HW2

summary report for the mushroom dataset

Tsung-Jiun Huang

2025-03-17

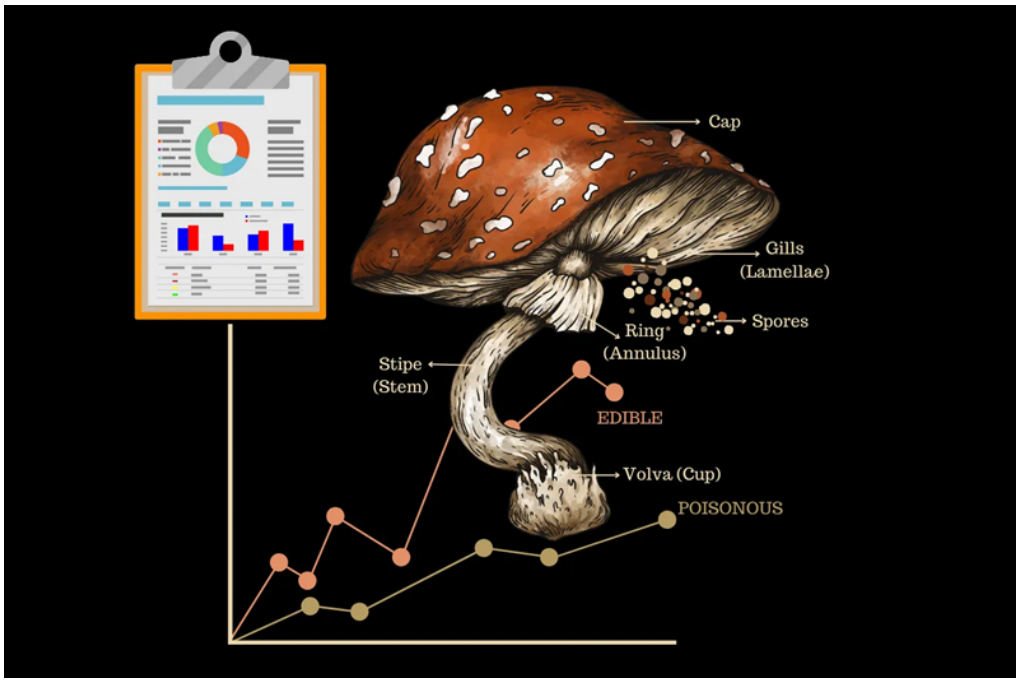
目錄

一、Data Dictionary

表 1: Mushroom Dataset Data Dictionary

Variable	DataType	Definition	Note
family	String	Name of the family of mushroom species	Multinomial
name	String	Mushroom species name	Multinomial
class	Binary	poisonous=p, edibile=e	Binary
cap-diameter	Float, Metric data	Cap diameter in cm	[Min, max] or mean
cap-shape	Nominal data	Shape of the cap	bell=b, conical=c, convex=x, flat=f, sunken=s, spherical=p, others=o
cap-surface	Nominal data	Surface type of the cap	fibrous=i, grooves=g, scaly=y, smooth=s, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e
cap-color	Nominal data	Color of the cap	brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k
does-bruise-bleed	Nominal data	Bruising or bleeding	t=yes, f=no
gill-attachment	Nominal data	Attachment of the gills	adnate=a, adnexed=x, decurrent=d, free=e, sinuate=s, pores=p, none=f, unknown=?

Variable	DataType	Definition	Note
gill-spacing	Nominal data	Spacing between gills	close=c, distant=d, none=f
gill-color	Nominal data	Color of the gills	brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k, none=f
stem-height	Float, Metric data	Height of the stem in cm	[Min, max] or mean
stem-width	Float, Metric data	Width of the stem in mm	[Min, max] or mean
stem-root	Nominal data	Root type of the stem	bulbous=b, swollen=s, club=c, cup=u, equal=e, rhizomorphs=z, rooted=r
stem-surface	Nominal data	Surface type of the stem	fibrous=i, grooves=g, scaly=y, smooth=s, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e, none=f
stem-color	Nominal data	Color of the stem	brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k, none=f
veil-type	Nominal data	Type of veil	p=partial, u=universal
veil-color	Nominal data	Color of the veil	brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k, none=f
has-ring	Nominal data	Presence of a ring	t=yes, f=no
ring-type	Nominal data	Type of ring	cobwebby=c, evanescent=e, flaring=r, grooved=g, large=l, pendant=p, sheathing=s, zone=z, scaly=y, movable=m, none=f, unknown=?
spore-print-color	Nominal data	Color of spore-print	brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o, black=k
habitat	Nominal data	Habitat type	grasses=g, leaves=l, meadows=m, paths=p, heaths=h, urban=u, waste=w, woods=d
season	Nominal data	Season of occurrence	spring=s, summer=u, autumn=a, winter=w



二、讀取資料

```
# R Interface to Python
library(reticulate)                # Make R and Python interoperable, allowing R to call Python code.
use_python("C:/Users/user/anaconda3/python.exe", required = TRUE)  # Finding Anaconda's Python path
# py_config()

library(Hmisc)                     # data analysis and report tools
library(ggplot2)                   # a system for creating graphics
library(tableone)                  # a tool for creating tableone
```

```
# !pip install tableone
```

```
import re  # Regular expressions (text processing)
import numpy as np  # Numerical computing
import pandas as pd  # Data analysis
import seaborn as sns  # Data visualization
import tensorflow as tf  # Deep learning framework
from tensorflow import keras  # High-level API for TensorFlow
from tableone import TableOne  # Summary tables for data analysis
import matplotlib.pyplot as plt  # Plotting library
from tensorflow.keras import layers, regularizers  # Keras layers & regularization
from sklearn.model_selection import train_test_split  # Split dataset
from sklearn.preprocessing import LabelEncoder  # Encode categorical data
from sklearn.ensemble import RandomForestClassifier  # Machine learning classifier
from sklearn.preprocessing import StandardScaler  # Standardize features
from sklearn.metrics import accuracy_score, classification_report  # Model evaluation
from tensorflow.keras import backend as K  # Low-level TensorFlow operations
```

三、資料前處理-轉換為結構化資料

```python
# read CSV
file_path = "C:/Users/user/Downloads/primary_data.csv"

with open(file_path, "r", encoding="utf-8") as file:
    raw_lines = file.readlines()

# get the feature name
header = raw_lines[0].strip().split(";")  # the first row

# processing data
data = []
for line in raw_lines[1:]:
    values = line.strip().split(";")  # spilt by `;`
    if len(values) != len(header):
        values += [""] * (len(header) - len(values))

    # handing `[]` let [10 20] to '10, 20'
    cleaned_values = [re.sub(r"\[|\]", "", v).replace("\t", " ") for v in values]
    data.append(cleaned_values)

# make a DataFrame
df1 = pd.DataFrame(data, columns=header)
```

四、資料描述

```r
# read dataset
df <- read.csv("C:/Users/user/Downloads/primary_data_cleaned.csv")
# data description
latex(describe(df), descript = "descriptive statistics", file = '', caption.placement = 'top')
```

df
23 Variables 173 Observations

family

n	missing	distinct
173	0	23

lowest : Amanita Family Bolbitius Family Bolete Family Bracket Fungi Chanterelle Family
highest: Russula Family Saddle-Cup Family Stropharia Family Tricholoma Family Wax Gill Family

name

n	missing	distinct
173	0	173

lowest : Amethyst Deceiver Aniseed Funnel Cap Apricot Fungus Bare-toothed Russula Bay Bolete
highest: Yellow-gilled Russula Yellow-staining Mushroom Yellow-stemmed Bell Cap Yellow Swamp Russula Yellow Wax cap

class

n	missing	distinct
173	0	2

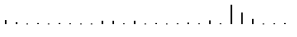
Value	e	p
Frequency	77	96
Proportion	0.445	0.555

cap.diameter

n	missing	distinct
173	0	51

```
lowest : 0.4, 1   0.5, 1   0.5, 1.5 0.7, 1.3 1, 1.5  , highest: 8, 14    8, 15    8, 20    8, 25    8, 30
```

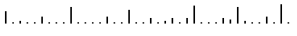
cap.shape



n	missing	distinct
173	0	27

```
lowest : b       b, f    b, f, s b, x    b, x, f, highest: x, f    x, f, s x, o    x, p    x, s
```

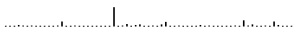
Cap.surface



n	missing	distinct
133	40	40

```
lowest : d          d, e, y, i d, k       d, k, s    d, s
highest: t, w, d    w          w, t       y          y, s
```

cap.color



n	missing	distinct
173	0	67

```
lowest : b            b, p, e, y    b, u          e            e, n
highest: y            y, n          y, o          y, o, g, n, r y, o, r, n
```

does.bruise.or.bleed

n	missing	distinct
173	0	2

Value	f	t
Frequency	143	30
Proportion	0.827	0.173

gill_attachment

n	missing	distinct
145	28	8

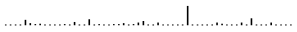
Value	a	a, d	d	e	f	p	s	x
Frequency	32	8	25	16	10	17	16	21
Proportion	0.221	0.055	0.172	0.110	0.069	0.117	0.110	0.145

gill_spacing

n	missing	distinct
102	71	3

Value	c	d	f
Frequency	70	22	10
Proportion	0.686	0.216	0.098

gill.color



n	missing	distinct
173	0	59

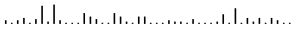
```
lowest : b       b, p, w b, u    e       f       , highest: y, n    y, o, e y, r    y, r, k y, w
```

stem.height

n	missing	distinct
173	0	46

```
lowest : 0      1, 2   1, 3   10, 12 10, 15, highest: 8, 12  8, 15  8, 20  8, 25  8, 30
```

stem.width



n	missing	distinct
173	0	48

```
lowest : 0      0.5, 1 1      1, 2   1, 3  , highest: 7, 15  8, 12  8, 15  8, 18  8, 20
```

stem_root

n	missing	distinct
27	146	5

Value	b	c	f	r	s
Frequency	9	2	3	4	9
Proportion	0.333	0.074	0.111	0.148	0.333

stem.surface

n	missing	distinct
65	108	14

Value	f	g	h	i	i, s	i, t	i, y	k	k, s	s	s, h	t	y	y, s
Frequency	3	5	1	11	1	1	1	4	1	15	1	7	13	1
Proportion	0.046	0.077	0.015	0.169	0.015	0.015	0.015	0.062	0.015	0.231	0.015	0.108	0.200	0.015

stem.color

n	missing	distinct
173	0	41

lowest : b, u e e, n e, u, y e, y , highest: w, y y y, e, n y, n y, o, k

veil.type

n	missing	distinct	value
9	164	1	u

Value	u
Frequency	9
Proportion	1

veil.color

n	missing	distinct
21	152	7

Value	e, n	k	n	u	w	y	y, w
Frequency	1	1	1	1	15	1	1
Proportion	0.048	0.048	0.048	0.048	0.714	0.048	0.048

has.ring

n	missing	distinct
173	0	2

Value	f	t
Frequency	130	43
Proportion	0.751	0.249

ring.type

n	missing	distinct
166	7	13

Value	e	e, g	f	g	g, p	l	l, e	l, p	l, r	m	p	r	z
Frequency	6	1	137	2	2	2	1	1	2	1	2	3	6
Proportion	0.036	0.006	0.825	0.012	0.012	0.012	0.006	0.006	0.012	0.006	0.012	0.018	0.036

Spore.print.color

n	missing	distinct
18	155	8

Value	g	k	k, r	k, u	n	p	p, w	w
Frequency	1	5	1	1	3	3	1	3
Proportion	0.056	0.278	0.056	0.056	0.167	0.167	0.056	0.167

habitat

n	missing	distinct
173	0	21

lowest : d d, h g g, d g, d, h, highest: m m, d m, h p, d w

season

n	missing	distinct
173	0	10

Value	a	a, w	s	s, a, w	s, u	s, u, a	s, u, a, w
Frequency	16	15	1	1	3	5	13
Proportion	0.092	0.087	0.006	0.006	0.017	0.029	0.075

Value	u	u, a	u, a, w
Frequency	1	106	12
Proportion	0.006	0.613	0.069

Through analysis, it was found that there are 23 different types of families, the most common of which is Tricholoma Family, followed by Russula Family. There are 2 different families in class, of which poisonous (p) accounts for 55.5% and edible (e) accounts for 44.5%. cap.diameter has 51 different diameters, 50 of which have different maximum and minimum values, and one that is expressed as an average. The largest diameter is in the range of 2 to 5, followed by 10 to 15. There are 27 different cap shapes in cap-shape, the most common one is (convex = x) with 48, followed by (convex = x, flat = f) with 29.

五、tableone

```
import pandas as pd
import numpy as np
from tableone import TableOne

df = pd.read_csv("C:/Users/user/Downloads/primary_data_cleaned.csv")
# define the column in table one
columns = [
    'class',          #
    'family',         #
    'cap-diameter',   #
    'stem-height',    #
    'stem-width',     #
    'cap-shape',      #
    'Cap-surface',    #
    'cap-color',      #
    'gill_spacing',   #
    'gill-color',     #
    'stem-color',     #
    'habitat',        #
    'season'          #
]

# define the continuous feature
continuous = ['cap-diameter', 'stem-height', 'stem-width']

# define the categorical feature
categorical = [
    'family', 'cap-shape', 'Cap-surface', 'cap-color',
    'gill_spacing', 'gill-color', 'stem-color', 'habitat', 'season'
]
groupby = 'class'

# processing the value
def convert_range_to_median(value):
```

```
    if pd.isna(value):
        return np.nan
    if isinstance(value, str) and ',' in value:
        try:
            values = [float(x.strip()) for x in value.split(',')]
            return np.median(values)
        except:
            return np.nan
    try:
        return float(value)
    except:
        return np.nan

# let data convert to median
for col in continuous:
    df[col] = df[col].apply(convert_range_to_median)

# create Table 1
table = TableOne(
    df, columns=columns, categorical=categorical, continuous=continuous,
    groupby=groupby, missing=True, decimals=2
)
```

```
library(table1)
df <- read.csv("C:/Users/user/Downloads/primary_data_cleaned.csv")
table1(~ family+gill_attachment+gill_spacing+stem_root+habitat+season| class,data=df)
```

	e	p	Overall
	(N=77)	(N=96)	(N=173)
family			
Amanita Family	3 (3.9%)	5 (5.2%)	8 (4.6%)
Bolbitius Family	1 (1.3%)	2 (2.1%)	3 (1.7%)
Bolete Family	11 (14.3%)	3 (3.1%)	14 (8.1%)
Bracket Fungi	1 (1.3%)	6 (6.3%)	7 (4.0%)
Chanterelle Family	3 (3.9%)	0 (0%)	3 (1.7%)
Entoloma Family	1 (1.3%)	6 (6.3%)	7 (4.0%)
Hydnum Family	1 (1.3%)	0 (0%)	1 (0.6%)
Ink Cap Family	6 (7.8%)	7 (7.3%)	13 (7.5%)
Lepiota Family	2 (2.6%)	1 (1.0%)	3 (1.7%)
Morel Family	1 (1.3%)	0 (0%)	1 (0.6%)
Mushroom Family	4 (5.2%)	1 (1.0%)	5 (2.9%)
Oyster Mushroom Family	2 (2.6%)	0 (0%)	2 (1.2%)
Pluteus Family	2 (2.6%)	0 (0%)	2 (1.2%)
Russula Family	11 (14.3%)	16 (16.7%)	27 (15.6%)
Stropharia Family	1 (1.3%)	7 (7.3%)	8 (4.6%)
Tricholoma Family	23 (29.9%)	20 (20.8%)	43 (24.9%)
Wax Gill Family	4 (5.2%)	4 (4.2%)	8 (4.6%)
Cortinarius Family	0 (0%)	11 (11.5%)	11 (6.4%)
Crepidotus Family	0 (0%)	1 (1.0%)	1 (0.6%)
Ear-Pick Family	0 (0%)	1 (1.0%)	1 (0.6%)
Jelly Discs Family	0 (0%)	1 (1.0%)	1 (0.6%)
Paxillus Family	0 (0%)	3 (3.1%)	3 (1.7%)

	e	p	Overall
Saddle-Cup Family	0 (0%)	1 (1.0%)	1 (0.6%)
gill_attachment			
	10 (13.0%)	18 (18.8%)	28 (16.2%)
a	11 (14.3%)	21 (21.9%)	32 (18.5%)
a, d	5 (6.5%)	3 (3.1%)	8 (4.6%)
d	9 (11.7%)	16 (16.7%)	25 (14.5%)
e	10 (13.0%)	6 (6.3%)	16 (9.2%)
f	4 (5.2%)	6 (6.3%)	10 (5.8%)
p	12 (15.6%)	5 (5.2%)	17 (9.8%)
s	7 (9.1%)	9 (9.4%)	16 (9.2%)
x	9 (11.7%)	12 (12.5%)	21 (12.1%)
gill_spacing			
	31 (40.3%)	40 (41.7%)	71 (41.0%)
c	29 (37.7%)	41 (42.7%)	70 (40.5%)
d	13 (16.9%)	9 (9.4%)	22 (12.7%)
f	4 (5.2%)	6 (6.3%)	10 (5.8%)
stem_root			
	67 (87.0%)	79 (82.3%)	146 (84.4%)
b	6 (7.8%)	3 (3.1%)	9 (5.2%)
s	4 (5.2%)	5 (5.2%)	9 (5.2%)
c	0 (0%)	2 (2.1%)	2 (1.2%)
f	0 (0%)	3 (3.1%)	3 (1.7%)
r	0 (0%)	4 (4.2%)	4 (2.3%)
habitat			
d	47 (61.0%)	57 (59.4%)	104 (60.1%)
d, h	1 (1.3%)	3 (3.1%)	4 (2.3%)
g	1 (1.3%)	10 (10.4%)	11 (6.4%)
g, d	6 (7.8%)	4 (4.2%)	10 (5.8%)
g, d, h	1 (1.3%)	0 (0%)	1 (0.6%)
g, h, d	1 (1.3%)	2 (2.1%)	3 (1.7%)
g, l, m, d	1 (1.3%)	0 (0%)	1 (0.6%)
g, m	3 (3.9%)	2 (2.1%)	5 (2.9%)
g, m, d	1 (1.3%)	4 (4.2%)	5 (2.9%)
g, u, d	1 (1.3%)	0 (0%)	1 (0.6%)
l	1 (1.3%)	0 (0%)	1 (0.6%)
l, d	7 (9.1%)	6 (6.3%)	13 (7.5%)
l, d, h	1 (1.3%)	0 (0%)	1 (0.6%)
l, h	1 (1.3%)	0 (0%)	1 (0.6%)
m	1 (1.3%)	1 (1.0%)	2 (1.2%)
m, d	2 (2.6%)	1 (1.0%)	3 (1.7%)
w	1 (1.3%)	0 (0%)	1 (0.6%)
g, l, d	0 (0%)	1 (1.0%)	1 (0.6%)
h, d	0 (0%)	2 (2.1%)	2 (1.2%)
m, h	0 (0%)	1 (1.0%)	1 (0.6%)
p, d	0 (0%)	2 (2.1%)	2 (1.2%)
season			
a	5 (6.5%)	11 (11.5%)	16 (9.2%)
a, w	9 (11.7%)	6 (6.3%)	15 (8.7%)
s	1 (1.3%)	0 (0%)	1 (0.6%)
s, a, w	1 (1.3%)	0 (0%)	1 (0.6%)
s, u	2 (2.6%)	1 (1.0%)	3 (1.7%)

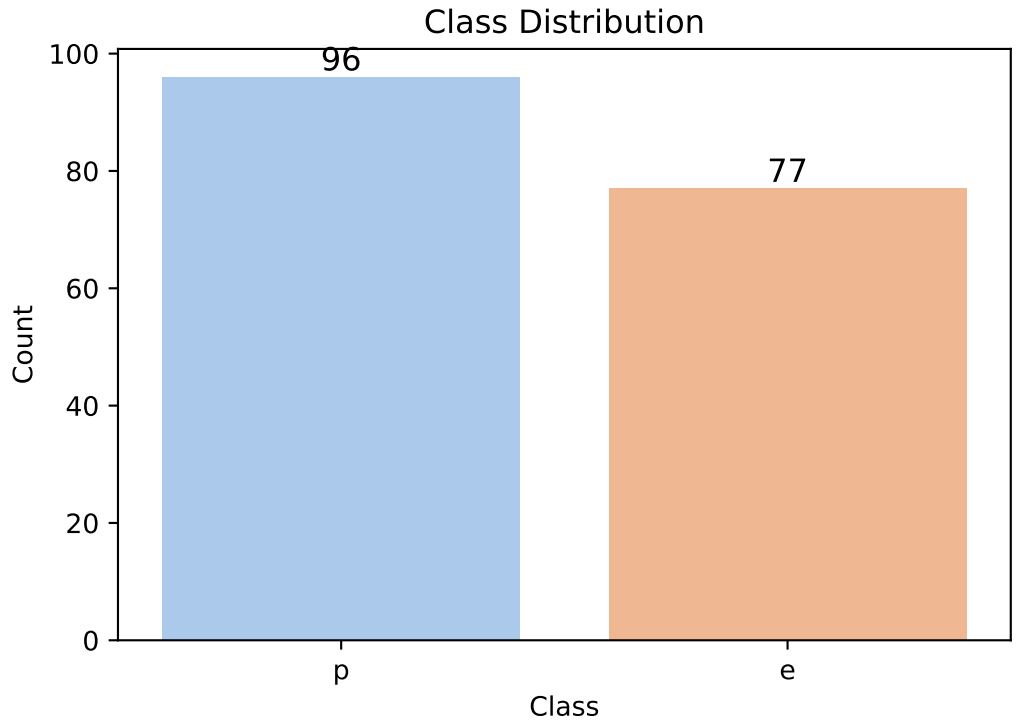
	e	p	Overall
s, u, a	1 (1.3%)	4 (4.2%)	5 (2.9%)
s, u, a, w	7 (9.1%)	6 (6.3%)	13 (7.5%)
u, a	43 (55.8%)	63 (65.6%)	106 (61.3%)
u, a, w	8 (10.4%)	4 (4.2%)	12 (6.9%)
u	0 (0%)	1 (1.0%)	1 (0.6%)

六、視覺化圖表分析

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# read CSV
df = pd.read_csv("C:/Users/user/Downloads/primary_data_cleaned.csv")

# print class's counts
print(df['class'].value_counts())
```

```
class
p    96
e    77
Name: count, dtype: int64
```

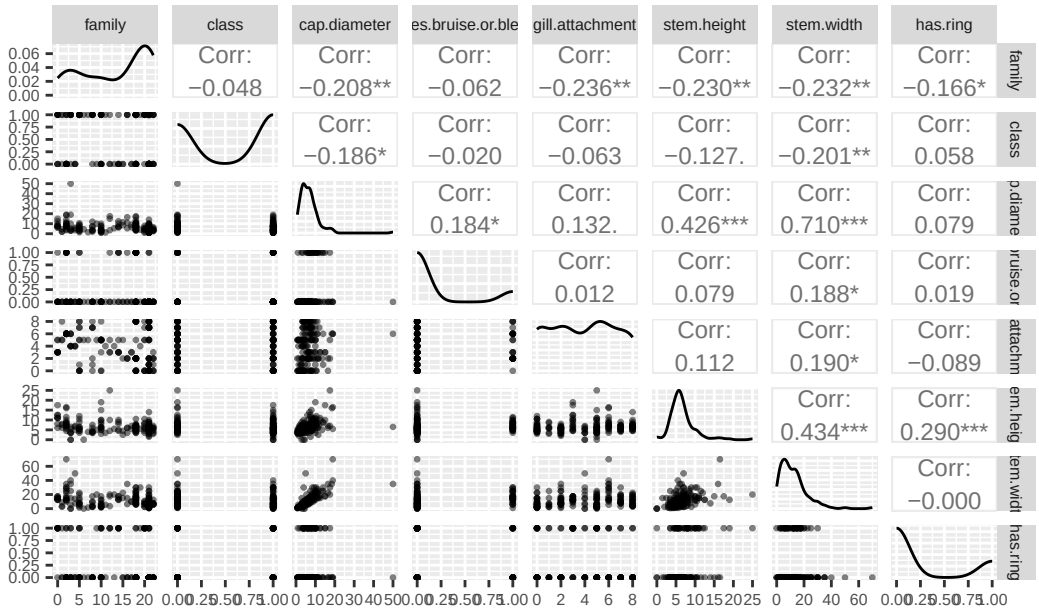
```
# make class distribution
plt.figure(figsize=(6, 4))
ax = sns.countplot(data=df, x="class", palette="pastel")
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Count")

# add the umber on the chart
for p in ax.patches:
    plt.text(p.get_x() + p.get_width() / 2., p.get_height(),
             f'{p.get_height():.0f}',
             ha='center', va='bottom', fontsize=12, color='black')

# show the chart
plt.show()
```



from this chart,we can see that there are 96 poisonous and 77 edibile.

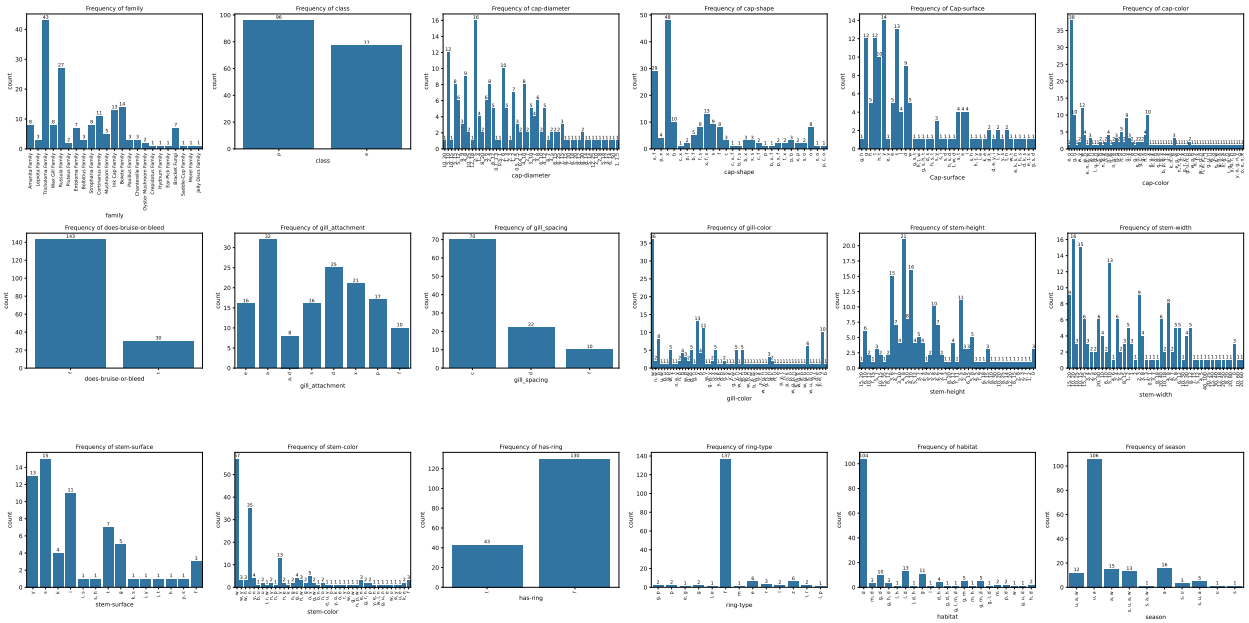
```
library(GGally)
library(ggplot2)

# read CSV
df <- read.csv("C:/Users/user/Downloads/processed_data11.csv")

# select the numerical_columns
numerical_columns <- sapply(df, is.numeric)

# create a ggpairs chart
plot <- ggpairs(df[, numerical_columns],
                upper = list(continuous = wrap("cor", size = 3)),  # cor-plot size
                lower = list(continuous = wrap("points", alpha = 0.5, size = 0.5)),  # pointsize
                title = "GGpairs Plot") +
        theme(text = element_text(size = 8))  # textsize

# show the plot
print(plot)
```

GGpairs Plot




```
import matplotlib.pyplot as plt
import seaborn as sns

# select the categorical_columns
categorical_columns = ['family', 'class', 'cap-diameter', 'cap-shape', 'Cap-surface', 'cap-color',
                       'does-bruise-or-bleed', 'gill_attachment', 'gill_spacing', 'gill-color',
                       'stem-height', 'stem-width', 'stem-surface', 'stem-color', 'has-ring',
                       'ring-type', 'habitat', 'season']

# df1 = pd.read_csv("C:/Users/user/Downloads/primary_data_cleaned.csv")

# create a 6×3 chart
fig, axes = plt.subplots(nrows=3, ncols=6, figsize=(30, 15))  # chartsize
axes = axes.flatten()  # Flatten a 2D array to 1D

# create all the chart
for i, column in enumerate(categorical_columns):
    ax = sns.countplot(data=df, x=column, ax=axes[i])
    ax.set_title(f'Frequency of {column}', fontsize=10)  # set title
    ax.tick_params(axis='x', rotation=90, labelsize=8)  # rotation and labelsize

    # add the umber on the chart
    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center',
                    fontsize=8, color='black',
                    xytext=(0, 5), textcoords='offset points')

# Adjust layout
plt.tight_layout()
```

```
# show the chart
plt.show()
```



七、資料前處理-補值+encoding

```
df = pd.read_csv("C:/Users/user/Downloads/primary_data_cleaned.csv")
# calcute the missing values
missing_values = df.isnull().sum()
print("   ")
```

```
print(missing_values[missing_values > 0])
```

```
Cap-surface          40
gill_attachment      28
gill_spacing         71
stem_root           146
stem-surface        108
veil-type           164
veil-color          152
ring-type             7
Spore-print-color   155
dtype: int64
```

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

```
df=pd.read_csv("C:/Users/user/Downloads/primary_data_cleaned.csv")
# 1. mice imputation
numerical_columns = ['cap-diameter', 'stem-height', 'stem-width']  # select the numberical feature

def convert_range_to_median(value):
    if pd.isna(value):
        return np.nan
    if isinstance(value, str) and ',' in value:
        try:
            values = [float(x.strip()) for x in value.split(',')]
            return np.median(values)
        except:
            return np.nan
    try:
        return float(value)
    except:
        return np.nan

# convert to median
for col in numerical_columns:
    df[col] = df[col].apply(convert_range_to_median)

# use MICE stagety
imputer = IterativeImputer(random_state=0)
df[numerical_columns] = imputer.fit_transform(df[numerical_columns])

# Label Encoding
label_columns = ['family', 'class', 'does-bruise-or-bleed', 'gill_attachment', 'has-ring']
label_encoder = LabelEncoder()

for col in label_columns:
    # if it have nan, change it to missing
    df[col] = df[col].fillna('missing')
    df[col] = label_encoder.fit_transform(df[col].astype(str))

# 3. delete the missing feature
columns_to_drop = ['veil-type', 'veil-color', 'Spore-print-color', 'stem-surface', 'stem_root', 'name']
df.drop(columns=[col for col in columns_to_drop if col in df.columns], inplace=True)

# 4. One-Hot Encoding
multi_columns = ['cap-shape', 'Cap-surface', 'cap-color', 'gill_spacing', 'gill-color',
                 'stem-color', 'ring-type', 'season', 'habitat']

for col in multi_columns:
    if col in df.columns:
        # fill in nan
        df[col] = df[col].fillna('').astype(str)  #
        # split the multi column
        dummies = df[col].str.split(',', expand=True).stack().str.strip()
        dummies = pd.get_dummies(dummies, prefix=col)
        # groupby it
        dummies = dummies.groupby(level=0).sum()
```

```
        # concat to DataFrame
        df = pd.concat([df, dummies], axis=1)
        # drop the original column
        df.drop(columns=[col], inplace=True)

# know that how size is it
print("   ", df.shape)
```

```
    (173, 89)
```

```
# get the processed_data1
df.to_csv('processed_data1.csv', index=False)
```

I first use MICE (IterativeImputer) to fill in the values. MICE (Multiple Imputation by Chained Equations) can infer missing values based on other numerical features, which is more accurate than simple mean filling.

Then because of cap-diameter, stem-height, stem-width Some of these numerical features are in range format, so I designed a function convert_range_to_median for subsequent processing. If it is a range (, separated), the median is calculated to fill the value. If it cannot be parsed, it is set to NaN to facilitate subsequent value filling.

Next, we use Label Encoding to process categorical data because they have fewer categories, such as family, class, does-bruise-or-bleed, gill-attachment, has-ring. If there are NaN values, they will be filled with 'missing' to avoid LabelEncoder errors.

Then we remove unimportant features or features with too many missing values, such as: veil-type, veil-color, pore-print-color, stem-surface, stem_root, name → These features may not contain enough information or cannot be directly quantified, so they are removed to simplify the model.

One-Hot Encoding is also used. The processing method for this part is special because some features (such as cap-shape, cap-surface, cap-color, gill-spacing...) may be multi-valued (comma-separated), for example: cap-color, season, etc. Therefore, I did the following steps to fill NaN with "" (empty string) to avoid str.split() errors. str.split(',') splits into multiple categories and then expands into multiple lines (stack()). Use pd.get_dummies() for One-Hot Encoding and convert it to 0/1 features. Use groupby(level=0).sum() to merge multiple rows of results back into the original DataFrame. Delete the original feature field to avoid duplication.

Finally, output the data size (df.shape) to check the final number of features and confirm whether One-Hot Encoding successfully added new features. Save it as processed_data1.csv for subsequent modeling.

八、模型訓練比較

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# read CSV
df = pd.read_csv("C:/Users/user/Downloads/processed_data1.csv")

# define X and y
X = df.drop(columns=['class'])  # feature
y = df['class']                 # target
```

```
# ensure that all is float
X = X.astype(float)

# split the train test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# make a randomforest model
model = RandomForestClassifier(n_estimators=40, random_state=42)

# train the model
model.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=40, random_state=42)
```

```
# predict!
y_pred = model.predict(X_test)

# calculating the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"   : {accuracy:.4f}")
```

```
   : 0.6000
```

```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# read CSV
df = pd.read_csv("C:/Users/user/Downloads/processed_data1.csv")

# define X and y
X = df.drop(columns=['class']).astype(float)
y = df['class']

# make a LogisticRegression model
model = LogisticRegression(max_iter=1000, random_state=42)

# 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# print the score
print(f"   5     : {scores.mean():.4f} (±{scores.std():.4f})")
```

```
   5     : 0.5492 (±0.0421)
```

```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

# read CSV
df = pd.read_csv("C:/Users/user/Downloads/processed_data1.csv")

# define X and y
X = df.drop(columns=['class']).astype(float)
```

```
y = df['class']

# make an advanced randomforest model
model = RandomForestClassifier(
    n_estimators=50,      # number of tree
    max_depth=5,
    min_samples_split=10,
    random_state=42
)

# 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# print the score
print(f"     5      : {scores.mean():.4f} (±{scores.std():.4f})")
```

```
     5      : 0.5550 (±0.0996)
```

```
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

# read CSV
df = pd.read_csv("C:/Users/user/Downloads/processed_data1.csv")

# define X and y
X = df.drop(columns=['class']).astype(float)
y = df['class']

# make a advanced LogisticRegression model
model = LogisticRegression(max_iter=1000, random_state=42)

# 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# print the score
print(f"   5      : {scores.mean():.4f} (±{scores.std():.4f})")
```

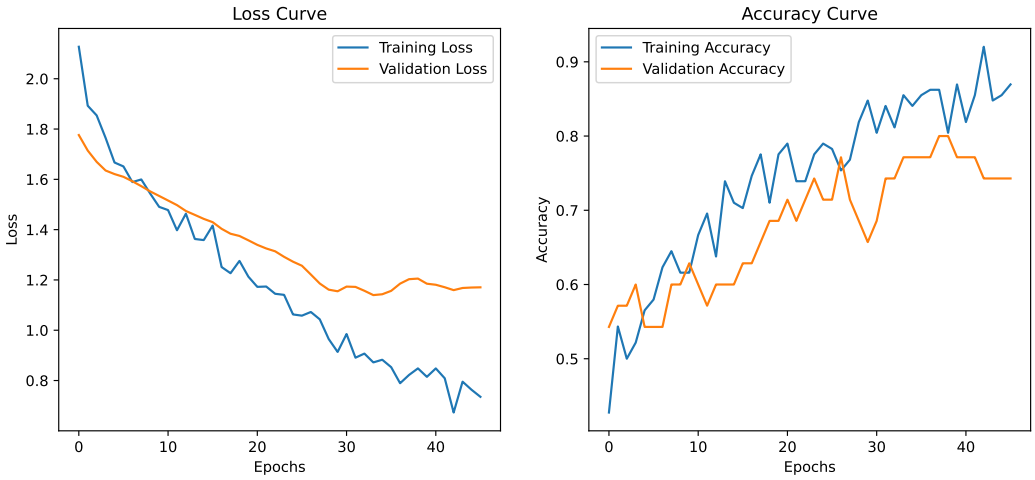
```
   5      : 0.5492 (±0.0421)
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers, regularizers
from sklearn.decomposition import PCA  #

# read CSV
df = pd.read_csv("C:/Users/user/Downloads/processed_data1.csv")

# define X and y
X = df.drop(columns=['class'])
y = df['class']
```

```
# StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dimensionality reduction (reducing the number of features)
pca = PCA(n_components=30)  # Keep 30 components
X_scaled = pca.fit_transform(X_scaled)
print(f"PCA  {sum(pca.explained_variance_ratio_):.4f}")
```

```
# split the train test data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# make a dnn model
model = keras.Sequential([
    layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01), input_shape=(X_train.s
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(16, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.BatchNormalization(),
    layers.Dropout(0.3),

    layers.Dense(1, activation='sigmoid')  # binary question can use sigmoid activation function
])

# use Adam optimizer and setting learning rate
optimizer = keras.optimizers.Adam(learning_rate=0.003)

# compile the model
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

# define the reduce function
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, min_lr=1e-6)
early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=12, restore_best_weights=Tru

# train the model
history = model.fit(X_train, y_train,
                    epochs=200, batch_size=16,
                    validation_data=(X_test, y_test),
                    callbacks=[early_stopping, reduce_lr],
                    verbose=2)
```

```
Deep Learning Accuracy: 0.7714
```


Loss Curve
Training Loss
Validation Loss
Loss
Epochs
Accuracy Curve
Training Accuracy
Validation Accuracy
Accuracy
Epochs