Usama Mehboob **Roll no 225** **Section E**

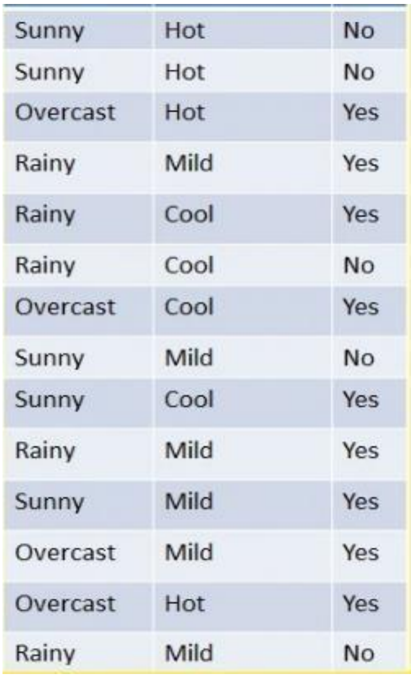
Lab 03

K-NEAREST NEIGHBOR (KNN) ALGORITHM

Objective:

Implementing K-Nearest Neighbor (KNN) algorithm to classify the data set.

Lab Task:

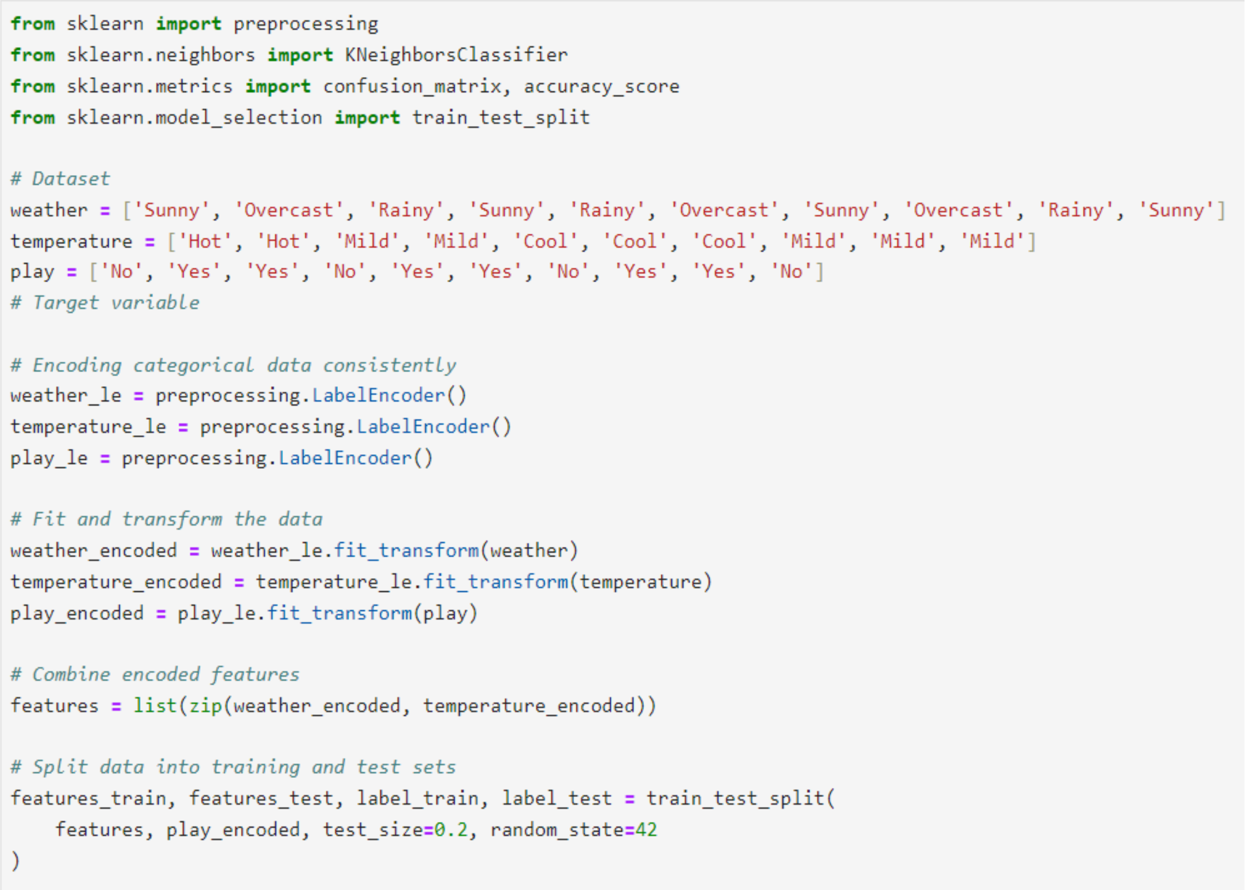


Weather	Temperature	Play
Sunny	Hot	No
Sunny	Hot	No
Overcast	Hot	Yes
Rainy	Mild	Yes
Rainy	Cool	Yes
Rainy	Cool	No
Overcast	Cool	Yes
Sunny	Mild	No
Sunny	Cool	Yes
Rainy	Mild	Yes
Sunny	Mild	Yes
Overcast	Mild	Yes
Overcast	Hot	Yes
Rainy	Mild	No

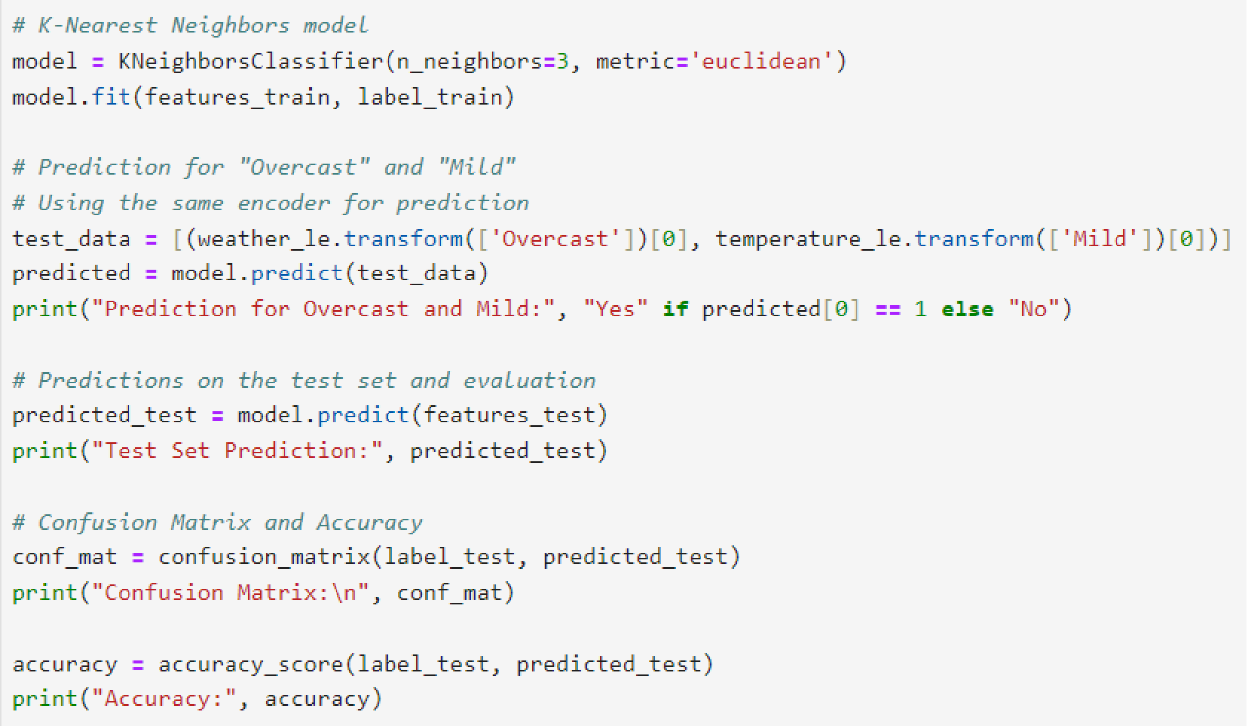
Fig 1

1. Implement K-Nearest Neighbor (KNN) Algorithm on the above dataset in Fig 1 to predict whether the players can play or not when the weather is overcast and the temperature is mild.Also apply confusion Matrix.

Code:

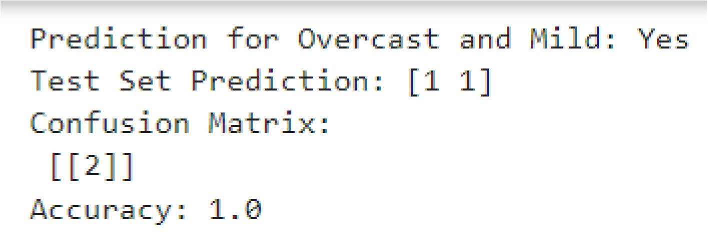


```
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split

# Dataset
weather = ['Sunny', 'Overcast', 'Rainy', 'Sunny', 'Rainy', 'Overcast', 'Sunny', 'Overcast', 'Rainy', 'Sunny']
temperature = ['Hot', 'Hot', 'Mild', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Mild', 'Mild']
play = ['No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No']
# Target variable

# Encoding categorical data consistently
weather_le = preprocessing.LabelEncoder()
temperature_le = preprocessing.LabelEncoder()
play_le = preprocessing.LabelEncoder()

# Fit and transform the data
weather_encoded = weather_le.fit_transform(weather)
temperature_encoded = temperature_le.fit_transform(temperature)
play_encoded = play_le.fit_transform(play)

# Combine encoded features
features = list(zip(weather_encoded, temperature_encoded))

# Split data into training and test sets
features_train, features_test, label_train, label_test = train_test_split(
    features, play_encoded, test_size=0.2, random_state=42
)
```



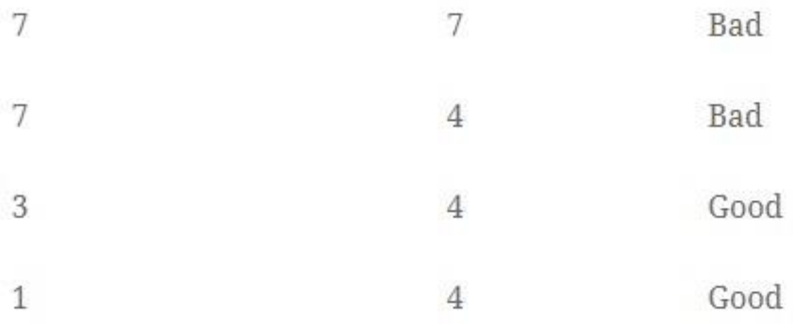
```
# K-Nearest Neighbors model
model = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
model.fit(features_train, label_train)

# Prediction for "Overcast" and "Mild"
# Using the same encoder for prediction
test_data = [(weather_le.transform(['Overcast'])[0], temperature_le.transform(['Mild'])[0])]
predicted = model.predict(test_data)
print("Prediction for Overcast and Mild:", "Yes" if predicted[0] == 1 else "No")

# Predictions on the test set and evaluation
predicted_test = model.predict(features_test)
print("Test Set Prediction:", predicted_test)

# Confusion Matrix and Accuracy
conf_mat = confusion_matrix(label_test, predicted_test)
print("Confusion Matrix:\n", conf_mat)

accuracy = accuracy_score(label_test, predicted_test)
print("Accuracy:", accuracy)
```

Output:



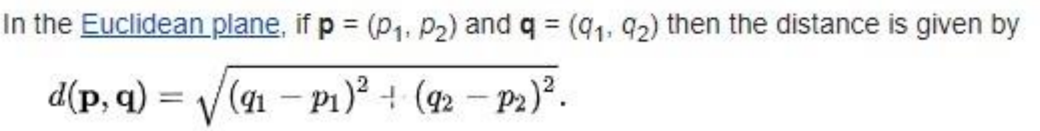
```
Prediction for Overcast and Mild: Yes
Test Set Prediction: [1 1]
Confusion Matrix:
 [[2]]
Accuracy: 1.0
```

2. Here are 4 training samples. The two attributes are acid durability and strength. Now the factory produces a new tissue paper that passes laboratory test with X1=3 and X2=7. Predict the classification of this new tissue.



X1= Acid durability (sec)	X2=Strength (kg/m^2)	Y=Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

 Calculate the Euclidean Distance between the query instance and all the training samples. Coordinate of query instance is (3,7).

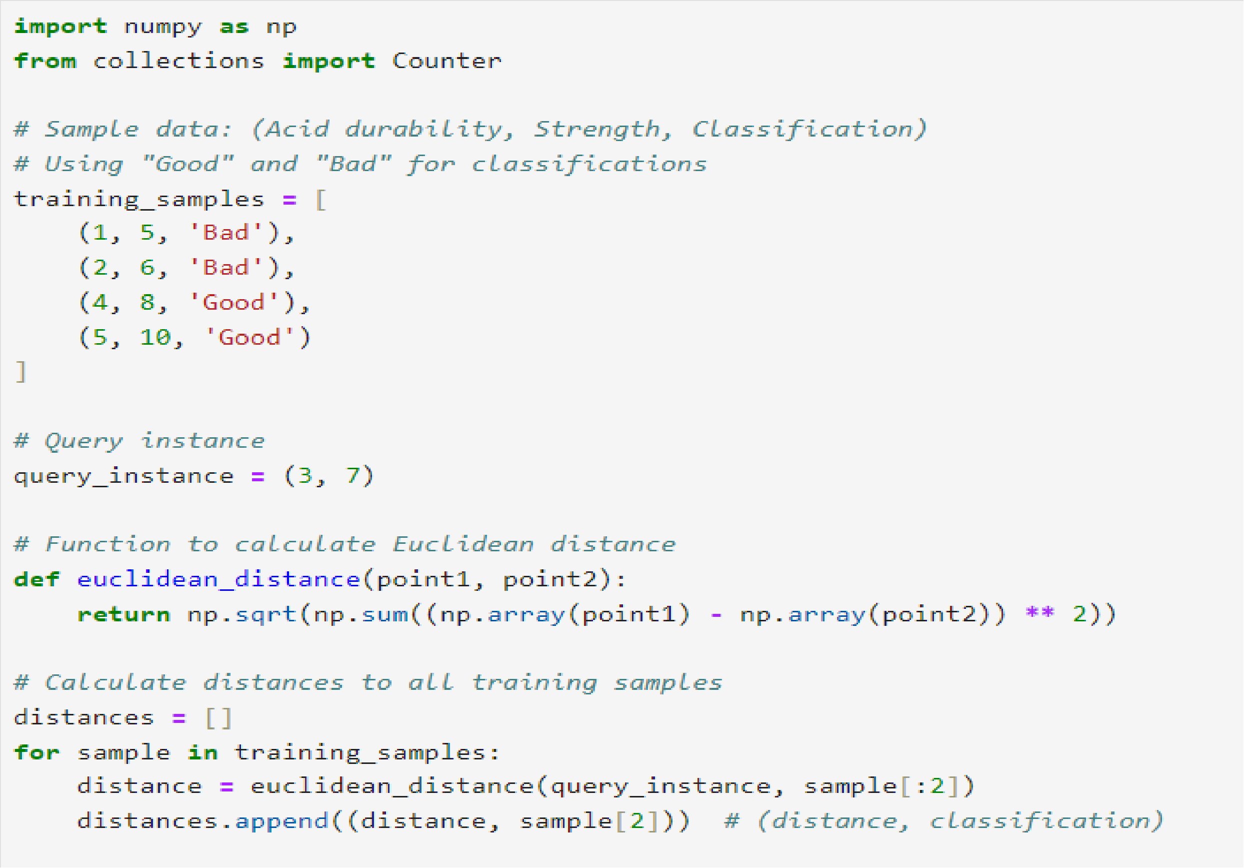


In the Euclidean plane, if **p** = (p_1, p_2) and **q** = (q_1, q_2) then the distance is given by

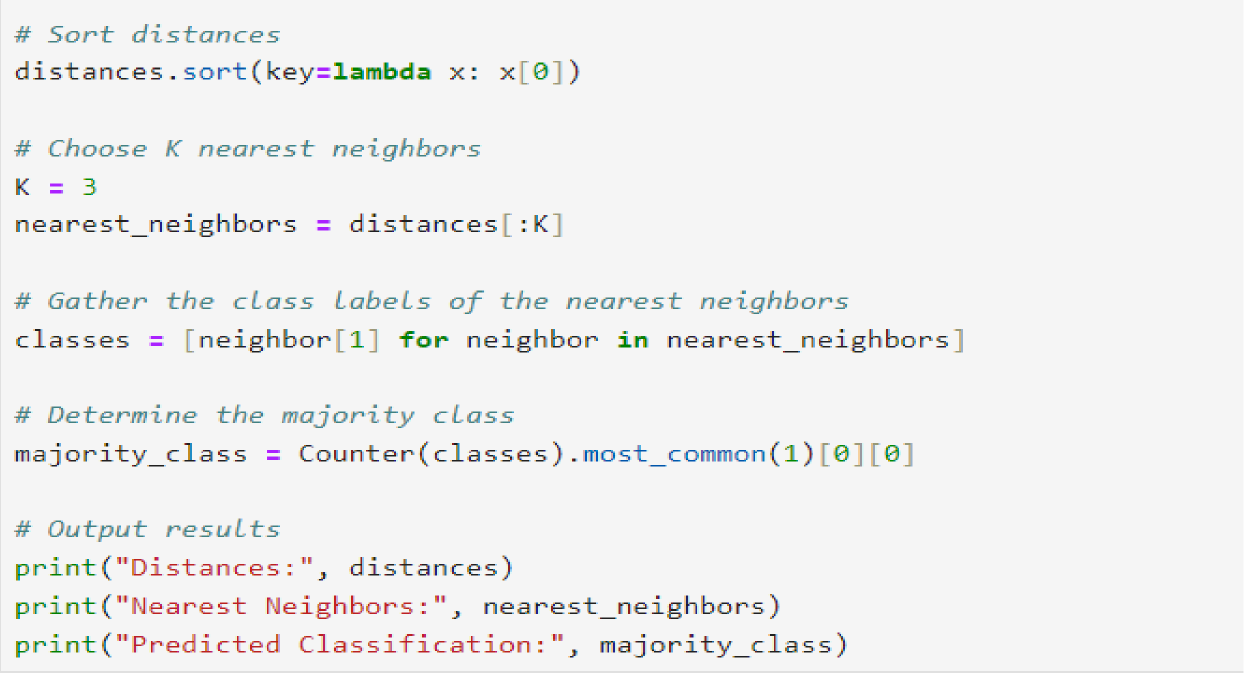
$$d(\mathbf{p},\mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

Suppose K = number of nearest neighbors = 3, sort the distances and determine nearest neighbors. Gather the class (Y) of the nearest neighbors. Use majority of the category of nearest neighbors as the prediction value of the query instance.

Code:

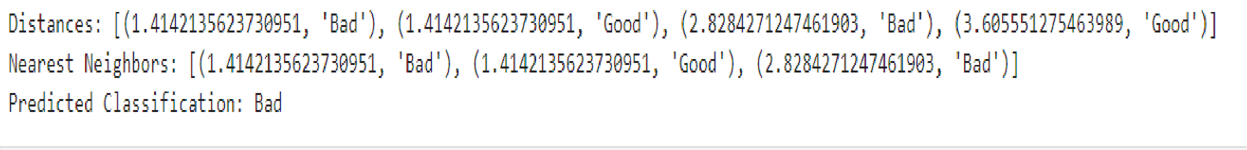


```python
import numpy as np
from collections import Counter

# Sample data: (Acid durability, Strength, Classification)
# Using "Good" and "Bad" for classifications
training_samples = [
    (1, 5, 'Bad'),
    (2, 6, 'Bad'),
    (4, 8, 'Good'),
    (5, 10, 'Good')
]

# Query instance
query_instance = (3, 7)

# Function to calculate Euclidean distance
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((np.array(point1) - np.array(point2)) ** 2))

# Calculate distances to all training samples
distances = []
for sample in training_samples:
    distance = euclidean_distance(query_instance, sample[:2])
    distances.append((distance, sample[2]))  # (distance, classification)
```



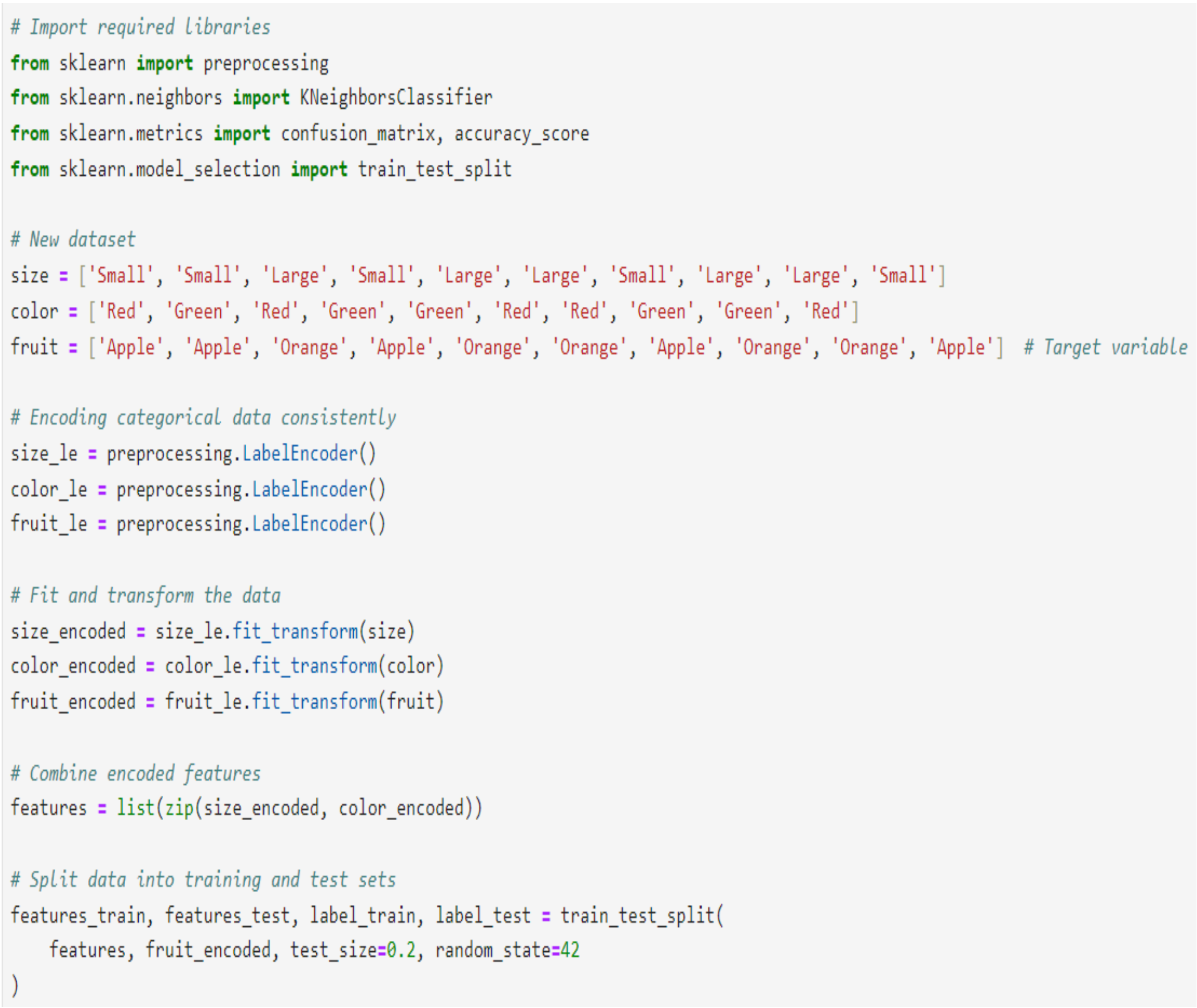
```python
# Sort distances
distances.sort(key=lambda x: x[0])

# Choose K nearest neighbors
K = 3
nearest_neighbors = distances[:K]

# Gather the class labels of the nearest neighbors
classes = [neighbor[1] for neighbor in nearest_neighbors]

# Determine the majority class
majority_class = Counter(classes).most_common(1)[0][0]

# Output results
print("Distances:", distances)
print("Nearest Neighbors:", nearest_neighbors)
print("Predicted Classification:", majority_class)
```

Output:

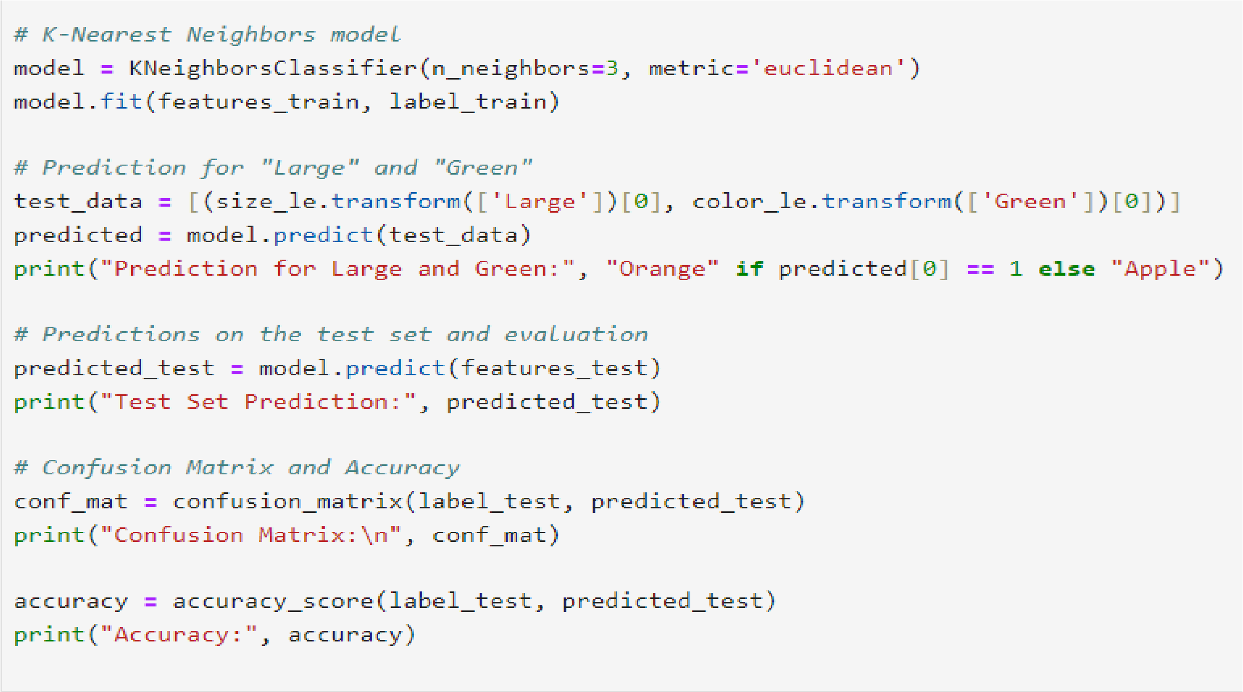


```
Distances: [(1.4142135623730951, 'Bad'), (1.4142135623730951, 'Good'), (2.8284271247461903, 'Bad'), (3.605551275463989, 'Good')]
Nearest Neighbors: [(1.4142135623730951, 'Bad'), (1.4142135623730951, 'Good'), (2.8284271247461903, 'Bad')]
Predicted Classification: Bad
```

Home Task:

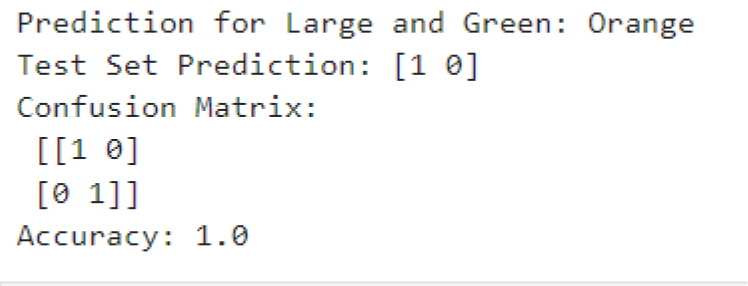


```python
# Import required libraries
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split

# New dataset
size = ['Small', 'Small', 'Large', 'Small', 'Large', 'Large', 'Small', 'Large', 'Large', 'Small']
color = ['Red', 'Green', 'Red', 'Green', 'Green', 'Red', 'Red', 'Green', 'Green', 'Red']
fruit = ['Apple', 'Apple', 'Orange', 'Apple', 'Orange', 'Orange', 'Apple', 'Orange', 'Orange', 'Apple']  # Target variable

# Encoding categorical data consistently
size_le = preprocessing.LabelEncoder()
color_le = preprocessing.LabelEncoder()
fruit_le = preprocessing.LabelEncoder()

# Fit and transform the data
size_encoded = size_le.fit_transform(size)
color_encoded = color_le.fit_transform(color)
fruit_encoded = fruit_le.fit_transform(fruit)

# Combine encoded features
features = list(zip(size_encoded, color_encoded))

# Split data into training and test sets
features_train, features_test, label_train, label_test = train_test_split(
    features, fruit_encoded, test_size=0.2, random_state=42
)
```



```
# K-Nearest Neighbors model
model = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
model.fit(features_train, label_train)

# Prediction for "Large" and "Green"
test_data = [(size_le.transform(['Large'])[0], color_le.transform(['Green'])[0])]
predicted = model.predict(test_data)
print("Prediction for Large and Green:", "Orange" if predicted[0] == 1 else "Apple")

# Predictions on the test set and evaluation
predicted_test = model.predict(features_test)
print("Test Set Prediction:", predicted_test)

# Confusion Matrix and Accuracy
conf_mat = confusion_matrix(label_test, predicted_test)
print("Confusion Matrix:\n", conf_mat)

accuracy = accuracy_score(label_test, predicted_test)
print("Accuracy:", accuracy)
```

Output:



```
Prediction for Large and Green: Orange
Test Set Prediction: [1 0]
Confusion Matrix:
 [[1 0]
 [0 1]]
Accuracy: 1.0
```