DATA MINING

Project

BY:

Mohammad Dabash(2035068)
Yazeed Alsahouri(2136866)
Abdulrahman Abuhani(2132462)
Belal Khaled(2136873)

Table Of Contents:

Topics: **Page:**

Introduction:

In this report, we delve into the comprehensive process of analyzing a dataset, starting from: initial exploration to data pre-processing and concluding with association rules mining.

Task A (Data exploration):

Here we will do the following using 'Community-Participation-DataSet(8).csv':

- ✓ ***Identify the type of each attribute(nominal, ordinal, interval or ratio).***
- ✓ **Statistical summaries and visualizations.**
- ✓ ***Clustering and detecting Outliers.***

Task B *(Data preprocessing)*:

Here we will do the following using *'Scoring-Dataset-8.csv'*:

- ✓ ***Binning with both (equal width and equal depth) then smoothing by bin means.***
- ✓ ***Normalization.***
- ✓ ***Discretization.***
- ✓ ***Converting Gender attribute to binary.***
- ✓ ***Final view of the preprocessed attributes.***

Task C (Association rules mining):

Here we will do the following using 'Community-Participation-DataSet(8).csv':

- ✓ ***Apriori.***
- ✓ ***Increasing the threshold of confidence to be > 0.70.***

➢ Importing all libraries we need:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from sklearn.preprocessing import  StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.cluster import KMeans
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
import warnings
warnings.filterwarnings("ignore")
```

TASK A(Data exploration):

A.1 Data Types:

- **Code:**

```
data1 = pd.read_csv('/content/Community-Participation-DataSet(8).csv')
data1
#The data contains 2000 `record` and 13 `attribute`, let's explore their types.

data1.info()
```

➢ Output for A.1(Data Types):

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 13 columns):
#   Column                Non-Null Count             Dtype
---  ------               -----------------------    -------------------------
0   Record#               2000 non-null              int64  → Ordinal
1   Elapsed_Time          2000 non-null              float64 → Ratio
2   Time_in_Community     2000 non-null              object → Ordinal
3   Gender                2000 non-null              object → Binary
4   Working               2000 non-null              object → Binary
5   Age                   2000 non-null              int64  → Ratio
6   Family                2000 non-null              object → Binary
7   Hobbies               2000 non-null              object → Binary
8   Social_Club           2000 non-null              object → Binary
9   Political             2000 non-null              object → Binary
10  Professional          2000 non-null              object → Binary
11  Religious             2000 non-null              object → Binary
12  Support_Group         2000 non-null              object → Binary

dtypes: float64(1), int64(2), object(10)
memory usage: 203.2+ KB
```

TASK A(Data exploration):

A.2 Statistical summaries and visualizations:

- ## Code:

#Let's start simple:

1- `data1[['Age', 'Elapsed_Time']].describe().T`

2- `data1.describe(include = "object").T`

3-
```
sns.histplot(data1['Elapsed_Time'], kde=True)
plt.title(f'Distribution of Elapsed_Time')
plt.xlabel('Elapsed_Time')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show() #Elapsed_Time distribution using Histogram plot.
```

4-
```
sns.histplot(data1['Age'], kde=True)
plt.title(f'Distribution of Elapsed_Time')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show() #Age distribution using Histogram plot.
```

5-
```
sns.scatterplot(data=data1, x='Elapsed_Time', y='Age')
plt.show()#Correlation between Age and Elapsed Time.
```

#Lets see the value counts of some attributes using word cloud and bar plot ➔

#Time_in_Community ➔

6-
```
text = data1['Time_in_Community']
text = ' '.join(list(text))
wordcloud = WordCloud().generate(text)
x = list(dict(data1['Time_in_Community'].value_counts()).keys())
counts = list(data1['Time_in_Community'].value_counts())
fig, axs = plt.subplots(1, 2, figsize=(15,7))
axs[0].imshow(wordcloud, interpolation='bilinear')
axs[0].axis("off")
axs[1].bar(x=x,height=counts, data=data1, color=['#5F0F40', '#FB8B24', '#E36414'])
plt.show()
```

#Gender ➔
```
text = data1['Gender']
text = ' '.join(list(text))
wordcloud = WordCloud().generate(text)
x = list(dict(data1['Gender'].value_counts()).keys())
counts = list(data1['Gender'].value_counts())
fig, axs = plt.subplots(1, 2, figsize=(15,7))
axs[0].imshow(wordcloud, interpolation='bilinear')
axs[0].axis("off")
axs[1].bar(x=x,height=counts, data=data1, color=['#5F0F40', '#FB8B24', '#E36414'])
plt.show()
```

➢ *Output for A.2* (Statistical summaries and visualizations):

1➔

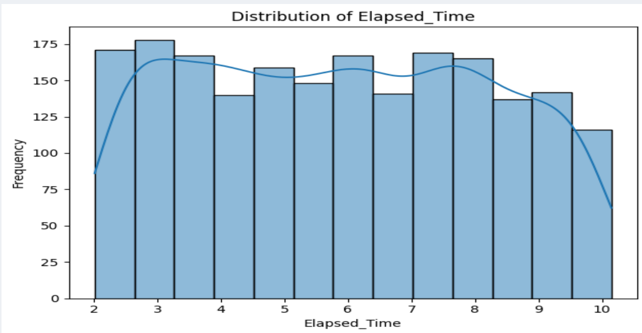
	count	mean	std	min	25%	50%	75%	max
Age	2000.0	36.727500	10.699368	17.00	27.00	37.000	46.0000	57.00
Elapsed_Time	2000.0	5.912215	2.324282	2.01	3.84	5.905	7.8625	10.15

2➔

	count	unique	top	freq
Time_in_Community	2000	3	Long	841
Gender	2000	2	F	1028
Working	2000	2	Yes	1001
Family	2000	2	No	1208
Hobbies	2000	2	No	1413
Social_Club	2000	2	No	1627
Political	2000	2	No	1834
Professional	2000	2	No	1342
Religious	2000	2	No	1167
Support_Group	2000	2	No	1691

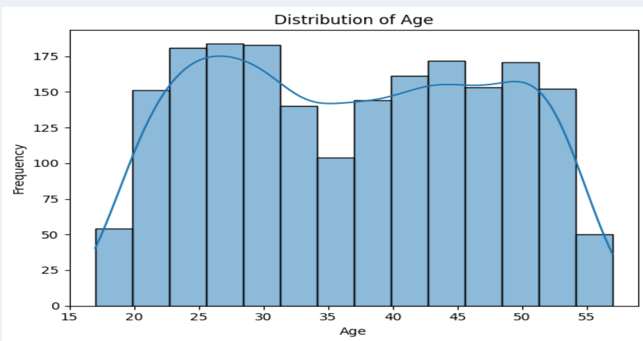
(The mean and median being close suggests the data might follow a normal distribution).

3➔



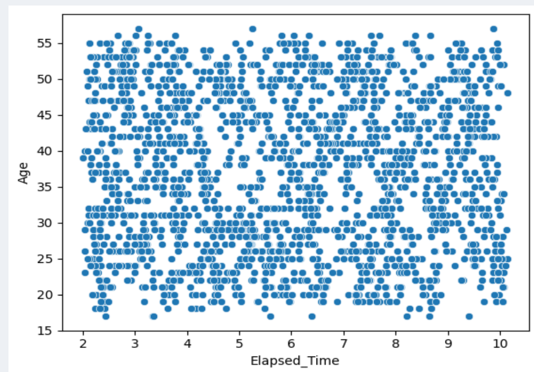
('Elapsed_Time' distribution using Histogram plot).

4➔



('Age' distribution using Histogram plot).

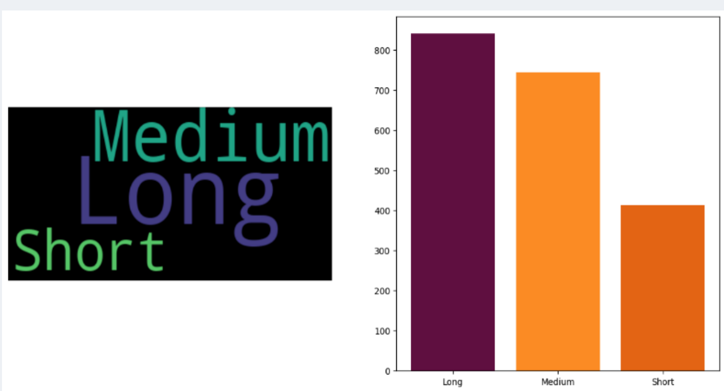
5➔



(Correlation between Age and Elapsed_Time)

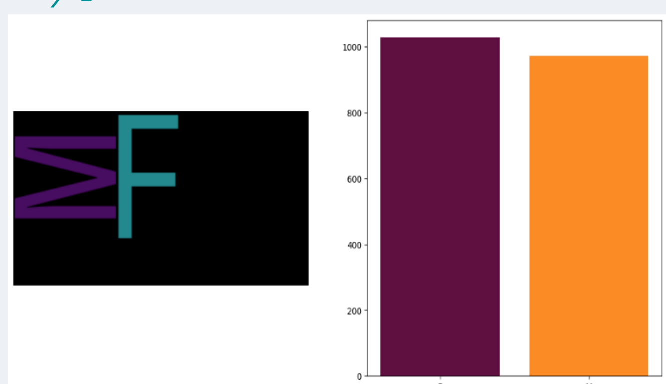
➔ There is no clear correlation.

6➔



(value counts for 'Time_in_Community')
using word cloud and bar plot.

7➔



(value counts for 'Gender')
using word cloud and bar plot.

TASK A(Data exploration):

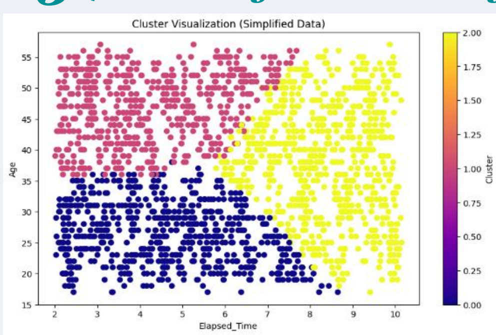
A.3 Clustering and detecting Outliers:

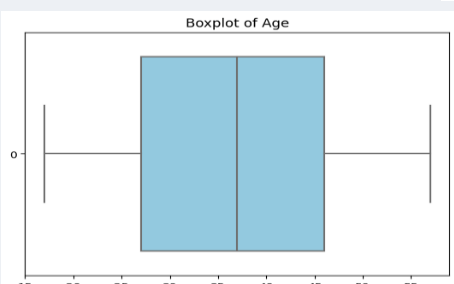
- ***Code:***

```
#Clustering:
numerical = data1.select_dtypes(include=['number']).drop('Record#', axis=1).columns
categorical = data1.select_dtypes(include=['object']).columns
preprocessor = ColumnTransformer(
transformers=[
('num', StandardScaler(), numerical),('cat', OneHotEncoder(), categorical) ] )
prep_data = preprocessor.fit_transform(data1)
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(prep_data)
plt.figure(figsize=(10,6))
plt.scatter(data1['Elapsed_Time'], data1['Age'], c=clusters, cmap='plasma', marker='o')
plt.title('Cluster Visualization (Simplified Data)')
plt.xlabel('Elapsed_Time')
plt.ylabel('Age')
plt.colorbar(label='Cluster')
plt.show()
#OutLiers(we will detect Outliers using Boxplot):
#Age➔
sns.boxplot(data=data1['Age'], color='skyblue', orient='h')
plt.title('Boxplot of Age')
plt.show()
#Elapsed_Time➔
sns.boxplot(data=data1['Elapsed_Time'], color='mediumseagreen', orient='h')
plt.title('Boxplot of Elapsed_Time')
plt.show()
```

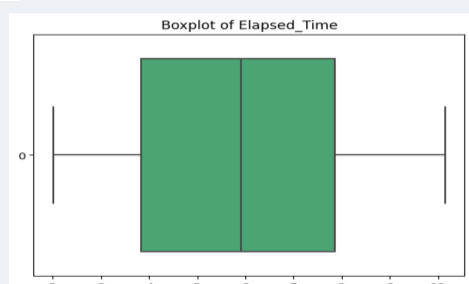
➢ *Output for A.3 (Clustering and detecting Outliers):*

Cluster visualization (simplified data).➔





➔ *There is no OutLiers in our data depending on the Boxplot.* 🡐



Boxplot of Age

Boxplot of Elapsed_Time

TASK B(Data preprocessing):

B.1 Binning:

- ## Code(Equal width binning):

```
#We Chose the 8th dataset.
 data2 = pd.read_csv('/content/Scoring-Dataset-8.csv')
 data2

#Performing Equal width binning ➔
 min_age = data2['Age'].min()
 max_age = data2['Age'].max()
 w = (max_age - min_age) / 3
 bin_boundaries = [min_age + i * w for i in range(3)]#bin boundaries = min + i * w
 bin_boundaries.append(max_age)
 data2['equi_width'] = pd.cut( data2['Age'], bins=bin_boundaries, labels=
                               ['Bin 1', 'Bin 2', 'Bin 3'], include_lowest=True )
 data2[['Age', 'equi_width']]
#Bin values ➔
 for bin_number in ['Bin 1', 'Bin 2', 'Bin 3']:
 bin_values = data2[data2['equi_width'] == bin_number]['Age'].tolist()
 print(f'{bin_number}: {bin_values}\n') ➔ #prints every bin and its values.
#Smoothing by bin means ➔
 for bin_number in ['Bin 1', 'Bin 2', 'Bin 3']:
 bin_mean = np.mean(data2[data2['equi_width'] == bin_number]['Age'].tolist())
 bin_mean = round(bin_mean)
 data2.loc[data2['equi_width'] == bin_number, 'Age_Smoothed'] = bin_mean
 data2[['Age', 'equi_width', 'Age_Smoothed']]
#We'll keep smoothed values only ➔
 data2.drop('equi_width', axis=1, inplace=True)
```

- ## Code(Equal depth binning):

```
#Performing Equal depth binning ➔
#first we will sort the data by age to perform equal depth binning ➔
 data2 = data2.sort_values(by='Age')
 data2.reset_index(drop=True, inplace=True)
 data2.loc[:99, 'equi_depth'] = 'Bin 1'
 data2.loc[100:199, 'equi_depth'] = 'Bin 2'
 data2.loc[200:, 'equi_depth'] = 'Bin 3'
 data2[['Age', 'equi_depth']]

#Making sure that they are equal ➔
 data2['equi_depth'].value_counts()

#Smoothing by bin means ➔
 for bin_number in ['Bin 1', 'Bin 2', 'Bin 3']:
 bin_mean = np.mean(data2[data2['equi_depth'] == bin_number]['Age'].tolist())
 bin_mean = round(bin_mean)
 data2.loc[data2['equi_depth'] == bin_number, 'Age_Smoothed_2'] = bin_mean
 data2[['Age', 'equi_depth', 'Age_Smoothed_2']]
#We'll keep smoothed values only ➔
 data2.drop('equi_depth', axis=1, inplace=True)
```

➢ *Output for B.1 (Binning):*

	Age	equi_width
0	54	Bin 3
1	24	Bin 1
2	45	Bin 2
3	33	Bin 1
4	37	Bin 2
...	...	...
295	37	Bin 2
296	46	Bin 2
297	66	Bin 3
298	37	Bin 2
299	26	Bin 1

300 rows × 2 columns

➔ **Performing Equal width binning**

Smoothing by bin means (Equal width) 🡐

	Age	equi_width	Age_Smoothed
0	54	Bin 3	60.0
1	24	Bin 1	27.0
2	45	Bin 2	45.0
3	33	Bin 1	27.0
4	37	Bin 2	45.0
...	...	...	...
295	37	Bin 2	45.0
296	46	Bin 2	45.0
297	66	Bin 3	60.0
298	37	Bin 2	45.0
299	26	Bin 1	27.0

300 rows × 3 columns

	Age	equi_depth
0	17	Bin 1
1	17	Bin 1
2	17	Bin 1
3	19	Bin 1
4	19	Bin 1
...	...	...
295	68	Bin 3
296	68	Bin 3
297	68	Bin 3
298	70	Bin 3
299	70	Bin 3

300 rows × 2 columns

➔ **Performing Equal depth binning**

Smoothing by bin means (Equal depth) 🡐

	Age	equi_depth	Age_Smoothed_2
0	17	Bin 1	30.0
1	17	Bin 1	30.0
2	17	Bin 1	30.0
3	19	Bin 1	30.0
4	19	Bin 1	30.0
...	...	...	...
295	68	Bin 3	60.0
296	68	Bin 3	60.0
297	68	Bin 3	60.0
298	70	Bin 3	60.0
299	70	Bin 3	60.0

300 rows × 3 columns

TASK B(Data preprocessing):

B.2 Normalization:

- ## *Code:*

```
#Min Max
min = data2['Age'].min()
max = data2['Age'].max()

data2['Age_minmax'] = (data2['Age'] - min) / (max - min) ➔ #calculate for each age its min_max.

#Z-score
mu = data2['Age'].mean()
std = data2['Age'].std()

data2['Age_Zscore'] = (data2['Age'] - mu) / std ➔ #calculate for each age its Z-score.
```

TASK B(Data preprocessing):

B.3 Discretization:

- ***Code:***

```
labels = ['Teenager', 'Young', 'Mid_Age', 'Mature', 'Old']
edges = [0, 16, 35, 55, 70, np.inf]
data2['Age_categorical'] = pd.cut(data2['Age'], bins=edges, labels=labels)
data2[['Age_categorical', 'Age']]
```

➢ *Output for B.3 (Discretization):*

	Age_categorical	Age
0	Young	17
1	Young	17
2	Young	17
3	Young	19
4	Young	19
...	...	...
295	Mature	68
296	Mature	68
297	Mature	68
298	Mature	70
299	Mature	70

300 rows × 2 columns

TASK B(Data preprocessing):

B.4 Converting Gender to binary:

- ***Code:***

```
data2['Gender_Binary'] = data2['Gender'].map({'M': 1, 'F': 0})
data2[['Gender', 'Gender_Binary']]
```

➢ *Output for B.4 (Converting Gender to binary):*

	Gender	Gender_Binary
0	M	1
1	M	1
2	M	1
3	M	1
4	M	1
...	...	...
295	F	0
296	M	1
297	F	0
298	M	1
299	M	1

300 rows × 2 columns

TASK B(Data preprocessing):

Final view of the preprocessed attributes:

- ***Code:***

```
data_preprocessed = data2[['Age', 'Age_categorical','Age_Smoothed', 'Age_Smoothed_2',
                                                   'Gender', 'Gender_Binary']]

data_preprocessed.sample(8)
```

➢ ***Output:***

	Age	Age_categorical	Age_Smoothed	Age_Smoothed_2	Gender	Gender_Binary
80	36	Mid_Age	45.0	30.0	M	1
85	37	Mid_Age	45.0	30.0	F	0
58	32	Young	27.0	30.0	M	1
194	52	Mid_Age	45.0	47.0	F	0
158	49	Mid_Age	45.0	47.0	F	0
14	23	Young	27.0	30.0	M	1
275	65	Mature	60.0	60.0	F	0
133	46	Mid_Age	45.0	47.0	F	0

TASK C(Association rules mining):

Apriori:

- ***Code:***

```
data = pd.read_csv('/content/Community-Participation-DataSet(8).csv')
data = data.drop(columns=['Age', 'Record#', 'Elapsed_Time'], axis=1)

#First we will convert the data into a list of transactions ➔
transactions = []

for i in range(len(data)):
  one_transaction = []
  row = data.iloc[i] #series
  one_transaction.append(row['Time_in_Community'])

  if row['Working'] == 'Yes': one_transaction.append('Working')
  if row['Family'] == 'Yes': one_transaction.append('Family')
  if row['Support_Group'] == 'Yes': one_transaction.append('Support_Group')
  if row['Religious'] == 'Yes': one_transaction.append('Religious')
  if row['Professional'] == 'Yes': one_transaction.append('Professional')
  if row['Political'] == 'Yes': one_transaction.append('Political')
  if row['Social_Club'] == 'Yes': one_transaction.append('Social_Club')
  if row['Hobbies'] == 'Yes': one_transaction.append('Hobbies')
  transactions.append(one_transaction)

#Declaring a transaction encoder to convert the data to a format that apriori algorithm accepts ➔
t_encoder = TransactionEncoder()
te_arr = t_encoder.fit_transform(transactions)
df = pd.DataFrame(te_arr, columns=t_encoder.columns_)

#Applying apriori algorithm on our transformed data ➔
frequent_itemsets = apriori(df, min_support=0.15, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

rules[['antecedents', 'consequents','support', 'confidence']] ➔ #printing the result.

#Increasing the threshold of confidence to be > 0.70 ➔
frequent_itemsets = apriori(df, min_support=0.15, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.70)

rules[['antecedents', 'consequents','support', 'confidence']] ➔ #printing the result.
```

➢ *Output for task C:*

rules[['antecedents', 'consequents','support', 'confidence']]

➔ the result:

	antecedents	consequents	support	confidence
0	(Hobbies)	(Family)	0.1880	0.640545
1	(Religious)	(Family)	0.2260	0.542617
2	(Family)	(Religious)	0.2260	0.570707
3	(Family)	(Working)	0.2010	0.507576
4	(Religious)	(Hobbies)	0.2375	0.570228
5	(Hobbies)	(Religious)	0.2375	0.809199
6	(Long)	(Working)	0.2275	0.541023
7	(Religious, Hobbies)	(Family)	0.1575	0.663158
8	(Religious, Family)	(Hobbies)	0.1575	0.696903
9	(Hobbies, Family)	(Religious)	0.1575	0.837766
10	(Hobbies)	(Religious, Family)	0.1575	0.536627

After Increasing the threshold of confidence to be > 0.70:

➔ the result:

	antecedents	consequents	support	confidence
0	(Hobbies)	(Religious)	0.2375	0.809199
1	(Hobbies, Family)	(Religious)	0.1575	0.837766

- **These rules suggest strong associations between being involved in hobby-oriented community organizations (and in combination with family-oriented organizations) and being a member of a religious organization.**

✓*References:*

https://www.analyticsvidhya.com/blog/2021/07/data-visualization-a-useful-tool-to-explore-data/#:~:text=What%20are%20the%20tools%20used,%2C%20Power%20BI%2C%20and%20D3.

https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6