**Experiment 2**

mysql -u root -p

show databases;

create database smart\_farming;

CREATE TABLE CropID (

CropID INT PRIMARY KEY,

CropName VARCHAR(50),

CropType VARCHAR(50),

Season VARCHAR(50)

);

CREATE TABLE DataID (

DataID DATE PRIMARY KEY,

Day INT,

Month INT,

Quarter INT,

Year INT

);

CREATE TABLE DateID (

DateID DATE PRIMARY KEY,

Day INT,

Month INT,

Quarter INT,

Year INT

);

CREATE TABLE SensorID (

SensorID INT PRIMARY KEY,

InViz BOOLEAN,

AverageTemperature DECIMAL(5,2),

HumidityPercentage DECIMAL(5,2),

SoilMoistureLevel DECIMAL(5,2),

SunlightHours DECIMAL(4,2)

);

show tables;

CREATE TABLE FarmID (

FarmID INT PRIMARY KEY,

FarmName VARCHAR(100),

Location VARCHAR(100),

Area DECIMAL(6,2),

Hector VARCHAR(10),

OwnerName VARCHAR(100)

);

INSERT INTO CropID (CropID, CropName, CropType, Season)

VALUES

(1, 'Wheat', 'Cereal', 'Rabi'),

(2, 'Rice', 'Cereal', 'Kharif'),

(3, 'Sugarcane', 'Commercial', 'Annual'),

(4, 'Potato', 'Vegetable', 'Rabi');

INSERT INTO DateID (DateID, Day, Month, Quarter, Year)

VALUES

('2025-07-01', 1, 7, 3, 2025),

('2025-08-15', 15, 8, 3, 2025),

('2025-11-10', 10, 11, 4, 2025),

('2025-12-25', 25, 12, 4, 2025);

INSERT INTO SensorID (SensorID, InViz, AverageTemperature, HumidityPercentage, SoilMoistureLevel, SunlightHours)

VALUES

(101, TRUE, 29.55, 68.40, 30.25, 6.50),

(102, FALSE, 31.20, 72.10, 28.00, 7.25),

(103, TRUE, 26.80, 65.50, 32.10, 5.90),

(104, FALSE, 30.00, 70.00, 29.50, 6.80);

INSERT INTO FarmID (FarmID, FarmName, Location, Area, Hector, OwnerName)

VALUES

(1, 'Green Acres', 'Pune, Maharashtra', 12.50, '12ha', 'Ramesh Patil'),

(2, 'Sunny Fields', 'Nashik, Maharashtra', 20.75, '21ha', 'Sita Deshmukh'),

(3, 'Eco Farm', 'Indore, MP', 15.30, '15ha', 'Amit Singh'),

(4, 'Fresh Land', 'Ludhiana, Punjab', 18.00, '18ha', 'Gurpreet Kaur');

SELECT \* FROM CropID;

SELECT \* FROM DateID;

SELECT \* FROM SensorID;

SELECT \* FROM FarmID;

CREATE TABLE CropYieldFact (

FactID INT PRIMARY KEY AUTO\_INCREMENT, -- Unique identifier for each record

SpaInVisFarmId INT, -- Foreign Key to FarmID

CropID INT, -- Foreign Key to CropID

DateID DATE, -- Foreign Key to DateID (date)

SensorID INT, -- Foreign Key to SensorID

YieldKg DECIMAL(10,2), -- Yield in kilograms

WaterLitre DECIMAL(10,2), -- Water used in litres

-- Foreign Key Constraints

FOREIGN KEY (SpaInVisFarmId) REFERENCES FarmID(FarmID),

FOREIGN KEY (CropID) REFERENCES CropID(CropID),

FOREIGN KEY (DateID) REFERENCES DateID(DateID),

FOREIGN KEY (SensorID) REFERENCES SensorID(SensorID)

);

INSERT INTO CropYieldFact (SpaInVisFarmId, CropID, DateID, SensorID, YieldKg, WaterLitre)

VALUES

(1, 1, '2025-07-01', 101, 1500.50, 3200.75),

(2, 2, '2025-08-15', 102, 1800.00, 4000.00),

(3, 3, '2025-11-10', 103, 2100.25, 3700.50),

(4, 4, '2025-12-25', 104, 1600.75, 2900.30);

SELECT \* FROM CropYieldFact;

exit

**Experiment 3**

**1. Roll-up**

SELECT

d.Year,

SUM(cyf.YieldKg) AS TotalYieldKg,

SUM(cyf.WaterLitre) AS TotalWaterLitre

FROM CropYieldFact AS cyf

JOIN DateID AS d

ON cyf.DateID = d.DateID

GROUP BY d.Year

ORDER BY d.Year;

**2. Drill-down**

SELECT

d.Year,

d.Month,

SUM(cyf.YieldKg) AS TotalYieldKg,

SUM(cyf.WaterLitre) AS TotalWaterLitre

FROM CropYieldFact AS cyf

JOIN DateID AS d

ON cyf.DateID = d.DateID

GROUP BY d.Year, d.Month

ORDER BY d.Year, d.Month;

**3. Slice**

SELECT

cyf.FactID,

farm.FarmName,

d.DateID,

cyf.YieldKg,

cyf.WaterLitre

FROM CropYieldFact AS cyf

JOIN FarmID AS farm

ON cyf.SpaInVisFarmId = farm.FarmID

JOIN DateID AS d

ON cyf.DateID = d.DateID

WHERE cyf.CropID = 1;

**5. Pivot**

SELECT

crop.CropName,

SUM(CASE WHEN d.Year = 2023 THEN cyf.YieldKg ELSE 0 END) AS Yield\_2023,

SUM(CASE WHEN d.Year = 2024 THEN cyf.YieldKg ELSE 0 END) AS Yield\_2024,

SUM(CASE WHEN d.Year = 2025 THEN cyf.YieldKg ELSE 0 END) AS Yield\_2025

FROM CropYieldFact AS cyf

JOIN CropID AS crop

ON cyf.CropID = crop.CropID

JOIN DateID AS d

ON cyf.DateID = d.DateID

GROUP BY crop.CropName

ORDER BY crop.CropName;

**Experiment 8: One Classifier in java/python**

Program:

dataset = [

[0, 0, 1, 0, 0],

[0, 0, 1, 1, 0],

[1, 0, 0, 1, 1],

[1, 0, 1, 0, 1],

[2, 2, 0, 0, 1],

[2, 2, 0, 1, 0],

[2, 1, 0, 1, 1],

[0, 2, 1, 0, 0],

[0, 2, 1, 1, 1],

[2, 2, 1, 0, 1],

[0, 0, 0, 0, 0],

[0, 1, 0, 0, 1],

[1, 0, 0, 0, 1],

[1, 0, 0, 1, 0],

[2, 1, 1, 1, 0]

]

mp = dict()

for i in range(len(dataset)):

row = dataset[i]

y = row[-1]

if y not in mp:

mp[y] = list()

mp[y].append(row)

for label in mp:

print(label)

for row in mp[label]:

print(row)

test = [2, 1, 0, 1]

probYes = 1

countYes = 0

totalYes = 0

for row in dataset:

if row[-1] == 1:

countYes += 1

totalYes += 1

print("Total yes: {}/{}".format(countYes, totalYes))

probYes \*= countYes / totalYes

for i in range(len(test)):

countYes = 0

totalYes = 0

for row in mp[1]:

if test[i] == row[i]:

countYes += 1

totalYes += 1

print("for feature {}: {}/{}".format(i + 1, countYes, totalYes))

probYes \*= countYes / totalYes

probNo = 1

countNo = 0

totalNo = 0

for row in dataset:

if row[-1] == 0:

countNo += 1

totalNo += 1

print("Total no: {}/{}".format(countNo, totalNo))

probNo \*= countNo / totalNo

for i in range(len(test)):

countNo = 0

totalNo = 0

for row in mp[0]:

if test[i] == row[i]:

countNo += 1

totalNo += 1

print("for feature {}: {}/{}".format(i + 1, countNo, totalNo))

probNo \*= countNo / totalNo

print("Probability of playing golf: {:.2f}%".format(probYes / (probYes + probNo) \* 100))

**Experiment 9: One Clustering Algorithm in java/python**

data = [

[5, 2], [2, 4], [9, 5], [3, 6], [3, 1], [5, 5], [1, 5], [6, 7], [4, 2], [2, 7],

[9, 2], [1, 5], [4, 6], [3, 6], [1, 7], [8, 4], [8, 7], [2, 2], [7, 2], [2, 1],

[2, 4], [1, 2], [1, 4], [2, 6], [7, 7], [2, 4], [3, 4], [1, 4]

]

X = [i[0] for i in data]

y = [i[1] for i in data]

import matplotlib.pyplot as plt

plt.scatter(X, y)

plt.show()

import math

def dist(center, point):

d = 0.0

for i in range(0, len(point)):

d += (center[i] - point[i]) \*\* 2

return math.sqrt(d)

def assignCenters(centers, dataset):

clusters = []

for i in range(len(dataset)):

distances = []

for center in centers:

distances.append(dist(center, dataset[i]))

temp = [z for z, val in enumerate(distances) if val == min(distances)]

clusters.append(temp[0])

return clusters

def mean\_center(k, dataset, clusters):

nCenters = []

for i in range(k):

x = 0.0

y = 0.0

count = 0

for j in range(len(clusters)):

if i

== clusters[j]:

x += dataset[j][0]

y += dataset[j][1]

count += 1

x = x / count

y = y / count

nCenters.append([x, y])

return nCenters

print("Enter k")

k = int(input())

centers = []

for i in range(k):

print("Enter center " + str(i))

temp = [int(x) for x in input().split()]

centers.append(temp)

print("Initial centers:")

print(centers)

print("Initial clusters:")

clusters = assignCenters(centers, data)

for i in range(k):

print("Cluster " + str(i))

for j in range(len(clusters)):

if i == clusters[j]:

print(data[j], end="")

print()

for itr in range(10):

print("Iteration " + str(itr))

centers = mean\_center(k, data, clusters)

print("Updated centers:")

print(centers)

clusters = assignCenters(centers, data)

print("Updated clusters:")

for i in range(k):

print("Cluster " + str(i))

for j in range(len(clusters)):

if i

== clusters[j]:

print(data[j], end="")

print()

**Experiment 10: One Clustering Algorithm in java/python**

Code:

import pandas as pd

from mlxtend.frequent\_patterns import apriori, association\_rules

# Sample transaction dataset

dataset = [

['milk', 'bread', 'butter'],

['bread', 'butter'],

['milk', 'bread'],

['milk', 'butter'],

['bread', 'butter', 'jam'],

['milk', 'bread', 'butter', 'jam'],

]

# Convert the dataset into a DataFrame manually (one-hot encoding)

all\_items = set(item for transaction in dataset for item in transaction)

one\_hot\_encoded\_data = []

for transaction in dataset:

transaction\_encoded = {item: (item in transaction) for item in all\_items}

one\_hot\_encoded\_data.append(transaction\_encoded)

df = pd.DataFrame(one\_hot\_encoded\_data)

# Apply Apriori to find frequent itemsets

frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)

# Generate association rules from frequent itemsets

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.2)

# Display results

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules)