Code

Mushroom Classification

Jack McColm-de Jong

2023-12-15

Introduction

Overview

The objective of this data science report is to explore and analyze a dataset concerning mushrooms, focusing on the critical task of classifying them based on their edibility. The data is located on Kaggle (https://www.kaggle.com/datasets/uciml/mushroom-classification/data), provided by UCI Machine Learning. The primary research question at the heart of this analysis is: “Can we accurately predict whether a mushroom is edible or poisonous based on its characteristics?” This question is not only of academic interest but also holds significant practical implications in fields like mycology, food safety, and outdoor recreation.

Dataset Description

The dataset is a comprehensive collection of data points related to 23 mushroom species. Each entry in the dataset represents a mushroom observation and includes a set of features that describe its physical characteristics. All of the features are categorical. The most crucial feature in this dataset is the “class”

variable, which categorizes each mushroom as either “edible” or “poisonous”. This binary classification forms the foundation of our analysis.

Features

The dataset includes several features, which include (but are not limited to) color, shape, habitat, and odor. These characteristics are essential as they potentially hold the key to distinguishing between edible and poisonous mushrooms. Understanding the correlation and patterns within these features in relation to the “class” variable is the aim of our analysis.

Background

Mushroom classification has been a subject of interest for centuries, primarily due to the high stakes involved in distinguishing edible mushrooms from poisonous ones. Incorrect classification can lead to severe health risks, making this an area of both scientific and practical importance. Traditional methods of classification rely on expert knowledge and physical inspection. However, with advancements in data science and machine learning, there is a significant opportunity to augment or even revolutionize these traditional methods.

Objective

The aim of this report is to apply various data analysis and machine learning techniques to this dataset to predict the class (edible or poisonous) of mushrooms. This involves data preprocessing, exploratory data analysis, feature selection, and ultimately, building and evaluating predictive models. The goal is to achieve a model with high accuracy and reliability, potentially providing a valuable tool for experts and laypersons in identifying mushroom edibility.

Dataset Preliminary Analysis

The mushrooms.csv dataset consists of 8124 entries, each representing a unique mushroom specimen. It includes 23 columns, each of which is a feature describing various aspects of the mushrooms. Importantly, all features, including the target variable ‘class’, are categorical in nature.

Key Features

Class: The primary variable of interest, indicating whether a mushroom is edible (e) or poisonous (p).

Physical Characteristics: These include cap-shape, cap-surface, cap-color, bruises, odor, and others, detailing the physical appearance and attributes of the mushrooms.

Gill Attributes: Features like gill-attachment, gill-spacing, gill-size, and gill-color provide information about the gills of the mushrooms.

Stalk Features: This includes stalk-shape, stalk-root, stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring, and stalk-color-below-ring.

Veil Characteristics: Details about the veil, including veil-type and veil-color.

Ring Features: Information about the ring(s) of the mushroom, captured in ring-number and ring-type.

Additional Attributes: spore-print-color, population, and habitat provide more context about the mushroom’s environment and reproduction.

Exploratory Data Analysis

To begin, we will read in the dataset and display the head for inspection.

Hide

```
mushrooms = read.csv("~/Final Project/mushrooms.csv", header = TRUE, sep = ",")
head(mushrooms)
```

```
##   class cap.shape cap.surface cap.color bruises odor gill.attachment
## 1     p         x           s         n       t    p               f
## 2     e         x           s         y       t    a               f
## 3     e         b           s         w       t    l               f
## 4     p         x           y         w       t    p               f
## 5     e         x           s         g       f    n               f
## 6     e         x           y         y       t    a               f
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1            c         n          k           e          e
## 2            c         b          k           e          c
## 3            c         b          n           e          c
## 4            c         n          n           e          e
## 5            w         b          k           t          e
## 6            c         b          n           e          c
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                        s                        s                       w
## 2                        s                        s                       w
## 3                        s                        s                       w
## 4                        s                        s                       w
## 5                        s                        s                       w
## 6                        s                        s                       w
##   stalk.color.below.ring veil.type veil.color ring.number ring.type
## 1                      w         p          w           o         p
## 2                      w         p          w           o         p
## 3                      w         p          w           o         p
## 4                      w         p          w           o         p
## 5                      w         p          w           o         e
## 6                      w         p          w           o         p
##   spore.print.color population habitat
## 1                 k          s       u
## 2                 n          n       g
## 3                 n          n       m
## 4                 k          s       u
## 5                 n          a       g
## 6                 k          n       g
```

The data is encoded with letters representing different attributes. Using the cipher from Kaggle, we can replace the coded letters with their true meanings. For the sake of tidyness, the decoding is attached at the bottom of the report. Here we can see the head of the cleaned dataset:

Hide

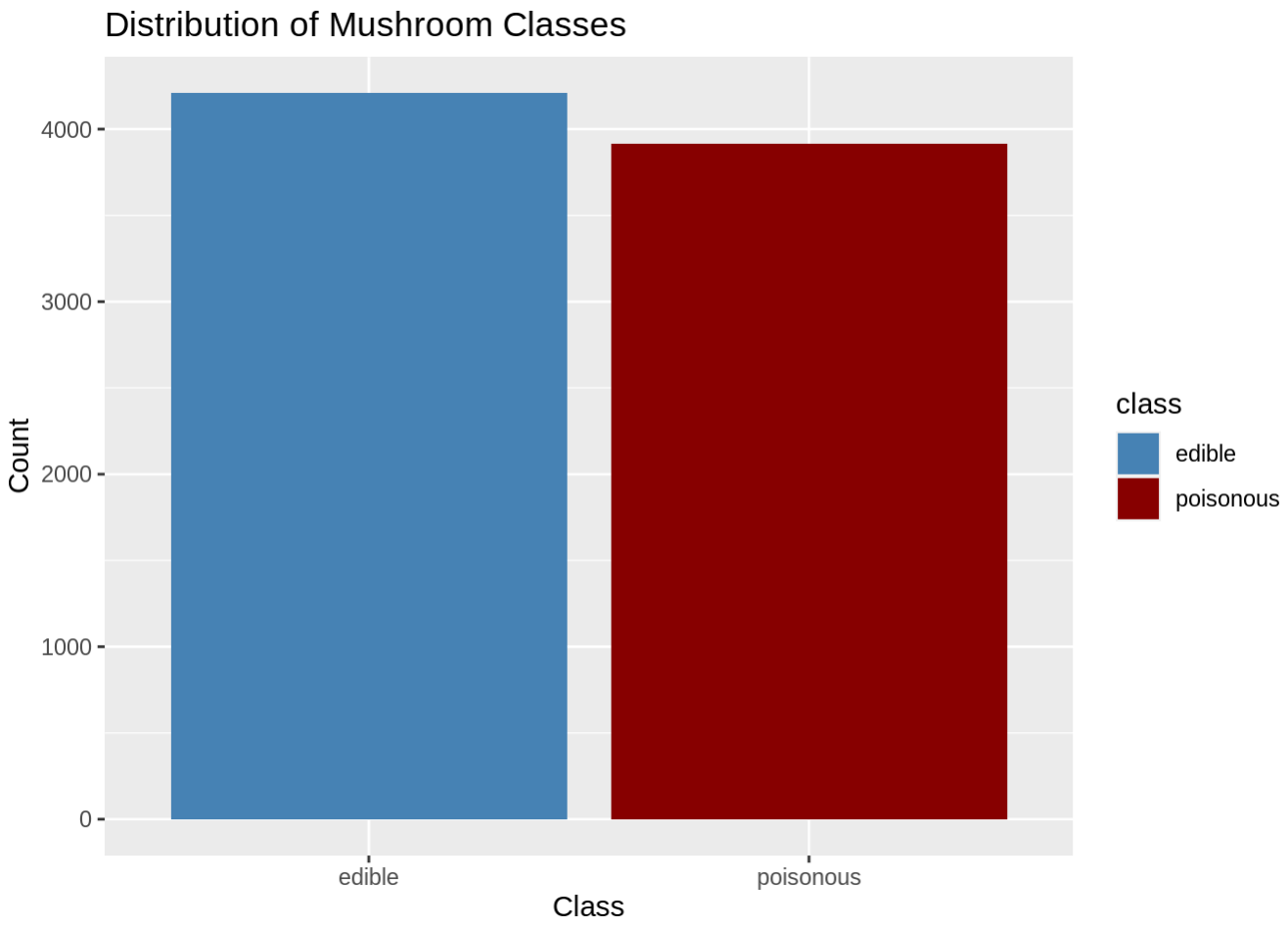
```
# View the transformed dataset
head(mushrooms_transformed)
```

```
##       class cap.shape cap.surface cap.color bruises    odor gill.attachment
## 1 poisonous    convex      smooth     brown bruises pungent            free
## 2    edible    convex      smooth    yellow bruises  almond            free
## 3    edible      bell      smooth     white bruises   anise            free
## 4 poisonous    convex       scaly     white bruises pungent            free
## 5    edible    convex      smooth      gray      no    none            free
## 6    edible    convex       scaly    yellow bruises  almond            free
##   gill.spacing gill.size gill.color stalk.shape stalk.root
## 1        close    narrow      black   enlarging      equal
## 2        close     broad      black   enlarging       club
## 3        close     broad      brown   enlarging       club
## 4        close    narrow      brown   enlarging      equal
## 5      crowded     broad      black    tapering      equal
## 6        close     broad      brown   enlarging       club
##   stalk.surface.above.ring stalk.surface.below.ring stalk.color.above.ring
## 1                   smooth                   smooth                  white
## 2                   smooth                   smooth                  white
## 3                   smooth                   smooth                  white
## 4                   smooth                   smooth                  white
## 5                   smooth                   smooth                  white
## 6                   smooth                   smooth                  white
##   stalk.color.below.ring veil.type veil.color ring.number  ring.type
## 1                  white   partial      white         one    pendant
## 2                  white   partial      white         one    pendant
## 3                  white   partial      white         one    pendant
## 4                  white   partial      white         one    pendant
## 5                  white   partial      white         one evanescent
## 6                  white   partial      white         one    pendant
##   spore.print.color population habitat
## 1             black  scattered   urban
## 2             brown   numerous grasses
## 3             brown   numerous meadows
## 4             black  scattered   urban
## 5             brown   abundant grasses
## 6             black   numerous grasses
```

Now we will take a closer look at our response variable (class) and the relationships between predictors. To begin we will look at the distribution of the ‘class’ variable.

Hide

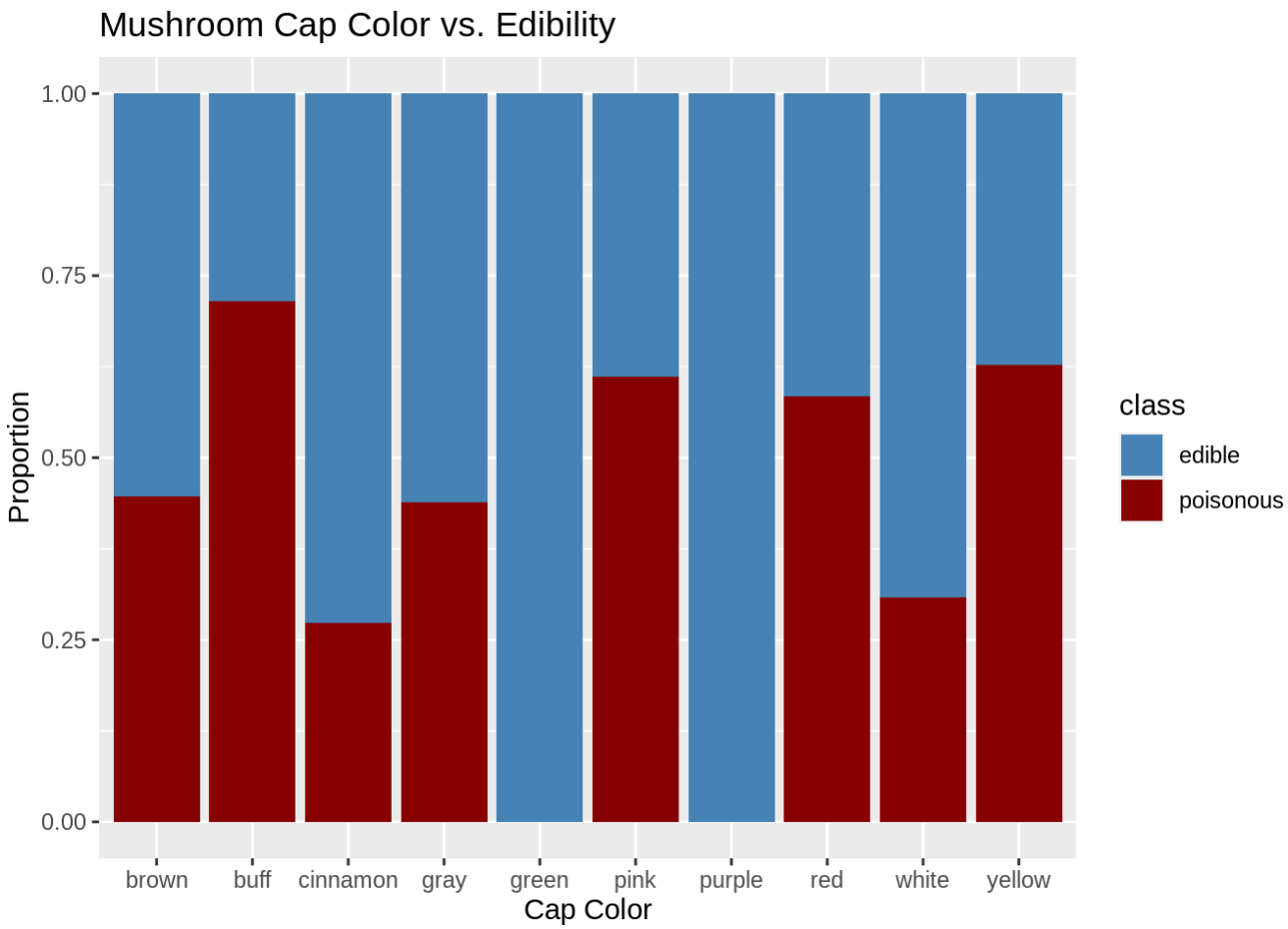
```
# 1. Univariate Visualization of the Outcome Variable
ggplot(mushrooms_transformed, aes(x = class, fill = class)) +
  geom_bar() +
  scale_fill_manual(values = c("edible" = "steelblue", "poisonous" = "darkred")) +
  labs(title = "Distribution of Mushroom Classes", x = "Class", y = "Count")
```



The graph shows that the proportions of the class variable are roughly equal, with slightly more ‘edible’ values than ‘poisonous’. Next we will examine the relationship between mushroom cap color and class.

Hide

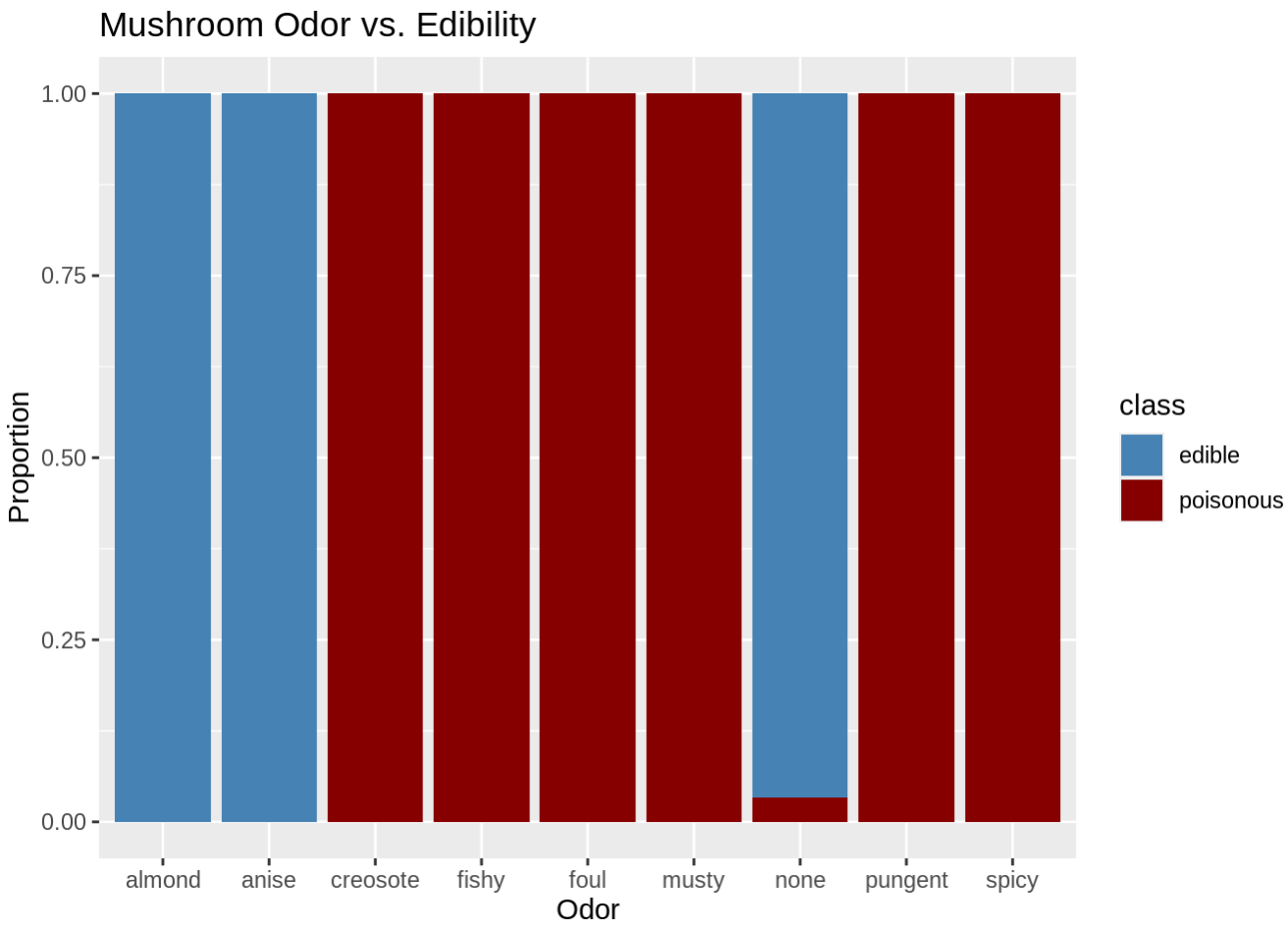
```
# 2. Bivariate Visualization - Cap Color vs. Edibility
ggplot(mushrooms_transformed, aes(fill = class, x = `cap.color`)) +
  geom_bar(position = "fill") +
  scale_fill_manual(values = c("edible" = "steelblue", "poisonous" = "darkred")) +
  labs(title = "Mushroom Cap Color vs. Edibility", x = "Cap Color", y = "Proportion")
```



The resulting graph reveals that certain colors are great predictors for class, with the colors green and purple having 100% edible classifications. Other colors like cinnamon, white, and buff are also pretty good predictors. Next we will look at the relationship between odor and class.

Hide

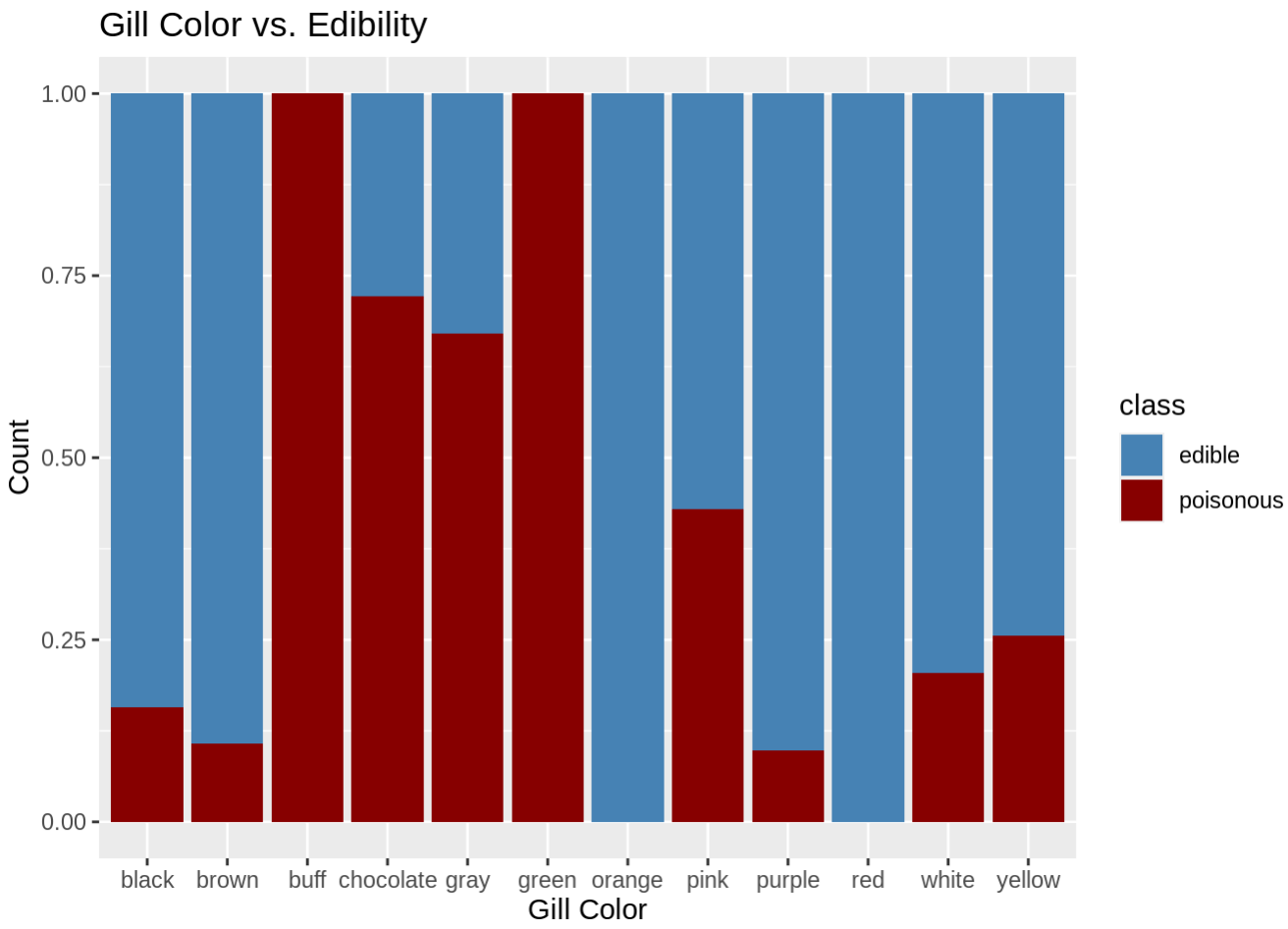
```
# 3. Bivariate Visualization - Odor vs. Edibility
ggplot(mushrooms_transformed, aes(fill = class, x = odor)) +
  geom_bar(position = "fill") +
  scale_fill_manual(values = c("edible" = "steelblue", "poisonous" = "darkred")) +
  labs(title = "Mushroom Odor vs. Edibility", x = "Odor", y = "Proportion")
```



The graph shows that odor is an excellent predictor for class, with the different odors having almost no overlap between edible and poisonous. This variable alone may be enough to cause data leakage and ruin our models. We will experiment with modeling without the odor variable. For our next visualization, we will look at gill color.

Hide

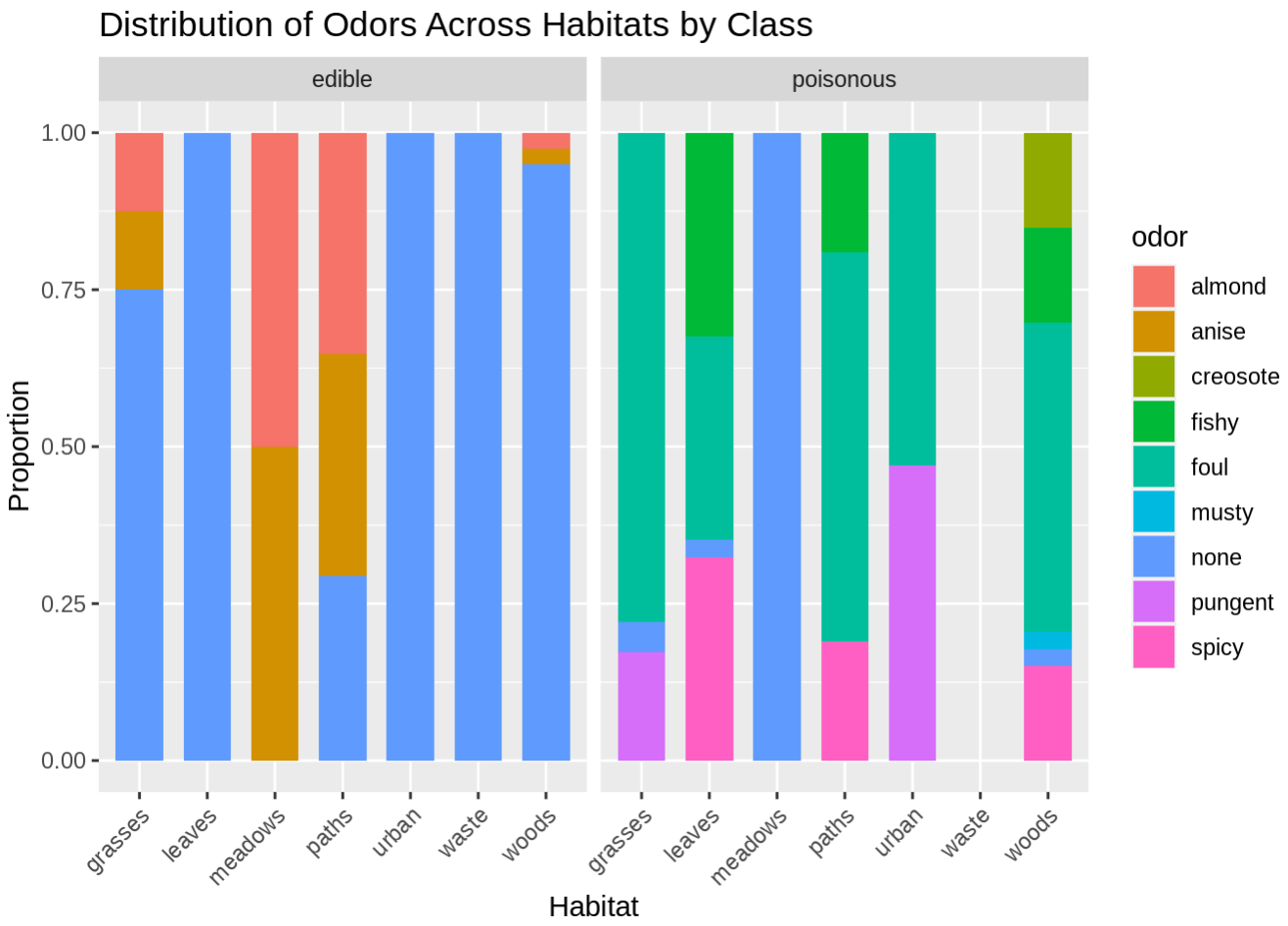
```
# 4. Bivariate Visualization - Gill Color vs. Edibility
ggplot(mushrooms_transformed, aes(x = `gill.color`, fill = class)) +
  geom_bar(position = "fill") +
  labs(title = "Gill Color vs. Edibility", x = "Gill Color", y = "Count") +
  scale_fill_manual(values = c("edible" = "steelblue", "poisonous" = "darkred"))
```



We can see from the graph that gill color is also an excellent predictor for class. With multiple variables showing strong correlation with our response variable, we should be able to train very accurate models. For our last visualization, we will examine a multivariate plot of habitat, odor, and class.

Hide

```
# 5. Multivariate Plot of Habitat, Odor, and Class
ggplot(mushrooms_transformed, aes(x = habitat, fill = odor)) +
  geom_bar(position = "fill", width = 0.7) +  # Adjust the width here
  facet_wrap(~class) +
  labs(title = "Distribution of Odors Across Habitats by Class", x = "Habitat", y = "Propo
rtion") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))  # Adjusting text angle for bet
ter readability
```



The addition of the habitat variable allows us to see how mushroom odors are distributed and differ across habitats, showing how a potential interaction effect could be useful in our model. Overall, now we have a better understanding of the data and can tune accordingly.

The data includes 23 variables, many of which are mycological metrics, uninterpretable for the average person. We want this model to be easy to use, and requiring so many predictors is very unintuitive for real world application. In addition, upon visual inspection, many of the predictors do not appear to be correlated with the response variable. Therefore we will remove most of the columns, only keeping a few intuitively measurable variables, including class, cap.shape, cap.surface, cap.color, bruises, gill.color, and habitat. In addition we will set class, our response variable, as a factor.

Hide

```
mushrooms_reduced <- mushrooms_transformed %>%
   select(
     class, cap.shape, cap.surface, cap.color, bruises, gill.color, habitat
   )

mushrooms_reduced$class <- as.factor(mushrooms_reduced$class) # Set class as a factor
```

Data Splitting and Cross Validation

Now we will split the data into two groups: one for training our models, and one for evaluating them. In our project, we have split the data such that 80% is used for training and the remaining 20% for testing. This split is a common practice, providing a substantial amount of data for training while still retaining enough for

an unbiased evaluation.

Cross-validation is a robust method to estimate the skill of a model on new data. We split the training data into smaller groups. In this case, the data is divided into 10 parts. For each “fold”, we train the model on 9 parts and validate (test) it on the 1 remaining part. We repeat this process so that each part is used for validation once. Finally we calculate the average performance across all 10 validation tests.

In our code, trainControl(method = “cv”, number = 10) specifies that we are using 10-fold cross-validation. This means our model will be trained and validated across 10 different subsets of our data to ensure its reliability and robustness.

Hide

```
set.seed(123) # for reproducibility
splitIndex <- createDataPartition(mushrooms_reduced$class, p = 0.8, list = FALSE)
trainingSet <- mushrooms_reduced[splitIndex, ]
testingSet <- mushrooms_reduced[-splitIndex, ]

# Cross validation setup
crossValidation <- trainControl(method = "cv", number = 10)
```

Model Fitting

Now we will move on to fitting models to our data. Using the recipes package, we can create a recipe using our cleaned data. We use step_dummy() to dummy code all of the variables, as the entire dataset is categorical. Then we apply the recipe to our training data, and we are ready for modelling.

Hide

```
# Create the recipe for modelling
mushroom_recipe <- recipe(class ~ ., data = trainingSet) %>%
  step_dummy(all_nominal(), -all_outcomes())

# Apply the recipe to the training data
prepared_recipe <- prep(mushroom_recipe, training = trainingSet)
training_processed <- bake(prepared_recipe, new_data = trainingSet)
```

I decided to fit three types of models to the data: a knn model, an elastic net model, and a logistic regression model.

Hide

```r
# 1. k-Nearest Neighbors
knnFit <- train(class ~ ., data = training_processed, method = "kknn",
                trControl = crossValidation, tuneLength = 10)

# 2. Elastic Net
enetFit <- train(class ~ ., data = training_processed, method = "glmnet",
                 trControl = crossValidation, tuneLength = 10)

# 3. Logistic Regression
logiFit <- train(class ~ ., data = training_processed, method = "glm",
                 trControl = crossValidation, tuneLength = 10)

# Output the model details
print(knnFit)
```

```
## k-Nearest Neighbors
##
## 6500 samples
##   35 predictor
##    2 classes: 'edible', 'poisonous'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5850, 5851, 5851, 5850, 5850, 5850, ...
## Resampling results across tuning parameters:
##
##   kmax  Accuracy   Kappa
##    5    0.9772319  0.9543693
##    7    0.9772319  0.9543693
##    9    0.9772319  0.9543693
##   11    0.9772319  0.9543693
##   13    0.9772319  0.9543693
##   15    0.9772319  0.9543693
##   17    0.9772319  0.9543693
##   19    0.9772319  0.9543693
##   21    0.9772319  0.9543693
##   23    0.9772319  0.9543693
##
## Tuning parameter 'distance' was held constant at a value of 2
## Tuning
##  parameter 'kernel' was held constant at a value of optimal
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were kmax = 23, distance = 2 and kernel
##  = optimal.
```

Hide

```r
print(enetFit$bestTune)
```

```
##    alpha      lambda
## 55   0.6 0.003524065
```

Hide

```
print(logiFit)
```

```
## Generalized Linear Model
##
## 6500 samples
##   35 predictor
##    2 classes: 'edible', 'poisonous'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5850, 5851, 5850, 5850, 5850, 5850, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9055365  0.8106964
```

From the model details, we can see the fitted parameters:

knn parameters are kmax = 23, distance = 2, and kernel = optimal

elastic net parameters are alpha = 0.2, lambda = 0.00352

logistic regression parameter is kappa = 0.812

Model Selection and Performance

In order to evaluate our model performance and choose the best one, we will look at model ROC AUC (Receiver Operating Characteristic Area Under the Curve). Taking on a value between 0 and 1, this performance measurement allows us to compare our models, with higher ROC AUC indicating a better model.

Hide

```
# Prepare testing data for comparison
testing_processed <- bake(prepared_recipe, new_data = testingSet)

# Predict using the KNN model
knnPredictions <- predict(knnFit, testing_processed, type = "prob")
rocKnn <- roc(testing_processed$class, knnPredictions[, "edible"])

# Predict using the Elastic Net model
enetPredictions <- predict(enetFit, testing_processed, type = "prob")
rocEnet <- roc(testing_processed$class, enetPredictions[, "edible"])

# Predict using the Logistic model
logPredictions <- predict(enetFit, testing_processed, type = "prob")
rocLog <- roc(testing_processed$class, logPredictions[, "edible"])

# AUC values
aucKnn <- auc(rocKnn)
aucEnet <- auc(rocEnet)
aucLog <- auc(rocLog)

# Output the AUC values
print(paste("AUC for KNN:", aucKnn))
```

```
## [1] "AUC for KNN: 0.973796626590919"
```

Hide

```
print(paste("AUC for Elastic Net:", aucEnet))
```

```
## [1] "AUC for Elastic Net: 0.945614522636951"
```

Hide

```
print(paste("AUC for Logistic:", aucLog))
```

```
## [1] "AUC for Logistic: 0.945614522636951"
```

Looking at the results of the the KNN fit, we can see that the accuracy is roughly the same for each kmax value. Therefore to reduce model complexity we can select and train our final model: a KNN model with 5 neighbors, distance = 2, and kernel = “optimal”.

Hide

```
knn_fit <- train(class ~ ., data = training_processed, method = "kknn",
                 trControl = crossValidation, tuneGrid = expand.grid(kmax = 5, distance =
2, kernel = "optimal"))

knnPredictions <- predict(knnFit, testing_processed, type = "prob")
rocKnn <- roc(testing_processed$class, knnPredictions[, "edible"])
aucKnn <- auc(rocKnn)
print(paste("AUC for KNN:", aucKnn))
```

```
## [1] "AUC for KNN: 0.973796626590919"
```

Next we can examine the confusion matrix of the KNN model to analyze its performance.

Hide

```
knn_predictions <- predict(knn_fit, newdata = testing_processed)
confusionMatrix <- confusionMatrix(knn_predictions, testing_processed$class)
print(confusionMatrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  edible poisonous
##   edible       827        28
##   poisonous     14       755
##
##                Accuracy : 0.9741
##                  95% CI : (0.9652, 0.9813)
##     No Information Rate : 0.5179
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9482
##
##  Mcnemar's Test P-Value : 0.04486
##
##             Sensitivity : 0.9834
##             Specificity : 0.9642
##          Pos Pred Value : 0.9673
##          Neg Pred Value : 0.9818
##              Prevalence : 0.5179
##          Detection Rate : 0.5092
##    Detection Prevalence : 0.5265
##       Balanced Accuracy : 0.9738
##
##        'Positive' Class : edible
##
```

Based on the values for ROC AUC, we can conclude that the knn model with 5 neighbors is the most accurate model with an ROC AUC of 0.9743, and an accuracy of 97.48%. This is a very accurate model that will provide quality predictions based on intuitive measurements.

Conclusion

In this data science project, our goal was to develop a predictive model that could accurately classify mushrooms as either edible or poisonous based on various mushroom characteristics. We explored and compared several machine learning models, ultimately finding that a k-Nearest Neighbors (KNN) model with 5 neighbors outperformed the others.

Model Performance

KNN Model: Our final model, a KNN with 5 neighbors, emerged as the top performer. This model's success can be attributed to its ability to effectively capture the nuances in the data, leveraging the high correlations observed between the class and predictors.

Logistic Regression: This model, while robust in many settings, was the least effective for our dataset. Its performance was somewhat expected, given that logistic regression can sometimes struggle with complex, non-linear relationships in data.

Elastic Net Regression: Falling between KNN and logistic regression, the elastic net model provided a balanced approach. It handled the correlations and feature interactions better than logistic regression but did not reach the accuracy level of the KNN model.

Insights

The performance of these models aligns well with our initial data exploration, which revealed high correlations between the target variable and several predictors. The KNN model's superior performance was anticipated due to its proficiency in capturing intricate patterns in datasets with closely correlated features.

The ability to predict mushroom edibility with high accuracy has significant implications for both mycology enthusiasts and public health. The models developed here can serve as a valuable tool in educational and preventative efforts, helping to reduce the risks associated with mushroom foraging. By simply noting easily identifiable physical characteristics, this model can easily distinguish mushroom edibility.

Code Appendix

Hide

```
# Replace coded values with actual values
mushrooms_transformed <- mushrooms %>%
  mutate(
    class = case_when(
      class == 'e' ~ 'edible',
      class == 'p' ~ 'poisonous'
    ),
    `cap.shape` = case_when(
      `cap.shape` == 'b' ~ 'bell',
      `cap.shape` == 'c' ~ 'conical',
      `cap.shape` == 'x' ~ 'convex',
      `cap.shape` == 'f' ~ 'flat',
      `cap.shape` == 'k' ~ 'knobbed',
      `cap.shape` == 's' ~ 'sunken'
    ),
    `cap.surface` = case_when(
      `cap.surface` == 'f' ~ 'fibrous',
      `cap.surface` == 'g' ~ 'grooves',
      `cap.surface` == 'y' ~ 'scaly',
      `cap.surface` == 's' ~ 'smooth'
    ),
    `cap.color` = case_when(
      `cap.color` == 'n' ~ 'brown',
      `cap.color` == 'b' ~ 'buff',
      `cap.color` == 'c' ~ 'cinnamon',
      `cap.color` == 'g' ~ 'gray',
      `cap.color` == 'r' ~ 'green',
      `cap.color` == 'p' ~ 'pink',
      `cap.color` == 'u' ~ 'purple',
      `cap.color` == 'e' ~ 'red',
      `cap.color` == 'w' ~ 'white',
      `cap.color` == 'y' ~ 'yellow'
    ),
    bruises = case_when(
      bruises == 't' ~ 'bruises',
      bruises == 'f' ~ 'no'
    ),
    odor = case_when(
      odor == 'a' ~ 'almond',
      odor == 'l' ~ 'anise',
      odor == 'c' ~ 'creosote',
      odor == 'y' ~ 'fishy',
      odor == 'f' ~ 'foul',
      odor == 'm' ~ 'musty',
      odor == 'n' ~ 'none',
      odor == 'p' ~ 'pungent',
      odor == 's' ~ 'spicy'
    ),
    `gill.attachment` = case_when(
      `gill.attachment` == 'a' ~ 'attached',
      `gill.attachment` == 'd' ~ 'descending',
      `gill.attachment` == 'f' ~ 'free',
```

```
    `gill.attachment` == 'n' ~ 'notched'
  ),
  `gill.spacing` = case_when(
    `gill.spacing` == 'c' ~ 'close',
    `gill.spacing` == 'w' ~ 'crowded',
    `gill.spacing` == 'd' ~ 'distant'
  ),
  `gill.size` = case_when(
    `gill.size` == 'b' ~ 'broad',
    `gill.size` == 'n' ~ 'narrow'
  ),
      # Continuing from gill.color onwards
  `gill.color` = case_when(
    `gill.color` == 'k' ~ 'black',
    `gill.color` == 'n' ~ 'brown',
    `gill.color` == 'b' ~ 'buff',
    `gill.color` == 'h' ~ 'chocolate',
    `gill.color` == 'g' ~ 'gray',
    `gill.color` == 'r' ~ 'green',
    `gill.color` == 'o' ~ 'orange',
    `gill.color` == 'p' ~ 'pink',
    `gill.color` == 'u' ~ 'purple',
    `gill.color` == 'e' ~ 'red',
    `gill.color` == 'w' ~ 'white',
    `gill.color` == 'y' ~ 'yellow'
  ),
  `stalk.shape` = case_when(
    `stalk.shape` == 'e' ~ 'enlarging',
    `stalk.shape` == 't' ~ 'tapering'
  ),
  `stalk.root` = case_when(
    `stalk.root` == 'b' ~ 'bulbous',
    `stalk.root` == 'c' ~ 'club',
    `stalk.root` == 'u' ~ 'cup',
    `stalk.root` == 'e' ~ 'equal',
    `stalk.root` == 'z' ~ 'rhizomorphs',
    `stalk.root` == 'r' ~ 'rooted',
    `stalk.root` == '?' ~ 'missing'
  ),
  `stalk.surface.above.ring` = case_when(
    `stalk.surface.above.ring` == 'f' ~ 'fibrous',
    `stalk.surface.above.ring` == 'y' ~ 'scaly',
    `stalk.surface.above.ring` == 'k' ~ 'silky',
    `stalk.surface.above.ring` == 's' ~ 'smooth'
  ),
  `stalk.surface.below.ring` = case_when(
    `stalk.surface.below.ring` == 'f' ~ 'fibrous',
    `stalk.surface.below.ring` == 'y' ~ 'scaly',
    `stalk.surface.below.ring` == 'k' ~ 'silky',
    `stalk.surface.below.ring` == 's' ~ 'smooth'
  ),
  `stalk.color.above.ring` = case_when(
```

```
    `stalk.color.above.ring` == 'n' ~ 'brown',
    `stalk.color.above.ring` == 'b' ~ 'buff',
    `stalk.color.above.ring` == 'c' ~ 'cinnamon',
    `stalk.color.above.ring` == 'g' ~ 'gray',
    `stalk.color.above.ring` == 'o' ~ 'orange',
    `stalk.color.above.ring` == 'p' ~ 'pink',
    `stalk.color.above.ring` == 'e' ~ 'red',
    `stalk.color.above.ring` == 'w' ~ 'white',
    `stalk.color.above.ring` == 'y' ~ 'yellow'
  ),
  `stalk.color.below.ring` = case_when(
    `stalk.color.below.ring` == 'n' ~ 'brown',
    `stalk.color.below.ring` == 'b' ~ 'buff',
    `stalk.color.below.ring` == 'c' ~ 'cinnamon',
    `stalk.color.below.ring` == 'g' ~ 'gray',
    `stalk.color.below.ring` == 'o' ~ 'orange',
    `stalk.color.below.ring` == 'p' ~ 'pink',
    `stalk.color.below.ring` == 'e' ~ 'red',
    `stalk.color.below.ring` == 'w' ~ 'white',
    `stalk.color.below.ring` == 'y' ~ 'yellow'
  ),
  `veil.type` = case_when(
    `veil.type` == 'p' ~ 'partial',
    `veil.type` == 'u' ~ 'universal'
  ),
  `veil.color` = case_when(
    `veil.color` == 'n' ~ 'brown',
    `veil.color` == 'o' ~ 'orange',
    `veil.color` == 'w' ~ 'white',
    `veil.color` == 'y' ~ 'yellow'
  ),
  `ring.number` = case_when(
    `ring.number` == 'n' ~ 'none',
    `ring.number` == 'o' ~ 'one',
    `ring.number` == 't' ~ 'two'
  ),
  `ring.type` = case_when(
    `ring.type` == 'c' ~ 'cobwebby',
    `ring.type` == 'e' ~ 'evanescent',
    `ring.type` == 'f' ~ 'flaring',
    `ring.type` == 'l' ~ 'large',
    `ring.type` == 'n' ~ 'none',
    `ring.type` == 'p' ~ 'pendant',
    `ring.type` == 's' ~ 'sheathing',
    `ring.type` == 'z' ~ 'zone'
  ),
  `spore.print.color` = case_when(
    `spore.print.color` == 'k' ~ 'black',
    `spore.print.color` == 'n' ~ 'brown',
    `spore.print.color` == 'b' ~ 'buff',
    `spore.print.color` == 'h' ~ 'chocolate',
    `spore.print.color` == 'r' ~ 'green',
```

```
    `spore.print.color` == 'o' ~ 'orange',
    `spore.print.color` == 'u' ~ 'purple',
    `spore.print.color` == 'w' ~ 'white',
    `spore.print.color` == 'y' ~ 'yellow'
  ),
  population = case_when(
    population == 'a' ~ 'abundant',
    population == 'c' ~ 'clustered',
    population == 'n' ~ 'numerous',
    population == 's' ~ 'scattered',
    population == 'v' ~ 'several',
    population == 'y' ~ 'solitary'
  ),
  habitat = case_when(
    habitat == 'g' ~ 'grasses',
    habitat == 'l' ~ 'leaves',
    habitat == 'm' ~ 'meadows',
    habitat == 'p' ~ 'paths',
    habitat == 'u' ~ 'urban',
    habitat == 'w' ~ 'waste',
    habitat == 'd' ~ 'woods'
  )
)
```