The University of Faisalabad

Lab Manual

Subject: Machine Learning

Course Code AI-414

By

Mohammad Mukedas 2023-BS-AI-008

Supervised by Sir Saeed

Degree name: BSAI-4A

DEPARTMENT OF COMPUTATIONAL SCIENCE

Table Of Content

1: YouTube Spam Comment Classification

Description:

This project involves collecting YouTube comments and analyzing them using a regression algorithm. The comments are turned into numerical features (using) techniques like TF-IDF or Bag of Words), and then a regression model is trained to predict whether a comment is spam (1) or not spam (0).

Tools & Technologies:

- Python
- Pandas, NumPy (for data handling)
- Scikit-learn (for regression models)
- Text processing tools (like CountVectorizer or TfidfVectorizer)

How It Works:

1.Data Collection: Gather labeled YouTube comments (spam or not spam).

2.Text Preprocessing: Clean the text (remove punctuation, lowercase, etc.

3.Feature Extraction: Convert comments into numerical vectors.

4.Model Training: Use a regression model (like Logistic Regression) to learn patterns.

5.Prediction: Test the model on new comments to classify them.

Expected Output:

The system should label new YouTube comments as spam (1) or not spam (0) with good accuracy.

Program:

Import Libraries ¶

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import joblib
```

Step 2: Load the Dataset

```
df = pd.read_csv("C:/Users/AS/Downloads/Youtube-Spam-Dataset.csv")
```

What is Basic EDA in Machine Learning?

EDA (Exploratory Data Analysis) is the **first step in any ML project**, where you **understand and explore your dataset** before building a model. It helps you find patterns, spot outliers, and understand data quality.

Step 3: Basic EDA

```
print("Dataset Shape:", df.shape)
```

```
Dataset Shape: (1956, 6)
```

```
print("\nMissing Values:\n", df.isnull().sum())
```

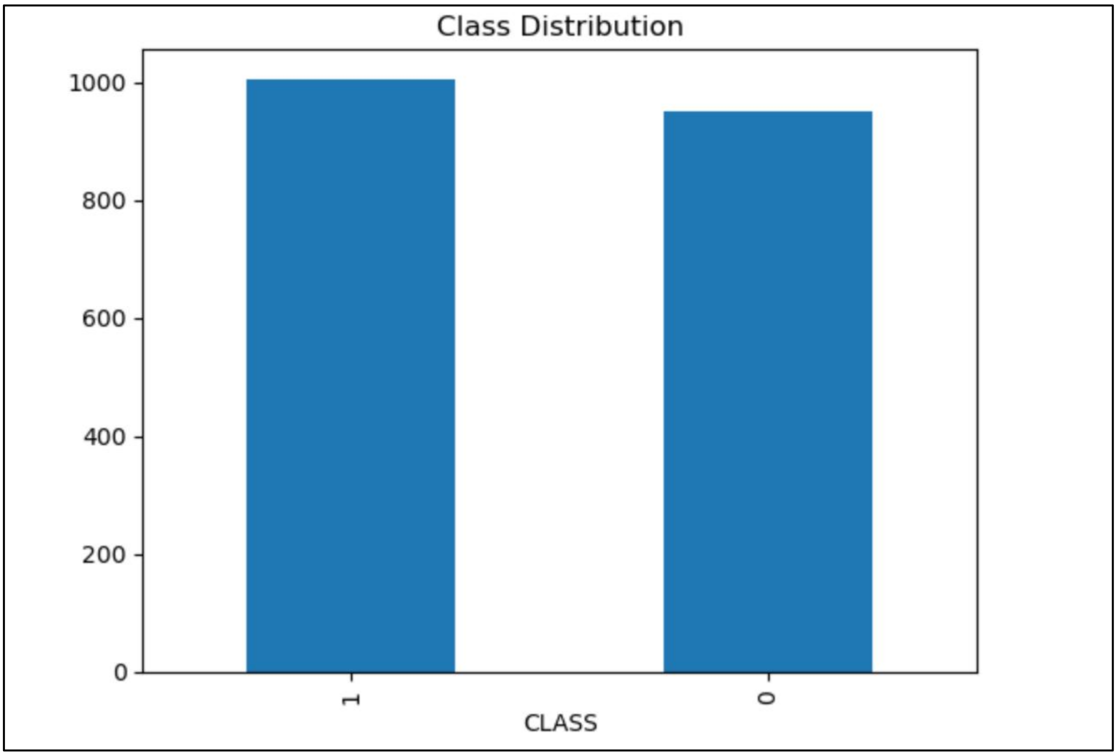
```

Missing Values:
 COMMENT_ID      0
AUTHOR          0
DATE          245
CONTENT         0
VIDEO_NAME      0
CLASS           0
dtype: int64
```

```
print("\nClass Distribution:\n", df['CLASS'].value_counts())
```

```
Class Distribution:
 CLASS
1    1005
0     951
Name: count, dtype: int64
```

```
df['CLASS'].value_counts().plot(kind='bar', title='Class Distribution')
```

```
<Axes: title={'center': 'Class Distribution'}, xlabel='CLASS'>
```




```
plt.show()
```

Step 4: Data Preprocessing

```
df = df[['CONTENT', 'CLASS']].dropna()
```

Step 5: Split the Dataset

```
X = df['CONTENT']
y = df['CLASS']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 6: Feature Extraction using TF-IDF

```
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
```

```
X_train_tfidf = tfidf.fit_transform(X_train)
```

```
X_test_tfidf = tfidf.transform(X_test)
```

Step 7: Model Training

```
model = LogisticRegression()
```

```
model.fit(X_train_tfidf, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Step 8: Model Evaluation

```
y_pred = model.predict(X_test_tfidf)
```

```
print("\nAccuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9489795918367347
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.94       176
           1       0.97      0.94      0.95       216

    accuracy                           0.95       392
   macro avg       0.95      0.95      0.95       392
weighted avg       0.95      0.95      0.95       392
```

```
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
 [[170   6]
 [ 14 202]]
```

Step 9: Save the Model and Vectorizer

```
joblib.dump(model, "spam_classifier_model.pkl")
```

```
['spam_classifier_model.pkl']
```

```
joblib.dump(tfidf, "tfidf_vectorizer.pkl")
```

```
['tfidf_vectorizer.pkl']
```

Project 2: Mushroom Classification Using Classifier

Introduction:

In this project, we will examine the data and build different **machine learning models** that will detect if the mushroom is **edible or poisonous** by its specifications like cap shape, cap color, gill color, etc. using different classifiers.

Dataset:

The dataset used in this project is that contains 8124 instances of mushrooms with 23 features like cap-shape, cap-surface, cap-color, bruises, odor, etc.

The **python libraries** and packages we'll use in this project are namely:

- NumPy
- Pandas

- Seaborn
- Matplotlib
- Graphviz
- Scikit-learn

We'll use the specifications like cap shape, cap color, gill color, etc. to classify the mushrooms into edible and poisonous.

Proram:

Importing Libraraies

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_curve, auc, roc_curve
```

```
df = pd.read_csv(r"C:\Users\HP\Documents\mushrooms.csv")
```

```
df.head()
```

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

5 rows × 23 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
```

```
df.describe()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	...	8124	8124	8124	8124	8124	8124	8124	8124
unique	2	6	4	10	2	9	2	2	2	12	...	4	9	9	1	4	3	5	9
top	e	x	y	n	f	n	f	c	b	b	...	s	w	w	p	w	o	p	w
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	...	4936	4464	4384	8124	7924	7488	3968	2388

4 rows × 23 columns

```
print("Dataset shape:", df.shape)
```

```
Dataset shape: (8124, 23)
```

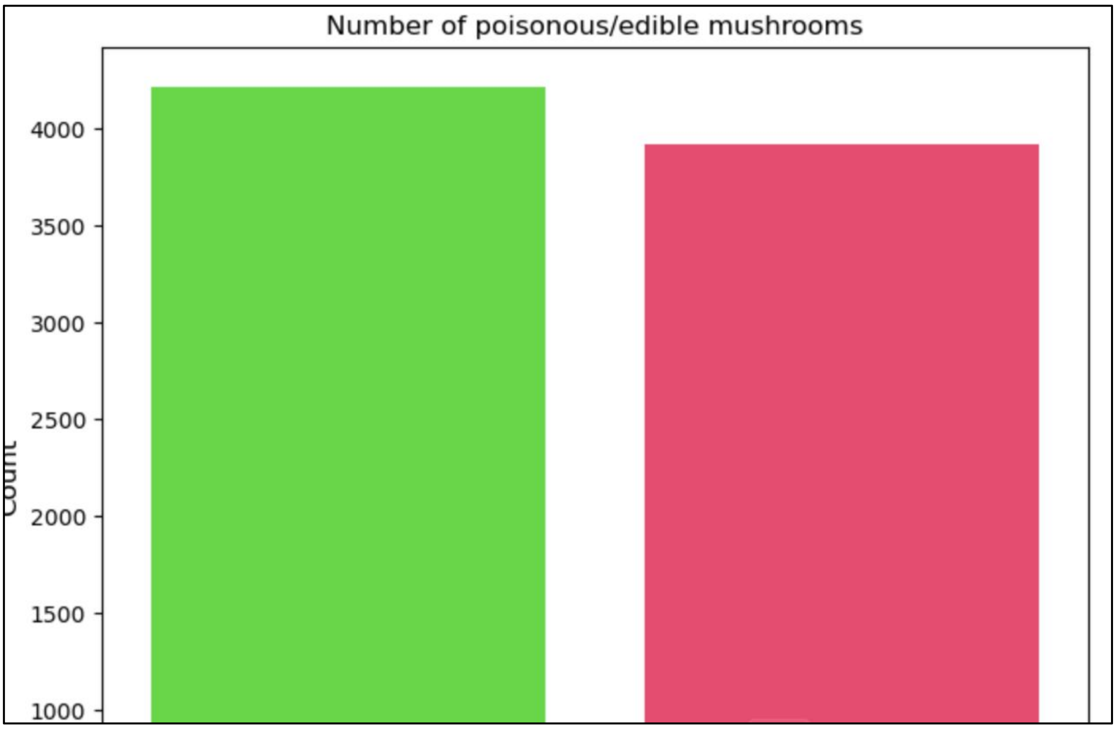
```
df['class'].unique()
```

```
array(['p', 'e'], dtype=object)
```

```
df['class'].value_counts()
```

```
class
e    4208
p    3916
Name: count, dtype: int64
```

```
count = df['class'].value_counts()
plt.figure(figsize=(8,7))
# Updated syntax for sns.barplot with x and y as named parameters
sns.barplot(x=count.index, y=count.values, alpha=0.8, palette="prism")
# Alternative syntax for newer seaborn versions:
# sns.barplot(data=count.reset_index(), x='index', y='class', alpha=0.8, palette="prism")
plt.ylabel('Count', fontsize=12)
plt.xlabel('Class', fontsize=12)
plt.title('Number of poisonous/edible mushrooms')
#plt.savefig("mushrooms1.png", format='png', dpi=500)
plt.show()
```



```python
df = df.astype('category')
df.dtypes
```

```
class                       category
cap-shape                   category
cap-surface                 category
cap-color                   category
bruises                     category
odor                        category
gill-attachment             category
gill-spacing                category
gill-size                   category
gill-color                  category
stalk-shape                 category
stalk-root                  category
stalk-surface-above-ring    category
stalk-surface-below-ring    category
stalk-color-above-ring      category
stalk-color-below-ring      category
veil-type                   category
veil-color                  category
ring-number                 category
ring-type                   category
spore-print-color           category
population                  category
```

```python
labelencoder=LabelEncoder()
for column in df.columns:
    df[column] = labelencoder.fit_transform(df[column])
```

```python
df.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color
0	1	5	2	4	1	6	1	0	1	4	...	2	7	7	0	2	1	4	2
1	0	5	2	9	1	0	1	0	0	4	...	2	7	7	0	2	1	4	3
2	0	0	2	8	1	3	1	0	0	5	...	2	7	7	0	2	1	4	3
3	1	5	3	8	1	6	1	0	1	5	...	2	7	7	0	2	1	4	2
4	0	5	2	3	0	5	1	1	0	4	...	2	7	7	0	2	1	0	3

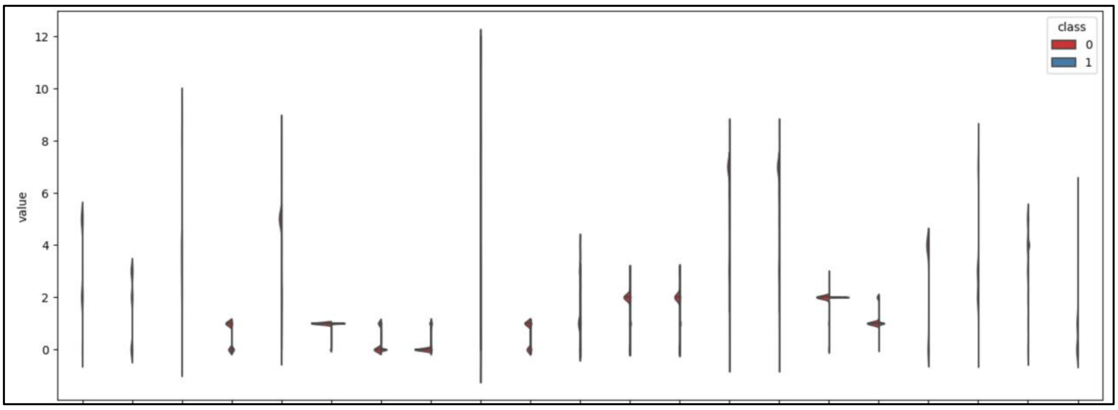
5 rows × 23 columns

```python
df['veil-type']
```

```
0       0
1       0
2       0
3       0
4       0
       ..
8119    0
8120    0
8121    0
8122    0
8123    0
Name: veil-type, Length: 8124, dtype: int32
```

```python
df = df.drop(["veil-type"],axis=1)
```

```python
df_div = pd.melt(df, "class", var_name="Characteristics")
fig, ax = plt.subplots(figsize=(16,6))
p = sns.violinplot(ax = ax, x="Characteristics", y="value", hue="class", split = True, data=df_div, inner = 'quartile', palette = 'Set1')
df_no_class = df.drop(["class"],axis = 1)
p.set_xticklabels(rotation = 90, labels = list(df_no_class.columns));
#plt.savefig("violinplot.png", format='png', dpi=500, bbox_inches='tight')
```

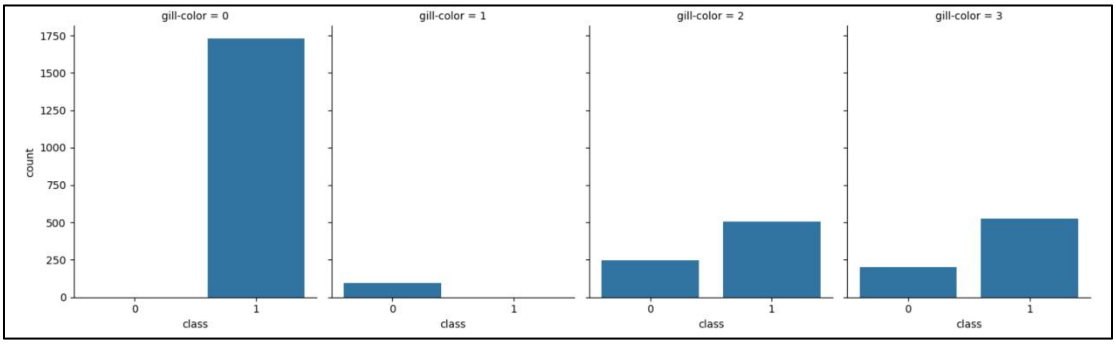



```python
plt.figure(figsize=(14,12))
sns.heatmap(df.corr(),linewidths=.1,cmap="Purples", annot=True, annot_kws={"size": 7})
plt.yticks(rotation=0);
#plt.savefig("corr.png", format='png', dpi=400, bbox_inches='tight')
```


```python
df[['class', 'gill-color']].groupby(['gill-color'], as_index=False).mean().sort_values(by='class', ascending=False)
```

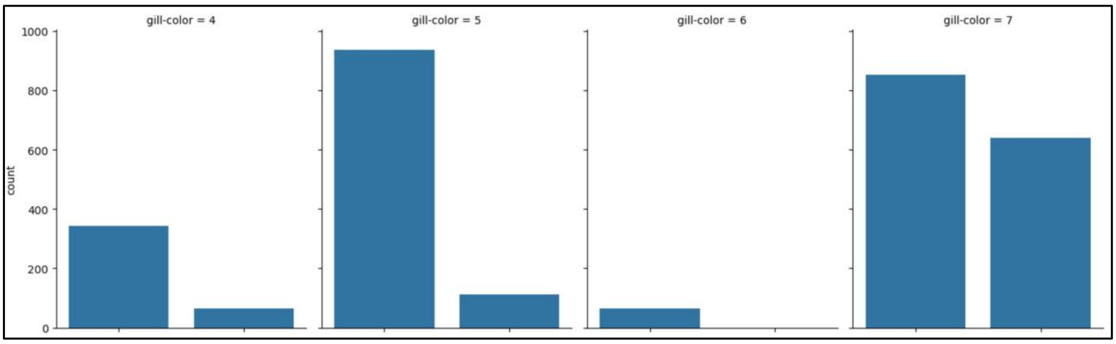
	gill-color	class
0	0	1.000000
8	8	1.000000
3	3	0.721311
2	2	0.670213
7	7	0.428954
11	11	0.255814
10	10	0.204659
4	4	0.156863
5	5	0.106870
9	9	0.097561
1	1	0.000000
6	6	0.000000

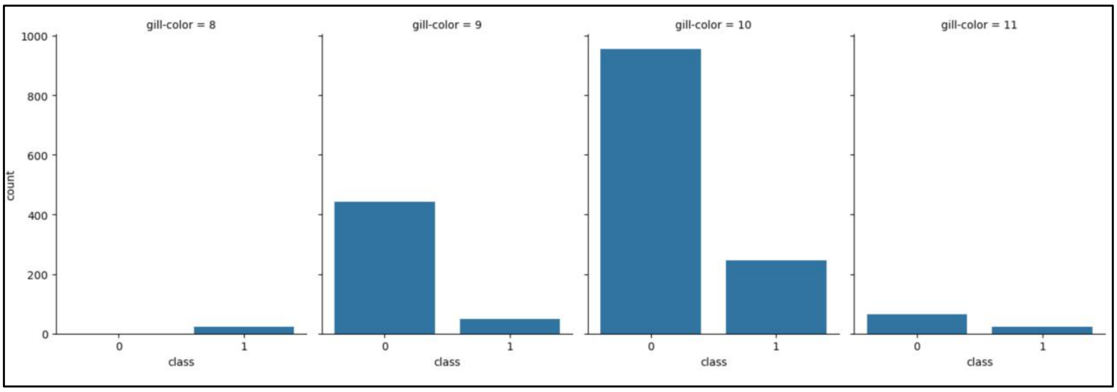
```python
new_var = df[['class', 'gill-color']]
new_var = new_var[new_var['gill-color']<=3.5]
# Changed factorplot to catplot as factorplot is deprecated in newer seaborn versions
# Fixed the parameter order in catplot function to avoid duplicate 'data' argument
sns.catplot(x='class', col='gill-color', data=new_var, kind='count', height=4.5, aspect=.8, col_wrap=4);
#plt.savefig("gillcolor1.png", format='png', dpi=500, bbox_inches='tight')
```




```python
new_var = df[['class', 'gill-color']]
new_var = new_var[new_var['gill-color'] > 3.5]

# Using catplot instead of factorplot (which is deprecated)
sns.catplot(x='class', col='gill-color', data=new_var, kind='count',
            height=4.5, aspect=.8, col_wrap=4)

#plt.savefig("gillcolor2.png", format='png', dpi=400, bbox_inches='tight')
```

```
<seaborn.axisgrid.FacetGrid at 0x1d9e3942690>
```





```
X = df.drop(['class'], axis=1)
y = df["class"]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.1)
```

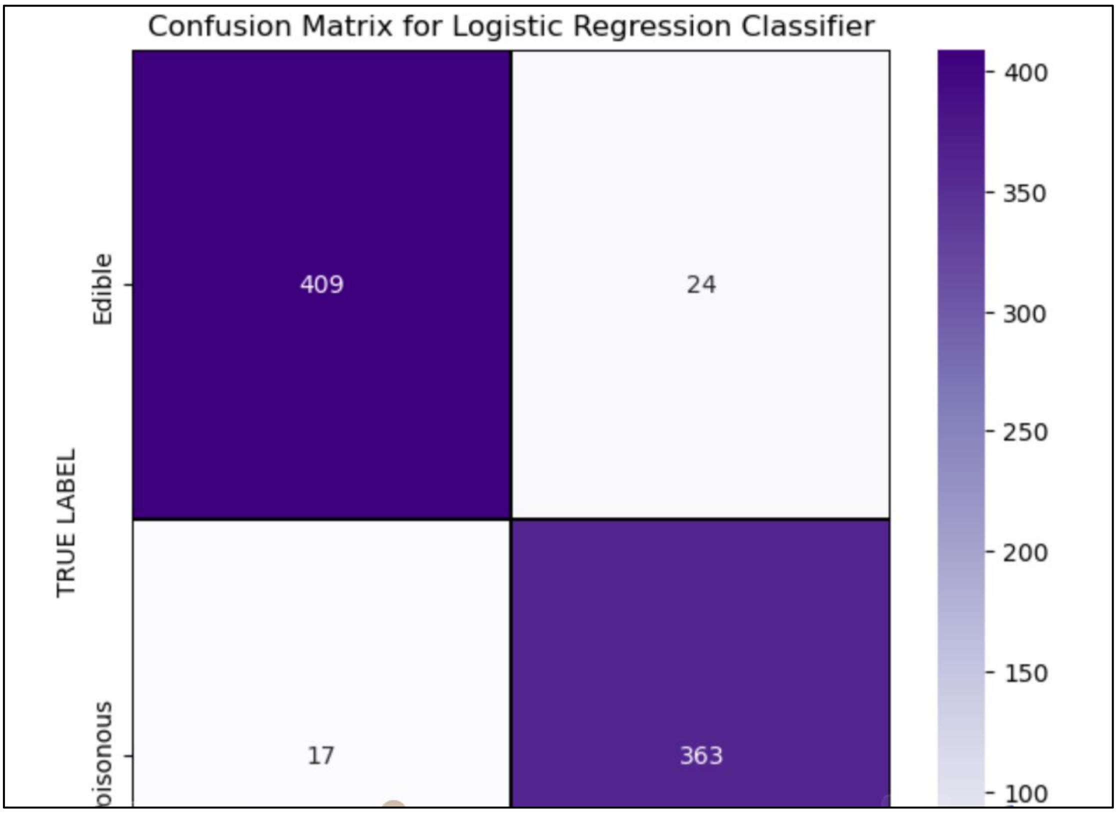
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver="lbfgs", max_iter=500)
lr.fit(X_train, y_train)
print("Test Accuracy: {}%".format(round(lr.score(X_test, y_test)*100,2)))
```

```
Test Accuracy: 94.96%
```

```
y_pred_lr = lr.predict(X_test)
print("Logistic Regression Classifier report: \n\n", classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Classifier report:

              precision    recall  f1-score   support

           0       0.96      0.94      0.95       433
           1       0.94      0.96      0.95       380

    accuracy                           0.95       813
   macro avg       0.95      0.95      0.95       813
weighted avg       0.95      0.95      0.95       813
```

```
cm = confusion_matrix(y_test, y_pred_lr)
x_axis_labels = ["Edible", "Poisonous"]
y_axis_labels = ["Edible", "Poisonous"]
f, ax = plt.subplots(figsize =(7,7))
sns.heatmap(cm, annot = True, linewidths=0.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Purples", xticklabels=x_axis_labels,
plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for Logistic Regression Classifier')
#plt.savefig("lrcm.png", format='png', dpi=500, bbox_inches='tight')
plt.show()
```


Confusion Matrix for Logistic Regression Classifier
TRUE LABEL
Edible
oisonous
409
24
17
363
400
350
300
250
200
150
100