```
import pandas as pd #pandas are used to data manupulation
import numpy as np


from google.colab import files #importing files from google colb
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving 19CSE305_LabData_Set3.1.xlsx to 19CSE305_LabData_Set3.1.xlsx

Anisha Part

```
#Question1
excel = pd.ExcelFile('19CSE305_LabData_Set3.1.xlsx') #aceesing the excel
df = pd.read_excel(excel, 'thyroid0387_UCI')


df
```

	Record ID	age	sex	on thyroxine	query on thyroxine	on antithyroid medication	sick	pregnant	thyroid surgery	I131 treatment	...	TT4 measured	TT4	T4U measured
0	840801013	29	F	f	f	f	f	f	f	f	...	f	?	
1	840801014	29	F	f	f	f	f	f	f	f	...	t	128	
2	840801042	41	F	f	f	f	f	f	f	f	...	f	?	
3	840803046	36	F	f	f	f	f	f	f	f	...	f	?	
4	840803047	32	F	f	f	f	f	f	f	f	...	f	?	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
995	841031002	41	F	f	f	f	f	f	f	f	...	t	148	
996	841031010	41	F	f	f	f	f	f	f	f	...	t	9.7	
997	841031030	20	F	f	f	f	f	t	f	f	...	f	?	
998	841031031	20	F	f	f	f	f	f	f	f	...	t	201	
999	841031032	73	F	f	f	f	f	f	f	f	...	t	85	

1000 rows × 31 columns

```
for attribute in df.columns:
  # Get the unique values of the attribute
  unique_values = df[attribute].unique()

  # Identify the datatype of the attribute
  datatype = None
  if len(unique_values) == 2 and unique_values[0] == "0" and unique_values[1] == "1":
    datatype = "binary"
  elif isinstance(unique_values[0], str):
    datatype = "nominal"
  elif all(isinstance(value, int) for value in unique_values):
    datatype = "ordinal"
  else:
    datatype = "numerical"

# Print the attribute name, unique values, and datatype
  print(f"Attribute: {attribute}")
  print(f"Unique values: {unique_values}")
  print(f"Datatype: {datatype}")
```

```
Attribute: Record ID
Unique values: [840801013 840801014 840801042 840803046 840803047 840803048 840803068
 840807019 840808060 840808073 840810016 840813022 840813060 840813068
 840814014 840814057 840815016 840815020 840815067 840815068 840815069
 840816001 840816002 840816003 840816004 840816005 840816006 840816007
 840816008 840816009 840816010 840816011 840816013 840816014 840816022
 840816025 840816028 840816029 840816046 840816047 840816049 840816052
 840816060 840816061 840816063 840816069 840816070 840816071 840816072
 840816076 840816077 840816078 840816093 840816095 840816097 840817001
```

```
840817002 840817003 840817004 840817006 840817008 840817009 840817010
840817012 840817028 840817038 840817056 840817058 840817059 840817060
840817061 840817062 840817064 840817065 840820001 840820002 840820003
840820004 840820009 840820014 840820020 840820021 840820022 840820027
840820047 840820052 840821009 840821010 840821011 840821012 840821013
840821014 840821016 840821017 840821022 840821023 840821024 840821042
840821043 840821047 840821049 840821054 840821055 840821056 840821063
840822006 840822008 840822023 840822025 840822026 840822027 840822028
840822029 840822030 840822031 840822033 840823001 840823002 840823003
840823004 840823005 840823006 840823007 840823008 840823010 840823011
840823019 840823025 840823030 840823032 840823038 840823039 840823040
840823041 840823042 840823043 840823044 840823054 840823055 840823063
840823077 840823078 840823084 840823085 840823086 840823087 840823089
840823090 840823091 840823092 840823093 840824001 840824002 840824009
840824010 840824011 840824015 840824043 840824046 840827001 840827002
840827003 840827004 840827005 840827006 840827007 840827008 840827019
840827023 840827025 840827028 840827035 840827036 840827041 840827053
840827054 840827055 840827060 840827062 840827065 840828001 840828002
840828003 840828024 840828031 840828033 840828034 840828035 840828036
840828037 840828039 840828040 840828042 840828046 840829005 840829012
840829013 840829014 840829015 840829021 840829035 840829036 840829037
840829038 840829039 840829040 840829041 840829042 840829043 840829044
840829052 840829056 840830002 840830003 840830004 840830008 840830031
840830034 840830045 840830052 840830053 840830055 840830063 840830064
840831006 840831007 840831037 840831047 840831049 840831054 840831055
840831058 840831059 840903001 840903002 840903003 840903004 840903005
840903006 840903007 840903008 840903009 840903010 840903053 840903054
840903058 840903065 840903068 840903071 840903072 840903073 840903074
840903075 840903083 840903084 840903085 840904003 840904005 840904006
840904007 840904009 840904017 840904027 840904028 840904029 840904034
840904037 840904042 840904045 840904046 840904047 840904048 840904049
840905002 840905003 840905004 840905005 840905019 840905025 840905027
840905029 840905030 840905032 840905037 840905038 840905041 840905046
840906002 840906003 840906005 840906006 840906007 840906011 840906012
840906015 840906016 840906018 840906021 840906022 840906024 840906026
840906027 840906029 840906034 840906044 840906049 840906050 840906051
840906057 840906059 840906060 840906063 840906064 840906071 840906077
840906081 840906089 840906095 840907001 840907002 840907012 840907028
840907050 840907052 840907053 840907054 840907055 840907056 840907057
840907060 840907075 840910006 840910007 840910008 840910009 840910011
840910012 840910013 840910023 840910024 840910037 840910038 840910044
840910045 840910046 840910047 840910049 840910050 840910056 840910057
840910060 840910069 840911005 840911016 840911020 840911029 840911036
840911037 840911038 840911039 840911040 840911041 840911042 840911045
840911047 840911059 840911060 840912001 840912002 840912004 840912005
840913002 840913004 840913006 840913009 840913015 840913016 840913017
840913018 840913019 840913020 840913021 840913032 840913033 840913034
840913038 840913040 840913041 840913042 840913043 840913045 840913047
840913051 840913052 840913053 840913059 840913061 840913063 840913068
```

```
import pandas as pd


# Study each attribute and associated values present. Identify the datatype (nominal etc.) for the attribute.
for column in df.columns:
    print(column, df[column].dtype, df[column].unique())

# For categorical attributes, identify the encoding scheme to be employed.
# For ordinal variables, use label encoding.
# For nominal variables, use one-hot encoding.
for column in df.columns:
    if df[column].dtype == "object":
        if df[column].dtype.name == "category":
            # Ordinal variable
            df[column] = df[column].cat.codes
        else:
            # Nominal variable
            df = pd.get_dummies(df, columns=[column])
```

```
T3 measured object ['f' 't']
T3 object ['?' 1.9 2.6 1.8 1.7 2.3 2.4 2.9 2 2.1 1.6 0.1 1.4 1.2 1.5 1.3 2.5 2.7 2.2
 2.8 3.2 0.4 0.8 1 1.1 3.7 4.4 3 3.1 3.6 7.6 0.9 4.2 0.5 0.6 0.3 0.7 3.8
 0.2 4.1 6.6 4.7 8.599999 3.3 4.3 0.05 3.4]
TT4 measured object ['f' 't']
TT4 object ['?' 128 116 76 83 133 105 122 48 90 79 104 88 107 126 113 150 93 157 80
 91 47 39 71 111 86 136 163 118 82 33 134 60 102 97 115 132 114 94 145 106
 87 120 121 152 98 109 139 131 112 64 7.5 68 99 92 95 100 184 81 308 57
 170 130 51 144 73 63 78 129 85 125 196 153 96 127 108 250 141 110 101 210
 140 74 188 3.9 117 84 138 119 27 89 236 32 55 72 44 158 69 9 6 15 149 13
 61 182 4 178 135 70 151 56 59 213 77 222 195 14 12 40 67 16 167 148 147
 123 50 65 66 7.6 4.1 49 75 202 3 169 200 24 242 162 103 143 186 124 58 35
 180 359 43 225 137 166 302 62 206 52 155 142 261 176 22 260 159 54 287
 230 36 172 296 21 154 183 198 174 208 245 217 11 207 42 228 46 201 212 23
 333 38 168 41 175 146 211 181 156 9.7]
T4U measured object ['f' 't']
T4U object ['?' 1.02 1.06 0.94 1.08 0.84 1.13 1.07 0.87 0.89 0.62 0.91 0.68 1 1.38
 0.79 0.95 1.57 0.92 1.48 1.1 0.7 1.01 1.05 0.96 0.78 1.4 0.66 0.86 0.76
 0.9 1.16 1.12 0.98 1.04 1.26 0.83 0.97 0.93 0.88 0.73 1.29 1.3 0.75 0.8
 1.83 1.03 0.61 1.44 1.18 0.59 0.81 0.64 1.2 0.82 1.19 0.99 1.56 1.22 0.71
 1.32 0.67 0.32 1.11 0.85 0.52 1.15 1.21 0.77 0.69 1.51 1.33 0.55 1.45
 1.24 1.79 0.72 1.73 1.27 1.68 1.09 1.43 0.35 0.3 1.28 0.2 1.41 1.14 0.53
 1.52 1.23 0.74 1.53 1.62 1.66 0.4 1.86 1.59 0.29 0.34 1.17 1.76 0.57 0.63
 1.71 0.31 0.49 1.31 1.34 0.5 1.75 1.36 0.36 1.42 0.6 1.74 1.46]
FTI measured object ['f' 't']
FTI object ['?' 47 85 84 96 105 95 106 176 129 100 69 39 91 90 93 66 121 92 173 117
 31 113 67 101 126 123 149 68 86 132 131 116 97 124 136 142 104 7.5 107 73
 110 130 88 128 122 102 134 163 63 354 81 109 114 133 170 99 111 108 161
 78 148 98 135 80 127 213 119 65 89 143 316 155 172 150 103 120 258 5 272
 263 166 138 52 164 337 94 118 182 41 70 144 10 4 13 87 140 74 152 77 3 82
 145 64 79 147 54 83 634 650 12 61 11 115 35 17 165 167 153 44 3.4 55 71
 253 75 2.5 197 24 156 237 203 112 141 3.5 190 37 45 193 57 76 160 6 200
 485 49 158 137 428 450 174 189 202 159 196 154 139 34 222 184 178 146 125
 21 157 51 839 332 151 305 299 266 32 53 370 22 168 60 187 171 220 169 232
 254 345 194 211 217 550 23 257 188 192 179 218 208 6.6 240]
TBG measured object ['f' 't']
TBG object ['?' 11 26 36 21 28 23 18 33 20 19 30 29 24 9.299999 25 53 27]
referral source object ['other' 'SVI' 'SVHC' 'STMW' 'SVHD' 'WEST']
Condition object ['NO CONDITION' 'S' 'F' 'AK' 'R' 'I' 'M' 'N' 'G' 'K' 'A' 'KJ' 'L' 'MK' 'Q'
 'J' 'C|I' 'O' 'LJ' 'H|K' 'D' 'GK' 'MI' 'P']
```

```
# Task 3: Study data range for numeric variables
numeric_attributes = df.select_dtypes(include=[np.number])
data_range = numeric_attributes.describe().loc[['min', 'max']]
print("Data Range for Numeric Variables:")
print(data_range)
```

```
Data Range for Numeric Variables:
       Record ID   age  sex_?  sex_F  sex_M  on thyroxine_f  on thyroxine_t  \
min  840801013.0   1.0    0.0    0.0    0.0             0.0             0.0
max  841031032.0  97.0    1.0    1.0    1.0             1.0             1.0

     query on thyroxine_f  query on thyroxine_t  on antithyroid medication_f  \
min                   0.0                   0.0                          0.0
max                   1.0                   1.0                          1.0

     ...  Condition_M  Condition_MI  Condition_MK  Condition_N  \
min  ...          0.0           0.0           0.0          0.0
max  ...          1.0           1.0           1.0          1.0

     Condition_NO CONDITION  Condition_O  Condition_P  Condition_Q  \
min                     0.0          0.0          0.0          0.0
max                     1.0          1.0          1.0          1.0

     Condition_R  Condition_S
min          0.0          0.0
max          1.0          1.0

[2 rows x 778 columns]
```

```
# Study the presence of missing values in each attribute.
print(df.isnull().sum())
```

```
Record ID      0
age            0
sex_?          0
sex_F          0
sex_M          0
              ..
Condition_O    0
Condition_P    0
Condition_Q    0
Condition_R    0
Condition_S    0
Length: 778, dtype: int64
```

```
# Task 4: Presence of missing values
#an other way to do it
```

```
missing_values = df.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
Record ID      0
age            0
sex_?          0
sex_F          0
sex_M          0
               ..
Condition_O    0
Condition_P    0
Condition_Q    0
Condition_R    0
Condition_S    0
Length: 778, dtype: int64
```

```
#task5

z_scores = np.abs((df - df.mean()) / df.std())
threshold = 50
outliers = z_scores > threshold
print(outliers)
```

```
     Condition    FTI  FTI measured  I131 treatment  Record ID     T3  \
0        False  False         False           False      False  False
1        False  False         False           False      False  False
2        False  False         False           False      False  False
3        False  False         False           False      False  False
4        False  False         False           False      False  False
..         ...    ...           ...             ...        ...    ...
995      False  False         False           False      False  False
996      False  False         False           False      False  False
997      False  False         False           False      False  False
998      False  False         False           False      False  False
999      False  False         False           False      False  False

     T3 measured    T4U  T4U measured    TBG  ...  pregnant  psych  \
0          False  False         False  False  ...     False  False
1          False  False         False  False  ...     False  False
2          False  False         False  False  ...     False  False
3          False  False         False  False  ...     False  False
4          False  False         False  False  ...     False  False
..           ...    ...           ...    ...  ...       ...    ...
995        False  False         False  False  ...     False  False
996        False  False         False  False  ...     False  False
997        False  False         False  False  ...     False  False
998        False  False         False  False  ...     False  False
999        False  False         False  False  ...     False  False

     query hyperthyroid  query hypothyroid  query on thyroxine  \
0                 False              False               False
1                 False              False               False
2                 False              False               False
3                 False              False               False
4                 False              False               False
..                  ...                ...                 ...
995               False              False               False
996               False              False               False
997               False              False               False
998               False              False               False
999               False              False               False

     referral source    sex   sick  thyroid surgery  tumor
0              False  False  False            False  False
1              False  False  False            False  False
2              False  False  False            False  False
3              False  False  False            False  False
4              False  False  False            False  False
..               ...    ...    ...              ...    ...
995            False  False  False            False  False
996            False  False  False            False  False
997            False  False  False            False  False
998            False  False  False            False  False
999            False  False  False            False  False

[1000 rows x 31 columns]
<ipython-input-7-5a1c5e20392c>:3: FutureWarning: The default value of numeric_only in DataFrame.mean is deprecated. In a future
  z_scores = np.abs((df - df.mean()) / df.std())
<ipython-input-7-5a1c5e20392c>:3: FutureWarning: The default value of numeric_only in DataFrame.std is deprecated. In a future v
  z_scores = np.abs((df - df.mean()) / df.std())
```

```
# For numeric variables, calculate the mean and variance (or standard deviation).
for column in df.columns:
    if df[column].dtype != "object":
        print(column, df[column].mean(), df[column].var())
```

```
Record ID 840934027.976 5819758750.219644
age 51.509 352.5584774774775
sex_? 0.024 0.023447447447447447
sex_F 0.672 0.22063663663663666
sex_M 0.304 0.2117957957957958
on thyroxine_f 0.845 0.13110610610610607
on thyroxine_t 0.155 0.13110610610610607
query on thyroxine_f 0.976 0.02344744744744745
query on thyroxine_t 0.024 0.02344744744744745
on antithyroid medication_f 0.986 0.013817817817817817
on antithyroid medication_t 0.014 0.013817817817817817
sick_f 0.971 0.028187187187187185
sick_t 0.029 0.028187187187187185
pregnant_f 0.986 0.013817817817817817
pregnant_t 0.014 0.013817817817817817
thyroid surgery_f 0.978 0.02153753753753754
thyroid surgery_t 0.022 0.02153753753753754
I131 treatment_f 0.976 0.023447447447447454
I131 treatment_t 0.024 0.02344744744744745
query hypothyroid_f 0.92 0.07367367367367367
query hypothyroid_t 0.08 0.07367367367367367
query hyperthyroid_f 0.925 0.06944444444444445
query hyperthyroid_t 0.075 0.06944444444444445
lithium_f 1.0 0.0
goitre_f 0.985 0.014789789789789795
goitre_t 0.015 0.014789789789789795
tumor_f 0.974 0.025349349349349348
tumor_t 0.026 0.025349349349349348
hypopituitary_f 0.999 0.0010000000000000005
hypopituitary_t 0.001 0.0010000000000000005
psych_f 0.975 0.024399399399399398
psych_t 0.025 0.024399399399399398
TSH measured_f 0.115 0.10187687687687688
TSH measured_t 0.885 0.10187687687687688
TSH_0.05 0.036 0.03473873873873873
TSH_0.1 0.143 0.12267367367367368
TSH_0.15 0.021 0.020579579579579576
TSH_0.2 0.049 0.04664564564564565
TSH_0.25 0.035 0.03380880880880881
TSH_0.3 0.03 0.029129129129129128
TSH_0.35 0.003 0.0029939939939939942
TSH_0.4 0.037 0.03566666666666665
TSH_0.5 0.022 0.02153753753753754
TSH_0.6 0.022 0.02153753753753754
TSH_0.7 0.032 0.03100700700700701
TSH_0.8 0.028 0.027243243243243242
TSH_0.9 0.028 0.027243243243243245
TSH_1 0.023 0.022493493493493492
TSH_1.1 0.017 0.016727727727727727
TSH_1.2 0.027 0.02629729729729729
TSH_1.3 0.02 0.019619619619619614
TSH_1.4 0.021 0.020579579579579576
TSH_1.5 0.015 0.014789789789789792
TSH_1.6 0.017 0.016727727727727723
TSH_1.7 0.011 0.010889889889889892
TSH_1.8 0.01 0.009909909909909991
TSH_1.83 0.001 0.0010000000000000002
TSH_1.9 0.012 0.01186786786786787
```

```
# Task 6: Calculate mean and variance (or standard deviation) for numeric variables
numeric_attributes_mean = numeric_attributes.mean()
numeric_attributes_variance = numeric_attributes.var()
numeric_attributes_stddev = numeric_attributes.std()
print("Mean of Numeric Variables:")
print(numeric_attributes_mean)
print("\nVariance of Numeric Variables:")
print(numeric_attributes_variance)
print("\nStandard Deviation of Numeric Variables:")
print(numeric_attributes_stddev)
```

```
Mean of Numeric Variables:
Record ID      8.409340e+08
age            5.150900e+01
sex_?          2.400000e-02
sex_F          6.720000e-01
sex_M          3.040000e-01
                   ...
Condition_O    3.000000e-03
Condition_P    1.000000e-03
Condition_Q    1.000000e-03
Condition_R    1.800000e-02
Condition_S    8.000000e-03
Length: 778, dtype: float64
```

```
Variance of Numeric Variables:
Record ID      5.819759e+09
age            3.525585e+02
sex_?          2.344745e-02
sex_F          2.206366e-01
sex_M          2.117958e-01
                   ...
Condition_O    2.993994e-03
Condition_P    1.000000e-03
Condition_Q    1.000000e-03
Condition_R    1.769369e-02
Condition_S    7.943944e-03
Length: 778, dtype: float64

Standard Deviation of Numeric Variables:
Record ID      76287.343316
age               18.776541
sex_?              0.153126
sex_F              0.469720
sex_M              0.460213
                   ...
Condition_O        0.054717
Condition_P        0.031623
Condition_Q        0.031623
Condition_R        0.133018
Condition_S        0.089129
Length: 778, dtype: float64
```

```
#q3)Data normalization is a technique used in data mining to transform the values of a dataset into a common scale.
# is the process of rescaling one or more attributes to the range of 0 to 1. This means that the largest value for each attribute is 1 ar
#used min max normalization:Min-Max normalization: This technique scales the values of a feature to a range between 0 and 1. This is done
from sklearn.preprocessing import MinMaxScaler#The MinMaxScaler from scikit-learn is a preprocessing technique used for feature scaling,
df = pd.DataFrame(data)#creating dataframe
age_data = df[['age']]#accessing the nedded coulmn
scaler = MinMaxScaler()#create a Min-Max scaler using MinMaxScaler()
normalized_age = scaler.fit_transform(age_data)#Fit the scaler to the 'Age' data and transform it
df['Age'] = normalized_age
print(df['Age'])#prinitng the normalized age attribute
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-76c9cde2a01c> in <cell line: 5>()
      3 #used min max normalization:Min-Max normalization: This technique scales the values of a feature to a range
between 0 and 1. This is done by subtracting the minimum value of the feature from each value, and then dividing by the
range of the feature.
      4 from sklearn.preprocessing import MinMaxScaler#The MinMaxScaler from scikit-learn is a preprocessing technique
used for feature scaling, specifically Min-Max scaling. It scales the features (attributes) of a dataset to a specific
range, usually [0, 1].
----> 5 df = pd.DataFrame(data)#creating dataframe
      6 age_data = df[['age']]#accessing the nedded coulmn
      7 scaler = MinMaxScaler()#create a Min-Max scaler using MinMaxScaler()

NameError: name 'data' is not defined
```

to know which column needs normalizationCalculate summary statistics (mean, median, standard deviation, etc.) for each attribute. High variance or large differences in the means of attributes can suggest the need for normalization. in easier words the column with the values more spreaded out

```
excel = pd.ExcelFile('19CSE305_LabData_Set3.1.xlsx') #aceesing the excel
data = pd.read_excel(excel, 'marketing_campaign')
```

```
#in the other data set the income,recency,MntWines,MntMeatProducts,MntFishProducts,MntSweetProducts,MntGoldProds due to theit high variar
from sklearn.preprocessing import MinMaxScaler
df = pd.DataFrame(data)#creating dataframe
norm_data = ['Income','Recency','MntWines','MntMeatProducts','MntFishProducts','MntSweetProducts','MntGoldProds']
scaler = MinMaxScaler()
df[norm_data]= scaler.fit_transform(df[norm_data])#Fit the scaler to the 'Age' data and transform it

print(df[norm_data])#prinitng the normalized age attribute
```

```
        Income   Recency  MntWines  MntMeatProducts  MntFishProducts  \
0     0.348178  0.585859  0.425603         0.316125         0.677165
1     0.274442  0.383838  0.007373         0.002900         0.007874
2     0.432423  0.262626  0.285523         0.073086         0.437008
3     0.151291  0.262626  0.007373         0.011021         0.039370
4     0.349147  0.949495  0.115952         0.067865         0.181102
..         ...       ...       ...              ...              ...
995   0.202420  0.656566  0.002681         0.005800         0.007874
996   0.456330  0.252525  0.532842         0.315545         0.374016
997   0.147821  0.848485  0.010054         0.010441         0.031496
998   0.315086  0.070707  0.257373         0.168794         0.511811
```

```
999  0.272829  0.343434  0.163539         0.017981         0.027559

     MntSweetProducts  MntGoldProds
0            0.334601      0.243094
1            0.003802      0.016575
2            0.079848      0.116022
3            0.011407      0.013812
4            0.102662      0.041436
..                ...           ...
995          0.000000      0.011050
996          0.220532      0.000000
997          0.064639      0.055249
998          0.155894      0.176796
999          0.007605      0.138122

[1000 rows x 7 columns]
```

Anisha's Part

```
#Q5)
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Load the data from the Excel file
file_name = r"19CSE305_LabData_Set3.1.xlsx"
worksheet_name = 'thyroid0387_UCI'
data = pd.read_excel(file_name, sheet_name=worksheet_name)

# Extract the feature vectors for the first two observations
vector1 = data.iloc[0, 1:].apply(lambda x: float(x) if str(x).replace('.', '', 1).isdigit() else np.nan)
vector2 = data.iloc[1, 1:].apply(lambda x: float(x) if str(x).replace('.', '', 1).isdigit() else np.nan)

# Calculate the dot product of the two vectors
dot_product = np.dot(vector1, vector2)

# Calculate the magnitude (length) of each vector
magnitude_vector1 = np.linalg.norm(vector1)
magnitude_vector2 = np.linalg.norm(vector2)

# Calculate the cosine similarity
cosine_similarity = dot_product / (magnitude_vector1 * magnitude_vector2)

# Print the cosine similarity
print("Cosine Similarity:", cosine_similarity)
```

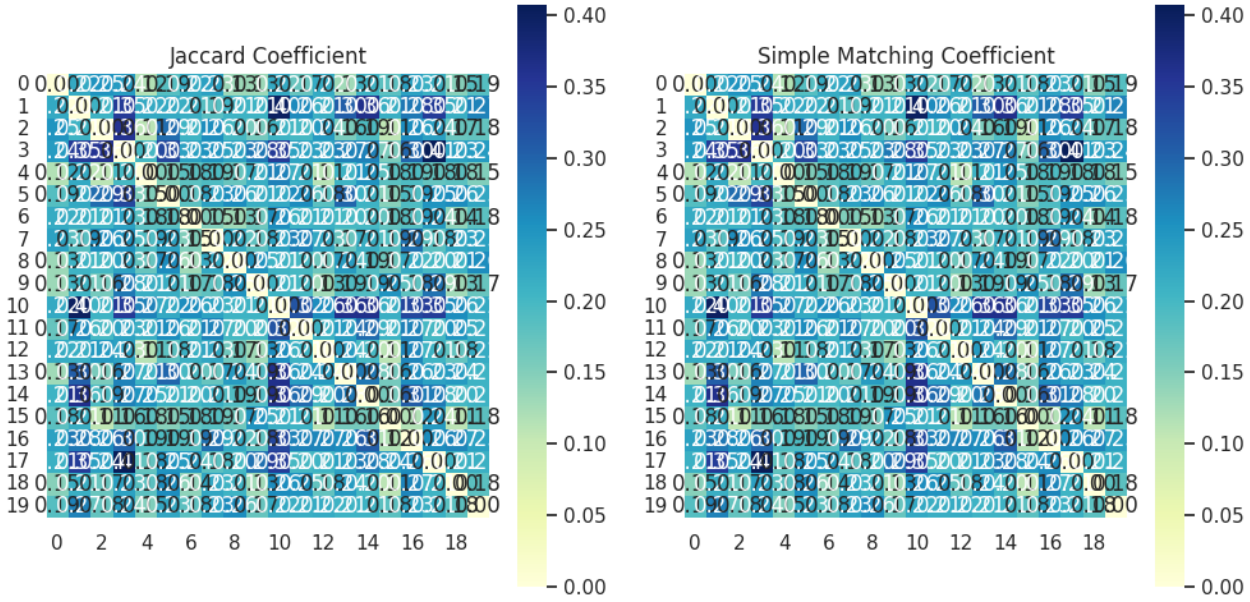
```
Cosine Similarity: nan
```

```
excel = pd.ExcelFile('19CSE305_LabData_Set3.1.xlsx') #aceesing the excel
data = pd.read_excel(excel, 'marketing_campaign')


#)q6
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import jaccard_score
from sklearn.metrics.pairwise import cosine_similarity


# Assuming the data is in a single column of the DataFrame.
observation_sets = [set(row) for _, row in data.iterrows()]
observation_vectors = [np.array(row) for _, row in data.iterrows()]

# Calculate JC (Jaccard Coefficient) :J(A, B) = |A ∩ B| / |A ∪ B|

jc_scores = [] #empty list

for i in range(len(observation_sets)):
    for j in range(i + 1, len(observation_sets)):
        intersection = len(observation_sets[i] & observation_sets[j])#calculating the intersection
        union = len(observation_sets[i] | observation_sets[j])#calculating union
        jc = intersection / union
        jc_scores.append((i, j, jc))
print("JACCARD COEFF:",jc)

# Calculate SMC (Simple Matching Coefficient) :The Simple Matching Coefficient (SMC) is a similarity measure used to compare two sets by
smc_scores = []

for i in range(len(observation_sets)):
    for j in range(i + 1, len(observation_sets)):
        matching = len(observation_sets[i] & observation_sets[j])
```

```
            non_matching = len(observation_sets[i] ^ observation_sets[j])
            smc = matching / (matching + non_matching)
            smc_scores.append((i, j, smc))
    print("SIMPLE MATICHING COEFF:",smc)
```

```
     JACCARD COEFF: 0.23333333333333334
     SIMPLE MATICHING COEFF: 0.23333333333333334
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Read data from the Excel sheet
excel = pd.ExcelFile('19CSE305_LabData_Set3.1.xlsx') #aceesing the excel
df = pd.read_excel(excel, 'marketing_campaign')

# Extract the first 20 observation vectors from the Excel sheet
observation_vectors = df.iloc[:20, :].values

# Step 2: Calculate JC, SMC, and COS
num_observations = len(observation_vectors)
jc_matrix = np.zeros((num_observations, num_observations))
smc_matrix = np.zeros((num_observations, num_observations))
cos_matrix = np.zeros((num_observations, num_observations))

for i in range(num_observations):
    for j in range(num_observations):
        if i != j:
            # Convert vectors to sets for Jaccard and SMC calculations
            set_i = set(observation_vectors[i])
            set_j = set(observation_vectors[j])

            # Calculate JC
            intersection_jc = len(set_i & set_j)
            union_jc = len(set_i | set_j)
            jc_matrix[i, j] = intersection_jc / union_jc

            # Calculate SMC
            matching_smc = len(set_i & set_j)
            non_matching_smc = len(set_i ^ set_j)
            smc_matrix[i, j] = matching_smc / (matching_smc + non_matching_smc)
            # Calculate COS
            #dot_product_cos = np.dot(observation_vectors[i], observation_vectors[j])
            #norm_i_cos = np.linalg.norm(observation_vectors[i])

            #norm_j_cos = np.linalg.norm(observation_vectors[j])
            #cos_matrix[i, j] = dot_product_cos / (norm_i_cos * norm_j_cos)


# Step 3: Create heatmap plots for JC, SMC, and COS matrices
plt.figure(figsize=(15, 5))

plt.subplot(131)
sns.heatmap(jc_matrix, cmap="YlGnBu", annot=True, fmt=".2f", square=True)
plt.title("Jaccard Coefficient")

plt.subplot(132)
sns.heatmap(smc_matrix, cmap="YlGnBu", annot=True, fmt=".2f", square=True)
plt.title("Simple Matching Coefficient")

#plt.subplot(133)
#sns.heatmap(cos_matrix, cmap="YlGnBu", annot=True, fmt=".2f", square=True)
#plt.title("Cosine Similarity")

plt.tight_layout()
plt.show()
```



Manogna's Part

```
column_names=data.select_dtypes(include=['float64','int64'])
variance={}
mean={}
column_names=data.select_dtypes(include=['float64','int64'])

for column in column_names:
    variance[column] = np.var(data[column])
    mean[column]=np.mean(data[column])
print(variance)
mean
import math

outliers={}
for column in column_names:
    count=0
    for a in data[column]:
        z=(a-mean[column])/math.sqrt(variance[column])
        if z<0:
            z*=-1
        if z>3:
            count+=1
    if count >0:
        outliers[column]='yes'
    else:
        outliers[column]='no'
print(outliers)
for column in data.columns:
    if column in column_names :
        if outliers[column]=='yes':
            data[column].fillna(data[column].median(), inplace=True)
        else:
            data[column].fillna(data[column].mean(), inplace=True)
    else:
        data[column].fillna(data[column].mode(), inplace=True)

data
```

```
{'ID': 10609003.698096002, 'Year_Birth': 153.77751099999998, 'Income': 478111436.3978776, 'Kidhome': 0.3119749999999999, 'T
{'ID': 'no', 'Year_Birth': 'yes', 'Income': 'yes', 'Kidhome': 'no', 'Teenhome': 'no', 'Recency': 'no', 'MntWines': 'yes', '
<ipython-input-26-c7b11c58b842>:17: RuntimeWarning: invalid value encountered in double_scalars
  z=(a-mean[column])/math.sqrt(variance[column])
<ipython-input-26-c7b11c58b842>:34: UserWarning: Unable to sort modes: '<' not supported between instances of 'datetime.dat
  data[column].fillna(data[column].mode(), inplace=True)
```

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVis
0	5524	1957	Graduation	Single	58138.0	0	0	2012-04-09 00:00:00	58	635	...	
1	2174	1954	Graduation	Single	46344.0	1	1	2014-08-03 00:00:00	38	11	...	
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	...	
3	6182	1984	Graduation	Together	26646.0	1	0	2014-10-02 00:00:00	26	11	...	

```
from google.colab import files #importing files from google colb
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving dataset.csv.xlsx to dataset.csv.xlsx

```
binary_attributes = ['onthyroxine', 'queryonthyroxine', 'onantithyroidmedication','sick', 'pregnant', 'thyroidsurgery', 'I131treatment',
v1=[]
v2=[]
v10c=0
v11c=0
v20c=0
v21c=0
for column  in binary_attributes :
    if data.loc[0, column] == 'f':
        v1.append(0)
        v10c+=1
    else:
        v1.append(1)
        v11c+=1
    if data.loc[1, column] == 'f':
        v2.append(0)
        v20c+=1
    else:
        v2.append(1)
        v21c+=1
f11=v11c+v21c
f01=v10c+v21c
f10=v11c+v20c
f00=v10c+v20c
if f01 + f10 + f11 != 0:
    jc = f11 / (f01 + f10 + f11)
else:
    jc = 0.0

if f00 + f01 + f10 + f11 != 0:
    smc = (f11 + f00) / (f00 + f01 + f10 + f11)
else:
    smc = 0.0
print(jc)
print (smc)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key, method, tolerance)
   3801             try:
```

```
import numpy as np

def jaccard_coefficient(vector1, vector2):
  """Calculates the Jaccard Coefficient between two vectors.

  Args:
    vector1: A vector of binary values.
    vector2: A vector of binary values.

  Returns:
    The Jaccard Coefficient between the two vectors.
  """

  intersection = np.sum(vector1 & vector2)
  union = np.sum(vector1 | vector2)
  return intersection / union

def simple_matching_coefficient(vector1, vector2):
  """Calculates the Simple Matching Coefficient between two vectors.

  Args:
    vector1: A vector of binary values.
    vector2: A vector of binary values.

  Returns:
    The Simple Matching Coefficient between the two vectors.
  """

  intersection = np.sum(vector1 & vector2)
  return intersection

# Load the dataset



# Get the first two observation vectors
vector1 = df[0, :]
vector2 = df[1, :]

# Calculate the Jaccard Coefficient and Simple Matching Coefficient
jc = jaccard_coefficient(vector1, vector2)
smc = simple_matching_coefficient(vector1, vector2)

# Print the results
print("Jaccard Coefficient:", jc)
print("Simple Matching Coefficient:", smc)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self,
key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                                  5 frames

TypeError: '(0, slice(None, None, None))' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                         Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
_check_indexing_error(self, key)
   5923             # if key is not a scalar, directly raise an error (the code
below
   5924             # would convert to numpy arrays and raise later any way) -
GH29926
-> 5925             raise InvalidIndexError(key)
   5926
   5927     @cache_readonly

InvalidIndexError: (0, slice(None, None, None))
```

2s completed at 5:56 PM