In [2]:
```
import csv

hypo = ['%'] * 6
data=[]
print("\nThe given training examples are:")

#Load the dataset
with open('p1_dataset.csv') as csv_file:
    readcsv = csv.reader(csv_file, delimiter=',')
    next(readcsv)
    for row in readcsv:
      print(row)
      if(row[-1]=="Yes"):
        data.append(row)

#Display positive examples
print("\nThe positive examples are:")
for x in data:
    print(x)

print("\nThe steps of the Find-s algorithm are\n",hypo)

#Implementing Find-S Algorithm
for i in range(len(data)):
    for j in range(len(data[i])-1):
        if hypo[j] == '%':
            hypo[j] = data[i][j]
        elif hypo[j] != data[i][j]:
            hypo[j] = '?'
    print(hypo)

print("\nThe maximally specific Find-s hypothesis for the given training examples is:")
print([h for h in hypo if h != '%'])
```

```
The given training examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The positive examples are:
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes']
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']

The steps of the Find-s algorithm are
 ['%', '%', '%', '%', '%', '%']
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', '?', '?']

The maximally specific Find-s hypothesis for the given training examples is:
['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

In [1]:
```
import csv

#Load the dataset
with open("p1_dataset.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)
    specific = data[1][:-1]   #gets the first row without the target
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    #Initalize general and specific hypothesis
    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        #If negative test case
        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"
        #Print the steps of Algorithm
        print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination Algorithm")
        print(specific)
        print(general)

    gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j != '?':
                gh.append(i)
                break
    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

```
Step 1 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 2 of Candidate Elimination Algorithm
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 3 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Step 4 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Sam
e']]

Step 5 of Candidate Elimination Algorithm
['Sunny', 'Warm', '?', 'Strong', '?', '?']
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Final Specific hypothesis:
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final General hypothesis:
 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

In [15]:
```
#Import the necessary libraries
from sklearn.datasets import load_iris
import pandas as pd
```

In [16]:
```
#Load the dataset and seperated the features  and Targets
tennis = pd.read_csv('P3_cleaning.csv')
X=tennis.iloc[:,0:4]
y=tennis.iloc[:,4:5]
print("FEATURES")
print(X)
print("TARGET")
print(y)
```

```
FEATURES
     outlook temperature humidity    wind
0      sunny         hot     high    weak
1      sunny         hot     high    weak
2   overcast         hot     high  strong
3       rain        mild     high    weak
4       rain        cool   normal    weak
5       rain        cool   normal  strong
6   overcast        cool   normal  strong
7      sunny        mild     high    weak
8      sunny        cool   normal    weak
9       rain        mild   normal    weak
10     sunny        mild   normal  strong
11  overcast        mild     high  strong
12  overcast         hot   normal    weak
13      rain        mild     high  strong
TARGET
   play
0    no
1    no
2   yes
3   yes
4   yes
5    no
6   yes
7    no
8   yes
9   yes
10  yes
11  yes
12  yes
13   no
```

In [18]:
```
#Data Cleaning - Features(Ordinal encoder) and Targets(Label encoder)
from sklearn.preprocessing import OrdinalEncoder,LabelEncoder
ordinal_encoder = OrdinalEncoder()  # for cleaning the features
label_encode = LabelEncoder()       # for cleaning the targets

X_ordinal_encoded= ordinal_encoder.fit_transform(X)
print("features\n" , X_ordinal_encoded)
y_label_encoded = label_encode.fit_transform(y.values.ravel())
print("Target\n",y_label_encoded)
```

```
features
 [[2. 1. 0. 1.]
 [2. 1. 0. 0.]
 [0. 1. 0. 1.]
 [1. 2. 0. 1.]
 [1. 0. 1. 1.]
 [1. 0. 1. 0.]
 [0. 0. 1. 0.]
 [2. 2. 0. 1.]
 [2. 0. 1. 1.]
 [1. 2. 1. 1.]
 [2. 2. 1. 0.]
 [0. 2. 0. 0.]
 [0. 1. 1. 1.]
 [1. 2. 0. 0.]]
Target
 [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

In [1]:
```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder,OrdinalEncoder
```

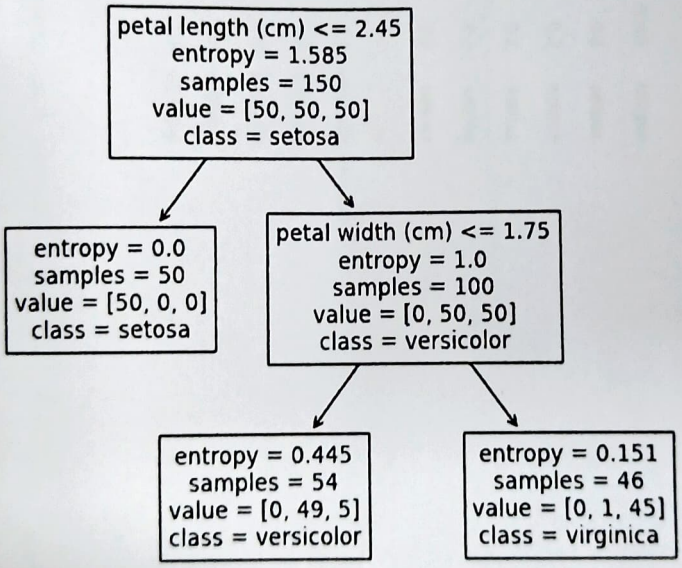
In [2]:
```
iris_data = load_iris()
X = iris_data.data[:,2:]
y = iris_data.target
print("Shape of X : "+str(X.shape)+"\nShape of y : "+str(y.shape))
```

```
Shape of X : (150, 2)
Shape of y : (150,)
```

In [3]:
```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='entropy',max_depth=2, random_state=100)
clf = tree_clf.fit(X,y)
```

In [4]:
```
from sklearn import tree
tree.plot_tree(clf,feature_names=iris_data.feature_names[2:],class_names=iris_data.target_names)
```

Out[4]:
```
[Text(0.4, 0.8333333333333334, 'petal length (cm) <= 2.45\nentropy = 1.585\nsamples = 150\nvalue = [5
0, 50, 50]\nclass = setosa'),
 Text(0.2, 0.5, 'entropy = 0.0\nsamples = 50\nvalue = [50, 0, 0]\nclass = setosa'),
 Text(0.6, 0.5, 'petal width (cm) <= 1.75\nentropy = 1.0\nsamples = 100\nvalue = [0, 50, 50]\nclass =
versicolor'),
 Text(0.4, 0.16666666666666666, 'entropy = 0.445\nsamples = 54\nvalue = [0, 49, 5]\nclass = versicolo
r'),
 Text(0.8, 0.16666666666666666, 'entropy = 0.151\nsamples = 46\nvalue = [0, 1, 45]\nclass = virginic
a')]
```



In [5]:
```
print( tree_clf.predict_proba([[5,1.5]]) )
```

```
[[0.         0.90740741 0.09259259]]
```

In [6]:
```
otp = tree_clf.predict([[5,1.5]])
print(iris_data.target_names[otp])
```

```
['versicolor']
```

In [1]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
```

In [2]:
```
iris_data=load_iris()
X = iris_data.data[:,2:]
y = iris_data.target
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

In [3]:
```
df = pd.DataFrame(iris_data.data,columns=iris_data.feature_names)
df['target'] = iris_data.target_names[iris_data.target]
df
```

Out[3]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
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

In [4]:
```
rnd_clf = RandomForestClassifier(n_estimators=500,max_leaf_nodes=16,n_jobs=-1)
rnd_clf.fit(X_train,y_train)
```

Out[4]:
```
RandomForestClassifier
RandomForestClassifier(max_leaf_nodes=16, n_estimators=500, n_jobs=-1)
```

In [5]:
```
y_pred_rf = rnd_clf.predict(X_test)
accuracy = accuracy_score(y_test,y_pred_rf)
```

In [6]:
```
print("actual : "+str(y_test))
print("predicted : "+str(y_pred_rf))
print("Accuracy : "+str(accuracy))
```

```
actual : [2 2 2 1 1 1 0 1 2 2 1 2 0 0 1 1 0 2 1 2 2 1 1 2 1 0 1 1 1 1]
predicted : [2 2 2 1 2 1 0 1 2 2 1 2 0 0 1 1 0 2 1 2 2 1 1 2 1 0 1 1 1 1]
Accuracy : 0.9666666666666667
```

In [16]:
```
#Importing the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
```

In [19]:
```
#Load the breast cancer Dataset
cancer = load_breast_cancer()
X= cancer.data
y= cancer.target
```

In [20]:
```
#Split the train and test with 0.2 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)
gnb = GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
accuracy = accuracy_score(y_test,y_pred)
```

In [21]:
```
#Display the actulual,predicted,Accuracy
print("actual : "+str(np.array(y_test)))
print("predicted : "+str(y_pred))
print("Accuracy : "+str(accuracy))
```

```
actual : [0 1 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1
 1 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0
 1 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 0
 1 0 1]
predicted : [0 1 0 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1
 1 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0
 1 1 1 0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 1 1 1 1 1 0
 1 0 1]
Accuracy : 0.9298245614035088
```

In [14]:
```
#Import the necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,confusion_matrix,precision_score,recall_score
```

In [15]:
```
#Load the libraries
msg=pd.read_csv('P7_dataset.csv',names=['message','label'])
print('The dimensions of the dataset',msg.shape)
#Map pos to 1 and neg to 0
msg['labelnum']=msg.label.map({'pos':1,'neg':0})
X=msg.message
y=msg.labelnum
#Display the dataframe with message and labelnum. head() shows only 5 records
msg[['message','labelnum']].head()
```

```
The dimensions of the dataset (18, 2)
```

Out[15]:

	message	labelnum
1	I love this sandwich	1
2	This is an amazing place	1
3	I feel very good about these beers	1
4	This is my best work	1
5	What an awesome view	1

In [16]:
```
#Split the dataset into train and test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=100)
```

In [17]:
```
#Make probability distribution of words and count the occurences
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_dtm = count_vect.fit_transform(X_train)
X_test_dtm=count_vect.transform(X_test)
clf = MultinomialNB().fit(X_train_dtm,y_train)
predicted = clf.predict(X_test_dtm)
```

In [18]:
```
#Display the accuracy , Confusion matrix, precision and recall
print('Accuracy metrics')
print('Accuracy of the classifer is : ',accuracy_score(y_test,predicted))
print('Confusion matrix')
print(confusion_matrix(y_test,predicted))
print('Recall and Precison ')
print(recall_score(y_test,predicted))
print(precision_score(y_test,predicted))
```

```
Accuracy metrics
Accuracy of the classifer is :  1.0
Confusion matrix
[[3 0]
 [0 2]]
Recall and Precison
1.0
1.0
```

```
! pip install pgmpy
! pip install torch
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pgmpy
  Downloading pgmpy-0.1.22-py3-none-any.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 36.5 MB/s eta 0:00:00
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.10.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.5.3)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.0.9)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from pgmpy) (2.0.1+cu118)
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-packages (from pgmpy) (0.13.5)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pgmpy) (4.65.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from pgmpy) (1.2.0)
Requirement already satisfied: opt-einsum in /usr/local/lib/python3.10/dist-packages (from pgmpy) (3.3.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->pgmpy) (2022.7.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->pgmpy)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (0.5.3)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels->pgmpy) (23.1
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.12.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (1.11.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch->pgmpy) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch->pgmpy) (3.25.
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch->pgmpy) (16.0.5)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.2->statsmodels->pgmpy) (1.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch->pgmpy) (2.
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->pgmpy) (1.3.0)
Installing collected packages: pgmpy
Successfully installed pgmpy-0.1.22
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.0.1+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.12.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.11.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: triton==2.0.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.0.0)
Requirement already satisfied: cmake in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (3.25.2)
Requirement already satisfied: lit in /usr/local/lib/python3.10/dist-packages (from triton==2.0.0->torch) (16.0.5)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.2)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```

```
import numpy as np
import pandas as pd
from pgmpy.inference import VariableElimination
from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
```

```
heartdisease = pd.read_csv('heart.csv')
heartdisease = heartdisease.replace('?',np.nan)
```

```
heartdisease.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca
0	63	1	1	145	233	1	2	150	0	2.3	3	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3
2	67	1	4	120	229	0	2	129	1	2.6	2	2
3	37	1	3	130	250	0	0	187	0	3.5	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0

```
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdis
```

```
/usr/local/lib/python3.10/dist-packages/pgmpy/models/BayesianModel.py:8: FutureWarning: BayesianModel has been renamed to
  warnings.warn(
```

```
model.fit(heartdisease,estimator= MaximumLikelihoodEstimator)
```

```
heartdisease_infer = VariableElimination(model)
```

```
print('\n 1.Probability of HeartDisease given evidence= restecg :1')
q1=heartdisease_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
```

```
 1.Probability of HeartDisease given evidence= restecg :1
+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease(0) |              0.1012 |
+----------------+---------------------+
| heartdisease(1) |              0.0000 |
+----------------+---------------------+
| heartdisease(2) |              0.2392 |
+----------------+---------------------+
| heartdisease(3) |              0.2015 |
+----------------+---------------------+
| heartdisease(4) |              0.4581 |
+----------------+---------------------+
/usr/local/lib/python3.10/dist-packages/pgmpy/models/BayesianModel.py:8: FutureWarning: BayesianModel has been renamed to
  warnings.warn(
```

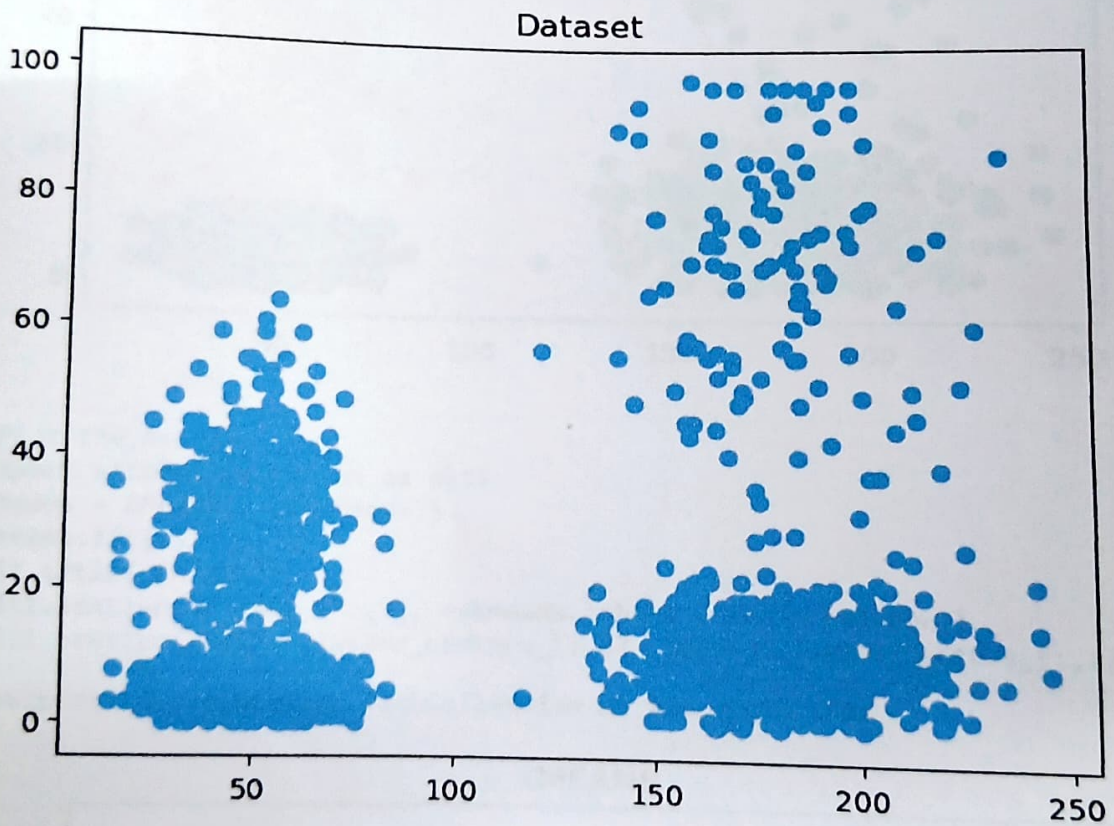
```
print('\n 1.Probability of HeartDisease given evidence= cp :2')
q2=heartdisease_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
 1.Probability of HeartDisease given evidence= cp :2
+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease(0) |              0.3610 |
+----------------+---------------------+
| heartdisease(1) |              0.2159 |
+----------------+---------------------+
| heartdisease(2) |              0.1373 |
+----------------+---------------------+
| heartdisease(3) |              0.1537 |
+----------------+---------------------+
| heartdisease(4) |              0.1321 |
+----------------+---------------------+
```

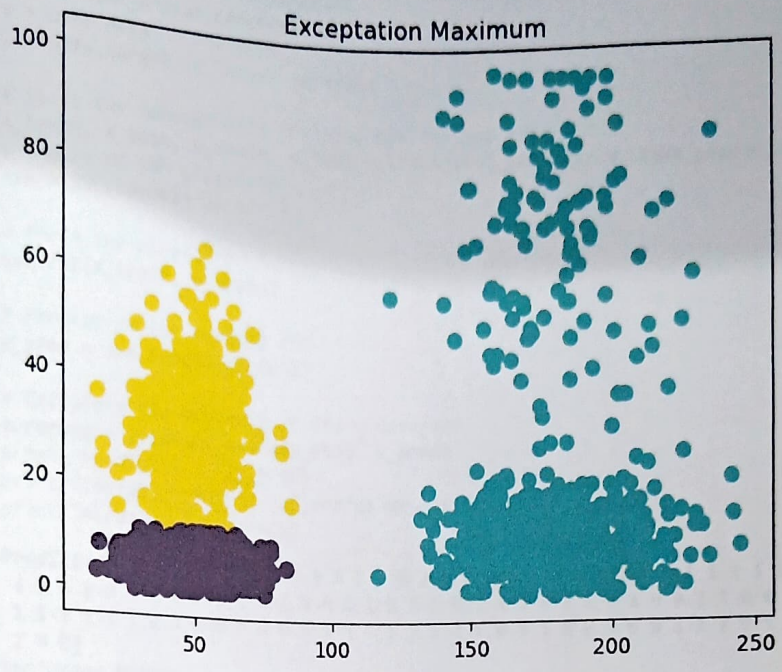
In [2]:
```
#Import the necessary libraries
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import pandas as pd
```

In [3]:
```
#Load the dataset
X = pd.read_csv('P9_dataset.csv')
x1 = X['Distance_Feature'].values
x2 = X['Speeding_Feature'].values
X= np.array(list(zip(x1,x2))).reshape(len(x1),2)
```

In [4]:
```
#Visualize the dataset
plt.plot()
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
```

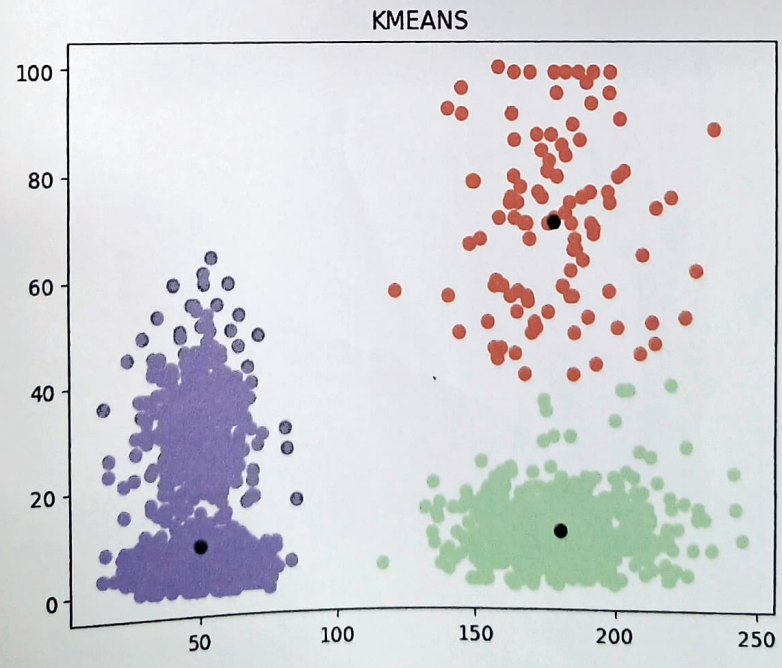


In [5]:
```
#Plot the Exceptation Maximum (EM)
gmm = GaussianMixture(n_components=3)
gmm.fit(X)
em_predictions = gmm.predict(X)
plt.title('Exceptation Maximum')
plt.scatter(X[:,0], X[:,1],c=em_predictions,s=50)
plt.show()
```



In [6]:
```
#Plot the Kmeans
import matplotlib.pyplot as plt1
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
plt.title('KMEANS')
plt1.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')
plt1.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

Out[6]: <matplotlib.collections.PathCollection at 0x1ca7b952550>



In [4]:
```
#Import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix

# Load the breast cancer dataset
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create an SVM classifier
svm = SVC(kernel='linear')

# Train the classifier
svm.fit(X_train, y_train)

# Predict on the testing set
y_pred = svm.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Predicted : ",y_pred)
print("Confusion Matrix \n",confusion_matrix(y_test,y_pred))
print("Accuracy:",accuracy)
```

```
Predicted :  [1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 0 0]
Confusion Matrix
 [[39  4]
 [ 1 70]]
Accuracy: 0.956140350877193
```