**Slip 15**

**Write a python program to Implement Naïve Bayes.**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

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

df = pd.read\_csv('iris.csv')

print("Dataset:")

print(df.head())

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

y = df['Species']

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

model = GaussianNB()

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

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

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

print('\nAccuracy:', accuracy)

print('\nClassification Report:')

print(classification\_report(y\_test, y\_pred))

print('\nConfusion Matrix:')

print(confusion\_matrix(y\_test, y\_pred))

**Q2.Write a python program to implement linear SVM.**

import pandas as pd

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

from sklearn.svm import SVC

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

df = pd.read\_csv('heart.csv')

print("Dataset:")

print(df.head())

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)

model = SVC(kernel='linear')

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

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

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

print('\nAccuracy:', accuracy)

print('\nClassification Report:')

print(classification\_report(y\_test, y\_pred))

print('\nConfusion Matrix:')

print(confusion\_matrix(y\_test, y\_pred))

**Slip 16**

**Write a Python program to implement to find all null values in a given Data set and remove them.(Use air quality dataset.)**

import pandas as pd

# Load the air quality dataset

df = pd.read\_csv('air\_quality.csv')

# Display the shape of the dataset before removing null values

print("Shape of dataset before removing null values:", df.shape)

# Display the count of null values in each column

print("\nCount of null values in each column:")

print(df.isnull().sum())

# Remove rows with any null values

df\_cleaned = df.dropna()

# Display the shape of the dataset after removing null values

print("\nShape of dataset after removing null values:", df\_cleaned.shape)

# Save the cleaned dataset to a new CSV file

df\_cleaned.to\_csv('air\_quality\_cleaned.csv', index=False)

print("\nCleaned dataset saved to air\_quality\_cleaned.csv")

**Q.2 Write a python program to implement Multiple Linear Regression for given dataset. (Use Position Salaries Dataset)**

import pandas as pd

from sklearn.linear\_model import LinearRegression

import numpy as np

# Load the dataset

df = pd.read\_csv('Position\_Salaries.csv')

# Display the dataset

print("Dataset:")

print(df)

# Split the dataset into features (X) and target variable (y)

X = df.iloc[:, 1:-1].values # Independent variables (features)

y = df.iloc[:, -1].values # Dependent variable (target)

# Train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Predict the salary for a new employee with a given level

new\_level = 6.5

predicted\_salary = model.predict([[new\_level]])

print(f"\nPredicted Salary for Level {new\_level}: ${predicted\_salary[0]:,.2f}")

# Coefficients and Intercept

print("\nCoefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

**Slip 17**

**Q.1 Write a python program to Implement Decision Tree whether or not to play tennis.**

from sklearn.tree import DecisionTreeClassifier

from sklearn import preprocessing

# Create dataset

weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny',

'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy']

temp = ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild']

humidity = ['High', 'High', 'High', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'High', 'Normal', 'High']

windy = ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong']

play\_tennis = ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

# Encode categorical variables

le = preprocessing.LabelEncoder()

weather\_encoded = le.fit\_transform(weather)

temp\_encoded = le.fit\_transform(temp)

humidity\_encoded = le.fit\_transform(humidity)

windy\_encoded = le.fit\_transform(windy)

play\_tennis\_encoded = le.fit\_transform(play\_tennis)

# Merge features

features = list(zip(weather\_encoded, temp\_encoded, humidity\_encoded, windy\_encoded))

# Create Decision Tree classifier

clf = DecisionTreeClassifier()

# Train the classifier

clf = clf.fit(features, play\_tennis\_encoded)

# Predict

predicted = clf.predict([[0, 2, 0, 1]]) # Example: Sunny, Mild, High, Strong

# Decode the predicted label

predicted\_label = le.inverse\_transform(predicted)

print("Predicted Label:", predicted\_label[0])

**Q.2 Write a python Program to transform data with Principal Component Analysis (PCA)**

import pandas as pd

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

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

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

y = data['Species']

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

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

targets = data['Species'].unique()

colors = ['r', 'g', 'b']

for target, color in zip(targets, colors):

indicesToKeep = y == target

plt.scatter(X\_pca[indicesToKeep, 0], X\_pca[indicesToKeep, 1], c=color, label=target)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('2 Component PCA')

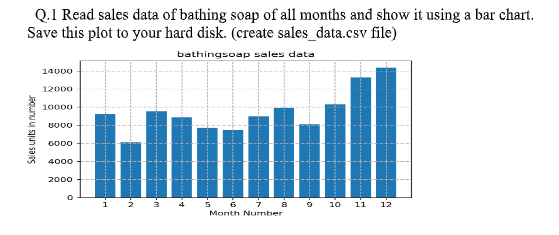
plt.legend(targets)

plt.grid()

plt.show()

**Slip 18**

**Q.1 Read sales data of bathing soap of all months and show it using a bar chart. Save this plot to your hard disk. (create sales\_data.csv file)**



import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Create sales\_data.csv file (you can replace the data with your actual sales data)

data = {

'Month Number': list(range(1, 13)),

'Sale Units in Number': [2000, 4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, 20000, 22000, 24000]

}

df = pd.DataFrame(data)

df.to\_csv('sales\_data.csv', index=False)

# Step 2: Read data from sales\_data.csv

df = pd.read\_csv('sales\_data.csv')

# Step 3: Plot the data

plt.figure(figsize=(10, 6))

plt.bar(df['Month Number'], df['Sale Units in Number'], color='skyblue')

plt.title('Bathing Soap Sales Data')

plt.xlabel('Month Number')

plt.ylabel('Sale Units in Number')

plt.grid(True)

# Step 4: Save the plot to hard disk

plt.savefig('bathing\_soap\_sales.png')

# Display the plot

plt.show()

**Q.2 Write a python program to implement Multiple Linear Regression for given dataset. (Create Own Dataset)**

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import numpy as np

# Generate synthetic dataset

np.random.seed(0)

n\_samples = 100

X1 = np.random.rand(n\_samples) \* 10 # Feature 1

X2 = np.random.rand(n\_samples) \* 5 # Feature 2

X3 = np.random.rand(n\_samples) \* 3 # Feature 3

epsilon = np.random.randn(n\_samples) # Noise

y = 3 \* X1 + 2 \* X2 + 5 \* X3 + 10 + epsilon # Target variable

# Create DataFrame

data = {'Feature 1': X1, 'Feature 2': X2, 'Feature 3': X3, 'Target': y}

df = pd.DataFrame(data)

# Display the dataset

print("Dataset:")

print(df.head())

# Split the dataset into features (X) and target variable (y)

X = df[['Feature 1', 'Feature 2', 'Feature 3']].values # Independent variables (features)

y = df['Target'].values # Dependent variable (target)

# Train the Multiple Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Predict on the training data

y\_pred = model.predict(X)

# Calculate performance metrics

mse = mean\_squared\_error(y, y\_pred)

r2 = r2\_score(y, y\_pred)

print('\nPerformance Metrics:')

print('Mean Squared Error:', mse)

print('R^2 Score:', r2)

# Coefficients and Intercept

print("\nCoefficients:", model.coef\_)

print("Intercept:", model.intercept\_)

**Slip 19**

**Q.1 Write a python program to Prepare Scatter Plot (Use Forge Dataset).**

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

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

df = pd.read\_csv('iris.csv')

print("Dataset:")

print(df.head())

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

y = df['Species']

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

model = GaussianNB()

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

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

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

print('\nAccuracy:', accuracy)

print('\nClassification Report:')

print(classification\_report(y\_test, y\_pred))

print('\nConfusion Matrix:')

print(confusion\_matrix(y\_test, y\_pred))

**Q.2 Write a python program to implement k-means algorithm on a synthetic Dataset.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans

data, labels = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60,

random\_state=42)

kmeans = KMeans(n\_clusters=4)

kmeans.fit(data)

centers = kmeans.cluster\_centers\_

predicted\_labels = kmeans.labels\_

plt.scatter(data[:, 0], data[:, 1], c=predicted\_labels, cmap='viridis', edgecolors='k')

plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('Clustering with K-Means')

plt.show()

**Slip 20**

**Q.1 Calculate total sale data for last year for each product and show it using a Pie chart. (Create sales\_data.csv file)**

import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Create sales\_data.csv file (you can replace the data with your actual sales data)

data = {

'Product': ['Product A', 'Product B', 'Product C', 'Product D'],

'Total Sales': [5000, 7000, 3000, 9000] # Replace with actual sales data for each product

}

df = pd.DataFrame(data)

df.to\_csv('sales\_data.csv', index=False)

# Step 2: Read data from sales\_data.csv

df = pd.read\_csv('sales\_data.csv')

# Step 3: Calculate total sales for each product

total\_sales = df['Total Sales']

# Step 4: Plot the total sales data using a pie chart

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

plt.pie(total\_sales, labels=df['Product'], autopct='%1.1f%%', startangle=140)

plt.title('Total Sales Data for Last Year')

plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

**Q.2 Write a python program to implement Eclat Algorithm (OnlineRetail.xlsx dataset)**

import pandas as pd

from collections import defaultdict

# Load the dataset

df = pd.read\_excel('OnlineRetail.xlsx')

# Data preprocessing

# Remove spaces in the description column

df['Description'] = df['Description'].str.strip()

# Remove rows with missing values in InvoiceNo

df.dropna(axis=0, subset=['InvoiceNo'], inplace=True)

# Remove credit transactions (invoices starting with 'C')

df = df[~df['InvoiceNo'].astype(str).str.startswith('C')]

# Group items by transaction

transactions = df.groupby(['InvoiceNo'])['Description'].apply(list)

# Define Eclat function

def eclat(transactions, min\_support):

# Count the occurrences of each item

item\_counts = defaultdict(int)

for transaction in transactions:

for item in transaction:

item\_counts[item] += 1

# Find frequent items

frequent\_items = {item for item, count in item\_counts.items() if count >= min\_support}

# Initialize the Eclat tree

eclat\_tree = {}

for item in frequent\_items:

eclat\_tree[item] = set()

# Build the Eclat tree

for transaction in transactions:

for i, item1 in enumerate(frequent\_items):

if item1 in transaction:

for item2 in frequent\_items[i+1:]:

if item2 in transaction:

eclat\_tree[item1].add(item2)

eclat\_tree[item2].add(item1)

return eclat\_tree

# Set minimum support

min\_support = 100 # Adjust this value as needed

# Run the Eclat algorithm

eclat\_tree = eclat(transactions, min\_support)

# Print the results

print("Frequent Itemsets:")

for item, neighbors in eclat\_tree.items():

print(f"{item}: {', '.join(neighbors)}")