2. k-means clustering

import pandas as pd

from sklearn.cluster import KMeans

data = pd.read\_csv('/content/data.csv')

X = data[['Variable 1', 'Variable 2']]

kmeans = KMeans(n\_clusters=2, random\_state=1)

data['Cluster'] = kmeans.fit\_predict(X)

print(data)

7. k nearest neighbour

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

data = pd.read\_csv('/content/data1.csv')

X = data[['Height', 'Weight']].values

y = data['Size'].values

k = 3

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X, y)

new\_customer = np.array([[172, 68]])

predicted\_size = knn.predict(new\_customer)

print(f"Predicted T-shirt size for the new customer: {predicted\_size[0]}")

3. linear regression

x = [0, 1, 2, 3, 4]

y = [2, 3, 5, 4, 6]

N = len(x)

sum\_x = sum(x)

sum\_y = sum(y)

sum\_x\_squared = sum([xi \*\* 2 for xi in x])

sum\_xy = sum([x[i] \* y[i] for i in range(N)])

a = (N \* sum\_xy - sum\_x \* sum\_y) / (N \* sum\_x\_squared - sum\_x \*\* 2)

b = (sum\_y - a \* sum\_x) / N

print(f"The regression line is: y = {a:.2f}x + {b:.2f}")

x\_new = 10

y\_estimated = a \* x\_new + b

print(f"Estimated value of y when x = 10: {y\_estimatejd:.2f}")

y\_pred = [a \* xi + b for xi in x]

mse = sum([(y\_pred[i] - y[i]) \*\* 2 for i in range(N)]) / N

print(f"Mean Squared Error: {mse:.2f}")

-----------------------------------------------------------------------------------

4.find s

import pandas as pd

data = pd.read\_csv('/content/sfinds.csv')

positive\_examples = data[data['EnjoySport'] == 'Yes']

hypothesis = positive\_examples.iloc[0, :-1].values

for i, row in positive\_examples.iterrows():

    for j in range(len(hypothesis)):

        if hypothesis[j] != row[j]:

            hypothesis[j] = '?'

print("Maximally Specific Hypothesis:", hypothesis)

5.GMM

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.mixture import GaussianMixture

iris = load\_iris()

X = iris.data

gmm = GaussianMixture(n\_components=3, random\_state=0)

gmm.fit(X)

labels = gmm.predict(X)

colors = ['red', 'green', 'yellow']

plt.figure(figsize=(8, 6))

for i, color in enumerate(colors):

plt.scatter(X[labels == i, 0], X[labels == i, 1], color=color, label=f'Cluster {i+1}')

plt.xlabel(iris.feature\_names[0])

plt.ylabel(iris.feature\_names[1])

plt.title("Gaussian Mixture Model Clustering on Iris Dataset")

plt.legend()

plt.show()

6.SVM

import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import SVC

X = np.array([

[1, 2],

[5, 8],

[1.5, 1.8],

[8, 8],

[1, 0.6],

[9, 11],

[7, 10],

[8.7, 9.4],

[2.3, 4],

[5.5, 3],

[7.7, 8.8],

[6.1, 7.5]

])

y = np.array([0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1])

model = SVC(kernel='linear')

model.fit(X, y)

w = model.coef\_[0]

b = model.intercept\_[0]

slope = -w[0] / w[1]

intercept = -b / w[1]

x\_vals = np.linspace(min(X[:, 0]), max(X[:, 0]), 100)

y\_vals = slope \* x\_vals + intercept

margin = 1 / np.sqrt(np.sum(w\*\*2))

y\_margin\_up = y\_vals + margin

y\_margin\_down = y\_vals - margin

plt.figure(figsize=(8, 6))

plt.scatter(X[:, 0], X[:, 1], c=y, cmap='coolwarm', s=50)

plt.plot(x\_vals, y\_vals, 'k-', label="Optimal Hyperplane")

plt.plot(x\_vals, y\_margin\_up, 'g--', label="Margin")

plt.plot(x\_vals, y\_margin\_down, 'g--')

plt.scatter(model.support\_vectors\_[:, 0], model.support\_vectors\_[:, 1],

s=100, facecolors='none', edgecolors='k', label="Support Vectors")

plt.xlabel("X")

plt.ylabel("Y")

plt.legend()

plt.title("SVM with Optimal Hyperplane and Margins")

plt.show()

1.DECISION TREE

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

data = pd.read\_csv('weather\_data.csv')

label\_encoder = LabelEncoder()

for column in data.columns:

data[column] = label\_encoder.fit\_transform(data[column])

X = data.drop('PlayTennis', axis=1)

y = data['PlayTennis']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy \* 100:.2f}%")