EXP 3 : Perform data pre-processing

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder

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

from sklearn.compose import ColumnTransformer

# Sample dataset with missing values, duplicate rows, and categorical values

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

# Create a DataFrame

df = pd.DataFrame(data)

print("Original Data:")

print(df)

# Step 1: Handling Missing Values (Filling up missing values)

# Use SimpleImputer to fill missing values for 'Age' and 'Salary' with mean

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

df[['Age', 'Salary']] = imputer.fit\_transform(df[['Age', 'Salary']])

print("\nData after handling missing values:")

print(df)

# Step 2: Removing Duplicate Data

df.drop\_duplicates(inplace=True)

print("\nData after removing duplicate rows:")

print(df)

# Step 3: Handling Noisy Data (If there's noisy data, it should be handled here)

# For demonstration, let's assume Salary less than 30000 is an error (noisy data)

# Handling noisy data by applying a threshold or fixing error-prone values

df['Salary'] = df['Salary'].apply(lambda x: x if x >= 30000 else np.nan)

# df['Salary'] = imputer.fit\_transform(df[['Salary']]) # Re-impute noisy data

print("\nData after handling noisy data:")

print(df)

# Step 4: Handling Outliers (if any)

# Let's assume 'Age' greater than 60 is an outlier

# We'll replace outliers with the mean value (same as filling missing data)

df['Age'] = df['Age'].apply(lambda x: x if x <= 60 else df['Age'].mean())

print("\nData after handling outliers:")

print(df)

# Step 5: Scaling the Data (Standardization)

# Apply standard scaling to numerical data (Age and Salary)

scaler = StandardScaler()

df[['Age', 'Salary']] = scaler.fit\_transform(df[['Age', 'Salary']])

print("\nData after scaling:")

print(df)

**Data.csv**

Country Age Salary Purchased

France 44 72000 No

Spain 27 48000 Yes

Germany 30 54000 No

Spain 38 61000 No

Germany 40 Yes

France 35 58000 Yes

Spain 52000 No

France 48 2000 Yes

Germany 50 83000 No

Germany 40 Yes

France 77 67000 Yes

France 44 72000 No

EXP 04 : Perform data visualization

# Importing required libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Creating a sample dataset

np.random.seed(0) # For reproducibility

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

# Convert to DataFrame

df = pd.DataFrame(data)

# 1. Scatter Plot

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

sns.scatterplot(x='Age', y='Salary', hue='Purchased', style='Purchased', data=df)

plt.title('Scatter Plot of Age vs Salary')

plt.xlabel('Age')

plt.ylabel('Salary')

plt.legend(title='Purchased')

plt.grid()

plt.show()

# 2. Histogram

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

sns.histplot(df['Salary'], bins=20, kde=True)

plt.title('Histogram of Salary Distribution')

plt.xlabel('Salary')

plt.ylabel('Frequency')

plt.grid()

plt.show()

# 3. Box Plot

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

sns.boxplot(x='Purchased', y='Salary', data=df)

plt.title('Box Plot of Salary by Purchase Decision')

plt.xlabel('Purchased')

plt.ylabel('Salary')

plt.grid()

plt.show()

Data.csv

Country Age Salary Purchased

France 44 72000 No

Spain 27 48000 Yes

Germany 30 54000 No

Spain 38 61000 No

Germany 40 Yes

France 35 58000 Yes

Spain 52000 No

France 48 2000 Yes

Germany 50 83000 No

Germany 40 Yes

France 77 67000 Yes

France 44 72000 No

EXP 05 : Implement classification algorithm(Naïve Bayes)

import pandas as pd

# Load the dataset

df = pd.read\_csv('golf\_data.csv', encoding='ISO-8859-1')

# Print column names to verify

print(df.columns)

# Use the correct column name 'play'

df\_encoded = pd.get\_dummies(df.drop('play', axis=1), drop\_first=True) # Drop the first category to avoid multicollinearity

df\_encoded['play'] = df['play'] # Make sure to add the target variable back

X = df\_encoded.drop('play', axis=1)

y = df\_encoded['play']

# Calculate prior probabilities

prior\_probs = y.value\_counts(normalize=True)

# Function to calculate conditional probabilities

def calc\_conditional\_probs(df, feature, target):

feature\_values = df[feature].unique()

target\_values = df[target].unique()

probs = {}

for t in target\_values:

subset = df[df[target] == t]

probs[t] = subset[feature].value\_counts(normalize=True).to\_dict()

return probs

# Calculate conditional probabilities for each feature

cond\_probs = {feature: calc\_conditional\_probs(df\_encoded, feature, 'play') for feature in X.columns}

print("Conditional Probabilities:")

for feature in cond\_probs:

print(f"{feature}: {cond\_probs[feature]}")

# Function to make predictions

def predict(row, prior\_probs, cond\_probs):

probs = {}

for target in prior\_probs.index:

prob = prior\_probs[target]

for feature in row.index:

value = row[feature]

if feature in cond\_probs:

prob \*= cond\_probs[feature].get(value, {}).get(target, 1e-6)

else:

prob \*= 1e-6

probs[target] = prob

return max(probs, key=probs.get)

# Test instance

test\_instance = pd.Series({

'outlook\_Overcast': 0,

'outlook\_Rain': 0,

'outlook\_Sunny': 1,

'temp\_Cool': 0,

'temp\_Hot': 0,

'temp\_Mild': 1,

'humidity\_High': 0,

'humidity\_Normal': 1,

'wind\_Weak': 1,

'wind\_Strong': 0

})

# Make prediction

prediction = predict(test\_instance, prior\_probs, cond\_probs)

print(f'Predicted Class: {prediction}')

**golf\_data.csv**

day outlook temp humidity wind play

D1 Sunny Hot High Weak No

D2 Sunny Hot High Strong No

D3 Overcast Hot High Weak Yes

D4 Rain Mild High Weak Yes

D5 Rain Cool Normal Weak Yes

D6 Rain Cool Normal Strong No

D7 Overcast Cool Normal Strong Yes

D8 Sunny Mild High Weak No

D9 Sunny Cool Normal Weak Yes

D10 Rain Mild Normal Weak Yes

D11 Sunny Mild Normal Strong Yes

D12 Overcast Mild High Strong Yes

D13 Overcast Hot Normal Weak Yes

D14 Rain Mild High Strong No

EXP 06 : Implement clustering algorithm(K-means)

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cluster import KMeans

# Load the dataset

dataset = pd.read\_csv('Mall\_Customers.csv')

# Selecting the features for clustering (Annual Income and Spending Score)

X = dataset.iloc[:, [3, 4]].values

# Using the Elbow method to find the optimal number of clusters

wcss = []

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_) # Inertia: sum of squared distances of samples to their closest cluster center

# Plotting the Elbow graph

plt.plot(range(1, 11), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS (Within-Cluster Sum of Squares)')

plt.show()

# Implementing K-Means with the optimal number of clusters (let's say 5)

optimal\_clusters = 5

kmeans = KMeans(n\_clusters=optimal\_clusters, init='k-means++', random\_state=42)

y\_kmeans = kmeans.fit\_predict(X)

# Visualizing the clusters

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

colors = ['red', 'blue', 'green', 'cyan', 'magenta']

for i in range(optimal\_clusters):

plt.scatter(X[y\_kmeans == i, 0], X[y\_kmeans == i, 1], s=100, c=colors[i], label=f'Cluster {i + 1}')

# Plotting the centroids

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='yellow', label='Centroids')

plt.title('Clusters of Customers')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

# Conclusion statement

print("We have successfully implemented the K-Means clustering algorithm.")

**Mall\_Customers.csv**

CustomerID Genre Age Annual Income (k$) Spending Score (1-100)

1 Male 19 15 39

2 Male 21 15 81

3 Female 20 16 6

4 Female 23 16 77

5 Female 31 17 40

6 Female 22 17 76

7 Female 35 18 6

8 Female 23 18 94

9 Male 64 19 3

10 Female 30 19 72

11 Male 67 19 14

12 Female 35 19 99

13 Female 58 20 15

14 Female 24 20 77

15 Male 37 20 13

16 Male 22 20 79

17 Female 35 21 35

18 Male 20 21 66

19 Male 52 23 29

20 Female 35 23 98

21 Male 35 24 35

22 Male 25 24 73

23 Female 46 25 5

24 Male 31 25 73

25 Female 54 28 14

26 Male 29 28 82

27 Female 45 28 32

28 Male 35 28 61

29 Female 40 29 31

30 Female 23 29 87

31 Male 60 30 4

32 Female 21 30 73

33 Male 53 33 4

34 Male 18 33 92

35 Female 49 33 14

36 Female 21 33 81

37 Female 42 34 17

38 Female 30 34 73

39 Female 36 37 26

40 Female 20 37 75

41 Female 65 38 35

42 Male 24 38 92

43 Male 48 39 36

44 Female 31 39 61

45 Female 49 39 28

46 Female 24 39 65

47 Female 50 40 55

48 Female 27 40 47

49 Female 29 40 42

50 Female 31 40 42

51 Female 49 42 52

52 Male 33 42 60

53 Female 31 43 54

54 Male 59 43 60

55 Female 50 43 45

56 Male 47 43 41

57 Female 51 44 50

58 Male 69 44 46

59 Female 27 46 51

60 Male 53 46 46

61 Male 70 46 56

62 Male 19 46 55

63 Female 67 47 52

64 Female 54 47 59

65 Male 63 48 51

66 Male 18 48 59

67 Female 43 48 50

68 Female 68 48 48

69 Male 19 48 59

70 Female 32 48 47

71 Male 70 49 55

72 Female 47 49 42

73 Female 60 50 49

74 Female 60 50 56

75 Male 59 54 47

76 Male 26 54 54

77 Female 45 54 53

78 Male 40 54 48

79 Female 23 54 52

80 Female 49 54 42

81 Male 57 54 51

82 Male 38 54 55

83 Male 67 54 41

84 Female 46 54 44

85 Female 21 54 57

86 Male 48 54 46

87 Female 55 57 58

88 Female 22 57 55

89 Female 34 58 60

90 Female 50 58 46

91 Female 68 59 55

92 Male 18 59 41

93 Male 48 60 49

94 Female 40 60 40

95 Female 32 60 42

96 Male 24 60 52

97 Female 47 60 47

98 Female 27 60 50

99 Male 48 61 42

100 Male 20 61 49

101 Female 23 62 41

102 Female 49 62 48

103 Male 67 62 59

104 Male 26 62 55

105 Male 49 62 56

106 Female 21 62 42

107 Female 66 63 50

108 Male 54 63 46

109 Male 68 63 43

110 Male 66 63 48

111 Male 65 63 52

112 Female 19 63 54

113 Female 38 64 42

114 Male 19 64 46

115 Female 18 65 48

116 Female 19 65 50

117 Female 63 65 43

118 Female 49 65 59

119 Female 51 67 43

120 Female 50 67 57

121 Male 27 67 56

122 Female 38 67 40

123 Female 40 69 58

124 Male 39 69 91

125 Female 23 70 29

126 Female 31 70 77

127 Male 43 71 35

128 Male 40 71 95

129 Male 59 71 11

130 Male 38 71 75

131 Male 47 71 9

132 Male 39 71 75

133 Female 25 72 34

134 Female 31 72 71

135 Male 20 73 5

136 Female 29 73 88

137 Female 44 73 7

138 Male 32 73 73

139 Male 19 74 10

140 Female 35 74 72

141 Female 57 75 5

142 Male 32 75 93

143 Female 28 76 40

144 Female 32 76 87

145 Male 25 77 12

146 Male 28 77 97

147 Male 48 77 36

148 Female 32 77 74

149 Female 34 78 22

150 Male 34 78 90

151 Male 43 78 17

152 Male 39 78 88

153 Female 44 78 20

154 Female 38 78 76

155 Female 47 78 16

156 Female 27 78 89

157 Male 37 78 1

158 Female 30 78 78

159 Male 34 78 1

160 Female 30 78 73

161 Female 56 79 35

162 Female 29 79 83

163 Male 19 81 5

164 Female 31 81 93

165 Male 50 85 26

166 Female 36 85 75

167 Male 42 86 20

168 Female 33 86 95

169 Female 36 87 27

170 Male 32 87 63

171 Male 40 87 13

172 Male 28 87 75

173 Male 36 87 10

174 Male 36 87 92

175 Female 52 88 13

176 Female 30 88 86

177 Male 58 88 15

178 Male 27 88 69

179 Male 59 93 14

180 Male 35 93 90

181 Female 37 97 32

182 Female 32 97 86

183 Male 46 98 15

184 Female 29 98 88

185 Female 41 99 39

186 Male 30 99 97

187 Female 54 101 24

188 Male 28 101 68

189 Female 41 103 17

190 Female 36 103 85

191 Female 34 103 23

192 Female 32 103 69

193 Male 33 113 8

194 Female 38 113 91

195 Female 47 120 16

196 Female 35 120 79

197 Female 45 126 28

198 Male 32 126 74

199 Male 32 137 18

200 Male 30 137 83

EXP 07: Implement frequent pattern mining algorithm (Apriori)

import numpy as np

import pandas as pd

from apyori import apriori

# Load the dataset

dataset = pd.read\_csv('Market\_Basket\_Optimisation.csv', header=None)

# Prepare the transactions

transactions = []

for i in range(len(dataset)):

# Use dataset.shape[1] to handle dynamic column count

transactions.append([str(dataset.values[i, j]) for j in range(dataset.shape[1]) if pd.notna(dataset.values[i, j])])

# Applying the Apriori algorithm

rules = apriori(transactions=transactions, min\_support=0.003, min\_confidence=0.2, min\_lift=3, min\_length=2)

# Convert the results to a list

results = list(rules)

# Displaying the results

for rule in results:

for ordered\_stat in rule.ordered\_statistics:

print(f"Rule: {rule.items}, Support: {rule.support}, Confidence: {ordered\_stat.confidence}, Lift: {ordered\_stat.lift}")

Market\_Basket\_Optimisation.csv

shrimp avocado vegetables mix green grapes whole wheat flour

turkey eggs french fries

shrimp pasta mineral water soup

EXP 10 : PAGE RANK Algorithm

import numpy as np

# Normalize the adjacency matrix (make it a probability matrix where all columns sum to 1)

def normalize\_adjacency\_matrix(A):

n = len(A) # number of rows/cols in A

for j in range(len(A[0])):

sum\_of\_col = sum(A[i][j] for i in range(n)) # Sum of the j-th column

if sum\_of\_col == 0: # Adjust for dangling nodes (columns of zeros)

for i in range(n):

A[i][j] = 1 / n

else:

for i in range(n):

A[i][j] /= sum\_of\_col # Normalize the column

return A

# Implement damping matrix using the formula

# M = dA + (1-d)(1/n)Q, where Q is an array of 1's and d is the damping factor

def damping\_matrix(A, damping\_factor=0.85):

n = len(A) # number of rows/cols in A

Q = np.ones((n, n)) / n # Array of 1's normalized

arrA = np.array(A)

arrM = damping\_factor \* arrA + (1 - damping\_factor) \* Q # Create damping matrix

return arrM

# Find the steady-state vector corresponding to the eigenvalue 1

def find\_steady\_state(M):

eigenvalues, eigenvectors = np.linalg.eig(M)

# Find the index of the eigenvector corresponding to the eigenvalue 1

idx\_with\_eval1 = np.isclose(eigenvalues, 1) # Use np.isclose for numerical stability

steady\_state\_vector = eigenvectors[:, idx\_with\_eval1]

# Normalize the steady state vector so its components sum to 1

steady\_state\_vector = steady\_state\_vector / np.sum(steady\_state\_vector)

return steady\_state\_vector.flatten() # Return as a 1D array

# PageRank function

def page\_rank(A):

A = normalize\_adjacency\_matrix(A)

M = damping\_matrix(A)

# Find steady state vector

steady\_state\_vector = find\_steady\_state(M)

return steady\_state\_vector

# TEST CASES

print("\nPage Rank Examples")

# 1) Corresponds to directed graph (1)

matrix1 = [

[0, 1, 0, 0],

[0, 0, 0, 0],

[0, 1, 0, 1],

[0, 0, 1, 0]

]

print("1) matrix 1 = ", matrix1)

print("steady state vector: ")

print(page\_rank(matrix1)) # expected output: approximately [0.077, 0.054, 0.441, 0.429]

# 2)

matrix2 = [

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

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

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

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

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

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

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

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

]

print("\n2) matrix 2 = ", matrix2)

print("steady state vector: ")

print(page\_rank(matrix2)) # expected output: approximately [0.03037, 0.0536, 0.02735, 0.0617, 0.1621, 0.2836, 0.2419, 0.1393]