Name: Berkay Yıldız

ID: 201104087

Course: Bil470

Importing the Dependincies

In [1]:
```
import numpy as np
import pandas as pd
import plotly.express as px
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from km import KMeansClustering
```

Data Loading and Basic Preprocess

In [2]:
```
iris_data = pd.read_csv('Iris.csv')
iris_data.head()
```

Out[2]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [4]:
```
iris_data = iris_data.drop('Id',axis=1)
```

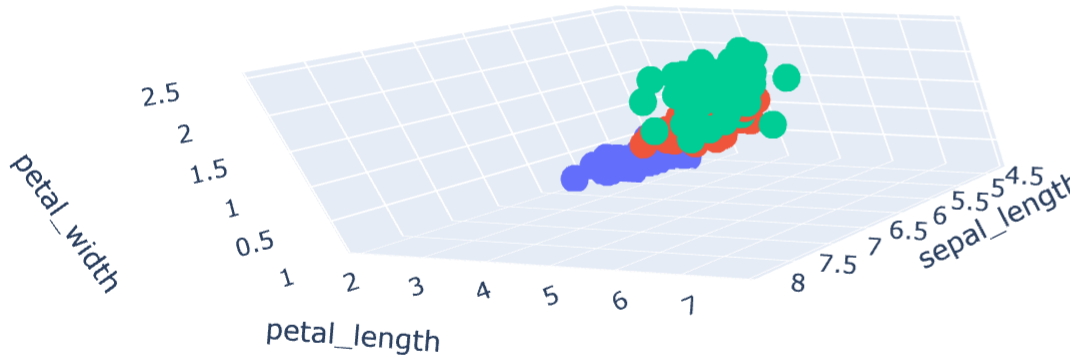
In [5]:
```
labels = LabelEncoder().fit_transform(iris_data.Species)
print(labels)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

In [6]:
```
iris_data['Target'] = labels
```

3D Cluster Plot

In [2]:
```
df = px.data.iris()
fig = px.scatter_3d(df, x='sepal_length', y='petal_length', z='petal_width', color='sp
fig.show()
```



Data Separation

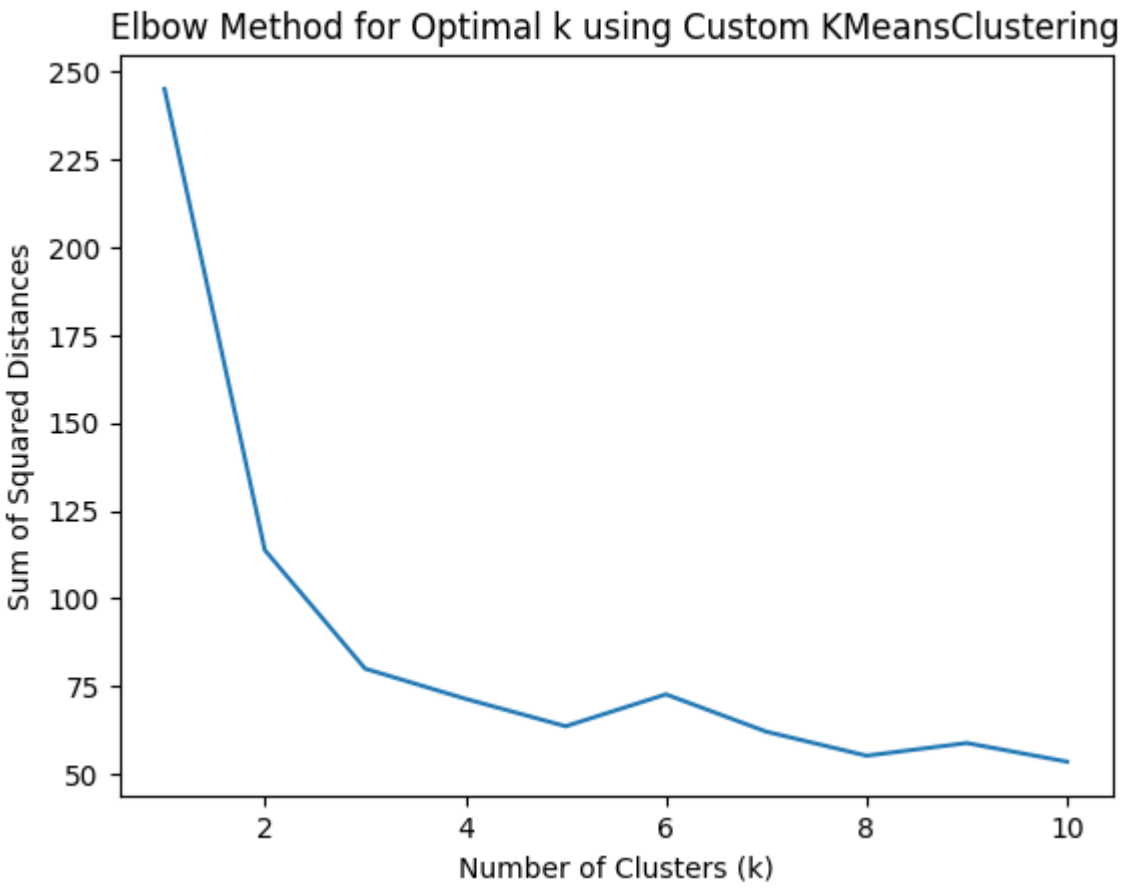
In [8]:
```
X = iris_data.drop(['Species'], axis=1)
Y = iris_data['Target']
```

In [9]:
```
X_train, X_test,Y_train,Y_test = train_test_split(X,Y, test_size = 0.2, shuffle=Tr
```

Elbow Method to find Optimal n_cluster

In [10]:
```
k_values = list(range(1, 11))
distances = []
```

```
for k in k_values:
    model = KMeansClustering(n_cluster=k)
    centroids, labels = model.fit(X_train.values.tolist())
    predictions = model.predict(X_train.values.tolist())
    distances.append(model.sumOfSquaredDistances(X_train.values.tolist(), predictions)

plt.plot(k_values, distances)
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Sum of Squared Distances')
plt.title('Elbow Method for Optimal k using Custom KMeansClustering')
plt.show()
```



Training Data

In [65]:
```
km_model = KMeansClustering(3)
centroids, labels = km_model.fit(X_train.values.tolist())
```

In [66]:
```
predictions = km_model.predict(X_test.values.tolist())
train_preds = km_model.predict(X_train.values.tolist())
```

Model Evaluation: Classification Report (Train-Test)

In [67]:
```
print(classification_report(train_preds, Y_train, target_names=['setosa', 'versico
```

```
                precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        39
  versicolor       1.00      0.93      0.97        46
   virginica       0.92      1.00      0.96        35

    accuracy                           0.97       120
   macro avg       0.97      0.98      0.98       120
weighted avg       0.98      0.97      0.98       120
```

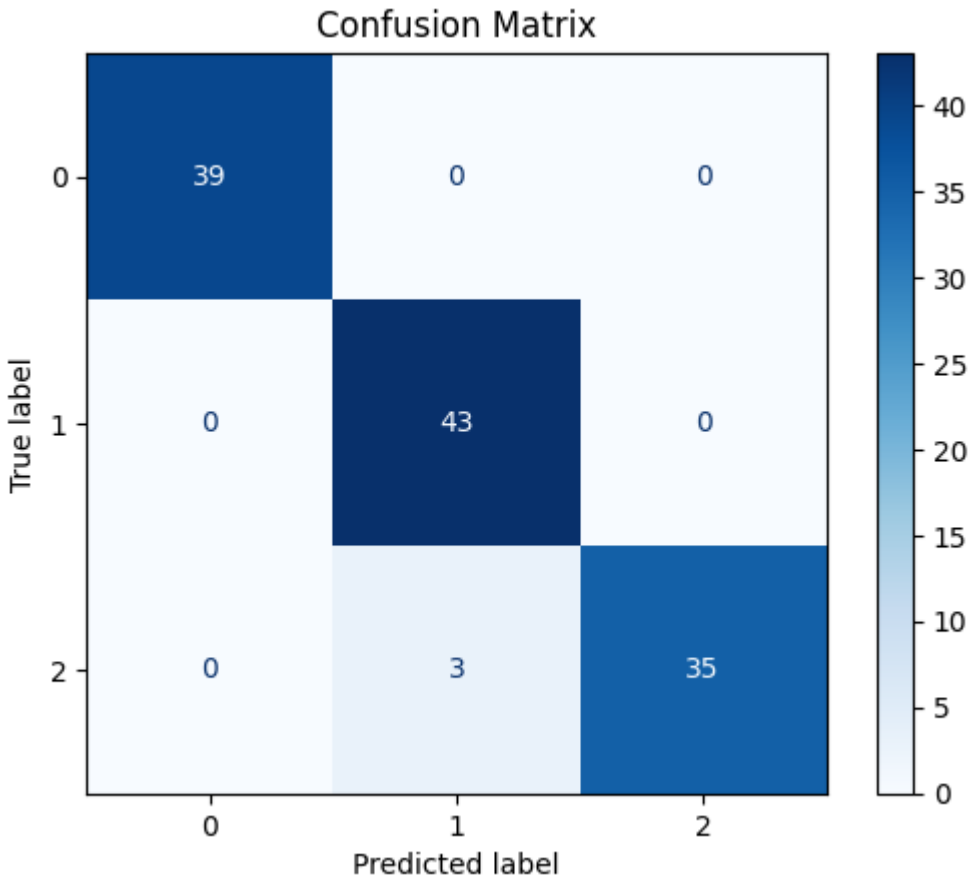
In [68]:
```
print(classification_report(predictions, Y_test, target_names=['setosa', 'versicol
```

```
                precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      1.00      1.00         7
   virginica       1.00      1.00      1.00        12

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Confusion Matrix Visualization (Train-Test)

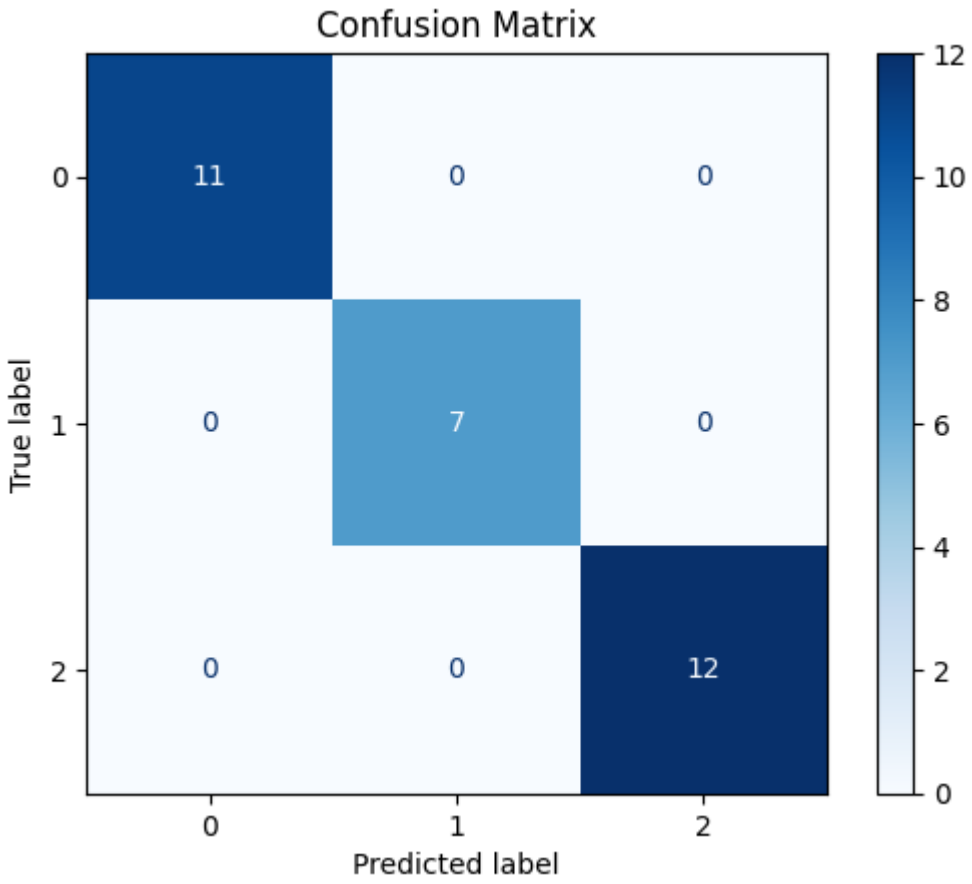
In [69]:
```
cm = confusion_matrix(Y_train, train_preds)
display_labels = np.unique(iris_data['Target'])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=display_labels)
plt.figure(figsize=(8, 6))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```



In [74]:
```
cm = confusion_matrix(Y_test, predictions)
display_labels = np.unique(iris_data['Target'])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=display_labels)
plt.figure(figsize=(8, 6))
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```

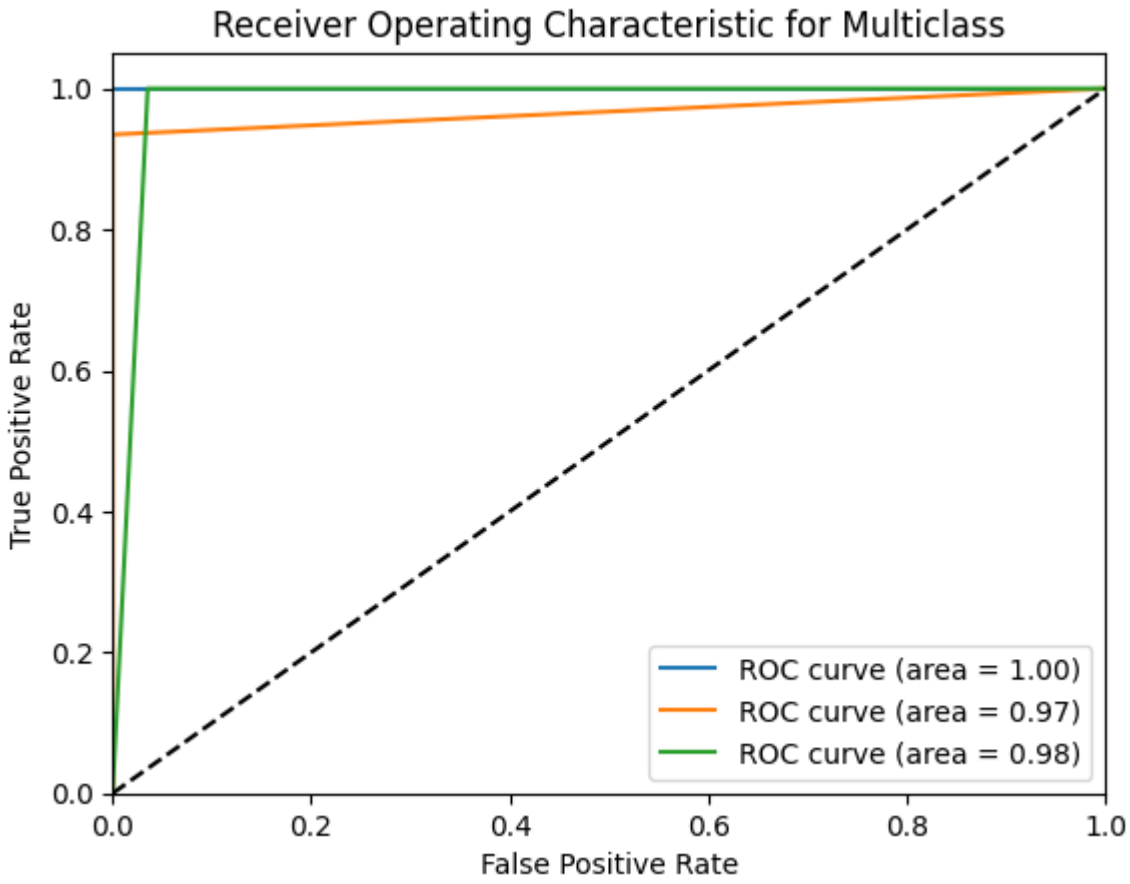
```
<Figure size 800x600 with 0 Axes>
```



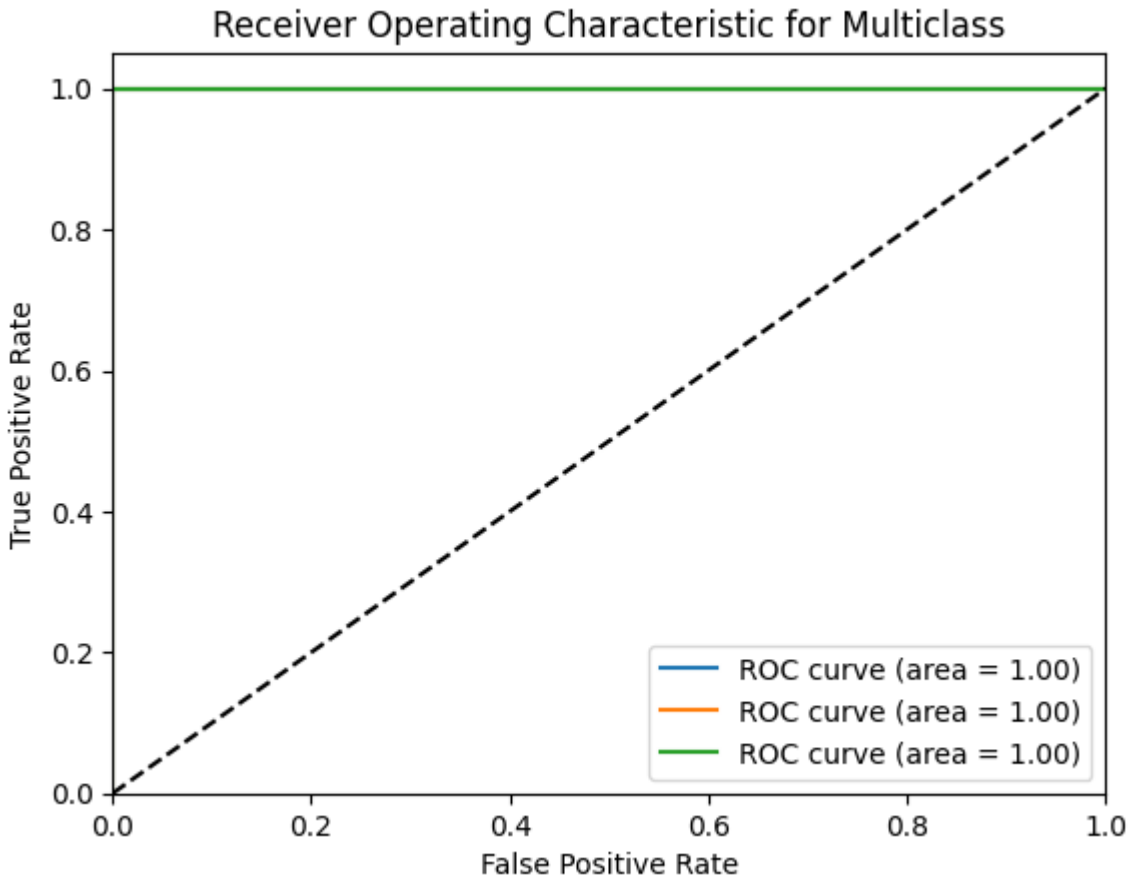
Multiclass ROC Curve Analysis (Train-Test)

In [72]:

```
y = label_binarize(pd.Series(train_preds), classes=[0, 1, 2])
y2 = label_binarize(pd.Series(Y_train), classes=[0, 1, 2])

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(y[:, i], y2[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure()

for i in range(3):
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Multiclass')
plt.legend(loc="lower right")
plt.show()
```



In [73]:
```
y = label_binarize(pd.Series(predictions), classes=[0, 1, 2])
y2 = label_binarize(pd.Series(Y_test), classes=[0, 1, 2])

fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(y[:, i], y2[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure()

for i in range(3):
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Multiclass')
plt.legend(loc="lower right")
plt.show()
```



Yorumlar

- Model hem eğitim hem de test verilerinde oldukça yüksek doğruluk ve performans gösteriyor.
- Modelin verileri iyi öğrendiği ve yeni verilere de iyi genellendiği görülüyor. -Precision, recall ve F1-Score gibi metriklerin yüksek olması, modelin sınıf tahminlerinde dengeyi sağladığını gösteriyor. -ROC eğrisi analizi ve AUC hesaplama sonuçları da bu modelin sınıflandırma yeteneğini vurguluyor. Sonuç olarak, K-Means kümeleme sınıflandırıcısı iyi çalışıyor gibi görünüyor ve test verilerinde de yüksek performans gösteriyor.

Karar Ağacı ve K-Means Clustering karşılaştırması

- Karar ağacı modelinin yorumlanabilmesi daha kolay olduğundan model anlaşılırlığı daha kolaydır.
- Karar ağacındaki modeli %100 doğrulukta çalışabilirken, K-means modelinde training için bu performansa ulaşamadım. Bunun nedenleri şunlar olabilir:
  - Karar ağacı modeli overifit olmuş olabilir. Çok fazla verimizin olmaması sonucu bu elimizdeki verilerle doğru çalışmıştır.
  - Karar ağacı ve K-means farklı türde algoritmalar ve problemler için kullanılır. Karar ağacı sınıflandırma problemleri için kullanılırken, K-means veriyi kümelemek için

kullanılır. -K-means etkietsiz veride de çalışabildiğinden veride desen keşfi ve gruplar oluşturma için kullanışlıdır. -K-means, karar ağacı modeline göre daha kolay yazılabilir. -K-means karmaşık, doğrusal olmayan veri dağılımlarıyla iyi sonuç vermeyebilir.