KNN Analysis for simulated dataset

Creating an artificial dataset:

For creating an artificial dataset, we used the make_blobs from the sk.learn datasets and we have also loaded the required packages. Make_blobs function creates a cluster of points with specific centers.

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

centers = [[2, 4], [6, 6], [1, 9]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=150,
                          centers=np.array(centers),
                          random_state=1)
```

Splitting the dataset:

Splitting of dataset into training and testing, by using the train_test_split function from the sklearn package I have divided the data into two sets. Which is 80% used for the training and the remaining 20% used for the testing.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=12)
```

Performing a KNN analysis of the simulated data:

Here to perform the KNN analysis by importing the Kneighbhorsclassifier from sklearn package. Using of Kneighorsclassifier determines the how many nearest k neighbhors to look at.

```
5]  from sklearn.neighbors import KNeighborsClassifier

    # classifier "out of the box", no parameters
    knn = KNeighborsClassifier()
    knn.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

Then did some interesting metrics to print like the accuracy, the predictions from the classifier and the target values.

```
17] print("Predictions from the classifier:")
    learn_data_predicted = knn.predict(X_train)
    print(learn_data_predicted)
    print("Target values:")
    print(y_train)
    print(accuracy_score(learn_data_predicted, y_train))
```

```
Predictions from the classifier:
[0 2 1 0 0 1 1 2 2 0 2 2 2 1 1 0 0 2 1 1 0 0 0 1 1 2 0 0 1 0 1 1 1 0 1 2 0
 1 0 1 2 2 2 0 2 0 2 2 0 0 0 1 2 2 2 2 1 1 0 1 2 1 2 2 2 0 0 0 0 0 0 0 1 1
 2 1 2 1 2 2 1 1 1 0 2 1 2 1 0 1 2 1 0 2 0 1 2 2 0 2 1 0 0 2 1 1 2 2 0 1 1
 1 2 2 2 1 1 2 1 2]
Target values:
[0 2 1 0 0 1 1 2 2 0 2 2 2 1 1 0 0 2 1 1 0 0 0 1 1 2 0 0 1 0 1 1 1 0 1 2 0
 1 0 1 2 2 2 0 2 0 2 2 0 0 0 1 2 2 2 2 1 1 0 1 2 1 2 2 2 0 0 0 0 0 0 0 1 1
 2 1 2 1 2 2 1 1 1 0 2 1 2 1 0 1 2 1 0 2 0 1 2 2 0 2 1 0 0 2 1 1 2 2 0 1 1
 1 2 2 2 1 1 2 1 2]
1.0
```

From the above we can say that the predictions from the classifier and the target values are same and the accuracy is 1. It tells us that the model perfectly fits the model, and it will work well on unseen data.

Evaluate K-nearest neighbors classifier for different values of k and determine the accuracy:

It is important to see how the classifier works for different values of k and determine the accuracy for different values of k. The values for k from 1-20

```
k_values = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
test_accuracy = []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    test_data_predicted = knn.predict(X_test)
    test_accuracy.append(accuracy_score(test_data_predicted, y_test))
print(test_accuracy)
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

From the output we can say that dataset was well separated. Generally the accuracy of K = 1 suggests that labels were assigned properly but in this case of our dataset is too small, hence the accuracy of k= 1 suggests that the dataset is well separated.

Plot your different results:

Here I have created three separate scatterplots which are the following.

1.Shows the original dataset.

2.Shows the predicted labels on training data.

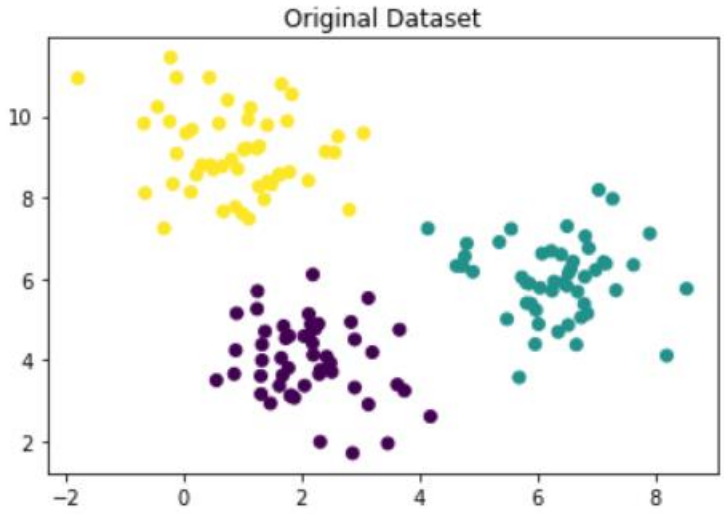
3.Shows the predicted labels on testing data.

4.Shows the K-nearest neighbhors classifier accuracy using different values of k

1.Scatter plot of original dataset

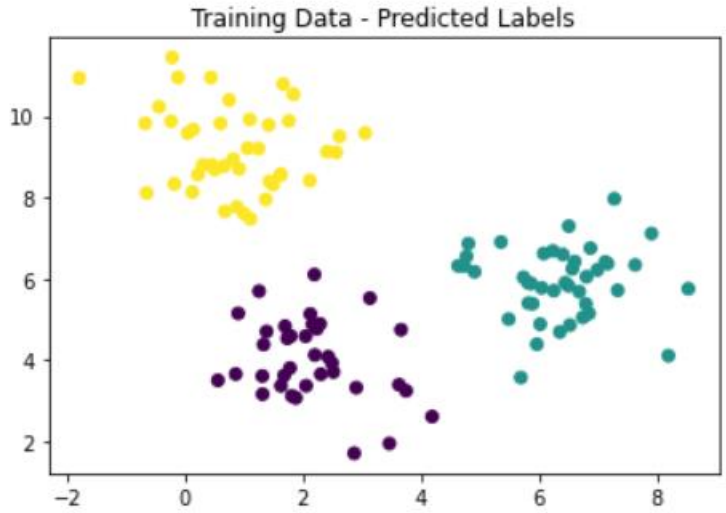
```
# Scatter plot of the original dataset

plt.scatter(data[:, 0], data[:, 1], c=labels)
plt.title("Original Dataset")
plt.show()
```



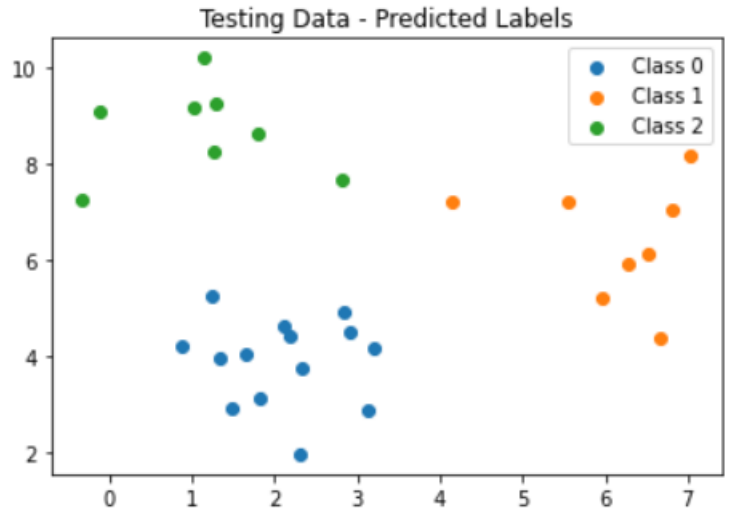
2.scatter plot of predicted labels on training data

```
# Scatter plot of the predicted labels on the training data
plt.scatter(X_train[:, 0], X_train[:, 1], c=learn_data_predicted)
plt.title("Training Data - Predicted Labels")
plt.show()
```



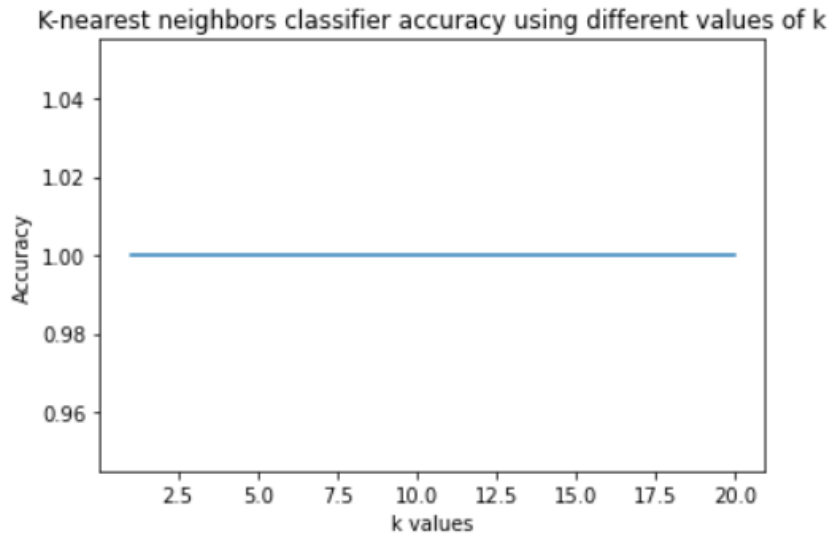
3.scatter plot of predicted labels on testing

```
# Scatter plot of the predicted labels on the testing data
fig, ax = plt.subplots()
for i in range(n_classes):
    X = X_test[y_test == i]
    y = test_data_predicted[y_test == i]
    ax.scatter(X[:, 0], X[:, 1], label=f"Class {i}")
ax.set_title("Testing Data - Predicted Labels")
ax.legend()
plt.show()
```



4.Plot showing the K-nearest neighbhors classifier accuracy using different values of k

```
# Accuracy using different values of k
plt.plot(k_values, test_accuracy)
plt.title("K-nearest neighbors classifier accuracy using different values of k")
plt.xlabel("k values")
plt.ylabel("Accuracy")
plt.show()
```



As discussed above, the accuracy of k is same for the different values of k which tells us that data has been well separated.