#1

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score

dataset = pd.read\_csv("diabetes.csv")

class\_0 = dataset[dataset['Outcome'] == 0].drop(columns=['Outcome'])

class\_1 = dataset[dataset['Outcome'] == 1].drop(columns=['Outcome'])

centroid\_0 = class\_0.mean(axis=0)

centroid\_1 = class\_1.mean(axis=0)

spread\_0 = class\_0.std(axis=0)

spread\_1 = class\_1.std(axis=0)

interclass\_distance = np.linalg.norm(centroid\_0- centroid\_1)

print("Centroid Class 0:\n", centroid\_0)

print("Centroid Class 1:\n", centroid\_1)

print("Spread Class 0:\n", spread\_0)

print("Spread Class 1:\n", spread\_1)

print("Interclass Distance:", interclass\_distance)

**OUTPUT:**

Centroid Class 0:

Pregnancies 3.298000

Glucose 109.980000

BloodPressure 68.184000

SkinThickness 19.664000

Insulin 68.792000

BMI 30.304200

DiabetesPedigreeFunction 0.429734

Age 31.190000

dtype: float64

Centroid Class 1:

Pregnancies 4.865672

Glucose 141.257463

BloodPressure 70.824627

SkinThickness 22.164179

Insulin 100.335821

BMI 35.142537

DiabetesPedigreeFunction 0.550500

Age 37.067164

dtype: float64

Spread Class 0:

Pregnancies 3.017185

Glucose 26.141200

BloodPressure 18.063075

SkinThickness 14.889947

Insulin 98.865289

BMI 7.689855

DiabetesPedigreeFunction 0.299085

Age 11.667655

dtype: float64

Spread Class 1:

Pregnancies 3.741239

Glucose 31.939622

BloodPressure 21.491812

SkinThickness 17.679711

Insulin 138.689125

BMI 7.262967

DiabetesPedigreeFunction 0.372354

Age 10.968254

dtype: float64

Interclass Distance: 45.24310842710197

#2

feature = 'Glucose'

hist\_values, bin\_edges = np.histogram(dataset[feature], bins=10)

plt.figure(figsize=(8, 5))

plt.hist(dataset[feature], bins=10, edgecolor='black', alpha=0.7)

plt.xlabel(feature)

plt.ylabel("Frequency")

plt.title(f"Histogram of {feature}")

plt.grid(True)

plt.show()

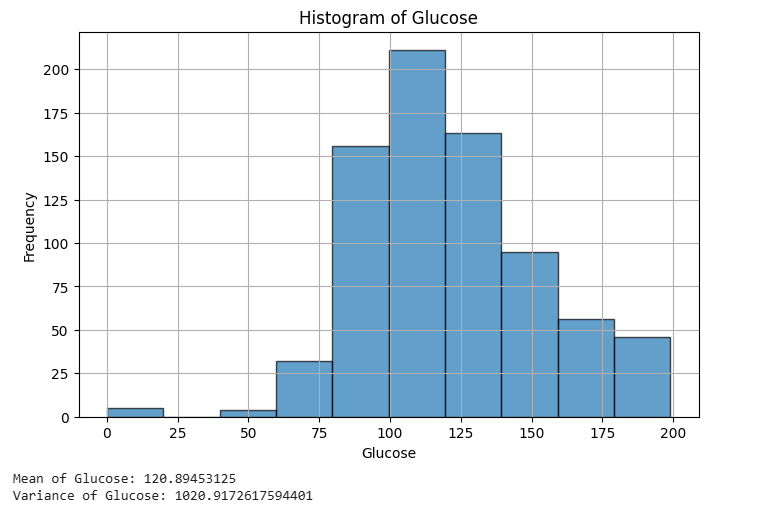
mean\_value = np.mean(dataset[feature])

variance\_value = np.var(dataset[feature])

print(f"Mean of {feature}: {mean\_value}")

print(f"Variance of {feature}: {variance\_value}")

**OUTPUT:**



#3

vector1 = dataset.iloc[2, :-1].values # First row (excluding Outcome)

vector2 = dataset.iloc[3, :-1].values # Second row (excluding Outcome)

r\_values = np.arange(1, 11)

minkowski\_distances = [np.sum(np.abs(vector1- vector2) \*\* r) \*\* (1/r) for r in

r\_values]

plt.figure(figsize=(8, 5))

plt.plot(r\_values, minkowski\_distances, marker='o', linestyle='-')

plt.xlabel("r (Minkowski Parameter)")

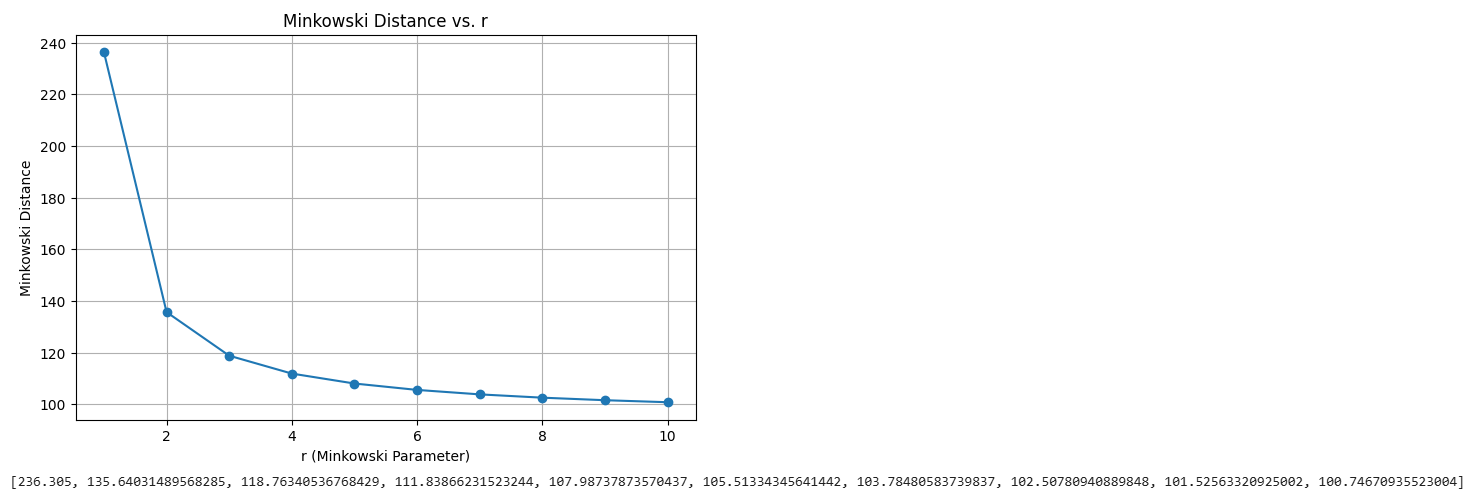
plt.ylabel("Minkowski Distance")

plt.title("Minkowski Distance vs. r")

plt.grid(True)

plt.show()

print(minkowski\_distances)

**OUTPUT:**

#4

X = dataset.drop(columns=['Outcome'])

y = dataset['Outcome'] # Target variable

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

random\_state=42)

print("Training features shape:", X\_train.shape)

print("Test features shape:", X\_test.shape)

print("Training target shape:", y\_train.shape)

print("Test target shape:", y\_test.shape)

**OUTPUT:**

**Training features shape: (537, 8)**

**Test features shape: (231, 8)**

**Training target shape: (537,)**

**Test target shape: (231,)**

#5

knn\_classifier = KNeighborsClassifier(n\_neighbors=3)

knn\_classifier.fit(X\_train, y\_train)

train\_accuracy = knn\_classifier.score(X\_train, y\_train)

print(f"Training Accuracy: {train\_accuracy}")

**OUTPUT:**

Training Accuracy: 0.8435754189944135

#6

test\_accuracy = knn\_classifier.score(X\_test, y\_test)

print(f"Test Accuracy: {test\_accuracy}")

**OUTPUT:**

Test Accuracy: 0.6753246753246753

#7

predictions = knn\_classifier.predict(X\_test)

print("Predictions for the test set:")

print(predictions)

test\_vector = X\_test.iloc[0].values.reshape(1,-1)

predicted\_class = knn\_classifier.predict(test\_vector)

print(f"Predicted class for the given test vector: {predicted\_class[0]}")

**OUTPUT:**

Predictions for the test set:

[0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 0 1 1

0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0

0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 1 1 0 1 0 1 0

1 0 1 1 1 1 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 0 0 0 0 1 0 0 0 0

1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0

0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 0

0 0 0 1 0 0 0 0 0]

Predicted class for the given test vector: 0

#8

def evaluate\_knn\_accuracy(X\_train, y\_train, X\_test, y\_test):

k\_values = np.arange(1, 12)

accuracies = []

for k in k\_values:

knn\_classifier = KNeighborsClassifier(n\_neighbors=k)

knn\_classifier.fit(X\_train, y\_train)

accuracy = knn\_classifier.score(X\_test, y\_test)

accuracies.append(accuracy)

return k\_values, accuracies

k\_values, accuracies = evaluate\_knn\_accuracy(X\_train, y\_train, X\_test, y\_test)

plt.figure(figsize=(8, 5))

plt.plot(k\_values, accuracies, marker='o', linestyle='-', color='b')

plt.xlabel('k (Number of Neighbors)')

plt.ylabel('Accuracy')

plt.title('kNN Accuracy vs. k (1 to 11)')

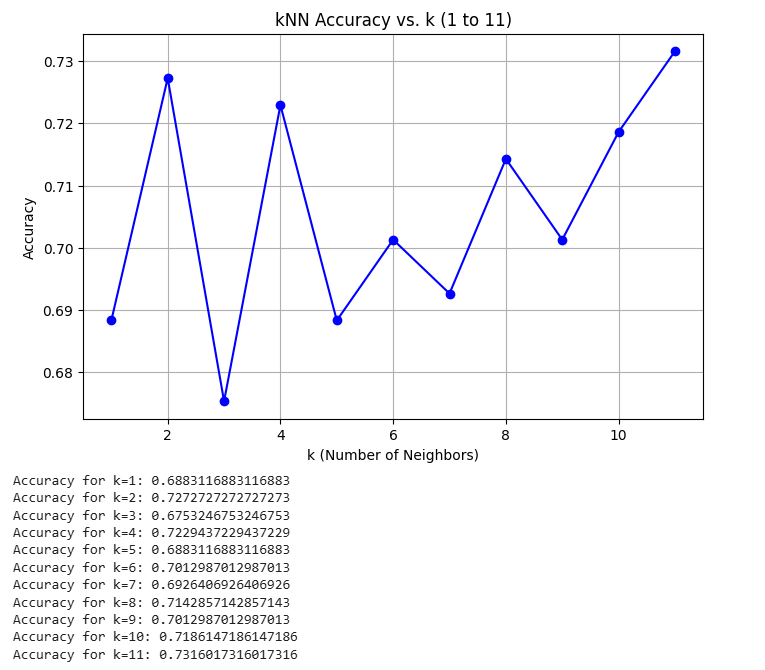
plt.grid(True)

plt.show()

for k, accuracy in zip(k\_values, accuracies):

print(f"Accuracy for k={k}: {accuracy}")

**OUTPUT:**



#9

y\_train\_pred = knn\_classifier.predict(X\_train)

y\_test\_pred = knn\_classifier.predict(X\_test)

train\_conf\_matrix = confusion\_matrix(y\_train, y\_train\_pred)

print("Confusion Matrix (Training Set):")

print(train\_conf\_matrix)

test\_conf\_matrix = confusion\_matrix(y\_test, y\_test\_pred)

print("Confusion Matrix (Test Set):")

print(test\_conf\_matrix)

train\_precision = precision\_score(y\_train, y\_train\_pred)

train\_recall = recall\_score(y\_train, y\_train\_pred)

train\_f1 = f1\_score(y\_train, y\_train\_pred)

test\_precision = precision\_score(y\_test, y\_test\_pred)

test\_recall = recall\_score(y\_test, y\_test\_pred)

test\_f1 = f1\_score(y\_test, y\_test\_pred)

print("\nPerformance Metrics (Training Set):")

print(f"Precision: {train\_precision}")

print(f"Recall: {train\_recall}")

print(f"F1-Score: {train\_f1}")

print("\nPerformance Metrics (Test Set):")

print(f"Precision: {test\_precision}")

print(f"Recall: {test\_recall}")

print(f"F1-Score: {test\_f1}")

if train\_f1 > test\_f1:

print("\n The dataset is overfitting.")

elif train\_f1 < test\_f1:

print("\n The dataset is underfitting.")

else:

print("\n The dataset is regular fit.")

**OUTPUT:**

Confusion Matrix (Training Set):

[[316 33]

[ 51 137]]

Confusion Matrix (Test Set):

[[108 43]

[ 32 48]]

Performance Metrics (Training Set):

Precision: 0.8058823529411765

Recall: 0.7287234042553191

F1-Score: 0.7653631284916201

Performance Metrics (Test Set):

Precision: 0.5274725274725275

Recall: 0.6

F1-Score: 0.5614035087719298

The dataset is overfitting.