Mushroom Midterm

### Predicting the edibility of mushrooms

### Philip Fowler

### date started: 2025-03-17

# Section 1 Load and Explore the Data

## 1.1 Load the Data

# import libraries  
import pandas as pd  
import numpy as np  
import scipy.stats as stats  
import statsmodels.api as sm  
import matplotlib.pyplot as plt  
import seaborn as sns  
import warnings  
from ucimlrepo import fetch\_ucirepo  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression, Lasso  
from sklearn.preprocessing import LabelEncoder, StandardScaler  
from sklearn.feature\_selection import chi2, SelectKBest, RFE  
from sklearn.metrics import recall\_score, precision\_score, f1\_score, classification\_report, confusion\_matrix, accuracy\_score  
from sklearn.tree import DecisionTreeClassifier, plot\_tree  
from sklearn.svm import SVC  
  
dont\_panic = 42  
warnings.filterwarnings('ignore')  
sns.set(style="whitegrid")

# fetch dataset   
mushroom = fetch\_ucirepo(id=73)   
   
# data (as pandas dataframes)   
x = mushroom.data.features   
y = mushroom.data.targets   
   
# metadata   
print(mushroom.metadata)   
# I love that UCI includes this metadata. It is in the readme as it is easier to access there.

{'uci\_id': 73, 'name': 'Mushroom', 'repository\_url': 'https://archive.ics.uci.edu/dataset/73/mushroom', 'data\_url': 'https://archive.ics.uci.edu/static/public/73/data.csv', 'abstract': 'From Audobon Society Field Guide; mushrooms described in terms of physical characteristics; classification: poisonous or edible', 'area': 'Biology', 'tasks': ['Classification'], 'characteristics': ['Multivariate'], 'num\_instances': 8124, 'num\_features': 22, 'feature\_types': ['Categorical'], 'demographics': [], 'target\_col': ['poisonous'], 'index\_col': None, 'has\_missing\_values': 'yes', 'missing\_values\_symbol': 'NaN', 'year\_of\_dataset\_creation': 1981, 'last\_updated': 'Thu Aug 10 2023', 'dataset\_doi': '10.24432/C5959T', 'creators': [], 'intro\_paper': None, 'additional\_info': {'summary': "This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and Ivy.", 'purpose': None, 'funded\_by': None, 'instances\_represent': None, 'recommended\_data\_splits': None, 'sensitive\_data': None, 'preprocessing\_description': None, 'variable\_info': ' 1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s\r\n 2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s\r\n 3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y\r\n 4. bruises?: bruises=t,no=f\r\n 5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s\r\n 6. gill-attachment: attached=a,descending=d,free=f,notched=n\r\n 7. gill-spacing: close=c,crowded=w,distant=d\r\n 8. gill-size: broad=b,narrow=n\r\n 9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y\r\n 10. stalk-shape: enlarging=e,tapering=t\r\n 11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?\r\n 12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s\r\n 13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s\r\n 14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y\r\n 15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y\r\n 16. veil-type: partial=p,universal=u\r\n 17. veil-color: brown=n,orange=o,white=w,yellow=y\r\n 18. ring-number: none=n,one=o,two=t\r\n 19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z\r\n 20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y\r\n 21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y\r\n 22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d', 'citation': None}}

## 1.2 Check for missing and display summary statistics

# variable information   
print(mushroom.variables)

name role type demographic \  
0 poisonous Target Categorical None   
1 cap-shape Feature Categorical None   
2 cap-surface Feature Categorical None   
3 cap-color Feature Binary None   
4 bruises Feature Categorical None   
5 odor Feature Categorical None   
6 gill-attachment Feature Categorical None   
7 gill-spacing Feature Categorical None   
8 gill-size Feature Categorical None   
9 gill-color Feature Categorical None   
10 stalk-shape Feature Categorical None   
11 stalk-root Feature Categorical None   
12 stalk-surface-above-ring Feature Categorical None   
13 stalk-surface-below-ring Feature Categorical None   
14 stalk-color-above-ring Feature Categorical None   
15 stalk-color-below-ring Feature Categorical None   
16 veil-type Feature Binary None   
17 veil-color Feature Categorical None   
18 ring-number Feature Categorical None   
19 ring-type Feature Categorical None   
20 spore-print-color Feature Categorical None   
21 population Feature Categorical None   
22 habitat Feature Categorical None   
  
 description units missing\_values   
0 None None no   
1 bell=b,conical=c,convex=x,flat=f, knobbed=k,su... None no   
2 fibrous=f,grooves=g,scaly=y,smooth=s None no   
3 brown=n,buff=b,cinnamon=c,gray=g,green=r, pink... None no   
4 bruises=t,no=f None no   
5 almond=a,anise=l,creosote=c,fishy=y,foul=f, mu... None no   
6 attached=a,descending=d,free=f,notched=n None no   
7 close=c,crowded=w,distant=d None no   
8 broad=b,narrow=n None no   
9 black=k,brown=n,buff=b,chocolate=h,gray=g, gre... None no   
10 enlarging=e,tapering=t None no   
11 bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,... None yes   
12 fibrous=f,scaly=y,silky=k,smooth=s None no   
13 fibrous=f,scaly=y,silky=k,smooth=s None no   
14 brown=n,buff=b,cinnamon=c,gray=g,orange=o, pin... None no   
15 brown=n,buff=b,cinnamon=c,gray=g,orange=o, pin... None no   
16 partial=p,universal=u None no   
17 brown=n,orange=o,white=w,yellow=y None no   
18 none=n,one=o,two=t None no   
19 cobwebby=c,evanescent=e,flaring=f,large=l, non... None no   
20 black=k,brown=n,buff=b,chocolate=h,green=r, or... None no   
21 abundant=a,clustered=c,numerous=n, scattered=s... None no   
22 grasses=g,leaves=l,meadows=m,paths=p, urban=u,... None no

# The features we will use in our model.   
x.head()

|  | cap-shape | cap-surface | cap-color | bruises | odor | gill-attachment | gill-spacing | gill-size | gill-color | stalk-shape | ... | stalk-surface-below-ring | stalk-color-above-ring | stalk-color-below-ring | veil-type | veil-color | ring-number | ring-type | spore-print-color | population | habitat |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | x | s | n | t | p | f | c | n | k | e | ... | s | w | w | p | w | o | p | k | s | u |
| 1 | x | s | y | t | a | f | c | b | k | e | ... | s | w | w | p | w | o | p | n | n | g |
| 2 | b | s | w | t | l | f | c | b | n | e | ... | s | w | w | p | w | o | p | n | n | m |
| 3 | x | y | w | t | p | f | c | n | n | e | ... | s | w | w | p | w | o | p | k | s | u |
| 4 | x | s | g | f | n | f | w | b | k | t | ... | s | w | w | p | w | o | e | n | a | g |

# Our target variable.  
y.head()

|  | poisonous |
| --- | --- |
| 0 | p |
| 1 | e |
| 2 | e |
| 3 | p |
| 4 | e |

# How many rows and columns?  
x.shape

(8124, 22)

y.shape

(8124, 1)

# Let's get some descriptive statistics.  
x.describe().T

|  | count | unique | top | freq |
| --- | --- | --- | --- | --- |
| cap-shape | 8124 | 6 | x | 3656 |
| cap-surface | 8124 | 4 | y | 3244 |
| cap-color | 8124 | 10 | n | 2284 |
| bruises | 8124 | 2 | f | 4748 |
| odor | 8124 | 9 | n | 3528 |
| gill-attachment | 8124 | 2 | f | 7914 |
| gill-spacing | 8124 | 2 | c | 6812 |
| gill-size | 8124 | 2 | b | 5612 |
| gill-color | 8124 | 12 | b | 1728 |
| stalk-shape | 8124 | 2 | t | 4608 |
| stalk-root | 5644 | 4 | b | 3776 |
| stalk-surface-above-ring | 8124 | 4 | s | 5176 |
| stalk-surface-below-ring | 8124 | 4 | s | 4936 |
| stalk-color-above-ring | 8124 | 9 | w | 4464 |
| stalk-color-below-ring | 8124 | 9 | w | 4384 |
| veil-type | 8124 | 1 | p | 8124 |
| veil-color | 8124 | 4 | w | 7924 |
| ring-number | 8124 | 3 | o | 7488 |
| ring-type | 8124 | 5 | p | 3968 |
| spore-print-color | 8124 | 9 | w | 2388 |
| population | 8124 | 6 | v | 4040 |
| habitat | 8124 | 7 | d | 3148 |

y.describe().T

|  | count | unique | top | freq |
| --- | --- | --- | --- | --- |
| poisonous | 8124 | 2 | e | 4208 |

# Do we have nulls?  
x.isnull().sum()

cap-shape 0  
cap-surface 0  
cap-color 0  
bruises 0  
odor 0  
gill-attachment 0  
gill-spacing 0  
gill-size 0  
gill-color 0  
stalk-shape 0  
stalk-root 2480  
stalk-surface-above-ring 0  
stalk-surface-below-ring 0  
stalk-color-above-ring 0  
stalk-color-below-ring 0  
veil-type 0  
veil-color 0  
ring-number 0  
ring-type 0  
spore-print-color 0  
population 0  
habitat 0  
dtype: int64

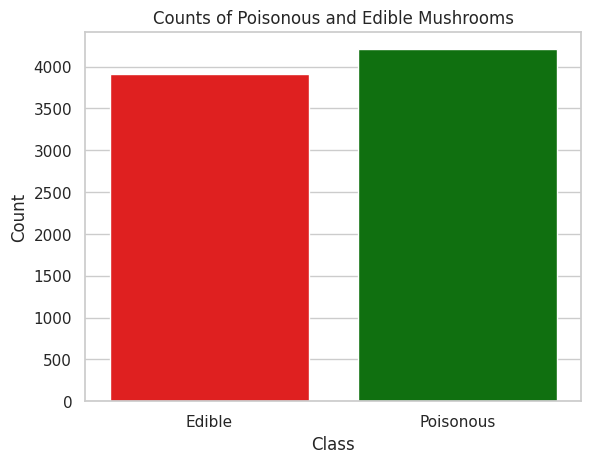
# Frequency of unique values for stalk-root  
stalk\_root\_counts = x['stalk-root'].value\_counts()  
print(stalk\_root\_counts)

stalk-root  
b 3776  
e 1120  
c 556  
r 192  
Name: count, dtype: int64

# Section 2 Visualize the Data

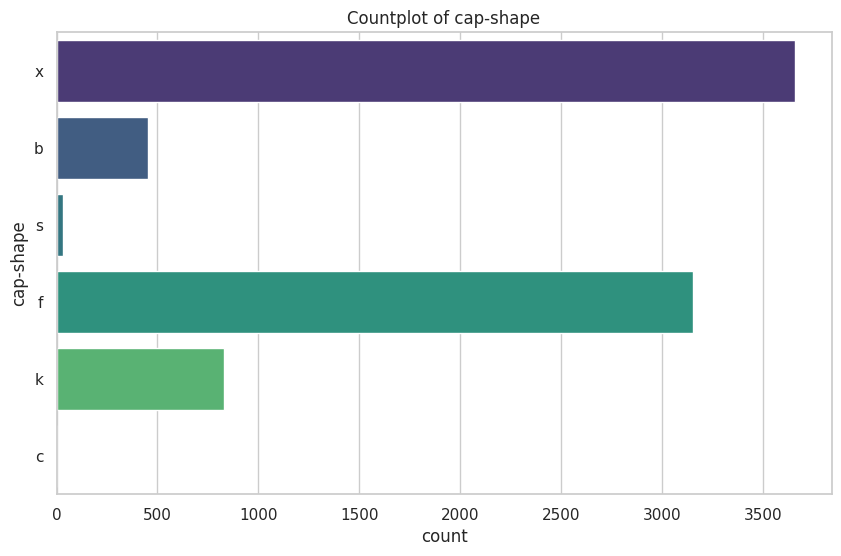
## 2.1 Create plots to visualize the data

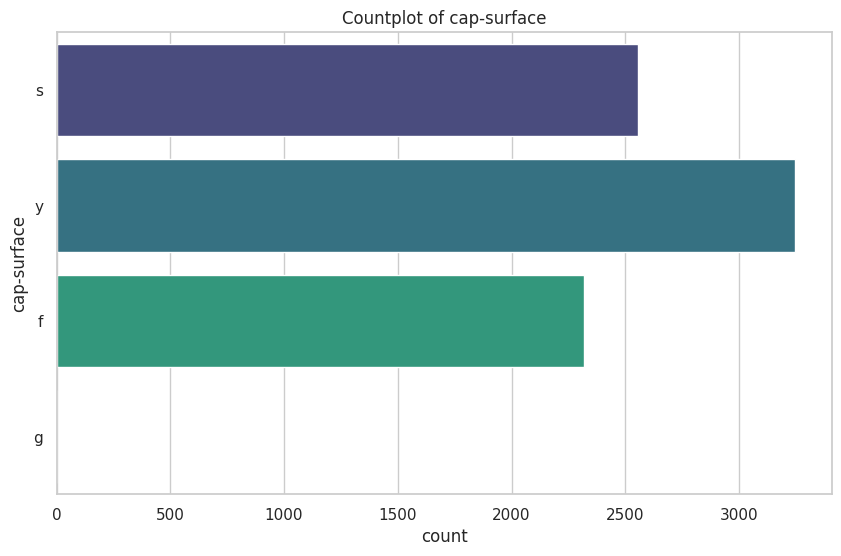
# Count the occurrences of each class in the target variable  
class\_counts = y['poisonous'].value\_counts()  
  
# Plot the bar chart using seaborn  
sns.countplot(x='poisonous', data=y, palette=['red', 'green'])  
plt.title('Counts of Poisonous and Edible Mushrooms')  
plt.xlabel('Class')  
plt.ylabel('Count')  
plt.xticks(ticks=[0, 1], labels=['Edible', 'Poisonous'], rotation=0)  
plt.show()

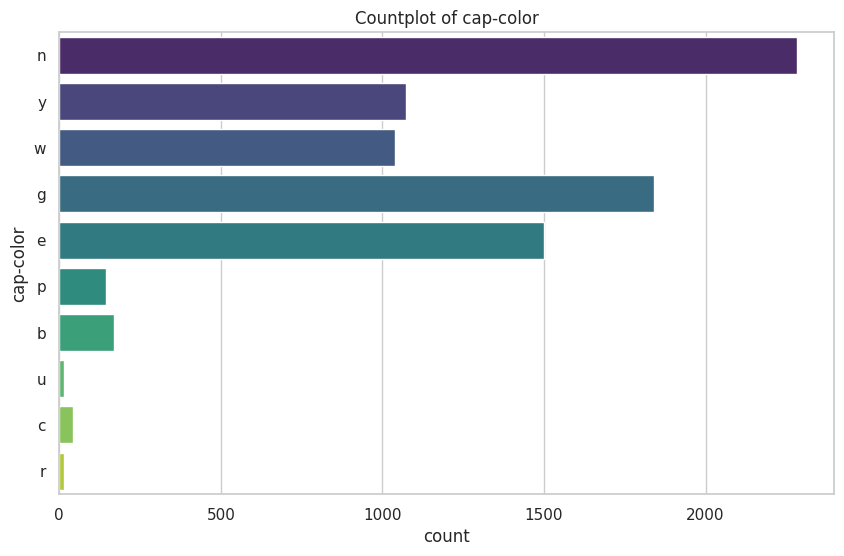


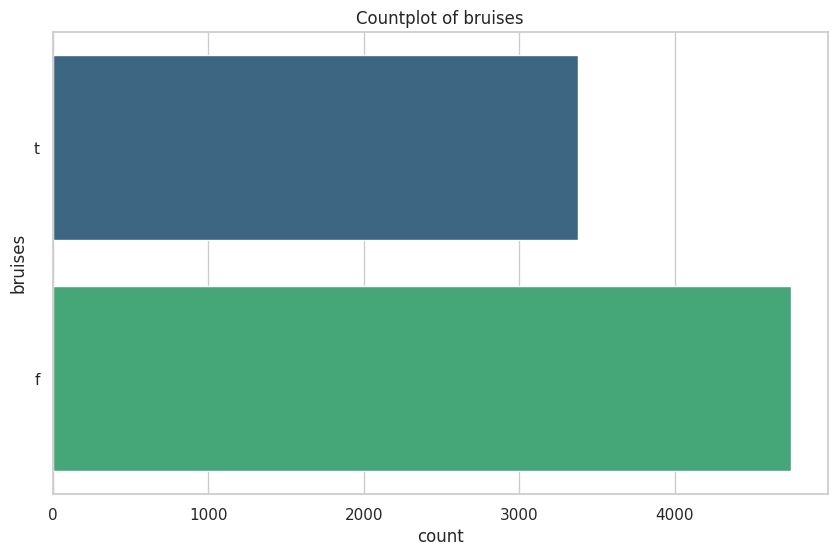
There are roughly equal numbers of edible and poisonous mushrooms in the dataset. The dataset is balanced.

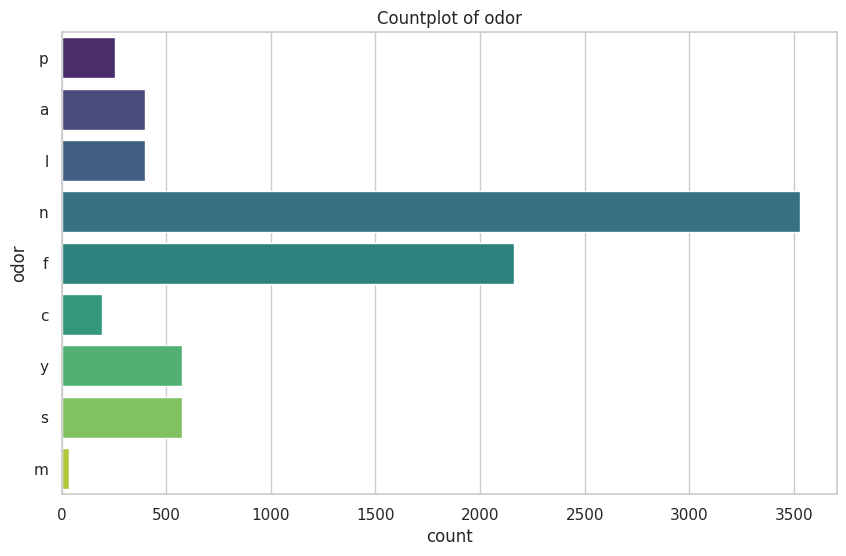
# Create countplots for each column in the x dataframe  
for column in x.columns:  
 plt.figure(figsize=(10, 6))  
 sns.countplot(y=column, data=x, palette='viridis')  
 plt.title(f'Countplot of {column}')  
 plt.show()

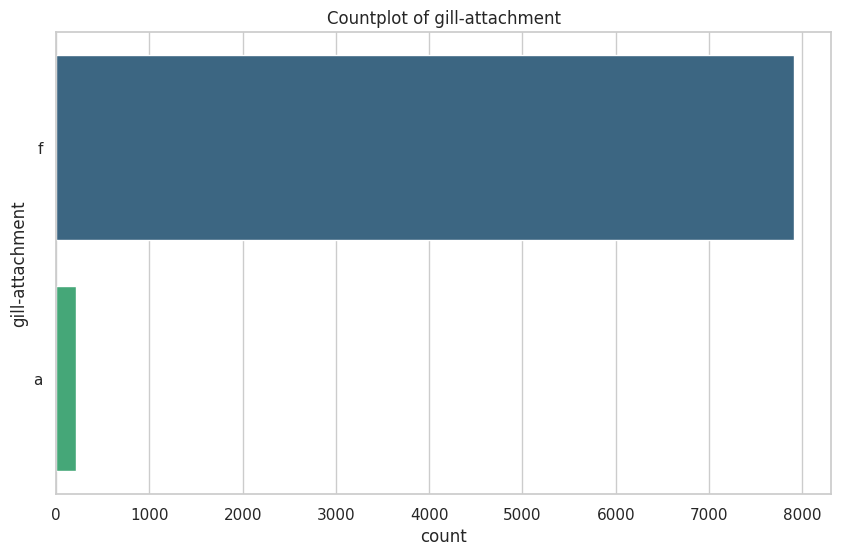


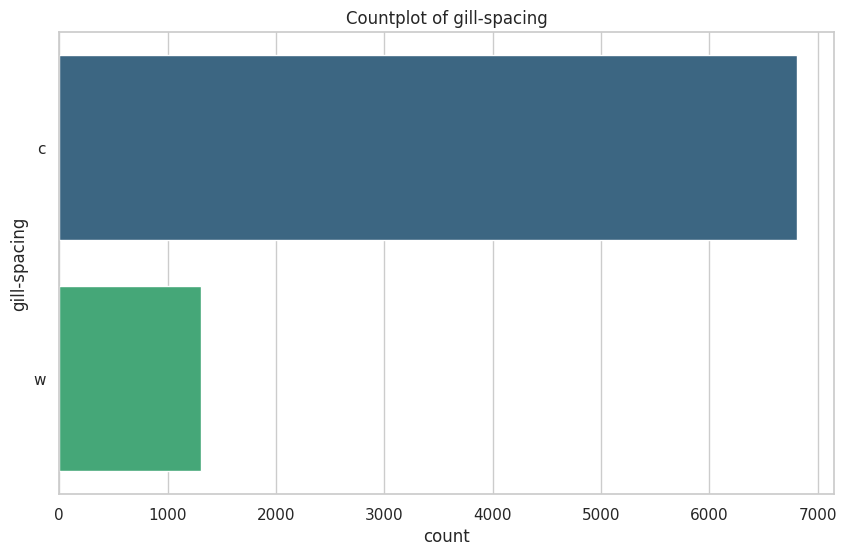


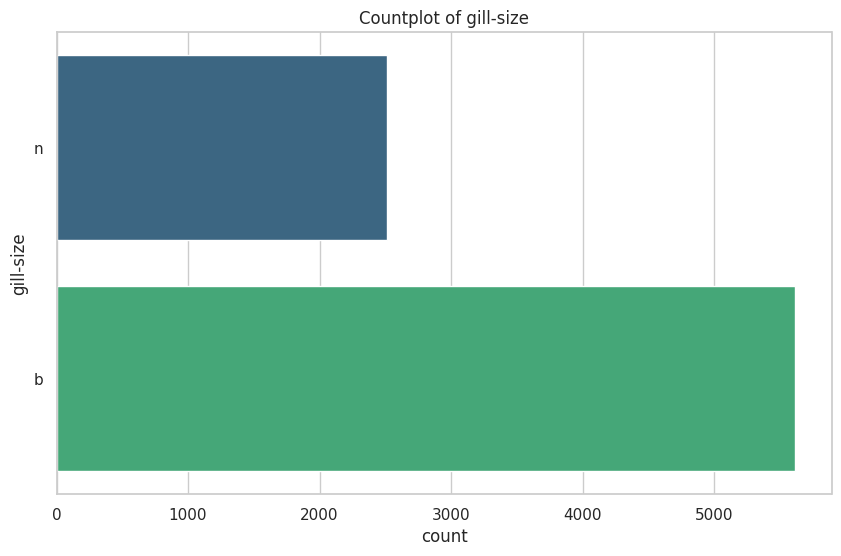


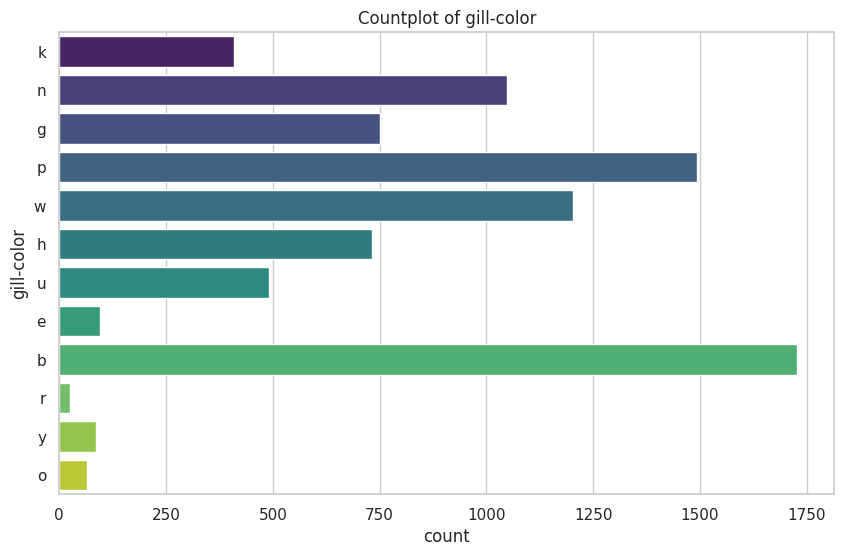


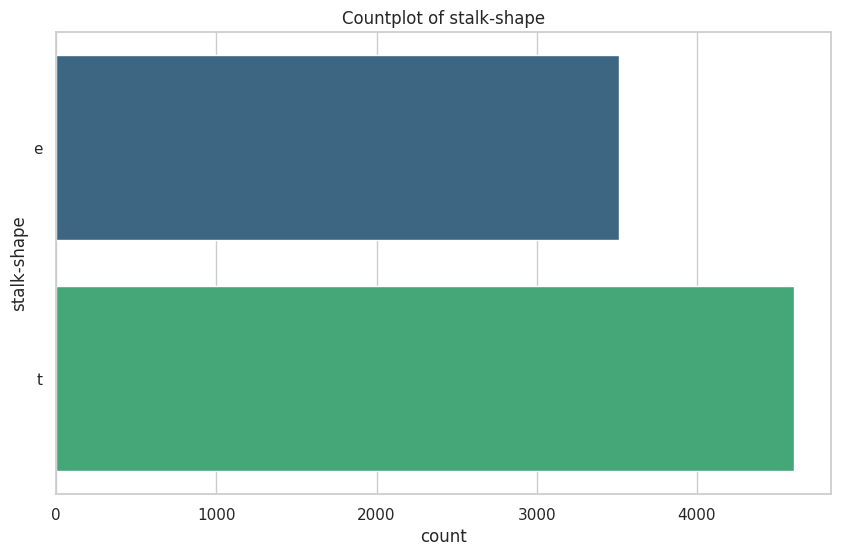


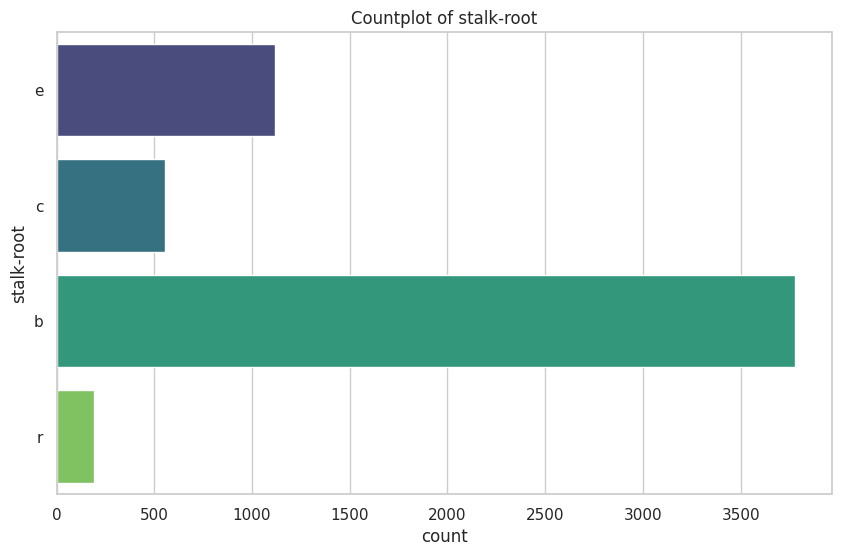


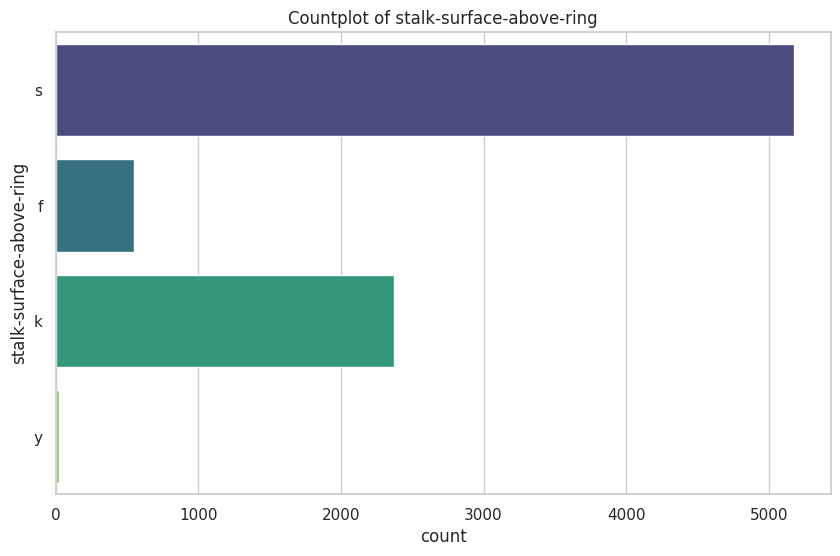


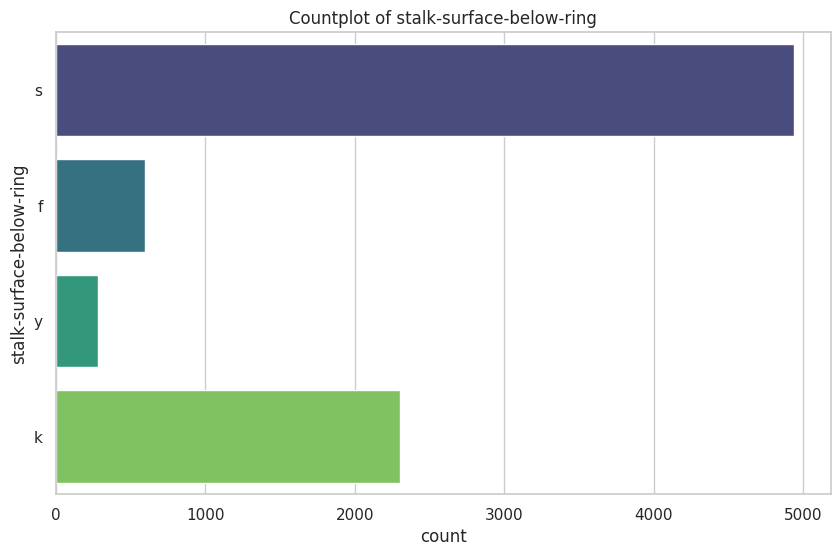


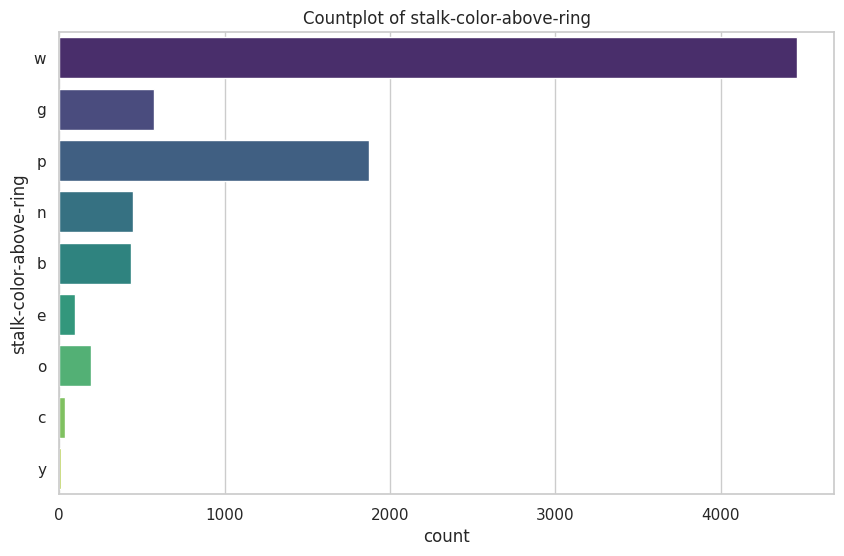


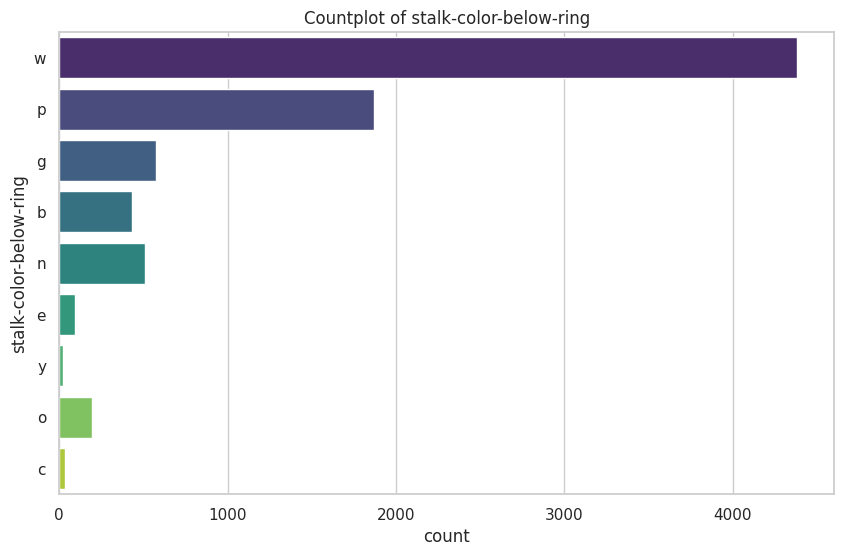


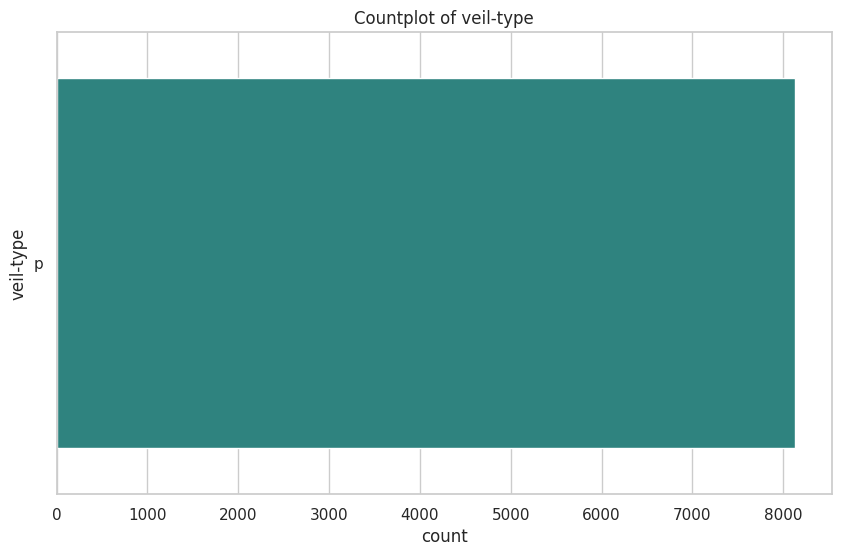


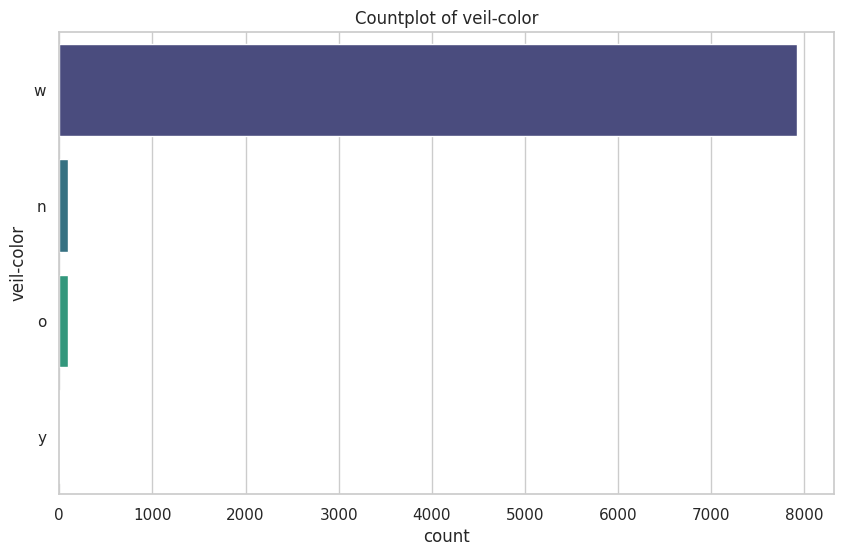


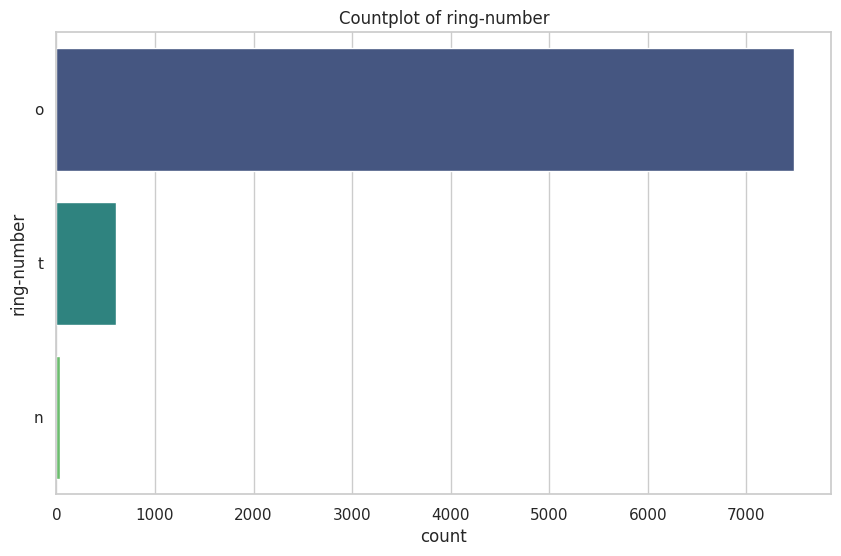


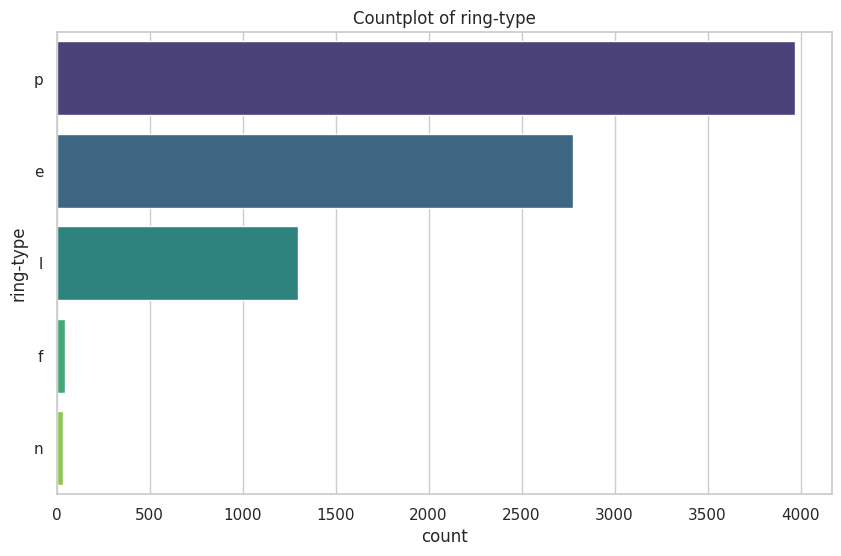


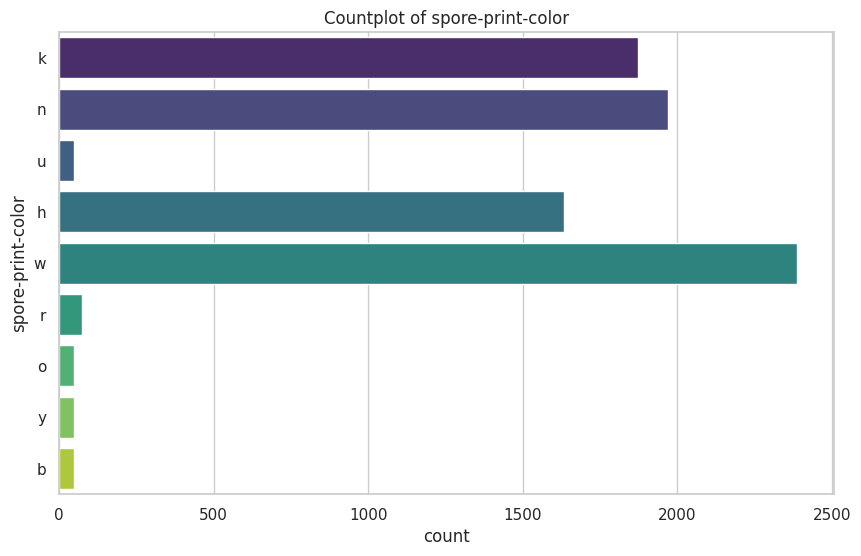


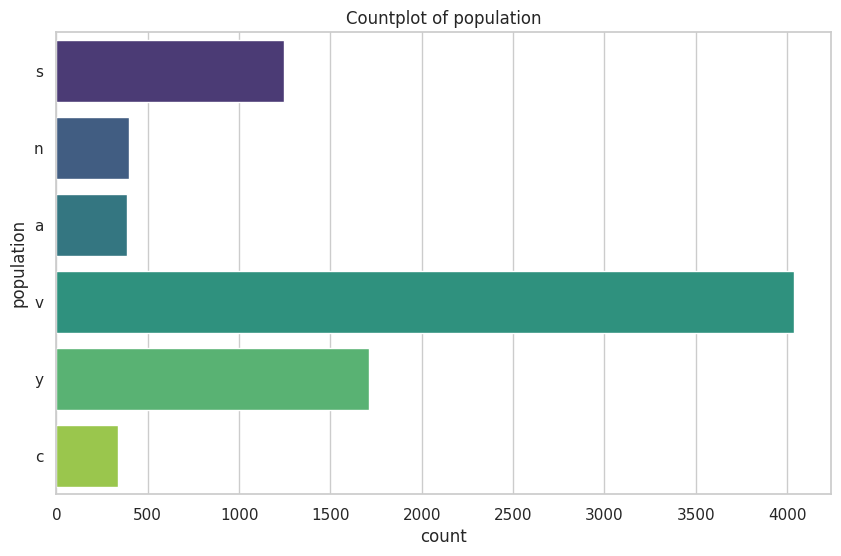


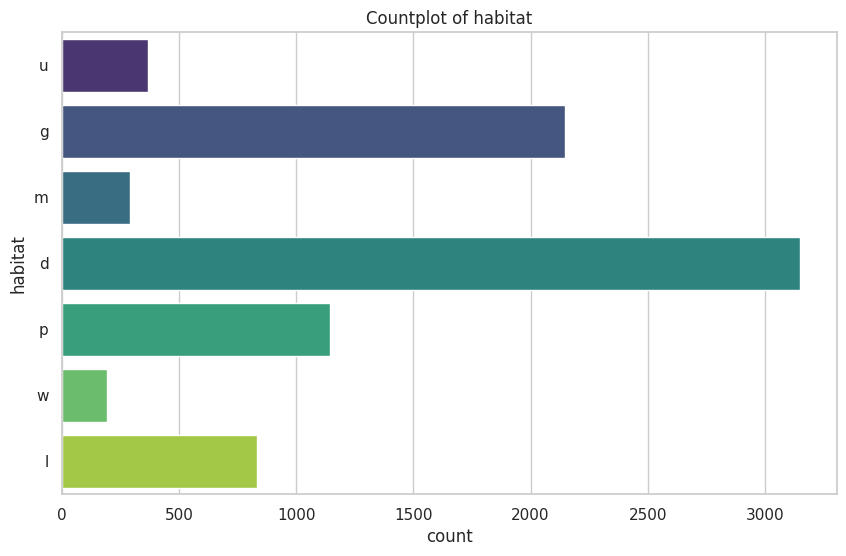












All of these values are categorical, so we will need to encode them before we can use them in a machine learning model. More specifically, all of these values are nominal, so we will need to use one-hot encoding. Ring number could be considered ordinal, but we will treat it as nominal for now.

# Section 3 Feature Selection and Justification

## 3.1 Select features and justify selection

# 1) Make a copy of x and name it x\_uncleaned  
x\_uncleaned = x.copy()  
  
# 2) Drop the column stalk-root and name it x\_dropcolumn  
x\_dropcolumn = x.drop(columns=['stalk-root'])

The uncleaned dataset has not been modified in any way. The dropcolumn dataset has had the stalk-root column removed as there were numerous (~30%) missing values in this column.

# Encode categorical features as numbers  
x\_encoded = x.apply(LabelEncoder().fit\_transform)  
  
# Perform chi-squared test  
chi2\_scores, p\_values = chi2(x\_encoded, y.apply(LabelEncoder().fit\_transform))  
  
# Create a DataFrame to display the results  
chi2\_results = pd.DataFrame({'Feature': x.columns, 'Chi2 Score': chi2\_scores, 'p-value': p\_values})  
  
# Display the results  
print(chi2\_results.sort\_values(by='p-value'))

Feature Chi2 Score p-value  
7 gill-size 1636.606833 0.000000e+00  
8 gill-color 5957.764469 0.000000e+00  
18 ring-type 1950.610146 0.000000e+00  
3 bruises 1194.277352 1.069037e-261  
6 gill-spacing 826.795274 8.061293e-182  
21 habitat 751.309489 2.082916e-165  
10 stalk-root 425.896927 1.270374e-94  
19 spore-print-color 379.132729 1.922119e-84  
20 population 311.766736 9.002366e-70  
11 stalk-surface-above-ring 222.982400 2.022392e-50  
1 cap-surface 214.068544 1.779362e-48  
12 stalk-surface-below-ring 206.648180 7.398869e-47  
13 stalk-color-above-ring 119.792216 7.024638e-28  
14 stalk-color-below-ring 109.789410 1.089740e-25  
4 odor 75.910163 2.968682e-18  
9 stalk-shape 36.594105 1.454709e-09  
17 ring-number 25.646335 4.100735e-07  
0 cap-shape 17.508364 2.860465e-05  
2 cap-color 11.511382 6.917135e-04  
16 veil-color 5.126826 2.355878e-02  
5 gill-attachment 3.505447 6.116734e-02  
15 veil-type NaN NaN

# Initialize the model  
model = LogisticRegression()  
  
# Initialize RFE with the model and the number of features to select  
rfe = RFE(model, n\_features\_to\_select=10)  
  
# Fit RFE  
rfe = rfe.fit(x\_encoded, y.apply(LabelEncoder().fit\_transform))  
  
# Get the ranking of the features  
ranking = rfe.ranking\_  
  
# Create a DataFrame to display the feature rankings  
feature\_ranking = pd.DataFrame({'Feature': x.columns, 'Ranking': ranking})  
  
# Display the features and their rankings  
print(feature\_ranking.sort\_values(by='Ranking'))

Feature Ranking  
3 bruises 1  
7 gill-size 1  
6 gill-spacing 1  
5 gill-attachment 1  
10 stalk-root 1  
11 stalk-surface-above-ring 1  
9 stalk-shape 1  
17 ring-number 1  
18 ring-type 1  
20 population 1  
16 veil-color 2  
4 odor 3  
1 cap-surface 4  
12 stalk-surface-below-ring 5  
8 gill-color 6  
13 stalk-color-above-ring 7  
14 stalk-color-below-ring 8  
0 cap-shape 9  
19 spore-print-color 10  
21 habitat 11  
2 cap-color 12  
15 veil-type 13

# Get the top features selected by RFE  
rfe\_top10\_features = feature\_ranking[feature\_ranking['Ranking'] <= 6]['Feature'].tolist()  
  
# Get the top features selected by chi-squared test  
chi2\_top10\_features = chi2\_results.nsmallest(14, 'p-value')['Feature'].tolist()  
  
# Find the common features in both lists  
common\_top10\_features = list(set(rfe\_top10\_features) & set(chi2\_top10\_features))  
  
print("Top 10 features selected by both RFE and chi-squared test:")  
print(common\_top10\_features)

Top 10 features selected by both RFE and chi-squared test:  
['cap-surface', 'bruises', 'gill-color', 'gill-size', 'gill-spacing', 'population', 'stalk-root', 'stalk-surface-above-ring', 'ring-type', 'stalk-surface-below-ring']

# 3) Make a copy of x with only the top 10 features and name it x\_top10  
x\_top10 = x[common\_top10\_features]  
x\_top10.head()

|  | cap-surface | bruises | gill-color | gill-size | gill-spacing | population | stalk-root | stalk-surface-above-ring | ring-type | stalk-surface-below-ring |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | s | t | k | n | c | s | e | s | p | s |
| 1 | s | t | k | b | c | n | c | s | p | s |
| 2 | s | t | n | b | c | n | c | s | p | s |
| 3 | y | t | n | n | c | s | e | s | p | s |
| 4 | s | f | k | b | w | a | e | s | e | s |

# Get the top features selected by RFE  
rfe\_top5\_features = feature\_ranking[feature\_ranking['Ranking'] <= 1]['Feature'].tolist()  
  
# Get the top features selected by chi-squared test  
chi2\_top5\_features = chi2\_results.nsmallest(7, 'p-value')['Feature'].tolist()  
  
# Find the common features in both lists  
common\_top5\_features = list(set(rfe\_top5\_features) & set(chi2\_top5\_features))  
  
print("Top 5 features selected by both RFE and chi-squared test:")  
print(common\_top5\_features)

Top 5 features selected by both RFE and chi-squared test:  
['bruises', 'gill-size', 'gill-spacing', 'stalk-root', 'ring-type']

# 4) Make a copy of x with only the top 5 features and name it x\_top5  
x\_top5 = x[common\_top5\_features]  
x\_top5.head()

|  | bruises | gill-size | gill-spacing | stalk-root | ring-type |
| --- | --- | --- | --- | --- | --- |
| 0 | t | n | c | e | p |
| 1 | t | b | c | c | p |
| 2 | t | b | c | c | p |
| 3 | t | n | c | e | p |
| 4 | f | b | w | e | e |

# One-hot encode the datasets  
x\_uncleaned\_encoded = pd.get\_dummies(x\_uncleaned)  
x\_dropcolumn\_encoded = pd.get\_dummies(x\_dropcolumn)  
x\_top10\_encoded = pd.get\_dummies(x\_top10)  
x\_top5\_encoded = pd.get\_dummies(x\_top5)  
  
# Display the first few rows of each encoded dataset  
print(x\_uncleaned\_encoded.head())  
print(x\_dropcolumn\_encoded.head())  
print(x\_top10\_encoded.head())  
print(x\_top5\_encoded.head())

cap-shape\_b cap-shape\_c cap-shape\_f cap-shape\_k cap-shape\_s \  
0 False False False False False   
1 False False False False False   
2 True False False False False   
3 False False False False False   
4 False False False False False   
  
 cap-shape\_x cap-surface\_f cap-surface\_g cap-surface\_s cap-surface\_y \  
0 True False False True False   
1 True False False True False   
2 False False False True False   
3 True False False False True   
4 True False False True False   
  
 ... population\_s population\_v population\_y habitat\_d habitat\_g \  
0 ... True False False False False   
1 ... False False False False True   
2 ... False False False False False   
3 ... True False False False False   
4 ... False False False False True   
  
 habitat\_l habitat\_m habitat\_p habitat\_u habitat\_w   
0 False False False True False   
1 False False False False False   
2 False True False False False   
3 False False False True False   
4 False False False False False   
  
[5 rows x 116 columns]  
 cap-shape\_b cap-shape\_c cap-shape\_f cap-shape\_k cap-shape\_s \  
0 False False False False False   
1 False False False False False   
2 True False False False False   
3 False False False False False   
4 False False False False False   
  
 cap-shape\_x cap-surface\_f cap-surface\_g cap-surface\_s cap-surface\_y \  
0 True False False True False   
1 True False False True False   
2 False False False True False   
3 True False False False True   
4 True False False True False   
  
 ... population\_s population\_v population\_y habitat\_d habitat\_g \  
0 ... True False False False False   
1 ... False False False False True   
2 ... False False False False False   
3 ... True False False False False   
4 ... False False False False True   
  
 habitat\_l habitat\_m habitat\_p habitat\_u habitat\_w   
0 False False False True False   
1 False False False False False   
2 False True False False False   
3 False False False True False   
4 False False False False False   
  
[5 rows x 112 columns]  
 cap-surface\_f cap-surface\_g cap-surface\_s cap-surface\_y bruises\_f \  
0 False False True False False   
1 False False True False False   
2 False False True False False   
3 False False False True False   
4 False False True False True   
  
 bruises\_t gill-color\_b gill-color\_e gill-color\_g gill-color\_h ... \  
0 True False False False False ...   
1 True False False False False ...   
2 True False False False False ...   
3 True False False False False ...   
4 False False False False False ...   
  
 stalk-surface-above-ring\_y ring-type\_e ring-type\_f ring-type\_l \  
0 False False False False   
1 False False False False   
2 False False False False   
3 False False False False   
4 False True False False   
  
 ring-type\_n ring-type\_p stalk-surface-below-ring\_f \  
0 False True False   
1 False True False   
2 False True False   
3 False True False   
4 False False False   
  
 stalk-surface-below-ring\_k stalk-surface-below-ring\_s \  
0 False True   
1 False True   
2 False True   
3 False True   
4 False True   
  
 stalk-surface-below-ring\_y   
0 False   
1 False   
2 False   
3 False   
4 False   
  
[5 rows x 45 columns]  
 bruises\_f bruises\_t gill-size\_b gill-size\_n gill-spacing\_c \  
0 False True False True True   
1 False True True False True   
2 False True True False True   
3 False True False True True   
4 True False True False False   
  
 gill-spacing\_w stalk-root\_b stalk-root\_c stalk-root\_e stalk-root\_r \  
0 False False False True False   
1 False False True False False   
2 False False True False False   
3 False False False True False   
4 True False False True False   
  
 ring-type\_e ring-type\_f ring-type\_l ring-type\_n ring-type\_p   
0 False False False False True   
1 False False False False True   
2 False False False False True   
3 False False False False True   
4 True False False False False

notes: one hot encoding was used over label encoding to avoid any ordinal assumptions. FRom what I am able to glean from the data, none of the featers are ordinal.

# Section 4 Train and Evaluate Models

## 4.1 Train Models

# Prepare one-hot encoded datasets  
datasets\_encoded = {  
 'Uncleaned': x\_uncleaned\_encoded,  
 'Drop Column': x\_dropcolumn\_encoded,  
 'Top 10 Features': x\_top10\_encoded,  
 'Top 5 Features': x\_top5\_encoded  
}  
  
# Function to train and evaluate models  
def evaluate\_models(x\_train, y\_train, x\_test, y\_test):  
 models = {  
 'Logistic Regression': LogisticRegression(class\_weight={0: 1, 1: 2}),  
 'Decision Tree': DecisionTreeClassifier(class\_weight={0: 1, 1: 2}, min\_samples\_leaf=2),  
 'SVM': SVC(kernel='linear', class\_weight={0: 1, 1: 2}, C=0.5)  
 }  
   
 results = {}  
   
 for model\_name, model in models.items():  
 model.fit(x\_train, y\_train)  
 predictions = model.predict(x\_test)  
   
 accuracy = accuracy\_score(y\_test, predictions)  
 precision = precision\_score(y\_test, predictions)  
 recall = recall\_score(y\_test, predictions)  
 f1 = f1\_score(y\_test, predictions)  
 confusion = confusion\_matrix(y\_test, predictions)  
   
 results[model\_name] = {  
 'Accuracy': accuracy,  
 'Precision': precision,  
 'Recall': recall,  
 'F1 Score': f1,  
 'Confusion Matrix': confusion  
 }  
   
 return results  
  
# Encode target variable  
y\_encoded = y.apply(LabelEncoder().fit\_transform)  
  
# Split datasets and evaluate models  
all\_results = {}  
  
for dataset\_name, dataset in datasets\_encoded.items():  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(dataset, y\_encoded, test\_size=0.2, random\_state=dont\_panic)  
 results = evaluate\_models(x\_train, y\_train, x\_test, y\_test)  
 all\_results[dataset\_name] = results  
  
# Display results  
for dataset\_name, results in all\_results.items():  
 print(f"Results for {dataset\_name} dataset:")  
 for model\_name, metrics in results.items():  
 print(f" {model\_name}:")  
 for metric\_name, value in metrics.items():  
 if metric\_name == 'Confusion Matrix':  
 print(f" {metric\_name}:\n{value}")  
 else:  
 print(f" {metric\_name}: {value:.4f}")  
 print()

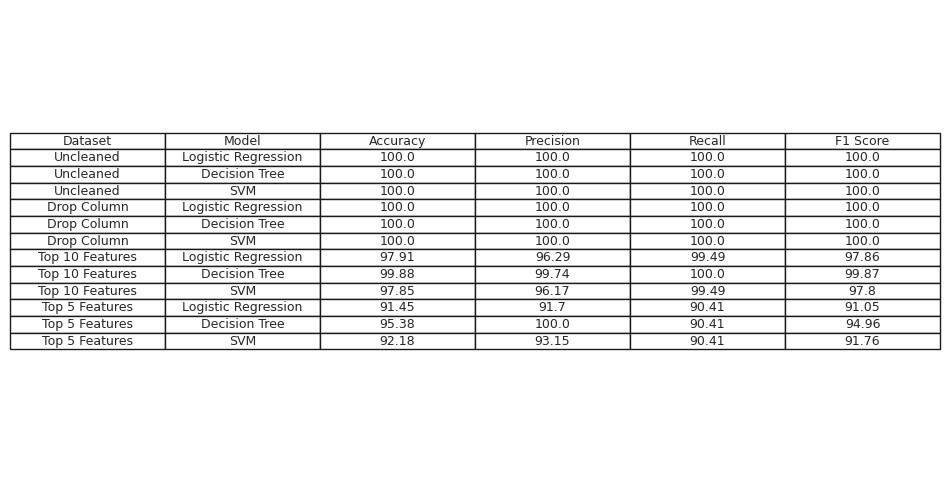
Results for Uncleaned dataset:  
 Logistic Regression:  
 Accuracy: 1.0000  
 Precision: 1.0000  
 Recall: 1.0000  
 F1 Score: 1.0000  
 Confusion Matrix:  
[[843 0]  
 [ 0 782]]  
 Decision Tree:  
 Accuracy: 1.0000  
 Precision: 1.0000  
 Recall: 1.0000  
 F1 Score: 1.0000  
 Confusion Matrix:  
[[843 0]  
 [ 0 782]]  
 SVM:  
 Accuracy: 1.0000  
 Precision: 1.0000  
 Recall: 1.0000  
 F1 Score: 1.0000  
 Confusion Matrix:  
[[843 0]  
 [ 0 782]]  
  
Results for Drop Column dataset:  
 Logistic Regression:  
 Accuracy: 1.0000  
 Precision: 1.0000  
 Recall: 1.0000  
 F1 Score: 1.0000  
 Confusion Matrix:  
[[843 0]  
 [ 0 782]]  
 Decision Tree:  
 Accuracy: 1.0000  
 Precision: 1.0000  
 Recall: 1.0000  
 F1 Score: 1.0000  
 Confusion Matrix:  
[[843 0]  
 [ 0 782]]  
 SVM:  
 Accuracy: 1.0000  
 Precision: 1.0000  
 Recall: 1.0000  
 F1 Score: 1.0000  
 Confusion Matrix:  
[[843 0]  
 [ 0 782]]  
  
Results for Top 10 Features dataset:  
 Logistic Regression:  
 Accuracy: 0.9791  
 Precision: 0.9629  
 Recall: 0.9949  
 F1 Score: 0.9786  
 Confusion Matrix:  
[[813 30]  
 [ 4 778]]  
 Decision Tree:  
 Accuracy: 0.9988  
 Precision: 0.9974  
 Recall: 1.0000  
 F1 Score: 0.9987  
 Confusion Matrix:  
[[841 2]  
 [ 0 782]]  
 SVM:  
 Accuracy: 0.9785  
 Precision: 0.9617  
 Recall: 0.9949  
 F1 Score: 0.9780  
 Confusion Matrix:  
[[812 31]  
 [ 4 778]]  
  
Results for Top 5 Features dataset:  
 Logistic Regression:  
 Accuracy: 0.9145  
 Precision: 0.9170  
 Recall: 0.9041  
 F1 Score: 0.9105  
 Confusion Matrix:  
[[779 64]  
 [ 75 707]]  
 Decision Tree:  
 Accuracy: 0.9538  
 Precision: 1.0000  
 Recall: 0.9041  
 F1 Score: 0.9496  
 Confusion Matrix:  
[[843 0]  
 [ 75 707]]  
 SVM:  
 Accuracy: 0.9218  
 Precision: 0.9315  
 Recall: 0.9041  
 F1 Score: 0.9176  
 Confusion Matrix:  
[[791 52]  
 [ 75 707]]

# Convert the results dictionary to a DataFrame  
results\_df = pd.DataFrame.from\_dict({(i, j): all\_results[i][j]   
 for i in all\_results.keys()   
 for j in all\_results[i].keys()},  
 orient='index')  
  
# Display the DataFrame  
print(results\_df)

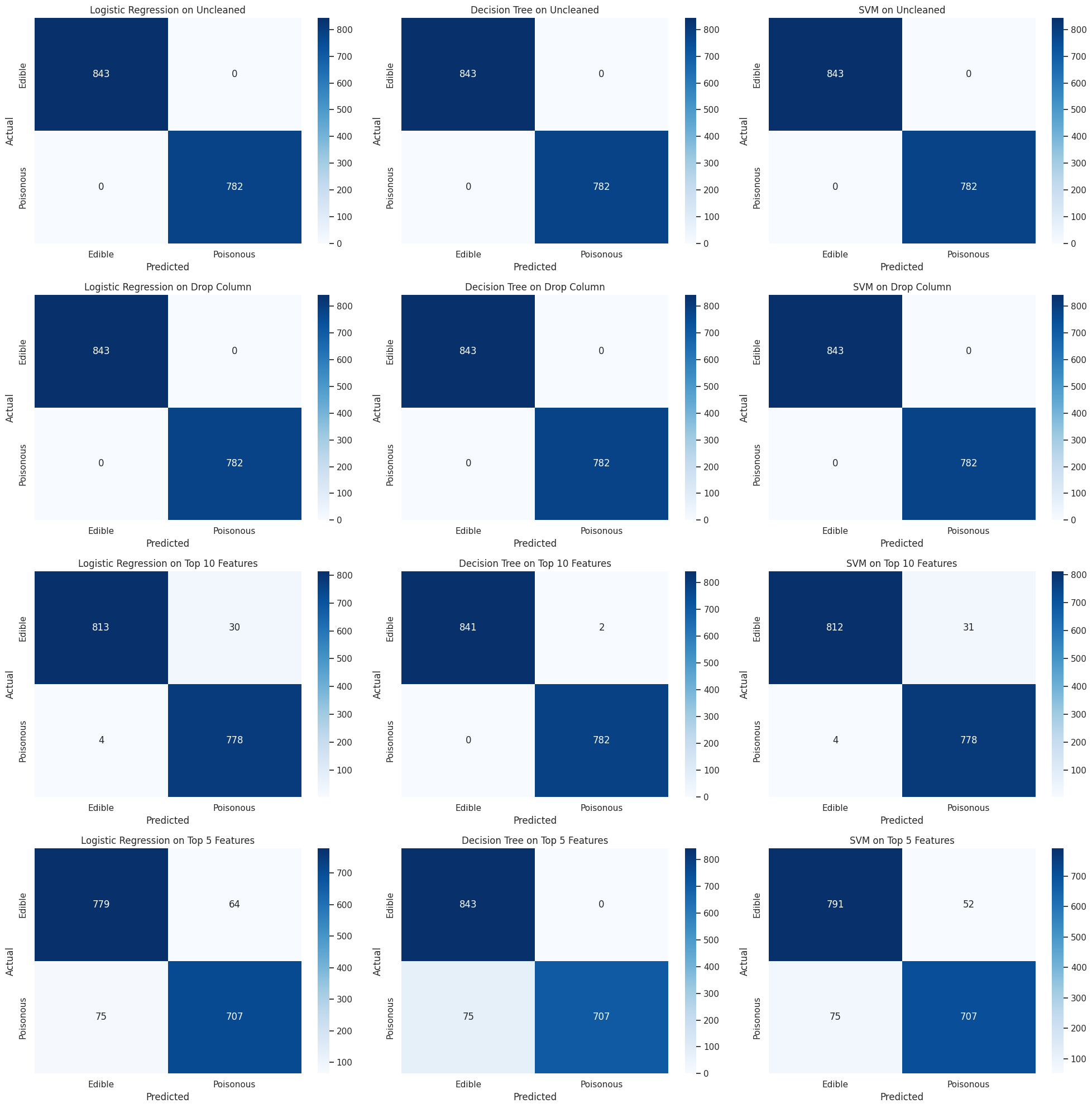
Accuracy Precision Recall F1 Score \  
Uncleaned Logistic Regression 1.000000 1.000000 1.000000 1.000000   
 Decision Tree 1.000000 1.000000 1.000000 1.000000   
 SVM 1.000000 1.000000 1.000000 1.000000   
Drop Column Logistic Regression 1.000000 1.000000 1.000000 1.000000   
 Decision Tree 1.000000 1.000000 1.000000 1.000000   
 SVM 1.000000 1.000000 1.000000 1.000000   
Top 10 Features Logistic Regression 0.979077 0.962871 0.994885 0.978616   
 Decision Tree 0.998769 0.997449 1.000000 0.998723   
 SVM 0.978462 0.961681 0.994885 0.978001   
Top 5 Features Logistic Regression 0.914462 0.916991 0.904092 0.910496   
 Decision Tree 0.953846 1.000000 0.904092 0.949631   
 SVM 0.921846 0.931489 0.904092 0.917586   
  
 Confusion Matrix   
Uncleaned Logistic Regression [[843, 0], [0, 782]]   
 Decision Tree [[843, 0], [0, 782]]   
 SVM [[843, 0], [0, 782]]   
Drop Column Logistic Regression [[843, 0], [0, 782]]   
 Decision Tree [[843, 0], [0, 782]]   
 SVM [[843, 0], [0, 782]]   
Top 10 Features Logistic Regression [[813, 30], [4, 778]]   
 Decision Tree [[841, 2], [0, 782]]   
 SVM [[812, 31], [4, 778]]   
Top 5 Features Logistic Regression [[779, 64], [75, 707]]   
 Decision Tree [[843, 0], [75, 707]]   
 SVM [[791, 52], [75, 707]]

## 4.2 Evaluate Models

# Create a new DataFrame with the desired columns, excluding 'Confusion Matrix'  
table\_df = results\_df.reset\_index()  
table\_df.columns = ['Dataset', 'Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'Confusion Matrix']  
table\_df = table\_df.drop(columns=['Confusion Matrix'])  
  
# Convert the metrics to percentages  
table\_df[['Accuracy', 'Precision', 'Recall', 'F1 Score']] = table\_df[['Accuracy', 'Precision', 'Recall', 'F1 Score']] \* 100  
  
# Round the metrics to two decimal places  
table\_df[['Accuracy', 'Precision', 'Recall', 'F1 Score']] = table\_df[['Accuracy', 'Precision', 'Recall', 'F1 Score']].round(2)  
  
# Plot the table  
fig, ax = plt.subplots(figsize=(12, 6))  
ax.axis('tight')  
ax.axis('off')  
table = ax.table(cellText=table\_df.values, colLabels=table\_df.columns, cellLoc='center', loc='center')  
  
# Display the table  
plt.show()  
  
# Save the table to an image  
fig.savefig('results\_table.png')



# Function to plot confusion matrix  
def plot\_confusion\_matrix(ax, conf\_matrix, model\_name, dataset\_name):  
 sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Edible', 'Poisonous'], yticklabels=['Edible', 'Poisonous'], ax=ax)  
 ax.set\_title(f'{model\_name} on {dataset\_name}')  
 ax.set\_xlabel('Predicted')  
 ax.set\_ylabel('Actual')  
  
# Create a figure with 4 rows and 3 columns of subplots  
fig, axes = plt.subplots(4, 3, figsize=(20, 20))  
axes = axes.flatten()  
  
# Plot confusion matrices for each model and dataset  
index = 0  
for dataset\_name, results in all\_results.items():  
 for model\_name, metrics in results.items():  
 plot\_confusion\_matrix(axes[index], metrics['Confusion Matrix'], model\_name, dataset\_name)  
 index += 1  
  
# Adjust layout  
plt.tight\_layout()  
plt.show()



# Function to train and plot decision tree  
def plot\_decision\_tree(x\_train, y\_train, dataset\_name):  
 # Initialize the model  
 tree\_model = DecisionTreeClassifier()  
   
 # Fit the model  
 tree\_model.fit(x\_train, y\_train)  
   
 # Plot the tree  
 plt.figure(figsize=(20, 10))  
 plot\_tree(tree\_model, filled=True, feature\_names=x\_train.columns, class\_names=['Edible', 'Poisonous'], rounded=True)  
 plt.title(f"Decision Tree for {dataset\_name} Dataset")  
 plt.show()  
  
# Plot decision tree for each dataset  
plot\_decision\_tree(x\_uncleaned\_encoded, y\_encoded, 'Uncleaned')  
plot\_decision\_tree(x\_dropcolumn\_encoded, y\_encoded, 'Drop Column')  
plot\_decision\_tree(x\_top10\_encoded, y\_encoded, 'Top 10 Features')  
plot\_decision\_tree(x\_top5\_encoded, y\_encoded, 'Top 5 Features')

