Experiment-15:

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_breast\_cancer

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import seaborn as sns

# Load data

data = load\_breast\_cancer()

df = pd.DataFrame(data.data, columns=data.feature\_names)

# Standardize

scaled = StandardScaler().fit\_transform(df)

# PCA with 2 components

pca = PCA(n\_components=2)

pca\_result = pca.fit\_transform(scaled)

# Scatter plot

plt.scatter(pca\_result[:, 0], pca\_result[:, 1], c=data.target, cmap='plasma')

plt.xlabel('PC1')

plt.ylabel('PC2')

plt.title('PCA - Breast Cancer')

plt.show()

# Heatmap of PCA components

sns.heatmap(pd.DataFrame(pca.components\_, columns=data.feature\_names), cmap='plasma')

plt.title('PCA Components')

plt.show()

Experiment-13:

import pandas as pd

import numpy as np

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.inference import VariableElimination

# Load and clean dataset

data = pd.read\_csv('heart.csv').replace('?', np.nan)

# Define model structure

model = BayesianModel([

('age', 'heartdisease'),

('sex', 'heartdisease'),

('exang', 'heartdisease'),

('cp', 'heartdisease'),

('heartdisease', 'restecg'),

('heartdisease', 'chol')

])

# Train model using MLE

model.fit(data, estimator=MaximumLikelihoodEstimator)

# Inference

infer = VariableElimination(model)

# Queries

print("P(HeartDisease | restecg=1)")

print(infer.query(['heartdisease'], evidence={'restecg': 1}))

print("\nP(HeartDisease | cp=2)")

print(infer.query(['heartdisease'], evidence={'cp': 2}))

Experiment-11:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

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

# Load dataset

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

data = pd.read\_csv("8-dataset.csv", names=names)

# Input features and labels

X = data.iloc[:, :-1]

labels = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [labels[i] for i in data['Class']]

# Color map

colors = np.array(['red', 'lime', 'black'])

# Plot Real Labels

plt.figure(figsize=(14, 5))

plt.subplot(1, 3, 1)

plt.title("Actual")

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colors[y])

# KMeans

kmeans = KMeans(n\_clusters=3, random\_state=0)

kmeans\_labels = kmeans.fit\_predict(X)

plt.subplot(1, 3, 2)

plt.title("KMeans")

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colors[kmeans\_labels])

print("KMeans Accuracy:", accuracy\_score(y, kmeans\_labels))

print("KMeans Confusion:\n", confusion\_matrix(y, kmeans\_labels))

# Gaussian Mixture

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

gmm\_labels = gmm.fit\_predict(X)

plt.subplot(1, 3, 3)

plt.title("GMM")

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colors[gmm\_labels])

plt.tight\_layout()

plt.show()

print("GMM Accuracy:", accuracy\_score(y, gmm\_labels))

print("GMM Confusion:\n", confusion\_matrix(y, gmm\_labels))

Experiment: 9

import numpy as np

import matplotlib.pyplot as plt

from moepy import lowess

# Generate data

x = np.linspace(0, 5, 150)

y = np.sin(x) + np.random.normal(0, 0.1, size=len(x))

# Fit LOWESS model

model = lowess.Lowess()

model.fit(x, y)

# Predict

x\_pred = np.linspace(0, 5, 26)

y\_pred = model.predict(x\_pred)

# Plot

plt.scatter(x, y, color='magenta', s=10, label='Noisy Data')

plt.plot(x\_pred, y\_pred, 'r--', label='LOWESS Fit')

plt.legend()

plt.show()

Experiment: 7

import numpy as np

# Input and output

X = np.array([[2, 9], [1, 5], [3, 6]], dtype=float)

y = np.array([[92], [86], [89]], dtype=float)

X = X / np.max(X, axis=0)

y = y / 100

# Activation functions

def sigmoid(x): return 1 / (1 + np.exp(-x))

def d\_sigmoid(x): return x \* (1 - x)

# Parameters

epoch = 7000

lr = 0.1

input\_neurons = 2

hidden\_neurons = 3

output\_neurons = 1

# Weights and biases

wh = np.random.rand(input\_neurons, hidden\_neurons)

bh = np.random.rand(1, hidden\_neurons)

wout = np.random.rand(hidden\_neurons, output\_neurons)

bout = np.random.rand(1, output\_neurons)

# Training

for \_ in range(epoch):

# Forward

hidden\_input = np.dot(X, wh) + bh

hidden\_output = sigmoid(hidden\_input)

final\_input = np.dot(hidden\_output, wout) + bout

output = sigmoid(final\_input)

# Backward

error = y - output

d\_output = error \* d\_sigmoid(output)

error\_hidden = d\_output.dot(wout.T)

d\_hidden = error\_hidden \* d\_sigmoid(hidden\_output)

# Update weights

wout += hidden\_output.T.dot(d\_output) \* lr

bout += np.sum(d\_output, axis=0, keepdims=True) \* lr

wh += X.T.dot(d\_hidden) \* lr

bh += np.sum(d\_hidden, axis=0, keepdims=True) \* lr

# Results

print("Input:\n", X)

print("Actual:\n", y)

print("Predicted:\n", output)

Experiment: 5

import pandas as pd

from sklearn.datasets import load\_boston

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import cross\_val\_score, train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Load dataset

data = load\_boston()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['TARGET'] = data.target

# Remove duplicates

df = df.drop\_duplicates()

# Split data

X = df.drop('TARGET', axis=1)

y = df['TARGET']

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

# Train model

model = LinearRegression()

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

# Bias: Training error

train\_pred = model.predict(X\_train)

train\_mse = mean\_squared\_error(y\_train, train\_pred)

# Variance: Testing error

test\_pred = model.predict(X\_test)

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

# Cross-validation

cv\_scores = cross\_val\_score(model, X, y, cv=5)

print("Training Error (Bias):", train\_mse)

print("Testing Error (Variance):", test\_mse)

print("Cross-Validation Scores:", cv\_scores)

print("Average CV Score:", cv\_scores.mean())

Experiment: 3

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import confusion\_matrix

# Load data

data = load\_iris()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['target'] = data.target

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[data.feature\_names], df['target'], random\_state=0)

# Train model

clf = DecisionTreeClassifier(max\_depth=2, random\_state=0)

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

# Predict and evaluate

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

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

# Plot tree

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

plot\_tree(clf, feature\_names=data.feature\_names, class\_names=data.target\_names, filled=True)

plt.show()  
  
  
Experiment: 1

import pandas as pd

import numpy as np

data = pd.read\_csv("ws.csv")

d = np.array(data)[:,:-1]

target = np.array(data)[:,-1]

def train(c, t):

for i in range(len(t)):

if t[i] == "Yes":

h = c[i].copy()

break

for i in range(len(t)):

if t[i] == "Yes":

for j in range(len(h)):

if c[i][j] != h[j]:

h[j] = '?'

return h

print("Final Hypothesis:", train(d, target))