**K-NN Classifier**

**CSE 303: Machine Learning**

Submitted by

Name: Roda Chinthapalli

Roll No: AP22110011496

Section: M

Lab Date: 17/09/24

End Date: 23/09/24

**A picture containing text

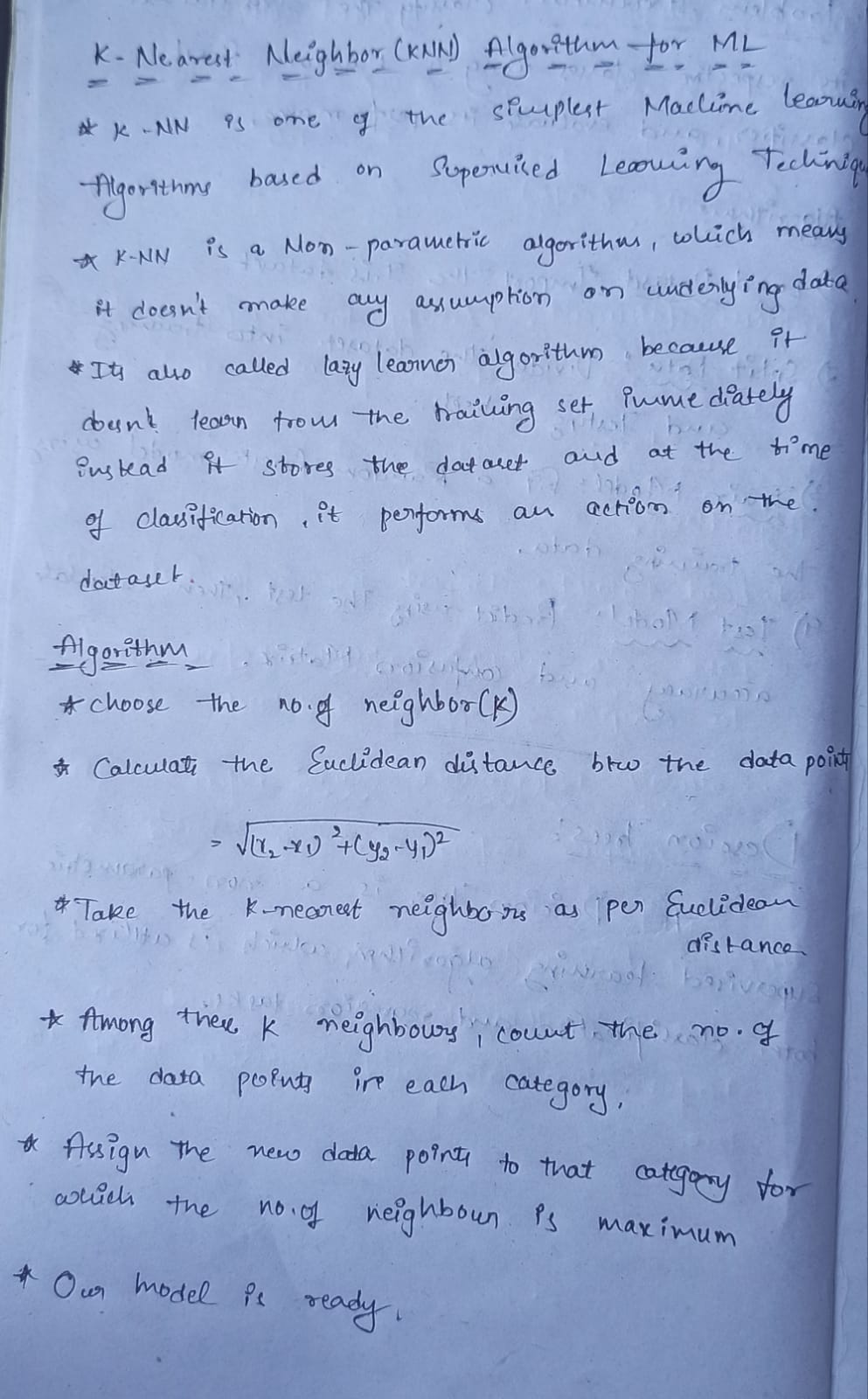
Description automatically generated**

**Department Computer Science and Engineering**

**School of Engineering and Sciences**

**SRM University–AP**

**Amaravati, Andhra Pradesh – 522 240, India**

****

**1.Question:**

1. Implement K-NN Classifier for classification of any dataset of your choice.

a. Load an existing data set

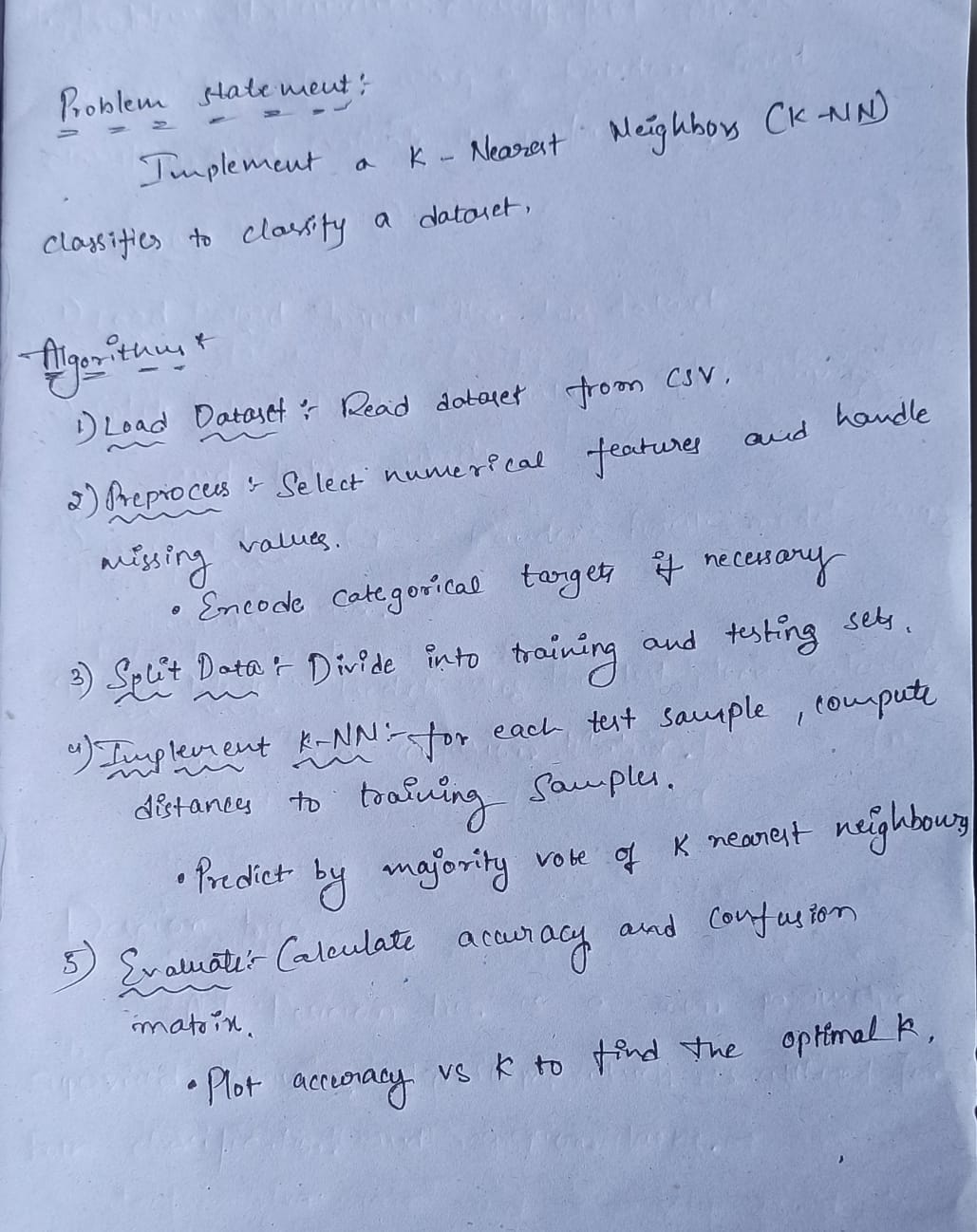
b. Split the data set to train and test sets

c. Test your model using test set. Find accuracy and confusion Matrix.

d. Examine the effect of the value of K on accuracy/performance. Plot the curve “k vs

accuracy” and find out the value of k for maximum accuracy for the test samples.

**Problem Statement and Algorithm:**



**Solution:**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from sklearn.preprocessing import LabelEncoder**

**Load the Dataset**

**# Step 1: Load the Dataset**

**def load\_dataset(file\_path):**

**df = pd.read\_csv(file\_path)**

**return df**

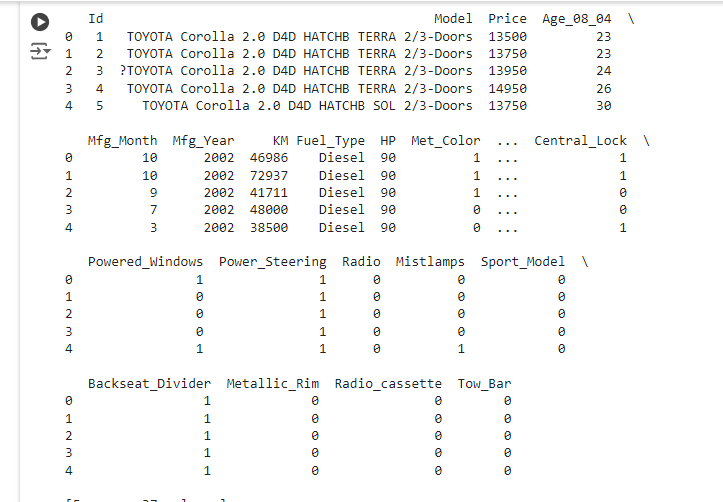
**# File path to the dataset**

**file\_path = '/content/ToyotaCorolla.csv'**

**data = load\_dataset(file\_path)**

**# Display the first few rows of the dataset**

**print(data.head())**

****

**# Get the number of rows and columns**

**num\_rows, num\_columns = data.shape**

**# Print the number of rows and columns**

**print(f"Number of rows: {num\_rows}")**

**print(f"Number of columns: {num\_columns}")**

****

**# Preprocess the data**

**def preprocess\_data(df):**

**# Select numerical columns only for simplicity**

**df = df.select\_dtypes(include=[np.number]).dropna() # Use only numerical columns and drop rows with NaN**

**# Encode categorical targets if necessary**

**if df.iloc[:, -1].dtype == 'object':**

**le = LabelEncoder()**

**df.iloc[:, -1] = le.fit\_transform(df.iloc[:, -1])**

**X = df.iloc[:, :-1].values**

**y = df.iloc[:, -1].values**

**return X, y**

**X, y = preprocess\_data(data)**

**b. Split the data set to train and test sets**

**# Step 2: Split Data**

**def train\_test\_split(X, y, test\_size=0.3):**

**np.random.seed(42)**

**indices = np.arange(X.shape[0])**

**np.random.shuffle(indices)**

**split\_index = int(X.shape[0] \* (1 - test\_size))**

**train\_indices = indices[:split\_index]**

**test\_indices = indices[split\_index:]**

**return X[train\_indices], X[test\_indices], y[train\_indices], y[test\_indices]**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)**

**c. Test your model using test set. Find accuracy and confusion Matrix.**

**# Step 3: Implement K-NN Classifier**

**def euclidean\_distance(x1, x2):**

**return np.sqrt(np.sum((x1 - x2) \*\* 2))**

**def knn\_predict(X\_train, y\_train, X\_test, k):**

**y\_pred = []**

**for test\_point in X\_test:**

**distances = [euclidean\_distance(test\_point, x\_train) for x\_train in X\_train]**

**k\_nearest\_indices = np.argsort(distances)[:k]**

**k\_nearest\_labels = [y\_train[i] for i in k\_nearest\_indices]**

**most\_common = np.bincount(k\_nearest\_labels).argmax()**

**y\_pred.append(most\_common)**

**return np.array(y\_pred)**

**# Calculate accuracy and confusion matrix**

**def accuracy(y\_true, y\_pred):**

**return np.mean(y\_true == y\_pred)**

**def confusion\_matrix(y\_true, y\_pred):**

**classes = np.unique(y\_true)**

**matrix = np.zeros((len(classes), len(classes)), dtype=int)**

**for true\_label, pred\_label in zip(y\_true, y\_pred):**

**matrix[true\_label, pred\_label] += 1**

**return matrix**

**d. Examine the effect of the value of K on accuracy/performance. Plot the curve “k vs accuracy” and find out the value of k for maximum accuracy for the test samples.**

**# Step 4: Evaluate Performance**

**def evaluate\_performance(X\_train, y\_train, X\_test, y\_test, k\_values):**

**accuracies = []**

**for k in k\_values:**

**y\_pred = knn\_predict(X\_train, y\_train, X\_test, k)**

**accuracy\_value = accuracy(y\_test, y\_pred)**

**accuracies.append(accuracy\_value)**

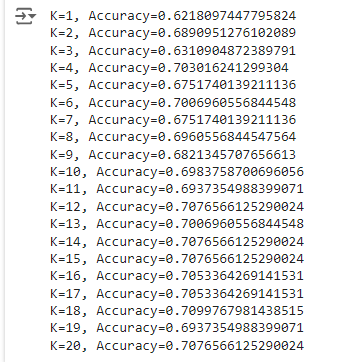
**print(f'K={k}, Accuracy={accuracy\_value}')**

**return accuracies**

**# Determine the optimal K**

**k\_values = range(1, 21)**

**accuracies = evaluate\_performance(X\_train, y\_train, X\_test, y\_test, k\_values)**

****

**# Plot the accuracy vs. K**

**plt.plot(k\_values, accuracies, marker='o')**

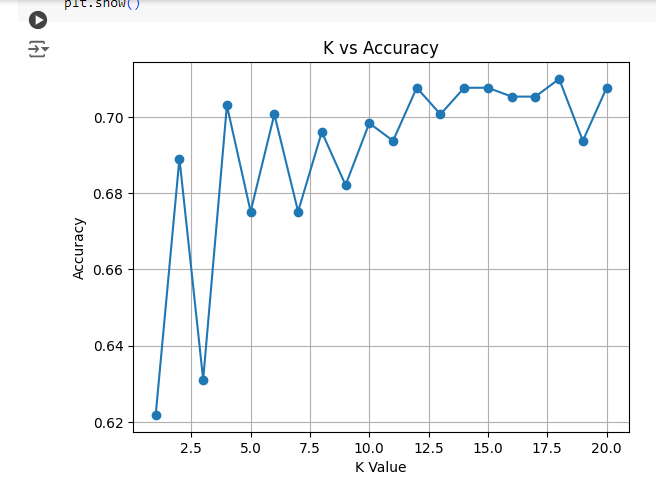
**plt.xlabel('K Value')**

**plt.ylabel('Accuracy')**

**plt.title('K vs Accuracy')**

**plt.grid(True)**

**plt.show()**

****

**# Find the minimum K with the highest accuracy**

**max\_accuracy = max(accuracies)**

**optimal\_ks = [k for k, acc in zip(k\_values, accuracies) if acc == max\_accuracy]**

**best\_k = min(optimal\_ks)**

**print(f'The optimal (smallest) K is {best\_k}')**

****

**# For confusion matrix and accuracy with the best K**

**y\_best\_pred = knn\_predict(X\_train, y\_train, X\_test, best\_k)**

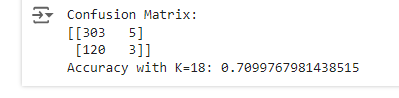
**conf\_matrix = confusion\_matrix(y\_test, y\_best\_pred)**

**accuracy\_value = accuracy(y\_test, y\_best\_pred)**

**print('Confusion Matrix:')**

**print(conf\_matrix)**

**print(f'Accuracy with K={best\_k}: {accuracy\_value}')**

****