UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

SECB3203-01(PROGRAMMING FOR BIOINFORMATIC)

Semester 01 2025/2026

Section 01

Final Report

Faculty of Computing

Dataset:

https://www.kaggle.com/datasets/uciml/mushroom-classification

STUDENT NAME	MATRIC NUMBER
NURUL NATALIA BINTI ROSNIZAM	A23CS0165
ADLINA NARISYA BINTI ISMAIL	A23CS0033
NAZATUL NADHIRAH BINTI SABTU	A23CS0144

Table of Contents

1.0 Introduction 1

1.1 Problem Background 1

1.2 Problem Statement 2

1.3 Objectives 2

1.4 Scopes 3

2.0 Data Collection and Preprocessing 4

2.1 Importing Dataset 4

2.2 Data Wrangling 10

2.3 Software and Hardware Requirements 12

3.0 Flowchart of Proposed Approach 13

3.1 Exploratory Data Analysis 14

3.1.1 Target Variable Distribution Analysis 14

3.1.2 Correlation Analysis 15

3.1.3 Visualization Correlation 16

3.1.4 Feature Selection based on Correlation 17

3.2 Model Development 21

3.2.1 Data Partitioning (Train-Test Split) 21

3.2.2 Model Training 22

3.2.2.1 Random Forest Classifier 22

3.2.2.2 Decision Tree Classifier 22

3.2.2.3 Logistic Regression 22

3.3 Model Evaluation 23

3.3.1 Evaluation Metrices 23

3.3.2 Evaluated Models 23

3.3.2.1 Random Forest 23

3.2.2.2 Decision Tree Classifier 28

3.2.2.3 Logistic Regression 32

4.0 Testing and Validation 39

4.1 Comparison of Tree-Based Models: Random Forest vs. Decision Tree 39

4.2 Comparison with the Linear Model: Tree-Based vs. Logistic Regression 40

5.0 Conclusion 41

1.0 Introduction

Mushrooms are widely consumed as a nutritious food source, but many wild species are poisonous and pose serious health risks when misidentified. The dataset used in this project is derived from *The Audubon Society Field Guide to North American Mushrooms (1981)* and contains descriptions of hypothetical samples representing 23 species of gilled mushrooms belonging to the *Agaricus* and *Lepiota* families.

Each species in the dataset is classified as definitely edible or definitely poisonous. For safety purposes, the "unknown" category has been merged with the "poisonous" class. Importantly, the Field Guide emphasizes that there is no simple universal rule to determine the edibility of a mushroom unlike plants such as Poison Ivy, where simple identification rules exist ("leaflets three, let it be"). It is because edible and poisonous mushrooms often share similar morphological features such as cap color, odor, gill structure, and stalk characteristics, manual identification is difficult for non-experts. This makes mushroom classification a challenging biological problem, highlighting the need for computational approaches to support safe identification and prevent accidental poisoning.

1.1 Problem Background

Mushrooms are eaten worldwide for food, nutrition, and sometimes medicine, but not all mushrooms are safe. Many poisonous mushrooms look very similar to edible ones, which often causes people to misidentify them. This can lead to serious health problems, including poisoning or even death. Because of this, it is important to understand the features that separate safe mushrooms from dangerous ones.

Researchers use data to study these differences. One useful dataset is the Mushroom Classification dataset, which contains information on more than 8,000 mushrooms. Each mushroom is labeled as edible or poisonous and includes details such as cap shape, color, odor, and gill size. These are features normally used to identify mushrooms in real life.

While many wild mushrooms are edible, a large number are toxic, with some being deadly. Distinguishing between edible and poisonous species is challenging and often requires expert knowledge. Moreover, misidentification can lead to severe illness or even death. Therefore, there is a critical need for a reliable and accessible tool to help individuals accurately classify mushrooms and ensure their safety

1.2 Problem Statement

The primary challenge in mushroom classification is the high risk associated with human error in prediction, as manual identification tend to make mistakes due to identical visualization between mushroom species. This problem always occurs in real-life situations which lead to life-threatening scenarios. On the other hand, there is a computational challenge where the classification fully depends on multiple categorical features that often overlap between classes, occurs when samples from different classes share similar characteristics. Thus, this project aims to address both human and computational challenges by developing an automated machine learning model that can classify mushrooms accurately and safely based on their characteristics. Therefore, reducing the risk of poisoning.

1.3 Objectives

1. To prepare the Mushroom Classification dataset by conducting data wrangling, including removing the useless features.
2. To perform exploratory data analysis to identify the most significant features for machine learning training.
3. To develop and train multiple models including Random Forest, Decision Tree, and Logistic Regression.
4. To evaluate and compare the performance of these models based on accuracy, AUC, and f1-score to find the optimal one.

1.4 Scopes

1. **Data Used:**

   This project uses the Mushroom Classification dataset from Kaggle, which contains 8,124 samples of mushrooms with categorical features and a class label indicating whether each mushroom is edible or poisonous.

2. **Techniques to Be Used:**

   The project applies basic exploratory data analysis, descriptive statistics, and simple data visualizations to understand the distribution and patterns in the dataset.

3. **Methodology:**

   The study involves loading and inspecting the dataset, cleaning the data, exploring each feature, visualizing important patterns, and summarizing the findings in a clear report.

4. **Limits of the Research:**

   The analysis is limited to the features provided in the dataset, which may not represent all mushroom species in real life.

5. **Expected Outcomes:**

   The project aims to provide a clear understanding of the class balance between edible and poisonous mushrooms, identify key feature patterns, and present meaningful visualizations that highlight important insights from the data.

2.0 Data Collection and Preprocessing

2.1 Importing Dataset

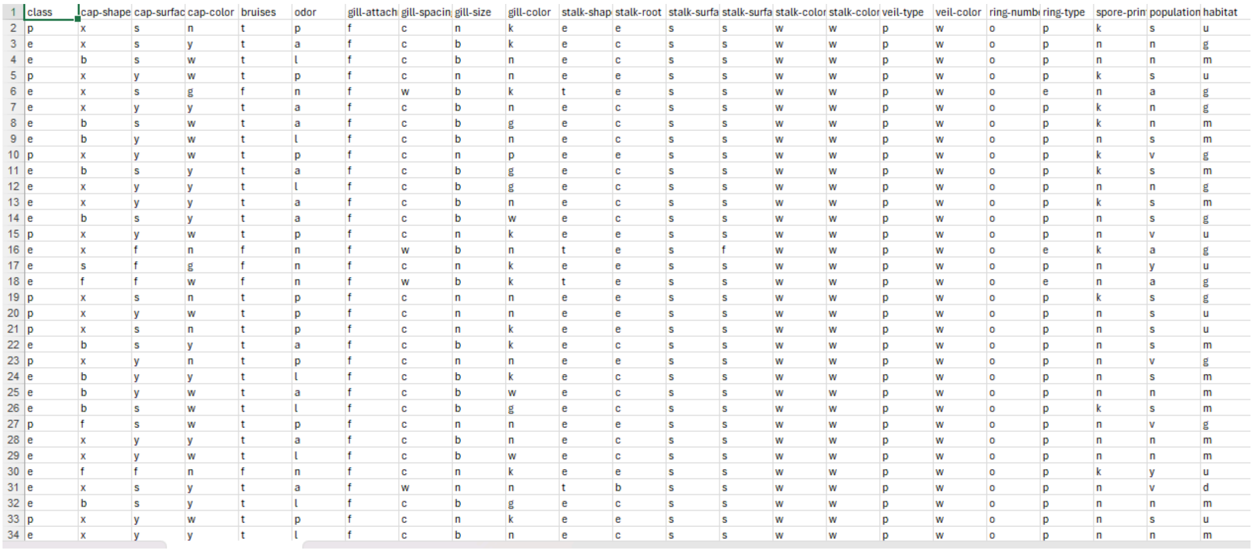
2.1.1 Understanding the Data

In this research, a mushroom classification dataset is utilized to predict whether a mushroom is edible or poisonous based on its physical characteristics. The dataset contains various categorical attributes describing mushroom properties such as class, cap shape, cap color, odor, gill size, stalk shape, habitat, and other morphological features. These attributes are critical in identifying patterns that distinguish edible mushrooms from poisonous ones.

The dataset was obtained from Kaggle and is widely used for classification and data mining research. It consists of 8124 mushroom samples (rows) with 24 features (columns), including one target attribute representing the class of the mushroom. Each feature plays an important role in understanding mushroom toxicity and edibility.

Our dataset from Kaggle: https://www.kaggle.com/code/rocklen/mushroom-classification/input

Figure 2.0: Raw dataset from kaggle



1	class	cap-shape	cap-surfac	cap-color	bruises	odor	gill-attach	gill-spacin	gill-size	gill-color	stalk-shap	stalk-root	stalk-surfa	stalk-surfa	stalk-color	stalk-color	veil-type	veil-color	ring-numb	ring-type	spore-prin	population	habitat
2	p	x	s	n	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	k	s	u
3	e	x	s	y	t	a	f	c	b	k	e	c	s	s	w	w	p	w	o	p	n	n	g
4	e	b	s	w	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n	n	m
5	p	x	y	w	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	k	s	u
6	e	x	s	g	f	n	f	w	b	k	t	e	s	s	w	w	p	w	o	e	n	a	g
7	e	x	y	y	t	a	f	c	b	n	e	c	s	s	w	w	p	w	o	p	k	n	g
8	e	b	s	w	t	a	f	c	b	g	e	c	s	s	w	w	p	w	o	p	k	n	m
9	e	b	y	w	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n	s	m
10	p	x	y	w	t	p	f	c	n	p	e	e	s	s	w	w	p	w	o	p	k	v	g
11	e	b	s	y	t	a	f	c	b	g	e	c	s	s	w	w	p	w	o	p	k	s	m
12	e	x	y	y	t	l	f	c	b	g	e	c	s	s	w	w	p	w	o	p	n	n	g
13	e	x	y	y	t	a	f	c	b	n	e	c	s	s	w	w	p	w	o	p	k	s	m
14	e	b	s	y	t	a	f	c	b	w	e	c	s	s	w	w	p	w	o	p	n	s	g
15	p	x	y	w	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	n	v	u
16	e	x	f	n	f	n	f	w	b	n	t	e	s	f	w	w	p	w	o	e	k	a	g
17	e	s	f	g	f	n	f	c	n	k	e	e	s	s	w	w	p	w	o	p	n	y	u
18	e	f	f	w	f	n	f	w	b	k	t	e	s	s	w	w	p	w	o	e	n	a	g
19	p	x	s	n	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	k	s	g
20	p	x	y	w	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	n	s	u
21	p	x	s	n	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	n	s	u
22	e	b	s	y	t	a	f	c	b	k	e	c	s	s	w	w	p	w	o	p	n	s	m
23	p	x	y	n	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	n	v	g
24	e	b	y	y	t	l	f	c	b	k	e	c	s	s	w	w	p	w	o	p	n	s	m
25	e	b	y	w	t	a	f	c	b	w	e	c	s	s	w	w	p	w	o	p	n	n	m
26	e	b	s	w	t	l	f	c	b	g	e	c	s	s	w	w	p	w	o	p	k	s	m
27	p	f	s	w	t	p	f	c	n	n	e	e	s	s	w	w	p	w	o	p	n	v	g
28	e	x	y	y	t	a	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n	n	m
29	e	x	y	w	t	l	f	c	b	w	e	c	s	s	w	w	p	w	o	p	n	n	m
30	e	f	f	n	f	n	f	c	n	k	e	e	s	s	w	w	p	w	o	p	k	y	u
31	e	x	s	y	t	a	f	w	n	n	t	b	s	s	w	w	p	w	o	p	n	v	d
32	e	b	s	y	t	l	f	c	b	g	e	c	s	s	w	w	p	w	o	p	n	n	m
33	p	x	y	w	t	p	f	c	n	k	e	e	s	s	w	w	p	w	o	p	n	s	u
34	e	x	y	y	t	l	f	c	b	n	e	c	s	s	w	w	p	w	o	p	n	n	m

Figure 2.0 illustrates the mushroom dataset obtained from Kaggle, consisting of 8124 instances and 23 attributes.

Table 2.1 List of selected attributes mushroom dataset

Feature	**Explanation**
Class	Indicates mushroom edibility: **e** – edible, **p** – poisonous.
Cap-shape	Describes the shape of the mushroom cap: **b** – bell, **c** – conical, **x** – convex, **f** – flat, **k** – knobbed, **s** – sunken.
Cap-surface	Describes the surface texture of the cap: **f** – fibrous, **g** – grooved, **y** – scaly, **s** – smooth.
Population	Indicates how the mushroom population appears in nature: **c** – clustered **n** – numerous **s** – scattered **v** – several **y** – solitary

	a - abundant
Habitat	Describes the environment where the mushroom commonly grows: **g** – grasses **l** – leaves **m** – meadows **p** – paths **u** – urban areas **w** – waste areas **d** – woods
Spore-print-color	Indicates the color of the mushroom's spore print: **k** – black **n** – brown **b** – buff **h** – chocolate **r** – green **o** – orange **u** – purple **w** – white **y** – yellow
Veil-color	Describes the color of the veil covering the mushroom when young: **n** – brown

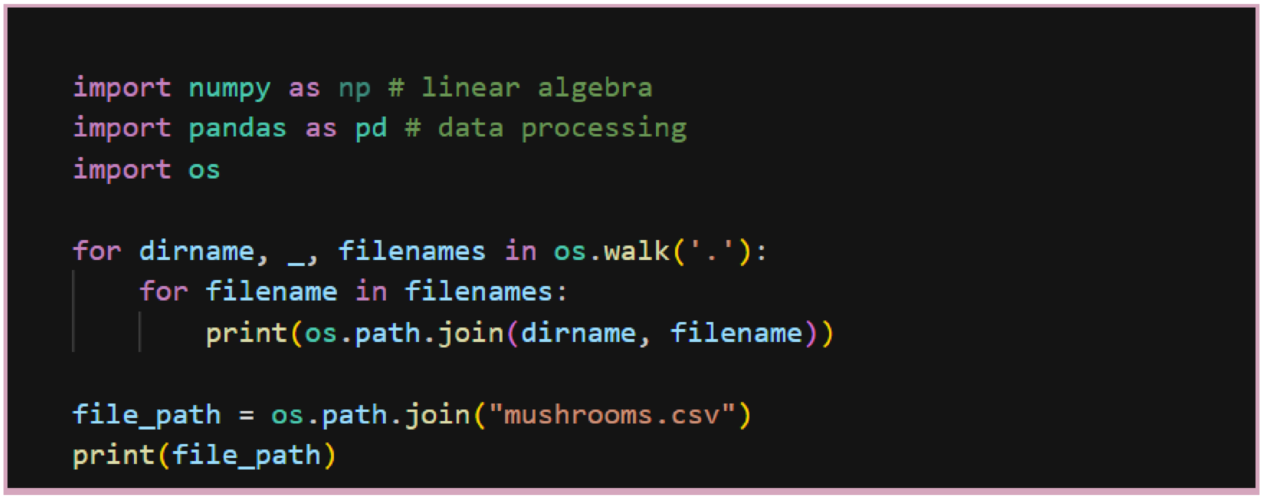
	o – orange **w** – white **y** – yellow
Gill-attachment	Describes how the gills are attached to the stalk: **a** – attached **f** – free
Gill-size	Describes the size of the gills under the mushroom cap: **b** – broad **n** – narrow

In **Table 2.1,** show selected features along with their explanations and name that are relevant for mushroom classification.

2.1.2 Importing and Exporting Data in Python

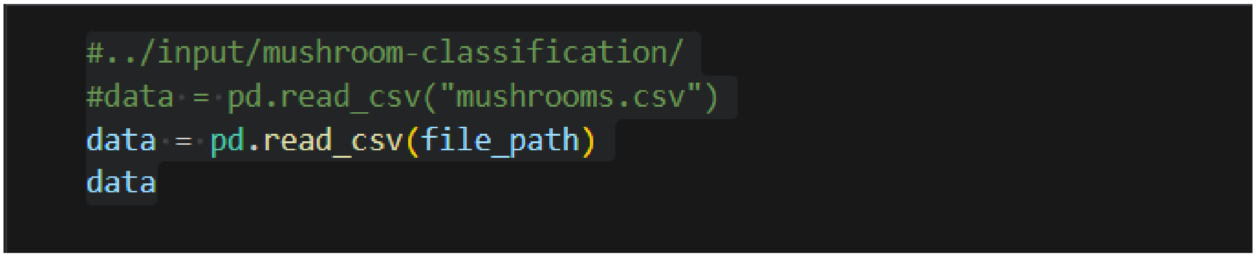
The dataset is provided in CSV format. We import it into Python using the numpy, pandas, os library:

Figure 2.1: Importing dataset using pandas library



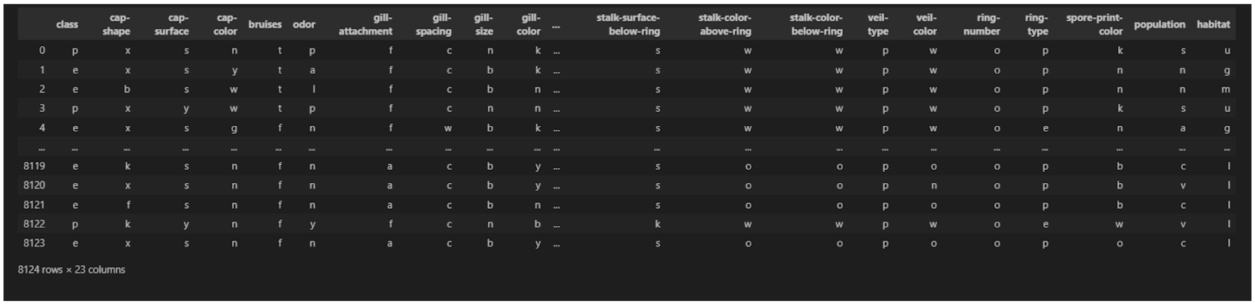
```python
import numpy as np # linear algebra
import pandas as pd # data processing
import os

for dirname, _, filenames in os.walk('.'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

file_path = os.path.join("mushrooms.csv")
print(file_path)
```

Figure 2.2: Read CSV file



```python
#../input/mushroom-classification/
#data = pd.read_csv("mushrooms.csv")
data = pd.read_csv(file_path)
data
```

Figure 2.3: Display the result of raw mushroom dataset



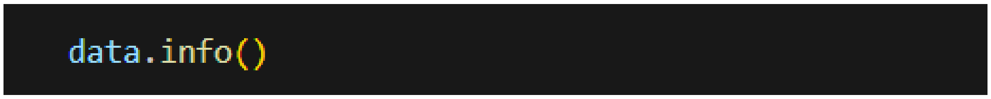
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	b	c	l
8120	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	n	o	p	b	v	l
8121	e	f	s	n	f	n	a	c	b	n	...	s	o	o	p	o	o	p	b	c	l
8122	p	k	y	n	f	y	f	c	n	b	...	k	w	w	p	w	o	e	w	v	l
8123	e	x	s	n	f	n	a	c	b	y	...	s	o	o	p	o	o	p	o	c	l

8124 rows × 23 columns

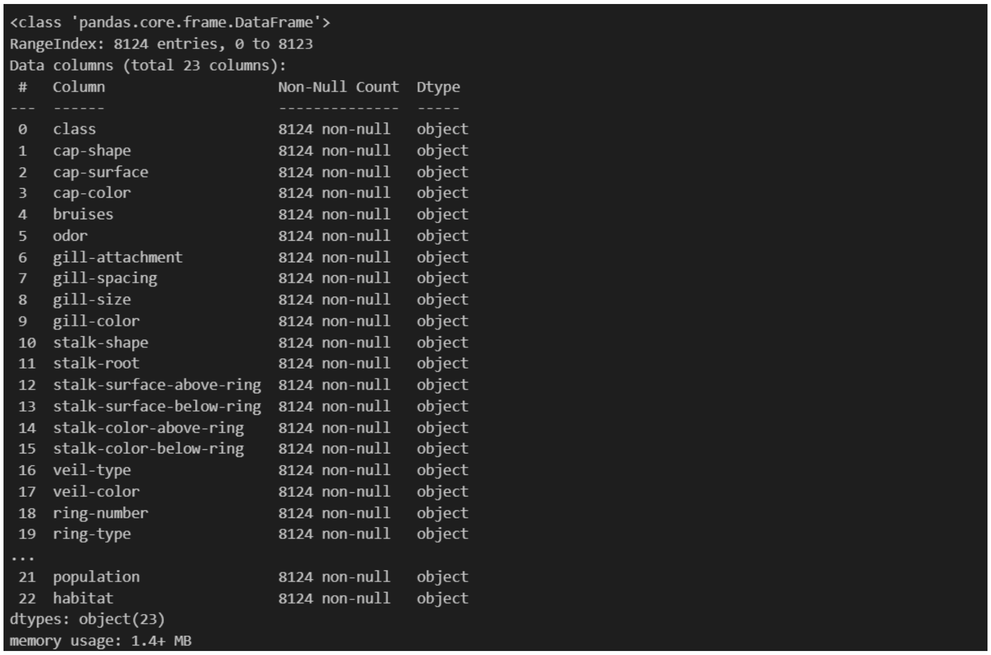
2.1.3 Getting Started Analyzing Data in Python

Initial data analysis began by identifying the total number of rows and columns in the dataset.

Figure 2.4: Displaying the total number of rows and columns in the dataset.



```
data.info()
```

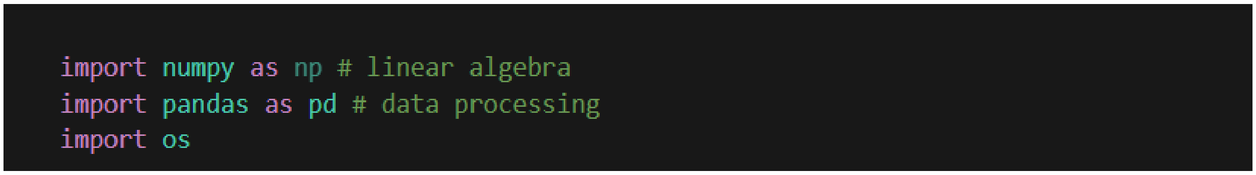


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 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
 19  ring-type                 8124 non-null   object
...
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

2.1.4 Python Packages for Data Science

We import the necessary libraries for data handling and preprocessing:

Figure 2.5: Import necessary libraries



```
import numpy as np # linear algebra
import pandas as pd # data processing
import os
```

1. **Numpy**

- Used to import the mushroom dataset from a CSV file.
- Displayed all columns to verify successful data loading.
- Checked dataset structure, data types, and missing values.
- Supported basic data manipulation and preprocessing tasks

2. **Pandas**

- used to handle numerical operations during data preprocessing.
- Supported representation of missing values using NaN.
- Assisted in preparing data for further analysis.

3. **Os**

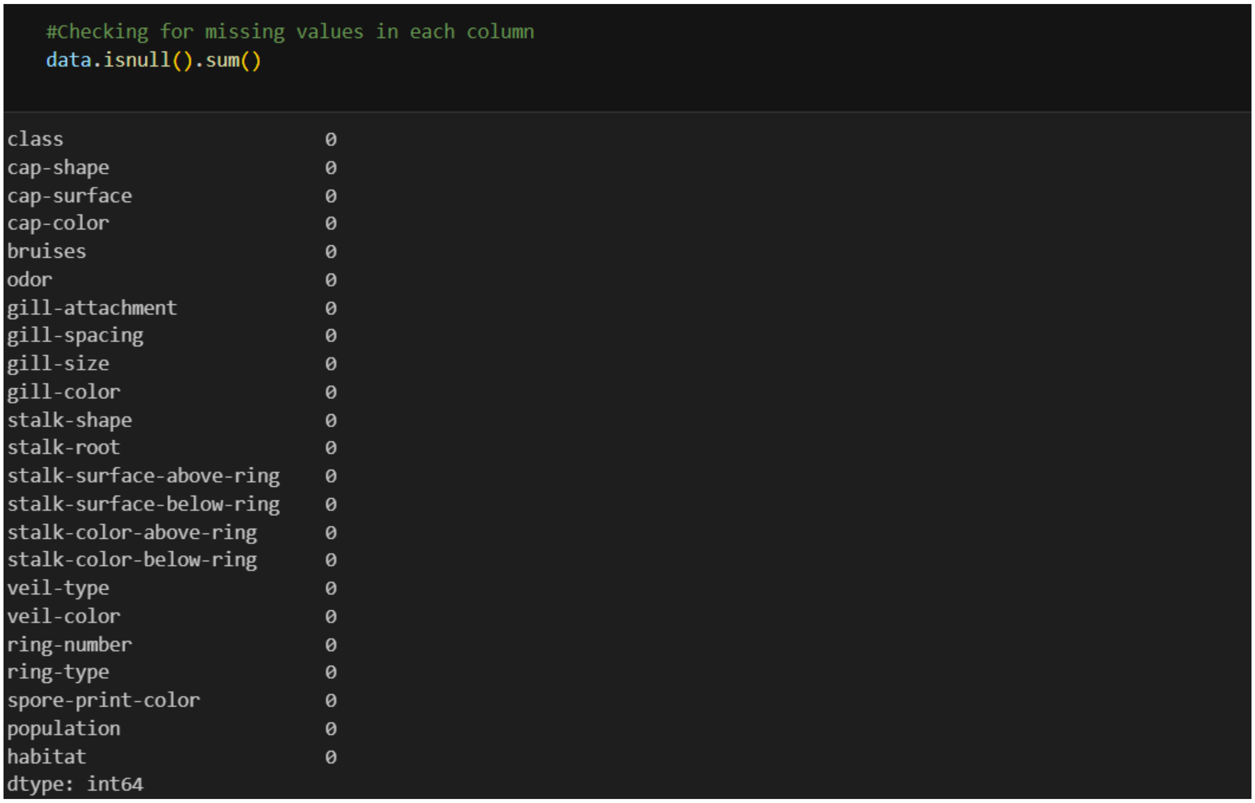
- Used to verify the presence of the dataset file in the working directory.
- Assisted in defining and managing the dataset file path before importing the data.

2.2 Data Wrangling

2.2.1 Identifying and Handling Missing Value

A thorough inspection was conducted to identify missing or null values in the dataset. After examination using **data.isnull.sum()**, it was confirmed that the mushroom dataset contains no missing values. All attributes are fully populated, allowing the dataset to be used directly without requiring data imputation or removal of records.

Figure 2.6: Display the result for confirmation of no missing values in each column.



```
#Checking for missing values in each column
data.isnull().sum()
```

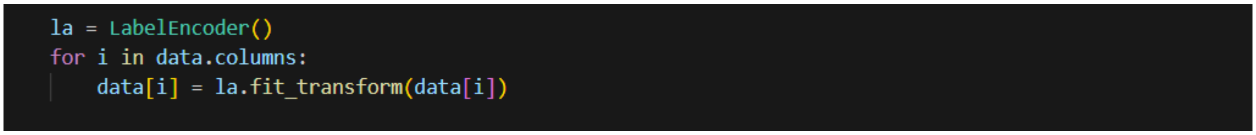
```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
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
```

2.2.2 Data Formatting

The data types of each attribute were examined to ensure consistency and compatibility with machine learning algorithms. Since the mushroom dataset consists entirely of categorical variables, Label Encoding was applied to convert all categorical attributes into numerical values. This process transformed each category into integer codes, ensuring a uniform numerical representation across all features and enabling the dataset to be directly used for model training.

We use Label Encoding for simplicity:

Figure 2.7: Convert dataset string to integer



```
la = LabelEncoder()
for i in data.columns:
    data[i] = la.fit_transform(data[i])
```

Figure 2.7 shows the target column class is encoded as 0 = edible, 1 = poisonous. Each categorical feature is converted into integer codes.

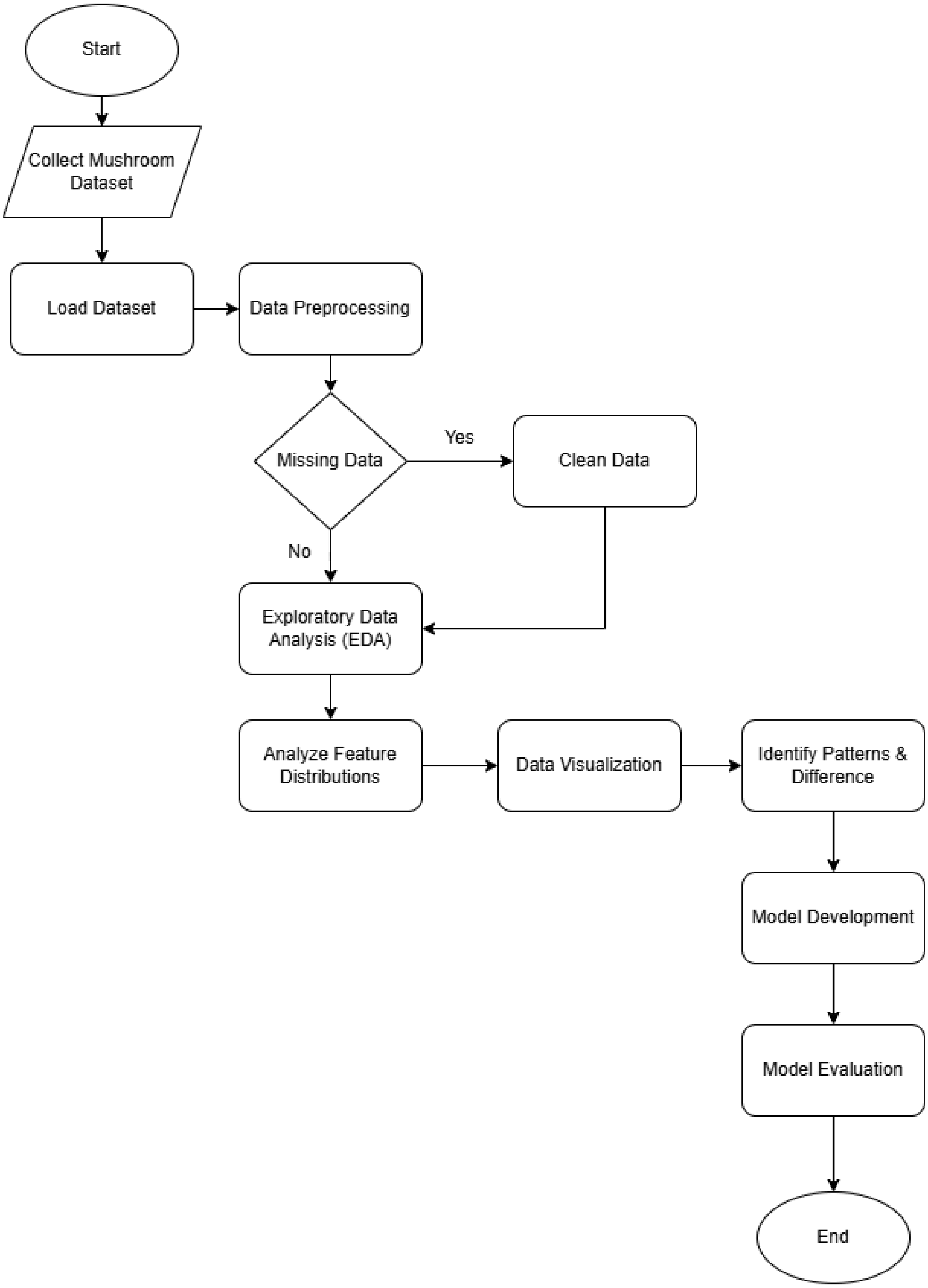
2.3 Software and Hardware Requirements

1. Software Requirements
   - Operating System : Windows 11
   - Programming Language : Python 3.13
   - Development Environment : Visual Studio Code
   - Libraries : pandas, numpy, matplotlib, scikit-learn, os, seaborn
   - Dataset Source : Kaggle

2. Hardware
   - **Processor (CPU):** The central processing unit is responsible for executing all program instructions and handling computations. A modern processor allows data operations and analysis tasks to be performed quickly.
   - **Memory (RAM):** Sufficient RAM is needed to load the dataset and perform multiple data operations without slowing down the system.
   - **Storage:** Storage is needed to save the dataset, code and results safely.
   - **Graphics Processing Unit (GPU):** Having a GPU can accelerate plotting and rendering of visualizations in some environments.

Component	**Specification**
Processor (CPU)	Intel® Core(™) i5-8350U or equivalent
Memory (RAM)	16.0 GB / 32.0 GB
Storage	At least 1 GB free space
GPU	8 GB

3.0 Flowchart of Proposed Approach



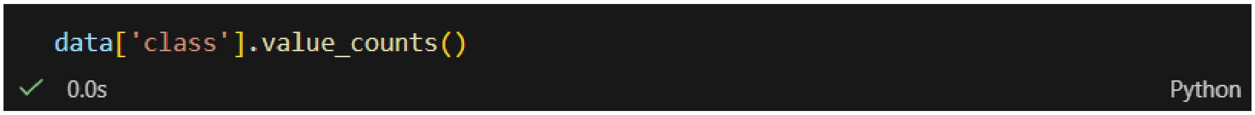
3.1 Exploratory Data Analysis

This section is focusing on understanding the data structure and the attributes' relationships. This helps in identifying important attributes of the data for the model. The results gained from this analysis will contribute to develop an effective model.

3.1.1 Target Variable Distribution Analysis

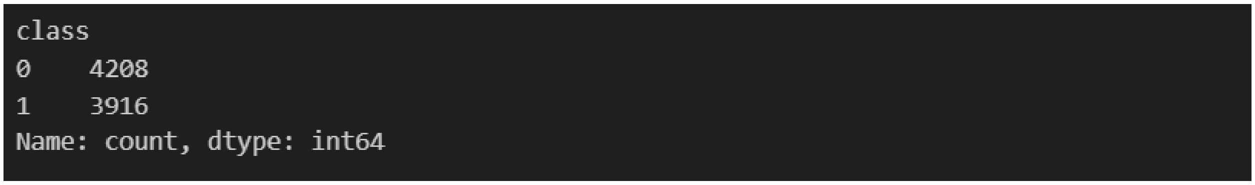
Examine the distribution of class (target variable) balance to avoid biased models. The distribution was identified using the value_counts() function, which give the total of each class (edible and poisonous)

Figure 3.0: value_counts() shows the class distribution



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

Result:



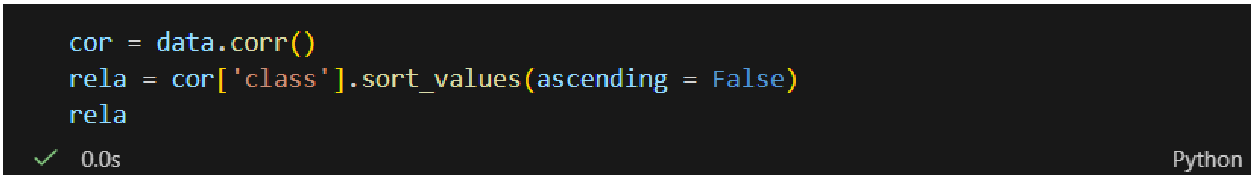
```
class
0    4208
1    3916
Name: count, dtype: int64
```

Figure 3.1: The analysis shows that the class's distribution is effectively balanced with the ratio as 1.07:1. The '0' is for edible, while '1' is for poisonous class. This is crucial for a classification because the imbalance dataset may lead to model bias, causing the algorithm to favor the majority class, affecting the result.

3.1.2 Correlation Analysis

To understand the linear relationships between all attributes and their individual effect on the target variable. A correlation matrix was computed. Using the data.corr() function to measure the relationship strength between two attributes. This implies all the attributes. The result is a value in between -1 and 1, where 1 indicates a perfect positive correlation, -1 for a perfect negative correlation, and 0 indicates no linear correlation. Meanwhile, sort_values(ascending = False) being used to sort the correlation values in descending order.

Figure 3.2: Calculate the correlation

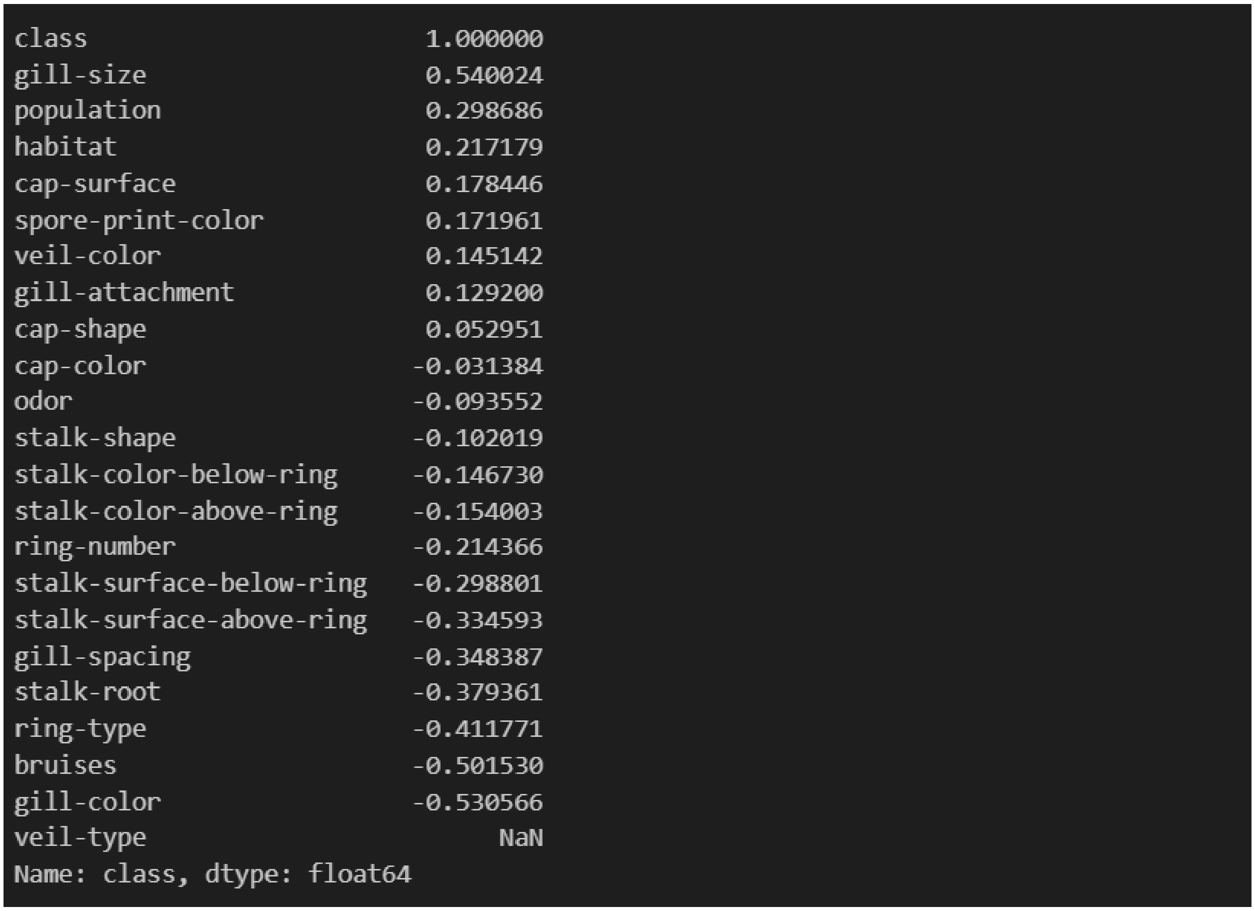


```
cor = data.corr()
rela = cor['class'].sort_values(ascending = False)
rela
```

✓ 0.0s Python

Result:

Figure 3.3: The correlation output



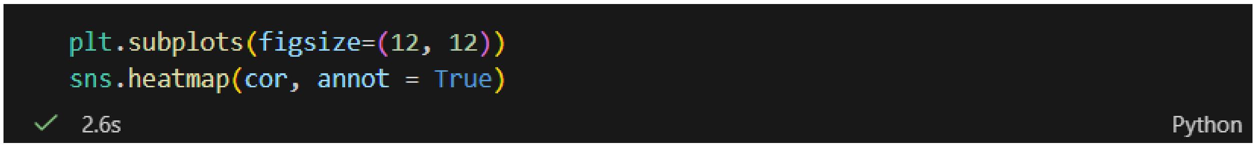
```
class                      1.000000
gill-size                  0.540024
population                 0.298686
habitat                    0.217179
cap-surface                0.178446
spore-print-color          0.171961
veil-color                 0.145142
gill-attachment            0.129200
cap-shape                  0.052951
cap-color                 -0.031384
odor                      -0.093552
stalk-shape               -0.102019
stalk-color-below-ring    -0.146730
stalk-color-above-ring    -0.154003
ring-number               -0.214366
stalk-surface-below-ring  -0.298801
stalk-surface-above-ring  -0.334593
gill-spacing              -0.348387
stalk-root                -0.379361
ring-type                 -0.411771
bruises                   -0.501530
gill-color                -0.530566
veil-type                       NaN
Name: class, dtype: float64
```

Based on the output, the most predictive features are gill-size (0.54), gill-color (-0.53), and bruises (-0.50) where the gill-size is positive correlation and opposite for gill-color and bruises. It is because these features have strong correlations which means the value is greater than 0.5, crucial in predicting the mushroom class. However, the veil-type is NaN (Not a Number) because it has no variation which makes it useless for the prediction. Therefore, veil-type cannot be used to differentiate the mushroom classes.

3.1.3 Visualization Correlation

Using heatmap visualization to easily interpret the correlation matrix. Allowing a quick overview of all relationships between all features. The heatmap uses color gradients to differentiate the intensity of the correlation. The lightest color represents -1 while the darkest color represents +1. Thus, it is easier to identify the most influential features.

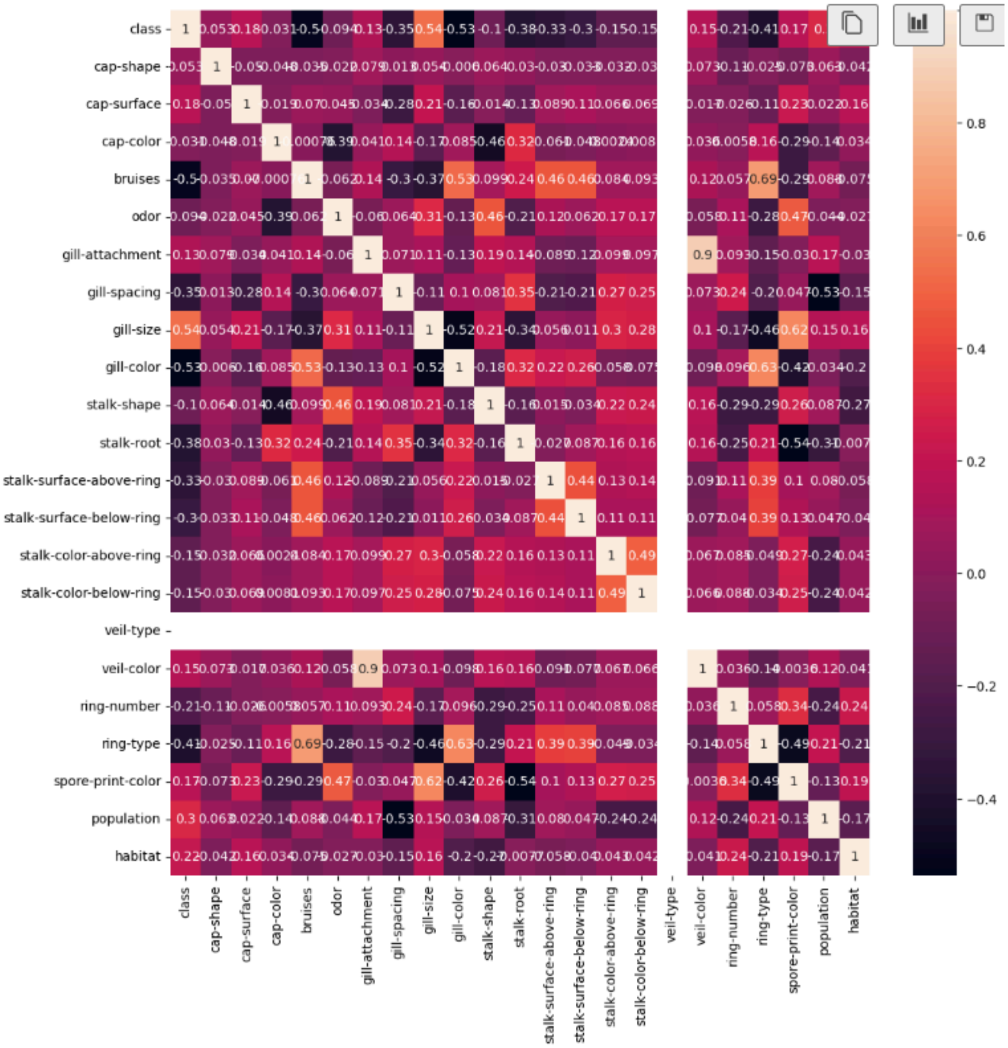
Figure 3.4: Functions to construct the heatmap visualization



```
plt.subplots(figsize=(12, 12))
sns.heatmap(cor, annot = True)
```

Result:

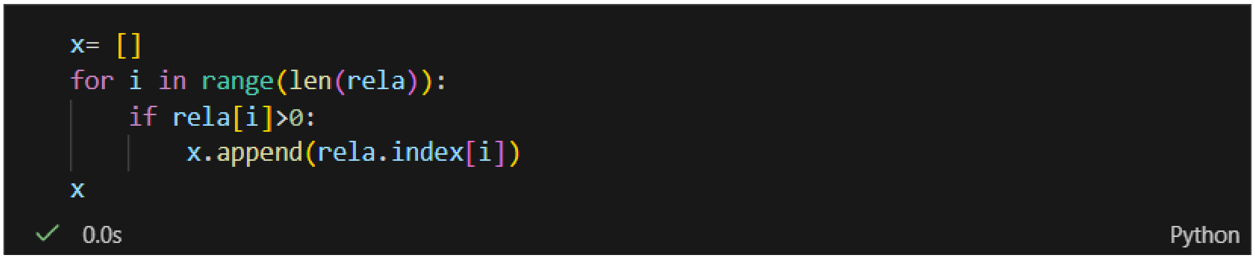
The heatmap highlighting the gill-size (0.54), gill-color (-0.53), and bruises (-0.50) among the most highly correlated with the class, a target variable. This proves that these features are strong indicators of whether a mushroom is edible or poisonous.



3.1.4 Feature Selection based on Correlation

Following the correlation analysis, a features selection process was conducted to extract the positive correlated variables and prepare the final dataset for model training. The objective is to create a dataset that would enhance model interpretability and efficiency, helping the model to find a perfect algorithm and gain 100% accuracy.

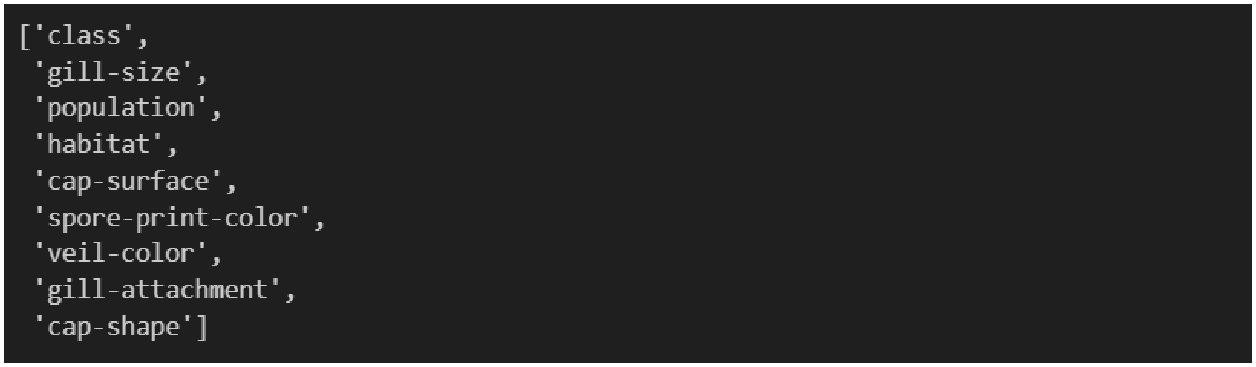
Figure 3.5: Selected variables with positive correlation



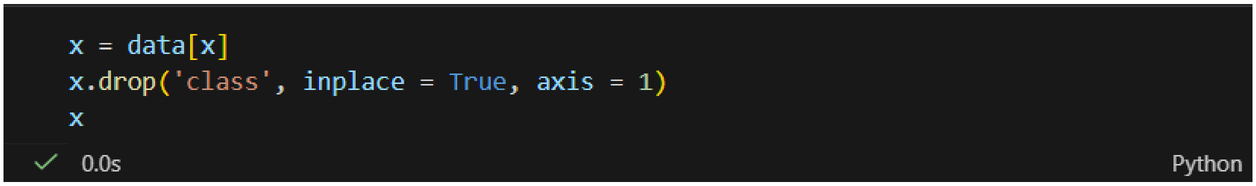
```python
x= []
for i in range(len(rela)):
    if rela[i]>0:
        x.append(rela.index[i])
x
```

Result:

Figure 3.6: The selected variables for model training and testing



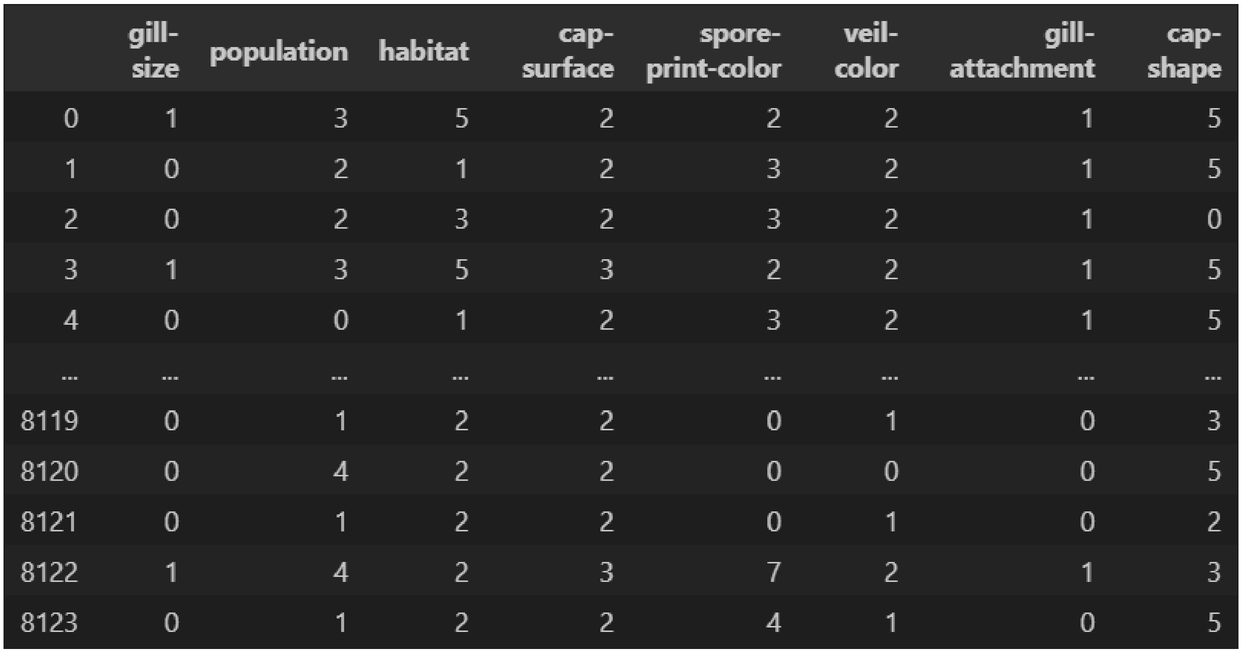
```
['class',
 'gill-size',
 'population',
 'habitat',
 'cap-surface',
 'spore-print-color',
 'veil-color',
 'gill-attachment',
 'cap-shape']
```



```
x = data[x]
x.drop('class', inplace = True, axis = 1)
x
```

✓ 0.0s Python

Result:

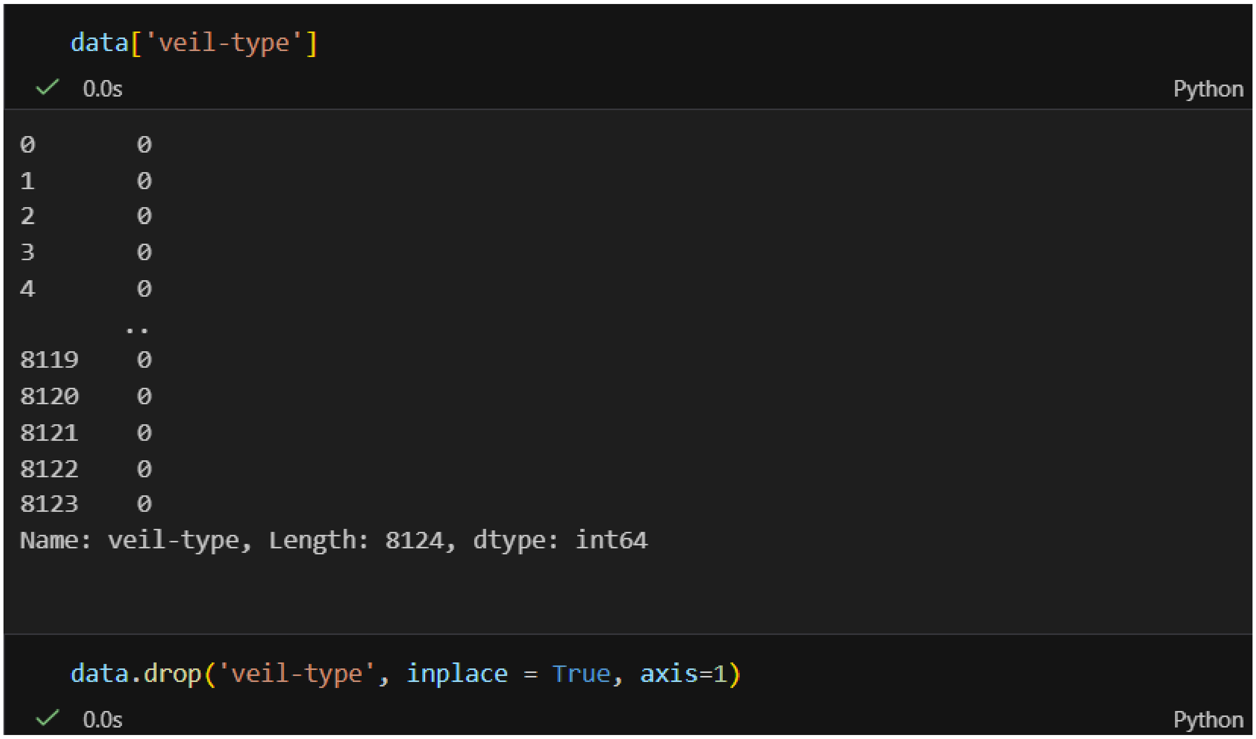


	gill-size	population	habitat	cap-surface	spore-print-color	veil-color	gill-attachment	cap-shape
0	1	3	5	2	2	2	1	5
1	0	2	1	2	3	2	1	5
2	0	2	3	2	3	2	1	0
3	1	3	5	3	2	2	1	5
4	0	0	1	2	3	2	1	5
...	...	...	...	...	...	...	...	...
8119	0	1	2	2	0	1	0	3
8120	0	4	2	2	0	0	0	5
8121	0	1	2	2	0	1	0	2
8122	1	4	2	3	7	2	1	3
8123	0	1	2	2	4	1	0	5

Result Interpretation:

Above are the variables (columns) that have positive correlation value extracted for model training. By selecting these variables, the model is built exclusively on characteristics that are expressing the poisonous class. Therefore, the model is trained to be highly aware to poisonous indicators and prioritize any feature that strongly associated with toxicity. This minimizes the False Negative where the model classifies a poisonous mushroom as edible.

Figure 3.7: Shows the veil-type attributes have the same value for all instances ('0' refers to 'partial')



```
data['veil-type']
```

✓ 0.0s Python

```
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: int64
```

```
data.drop('veil-type', inplace = True, axis=1)
```

✓ 0.0s Python

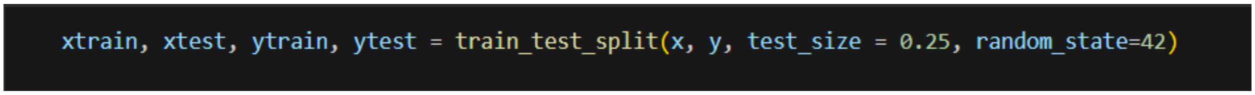
Therefore, the veil-type was removed from the dataset because it was completely useless in the model training. It doesn't help the model to differentiate between edible or poisonous mushrooms.

3.2 Model Development

The primary objective of Model Development is to develop, train, and prepare a machine learning classification models that capable of accurately predicting mushroom edibility based on the selected features. The goal is to implement and train several models (Random Forest, Decision Tree, and Logistic Regression) to establish a baseline for performance evaluation in the subsequent phase.

3.2.1 Data Partitioning (Train-Test Split)

To avoid biased model evaluation, the prepared dataset was divided into two subsets, a training set and a testing set. This was accomplished by the train_test_split() function from the scikit-learn library. The model trains on a portion of the data and objectively evaluating its performance on unseen data. This helps prevent overfitting where the model only memorizes the training data but performs poorly using the new or unseen data and underfitting.



```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25, random_state=42)
```

Training Set (75%): This subset consists of xtrain and ytrain, used to train the models. The models analyze this data to learn the algorithms and relationships between the features and the target class.

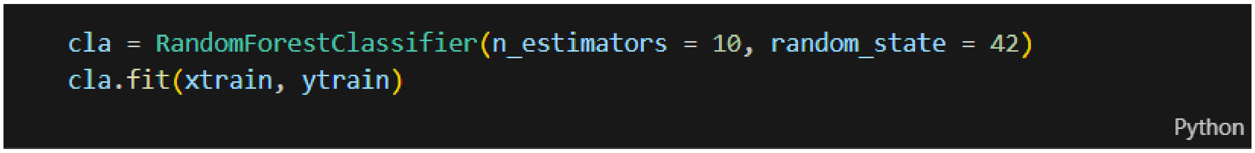
Testing Set (25%): This subset consists of xtest and ytest, and was kept separate during the training process. It serves as unseen data to evaluate the generalization performance of the trained models fairly.

A random_state = 42 was used to ensure that the data split was identical every time the code was run.

3.2.2 Model Training

A diverse set of three classification algorithms was chosen:

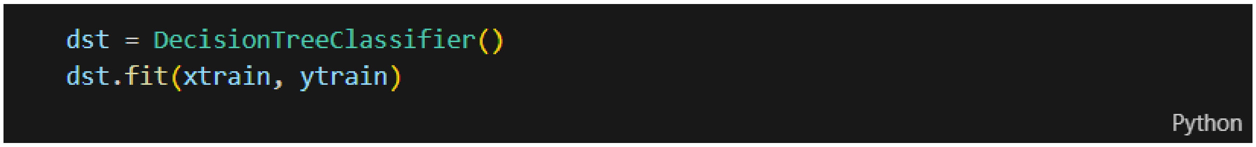
3.2.2.1 Random Forest Classifier



```python
cla = RandomForestClassifier(n_estimators = 10, random_state = 42)
cla.fit(xtrain, ytrain)
```

A Random Forest Classifier model is instantiated. A method operates by constructing a number of decision trees during training. n_estimators = 10 means the forest will be composed of 10 decision trees. Known for its high accuracy, and robustness to overfitting.

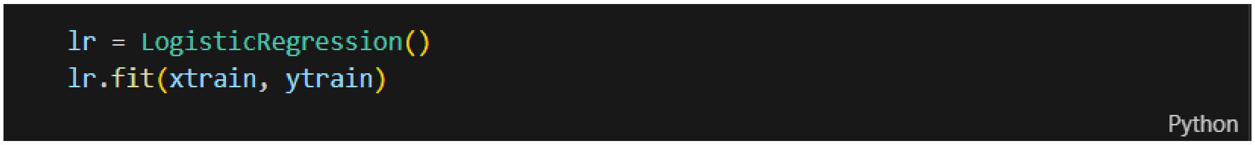
3.2.2.2 Decision Tree Classifier



```python
dst = DecisionTreeClassifier()
dst.fit(xtrain, ytrain)
```

A simple, non-parametric machine learning algorithm that serves as an excellent baseline. It uses a tree-like structure to make a decision and classify data into categories. Split the dataset into smaller, more homogeneous subsets based on the most significant features until leaf nodes (terminal nodes) are decided.

3.2.2.3 Logistic Regression



```python
lr = LogisticRegression()
lr.fit(xtrain, ytrain)
```

A linear model used for binary classification. This model examines the relationship between all variables and target variables. It is a linear model, thus it used when the output variable or target variable is continuous.

All selected models were instantiated and trained using the .fit() function. Each model trained using the training data (xtrain, ytrain), allowing it to learn and know the pattern of mapping from input features to the target output. The models also independent during the training process by trained separately using the same training data.

3.3 Model Evaluation

The performance evaluation for each trained model was strictly evaluated on the test set. For each model (Random Forest, Decision Tree and Logistic Regression), the evaluation was calculated based on the evaluation matrices.

3.3.1 Evaluation Metrices

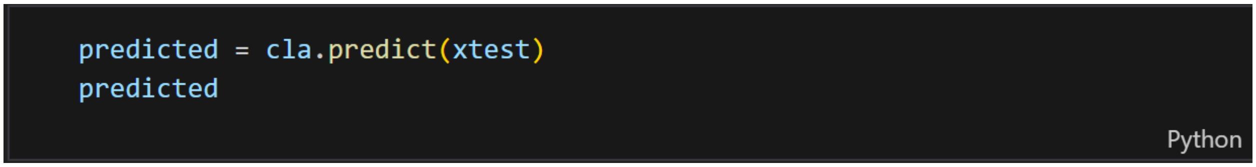
The following metrics were calculated to evaluate the models:

1. **Accuracy:** The overall percentage of correct predictions.
2. **ROC Curve and AUC Score:** The ROC curve was plotted to visualize model performance and the AUC score was calculated to quantify it. Models that achieved an AUC 1.0 , indicating the perfect classification.
3. **Confusion Matrix:** Confusion matrix was generated to analyze the types of errors that were made by each of the models.
4. **Classification Report:** Precision, recall and F1-score were also computed for each class. All of which were 1.0 confirming that the model is perfect in performance on this dataset.

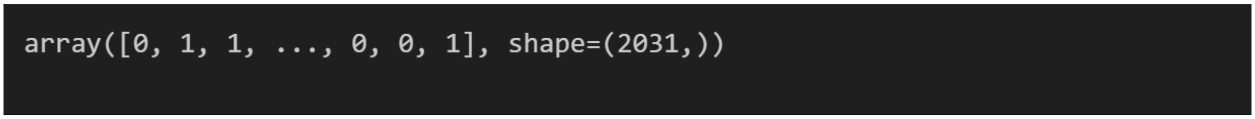
3.3.2 Evaluated Models

3.3.2.1 Random Forest

Figure 5.0: Making predictions on testing data



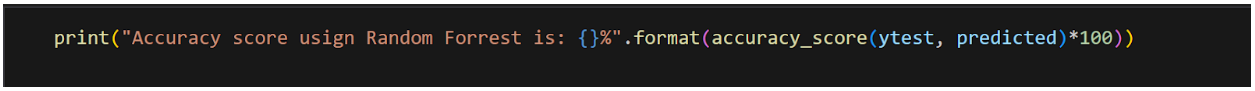
```
predicted = cla.predict(xtest)
predicted
```



```
array([0, 1, 1, ..., 0, 0, 1], shape=(2031,))
```

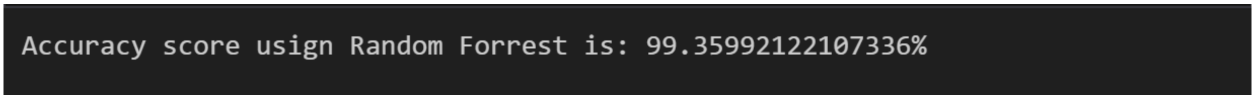
Based on **Figure 5.0,** cla refers to trained Random Forest Classifiers. All models use the predict() function to generate the predicted class labels (dst as Decision Tree Classifier and lr as Logistic Regression) based on the xtest that contains values for all selected variables. The output array predicted label for 2031 test samples (0 = edible, 1 = poisonous).

Figure 5.1: Accuracy Calculation



```
print("Accuracy score usign Random Forrest is: {}%".format(accuracy_score(ytest, predicted)*100))
```

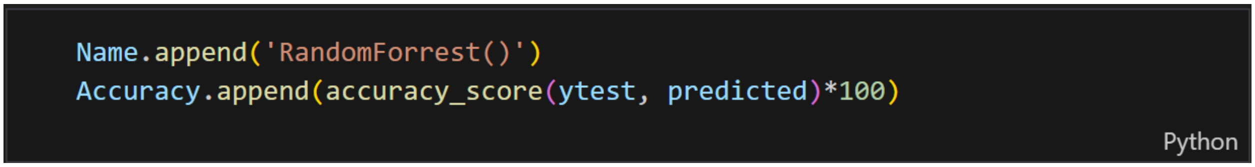
accuracy_score() function is comparing the predicted labels with the true labels. The results are multiplied by 100 to express the accuracy as a percentage.



```
Accuracy score usign Random Forrest is: 99.35992122107336%
```

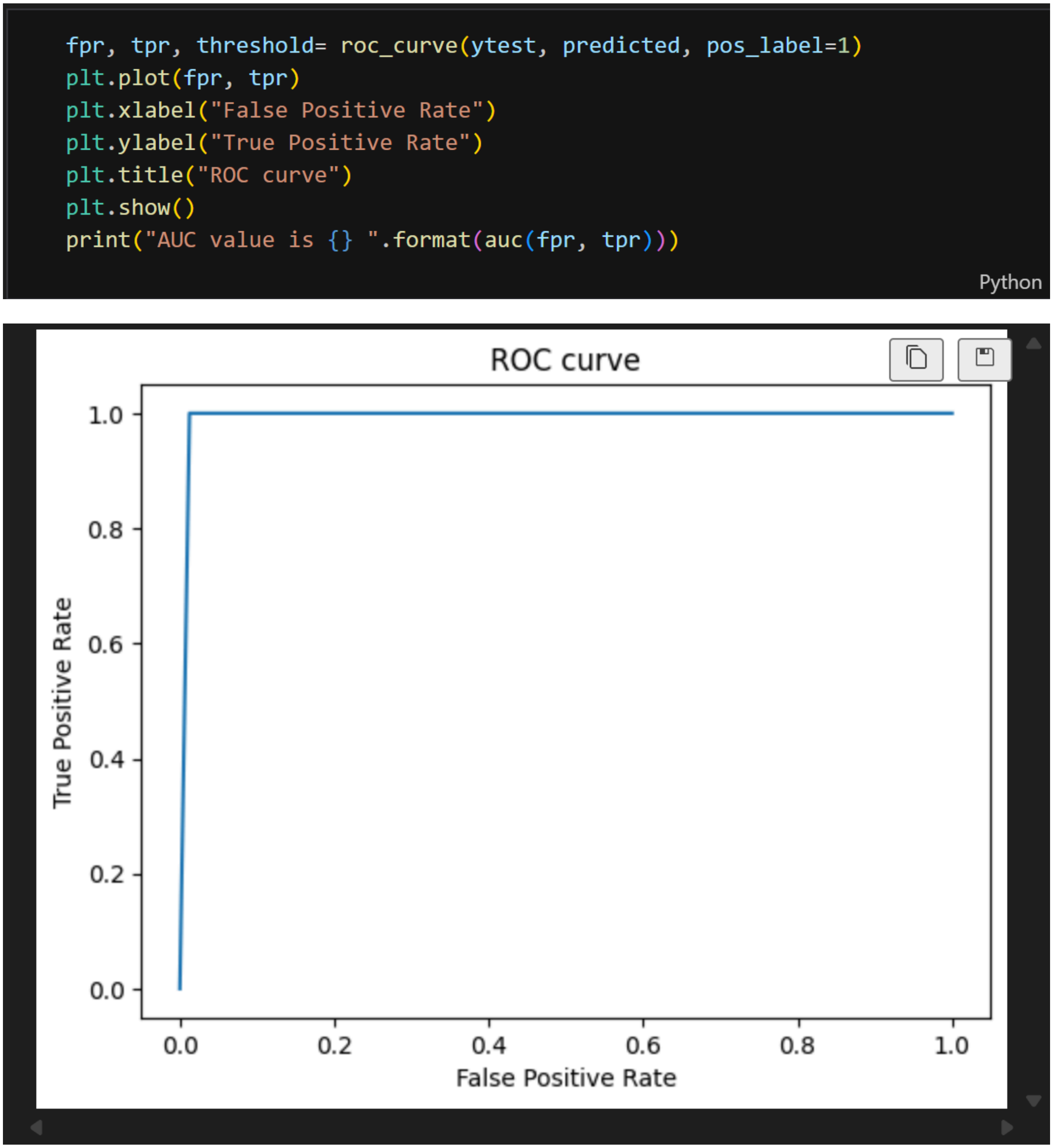
The accuracy showed that almost all mushrooms were correctly classified by the Random Forest model.

Figure 5.2: Adding Random Forest model in the name list and the accuracy for Random Forest model in Accuracy list



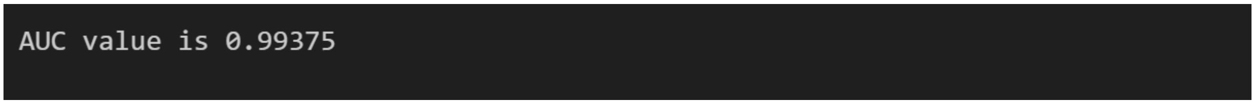
```
Name.append('RandomForrest()')
Accuracy.append(accuracy_score(ytest, predicted)*100)
```

Figure 5.3: ROC Curve



```
fpr, tpr, threshold= roc_curve(ytest, predicted, pos_label=1)
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curve")
plt.show()
print("AUC value is {} ".format(auc(fpr, tpr)))
```

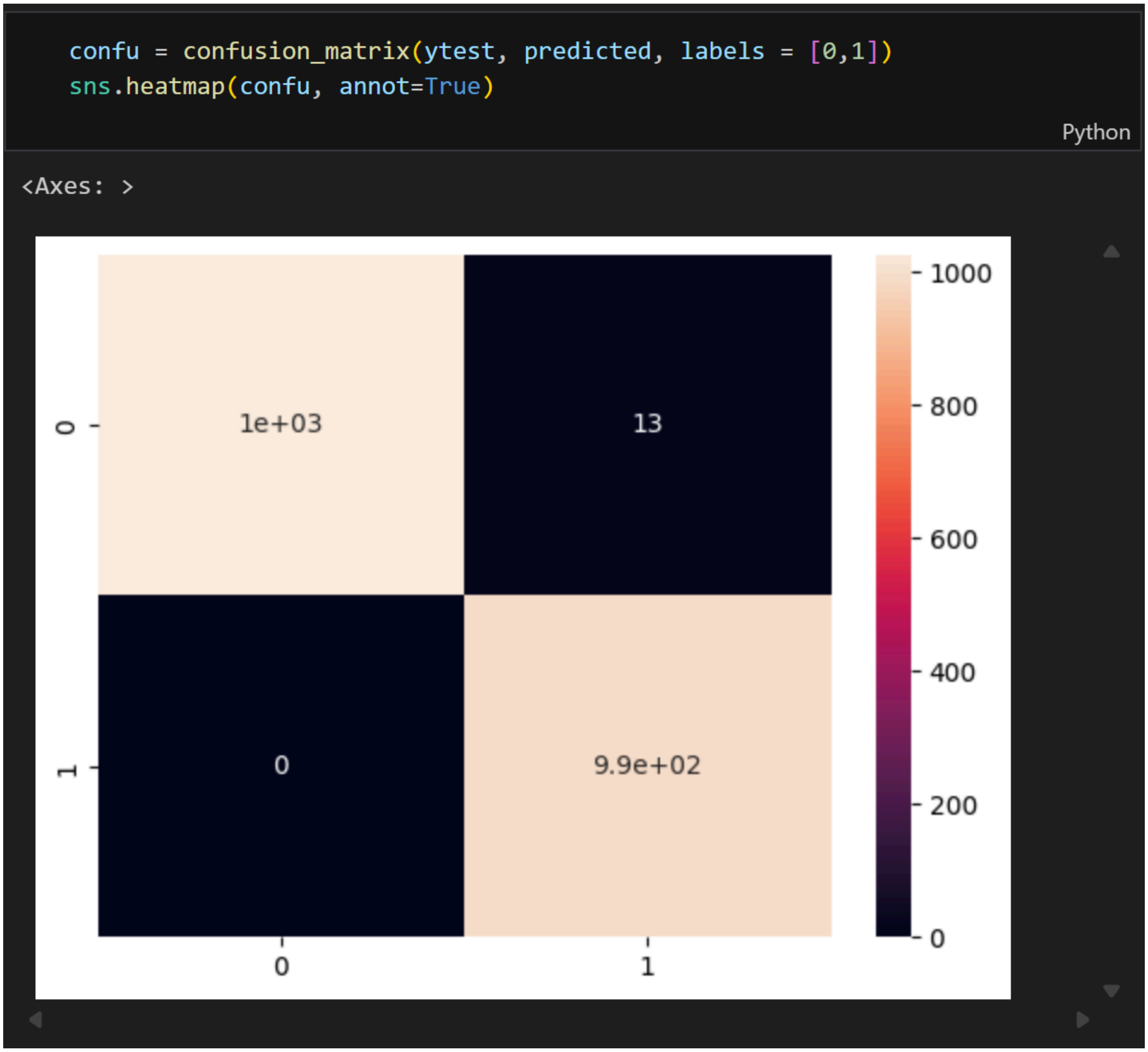
The x-axis represents the false positive rate while the y-axis represents the true positive rate. A curve is closer to the top left corner showing the false positive rate is low while the true positive rate is high indicating the better classification performance.



```
AUC value is 0.99375
```

The AUC score summarizes the ROC curve into a single value and indicates the excellent discriminative ability showing that Random Forest can distinguish between two classes.

Figure 5.4: Confusion matrix

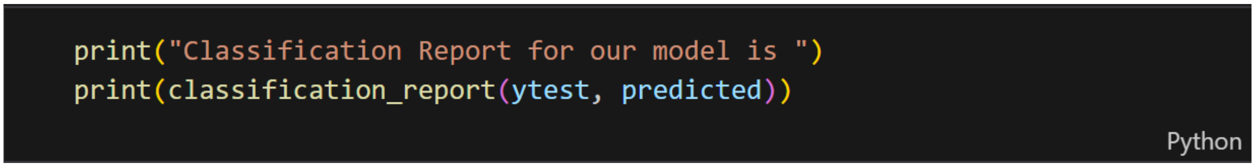


```python
confu = confusion_matrix(ytest, predicted, labels = [0,1])
sns.heatmap(confu, annot=True)
```

```
<Axes: >
```

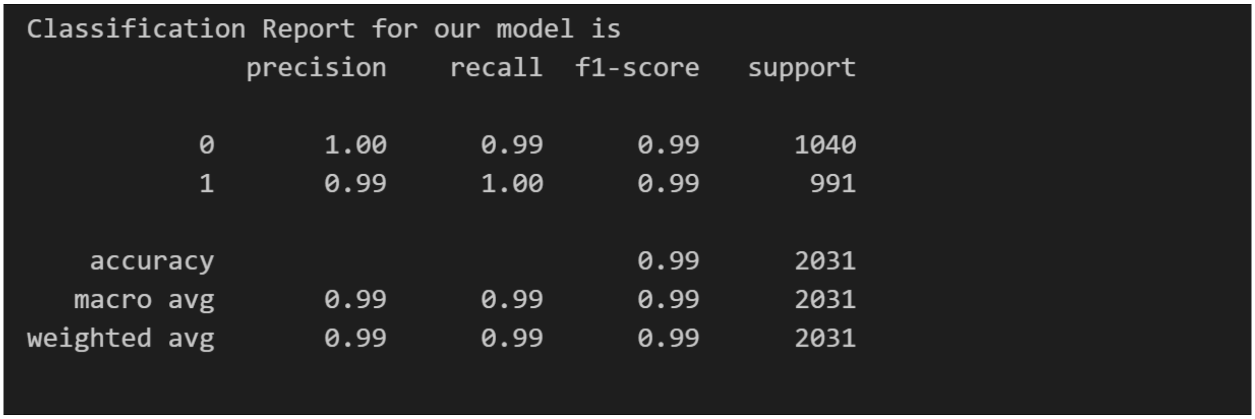
A confusion matrix compares actual labels with the predicted labels. It contains True Positives (Bottom-Right cell), True Negatives (Top-Left cell), False Positive (Top-Right cell), False Negatives (Bottom-Left cell). This visualizes the confusion matrix using a heatmap. annot=True displays the numeric in each cell. The models correctly identified approximately 100 edible mushrooms (True Negative) and 990 poisonous mushrooms (True Positive). There were zero instances where the model incorrectly classified poisonous mushrooms as edible mushrooms. This proof that the model are perfectly safe and did not produce errors that are life threatening. However, there were 13 instances of edible mushrooms incorrectly identified as poisonous.

Figure 5.5: Classification Report



```python
print("Classification Report for our model is ")
print(classification_report(ytest, predicted))
```

Result:



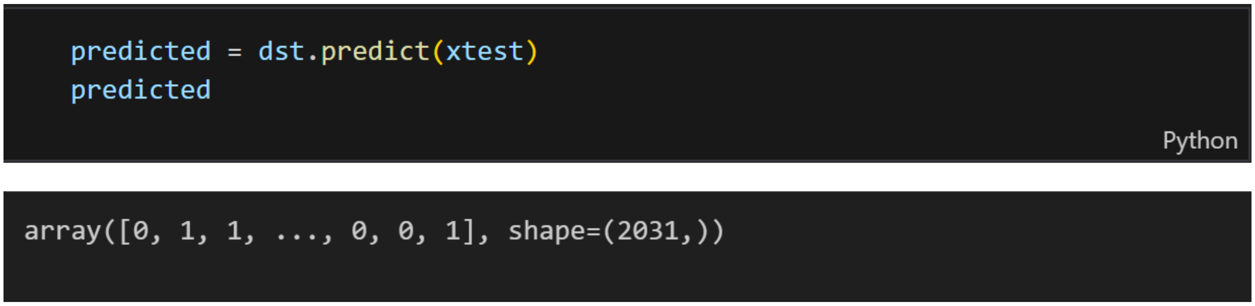
```
Classification Report for our model is 
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1040
           1       0.99      1.00      0.99       991

    accuracy                           0.99      2031
   macro avg       0.99      0.99      0.99      2031
weighted avg       0.99      0.99      0.99      2031
```

**(Note: 0=edible, 1=poisonous)

Precision, recall, and F1-score values are all approximately 0.99 (macro average and weighted average), confirming consistent classification quality. This indicates balanced and reliable performance across both classes.

3.3.2.2 Decision Tree

Figure 5.6: Making prediction on testing data

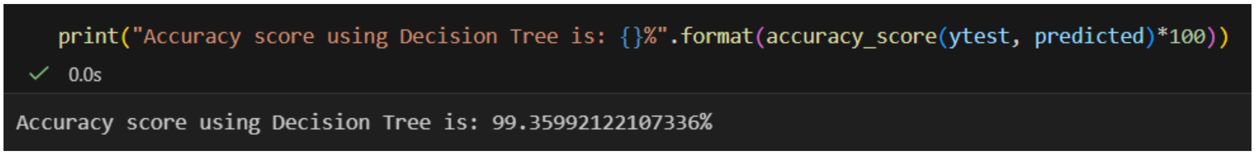


```
predicted = dst.predict(xtest)
predicted
```

```
array([0, 1, 1, ..., 0, 0, 1], shape=(2031,))
```

Based on **Figure 5.6** dst refers to trained Decision Tree Classifiers. The output array predicted label for 2031 test samples (0 = edible, 1 = poisonous).

Figure 5.7: Accuracy Calculation

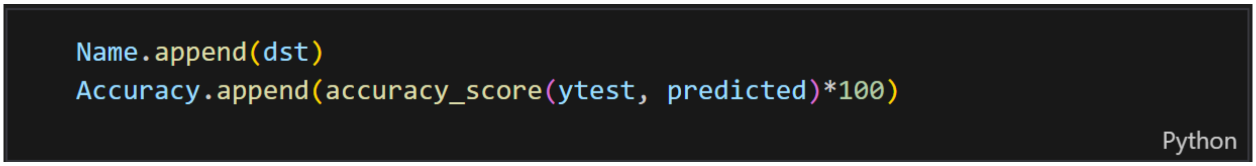


```
print("Accuracy score using Decision Tree is: {}%".format(accuracy_score(ytest, predicted)*100))
```

```
Accuracy score using Decision Tree is: 99.35992122107336%
```

accuracy_score() function is comparing the predicted labels with the true labels(ytest). The results are multiplied by 100 to express the accuracy as a percentage. The accuracy of 99.36% showed that almost all mushrooms were correctly classified by the Decision Tree model.

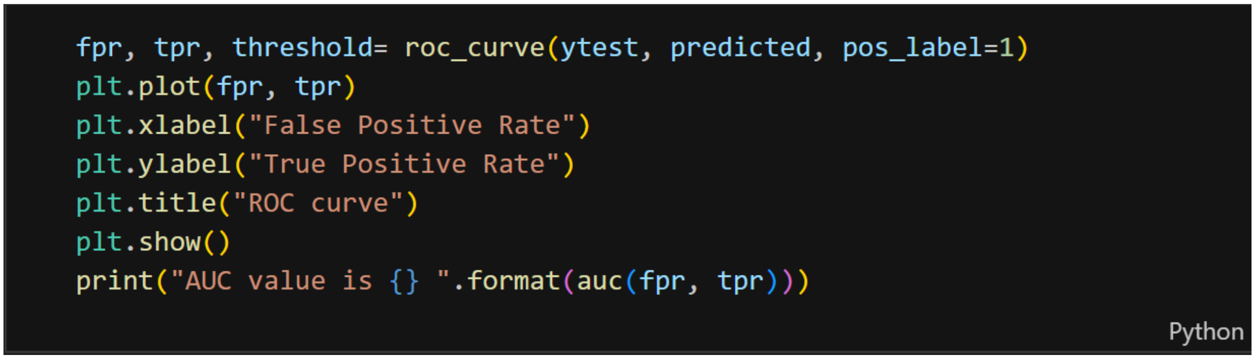
Figure 5.8: Adding Decision Tree model in the name list and the accuracy for Decision Tree model in Accuracy list



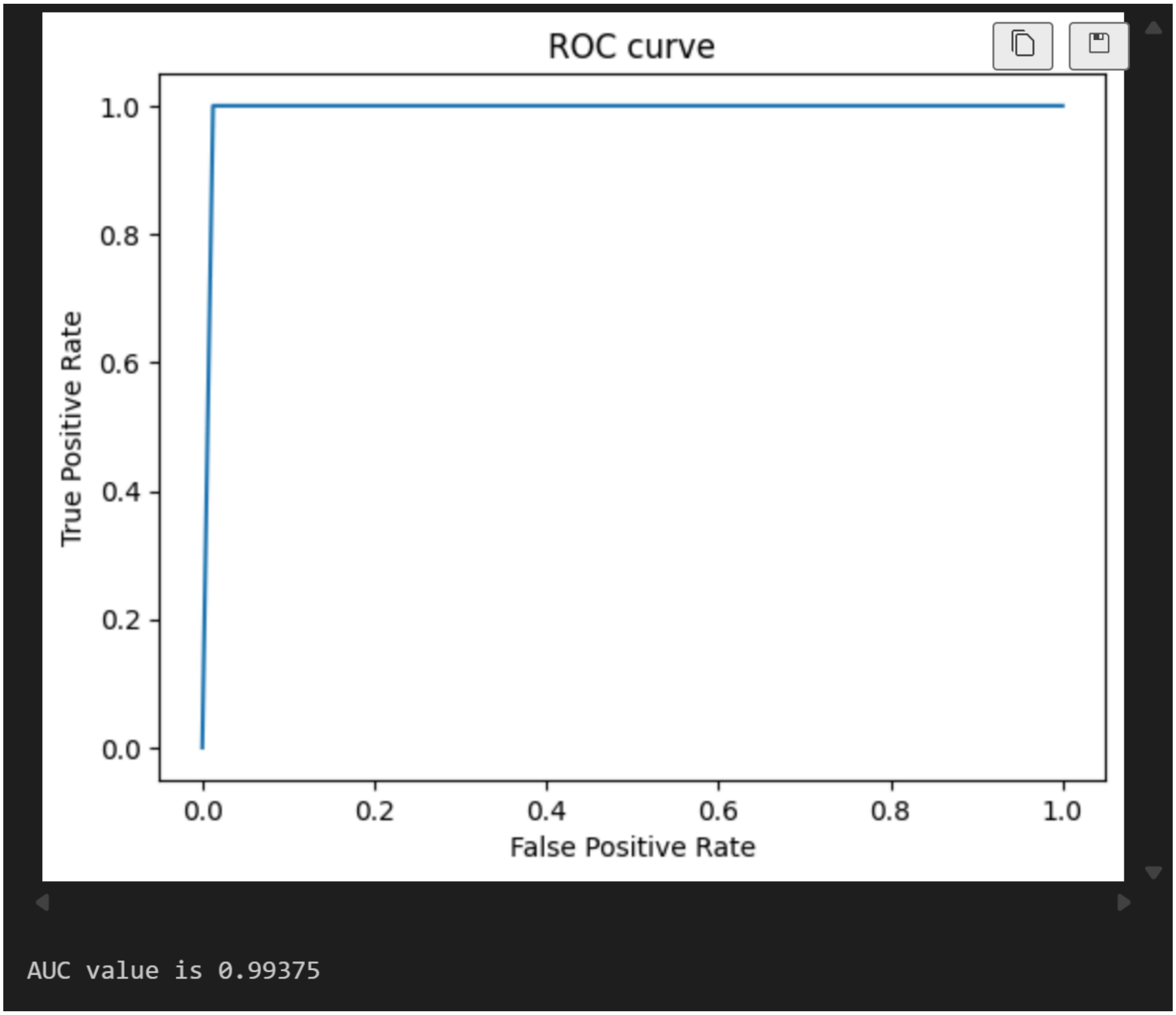
```
Name.append(dst)
Accuracy.append(accuracy_score(ytest, predicted)*100)
```

These lines save the name and the accuracy value of the Decision Tree model that can be used later to compare with other models. This enables more than one model to be assessed and compared in a systematic way.

Figure 5.9: ROC Curve



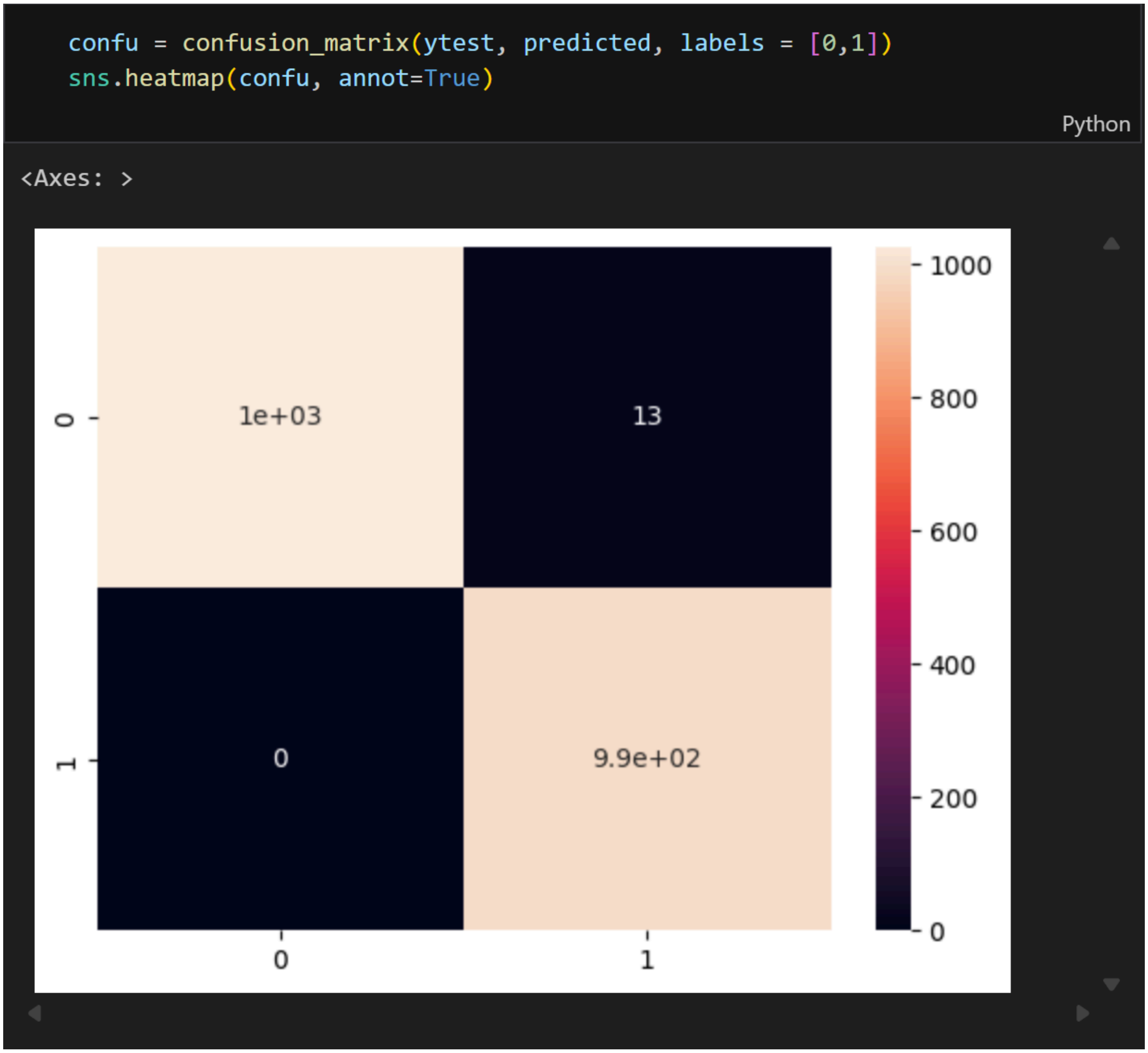
```python
fpr, tpr, threshold= roc_curve(ytest, predicted, pos_label=1)
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curve")
plt.show()
print("AUC value is {} ".format(auc(fpr, tpr)))
```



```
AUC value is 0.99375
```

The x-axis represents the false positive rate while the y-axis represents the true positive rate. A curve closer to the top left corner indicates better classification performance. An AUC score of 0.99375 indicates excellent classification ability and strong separation between classes.

Figure 5.10: Confusion matrix

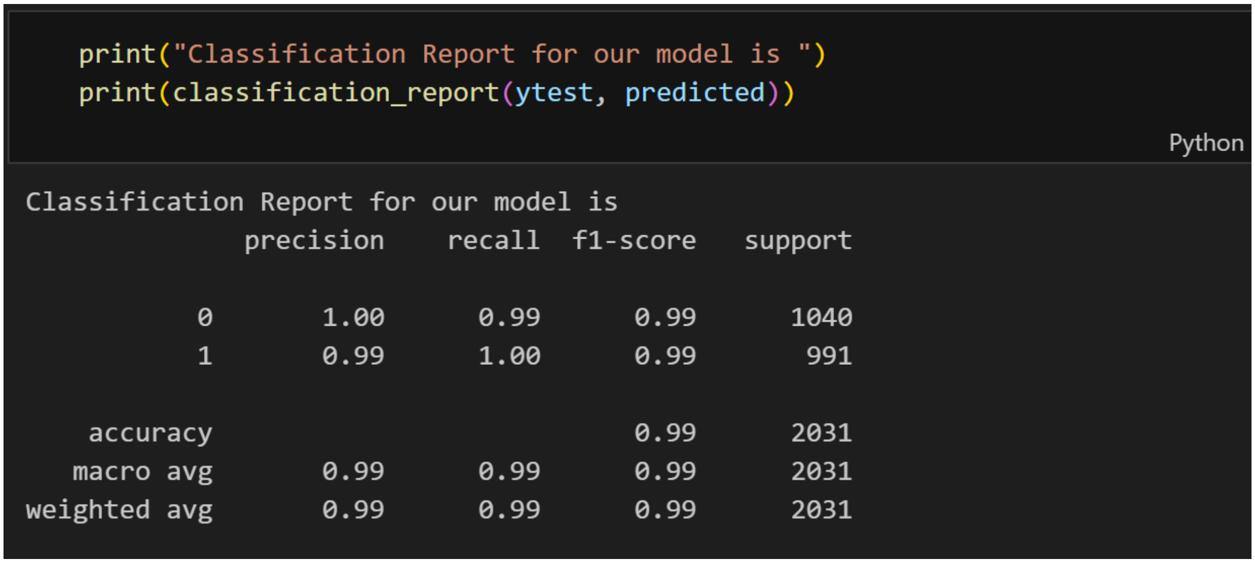


```
confu = confusion_matrix(ytest, predicted, labels = [0,1])
sns.heatmap(confu, annot=True)
```

```
<Axes: >
```

The result is the same as the Random Forest model. This model correctly identified approximately 100 edible mushrooms (True Negatives) and 990 poisonous mushrooms (True Positives). There were zero instances where the model incorrectly classified poisonous mushrooms as edible mushrooms (False Negatives). Lastly, there were 13 instances of edible mushrooms incorrectly identified as poisonous which is an acceptable trade-off, as it only results in discarding the edible mushrooms rather than consuming the poisonous ones.

Figure 5.11: Classification Report



```python
print("Classification Report for our model is ")
print(classification_report(ytest, predicted))
```

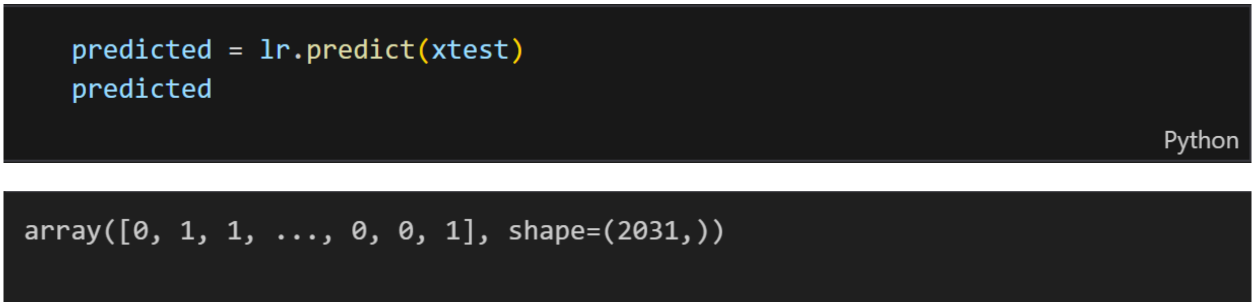
```
Classification Report for our model is 
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1040
           1       0.99      1.00      0.99       991

    accuracy                           0.99      2031
   macro avg       0.99      0.99      0.99      2031
weighted avg       0.99      0.99      0.99      2031
```

**(Note: 0=edible, 1=poisonous)

The classification results showing the detailed performance measures precision, recall, and F1 score for each class. With the macro and weighted averages around 0.99, the model showed the balanced and reliable performance across classes.

3.3.2.3 Logistic Regression

Figure 5.12: Making prediction on testing data

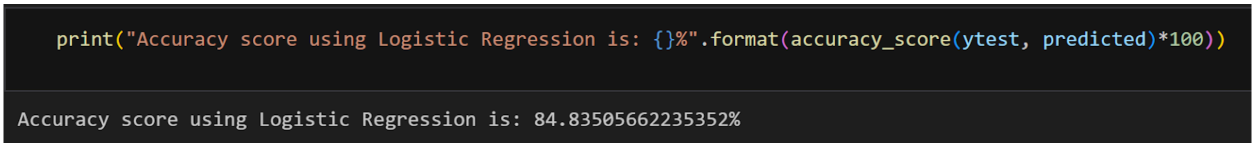


```python
predicted = lr.predict(xtest)
predicted
```

```
array([0, 1, 1, ..., 0, 0, 1], shape=(2031,))
```

Based on **Figure 5.12** lr refers to trained Logistic Regression. The output array predicted label for 2031 test samples (0 = edible, 1 = poisonous)

Figure 5.13: Accuracy Calculation

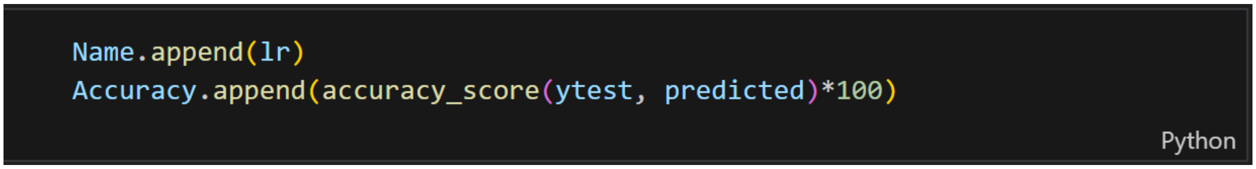


```python
print("Accuracy score using Logistic Regression is: {}%".format(accuracy_score(ytest, predicted)*100))
```

```
Accuracy score using Logistic Regression is: 84.83505662235352%
```

accuracy_score() function is comparing the predicted labels with the true labels(ytest). The results are multiplied by 100 to express the accuracy as a percentage. The accuracy of 84.84% showed that the Logistic Regression model has made the correct classification of the test samples but not all.

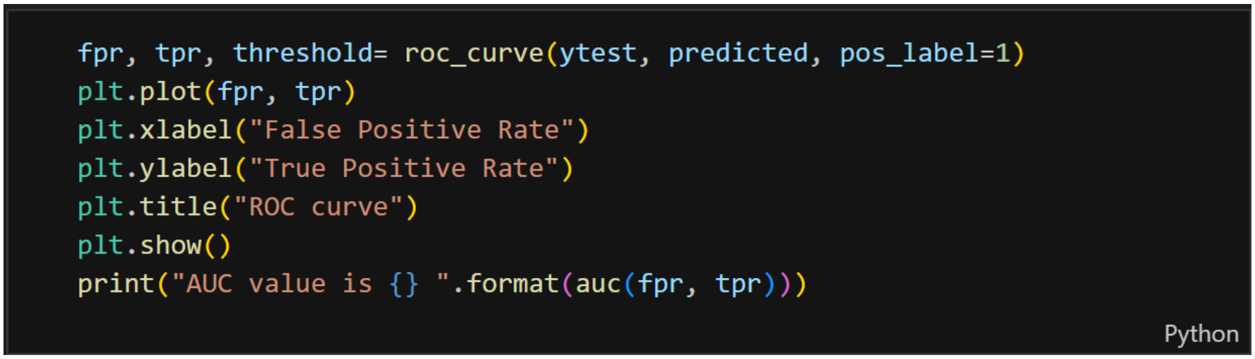
Figure 5.14: Adding Logistic Regression model in the name list and the accuracy for Logistic Regression model in Accuracy list



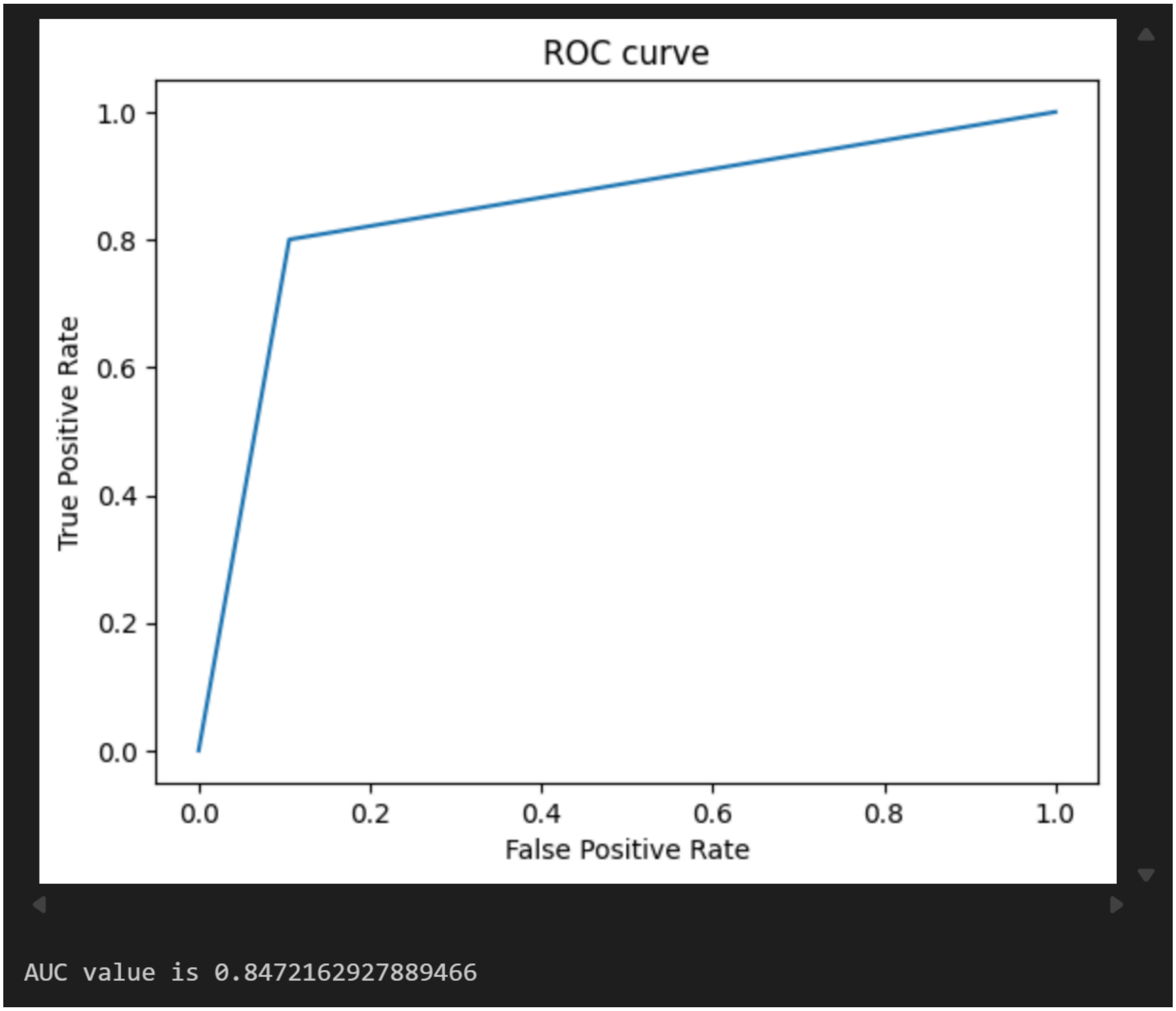
```python
Name.append(lr)
Accuracy.append(accuracy_score(ytest, predicted)*100)
```

These lines are to store the Logistic Regression model and its accuracy value for later comparison with other classification models.

Figure 5.15: Roc Curve



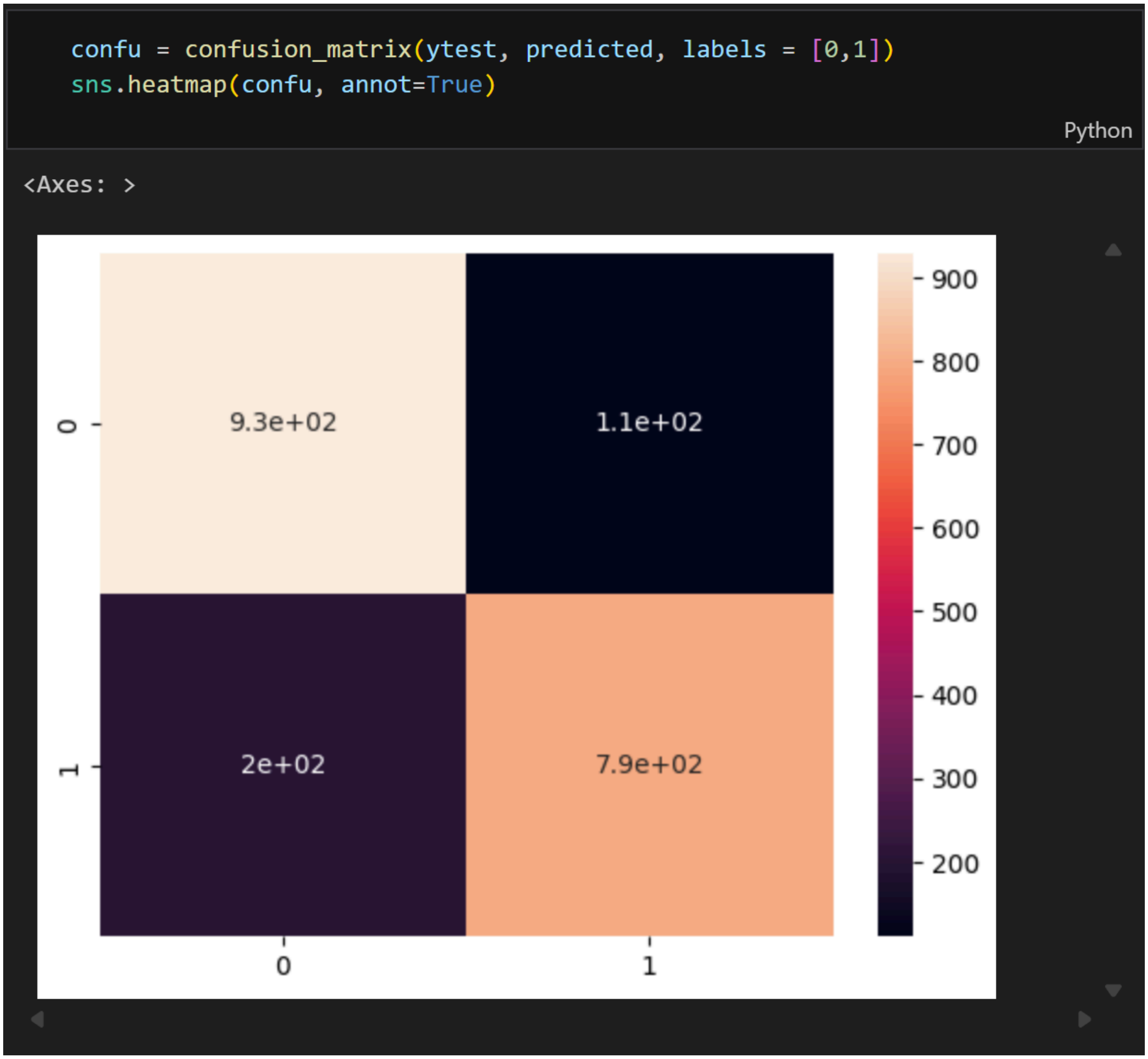
```
fpr, tpr, threshold= roc_curve(ytest, predicted, pos_label=1)
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curve")
plt.show()
print("AUC value is {} ".format(auc(fpr, tpr)))
```



```
AUC value is 0.8472162927889466
```

The x-axis represents the false positive rate while the y-axis represents the true positive rate. The curve is closer to the diagonal line compared to the curve resulting from Decision Tree and Random Forest models. The AUC is 0.8472, summarizing the ROC curve's performance. The result shows that the Logical Regression model has reasonable discriminative ability but slightly worse compared to the other two models.

Figure 5.16: Confusion matrix

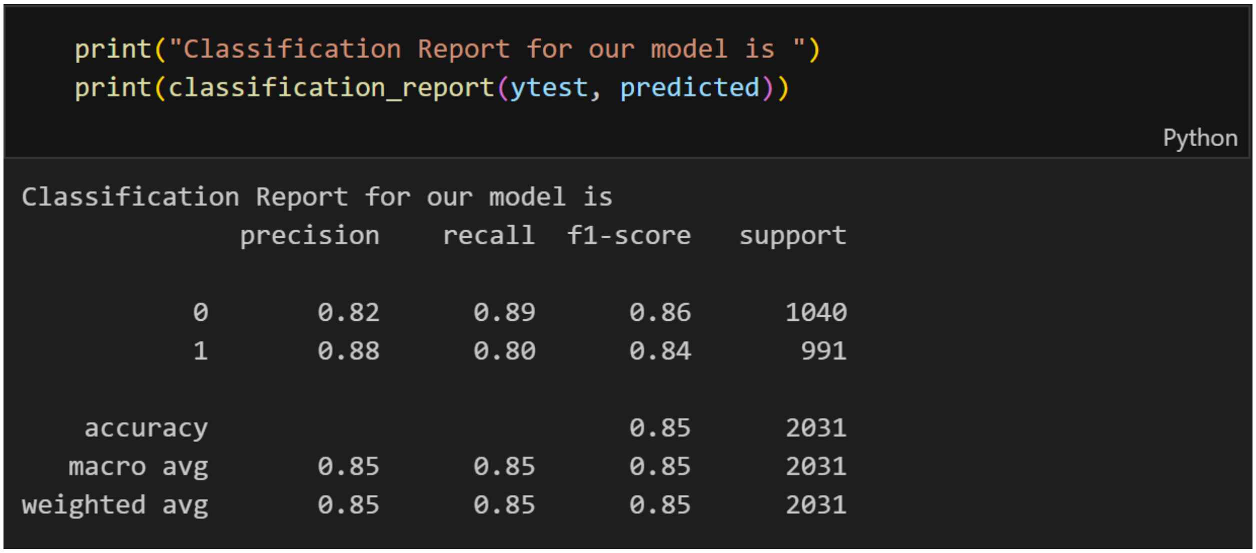


```
confu = confusion_matrix(ytest, predicted, labels = [0,1])
sns.heatmap(confu, annot=True)
```

```
<Axes: >
```

The model correctly identified approximately 930 edible mushrooms (True Negatives) and approximately 790 poisonous mushrooms successfully identified (True Positives). Meanwhile, there were 110 edible mushrooms that were incorrectly classified as poisonous (False Positives). Critically, the model made 200 dangerous errors by classifying poisonous mushrooms as edible. This result highlights the model's inability in making predictions which are dangerously unreliable for this application.

Figure 5.17: Classification Report



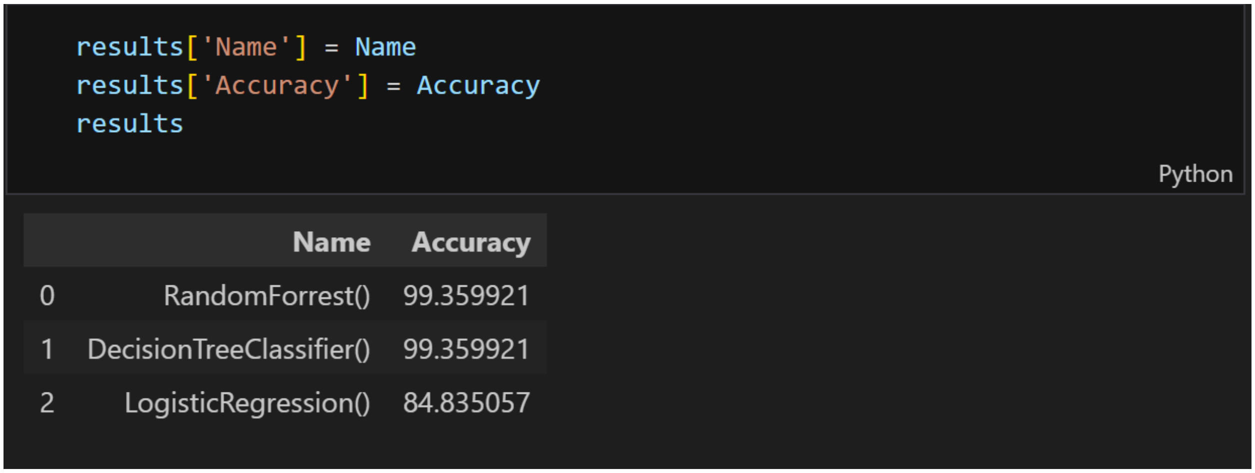
```python
print("Classification Report for our model is ")
print(classification_report(ytest, predicted))
```

```
Classification Report for our model is
              precision    recall  f1-score   support

           0       0.82      0.89      0.86      1040
           1       0.88      0.80      0.84       991

    accuracy                           0.85      2031
   macro avg       0.85      0.85      0.85      2031
weighted avg       0.85      0.85      0.85      2031
```

**(Note: 0=edible, 1=poisonous)

The classification report is a detailed report that gives the metrics of precision, recall and the F1-score of each classification. The findings show that the model is a little better on class 0 than with class 1 which reflects the limitation of a linear model in handling the complex features interaction. Overall accuracy is 0.85 with both macro and weighted averages also at 0.85 indicate consistent but not perfectly balanced performance across the class.

Figure 5.18: Final Results



```python
results['Name'] = Name
results['Accuracy'] = Accuracy
results
```

Python

```
                     Name   Accuracy
0          RandomForrest()  99.359921
1  DecisionTreeClassifier() 99.359921
2     LogisticRegression()  84.835057
```

3.3.3 Model Performance Results

A final comparison table was created to summarize the accuracy of each model. All of the three models achieved accuracy on the test data showing that the selected features were highly predictive for this classification problem.

Model Name	Accuracy	AUC Score	F1-score
Random Forest	99.36%	0.99375%	0: 0.99 1: 0.99
Decision Tree	99.36%	0.99375%	0: 0.99 1: 0.99
Logistic Regression	84.84%	0.8472%	0: 0.86 1: 0.84

**(Note: 0=edible, 1=poisonous)

The final results table summarizes the classification accuracy achieved by each model evaluated in this study. The table compares Random Forest, Decision Tree, and Logistic Regression based on their performance on the test dataset.

The Decision Tree and the Random Forest models showed the highest accuracy of 99.36% which implies that they performed very well in terms of classification. This indicates that the tree based models proved to be very effective in capturing the complex and nonlinear relationships in the data. This could be because the decision rules are able to use categorical features that carry a lot of information.

In contrast, Logistic Regression achieved a lower accuracy of 84.84% compared to other models. This highlights the limitations of a linear model when applied to a dataset with the complex feature interactions. However, Logistic Regression remains an important baseline model because of its simplicity and ease of interpretation.

Overall, the comparison shows that tree-based models performed much better than Logistic Regression for this classification task. Among them, the **Decision Tree** was selected as the final model because of its high accuracy combined with clear interpretability making it suitable for understanding the decision making process of the classifier.

3.3.4 Discussion on Model Performance

Although Random Forest and Decision Tree models achieved perfect accuracy and AUC scores, this outcome should be interpreted cautiously. The mushroom dataset is known to be highly separable, where some categorical features such as odor showing strongly determine whether a mushroom is edible or poisonous. As a result, the tree based models can easily learn almost deterministic patterns resulting in a perfect accuracy on the test sets.

Logistic regression achieved slightly lower performance which may suggest better generalization because of its simpler linear decision boundary. This shows the importance of considering model complexity and generalization ability rather than focusing only on accuracy.

3.3.5 Overfitting, Underfitting and Model Selection

The Decision Tree model was chosen as the preferred model because of its high performance as well as the fact that it is able to outline the decision rules in an understandable and clear way. Even the concept of Decision Trees is normally related to overfitting. The risks have been addressed by testing the model with another test set where the model took a ninety nine percent classification. This shows that the trained decision rules were able to generalize effectively unknown data in this dataset.

The Decision Tree could easily estimate nonlinear relationships among categorical features whereas the Logistic Regression cannot be used to capture such complex features interactions and thus it tends to underfit. Although the accuracy of the Random Forest was also high, the complexity and decreased interpretability of the model made it not be chosen as the final model.

It is possible to state that the Decision Tree model offered the most appropriate combination of the accuracy, simplicity and interpretability and it was an appropriate tool to use the mushroom classification problem.

- Model complexity
- Risk of overfitting
- Interpretability
- Generalization capability

4.0 Testing and Validation

This phase involved the testing and validation of the three trained models: Random Forest, Decision Tree, and Logistic Regression. The performance of each model was evaluated using a confusion matrix to analyze the True Positive, False Positive, True Negative and False Negative. Moreover, the classification report is also generated, producing the precision, recall, F1-score, and accuracy for each model.

4.1 Comparison of Tree-Based Models: Random Forest vs. Decision Tree

From the confusion matrix, both models produced the same performance. The True Positives (TP), representing the number of poisonous mushrooms correctly identified, was approximately 990 for both models. Furthermore, the False Negatives (FN) was zero for both, incorrectly predicted the poisonous mushroom as edible. A value zero indicates that these models are safe to be used and not risking a life.

Both models had 13 instances of False Positives (FP), which predicts the mushroom is poisonous when it is actually edible. However, the impact of the errors in the real-world is minimal because it only removes edible mushrooms. Meanwhile, the True Negatives (TN) value is approximately 1000.

The accuracy for both the Random Forest and Decision Tree models was 99.36%, and f1-score is 0.99. In summary, both tree-based models proved to be accurate and safe for classification task.

4.2 Comparison with the Linear Model: Tree-Based vs. Logistic Regression

The Logistics Regression model demonstrated poor performance by incorrectly classified 200 poisonous mushrooms as edible (False Negatives), contrary to the tree-based models. Thus, this model would lead to life-threatening scenario in real-world.

The True Positive (TP) and True Negatives (TN), both record 790 and 930 which is significantly lower than the tree-tased models. Moreover, its False Positive (FP) value was 110, indicating it was less reliable at classification task.

The accuracy of the Logistic Regression model was 84.84% much lower than the 99.36% achieved by the other two models. The f1-score for poisonous was only 0.86%, proving why the model's failed to identify all the poisonous mushrooms.

This poor performance highlights the limitation of using a linear model like Logistic Regression on a complex, non-linear feature interactions dataset.

5.0 Conclusion

This project successfully developed and evaluated machine learning models for mushroom classification. The primary objective was to create a model capable of differentiating between edible or poisonous mushrooms based on the characteristics. Through a process of data collection and pre-processing, exploratory analysis, model development and evaluation, an effective model successfully developed.

The analysis revealed that the dataset is complex and non-linear which is suitable for tree-based models, Random Forest and Decision Tree. The accuracy achieved from both models is 99.36% by capturing the pattern that linear models cannot. Moreover, these models are safe to use in real-life, since no poisonous mushrooms incorrectly classified as edible mushrooms.

In contrast, the Logistic Regression model was less accurate with 88.84% accuracy and dangerous to implement in real-life scenario because of the 200 poisonous mushrooms wrongly identified as edible mushrooms.

Therefore, the Random Forest and Decision Tree models are the optimal choice with the exceptional accuracy which make them a dependable solution.