Name: Krishna Santosh Kabra(27)

Practical6: Implement simple Navie Bayes Classifications algorithm. Using python on iris.csv dataset.

Import Liabraries

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay,classification_report,accuracy_score, precision_score, recall_score,
from sklearn.preprocessing import LabelEncoder
```

Load the Iris dataset using seaborn

```
data = sns.load_dataset('iris')
```

Display the dataset

```
data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Display the first 5 rows of the dataset

```
data.head(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Display the last 5 rows of the dataset

```
data.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

Display summary statistics for the dataset

```
data.describe(include='all')
```

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	setosa
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.057333	3.758000	1.199333	NaN
std	0.828066	0.435866	1.765298	0.762238	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

Display information about the dataset, including data types and missing values

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Print the shape of the dataset

```
print(data.shape)
```

```
(150, 5)
```

Get unique values of the 'species' column

```
data['species'].unique()
```

```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

Check for missing values in the dataset

```
data.isnull().sum()
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
```

```
species         0
dtype: int64
```

Split the dataset into features (x) and target variable (y)

```
x = data.iloc[:,1:5]
y = data.iloc[:,5:]
```

Split the data into training and testing sets

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 0)
```

```
print(x_train)
print(y_train)
print(type(x_train))
print(type(y_train))
```

```
     sepal_width  petal_length  petal_width     species
60           2.0           3.5          1.0  versicolor
116          3.0           5.5          1.8   virginica
144          3.3           5.7          2.5   virginica
119          2.2           5.0          1.5   virginica
108          2.5           5.8          1.8   virginica
..           ...           ...          ...         ...
9            3.1           1.5          0.1      setosa
103          2.9           5.6          1.8   virginica
67           2.7           4.1          1.0  versicolor
117          3.8           6.7          2.2   virginica
47           3.2           1.4          0.2      setosa

[105 rows x 4 columns]
Empty DataFrame
Columns: []
Index: [60, 116, 144, 119, 108, 69, 135, 56, 80, 123, 133, 106, 146, 50, 147, 85, 30, 101, 94, 64, 89, 91, 125, 48, 13, 111, 95, 20,

[105 rows x 0 columns]
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

```
# Drop the 'species' column from the DataFrame to obtain the feature matrix (x)
x = data.drop(columns=['species'])

# Select the 'species' column as the target variable (y)
y = data['species']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)

# Initialize LabelEncoder
encode = LabelEncoder()

# Encode the target variable y
y_train_encoded = encode.fit_transform(y_train)
y_test_encoded = encode.transform(y_test)

# Instantiate Gaussian Naive Bayes classifier
naive_bayes = GaussianNB()

# Train the classifier
naive_bayes.fit(x_train, y_train_encoded)

# Make predictions
pred = naive_bayes.predict(x_test)
```

Make predictions

```
pred
```

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 1, 1, 1, 2, 0, 2, 0,
       0])
```

```
y_test
```

```
114     virginica
62     versicolor
33         setosa
107     virginica
7          setosa
100     virginica
40         setosa
86     versicolor
76     versicolor
71     versicolor
134     virginica
51     versicolor
73     versicolor
54     versicolor
63     versicolor
37         setosa
78     versicolor
90     versicolor
45         setosa
16         setosa
121     virginica
66     versicolor
24         setosa
8          setosa
126     virginica
22         setosa
44         setosa
97     versicolor
93     versicolor
26         setosa
137     virginica
84     versicolor
27         setosa
127     virginica
132     virginica
59     versicolor
18         setosa
83     versicolor
61     versicolor
92     versicolor
112     virginica
2          setosa
141     virginica
43         setosa
10         setosa
Name: species, dtype: object
```

Check if any unseen labels are present

```
# Check unique values in y
unique_labels_y = set(y.unique())

# Check if any unseen labels are present
unseen_labels = unique_labels_y - set(encode.classes_)

if unseen_labels:
    print("Unseen labels in y:", unseen_labels)
    # Handle unseen labels here, e.g., remove instances or encode them differently
else:
    print("No unseen labels in y.")
```

```
No unseen labels in y.
```

Print classes seen by LabelEncoder during fitting and unique values in y

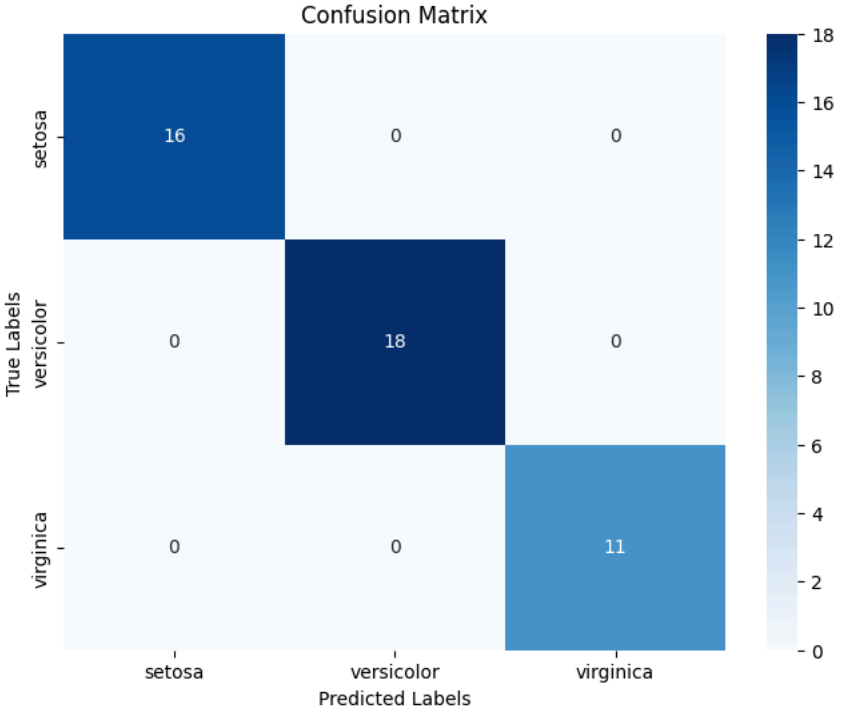
```
print("Classes seen by LabelEncoder during fitting:", encode.classes_)
print("Unique values in y:", y.unique())
```

```
Classes seen by LabelEncoder during fitting: ['setosa' 'versicolor' 'virginica']
Unique values in y: ['setosa' 'versicolor' 'virginica']
```

```
# Train the classifier
naive_bayes.fit(x_train, y_train_encoded)

# Make predictions on the test data
pred = naive_bayes.predict(x_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test_encoded, pred)
precision = precision_score(y_test_encoded, pred, average='weighted')
recall = recall_score(y_test_encoded, pred, average='weighted')
f1 = f1_score(y_test_encoded, pred, average='weighted')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

```
# Create confusion matrix
matrix = confusion_matrix(y_test_encoded, pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(matrix, annot=True, fmt='d', cmap='Blues', xticklabels=encode.classes_, yticklabels=encode.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```




```
# Encode the string labels in y_test to numeric labels
y_test_encoded = encode.transform(y_test)

# Print the classification report using the encoded labels
print(classification_report(y_test_encoded, pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        16
           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        11

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Double-click (or enter) to edit

```
from sklearn.metrics import classification_report

# Print classification report
print(classification_report(y_test_encoded, pred, target_names=encode.classes_))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       1.00      1.00      1.00        18
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
print(matrix.shape)
```

```
(3, 3)
```

```
from sklearn.metrics import classification_report

# Print classification report
print(classification_report(y_test_encoded, pred, target_names=encode.classes_))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        16
  versicolor       1.00      1.00      1.00        18
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

```
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix using the encoded labels
matrix = confusion_matrix(y_test_encoded, pred)

# Calculate accuracy
accuracy = np.diag(matrix).sum() / matrix.sum()

# Print the metrics
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Accuracy: 1.00
```

```
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix using the encoded labels
matrix = confusion_matrix(y_test_encoded, pred)

# Calculate error rate
error_rate = 1 - np.diag(matrix).sum() / matrix.sum()

# Print the error rate
print('Error Rate:', error_rate)
```

```
Error Rate: 0.0
```

```
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix using the encoded labels
matrix = confusion_matrix(y_test_encoded, pred)

# Calculate sensitivity for each class
sensitivity = np.diag(matrix) / matrix.sum(axis=1)

# Print sensitivity for each class
for i, class_label in enumerate(encode.classes_):
    print(f'Sensitivity (Recall) for class {class_label}: {sensitivity[i]}')
```

```
Sensitivity (Recall) for class setosa: 1.0
Sensitivity (Recall) for class versicolor: 1.0
Sensitivity (Recall) for class virginica: 1.0
```

```
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix using the encoded labels
matrix = confusion_matrix(y_test_encoded, pred)

# Calculate precision for each class
precision = np.diag(matrix) / matrix.sum(axis=0)

# Print precision for each class
for i, class_label in enumerate(encode.classes_):
    print(f'Precision for class {class_label}: {precision[i]}')
```

```
Precision for class setosa: 1.0
Precision for class versicolor: 1.0
Precision for class virginica: 1.0
```

```
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix using the encoded labels
matrix = confusion_matrix(y_test_encoded, pred)

# Calculate specificity for each class
specificity = []
for i, class_label in enumerate(encode.classes_):
    # Get the indices of all other classes except the current one
    other_indices = [j for j in range(matrix.shape[0]) if j != i]

    # Calculate true negatives (TN) for the current class
    tn = np.sum(matrix[other_indices][:, other_indices])

    # Calculate false positives (FP) for the current class
    fp = np.sum(matrix[other_indices][:, i])

    # Calculate specificity for the current class
    specificities = tn / (tn + fp)

    specificity.append(specificities)

# Print specificity for each class
for i, class_label in enumerate(encode.classes_):
    print(f'Specificity for class {class_label}: {specificity[i]}')
```

```
Specificity for class setosa: 1.0
Specificity for class versicolor: 1.0
Specificity for class virginica: 1.0
```

```
from sklearn.metrics import confusion_matrix

# Calculate confusion matrix using the encoded labels
matrix = confusion_matrix(y_test_encoded, pred)

# Calculate false positive rate for each class
fpr = []
```