In [ ]:
```python
# Implementasi dengan KNN

import numpy as np
import pandas as pd
import pickle

dataset1 = pd.read_csv('data_train.csv')
dataset2 = pd.read_csv('data_validation.csv')
```

In [ ]:
```python
from sklearn.metrics import accuracy_score
class KNN:
  def __init__(self, training_dataset, validation_dataset):
    self.training_dataset = training_dataset
    self.validation_dataset = validation_dataset
    self.potential_k = []
    self.k = 0

    #variables for showing training result
    self.potential_k_correct_counts = {}
    self.validation_iteration = 0

  def save_model(self, filename):
    with open(filename, 'wb') as file:
        pickle.dump(self, file)

  @staticmethod
  def load_model(filename):
    with open(filename, 'rb') as file:
        return pickle.load(file)

  # training from dataset train and validation to find k
  def train(self):
    # Determining best K value
    # K ditentukan dengan menentukan distance dari tiap data di dataset valid dengan data di dataset train, dengan k data dengan distance terdekat.
    # Penghitungan k yang menghasilkan prediksi yang benar (atau sesuai dengan di dataset valid) akan dipetakan dengan berapa kali ia menghasilkan benar
    # dan di akhir, k ditentukan dari k yang menghasilkan benar paling banyak

    # Split dataset menjadi predictors and target
    training_predictors = self.training_dataset.iloc[:, :-1]
    test_predictors = self.validation_dataset.iloc[:, :-1]
    training_target = self.training_dataset.iloc[:, -1]
    validation_target = self.validation_dataset.iloc[:, -1]

    # Fungsi untuk menghitung key yang menghasilkan prediksi yang benar
    def correctKIncrement(key):
      if key in self.potential_k_correct_counts:
        self.potential_k_correct_counts[key] += 1
      else:
        self.potential_k_correct_counts[key] = 1

    # Training dataset, ambil kolom target
    df_euclidean_distance = self.training_dataset.copy()
    df_euclidean_distance = df_euclidean_distance[['price_range']]

    for i in range(len(self.validation_dataset)):
      # Initiate variabel
      k2distance_dict = {}
      temporary_column_name = "eu_dist_to_valid_row"+str(i)

      euclidean_distances = []
      differences = test_predictors.iloc[i].values - training_predictors.values
      distances = np.linalg.norm(differences, axis=1)
      euclidean_distances = distances.tolist()
      df_euclidean_distance[temporary_column_name] = euclidean_distances

      # Cari K value dengan menghitung rata-rata dari k data dengan distance terdekat
      for j in range(1,len(self.training_dataset)+1):
        k2distance_dict[j] = df_euclidean_distance.sort_values(temporary_column_name)["price_range"].head(j).mean()

      # Proses fitness K dengan menghitung prediksi yang benar dan memasukkannya ke dalam map
      for j in k2distance_dict:
        if (k2distance_dict.get(j) == validation_target[i]):
          correctKIncrement(j)
        elif (validation_target[i]-0.5 <= k2distance_dict.get(j) < validation_target[i]+0.5 and 0 <= k2distance_dict.get(j) <= 3):
          correctKIncrement(j)

      # Drop kolom temporary
      df_euclidean_distance = df_euclidean_distance.drop(columns=temporary_column_name)

      # Setting nilai K
      self.validation_iteration = i+1
      self.potential_k = [key for key, value in self.potential_k_correct_counts.items() if value == max(self.potential_k_correct_counts.values())]
      self.k = min(self.potential_k)  # K terbaik dengan K terkecil untuk mencegah overfitting

  # Show training result
  def showTrainingResult(self):
    print(f"K values with the most correct predictions are K = {self.potential_k}")
    correct_count = max(self.potential_k_correct_counts.values())
    validation_count = self.validation_iteration
    print(f"With said K value(s) being correct {correct_count} times out of {validation_count} ({correct_count/validation_count})")
    print(f'According to sklearn.metrics, the validity score of the model with accuracy_score() is {accuracy_score(self.validation_dataset.iloc[:,-1].values, self.validate())}')

  def validate(self):
    valid_dataset = self.validation_dataset
    return self.predict(valid_dataset)

  # Prediksi dari dataset test
  def predict(self, test_dataset):
    # splitting datasets
    training_predictors = self.training_dataset.iloc[:, :-1].copy()
    test_predictors = test_dataset.iloc[:,:-1].copy()
    training_target = self.training_dataset.iloc[:, -1].copy()
    validation_target = test_dataset.iloc[:,-1].copy()

    # Inisiasi variabel
    predicted_values = []
    df_euclidean_distance = self.training_dataset.copy()
    df_euclidean_distance = df_euclidean_distance[['price_range']]

    for i in range(len(test_dataset)):
      # Inisiasi variabel yang digunakan untuk setiap baris di dataset test
      temporary_column_name = "eu_dist_to_valid_row"+str(i)

      euclidean_distances = []
      differences = test_predictors.iloc[i].values - training_predictors.values
      distances = np.linalg.norm(differences, axis=1)
      euclidean_distances = distances.tolist()
      df_euclidean_distance[temporary_column_name] = euclidean_distances

      # Hitung prediksi KNN dan masukkan ke dalam array
      prediction = df_euclidean_distance.sort_values(temporary_column_name)["price_range"].head(self.k).mean()

      # putting statistical values to round categorial values (following dataset values)
      prediction = round(prediction)

      # prediction_floatval -= math.floor(prediction)
      # if float value >= 0.5 then round them upward
      # if float value is <0.5 then round them downward

      predicted_values.append(prediction)

      # Drop kolom temporary
      df_euclidean_distance = df_euclidean_distance.drop(columns=temporary_column_name)

    return predicted_values
```

Training KNN Model

In [ ]:
```python
KNN_Model = KNN(dataset1, dataset2)
KNN_Model.train()
```

Showing Training K Results

In [ ]:
```python
KNN_Model.showTrainingResult()
# Putting K dictionary into a dataframe
k_prediction_frequency = pd.DataFrame(list(KNN_Model.potential_k_correct_counts.items()), columns=["k","frequency"])
display(k_prediction_frequency)
print("Result of 50 best k values")
display(k_prediction_frequency.sort_values("frequency", ascending=False).head(50))

KNN_Model.save_model('knn_model.txt')
```

Loading KNN from file

In [ ]:
```python
loaded_knn = KNN.load_model('knn_model.txt')

loaded_knn.showTrainingResult()
```

```
k_prediction_frequency = pd.DataFrame(list(loaded_knn.potential_k_correct_counts.items()), columns=["k","frequency"])
display(k_prediction_frequency)
print("Result of 50 best k values")
display(k_prediction_frequency.sort_values("frequency", ascending=False).head(50))
```