M.Naveen kumar

AIE-23168

1.Please refer to the “Purchase Data” worksheet of Lab Session Data.xlsx. Please load the data and segregate them into 2 matrices A & C (following the nomenclature of AX = C). Do the following activities.

• What is the dimensionality of the vector space for this data? • How many vectors exist in this vector space? • What is the rank of Matrix A? • Using Pseudo-Inverse find the cost of each product available for sale. (Suggestion: If you use Python, you can use numpy.linalg.pinv() function to get a pseudo-inverse.)

```
import pandas as pd

import numpy as np


data = pd.read_excel(r"C:\Users\navee\Downloads\Lab Session Data.xlsx")


A = data.loc[:, ['Candies (#)', 'Mangoes (Kg)', 'Milk Packets (#)']].values

C = data[['Payment (Rs)']].values


print(f"A = {A}")

print(f"C = {C}")


dim = A.shape[1]

num_vectors = A.shape[0]

rank_A = np.linalg.matrix_rank(A)

A_pinv = np.linalg.pinv(A)

cost_vector = A_pinv @ C


print(f"The dimensionality of the vector space is = {dim}")

print(f"The number of vectors in the vector space is = {num_vectors}")

print(f"The rank of the matrix A is = {rank_A}")

print(f"The pseudo-inverse of matrix A is =\n{A_pinv}")

print(f"The cost of each product that is available for sale is = {cost_vector.flatten()}")
```

```
A = [[20  6  2]
 [16  3  6]
 [27  6  2]
 [19  1  2]
 [24  4  2]
 [22  1  5]
 [15  4  2]
 [18  4  2]
 [21  1  4]
 [16  2  4]]
C = [[386]
 [289]
 [393]
 [110]
 [280]
 [167]
 [271]
 [274]
 [148]
 [198]]
The dimensionality of the vector space is = 3
The number of vectors in the vector space is = 10
The rank of the matrix A is = 3
The pseudo-inverse of matrix A is =
[[-0.01008596 -0.03124505  0.01013951  0.0290728   0.0182907   0.01161794
  -0.00771348  0.00095458  0.01743623 -0.00542016]
 [ 0.09059668  0.07263726  0.03172933 -0.09071908 -0.01893196 -0.06926996
   0.05675464  0.03152577 -0.07641966  0.00357352]
 [ 0.00299878  0.15874243 -0.05795468 -0.06609024 -0.06295043  0.03348017
   0.01541831 -0.01070461  0.00029003  0.05938755]]
The cost of each product that is available for sale is = [ 1. 55. 18.]
```

A2.Use the Pseudo-inverse to calculate the model vector X for predicting the cost of the products available with the vendor.

```
import pandas as pd

import numpy as np


data = pd.read_excel(r"C:\Users\navee\Downloads\Lab Session Data.xlsx")


data = data.iloc[:, :5].drop(columns=["Customer"])

data.columns = ["Candies", "Mangoes", "Milk_Packets", "Payment"]


X = np.column_stack((np.ones(len(data)), data[["Candies", "Mangoes", "Milk_Packets"]].values))

Y = data["Payment"].values


X_pinv = np.linalg.pinv(X)

model_parameters = X_pinv @ Y


print("Estimated Model Parameters (Intercept and Coefficients):")

print(model_parameters)
```

```
Estimated Model Parameters (Intercept and Coefficients):

[-5.68434189e-14  1.00000000e+00  5.50000000e+01  1.80000000e+01]
```

A3. Mark all customers (in “Purchase Data” table) with payments above Rs. 200 as RICH and others as POOR. Develop a classifier model to categorize customers into RICH or POOR class based on purchase behavior.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report
```

```
data = pd.read_excel(r"C:\Users\navee\Downloads\Lab Session Data.xlsx")

data["Customer Type"] = ["RICH" if amount > 200 else "POOR" for amount in data["Payment (Rs)"]]

features = data[["Candies (#)", "Mangoes (Kg)", "Milk Packets (#)"]]
target = data["Customer Type"]

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.5, random_state=42)

knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

predictions = knn_model.predict(X_test)

print("Classification Report")
print(classification_report(y_test, predictions))
```

```
     precision   recall  f1-score   support

    POOR      0.00      0.00      0.00        2
    RICH      0.60      1.00      0.75        3

  accuracy                        0.60        5
 macro avg      0.30      0.50      0.38        5
weighted avg      0.36      0.60      0.45        5
```

A4.Please refer to the data present in “IRCTC Stock Price” data sheet of the above excel file. Do the following after loading the data to your programming platform. • Calculate the mean and variance of the Price data present in column D. (Suggestion: if you use Python, you may use statistics.mean() & statistics.variance() methods). • Select the price data for all Wednesdays and calculate the sample mean. Compare the mean with the population mean and note your observations.

Select the price data for the month of Apr and calculate the sample mean. Compare the mean with the population mean and note your observations. • From the Chg% (available in column I) find the probability of making a loss over the stock. (Suggestion: use lambda function to find negative values) • Calculate the probability of making a profit on Wednesday. • Calculate the conditional probability of making profit, given that today is Wednesday. • Make a scatter plot of Chg% data against the day of the week

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from statistics import mean, variance


data = pd.read_excel(r"C:\Users\navee\Downloads\Lab Session Data.xlsx",sheet_name="IRCTC
Stock Price")


price_column = data["Price"]

print(f"D = {price_column}")


price_mean = mean(price_column)

price_variance = variance(price_column)

print(f"The mean of column D is = {price_mean}")

print(f"The variance of column D is = {price_variance}")


data["Date"] = pd.to_datetime(data["Date"])

data["Weekday"] = data["Date"].dt.weekday


wednesday_prices = data[data["Weekday"] == 2]["Price"]

wednesday_mean = wednesday_prices.mean()

print(f"The sample mean for all Wednesdays in the dataset is = {wednesday_mean}")
```

```
data["Month"] = data["Date"].dt.month

april_prices = data[data["Month"] == 4]["Price"]

april_mean = mean(april_prices)

print(f"The sample mean for April in the dataset is = {april_mean}")


loss_probability = (data["Chg%"] < 0).mean()

print(f"The probability of making a loss in the stock is {loss_probability}")


profit_wednesdays = (data.loc[data["Weekday"] == 2, "Chg%"] > 0).mean()

print(f"The probability of making a profit in the stock on Wednesday is {profit_wednesdays}")


num_wed = len(wednesday_prices)

num_profitable_wed = (wednesday_prices > 0).sum()

conditional_prob_wed = num_profitable_wed / num_wed

print(f"The conditional probability of making a profit, given that today is Wednesday =
{conditional_prob_wed}")


sns.scatterplot(x="Weekday", y="Chg%", data=data, hue="Weekday", palette="hls")

plt.xlabel("Day of the Week")

plt.ylabel("Chg%")

plt.title("Chg% Distribution by Day of the Week")

plt.show()
```

```
D = 0    2081.85

1    2077.75

2    2068.85

3    2072.95

4    2078.25

     ...

244   1397.40

245   1400.75
```

246 1405.10

247 1412.35

248 1363.05

Name: Price, Length: 249, dtype: float64

The mean of column D is = 1560.663453815261

The variance of column D is = 58732.365352539186

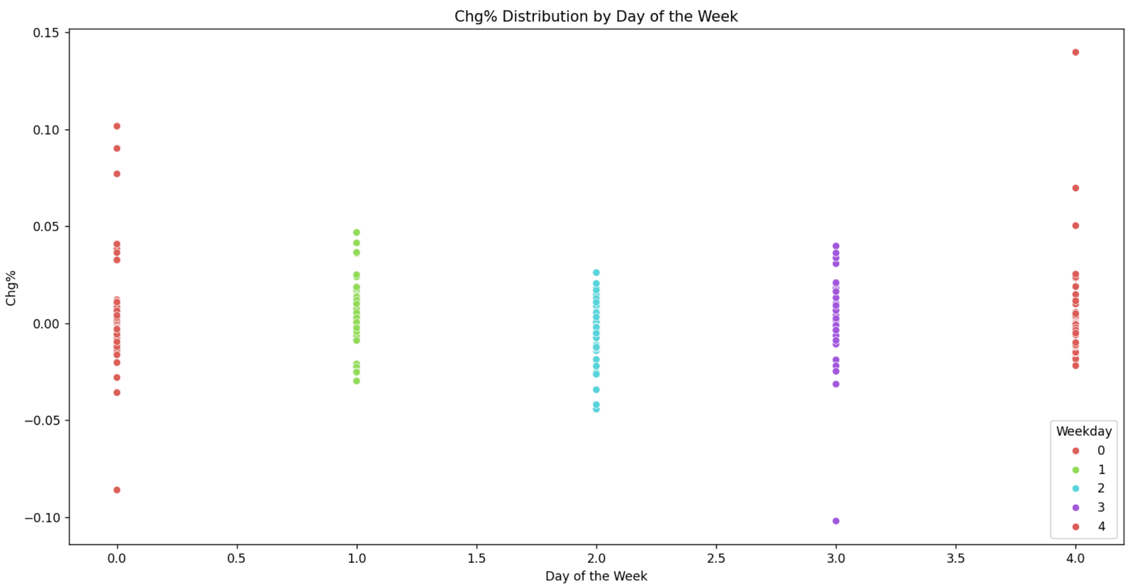
The sample mean for all Wednesdays in the dataset is = 1550.7060000000001

The sample mean for April in the dataset is = 1698.9526315789474

The probability of making a loss in the stock is 0.4979919678714859

The probability of making a profit in the stock on Wednesday is 0.42

The conditional probability of making a profit, given that today is Wednesday = 1.0



A5. Data Exploration: Load the data available in “thyroid0387_UCI” worksheet. Perform the following tasks: • Study each attribute and associated values present. Identify the datatype (nominal etc.) for the attribute. • For categorical attributes, identify the encoding scheme to be employed. (Guidance: employ label encoding for ordinal variables while One-Hot encoding may be employed for nominal variables). • Study the data range for numeric variables. • Study the presence of missing values in each attribute. • Study presence of outliers in data. • For numeric variables, calculate the mean and variance (or standard deviation).

import pandas as pd

import numpy as np

```
import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder, OneHotEncoder



df = pd.read_excel(r"C:\Users\navee\Downloads\Lab Session
Data.xlsx",sheet_name="thyroid0387_UCI")


print(df.info())

categorical_cols = df.select_dtypes(include=['object']).columns

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

label_encoders = {}

for col in categorical_cols:

  unique_values = df[col].nunique()

  if unique_values <= 10:  # Assuming ordinal encoding for variables with limited unique values

    le = LabelEncoder()

    df[col] = le.fit_transform(df[col])

    label_encoders[col] = le

  else:

    df = pd.get_dummies(df, columns=[col], drop_first=True)

numeric_ranges = df[numeric_cols].describe()


missing_values = df.isnull().sum()


plt.figure(figsize=(12, 6))

sns.boxplot(data=df[numeric_cols])

plt.xticks(rotation=90)

plt.show()


numeric_stats = df[numeric_cols].agg(['mean', 'var'])
```

```
print("Data Types:\n", df.dtypes)
print("\nNumeric Ranges:\n", numeric_ranges)
print("\nMissing Values:\n", missing_values)
print("\nMean and Variance:\n", numeric_stats)
```

```
RangeIndex: 9172 entries, 0 to 9171
Data columns (total 31 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Record ID                  9172 non-null   int64
 1   age                        9172 non-null   int64
 2   sex                        9172 non-null   object
 3   on thyroxine               9172 non-null   object
 4   query on thyroxine         9172 non-null   object
 5   on antithyroid medication  9172 non-null   object
 6   sick                       9172 non-null   object
 7   pregnant                   9172 non-null   object
 8   thyroid surgery            9172 non-null   object
 9   I131 treatment             9172 non-null   object
 10  query hypothyroid          9172 non-null   object
 11  query hyperthyroid         9172 non-null   object
 12  lithium                    9172 non-null   object
 13  goitre                     9172 non-null   object
 14  tumor                      9172 non-null   object
 15  hypopituitary              9172 non-null   object
 16  psych                      9172 non-null   object
 17  TSH measured               9172 non-null   object
 18  TSH                        9172 non-null   object
 19  T3 measured                9172 non-null   object
 20  T3                         9172 non-null   object
```

```
 21  TT4 measured           9172 non-null   object
 22  TT4                    9172 non-null   object
 23  T4U measured           9172 non-null   object
 24  T4U                    9172 non-null   object
 25  FTI measured           9172 non-null   object
 26  FTI                    9172 non-null   object
 27  TBG measured           9172 non-null   object
 28  TBG                    9172 non-null   object
 29  referral source        9172 non-null   object
 30  Condition              9172 non-null   object
dtypes: int64(2), object(29)
memory usage: 2.2+ MB
None
Data Types:
 Record ID          int64
age                 int64
sex                 int64
on thyroxine        int64
query on thyroxine  int64
                    ...
Condition_OI        bool
Condition_P         bool
Condition_Q         bool
Condition_R         bool
Condition_S         bool
Length: 1361, dtype: object

Numeric Ranges:
          Record ID          age
count  9.172000e+03  9172.000000
mean   8.529473e+08    73.555822
```

std 7.581969e+06 1183.976718

min 8.408010e+08 1.000000

25% 8.504090e+08 37.000000

50% 8.510040e+08 55.000000

75% 8.607110e+08 68.000000

max 8.701190e+08 65526.000000

Missing Values:

Record ID 0

age 0

sex 0

on thyroxine 0

query on thyroxine 0

..

Condition_OI 0

Condition_P 0

Condition_Q 0

Condition_R 0

Condition_S 0

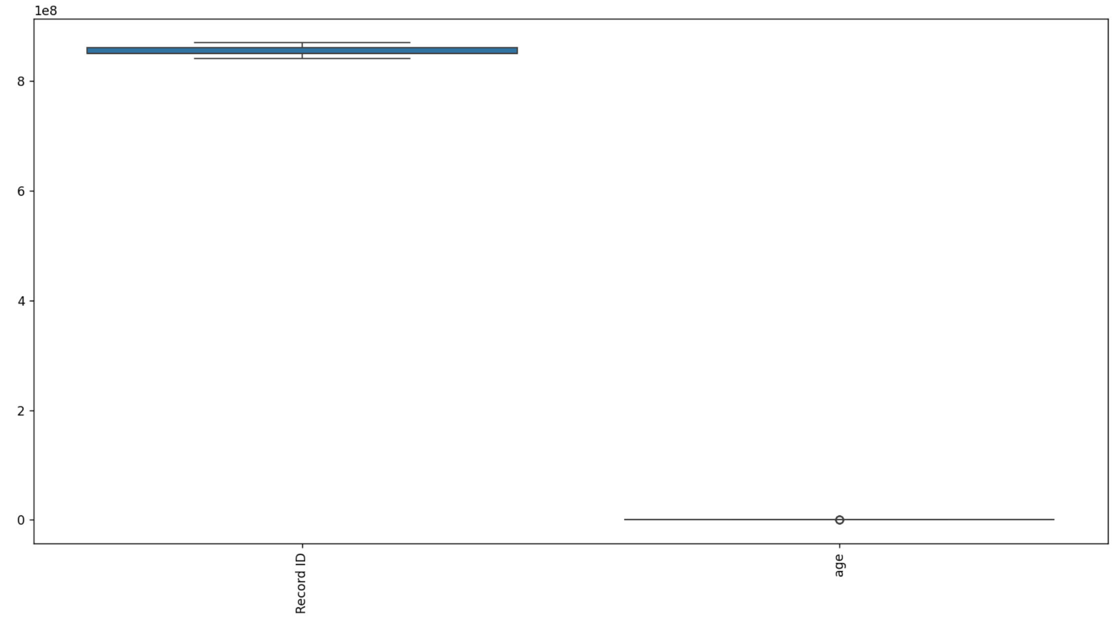
Length: 1361, dtype: int64

Mean and Variance:

Record ID age

mean 8.529473e+08 7.355582e+01

var 5.748625e+13 1.401801e+06



A6. Data Imputation: employ appropriate central tendencies to fill the missing values in the data variables. Employ following guidance. • Mean may be used when the attribute is numeric with no outliers • Median may be employed for attributes which are numeric and contain outliers • Mode may be employed for categorical attributes

```
categorical_cols = df.select_dtypes(include=['object']).columns

numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns


for col in df.columns:

    if df[col].isnull().sum() > 0:

        if col in numeric_cols:

            if np.any((df[col] - df[col].median()).abs() > 3 * df[col].std()):

                df[col].fillna(df[col].median(), inplace=True)

            else:

                df[col].fillna(df[col].mean(), inplace=True)

        else:

            df[col].fillna(df[col].mode()[0], inplace=True)


print("Missing values after imputation:\n", df.isnull().sum())
```

Missing values after imputation:

Record ID 0

age 0

sex 0

on thyroxine 0

query on thyroxine 0

..

Condition_OI 0

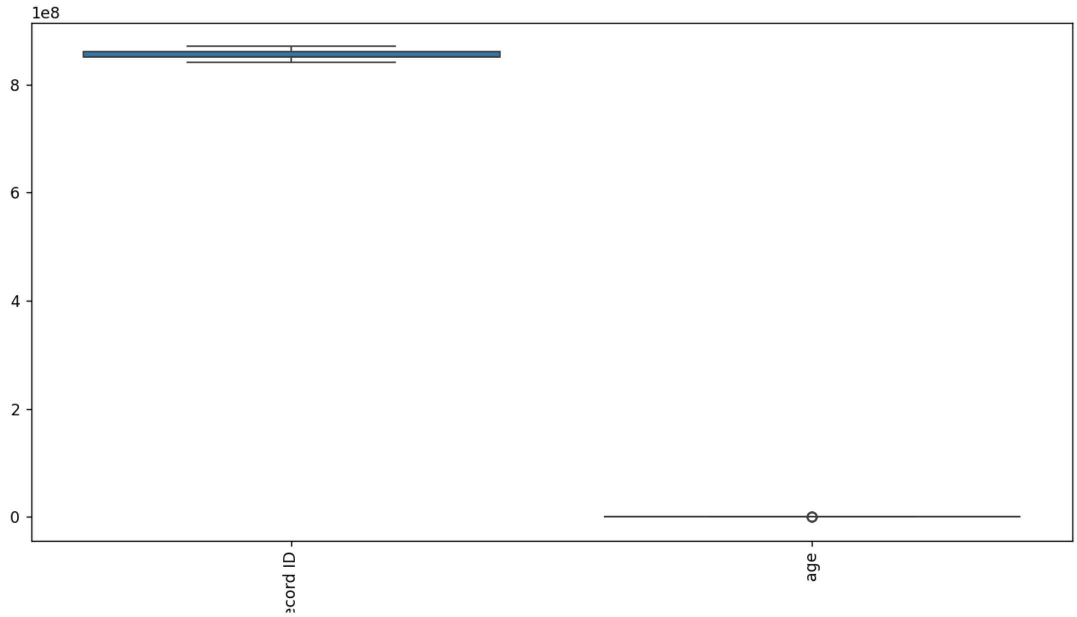
Condition_P 0

Condition_Q 0

Condition_R 0

Condition_S 0

Length: 1361, dtype: int64



A7. Data Normalization / Scaling: from the data study, identify the attributes which may need normalization. Employ appropriate normalization techniques to create normalized set of data.

from sklearn.preprocessing import MinMaxScaler, StandardScaler

```
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

outlier_cols = [col for col in numeric_cols if np.any((df[col] - df[col].median()).abs() > 3 * df[col].std())]

non_outlier_cols = list(set(numeric_cols) - set(outlier_cols))


scaler_standard = StandardScaler()

scaler_minmax = MinMaxScaler()


df[non_outlier_cols] = scaler_standard.fit_transform(df[non_outlier_cols])

df[outlier_cols] = scaler_minmax.fit_transform(df[outlier_cols])


print("Normalized data sample:\n", df.head())
```

```
Normalized data sample:
   Record ID      age      sex  on thyroxine  query on thyroxine  ...  Condition_OI  Condition_P  Condition_Q  Condition_R  Condition_S
0  -1.602090  0.000427 -0.526833    -0.395384             0.0  ...       False       False       False       False       False
1  -1.602090  0.000427 -0.526833    -0.395384             0.0  ...       False       False       False       False       False
2  -1.602086  0.000610 -0.526833    -0.395384             0.0  ...       False       False       False       False       False
3  -1.601822  0.000534 -0.526833    -0.395384             0.0  ...       False       False       False       False       False
4  -1.601822  0.000473 -0.526833    -0.395384             0.0  ...       False       False       False       False        True
```

A8.Take the first 2 observation vectors from the dataset. Consider only the attributes (direct or derived) with binary values for these vectors (ignore other attributes). Calculate the Jaccard Coefficient (JC) and Simple Matching Coefficient (SMC) between the document vectors. Use first vector for each document for this. Compare the values for JC and SMC and judge the appropriateness of each of them.

```
import numpy as np
```

```
binary_cols = [col for col in df.columns if set(df[col].unique()).issubset({0, 1})]
vector1 = df.iloc[0][binary_cols].values
vector2 = df.iloc[1][binary_cols].values

f11 = np.sum((vector1 == 1) & (vector2 == 1))
f00 = np.sum((vector1 == 0) & (vector2 == 0))
f01 = np.sum((vector1 == 0) & (vector2 == 1))
f10 = np.sum((vector1 == 1) & (vector2 == 0))

JC = f11 / (f01 + f10 + f11)
SMC = (f11 + f00) / (f00 + f01 + f10 + f11)

print(f"Jaccard Coefficient (JC): {JC}")
print(f"Simple Matching Coefficient (SMC): {SMC}")
```

```
Jaccard Coefficient (JC): 0.38461538461538464
Simple Matching Coefficient (SMC): 0.9940959409594096
```

A9. Cosine Similarity Measure: Now take the complete vectors for these two observations (including all the attributes). Calculate the Cosine similarity between the documents by using the second feature vector for each document.

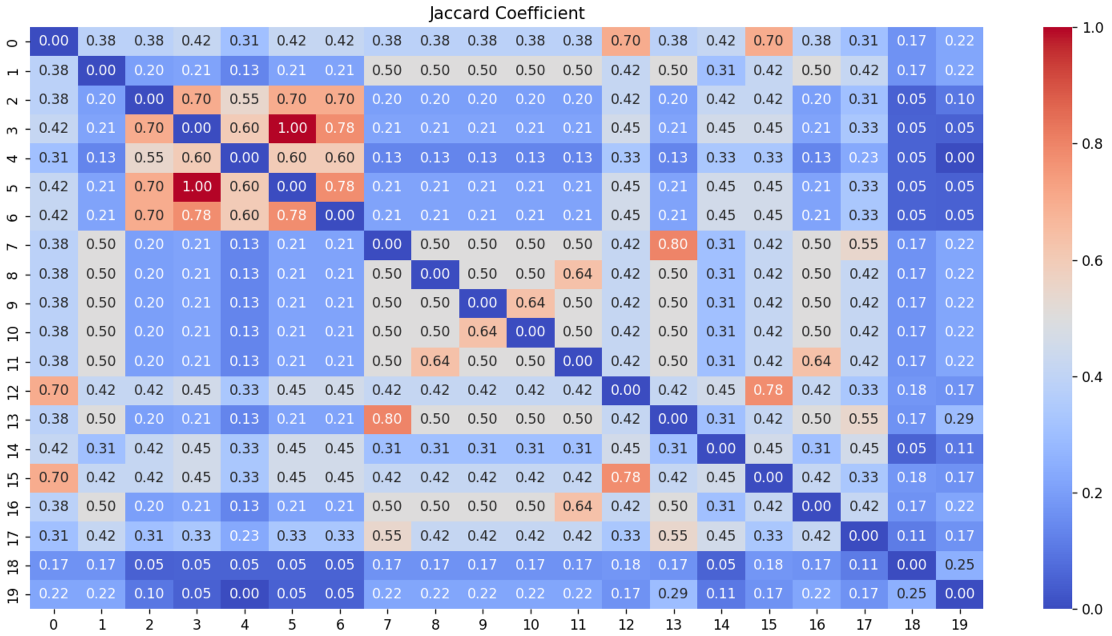
```
from sklearn.metrics.pairwise import cosine_similarity

vector1 = df.iloc[0].values.reshape(1, -1)
vector2 = df.iloc[1].values.reshape(1, -1)

cosine_sim = cosine_similarity(vector1, vector2)[0][0]

print(f"Cosine Similarity: {cosine_sim}")
```

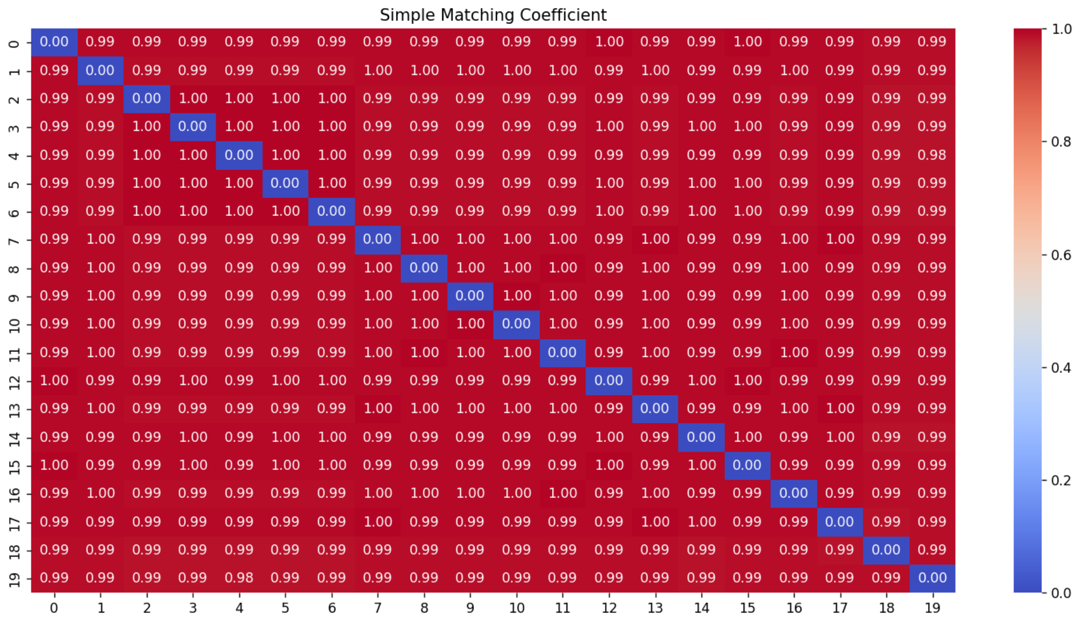
```
Cosine Similarity: 0.5547930327355779
```

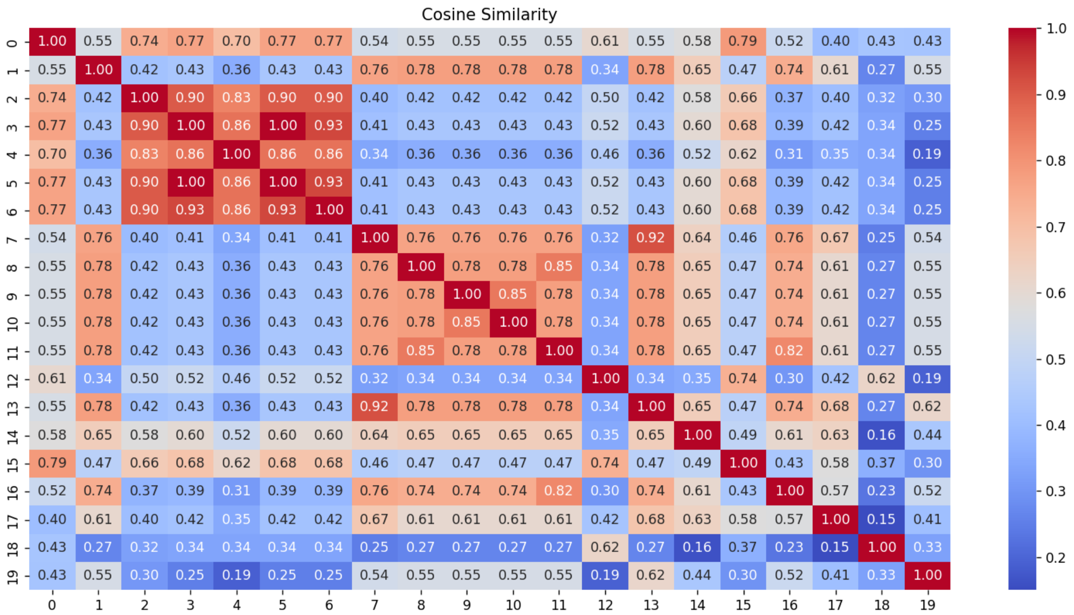
A10. Heatmap Plot: Consider the first 20 observation vectors. Calculate the JC, SMC and COS between the pairs of vectors for these 20 vectors. Employ similar strategies for coefficient calculation as in A4 & A5. Employ a heatmap plot to visualize the similarities.

```
import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import cosine_similarity


subset_df = df.iloc[:20]


binary_cols = [col for col in df.columns if set(df[col].unique()).issubset({0, 1})]


def compute_jc_smc(matrix):

    n = len(matrix)

    jc_matrix = np.zeros((n, n))

    smc_matrix = np.zeros((n, n))


    for i in range(n):

        for j in range(n):

            if i != j:

                f11 = np.sum((matrix[i] == 1) & (matrix[j] == 1))

                f00 = np.sum((matrix[i] == 0) & (matrix[j] == 0))

                f01 = np.sum((matrix[i] == 0) & (matrix[j] == 1))

                f10 = np.sum((matrix[i] == 1) & (matrix[j] == 0))


                jc_matrix[i, j] = f11 / (f01 + f10 + f11) if (f01 + f10 + f11) > 0 else 0

                smc_matrix[i, j] = (f11 + f00) / (f00 + f01 + f10 + f11) if (f00 + f01 + f10 + f11) > 0 else 0


    return jc_matrix, smc_matrix


binary_matrix = subset_df[binary_cols].values
```

```
jc_matrix, smc_matrix = compute_jc_smc(binary_matrix)

cos_matrix = cosine_similarity(subset_df.values)

plt.figure(figsize=(8, 6))
sns.heatmap(jc_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Jaccard Coefficient")
plt.show()

plt.figure(figsize=(8, 6))
sns.heatmap(smc_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Simple Matching Coefficient")
plt.show()

plt.figure(figsize=(8, 6))
sns.heatmap(cos_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Cosine Similarity")
plt.show()
```


Jaccard Coefficient
0.00 0.38 0.38 0.42 0.31 0.42 0.42 0.38 0.38 0.38 0.38 0.38 0.70 0.38 0.42 0.70 0.38 0.31 0.17 0.22
0.38 0.00 0.20 0.21 0.13 0.21 0.21 0.50 0.50 0.50 0.50 0.50 0.42 0.50 0.31 0.42 0.50 0.42 0.17 0.22
0.38 0.20 0.00 0.70 0.55 0.70 0.70 0.20 0.20 0.20 0.20 0.20 0.42 0.20 0.42 0.42 0.20 0.31 0.05 0.10
0.42 0.21 0.70 0.00 0.60 1.00 0.78 0.21 0.21 0.21 0.21 0.21 0.45 0.21 0.45 0.45 0.21 0.33 0.05 0.05
0.31 0.13 0.55 0.60 0.00 0.60 0.60 0.13 0.13 0.13 0.13 0.13 0.33 0.13 0.33 0.33 0.13 0.23 0.05 0.00
0.42 0.21 0.70 1.00 0.60 0.00 0.78 0.21 0.21 0.21 0.21 0.21 0.45 0.21 0.45 0.45 0.21 0.33 0.05 0.05
0.42 0.21 0.70 0.78 0.60 0.78 0.00 0.21 0.21 0.21 0.21 0.21 0.45 0.21 0.45 0.45 0.21 0.33 0.05 0.05
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.00 0.50 0.50 0.50 0.50 0.42 0.80 0.31 0.42 0.50 0.55 0.17 0.22
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.50 0.00 0.50 0.50 0.64 0.42 0.50 0.31 0.42 0.50 0.42 0.17 0.22
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.50 0.50 0.00 0.64 0.50 0.42 0.50 0.31 0.42 0.50 0.42 0.17 0.22
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.50 0.50 0.64 0.00 0.50 0.42 0.50 0.31 0.42 0.50 0.42 0.17 0.22
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.50 0.64 0.50 0.50 0.00 0.42 0.50 0.31 0.42 0.64 0.42 0.17 0.22
0.70 0.42 0.42 0.45 0.33 0.45 0.45 0.42 0.42 0.42 0.42 0.42 0.00 0.42 0.45 0.78 0.42 0.33 0.18 0.17
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.80 0.50 0.50 0.50 0.50 0.42 0.00 0.31 0.42 0.50 0.55 0.17 0.29
0.42 0.31 0.42 0.45 0.33 0.45 0.45 0.31 0.31 0.31 0.31 0.31 0.45 0.31 0.00 0.45 0.31 0.45 0.05 0.11
0.70 0.42 0.42 0.45 0.33 0.45 0.45 0.42 0.42 0.42 0.42 0.42 0.78 0.42 0.45 0.00 0.42 0.33 0.18 0.17
0.38 0.50 0.20 0.21 0.13 0.21 0.21 0.50 0.50 0.50 0.50 0.64 0.42 0.50 0.31 0.42 0.00 0.42 0.17 0.22
0.31 0.42 0.31 0.33 0.23 0.33 0.33 0.55 0.42 0.42 0.42 0.42 0.33 0.55 0.45 0.33 0.42 0.00 0.11 0.17
0.17 0.17 0.05 0.05 0.05 0.05 0.05 0.17 0.17 0.17 0.17 0.17 0.18 0.17 0.05 0.18 0.17 0.11 0.00 0.25
0.22 0.22 0.10 0.05 0.00 0.05 0.05 0.22 0.22 0.22 0.22 0.22 0.17 0.29 0.11 0.17 0.22 0.17 0.25 0.00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1.0
0.8
0.6
0.4
0.2
0.0


Simple Matching Coefficient
0.00 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 1.00 0.99 0.99 1.00 0.99 0.99 0.99 0.99
0.99 0.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 1.00 1.00 1.00 0.99 1.00 0.99 0.99 1.00 0.99 0.99 0.99
0.99 0.99 0.00 1.00 1.00 1.00 1.00 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99
0.99 0.99 1.00 0.00 1.00 1.00 1.00 0.99 0.99 0.99 0.99 0.99 1.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99
0.99 0.99 1.00 1.00 0.00 1.00 1.00 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.98
0.99 0.99 1.00 1.00 1.00 0.00 1.00 0.99 0.99 0.99 0.99 0.99 1.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99
0.99 0.99 1.00 1.00 1.00 1.00 0.00 0.99 0.99 0.99 0.99 0.99 1.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 0.00 1.00 1.00 1.00 1.00 0.99 1.00 0.99 0.99 1.00 1.00 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 0.00 1.00 1.00 1.00 0.99 1.00 0.99 0.99 1.00 0.99 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 0.00 1.00 1.00 0.99 1.00 0.99 0.99 1.00 0.99 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 1.00 0.00 1.00 0.99 1.00 0.99 0.99 1.00 0.99 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 1.00 1.00 0.00 0.99 1.00 0.99 0.99 1.00 0.99 0.99 0.99
1.00 0.99 0.99 1.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99 0.99 0.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 1.00 1.00 1.00 0.99 0.00 0.99 0.99 1.00 1.00 0.99 0.99
0.99 0.99 0.99 1.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99 0.99 1.00 0.99 0.00 1.00 0.99 1.00 0.99 0.99
1.00 0.99 0.99 1.00 0.99 1.00 1.00 0.99 0.99 0.99 0.99 0.99 1.00 0.99 1.00 0.00 0.99 0.99 0.99 0.99
0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 1.00 1.00 1.00 0.99 1.00 0.99 0.99 0.00 0.99 0.99 0.99
0.99 0.99 0.99 0.99 0.99 0.99 0.99 1.00 0.99 0.99 0.99 0.99 0.99 1.00 1.00 0.99 0.99 0.00 0.99 0.99
0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.00 0.99
0.99 0.99 0.99 0.99 0.98 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.99 0.00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1.0
0.8
0.6
0.4
0.2
0.0


Cosine Similarity
0 1.00 0.55 0.74 0.77 0.70 0.77 0.77 0.54 0.55 0.55 0.55 0.55 0.61 0.55 0.58 0.79 0.52 0.40 0.43 0.43
1 0.55 1.00 0.42 0.43 0.36 0.43 0.43 0.76 0.78 0.78 0.78 0.78 0.34 0.78 0.65 0.47 0.74 0.61 0.27 0.55
2 0.74 0.42 1.00 0.90 0.83 0.90 0.90 0.40 0.42 0.42 0.42 0.42 0.50 0.42 0.58 0.66 0.37 0.40 0.32 0.30
3 0.77 0.43 0.90 1.00 0.86 1.00 0.93 0.41 0.43 0.43 0.43 0.43 0.52 0.43 0.60 0.68 0.39 0.42 0.34 0.25
4 0.70 0.36 0.83 0.86 1.00 0.86 0.86 0.34 0.36 0.36 0.36 0.36 0.46 0.36 0.52 0.62 0.31 0.35 0.34 0.19
5 0.77 0.43 0.90 1.00 0.86 1.00 0.93 0.41 0.43 0.43 0.43 0.43 0.52 0.43 0.60 0.68 0.39 0.42 0.34 0.25
6 0.77 0.43 0.90 0.93 0.86 0.93 1.00 0.41 0.43 0.43 0.43 0.43 0.52 0.43 0.60 0.68 0.39 0.42 0.34 0.25
7 0.54 0.76 0.40 0.41 0.34 0.41 0.41 1.00 0.76 0.76 0.76 0.76 0.32 0.92 0.64 0.46 0.76 0.67 0.25 0.54
8 0.55 0.78 0.42 0.43 0.36 0.43 0.43 0.76 1.00 0.78 0.78 0.85 0.34 0.78 0.65 0.47 0.74 0.61 0.27 0.55
9 0.55 0.78 0.42 0.43 0.36 0.43 0.43 0.76 0.78 1.00 0.85 0.78 0.34 0.78 0.65 0.47 0.74 0.61 0.27 0.55
10 0.55 0.78 0.42 0.43 0.36 0.43 0.43 0.76 0.78 0.85 1.00 0.78 0.34 0.78 0.65 0.47 0.74 0.61 0.27 0.55
11 0.55 0.78 0.42 0.43 0.36 0.43 0.43 0.76 0.85 0.78 0.78 1.00 0.34 0.78 0.65 0.47 0.82 0.61 0.27 0.55
12 0.61 0.34 0.50 0.52 0.46 0.52 0.52 0.32 0.34 0.34 0.34 0.34 1.00 0.34 0.35 0.74 0.30 0.42 0.62 0.19
13 0.55 0.78 0.42 0.43 0.36 0.43 0.43 0.92 0.78 0.78 0.78 0.78 0.34 1.00 0.65 0.47 0.74 0.68 0.27 0.62
14 0.58 0.65 0.58 0.60 0.52 0.60 0.60 0.64 0.65 0.65 0.65 0.65 0.35 0.65 1.00 0.49 0.61 0.63 0.16 0.44
15 0.79 0.47 0.66 0.68 0.62 0.68 0.68 0.46 0.47 0.47 0.47 0.47 0.74 0.47 0.49 1.00 0.43 0.58 0.37 0.30
16 0.52 0.74 0.37 0.39 0.31 0.39 0.39 0.76 0.74 0.74 0.74 0.82 0.30 0.74 0.61 0.43 1.00 0.57 0.23 0.52
17 0.40 0.61 0.40 0.42 0.35 0.42 0.42 0.67 0.61 0.61 0.61 0.61 0.42 0.68 0.63 0.58 0.57 1.00 0.15 0.41
18 0.43 0.27 0.32 0.34 0.34 0.34 0.34 0.25 0.27 0.27 0.27 0.27 0.62 0.27 0.16 0.37 0.23 0.15 1.00 0.33
19 0.43 0.55 0.30 0.25 0.19 0.25 0.25 0.54 0.55 0.55 0.55 0.55 0.19 0.62 0.44 0.30 0.52 0.41 0.33 1.00
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
1.0 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2