Exercise 11

Use wine dataset from sklearn.datasets to classify wines into 3 categories. Load the dataset and split it into test and train. After that train the model using Gaussian and Multinominal classifier and post which model performs better. Use the trained model to perform some predictions on test data.

Import all essentials

In [109…

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
import seaborn as sn
```

Load the wine data

In [110…

```
data = load_wine()
dir(data)
```

Out[110]:

```
['DESCR', 'data', 'feature_names', 'frame', 'target', 'target_names']
```

In [111…

```
data.data
```

Out[111]:

```
array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]])
```

Make it into a DataFrame and append the target column

In [112…

```
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[112]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	col
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	

GroupBy the target column

In [113…

```
df.groupby('target').describe()
```

Out[113]:

	alcohol								malic_acid		...	od280/od315_of_diluted_wines		
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	coun
target														
0	59.0	13.744746	0.462125	12.85	13.400	13.750	14.100	14.83	59.0	2.010678	...	3.42	4.00	59.(
1	71.0	12.278732	0.537964	11.03	11.915	12.290	12.515	13.86	71.0	1.932676	...	3.16	3.69	71.(
2	48.0	13.153750	0.530241	12.20	12.805	13.165	13.505	14.34	48.0	3.333750	...	1.82	2.47	48.(

3 rows × 104 columns

In [114…

```
df.shape
```

Out[114]:

```
(178, 14)
```

Split into X and y

In [115…

```
X = df.drop(['target'], axis=1)
y = df['target']
```

Split into Training and Testing dataset

In [116…

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

K Fold Cross Validation

In [117…

```
kf = StratifiedKFold(n_splits=10)
```

In [118…

```
gaussian = cross_val_score(GaussianNB(), X, y, cv=kf)
```

In [119…

```
multinomial = cross_val_score(MultinomialNB(), X, y, cv=kf)
```

In [120…

```
print(f"Gaussian NB average score {np.mean(gaussian)}")
print(f"Multinomial NB average score {np.mean(multinomial)}")
```

```
Gaussian NB average score 0.9777777777777779
Multinomial NB average score 0.8496732026143791
```

So we can see clearly that Gaussian NB is doing better here

Using Gaussian Naive Bayes model

In [121…

```
model = GaussianNB()
model.fit(X_train, y_train)
```

Out[121]:

```
▾ GaussianNB
GaussianNB()
```

In [122…

```
model.score(X_test, y_test)
```

Out[122]:

```
1.0
```

Predictions

Checking the value at index 99

In [123…

```
test = np.array(X.iloc[99])
test
```

Out[123]:

```
array([ 12.29,   3.17,   2.21,  18.  ,  88.  ,   2.85,   2.99,   0.45,
         2.81,   2.3 ,   1.42,   2.83, 406.  ])
```

Checking the target at index 99

In [124

```
y[99]
```

```
In [124…: y[99]
```

```
Out[124]: 1
```

Predicting the value at index 99

```
In [125…: model.predict([test])
```

```
C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:465: UserWarning: X does
not have valid feature names, but GaussianNB was fitted with feature names
  warnings.warn(
```

```
Out[125]: array([1])
```

```
In [126…: predict = model.predict(X_test)
          predict
```

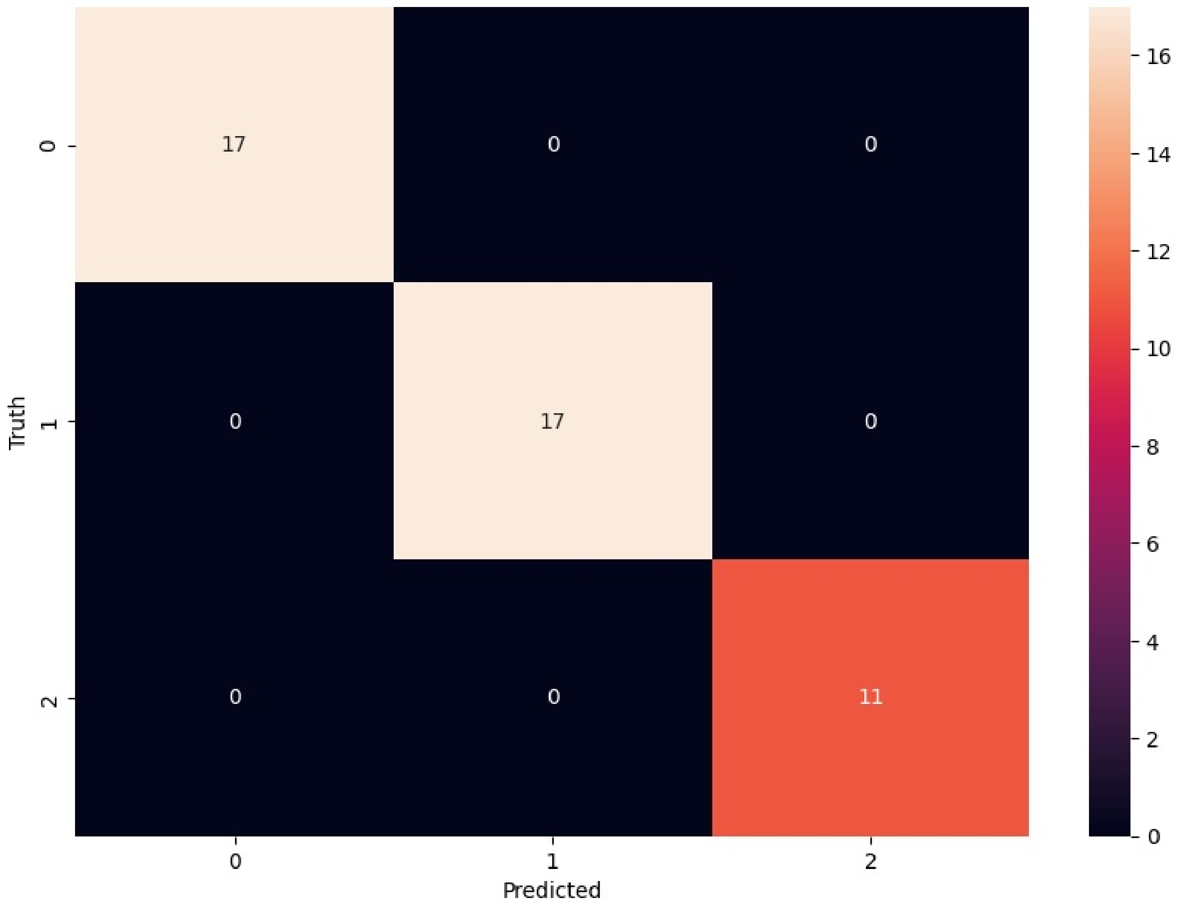
```
Out[126]: array([0, 0, 2, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 2, 1, 2, 1, 2, 1, 0, 0,
                 2, 2, 1, 0, 1, 1, 1, 1, 2, 1, 2, 0, 0, 1, 0, 1, 0, 0, 2, 2, 0, 1,
                 2])
```

```
In [149…: # model.predict_proba(X_test)
```

Using Confusion Matrix

```
In [141…: cm = confusion_matrix(y_test, predict)
          plt.figure(figsize=(10, 7))
          sn.heatmap(cm, annot=True)
          plt.xlabel("Predicted")
          plt.ylabel("Truth")
```

```
Out[141]: Text(95.72222222222221, 0.5, 'Truth')
```



Making a new DataFrame from Actual and Predicted Values

```
In [150…: obj = {
              "Actual Value" : y_test,
              "Predicted Value" : predict
          }

          valuedf = pd.DataFrame(obj)
          valuedf
```

Out[150]:

	Actual Value	Predicted Value
1	0	0
36	0	0
174	2	2
115	1	1
56	0	0
47	0	0
42	0	0
19	0	0
96	1	1
30	0	0
78	1	1
98	1	1
86	1	1
17	0	0
138	2	2
91	1	1
171	2	2
106	1	1
172	2	2
84	1	1
51	0	0
38	0	0
130	2	2
161	2	2
94	1	1
49	0	0
112	1	1
111	1	1
116	1	1
80	1	1
155	2	2
129	1	1
177	2	2
24	0	0
52	0	0
88	1	1
35	0	0
76	1	1
11	0	0

18	0	0
159	2	2
132	2	2
0	0	0
117	1	1
167	2	2

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js