```python
import pandas as pd
import numpy as np
data=
pd.read_excel(r"19CSE305_LabData_Set3.1.xlsx",sheet_name="thyroid0387_
UCI")

print(data)
```

```
     Record ID  age sex on thyroxine query on thyroxine  \
0    840801013   29   F            f                  f
1    840801014   29   F            f                  f
2    840801042   41   F            f                  f
3    840803046   36   F            f                  f
4    840803047   32   F            f                  f
..         ...  ...  ..          ...                ...
995  841031002   41   F            f                  f
996  841031010   41   F            f                  f
997  841031030   20   F            f                  f
998  841031031   20   F            f                  f
999  841031032   73   F            f                  f

    on antithyroid medication sick pregnant thyroid surgery I131
treatment  \
0                           f    f        f               f
f
1                           f    f        f               f
f
2                           f    f        f               f
f
3                           f    f        f               f
f
4                           f    f        f               f
f
..                        ...  ...      ...             ...
...
995                         f    f        f               f
f
996                         f    f        f               f
f
997                         f    f        t               f
f
998                         f    f        f               f
f
999                         f    f        f               f
f

     ... TT4 measured  TT4 T4U measured   T4U FTI measured   FTI TBG
measured  \
0    ...            f    ?             f     ?            f    ?
f
```

```
1    ...              t  128            f     ?             f    ?
f
2    ...              f    ?            f     ?             f    ?
t
3    ...              f    ?            f     ?             f    ?
t
4    ...              f    ?            f     ?             f    ?
t
..   ...            ...  ...          ...   ...           ...  ...
...
995  ...              t  148            t  1.44             t  103
f
996  ...              t  9.7            t  1.46             t  6.6
f
997  ...              f    ?            f     ?             f    ?
t
998  ...              t  201            t  0.84             t  240
f
999  ...              t   85            t   0.9             t   94
f

    TBG referral source     Condition
0     ?           other  NO CONDITION
1     ?           other  NO CONDITION
2    11           other  NO CONDITION
3    26           other  NO CONDITION
4    36           other             S
..   ..             ...           ...
995   ?            STMW  NO CONDITION
996   ?           other             F
997  30           other  NO CONDITION
998   ?             SVI            AK
999   ?             SVI  NO CONDITION

[1000 rows x 31 columns]

print("Record ID: nominal/numerical attribute")
print("Age: Numerical attribute")
print("Sex: Nominal attribute")
print("On Thyroxine: nominal attribute")
print("Query on Thyroxine: nominal attribute")
print("On Antithyroid Medication: nominal attribute")
print("Sick: nominal attribute")
print("Pregnant: nominal attribute")
print("Thyroid Surgery: nominal attribute")

print("I131 Treatment: nominal attribute")
print("TT4 Measured: nominal attribute")
print("TT4: numerical attribute")
print("T4U: continuous attribute")
```

```python
print("FTI Measured: nominal attribute")
print("FTI: numerical attribute")
print("TBG Measured: nominal attribute")
print("TBG : numerical attribute")
print("Condition : nominal attribute")
```

```
Record ID: nominal/numerical attribute
Age: Numerical attribute
Sex: Nominal attribute
On Thyroxine: nominal attribute
Query on Thyroxine: nominal attribute
On Antithyroid Medication: nominal attribute
Sick: nominal attribute
Pregnant: nominal attribute
Thyroid Surgery: nominal attribute
I131 Treatment: nominal attribute
TT4 Measured: nominal attribute
TT4: numerical attribute
T4U: continuous attribute
FTI Measured: nominal attribute
FTI: numerical attribute
TBG Measured: nominal attribute
TBG : numerical attribute
Condition : nominal attribute
```

```python
print("Sex: 0 AND 1 value for MALE & FEMALE")
print("On Thyroxine:0 and 1 value for t & f")
print("Query on Thyroxine: 0 and 1 value for t & f")
print("On Antithyroid Medication: 0 and 1 value for t & f")
print("Sick: 0 and 1 value for t & f")
print("Pregnant: 0 and 1 value for t & f")
print("Thyroid Surgery: 0 and 1 value for t & f")

print("I131 Treatment: 0 and 1 value for t & f")
print("TT4 Measured: 0 and 1 value for t & f")

print("FTI Measured: 0 and 1 value for t & f")
print("FTI: numerical attribute")
print("TBG Measured: 0 and 1 value for t & f")
print("Condition : 0 and 1 value for t & f")
```

```
Sex: 0 AND 1 value for MALE & FEMALE
On Thyroxine:0 and 1 value for t & f
Query on Thyroxine: 0 and 1 value for t & f
On Antithyroid Medication: 0 and 1 value for t & f
Sick: 0 and 1 value for t & f
Pregnant: 0 and 1 value for t & f
Thyroid Surgery: 0 and 1 value for t & f
I131 Treatment: 0 and 1 value for t & f
```

```
TT4 Measured: 0 and 1 value for t & f
FTI Measured: 0 and 1 value for t & f
FTI: numerical attribute
TBG Measured: 0 and 1 value for t & f
Condition : 0 and 1 value for t & f
```

```python
print(f'Range of age column is {data["age"].min()} to
{data["age"].max()}')

data["TSH"]= pd.to_numeric(data['TSH'], errors='coerce')
print(f'Range of TSH column is {data["TSH"].min()} to
{data["TSH"].max()}')

data["T3"]= pd.to_numeric(data['T3'], errors='coerce')
print(f'Range of T3 column is {data["T3"].min()} to
{data["T3"].max()}')

data["TT4"]= pd.to_numeric(data['TT4'], errors='coerce')
print(f'Range of TT4 column is {data["TT4"].min()} to
{data["TT4"].max()}')

data["T4U"]= pd.to_numeric(data['T4U'], errors='coerce')
print(f'Range of T4U column is {data["T4U"].min()} to
{data["T4U"].max()}')

data["FTI"]= pd.to_numeric(data['FTI'], errors='coerce')
print(f'Range of FTI column is {data["FTI"].min()} to
{data["FTI"].max()}')

data["TBG"]= pd.to_numeric(data['TBG'], errors='coerce')
print(f'Range of TBG column is {data["TBG"].min()} to
{data["TBG"].max()}')
```

```
Range of age column is 1 to 97
Range of TSH column is 0.05 to 430.0
Range of T3 column is 0.05 to 8.599999
Range of TT4 column is 3.0 to 359.0
Range of T4U column is 0.2 to 1.86
Range of FTI column is 2.5 to 839.0
Range of TBG column is 9.299999 to 53.0
```

```python
import pandas as pd

print(f'Mean of age column {(data["age"]).mean()}')
print(f'Variance of age column {(data["age"]).var()}')

data["TSH"]= pd.to_numeric(data['TSH'], errors='coerce')
print(f'Mean of TSH column {(data["TSH"]).mean()}')
print(f'Variance of TSH column {(data["TSH"]).var()}')
```

```
data["T3"]= pd.to_numeric(data['T3'], errors='coerce')
print(f'Mean of T3 column {(data["T3"]).mean()}')
print(f'Variance of T3 column {(data["T3"]).var()}')

data["TT4"]= pd.to_numeric(data['TT4'], errors='coerce')
print(f'Mean of TT4 column {(data["TT4"]).mean()}')
print(f'Variance of TT4 column {(data["TT4"]).var()}')

data["T4U"]= pd.to_numeric(data['T4U'], errors='coerce')
print(f'Mean of T4U column {(data["T4U"]).mean()}')
print(f'Variance of T4U column {(data["T4U"]).var()}')

data["FTI"]= pd.to_numeric(data['FTI'], errors='coerce')
print(f'Mean of FTI column {(data["FTI"]).mean()}')
print(f'Variance of FTI column {(data["FTI"]).var()}')

data["TBG"]= pd.to_numeric(data['TBG'], errors='coerce')
print(f'Mean of TBG column {(data["TBG"]).mean()}')
print(f'Variance of TBG column {(data["TBG"]).var()}')

Mean of age column 51.509
Variance of age column 352.5584774774775
Mean of TSH column 6.5596384079096035
Variance of TSH column 865.3461522583341
Mean of T3 column 1.8222431065162907
Variance of T3 column 0.6502605662253198
Mean of TT4 column 106.44770833333334
Variance of TT4 column 1739.129400373653
Mean of T4U column 0.972039911308204
Variance of T4U column 0.04283467987035838
Mean of FTI column 114.60088691796008
Variance of FTI column 3833.3445497119487
Mean of TBG column 26.23055552777778
Variance of TBG column 65.19932636428574

#A2

data = data.replace(['?', ' '], np.nan)
numeric_columns = data.select_dtypes(include=[np.number]).columns

column_means = data[numeric_columns].mean()
data[numeric_columns] = data[numeric_columns].fillna(column_means)


column_medians = data[numeric_columns].median()
def replace_outliers(column):
    median = column.median()
    lower_bound = column.quantile(0.25) - 1.5 * (column.quantile(0.75)
- column.quantile(0.25))
    upper_bound = column.quantile(0.75) + 1.5 * (column.quantile(0.75)
- column.quantile(0.25))

    column[column < lower_bound] = median
    column[column > upper_bound] = median

    return column

data[numeric_columns] = data[numeric_columns].apply(replace_outliers)


categorical_columns = data.select_dtypes(include=['object']).columns
def replace_outliers_categories(column):
    mode = column.mode().values[0]
    column[column != mode] = mode
    return column

data[categorical_columns] =
data[categorical_columns].apply(replace_outliers_categories)




#A3

column_stats = pd.DataFrame({
    'Column': numeric_columns,
    'Range': int(numeric_columns.max()) - (numeric_columns.min())
})

threshold = 1
columns_to_normalize = column_stats[column_stats['Range'] > threshold]
['Column']

print("Columns that may need normalization:")
print(columns_to_normalize)
```

```
----------------------------------------------------------------------
-----
ValueError                               Traceback (most recent call
last)
c:\Users\Win10\Downloads\lab2.ipynb Cell 8 line 5
      <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=0'>1</a> #A3
      <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=2'>3</a> column_stats = pd.DataFrame({
      <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=3'>4</a>     'Column': numeric_columns,
----> <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=4'>5</a>     'Range': int(numeric_columns.max()) -
(numeric_columns.min())
      <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=5'>6</a> })
      <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=7'>8</a> threshold = 1
      <a
href='vscode-notebook-cell:/c%3A/Users/Win10/Downloads/lab2.ipynb#X10s
ZmlsZQ%3D%3D?line=8'>9</a> columns_to_normalize =
column_stats[column_stats['Range'] > threshold]['Column']

ValueError: invalid literal for int() with base 10: 'age'
```

```python
#A4
# Extract the first two observation vectors (rows)
vector1 = data.iloc[0]
vector2 = data.iloc[1]

# Define a function to calculate the Jaccard Coefficient
def jaccard_coefficient(vec1, vec2):
    intersection = sum((vec1 == 't') & (vec2 == 't'))
    union = sum((vec1 == 't') | (vec2 == 't'))
    return intersection / union

# Define a function to calculate the Simple Matching Coefficient
def simple_matching_coefficient(vec1, vec2):
    match = sum(vec1 == vec2)
    total = len(vec1)
    return match / total

# Calculate JC and SMC for the two vectors
jc_value = jaccard_coefficient(vector1, vector2)
smc_value = simple_matching_coefficient(vector1, vector2)

# Print the calculated values
print(f"Jaccard Coefficient (JC): {jc_value}")
print(f"Simple Matching Coefficient (SMC): {smc_value}")

# Compare the values and make a judgment
if jc_value > smc_value:
    print("JC is more appropriate for measuring similarity.")
elif smc_value > jc_value:
    print("SMC is more appropriate for measuring similarity.")
else:
    print("JC and SMC are equally appropriate for measuring
similarity.")
```

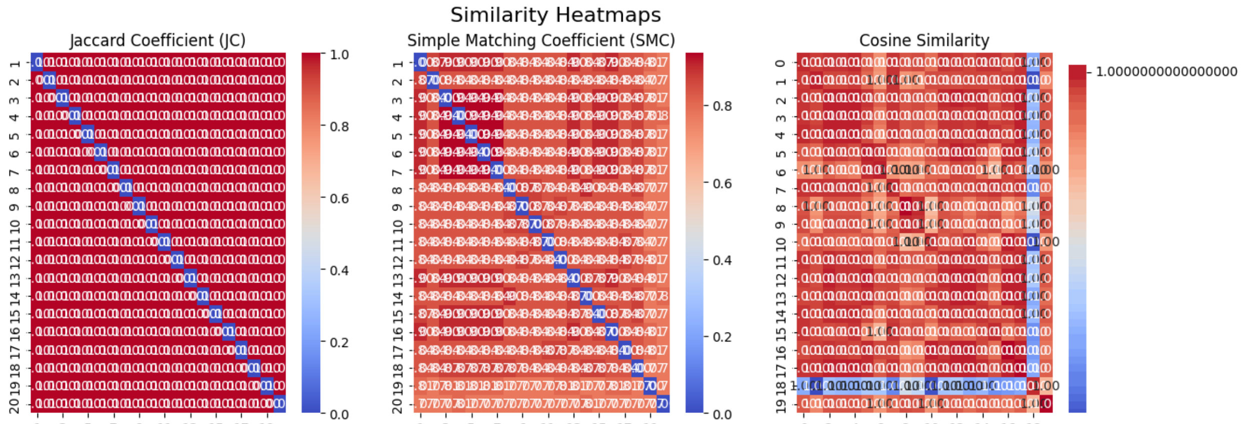
```
Jaccard Coefficient (JC): 1.0
Simple Matching Coefficient (SMC): 0.8709677419354839
JC is more appropriate for measuring similarity.
```

```
#A5
from sklearn.metrics.pairwise import cosine_similarity
vector1 = data[numeric_columns].iloc[0, 2:].values  # Start from the
third column (excluding 'Record ID' and 'age')
vector2 = data[numeric_columns].iloc[1, 2:].values

# Reshape the vectors into 2D arrays for cosine similarity calculation
vector1 = vector1.reshape(1, -1)
vector2 = vector2.reshape(1, -1)

# Calculate the cosine similarity
cosine_sim = cosine_similarity(vector1, vector2)

# Print the cosine similarity value
print(f"Cosine Similarity: {cosine_sim[0][0]}")
```

```
Cosine Similarity: 0.9957396988804667
```

```
#A6
from sklearn.metrics import jaccard_score
from sklearn.metrics import pairwise_distances
import seaborn as sns
import matplotlib.pyplot as plt
# Example: Selecting the first 20 rows and relevant columns (replace
with your actual columns)
df_subset = data.iloc[:20]

def jaccard_coefficient(vec1, vec2):
    intersection = sum((vec1 == 't') & (vec2 == 't'))
    union = sum((vec1 == 't') | (vec2 == 't'))
    return intersection / union

# Calculate Jaccard Coefficient (JC)
jc_matrix = pd.DataFrame(np.zeros((20, 20)), index=range(1, 21),
columns=range(1, 21))
for i in range(20):
    for j in range(i+1, 20):
        jc = jaccard_coefficient(df_subset.iloc[i], df_subset.iloc[j])
        jc_matrix.iloc[i, j] = jc
        jc_matrix.iloc[j, i] = jc

# Calculate Simple Matching Coefficient (SMC)
def smc_similarity(vec1, vec2):
    match = sum(vec1 == vec2)
    total = len(vec1)
    return match / total
```

```python
smc_matrix = pd.DataFrame(np.zeros((20, 20)), index=range(1, 21),
columns=range(1, 21))
for i in range(20):
    for j in range(i+1, 20):
        smc = smc_similarity(df_subset.iloc[i], df_subset.iloc[j])
        smc_matrix.iloc[i, j] = smc
        smc_matrix.iloc[j, i] = smc

# Calculate Cosine Similarity
cosine_sim_matrix = cosine_similarity(data[numeric_columns].head(20))

# Convert to DataFrames for visualization
jc_df = pd.DataFrame(jc_matrix, index=range(1, 21), columns=range(1,
21))
smc_df = pd.DataFrame(smc_matrix, index=range(1, 21), columns=range(1,
21))


# Create subplots for each similarity measure
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
fig.suptitle('Similarity Heatmaps', fontsize=16)

# JC Heatmap
sns.heatmap(jc_df, annot=True, cmap='coolwarm', fmt='.2f', ax=axes[0])
axes[0].set_title('Jaccard Coefficient (JC)')

# SMC Heatmap
sns.heatmap(smc_df, annot=True, cmap='coolwarm', fmt='.2f',
ax=axes[1])
axes[1].set_title('Simple Matching Coefficient (SMC)')

# Cosine Similarity Heatmap
sns.heatmap(cosine_sim_matrix, annot=True, cmap='coolwarm', fmt='.2f',
ax=axes[2])
axes[2].set_title('Cosine Similarity')

plt.show()
```


Similarity Heatmaps
Jaccard Coefficient (JC)
Simple Matching Coefficient (SMC)
Cosine Similarity
1.0000000000000000