Assessed Practical 3

Julian Bara-Mason | sID: 201674483

2023-06-09

Section 1

Introduction

This report presents an analysis of the classic machine learning mushroom dataset, which contains observations of gilled mushrooms along with their visual and olfactory characteristics. The objective of this report and analysis is to develop machine learning models that can accurately predict the edibility of mushrooms based on these attributes.

To predict the edibility of mushrooms, we apply logistic regression, decision trees, and random forests to the data. These models will be tuned for maximal predictive accuracy. Cross-validation will be utilised to evaluate the models and select the best-performing model.

The remainder of this report is organised as follows: Section 2 provides a brief overview of the data, the data preprocessing, model implementation, tuning, and evaluation. Section 3 presents and concludes the results of the analysis, highlighting the performance metrics and model selection outcomes.

Section 2

Data Overview

```
# Load data
data <- read.csv("mushrooms.csv")

# Loads required packages
library(caret)
```

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

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

```
library(pROC)
```

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

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

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

```
library(rpart)
library(ggplot2)
library(MASS)
library(rpart)
library(gridExtra)
library(randomForest)
```

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

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

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

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

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

```
# View data
head(data)
```

```
##      Edible CapShape CapSurface CapColor    Odor Height
## 1 Poisonous   Convex     Smooth    Brown Pungent   Tall
## 2    Edible   Convex     Smooth   Yellow  Almond  Short
## 3    Edible     Bell     Smooth    White   Anise   Tall
## 4 Poisonous   Convex      Scaly    White Pungent  Short
## 5    Edible   Convex     Smooth     Gray    None  Short
## 6    Edible   Convex      Scaly   Yellow  Almond  Short
```

```
summary(data)
```

```
##     Edible            CapShape          CapSurface          CapColor
##  Length:8124        Length:8124        Length:8124        Length:8124
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##      Odor              Height
##  Length:8124        Length:8124
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

```
str(data)
```

```
## 'data.frame':    8124 obs. of  6 variables:
##  $ Edible    : chr  "Poisonous" "Edible" "Edible" "Poisonous" ...
##  $ CapShape  : chr  "Convex" "Convex" "Bell" "Convex" ...
##  $ CapSurface: chr  "Smooth" "Smooth" "Smooth" "Scaly" ...
##  $ CapColor  : chr  "Brown" "Yellow" "White" "White" ...
##  $ Odor      : chr  "Pungent" "Almond" "Anise" "Pungent" ...
##  $ Height    : chr  "Tall" "Short" "Tall" "Short" ...
```

The dataset comprises of 8,124 observations of gilled mushrooms. Each observation is described by six categorical variables: Edible, CapShape, CapSurface, CapColor, Odor, and Height. These variables capture important visual and olfactory characteristics of the mushrooms that will serve as predictors for determining their edibility.

The target variable, Edible, indicates whether a mushroom is classified as edible or poisonous.

The remaining five fariables provide information about the physical characteristics of the mushrooms: 1. CapShape: describes the shape of the mushrooms cap, which can take on one of the following values: convex, bell, sunken, flat, knobbed, or conical. 2. CapSurface: represents the texture of the mushrooms cap, whcih can bee smooth, scaly, fibrous, or grooved. 3. CapColor: describes the color of the mushrooms cap and can assume various values like brown, yellow, white, gray, red, pink, purple, and green. 4. Odor: describes the odor type of the mushroom, which can assume one of the following: almond, anise, creosote, fishy, foul, musty, pungent, or none. 5. Height: indicates the height of the mushroom, classified as tall or short.

The categorical nature of the variables necessitates appropriate preprocessing and modelling techniques to effectively handle them.

Data Preprocessing

Missing Values

```
# Checks for missing values in the dataset
missing <- sum(is.na(data))
cat("There are", missing, "missing values in the dataset \n")
```

```
## There are 0 missing values in the dataset
```

The dataset has no missing values and is therefore considered complete.

Converting categorical data

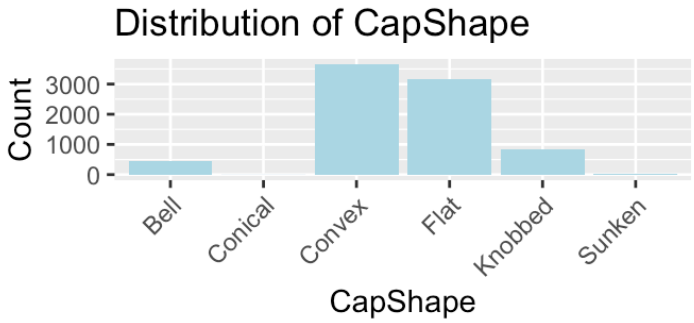
To correctly and efficiently handle categorical data, they must be converted to strings factors.

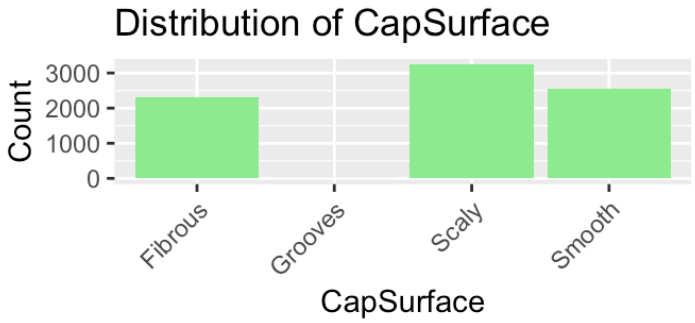
```
# Convert categorical data to factors
data$Edible <- as.factor(data$Edible)
data$CapShape <- as.factor(data$CapShape)
data$CapSurface <- as.factor(data$CapSurface)
data$CapColor <- as.factor(data$CapColor)
data$Odor <- as.factor(data$Odor)
data$Height <- as.factor(data$Height)

summary(data)
```

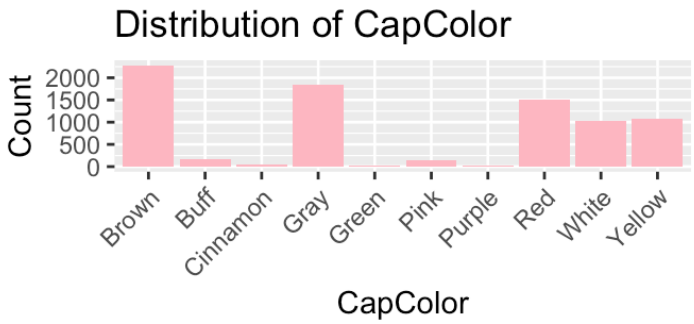
```
##        Edible        CapShape      CapSurface      CapColor         Odor
##  Edible   :4208   Bell   : 452   Fibrous:2320   Brown  :2284   None   :3528
##  Poisonous:3916   Conical:   4   Grooves:   4   Gray   :1840   Foul   :2160
##                   Convex :3656   Scaly  :3244   Red    :1500   Fishy  : 576
##                   Flat   :3152   Smooth :2556   Yellow :1072   Spicy  : 576
##                   Knobbed: 828                  White  :1040   Almond : 400
##                   Sunken :  32                  Buff   : 168   Anise  : 400
##                                                 (Other): 220   (Other): 484
##    Height
##  Short:4043
##  Tall :4081
##
##
##
##
##
```

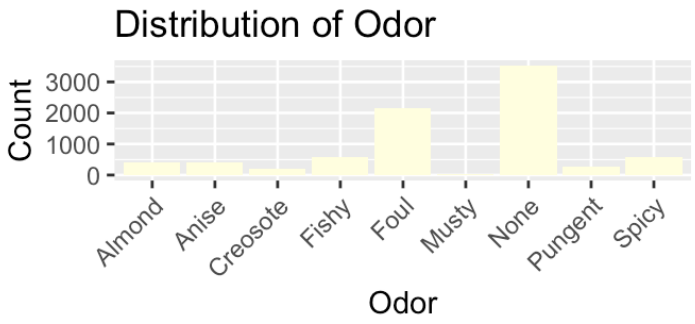
Data Visualisation

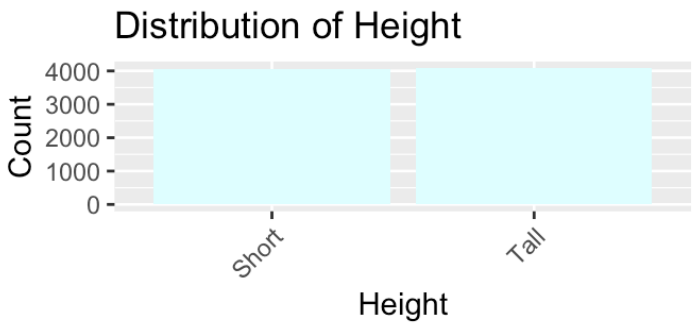
```
# Bar plot for CapShape
plot_cap_shape <- ggplot(data, aes(x = CapShape)) +
  geom_bar(fill = "lightblue") +
  labs(title = "Distribution of CapShape") +
  xlab("CapShape") +
  ylab("Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Bar plot for CapSurface
plot_cap_surface <- ggplot(data, aes(x = CapSurface)) +
  geom_bar(fill = "lightgreen") +
  labs(title = "Distribution of CapSurface") +
  xlab("CapSurface") +
  ylab("Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Bar plot for CapColor
plot_cap_color <- ggplot(data, aes(x = CapColor)) +
  geom_bar(fill = "lightpink") +
  labs(title = "Distribution of CapColor") +
  xlab("CapColor") +
  ylab("Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Bar plot for Odor
plot_odor <- ggplot(data, aes(x = Odor)) +
  geom_bar(fill = "lightyellow") +
  labs(title = "Distribution of Odor") +
  xlab("Odor") +
  ylab("Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Bar plot for Height
plot_height <- ggplot(data, aes(x = Height)) +
  geom_bar(fill = "lightcyan") +
  labs(title = "Distribution of Height") +
  xlab("Height") +
  ylab("Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Arrange the plots in a grid layout
grid_arrange <- grid.arrange(plot_cap_shape, plot_cap_surface, plot_cap_color, plot
_odor, plot_height, ncol = 2)
```





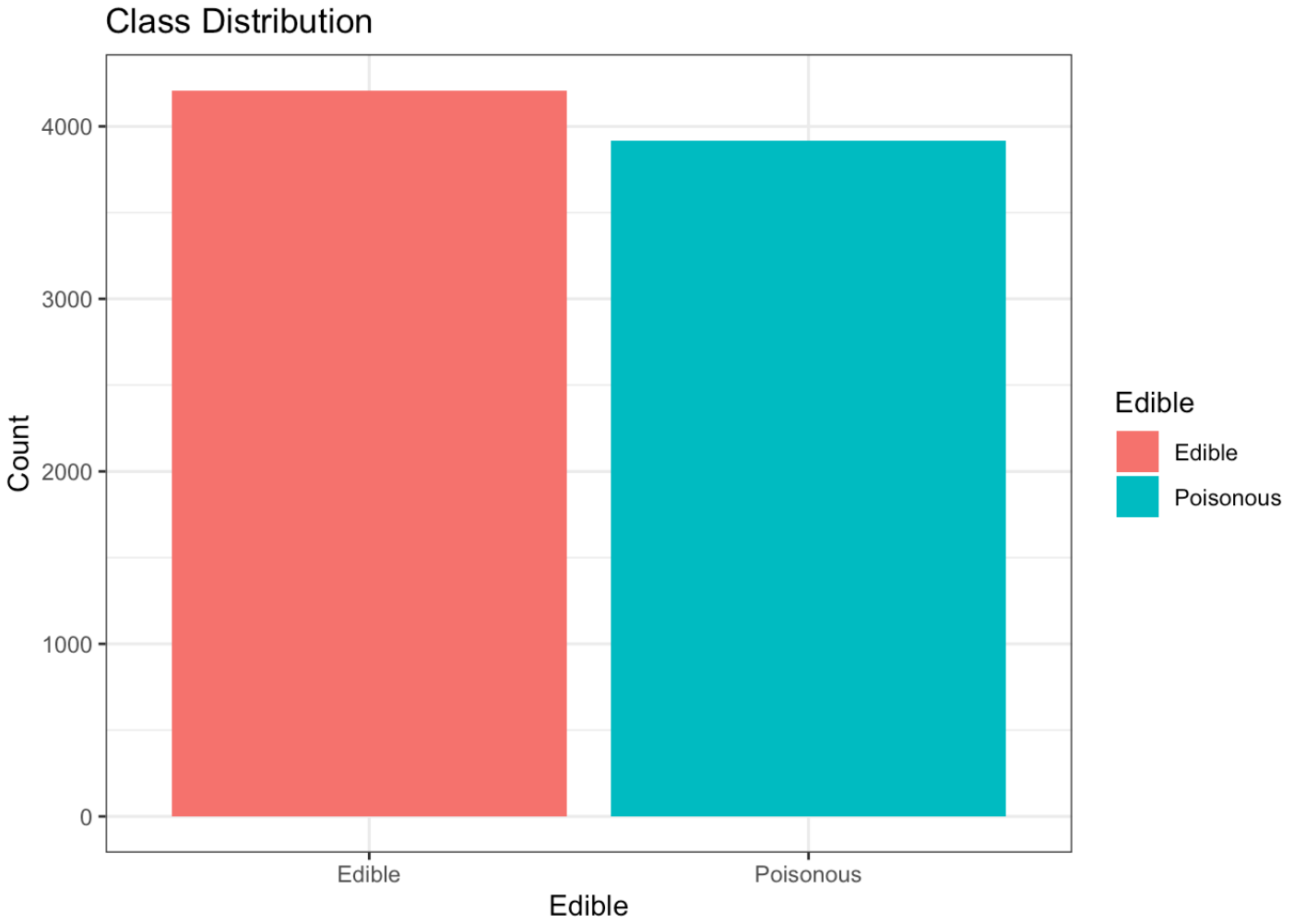






The visualizations provide insight into the distribution of all the input variables. From the output above, we can see: The CapShape of most mushrooms is Convex The CapSurface of most mushrooms is Scaly The CapColor of most mushrooms is Brown The Odor of most mushrooms is None The Height of mushrooms are about the same height.

```
# Bar plot for class distribution
plot_class <- ggplot(data, aes(x = Edible, fill = Edible)) +
  geom_bar() +
  labs(title = "Class Distribution") +
  xlab("Edible") +
  ylab("Count") +
  theme_bw()

# Display the plot
plot_class
```



From the Class Distribution plot above, there is a fair distribution of the classes, so there is no class imbalance in the dataset.

Split dataset to train and test set

The dataset was split into a train set to model & learn the data with, and a test set to predict and compare the predictive accuracy of the models.

A 70:30 split was used.

```
# Set the seed for reproducibility
set.seed(123)

#Split the data into training and testing sets
trainIndex <- createDataPartition(data$Edible, p = 0.7, list = FALSE)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]
```

Model Implementation

Logistic Regression Model

```
logistic_model <- glm(Edible ~ ., data=train_data, family=binomial)
```

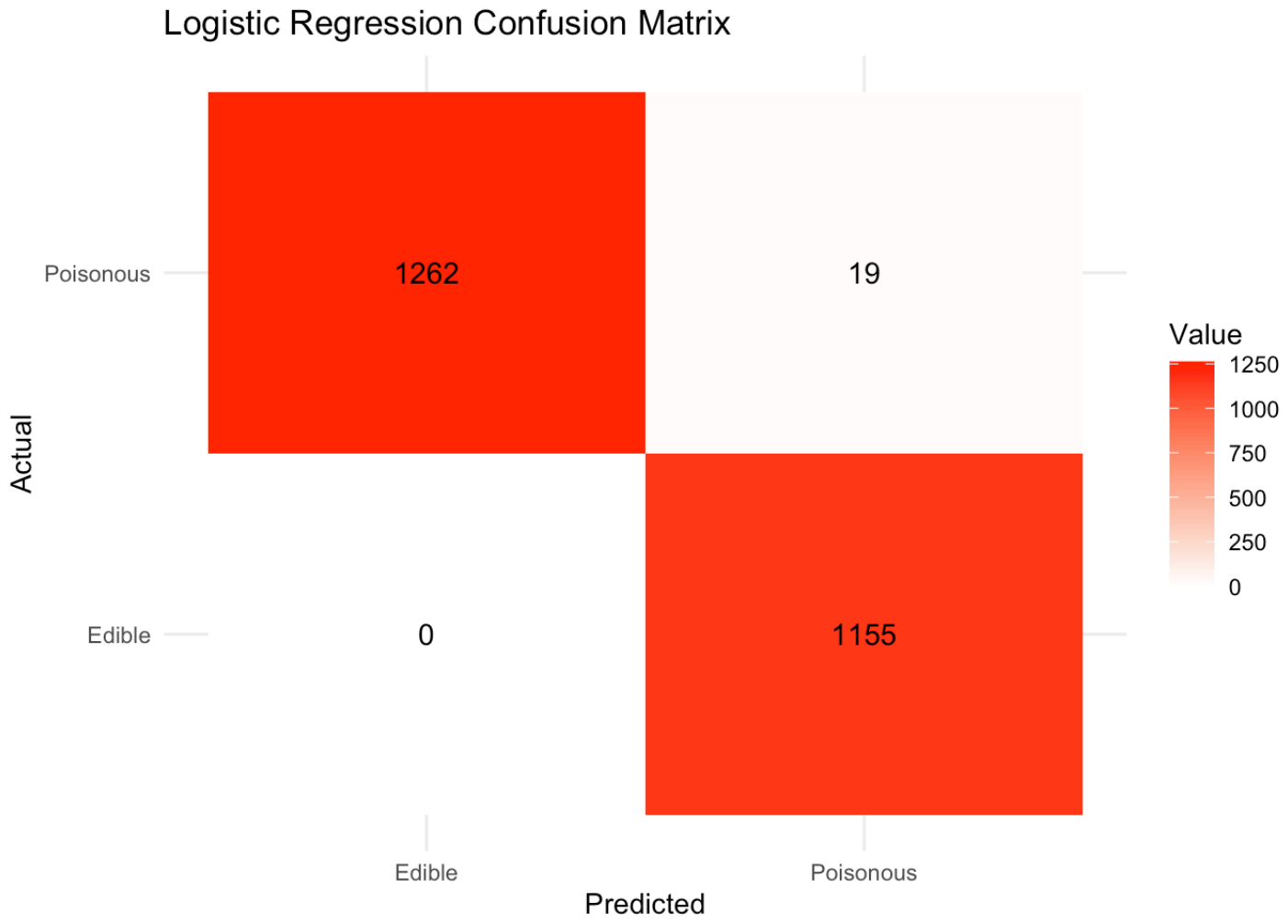
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Important to note the warning produced by the logistic model fit. The warning indicates that the logistic regression model encountered perfect separation, leading to numerical instability when estimating the model parameters. The model needs to be evaluated and tuned for better prediction power.

```
# Model Summary
summary(logistic_model)
```

```
##
## Call:
## glm(formula = Edible ~ ., family = binomial, data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.128    0.000    0.000    0.000    3.792
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -4.772e+01  4.224e+03  -0.011 0.990986
## CapShapeConical   -8.819e-01  3.554e+04   0.000 0.999980
## CapShapeConvex    -3.857e+00  5.967e-01  -6.465 1.02e-10 ***
## CapShapeFlat      -1.995e+00  3.775e-01  -5.284 1.26e-07 ***
## CapShapeKnobbed   -3.099e+00  6.431e-01  -4.820 1.44e-06 ***
## CapShapeSunken    -1.950e+01  1.462e+04  -0.001 0.998936
## CapSurfaceGrooves  2.712e+01  3.470e+04   0.001 0.999376
## CapSurfaceScaly    2.331e+00  5.784e-01   4.030 5.57e-05 ***
## CapSurfaceSmooth   9.511e-01  5.457e-01   1.743 0.081347 .
## CapColorBuff       2.773e+00  4.824e-01   5.749 8.99e-09 ***
## CapColorCinnamon  -1.893e+01  1.211e+04  -0.002 0.998753
## CapColorGray      -1.808e+01  1.566e+03  -0.012 0.990788
## CapColorGreen     -1.997e+01  2.225e+04  -0.001 0.999284
## CapColorPink       2.860e+00  4.545e-01   6.292 3.13e-10 ***
## CapColorPurple    -1.997e+01  2.060e+04  -0.001 0.999226
## CapColorRed       -1.832e+01  1.954e+03  -0.009 0.992518
## CapColorWhite      1.575e+00  4.184e-01   3.764 0.000167 ***
## CapColorYellow     2.460e+01  2.369e+03   0.010 0.991715
## OdorAnise         -4.689e-02  4.976e+03   0.000 0.999992
## OdorCreosote       9.146e+01  7.325e+03   0.012 0.990037
## OdorFishy          9.047e+01  5.794e+03   0.016 0.987543
## OdorFoul           9.072e+01  4.739e+03   0.019 0.984728
## OdorMusty          9.050e+01  1.820e+04   0.005 0.996034
## OdorNone           4.440e+01  4.224e+03   0.011 0.991612
## OdorPungent        7.240e+01  7.182e+03   0.010 0.991957
## OdorSpicy          9.043e+01  5.741e+03   0.016 0.987432
## HeightTall        -1.911e-02  3.154e-01  -0.061 0.951700
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 7877.92  on 5687  degrees of freedom
## Residual deviance:  310.56  on 5661  degrees of freedom
## AIC: 364.56
##
## Number of Fisher Scoring iterations: 22
```

```
actual_lr <- test_data$Edible

# Predict using the model on the test data
predicted_lr <- predict(logistic_model, newdata=test_data, type="response")

# Convert predicted and reference variables to factors with the same levels
predicted_lr <- ifelse(predicted_lr > 0.5, "Edible", "Poisonous")
predicted_lr_labels <- factor(predicted_lr, levels = c("Edible", "Poisonous"))

# Dataframe for actual and predicted values
lr_pred_df <- data.frame(
  Actual = actual_lr,
  Predicted = predicted_lr_labels
)

# Create a confusion matrix
confusion_matrix <- as.matrix(table(lr_pred_df$Actual, lr_pred_df$Predicted))
row_names <- rownames(confusion_matrix)

# Create the confusion matrix data frame
confusion_matrix_df <- data.frame(
  Actual = rep(row_names, each=length(row_names)),
  Predicted = rep(row_names, length(row_names)),
  Value = as.vector(confusion_matrix)
)
# Plot the confusion matrix
ggplot(data = confusion_matrix_df, aes(x = Predicted, y = Actual, fill = Value)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "red") +
  labs(x = "Predicted", y = "Actual", title = "Logistic Regression Confusion Matrix
") +
  theme_minimal()
```




```
# Extract performance metrics
cm <- confusionMatrix(lr_pred_df$Predicted, lr_pred_df$Actual)
accuracy_lr <- cm$overall["Accuracy"]
precision <- cm$byClass["Pos Pred Value"]
recall <- cm$byClass["Sensitivity"]
f1_score <- cm$byClass["F1"]

# Print the performance metrics
cat("Accuracy:", round(accuracy_lr*100, 3), "%")
```

```
## Accuracy: 0.78 %
```

As seen in the confusion matrix above and the Accuracy output, this model performs poorly in predicting the mushrooms edibility. Its accuracy is only 0.78%.

Logistic Regression Model Tuning

To tune and enhance the model's performance, ridge regression for logistic regression was used. By incorporating ridge regression, we mitigated the earlier warning about fitted probability being numerically 0 or 1, causing instability. Ridge regression's regularization helped to prevent such extreme predictions by shrinking the regression coefficients towards 0.

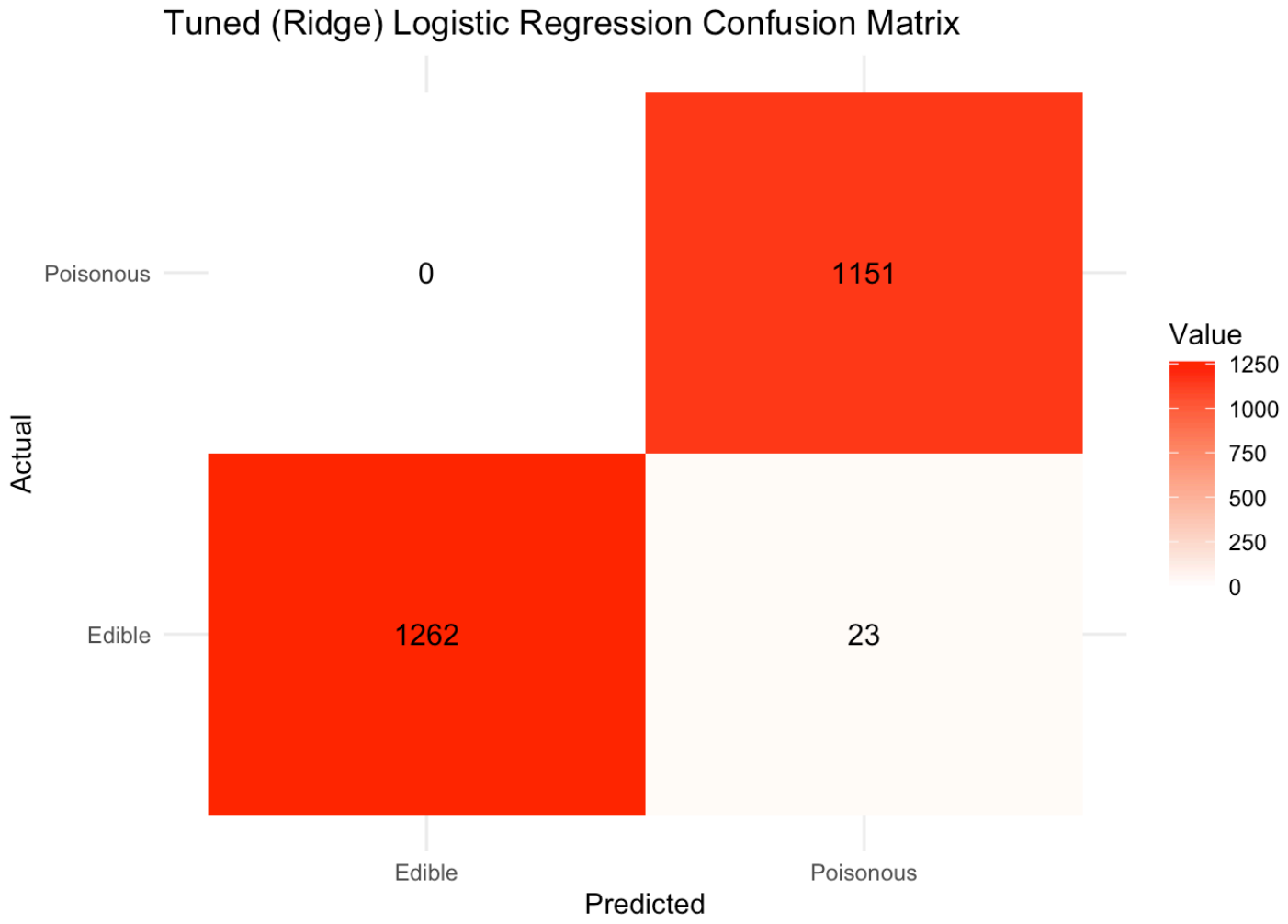
To determine the optimal alpha value for the regularization, cross-validation was employed to identify the alpha vaue that yields the best performance. performance.

```
# Convert the response variable to a factor with two levels
response <- factor(train_data$Edible, levels = c("Edible", "Poisonous"))

# Convert the predictor variables to a matrix
train_matrix <- model.matrix(Edible ~ ., data = train_data)

# Create train control with resampling
train_control <- trainControl(method = "cv", number = 5)

# Fit the ridge logistic regression model with cross-validation
ridge_model <- train(
  x = train_matrix,
  y = response,
  method = "glmnet",
  family = "binomial",
  trControl = train_control,
  tuneLength = 5,
  metric = "Accuracy"
)

# Extract the best alpha value
best_alpha <- ridge_model$bestTune$alpha
# Print the best alpha
cat("Best Alpha:", best_alpha, "\n")
```

```
## Best Alpha: 1
```

From the cross-validation, the best alph value for the ridge regression is 1.

```
# Predict on test data
test_matrix <- model.matrix(Edible ~ ., data = test_data)
predictions <- predict(ridge_model, newdata = test_matrix, type = "raw")  # Use "ra
w" instead of "response"

# Convert predictions to binary labels
predicted_labels <- ifelse(predictions == "Edible", "Edible", "Poisonous")
predicted_lr_labels_ridge <- factor(predicted_labels, levels = c("Edible", "Poisono
us"))

# Dataframe for actual and predicted values
lr_pred_df_ridge <- data.frame(
  Actual = test_data$Edible,
  Predicted = predicted_lr_labels_ridge
)

# Create confusion matrix
cm_ridge <- table(lr_pred_df_ridge$Actual, lr_pred_df_ridge$Predicted)

# Create the confusion matrix data frame
confusion_matrix_ridge_df <- data.frame(
  Actual = rep(rownames(cm_ridge), each = nrow(cm_ridge)),
  Predicted = rep(colnames(cm_ridge), ncol(cm_ridge)),
  Value = as.vector(cm_ridge)
)

# Plot the confusion matrix
ggplot(data = confusion_matrix_ridge_df, aes(x = Predicted, y = Actual, fill = Valu
e)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "red") +
  labs(x = "Predicted", y = "Actual", title = "Tuned (Ridge) Logistic Regression Co
nfusion Matrix") +
  theme_minimal()
```




```
# Calculate accuracy
accuracy <- sum(predicted_labels == test_data$Edible) / length(test_data$Edible) *
100
# Print accuracy
print(paste("Accuracy of Tuned (Ridge) Logistic Regression Model:", round(accuracy,
2), "%"))
```

```
## [1] "Accuracy of Tuned (Ridge) Logistic Regression Model: 99.06 %"
```

Logistic Regression Model Evaluation

The initial logistic regression was a poor model with only a 0.78% accuracy. However, by tuning it with Ridge regression (alpha = 0.039), the model was significantly improved with a new accuracy of 99.1%.

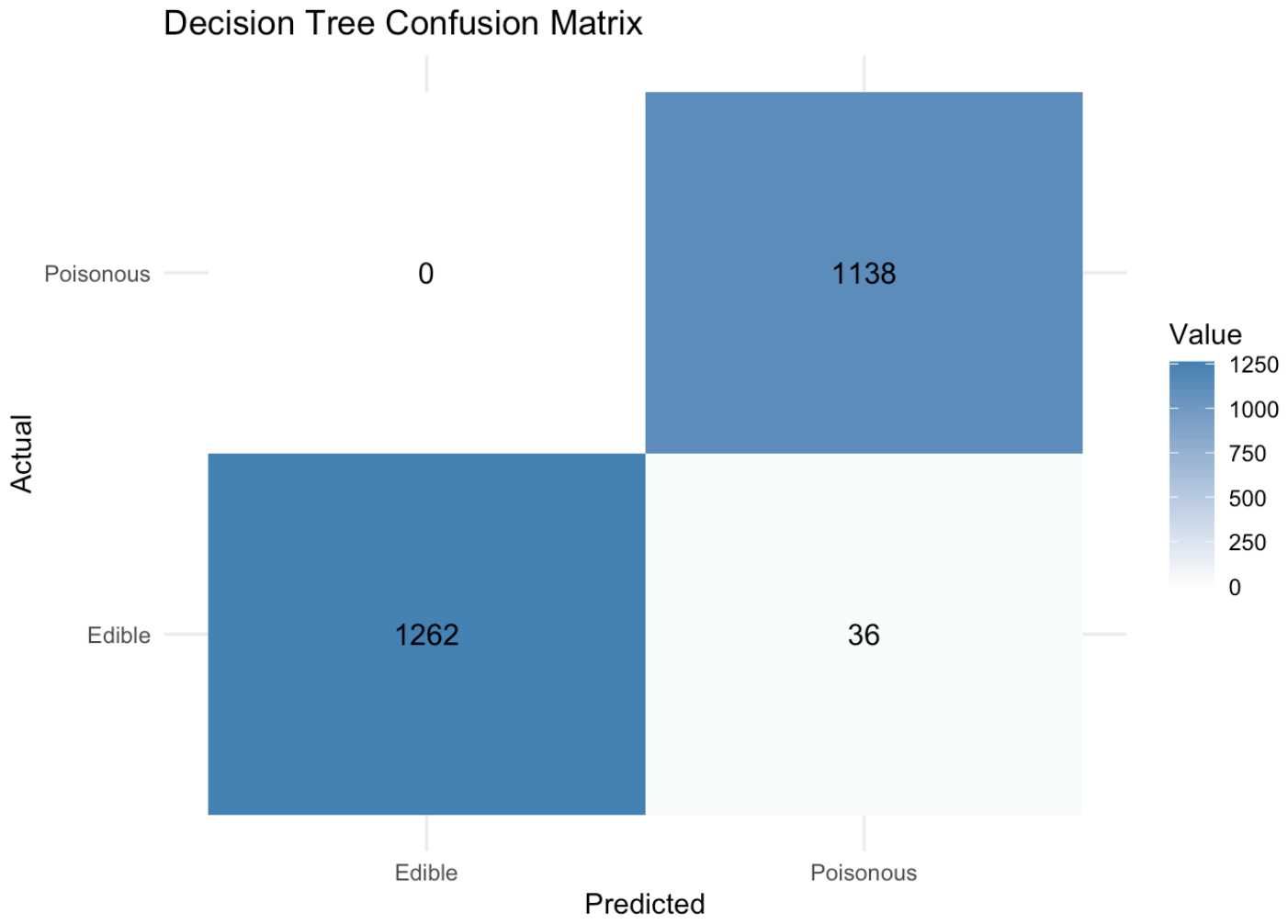
Decision Tree Model

A decision tree model was employed to classify mushrooms as either edible or poisonous.

```
# Train and tune the decision tree model
decisionTreeModel <- rpart(Edible ~ ., data = train_data, method = "class")
summary(decisionTreeModel)
```

```
## Call:
## rpart(formula = Edible ~ ., data = train_data, method = "class")
##   n= 5688
##
##          CP nsplit  rel error     xerror        xstd
## 1 0.9693654      0 1.00000000 1.00000000 0.013743674
## 2 0.0100000      1 0.03063457 0.03063457 0.003317733
##
## Variable importance
##       Odor   CapColor   CapShape CapSurface
##         77          9          8          6
##
## Node number 1: 5688 observations,    complexity param=0.9693654
##   predicted class=Edible     expected loss=0.4820675  P(node) =1
##     class counts:  2946  2742
##    probabilities: 0.518 0.482
##   left son=2 (3030 obs) right son=3 (2658 obs)
##   Primary splits:
##       Odor       splits as  LLRRRRLRR,  improve=2676.9990000, (0 missing)
##       CapSurface splits as  LRRR,       improve= 109.3526000, (0 missing)
##       CapShape   splits as  LRRRRL,     improve= 103.4139000, (0 missing)
##       CapColor   splits as  LRLLLRLRLR, improve=  88.9771100, (0 missing)
##       Height     splits as  LR,         improve=   0.3097844, (0 missing)
##   Surrogate splits:
##       CapColor   splits as  LRLLLLLRLR, agree=0.589, adj=0.120, (0 split)
##       CapShape   splits as  LLLLRL,     agree=0.580, adj=0.102, (0 split)
##       CapSurface splits as  LLRR,       agree=0.570, adj=0.081, (0 split)
##
## Node number 2: 3030 observations
##   predicted class=Edible     expected loss=0.02772277  P(node) =0.5327004
##     class counts:  2946    84
##    probabilities: 0.972 0.028
##
## Node number 3: 2658 observations
##   predicted class=Poisonous  expected loss=0  P(node) =0.4672996
##     class counts:     0  2658
##    probabilities: 0.000 1.000
```

```
# Predict on test data using the decision tree model
predicted_labels_tree <- predict(decisionTreeModel, newdata = test_data, type = "cl
ass")

# Convert predicted labels to factor
predicted_tree_labels <- factor(predicted_labels_tree, levels = c("Edible", "Poison
ous"))

# Create dataframe for actual and predicted values
tree_pred_df <- data.frame(
  Actual = test_data$Edible,
  Predicted = predicted_tree_labels
)

# Create confusion matrix
cm_tree <- table(tree_pred_df$Actual, tree_pred_df$Predicted)

# Create the confusion matrix data frame
confusion_matrix_tree_df <- data.frame(
  Actual = rep(rownames(cm_tree), each = nrow(cm_tree)),
  Predicted = rep(colnames(cm_tree), ncol(cm_tree)),
  Value = as.vector(cm_tree)
)

# Plot the confusion matrix
ggplot(data = confusion_matrix_tree_df, aes(x = Predicted, y = Actual, fill = Value
)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(x = "Predicted", y = "Actual", title = "Decision Tree Confusion Matrix") +
  theme_minimal()
```




```
# Calculate accuracy
accuracy_tree <- sum(predicted_labels_tree == test_data$Edible) / nrow(test_data)

# Print the accuracy
print(paste("Accuracy of Decision Tree:", round(accuracy_tree * 100, 2), "%"))
```

```
## [1] "Accuracy of Decision Tree: 98.52 %"
```

The decision tree is appears to be a good predictive model with an accuracy of 98.52%. It can be tuned for max performance by adjusting its parameters and using cross-validation to evaluate different combinations of the parameter values. The complexity parameter, which controls the complexity of the tree, will be adjusted.

Decision Tree Model Tuning

The decision tree model was fine-tuned using random search and grid search techniques to classify mushrooms as either edible or poisonous. The random search involved randomly selecting hyperparameters within a predefined range, while the grid search exhaustively tested a range of hyperparameters. Cross-validation was employed to evaluate each model's performance and the better model was selected as the final decision tree model.

```
# Set up the tuning grid for random search
random_tune_grid <- expand.grid(cp = runif(10, 0.01, 0.5))

# Set up the tuning grid for grid search
grid_tune_grid <- expand.grid(cp = seq(0.01, 0.5, by = 0.01))

num_folds <- 5  # Define the number of cross-validation folds
# Perform cross-validation with parameter tuning for random search
random_train_control <- trainControl(method = "cv", number = num_folds)
random_tuned_tree_model <- train(
  Edible ~ .,
  data = train_data,
  method = "rpart",
  tuneGrid = random_tune_grid,
  trControl = random_train_control
)

# Perform cross-validation with parameter tuning for grid search
grid_train_control <- trainControl(method = "cv", number = num_folds)
grid_tuned_tree_model <- train(
  Edible ~ .,
  data = train_data,
  method = "rpart",
  tuneGrid = grid_tune_grid,
  trControl = grid_train_control
)

#print(random_tuned_tree_model)
#print(grid_tuned_tree_model)

# Access the resampling results for each method
random_resampling_results <- random_tuned_tree_model$resample
grid_resampling_results <- grid_tuned_tree_model$resample

# Calculate the average accuracy for each method
random_average_accuracy <- mean(random_resampling_results$Accuracy)
grid_average_accuracy <- mean(grid_resampling_results$Accuracy)

# Print the average accuracies
#cat("Random Search Average Accuracy:", random_average_accuracy, "\n")
#cat("Grid Search Average Accuracy:", grid_average_accuracy, "\n")

### Confusion Matrix
# Predict on test data using the random_tuned_tree_model
predicted_labels_random <- predict(random_tuned_tree_model, newdata = test_data, ty
pe = "raw")
predicted_labels_grid <- predict(grid_tuned_tree_model, newdata = test_data, type =
"raw")

# Convert predicted labels to factor
```

```
predicted_random_labels <- factor(predicted_labels_random, levels = c("Edible", "Po
isonous"))
predicted_grid_labels <- factor(predicted_labels_grid, levels = c("Edible", "Poison
ous"))

# Create dataframe for actual and predicted values
random_pred_df <- data.frame(
  Actual = test_data$Edible,
  Predicted = predicted_random_labels
)
grid_pred_df <- data.frame(
  Actual = test_data$Edible,
  Predicted = predicted_grid_labels
)

# Create confusion matrix
cm_random <- table(random_pred_df$Actual, random_pred_df$Predicted)
cm_grid <- table(grid_pred_df$Actual, grid_pred_df$Predicted)

# Create the confusion matrix data frame
confusion_matrix_random_df <- data.frame(
  Actual = rep(rownames(cm_random), each = nrow(cm_random)),
  Predicted = rep(colnames(cm_random), ncol(cm_random)),
  Value = as.vector(cm_random)
)
confusion_matrix_grid_df <- data.frame(
  Actual = rep(rownames(cm_grid), each = nrow(cm_grid)),
  Predicted = rep(colnames(cm_grid), ncol(cm_grid)),
  Value = as.vector(cm_grid)
)

# Plot the confusion matrix for the random_tuned_tree_model
plot_random <- ggplot(data = confusion_matrix_random_df, aes(x = Predicted, y = Act
ual, fill = Value)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(x = "Predicted", y = "Actual", title = "Random") +
  theme_minimal()

# Plot the confusion matrix for the grid_tuned_tree_model
plot_grid <- ggplot(data = confusion_matrix_grid_df, aes(x = Predicted, y = Actual,
fill = Value)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(x = "Predicted", y = "Actual", title = "Grid") +
  theme_minimal()
```

```
# Print the accuracy
#print(paste("Accuracy of Decision Tree (Tuned):", round(average_accuracy * 100, 2)
, "%"))
# Create Confusion Matrix for each model
dt_cm <- confusionMatrix(data=tree_pred_df$Predicted, reference = tree_pred_df$Actu
al)
random_cm <- confusionMatrix(data=random_pred_df$Predicted, reference = random_pred
_df$Actual)
grid_cm <- confusionMatrix(data=grid_pred_df$Predicted, reference = grid_pred_df$Ac
tual)

# Calculate the performance metrics for random_tuned_tree_model
random_accuracy <- random_cm$overall["Accuracy"]
random_precision <- random_cm$byClass["Precision"]
random_recall <- random_cm$byClass["Recall"]
random_f1 <- random_cm$byClass["F1"]

# Calculate the performance metrics for grid_tuned_tree_model
grid_accuracy <- grid_cm$overall["Accuracy"]
grid_precision <- grid_cm$byClass["Precision"]
grid_recall <- grid_cm$byClass["Recall"]
grid_f1 <- grid_cm$byClass["F1"]

# Create a data frame to hold the performance metrics
metrics_df <- data.frame(
  Metric = rep(c("Accuracy", "Precision", "Recall", "F1"), 2),
  Model = rep(c("Random Tuned Tree", "Grid-search Tuned Tree"), each = 4),
  Score = c(
    random_accuracy, grid_accuracy,
    random_precision, grid_precision,
    random_recall, grid_recall,
    random_f1, grid_f1
  )
)

# Plot the bar plot with value labels
bar_plot <- ggplot(metrics_df, aes(x = Metric, y = Score, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(Score, 2)), position = position_dodge(width = 0.9), v
just = -0.5) +
  labs(title = "Performance Metrics Comparison", x = "Metric", y = "Score") +
  scale_fill_manual(values = c("Random Tuned Tree" = "steelblue", "Grid-search Tune
d Tree" = "darkorange")) +
  theme_minimal()
```

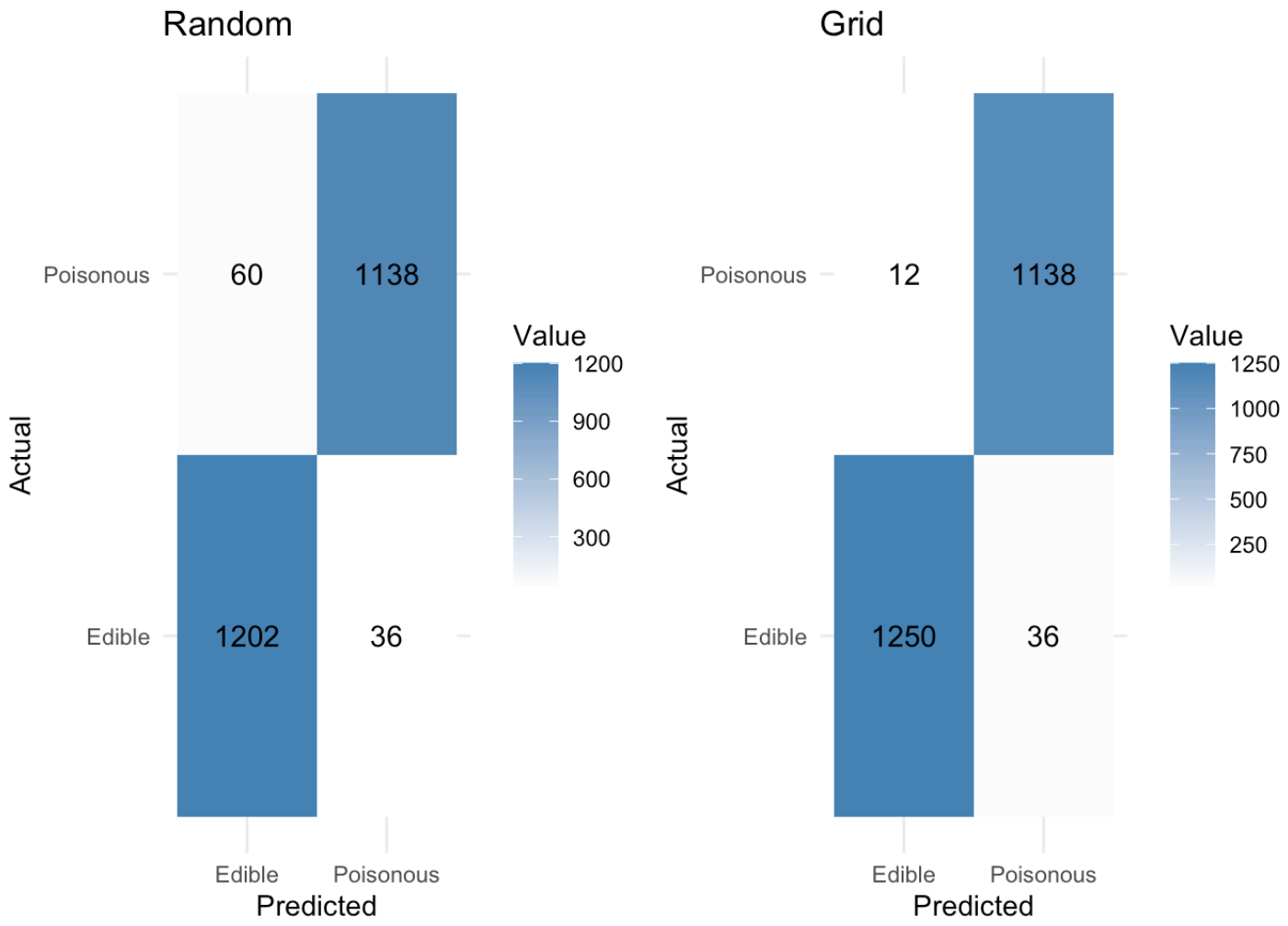
Decision Tree Model Evaluation

The fine-tuned decision tree models (Random and Grid) perform slightly worse than the first decision tree model, which had an accuracy score of 98.52%, while Random and Grid 96% and 95% respectively (See barplot above).

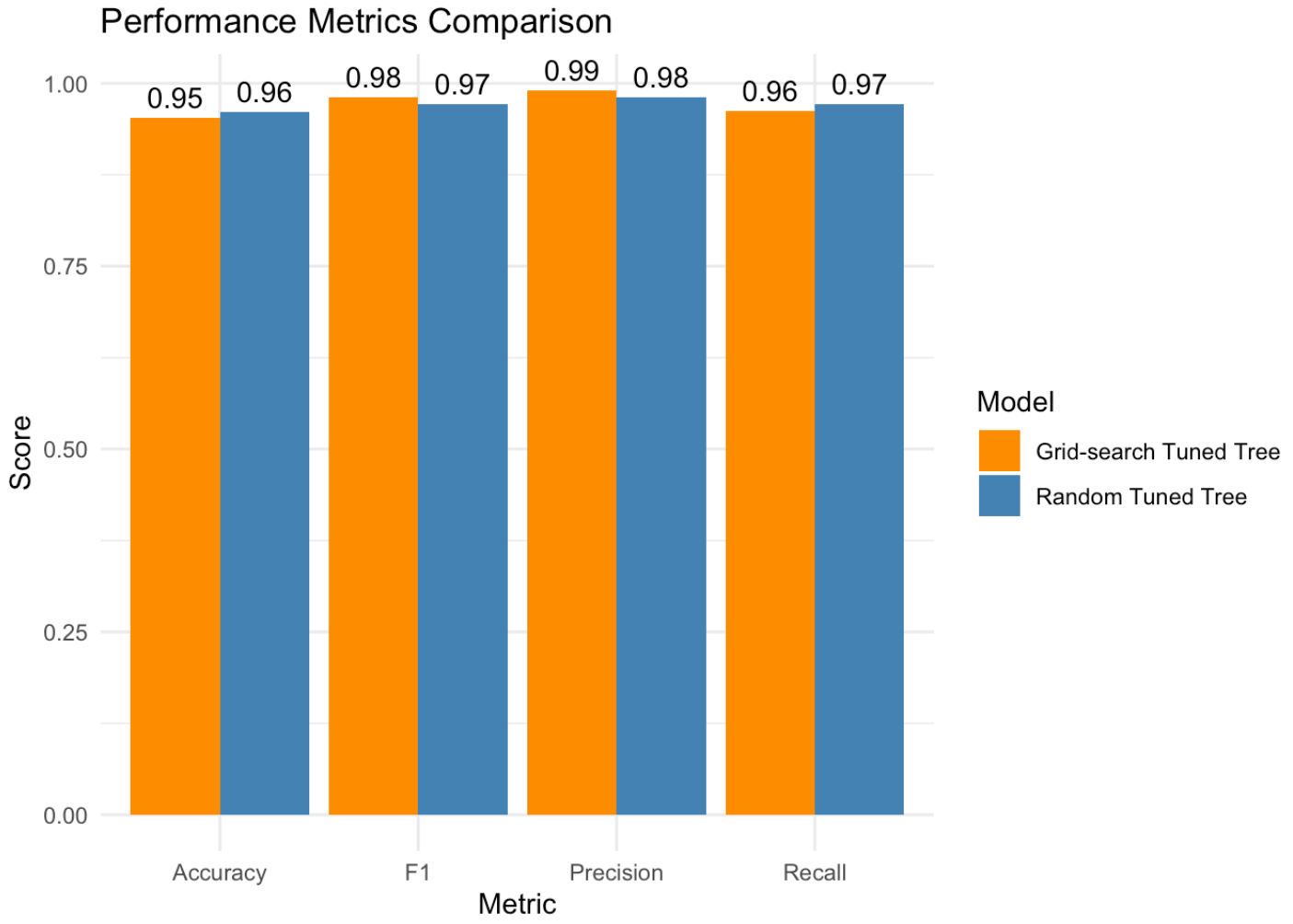
While Random and Grid were less accurate, one of them was selected as the final model going forward because they were both cross-validated 5-fold while the original decision tree model was not.

By examining the Confusion Matrix of both models in the plot below, it can be seen that the Random Tuned model misclassified 60 edible mushrooms as poisonous and 36 poisonous mushrooms as edible. On the other hand, the Grid Searched Model misclassified 12 edible mushrooms as poisonous and 36 poisonous mushrooms as edible.

```
# Arrange the plots side by side
combined_plots <- grid.arrange(plot_random, plot_grid, ncol = 2)
```




```
# Display the bar plot
print(bar_plot)
```



The barplot above shows the performance metrics of the Random Tuned Tree and the Grid Searched Tree.

The Random Tuned Tree has a higher accuracy score at 96% than the Grid Searched Tree's 95%. However, the Random Tuned Tree had a lower F1 score at 97% than the Grid Searched Tree's 98%. The F1 score balances the precision score and recall score.

The Random Tuned Tree misclassified a total of 96 mushrooms, while the Grid Searched Tree misclassified a total of 48 mushrooms. Of these misclassifications, both models misclassified 36 poisonous mushrooms as edible.

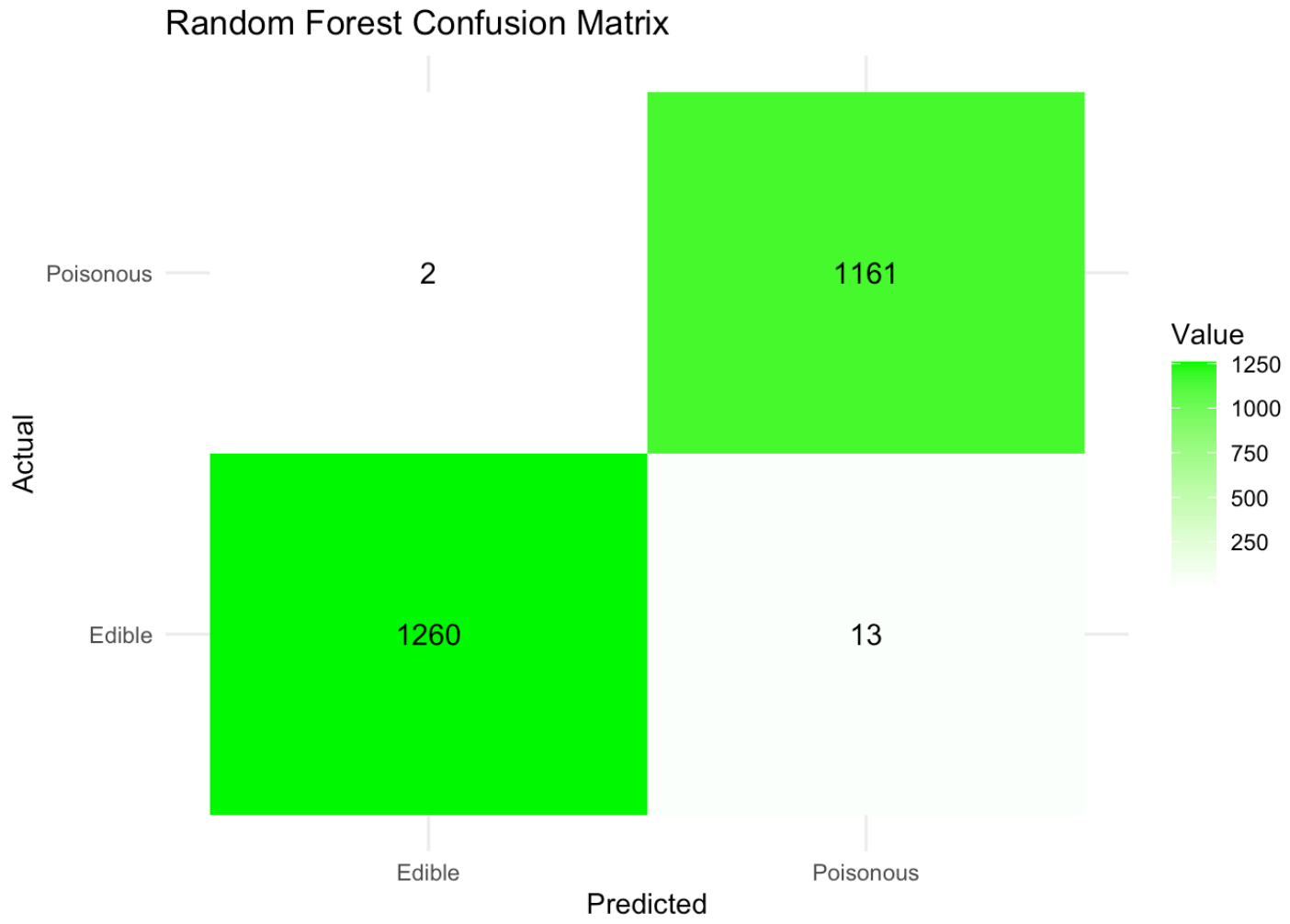
Based on these results, we can conclude that both models perform well in classifying mushrooms as edible or poisonous. However, the Grid Searched Model shows better overall performance with higher F1 score and lower misclassifications. The Grid Searched Decision Tree model is the final Decision Tree.

Random Forest Model

Random Forests is an ensemble method that combines the predictions of multiple individual decision trees to make more accurate and robust predictions.

```
# Define the train control for cross-validation
ctrl <- trainControl(method = "cv", number = 5)

# Fit Random Forest
rf_model <- train(
  Edible ~ .,
  data = train_data,
  method = "rf",
  trControl = ctrl
)
#summary(rf_model)
```

```
# Predict Random Forest Model on test data
rf_predictions <- predict(rf_model, newdata = test_data)
# Convert predicted labels to factor
rf_labels <- factor(rf_predictions, levels = c("Edible", "Poisonous"))

# Create dataframe for actual and predicted values
rf_pred_df <- data.frame(
  Actual = test_data$Edible,
  Predicted = rf_labels
)

# Create confusion matrix
cm_rf <- table(rf_pred_df$Actual, rf_pred_df$Predicted)

# Create the confusion matrix data frame
confusion_matrix_rf_df <- data.frame(
  Actual = rep(rownames(cm_rf), each = nrow(cm_rf)),
  Predicted = rep(colnames(cm_rf), ncol(cm_rf)),
  Value = as.vector(cm_rf)
)

# Calculate accuracy
accuracy_rf <- sum(diag(cm_rf)) / sum(cm_rf)

# Plot the confusion matrix
ggplot(data = confusion_matrix_rf_df, aes(x = Predicted, y = Actual, fill = Value))
+
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "green") +
  labs(x = "Predicted", y = "Actual", title = "Random Forest Confusion Matrix") +
  theme_minimal()
```




```
# Access the resampling results from the random forest model
rf_resamples <- rf_model$resample

# Calculate the average accuracy across all resamples
rf_accuracy <- mean(rf_resamples$Accuracy)

# Print the accuracy percentage of the random forest model
cat("Random Forest Accuracy: ", rf_accuracy * 100, "%\n")
```

```
## Random Forest Accuracy:  99.1737 %
```

The Random Forest model appears to be a strong predictive model for the mushroom data with an accuracy of 99.16% and only 18 wrong predictions (refer to confusion matrix plot above in green).

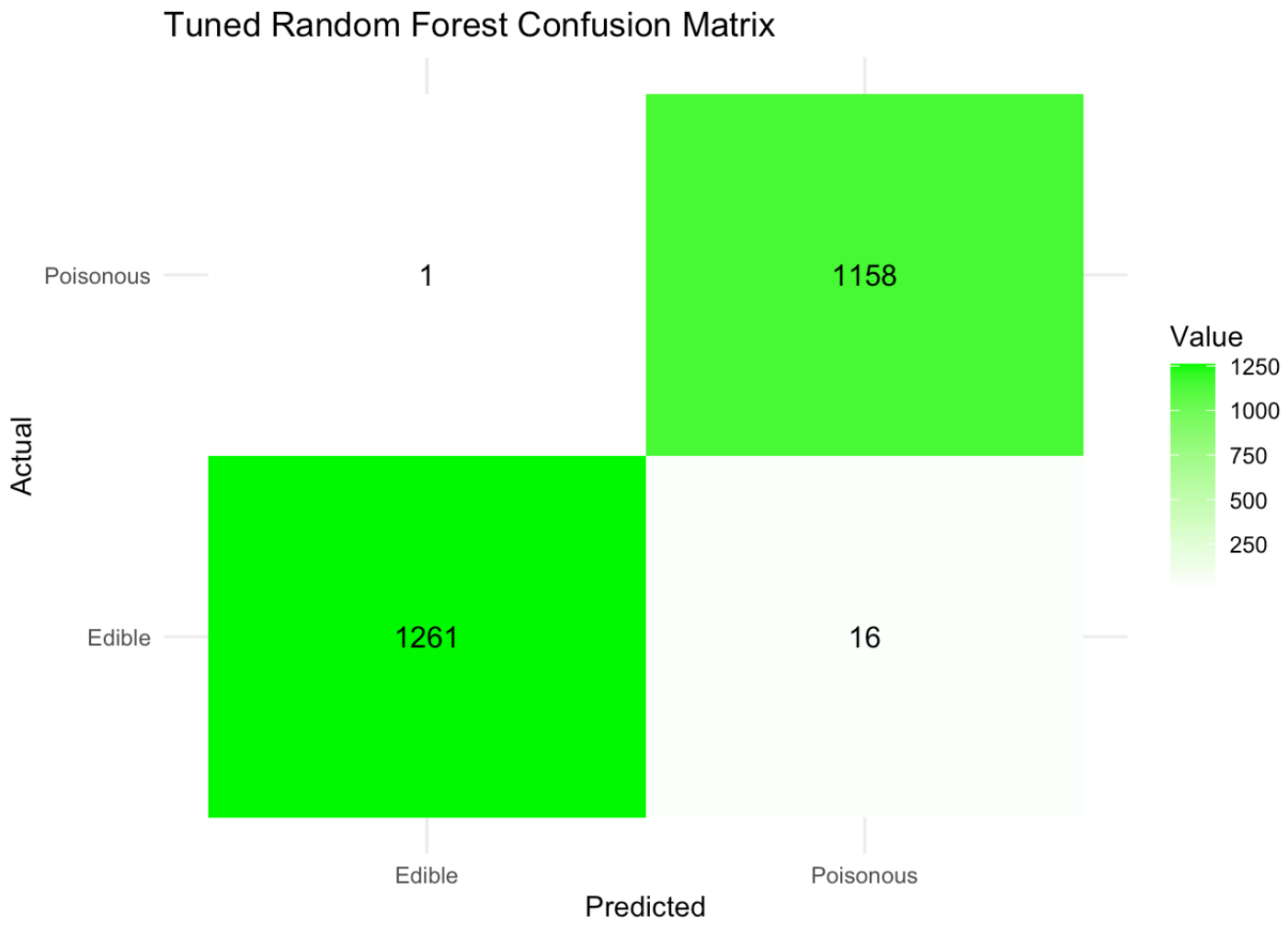
Random Forest Model

The random forest model was tuned by exploring different values for the mtry parameter, which determines the number of features randomly sampled at each split. A tuning grid was created with mtry values ranging from 1 to 5. Cross-validation was performed using 5-fold validation to evaluate the model's performance.

```
# Creates a tuning grid for the complexity parameter
```

```
tune_grid <- expand.grid(mtry = c(1, 2, 3, 4, 5))

# Perform cross-validation with parameter tuning
train_control <- trainControl(method = "cv", number = 5)  # 5-fold cross-validation
rf_model_tuned <- train(
  Edible ~ .,
  data = train_data,
  method = "rf",
  trControl = train_control,
  tuneGrid = tune_grid
)

# Get the best parameter value
best_mtry <- rf_model_tuned$bestTune$mtry

# Train the final random forest model with the best parameter
final_rf_model <- randomForest(
  Edible ~ .,
  data = train_data,
  mtry = best_mtry,
  importance = TRUE
)

# Make predictions on the test data using the final model
rf_predictions <- predict(final_rf_model, newdata = test_data, type = "class")

# Convert predicted labels to factor
rf_labels <- factor(rf_predictions, levels = c("Edible", "Poisonous"))

# Create dataframe for actual and predicted values
rf_pred_df <- data.frame(
  Actual = test_data$Edible,
  Predicted = rf_labels
)

# Create confusion matrix
rf_confusion_matrix <- table(rf_pred_df$Actual, rf_pred_df$Predicted)

# Create the confusion matrix data frame
rf_confusion_matrix_df <- data.frame(
  Actual = rep(rownames(rf_confusion_matrix), each = nrow(rf_confusion_matrix)),
  Predicted = rep(colnames(rf_confusion_matrix), ncol(rf_confusion_matrix)),
  Value = as.vector(rf_confusion_matrix)
)

# Plot the confusion matrix
ggplot(data = rf_confusion_matrix_df, aes(x = Predicted, y = Actual, fill = Value))
+
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "green") +
```

```
  labs(x = "Predicted", y = "Actual", title = "Tuned Random Forest Confusion Matrix
") +
  theme_minimal()
```




```
# Access the resampling results from the random forest model
rf_resamples_tuned <- rf_model_tuned$resample

# Calculate the average accuracy across all resamples
rf_accuracy_tuned <- mean(rf_resamples_tuned$Accuracy)

# Print the accuracy percentage of the random forest model
cat("Random Forest Accuracy: ", rf_accuracy_tuned * 100, "%\n")
```

```
## Random Forest Accuracy:  98.85721 %
```

Looking at the Tuned Random Forest Confusion Matrix and its accuracy (98.86%), it is less accurate than the first Random Forest Model.

Section 3

Model Selection

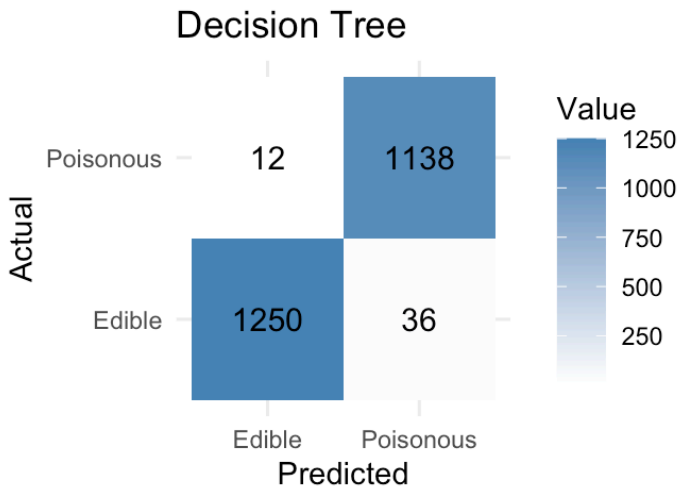
All three models have been tuned for better predictive power. The tuned models will be evaluated using cross-validation to determine the best model. The exception is the random forest model, whose tuned model performed slightly worse. So the initial random forest model will be used here.

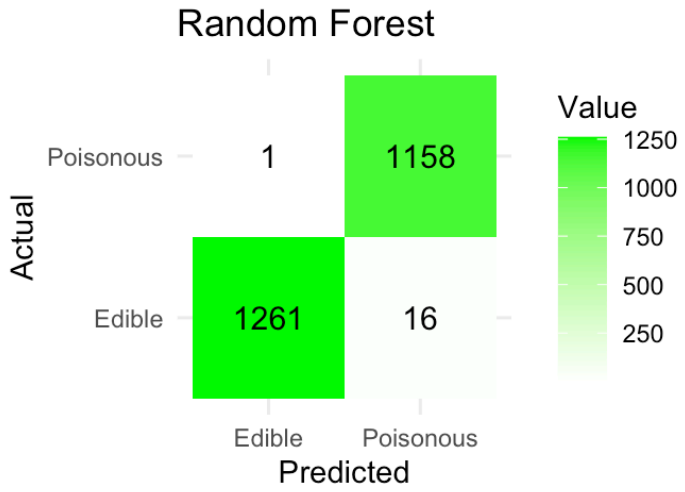
```
# Create list of models
models <- list(
  Ridge_Logistic = ridge_model,
  Decision_Tree = grid_tuned_tree_model,
  Random_Forest = rf_model
)
# Compare models
results <- resamples(models)
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: Ridge_Logistic, Decision_Tree, Random_Forest
## Number of resamples: 5
##
## Accuracy
##                      Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Ridge_Logistic 0.9859402 0.9894459 0.9894644 0.9903316 0.9912049 0.9956025    0
## Decision_Tree  0.9718805 0.9789104 0.9806508 0.9789037 0.9815303 0.9815466    0
## Random_Forest  0.9850615 0.9912049 0.9920844 0.9917370 0.9947276 0.9956063    0
##
## Kappa
##                      Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## Ridge_Logistic 0.9718178 0.9788482 0.9788855 0.9806236 0.9823758 0.9911907    0
## Decision_Tree  0.9436566 0.9577372 0.9612217 0.9577275 0.9630055 0.9630164    0
## Random_Forest  0.9700657 0.9823780 0.9841412 0.9834443 0.9894367 0.9912002    0
```

```
ridge_cm <- confusionMatrix(data=lr_pred_df_ridge$Predicted, reference = lr_pred_df
_ridge$Actual)
grid_cm <- confusionMatrix(data=grid_pred_df$Predicted, reference = grid_pred_df$Ac
tual)
forest_cm <- confusionMatrix(data = rf_pred_df$Predicted, reference = rf_pred_df$Ac
tual)

cm_list <- list(
  Logistic_Regression = ridge_cm,
  Decision_Tree = grid_cm,
  Random_Forest = forest_cm
)

# Extract the performance metrics from the confusion matrix objects
metrics <- lapply(seq_along(cm_list), function(i) {
  data.frame(
    Model = names(cm_list)[i],
    Accuracy = cm_list[[i]]$overall["Accuracy"],
    F1 = cm_list[[i]]$byClass["F1"],
    Precision = cm_list[[i]]$byClass["Precision"],
    Recall = cm_list[[i]]$byClass["Recall"],
    row.names = NULL
  )
})

# Combine the metrics into a single data frame
model_metrics <- do.call(rbind, metrics)
# Reshape the data for plotting
metrics_long <- tidyr::pivot_longer(model_metrics, cols = c(Accuracy, F1, Precision
, Recall), names_to = "Metric", values_to = "Score")

# Plot the bar plot
bar_plot <- ggplot(metrics_long, aes(x = Metric, y = Score, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(Score, 2)), position = position_dodge(width = 0.9), v
just = -0.5) +
  labs(title = "Model Performance Metrics Comparison", x = "Metric", y = "Score") +
  scale_fill_manual(values = c("Logistic_Regression" = "red", "Decision_Tree" = "st
eelblue", "Random_Forest" = "green")) +
  theme_minimal()
```

```
# Plot the logistic regression confusion matrix
plot_logistic <- ggplot(data = confusion_matrix_ridge_df, aes(x = Predicted, y = Ac
tual, fill = Value)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "red") +
  labs(x = "Predicted", y = "Actual", title = "Logistic Regression") +
  theme_minimal()

# Plot the confusion matrix for the grid_tuned_tree_model
plot_decision_tree <- ggplot(data = confusion_matrix_grid_df, aes(x = Predicted, y
= Actual, fill = Value)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "steelblue") +
  labs(x = "Predicted", y = "Actual", title = "Decision Tree") +
  theme_minimal()

# Plot the random forest confusion matrix
plot_random_forest <- ggplot(data = rf_confusion_matrix_df, aes(x = Predicted, y =
Actual, fill = Value)) +
  geom_tile() +
  geom_text(aes(label = Value), color = "black", size = 4) +
  scale_fill_gradient(low = "white", high = "green") +
  labs(x = "Predicted", y = "Actual", title = "Random Forest") +
  theme_minimal()

# Arrange the plots side by side
combined_plots <- grid.arrange(plot_logistic, plot_decision_tree, plot_random_fores
t, nrow = 2)
```





Above are the confusion matrix plots for all three models. From the plots, we can see: 1. The Logistic Regression model misclassifies 23 mushrooms 2. The Decision Tree model misclassifies 48 mushrooms 3. The Random Forest model misclassifies 17 mushrooms

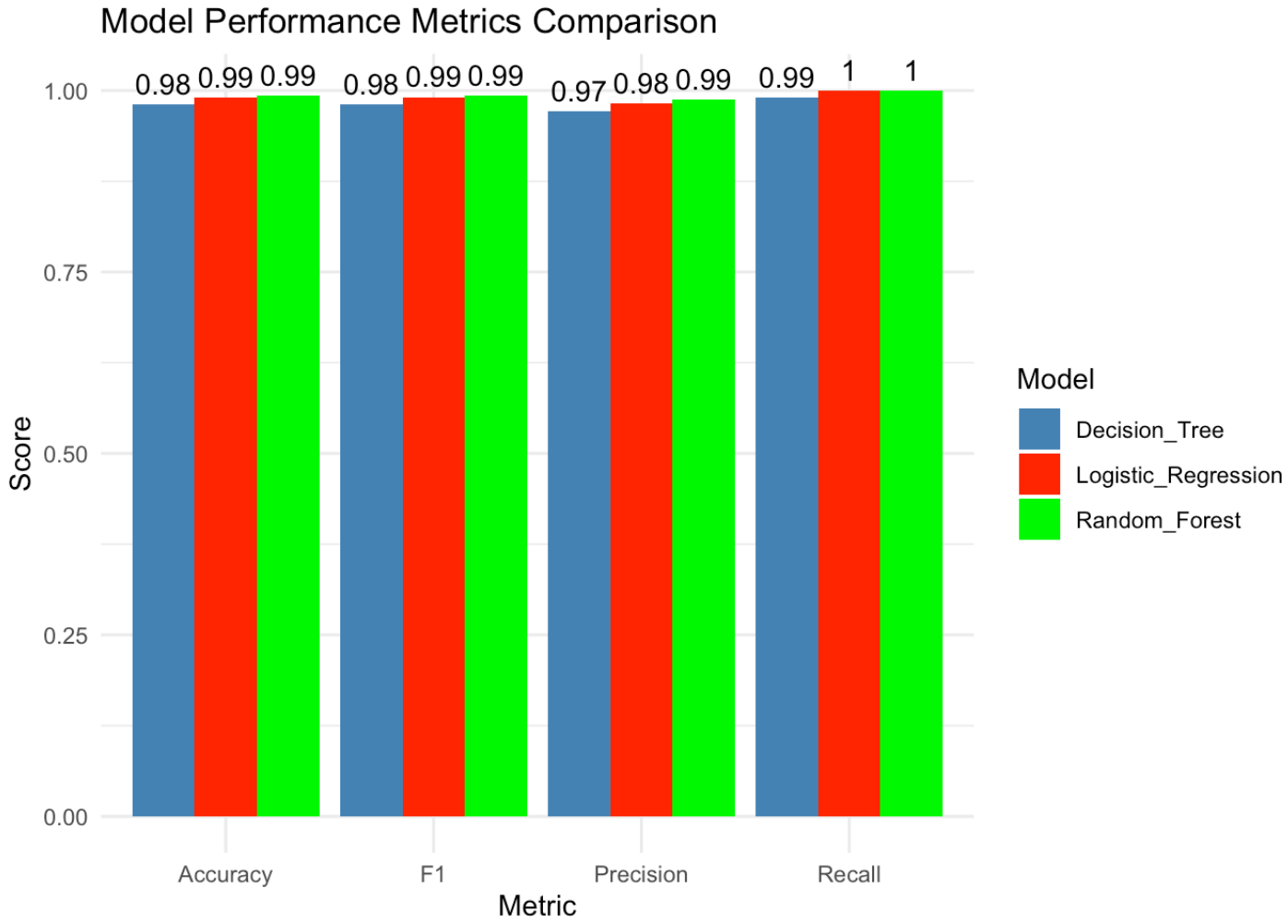
By looking at the confusion matrix plots, the Random Forest model appears to be the best model.

The barplot below is a comparison of the performance metrics of all three models.

From the plot, the Random Forest appears to be the best performing model, with the Logistic Regression model being a close second, and the Decision Tree being the worst performing model. They are all good models.

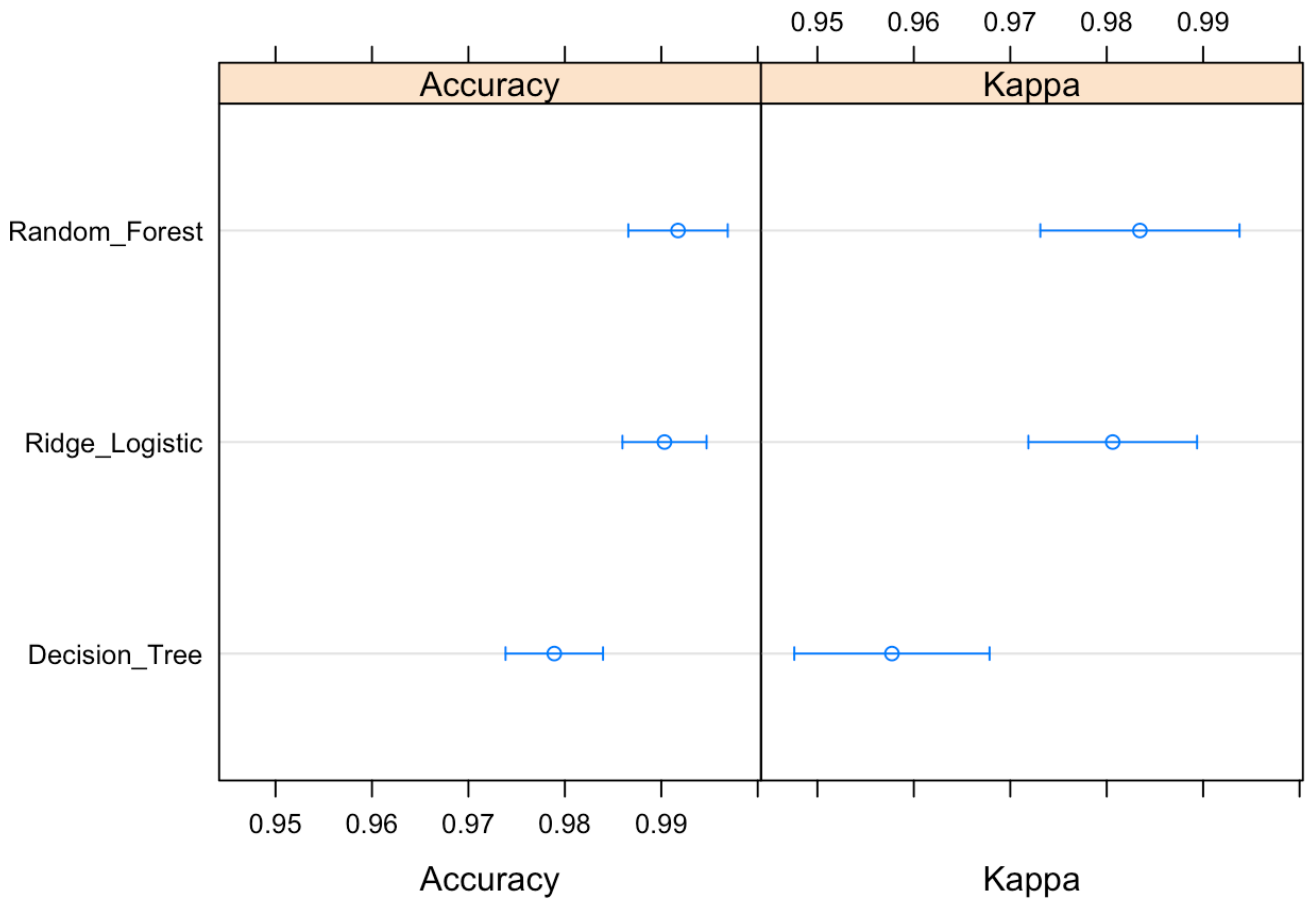
The Random Forest model and Logistic Regression model perform at the same level but the Random Forest model is slightly more precise with a precision score of 99% while the Logistic Regression's precisions score is 98%.

```
# Print the bar plot
print(bar_plot)
```



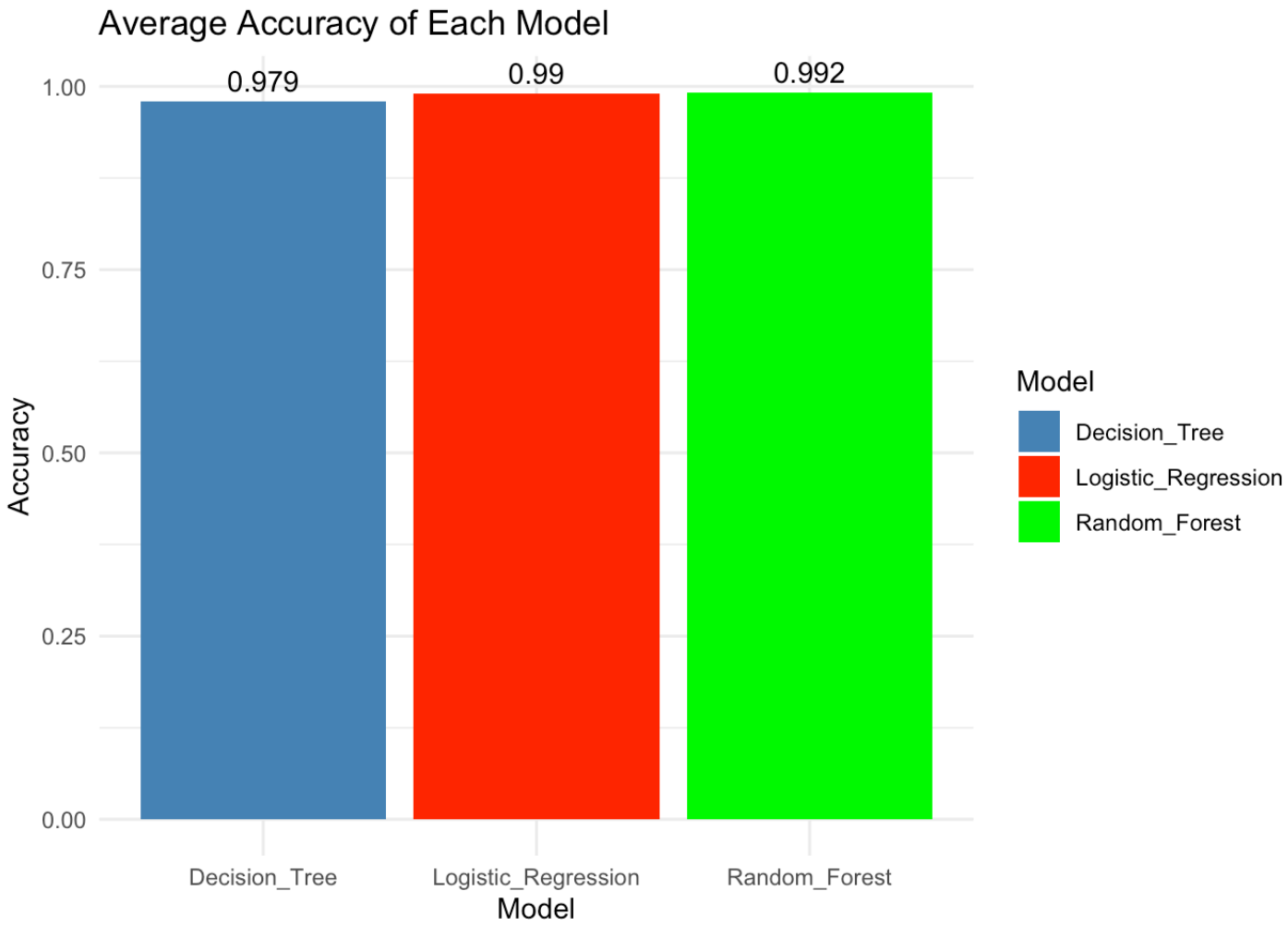
The dotplot below shows the distribution of the Accuracy scores of all three models. We can see that the Random Forest has the highest Accuracy score, followed by the Logistic Regression model, and then the Decision Tree. The dotplot shows a 95% confidence level.

```
# Visualise comparison
dotplot(results)
```



Confidence Level: 0.95

```
# Extract the accuracy values for each model
accuracy_metrics = data.frame(
  Logistic_Regression = ridge_model$resample$Accuracy,
  Decision_Tree = grid_tuned_tree_model$resample$Accuracy,
  Random_Forest = rf_model$resample$Accuracy
)

# Calculate average accuracy for each model
average_accuracy <- colMeans(accuracy_metrics)

# Create a data frame for plotting
plot_data <- data.frame(Model = names(average_accuracy), Accuracy = average_accurac
y)

# Create a bar plot of the average accuracy
bar_plot <- ggplot(plot_data, aes(x = Model, y = Accuracy, fill = Model)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = round(Accuracy, 3)), vjust = -0.5, color = "black") +
  labs(title = "Average Accuracy of Each Model", x = "Model", y = "Accuracy") +
  scale_fill_manual(values = c("Logistic_Regression" = "red", "Decision_Tree" = "st
eelblue", "Random_Forest" = "green")) +
  theme_minimal()

bar_plot
```



The barplot above is the average accuracy scores of the three models. Random Forest has the highest average accuracy score.

To provide further validation and assess the statistical significance of the performance differences observed among the models, a statistical test was conducted.

The aim of the test was to evaluate if the variations in the model performance were statistically significant in order to provide objective evidence to support the determination of the best performing model.

By comparing the accuracy scores of the models using a pairwise t-test with Bonferroni adjustment, the statistical test examined the hypothesis that there were significant differences in performance between the models.

Null Hypothesis (H0): There is no significant difference in performance between the models. Alternative Hypothesis (H1): There is significant diffference in performance between at least one pair of models.

Significance level: 5%.

```
# Perform a statistical test to determine if the performance differences between th
e models are statistically significant
# Convert the accuracy scores into a matrix
accuracy_matrix <- as.matrix(accuracy_metrics)

# Create a grouping factor indicating the model for each observation
models <- rep(colnames(accuracy_matrix), each = nrow(accuracy_matrix))

# Perform pairwise t-tests with Bonferroni adjustment
test_results <- pairwise.t.test(as.vector(accuracy_matrix), g = models, p.adjust.me
thod = "bonferroni")

# Print the test results
print(test_results)
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  as.vector(accuracy_matrix) and models
##
##                     Decision_Tree Logistic_Regression
## Logistic_Regression 0.00181       -
## Random_Forest       0.00069       1.00000
##
## P value adjustment method: bonferroni
```

The output above is the result of the t-test with Bonferroni adjustment.

The p-values for the compariosns are: 1. Comparison between Logistic Regression and Decision Tree: p-value: 0.00181, which is less than 0.05 (5%) significance level, indicating a statistically significant difference between these two model's accuracy. 2. Comparison between Logistic Regression and Random Forest: p-value: 0.00069, which is less than 0.05 (5%) significance level, indicating a statistically significant difference between these two model's accuracy.

The statistical test results support our observation that Random Forest appears to be the best model.

Our observation of the dotplot aligns with the statistical findings. Random Forest has the highest accuracy score and average accuracy among the three models. The confusion matrix further support this observation as the Random Forest model demonstrates the highest metrics across the board.

In conclusion, the statistical test results, along with the visual representation and performance metrics provide evidence that the Random Forest model is the best performing model.