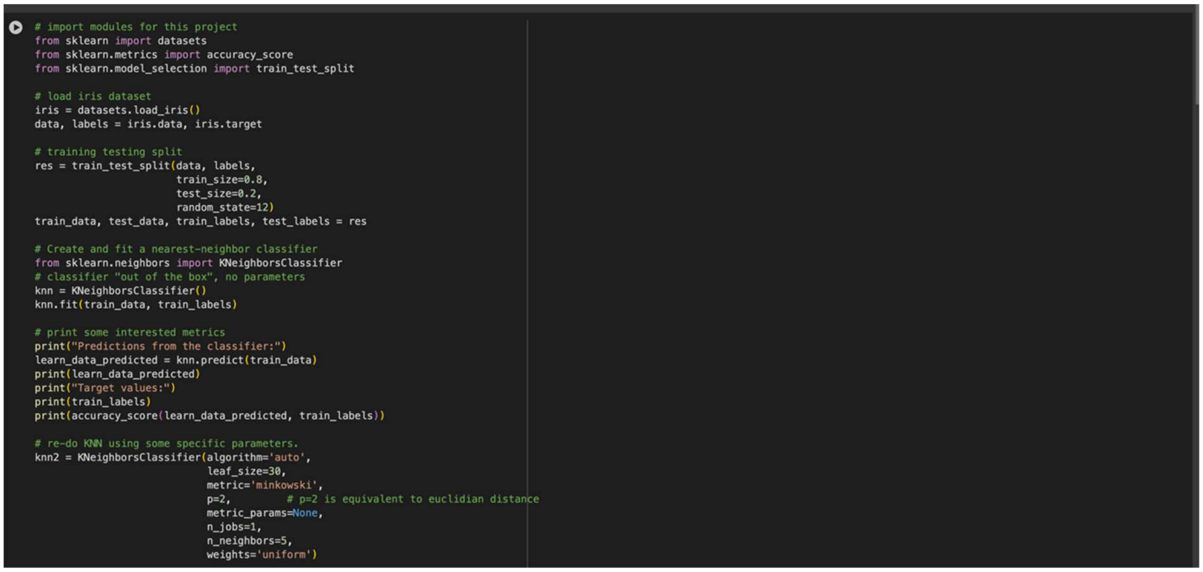
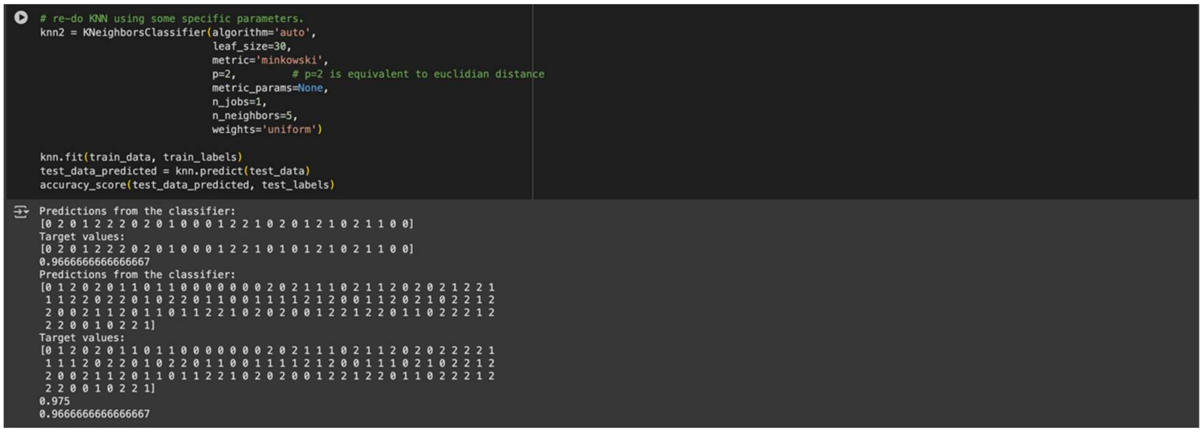
Tejasvin Maddineni
AML Assignment 1
GitHub :

1. **Understanding the Python Code in the template using iris dataset.**

- The Iris dataset was imported into Python and divided into two segments, with 80% designated for training and 20% for testing.
- A K-Nearest Neighbors (KNN) model was initially configured with default parameters and trained using the training dataset.
- The model was subsequently optimized by experimenting with various parameter adjustments on the test dataset.
- Ultimately, the accuracy of each model was computed and presented to assess overall performance.



```
# import modules for this project
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# load iris dataset
iris = datasets.load_iris()
data, labels = iris.data, iris.target

# training testing split
res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=12)
train_data, test_data, train_labels, test_labels = res

# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
# classifier "out of the box", no parameters
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)

# print some interested metrics
print("Predictions from the classifier:")
learn_data_predicted = knn.predict(train_data)
print(learn_data_predicted)
print("Target values:")
print(train_labels)
print(accuracy_score(learn_data_predicted, train_labels))

# re-do KNN using some specific parameters.
knn2 = KNeighborsClassifier(algorithm='auto',
                            leaf_size=30,
                            metric='minkowski',
                            p=2,         # p=2 is equivalent to euclidian distance
                            metric_params=None,
                            n_jobs=1,
                            n_neighbors=5,
                            weights='uniform')
```

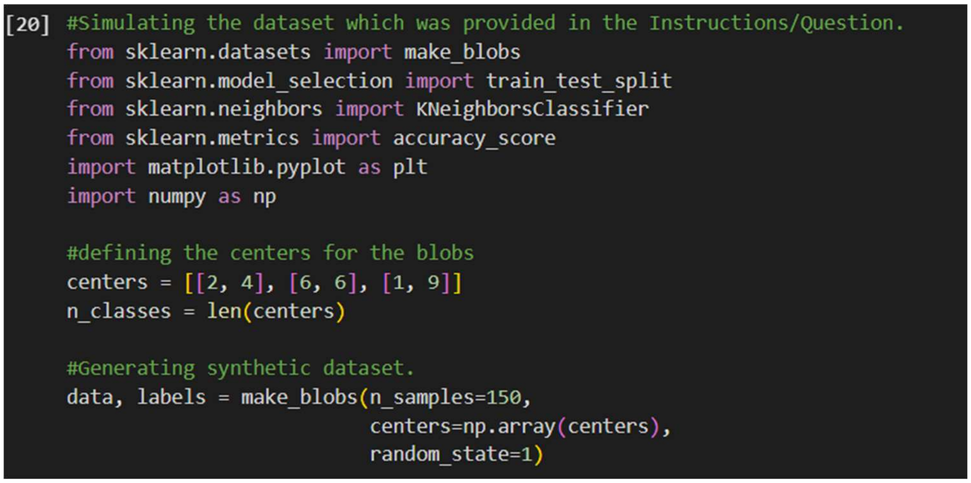


```
# re-do KNN using some specific parameters.
knn2 = KNeighborsClassifier(algorithm='auto',
                            leaf_size=30,
                            metric='minkowski',
                            p=2,         # p=2 is equivalent to euclidian distance
                            metric_params=None,
                            n_jobs=1,
                            n_neighbors=5,
                            weights='uniform')


knn.fit(train_data, train_labels)
test_data_predicted = knn.predict(test_data)
accuracy_score(test_data_predicted, test_labels)
```

```
Predictions from the classifier:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 2 0 1 2 1 0 2 1 1 0 0]
Target values:
[0 2 0 1 2 2 2 0 2 0 1 0 0 0 1 2 2 1 0 1 0 1 2 1 0 2 1 1 0 0]
0.9666666666666667
Predictions from the classifier:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 1 2 2 1
 1 1 2 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 2 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
Target values:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 2 2 2 1
 1 1 1 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 1 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
0.975
0.9666666666666667
```

2. Replicate the Study using a new simulated dataset.

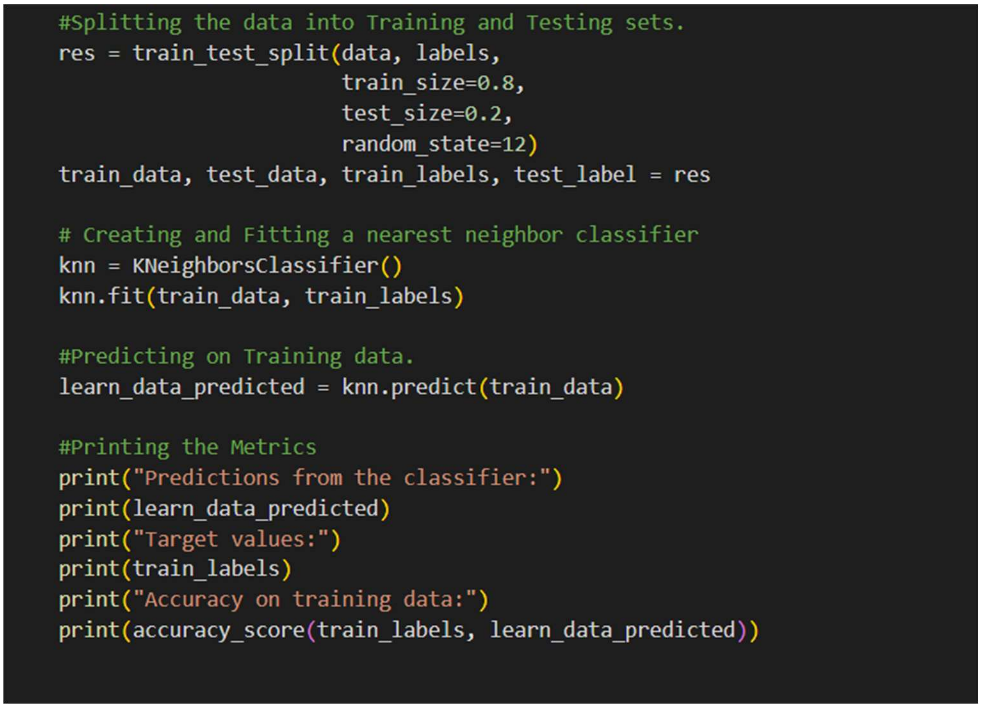
Step 1



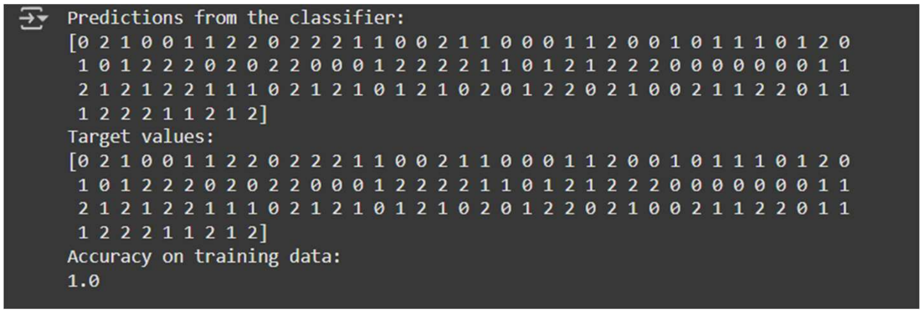
```
[20] #Simulating the dataset which was provided in the Instructions/Question.
     from sklearn.datasets import make_blobs
     from sklearn.model_selection import train_test_split
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score
     import matplotlib.pyplot as plt
     import numpy as np

     #defining the centers for the blobs
     centers = [[2, 4], [6, 6], [1, 9]]
     n_classes = len(centers)

     #Generating synthetic dataset.
     data, labels = make_blobs(n_samples=150,
                               centers=np.array(centers),
                               random_state=1)
```

We generated a synthetic dataset based on the provided instructions to simulate real-world data for analysis and model evaluation.

Step 2



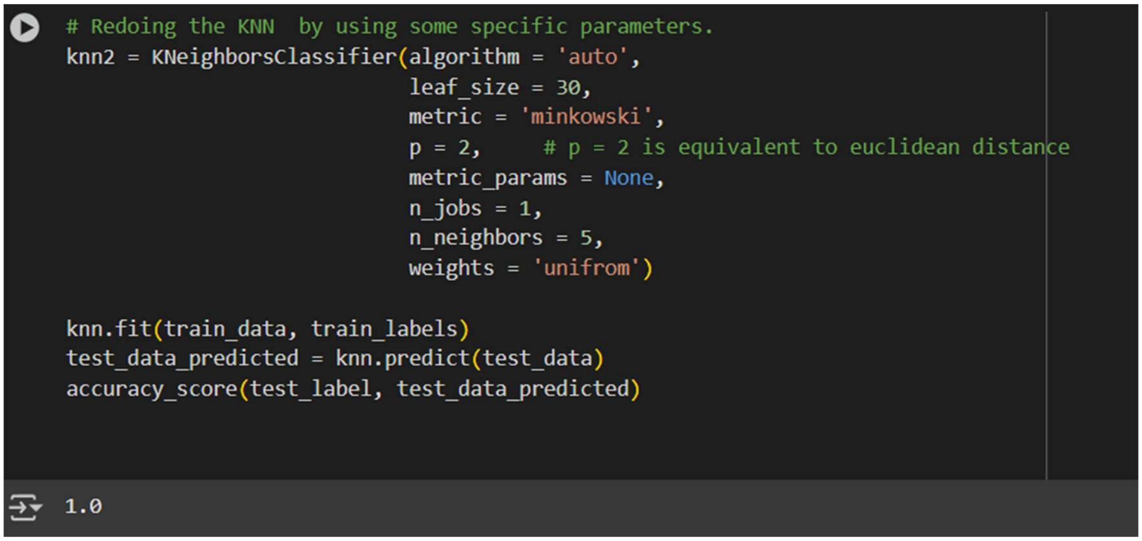
```
#Splitting the data into Training and Testing sets.
res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=12)
train_data, test_data, train_labels, test_label = res

# Creating and Fitting a nearest neighbor classifier
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)

#Predicting on Training data.
learn_data_predicted = knn.predict(train_data)

#Printing the Metrics
print("Predictions from the classifier:")
print(learn_data_predicted)
print("Target values:")
print(train_labels)
print("Accuracy on training data:")
print(accuracy_score(train_labels, learn_data_predicted))
```



```
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]
Accuracy on training data:
1.0
```

We divided the dataset into two subsets, allocating 80% for training and 20% for testing. We then trained a K-Nearest Neighbors (KNN) classifier on the training set and evaluated its performance. The classifier achieved an accuracy of 100% on the training data, indicating that it perfectly memorized the training labels.

Step 3

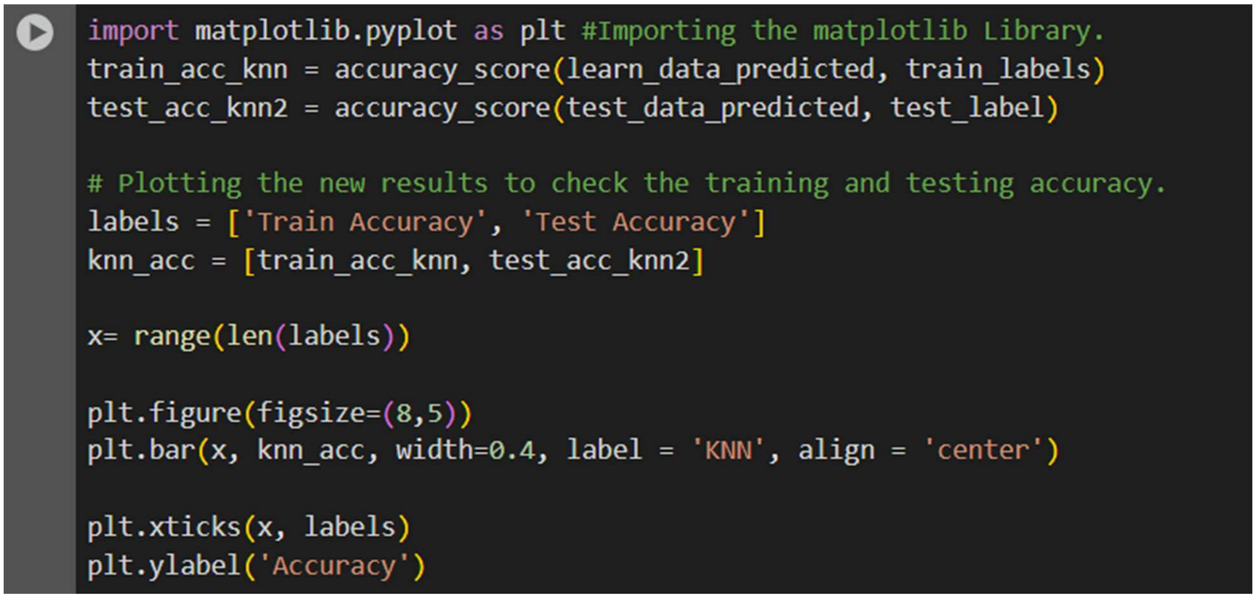


```
# Redoing the KNN  by using some specific parameters.
knn2 = KNeighborsClassifier(algorithm = 'auto',
                            leaf_size = 30,
                            metric = 'minkowski',
                            p = 2,      # p = 2 is equivalent to euclidean distance
                            metric_params = None,
                            n_jobs = 1,
                            n_neighbors = 5,
                            weights = 'unifrom')

knn.fit(train_data, train_labels)
test_data_predicted = knn.predict(test_data)
accuracy_score(test_label, test_data_predicted)
```

```
1.0
```

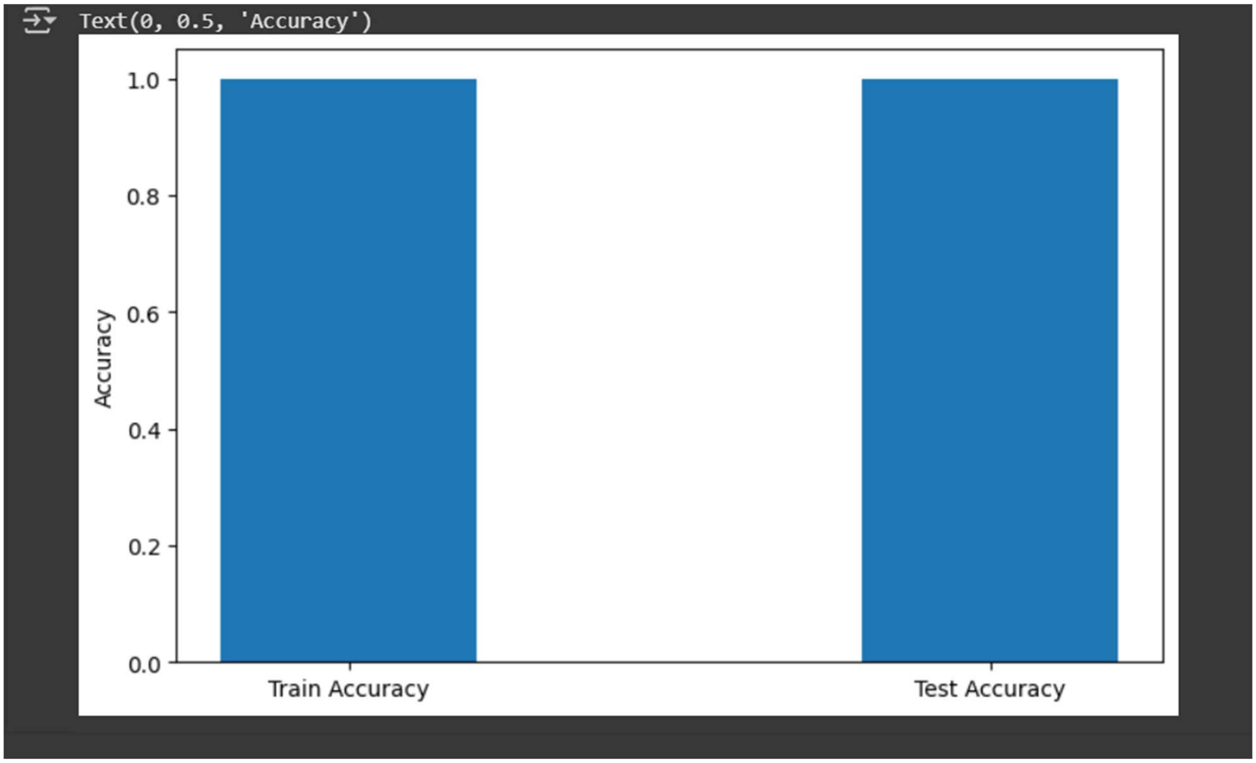
We reconfigured the K-Nearest Neighbors (KNN) model with specific parameters, using Euclidean distance (p=2) as the metric. The model was trained and then tested on the testing dataset, achieving an accuracy of 100%. The high accuracy suggests that the simulated dataset is well-structured and separable, leading to perfect classification for both training and testing data.

Step 4



```
import matplotlib.pyplot as plt #Importing the matplotlib Library.
train_acc_knn = accuracy_score(learn_data_predicted, train_labels)
test_acc_knn2 = accuracy_score(test_data_predicted, test_label)

# Plotting the new results to check the training and testing accuracy.
labels = ['Train Accuracy', 'Test Accuracy']
knn_acc = [train_acc_knn, test_acc_knn2]

x= range(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x, knn_acc, width=0.4, label = 'KNN', align = 'center')

plt.xticks(x, labels)
plt.ylabel('Accuracy')
```

A bar chart was generated to visually compare the training and testing accuracy of the KNN model, providing an intuitive way to analyze model performance.



```
Text(0, 0.5, 'Accuracy')
```

Project Summary: KNN Classification on a Simulated Dataset

In this project, we worked with a K-Nearest Neighbors (KNN) classifier to classify data points into different groups. Since we didn’t have a real dataset, we created a simulated dataset using the make_blobs function, which generated clusters of points in a 2D space.

We then split the dataset into 80% training data and 20% testing data. The KNN model was trained on the training data and then tested to see how well it could classify new points. At first, we ran the KNN with its default settings, and it got 100% accuracy on the training set, meaning it perfectly learned the data.

Next, we tweaked the model settings, using Euclidean distance (p=2) and setting the number of neighbors to 5. After retraining, the model also achieved 100% accuracy on the test set, showing that the dataset was well-structured and easy to separate.

Finally, we plotted a bar chart comparing training and testing accuracy to make the results more visual. The perfect accuracy suggests that the dataset was too simple, and in a real-world scenario, models might not perform this well. But overall, this project helped us understand how KNN works and how different settings affect its performance.