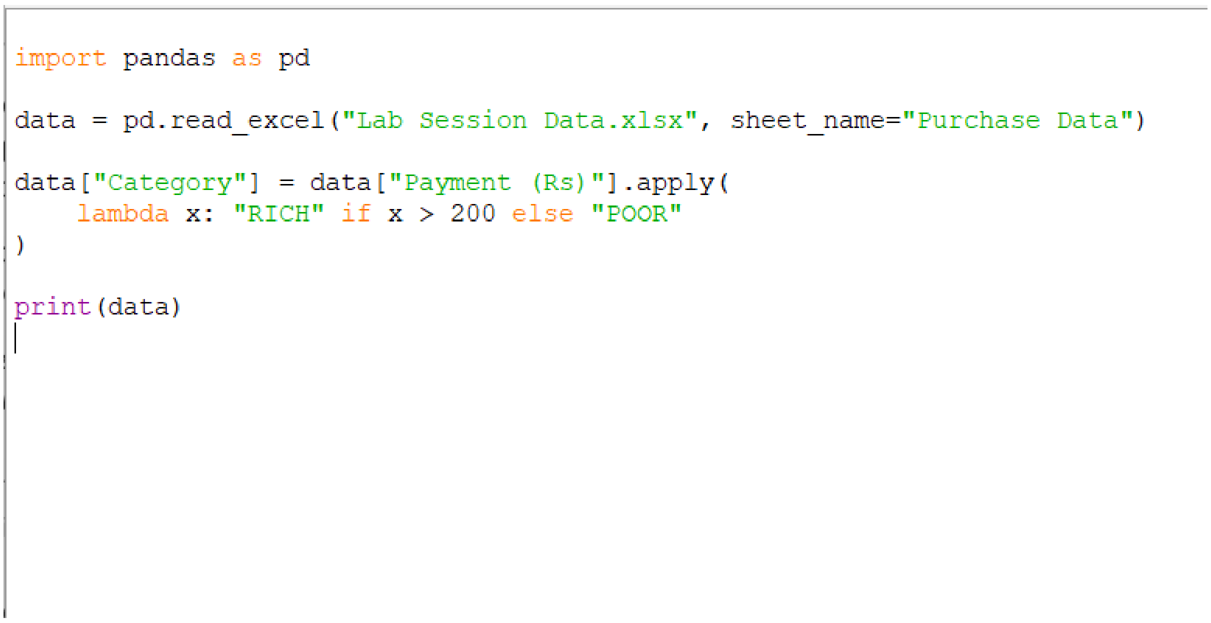
BL.SC.U4AIE24116
LAB-2

Q1

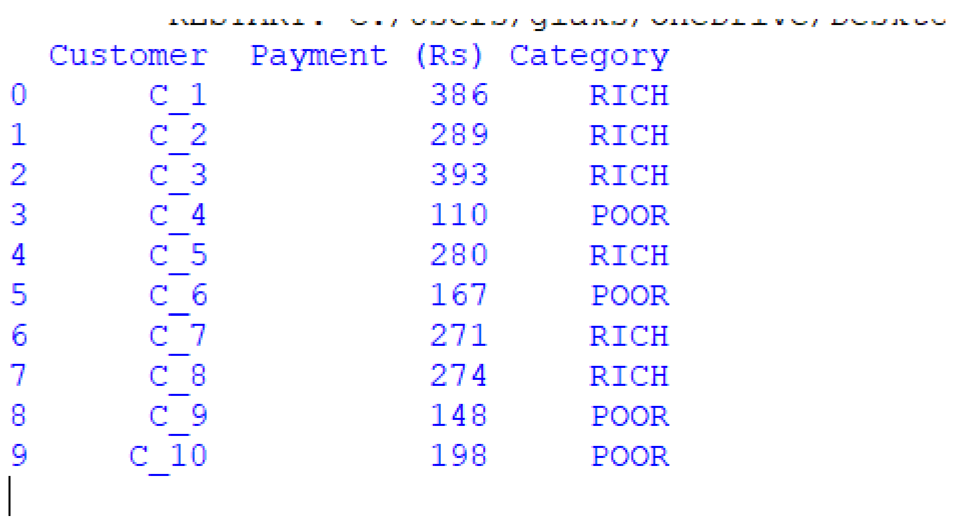
```
import numpy as np
X = np.array([
    [20, 6, 2],
    [16, 3, 6],
    [27, 6, 2],
    [19, 1, 2],
    [24, 4, 2],
    [22, 1, 5],
    [15, 4, 2],
    [18, 4, 2],
    [21, 1, 4],
    [16, 2, 4]
])

cost= np.array([386, 289, 393, 110, 280, 167, 271, 274, 148, 198])

dimensionality = X.shape[1]
num_vectors = X.shape[0]

print("Dimensionality of vector space:", dimensionality)
print("Number of vectors:", num_vectors)



rank_X = np.linalg.matrix_rank(X)
print("Rank of feature matrix:", rank_X)



X_pseudo_inverse = np.linalg.pinv(X)
cost = X_pseudo_inverse @ cost


print("Cost of Candies (Rs per unit):", round(cost[0], 2))
print("Cost of Mangoes (Rs per Kg):", round(cost[1], 2))
print("Cost of Milk Packets (Rs per unit):", round(cost[2], 2))
```

```
-----------------------------------------------------
Dimensionality of vector space: 3
Number of vectors: 10
Rank of feature matrix: 3

Cost of each product:
Cost of Candies (Rs per unit): 1.0
Cost of Mangoes (Rs per Kg): 55.0
Cost of Milk Packets (Rs per unit): 18.0
```

2nd



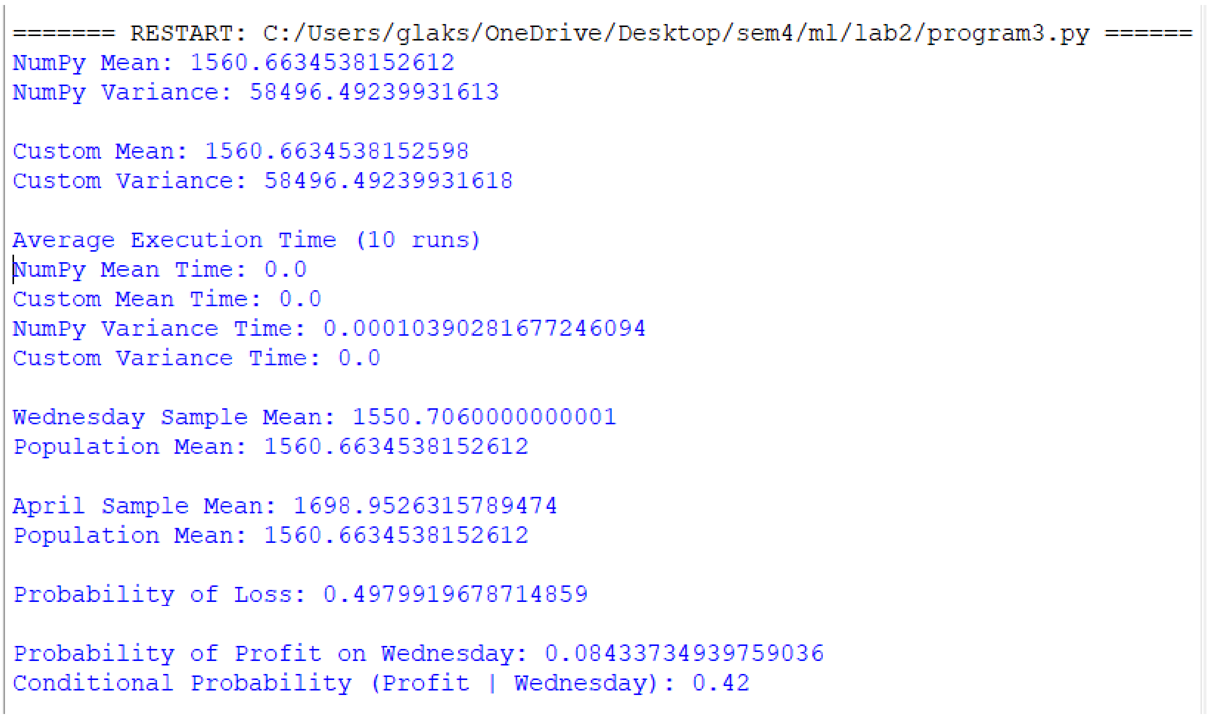
```python
import pandas as pd

data = pd.read_excel("Lab Session Data.xlsx", sheet_name="Purchase Data")

data["Category"] = data["Payment (Rs)"].apply(
    lambda x: "RICH" if x > 200 else "POOR"
)

print(data)
```



```
  Customer  Payment (Rs) Category
0      C_1           386     RICH
1      C_2           289     RICH
2      C_3           393     RICH
3      C_4           110     POOR
4      C_5           280     RICH
5      C_6           167     POOR
6      C_7           271     RICH
7      C_8           274     RICH
8      C_9           148     POOR
9     C_10           198     POOR
```

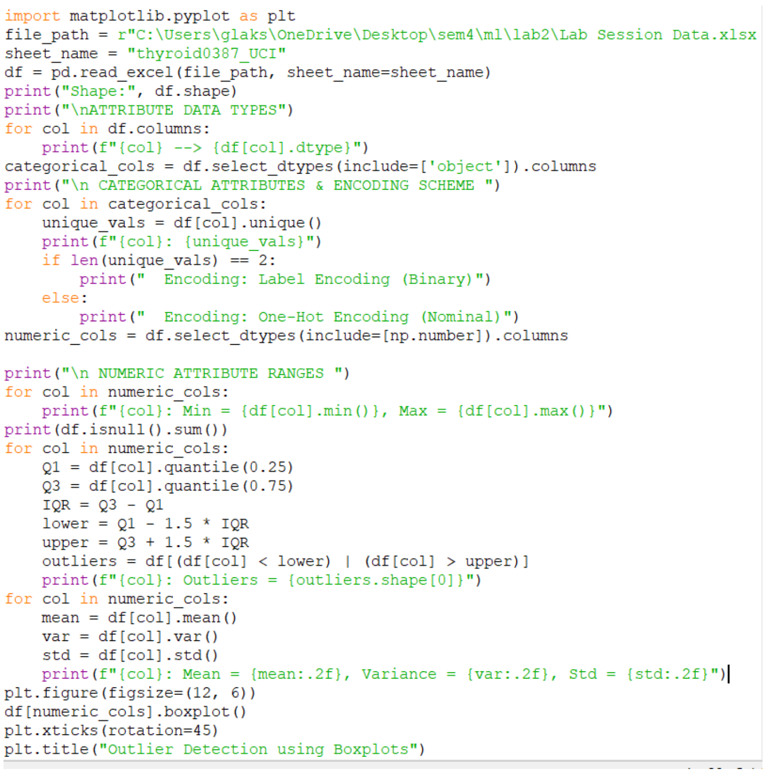
Q-3

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import time
file_path = "Lab Session Data.xlsx"
sheet_name = "IRCTC Stock Price"
df = pd.read_excel(file_path, sheet_name=sheet_name)
df['Date'] = pd.to_datetime(df['Date'])
df['Day'] = df['Date'].dt.day_name()
df['Month'] = df['Date'].dt.month_name()
price = df['Price']
chg = df['Chg%']
np_mean = np.mean(price)
np_var = np.var(price)
print("NumPy Mean:", np_mean)
print("NumPy Variance:", np_var)
def custom_mean(data):
    return sum(data) / len(data)
def custom_variance(data):
    mu = custom_mean(data)
    return sum((x - mu) ** 2 for x in data) / len(data)
cust_mean = custom_mean(price)
cust_var = custom_variance(price)
print("\nCustom Mean:", cust_mean)
print("Custom Variance:", cust_var)
def time_function(func, data, runs=10):
    times = []
    for _ in range(runs):
        start = time.time()
        func(data)
        end = time.time()
        times.append(end - start)
    return sum(times) / runs
np_mean_time = time_function(np.mean, price)
cust_mean_time = time_function(custom_mean, price)
np_var_time = time_function(np.var, price)
cust_var_time = time_function(custom_variance, price)
print("\nAverage Execution Time (10 runs)")
print("NumPy Mean Time:", np_mean_time)
print("Custom Mean Time:", cust_mean_time)
print("NumPy Variance Time:", np_var_time)
print("Custom Variance Time:", cust_var_time)
wed_price = df[df['Day'] == 'Wednesday']['Price']
wed_mean = wed_price.mean()
print("\nWednesday Sample Mean:", wed_mean)
print("Population Mean:", np_mean)
apr_price = df[df['Month'] == 'April']['Price']
```

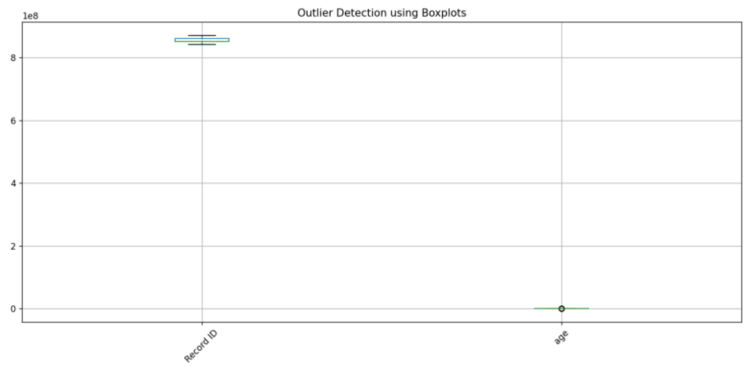


```
======= RESTART: C:/Users/glaks/OneDrive/Desktop/sem4/ml/lab2/program3.py ======
NumPy Mean: 1560.6634538152612
NumPy Variance: 58496.49239931613

Custom Mean: 1560.6634538152598
Custom Variance: 58496.49239931618

Average Execution Time (10 runs)
NumPy Mean Time: 0.0
Custom Mean Time: 0.0
NumPy Variance Time: 0.00010390281677246094
Custom Variance Time: 0.0

Wednesday Sample Mean: 1550.7060000000001
Population Mean: 1560.6634538152612

April Sample Mean: 1698.9526315789474
Population Mean: 1560.6634538152612

Probability of Loss: 0.4979919678714859

Probability of Profit on Wednesday: 0.08433734939759036
Conditional Probability (Profit | Wednesday): 0.42
```

Q-4



```
import matplotlib.pyplot as plt
file_path = r"C:\Users\glaks\OneDrive\Desktop\sem4\ml\lab2\Lab Session Data.xlsx
sheet_name = "thyroid0387_UCI"
df = pd.read_excel(file_path, sheet_name=sheet_name)
print("Shape:", df.shape)
print("\nATTRIBUTE DATA TYPES")
for col in df.columns:
    print(f"{col} --> {df[col].dtype}")
categorical_cols = df.select_dtypes(include=['object']).columns
print("\n CATEGORICAL ATTRIBUTES & ENCODING SCHEME ")
for col in categorical_cols:
    unique_vals = df[col].unique()
    print(f"{col}: {unique_vals}")
    if len(unique_vals) == 2:
        print("  Encoding: Label Encoding (Binary)")
    else:
        print("  Encoding: One-Hot Encoding (Nominal)")
numeric_cols = df.select_dtypes(include=[np.number]).columns

print("\n NUMERIC ATTRIBUTE RANGES ")
for col in numeric_cols:
    print(f"{col}: Min = {df[col].min()}, Max = {df[col].max()}")
print(df.isnull().sum())
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    outliers = df[(df[col] < lower) | (df[col] > upper)]
    print(f"{col}: Outliers = {outliers.shape[0]}")
for col in numeric_cols:
    mean = df[col].mean()
    var = df[col].var()
    std = df[col].std()
    print(f"{col}: Mean = {mean:.2f}, Variance = {var:.2f}, Std = {std:.2f}")
plt.figure(figsize=(12, 6))
df[numeric_cols].boxplot()
plt.xticks(rotation=45)
plt.title("Outlier Detection using Boxplots")
```



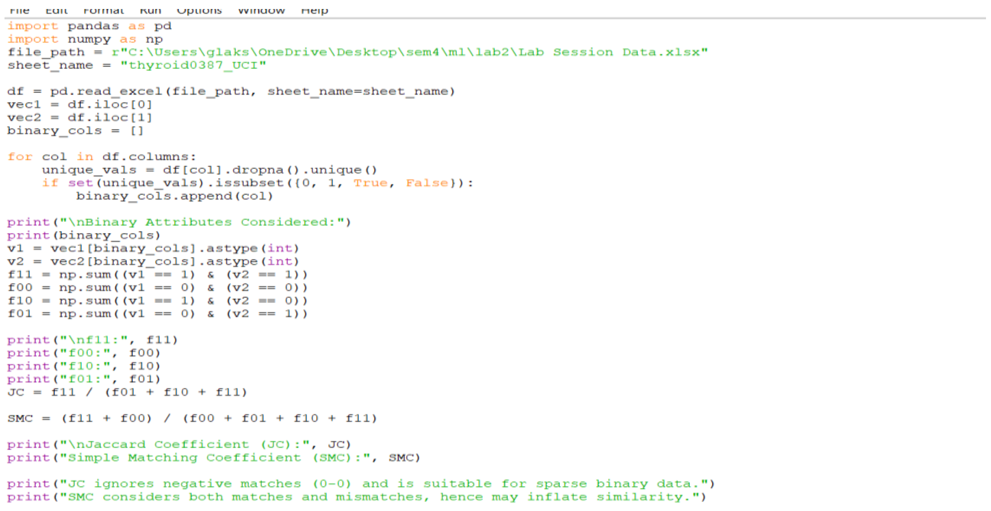
Record ID: Outliers = 0

age: Outliers = 4

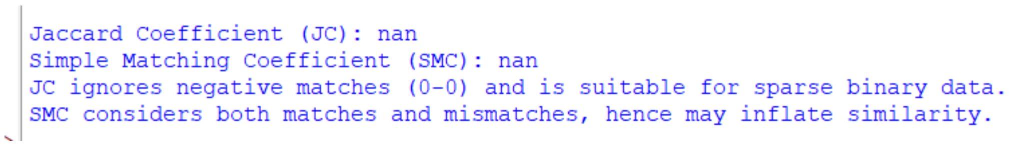
Record ID: Mean = 852947346.61, Variance = 57486250586150.28, Std = 7581968.78

age: Mean = 73.56, Variance = 1401800.87, Std = 1183.98

5Q

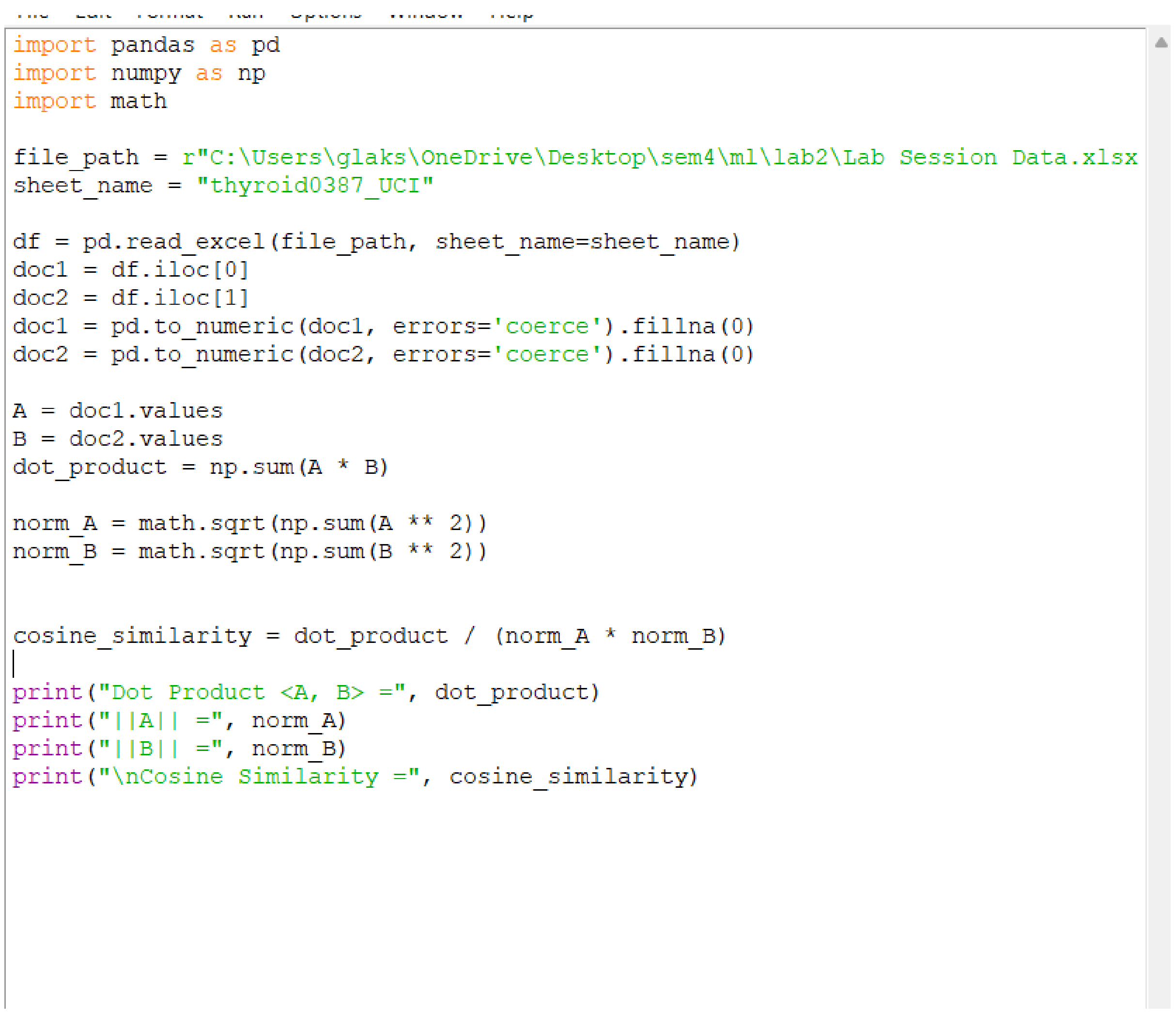


```
import pandas as pd
import numpy as np
file_path = r"C:\Users\glaks\OneDrive\Desktop\sem4\ml\lab2\Lab Session Data.xlsx"
sheet_name = "thyroid0387_UCI"

df = pd.read_excel(file_path, sheet_name=sheet_name)
vec1 = df.iloc[0]
vec2 = df.iloc[1]
binary_cols = []

for col in df.columns:
    unique_vals = df[col].dropna().unique()
    if set(unique_vals).issubset({0, 1, True, False}):
        binary_cols.append(col)

print("\nBinary Attributes Considered:")
print(binary_cols)
v1 = vec1[binary_cols].astype(int)
v2 = vec2[binary_cols].astype(int)
f11 = np.sum((v1 == 1) & (v2 == 1))
f00 = np.sum((v1 == 0) & (v2 == 0))
f10 = np.sum((v1 == 1) & (v2 == 0))
f01 = np.sum((v1 == 0) & (v2 == 1))

print("\nf11:", f11)
print("f00:", f00)
print("f10:", f10)
print("f01:", f01)
JC = f11 / (f01 + f10 + f11)

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

print("\nJaccard Coefficient (JC):", JC)
print("Simple Matching Coefficient (SMC):", SMC)

print("JC ignores negative matches (0-0) and is suitable for sparse binary data.")
print("SMC considers both matches and mismatches, hence may inflate similarity.")
```



```
Jaccard Coefficient (JC): nan
Simple Matching Coefficient (SMC): nan
JC ignores negative matches (0-0) and is suitable for sparse binary data.
SMC considers both matches and mismatches, hence may inflate similarity.
```

Q-6

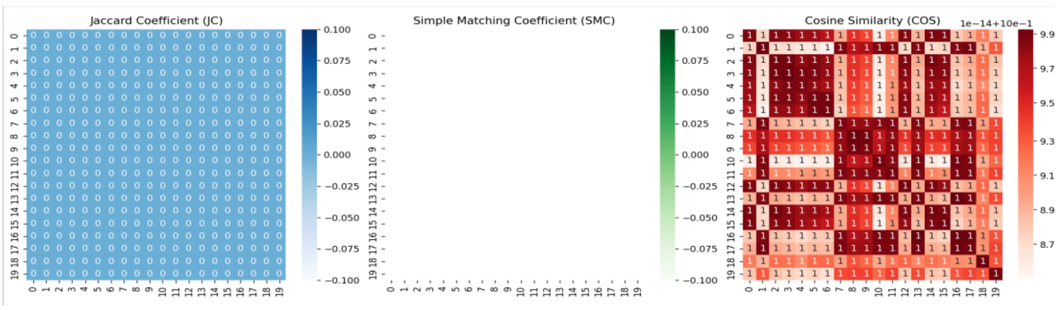


```
import pandas as pd
import numpy as np
import math

file_path = r"C:\Users\glaks\OneDrive\Desktop\sem4\ml\lab2\Lab Session Data.xlsx
sheet_name = "thyroid0387_UCI"

df = pd.read_excel(file_path, sheet_name=sheet_name)
doc1 = df.iloc[0]
doc2 = df.iloc[1]
doc1 = pd.to_numeric(doc1, errors='coerce').fillna(0)
doc2 = pd.to_numeric(doc2, errors='coerce').fillna(0)

A = doc1.values
B = doc2.values
dot_product = np.sum(A * B)

norm_A = math.sqrt(np.sum(A ** 2))
norm_B = math.sqrt(np.sum(B ** 2))


cosine_similarity = dot_product / (norm_A * norm_B)

print("Dot Product <A, B> =", dot_product)
print("||A|| =", norm_A)
print("||B|| =", norm_B)
print("\nCosine Similarity =", cosine_similarity)
```

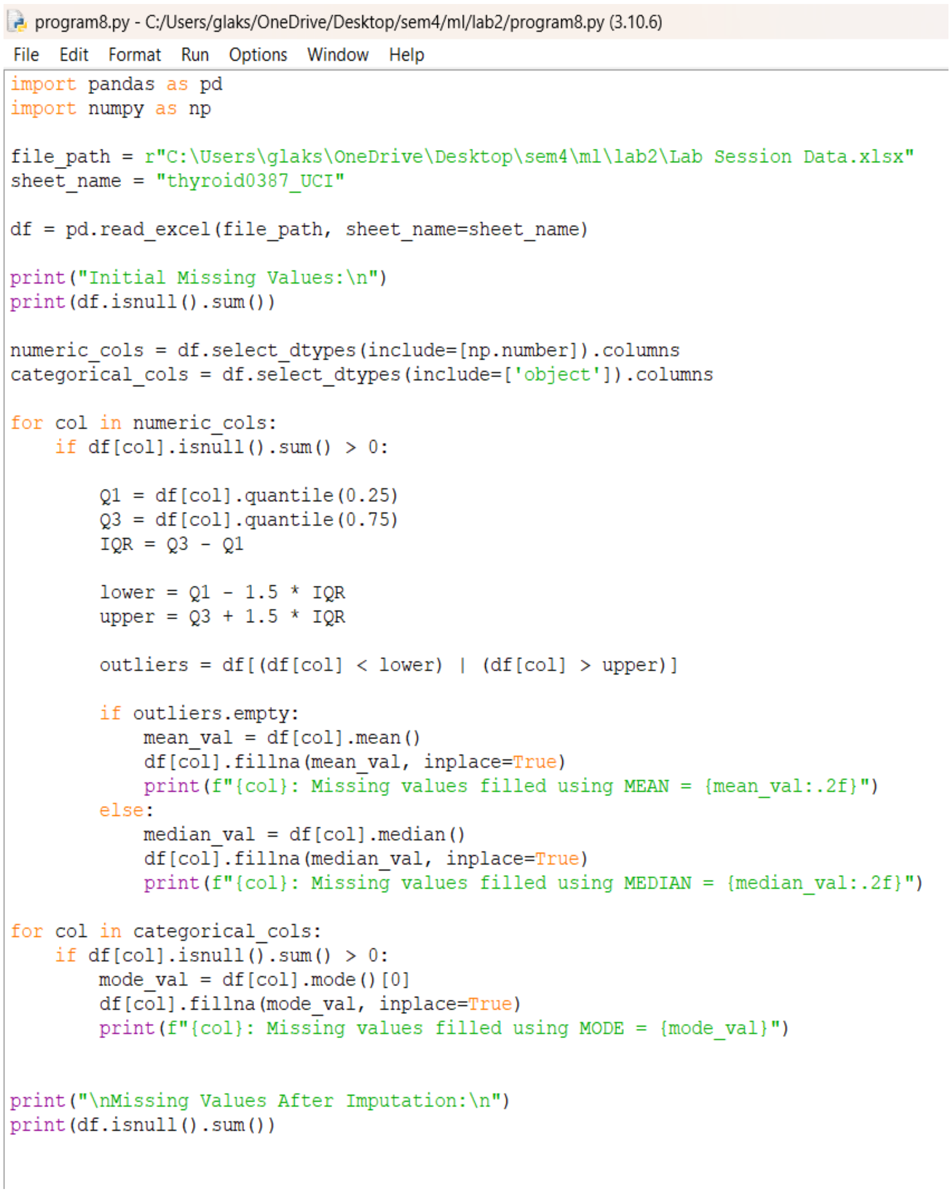
```
Dot Product <A, B> = 7.069463443026281e+17
||A|| = 840801013.0000006
||B|| = 840801014.0000103

Cosine Similarity = 0.9999999999999885
```

7Q

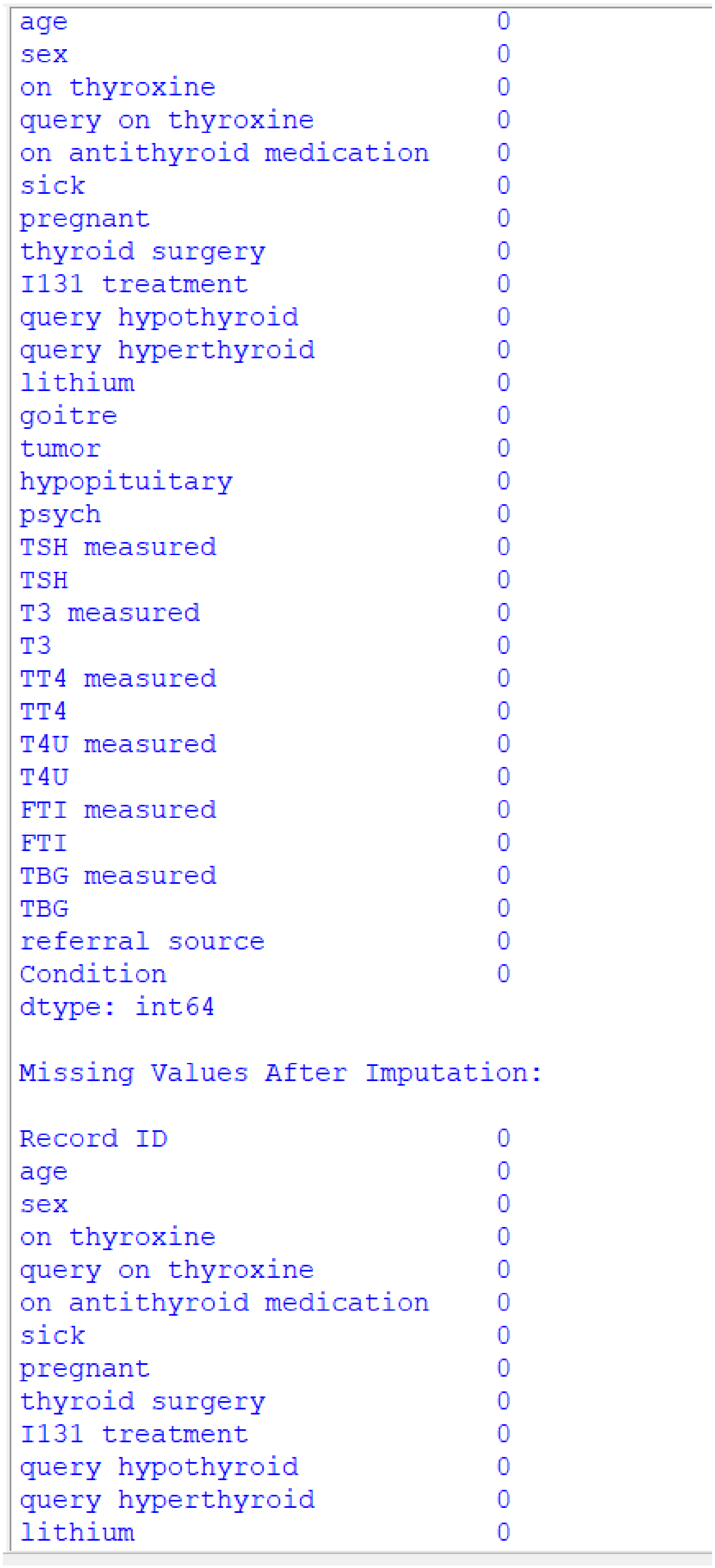
```
file_path = r"C:\Users\glaks\OneDrive\Desktop\sem4\ml\lab2\Lab Session Data.xlsx"
sheet_name = "thyroid0387_UCI"

df = pd.read_excel(file_path, sheet_name=sheet_name)
data_20 = df.iloc[:20]
binary_cols = []
for col in df.columns:
    unique_vals = df[col].dropna().unique()
    if set(unique_vals).issubset({0, 1, True, False}):
        binary_cols.append(col)

binary_data = data_20[binary_cols].astype(int)
numeric_data = data_20.apply(pd.to_numeric, errors='coerce').fillna(0)

n = 20
JC = np.zeros((n, n))
SMC = np.zeros((n, n))
COS = np.zeros((n, n))

for i in range(n):
    for j in range(n):
        v1 = binary_data.iloc[i]
        v2 = binary_data.iloc[j]

        f11 = np.sum((v1 == 1) & (v2 == 1))
        f00 = np.sum((v1 == 0) & (v2 == 0))
        f10 = np.sum((v1 == 1) & (v2 == 0))
        f01 = np.sum((v1 == 0) & (v2 == 1))

        JC[i, j] = f11 / (f11 + f10 + f01) if (f11 + f10 + f01) != 0 else 0
        SMC[i, j] = (f11 + f00) / (f11 + f00 + f10 + f01)
        A = numeric_data.iloc[i].values
        B = numeric_data.iloc[j].values

        dot = np.sum(A * B)
        normA = math.sqrt(np.sum(A ** 2))
        normB = math.sqrt(np.sum(B ** 2))

        COS[i, j] = dot / (normA * normB) if normA != 0 and normB != 0 else 0

plt.figure(figsize=(18, 5))

plt.subplot(1, 3, 1)
sns.heatmap(JC, annot=True, cmap="Blues")
plt.title("Jaccard Coefficient (JC)")

plt.subplot(1, 3, 2)
```



Q-8

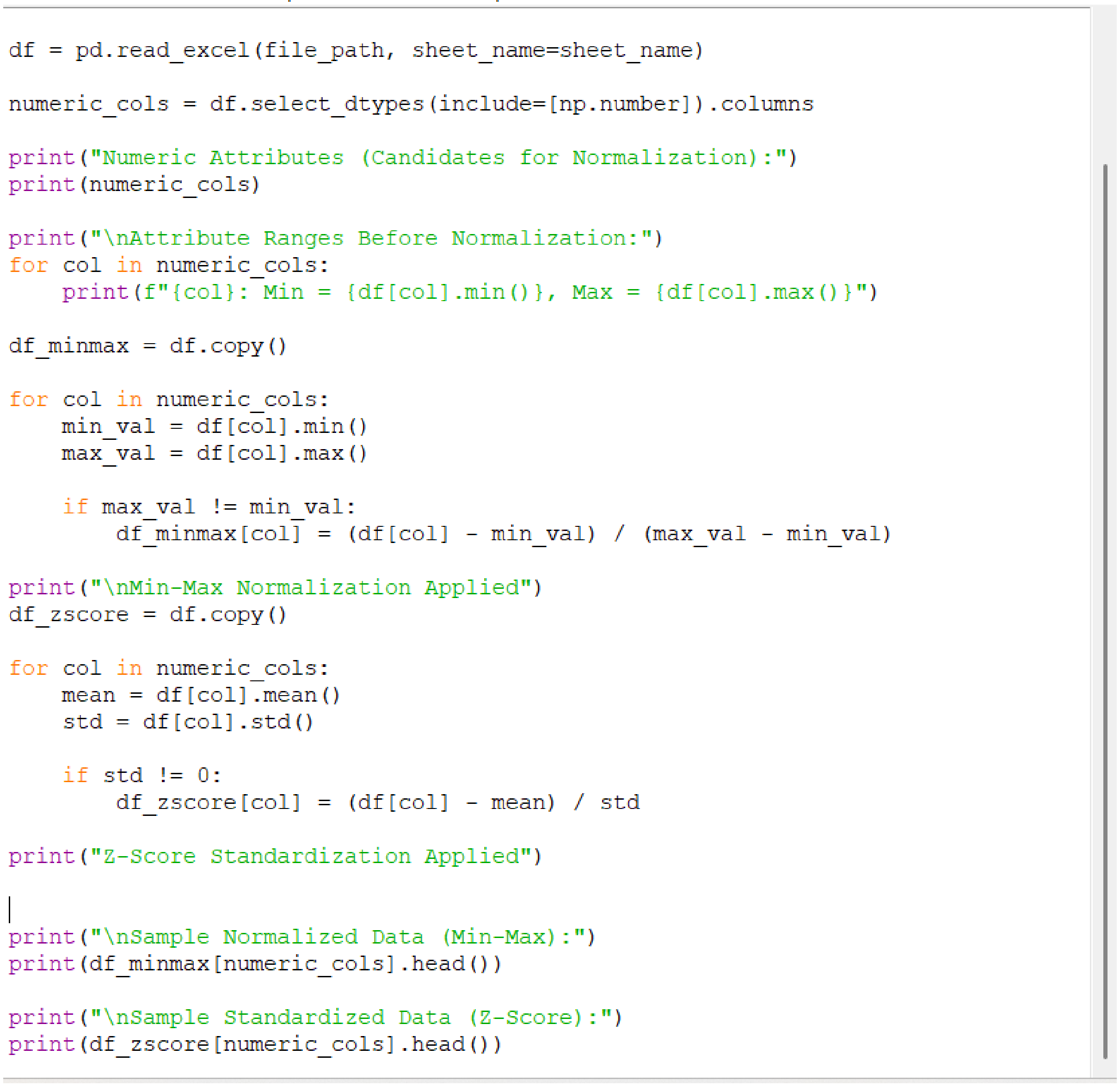

program8.py - C:/Users/glaks/OneDrive/Desktop/sem4/ml/lab2/program8.py (3.10.6)
File Edit Format Run Options Window Help

```
import pandas as pd
import numpy as np

file_path = r"C:\Users\glaks\OneDrive\Desktop\sem4\ml\lab2\Lab Session Data.xlsx"
sheet_name = "thyroid0387_UCI"

df = pd.read_excel(file_path, sheet_name=sheet_name)

print("Initial Missing Values:\n")
print(df.isnull().sum())

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

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

        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR

        outliers = df[(df[col] < lower) | (df[col] > upper)]

        if outliers.empty:
            mean_val = df[col].mean()
            df[col].fillna(mean_val, inplace=True)
            print(f"{col}: Missing values filled using MEAN = {mean_val:.2f}")
        else:
            median_val = df[col].median()
            df[col].fillna(median_val, inplace=True)
            print(f"{col}: Missing values filled using MEDIAN = {median_val:.2f}")

for col in categorical_cols:
    if df[col].isnull().sum() > 0:
        mode_val = df[col].mode()[0]
        df[col].fillna(mode_val, inplace=True)
        print(f"{col}: Missing values filled using MODE = {mode_val}")


print("\nMissing Values After Imputation:\n")
print(df.isnull().sum())
```

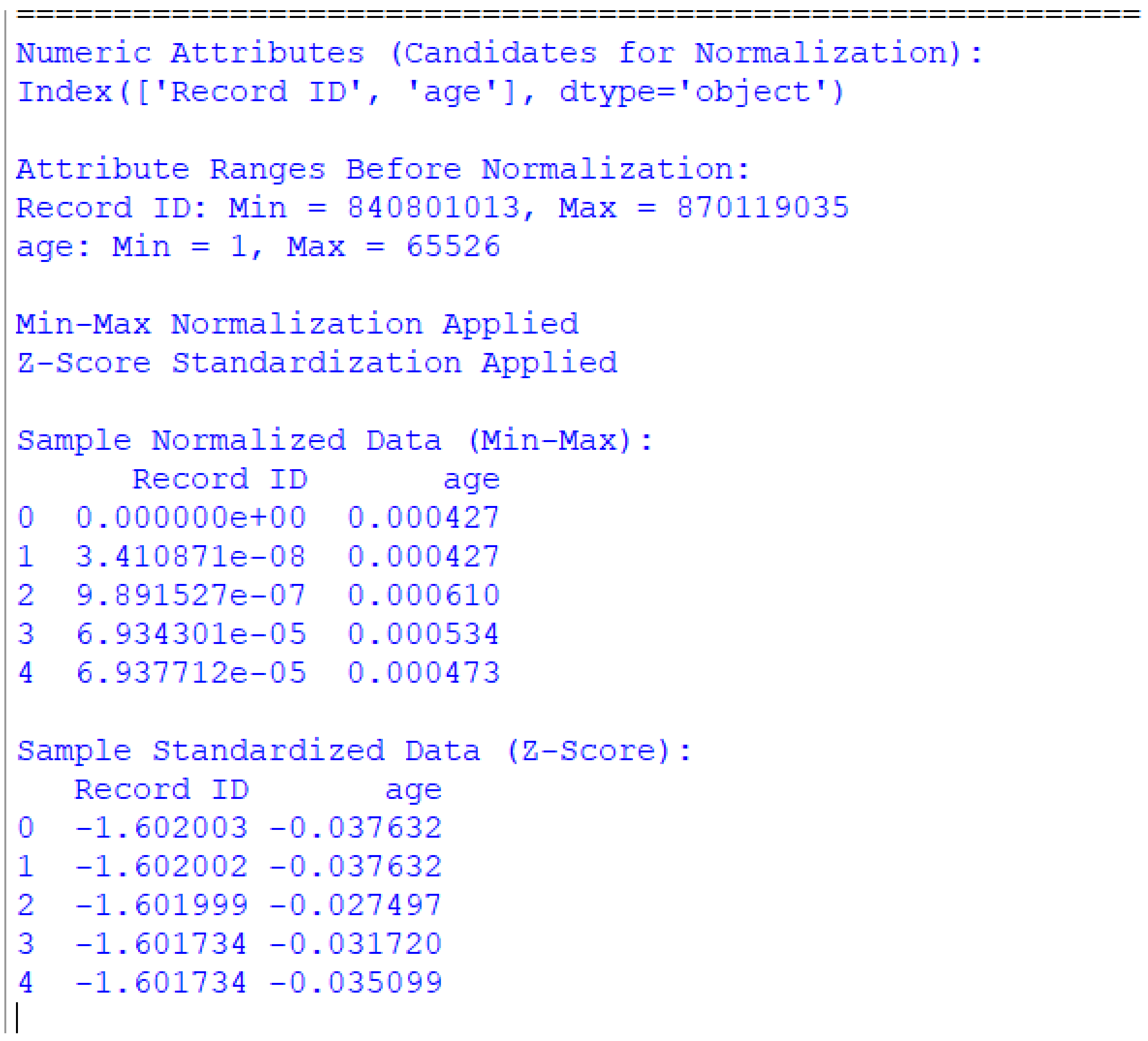


```
age                           0
sex                           0
on thyroxine                  0
query on thyroxine            0
on antithyroid medication     0
sick                          0
pregnant                      0
thyroid surgery               0
I131 treatment                0
query hypothyroid             0
query hyperthyroid            0
lithium                       0
goitre                        0
tumor                         0
hypopituitary                 0
psych                         0
TSH measured                  0
TSH                           0
T3 measured                   0
T3                            0
TT4 measured                  0
TT4                           0
T4U measured                  0
T4U                           0
FTI measured                  0
FTI                           0
TBG measured                  0
TBG                           0
referral source               0
Condition                     0
dtype: int64

Missing Values After Imputation:

Record ID                     0
age                           0
sex                           0
on thyroxine                  0
query on thyroxine            0
on antithyroid medication     0
sick                          0
pregnant                      0
thyroid surgery               0
I131 treatment                0
query hypothyroid             0
query hyperthyroid            0
lithium                       0
```

9Q



```
df = pd.read_excel(file_path, sheet_name=sheet_name)

numeric_cols = df.select_dtypes(include=[np.number]).columns

print("Numeric Attributes (Candidates for Normalization):")
print(numeric_cols)

print("\nAttribute Ranges Before Normalization:")
for col in numeric_cols:
    print(f"{col}: Min = {df[col].min()}, Max = {df[col].max()}")

df_minmax = df.copy()

for col in numeric_cols:
    min_val = df[col].min()
    max_val = df[col].max()

    if max_val != min_val:
        df_minmax[col] = (df[col] - min_val) / (max_val - min_val)

print("\nMin-Max Normalization Applied")
df_zscore = df.copy()

for col in numeric_cols:
    mean = df[col].mean()
    std = df[col].std()

    if std != 0:
        df_zscore[col] = (df[col] - mean) / std

print("Z-Score Standardization Applied")


print("\nSample Normalized Data (Min-Max):")
print(df_minmax[numeric_cols].head())

print("\nSample Standardized Data (Z-Score):")
print(df_zscore[numeric_cols].head())
```



```
=============================================================
Numeric Attributes (Candidates for Normalization):
Index(['Record ID', 'age'], dtype='object')

Attribute Ranges Before Normalization:
Record ID: Min = 840801013, Max = 870119035
age: Min = 1, Max = 65526

Min-Max Normalization Applied
Z-Score Standardization Applied

Sample Normalized Data (Min-Max):
      Record ID       age
0  0.000000e+00  0.000427
1  3.410871e-08  0.000427
2  9.891527e-07  0.000610
3  6.934301e-05  0.000534
4  6.937712e-05  0.000473

Sample Standardized Data (Z-Score):
   Record ID       age
0  -1.602003 -0.037632
1  -1.602002 -0.037632
2  -1.601999 -0.027497
3  -1.601734 -0.031720
4  -1.601734 -0.035099
```