Mushroom Poison
Decision Tree Analysis

Heejae Roh

Northeastern University

ALY 6040: Data Mining

Ahmadi Behzad

February 23, 2023

Introduction

In this analysis, I want to analyze whether mushrooms are edible or poisonous through a decision tree. The meaning of this analysis is that it determines the binary factor through various factors. In addition, I think this analysis is more meaningful in that a more accurate analysis is required because even one wrong analysis of mushrooms can lead to fatal results for humans.

R-code is already given, so I could easily write a report just by interpreting it, but I wanted to create a decision tree using python also. I want to study python more because there are cases where I feel that some analysis is more convenient in python (or R).

Mushroom edible vs poison dataset

Mushrooms dataset contains information about class, shape, odor, gill, stalk, veil, ring, spore, population, and habitat. It contains 8,124 observations and 23 variables. It does not include a specific type of mushroom. All variables consist of categorical data.

This dataset includes information about:

- The 'class' is the first variable in this data and the most important data. Whether it is edible or poisonous is indicated by e and p.
- All other variables represent categorical data for shape, odor, gill, stalk, veil, ring, spore, population, and habitat, respectively.

Understanding the Dataset

This dataset is given dataset and information about mushrooms' edible and poisonous status and other factors.

First Look of dataset is shape: (8,124, 23). There are 8,124 observations and 23 variables. Most of variables contains categorical data and the 'class' variable only contains 'e (edible)' and 'p (poisonous)', it could be said that this is binary data. Target variable is 'class' which is most important attribute to human.

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family Mushroom drawn from The Audubon Society Field Guide to North American Mushrooms (1981) (UCI, 2017).

Attribute Information (UCI, 2017)

No.	Feature	Attribute Information
1.	class	edible=e, poisonous=p
2.	cap-shape	bell=b, conical=c, convex=x, flat=f, knobbed=k, sunken=s
3.	cap-surface	fibrous=f, grooves=g, scaly=y, smooth=s
4.	cap-color	brown=n, buff=b, cinnamon=c, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y
5.	bruises	bruises=t, no=f
6.	odor	almond=a, anise=l, creosote=c, fishy=y, foul=f, musty=m, none=n, pungent=p, spicy=s
7.	gill-attachment	attached=a, descending=d, free=f, notched=n
8.	gill-spacing	close=c, crowded=w, distant=d
9.	gill-size	broad=b, narrow=n
10.	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
11.	stalk-shape	enlarging=e, tapering=t
12.	stalk-root	bulbous=b, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r, missing=?
13.	stalk-surface-above-ring	fibrous=f, scaly=y, silky=k, smooth=s
14.	stalk-surface-below-ring	fibrous=f, scaly=y, silky=k, smooth=s
15.	stalk-color-above-ring	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
16.	stalk-color-below-ring	brown=n, buff=b, cinnamon=c, gray=g, orange=o, pink=p, red=e, white=w, yellow=y
17.	veil-type	partial=p, universal=u
18.	veil-color	brown=n, orange=o, white=w, yellow=y
19.	ring-number	none=n, one=o, two=t
20.	ring-type	cobwebby=c, evanescent=e, flaring=f, large=l, none=n, pendant=p, sheathing=s, zone=z
21.	spore-print-color	black=k, brown=n, buff=b, chocolate=h, green=r, orange=o, purple=u, white=w, yellow=y
22.	population	bundant=a, clustered=c, numerous=n, scattered=s, several=v, solitary=y
23.	habitat	grasses=g, leaves=l, meadows=m, paths=p, urban=u, waste=w, woods=d

Unique values and Missing value of Features

No.	Feature	Number of Uniques	Missing_value
1.	class	2	0 (0.00)
2.	cap-shape	6	0 (0.00)
3.	cap-surface	4	0 (0.00)
4.	cap-color	10	0 (0.00)
5.	bruises	2	0 (0.00)
6.	odor	9	0 (0.00)
7.	gill-attachment	2	0 (0.00)
8.	gill-spacing	2	0 (0.00)
9.	gill-size	2	0 (0.00)
10.	gill-color	12	0 (0.00)
11.	stalk-shape	2	0 (0.00)
12.	stalk-root	5	0 (0.00)
13.	stalk-surface-above-ring	4	0 (0.00)
14.	stalk-surface-below-ring	4	0 (0.00)
15.	stalk-color-above-ring	9	0 (0.00)
16.	stalk-color-below-ring	9	0 (0.00)
17.	veil-type	1	0 (0.00)
18.	veil-color	4	0 (0.00)
19.	ring-number	3	0 (0.00)
20.	ring-type	5	0 (0.00)
21.	spore-print-color	9	0 (0.00)
22.	population	6	0 (0.00)
23.	habitat	7	0 (0.00)

Head and Sample of data

df.head(2)

	class	cap (3) -shape	bruises	odor	gill (4) -size	stalk (4) -surface-above-ring
0	p	x	t	p	n	s
1	e	x	t	a	b	s

	veil-color	ring_number	ring_type	spore-print-color	population	habitat
0	w	o	p	k	s	u
1	w	o	p	n	n	g

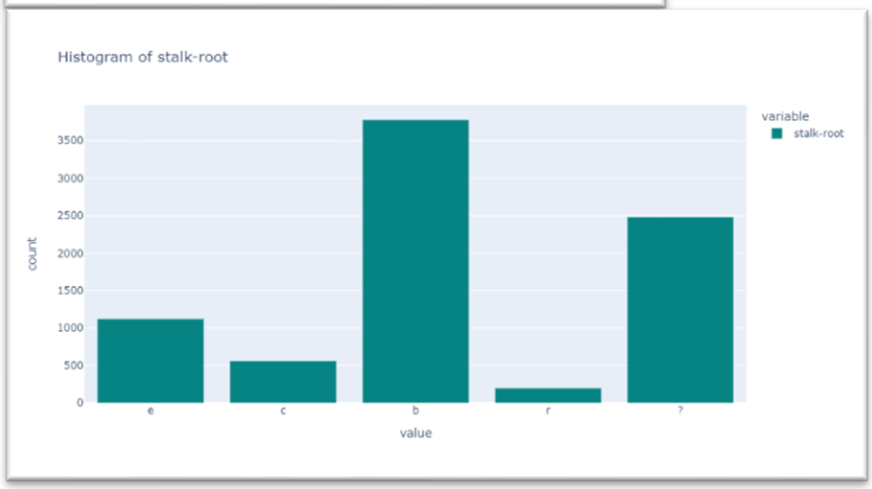
df.sample(3)

	class	cap (3) -shape	bruises	odor	gill (4) -size	stalk (4) -surface-above-ring
7121	p	k	f	y	n	s
4654	p	f	f	f	b	k

	veil-color	ring_number	ring_type	spore-print-color	population	habitat
0	w	o	e	w	v	d
1	w	o	l	h	y	d

Data Cleaning

1. There was no Missing_value, and it was confirmed that the data was clean.
2. I need to make a decision tree through classification. Therefore, cells with only one unique value should be removed.
3. Remove the veil-type identified as having one unique value through the first_looking function. I checked again through the histogram as follows.

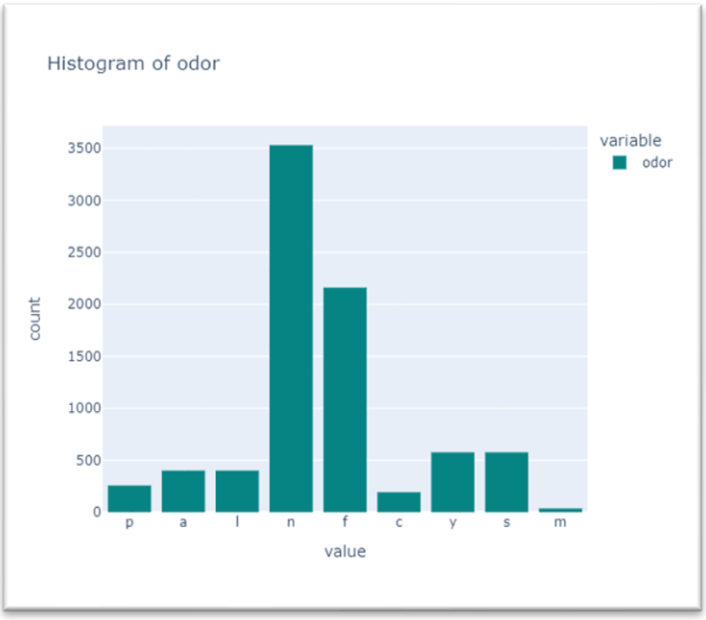


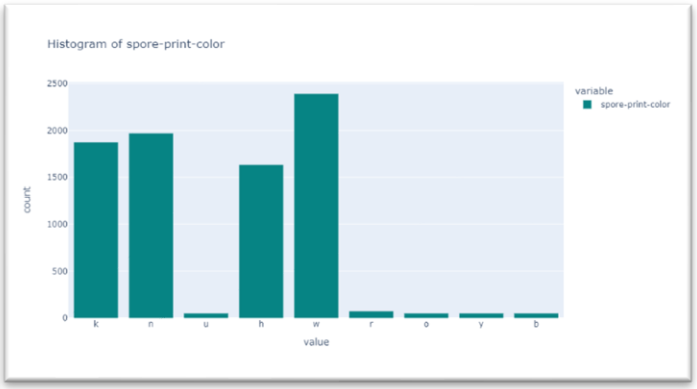
4. Using the function, subset(mushrooms,mushrooms$`stalk-root` == '?'), there are 2,480 '?' in stalk-root variable, but I will analyze it assuming that this is also one of values. And since we have many values as '?' in stalk-root, we will use them to see if we can analyze even though there is a '?'.

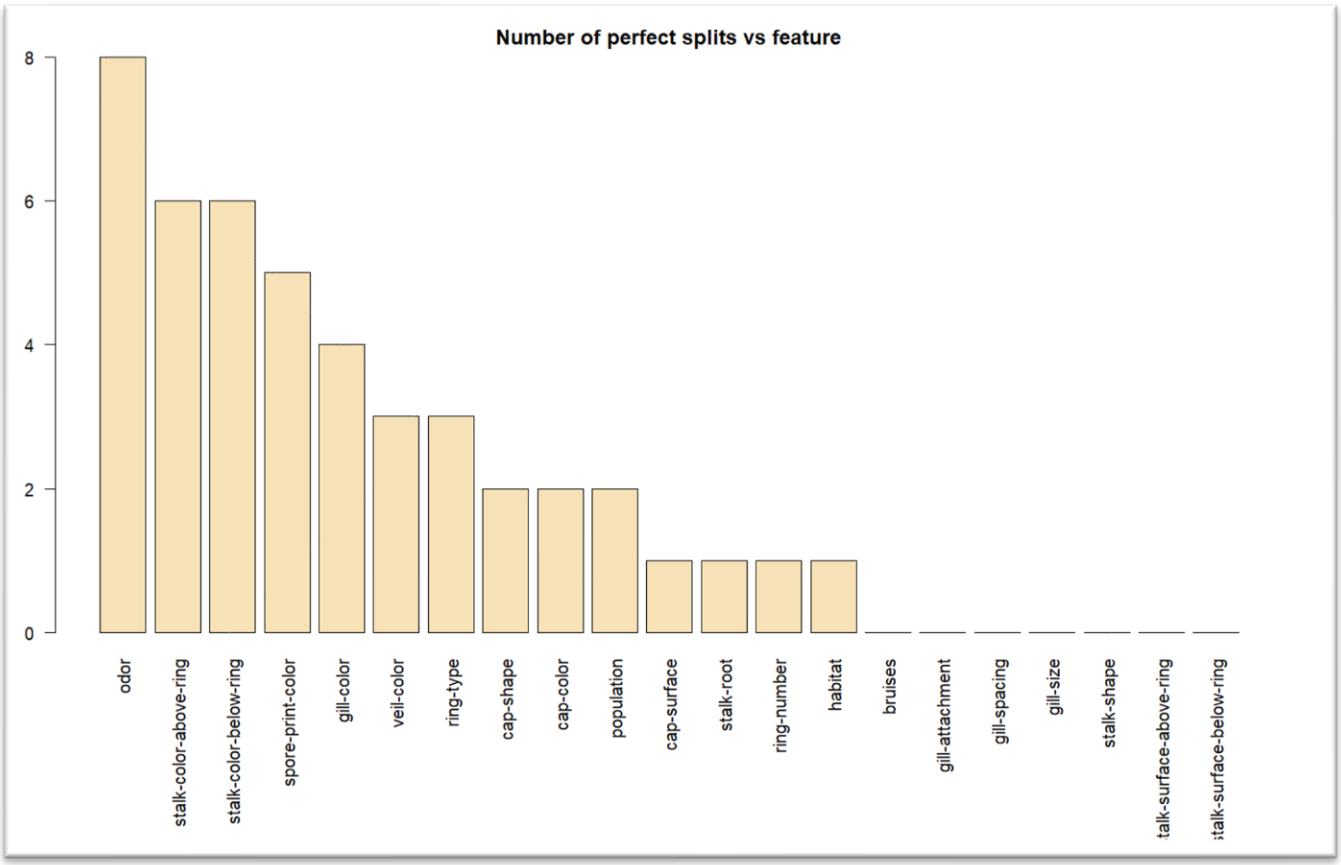
Exploratory Data Analysis:

(1) Descriptive Analysis

Histogram of some variables







Odor x Class contingency Table									
Class \ Odor	a	c	f	l	m	n	p	s	y
edible	400	0	0	400	0	3408	0	0	0
poison	0	192	2160	0	36	120	256	576	576

Interpretation

1. 'Number of perfects splits vs feature' indicates the number of categories that perfectly divide edible and poisonous in the Odor x Class contingency table.
2. For odor, among the 9 unique values examined in EDA, 8 values perfectly divide edible and poisonous. The stalk-color-above-ring has 6 out of 9 unique values, perfectly dividing edible and poisonous. In this way, the number of categories belonging to each variable shows how well we divide edible and poisonous targets.
3. The histogram of the three 'teal' colors above shows how each category variable is configured as a histogram. Odor is mostly occupied by n and f values.

Pre-processing for Decision tree

(1) Split the dataset

Train_test_split(X, y, test_size=0.2, random_state=1)			
X_train.shape	X_test.shape	y_train.shape	y_test.shape
(6499, 116)	(1625, 116)	(6499,)	(1625,)

Interpretation

1. The test_size parameter is set to 0.2, which means that 20% of the data is randomly selected for testing and the remaining 80% is used for training the model.
2. The table shows the output of this code which is the shape of the training and testing sets for both X and Y.
3. The X_train array has a shape of (6499, 117), which means that there are 6499 samples in the training set and each sample has 117 features. The X_test array has a shape of (1625, 117), which means that there are 1625 samples in the testing set and each sample has 117 features.
4. The y_train array has a shape of (6499,), which means that there are 6499 target values in the training set. The y_test array has a shape of (1625,), which means that there are 1625 target values in the testing set.

(2) Check the correlation between each cell

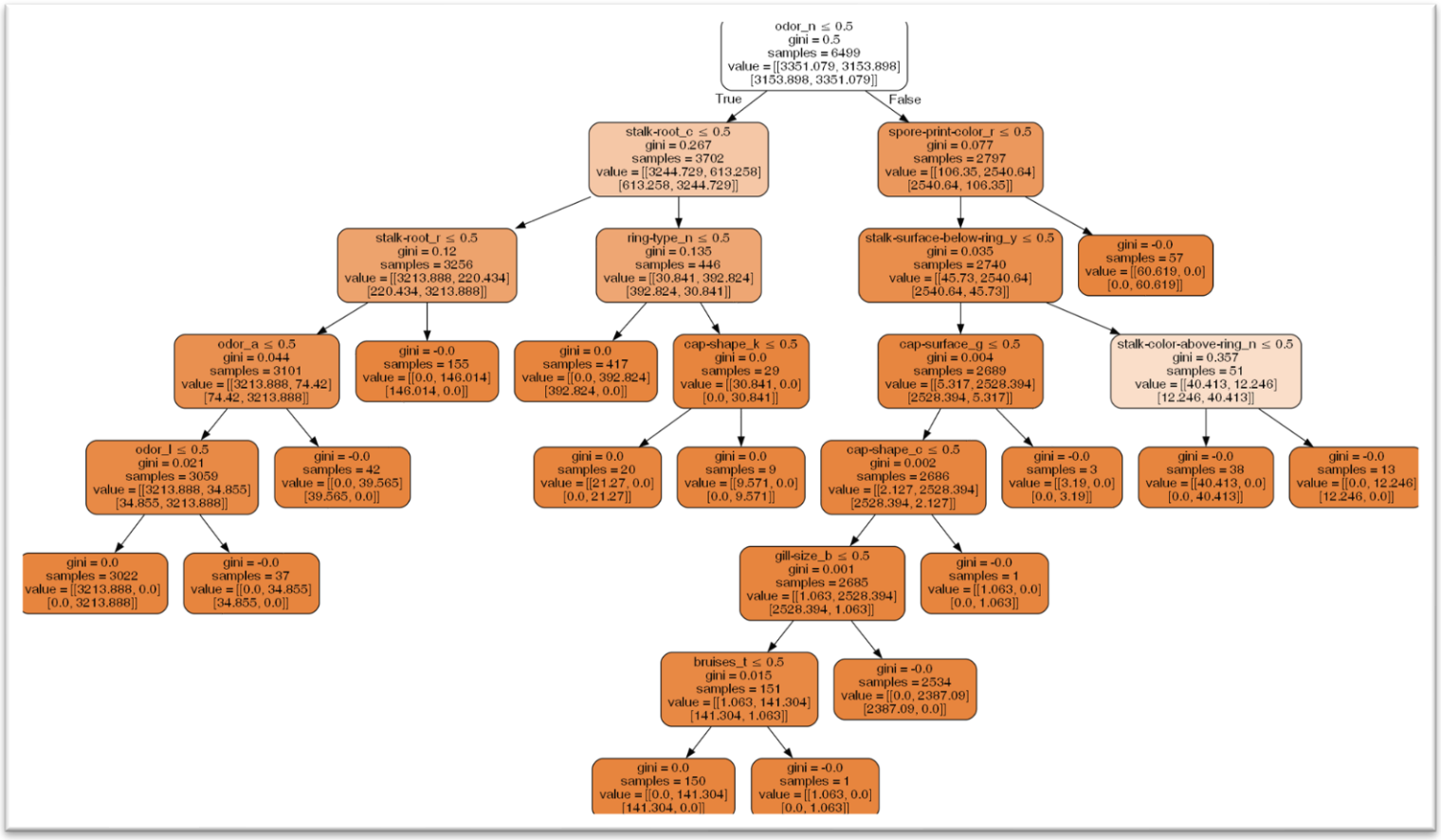
Corr_matrix		
stalk-shape_t	stalk-shape_e	1.000000
gill-spacing_c	gill-spacing_w	1.000000
…	…	
veil-color_w	stalk-color-below-ring_o	0.979302
	stalk-color-above-ring_o	0.979302

Interpretation

1. In general, it is not very meaningful to check the correlation between dummy variables. The reason is that dummy variables are binary variables that only take on the values of 0 or 1.
2. But I wanted to check if there are values in different categories that perfectly match each other. For example, if stalk-shape is t, gill-spacing is both w. Therefore, correlation was used to check if these cells exist.
3. If there are such cells, they can be merged, or one dummy cell can be removed. However, as shown in the table above, cells with two unique values were perfectly correlated with each other in some cases, but not in other cases.

Result of Decision tree

(1) Python result

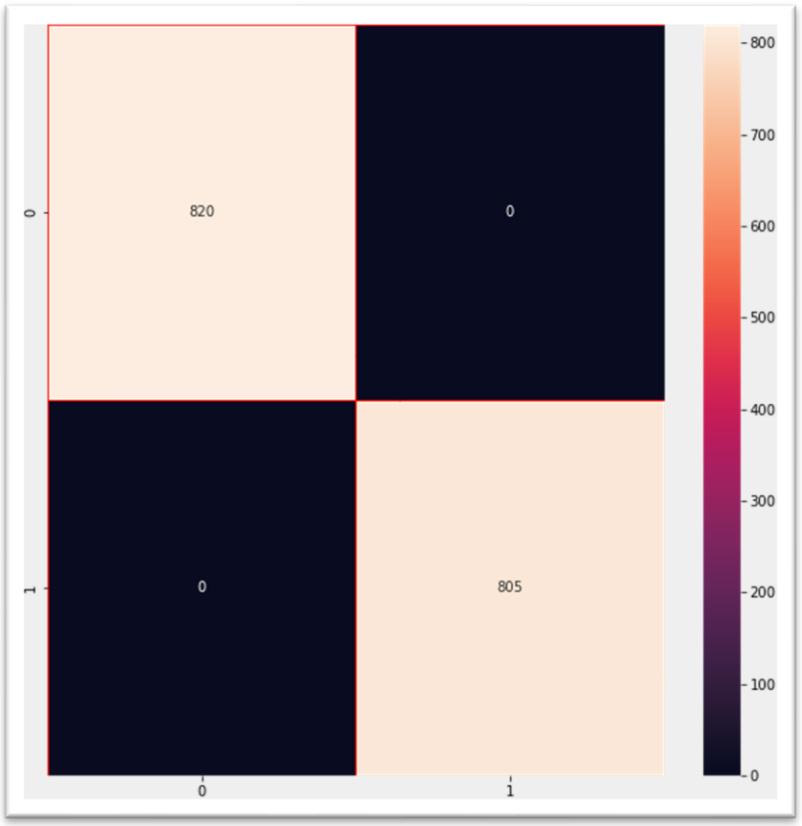


(2) Interpretation of Python result

Interpretation

1. The top of each node indicates what the node classifies. For example, odor_n <= 0.5 in the first node, where True indicates that odor is not n, and False indicates that odor is n.
2. The value of each node represents our target value, edible or poisonous. Edible on the left, poisonous on the right.
3. If the first node is n, the sample goes to the right node, which is 'odor_n <= 0 is False (=1)'. Like this, depending on whether odor is n or not, samples go left and right. The value indicates whether the sample at each node is edible or poisonous.
4. Finally, the node on the right follows the second of the values. And the values A and B, respectively, A means edible and B means poisonous.
5. The node on the left follows the first value of the values. And values A and B are sequentially, A stands for edible and B stands for poisonous.
6. Score of model trained through train data = model.score(X_test, y_test) 100%. Distinguish between 100% edible and poisonous.

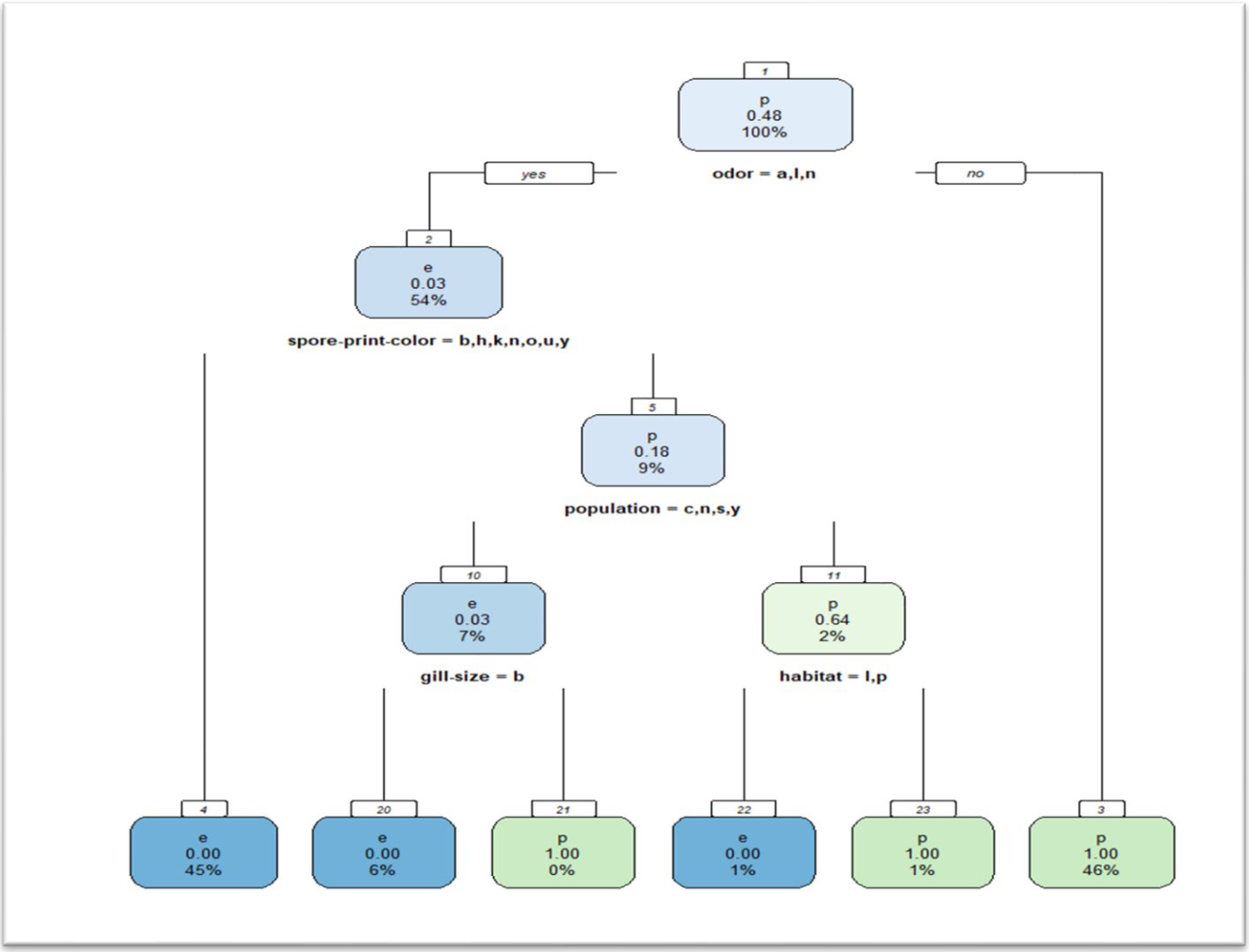
Confusion Matrix of Model



Interpretation

1. The above confusion matrix consists of y_pred_en based on X_test and confusion_matrix based on y_test.
2. The confusion matrix indicates that the model predicted every sample correctly. There were no false positives or false negatives in either class, which means that the model performed perfectly on the test set.

(3) R result



Interpretation

1. This is the result of configuring and running 80% of the train set and 20% of the test set based on R.
2. The confusion matrix indicates that the model predicted every sample correctly. There were no false positives or false negatives in either class, which means that the model performed perfectly on the test set.
3. Unlike Python, you can see that various elements are applied at once. For example, in the first node, odor = a, l, and n are considered at once, and if they are no, they are immediately analyzed as poisonous.
4. Unlike python, which is binary, it can analyze elements with similar results at once, so it is simpler, but it can easily show the result.

Confusion Matrix		
	e	p
e	829	0
p	0	795
	Accuracy	1.00
	Positive class	eidible

Interpretation

1. This is also a confusion matrix based on the test dataset.
2. The confusion matrix indicates that the model predicted every sample correctly. There were no false positives or false negatives in either class, which means that the model performed perfectly on the test set.

Recommendation and Interpretation of whole results

Interpretation

1. This data can be of great value to mushroom pickers. Mushrooms can kill a person depending on the strength of their poison.
2. The confusion matrix indicates that the model predicted every sample correctly. There were no false positives or false negatives in either class, which means that the model performed perfectly on the test set.
3. As explained in the indroduction earlier, this analysis had to aim for 100%, and the decision tree was constructed perfectly 100%.

Recommendation

1. I will use a decision tree using R code because it is simpler to explain to people.
2. We should use that information to create an infographic that tells people which mushrooms are safe and which ones are dangerous.
3. When we create an infographic, we need to write it in more detail so that people can clearly identify the characteristics of the mushroom. For example, odor_n (none) should be more clearly descriptive.
4. And we shouldn't assume that the results we create are always right. Periodic observations should be made of new species of mushroom classifications that may continue to appear.

Conclusion

Through this assignment, I have constructed a decision tree that must satisfy 100%. I also thought about the difference between the two programs while constructing this not only through R-code but also through python. We also learned about interpreting the decision tree in both programs. And based on this, I thought about what recommendations I could actually make about real-world problem. I learn that we should think about the limits our data can always have. Our current model may be perfect, but we recognized the fact that it may not be perfect in the future, and thought we should continue to revise and supplement it.

REFERENCE

UCI MACHINE LEARNING. (2017). Mushroom classification. Kaggle. Retrieved from https://www.kaggle.com/datasets/uciml/mushroom-classification

SANDHYAKRISHNAN02. (2023). Mushroom classification-decision tree classifier. Kaggle. Retrieved from https://www.kaggle.com/code/sandhyakrishnan02/mushroom-classification-decision-tree-classifier

Magdalena, Konkiewicz. (October 23, 2019). Splitting your data to fit any machine learning model. ABOUT DATA BLOG. Retrieved from https://www.aboutdatablog.com/post/splitting-your-data-to-fit-any-machine-learning-model

JNDULI. (2018). Decision tree classifier for mushroom dataset. Kaggle. Retrieved from https://www.kaggle.com/code/jnduli/decision-tree-classifier-for-mushroom-dataset

MIGUELDOMINGUES. (2018). Mushroom classification. Kaggle. Retrieved from https://www.kaggle.com/code/mig555/mushroom-classification

Koolac. (2023). Plot decision tree graph in python sklearn (visualization and interpretation). YouTube. Retrieved from https://www.youtube.com/watch?v=D1ZmxwHAEJA&t=193s

Graphviz. (2023). Download. Retreived from https://graphviz.org/download/

Python-Code

```
# In[2]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn  as sns

# In[3]:
df = pd.read_excel('mushrooms.xlsx')


# In[4]:
df1 = pd.read_excel('mushrooms.xlsx')


# In[5]:
mushrooms = pd.read_excel('mushrooms.xlsx')

# In[6]:
df.head()

# In[7]:
df.sample(10)


# In[8]:
pip install -U pandas-profiling

# In[9]:
import pandas_profiling


# In[10]:
def first_looking(df):
    print(colored("Shape:", attrs=['bold']), df.shape,'\n',
          colored('-'*79, 'red', attrs=['bold']),
          colored("\nInfo:\n", attrs=['bold']), sep='')
    print(df.info(), '\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Number of Uniques:\n", attrs=['bold']),
df.nunique(),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("Missing Values:\n", attrs=['bold']),
```

```
missing_values(df),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')
    print(colored("All Columns:", attrs=['bold']),
list(df.columns),'\n',
          colored('-'*79, 'red', attrs=['bold']), sep='')

#########################################################################
#####

def missing_values(df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent =
(df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
    missing_values = pd.concat([missing_number, missing_percent],
axis=1, keys=['Missing_Number', 'Missing_Percent'])
    return missing_values[missing_values['Missing_Number']>0]

# In[11]:
pip install termcolor

# In[12]:
import colorama
from colorama import Fore, Style  # maakes strings colored
from termcolor import colored

# In[13]:
first_looking(df)

# In[14]:
df.profile_report()

# In[15]:
missing_values(df)

# In[16]:
df.drop("veil-type", axis=1, inplace=True)

# In[17]:
df.head()

# In[18]:
df.sample(3)

# In[19]:
```

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

# In[20]:
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt

# In[21]:
px.histogram(df['class'], color_discrete_sequence=['purple'],
title="Histogram of Edible vs Poisonous")

# In[22]:
px.histogram(df['cap-shape'], color_discrete_sequence=['teal'],
title="Histogram of cap-shape")

# In[23]:
px.histogram(df['bruises'], color_discrete_sequence=['teal'],
title="Histogram of bruises")

# In[24]:
px.histogram(df['odor'], color_discrete_sequence=['teal'],
title="Histogram of odor")

# In[25]:
px.histogram(df['stalk-root'], color_discrete_sequence=['teal'],
title="Histogram of stalk-root")

# In[26]:
px.histogram(df['spore-print-color'], color_discrete_sequence=['teal'],
title="Histogram of spore-print-color")

# In[27]:
px.histogram(df1['veil-type'], color_discrete_sequence=['teal'],
title="veil-type")


# In[28]:
X = df.drop(['class'],axis=1)
y = df['class']

# In[29]:
X = pd.get_dummies(X)
X.head()
```

```
# In[30]:
corr_matrix = X.corr()
print(corr_matrix)


# In[31]:
# Sort the correlation matrix by absolute value of correlation
coefficients
corr_matrix_abs = corr_matrix.abs()
sorted_corr = corr_matrix_abs.unstack().sort_values(kind="quicksort",
ascending=False)

# Print the top 10 most highly correlated pairs of dummy variables
top_pairs = sorted_corr[sorted_corr != 1.0].head(25)
print(top_pairs)

# In[32]:
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
y = encoder.fit_transform(y)
print(y)


# In[33]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=1)

# In[34]:
X_train.shape , X_test.shape

# In[35]:
y_train.shape , y_test.shape

# In[36]:
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree


# In[37]:
from sklearn.metrics import accuracy_score

# In[38]:
#Using the Decision Tree Classifier with splitting criterion as Gini
```

```
impurity, the maximum depth of the tree is 3.
clf_gini = DecisionTreeClassifier(criterion='gini', random_state=0)

# fit the model
clf_gini.fit(X_train, y_train)


# In[39]:
#Plot the tree
plt.figure(figsize=(12,8))
tree.plot_tree(clf_gini.fit(X_train, y_train))

# In[40]:
pip install graphviz

# In[41]:
#Predict the values
y_pred_gini = clf_gini.predict(X_test)

# In[42]:
#Predict the value using X train for accuracy comparision
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
# In[43]:

#Determine the accuracy score
print('Model accuracy score with criterion gini index: {0:0.4f}'.
format(accuracy_score(y_test, y_pred_gini)))
#Accuracy Score for training set
print('Training-set accuracy score: {0:0.4f}'.
format(accuracy_score(y_train, y_pred_train_gini)))

# In[44]:
clf_en = DecisionTreeClassifier(criterion='gini', random_state=0)


# fit the model
clf_en.fit(X_train, y_train)


# In[45]:
plt.figure(figsize=(12,8))
tree.plot_tree(clf_en.fit(X_train, y_train))
```

```
# In[46]:
#Predict the values
y_pred_en = clf_en.predict(X_test)

# In[47]:
#Predict the value using X train for accuracy comparision
y_pred_train_en = clf_en.predict(X_train)

# In[48]:
print('Model accuracy score with criterion entropy: {0:0.4f}'.
format(accuracy_score(y_test, y_pred_en)))
print('Training-set accuracy score: {0:0.4f}'.
format(accuracy_score(y_train, y_pred_train_en)))

# In[49]:
print('Training set score: {:.4f}'.format(clf_en.score(X_train,
y_train)))
print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))

# In[50]:
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import  f1_score

# In[51]:
cm = confusion_matrix(y_test, y_pred_en)
print('Confusion matrix\n\n', cm)

# In[52]:
f,ax = plt.subplots(figsize=(10, 10))
sns.heatmap(cm, annot=True, linewidths=0.5,linecolor="red", fmt=
'.0f',ax=ax)
plt.show()
plt.savefig('ConfusionMatrix.png')

# In[53]:
print(classification_report(y_test, y_pred_en))

# In[54]:
f1_score = f1_score(y_test, y_pred_en)
print("F1 Score:",f1_score)

# In[55]:
```

```
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)
feature_names = X_train.columns.tolist()

# Find the meaning of X[27]
print("X[27] corresponds to the feature:", feature_names[27])


# In[56]:
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)

feature_names = X_train.columns.tolist()
# Find the meaning of X[53]
print("X[53] corresponds to the feature:", feature_names[53])


# In[57]:
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)

feature_names = X_train.columns.tolist()

# Find the meaning of X[99]
print("X[99] corresponds to the feature:", feature_names[99])

# In[58]:
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)

feature_names = X_train.columns.tolist()

# Find the meaning of X[55]
print("X[55] corresponds to the feature:", feature_names[55])

# In[59]:
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)

feature_names = X_train.columns.tolist()
```

```
# Find the meaning of X[80]
print("X[80] corresponds to the feature:", feature_names[80])

# In[60]:
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)

feature_names = X_train.columns.tolist()


# Find the meaning of X[63]
print("X[63] corresponds to the feature:", feature_names[63])
# In[61]:
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3,
random_state=0)
clf_gini.fit(X_train, y_train)

feature_names = X_train.columns.tolist()
# Find the meaning of X[53]
print("X[53] corresponds to the feature:", feature_names[53])

# In[62]:
df1.head()

# In[63]:
# Create the contingency table
contingency_table = pd.crosstab(df1['class'], df1['odor'])

# Print the contingency table
print(contingency_table)

# In[64]:
# Create the contingency table
contingency_table = pd.crosstab(df1['class'], col)

# Print the contingency table
print(contingency_table)


# In[65]:
df = pd.read_excel('mushrooms.xlsx')


# In[66]:
```

```
from sklearn.model_selection import train_test_split

# split the data int x(training data) and y (results)
y = df['class']
x = df.drop(['class'], axis=1)
x = pd.get_dummies(x)
y = pd.get_dummies(y)
x.info()
y.info()
# x.info()
# y.info()
# x.dtypes


# In[67]:
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
# df['habitat_cat'] = df['habitat'].cat.codes
# df['class_cat'] = df['class'].cat.codes
# df.dtypes
# df['habitat_cat'].unique()
# sns.stripplot(x='class', y='habitat_cat', data=df, jitter=True)

# In[83]:
from sklearn.model_selection import cross_val_score
from sklearn import tree
from sklearn.model_selection import GridSearchCV

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=2)

parameters = {'criterion':('gini', 'entropy'),
              'min_samples_split':[2,3,4,5],
              'max_depth':[7,8,9],
              'class_weight':('balanced', None)
             }
tr = tree.DecisionTreeClassifier()
gsearch = GridSearchCV(tr, parameters)
gsearch.fit(X_train, y_train)
model = gsearch.best_estimator_
model

# In[84]:
```

```
# The scores are really great, so fit the model and predict
# model.fit(X_train, y_train)
score = model.score(X_test, y_test)
score

# In[85]:
import graphviz
dot_data = tree.export_graphviz(model, out_file=None,
                                feature_names=X_test.columns,
                                class_names=y_test.columns,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.format = 'png'

filename = 'decision_tree.png'
graph.render(filename)

graph

# In[88]:
print(X_train)

# In[96]:
from sklearn.preprocessing import label_binarize

# Assuming y_test is a multilabel target variable
y_test_binary = label_binarize(y_test, classes=[0, 1, 2])

# Now you can pass y_test_binary to the confusion matrix function
y_pred_binary = label_binarize(y_pred, classes=[0, 1, 2])
cm = confusion_matrix(y_test_binary, y_pred_binary)
print('Confusion matrix:\n', cm)

# In[98]:

from sklearn.metrics import precision_score, recall_score, f1_score

# Assuming y_test is a multilabel target variable
y_pred_binary = (y_pred > 0.5).astype(int)
precision = precision_score(y_test, y_pred_binary, average='micro')
recall = recall_score(y_test, y_pred_binary, average='micro')
f1 = f1_score(y_test, y_pred_binary, average='micro')
```

```
print('Precision:', precision)
print('Recall:', recall)
print('F1-score:', f1)
```

R-Code

```
#Installing libraries
install.packages('rattle')

#Loading libraries
library(rpart,quietly = TRUE)
library(caret,quietly = TRUE)
library(rpart.plot,quietly = TRUE)
library(rattle)
library(readxl)
library(psych)
library(DataExplorer)

#Reading the data set as a dataframe
setwd("C:\\Users\\14083\\Desktop")
mushrooms <- read_excel("mushrooms.xlsx")


# structure of the data
str(mushrooms)
describe(mushrooms)

# (added) profiling
create_report(mushrooms)

# number of rows with missing values
nrow(mushrooms) - sum(complete.cases(mushrooms))

# deleting redundant variable `veil.type`
mushrooms$`veil-type` <- NULL

# checking the missing value
subset(mushrooms,mushrooms$`stalk-root` == '?')

#analyzing the odor variable
table(mushrooms$class,mushrooms$odor)

number.perfect.splits <- apply(X=mushrooms[-1], MARGIN = 2, FUN =
```

```r
function(col){
  t <- table(mushrooms$class,col)
  sum(t == 0)
})

# Descending order of perfect splits
order <- order(number.perfect.splits,decreasing = TRUE)
number.perfect.splits <- number.perfect.splits[order]

# Plot graph
par(mar=c(10,2,2,2))
barplot(number.perfect.splits,
        main="Number of perfect splits vs feature",
        xlab="",ylab="Feature",las=2,col="wheat")


#data splicing
set.seed(12345)
train <- sample(1:nrow(mushrooms),size =
ceiling(0.80*nrow(mushrooms)),replace = FALSE)
# training set
mushrooms_train <- mushrooms[train,]
# test set
mushrooms_test <- mushrooms[-train,]


# penalty matrix
penalty.matrix <- matrix(c(0,1,10,0), byrow=TRUE, nrow=2)


# building the classification tree with rpart
tree <- rpart(class~.,
              data=mushrooms_train,
              parms = list(loss = penalty.matrix),
              method = "class")


# Visualize the decision tree with rpart.plot
rpart.plot(tree, nn=TRUE)


# choosing the best complexity parameter "cp" to prune the tree
cp.optim <- tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
# tree prunning using the best complexity parameter. For more in
```

```
tree <- prune(tree, cp=cp.optim)


#Testing the model
pred <- predict(object=tree,mushrooms_test[-1],type="class")


#Calculating accuracy
t <- table(mushrooms_test$class,pred)
confusionMatrix(t)
```