Program 7

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
from sklearn.neighbors import KNeighborsClassifier

df=pd.read_csv(r"D:\PRIYA\SEM 05\dataset\iris.csv")
df.head()
```

Out[1]:

	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

In [2]:
```
x=df.drop(['species'],axis=1)
y=df['species']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)

Scaler=StandardScaler()
x_train=Scaler.fit_transform(x_train)
x_test=Scaler.transform(x_test)

classifier=KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
```

In [6]:
```
accuracy = accuracy_score(y_test,y_pred)
print(f"Accuracy: {accuracy:.2f}\n")
print("Confusion Matrix:")
print(confusion_matrix(y_test,y_pred))
print("Classification Report:")
print(classification_report(y_test,y_pred))
```

```
Accuracy: 1.00

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

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

In [7]:
```
for true,pred in zip (y_test,y_pred):
    if true==pred:
        print(f"correct:true label={true}, predicted label={pred}")
    else:
        print("false:true label={true}, predicted label={pred}")
```

Program 8

In [8]:
```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score, f1_score

data=pd.read_csv(r"D:\PRIYA\SEM 05\dataset\NaiveText.csv")
#data=pd.read_csv(r'/content/IMDB Dataset.csv')

data.head()
```

Out[8]:

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

In [10]:
```
x=data['message']
y=data['label']

#x=data['review']
#y=data['sentiment']

x_train,x_test, y_train,y_test= train_test_split(x,y,test_size=0.2,random_state=42)

vectorizer = CountVectorizer()
x_train_vectorized = vectorizer.fit_transform(x_train)
x_test_vectorized = vectorizer.transform(x_test)

classifier = MultinomialNB()
classifier.fit(x_train_vectorized, y_train)
y_pred = classifier.predict(x_test_vectorized)
print(y_pred)


print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average = 'weighted'))
print("Recall:", recall_score(y_test, y_pred, average = 'weighted'))
print("F1 Score:", f1_score(y_test, y_pred, average = 'weighted'))
print("Confusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
[1 1 0 0]

Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
Confusion Matrix
[[2 0]
 [0 2]]
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00         2

    accuracy                           1.00         4
   macro avg       1.00      1.00      1.00         4
weighted avg       1.00      1.00      1.00         4
```

Program 9

```
In [ ]: pip install pgmpy
```

```
In [ ]: from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination

burglary_model=BayesianNetwork([('Burglary','Alarm'),
                                ('Earthquake','Alarm'),
                                ('Alarm','DavidCalls'),
                                ('Alarm','SophiaCalls')
                                ])
cpd_burglary=TabularCPD(
                            variable='Burglary',
                            variable_card = 2,
                            values=[[0.999],[0.001]])

cpd_earthquake=TabularCPD(
                            variable='Earthquake',
                            variable_card=  2,
                            values=[[0.998],[0.002]])
cpd_alarm=TabularCPD(
                            variable='Alarm',
                            variable_card = 2,
                            values=[[0.999,0.71,0.06,0.05],[0.001,0.29,0.94,0.95]],
                            evidence=['Burglary','Earthquake'],
                            evidence_card=[2,2])

cpd_david_calls=TabularCPD(
                            variable='DavidCalls',
                            variable_card=2,
                            values=[[0.8,0.1],[0.2,0.9]],
                            evidence=['Alarm'],
                            evidence_card=[2] )

cpd_sophia_calls=TabularCPD(
                            variable='SophiaCalls',
                            variable_card=2,
                            values=[[0.7,0.3],[0.3,0.7]],
                            evidence=['Alarm'],
                            evidence_card=[2])

burglary_model.add_cpds(cpd_burglary,cpd_earthquake,cpd_alarm,cpd_david_calls,cpd_sophia_calls)

if not burglary_model.check_model():
    raise ValueError("the bayesian Network structure or CPTs are invaild")
inference=VariableElimination(burglary_model)
prob_burglary=inference.query(variables=['Burglary'],
                             evidence={'DavidCalls':1,
                                      'SophiaCalls':1})
print(prob_burglary)
```

Program 10

In [11]:
```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

data = pd.read_csv(r"D:\PRIYA\SEM 05\dataset\PlayTennis.csv")
data.head()
```

Out[11]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes

In [12]:
```
le = LabelEncoder()
categorical_cols = ['Outlook', 'Temperature', 'Humidity', 'Wind', 'Play Tennis']

for col in categorical_cols:
    data[col] = le.fit_transform(data[col])
data.head(5)

X = data.drop('Play Tennis', axis=1)
y = data['Play Tennis']
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.33,random_state=42)

clf = GaussianNB()
clf.fit(x_train,y_train)
ypred = clf.predict(x_test)
print(ypred)

accuracy = accuracy_score(y_test, ypred)
print("Accuracy:", accuracy)
```

```
[1 0 0 1 1]
Accuracy: 0.6
```

Program 11

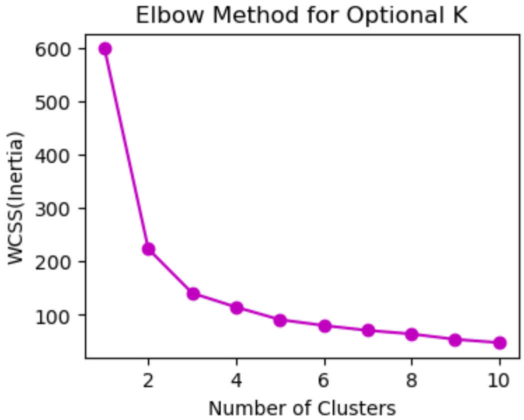
In [23]:
```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder,StandardScaler
from sklearn.cluster import KMeans,DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv(r"D:\PRIYA\SEM 05\dataset\iris.csv")
df.head(5)
```

Out[23]:

	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

In [52]:
```python
label_encoder=LabelEncoder()
df["species"]=label_encoder.fit_transform(df["species"])
features=['sepal_length','sepal_width','petal_length','petal_width']
x=df[features]
scaler=StandardScaler()
x_scaled=scaler.fit_transform(x)
wcss=[]
for i in range(1,11):
    km=KMeans(n_clusters=i)
    km.fit(x_scaled)
    wcss.append(km.inertia_)

plt.figure(figsize=(4,3))
plt.plot(range(1,11),wcss,marker='o',linestyle='-',color='m')
plt.title("Elbow Method for Optional K")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS(Inertia)")
plt.show()
```

```
C:\Users\Dell\Downloads\priya python\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environ ment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

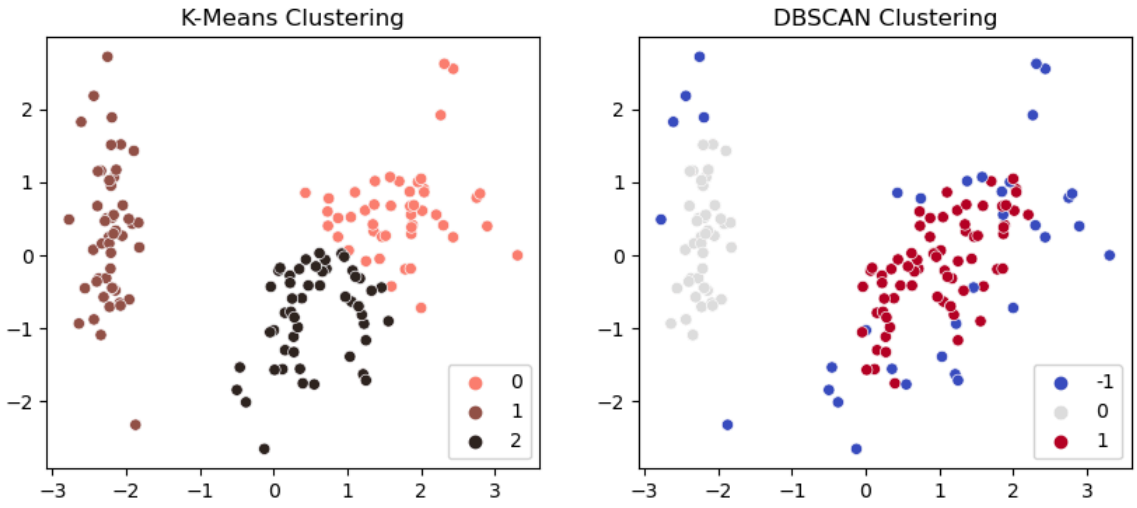


In [25]:
```python
kmeans=KMeans(n_clusters=3,random_state=42)
kmeans_labels=kmeans.fit_predict(x_scaled)

dbscan=DBSCAN(eps=0.5,min_samples=5)
dbscan_labels=dbscan.fit_predict(x_scaled)

kmeans_silhouette=silhouette_score(x_scaled,kmeans_labels)
dbscan_silhouette=silhouette_score(x_scaled[dbscan_labels != -1],dbscan_labels[dbscan_labels != -1])

print(f"silhouette_score for kmeans:{kmeans_silhouette:.3f}")
print(f"silhouette_score for dbscan:{dbscan_silhouette:.3f}")
```

```
silhouette_score for kmeans:0.459
silhouette_score for dbscan:0.653
```

In [55]:
```python
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
x_reduced=pca .fit_transform(x_scaled)
fig,axs=plt.subplots(1,2,figsize=(10,4))

sns.scatterplot(x=x_reduced[:,0],y=x_reduced[:,1],hue=kmeans_labels,palette="dark:salmon_r",ax=axs[0])
axs[0].set_title("K-Means Clustering")
```

```
sns.scatterplot(x=x_reduced[:,0],y=x_reduced[:,1],hue=dbscan_labels,palette="coolwarm",ax=axs[1])
axs[1].set_title("DBSCAN Clustering")
plt.show()
```



In [56]:
```
if "species" in df.columns:
    ground_truth=df["species"]
    print(f"Ground Truth Comparison:\nK-Means Labels vs Species:\n {pd.crosstab(ground_truth,kmeans_labels)}")
    print(f"\nGround Truth Comparison:\nDBSCAN Labels vs Species:\n{pd.crosstab(ground_truth[dbscan_labels!=-1],dbscan_labels[
```

```
Ground Truth Comparison:
K-Means Labels vs Species:
 col_0     0   1   2
species
0         0  50   0
1        11   0  39
2        36   0  14

Ground Truth Comparison:
DBSCAN Labels vs Species:
col_0     0   1
species
0        44   0
1         0  39
2         0  32
```

In [ ]:

In [ ]:

Program 12

In [ ]:
```
!pip install mlxtend
```

In [13]:
```
import pandas as pd
from mlxtend.frequent_patterns import apriori,association_rules
from mlxtend.preprocessing import TransactionEncoder

grocery_data=pd.read_csv(r"D:\PRIYA\SEM 05\dataset\groceries - groceries.csv")
transactions=grocery_data.iloc[:,1:].values.tolist()
transactions=[[item for item in transaction if isinstance(item,str)]
              for transaction in transactions]

te=TransactionEncoder()
te_ary=te.fit(transactions).transform(transactions)
one_hot_data=pd.DataFrame(te_ary,columns=te.columns_)

min_support=0.05
frequent_itemsets=apriori(one_hot_data,min_support=min_support,use_colnames=True)

min_confidence=0.1
rules=association_rules(frequent_itemsets,metric="confidence",min_threshold=min_confidence,num_itemsets=frequent_itemsets)

print("frequent itemsets:")
print(frequent_itemsets)
print("\n association rules:")
print(rules)
```

```
frequent itemsets:
     support                        itemsets
0   0.052466                          (beef)
1   0.080529                  (bottled beer)
2   0.110524                 (bottled water)
3   0.064870                   (brown bread)
4   0.055414                        (butter)
5   0.077682                   (canned beer)
6   0.082766                  (citrus fruit)
7   0.058058                        (coffee)
8   0.053279                          (curd)
9   0.063447                 (domestic eggs)
10  0.058973                   (frankfurter)
11  0.072293         (fruit/vegetable juice)
12  0.058566                     (margarine)
13  0.052364                       (napkins)
14  0.079817                    (newspapers)
15  0.193493              (other vegetables)
16  0.088968                        (pastry)
17  0.075648                     (pip fruit)
18  0.057651                          (pork)
19  0.183935                    (rolls/buns)
20  0.108998               (root vegetables)
21  0.093950                       (sausage)
22  0.098526                 (shopping bags)
23  0.174377                          (soda)
24  0.104931                (tropical fruit)
25  0.071683             (whipped/sour cream)
26  0.255516                    (whole milk)
27  0.139502                        (yogurt)
28  0.074835  (whole milk, other vegetables)
29  0.056634        (whole milk, rolls/buns)
30  0.056024            (whole milk, yogurt)

 association rules:
          antecedents         consequents  antecedent support  \
0        (whole milk)  (other vegetables)            0.255516   
1  (other vegetables)        (whole milk)            0.193493   
2        (whole milk)        (rolls/buns)            0.255516   
3        (rolls/buns)        (whole milk)            0.183935   
4        (whole milk)            (yogurt)            0.255516   
5            (yogurt)        (whole milk)            0.139502   

   consequent support   support  confidence      lift  representativity  \
0            0.193493  0.074835    0.292877  1.513634               1.0   
1            0.255516  0.074835    0.386758  1.513634               1.0   
2            0.183935  0.056634    0.221647  1.205032               1.0   
3            0.255516  0.056634    0.307905  1.205032               1.0   
4            0.139502  0.056024    0.219260  1.571735               1.0   
5            0.255516  0.056024    0.401603  1.571735               1.0   

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski  
0  0.025394    1.140548       0.455803  0.200000   0.123228    0.339817  
1  0.025394    1.214013       0.420750  0.200000   0.176286    0.339817  
2  0.009636    1.048452       0.228543  0.147942   0.046213    0.264776  
3  0.009636    1.075696       0.208496  0.147942   0.070369    0.264776  
4  0.020379    1.102157       0.488608  0.165267   0.092688    0.310432  
5  0.020379    1.244132       0.422732  0.165267   0.196226    0.310432  
```