**Slip 4**

**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 5**

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

import pandas as pd

import matplotlib.pyplot as plt

# Load the Iris dataset

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

# Select two features for the scatter plot

feature1 = 'sepal\_length'

feature2 = 'sepal\_width'

# Plot the scatter plot

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

plt.scatter(df[feature1], df[feature2], c='blue', marker='o', edgecolors='k', alpha=0.8)

plt.title('Scatter Plot of Iris Dataset')

plt.xlabel(feature1)

plt.ylabel(feature2)

plt.grid(True)

plt.show()

**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 6**

**Q.1 Write a Python program to implement to find all null values in a given Dataset 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 k-nearest Neighbors ML algorithm to build prediction model (Use Forge Dataset)**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

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

from sklearn.neighbors import KNeighborsClassifier

# Generate synthetic dataset (Forge dataset)

X, y = make\_classification(n\_samples=100, n\_features=2, n\_redundant=0, n\_clusters\_per\_class=1, random\_state=42)

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the KNN classifier

k = 5 # Number of neighbors

knn = KNeighborsClassifier(n\_neighbors=k)

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

# Plot the decision boundary

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(y\_min, y\_max, 0.02))

Z = knn.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.4)

plt.scatter(X[:, 0], X[:, 1], c=y, marker='o', edgecolors='k')

plt.title('KNN Decision Boundary')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.grid(True)

plt.show()

# Evaluate the model

accuracy = knn.score(X\_test, y\_test)

print("Accuracy:", accuracy)

**Slip 7**

**Q.1 Read Total profit of all months and show it using a line plot. (create sales\_data.csv file) Total profit data provided for each month. Generated line plot must include the following properties: –X label name = Month Number, Y label name = Total profit**

import pandas as pd

import matplotlib.pyplot as plt

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

data = {

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

'Total Profit': [20000, 22000, 25000, 24000, 26000, 27000, 30000, 32000, 31000, 33000, 35000, 37000]

}

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 total profit data using a line plot

plt.plot(df['Month Number'], df['Total Profit'], marker='o')

plt.title('Total Profit of All Months')

plt.xlabel('Month Number')

plt.ylabel('Total Profit')

plt.grid(True)

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)}")