1.import numpy as np

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

from sklearn.tree import DecisionTreeRegressor, export\_text

from sklearn.metrics import mean\_squared\_error

np.random.seed(42)

n\_students = 100

math\_marks = np.random.randint(50, 100, n\_students)

english\_marks = np.random.randint(50, 100, n\_students)

science\_marks = np.random.randint(50, 100, n\_students)

social\_studies\_marks = np.random.randint(50, 100, n\_students)

# Calculating total and average marks

total\_marks = math\_marks + english\_marks + science\_marks + social\_studies\_marks

average\_marks = total\_marks / 4

X = np.column\_stack((math\_marks, english\_marks, science\_marks, social\_studies\_marks))

y\_total = total\_marks

y\_avg = average\_marks

X\_train, X\_test, y\_train\_total, y\_test\_total = train\_test\_split(X, y\_total, test\_size=0.2, random\_state=42)

\_, \_, y\_train\_avg, y\_test\_avg = train\_test\_split(X, y\_avg, test\_size=0.2, random\_state=42)

total\_model = DecisionTreeRegressor(random\_state=42)

total\_model.fit(X\_train, y\_train\_total)

avg\_model = DecisionTreeRegressor(random\_state=42)

avg\_model.fit(X\_train, y\_train\_avg)

y\_pred\_total = total\_model.predict(X\_test)

y\_pred\_avg = avg\_model.predict(X\_test)

total\_mse = mean\_squared\_error(y\_test\_total, y\_pred\_total)

avg\_mse = mean\_squared\_error(y\_test\_avg, y\_pred\_avg)

print(f"Mean Squared Error for Total marks prediction: {total\_mse}")

print(f"Mean Squared Error for Average marks prediction: {avg\_mse}")

tree\_text = export\_text(total\_model, feature\_names=['Math', 'English', 'Science', 'Social\_Studies'])

print("Decision Tree for Total Marks Prediction:")

print(tree\_text)

2. import numpy as np

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report

np.random.seed(42) # For reproducibility

n\_samples = 1000

income = np.random.randint(30000, 100000, n\_samples)

age = np.random.randint(18, 70, n\_samples)

loan\_amount = np.random.randint(1000, 50000, n\_samples)

credit\_score = np.random.choice(['Poor', 'Fair', 'Good', 'Excellent'], n\_samples, p=[0.2, 0.3, 0.3, 0.2])

label\_map = {'Poor': 0, 'Fair': 1, 'Good': 2, 'Excellent': 3}

credit\_score = np.vectorize(label\_map.get)(credit\_score)

X = np.column\_stack((income, age, loan\_amount))

y = credit\_score

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

y\_pred = nb\_model.predict(X\_test)

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

report = classification\_report(y\_test, y\_pred, target\_names=['Poor', 'Fair', 'Good', 'Excellent'], zero\_division=0)

print(f"Accuracy: {accuracy}")

print("Classification Report:")

print(report)

3. import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import pandas as pd

iris = load\_iris()

X = iris.data

y = iris.target

class\_names = iris.target\_names

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

k = 5

knn = KNeighborsClassifier(n\_neighbors=k)

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

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

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

report = classification\_report(y\_test, y\_pred, target\_names=class\_names)

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

print(f"Accuracy: {accuracy}")

print("\nClassification Report:")

print(report)

conf\_matrix\_df = pd.DataFrame(conf\_matrix, index=class\_names, columns=class\_names)

print("\nConfusion Matrix:")

print(conf\_matrix\_df)

4.

input\_neurons = 5

hidden\_neurons = 10

output\_neurons = 3

weights\_hidden\_layer = input\_neurons \* hidden\_neurons

biases\_hidden\_layer = hidden\_neurons

total\_hidden\_layer\_params = weights\_hidden\_layer + biases\_hidden\_layer

weights\_output\_layer = hidden\_neurons \* output\_neurons

biases\_output\_layer = output\_neurons

total\_output\_layer\_params = weights\_output\_layer + biases\_output\_layer

total\_trainable\_params = total\_hidden\_layer\_params + total\_output\_layer\_params

print(f"Total number of trainable parameters: {total\_trainable\_params}")

5. import numpy as np

P\_A = 1 / 5

P\_not\_A = 1 - P\_A

P\_B\_given\_A = 1 / 5

P\_B\_given\_not\_A = 4 / 5

P\_C\_given\_B = 1 / 4

P\_C\_given\_not\_B = 3 / 4

P\_D\_given\_B = 1 / 2

P\_D\_given\_not\_B = 1 / 2

samples = [

{'A': None, 'B': None, 'C': None, 'D': 'd'}, # s1

{'A': 'a', 'B': 'b', 'C': 'c', 'D': 'd'}, # s2

{'A': 'a', 'B': 'b', 'C': 'c', 'D': 'd'}, # s3

{'A': 'neg\_a', 'B': 'b', 'C': 'neg\_c', 'D': 'd'} # s4

]

evidence = {'A': 'neg\_a', 'C': 'neg\_c'}

def likelihood\_weight(sample, evidence):

weight = 1.0

if evidence['A'] == 'neg\_a':

weight \*= P\_not\_A

else:

weight \*= P\_A

if evidence['C'] == 'neg\_c':

if sample['B'] == 'b':

weight \*= 1 - P\_C\_given\_B

else:

weight \*= 1 - P\_C\_given\_not\_B

else:

if sample['B'] == 'b':

weight \*= P\_C\_given\_B

else:

weight \*= P\_C\_given\_not\_B

return weight

weights = []

for sample in samples:

if sample['A'] == evidence['A'] and sample['C'] == evidence['C']:

weight = likelihood\_weight(sample, evidence)

weights.append((sample['B'], weight))

total\_weight = sum(weight for \_, weight in weights)

P\_B\_given\_evidence = {}

for B\_value, weight in weights:

if B\_value not in P\_B\_given\_evidence:

P\_B\_given\_evidence[B\_value] = 0

P\_B\_given\_evidence[B\_value] += weight / total\_weight

print(f"Estimated probabilities for P(B | neg A, neg C): {P\_B\_given\_evidence}")