In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler
import warnings
warnings.filterwarnings('ignore')
from sklearn.tree import DecisionTreeRegressor
from sklearn.feature_selection import RFE
from sklearn.preprocessing import MinMaxScaler,StandardScaler
import joblib
import datetime
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import cross_val_score,GridSearchCV
from sklearn.linear_model import Ridge,Lasso
```

In [2]:

```
df= pd.read_csv('mushrooms.csv')
```

In [3]:

```
df
```

Out[3]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring
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
...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s
8120	e	x	s	n	f	n	a	c	b	y	...	s
8121	e	f	s	n	f	n	a	c	b	n	...	s
8122	p	k	y	n	f	y	f	c	n	b	...	k
8123	e	x	s	n	f	n	a	c	b	y	...	s

8124 rows × 23 columns

In [4]:
```
df.head()
```

Out[4]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	a
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 [5]:
```
df.tail()
```

Out[5]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring
8119	e	k	s	n	f	n	a	c	b	y	...	s
8120	e	x	s	n	f	n	a	c	b	y	...	s
8121	e	f	s	n	f	n	a	c	b	n	...	s
8122	p	k	y	n	f	y	f	c	n	b	...	k
8123	e	x	s	n	f	n	a	c	b	y	...	s

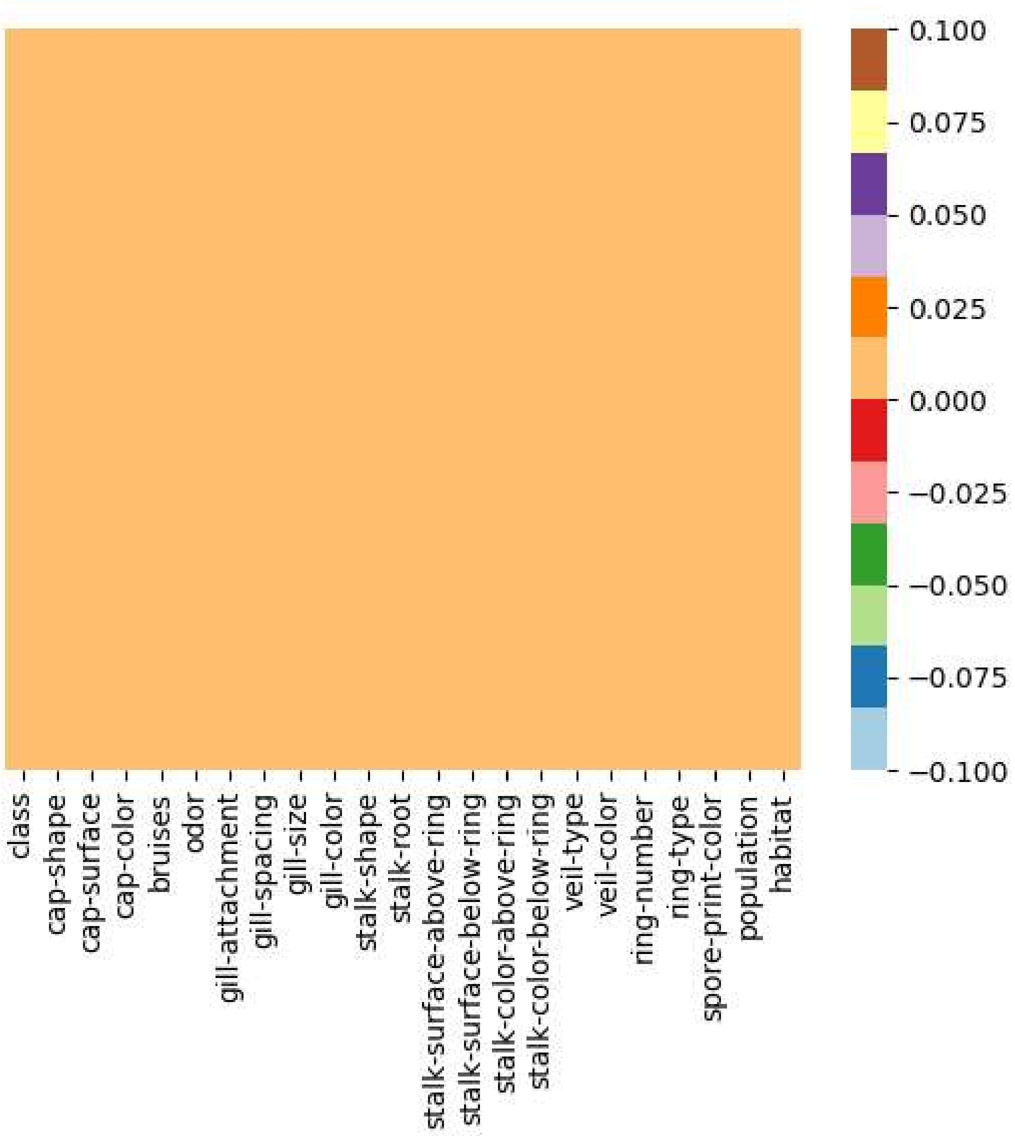
5 rows × 23 columns

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
class                       0
cap-shape                   0
cap-surface                 0
cap-color                   0
bruises                     0
odor                        0
gill-attachment             0
gill-spacing                0
gill-size                   0
gill-color                  0
stalk-shape                 0
stalk-root                  0
stalk-surface-above-ring    0
stalk-surface-below-ring    0
stalk-color-above-ring      0
stalk-color-below-ring      0
veil-type                   0
veil-color                  0
ring-number                 0
ring-type                   0
spore-print-color           0
population                  0
habitat                     0
dtype: int64
```

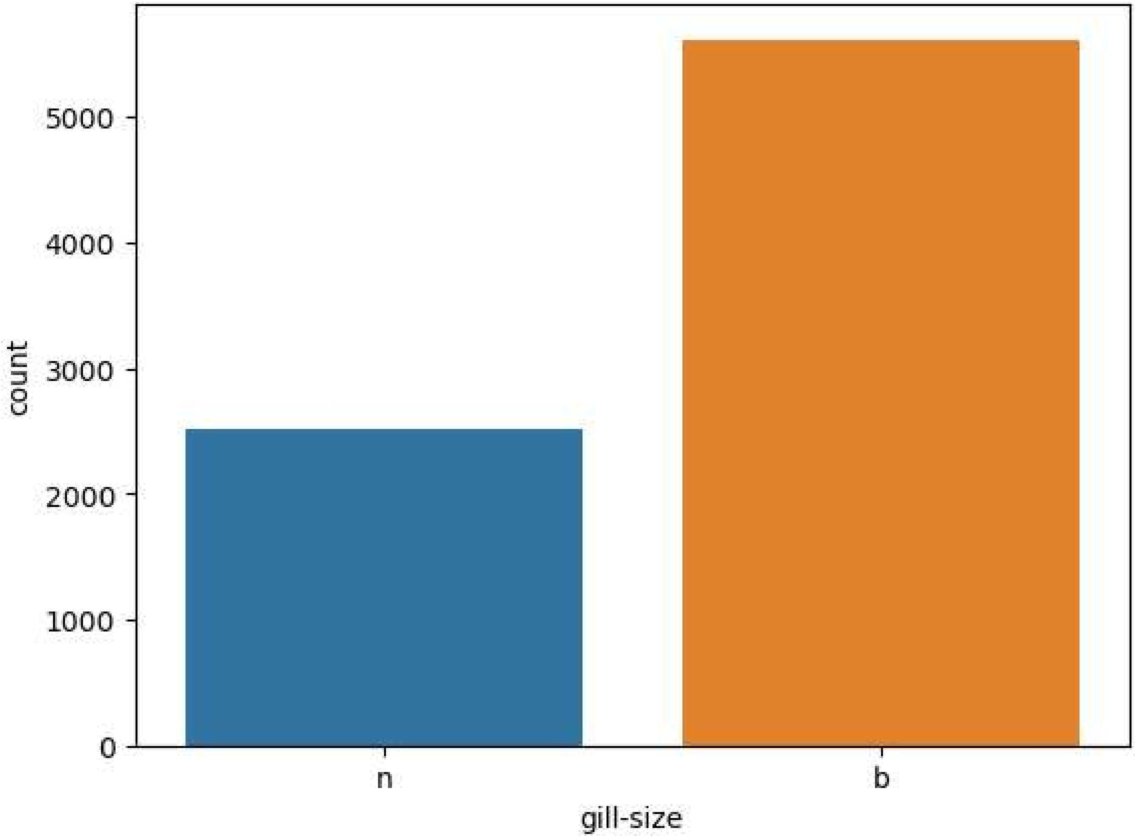
```
In [7]: sns.heatmap(df.isnull(),yticklabels=False,cmap="Paired")
```

```
Out[7]: <Axes: >
```



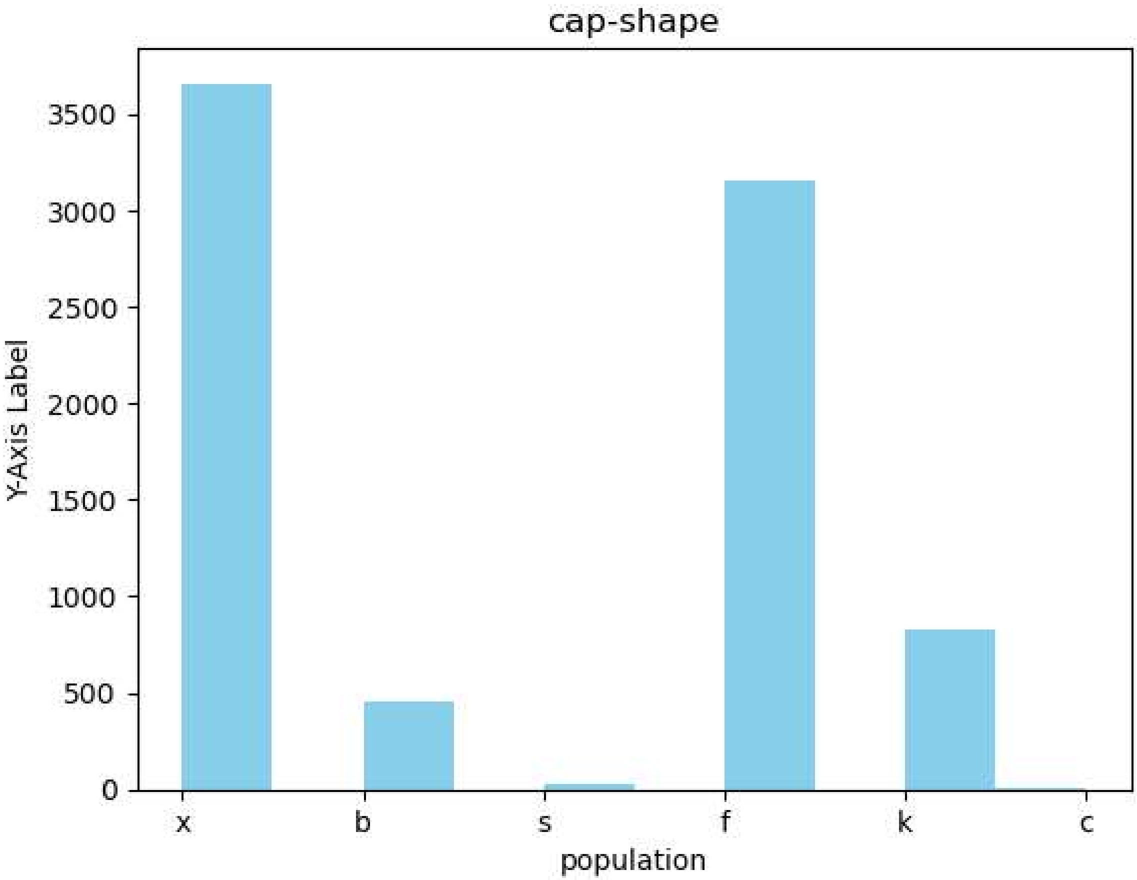
PERFORMING EDA

In [10]:
```
sns.countplot(x=df['gill-size'])
plt.show()
```



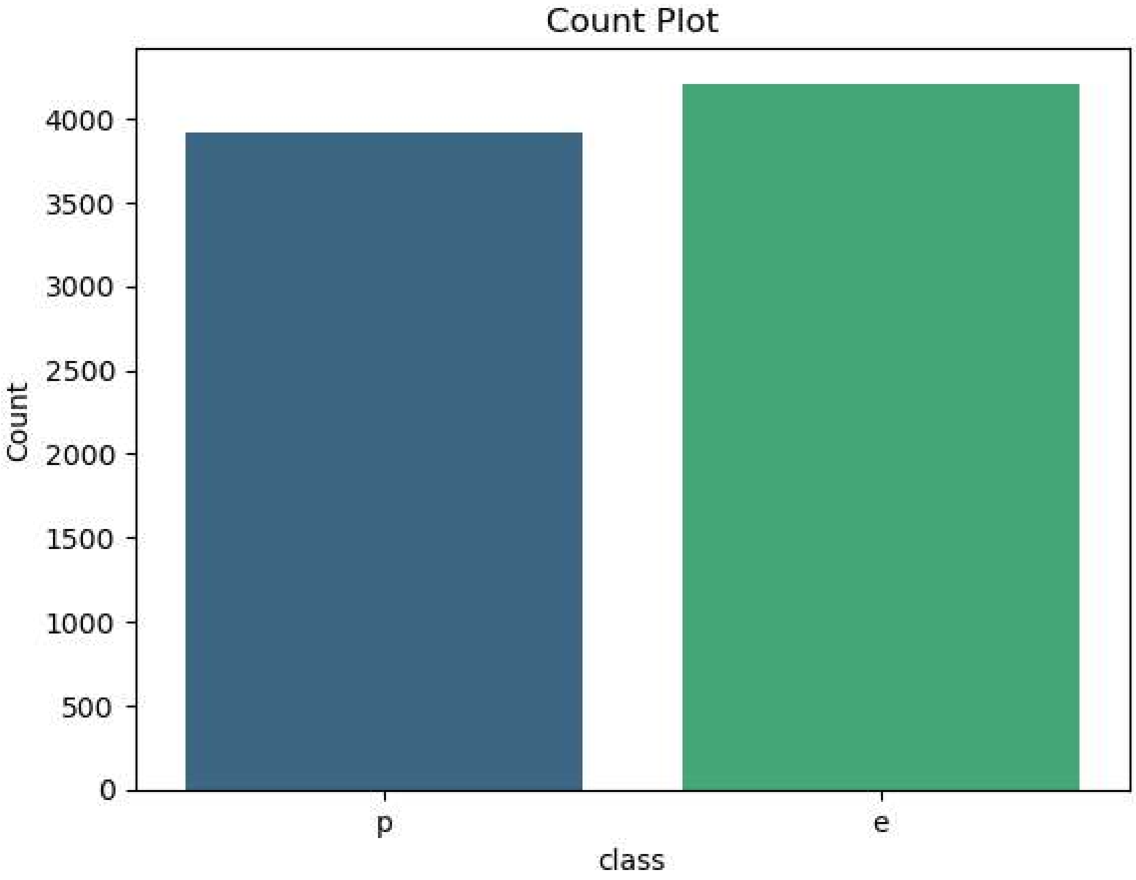
In [12]:

```
plt.hist(df['cap-shape'], bins=10, color='skyblue')

plt.xlabel('population')
plt.ylabel('Y-Axis Label')
plt.title('cap-shape')
plt.show()
```



In [18]:

```
sns.countplot(x='class', data=df, palette='viridis')
plt.xlabel('class')
plt.ylabel('Count')
plt.title('Count Plot')

plt.show()
```



In [21]:

```
df_area=df['cap-shape'].value_counts()
```

In [22]:

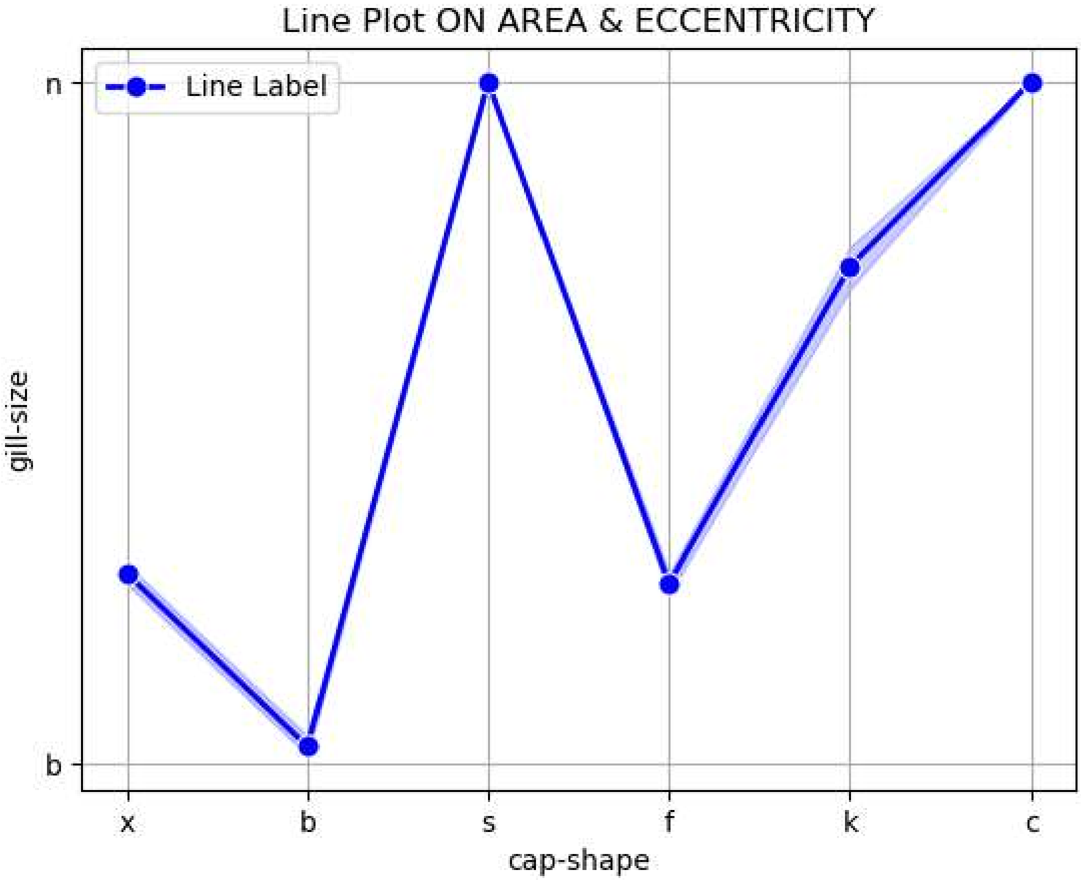
```
df_area
```

Out[22]:

```
x    3656
f    3152
k     828
b     452
s      32
c       4
Name: cap-shape, dtype: int64
```

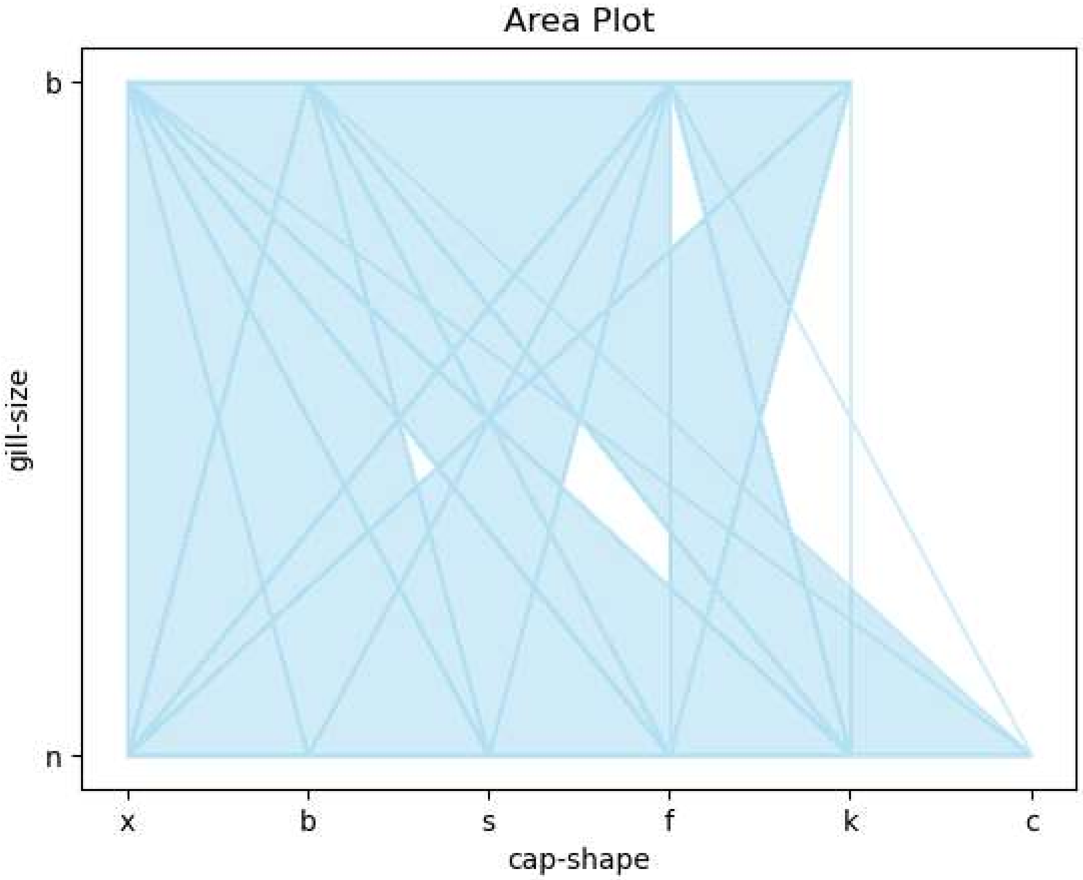
In [23]:

```
sns.lineplot(x='cap-shape', y='gill-size', data=df, marker='o', color='b', lin

plt.xlabel('cap-shape')
plt.ylabel('gill-size')
plt.title('Line Plot ON AREA & ECCENTRICITY')
plt.legend()
plt.grid(True)
plt.show()
```



In [27]:

```
plt.fill_between(df['cap-shape'], df['gill-size'], color='skyblue', alpha=0.4)
plt.xlabel('cap-shape')
plt.ylabel('gill-size')
plt.title('Area Plot')
plt.show()
```



In [31]:

```
df_pie =df['cap-shape'].value_counts()
```

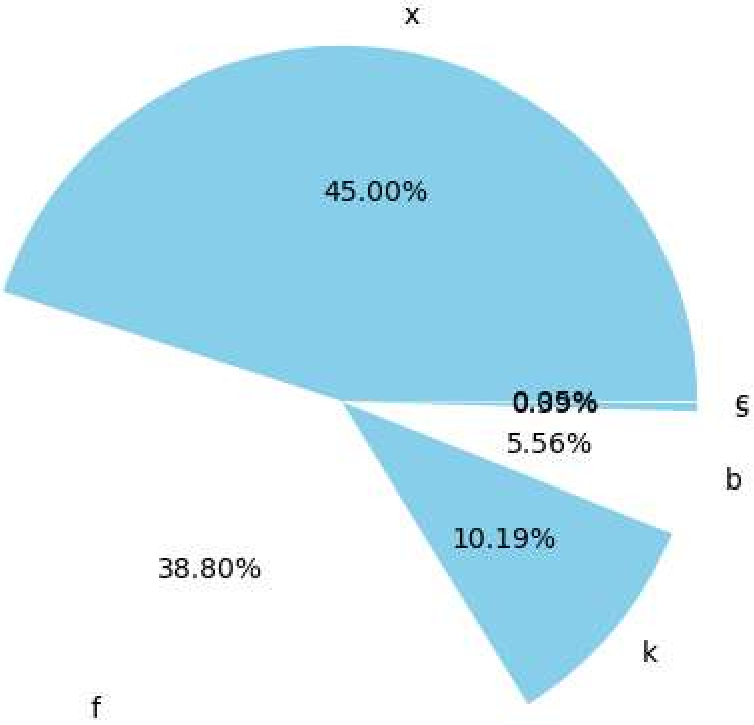
In [32]:

```
df_pie
```

Out[32]:

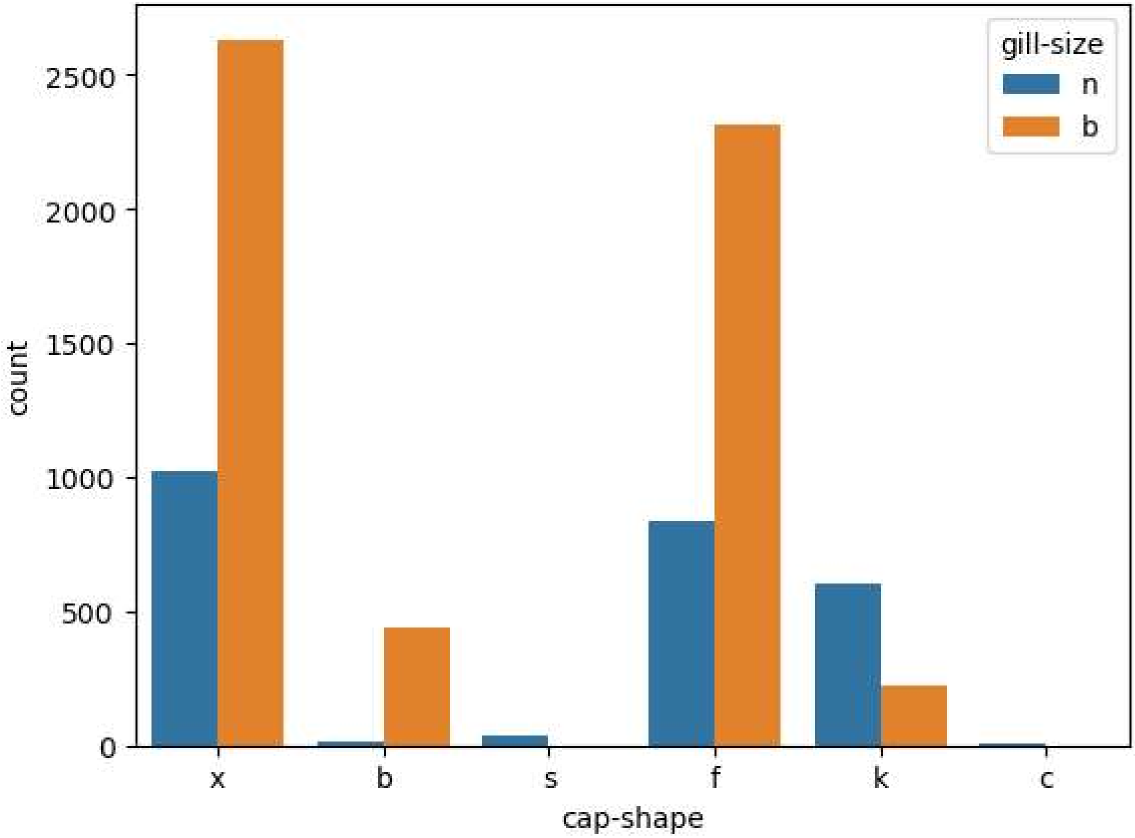
```
x    3656
f    3152
k     828
b     452
s      32
c       4
Name: cap-shape, dtype: int64
```

In [36]:
```
plt.pie(df_pie,labels=df_pie.index,autopct="%.2f%%",radius =1.2,colors=['skybl

plt.show()
```



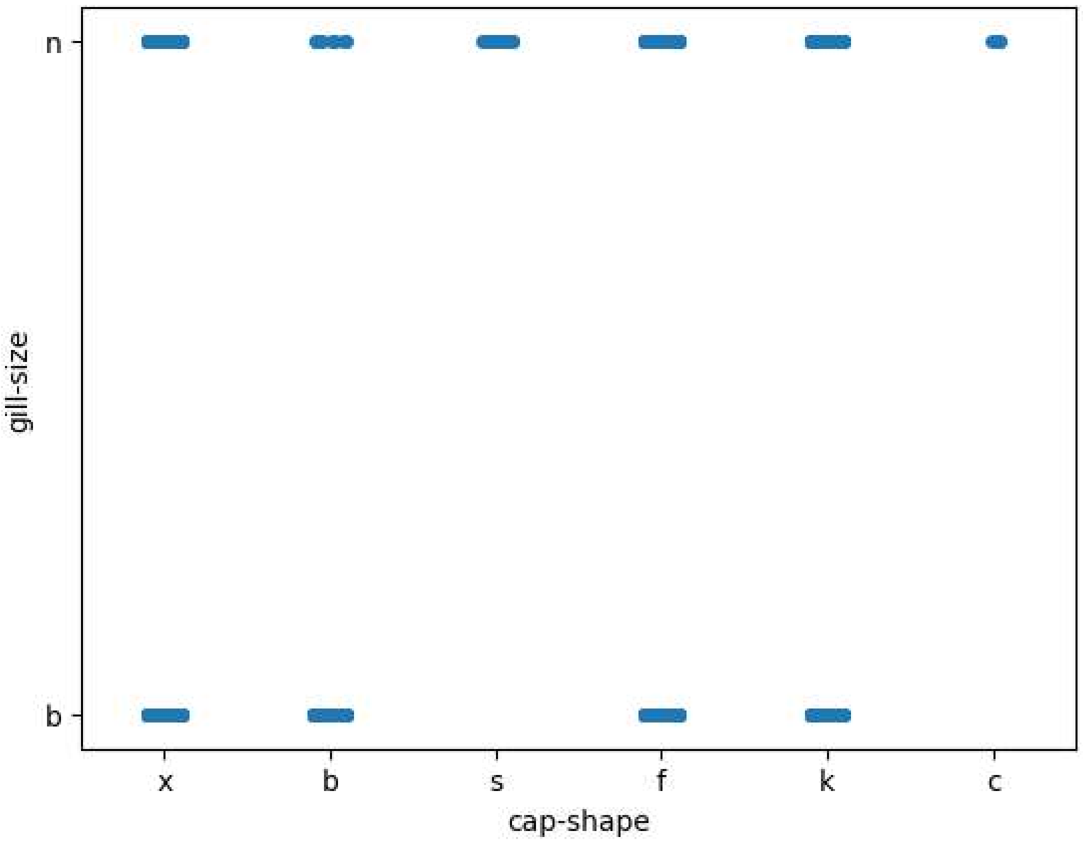
In [42]:

```
sns.countplot(x=df['cap-shape'],hue=df['gill-size'])
plt.show()
```



In [44]:
```
sns.stripplot(x='cap-shape',y='gill-size',data=df)
```

Out[44]:
```
<Axes: xlabel='cap-shape', ylabel='gill-size'>
```



In [48]:
```
cat_data=df.select_dtypes(include=object)
num_data=df.select_dtypes(exclude=object)
```

In [49]:
```
num_data
```

Out[49]:
```
   0
   1
   2
   3
   4
 ...
8119
8120
8121
8122
8123
```

8124 rows × 0 columns

In [50]:
```
cat_data
```

Out[50]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring
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
...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s
8120	e	x	s	n	f	n	a	c	b	y	...	s
8121	e	f	s	n	f	n	a	c	b	n	...	s
8122	p	k	y	n	f	y	f	c	n	b	...	k
8123	e	x	s	n	f	n	a	c	b	y	...	s

8124 rows × 23 columns

In [52]:
```
df.dtypes
```

Out[52]:
```
class                       object
cap-shape                   object
cap-surface                 object
cap-color                   object
bruises                     object
odor                        object
gill-attachment             object
gill-spacing                object
gill-size                   object
gill-color                  object
stalk-shape                 object
stalk-root                  object
stalk-surface-above-ring    object
stalk-surface-below-ring    object
stalk-color-above-ring      object
stalk-color-below-ring      object
veil-type                   object
veil-color                  object
ring-number                 object
ring-type                   object
spore-print-color           object
population                  object
habitat                     object
dtype: object
```

In [53]:
```
df
```

Out[53]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring
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
...	...	...	...	...	...	...	...	...	...	...	...	...
8119	e	k	s	n	f	n	a	c	b	y	...	s
8120	e	x	s	n	f	n	a	c	b	y	...	s
8121	e	f	s	n	f	n	a	c	b	n	...	s
8122	p	k	y	n	f	y	f	c	n	b	...	k
8123	e	x	s	n	f	n	a	c	b	y	...	s

8124 rows × 23 columns

In [54]:
```
encoder=LabelEncoder()
```

In [55]:
```
categorical_col = ['class','cap-shape','cap-surface','cap-color','bruises','od
encoder = OneHotEncoder(drop='first',sparse=False)
encoder_cols = pd.DataFrame(encoder.fit_transform(df[categorical_col]),columns
```

In [56]:
```
encoder_cols
```

Out[56]:

	class_p	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_g	cap-surface_s	cap-surface_y
0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...
8119	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
8120	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
8121	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
8122	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
8123	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0

8124 rows × 88 columns

In [64]:
```
x = pd.concat([encoder_cols],axis=1)
y = df['class']
```

In [65]:
```
x
```

Out[65]:

	class_p	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	cap-shape_x	cap-surface_g	cap-surface_s	cap-surface_y
0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
4	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...
8119	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
8120	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0
8121	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
8122	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
8123	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0

8124 rows × 88 columns

In [67]:
```
y
```

Out[67]:
```
0       p
1       e
2       e
3       p
4       e
       ..
8119    e
8120    e
8121    e
8122    p
8123    e
Name: class, Length: 8124, dtype: object
```

In [68]:
```
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,random_state
```

In [69]:
```
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler
import warnings
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
```

In [70]:
```
scaler = StandardScaler()
```

In [71]:
```
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

In [72]:
```
svcm = SVC(kernel='linear')
```

In [73]:
```
svcm.fit(x_train,y_train)
```

Out[73]:
```
SVC(kernel='linear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [74]:
```
y_pred = svcm.predict(x_test)
```

In [75]:
```
y_pred
```

Out[75]:
```
array(['e', 'p', 'p', ..., 'p', 'p', 'p'], dtype=object)
```

In [76]:
```
acc = accuracy_score(y_test,y_pred)
print('Accuracy:{:.2f}%'. format(acc*100))
```

```
Accuracy:100.00%
```

In [77]:
```
print(classification_report(y_test,y_pred))
```

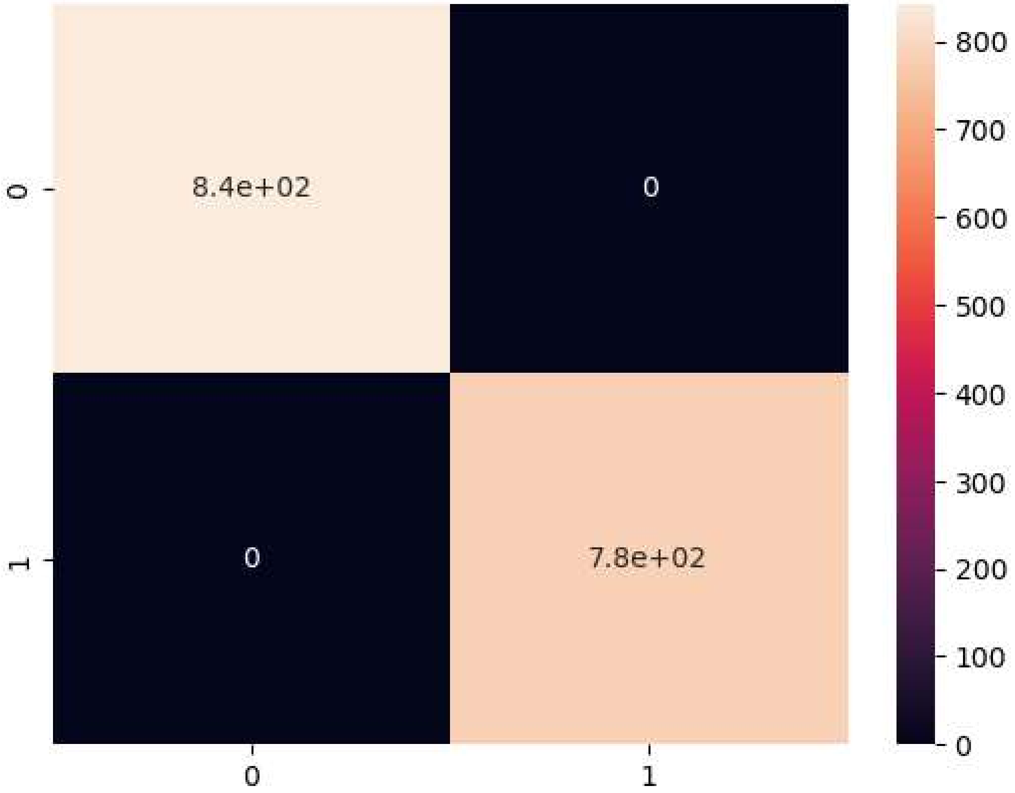
```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       843
           p       1.00      1.00      1.00       782

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [78]:
```
cm = confusion_matrix(y_test,y_pred)
print('confusion matrix:')
print(cm)
```

```
confusion matrix:
[[843   0]
 [  0 782]]
```

In [79]:
```
sns.heatmap(cm,annot=True)
```

Out[79]:
```
<Axes: >
```



In [80]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from sklearn.multiclass import OneVsRestClassifier
```

In [81]:
```
yb = label_binarize(y,classes=[0,1,2])
```

In [82]:
```
nc = yb.shape[1]
```

In [83]:
```
classifier = OneVsRestClassifier(SVC(kernel='linear',probability=True,random_s
```

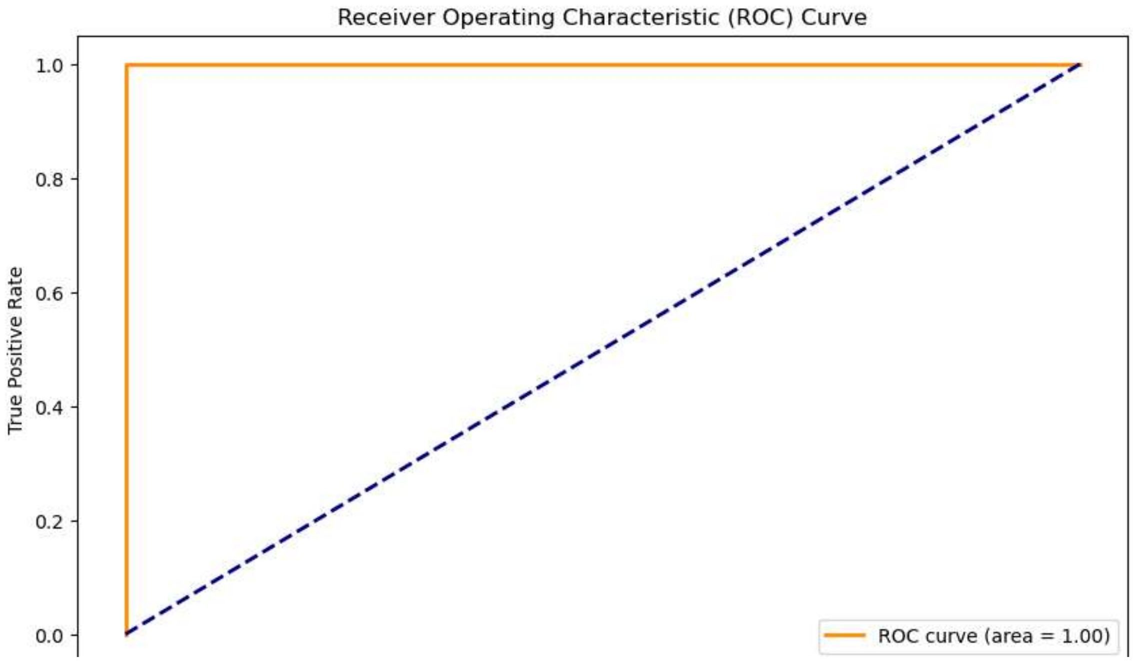
In [84]:
```
y_score = classifier.fit(x_train,y_train).decision_function(x_test)
```

In [85]:
```
y_score_2d = y_score.reshape(-1, 1)
```

In [87]:
```
y_test_binary = (y_test == 'p').astype(int)
fpr, tpr, _ = roc_curve(y_test_binary, y_score)
```

In [88]:
```
roc_auc = auc(fpr, tpr)
```

In [89]:
```
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_au
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
In
```



In [90]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
param_grid = {
 'C': [0.1, 1, 10, 100],
 'kernel': ['linear'],
}
svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, n_jobs=-1)
grid_search.fit(x_train, y_train)
print("Best hyperparameters found: ", grid_search.best_params_)
print("Best accuracy on the validation set: {:.2f}".format(grid_search.best_sc
```

```
Best hyperparameters found:  {'C': 0.1, 'kernel': 'linear'}
Best accuracy on the validation set: 1.00
```

In [91]:
```
grid_search = GridSearchCV(svm, param_grid, cv=5, n_jobs=-1)
grid_search.fit(x_train, y_train)
```

Out[91]:
```
GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100], 'kernel': ['linear']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [92]:
```
print("Best hyperparameters found: ", grid_search.best_params_)
print("Best accuracy on the validation set: {:.2f}".format(grid_search.best_sc
```

```
Best hyperparameters found:  {'C': 0.1, 'kernel': 'linear'}
Best accuracy on the validation set: 1.00
```

In [93]:
```
best_svm = grid_search.best_estimator_
best_svm.fit(x_train, y_train)
```

Out[93]:
```
SVC(C=0.1, kernel='linear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [94]:
```
test_accuracy = best_svm.score(x_test, y_test)
print("Test accuracy: {:.2f}".format(test_accuracy))
```

```
Test accuracy: 1.00
```

In [95]:
```
print("Test accuracy: {:.2f}".format(test_accuracy))
```

```
Test accuracy: 1.00
```

In [96]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
```

In [97]:
```
svm=SVC(kernel='linear')
param_dist={
    'C':uniform(loc=0,scale=10),
    'gamma':['scale','auto']+list(uniform(loc=0,scale=1).rvs(10)),
}
```

In [98]:

```
n_iter_search=20
random_search = RandomizedSearchCV(svcm, param_distributions=param_dist, n_ite
random_search.fit(x_train,y_train)
```

Out[98]:

```
RandomizedSearchCV(cv=5, estimator=SVC(kernel='linear'), n_iter=20, n_jobs=-
1,
                   param_distributions={'C': <scipy.stats._distn_infrastructu
re.rv_continuous_frozen object at 0x0000020E8231F890>,
                                        'gamma': ['scale', 'auto',
                                                  0.6084981184617801,
                                                  0.1527775985145413,
                                                  0.8245404219416449,
                                                  0.7976862097920863,
                                                  0.6878790088079055,
                                                  0.7418297071837214,
                                                  0.04358746495667076,
                                                  0.644865554378094,
                                                  0.05826150036667787 6,
                                                  0.1667991441025477]},
                   random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [99]:

```
best_param = random_search.best_params_
best_model = random_search.best_estimator_
y_pred_2=best_model.predict(x_test)
```

In [100]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       843
           p       1.00      1.00      1.00       782

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [101]:

```
cm= confusion_matrix(y_test,y_pred_2)
print(cm)
```

```
[[843   0]
 [  0 782]]
```

In [102]:
```
 from sklearn import model_selection, naive_bayes, svm, metrics,feature_extrac
```

In [103]:
```
x = pd.concat([encoder_cols],axis=1)
y = df['class']
```

In [104]:
```
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.2,random_state
```

In [105]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

In [106]:
```
bayes = naive_bayes.MultinomialNB()
```

In [107]:
```
bayes.fit(x_train,y_train)
```

Out[107]:
```
MultinomialNB()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [108]:
```
y_pred_nb=bayes.predict(x_test)
```

In [109]:
```
accuracy=metrics.accuracy_score(y_test,y_pred_nb)
accuracy
```

Out[109]:
```
0.9993846153846154
```

In [110]:
```
 print(metrics.classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       843
           p       1.00      1.00      1.00       782

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [111]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[111]:
```
array([[843,   0],
       [  0, 782]], dtype=int64)
```

In [112]:
```
yb=label_binarize(y, classes=[0,1,2])
nc = yb.shape[1]
```

In [120]:
```
 param_grid = {
'alpha': [0.1, 1, 10, 100],
'fit_prior': [True, False]
}
```

In [121]:
```
bayes = naive_bayes.MultinomialNB()
grid_search = GridSearchCV(bayes, param_grid, cv=5)
grid_search.fit(x_train, y_train)
```

Out[121]:
```
GridSearchCV(cv=5, estimator=MultinomialNB(),
             param_grid={'alpha': [0.1, 1, 10, 100],
                         'fit_prior': [True, False]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [122]:
```
best_param = grid_search.best_params_
best_nb = naive_bayes.MultinomialNB(alpha = best_param['alpha'], fit_prior = b
best_nb.fit(x_train, y_train)
y_pred = best_nb.predict(x_test)
```

In [123]:
```
print("Best Hyperparameter : ", best_param)
```

```
Best Hyperparameter :  {'alpha': 0.1, 'fit_prior': True}
```

In [124]:
```
acc = accuracy_score(y_test, y_pred)
acc
```

Out[124]:
```
0.9993846153846154
```

In [125]:
```
print (classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       843
           p       1.00      1.00      1.00       782

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [126]:

```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[126]:

```
array([[842,   1],
       [  0, 782]], dtype=int64)
```

In [127]:

```
param_dist = {
    'alpha': uniform(0.1, 2.0),  # Example: Uniform distribution for alpha
    'fit_prior':[True,False]
}
```

In [128]:

```
 bayes = naive_bayes.MultinomialNB()
```

In [129]:

```
x=scaler.fit_transform(x)
```

In [130]:

```
from sklearn.utils.validation import check_non_negative
check_non_negative(x, "MultinomialNB (input x)")
```

In [131]:

```
randomized_search = RandomizedSearchCV(bayes, param_distributions=param_dist,
randomized_search.fit(x, y)
```

Out[131]:

```
RandomizedSearchCV(cv=5, estimator=MultinomialNB(),
                   param_distributions={'alpha': <scipy.stats._distn_infrastr
ucture.rv_continuous_frozen object at 0x0000020E8513D290>,
                                        'fit_prior': [True, False]},
                   scoring='accuracy')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [132]:

```
best_param = randomized_search.best_params_
print("Best Hyperparameter : ", best_param)
```

```
Best Hyperparameter :  {'alpha': 0.28467359617041776, 'fit_prior': True}
```

In [133]:

```
best_nb = naive_bayes.MultinomialNB(alpha = best_param['alpha'], fit_prior = b
best_nb.fit(x_train, y_train)
y_pred = best_nb.predict(x_test)
```

In [134]:

```
acc = accuracy_score(y_test, y_pred)
acc
```

Out[134]:

```
0.9993846153846154
```

In [135]:

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           e       1.00      1.00      1.00       843
           p       1.00      1.00      1.00       782

    accuracy                           1.00      1625
   macro avg       1.00      1.00      1.00      1625
weighted avg       1.00      1.00      1.00      1625
```

In [136]:

```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[136]:

```
array([[842,   1],
       [  0, 782]], dtype=int64)
```

In [ ]:

```

```