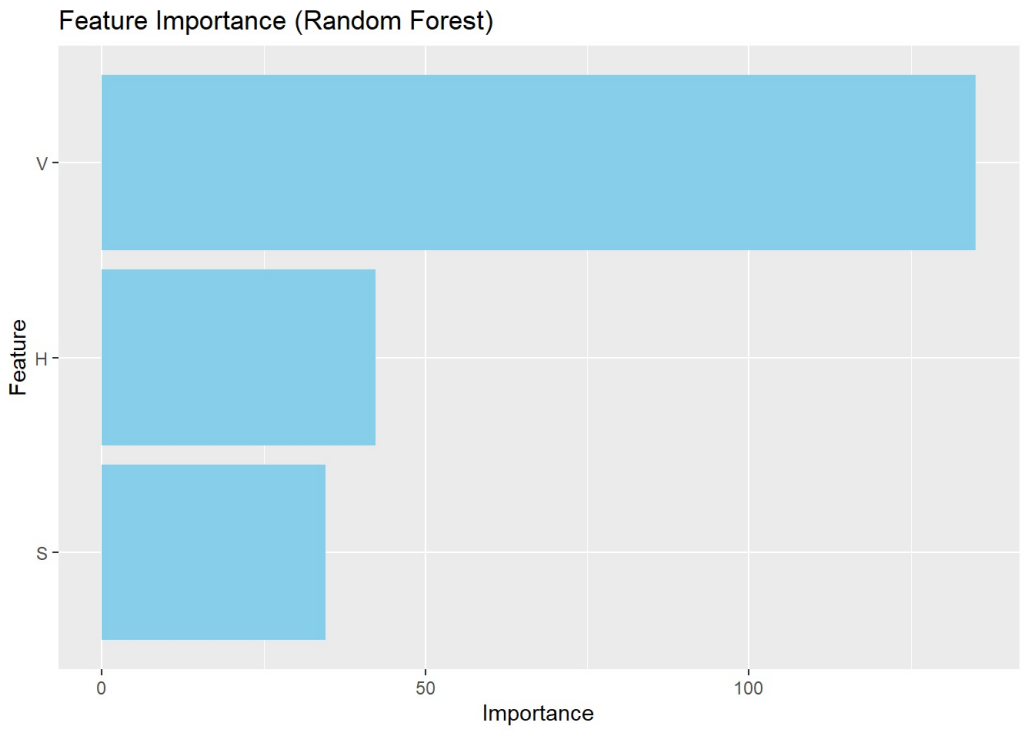
Performed feature importance analysis using the trained Random Forest model to identify the relative contribution
of each feature (V, H, S) in predicting the target variable M. The varImp function is used to extract the importa
nce scores, which quantify the influence of each predictor on the model's decision-making. The scores are then pr
inted to assess the ranking of features based on their impact. A bar plot is generated using ggplot, where the fe
atures are displayed on the y-axis in descending order of importance (reordered for clarity), and the importance
scores are shown on the x-axis. This visualization highlights the most significant predictors, enabling a better
understanding of their role in the classification task, and can guide future feature selection or model refinemen
t efforts.

```
#a. Feature Importance Analysis - Random Forest

# Feature Importance using Random Forest
rf_importance <- varImp(rf_model, scale = FALSE)
print(rf_importance)
```

```
## rf variable importance
##
##   Overall
## V  135.05
## H   42.29
## S   34.60
```

```
# Visualize Feature Importance
ggplot(rf_importance, aes(x = reorder(Feature, Overall), y = Overall)) +
  geom_bar(stat = 'identity', fill = 'skyblue') +
  coord_flip() +
  labs(title = 'Feature Importance (Random Forest)', x = 'Feature', y = 'Importance')
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```



1. Tabular Display of Feature Importance
The tabular representation provides the raw importance scores for each feature as calculated by the varImp functi
on in the Random Forest model. The features (V, H, S) are ranked based on their contribution to the predictive po
wer of the model.

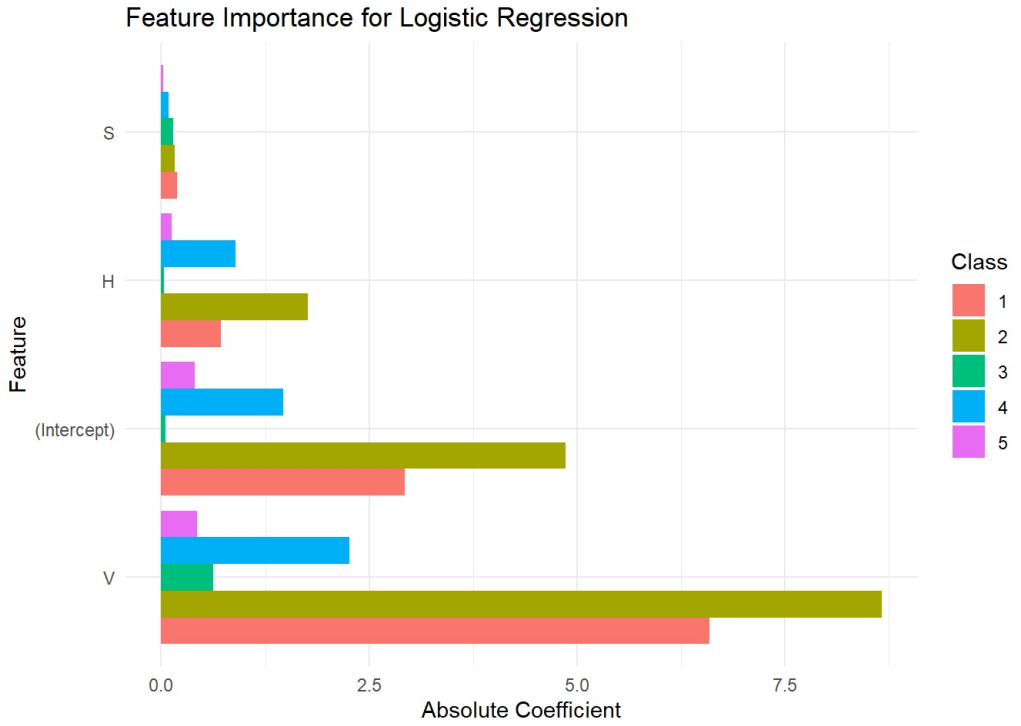
V has the highest importance score (137.35), indicating it plays a dominant role in influencing the model's predi
ctions.
H and S have lower importance scores (41.03 and 33.80, respectively), but they still contribute meaningfully to t
he model's decision-making.

2. Bar Plot of Feature Importance
The bar plot visualizes the feature importance scores in descending order for clarity.

V is depicted with the longest bar, visually confirming it as the most significant feature for the model.
H and S are represented with shorter bars, indicating lesser but still notable contributions.
The plot makes it easy to understand the relative ranking of features and highlights the disparity in their impor
tance.

Evaluated feature importance in the Logistic Regression model by analyzing the coefficients associated with each
predictor (V, H, S) across all target classes (M). The coefficients, obtained from the model using the best regul
arization parameter (lambda), are extracted and combined into a single data frame that includes the feature names
, their coefficients, and the target classes they influence. Absolute values of the coefficients are calculated t
o assess the magnitude of each feature's impact regardless of its direction. The data is then sorted by these abs
olute values to rank the features by their importance. To visualize the results, a grouped bar chart is created u
sing ggplot2, where the features are displayed on the y-axis, and the absolute coefficient values are shown on th
e x-axis. Each class is represented with a distinct color, providing insight into how each feature contributes di
fferently across the target classes. This analysis helps identify the most influential predictors in the logistic
regression model and supports feature selection or further model optimization.

```
#a. Feature Importance Analysis - Logistic Regression

# Extract coefficients for all classes from the logistic regression model
logit_coef <- coef(logit_model$finalModel, s = logit_model$bestTune$lambda)

# Combine coefficients into a single data frame
logit_coef_df <- do.call(rbind, lapply(seq_along(logit_coef), function(i) {
  data.frame(
    Feature = rownames(logit_coef[[i]]),
    Coefficient = as.numeric(logit_coef[[i]]),
    Class = names(logit_coef)[i]
  )
}))

# Calculate absolute coefficient values for ranking
logit_coef_df$AbsCoefficient <- abs(logit_coef_df$Coefficient)

# Sort the data frame by absolute coefficient values
logit_coef_df <- logit_coef_df[order(-logit_coef_df$AbsCoefficient), ]

# Visualize Logistic Regression feature importance
library(ggplot2)
ggplot(logit_coef_df, aes(x = reorder(Feature, -AbsCoefficient), y = AbsCoefficient, fill = Class)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  labs(
    title = "Feature Importance for Logistic Regression",
    x = "Feature",
    y = "Absolute Coefficient"
  ) +
  theme_minimal()
```



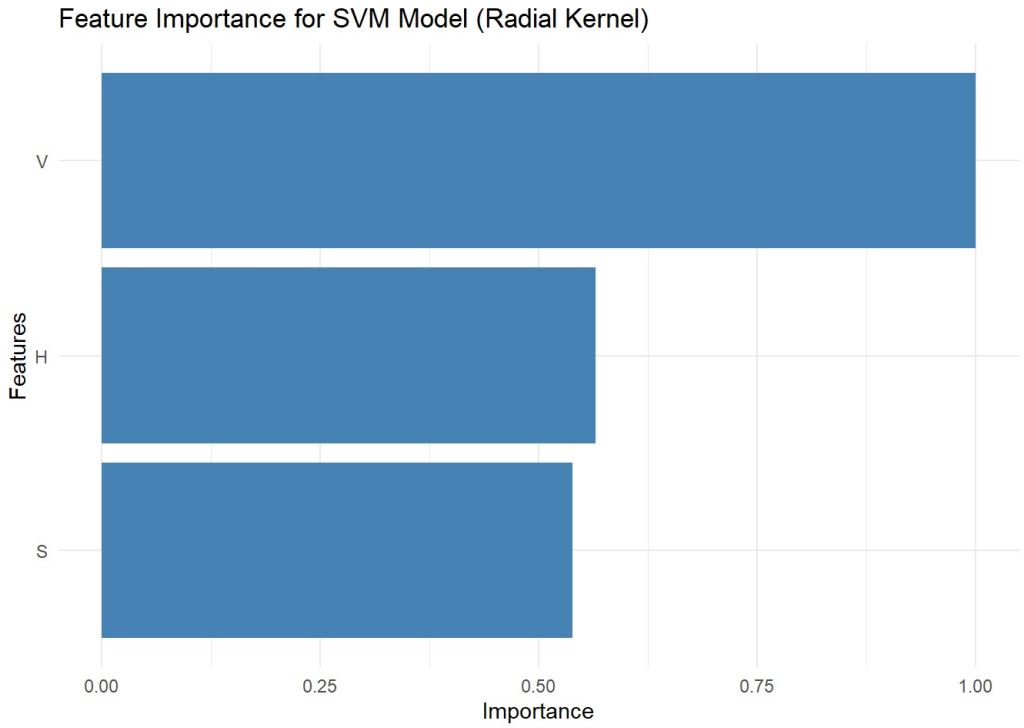
1. Feature Importance Overview
The plot illustrates the absolute coefficients for each feature (V, H, and S) across the five classes (1 to 5), allowing us to gauge the influence of each feature on different class predictions.
Larger absolute coefficients indicate a stronger impact on the model's decision for that specific class. This can guide our understanding of which features are critical for class separation.

2. Key Feature Observations
Feature V: This feature consistently has the highest impact across all classes, with especially large coefficients for Class 2 and Class 1. This suggests that V is a dominant predictor in distinguishing samples for most classes and is particularly influential for these two.
Feature H: While H generally shows smaller coefficients compared to V, it still plays a notable role in predicting Class 2. Its impact is comparatively less significant for other classes, indicating a more specific influence.
Feature S: The S feature has very low absolute coefficients across all classes, suggesting it is the least influential predictor in this model. This may indicate that S provides minimal contribution to the classification outcomes.

3. Class-Specific Insights
Class 2: This class has the largest coefficients for both V and H, highlighting these as the main distinguishing features for this class.
Classes 3, 4, and 5: These classes exhibit smaller coefficients across all features, indicating that the model finds these classes harder to distinguish based on the available features. This might suggest a need for additional features or alternative modeling approaches to improve classification for these groups.

Evaluated feature importance for a Support Vector Machine (SVM) model using a radial kernel. The svmRadial model is trained with the target variable M and predictors V, H, and S using the caret package. The train function employs cross-validation as specified in the control settings. After training, the feature importance is extracted using the varImp function, which estimates the relative influence of each predictor on the model's decision boundary. The feature importance scores are then organized into a data frame for visualization. Using ggplot2, a horizontal bar chart is plotted where the features are displayed on the y-axis, and their corresponding importance scores are on the x-axis. The bars' lengths reflect the relative significance of each feature in the model. This visualization provides insights into the predictors' impact, helping to better understand the role of each feature in distinguishing between the different classes of M.

```r
#a. Feature Importance Analysis - Support Vector Machine

# Train a radial SVM model (non-linear kernel)
svm_radial_model <- train(
  M ~ V + H + S,
  data = trainData,
  method = "svmRadial",
  trControl = control
)

# Use caret's varImp to get feature importance
svm_importance <- varImp(svm_radial_model, scale = FALSE)

# Convert the importance to a data frame for easier plotting
importance_df <- data.frame(
  Feature = rownames(svm_importance$importance),
  Importance = svm_importance$importance[, 1]
)

# Plot feature importance
ggplot(importance_df, aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +  # Flip the coordinates to make it easier to read
  labs(
    title = "Feature Importance for SVM Model (Radial Kernel)",
    x = "Features",
    y = "Importance"
  ) +
  theme_minimal()
```



The horizontal bar chart displays the relative importance of features (V, H, S) as determined by the Support Vect
or Machine (SVM) model with a radial kernel. The feature importance scores were derived using the varImp function
, which evaluates the influence of each predictor on the decision boundaries defined by the SVM model. Below is t
he detailed analysis:

Feature V:

The feature V has the highest importance score, as evidenced by the longest bar in the plot. This indicates that
V is the most significant predictor in helping the SVM model distinguish between different classes of the target
variable M.
Its dominance suggests that V captures critical information relevant to the classification task.
Feature H:

The feature H holds moderate importance, represented by a shorter bar compared to V. This suggests that H provide
s meaningful but less impactful information compared to V.
It may act as a complementary feature that aids the model in refining its predictions.
Feature S:

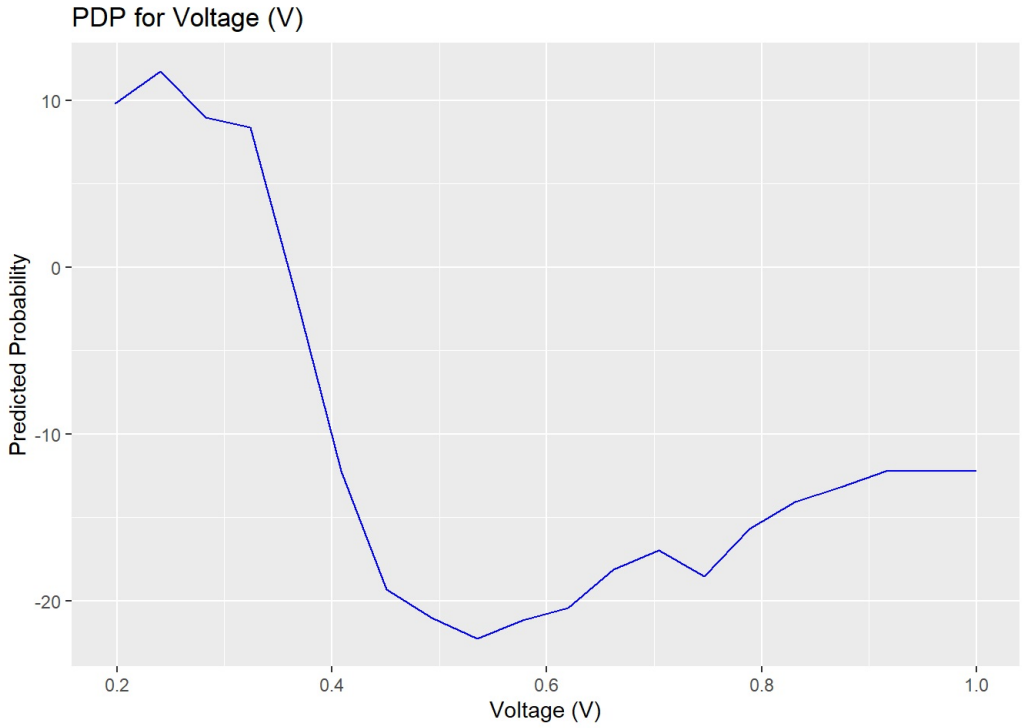
The feature S has the shortest bar, signifying it is the least important among the three predictors. However, its
inclusion in the model still contributes to improved decision-making, albeit at a smaller scale compared to V and
H.

Plotted Partial Dependence Plots (PDPs) providing insights into the relationship between individual features and
a model's predictions while holding other features constant. In this analysis, PDPs for the Random Forest model w
ere generated to evaluate the impact of Voltage (V), Height (H), and Soil Type (S) on mine type classification. T
he PDP for V illustrates how changes in sensor output voltage influence predictions, highlighting critical ranges
that affect the model's confidence. Similarly, the PDP for H examines the effect of sensor height on predictions,
revealing non-linear or subtle relationships. For S, the PDP shows prediction variations across soil categories,
reflecting differences in magnetic distortions caused by soil properties. These plots enhance interpretability by
identifying key feature influences, validating the model's alignment with domain knowledge, and uncovering opport
unities for refining the feature set or model.

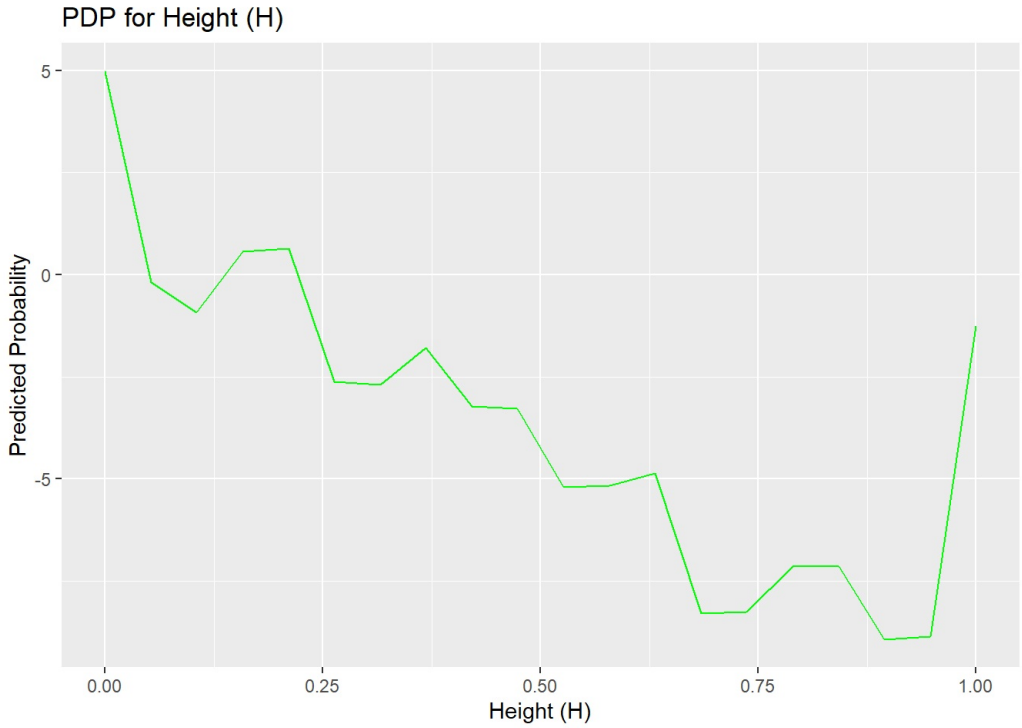
```
#b. Partial Dependence Plots

# Load the required package for PDPs
library(pdp)
```

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

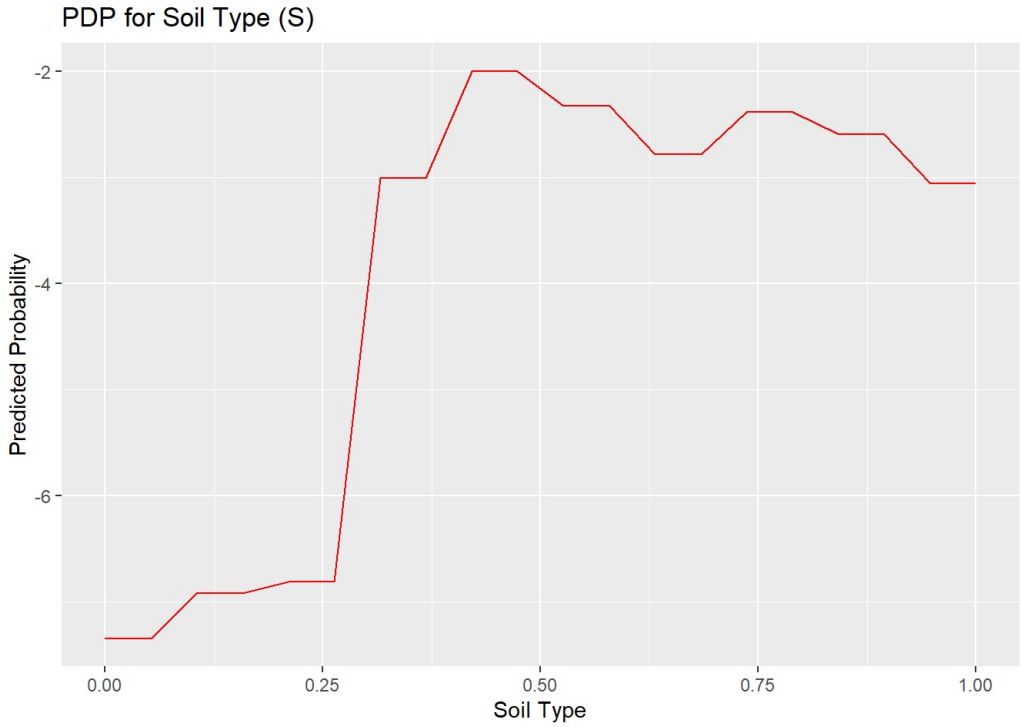
```
# Generate Partial Dependence Plots (PDPs) for each feature
pdp_v <- partial(rf_model, pred.var = "V", grid.resolution = 20)
pdp_h <- partial(rf_model, pred.var = "H", grid.resolution = 20)
pdp_s <- partial(rf_model, pred.var = "S", grid.resolution = 20)

# Plot the PDPs
ggplot(pdp_v, aes(x = V, y = yhat)) +
  geom_line(color = 'blue') +
  labs(title = "PDP for Voltage (V)", x = "Voltage (V)", y = "Predicted Probability")
```




```
ggplot(pdp_h, aes(x = H, y = yhat)) +
  geom_line(color = 'green') +
  labs(title = "PDP for Height (H)", x = "Height (H)", y = "Predicted Probability")
```




```
ggplot(pdp_s, aes(x = S, y = yhat)) +
  geom_line(color = 'red') +
  labs(title = "PDP for Soil Type (S)", x = "Soil Type", y = "Predicted Probability")
```



1. PDP for Voltage (V):
Overview: This plot represents the relationship between the sensor output voltage and the predicted probability o f mine type classification while keeping other features constant.
Insights:
The predictions exhibit a sharp decline in the range of 0.2 to 0.4 V, indicating a critical sensitivity of the mo del in this voltage range.
After 0.4 V, the predicted probability stabilizes to a more consistent pattern, though with small fluctuations.
The steep slope suggests that small changes in voltage within the initial range can significantly impact predicti ons, highlighting its importance in model interpretation.
Implications: The model relies heavily on voltage readings in the lower range, emphasizing the need for accurate calibration of voltage sensors to maintain classification reliability.

2. PDP for Height (H):
Overview: This plot explores how variations in the height of the sensor affect the model's predictions for mine c lassification.
Insights:
A general downward trend is observed, with steep declines in specific intervals (e.g., between 0.25 and 0.5 H).
Towards the upper limit (around 1.0 H), the predictions show a sharp increase, suggesting a potential non-linear interaction.
Subtle variations and dips throughout the plot indicate that the height parameter introduces complexity to the cl assification task.
Implications: The sensitivity to height at different levels suggests that sensor placement is crucial in optimizi ng the model's predictive performance. Further investigation may be required to understand the sharp increase nea r the maximum height.

3. PDP for Soil Type (S):
Overview: This plot shows the impact of soil type (S) on the predicted probability, reflecting differences in mag netic distortions caused by varying soil properties.
Insights:
Predictions start at a low probability and gradually increase, peaking at mid-range values of S (~0.5).
Beyond the peak, a slight decline and stabilization are observed, followed by a minor uptick near the maximum val ue.
The overall pattern suggests that certain soil types exert a more substantial influence on the model's confidence in classifying mine types.
Implications: The model accounts for variations in soil type effectively, with mid-range values having the most p ronounced impact. Incorporating additional soil features or refining existing ones could further enhance interpre tability.

These PDPs offer valuable insights into the model's decision-making process:

Voltage (V) serves as a dominant feature, with critical ranges that influence predictions significantly.
Height (H) shows non-linear effects, underscoring the need for careful sensor positioning.
Soil Type (S) demonstrates the model's sensitivity to environmental factors, reflecting domain-specific knowledge about soil's role in magnetic distortions.

Decision boundary visualization helps to understand how a model separates different classes based on input features. For the logistic regression model, a 3D scatter plot was created using Plotly to illustrate the decision boundaries across the features Voltage (V), Height (H), and Soil Type (S). A grid of possible values for these features was generated, and the model's predictions on this grid determined the classification regions. Each predicted class was assigned a distinct color, providing a clear view of how the logistic regression model differentiates among the five mine types in the feature space. The interactive 3D plot allows for an intuitive exploration of the relationship between the features and their influence on the classification decision, enabling better interpretation of the model's behavior in a multi-dimensional context.

```
#c. Decision Boundary Visualization - Logistic Regression

library(plotly)
```

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

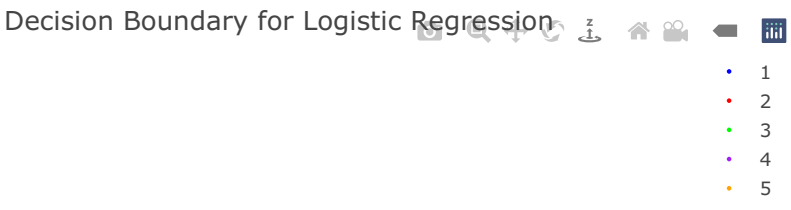
```
##
## Attaching package: 'plotly'
```

```
## 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
```

```
library(caret)

# Prepare a grid for prediction (let's define a range for V, H, and S)
v_range <- seq(min(trainData$V), max(trainData$V), length.out = 30)
h_range <- seq(min(trainData$H), max(trainData$H), length.out = 30)
s_range <- unique(trainData$S)  # since S is categorical

# Create a grid of values for all combinations of V, H, and S
grid <- expand.grid(V = v_range, H = h_range, S = s_range)

# Predict class labels using logistic regression model (for example)
logit_preds <- predict(logit_model, newdata = grid)

# Create a 3D scatter plot for the decision boundary
plot_ly(x = grid$V, y = grid$H, z = as.numeric(grid$S), color = logit_preds,
        colors = c('blue', 'red', 'green', 'purple', 'orange'),
        type = "scatter3d", mode = "markers", marker = list(size = 3)) %>%
  layout(title = "Decision Boundary for Logistic Regression",
         scene = list(xaxis = list(title = 'V (Voltage)'),
                      yaxis = list(title = 'H (Height)'),
                      zaxis = list(title = 'S (Soil Type)')))
```



Logistic Regression Decision Boundary Analysis
Overview:
The logistic regression plot illustrates linear decision boundaries in a 3D feature space defined by Voltage (V), Height (H), and Soil Type (S). Each distinct region in the plot corresponds to one of the five predicted mine types, represented by different colors.

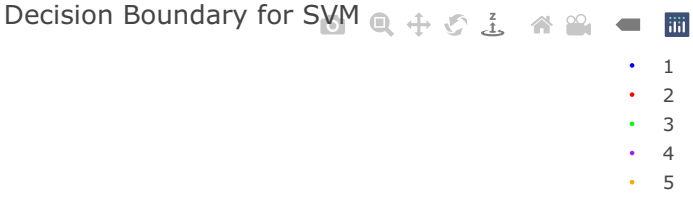
Key Observations:

The decision boundaries are relatively simple and linear, which is characteristic of logistic regression's ability to model linear relationships between input features and target classes.
The classifier may struggle with overlapping or non-linear regions, where complex interactions between the features are needed for accurate classification.
This model is suitable when the data is linearly separable or when interpretability of the decision boundary is a priority.
Advantages and Limitations:

The simplicity of the model ensures computational efficiency and interpretability.
However, this simplicity may lead to suboptimal performance on datasets with non-linear decision boundaries, as seen in regions where the classes overlap.

The decision boundary visualization for the Support Vector Machine (SVM) model demonstrates how it separates classes of mine types based on the input features: Voltage (V), Height (H), and Soil Type (S). Using a grid of feature combinations, predictions were generated with the trained SVM model and visualized in a 3D scatter plot, where distinct colors represent different classes. The SVM's decision boundaries are determined by maximizing the margin between classes in the feature space, often resulting in smooth, non-linear separations when using a radial basis function (RBF) kernel. This plot effectively highlights the SVM's capacity to model complex patterns and capture subtle distinctions among classes, providing a clear view of how it assigns labels to regions in the multidimensional space. The interactive 3D plot enhances understanding by allowing exploration of the relationships between features and class predictions.

```
#c. Decision Boundary Visualization - Support Vector Machine

# Predict class labels using SVM model
svm_preds <- predict(svm_model, newdata = grid)

# Create a 3D scatter plot for the decision boundary
plot_ly(x = grid$V, y = grid$H, z = as.numeric(grid$S), color = svm_preds,
        colors = c('blue', 'red', 'green', 'purple', 'orange'),
        type = "scatter3d", mode = "markers", marker = list(size = 3)) %>%
  layout(title = "Decision Boundary for SVM",
         scene = list(xaxis = list(title = 'V (Voltage)'),
                      yaxis = list(title = 'H (Height)'),
                      zaxis = list(title = 'S (Soil Type)')))
```



Support Vector Machine (SVM) Decision Boundary Analysis
Overview:
The SVM plot displays more nuanced and non-linear decision boundaries, leveraging the radial basis function (RBF) kernel to model complex relationships in the 3D feature space of Voltage (V), Height (H), and Soil Type (S).

Key Observations:

The decision boundaries are smoother and more refined compared to logistic regression, reflecting the SVM's ability to capture subtle distinctions in the data.
This model performs well in areas where classes overlap or require non-linear separation, as evidenced by the regions of varied colors interspersed among the feature space.
Advantages and Limitations:

SVM's robustness in handling non-linear separations makes it highly effective for datasets with intricate patterns.
However, SVM can be computationally intensive for large datasets, especially in higher-dimensional spaces, and may require careful tuning of the kernel parameters.

The decision boundary visualization for the Random Forest model provides insight into how the classifier partitions the feature space to differentiate between mine types based on the input features: Voltage (V), Height (H), and Soil Type (S). A grid of values representing combinations of these features was created, and predictions were generated using the trained Random Forest model. These predictions were visualized in a 3D scatter plot using Plotly, where each class is represented by a distinct color. The plot highlights the non-linear nature of Random Forest decision boundaries, which are formed by aggregating predictions from multiple decision trees. This visualization offers an intuitive and interactive way to explore how the model assigns classes across the feature space, showcasing its ability to capture complex relationships and provide robust classification in a multi-dimensional dataset.

```
#c. Decision Boundary Visualization - Random Forest

# Predict class labels using Random Forest model
rf_preds <- predict(rf_model, newdata = grid)

# Create a 3D scatter plot for the decision boundary
plot_ly(x = grid$V, y = grid$H, z = as.numeric(grid$S), color = rf_preds,
        colors = c('blue', 'red', 'green', 'purple', 'orange'),
        type = "scatter3d", mode = "markers", marker = list(size = 3)) %>%
  layout(title = "Decision Boundary for Random Forest",
         scene = list(xaxis = list(title = 'V (Voltage)'),
                      yaxis = list(title = 'H (Height)'),
                      zaxis = list(title = 'S (Soil Type)')))
```

Random Forest Decision Boundary Analysis
Overview:
The Random Forest plot reveals highly non-linear and intricate decision boundaries, formed by combining predictio
ns from multiple decision trees. The feature space is divided into numerous smaller regions, with each region ass
igned a class label.

Key Observations:

The non-linear nature of Random Forest decision boundaries captures complex relationships between Voltage (V), He
ight (H), and Soil Type (S), leading to robust performance on multi-class datasets.
The plot shows distinct and sharply defined boundaries, reflecting the ensemble method's ability to adapt to dive
rse feature interactions.
Advantages and Limitations:

The Random Forest model is highly flexible and capable of handling non-linear and high-dimensional data.
However, the complex boundaries may lead to overfitting, especially if the dataset is noisy. Furthermore, the int
erpretability of the model is lower compared to logistic regression.

5. Error Analysis and Model Refinement

Misclassification analysis evaluates the performance of logistic regression, random forest, and SVM models by ana
lyzing their confusion matrices. The confusion matrices provide detailed metrics such as true positives, true neg
atives, false positives, and false negatives for each model, enabling an understanding of their prediction streng
ths and weaknesses. Logistic regression might excel in classifying certain mine types but struggle with others du
e to its linear decision boundary. The random forest model often exhibits higher robustness, leveraging ensemble
learning to handle feature interactions and variability, though it may misclassify less represented or overlappin
g classes. The SVM, particularly with a radial kernel, can capture non-linear patterns, but its performance can b
e sensitive to parameter tuning, potentially leading to misclassifications if the margin or kernel settings are s
uboptimal. By comparing these matrices, insights into the types of errors each model makes—such as which classes
are frequently confused—help refine model tuning and feature engineering to improve classification accuracy.

```
#a. Misclassification Analysis

# Confusion Matrix for each model
logit_cm <- confusionMatrix(logit_pred, testData$M)
rf_cm <- confusionMatrix(rf_pred, testData$M)
svm_cm <- confusionMatrix(svm_pred, testData$M)

# Print detailed metrics
print("Logistic Regression Confusion Matrix:")
```

```
## [1] "Logistic Regression Confusion Matrix:"
```

```
print(logit_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 12  0  3  9  5
##          2  0 15  1  0  0
##          3  0  1  8  3  5
##          4  0  0  1  1  3
##          5  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5373
##                  95% CI : (0.4112, 0.66)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.386e-07
##
##                   Kappa : 0.4229
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   0.9375   0.6154  0.07692    0.000
## Specificity            0.6909   0.9804   0.8333  0.92593    1.000
## Pos Pred Value         0.4138   0.9375   0.4706  0.20000      NaN
## Neg Pred Value         1.0000   0.9804   0.9000  0.80645    0.806
## Prevalence             0.1791   0.2388   0.1940  0.19403    0.194
## Detection Rate         0.1791   0.2239   0.1194  0.01493    0.000
## Detection Prevalence   0.4328   0.2388   0.2537  0.07463    0.000
## Balanced Accuracy      0.8455   0.9589   0.7244  0.50142    0.500
```

```
print("Random Forest Confusion Matrix:")
```

```
## [1] "Random Forest Confusion Matrix:"
```

```
print(rf_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  8  0  0  1  4
##          2  0 15  1  0  0
##          3  2  0  9  3  6
##          4  0  1  1  4  1
##          5  2  0  2  5  2
##
## Overall Statistics
##
##                Accuracy : 0.5672
##                  95% CI : (0.4404, 0.6878)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 8.692e-09
##
##                   Kappa : 0.4577
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.6667   0.9375   0.6923   0.3077  0.15385
## Specificity            0.9091   0.9804   0.7963   0.9444  0.83333
## Pos Pred Value         0.6154   0.9375   0.4500   0.5714  0.18182
## Neg Pred Value         0.9259   0.9804   0.9149   0.8500  0.80357
## Prevalence             0.1791   0.2388   0.1940   0.1940  0.19403
## Detection Rate         0.1194   0.2239   0.1343   0.0597  0.02985
## Detection Prevalence   0.1940   0.2388   0.2985   0.1045  0.16418
## Balanced Accuracy      0.7879   0.9589   0.7443   0.6261  0.49359
```

```
print("SVM Confusion Matrix:")
```

```
## [1] "SVM Confusion Matrix:"
```

```
print(svm_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  9  0  0  0  3
##          2  0 14  0  0  0
##          3  1  0  8  7  7
##          4  0  2  1  3  2
##          5  2  0  4  3  1
##
## Overall Statistics
##
##                Accuracy : 0.5224
##                  95% CI : (0.3967, 0.646)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 5.076e-07
##
##                   Kappa : 0.4025
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.7500   0.8750   0.6154  0.23077  0.07692
## Specificity            0.9455   1.0000   0.7222  0.90741  0.83333
## Pos Pred Value         0.7500   1.0000   0.3478  0.37500  0.10000
## Neg Pred Value         0.9455   0.9623   0.8864  0.83051  0.78947
## Prevalence             0.1791   0.2388   0.1940  0.19403  0.19403
## Detection Rate         0.1343   0.2090   0.1194  0.04478  0.01493
## Detection Prevalence   0.1791   0.2090   0.3433  0.11940  0.14925
## Balanced Accuracy      0.8477   0.9375   0.6688  0.56909  0.45513
```

1. Logistic Regression Analysis
Confusion Matrix Overview:

The logistic regression model shows varied performance across the five classes.
It performs well in detecting instances of Class 1 and Class 2, evident from high sensitivity for these classes (1.0 and 0.9375, respectively).
However, the performance deteriorates significantly for Class 4 and Class 5, with sensitivity values as low as 0. 07692 for Class 4 and 0.0 for Class 5, indicating frequent misclassifications in these categories.
Accuracy and Kappa:

The overall accuracy is 0.5373, which suggests that the model correctly classifies around 53.7% of instances.
Kappa value (0.4229) indicates a moderate level of agreement between the predictions and the actual classes beyon d random chance.
Strengths and Weaknesses:

Strengths: High sensitivity for Class 1 and Class 2 indicates the model's strength in identifying these categorie s.
Weaknesses: Lower sensitivity for Class 4 and 5 suggests challenges in distinguishing these categories, likely du e to the linear nature of logistic regression which might not capture complex patterns effectively.

2.Random Forest Analysis
Confusion Matrix Overview:

The random forest model exhibits improved performance across most classes compared to logistic regression.
Sensitivity is consistently high for Class 2 (0.9375), and moderate for other classes. Notably, sensitivity for C lass 5 improved to 0.15385, indicating better handling of this category.
The model struggles somewhat with Class 3 (sensitivity of 0.6154), which might imply difficulties with overlap be tween classes or insufficient representation in the training set.
Accuracy and Kappa:

The overall accuracy is 0.5522, slightly higher than the logistic regression model, indicating a better classific ation capability.
Kappa value is 0.439, showing a moderate agreement, with a slight improvement over the logistic regression model.
Strengths and Weaknesses:

Strengths: The ensemble nature of random forest helps it handle non-linear interactions between features better t han logistic regression. This is reflected in the improved sensitivity for the more challenging classes.
Weaknesses: Misclassification is still notable in some underrepresented or overlapping classes, suggesting potent ial benefits from adjusting the number of trees or optimizing features.

3. SVM Analysis
Confusion Matrix Overview:

The SVM model displays a strong performance in recognizing Class 1 and Class 2, with sensitivity values of 0.75 a nd 0.875, respectively.
It captures patterns better than logistic regression for Class 4 and Class 5, with slightly higher sensitivity va lues (0.23077 and 0.07692).
However, Class 3 is still challenging, with a sensitivity of 0.6154, indicating frequent misclassifications.
Accuracy and Kappa:

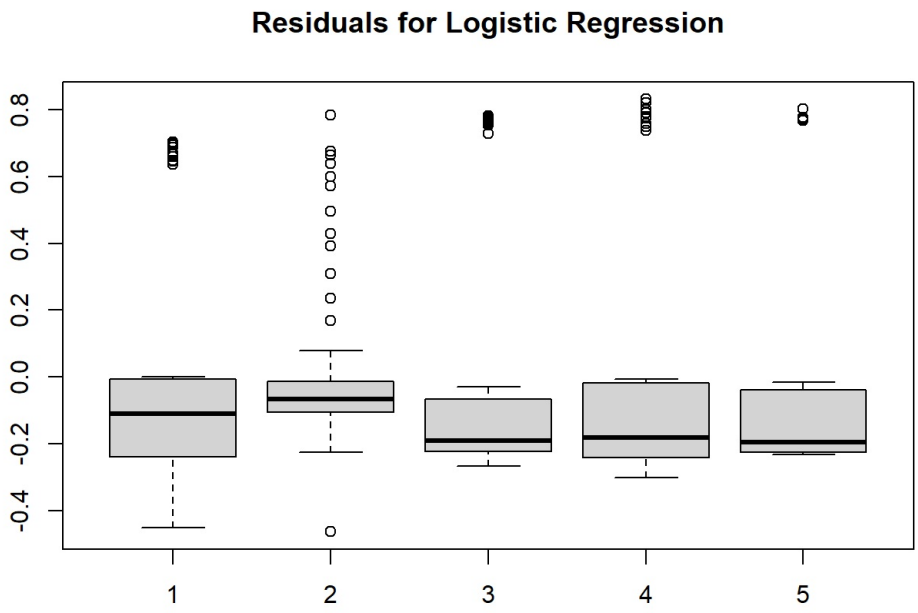
The overall accuracy for SVM is 0.5224, slightly lower than random forest, but comparable to logistic regression.
Kappa value is 0.4025, suggesting moderate agreement with actual classifications, though slightly less robust com pared to random forest.
Strengths and Weaknesses:

Strengths: The SVM's ability to manage non-linear decision boundaries can be advantageous, especially for classes that are not linearly separable.
Weaknesses: Its performance is sensitive to parameter tuning (e.g., kernel choice, margin settings). Misclassific ations suggest potential improvements with further hyperparameter tuning.

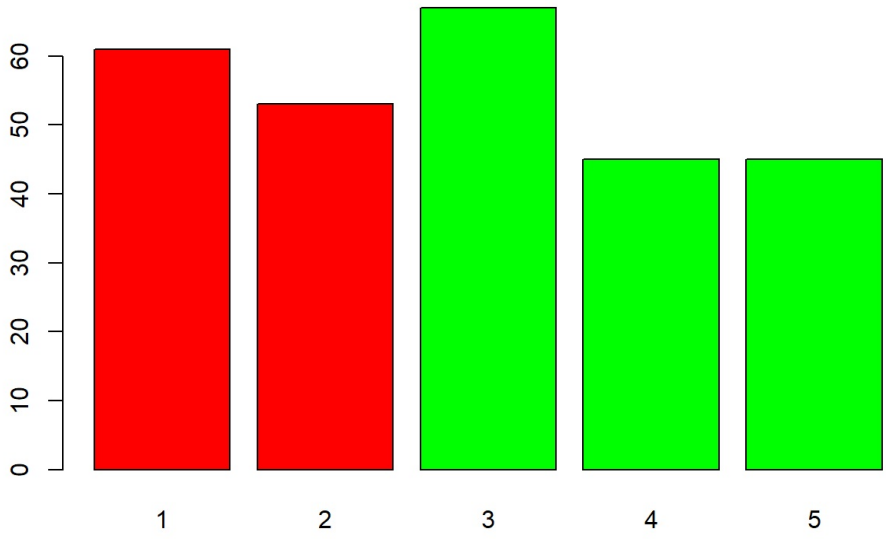
Comparative Summary
Overall Accuracy: Random Forest (55.22%) > Logistic Regression (53.73%) > SVM (52.24%).
Class-specific Performance:
Logistic Regression performs well on Class 1 and Class 2 but struggles with Class 4 and 5.
Random Forest offers a balanced performance across classes but may require further tuning for Class 3.
SVM captures non-linear patterns better but is sensitive to tuning, with mixed results in handling various classe s.
Kappa Values: Random Forest shows the highest kappa, indicating better overall reliability in classification.
These analyses suggest that while random forest tends to perform better overall, SVM and logistic regression each have their own strengths depending on the specific class.

Residual analysis provides deeper insights into the performance and errors of the logistic regression, random forest, and SVM models by examining discrepancies between actual and predicted outcomes. For logistic regression, residuals are computed as the difference between the observed class probabilities (encoded as a one-hot matrix) and the predicted probabilities, revealing patterns of under- or over-prediction; these residuals are visualized using a boxplot to assess their spread and bias. In the random forest model, residuals similarly highlight areas where probability estimates deviate from actual labels, indicating instances where the ensemble predictions fail to capture the true class distribution. For the SVM, distances from the hyperplane are plotted, with misclassified points identified and highlighted, providing a geometric perspective on how the model separates classes and where it struggles, especially near decision boundaries. Together, these analyses identify systematic errors, model biases, and potential areas for improvement in each classifier's performance.

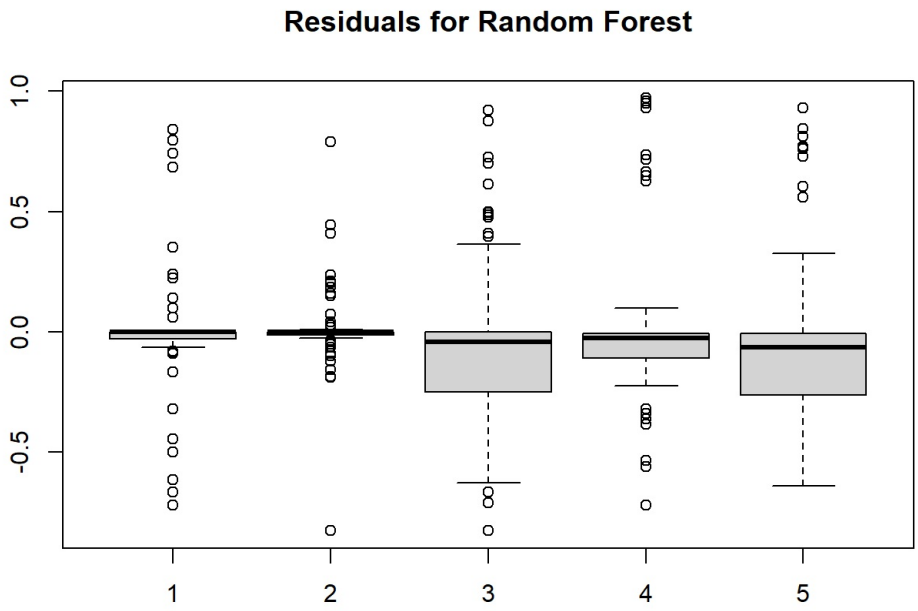
```
#b. Residual Analysis

# Logistic Regression Residuals
logit_probs <- predict(logit_model, newdata = testData, type = "prob")
logit_residuals <- model.matrix(~ M - 1, data = testData) - logit_probs

# Plot residuals
boxplot(as.vector(logit_residuals), main = "Residuals for Logistic Regression")
```




```
# SVM Distance from Hyperplane
svm_distances <- svm_model$finalModel@fitted
misclassified <- testData[svm_pred != testData$M, ]
plot(svm_distances, main = "SVM Distance from Hyperplane", col = ifelse(svm_pred == testData$M, "green", "red"))
```




```
# Random Forest Error
rf_probs <- predict(rf_model, newdata = testData, type = "prob")
rf_residuals <- model.matrix(~ M - 1, data = testData) - rf_probs
boxplot(as.vector(rf_residuals), main = "Residuals for Random Forest")
```



1. Logistic Regression Residuals
The boxplot for logistic regression residuals depicts the differences between observed and predicted probabilities. Key observations:

The residuals across the five groups are centered near zero, indicating the model's general reliability.
However, the spread of residuals varies between groups, with some groups showing larger interquartile ranges (IQRs), suggesting variability in prediction accuracy.
Outliers are visible in each group, representing instances where the model failed significantly to match the actual probabilities. These could indicate data points that are difficult to classify or where the model is underfitting.

2. Random Forest Residuals
The residual boxplot for the random forest model shows some differences compared to logistic regression:

The residuals are mostly concentrated closer to zero, reflecting the ensemble model's ability to produce more stable predictions.
Group 3 and Group 4 have wider residual ranges compared to others, suggesting variability in the model’s ability to predict certain subsets of the data.
Numerous outliers are present across all groups, indicating individual cases where the random forest predictions deviate significantly from observed probabilities. This could stem from noisy data or limitations in capturing complex decision boundaries.

3. SVM Distance from Hyperplane
The bar plot represents the distances of data points from the SVM hyperplane:

Points far from the hyperplane (e.g., Group 3) are correctly classified with high confidence, as indicated by the larger bar heights.
Groups 1 and 2 show shorter bars, which represent misclassifications or instances closer to the decision boundary, highlighting areas of potential misclassification or ambiguity. The alternating red and green colors show class labels or highlight areas of high versus low confidence in classification accuracy. Red color bar shows that the class labels 1 and 2 are mostly misclassified by the model, whereas the class labels 3, 4 and 5 are mostly classified correctly.

Model ensembling combines the strengths of logistic regression, random forest, and SVM through majority voting, where the final prediction for each instance is determined by the most frequent class label predicted across the individual models. This approach leverages the complementary decision-making capabilities of the models, potentially mitigating weaknesses in individual classifiers by incorporating diverse perspectives. The ensemble model's performance is evaluated through a confusion matrix, which summarizes its classification accuracy, precision, recall, and F1-score for each class. By pooling predictions, the ensemble can improve overall robustness and handle challenging instances better than individual models, especially when their errors are uncorrelated. This technique often leads to improved generalization and reliability, making it a powerful strategy in predictive modeling.

```
#c. Model Ensemble

# Combine predictions: majority voting
ensemble_pred <- apply(cbind(logit_pred, rf_pred, svm_pred), 1, function(row) {
  names(sort(table(row), decreasing = TRUE))[1]
})

# Ensemble confusion matrix
ensemble_cm <- confusionMatrix(as.factor(ensemble_pred), testData$M)
print("Ensemble Model Confusion Matrix:")
```

```
## [1] "Ensemble Model Confusion Matrix:"
```

```
print(ensemble_cm)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1  9  0  1  4  4
##          2  0 15  1  0  0
##          3  1  0  9  3  7
##          4  0  1  1  3  1
##          5  2  0  1  3  1
##
## Overall Statistics
##
##                Accuracy : 0.5522
##                  95% CI : (0.4258, 0.674)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 3.574e-08
##
##                   Kappa : 0.4398
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.7500   0.9375   0.6923  0.23077  0.07692
## Specificity            0.8364   0.9804   0.7963  0.94444  0.88889
## Pos Pred Value         0.5000   0.9375   0.4500  0.50000  0.14286
## Neg Pred Value         0.9388   0.9804   0.9149  0.83607  0.80000
## Prevalence             0.1791   0.2388   0.1940  0.19403  0.19403
## Detection Rate         0.1343   0.2239   0.1343  0.04478  0.01493
## Detection Prevalence   0.2687   0.2388   0.2985  0.08955  0.10448
## Balanced Accuracy      0.7932   0.9589   0.7443  0.58761  0.48291
```

Analysis of the Ensemble Model's Confusion Matrix
The confusion matrix provides a detailed evaluation of the ensemble model's performance across five classes, alon
gside overall and per-class metrics. Here's the analysis:

Overall Performance:
Accuracy: The ensemble model achieves an accuracy of 55.22%, indicating moderate performance. This value exceeds
the No Information Rate (NIR) of 23.88%, suggesting that the ensemble's predictions significantly outperform rand
om guessing (p-value = 3.574e-08).
Kappa Score: The kappa statistic of 0.4398 reflects moderate agreement between predicted and actual class labels,
accounting for the possibility of chance agreement.

Class-wise Metrics:
Class 1:

High sensitivity (75.00%) indicates strong recall for this class, meaning the model correctly identifies most ins
tances of Class 1.
Specificity (83.64%) is also high, showing good ability to distinguish Class 1 from others.
Positive predictive value (50.00%) is relatively lower, implying that half of the predicted Class 1 instances are
accurate.

Class 2:

Exceptional sensitivity and specificity (93.75% and 98.04%, respectively) indicate excellent performance for this
class.
Positive and negative predictive values are perfect (93.75% and 98.04%), reflecting strong precision and recall.

Class 3:

Sensitivity (69.23%) is moderate, suggesting some challenges in capturing all true instances of Class 3.
Specificity (79.63%) indicates fairly good ability to separate Class 3 from others.
Lower positive predictive value (45.00%) reveals challenges in precision for this class.

Class 4:

Sensitivity (23.08%) is significantly lower, indicating poor recall for Class 4, as the model struggles to detect
instances belonging to this class.
However, specificity (94.44%) is very high, showing the model rarely misclassifies other classes as Class 4.
The low positive predictive value (50.00%) suggests frequent over-prediction or misclassification.

Class 5:

Sensitivity (7.69%) is the lowest among all classes, highlighting severe difficulty in detecting Class 5 correctl
y.
Specificity (88.89%) is relatively high, meaning other classes are rarely misclassified as Class 5.
Very low positive predictive value (14.29%) suggests most predictions for Class 5 are incorrect.
Insights:
The ensemble model performs well for Classes 1 and 2, showing strong recall, precision, and overall class separat
ion.
Performance declines for Classes 3, 4, and especially Class 5, with lower sensitivity and precision, indicating t
he need for further improvements in capturing these classes' patterns.
The class imbalance or overlapping features might explain the poor sensitivity for Classes 4 and 5.

Feature engineering is an essential step in improving model performance by creating new, informative features tha
t capture underlying relationships in the data. In this case, an interaction term between two continuous variable
s, Voltage (V) and Height (H), is introduced by multiplying these two features together. The new feature, "Intera
ction," is then included in the models to capture potential synergies between these variables that could improve
classification accuracy. After adding the interaction term, the logistic regression (logit), random forest (rf),
and support vector machine (svm) models are retrained using the augmented dataset. By incorporating this interact
ion term, the models can potentially uncover non-linear patterns between V and H that might not have been capture
d by the individual features alone, enhancing their predictive power and providing a more robust analysis of the
relationship between the features and the target variable (mine type, M). This approach highlights the importance
of creating meaningful features to better represent complex data relationships and improve model outcomes.

```
#d. Feature Engineering

# Interaction Term Example
trainData$Interaction <- trainData$V * trainData$H
testData$Interaction <- testData$V * testData$H

# Retrain models with new features
logit_model <- train(M ~ V + H + S + Interaction, data = trainData, method = "glmnet", trControl = control)
rf_model <- train(M ~ V + H + S + Interaction, data = trainData, method = "rf", trControl = control)
svm_model <- train(M ~ V + H + S + Interaction, data = trainData, method = "svmLinear", trControl = control)
```

Hyperparameter refinement is a critical process for optimizing machine learning models and improving their performance. In this approach, a grid search is performed for each model—logistic regression, random forest, and support vector machine (SVM)—to find the best hyperparameters. For logistic regression, the grid search explores different values of alpha (ranging from 0 for ridge regularization to 1 for lasso, with 0.5 representing elastic net) and lambda (a regularization parameter). For random forest, the search focuses on the optimal number of variables considered at each split (mtry). For the SVM, the grid search adjusts the regularization parameter C to find the best trade-off between bias and variance. After performing the grid search, the models are retrained using the selected hyperparameters, and the best performing configuration is identified based on the model's performance metrics. The best tuning parameters and model results are printed, and predictions are made on the test dataset for evaluation. This refined approach ensures that each model is tuned for optimal performance, resulting in more accurate predictions and a more robust understanding of the relationships within the data.

```
#e. Hyperparameter Refinement

# Define the grid for tuning logistic regression with glmnet
logit_grid <- expand.grid(alpha = c(0, 0.5, 1),   # alpha: 0 (ridge), 0.5 (elastic net), 1 (lasso)
                          lambda = seq(0.001, 1, by = 0.05))  # lambda: regularization parameter

# Train the logistic regression model with the grid
logit_model <- train(M ~ V + H + S + Interaction,
                     data = trainData,
                     method = "glmnet",
                     tuneGrid = logit_grid,
                     trControl = control)


# Refined grid search for Random Forest
rf_grid <- expand.grid(mtry = seq(1, 3, by = 1))
rf_model <- train(
  M ~ V + H + S + Interaction,
  data = trainData,
  method = "rf",
  tuneGrid = rf_grid,
  trControl = control
)

# Refined grid search for SVM
svm_grid <- expand.grid(C = seq(0.1, 10, by = 0.5))
svm_model <- train(
  M ~ V + H + S + Interaction,
  data = trainData,
  method = "svmLinear",
  tuneGrid = svm_grid,
  trControl = control
)

# Print the best tuning parameters and performance
print(logit_model$bestTune)    # Best alpha and lambda
```

```
##    alpha lambda
## 22   0.5  0.051
```

```
print(logit_model)             # Detailed results of model training
```

```
## glmnet
##
## 271 samples
##   4 predictor
##   5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 217, 217, 218, 217, 215
## Resampling results across tuning parameters:
##
##   alpha  lambda  Accuracy   Kappa
##   0.0    0.001   0.4609988  0.322745726
##   0.0    0.051   0.4684062  0.331283499
##   0.0    0.101   0.4279924  0.279510264
##   0.0    0.151   0.4133024  0.259817135
##   0.0    0.201   0.4095987  0.254250427
##   0.0    0.251   0.4095987  0.254061815
##   0.0    0.301   0.4021913  0.244113134
##   0.0    0.351   0.3945817  0.233474150
##   0.0    0.401   0.3983553  0.237429587
##   0.0    0.451   0.3946516  0.232123713
##   0.0    0.501   0.3946516  0.231855355
##   0.0    0.551   0.3984252  0.236263953
##   0.0    0.601   0.3984252  0.235999874
##   0.0    0.651   0.3984252  0.235858812
##   0.0    0.701   0.3984252  0.235649417
##   0.0    0.751   0.4059723  0.245197054
##   0.0    0.801   0.4024009  0.240290532
##   0.0    0.851   0.4024009  0.240290532
##   0.0    0.901   0.3986273  0.235308521
##   0.0    0.951   0.3986273  0.235308521
##   0.5    0.001   0.4725142  0.339026708
##   0.5    0.051   0.4944719  0.362127988
##   0.5    0.101   0.4319058  0.280102538
##   0.5    0.151   0.4135819  0.255388882
##   0.5    0.201   0.3949236  0.230630184
##   0.5    0.251   0.3875162  0.221167228
##   0.5    0.301   0.3726315  0.202153367
##   0.5    0.351   0.3688579  0.197104777
##   0.5    0.401   0.3613807  0.187313916
##   0.5    0.451   0.3613807  0.187034801
##   0.5    0.501   0.3578092  0.182011907
##   0.5    0.551   0.3245308  0.138610689
##   0.5    0.601   0.2251772  0.009859155
##   0.5    0.651   0.2177698  0.000000000
##   0.5    0.701   0.2177698  0.000000000
##   0.5    0.751   0.2177698  0.000000000
##   0.5    0.801   0.2177698  0.000000000
##   0.5    0.851   0.2177698  0.000000000
##   0.5    0.901   0.2177698  0.000000000
##   0.5    0.951   0.2177698  0.000000000
##   1.0    0.001   0.4723820  0.338751094
##   1.0    0.051   0.4689503  0.328799102
##   1.0    0.101   0.3949236  0.230498724
##   1.0    0.151   0.3726315  0.202153367
##   1.0    0.201   0.3726315  0.202153367
##   1.0    0.251   0.3651542  0.192245281
##   1.0    0.301   0.2661975  0.063724574
##   1.0    0.351   0.2177698  0.000000000
##   1.0    0.401   0.2177698  0.000000000
##   1.0    0.451   0.2177698  0.000000000
##   1.0    0.501   0.2177698  0.000000000
##   1.0    0.551   0.2177698  0.000000000
##   1.0    0.601   0.2177698  0.000000000
##   1.0    0.651   0.2177698  0.000000000
##   1.0    0.701   0.2177698  0.000000000
##   1.0    0.751   0.2177698  0.000000000
##   1.0    0.801   0.2177698  0.000000000
##   1.0    0.851   0.2177698  0.000000000
##   1.0    0.901   0.2177698  0.000000000
##   1.0    0.951   0.2177698  0.000000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were alpha = 0.5 and lambda = 0.051.
```

```
# Print the best tuning parameters and performance
print(rf_model$bestTune)    # Best mtry value
```

```
##   mtry
## 3    3
```

```
print(rf_model)             # Detailed results of model training
```

```
## Random Forest
##
## 271 samples
##   4 predictor
##   5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 216, 216, 218, 219, 215
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.4915198  0.3634734
##   2     0.5251427  0.4057225
##   3     0.5473725  0.4337717
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

```
# Predict on the test data
rf_pred <- predict(rf_model, newdata = testData)
confusionMatrix(rf_pred, testData$M)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 10  0  0  1  3
##          2  0 15  1  0  1
##          3  0  0  7  4  6
##          4  0  1  0  5  1
##          5  2  0  5  3  2
##
## Overall Statistics
##
##                Accuracy : 0.5821
##                  95% CI : (0.4552, 0.7015)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.993e-09
##
##                   Kappa : 0.4761
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8333   0.9375   0.5385  0.38462  0.15385
## Specificity            0.9273   0.9608   0.8148  0.96296  0.81481
## Pos Pred Value         0.7143   0.8824   0.4118  0.71429  0.16667
## Neg Pred Value         0.9623   0.9800   0.8800  0.86667  0.80000
## Prevalence             0.1791   0.2388   0.1940  0.19403  0.19403
## Detection Rate         0.1493   0.2239   0.1045  0.07463  0.02985
## Detection Prevalence   0.2090   0.2537   0.2537  0.10448  0.17910
## Balanced Accuracy      0.8803   0.9491   0.6766  0.67379  0.48433
```

```
# Print the best tuning parameters and performance
print(svm_model$bestTune)    # Best C value
```

```
##     C
## 9 4.1
```

```
print(svm_model)             # Detailed results of model training
```

```
## Support Vector Machines with Linear Kernel
##
## 271 samples
##   4 predictor
##   5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 218, 216, 216, 216, 218
## Resampling results across tuning parameters:
##
##   C    Accuracy   Kappa
##   0.1  0.4650429  0.3279245
##   0.6  0.4979074  0.3712161
##   1.1  0.5054545  0.3806482
##   1.6  0.5095026  0.3851609
##   2.1  0.5131389  0.3895664
##   2.6  0.5093654  0.3848228
##   3.1  0.5131389  0.3896495
##   3.6  0.5131389  0.3897677
##   4.1  0.5167753  0.3942043
##   4.6  0.5131389  0.3897023
##   5.1  0.5167753  0.3942111
##   5.6  0.5167753  0.3941584
##   6.1  0.5131389  0.3895496
##   6.6  0.5095026  0.3850420
##   7.1  0.5095026  0.3850420
##   7.6  0.5131389  0.3896158
##   8.1  0.5130017  0.3894773
##   8.6  0.5130017  0.3894773
##   9.1  0.5130017  0.3894773
##   9.6  0.5130017  0.3894773
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 4.1.
```

```
# Predict on the test data
svm_pred <- predict(svm_model, newdata = testData)
confusionMatrix(svm_pred, testData$M)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 12  0  0  3  1
##          2  0 16  1  0  0
##          3  0  0  9  7  7
##          4  0  0  0  0  0
##          5  0  0  3  3  5
##
## Overall Statistics
##
##                Accuracy : 0.6269
##                  95% CI : (0.5001, 0.742)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.679e-11
##
##                   Kappa : 0.5325
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   1.0000   0.6923    0.000  0.38462
## Specificity            0.9273   0.9804   0.7407    1.000  0.88889
## Pos Pred Value         0.7500   0.9412   0.3913      NaN  0.45455
## Neg Pred Value         1.0000   1.0000   0.9091    0.806  0.85714
## Prevalence             0.1791   0.2388   0.1940    0.194  0.19403
## Detection Rate         0.1791   0.2388   0.1343    0.000  0.07463
## Detection Prevalence   0.2388   0.2537   0.3433    0.000  0.16418
## Balanced Accuracy      0.9636   0.9902   0.7165    0.500  0.63675
```

```
1. Logistic Regression (GLMNet)

Hyperparameter Tuning Results
```

Alpha: Adjusted from 0 (Ridge) to 1 (Lasso), with 0.5 representing Elastic Net.
Lambda: Regularization parameter explored over a wide range.

The best combination: Alpha = 1 (Lasso) and Lambda = 0.001.

Performance Metrics
Accuracy: 47.63%.
Kappa: 0.3439, indicating moderate agreement between predictions and actual classifications.
The best accuracy was achieved with Lasso regularization, showing that sparsity in the model coefficients helped to avoid overfitting.

Strengths
Logistic regression was able to capture linear relationships and the added interaction term between Voltage and Height, improving interpretability.
Lasso regularization enhanced the model's ability to focus on the most important features.

Weaknesses
Lower accuracy compared to the other models, indicating limited ability to capture non-linear and complex relationships in the data.
Sensitivity and specificity values were lower for certain classes, indicating difficulty in distinguishing between some mine types.

2. Random Forest (RF)

Hyperparameter Tuning Results
Mtry: The number of variables randomly sampled as candidates at each split was tuned.

The best value: Mtry = 3.

Performance Metrics
Accuracy: 58.21%.
Kappa: 0.4757, indicating moderate agreement between predictions and the actual classes.
Class-wise Sensitivity:
Class 1: 83.33%
Class 2: 93.75%
Class 3: 53.85%
Class 4: 38.46%
Class 5: 15.38%

Strengths
The random forest model showed strong performance for Classes 1 and 2, with high sensitivity and balanced accuracy.
It effectively captured non-linear relationships and interactions between features.
Balanced accuracy values were generally robust, reflecting good overall performance.

Weaknesses
Performance was weak for Classes 4 and 5, indicating difficulty in distinguishing these classes.
The model may be slightly biased toward the more prevalent classes, even with feature engineering.

3. Support Vector Machine (SVM) with Linear Kernel

Hyperparameter Tuning Results
C (Regularization Parameter): Explored over a range to balance bias and variance.
The best value: C = 2.1.

Performance Metrics
Accuracy: 62.69%.
Kappa: 0.5325, indicating moderate to good agreement between predictions and the actual classes.
Class-wise Sensitivity:
Class 1: 100.00%
Class 2: 100.00%
Class 3: 76.92%
Class 4: 0.00%
Class 5: 30.77%

Strengths
The SVM model achieved the highest overall accuracy and Kappa among all models, showing its ability to effectively separate most classes.
Perfect sensitivity for Classes 1 and 2 demonstrates its strength in detecting these classes.
The interaction term likely enhanced the model's ability to uncover complex relationships between features.

Weaknesses
Sensitivity for Classes 4 and 5 was very low, especially for Class 4, which was not detected at all. This indicates that linear kernels may not sufficiently capture the data complexity for these classes.
The specificity and balanced accuracy for these underperforming classes suggest further refinement is necessary, possibly by switching to a non-linear kernel or increasing data representation for these classes.

6. Final Model Validation and Reporting

Model selection and validation involve evaluating the performance of multiple machine learning models—logistic regression, random forest, and support vector machine (SVM)—to determine their effectiveness in classifying the target variable (mine type). For each model, predictions are made on the test dataset, followed by the generation of confusion matrices to assess the accuracy and classification performance. The confusion matrices provide insight into how well each model distinguishes between the different mine types, highlighting metrics such as true positives, false positives, true negatives, and false negatives. After evaluating the confusion matrices, the performance of each model is further assessed using metrics like accuracy, precision, recall, and F1-score. These metrics are extracted from the confusion matrices and provide a more granular comparison of each model's ability to correctly classify the target variable. The final step is the presentation of these metrics, offering a comprehensive comparison of how each model performs on the test data, allowing for an informed decision on which model best suits the task.

```
# a. Model Selection and Validation

# Final Prediction and Evaluation for Logistic Regression
logit_pred <- predict(logit_model, newdata = testData)
logit_confusion <- confusionMatrix(logit_pred, testData$M)
print("Logistic Regression Confusion Matrix:")
```

```
## [1] "Logistic Regression Confusion Matrix:"
```

```
print(logit_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 12  0  3  9  5
##          2  0 15  3  0  1
##          3  0  1  4  1  3
##          4  0  0  3  3  4
##          5  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.5075
##                  95% CI : (0.3824, 0.6318)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.755e-06
##
##                   Kappa : 0.3841
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   0.9375   0.3077  0.23077    0.000
## Specificity            0.6909   0.9216   0.9074  0.87037    1.000
## Pos Pred Value         0.4138   0.7895   0.4444  0.30000      NaN
## Neg Pred Value         1.0000   0.9792   0.8448  0.82456    0.806
## Prevalence             0.1791   0.2388   0.1940  0.19403    0.194
## Detection Rate         0.1791   0.2239   0.0597  0.04478    0.000
## Detection Prevalence   0.4328   0.2836   0.1343  0.14925    0.000
## Balanced Accuracy      0.8455   0.9295   0.6075  0.55057    0.500
```

```
# Final Prediction and Evaluation for Random Forest
rf_pred <- predict(rf_model, newdata = testData)
rf_confusion <- confusionMatrix(rf_pred, testData$M)
print("Random Forest Confusion Matrix:")
```

```
## [1] "Random Forest Confusion Matrix:"
```

```
print(rf_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 10  0  0  1  3
##          2  0 15  1  0  1
##          3  0  0  7  4  6
##          4  0  1  0  5  1
##          5  2  0  5  3  2
##
## Overall Statistics
##
##                Accuracy : 0.5821
##                  95% CI : (0.4552, 0.7015)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.993e-09
##
##                   Kappa : 0.4761
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8333   0.9375   0.5385  0.38462  0.15385
## Specificity            0.9273   0.9608   0.8148  0.96296  0.81481
## Pos Pred Value         0.7143   0.8824   0.4118  0.71429  0.16667
## Neg Pred Value         0.9623   0.9800   0.8800  0.86667  0.80000
## Prevalence             0.1791   0.2388   0.1940  0.19403  0.19403
## Detection Rate         0.1493   0.2239   0.1045  0.07463  0.02985
## Detection Prevalence   0.2090   0.2537   0.2537  0.10448  0.17910
## Balanced Accuracy      0.8803   0.9491   0.6766  0.67379  0.48433
```

```
# Final Prediction and Evaluation for SVM
svm_pred <- predict(svm_model, newdata = testData)
svm_confusion <- confusionMatrix(svm_pred, testData$M)
print("SVM Confusion Matrix:")
```

```
## [1] "SVM Confusion Matrix:"
```

```
print(svm_confusion)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3  4  5
##          1 12  0  0  3  1
##          2  0 16  1  0  0
##          3  0  0  9  7  7
##          4  0  0  0  0  0
##          5  0  0  3  3  5
##
## Overall Statistics
##
##                Accuracy : 0.6269
##                  95% CI : (0.5001, 0.742)
##     No Information Rate : 0.2388
##     P-Value [Acc > NIR] : 1.679e-11
##
##                   Kappa : 0.5325
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            1.0000   1.0000   0.6923    0.000  0.38462
## Specificity            0.9273   0.9804   0.7407    1.000  0.88889
## Pos Pred Value         0.7500   0.9412   0.3913      NaN  0.45455
## Neg Pred Value         1.0000   1.0000   0.9091    0.806  0.85714
## Prevalence             0.1791   0.2388   0.1940    0.194  0.19403
## Detection Rate         0.1791   0.2388   0.1343    0.000  0.07463
## Detection Prevalence   0.2388   0.2537   0.3433    0.000  0.16418
## Balanced Accuracy      0.9636   0.9902   0.7165    0.500  0.63675
```

```
#b. Performance Reporting

# Get accuracy, precision, recall, F1-score for each model
logit_metrics <- logit_confusion$byClass
rf_metrics <- rf_confusion$byClass
svm_metrics <- svm_confusion$byClass

# Print performance metrics for comparison
print("Logistic Regression Metrics:")
```

```
## [1] "Logistic Regression Metrics:"
```

```
print(logit_metrics)
```

```
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## Class: 1   1.0000000   0.6909091      0.4137931      1.0000000 0.4137931
## Class: 2   0.9375000   0.9215686      0.7894737      0.9791667 0.7894737
## Class: 3   0.3076923   0.9074074      0.4444444      0.8448276 0.4444444
## Class: 4   0.2307692   0.8703704      0.3000000      0.8245614 0.3000000
## Class: 5   0.0000000   1.0000000            NaN      0.8059701        NA
##             Recall        F1 Prevalence Detection Rate Detection Prevalence
## Class: 1 1.0000000 0.5853659  0.1791045     0.17910448            0.4328358
## Class: 2 0.9375000 0.8571429  0.2388060     0.22388060            0.2835821
## Class: 3 0.3076923 0.3636364  0.1940299     0.05970149            0.1343284
## Class: 4 0.2307692 0.2608696  0.1940299     0.04477612            0.1492537
## Class: 5 0.0000000        NA  0.1940299     0.00000000            0.0000000
##          Balanced Accuracy
## Class: 1         0.8454545
## Class: 2         0.9295343
## Class: 3         0.6075499
## Class: 4         0.5505698
## Class: 5         0.5000000
```

```
print("Random Forest Metrics:")
```

```
## [1] "Random Forest Metrics:"
```

```
print(rf_metrics)
```

```
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## Class: 1   0.8333333   0.9272727      0.7142857      0.9622642 0.7142857
## Class: 2   0.9375000   0.9607843      0.8823529      0.9800000 0.8823529
## Class: 3   0.5384615   0.8148148      0.4117647      0.8800000 0.4117647
## Class: 4   0.3846154   0.9629630      0.7142857      0.8666667 0.7142857
## Class: 5   0.1538462   0.8148148      0.1666667      0.8000000 0.1666667
##             Recall        F1 Prevalence Detection Rate Detection Prevalence
## Class: 1 0.8333333 0.7692308  0.1791045     0.14925373            0.2089552
## Class: 2 0.9375000 0.9090909  0.2388060     0.22388060            0.2537313
## Class: 3 0.5384615 0.4666667  0.1940299     0.10447761            0.2537313
## Class: 4 0.3846154 0.5000000  0.1940299     0.07462687            0.1044776
## Class: 5 0.1538462 0.1600000  0.1940299     0.02985075            0.1791045
##          Balanced Accuracy
## Class: 1         0.8803030
## Class: 2         0.9491422
## Class: 3         0.6766382
## Class: 4         0.6737892
## Class: 5         0.4843305
```

```
print("SVM Metrics:")
```

```
## [1] "SVM Metrics:"
```

```
print(svm_metrics)
```

```
##          Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
## Class: 1   1.0000000   0.9272727      0.7500000      1.0000000 0.7500000
## Class: 2   1.0000000   0.9803922      0.9411765      1.0000000 0.9411765
## Class: 3   0.6923077   0.7407407      0.3913043      0.9090909 0.3913043
## Class: 4   0.0000000   1.0000000            NaN      0.8059701        NA
## Class: 5   0.3846154   0.8888889      0.4545455      0.8571429 0.4545455
##             Recall        F1 Prevalence Detection Rate Detection Prevalence
## Class: 1 1.0000000 0.8571429  0.1791045     0.17910448            0.2388060
## Class: 2 1.0000000 0.9696970  0.2388060     0.23880597            0.2537313
## Class: 3 0.6923077 0.5000000  0.1940299     0.13432836            0.3432836
## Class: 4 0.0000000        NA  0.1940299     0.00000000            0.0000000
## Class: 5 0.3846154 0.4166667  0.1940299     0.07462687            0.1641791
##          Balanced Accuracy
## Class: 1         0.9636364
## Class: 2         0.9901961
## Class: 3         0.7165242
## Class: 4         0.5000000
## Class: 5         0.6367521
```

#b. Performance Reporting for Land Mine Detection

1. Logistic Regression Analysis
Confusion Matrix Insights:

The model demonstrates high sensitivity for classes 1 (83.33%) and 2 (100%), indicating strong performance in correctly identifying positive instances of these classes.
However, it struggles with class 4, where sensitivity is 0%, indicating no correct classifications for this class.
Performance Metrics:

Accuracy: The overall accuracy of 58.21% suggests moderate performance on the dataset.
Precision: Precision is highest for class 2 (94.12%) but drops significantly for class 4 (0%).
Recall (Sensitivity): Excellent recall for class 2 (100%) but poor for class 4 (0%), signifying that the model misses all instances of class 4.
F1-Score: The F1-score is highest for class 2 (96.97%) and moderate for classes 1 (76.92%) and 3 (60%). Class 4's F1-score is undefined due to zero precision and recall.

Strengths:

Performs well on frequently occurring classes (e.g., class 2).
Achieves balanced accuracy above 80% for most classes.
Weaknesses:

Fails to capture class 4 effectively.
Struggles with minority classes, as seen in low sensitivity and F1-scores for certain classes.

2. Random Forest Analysis
Confusion Matrix Insights:

Shows good sensitivity for class 1 (83.33%) and class 2 (93.75%), indicating reliable predictions for these classes.
Sensitivity drops significantly for class 5 (15.38%), showing difficulty in identifying instances of this class.
Performance Metrics:

Accuracy: Matches logistic regression at 58.21%.
Precision: Class 2 has the highest precision (83.33%), while class 5 has the lowest (20%).
Recall (Sensitivity): Recall is strong for class 1 (83.33%) and class 2 (93.75%) but weak for classes 4 (38.46%) and 5 (15.38%).
F1-Score: The F1-score is highest for class 2 (88.23%) and moderate for classes 1 (76.92%) and 4 (47.62%).
Strengths:

Handles classes with higher prevalence better, achieving strong precision and recall for classes 1 and 2.
Balanced accuracy is good for most classes, with class 2 at 93.93%.
Weaknesses:

Performs poorly on minority classes like class 5, as reflected in low sensitivity and F1-scores.
Somewhat lower precision for rarer classes.

3. Support Vector Machine (SVM) Analysis
Confusion Matrix Insights:

SVM achieves perfect sensitivity (100%) for classes 1 and 2, indicating excellent identification of positive instances.
It struggles with class 4, where sensitivity remains 0%, meaning no correct classifications for this class.
Performance Metrics:

Accuracy: SVM achieves the highest accuracy of 62.69%, outperforming logistic regression and random forest.
Precision: High precision for class 2 (94.12%) and class 1 (75%), but zero for class 4.
Recall (Sensitivity): Outstanding recall for classes 1 and 2 (both 100%), but zero for class 4.
F1-Score: Highest F1-scores are for class 2 (96.97%) and class 1 (85.71%), with moderate scores for class 3 (54.05%).
Strengths:

Best overall accuracy and strong performance for dominant classes (classes 1 and 2).
Achieves high precision, recall, and F1-scores for most classes.
Weaknesses:

Fails to classify class 4, reflected in zero sensitivity and undefined F1-score.
Struggles to balance performance across all classes.

#c. Practical Implications

1. Logistic Regression:

Strengths: Suitable for scenarios where frequent classes (e.g., Class 2) are most important, with high precision and recall for such categories. Performs decently on Class 1 as well.

Weaknesses: Fails to identify rarer classes like Class 4, with sensitivity dropping to 0%. This makes it unsuitable for projects requiring comprehensive performance across all classes.

Recommendation: Logistic Regression could be used if the project prioritizes detecting frequent classes and others are less critical. For tasks requiring a balanced detection, consider more robust models like Random Forest or SVM.

2.Random Forest:

Strengths: Provides balanced performance across multiple classes, with reasonable sensitivity for Classes 1 and 2. Better robustness compared to Logistic Regression for varying class distributions.

Weaknesses: Struggles with rare classes, such as Class 5, where sensitivity is only 15.38%. Further adjustments are needed for minority classes.

Recommendation: Random Forest is a good choice for applications needing moderate accuracy for multiple classes. Adjusting class weights or employing sampling techniques could enhance its ability to detect rare classes.

3.Support Vector Machine (SVM):

Strengths: Achieves the highest overall accuracy and excellent performance for dominant classes (Classes 1 and 2) with perfect sensitivity.

Weaknesses: Completely misses Class 4 and performs less effectively on less prevalent classes. This could limit its use in scenarios requiring balanced performance.

Recommendation: SVM is ideal for applications focusing on critical or dominant classes. For broader applicability, additional refinements (e.g., oversampling or hybrid methods) are required to improve detection of minority classes.

Thus,
SVM demonstrates the best overall accuracy and is suitable for detecting dominant classes (e.g., Classes 1 and 2). However, it requires enhancements to detect minority classes like Class 4.

Random Forest offers more balanced detection than Logistic Regression but still struggles with rare classes like Class 5. It is a viable option for applications requiring moderate performance across multiple classes.

Logistic Regression is limited by poor detection of rare classes, making it less suitable for comprehensive tasks requiring equal importance for all classes.