Poisonous Mushroom Classification

Shirley Mach & Gary Vartanian

November 25th, 2024

Table of Contents

Abstract

Identifying poisonous mushrooms using machine learning models is crucial for preventing potentially life-threatening mistakes. Flexible models such as Random Forest, Decision Trees, and XGBoost demonstrate superior performance in accurately classifying dangerous mushrooms compared to simpler models like Logistic Regression. While simpler models offer better interpretability, the critical nature of this classification task, where the consequence could be severe illness or loss of life, emphasizes the importance of predictive accuracy over interpretability. Thus, leveraging advanced models enhances safety and ensures more reliable identification of poisonous mushrooms. Important metrics utilized in this study include Accuracy, Recall, and F1-score, as they effectively account for false negatives—instances where poisonous mushrooms are incorrectly classified as edible—ensuring a comprehensive evaluation of each model's performance.

Introduction

Mushrooms, the delightful fungi, are a staple in many diets, from oyster and enoki to portobello and even the so-called magic mushrooms. As a diverse species, mushrooms can be incredibly beneficial, offering a rich source of ergothioneine—an amino acid and antioxidant that combats cellular damage, providing better gut health, and promoting lower cholesterol (UCLA Health 2024). However, they can also be deadly, making it crucial to understand the various features that distinguish safe, edible mushrooms from their toxic counterparts.

While general guidelines exist for identifying poisonous mushrooms, such as avoiding those with red caps, white gills, or a volva, these rules are not foolproof. Mushrooms lacking these traits can still be toxic (Visser 2016), therefore, finding better ways to detect would be helpful.

Problem Statement

The goal of this project is to develop machine learning models that can accurately classify mushrooms as edible or poisonous, providing a more reliable method of identification. Information obtained from this study may further the Agricultural Research Service's (ARS) efforts in developing portable mushroom tests, contributing to faster and more reliable identification of toxic species. Additionally, the project seeks to create a well-developed dichotomous key for scientists, particularly mycologists, to facilitate deeper research into poisonous mushrooms. Beyond scientific applications, the study aims to develop an accessible method for non-experts, enabling them to recognize key features that indicate whether a mushroom is safe or potentially hazardous, promoting safer foraging practices.

Dataset

The dataset used in this project is sourced from a Kaggle competition, based on the UCI Mushroom dataset. It consists of both training and testing datasets. The training dataset primarily includes around 3 million rows and 22 columns, with one column representing the mushroom classification and 20 columns detailing the mushrooms' features with 1 observation column (Id). With 3 million rows, our models will achieve stronger predictive power; however, additional resources will be needed to refine the dataset before implementing machine learning models, due to extreme volume of data.

Id	Cap Color	Stem Height	Veil Type	Habitat
Class	Bruise or Bleed	Stem Width	Veil Color	Season
Cap Diameter	Gill Attachment	Stem Root	Has Ring	
Cap Shape	Gill Spacing	Stem Surface	Ring Type	
Cap Surface	Gill Color	Stem Color	Spore Print Color	

Please refer to the UCI Mushroom dataset for the data dictionary.

Data Cleaning Process

Data Types, Data Cleaning, Data Removals

It is important to understand the makeup of the data. In the training data, all features are objects except cap-diameter, stem-height, and stem-width, which are numeric. Additionally, features that had incorrect entries, such as cap-diameters having non numeric values or cap-shape having numerical values, were cleaned. Lastly, the values that do not have relevance to their assigned columns were removed from the dataset.

Addressing Missing Fields

Additionally, it is important to note that there is a lot of missing data (Table 1). For the purpose of our data, if there are columns with beyond 25% missing entries, they were dropped from our dataset. For the remaining columns that have less than 25% missing entries, the missing values were filled in with the median (numeric) and mode (categorical), with the intention to preserve as much information as possible dealing with gaps in the dataset[1].

Table 1 - Features and Percent Missing

Features	Percent Missing (%)	Features	Percent Missing (%)
cap-diameter	0.000128	stem-root	88.452732
cap-shape	0.001283	stem-surface	63.551362
cap-surface	21.528227	stem-color	0.001219
cap-color	0.000385	veil-type	94.88435
does-bruise-or-bleed	0.000257	veil-color	87.93697
gill-attachment	16.80928	has-ring	0.00077
gill-spacing	40.373988	ring-type	4.134818

[1] Note: While there are other methods to impute numeric data: such as using mean, regression modeling, etc. Median was chosen due to the simplicity and relevance with skewed numeric data. Testing different methods (mean vs median vs leave out): imputing via the median (numeric) and mode (categorical) was found to achieve best results.

gill-color	0.001829	spore-print-color	91.425482
habitat	0.001444	habitat	0.001444

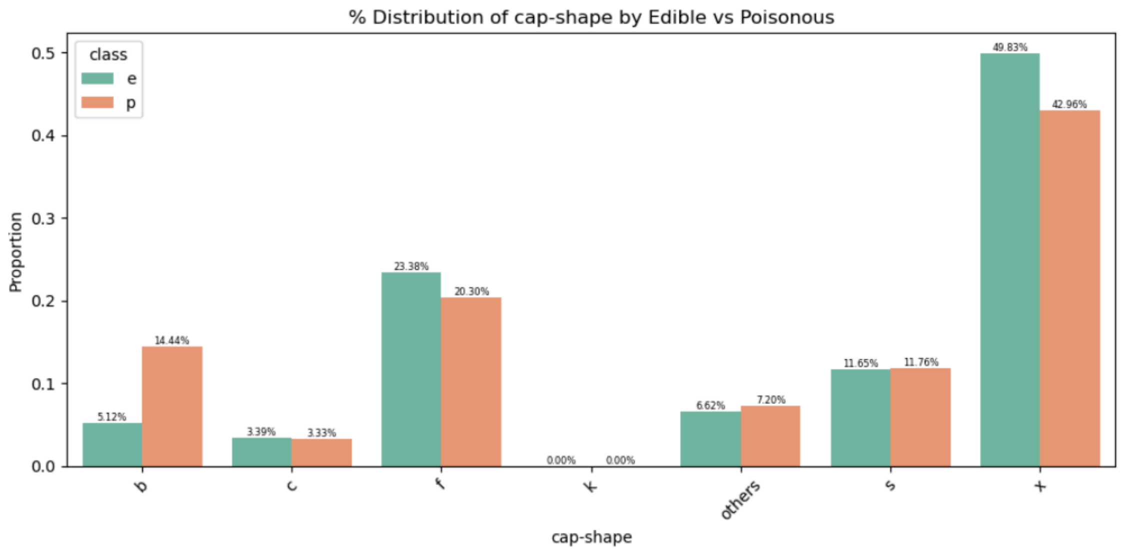
Class Balance

It is important to understand that most of the mushrooms in the cleaned trained dataset are poisonous (p). About 55% of the data consists of poisonous mushrooms (p) and 45% of edible ones (e). Given that there are more data points for poisonous mushrooms, our models may be better at predicting which mushrooms are poisonous than which ones are edible (better at predicting positives than negatives). The slight imbalance is noted, however, not too concerning.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) was conducted to help with better understanding and insights of the dataset and variables. A breakdown of edible (e) and poisonous (p) classes is done via contingency tables (this case contingency charts), which helps outline the difference in percentage occurrences of edible or poisonous on the different categorical variables. Main findings, results and observations are shown below. Additionally, fields with the biggest differences were tested for statistical significance via a 2 proportion z test with alpha of 1%[2].

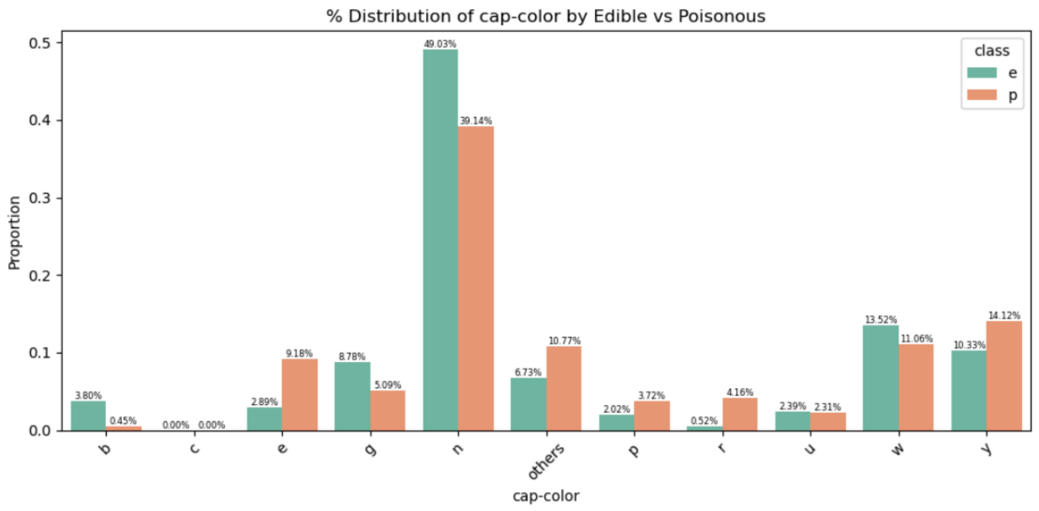
Cap Shape



[2] Other tests like: Chi-Square, Cohen's d, odds ratio were considered as a comparison. 2 proportion z test was chosen as it allowed: for individual comparison, ease of interpretation, and had a p value for each group comparison.

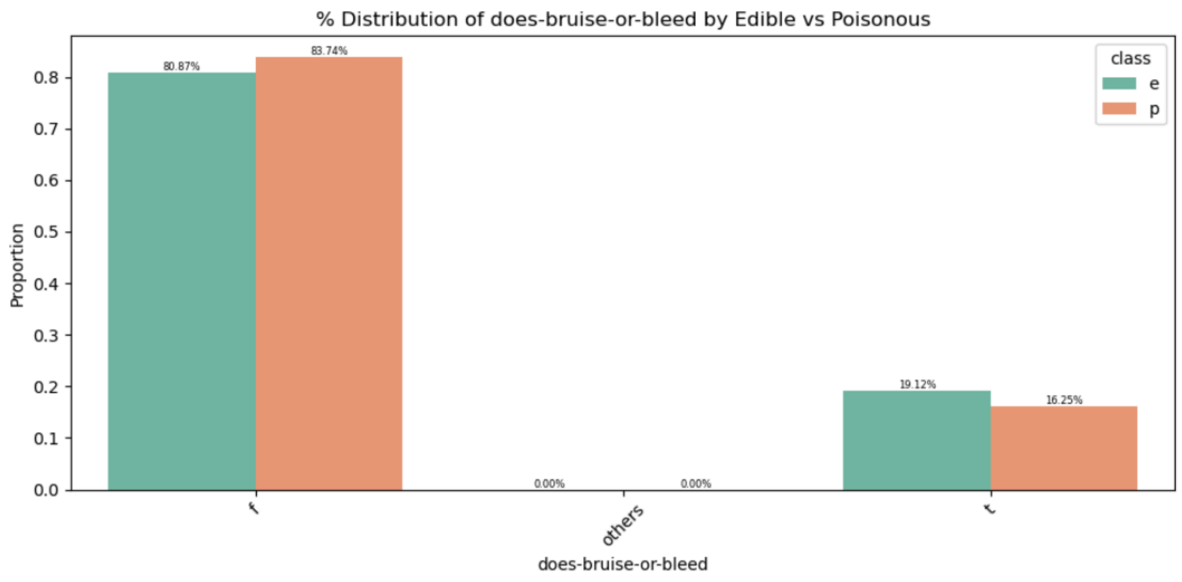
- The above compares the percentage breakdown of each category between e and p for this field. The biggest gaps are in the (x) and (b) category and (f). Rest are about the same.
- Statistical significance was observed for (x) and (b) (far right and far left). However, the biggest difference can be found for bell-shaped capped (b) mushrooms, which is almost 3 times as likely (14.44% vs 5.12%) for this mushroom to be poisonous than edible.
    - (x) and (f) had ~ 16% more likely to be edible vs poisonous.
    - (x), (f), (b) tested statistically significant, while the other variables did not.

<u>Cap Surface</u>



- Most mushrooms are brown (n). You can also see that brown (n) mushrooms are 25% more likely to be edible than poisonous.
- For yellow (y) mushrooms, you can see its ~ 40% more likely to be (p) than (e).
- If presented a red mushroom (e), it is 3 times as likely that the mushroom is poisonous than edible (9.18% vs 2.89%)
- It is 8 times as likely to see buff-colored (b) mushroom be edible than poisonous (3.8% vs 0.45%)
- It is also 8 times as likely to see a poisonous mushroom that is green (r) than edible (0.52% vs 4.16%)
- Everything but (u) and (c) was determined to be statistically significant.

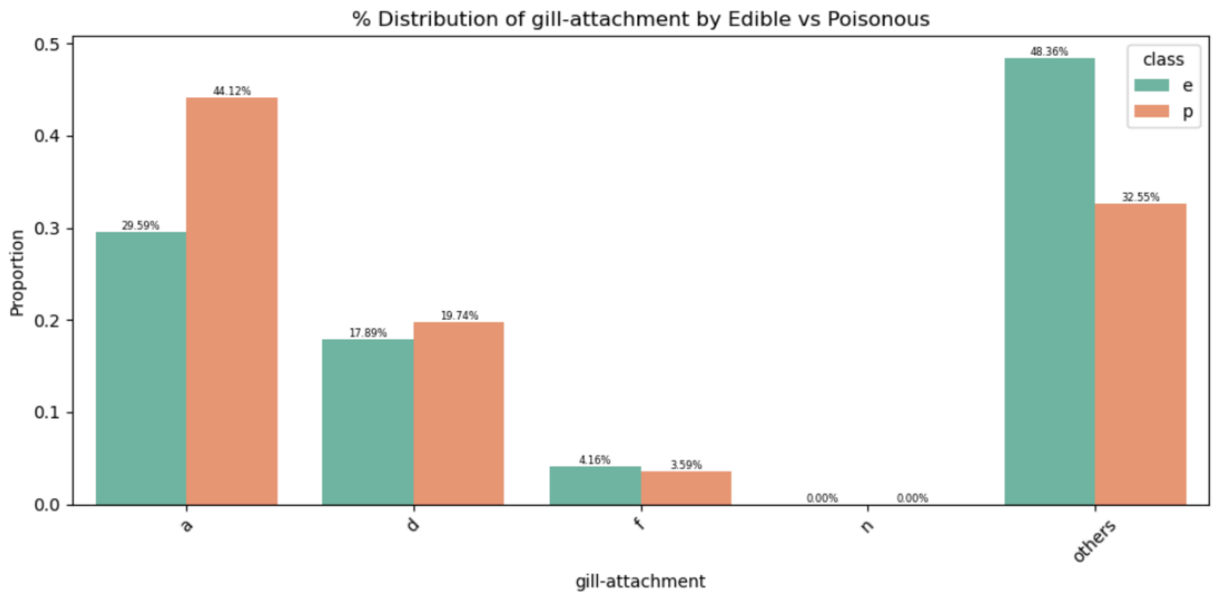
Bruise or Bleed



- Most mushrooms (~80%) don't bruise or bleed (f) while ~20% do (t).
  - There are some minor percentage differences but not a strong separator of classes.

While the difference between (f) and (t) did come out statistically significant, it is more likely due to the number of observations and sample size (millions) than lift. As a result, while statistically significant, it's likely that this field is not as predictive as the others.

Gill Attachment

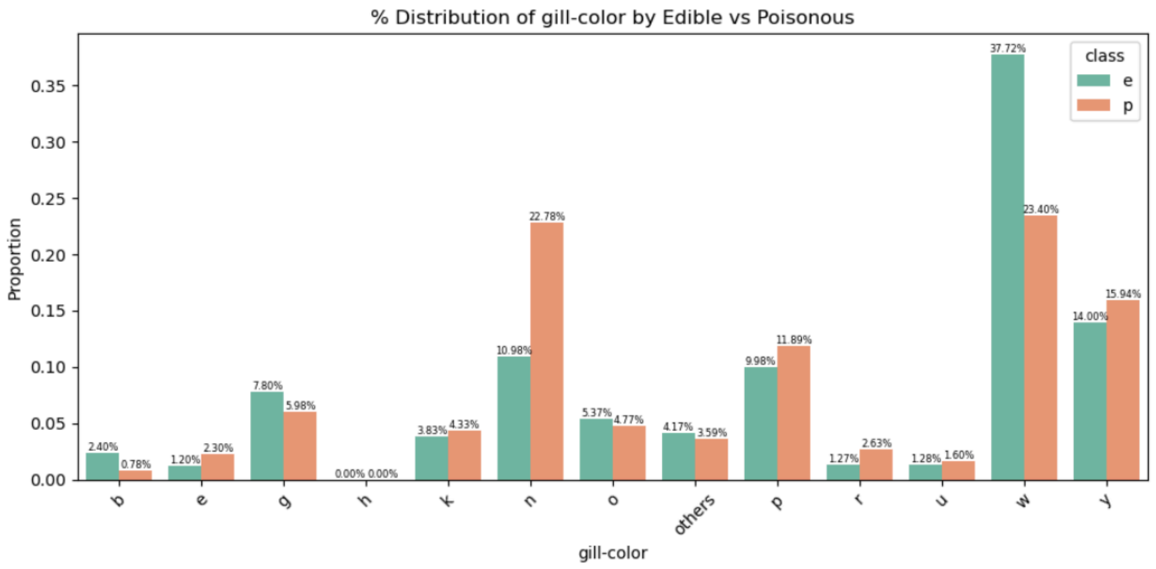


- Adnate mushrooms (a) are 1.5 times as likely to be poisonous than not (29.59% vs

44.12%).

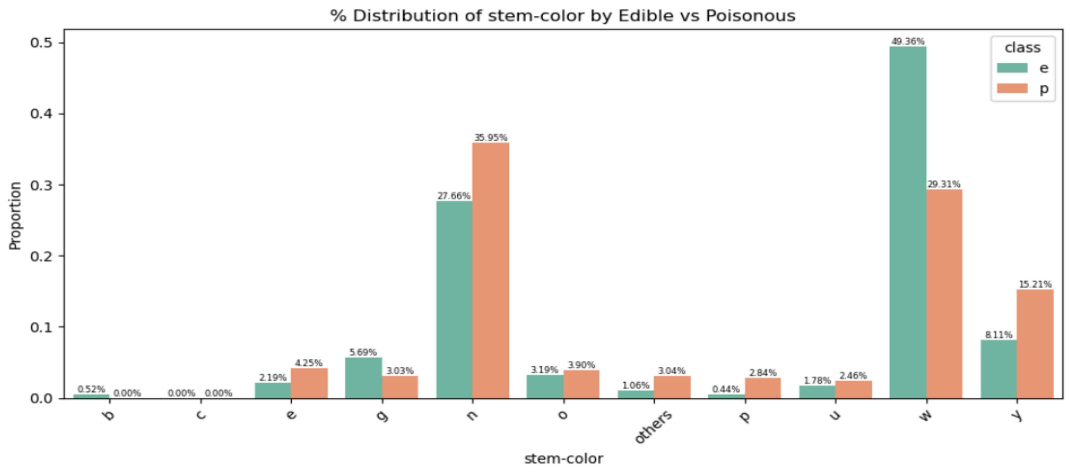
- For gill-attachment, others also contained a big deviance, however, caution was also undertaken when considering this variable due to possible mixed heterogeneity in this category of everything being unlabeled in others.
- (a) and others are considered statistically significant.

<u>Gill Color</u>



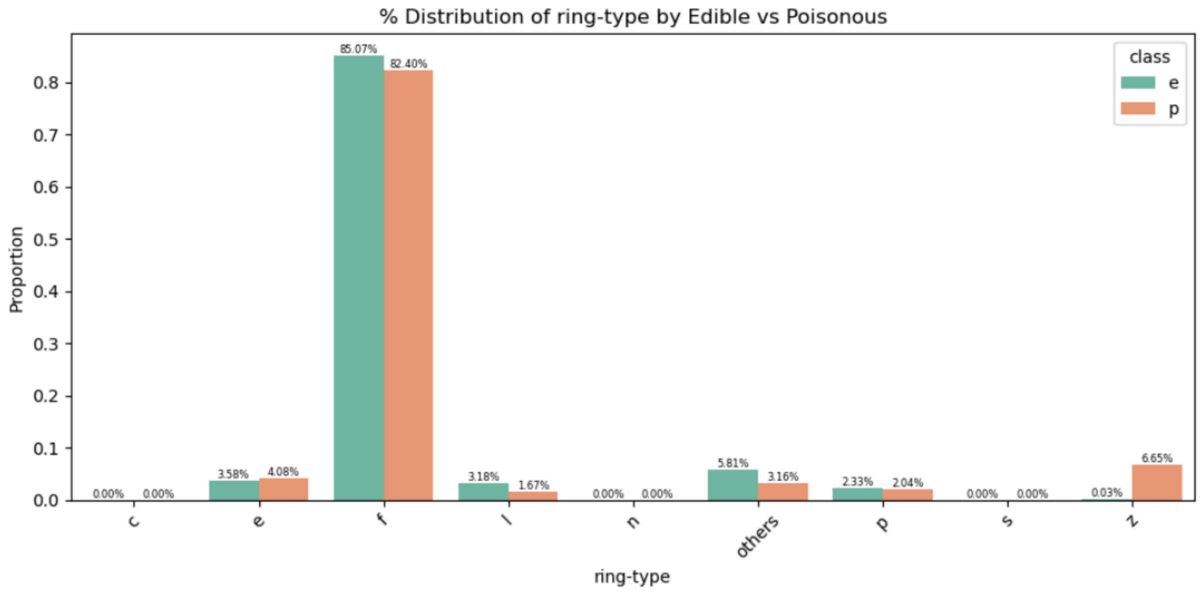
- The most frequent counts for gill are the white (w) and brown (n). They also have the biggest deviance when it comes to poisonous and edible percent frequency.
  - If the gill is white (w), the mushroom is almost twice as likely to be edible than poisonous.
  - If brown (n), then it is about two times more likely for the mushroom to be poisonous than edible.
- Other notable difference (although much less frequent) were:
  - If presented a buff-colored gill (b), the mushroom is more than twice as likely that the mushroom is edible than poisonous (2.4% vs 0.78%)
  - If red (e), then it is twice as likely to be poisonous than edible (1.20% vs 2.30%)
- (g), (p), (y) also had a small difference but not as great as the differences in the previous categories mentioned.
- The categories mentioned above, all passed were registered as statistical significant.

Stem Color



- If the stem color is white (w), the mushroom is 70% more likely to be edible than poisonous (49.36% vs 29.31%)
- If the mushroom's stem color is brown (n), the mushroom is 30% more likely to be poisonous (27.66% vs 35.95%)
- If it's gray (g), the mushroom is almost twice as likely to be edible than poisonous.
- If it's yellow (y), the mushroom is almost twice as likely to be poisonous than edible.
- If it's pink (p), the mushroom is almost 6 times more likely to be edible than poisonous.
- The others categories have a near 3x as likely to be poisonous vs edible.
- It was interesting to see that (b) only contained edible mushrooms.
- (w), (n), (g), (y), (p), (e), (b) and others also passed statistical significance test

Ring Type



- If the mushroom has a zone ring type (z), then it is EXTREMELY likely (221x) (see far right graph) that the mushroom is poisonous.
- If the ring type is large (l), then the mushroom is around two times as likely to be edible than poisonous
- Others show a 60% more likely to be edible than poisonous.
- Others, (l), (z), also passed statistical significance.

Model Implementation

When selecting models, we need to evaluate a range of options, balancing interpretability and flexibility. Interpretable models, like Logistic Regression, are easier to understand and implement, whereas flexible models, such as Random Forest, Decision Trees, and XGBoost, often offer higher accuracy. Given the severe consequences of false negatives (misclassifying poisonous mushrooms as edible), we will prioritize testing models known for their flexibility and predictive strength. Therefore, we have chosen to assess Logistic Regression, Decision Trees, Random Forest, and XGBoost. The data will be split into 70% training and 30% test. The 70/30 training/testing split was chosen as it aligns with traditional training/testing splits done by other studies on the topic (Rahma 2023), while leaving ample amount of test points for assessment.

Please see the **Results** section to see all the consolidated information on each model's cross validation and other scoring metrics.

Logistic Regression

This model is straightforward, highly interpretable, and computationally efficient, making it ideal for simpler datasets. However, it struggles with complex, non-linear relationships, potentially limiting its accuracy in more intricate scenarios. In the situation with our project, it is really good for interpretability and it would be super easy to implement by nontechnical people, but perform poorly in terms of its accuracy.

Decision Trees

Decision Trees are easy to interpret and visualize, capable of handling both numerical and categorical data. In the purpose of the mushroom study, decision trees are the models that most resemble dichotomous keys that many mycologists use. However, this type of model is prone to overfitting and generally offers lower accuracy compared to ensemble methods like Random Forest or XGBoost.

Random Forest

This model is known to deliver high accuracy and handle non-linear data while reducing overfitting through ensemble learning. The model works by creating a bunch of decision trees, and taking the overall averages of it. Its downside is reduced interpretability and higher computational demands, especially with large datasets. In the case of our project, this model may be good in performance, but not so much in terms of interpretation and implementation.

XGBoosting

XGBoost stands out for its exceptional performance on large datasets, excelling at managing non-linear relationships and missing data. On the other hand, it is complex to implement and fine-tune, requiring significant computational resources. We initially believed this model would have the best performance.

Metrics

Due to the classification nature of this problem, metrics such as Accuracy, Recall, and F1-score will be evaluated to assess model performance. While Precision is a great metric to measure the number of false positives, discarding a nontoxic mushroom is a better scenario than eating a toxic mushroom.

Accuracy measures the overall correctness of the model by calculating the ratio of correctly predicted instances to the total number of instances. While useful, it can be misleading in imbalanced datasets, like this one because there are more poisonous mushrooms than edible ones. The imbalance is not that bad here, so Accuracy still stands as a good benchmark. *Precision* focuses on the quality of positive predictions, calculated as the ratio of true positives to the sum of true and false positives. This metric is essential when the cost of false positives (incorrectly identifying an edible mushroom as poisonous) needs to be minimized, but in our case it is not as important as the other metrics. *Recall* measures the ability to correctly identify all positive instances. It is the ratio of true positives to the sum of true positives and false negatives. This metric is particularly crucial in this context, as false negatives (incorrectly classifying a poisonous mushroom as edible) pose severe safety risks. *F1-score* is the balance of Precision and Recall, offering a balanced measure that accounts for both false positives and false negatives. It is especially valuable when dealing with imbalanced datasets or when the cost of false positives and false negatives is high. By considering these metrics, we ensure a comprehensive evaluation that prioritizes safety and minimizes the risk of misclassification.

Cross Validation

To mitigate overfitting, we applied 5-fold cross-validation (CV) to each model. This approach reduces variance, optimizes data utilization, balances bias and variance, and enhances the model's robustness. Given the large size of our dataset (3 million rows), the CV results closely align with those from the training process for most models. However, in the case of the Decision

Tree, the performance metrics differ significantly, with the training data showing better results compared to the cross-validated version. This can be due to some rare cases: groupings that the CV model might have missed due to sampling or overfitting. Either case, the CV results show the results of more robust testing and will be used as the ultimate score board.

Results

Scoring Metrics	Logistic	Logistic CV'd	Random Forest	Random Forest CV'd	XGBoosting	XGBoosting CV'd	Decision Tree	Decision Tree CV'd
Accuracy	0.5471	0.5471	0.98705	0.98703	0.98398	0.98428	0.97505	0.87532
Precision (False)	0	0	0.98	0.98	0.98	0.98	0.97	0.82
Recall (False)	0	0	0.99	0.99	0.98	0.98	0.97	0.92
F1 Score (False)	0	0	0.99	0.99	0.98	0.98	0.97	0.87
Support (False)	423456	423456	423456	423456	423456	423456	423456	423456
Precision (True)	0.57	0.55	0.99	0.99	0.99	0.99	0.98	0.93
Recall (True)	1	1	0.99	0.99	0.98	0.98	0.98	0.84
F1 Score (True)	0.71	0.71	0.99	0.99	0.99	0.99	0.98	0.88
Support (True)	511628	511628	511628	511628	511628	511628	511628	511628

The performance evaluation of various models shows clear trends in their effectiveness at classifying mushrooms as either poisonous or edible. Logistic Regression performed poorly, with an accuracy of only 54.7%. It failed to identify any edible mushrooms correctly (False class), achieving 0% precision and recall for this class. Although it correctly classified all poisonous mushrooms (True class) with a 100% recall, its inability to handle the dataset's complexity makes it unsuitable for this task.

In contrast, Decision Tree performed well on the training data with an accuracy of 97.5%, but its cross-validated accuracy dropped significantly to 87.5%, suggesting potential overfitting. Random Forest and XGBoost demonstrated superior performance, achieving accuracies of 98.7% and 98.4%, respectively. Both models showed near-perfect precision and recall for both classes, indicating robust and consistent performance even under cross-validation, making them ideal choices for minimizing false negatives—critical for this application. The decline in precision and recall in the cross-validated version highlights its reduced reliability on unseen data. Overall, Random Forest and XGBoost offer the best balance between accuracy and robustness, ensuring safer and more reliable classification of poisonous mushrooms compared to simpler models like Logistic Regression and Decision Tree.

Thoughts, Theories, and Explanation for Results

This section tries to provide an explanation of why the non parametric models (Decision Tree, Random Forest, and Gradient Boosted) did better, it is presumed that this is due to the non-linearity in the data and the categorical buckets. So for example, logistic regression is good with taking average effects of each category but when it comes to interactions effects or sub categorical changes it's a bit hard for it to pick up. On the other hand, it's a lot easier for the tree based models to pick up on categories, subcategories, and sub subcategories for fields due to the category split and tree based models. So for example, if you have a mushroom with cap shape (X), Stem color (Y), and Gill (w), vs cap shape (X), Stem color (Y), and Gill (y), and there is a special case where for cap shape (X) and stem color (Y) gill changing plays a greater effect, the tree based would be able to pick up and adjust faster. These special cases can make a difference with performance and probably why the tree based performed better. This would also explain why the random forest and gradient boosted ensemble performed better, because there is certain layering (via ensembling) that the decision tree would not have been able to capture by itself and the additional models from the ensemble of decision trees would have helped with this. Another theory could be the class imbalance, but while the non parametric models are more robust to class imbalance, the class imbalance was not too off, so while this may play a part, it might not have contributed as much as other factors.

Please see the appendix to see the confusion matrices that were made for each model and its cross-validated version. For the purpose of readability, we chose to put that at the appendix.

Conclusion

Key metrics such as recall, F1 score, and overall accuracy are crucial in ensuring reliable classification of poisonous mushrooms. Although Random Forest demonstrated the highest performance, considerations around interpretability are also important; a model like Decision Tree offers a simpler, more transparent approach. Despite the Decision Tree having lower performance while providing better interpretability, the critical nature of this task underscores the importance of minimizing false negatives. Therefore, selecting a model with the highest predictive accuracy, such as Random Forest, is the safer choice to mitigate the potentially severe consequences of misclassifying a poisonous mushroom.

Future Recommendations

This section talks about ways this can be improved in the future.

Modeling Improvements

One can incorporate more advanced modeling techniques like neural networks and see if those can cause an improvement. Combining and ensemble different models (random forest and neural network) might improve the score in that last few percentages. Another option can be creating a residual correcting model that one can attach to this current model (form of model boosting). One would examine where the model is ineffective (misclassified), examine attributes associated with those errors and build a model as an extension to help better handle those edge cases. One can also use additional metadata to help with classification (Sladojevic 2016).

Improvements via Data, Domain, & Expert Input

Other options would be: expanding the dataset with diverse, real-world mushroom species and environmental variables to improve the model robustness. Developing an interactive, user-friendly application that translates complex model outputs into easily understandable information for non-scientists would further bridge the gap between research and practical use. Finally, continuous collaboration with mycologists and field experts can ensure the models evolve with the latest research, maintaining high accuracy and relevance in mushroom classification.

Lessons We Have Learned

Throughout this project, we also gained important lessons about the principles and practices emphasized in our class. We learned the value of applying a structured, end-to-end approach to problem-solving, from data preprocessing and model selection to evaluation and interpretation. The importance of choosing appropriate metrics, such as accuracy, recall, and F1 score, was reinforced, particularly in scenarios where the cost of false negatives is high. This project highlighted the real-world relevance of concepts like bias-variance trade-off and the critical role of cross-validation in assessing model performance and preventing overfitting.

We also recognized the importance of balancing predictive accuracy with interpretability, a recurring theme in class discussions. While powerful models like Random Forest and XGBoost offer superior performance, simpler models like Decision Trees demonstrate the trade-offs between transparency and complexity. This aligns with the class's focus on understanding not just how to build models, but also how to interpret and explain them to stakeholders.

Additionally, the knowledge from class helped us better understand and theorize why the models performed as well as they have in general and also comparative to each other.

Overall, this project brought classroom concepts to life, deepening our understanding of machine learning workflows and the importance of ethical and contextual considerations in real-world applications.

Citations

Visser, M. (2023, May 16). *How to identify poisonous mushrooms*. Environment Co. https://environment.co/how-to-identify-poisonous-mushrooms/

UCLA Health. *7 health benefits of mushrooms*. (2022, January 24). https://www.uclahealth.org/news/article/7-health-benefits-of-mushrooms

Mushroom Dataset. UCI Machine Learning Repository. (n.d.). https://archive.ics.uci.edu/dataset/848/secondary+mushroom+dataset

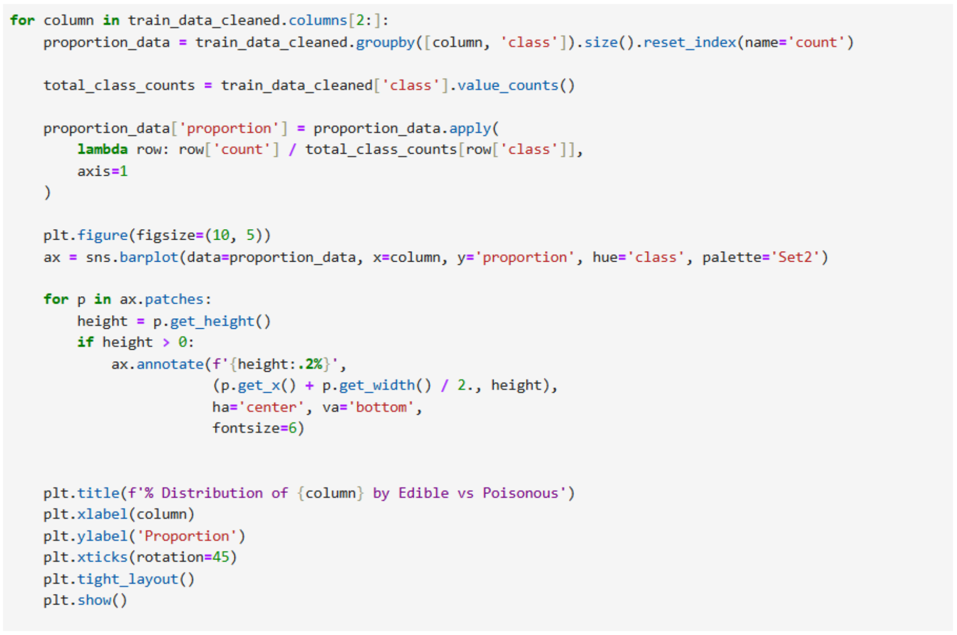
Rahma (2023 November) *The classification of Mushroom using ML* https://kjis.journals.ekb.eg/article_327330_afd343addc365a1440bbf1bf24d683bf.pdf

Sladojevic S, Arsenovic M, Anderla A, Culibrk D and Stefanovic D (2016 Deep neural networks based recognition of plant diseases by leaf image classification. Computational Intelligence and Neuroscience 2016).

Appendix

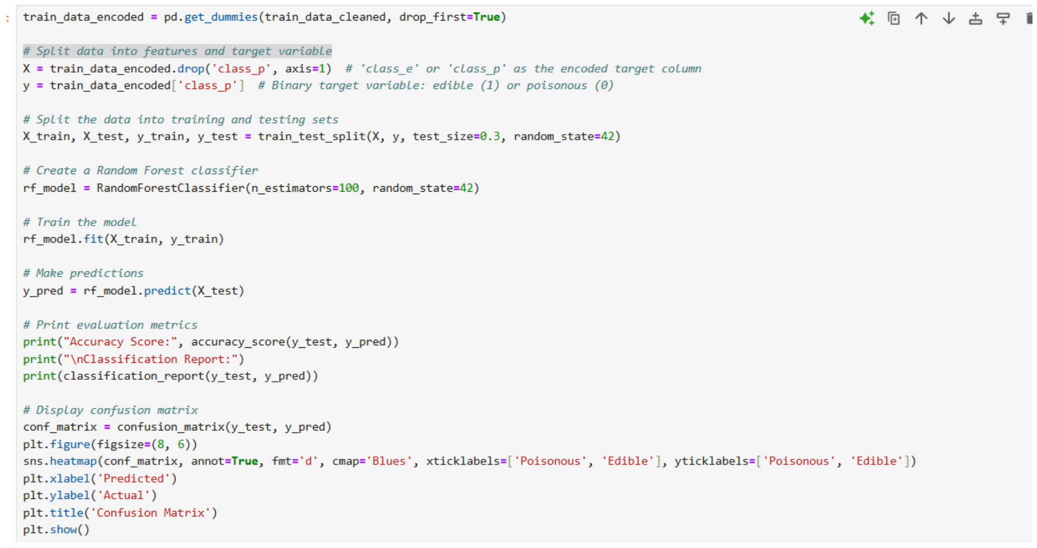
EDA Charts

all within one loop since most are categorical variables



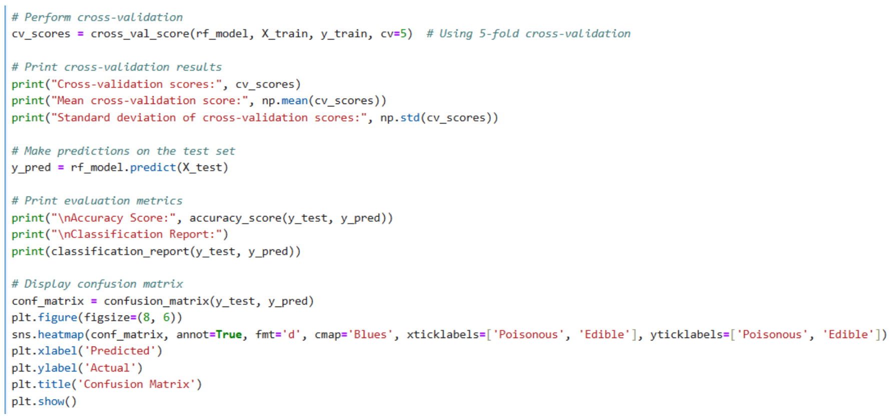
```
for column in train_data_cleaned.columns[2:]:
    proportion_data = train_data_cleaned.groupby([column, 'class']).size().reset_index(name='count')

    total_class_counts = train_data_cleaned['class'].value_counts()

    proportion_data['proportion'] = proportion_data.apply(
        lambda row: row['count'] / total_class_counts[row['class']],
        axis=1
    )

    plt.figure(figsize=(10, 5))
    ax = sns.barplot(data=proportion_data, x=column, y='proportion', hue='class', palette='Set2')

    for p in ax.patches:
        height = p.get_height()
        if height > 0:
            ax.annotate(f'{height:.2%}',
                        (p.get_x() + p.get_width() / 2., height),
                        ha='center', va='bottom',
                        fontsize=6)


    plt.title(f'% Distribution of {column} by Edible vs Poisonous')
    plt.xlabel(column)
    plt.ylabel('Proportion')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

Random Forest Code



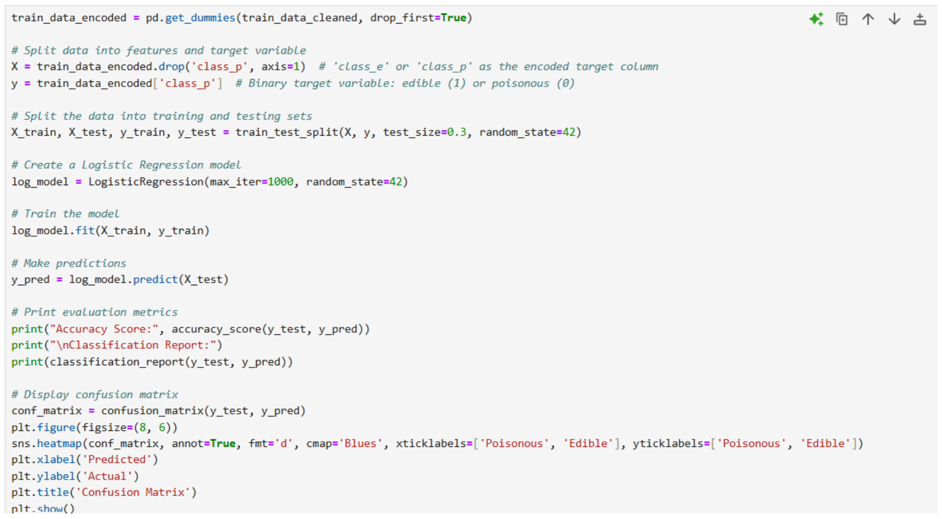
```python
train_data_encoded = pd.get_dummies(train_data_cleaned, drop_first=True)

# Split data into features and target variable
X = train_data_encoded.drop('class_p', axis=1)  # 'class_e' or 'class_p' as the encoded target column
y = train_data_encoded['class_p']  # Binary target variable: edible (1) or poisonous (0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Print evaluation metrics
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Poisonous', 'Edible'], yticklabels=['Poisonous', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Random Forest 5-Fold Cross Validation Code



```python
# Perform cross-validation
cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5)  # Using 5-fold cross-validation

# Print cross-validation results
print("Cross-validation scores:", cv_scores)
print("Mean cross-validation score:", np.mean(cv_scores))
print("Standard deviation of cross-validation scores:", np.std(cv_scores))

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Print evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Poisonous', 'Edible'], yticklabels=['Poisonous', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Logistic Regression Code



```python
train_data_encoded = pd.get_dummies(train_data_cleaned, drop_first=True)

# Split data into features and target variable
X = train_data_encoded.drop('class_p', axis=1)  # 'class_e' or 'class_p' as the encoded target column
y = train_data_encoded['class_p']  # Binary target variable: edible (1) or poisonous (0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Logistic Regression model
log_model = LogisticRegression(max_iter=1000, random_state=42)

# Train the model
log_model.fit(X_train, y_train)

# Make predictions
y_pred = log_model.predict(X_test)

# Print evaluation metrics
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Poisonous', 'Edible'], yticklabels=['Poisonous', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

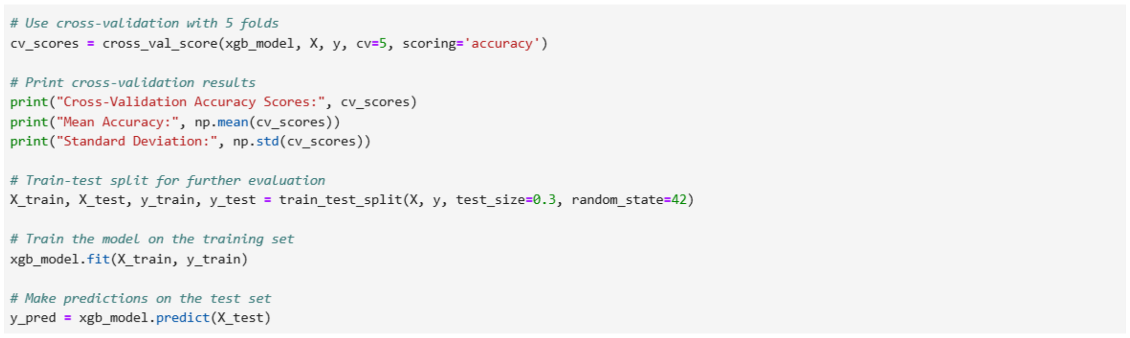
Logistic Regression 5-Fold Cross Validation Code

```python
# Create a Logistic Regression model
log_model = LogisticRegression(max_iter=1000, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(log_model, X_train, y_train, cv=5, scoring='accuracy')

# Print cross-validation scores
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", cv_scores.mean())

# Train the model on the entire training set
log_model.fit(X_train, y_train)

# Make predictions
y_pred = log_model.predict(X_test)

# Print evaluation metrics
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Poisonous', 'Edible'], yticklabels=['Poisonous', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

XGBoost Model Code

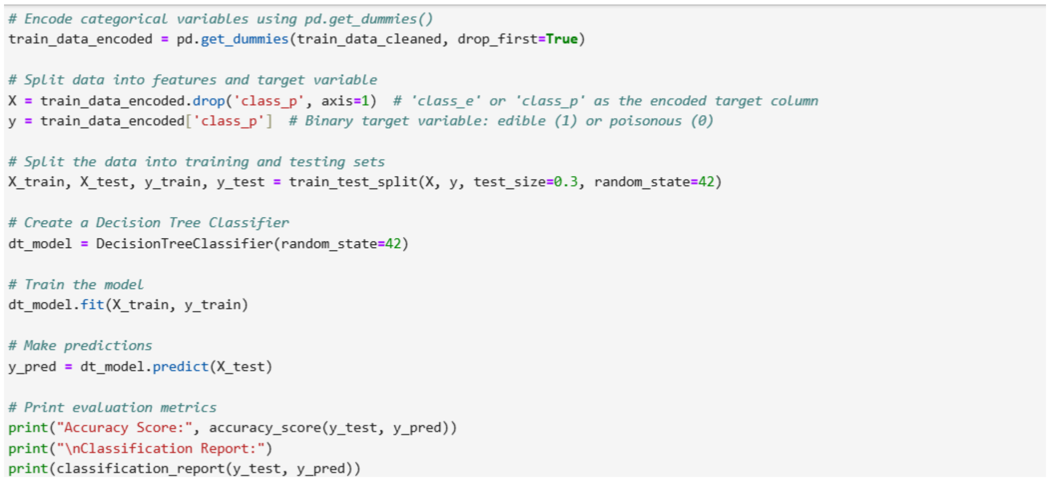
```python
train_data_encoded = pd.get_dummies(train_data_cleaned, drop_first=True)

# Split data into features and target variable
X = train_data_encoded.drop('class_p', axis=1)
y = train_data_encoded['class_p']

# Perform cross-validation
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

# Use cross-validation with 5 folds
cv_scores = cross_val_score(xgb_model, X, y, cv=5, scoring='accuracy')

# Print cross-validation results
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean Accuracy:", np.mean(cv_scores))
print("Standard Deviation:", np.std(cv_scores))

# Train-test split for further evaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model on the training set
xgb_model.fit(X_train, y_train)
```

XGBoosting Model 5-Fold Cross Validation Code



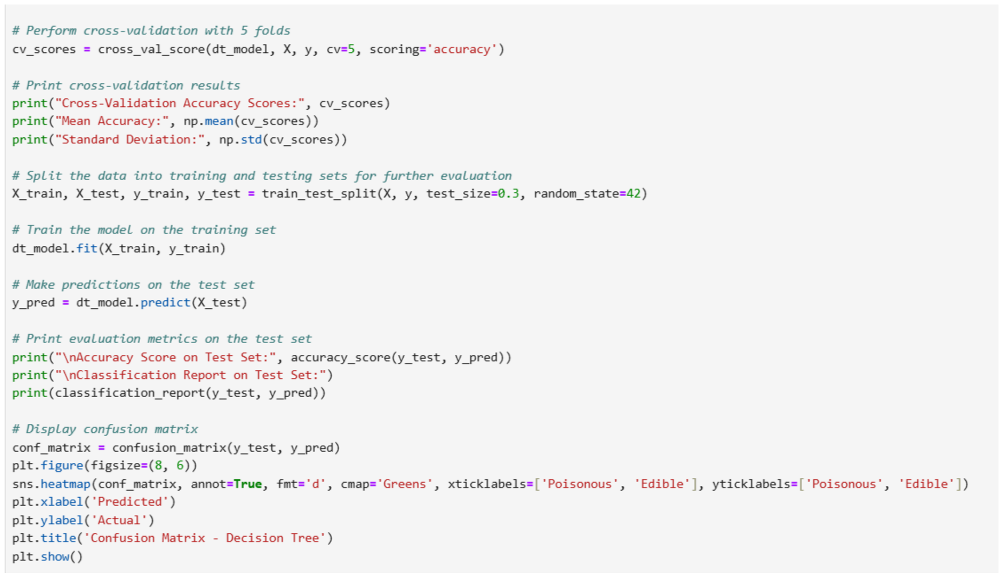
```
# Use cross-validation with 5 folds
cv_scores = cross_val_score(xgb_model, X, y, cv=5, scoring='accuracy')

# Print cross-validation results
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean Accuracy:", np.mean(cv_scores))
print("Standard Deviation:", np.std(cv_scores))

# Train-test split for further evaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model on the training set
xgb_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = xgb_model.predict(X_test)
```

Decision Tree Model Code



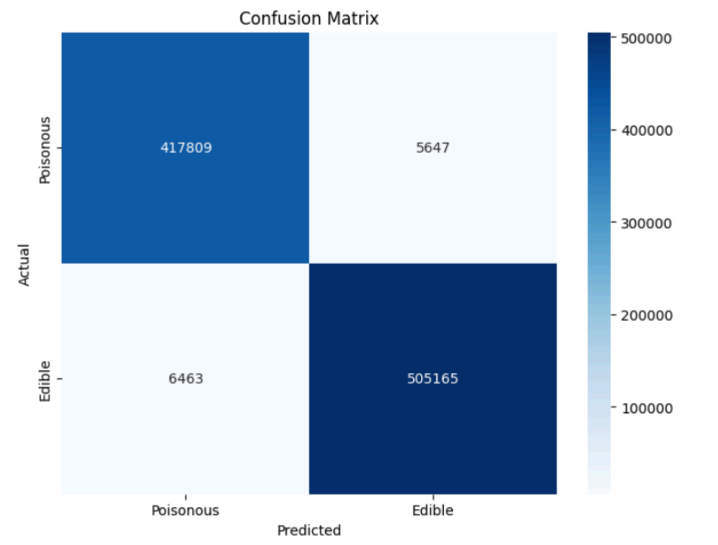
```
# Encode categorical variables using pd.get_dummies()
train_data_encoded = pd.get_dummies(train_data_cleaned, drop_first=True)

# Split data into features and target variable
X = train_data_encoded.drop('class_p', axis=1)  # 'class_e' or 'class_p' as the encoded target column
y = train_data_encoded['class_p']  # Binary target variable: edible (1) or poisonous (0)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
dt_model.fit(X_train, y_train)

# Make predictions
y_pred = dt_model.predict(X_test)

# Print evaluation metrics
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Decision Tree 5-Fold Cross Validation Code

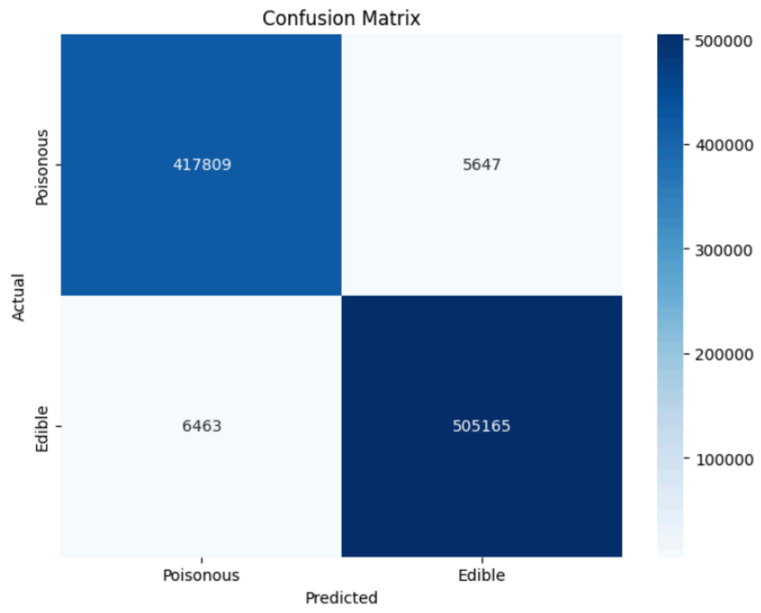


```
# Perform cross-validation with 5 folds
cv_scores = cross_val_score(dt_model, X, y, cv=5, scoring='accuracy')

# Print cross-validation results
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean Accuracy:", np.mean(cv_scores))
print("Standard Deviation:", np.std(cv_scores))

# Split the data into training and testing sets for further evaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train the model on the training set
dt_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = dt_model.predict(X_test)

# Print evaluation metrics on the test set
print("\nAccuracy Score on Test Set:", accuracy_score(y_test, y_pred))
print("\nClassification Report on Test Set:")
print(classification_report(y_test, y_pred))

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens', xticklabels=['Poisonous', 'Edible'], yticklabels=['Poisonous', 'Edible'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Decision Tree')
plt.show()
```

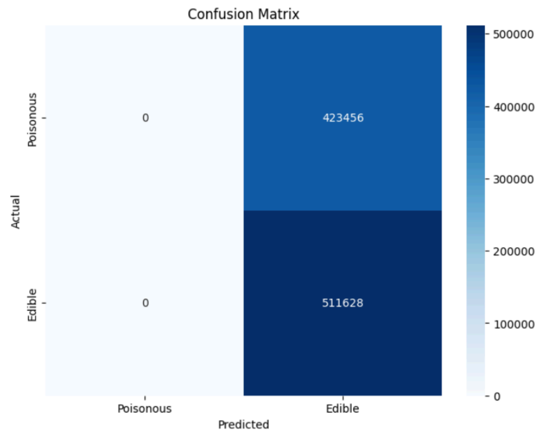
Random Forest Confusion Matrix



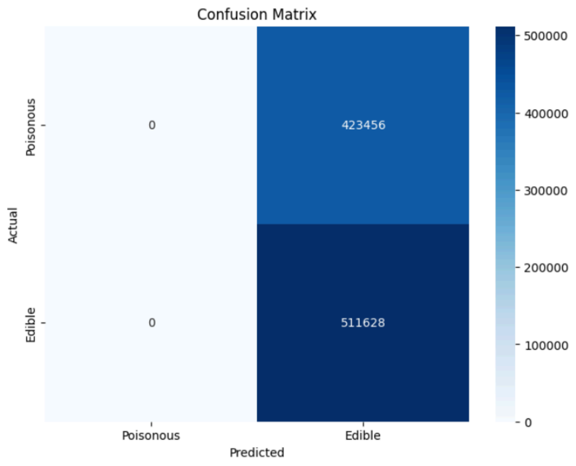
Random Forest 5-Fold Cross Validation Confusion Matrix



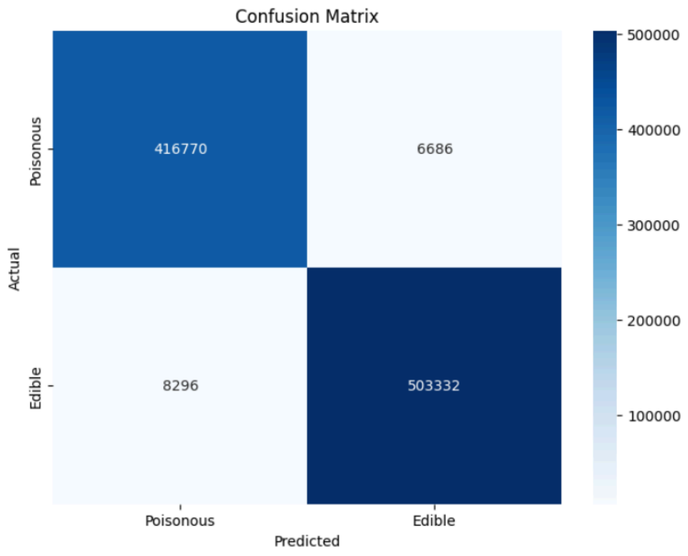
Logistic Regression Confusion Matrix



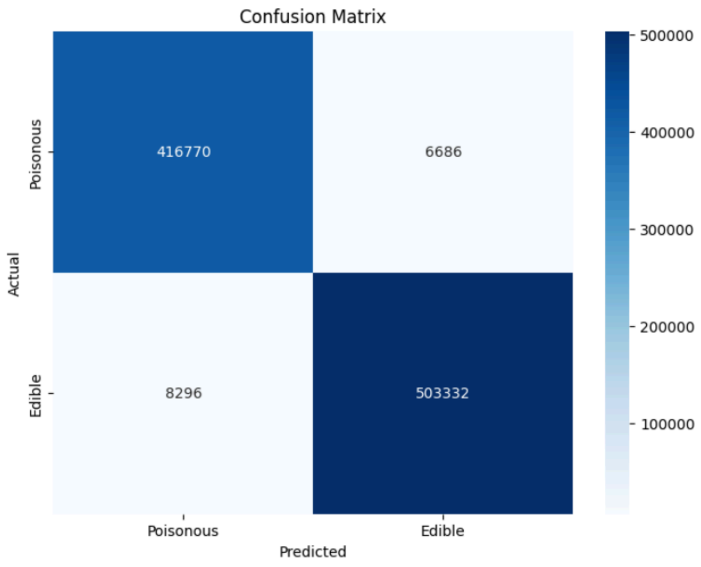
Logistic Regression 5-Fold Cross Validation Confusion Matrix



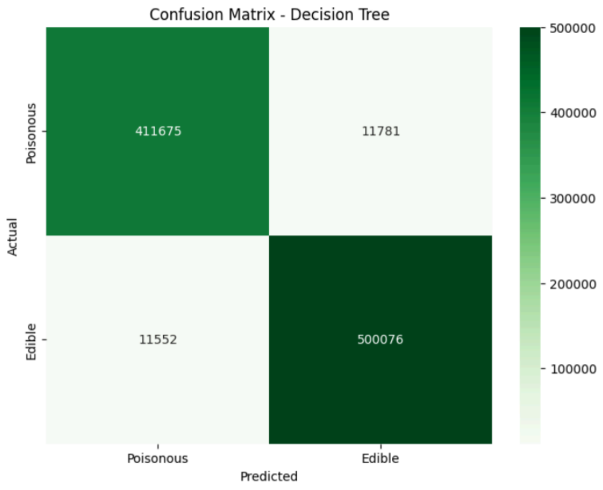
XGBoost Confusion Matrix



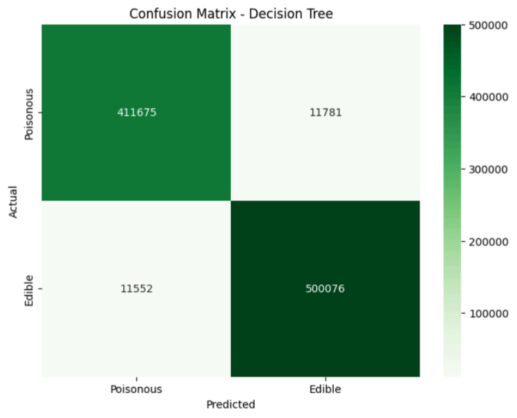
XGBoost Cross Validation Confusion Matrix



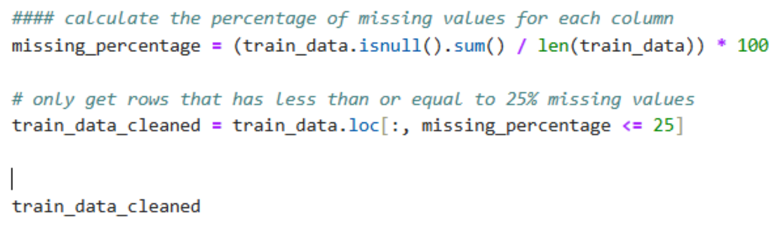
Decision Tree Confusion Matrix



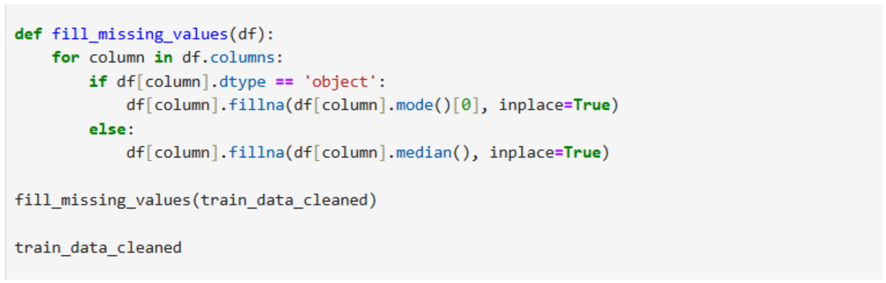
Decision Tree Cross Validation Confusion Matrix



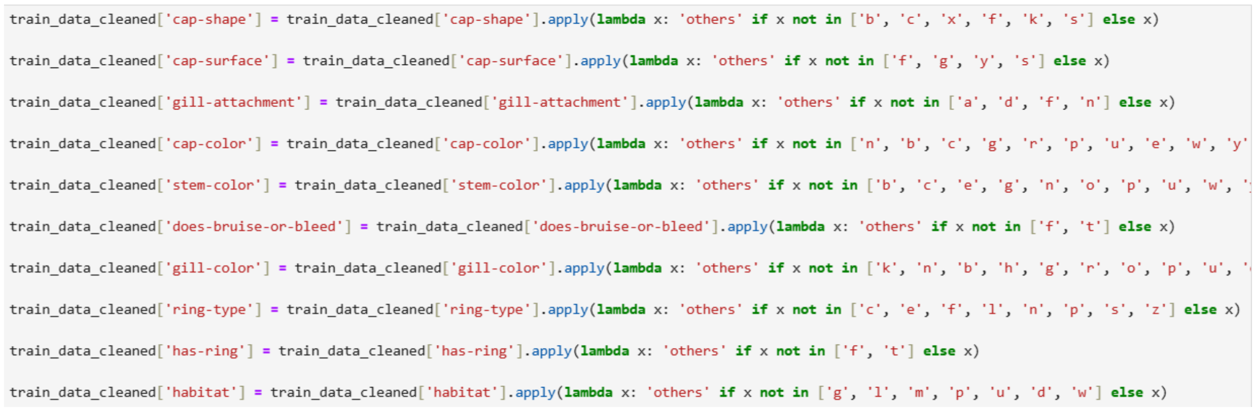
Data Cleaning Code



```python
#### calculate the percentage of missing values for each column
missing_percentage = (train_data.isnull().sum() / len(train_data)) * 100

# only get rows that has less than or equal to 25% missing values
train_data_cleaned = train_data.loc[:, missing_percentage <= 25]

|
train_data_cleaned
```



```python
def fill_missing_values(df):
    for column in df.columns:
        if df[column].dtype == 'object':
            df[column].fillna(df[column].mode()[0], inplace=True)
        else:
            df[column].fillna(df[column].median(), inplace=True)

fill_missing_values(train_data_cleaned)

train_data_cleaned
```



```python
train_data_cleaned['cap-shape'] = train_data_cleaned['cap-shape'].apply(lambda x: 'others' if x not in ['b', 'c', 'x', 'f', 'k', 's'] else x)

train_data_cleaned['cap-surface'] = train_data_cleaned['cap-surface'].apply(lambda x: 'others' if x not in ['f', 'g', 'y', 's'] else x)

train_data_cleaned['gill-attachment'] = train_data_cleaned['gill-attachment'].apply(lambda x: 'others' if x not in ['a', 'd', 'f', 'n'] else x)

train_data_cleaned['cap-color'] = train_data_cleaned['cap-color'].apply(lambda x: 'others' if x not in ['n', 'b', 'c', 'g', 'r', 'p', 'u', 'e', 'w', 'y'

train_data_cleaned['stem-color'] = train_data_cleaned['stem-color'].apply(lambda x: 'others' if x not in ['b', 'c', 'e', 'g', 'n', 'o', 'p', 'u', 'w', '

train_data_cleaned['does-bruise-or-bleed'] = train_data_cleaned['does-bruise-or-bleed'].apply(lambda x: 'others' if x not in ['f', 't'] else x)

train_data_cleaned['gill-color'] = train_data_cleaned['gill-color'].apply(lambda x: 'others' if x not in ['k', 'n', 'b', 'h', 'g', 'r', 'o', 'p', 'u', '

train_data_cleaned['ring-type'] = train_data_cleaned['ring-type'].apply(lambda x: 'others' if x not in ['c', 'e', 'f', 'l', 'n', 'p', 's', 'z'] else x)

train_data_cleaned['has-ring'] = train_data_cleaned['has-ring'].apply(lambda x: 'others' if x not in ['f', 't'] else x)

train_data_cleaned['habitat'] = train_data_cleaned['habitat'].apply(lambda x: 'others' if x not in ['g', 'l', 'm', 'p', 'u', 'd', 'w'] else x)
```