In [16]:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

```

In [27]:

```
data=pd.read_csv("C:/Users/kriti/OneDrive/Desktop/machine Learning/experiments/play_ter
```

In [28]:

```
data
```

Out[28]:

	day	outlook	temp	humidity	wind	play
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

In [29]:

```
#Encoding the data
#Transform labels to normalized encoding.First, the categorical variables\
#will need to be encoded.
#LabelEncoder is a utility class to help normalize labels.
#fit_transform -Fit to data, then transform it.

from sklearn.preprocessing import LabelEncoder
number = LabelEncoder()
data['outlook'] = number.fit_transform(data['outlook'])
data['temp'] = number.fit_transform(data['temp'])
data['humidity'] = number.fit_transform(data['humidity'])
data['wind'] = number.fit_transform(data['wind'])
data['play']=number.fit_transform(data['play'])
```

In [4]:

```
data
```

Out[4]:

	day	outlook	temp	humidity	wind	play
0	D1	2	1	0	1	0
1	D2	2	1	0	0	0
2	D3	0	1	0	1	1
3	D4	1	2	0	1	1
4	D5	1	0	1	1	1
5	D6	1	0	1	0	0
6	D7	0	0	1	0	1
7	D8	2	2	0	1	0
8	D9	2	0	1	1	1
9	D10	1	2	1	1	1
10	D11	2	2	1	0	1
11	D12	0	2	0	0	1
12	D13	0	1	1	1	1
13	D14	1	2	0	0	0

In [5]:

```
features = ["outlook", "temp", "humidity", "wind"]
target = "play"
```

x = data.iloc[:,0:5] # X is the features in our dataset y = data.iloc[:,-1] # y is the Labels in our dataset

In [6]:

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(data[features], data[target],

test_size = 0.33,

random_state = 54)

```

In [7]:

```
# gaussian means it is normally destribured
#apply naive bayes theoram

from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
#Fit Gaussian Naive Bayes according to  training set
model.fit(x_train, y_train)

```

Out[7]:

```
GaussianNB()
```

In [8]:

```
pred = model.predict(x_test)
pred
```

Out[8]:

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

In [9]:

```
# comapre actual and predicted values
df = pd.DataFrame({'Actual value': y_test, 'predicted value': pred})
df
```

Out[9]:

	Actual value	predicted value
4	1	1
12	1	1
9	1	1
3	1	1
13	0	1

In [10]:

```
# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, pred)

```

In [11]:

```
accuracy
```

Out[11]:

0.8

In [52]:

```
# Do the prediction for the the following unput instance
print(model.predict([[1,2,0,1]]))

```

[1]

In [12]:

```
# print classification report
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score

print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.80      1.00      0.89         4

    accuracy                           0.80         5
   macro avg       0.40      0.50      0.44         5
weighted avg       0.64      0.80      0.71         5


C:\Users\kriti\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1221: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [13]:

```
# Print confusion matrix
confusion = confusion_matrix(y_test, pred)
print(confusion)
```

```
[[0 1]
 [0 4]]
```

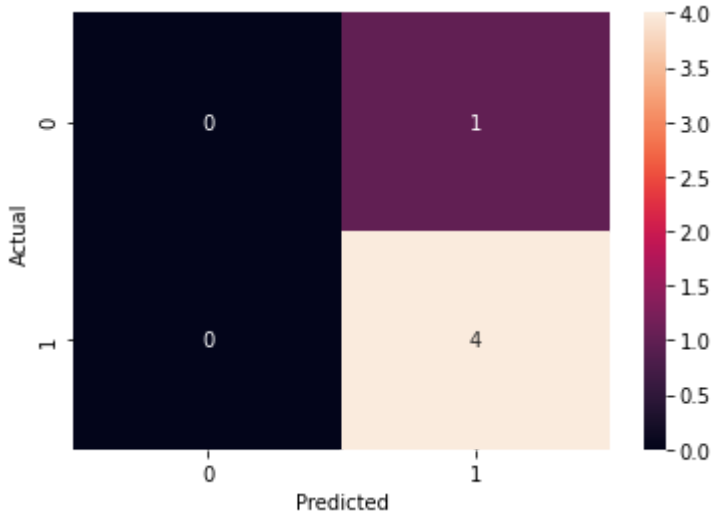
In [14]:

```
# Print confusion matrix using seaborn library
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(confusion, annot=True)

plt.ylabel('Actual')
plt.xlabel('Predicted')
```

Out[14]:

```
Text(0.5, 15.0, 'Predicted')
```



In [15]:

```
# We can also print the probability of an instance belonging to a particular class
pred = model.predict_proba(x_test)
pred
```

Out[15]:

```
array([[1.37124777e-01, 8.62875223e-01],
       [4.02044587e-04, 9.99597955e-01],
       [1.37124777e-01, 8.62875223e-01],
       [4.77670162e-01, 5.22329838e-01],
       [3.76121016e-01, 6.23878984e-01]])
```

In [ ]:

```

```