Classification Assignment

By: Pankajan T.

The Libraries used in this project

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from statsmodels.tools.tools import add_constant
from scipy.special import logit
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.ensemble import AdaBoostClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn import preprocessing
from sklearn import tree
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
from sklearn.ensemble import GradientBoostingClassifier
from IPython.display import Image
import pydotplus
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
```

We are taking the Dataset about mushrooms

The dataset have 8124 mushroom data each having 23 different properties to themselves.
link for the dataset is: https://www.kaggle.com/datasets/uciml/mushroom-classification

class edible=e, poisonous=p

cap-shape bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s

cap-surface fibrous=f,grooves=g,scaly=y,smooth=s

cap-color
brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y

bruises bruises=t,no=f

odor almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

gill-attachment attached=a, descending=d, free=f, notched=n

gill-spacing close=c,crowded=w,distant=d

gill-size broad=b,narrow=n

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

In [ ]:
```
df = pd.read_csv('mushrooms.csv', header=0)
df = pd.DataFrame(df)
```

In [ ]:
```
df.head()
```

Out[ ]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	st co abo r
0	p	x	s	n	t	p	f	c	n	k	...	s	
1	e	x	s	y	t	a	f	c	b	k	...	s	
2	e	b	s	w	t	l	f	c	b	n	...	s	
3	p	x	y	w	t	p	f	c	n	n	...	s	
4	e	x	s	g	f	n	f	w	b	k	...	s	

5 rows × 23 columns

In [ ]:
```
pd.options.display.max_columns = 300
pd.options.display.max_rows = 100
```

With this data we are going to predict whether a mushroom is edible by its physical attributtes and this is our core mission. So that people can understand the nature of the mushroom fungal growths in their environment. Importantly this project's purpose is predicatbility not interpretablity.

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(8124, 23)
```

Here we are going to find out that is the mushroom is poisonouse.

we are using the all the data to find out that if a new mushroom is presented can we identify is it possible to eat the stuff and survive(just for research not recommended for real world application). Here all other 22 properties can be idenitified by physical presence of the mushroom itself.

Preprocessing and data cleaning

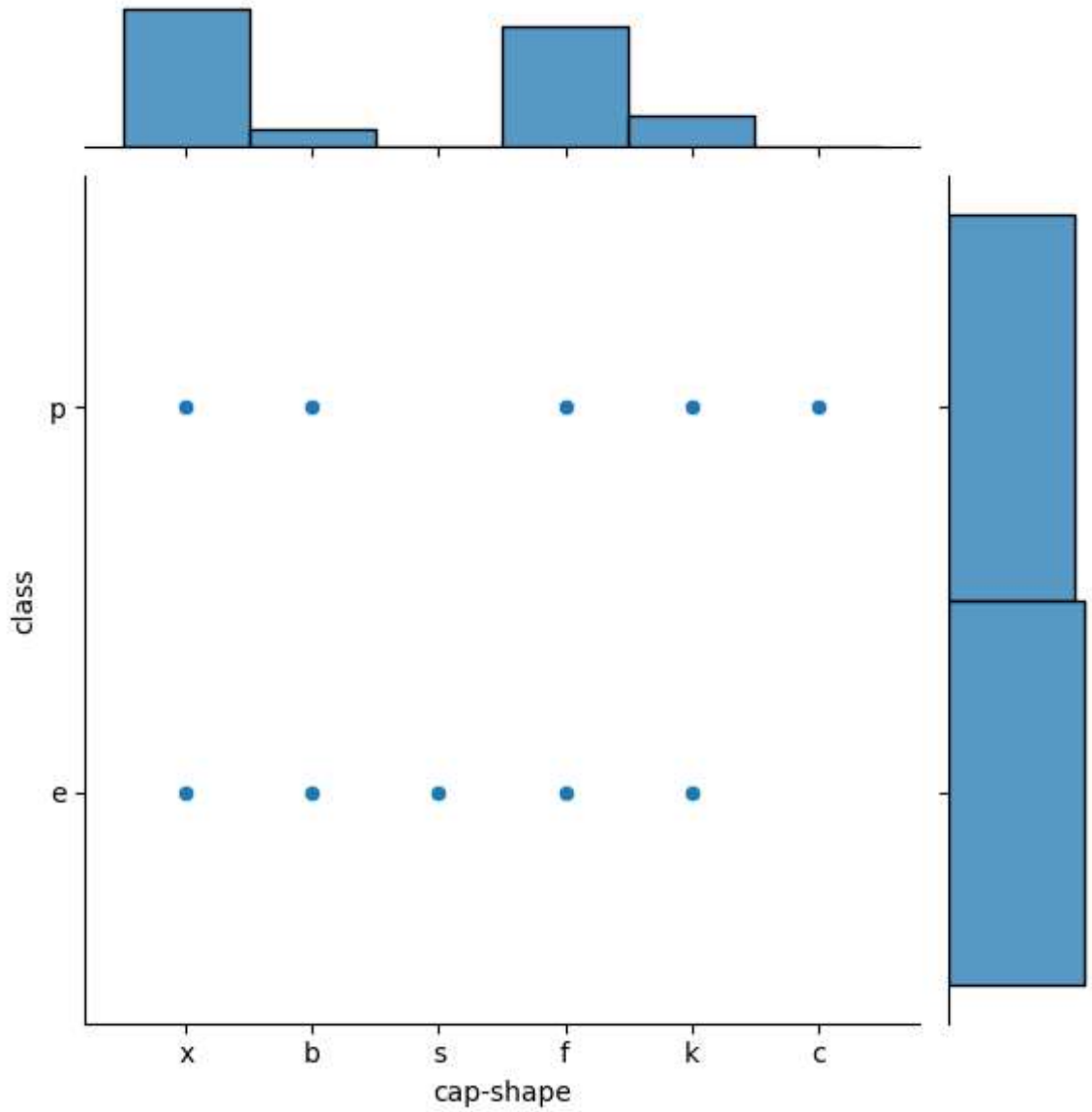
In [ ]:
```
df.describe()
```

Out[ ]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stal ro
count	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	8124	81
unique	2	6	4	10	2	9	2	2	2	12	2	
top	e	x	y	n	f	n	f	c	b	b	t	
freq	4208	3656	3244	2284	4748	3528	7914	6812	5612	1728	4608	37

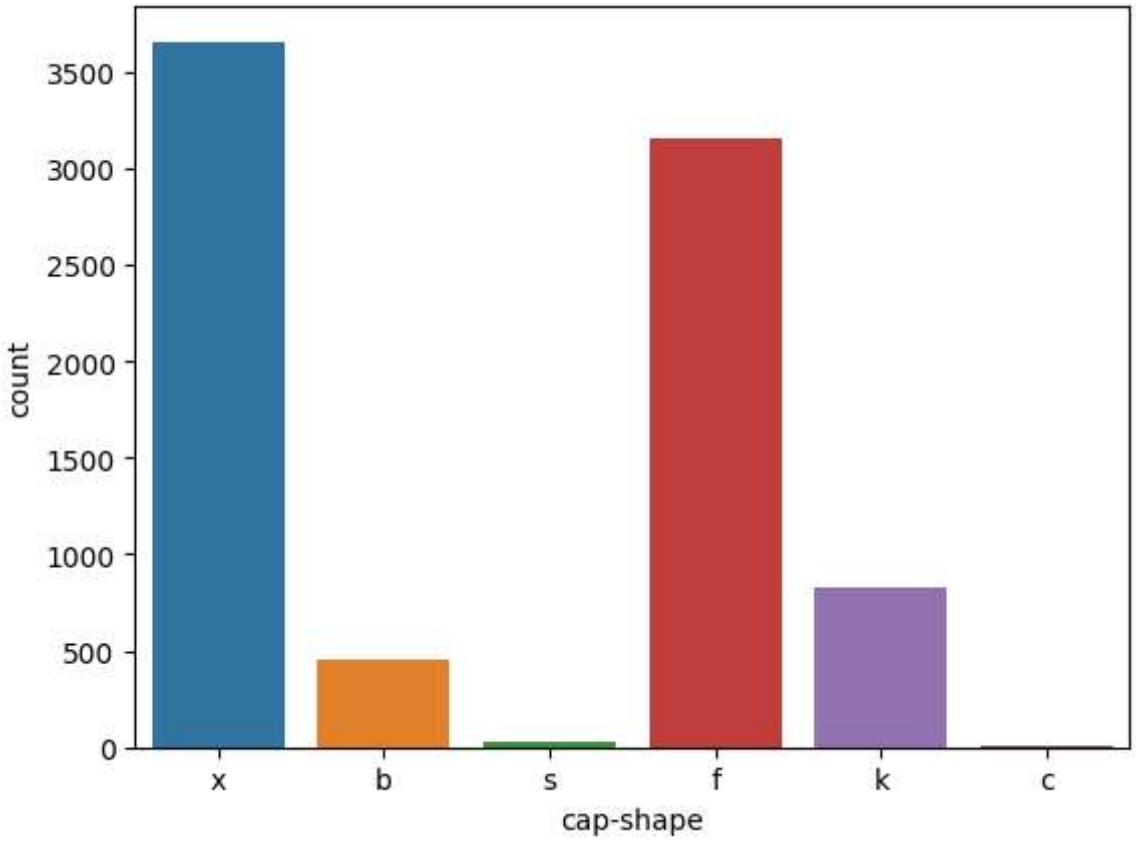
In [ ]:
```
sns.jointplot(x="cap-shape", y="class", data=df)
```

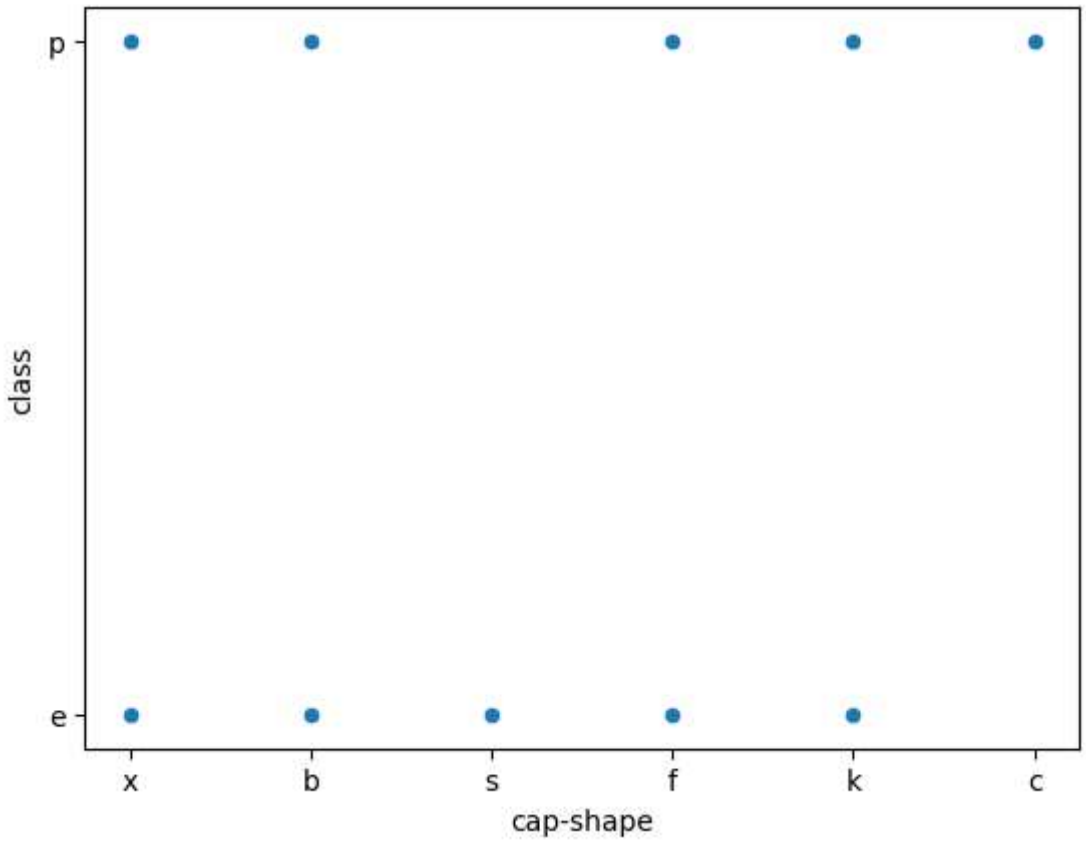
Out[ ]:
```
<seaborn.axisgrid.JointGrid at 0x1f75fca1b20>
```

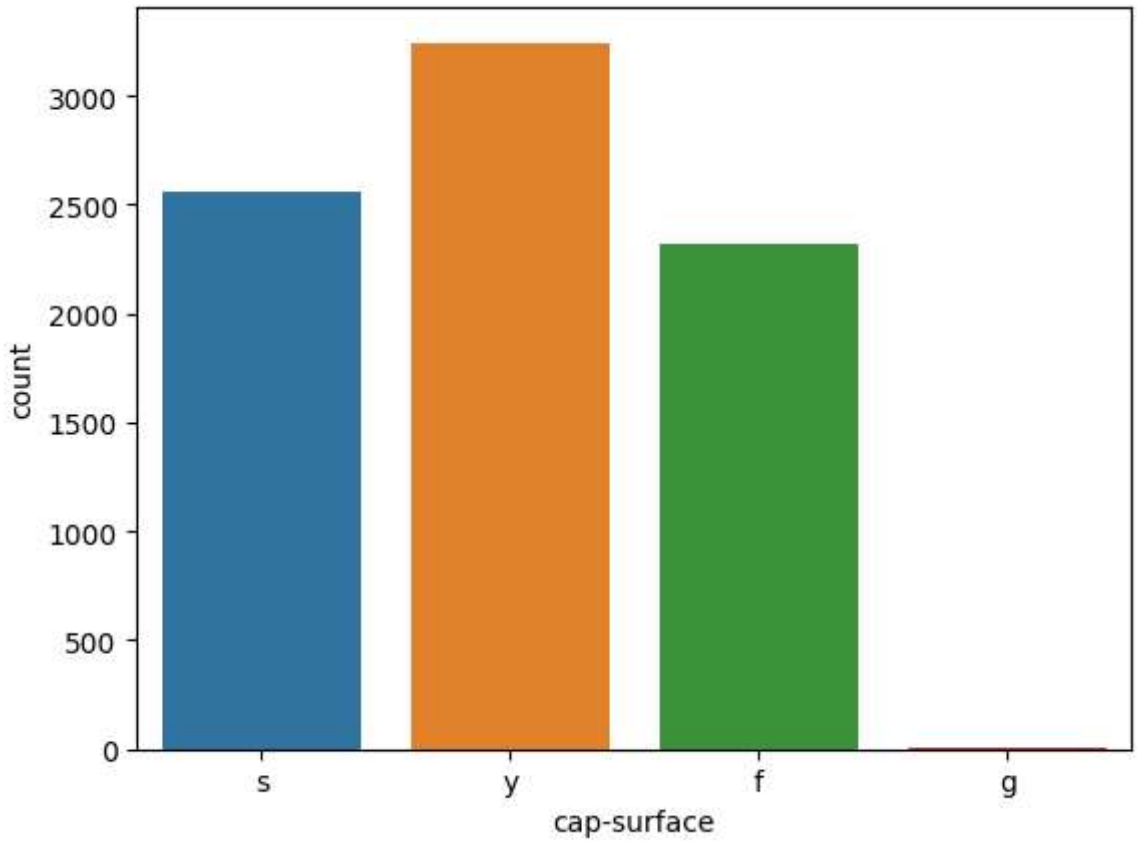


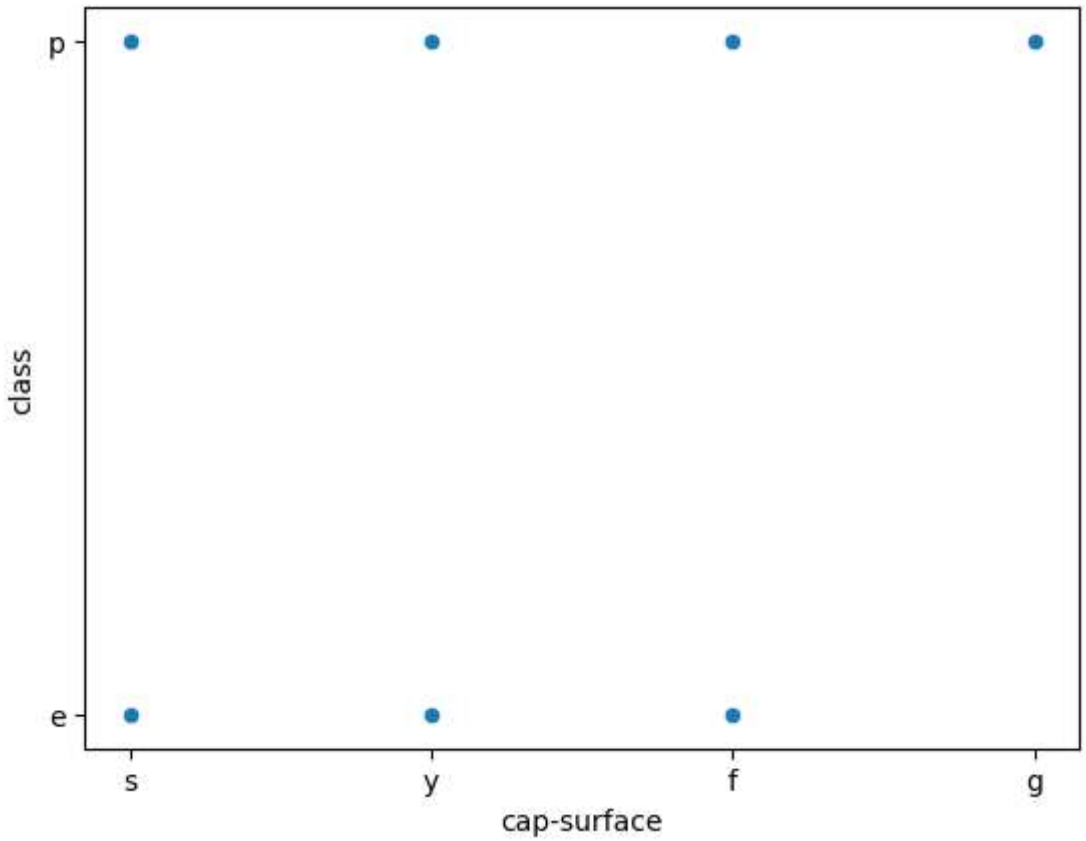
In [ ]:
```
y_col = 'class'

# Iterate over the columns of the dataset
```

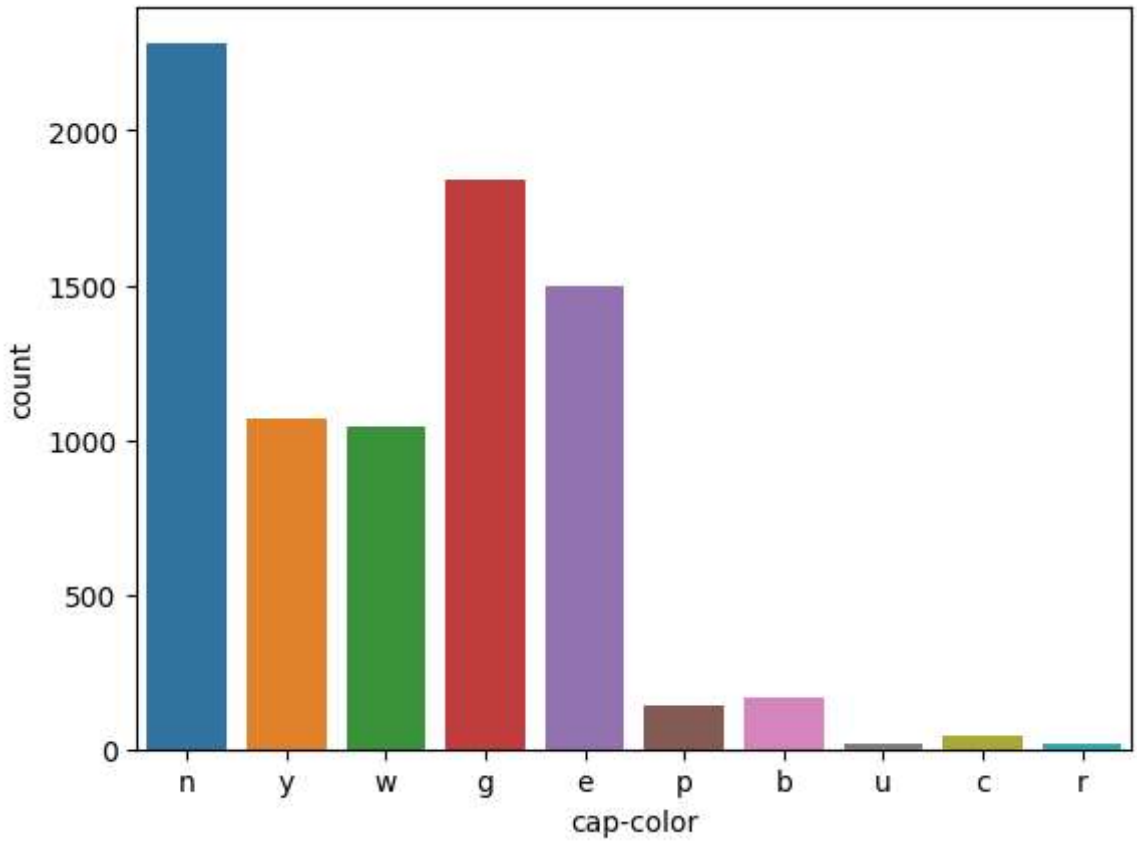
```
for col in df.columns:

    # Skip the y-axis column
    if col == y_col:
        continue

    # Create a scatter plot for the current column


    sns.countplot(x=col, data=df,)
    plt.show()
    sns.scatterplot(data=df,x=col,y=y_col)
    plt.show()
```

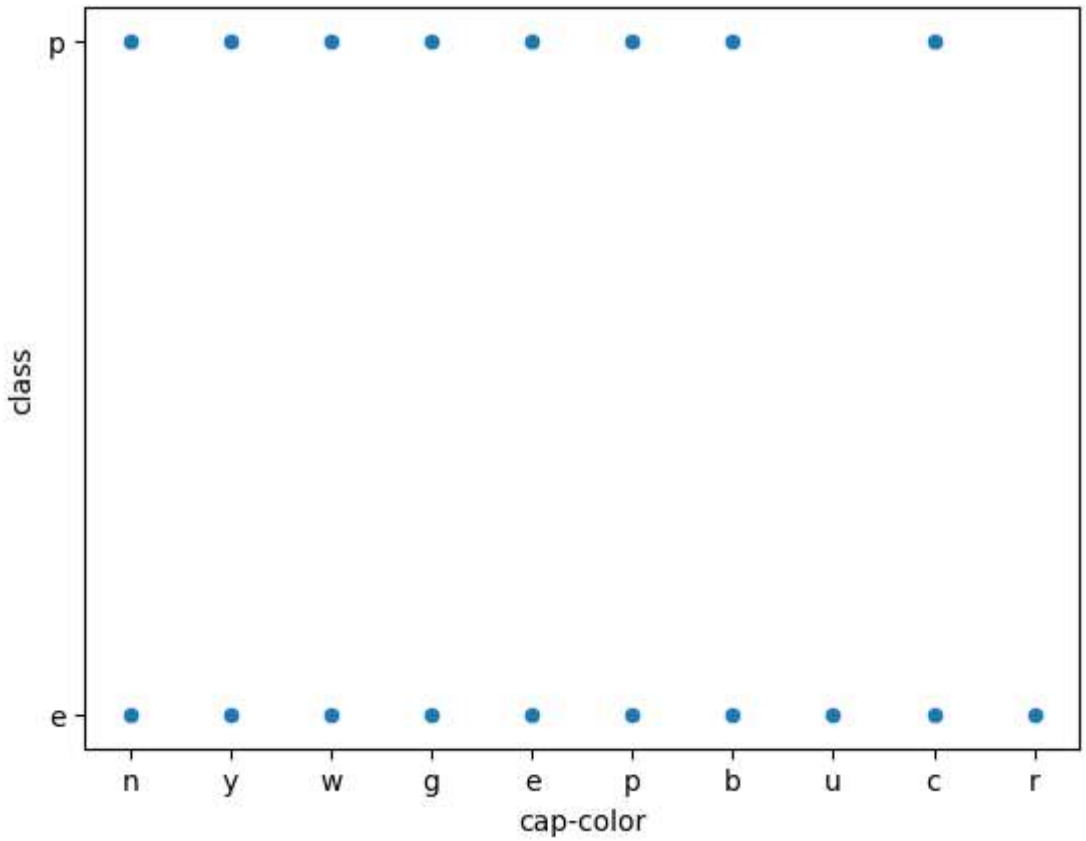


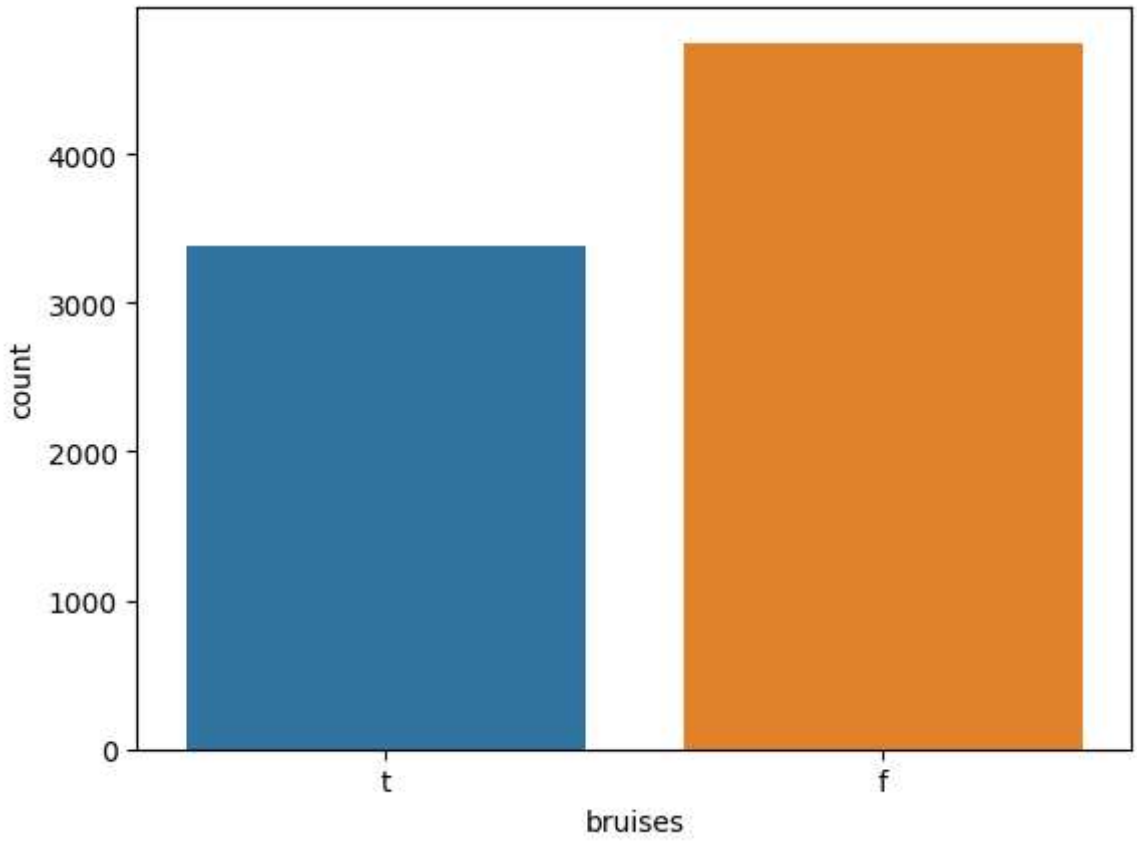

p
e
class
x
b
s
f
k
c
cap-shape

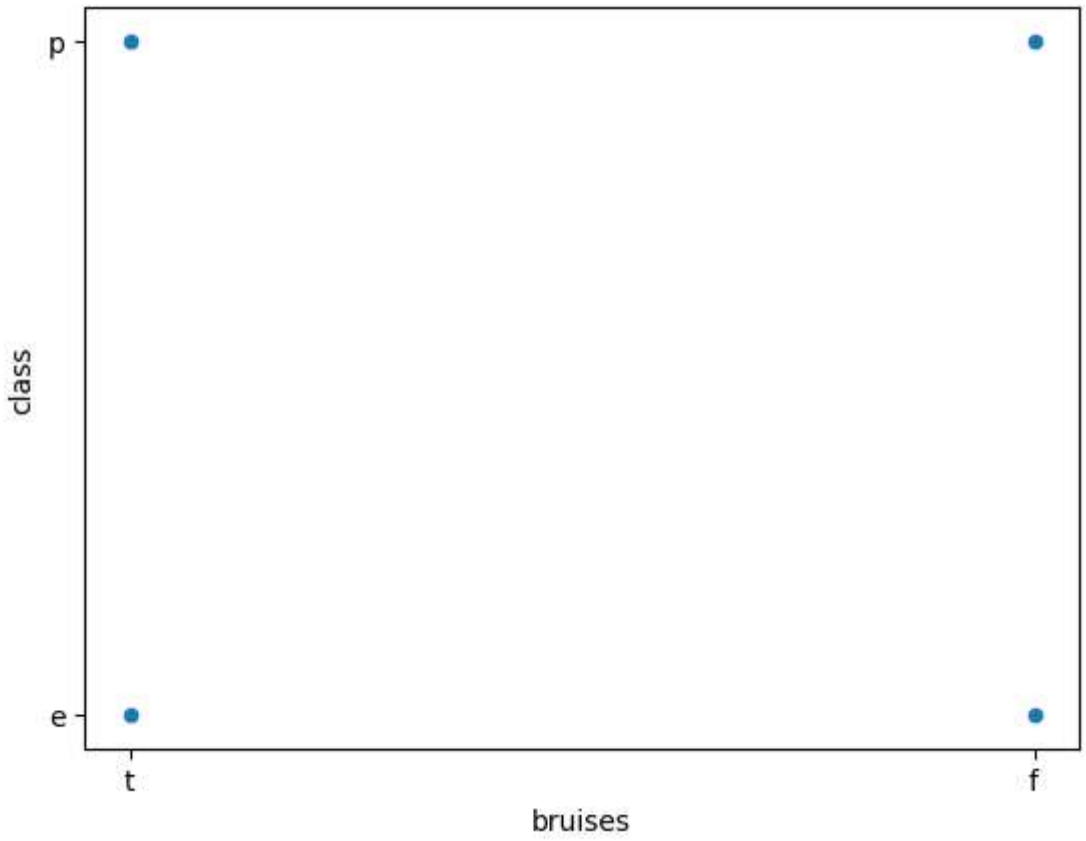

3000
2500
2000
1500
1000
500
0
count
s
y
f
g
cap-surface

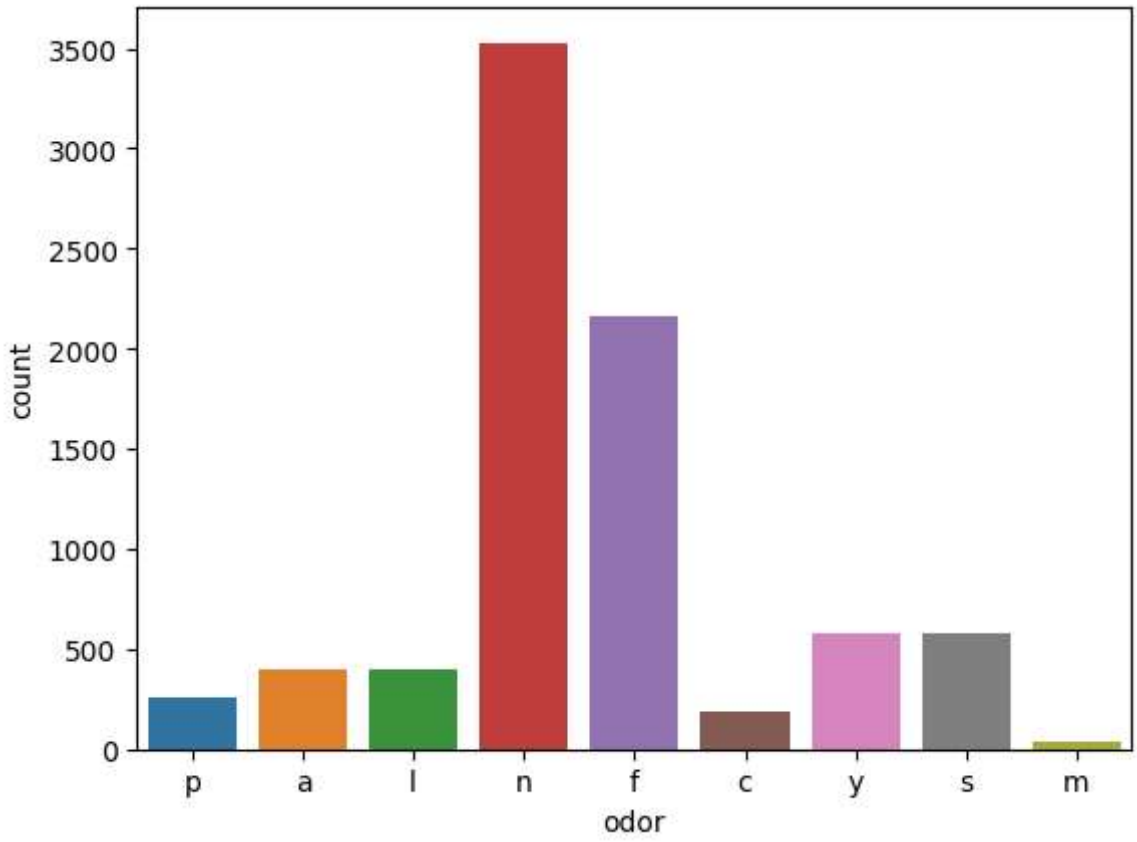

p
e
class
s
y
f
g
cap-surface

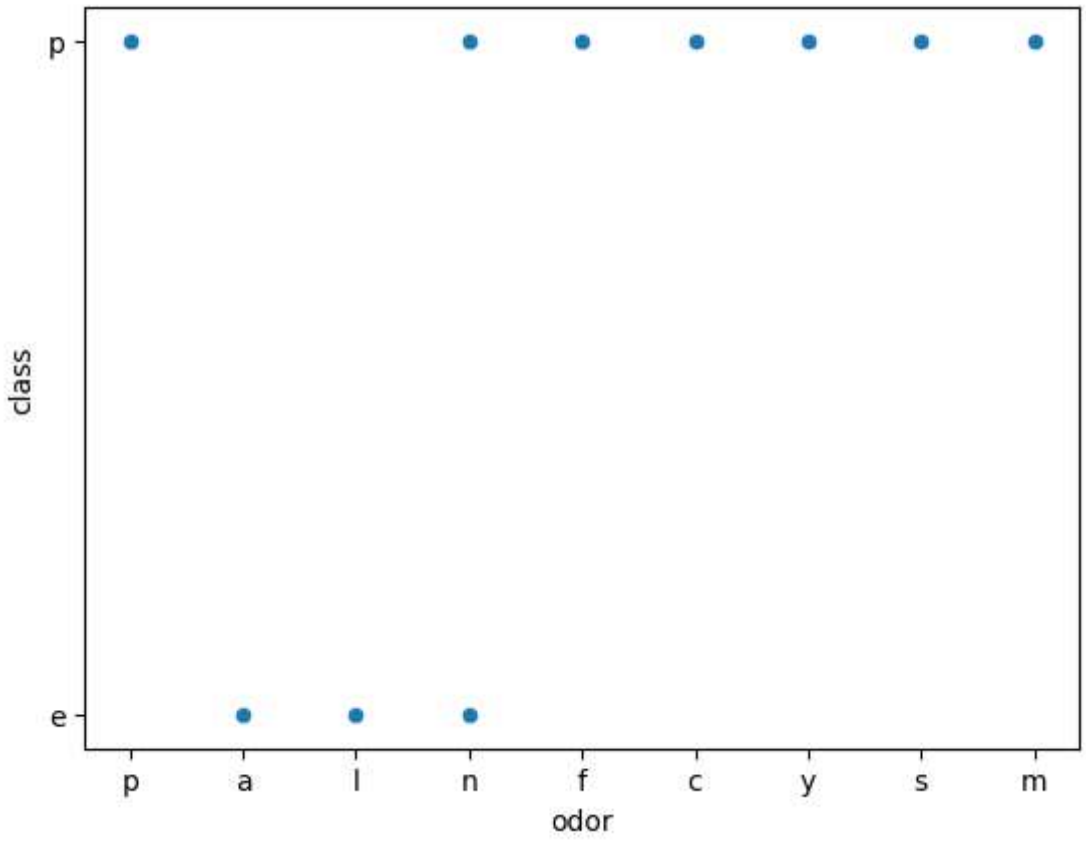

2000
1500
1000
500
0
count
n
y
w
g
e
p
b
u
c
r
cap-color

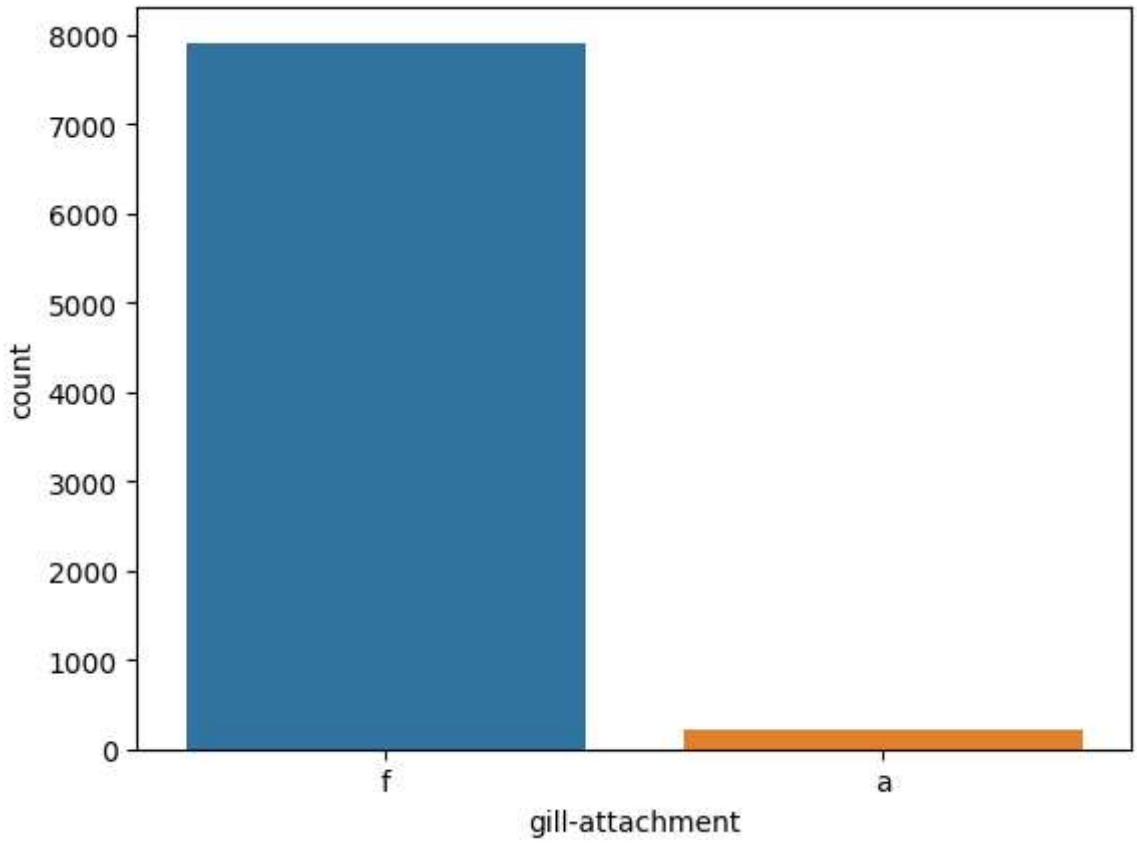

p
e
class
n
y
w
g
e
p
b
u
c
r
cap-color

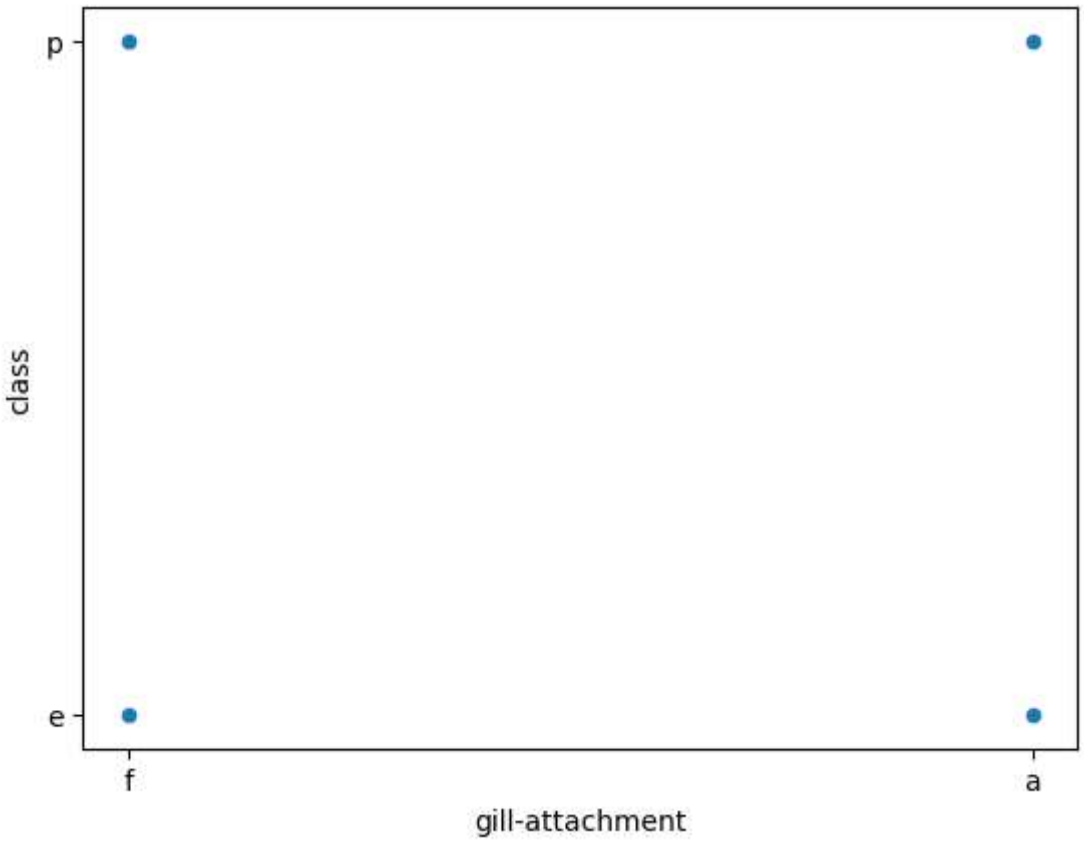

4000
3000
2000
1000
0
count
t
f
bruises

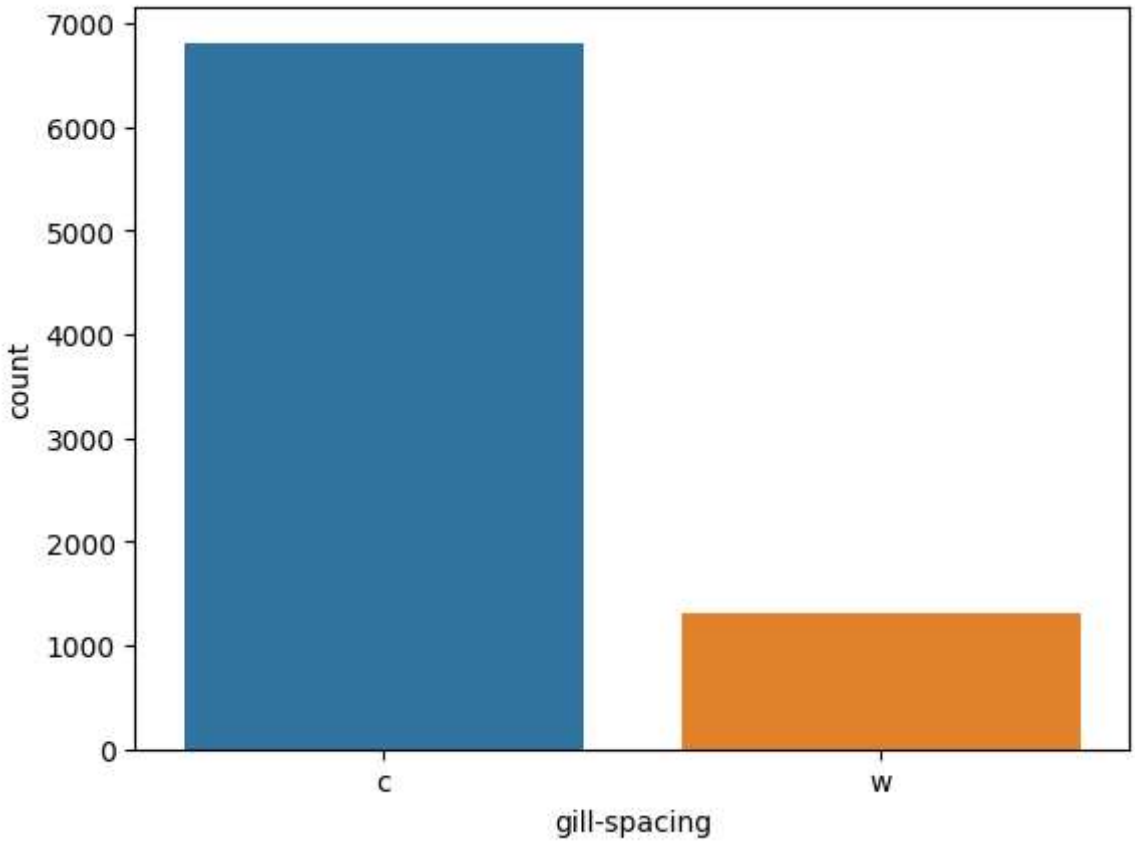

p
e
class
t
f
bruises

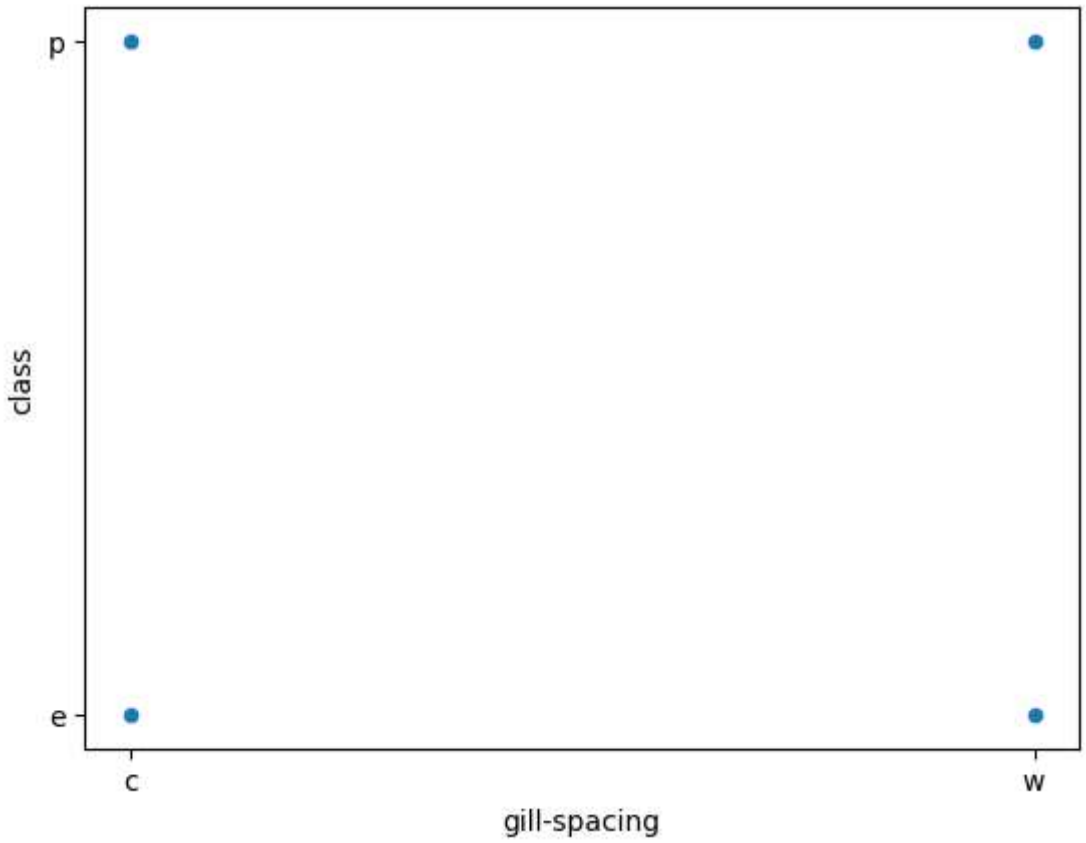

3500
3000
2500
2000
1500
1000
500
0
count
p
a
l
n
f
c
y
s
m
odor

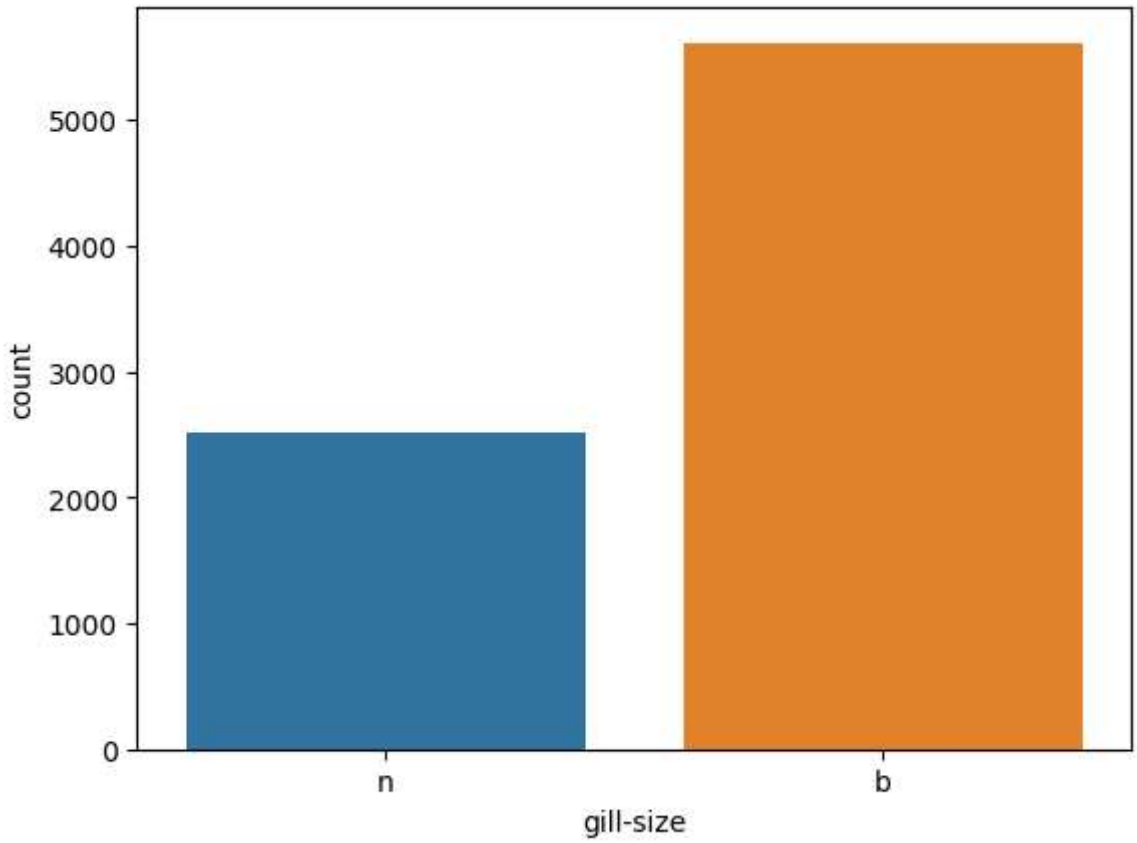

p
e
class
p
a
l
n
f
c
y
s
m
odor

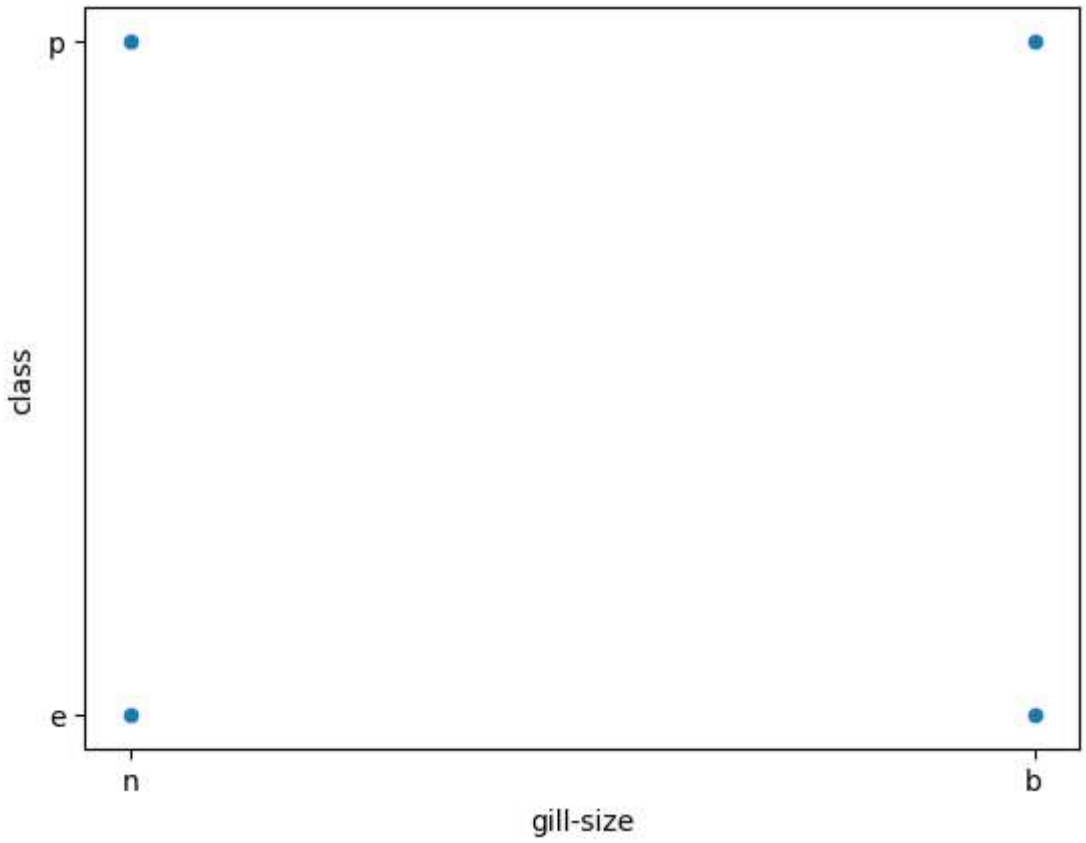

8000
7000
6000
5000
4000
3000
2000
1000
0
count
f
a
gill-attachment

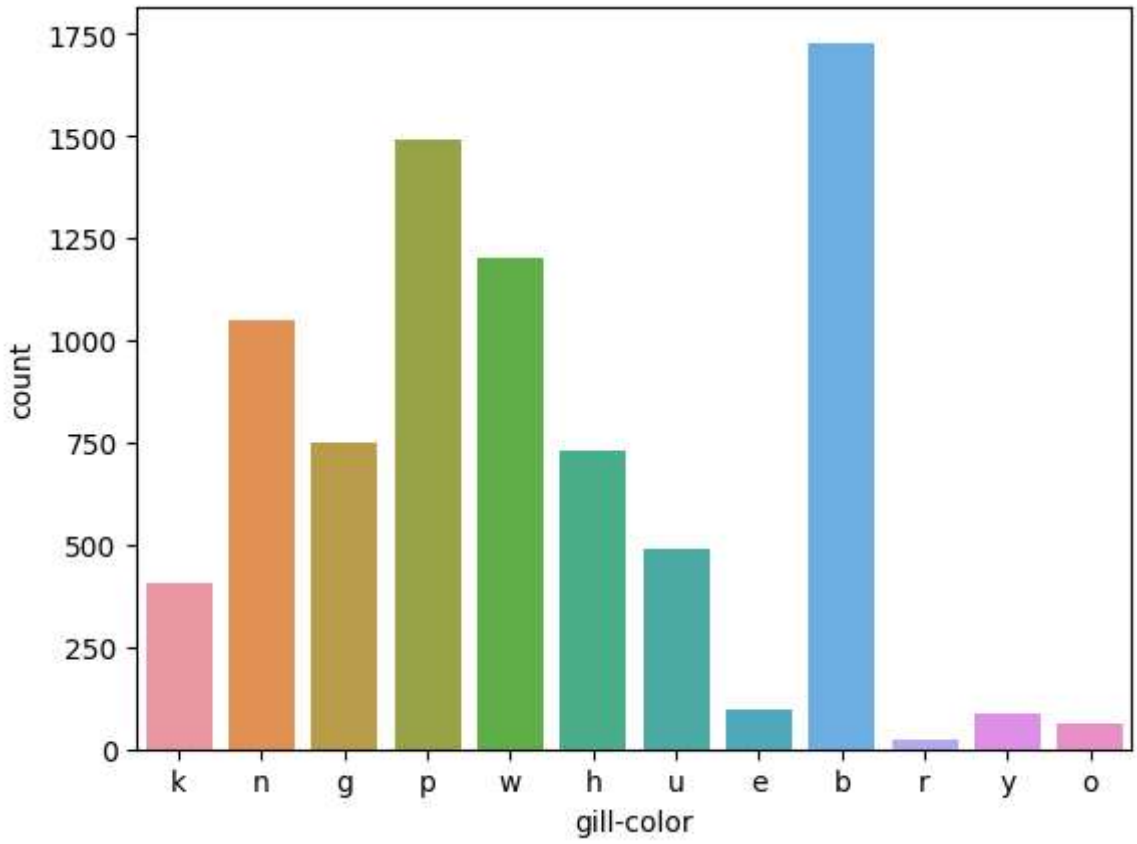

p
e
class
f
a
gill-attachment

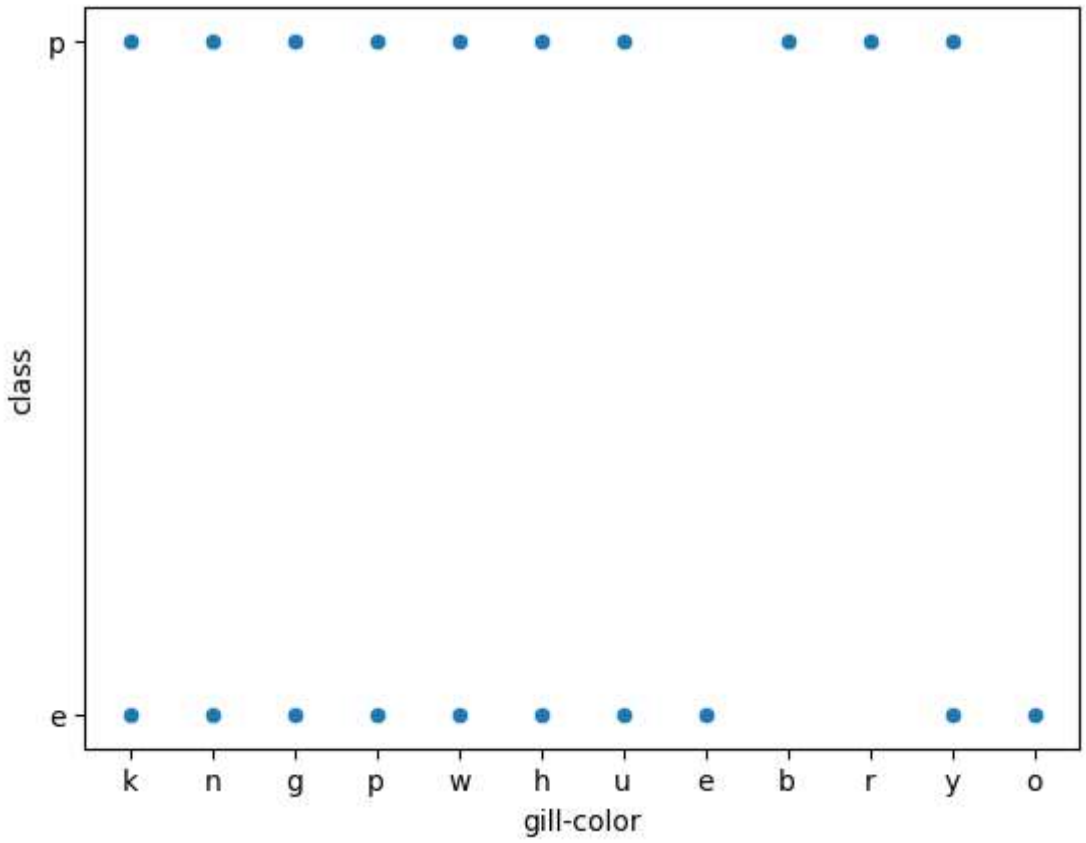

7000
6000
5000
4000
3000
2000
1000
0
count
c
w
gill-spacing

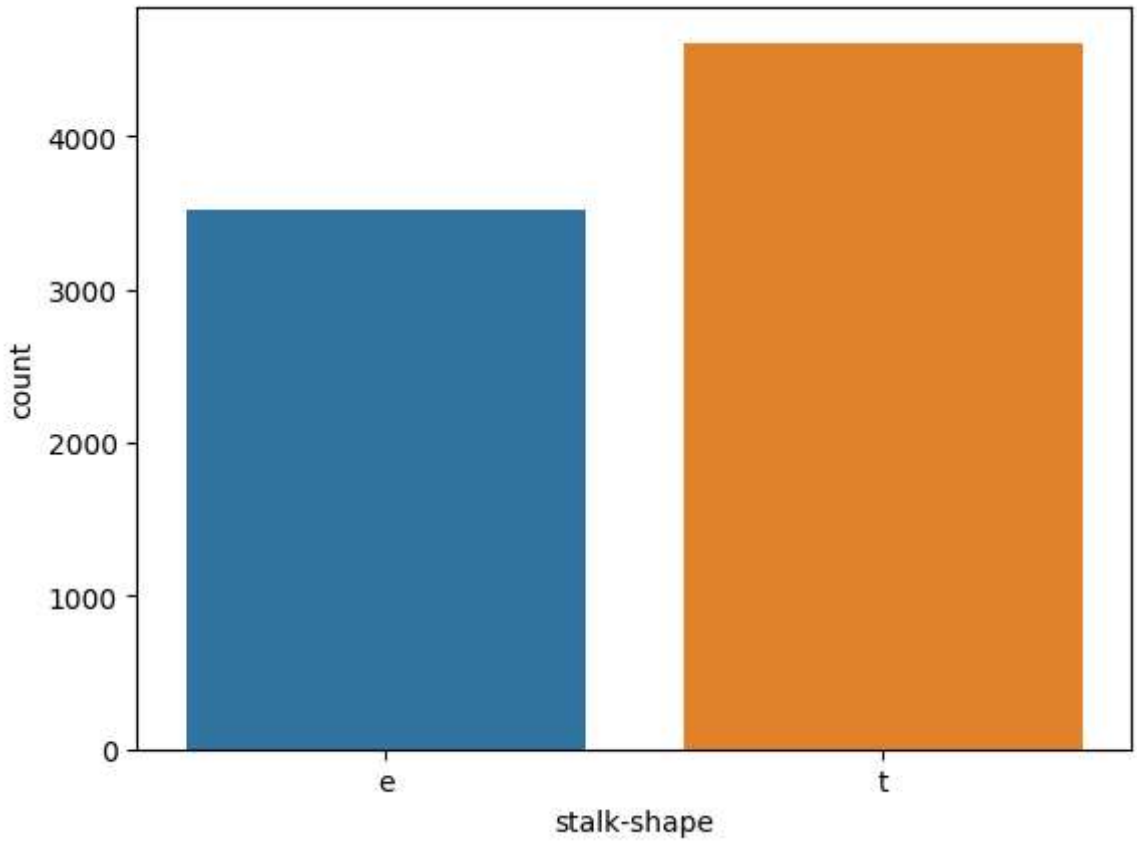

p
e
class
c
w
gill-spacing

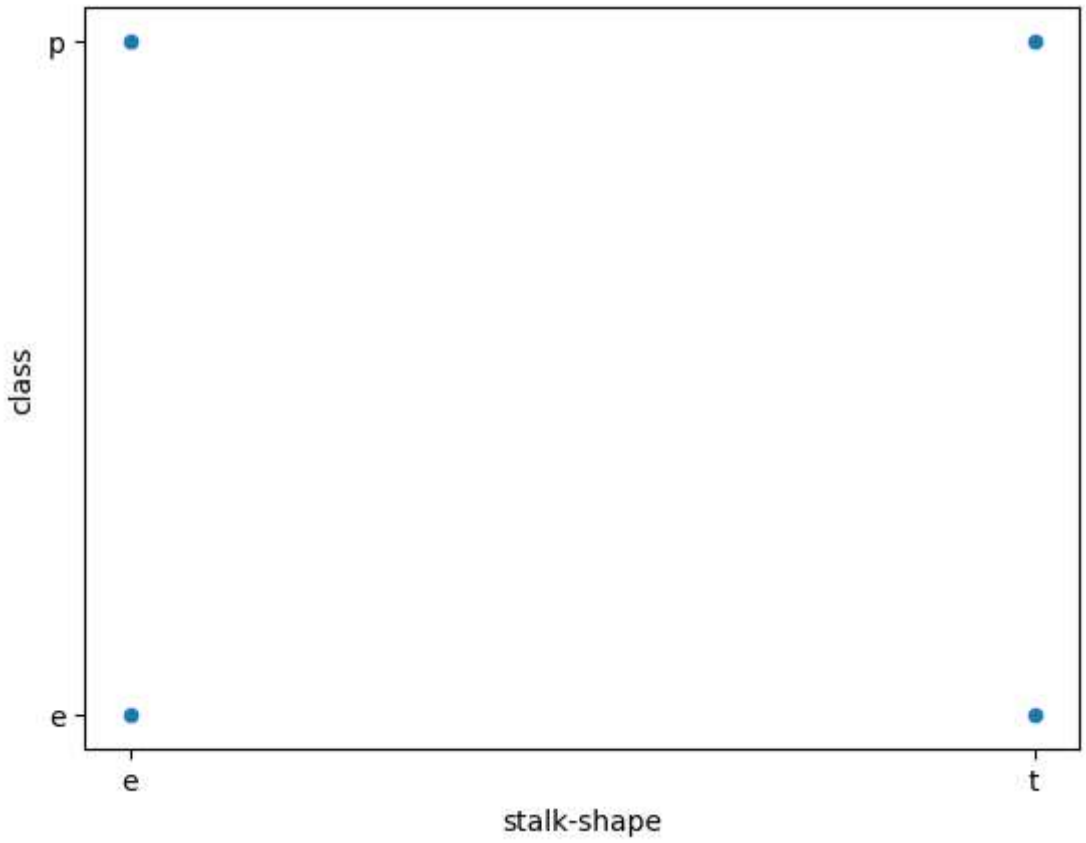

5000
4000
3000
2000
1000
0
count
n
b
gill-size

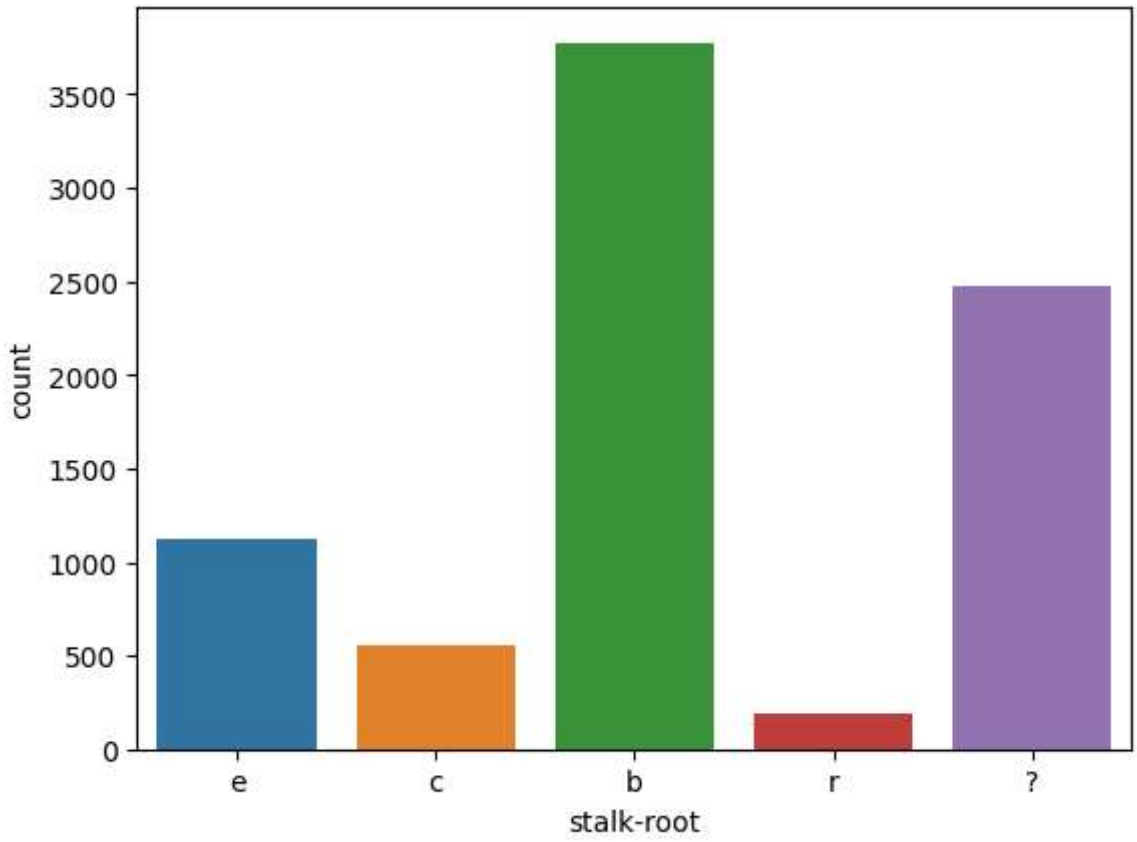

p
e
class
n
b
gill-size

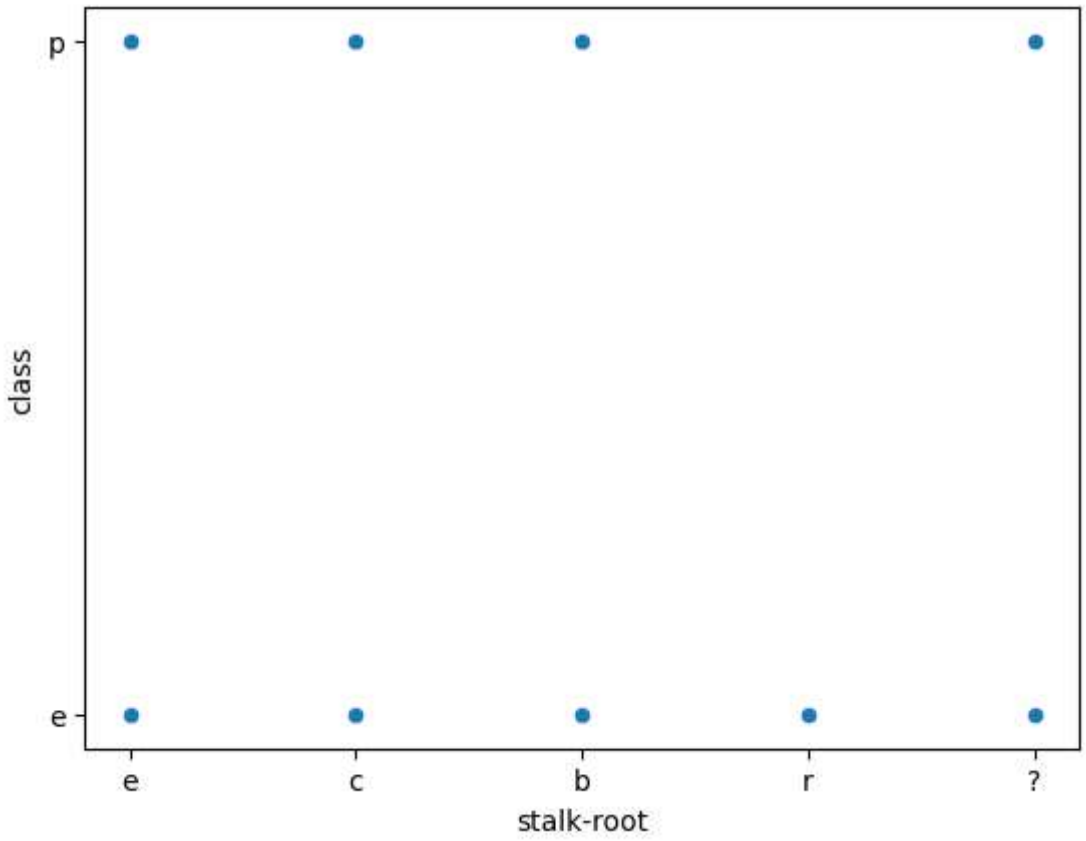

1750
1500
1250
1000
750
500
250
0
count
k
n
g
p
w
h
u
e
b
r
y
o
gill-color

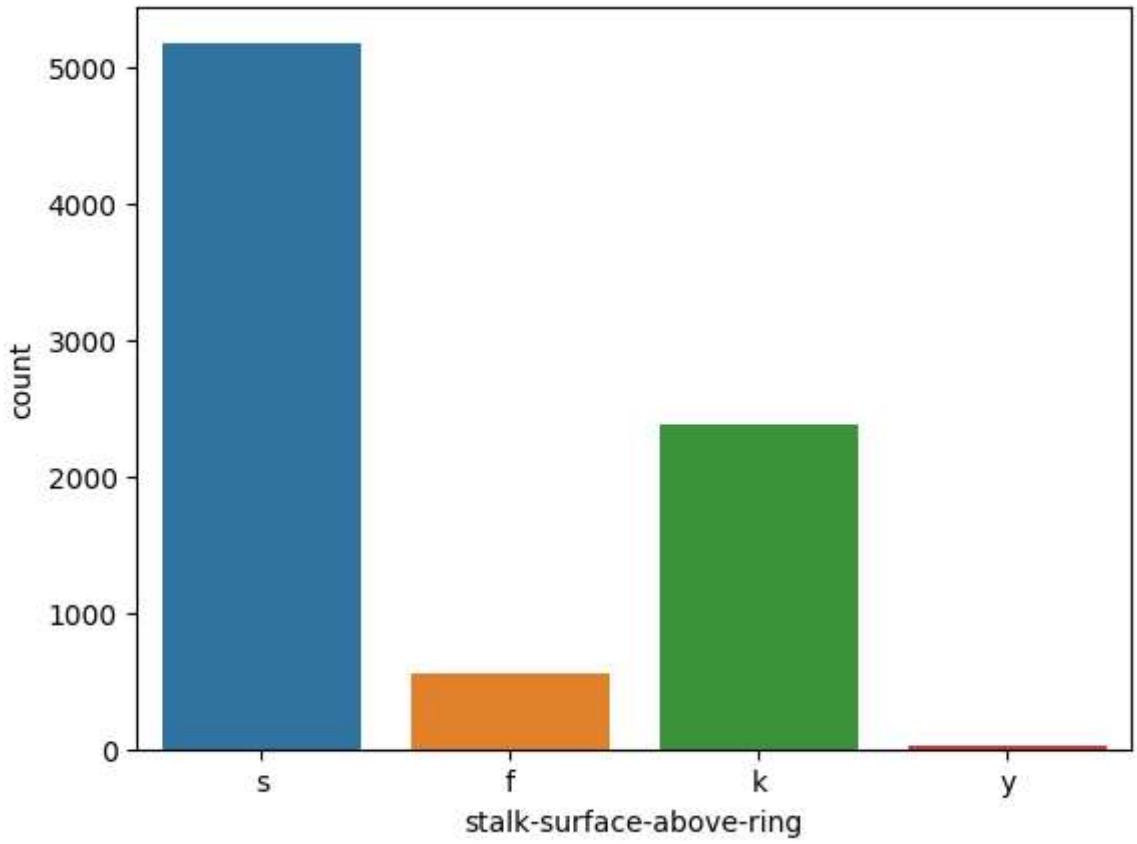

p
e
class
k
n
g
p
w
h
u
e
b
r
y
o
gill-color

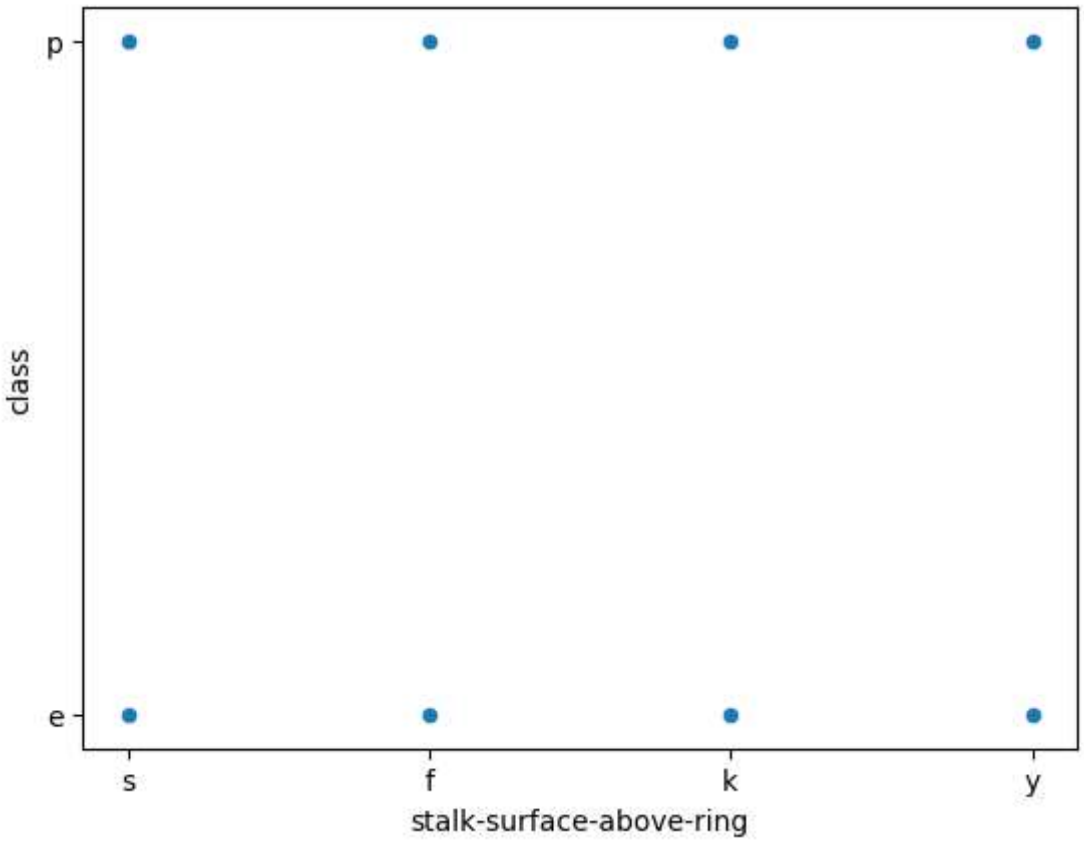

4000
3000
2000
1000
0
count
e
t
stalk-shape

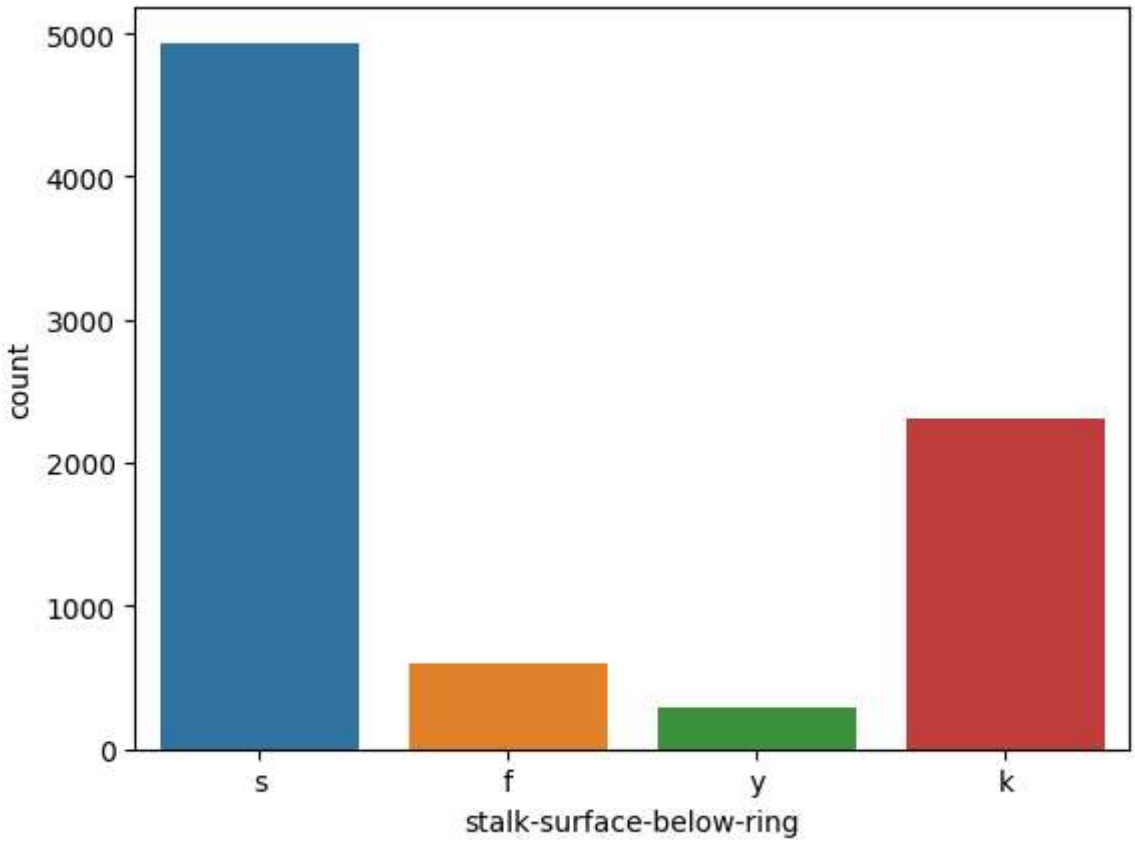

p
e
class
e
t
stalk-shape

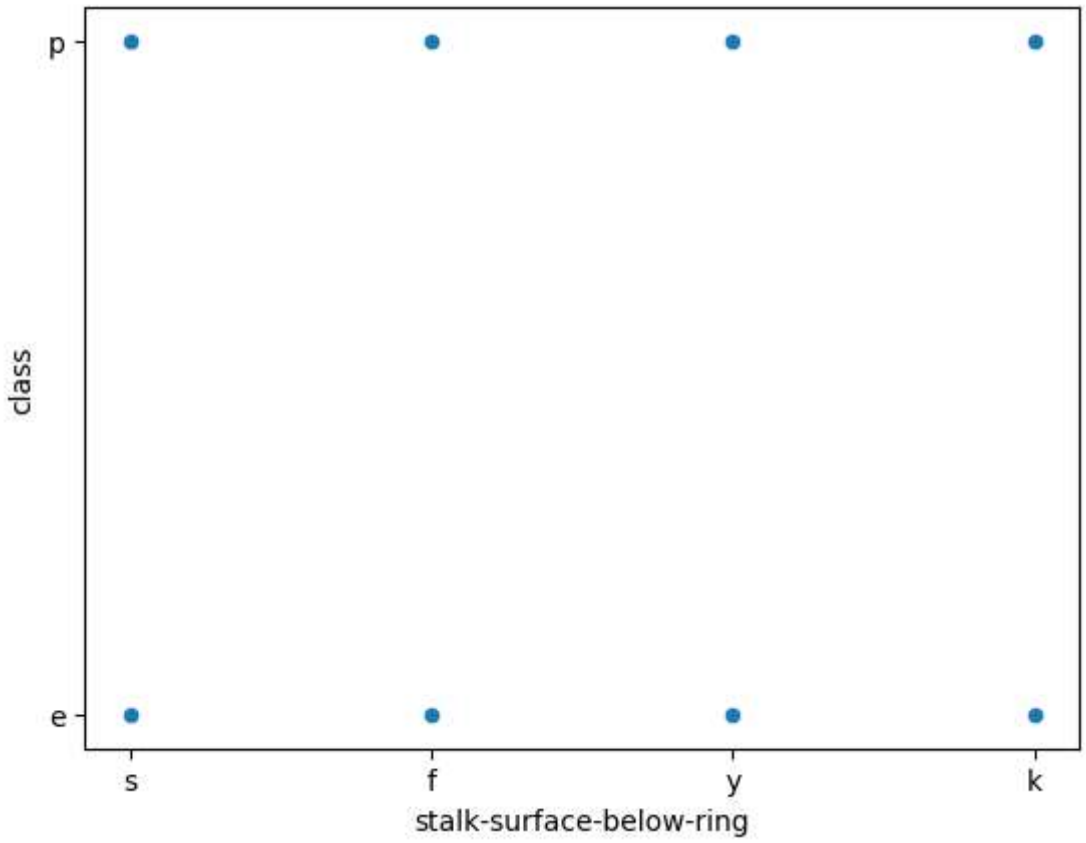

3500
3000
2500
2000
1500
1000
500
0
count
e
c
b
r
?
stalk-root

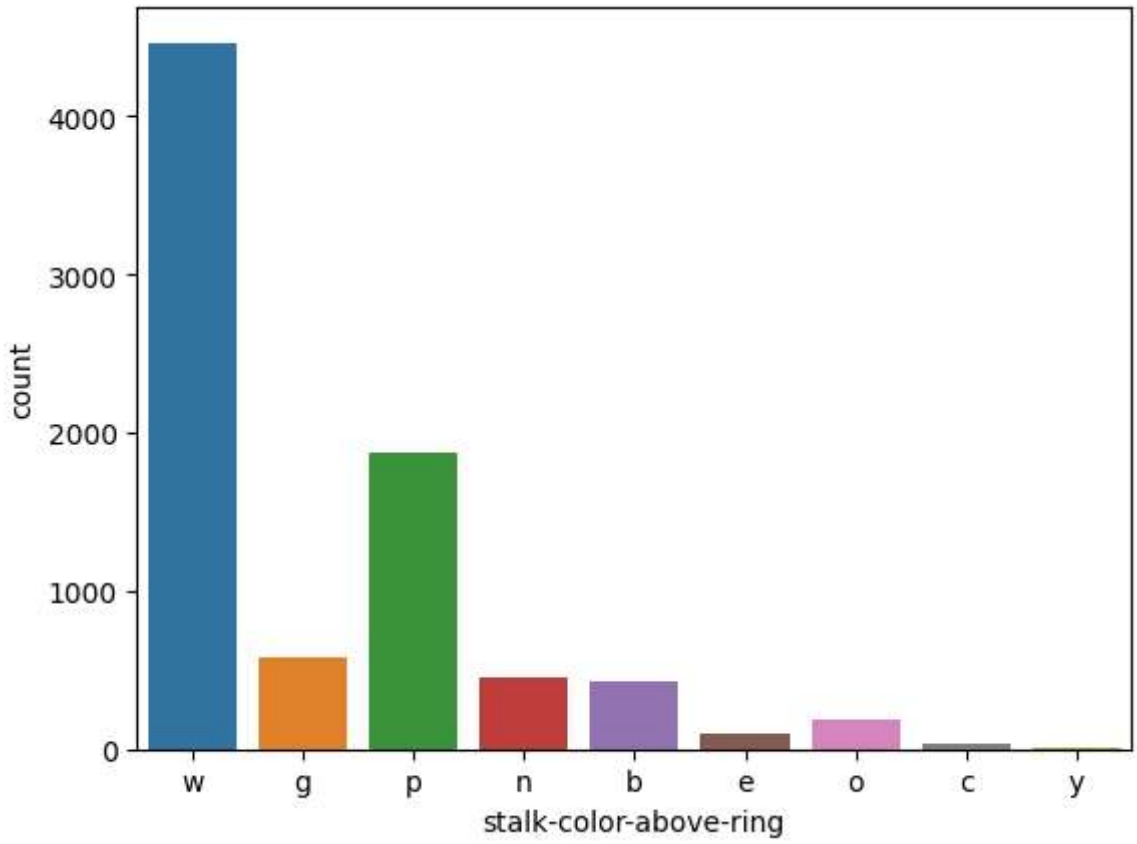

p
e
class
e
c
b
r
?
stalk-root

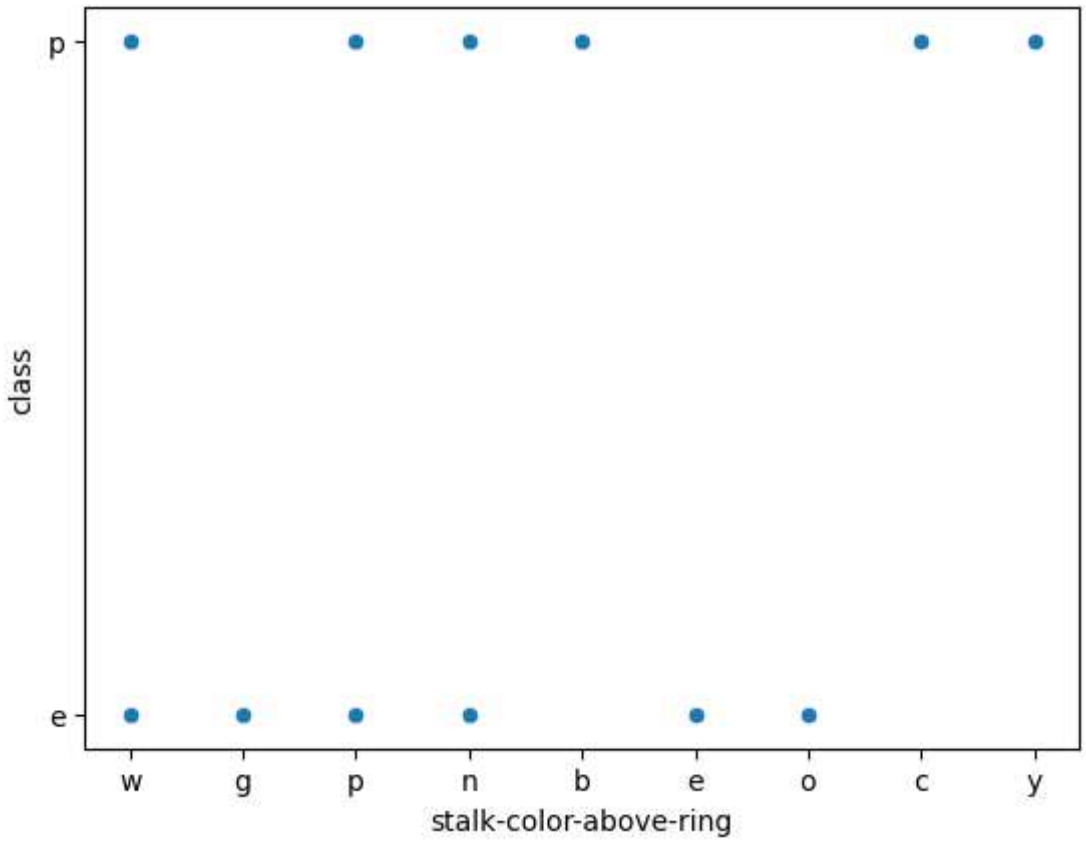

5000
4000
3000
2000
1000
0
count
s
f
k
y
stalk-surface-above-ring

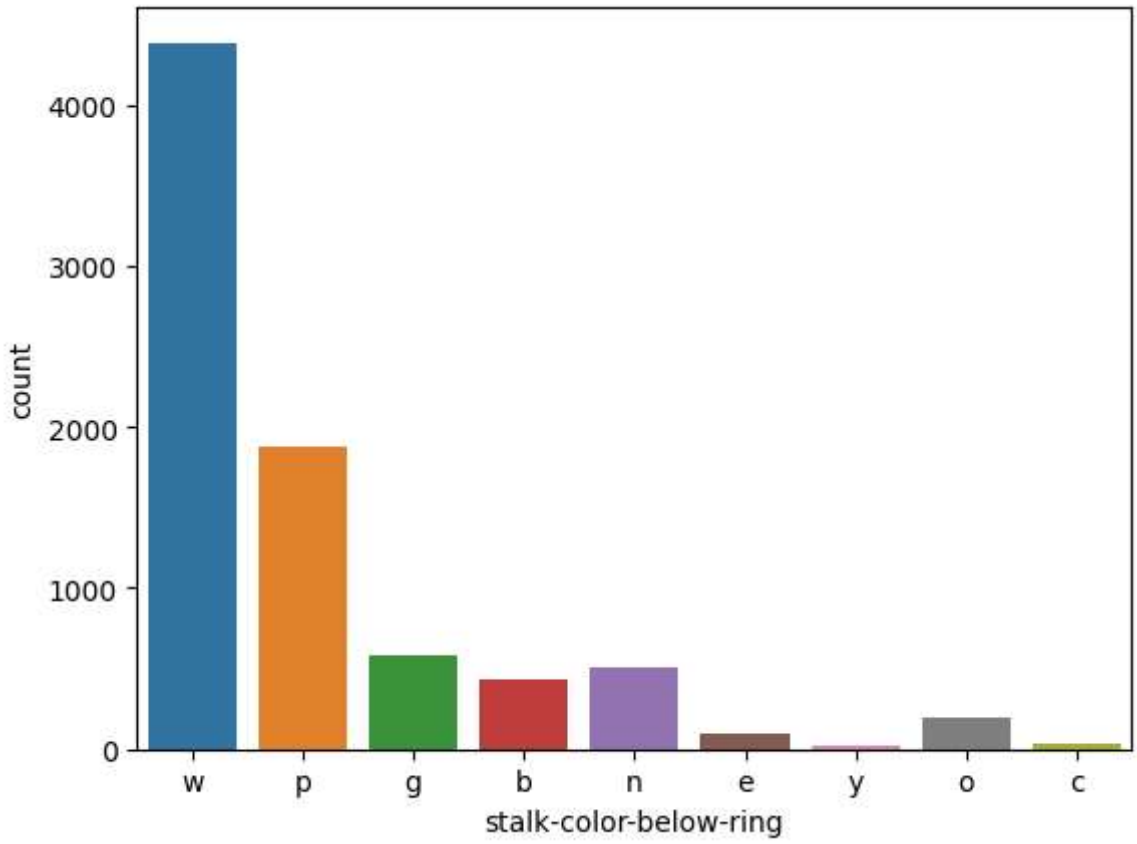

p
e
class
s
f
k
y
stalk-surface-above-ring

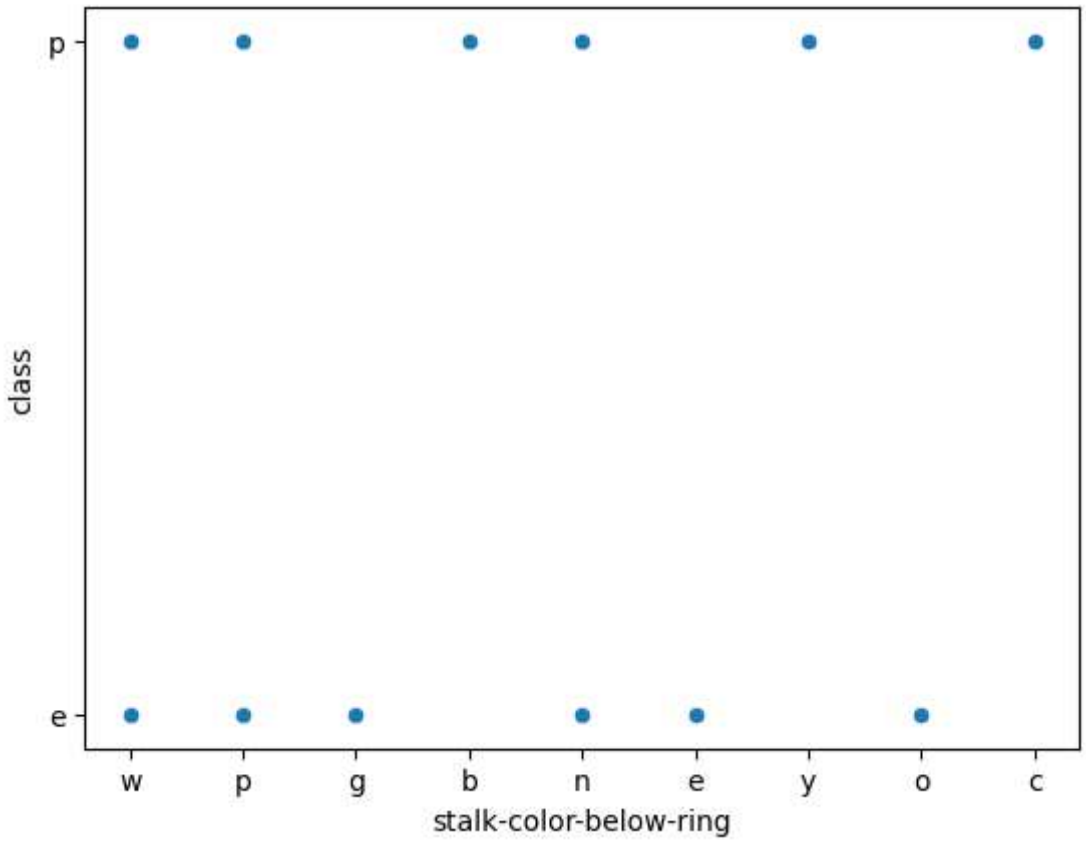

5000
4000
3000
2000
1000
0
count
s
f
y
k
stalk-surface-below-ring

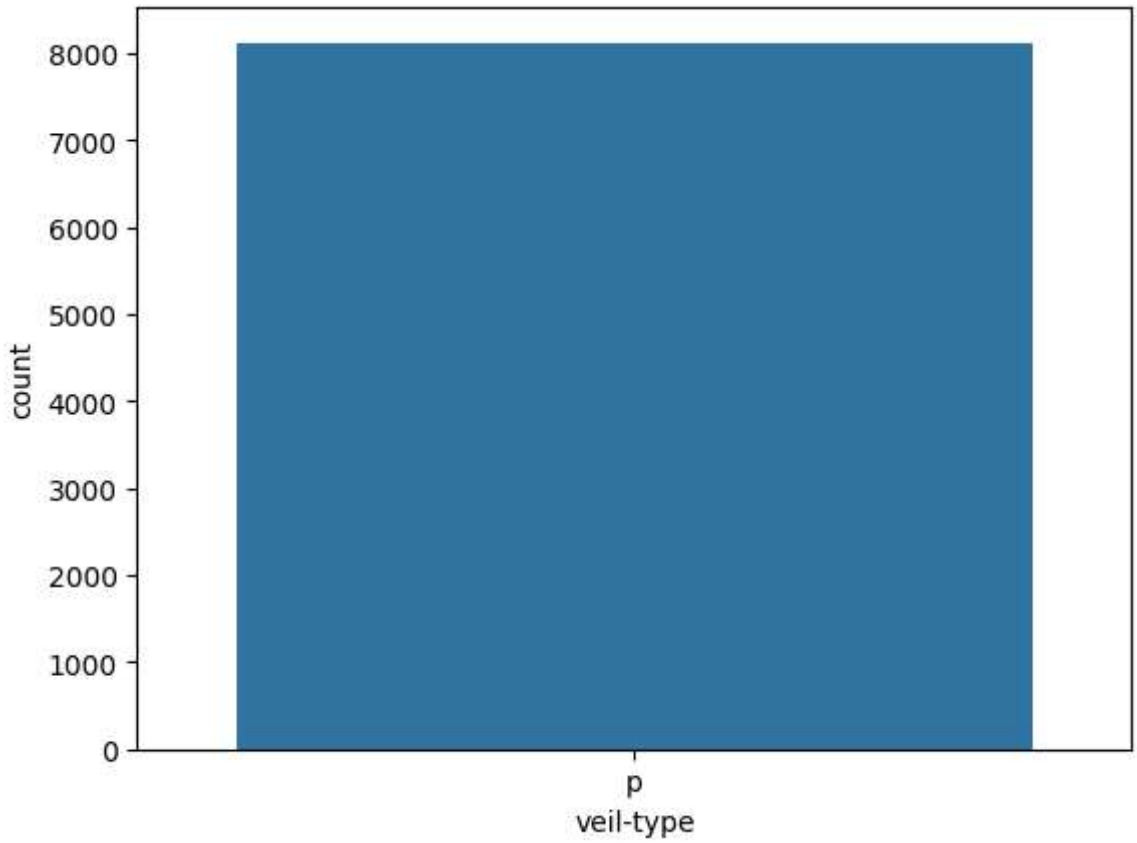

p
e
class
s
f
y
k
stalk-surface-below-ring

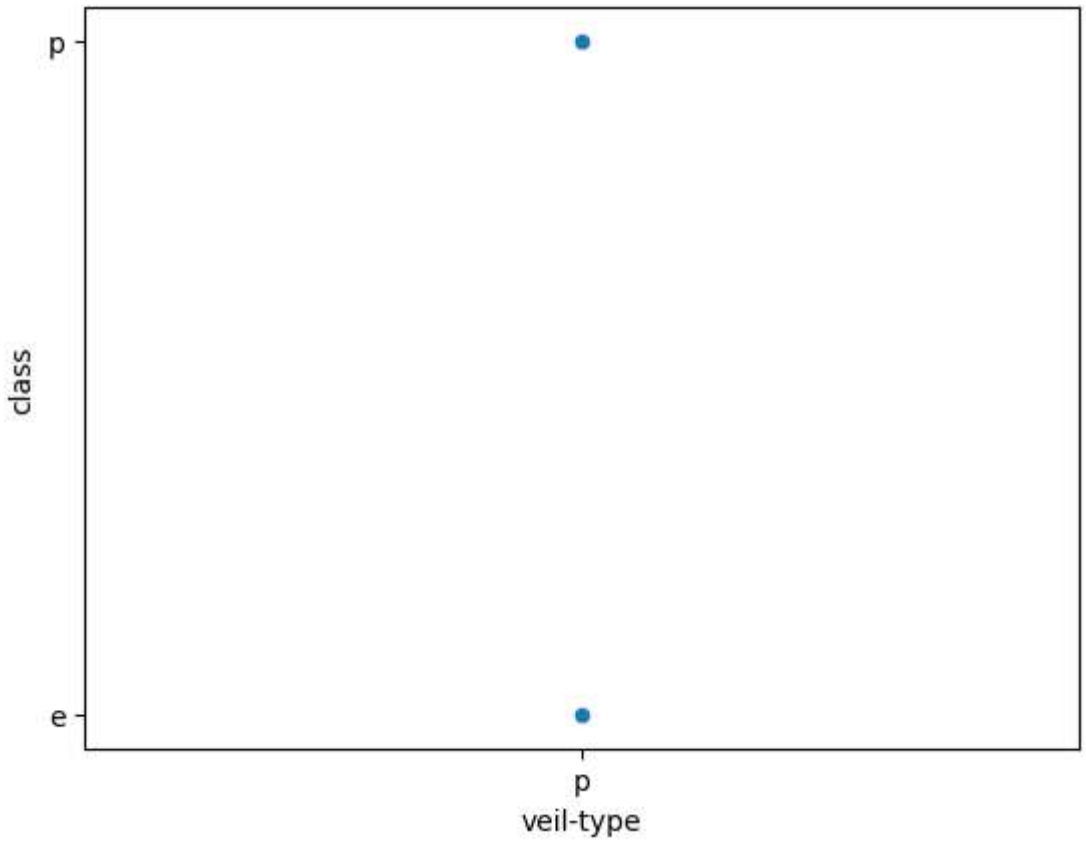

4000
3000
2000
1000
0
count
w
g
p
n
b
e
o
c
y
stalk-color-above-ring

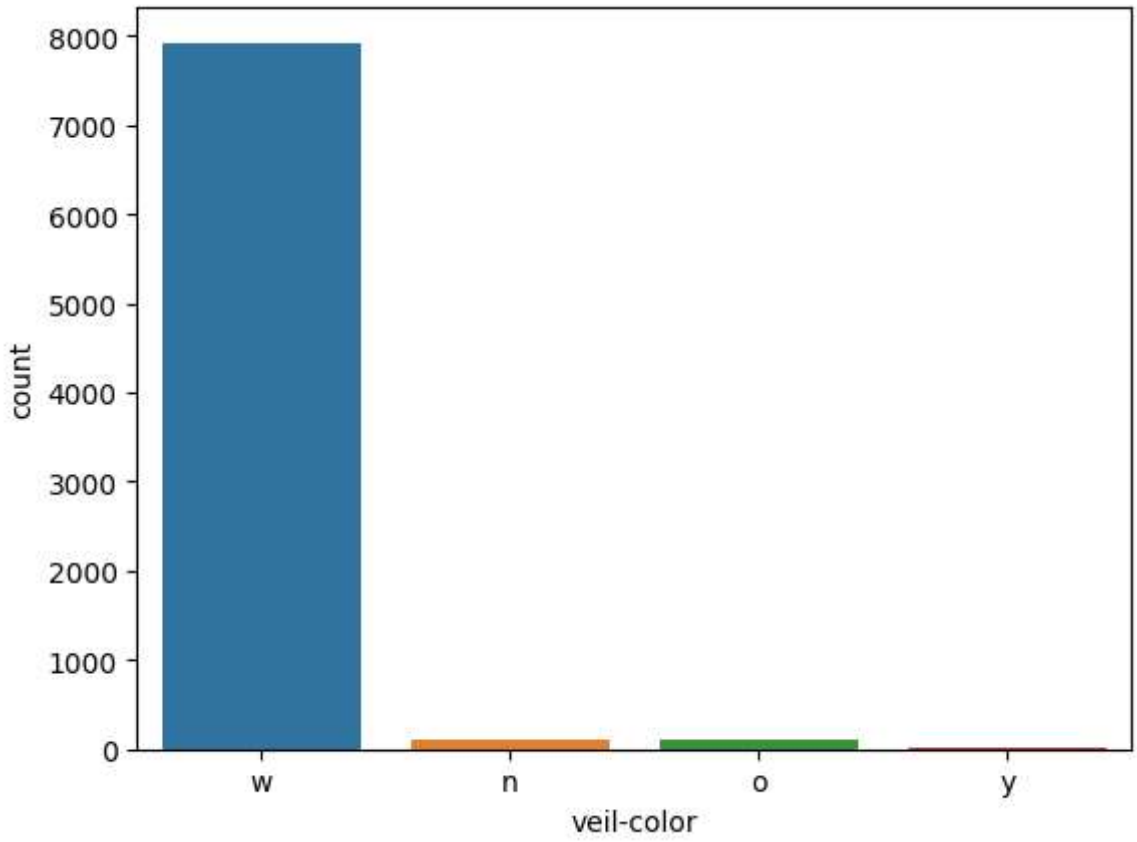

p
e
class
w
g
p
n
b
e
o
c
y
stalk-color-above-ring

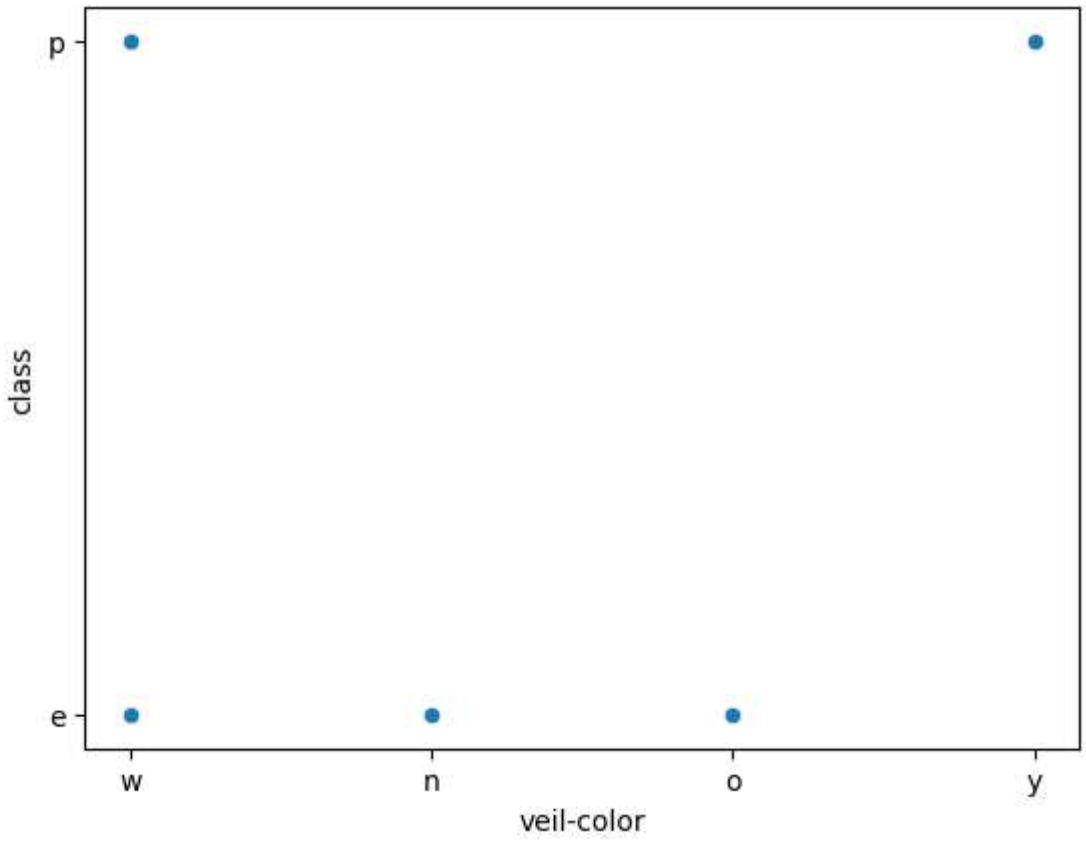

4000
3000
2000
1000
0
count
w
p
g
b
n
e
y
o
c
stalk-color-below-ring

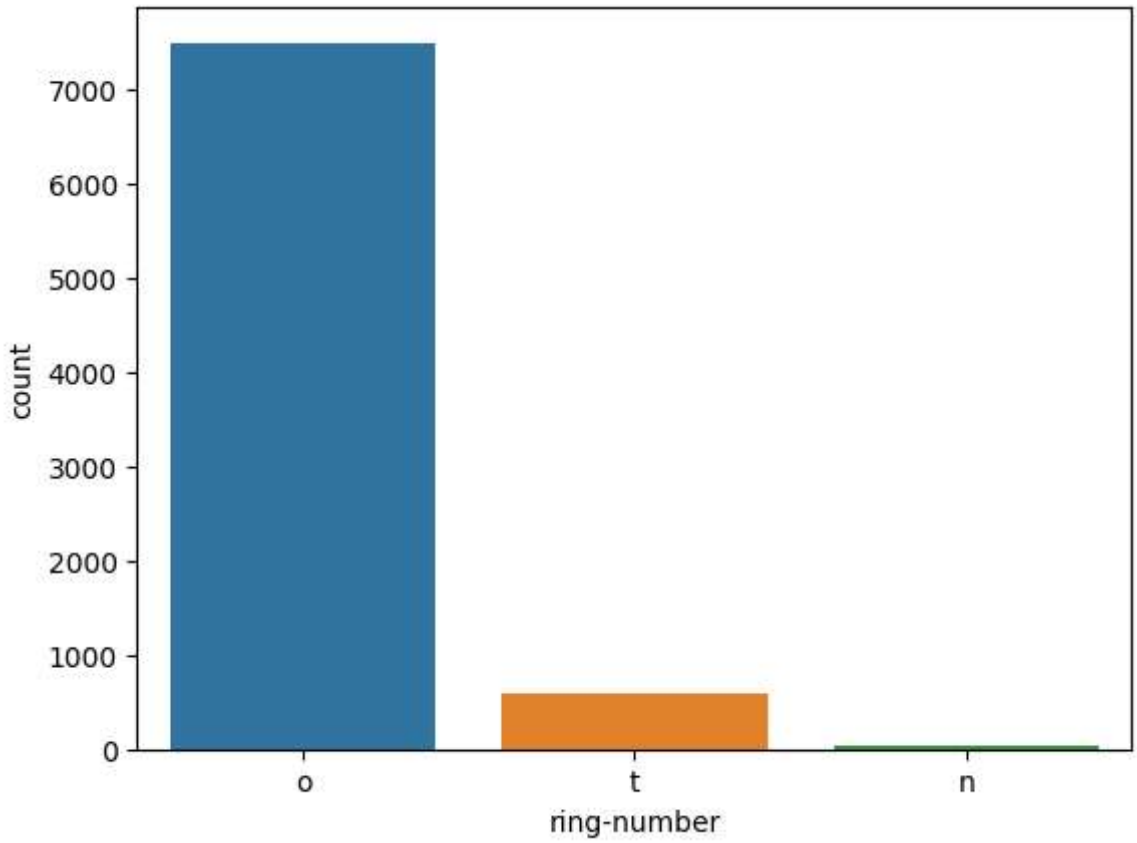

p
e
class
w
p
g
b
n
e
y
o
c
stalk-color-below-ring

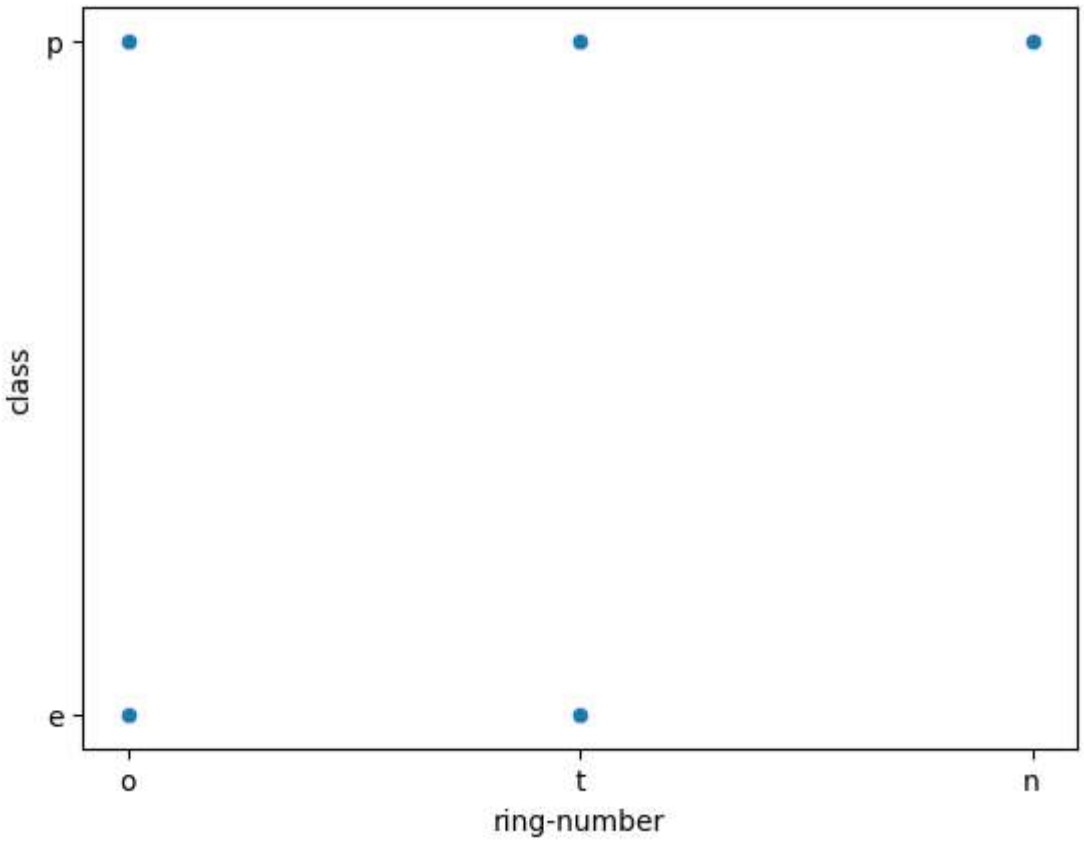

8000
7000
6000
5000
4000
3000
2000
1000
0
count
p
veil-type

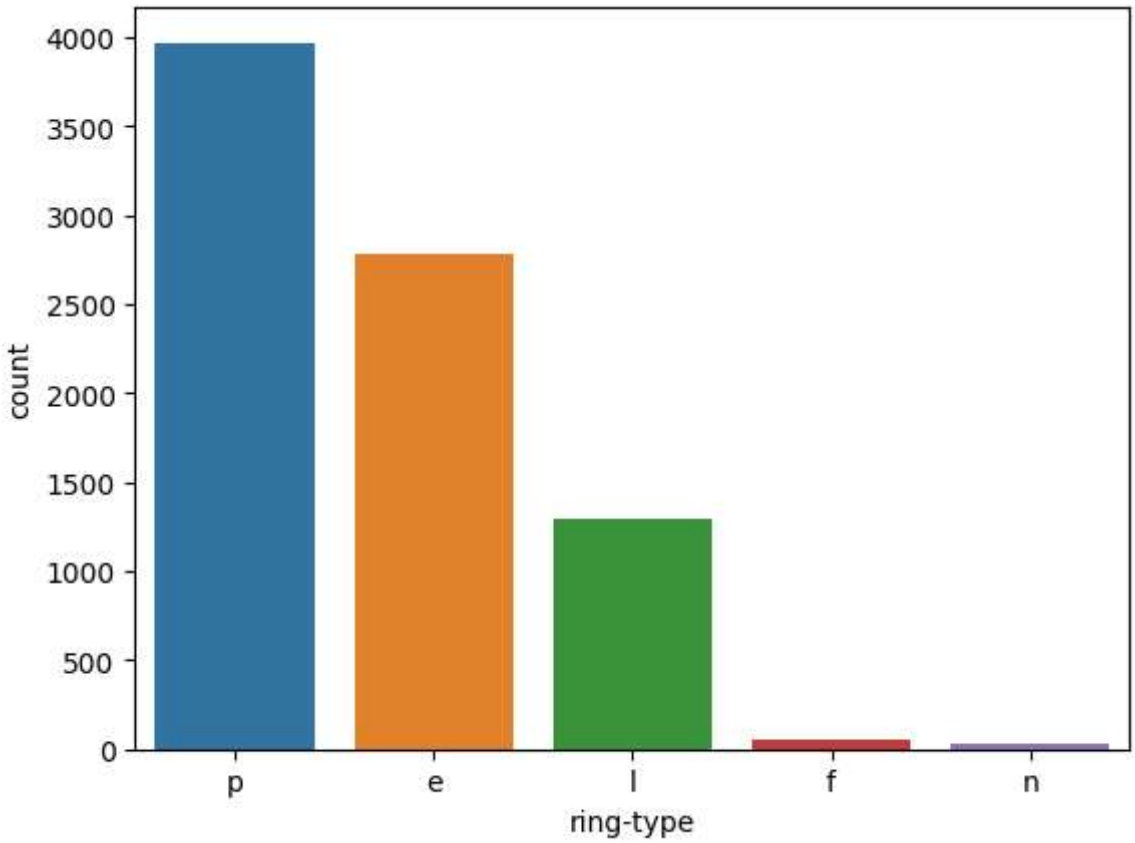

p
e
class
p
veil-type

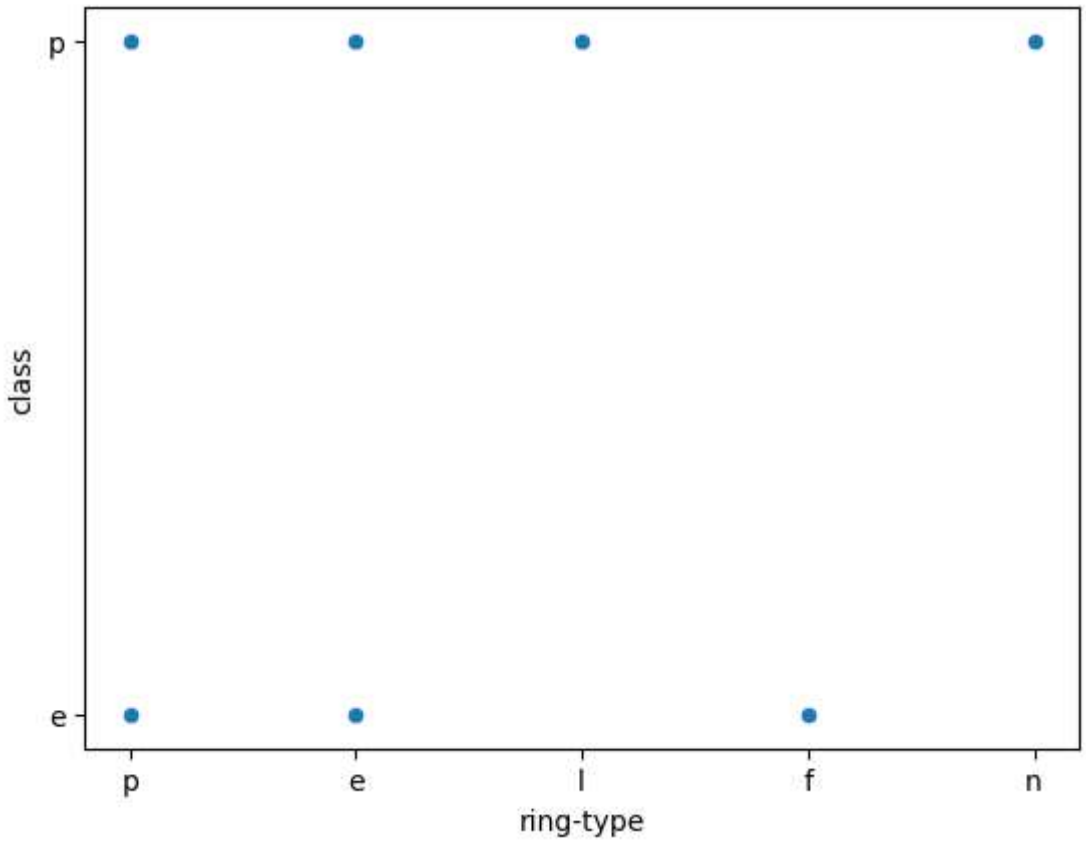

8000
7000
6000
5000
4000
3000
2000
1000
0
count
w
n
o
y
veil-color

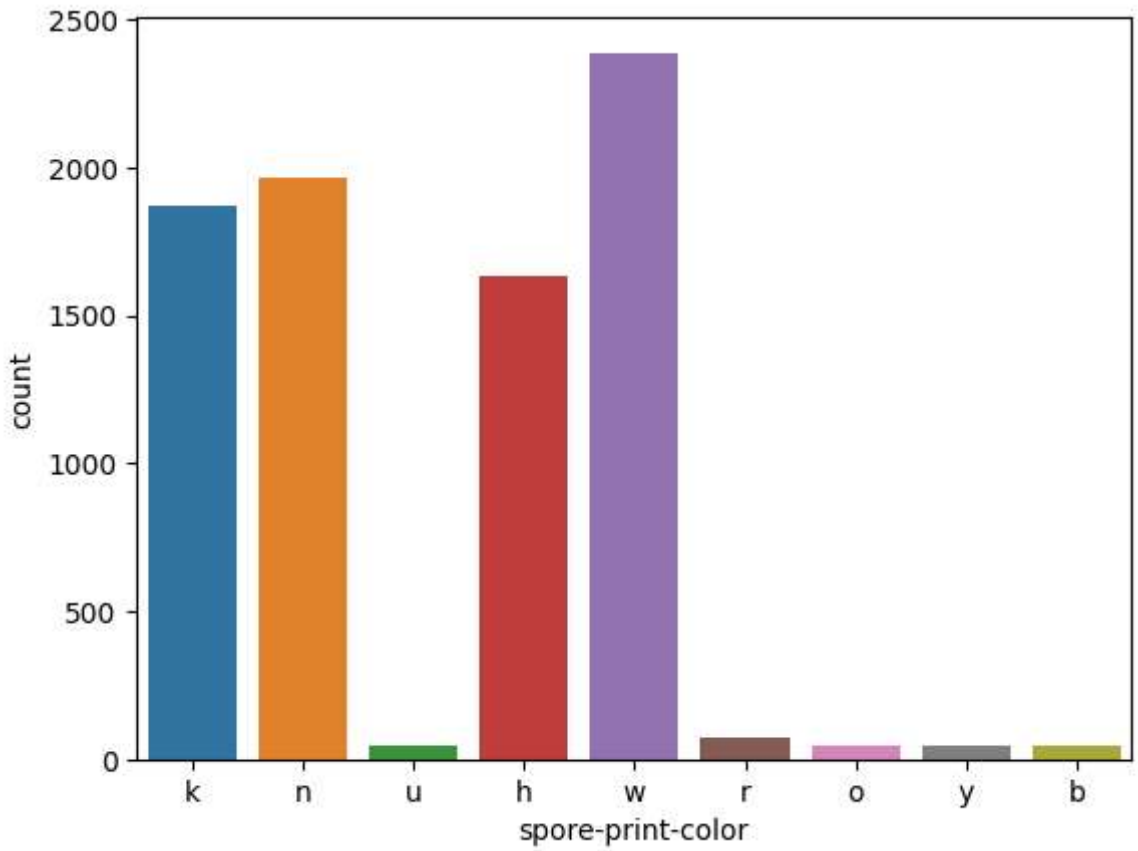

p
e
class
w
n
o
y
veil-color

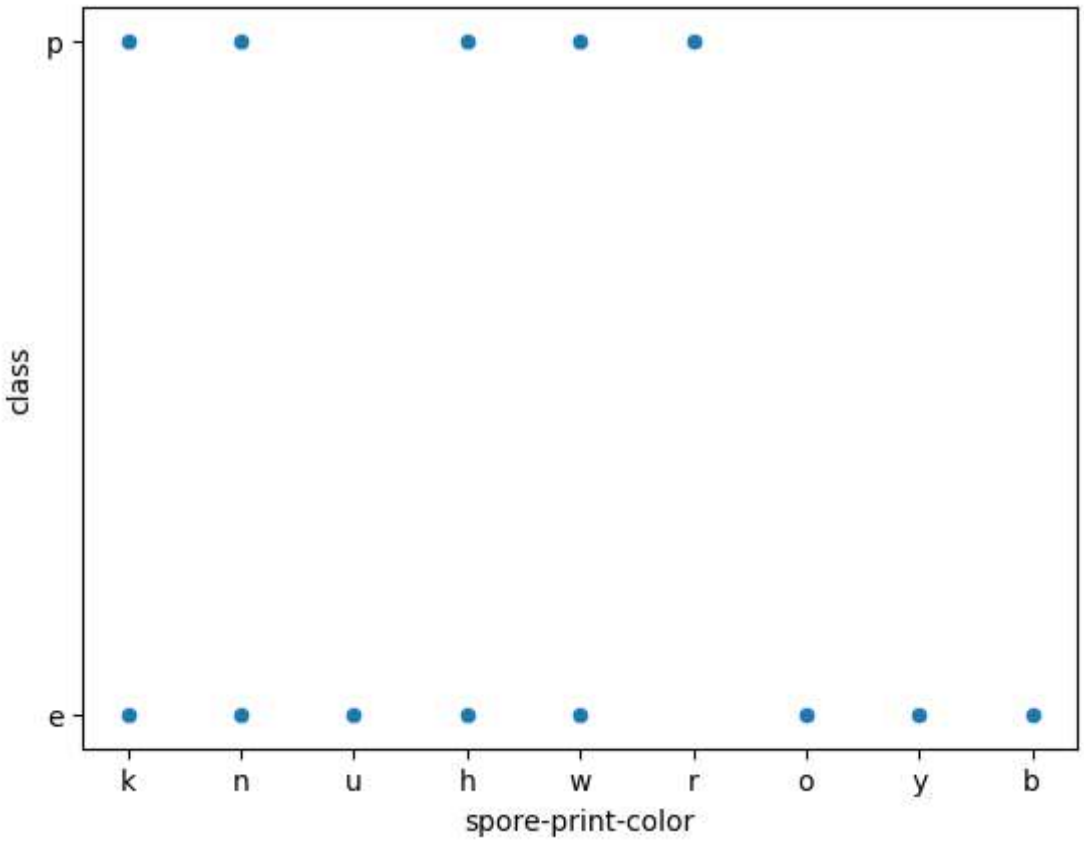

7000
6000
5000
4000
3000
2000
1000
0
count
o
t
n
ring-number

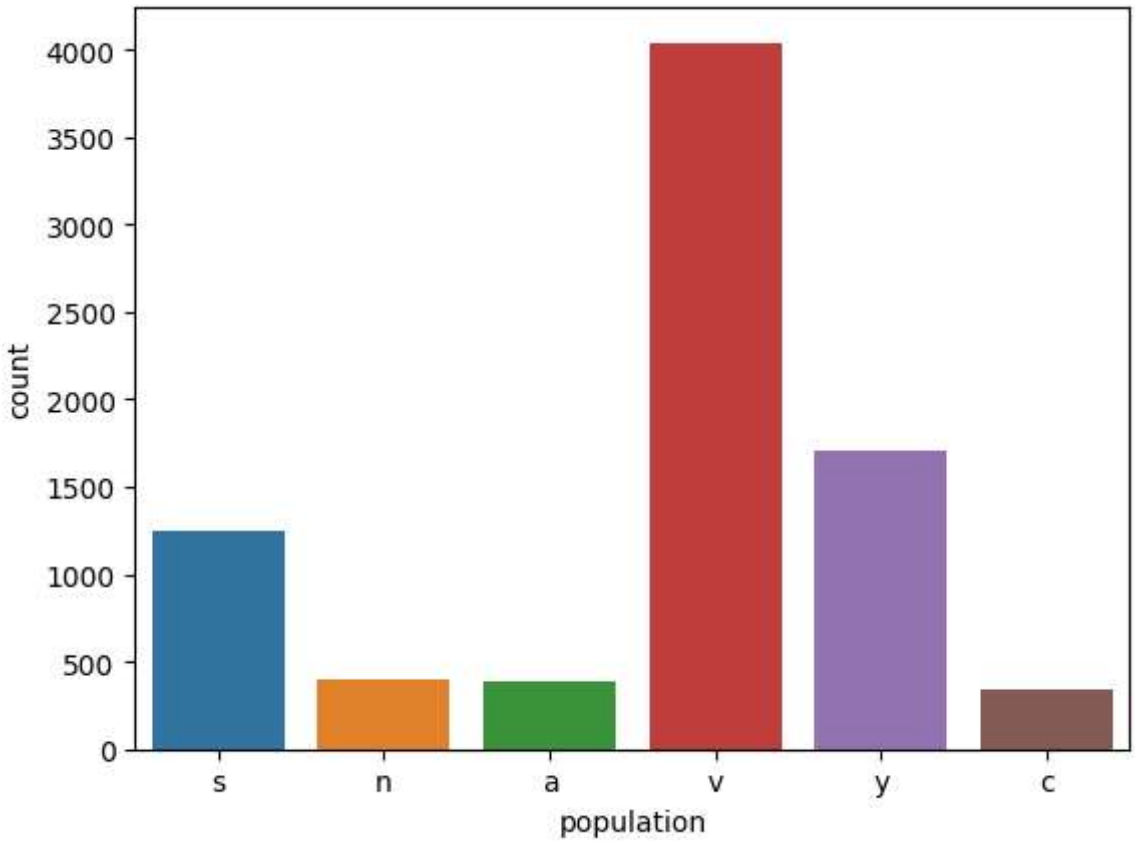

p
e
class
o
t
n
ring-number

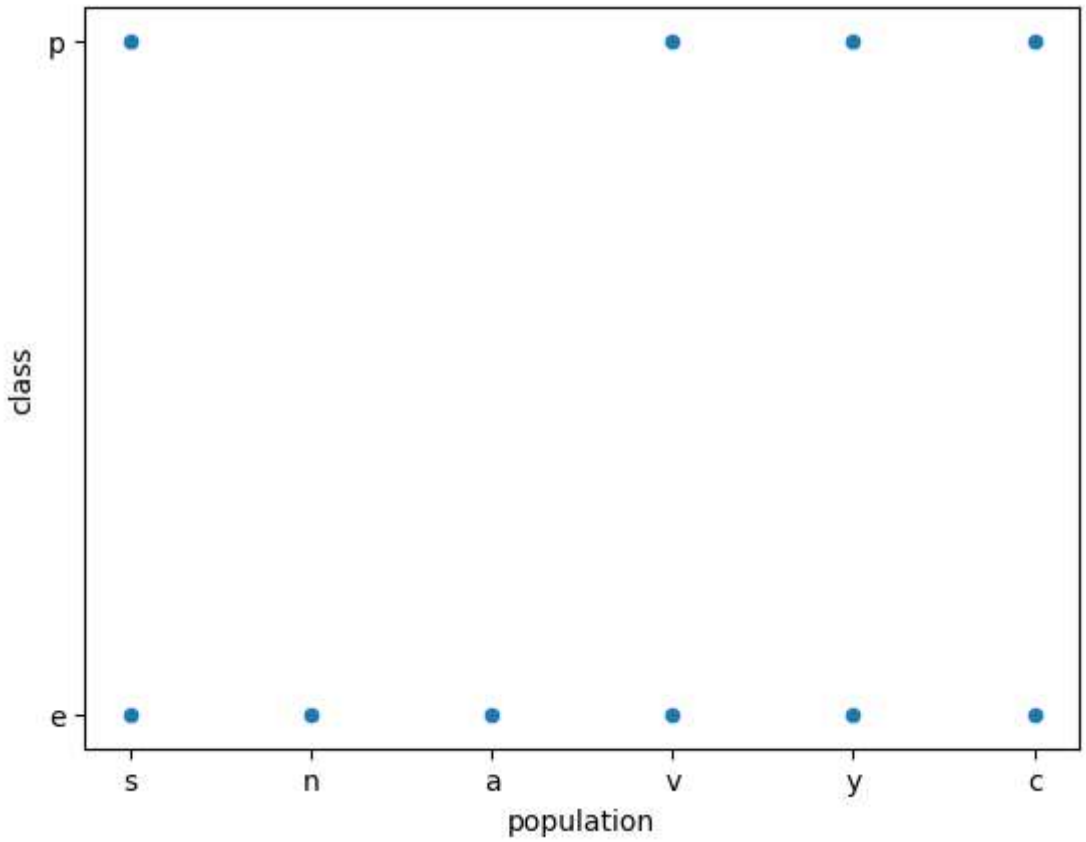

4000
3500
3000
2500
2000
1500
1000
500
0
count
p
e
l
f
n
ring-type

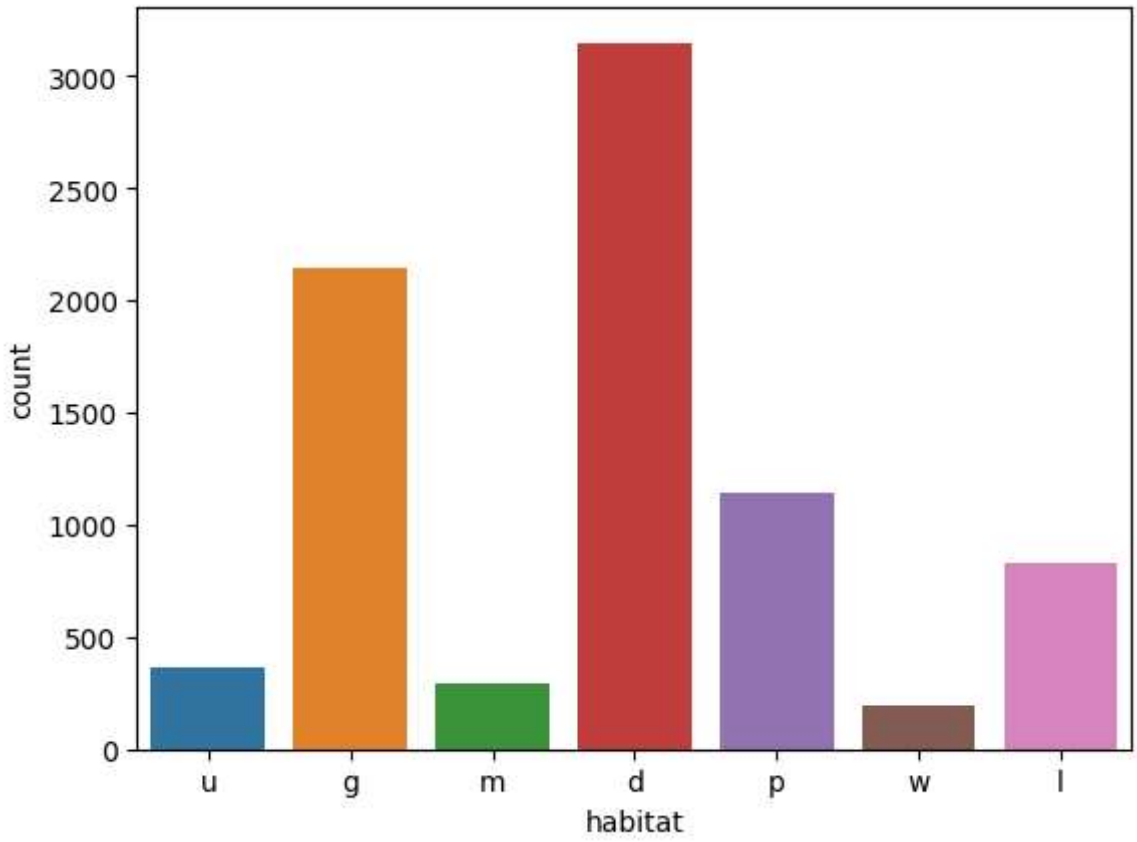

p
e
class
p
e
l
f
n
ring-type

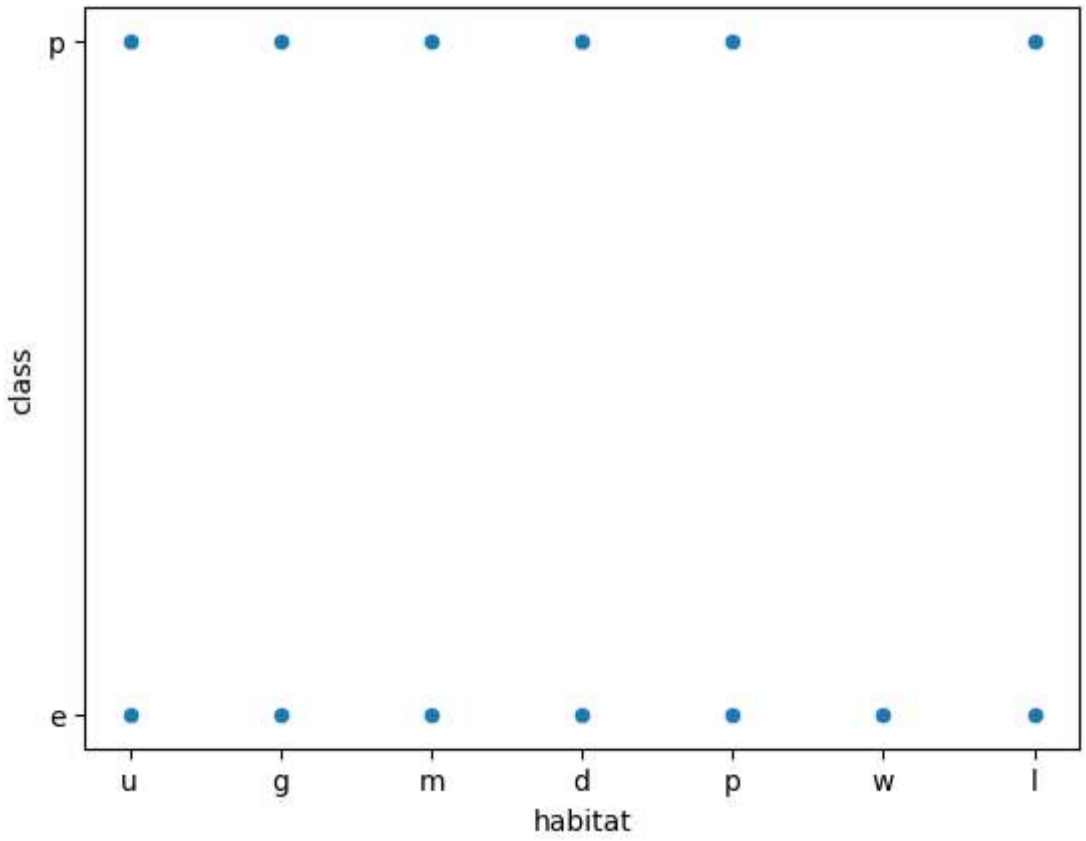

2500
2000
1500
1000
500
0
count
k
n
u
h
w
r
o
y
b
spore-print-color


p
e
class
k
n
u
h
w
r
o
y
b
spore-print-color


4000
3500
3000
2500
2000
1500
1000
500
0
count
s
n
a
v
y
c
population


p
e
class
s
n
a
v
y
c
population


3000
2500
2000
1500
1000
500
0
count
u
g
m
d
p
w
l
habitat



Observation

1. Some of the properties are qualitative and we need it to be quantitave so that we can use them in classification models.
2. Here veil-type only have p category meaning there is no information to gather from that data.
3. Other than that from the graph we can conclude that there is no noticable outliers
4. The dataset doesnt have any unfilled or NA cell

In [ ]:
```
df = pd.get_dummies(df)
df.head()
```

Out[ ]:

	class_e	class_p	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_f	cap-surface_g	su
0	0	1	0	0	0	0	0	1	0	0	
1	1	0	0	0	0	0	0	1	0	0	
2	1	0	1	0	0	0	0	0	0	0	
3	0	1	0	0	0	0	0	1	0	0	
4	1	0	0	0	0	0	0	1	0	0	

Here some details are unescacary creating some extra detail such as if poisonours it cant be edible. So we can remove one property of in the multiple categorical data.

In [ ]:
```
del df['class_p']
del df['cap-shape_x']
del df['cap-surface_y']
del df['cap-color_y']
del df['bruises_t']
del df['odor_y']
del df['gill-attachment_f']
del df['gill-spacing_w']
del df['gill-size_n']
del df['gill-color_y']
del df['stalk-shape_t']
del df['stalk-root_r']
del df['stalk-surface-above-ring_y']
del df['stalk-surface-below-ring_y']
del df['stalk-color-above-ring_y']
del df['stalk-color-below-ring_y']
del df['veil-color_y']
del df['ring-number_t']
del df['ring-type_p']
del df['spore-print-color_y']
del df['population_y']
del df['habitat_w']
```

In [ ]:
```
df.describe()
```

Out[ ]:

	class_e	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-surface_f
count	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000	8124.000000
mean	0.517971	0.055638	0.000492	0.387986	0.101920	0.003939	0.285574
std	0.499708	0.229235	0.022185	0.487321	0.302562	0.062641	0.451715
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 97 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   class_e                     8124 non-null   uint8
 1   cap-shape_b                 8124 non-null   uint8
 2   cap-shape_c                 8124 non-null   uint8
 3   cap-shape_f                 8124 non-null   uint8
 4   cap-shape_k                 8124 non-null   uint8
 5   cap-shape_s                 8124 non-null   uint8
 6   cap-surface_f               8124 non-null   uint8
 7   cap-surface_g               8124 non-null   uint8
 8   cap-surface_s               8124 non-null   uint8
 9   cap-color_b                 8124 non-null   uint8
 10  cap-color_c                 8124 non-null   uint8
 11  cap-color_e                 8124 non-null   uint8
 12  cap-color_g                 8124 non-null   uint8
 13  cap-color_n                 8124 non-null   uint8
 14  cap-color_p                 8124 non-null   uint8
 15  cap-color_r                 8124 non-null   uint8
 16  cap-color_u                 8124 non-null   uint8
 17  cap-color_w                 8124 non-null   uint8
 18  bruises_f                   8124 non-null   uint8
 19  odor_a                      8124 non-null   uint8
 20  odor_c                      8124 non-null   uint8
 21  odor_f                      8124 non-null   uint8
 22  odor_l                      8124 non-null   uint8
 23  odor_m                      8124 non-null   uint8
 24  odor_n                      8124 non-null   uint8
 25  odor_p                      8124 non-null   uint8
 26  odor_s                      8124 non-null   uint8
 27  gill-attachment_a           8124 non-null   uint8
 28  gill-spacing_c              8124 non-null   uint8
 29  gill-size_b                 8124 non-null   uint8
 30  gill-color_b                8124 non-null   uint8
 31  gill-color_e                8124 non-null   uint8
 32  gill-color_g                8124 non-null   uint8
 33  gill-color_h                8124 non-null   uint8
 34  gill-color_k                8124 non-null   uint8
 35  gill-color_n                8124 non-null   uint8
 36  gill-color_o                8124 non-null   uint8
 37  gill-color_p                8124 non-null   uint8
 38  gill-color_r                8124 non-null   uint8
 39  gill-color_u                8124 non-null   uint8
 40  gill-color_w                8124 non-null   uint8
 41  stalk-shape_e               8124 non-null   uint8
 42  stalk-root_?                8124 non-null   uint8
 43  stalk-root_b                8124 non-null   uint8
 44  stalk-root_c                8124 non-null   uint8
 45  stalk-root_e                8124 non-null   uint8
 46  stalk-surface-above-ring_f  8124 non-null   uint8
 47  stalk-surface-above-ring_k  8124 non-null   uint8
 48  stalk-surface-above-ring_s  8124 non-null   uint8
 49  stalk-surface-below-ring_f  8124 non-null   uint8
 50  stalk-surface-below-ring_k  8124 non-null   uint8
```

```
 51  stalk-surface-below-ring_s  8124 non-null   uint8
 52  stalk-color-above-ring_b    8124 non-null   uint8
 53  stalk-color-above-ring_c    8124 non-null   uint8
 54  stalk-color-above-ring_e    8124 non-null   uint8
 55  stalk-color-above-ring_g    8124 non-null   uint8
 56  stalk-color-above-ring_n    8124 non-null   uint8
 57  stalk-color-above-ring_o    8124 non-null   uint8
 58  stalk-color-above-ring_p    8124 non-null   uint8
 59  stalk-color-above-ring_w    8124 non-null   uint8
 60  stalk-color-below-ring_b    8124 non-null   uint8
 61  stalk-color-below-ring_c    8124 non-null   uint8
 62  stalk-color-below-ring_e    8124 non-null   uint8
 63  stalk-color-below-ring_g    8124 non-null   uint8
 64  stalk-color-below-ring_n    8124 non-null   uint8
 65  stalk-color-below-ring_o    8124 non-null   uint8
 66  stalk-color-below-ring_p    8124 non-null   uint8
 67  stalk-color-below-ring_w    8124 non-null   uint8
 68  veil-type_p                 8124 non-null   uint8
 69  veil-color_n                8124 non-null   uint8
 70  veil-color_o                8124 non-null   uint8
 71  veil-color_w                8124 non-null   uint8
 72  ring-number_n               8124 non-null   uint8
 73  ring-number_o               8124 non-null   uint8
 74  ring-type_e                 8124 non-null   uint8
 75  ring-type_f                 8124 non-null   uint8
 76  ring-type_l                 8124 non-null   uint8
 77  ring-type_n                 8124 non-null   uint8
 78  spore-print-color_b         8124 non-null   uint8
 79  spore-print-color_h         8124 non-null   uint8
 80  spore-print-color_k         8124 non-null   uint8
 81  spore-print-color_n         8124 non-null   uint8
 82  spore-print-color_o         8124 non-null   uint8
 83  spore-print-color_r         8124 non-null   uint8
 84  spore-print-color_u         8124 non-null   uint8
 85  spore-print-color_w         8124 non-null   uint8
 86  population_a                8124 non-null   uint8
 87  population_c                8124 non-null   uint8
 88  population_n                8124 non-null   uint8
 89  population_s                8124 non-null   uint8
 90  population_v                8124 non-null   uint8
 91  habitat_d                   8124 non-null   uint8
 92  habitat_g                   8124 non-null   uint8
 93  habitat_l                   8124 non-null   uint8
 94  habitat_m                   8124 non-null   uint8
 95  habitat_p                   8124 non-null   uint8
 96  habitat_u                   8124 non-null   uint8
dtypes: uint8(97)
memory usage: 769.7 KB
```

In [ ]:
```
df.head()
```

Out[ ]:

	class_e	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-surface_f	cap-surface_g	cap-surface_s	cap-color_b
0	0	0	0	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	0	1	0
2	1	1	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	1	0

We are finished with the data preprocessing and cleaning. Now for the next stage model generation.

1. Logistic regression

Logistic regression is a statistical method that models the relationship between a dependent variable and one or more independent variables, with the goal of predicting the probability of a binary outcome. It involves estimating the parameters of a logistic function that maps the input variables to the output probability, which is an S-shaped curve ranging from 0 to 1. Logistic regression is commonly used for classification

In [ ]:
```
x = df.loc[:, df.columns != 'class_e']
```

In [ ]:
```
y = df['class_e']
```

The dataset need to be seperated for both training and testing seperately

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state
```

In [ ]:
```
clf_lr = LogisticRegression()
clf_lr.fit(x_train,y_train)
```

Out[ ]:
```
LogisticRegression()
```

In [ ]:
```
clf_lr.coef_
```

Out[ ]:
```
array([[-3.69795044e-01, -6.42273299e-01, -4.18302461e-02,
        -5.83909616e-02,  4.50087706e-01,  8.42361768e-01,
        -7.95531066e-01, -1.17853158e-01, -8.30933009e-01,
         9.90635777e-01,  3.39597932e-02,  4.49434312e-02,
         4.13572259e-01, -8.21681854e-01,  3.26060735e-01,
         3.30693882e-01, -5.22137015e-01,  5.09963189e-03,
         3.01425232e+00, -2.84619228e+00, -2.58665054e+00,
         3.07782844e+00, -3.24496851e-01,  4.13236390e+00,
        -2.47201130e+00, -6.86463244e-01,  2.21418196e-01,
        -2.31578589e+00,  3.62943750e+00, -2.09565670e+00,
         7.10477148e-01,  3.88863675e-02, -4.02347034e-02,
         4.30466853e-01,  6.56587246e-01,  1.51056799e-01,
         5.43847711e-01, -6.94718848e-01,  3.44183500e-01,
         3.21910080e-01, -9.80524558e-01,  7.40388039e-01,
        -1.98301801e+00,  4.94543848e-01, -3.32764770e-01,
         1.01685599e+00, -1.78811688e+00,  1.49207080e+00,
         1.53006751e+00,  1.82210295e-01,  1.00513213e+00,
        -1.22274128e-01, -3.24496851e-01,  6.62223602e-01,
         4.49099571e-01,  2.26751398e-01,  3.97634289e-01,
        -1.90475300e-01,  3.11425631e-02, -7.81729345e-02,
        -3.24496851e-01,  5.65895230e-01,  4.06062339e-01,
         1.24460164e+00,  3.97634289e-01, -2.67565996e-01,
        -1.70577530e-01,  3.42168469e-04,  2.08185971e-01,
         1.89448318e-01,  7.31970854e-01, -3.24496851e-01,
        -6.10354631e-01, -8.51657197e-01,  1.55421885e+00,
        -4.22329717e-01, -3.24496851e-01,  1.37038080e-01,
        -9.31526518e-01,  1.57739225e+00,  1.76320536e+00,
         1.04859873e-01, -3.99251401e+00,  1.56653707e+00,
        -3.54369513e-01,  4.09047504e-02, -1.62120204e+00,
         7.72094070e-01, -5.58526103e-01, -9.20230365e-01,
         3.67625220e-02, -6.68396629e-01, -2.11113198e-01,
        -8.03104713e-01,  2.75487921e-01, -4.18416844e-01]])
```

In [ ]:
```
clf_lr.intercept_
```

Out[ ]:
```
array([-1.59972312])
```

Predicting and confusion matrix

In [ ]:
```
clf_lr.predict_proba(x)
```

Out[ ]:
```
array([[0.98005144, 0.01994856],
       [0.00100532, 0.99899468],
       [0.00209836, 0.99790164],
       ...,
       [0.0026786 , 0.9973214 ],
       [0.99628281, 0.00371719],
       [0.00510283, 0.99489717]])
```

In [ ]:
```
y_pred = clf_lr.predict(x_test)
y_pred
```

Out[ ]:
```
array([0, 0, 1, ..., 1, 1, 1], dtype=uint8)
```

In [ ]:
```
confusion_matrix(y_test, y_pred)
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

In [ ]:
```
precision_score(y_test, y_pred)
```

Out[ ]:
```
1.0
```

In [ ]:
```
recall_score(y_test, y_pred)
```

Out[ ]:
```
1.0
```

In [ ]:
```
roc_auc_score(y_test, y_pred)
```

Out[ ]:
```
1.0
```

Here the model predicts the testing portion accuratley.

2. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a statistical method that analyzes the relationship between a dependent variable and one or more independent variables, with the goal of finding a linear combination of the input variables that maximizes the separation between two or more classes. It involves modeling the distribution of the input variables for each class and using this information to estimate the probability of a new observation belonging to each class. LDA assumes that the input variables are normally distributed and that the covariance matrix is equal across classes. LDA is commonly used for dimensionality reduction, pattern recognition, and classification problems

In [ ]:
```
clf_lda = LinearDiscriminantAnalysis()
clf_lda.fit(x_train, y_train)
```

Out[ ]:
```
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

In [ ]:
```
y_pred_lda = clf_lda.predict(x_test)
```

In [ ]:
```
y_pred_lda
```

Out[ ]:
```
array([0, 0, 1, ..., 1, 1, 1], dtype=uint8)
```

In [ ]:
```
confusion_matrix(y_test, y_pred_lda)
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

In [ ]:
```
precision_score(y_test, y_pred)
```

Out[ ]:
```
1.0
```

In [ ]:
```
recall_score(y_test, y_pred)
```

Out[ ]:
```
1.0
```

In [ ]:
```
roc_auc_score(y_test, y_pred)
```

Out[ ]:
```
1.0
```

The model predicts the test portion accurately

3. K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric machine learning algorithm used for classification and regression problems. It works by finding the k-nearest data points in the training set to a new observation, and then classifying or predicting the outcome based on the most common class or average value of the k-nearest neighbors. The value of k is a user-defined parameter that affects the model performance and can be tuned using cross-validation. KNN can handle complex decision boundaries and can be used for both binary and multi-class classification problems. It is a simple algorithm that does not require any assumptions about the underlying distribution of the data, but it can be computationally expensive and sensitive to the choice of distance metric and the scaling of the input variables.

In [ ]:
```
scaler = preprocessing.StandardScaler().fit(x_train)
X_train_s= scaler.transform(x_train)
```

In [ ]:
```
scaler = preprocessing.StandardScaler().fit(x_test)
x_test_s= scaler.transform(x_test)
```

In [ ]:
```
x_test_s
```

Out[ ]:
```
array([[-0.24638377,  0.        , -0.82069022, ..., -0.18361648,
        -0.39362366, -0.24356115],
       [-0.24638377,  0.        ,  1.21848655, ..., -0.18361648,
        -0.39362366, -0.24356115],
       [-0.24638377,  0.        , -0.82069022, ..., -0.18361648,
        -0.39362366, -0.24356115],
       ...,
       [-0.24638377,  0.        , -0.82069022, ..., -0.18361648,
        -0.39362366, -0.24356115],
       [-0.24638377,  0.        ,  1.21848655, ..., -0.18361648,
        -0.39362366, -0.24356115],
       [-0.24638377,  0.        ,  1.21848655, ..., -0.18361648,
        -0.39362366, -0.24356115]])
```

In [ ]:
```
clf_knn_1 = KNeighborsClassifier(n_neighbors=1)
clf_knn_1.fit(x_train, y_train)
```

Out[ ]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

In [ ]:
```
confusion_matrix(y_test, clf_knn_1.predict(x_test_s))
```

```
e:\programes\python\lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

In [ ]:
```
accuracy_score(y_test, clf_knn_1.predict(x_test_s))
```

```
e:\programes\python\lib\site-packages\sklearn\base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

Out[ ]:
```
1.0
```

In [ ]:
```
clf_knn_3 = KNeighborsClassifier(n_neighbors=3)
clf_knn_3.fit(X_train_s, y_train)
accuracy_score(y_test, clf_knn_3.predict(x_test_s))
```

Out[ ]:
```
1.0
```

In [ ]:
```
params = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20,30]}
```

In [ ]:
```
grid_search_cv = GridSearchCV(KNeighborsClassifier(), params)
```

In [ ]:
```
grid_search_cv.fit(X_train_s, y_train)
```

Out[ ]:
```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

In [ ]:
```
grid_search_cv.best_params_
```

Out[ ]:
```
{'n_neighbors': 1}
```

In [ ]:
```
optimised_KNN = grid_search_cv.best_estimator_
```

In [ ]:
```
y_test_pred = optimised_KNN.predict(x_test_s)
```

In [ ]:
```
confusion_matrix(y_test, y_test_pred)
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

In [ ]:
```
accuracy_score(y_test, y_test_pred)
```

Out[ ]:
```
1.0
```

The model predicts the testing portion accurately, but it requires some time to train it.

4. Classification Tree

Decision tree is a machine learning algorithm used for both classification and regression tasks. It works by constructing a tree-like model of decisions and their possible consequences based on the input variables. Each internal node of the tree represents a decision based on a specific input variable, while each leaf node represents the outcome or prediction. The algorithm builds the tree recursively by splitting the data at each node based on the input variable that provides the most information gain or reduction in entropy. Decision trees can handle both categorical and continuous input variables and can handle missing values. They can be easily visualized and interpreted, and can handle non-linear relationships between input and output variables. However, decision trees can be prone to overfitting, especially if the tree is too deep or if there are many noisy or irrelevant input variables.

In [ ]:
```
clftree = tree.DecisionTreeClassifier(max_depth = 3)
```

In [ ]:
```
clftree.fit(x_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3)
```

In [ ]:
```
y_train_pred = clftree.predict(x_train)
y_test_pred = clftree.predict(x_test)
```

In [ ]:
```
confusion_matrix(y_test, y_test_pred)
```

Out[ ]:
```
array([[774,   9],
       [ 32, 810]], dtype=int64)
```

In [ ]:
```
accuracy_score(y_test, y_test_pred)
```

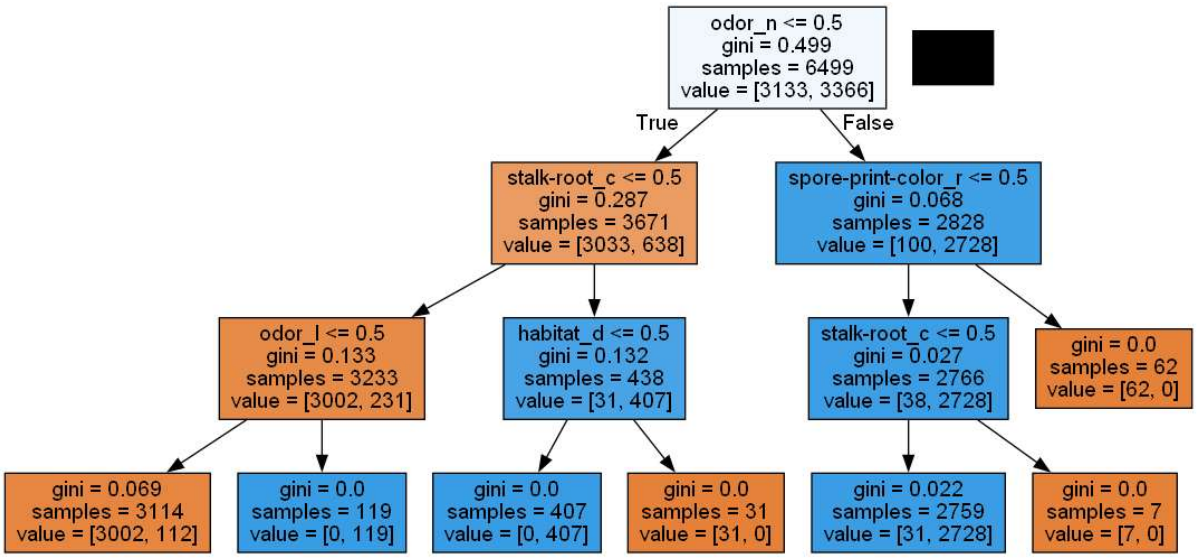
Out[ ]:
```
0.9747692307692307
```

Decision tree plot

In [ ]:
```
dot_data = tree.export_graphviz(clftree, out_file=None,feature_names= x_train.colum
```

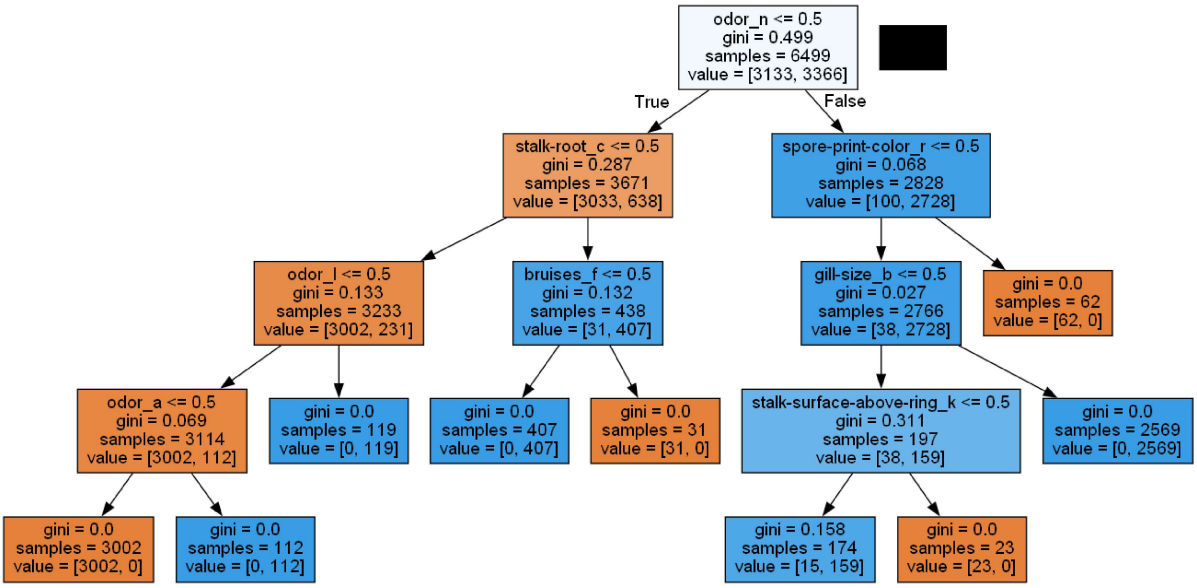
In [ ]:
```
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

Out[ ]:



In [ ]:
```
clftree2 = tree.DecisionTreeClassifier(min_samples_leaf = 20, max_depth=4)
clftree2.fit(x_train, y_train)
dot_data = tree.export_graphviz(clftree2, out_file=None,feature_names= x_train.colu
graph2 = pydotplus.graph_from_dot_data(dot_data)
Image(graph2.create_png())
```

Out[ ]:



In [ ]:
```
accuracy_score(y_test, clftree2.predict(x_test))
```

Out[ ]:
```
0.9993846153846154
```

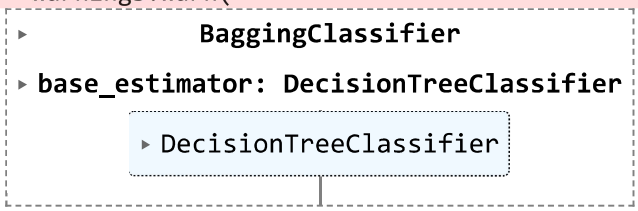
Bagging

In [ ]:
```
bag_clf = BaggingClassifier(base_estimator=clftree, n_estimators=1000,
                            bootstrap=True, n_jobs=-1,
                            random_state=42)
```

In [ ]:
```
bag_clf.fit(x_train, y_train)
```

```
e:\programes\python\lib\site-packages\sklearn\ensemble\_base.py:166: FutureWarnin
g: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed
in 1.4.
  warnings.warn(
```

Out[ ]:



In [ ]:
```
confusion_matrix(y_test, bag_clf.predict(x_test))
```

Out[ ]:
```
array([[774,   9],
       [ 32, 810]], dtype=int64)
```

In [ ]:
```
accuracy_score(y_test, bag_clf.predict(x_test))
```

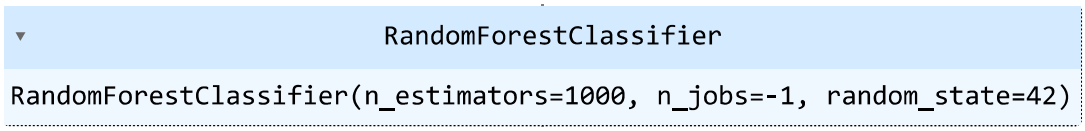
Out[ ]:
```
0.9747692307692307
```

5. Random Forest

In [ ]:
```
rf_clf = RandomForestClassifier(n_estimators=1000, n_jobs=-1 ,random_state=42)
```

In [ ]:
```
rf_clf.fit(x_train, y_train)
```

Out[ ]:



In [ ]:
```
confusion_matrix(y_test, rf_clf.predict(x_test))
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

In [ ]:
```
accuracy_score(y_test, rf_clf.predict(x_test))
```

Out[ ]:
```
1.0
```

In [ ]:
```
rf_clf = RandomForestClassifier(n_estimators=250,random_state=42)
```

In [ ]:
```
params_grid = {"max_features" : [4,5,6,7,8,9,10],
               "min_samples_split": [2, 3, 10],
               }
grid_search = GridSearchCV(rf_clf, params_grid,
                           n_jobs=-1, cv=5, scoring='accuracy')
```

In [ ]:
```
grid_search.fit(x_train, y_train)
```

Out[ ]:
```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [ ]:
```
cvrf_clf = grid_search.best_estimator_
accuracy_score(y_test, cvrf_clf.predict(x_test))
```

Out[ ]:
```
1.0
```

In [ ]:
```
confusion_matrix(y_test, cvrf_clf.predict(x_test))
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

Gradient Boosting

In [ ]:
```
gbc_clf = GradientBoostingClassifier()
gbc_clf.fit(x_train, y_train)
```

Out[ ]:
```
GradientBoostingClassifier
GradientBoostingClassifier()
```

In [ ]:
```
accuracy_score(y_test, gbc_clf.predict(x_test))
```

Out[ ]:
```
1.0
```

In [ ]:
```
gbc_clf2 = GradientBoostingClassifier(learning_rate =0.02, n_estimators =1000, max_
gbc_clf2.fit(x_train, y_train)
```

Out[ ]:
```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.02, max_depth=1, n_estimators=1
000)
```

In [ ]:
```
accuracy_score(y_train, gbc_clf2.predict(x_train))
```

Out[ ]:
```
0.9950761655639329
```

In [ ]:
```
accuracy_score(y_test, gbc_clf2.predict(x_test))
```

Out[ ]:
```
0.9938461538461538
```

Ada boost

In [ ]:
```
ada_clf = AdaBoostClassifier(learning_rate =0.02, n_estimators =5000)
```

In [ ]:
```
ada_clf.fit(x_train, y_train)
```

Out[ ]:
```
AdaBoostClassifier(learning_rate=0.02, n_estimators=5000)
```

In [ ]:
```
accuracy_score(y_test, ada_clf.predict(x_test))
```

Out[ ]:
```
1.0
```

XG boost

In [ ]:
```
xgb_clf = xgb.XGBClassifier(max_depth=5, n_estimators=10000, learning_rate=0.3,
                            n_jobs=-1)
```

In [ ]:
```
xgb_clf.fit(x_train, y_train)
```

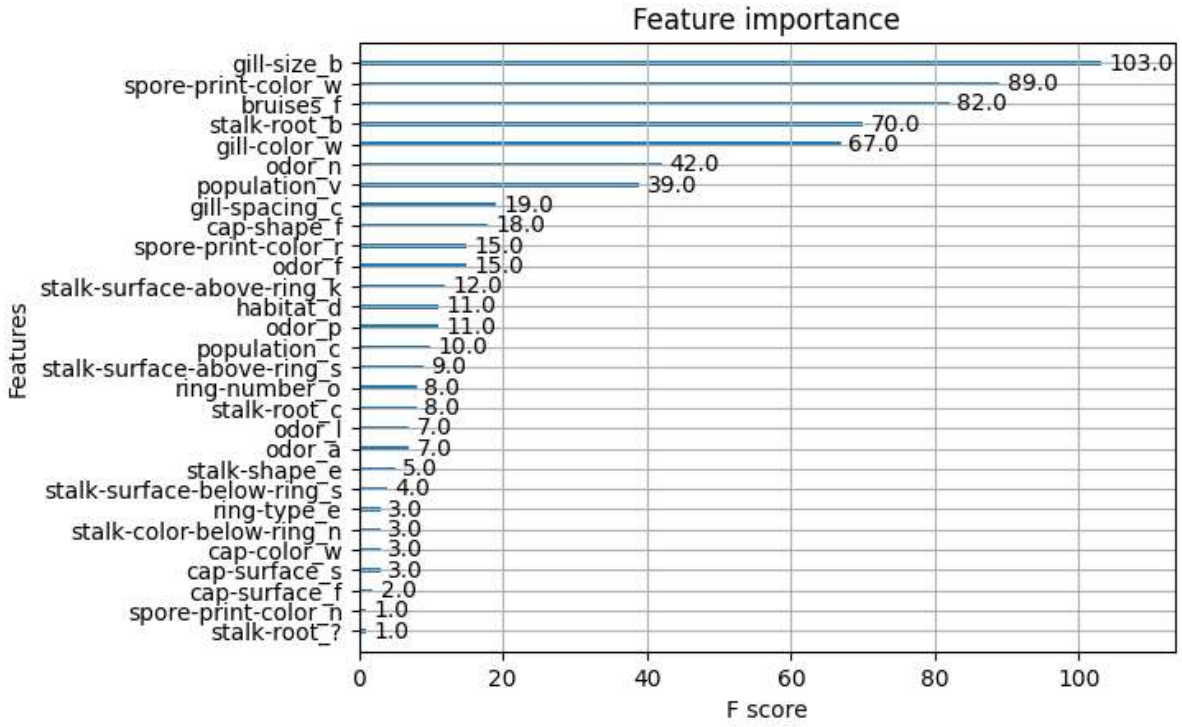
Out[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types
=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_typ
e=None,
              interaction_constraints=None, learning_rate=0.3, max_bin=
None,
```

In [ ]:
```
accuracy_score(y_test, xgb_clf.predict(x_test))
```

Out[ ]:
```
1.0
```

In [ ]:
```
xgb.plot_importance(xgb_clf)
```

Out[ ]:
```
<AxesSubplot: title={'center': 'Feature importance'}, xlabel='F score', ylabel='Fe
atures'>
```



In [ ]:
```
xgb_clf = xgb.XGBClassifier(n_estimators=250,learning_rate=0.1, random_state=42)
```

In [ ]:
```
param_test1 = {
 'max_depth':range(3,10,2),
    'gamma' : [0.1,0.2,0.3],
    'subsample':[0.8,0.9],
    'colsample_bytree':[0.8,0.9],
    'reg_alpha':[ 1e-2, 0.1, 1]
}
```

In [ ]:
```
grid_search = GridSearchCV(xgb_clf, param_test1,
                           n_jobs=-1, cv=5, scoring='accuracy')
```

This model predicts the test portion accurately, but it takes some time until it can be trained.

6. Support vector machine

Support Vector Machine (SVM) is a powerful machine learning algorithm used for classification, regression, and outlier detection tasks. It works by finding the optimal hyperplane that separates the input data into different classes or groups, with the maximum margin between the hyperplane and the closest data points of each class. SVM can handle both linear and non-linear decision boundaries by transforming the input data into a higher-dimensional space using kernel functions. SVM aims to maximize the margin while minimizing the classification error and can handle both binary and multi-class classification problems. SVM is a popular algorithm due to its ability to handle high-dimensional and complex data, and its effectiveness even with relatively small datasets. However, SVM can be

computationally expensive, especially for large datasets, and the choice of the kernel function and hyperparameters can significantly affect the model's performance.

In [ ]:
```
sc = StandardScaler().fit(x_train)
```

In [ ]:
```
x_train_std = sc.transform(x_train)
```

In [ ]:
```
x_test_std = sc.transform(x_test)
```

In [ ]:
```
clf_svm_l = svm.SVC(kernel='linear', C=100)
clf_svm_l.fit(x_train_std, y_train)
```

Out[ ]:
```
SVC
SVC(C=100, kernel='linear')
```

In [ ]:
```
y_train_pred = clf_svm_l.predict(x_train_std)
y_test_pred = clf_svm_l.predict(x_test_std)
```

In [ ]:
```
confusion_matrix(y_test, y_test_pred)
```

Out[ ]:
```
array([[783,   0],
       [  0, 842]], dtype=int64)
```

In [ ]:
```
accuracy_score(y_test, y_test_pred)
```

Out[ ]:
```
1.0
```

In [ ]:
```
clf_svm_l.n_support_
```

Out[ ]:
```
array([82, 96])
```

In [ ]:
```
params = {'C':(0.001,0.005,0.01,0.05, 0.1, 0.5, 1, 5, 10, 50,100,500,1000)}
```

In [ ]:
```
clf_svm_l = svm.SVC(kernel='linear')
```

In [ ]:
```
svm_grid_lin = GridSearchCV(clf_svm_l, params, n_jobs=-1,
                            cv=10, verbose=1, scoring='accuracy')
```

In [ ]:
```
svm_grid_lin.fit(x_train_std, y_train)
```

Out[ ]:
```
Fitting 10 folds for each of 13 candidates, totalling 130 fits
GridSearchCV
estimator: SVC
SVC
```

In [ ]:
```
svm_grid_lin.best_params_
```

Out[ ]:
```
{'C': 0.1}
```

In [ ]:
```
linsvm_clf = svm_grid_lin.best_estimator_
```

In [ ]:
```
accuracy_score(y_test, linsvm_clf.predict(x_test_std))
```

Out[ ]:
```
1.0
```

This model predicts the test portion.

Conclusion

Here all the models provided the same accuracy to the data of 1.0. But the core of this is to find the best predicatable model for the data. For predicatability and best complex relation. Due to higher ammount of data and multiple variables with higher correlation all the models tested have best predictions. So the performance and time taken are considered here. Here logistic regression have the lowest time taken. So, for this requirment even logistic regression is best.

Insights and keyfindings

Here whether gill size being broad or narrower influences the result more than any other feature and the stalker root bieng unknown being the least. <b> Due to being logistic regression having a good accuracy data have a linear relation to the edibility.

Due to perfect accuracy and prediction there is no more additional features needed for improving the model.