Section C (Algorithm implementation using packages)

```
#@title Import libraries
import pandas as pd
import seaborn as sb
import numpy as np
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

#@title Reading Data
data = pd.read_csv('/content/Thyroid_data.csv')

data.head()

   age sex on thyroxine query on thyroxine on antithyroid medication
sick  \
0   41   F            f                  f                         f
f
1   23   F            f                  f                         f
f
2   46   M            f                  f                         f
f
3   70   F            t                  f                         f
f
4   70   F            f                  f                         f
f

  pregnant thyroid surgery I131 treatment query hypothyroid  ... TT4
measured  \
0        f               f              f                 f  ...
t
1        f               f              f                 f  ...
t
2        f               f              f                 f  ...
t
3        f               f              f                 f  ...
t
4        f               f              f                 f  ...
t

     TT4 T4U measured   T4U FTI measured    FTI TBG measured  TBG  \
0  125.0            t  1.14            t  109.0            f    0
1  102.0            f  0.00            f    0.0            f    0
2  109.0            t  0.91            t  120.0            f    0
```

```
3  175.0            f  0.00            f    0.0            f    0
4   61.0            t  0.87            t   70.0            f    0

  referral source     label
0            SVHC  negative
1           other  negative
2           other  negative
3           other  negative
4             SVI  negative

[5 rows x 30 columns]
```

```
data.shape
```

```
(2800, 30)
```

```
data.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
label                        0
dtype: int64
```

```
count1 = (data['sex'] == '0').sum()
print(count1)
```

```
110
```

replacing the null values in sex to the most common sex

```
most_common_sex = data['sex'].mode()[0]
# print(most_common_sex)
#i.e. = F
data['sex'] = data['sex'].replace('0', most_common_sex)

count2 = (data['sex'] == '0').sum()
print(count2)
```

```
0
```

```
print(data.dtypes)
```

```
age                              int64
sex                             object
on thyroxine                    object
query on thyroxine              object
on antithyroid medication       object
sick                            object
pregnant                        object
thyroid surgery                 object
I131 treatment                  object
query hypothyroid               object
query hyperthyroid              object
lithium                         object
goitre                          object
tumor                           object
hypopituitary                   object
psych                           object
TSH measured                    object
TSH                            float64
T3 measured                     object
T3                             float64
TT4 measured                    object
TT4                            float64
T4U measured                    object
T4U                            float64
FTI measured                    object
FTI                            float64
TBG measured                    object
TBG                              int64
referral source                 object
label                           object
dtype: object
```

```
for cols in data.columns:
  print(f'{cols}: {data[cols].unique()}')
  print()
```

```
age: [ 41  23  46  70  18  59  80  66  68  84  67  71  28  65  42  63
51  81
  54  55  60  25  73  34  78  37  85  26  58  64  44  48  61  35  83
21
  87  53  77  27  69  74  38  76  45  36  22  43  72  82  31  39  49
62
  57   1  50  30  29  75  19   7  79  17  24  15  32  47  16  52  33
13
  10  89  56  20  90  40  88  14  86  94  12   4  11   8   5 455   2
91
   6   0  93  92]

sex: ['F' 'M']

on thyroxine: ['f' 't']

query on thyroxine: ['f' 't']

on antithyroid medication: ['f' 't']

sick: ['f' 't']

pregnant: ['f' 't']

thyroid surgery: ['f' 't']

I131 treatment: ['f' 't']

query hypothyroid: ['f' 't']

query hyperthyroid: ['f' 't']

lithium: ['f' 't']

goitre: ['f' 't']

tumor: ['f' 't']

hypopituitary: ['f' 't']

psych: ['f' 't']

TSH measured: ['t' 'f']

TSH: [1.30e+00 4.10e+00 9.80e-01 1.60e-01 7.20e-01 3.00e-02 0.00e+00
2.20e+00
 6.00e-01 2.40e+00 1.10e+00 2.80e+00 3.30e+00 1.20e+01 1.20e+00
```

```
1.50e+00
 6.00e+00 2.10e+00 1.00e-01 8.00e-01 1.90e+00 3.10e+00 2.00e-01
1.30e+01
 3.00e-01 3.50e-02 2.50e+00 5.00e-01 1.70e+00 7.30e+00 1.80e+00 2.60e-
01
 4.50e+01 5.40e+00 9.90e-01 2.50e-01 9.20e-01 1.50e-01 6.40e-01
1.00e+00
 4.00e-01 2.00e+00 2.60e+00 1.48e+01 1.50e+01 1.90e+01 2.00e-02
3.00e+00
 2.90e+00 3.20e+00 9.00e+00 1.60e+00 4.30e+00 5.00e-03 3.10e-01 6.10e-
01
 5.00e-02 7.80e+00 1.60e+02 2.50e-02 1.40e+00 1.00e-02 8.80e+00
1.51e+02
 4.00e-02 3.90e+00 9.40e+00 2.70e+00 2.30e+00 9.40e-01 4.50e-02
3.50e+00
 8.80e-01 8.00e-02 4.50e+00 6.80e-01 7.00e-01 6.70e-01 2.70e+01
6.10e+00
 7.50e-01 5.50e-01 2.60e+01 5.20e+00 7.70e-01 7.00e-02 9.00e-01
1.14e+01
 1.43e+02 4.50e-01 5.70e-01 6.50e-01 1.50e-02 1.60e+01 1.08e+02 8.30e-
01
 9.20e+00 8.60e+01 6.20e-01 5.90e-01 9.10e+00 5.90e+00 5.20e+01 3.30e-
01
 3.10e+01 5.80e+00 2.80e-01 5.10e+01 6.30e+00 4.40e+00 9.60e+00
3.40e+00
 9.00e-02 2.40e+01 7.60e-01 4.20e+01 2.50e+01 1.00e+01 4.60e+00
8.60e+00
 6.60e-01 6.20e+00 7.90e-01 2.80e+01 8.60e-01 9.70e+00 8.40e-01
1.70e+01
 1.80e+01 5.50e+01 1.40e+01 3.70e+00 8.70e-01 6.70e+00 7.40e-01
7.60e+00
 6.50e-02 2.90e-01 3.70e-01 8.00e+00 1.10e+01 4.80e-01 4.40e+01
7.90e+00
 5.00e+00 7.20e+00 8.90e-01 9.30e-01 9.70e-01 1.20e-01 6.40e+00
3.30e+01
 8.50e-01 7.10e+00 7.30e-01 1.99e+02 8.20e+00 1.88e+02 2.20e-01
9.80e+01
 2.20e+01 6.60e+00 5.10e+00 6.00e-02 4.20e-01 3.80e+00 3.50e+01
4.00e+00
 7.80e-01 6.30e-01 5.20e-01 6.00e+01 4.30e-01 5.60e+00 6.90e+00
3.60e+00
 2.90e+01 3.80e-01 4.90e+00 4.10e-01 9.90e+00 7.50e+00 3.40e+01
6.50e+00
 4.70e+00 1.03e+02 9.50e-01 1.40e-01 3.50e-01 4.20e+00 8.10e-01 5.40e-
01
 5.80e-01 8.90e+00 5.50e+00 3.40e-01 9.30e+00 1.30e-01 5.40e+01 3.90e-
01
 8.30e+00 4.78e+02 2.10e+01 6.80e+00 3.20e-01 2.30e-01 2.40e-01
8.10e+00
```

```
 9.10e-01 5.30e+00 1.00e+02 2.70e-01 1.01e+00 5.80e+01 4.10e+01
1.83e+02
 1.84e+01 4.70e-01 1.70e-01 1.21e+01 1.90e-01 8.20e-01 4.30e+01 4.40e-
01
 7.00e+01 7.70e+00 8.40e+00 6.90e-01 8.50e+00 2.10e-01 8.20e+01 5.50e-
02
 9.60e-01 7.10e-01 3.80e+01 3.60e-01 9.80e+00 7.00e+00 4.60e-01
1.11e+01
 3.90e+01 7.60e+01 5.70e+00 3.20e+01 1.26e+02 2.64e+01 5.30e-01 4.90e-
01
 3.60e+01 1.78e+02 1.45e+02 4.70e+01 4.80e+00 1.03e+01 8.90e+01
7.40e+00
 4.72e+02 5.10e-01 1.16e+02 6.10e+01 9.90e+01 4.60e+01 7.80e+01
4.68e+02]

T3 measured: ['t' 'f']

T3: [ 2.5   2.    0.    1.9   1.2   0.6   2.2   1.6   3.8   1.7   1.8
2.6
  2.1   0.3   5.5   1.4   3.1   1.5   2.3   2.4   2.7   0.9   1.
2.8
  2.9   0.8   1.3   0.4   3.3   3.5   3.4   1.1   4.2   3.7   3.
0.7
  4.8   4.3   0.05  3.2   5.4   4.    0.5   0.2   3.6   5.2   5.    6.
  5.3   3.9   4.6   4.5   7.3   4.7   6.7   4.1   6.1   0.1   4.9
10.6
  5.1   7.    6.2   4.4   7.1 ]

TT4 measured: ['t' 'f']

TT4: [125.  102.  109.  175.   61.  183.   72.   80.  123.   83.  115.
152.
 171.   97.   99.   70.  117.  121.  130.  108.  104.  134.  199.
57.
 129.  113.  119.   84.   81.   95.   66.  101.  147.  120.   69.
0.
  39.   87.   63.  133.   86.  163.  162.  103.   96.  151.  112.
82.
 138.   71.   77.   93.  107.  237.  110.   67.   88.  160.  118.
136.
 114.  116.   94.  161.   11.   32.  124.  137.   92.  135.  105.
150.
 126.  146.   91.  217.  141.  159.  122.  100.  111.  140.  205.
225.
  85.   90.   74.  219.  127.  132.  128.  106.  144.  131.   56.
79.
 142.   98.  177.  139.   78.  189.  180.   73.  145.  184.   38.
156.
  75.  148.   14.   76.   54.   58.   27.   65.  193.   13.  143.
12.
```

```
  64.  257.  164.   59.  167.   18.   41.  176.   37.   33.   44.
45.
 154.  174.  203.  244.   62.  158.   60.  187.  250.  181.  157.
223.
 272.  166.  213.  235.   10.   68.  231.  191.   48.    5.8 169.
149.
 210.   40.  155.  232.   42.  204.  430.  198.  230.   15.  170.
165.
  47.  168.  194.   89.   52.  179.  192.  172.    4.8  50.  182.
197.
 214.  246.  196.  207.   19.  153.   22.   46.  200.   35.  226.
201.
 233.  206.   31.  255.  178.  239.  195.    6.   36.    2.    3.
289.
 240.  209.   43.   34.  252.   29.  263.  301.   23.  188.  211.
253.
  21.  173. ]

T4U measured: ['t' 'f']

T4U: [1.14  0.    0.91  0.87  1.3   0.92  0.7   0.93  0.89  0.95  0.99
1.13
 0.86  0.96  0.94  0.9   1.02  1.05  0.62  1.06  1.55  0.83  1.09
1.07
 1.27  0.76  1.16  1.    0.56  0.81  0.68  0.78  0.85  1.35  1.15
0.82
 1.03  1.58  0.79  1.17  0.71  0.72  0.88  1.11  1.2   1.1   1.33
0.77
 1.24  0.53  1.44  1.63  1.51  1.42  1.23  1.01  0.98  0.61  1.12
1.43
 1.25  1.41  1.68  0.97  0.84  0.8   1.04  0.73  1.08  1.26  1.46
1.29
 1.34  1.66  1.21  1.19  0.75  0.52  1.83  1.39  1.5   1.93  1.18
0.74
 0.58  1.82  0.6   1.67  1.22  0.66  0.67  1.31  0.54  1.77  1.59
1.97
 1.69  1.38  1.28  1.4   0.69  0.65  1.74  2.03  1.73  1.65  1.36
1.52
 0.57  1.53  1.84  1.57  1.75  1.32  1.37  0.64  1.79  1.8   0.48
1.71
 1.62  1.76  1.56  1.48  0.59  0.31  1.94  2.12  1.47  0.63  0.944
0.49
 1.88  0.5   0.38  1.49  0.41  1.61  1.7  ]

FTI measured: ['t' 'f']

FTI: [109.    0.  120.   70.  141.   78.  115.  132.   93.  121.  153.
151.
 107.  119.   87.   81.  104.  130.  106.  116.  131.  190.   92.
102.
```

```
  76.   98.   90.   61.   94.  129.   95.   91.   33.  113.  148.
140.
 171.  155.  186.  122.  136.  110.  111.   97.   72.  100.   88.
67.
  84.  103.  135.  203.  112.  117.  180.  142.  145.  156.   96.
134.
   8.9  60.  139.   41.   99.   89.  146.  124.  105.   85.  157.
143.
  71.  221.   28.  108.  137.   83.   74.  170.   65.  101.  127.
274.
 154.  114.   62.   86.  126.  125.   64.  172.  162.   79.  118.
73.
 152.  163.  149.   14.   51.  165.   77.   32.   69.   80.   11.
54.
 164.  123.  144.   10.  214.  200.  160.   53.   16.  138.  169.
56.
  47.  133.   43.   68.  179.  224.  220.   82.  362.  182.   75.
66.
 161.   57.   58.  312.   63.  128.  147.  158.  281.  207.  216.
251.
 194.   46.    7.   42.  174.  395.  185.   13.  201.   48.  173.
167.
 188.  150.  235.  175.  159.    5.4 189.   59.  166.   34.  228.
232.
 217.  177.  176.  195.  219.   17.  210.  168.  205.   39.  187.
50.
 349.   52.  206.  253.  242.  244.  213.  178.  247.  215.  198.
19.
 237.   37.    7.6  24.    2.    3.  191.  223.    9.   29.  222.
204.
  26.  218.  197.   49.  209.  183. ]

TBG measured: ['f']

TBG: [0]

referral source: ['SVHC' 'other' 'SVI' 'STMW' 'SVHD']

label: ['negative' 'hyperthyroid' 'T3 toxic' 'goitre']
```

```
len(data['label'])
```

```
2800
```

```
count1 = (data['label'] == 'negative').sum()
count2 = (data['label'] == 'hyperthyroid').sum()
count3 = (data['label'] == 'goitre').sum()
count4 = (data['label'] == 'T3 toxic').sum()
print("count of negative: ", count1)
```

```
print("count of hyperthyroid: ", count2)
print("count of goitre: ", count3)
print("count of T3 toxic: ", count4)
```

```
count of negative:  2723
count of hyperthyroid:  62
count of goitre:  7
count of T3 toxic:  8
```

```
data.describe()
```

```
               age          TSH           T3          TT4          T4U  \
count  2800.000000  2800.000000  2800.000000  2800.000000  2800.000000

mean     51.825714     4.198261     1.601893   101.904786     0.892062

std      20.480953    20.381055     1.102638    43.599948     0.358101

min       0.000000     0.000000     0.000000     0.000000     0.000000

25%      36.000000     0.200000     0.800000    84.000000     0.830000

50%      54.000000     1.200000     1.800000   102.000000     0.955000

75%      67.000000     2.400000     2.300000   123.000000     1.070000

max     455.000000   478.000000    10.600000   430.000000     2.120000


               FTI     TBG
count  2800.000000  2800.0
mean     99.115679     0.0
std      46.094566     0.0
min       0.000000     0.0
25%      86.750000     0.0
50%     104.000000     0.0
75%     122.000000     0.0
max     395.000000     0.0
```

```
#@title Dropping redundant columns
new_data = data.drop(['TBG', 'TBG measured', 'referral source', 
'age'], axis = 1)
# Since both of the data points are constant and are of no use in the 
model.
```

```
new_data.nunique()
```

```
age                        94
sex                         2
on thyroxine                2
```

```
query on thyroxine              2
on antithyroid medication       2
sick                            2
pregnant                        2
thyroid surgery                 2
I131 treatment                  2
query hypothyroid               2
query hyperthyroid              2
lithium                         2
goitre                          2
tumor                           2
hypopituitary                   2
psych                           2
TSH measured                    2
TSH                           264
T3 measured                     2
T3                             65
TT4 measured                    2
TT4                           218
T4U measured                    2
T4U                           139
FTI measured                    2
FTI                           210
label                           4
dtype: int64
```

```python
#@title Label encoding the categorical columns

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for col in new_data.columns:
    if new_data[col].nunique() == 2:  # Check if there are exactly 2 unique values
        new_data[col] = le.fit_transform(new_data[col])

new_data.head(10)
```

```
   age  sex  on thyroxine  query on thyroxine  on antithyroid medication  \
0   41    0             0                   0
0
1   23    0             0                   0
0
2   46    1             0                   0
0
3   70    0             1                   0
0
4   70    0             0                   0
```

```
0
5   18    0             1                  0
0
6   59    0             0                  0
0
7   80    0             0                  0
0
8   66    0             0                  0
0
9   68    1             0                  0
0

   sick  pregnant  thyroid surgery  I131 treatment  query hypothyroid
...  \
0     0         0                0               0                  0
...
1     0         0                0               0                  0
...
2     0         0                0               0                  0
...
3     0         0                0               0                  0
...
4     0         0                0               0                  0
...
5     0         0                0               0                  0
...
6     0         0                0               0                  0
...
7     0         0                0               0                  0
...
8     0         0                0               0                  0
...
9     0         0                0               0                  0
...

    TSH  T3 measured   T3  TT4 measured    TT4  T4U measured   T4U  \
0  1.30            1  2.5             1  125.0             1  1.14
1  4.10            1  2.0             1  102.0             0  0.00
2  0.98            0  0.0             1  109.0             1  0.91
3  0.16            1  1.9             1  175.0             0  0.00
4  0.72            1  1.2             1   61.0             1  0.87
5  0.03            0  0.0             1  183.0             1  1.30
6  0.00            0  0.0             1   72.0             1  0.92
7  2.20            1  0.6             1   80.0             1  0.70
8  0.60            1  2.2             1  123.0             1  0.93
9  2.40            1  1.6             1   83.0             1  0.89

   FTI measured    FTI     label
0             1  109.0  negative
1             0    0.0  negative
```

```
2               1  120.0  negative
3               0    0.0  negative
4               1   70.0  negative
5               1  141.0  negative
6               1   78.0  negative
7               1  115.0  negative
8               1  132.0  negative
9               1   93.0  negative

[10 rows x 27 columns]
```

```
#@title Normalising the data using StandardScaler

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

for col in new_data.columns:
    if new_data[col].nunique() > 2 and new_data[col].dtype == 
'float64':
        new_data[col] = 
scaler.fit_transform(new_data[col].values.reshape(-1, 1))

new_data.head(10)
```

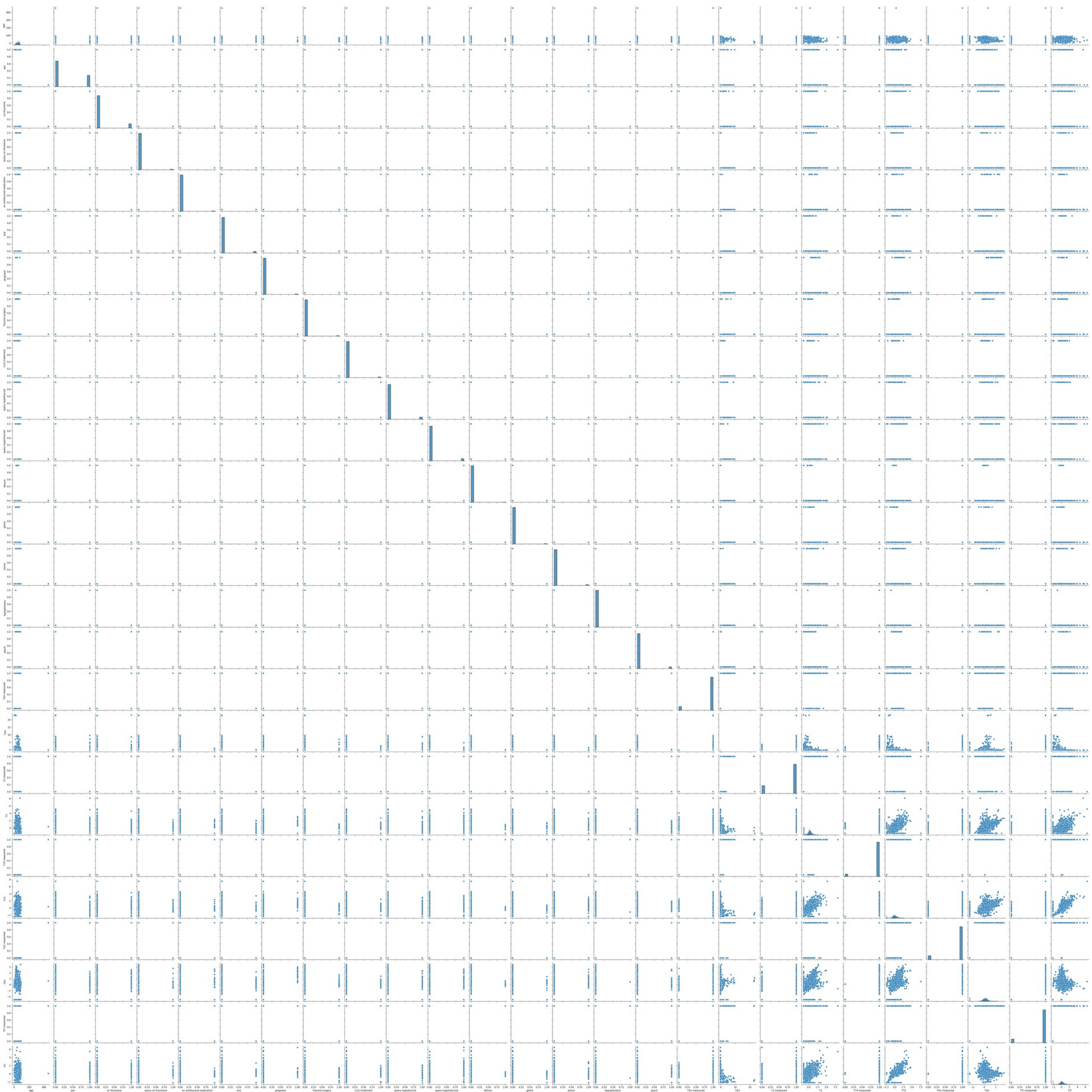
```
   age  sex  on thyroxine  query on thyroxine  on antithyroid 
medication  \
0   41    0             0                   0
0
1   23    0             0                   0
0
2   46    1             0                   0
0
3   70    0             1                   0
0
4   70    0             0                   0
0
5   18    0             1                   0
0
6   59    0             0                   0
0
7   80    0             0                   0
0
8   66    0             0                   0
0
9   68    1             0                   0
0

   sick  pregnant  thyroid surgery  I131 treatment  query hypothyroid 
...  \
```

```
0     0         0                 0               0                   0
...
1     0         0                 0               0                   0
...
2     0         0                 0               0                   0
...
3     0         0                 0               0                   0
...
4     0         0                 0               0                   0
...
5     0         0                 0               0                   0
...
6     0         0                 0               0                   0
...
7     0         0                 0               0                   0
...
8     0         0                 0               0                   0
...
9     0         0                 0               0                   0
...

        TSH  T3 measured        T3  TT4 measured       TT4  T4U
measured  \
0 -0.142229            1  0.814653             1  0.529802
1
1 -0.004822            1  0.361114             1  0.002184
0
2 -0.157933            0 -1.453041             1  0.162764
1
3 -0.198173            1  0.270406             1  1.676797
0
4 -0.170692            1 -0.364548             1 -0.938352
1
5 -0.204553            0 -1.453041             1  1.860316
1
6 -0.206025            0 -1.453041             1 -0.686013
1
7 -0.098063            1 -0.908795             1 -0.502494
1
8 -0.176581            1  0.542530             1  0.483922
1
9 -0.088248            1 -0.001717             1 -0.433674
1

        T4U  FTI measured       FTI     label
0  0.692492             1  0.214474  negative
1 -2.491534             0 -2.150652  negative
2  0.050101             1  0.453156  negative
3 -2.491534             0 -2.150652  negative
```

```
4 -0.061620             1 -0.631764  negative
5  1.139372             1  0.908823  negative
6  0.078031             1 -0.458177  negative
7 -0.536431             1  0.344664  negative
8  0.105961             1  0.713537  negative
9 -0.005760             1 -0.132700  negative

[10 rows x 27 columns]
```

```python
import seaborn as sns

sns.pairplot(new_data)
```

```
<seaborn.axisgrid.PairGrid at 0x7b31fcfe0be0>
```

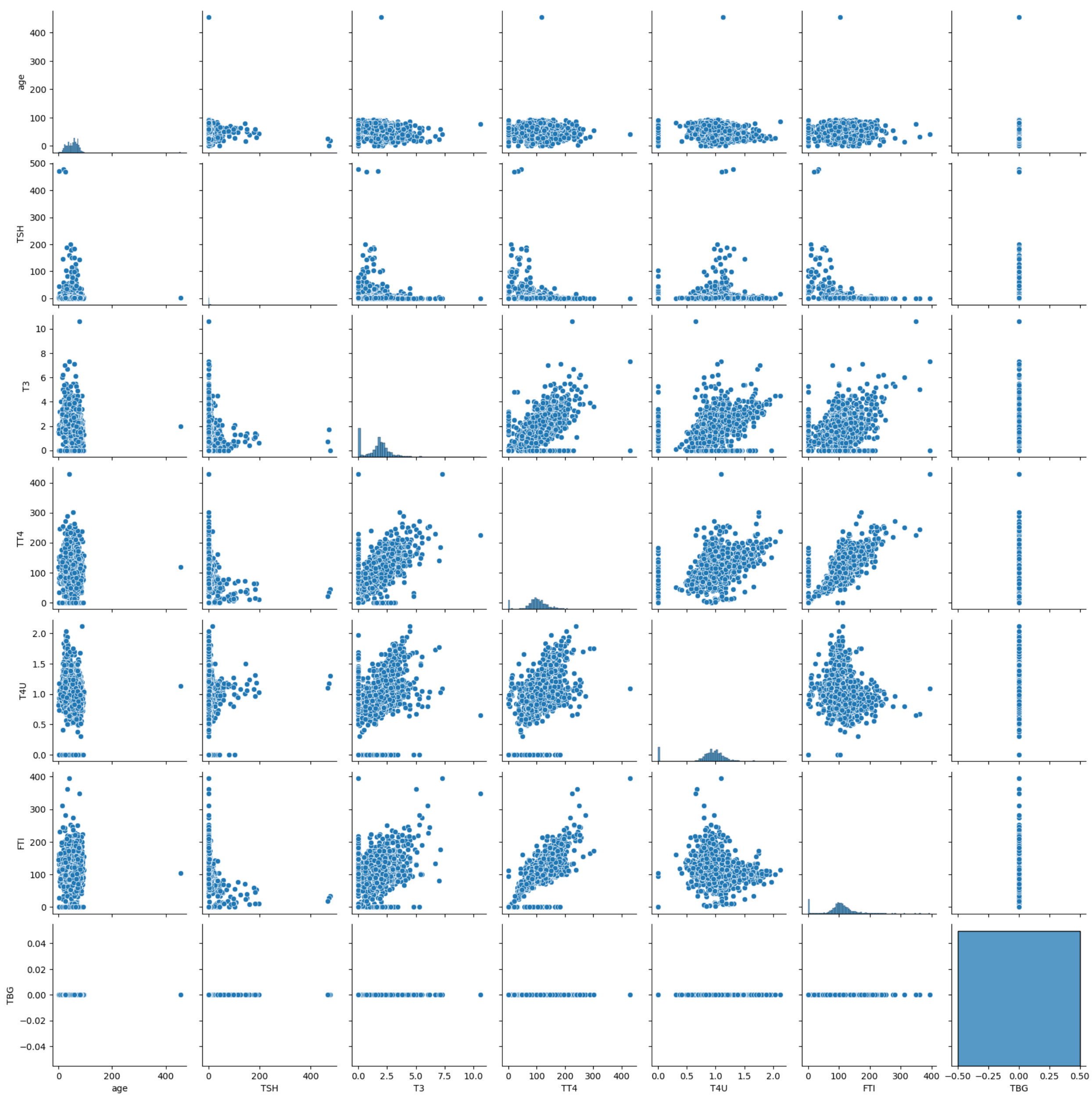


```
new_data.shape
```

```
(2800, 27)
```

```
sns.pairplot(data)
```

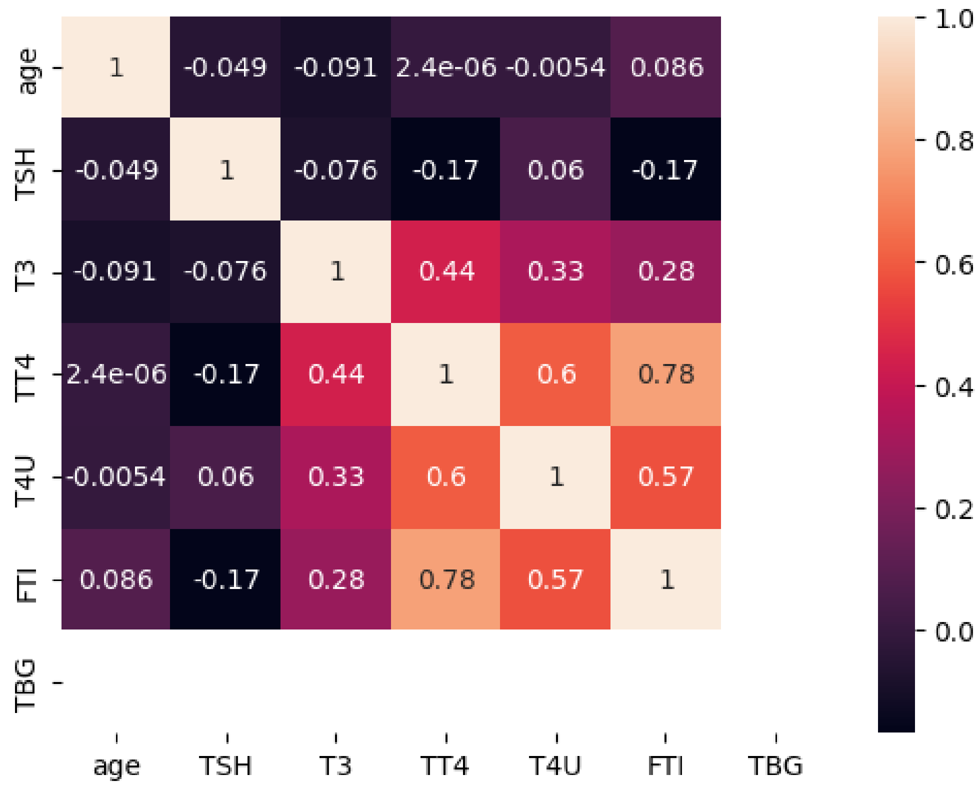
```
<seaborn.axisgrid.PairGrid at 0x7b31e35f9570>
```



```
#@title EDA
import seaborn as sns

correlation = data.corr()
# sns.heatmap(correlation, annot=True, cmap='coolwarm')

sns.heatmap(correlation, xticklabels=correlation.columns,
yticklabels=correlation.columns, annot=True)
```

```
<ipython-input-47-576286b99232>:4: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
```

```
of numeric_only to silence this warning.
  correlation = data.corr()

<Axes: >
```



First attempt

```
# import numpy as np
# import pandas as pd

# #@title Node Class

# class Node():
#     def __init__(self, feature=None, threshold=None, left=None, right=None, ig=None, value=None):
#         '''Constructor for the Node class'''
#         # For decision nodes
#         self.left = left
#         self.right = right
#         self.feature = feature
#         self.threshold = threshold
#         self.ig = ig

#         # For leaf nodes
#         self.value = value
```

```
# #@title Decision Tree Class

# class DecisionTree():
#     def __init__(self, minSamplesSplit=2, maxDepth=2):
#         '''Constructor for the DecisionTree class'''
#         # Initialize the root of the tree to be None
#         self.root = None

#         # Stopping conditions
#         self.minSamplesSplit = minSamplesSplit
#         self.maxDepth = maxDepth

#     def buildTree(self, datapoint, currDepth=0):
#         '''Recursive method which builds out the decision tree and
splits the data'''
#         X, Y = datapoint[:,:-1], datapoint[:,-1]
#         numSamples, numFeatures = np.shape(X)

#         # Split
#         if numSamples >= self.minSamplesSplit and currDepth <=
self.maxDepth:
#             # Best split
#             bestSplit = self.getBestSplit(datapoint, numSamples,
numFeatures)
#             # Checking if information gain is positive
#             if bestSplit["ig"] > 0:
#                 # Recur left
#                 leftSubtree = self.buildTree(bestSplit["leftData"],
currDepth+1)
#                 # Recur right
#                 rightSubtree =
self.buildTree(bestSplit["rightData"], currDepth+1)
#                 # Return decision node
#                 return Node(bestSplit["feature"],
bestSplit["threshold"], leftSubtree, rightSubtree, bestSplit["ig"])

#         # Leaf node
#         leafValue = self.calculateLeafValue(Y)

#         return Node(value=leafValue)

#     def getBestSplit(self, datapoint, numSamples, numFeatures):
#         '''Method to find the best split'''
#         bestSplit = {}
#         maxIG = -float("inf")

#         for feature in range(numFeatures):
#             featureValues = datapoint[:, feature]
#             possibleThresholds = np.unique(featureValues)
```

```
#             for threshold in possibleThresholds:
#                 leftData, rightData = self.make_split(datapoint, 
feature, threshold)

#                 # Check if children are not null
#                 if len(leftData) > 0 and len(rightData) > 0:
#                     y, leftY, rightY = datapoint[:, -1], leftData[:, 
-1], rightData[:, -1]
#                     # Information gain
#                     currIG = self.cost_function(y, leftY, rightY, 
"gini")

#                     # Update the best split if needed
#                     if currIG > maxIG:
#                         bestSplit["feature"] = feature
#                         bestSplit["threshold"] = threshold
#                         bestSplit["leftData"] = leftData
#                         bestSplit["rightData"] = rightData
#                         bestSplit["ig"] = currIG
#                         maxIG = currIG

#         return bestSplit

#     def cost_function(self, parent, lChild, rChild, mode="entropy"):
#         '''Method to compute information gain'''
#         weight_l = len(lChild) / len(parent)
#         weight_r = len(rChild) / len(parent)

#         if mode == "giniIndex":
#             gain = self.giniIndex(parent) - (weight_l * 
self.giniIndex(lChild) + weight_r * self.giniIndex(rChild))
#         else:
#             gain = self.entropy(parent) - (weight_l * 
self.entropy(lChild) + weight_r * self.entropy(rChild))

#         return gain

#     def make_split(self, datapoint, feature, threshold):
#         '''Method to split the data'''
#         leftData = np.array([row for row in datapoint if 
row[feature] <= threshold])
#         rightData = np.array([row for row in datapoint if 
row[feature] > threshold])

#         return leftData, rightData

#     def max_depth(self, node):
#         '''Method to calculate the maximum depth of the tree'''
#         if node is None:
```

```
#             return 0

#         leftDepth = self.max_depth(node.left)
#         rightDepth = self.max_depth(node.right)

#         return max(leftDepth, rightDepth) + 1

#     def pruning(self, pruningFactor):
#         '''Method to prune the tree'''
#         self.max_depth = self.max_depth(self.root)
#         numNodes = 0.5 * pruningFactor * self.max_depth
#         self.prune(self.root, numNodes)

#     def prune(self, node, numNodes):
#         '''Method to prune the tree'''
#         if node.left:
#             if node.left.value is None:
#                 self.prune(node.left, numNodes)
#         if node.right:
#             if node.right.value is None:
#                 self.prune(node.right, numNodes)

#         if node.left.value is not None and node.right.value is not
None:
#             leftValue, rightValue = node.left.value,
node.right.value
#             node.left, node.right = None, None
#             node.value = max(leftValue, rightValue, key=lambda x:
leftValue.count + rightValue.count)

#         if node.value is not None:
#             numNodes -= 1

#         return numNodes

#     def predict(self, X_test):
#       '''Method to predict the class labels'''
#       predictions = [self.prediction(x, self.root) for x in X_test]
#       return predictions

#     def prediction(self, x, tree):
#       if tree.value is not None: return tree.value

#       featureVal = x[tree.feature]

#       if featureVal <= tree.threshold:
#           return self.prediction(x, tree.left)
#       else:
#           return self.prediction(x, tree.right)
```

```
#     def score(self, X_test, y_test):
#         '''Method to calculate accuracy score'''
#         predictions = self.predict(X_test)
#         correct_predictions = 0

#         for i in range(len(predictions)):
#             if predictions[i] == y_test[i]:
#                 correct_predictions += 1

#         accuracy = correct_predictions / len(y_test)

#         return accuracy


#     def entropy(self, y):
#         '''Method to compute entropy'''
#         entropy = 0
#         classLabels = np.unique(y)

#         for labels in classLabels:
#             probLabels = len(y[y == labels]) / len(y)
#             entropy += -probLabels * np.log2(probLabels)

#         return entropy

#     def giniIndex(self, y):
#         '''Method to compute Gini index'''
#         gini = 0
#         classLabels = np.unique(y)

#         for labels in classLabels:
#             probLabels = len(y[y == labels]) / len(y)
#             gini += probLabels**2

#         return 1 - gini  # (since the Gini index ranges from 0 to 1
it can be achieved by subtracting from 1)

#     def calculateLeafValue(self, Y):
#         '''Method to calculate leaf value'''
#         Y = list(Y)
#         return max(Y, key=Y.count)

#     def printTree(self, tree=None, indent=" "):
#         '''Method to print the tree'''
#         if not tree:
#             tree = self.root

#         if tree.value is not None:
#             print(tree.value)
```

```
#         else:
#             print(f"{tree.feature} <= {tree.threshold}?")
#             print(f"{indent}T->", end="")
#             self.printTree(tree.left, indent + " ")
#             print(f"{indent}F->", end="")
#             self.printTree(tree.right, indent + " ")

#     def fit(self, X, Y):
#       '''Method to train the decision tree'''
#       self.root = self.buildTree(np.concatenate((X, Y.reshape(-1, 
1)), axis = 1))
```

Final Code

```
import numpy as np
import pandas as pd

class Node():
    def __init__(self, feature=None, threshold=None, left=None, 
right=None, gini=None, value=None):
        '''Constructor for the Node class'''
        # For decision nodes
        self.left = left
        self.right = right
        self.feature = feature
        self.threshold = threshold
        self.gini = gini

        # For leaf nodes
        self.value = value

class MyDecisionTree():
    def __init__(self, min_samples_split=2, max_depth=None):
        '''Constructor for the MyDecisionTree class'''
        # Initialize the root of the tree to be None
        self.root = None

        # Stopping conditions
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def buildTree(self, X, Y, curr_depth=0):
        '''Recursive method which builds out the decision tree and 
splits the data'''
        num_samples, num_features = np.shape(X)

        # Stopping conditions
        if (curr_depth >= self.max_depth) or (num_samples < 
self.min_samples_split) or (len(np.unique(Y)) == 1):
```

```
            leaf_value = self.calculateLeafValue(Y)
            return Node(value=leaf_value)

        best_split = self.getBestSplit(X, Y)
        left_data, right_data = best_split["leftData"],
best_split["rightData"]

        if len(left_data) == 0 or len(right_data) == 0:
            leaf_value = self.calculateLeafValue(Y)
            return Node(value=leaf_value)

        left_subtree = self.buildTree(left_data[:, :-1], left_data[:,
-1], curr_depth + 1)
        right_subtree = self.buildTree(right_data[:, :-1],
right_data[:, -1], curr_depth + 1)

        return Node(best_split["feature"], best_split["threshold"],
left_subtree, right_subtree, best_split["gini"])

    def getBestSplit(self, X, Y):
        '''Method to find the best split'''
        num_samples, num_features = np.shape(X)
        best_split = {}
        min_gini = float('inf')

        for feature in range(num_features):
            feature_values = X[:, feature]
            unique_thresholds = np.unique(feature_values)

            for threshold in unique_thresholds:
                left_data, right_data = self.make_split(X, Y, feature,
threshold)
                if len(left_data) > 0 and len(right_data) > 0:
                    gini = self.giniIndex(Y) - (len(left_data) /
num_samples) * self.giniIndex(left_data[:, -1]) - \
                            (len(right_data) / num_samples) *
self.giniIndex(right_data[:, -1])

                    if gini < min_gini:
                        best_split["feature"] = feature
                        best_split["threshold"] = threshold
                        best_split["leftData"] = left_data
                        best_split["rightData"] = right_data
                        best_split["gini"] = gini
                        min_gini = gini

        return best_split

    def cost_function(self, Y):
        '''Method to compute Gini impurity'''
```

```
        gini = 1.0
        class_labels = np.unique(Y)
        num_samples = len(Y)

        for label in class_labels:
            proportion = len(Y[Y == label]) / num_samples
            gini -= proportion ** 2

        return gini

    def make_split(self, X, Y, feature, threshold):
        '''Method to split the data'''
        left_data = np.array([row for row in np.hstack((X, Y.reshape(-
1, 1))) if row[feature] <= threshold])
        right_data = np.array([row for row in np.hstack((X,
Y.reshape(-1, 1))) if row[feature] > threshold])

        return left_data, right_data

    def max_depth(self, node):
        '''Method to calculate the maximum depth of the tree'''
        if node is None:
            return 0

        left_depth = self.max_depth(node.left)
        right_depth = self.max_depth(node.right)

        return max(left_depth, right_depth) + 1

    def predict(self, X_test):
        '''Method to predict the class labels'''
        predictions = [self.prediction(x, self.root) for x in X_test]
        return predictions

    def prediction(self, x, tree):
        if tree.value is not None:
            return tree.value

        feature_val = x[tree.feature]

        if feature_val <= tree.threshold:
            return self.prediction(x, tree.left)
        else:
            return self.prediction(x, tree.right)

    def score(self, X_test, y_test):
        '''Method to calculate accuracy score'''
        predictions = self.predict(X_test)
        correct_predictions = sum(1 for i in range(len(predictions))
if predictions[i] == y_test[i])
```

```
        accuracy = correct_predictions / len(y_test)
        return accuracy

    def giniIndex(self, Y):
        '''Method to compute Gini index'''
        gini = 1.0
        class_labels = np.unique(Y)
        num_samples = len(Y)

        for label in class_labels:
            proportion = len(Y[Y == label]) / num_samples
            gini -= proportion ** 2

        return gini

    def calculateLeafValue(self, Y):
        '''Method to calculate leaf value'''
        Y = list(Y)
        return max(Y, key=Y.count)

    def fit(self, X, Y):
        '''Method to train the decision tree'''
        self.root = self.buildTree(X, Y)

from sklearn.model_selection import train_test_split

X = new_data.drop(['label'], axis=1)
Y = new_data['label']

# Spliting the data into a training set (70%) and a testing set (30%)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.3, random_state=42)

# Create and train your decision tree model using MyDecisionTree
decision_tree = MyDecisionTree(max_depth=2)
decision_tree.fit(X_train.values, Y_train.values)  # Convert Pandas
Series to NumPy arrays

# Make predictions on the test data
Y_pred = decision_tree.predict(X_test.values)

# Calculate accuracy using the score method
accuracy = decision_tree.score(X_test.values, Y_test.values)  #
Convert Pandas Series to NumPy arrays

# Print the accuracy
print("Accuracy:", accuracy)
print(accuracy*100, "%")

Accuracy: 0.969047619047619
96.9047619047619 %
```

Section B (Library Implementation)

```
#@title Import libraries
import pandas as pd
import seaborn as sb
import numpy as np
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

#@title Reading Data
data = pd.read_csv('/content/processed.cleveland.data')

data.head()
```

```
   63.0  1.0  1.0.1  145.0  233.0  1.0.2  2.0  150.0  0.0  2.3  3.0
0.0.1  \
0  67.0  1.0    4.0  160.0  286.0    0.0  2.0  108.0  1.0  1.5  2.0
3.0
1  67.0  1.0    4.0  120.0  229.0    0.0  2.0  129.0  1.0  2.6  2.0
2.0
2  37.0  1.0    3.0  130.0  250.0    0.0  0.0  187.0  0.0  3.5  3.0
0.0
3  41.0  0.0    2.0  130.0  204.0    0.0  2.0  172.0  0.0  1.4  1.0
0.0
4  56.0  1.0    2.0  120.0  236.0    0.0  0.0  178.0  0.0  0.8  1.0
0.0

   6.0  0
0  3.0  2
1  7.0  1
2  3.0  0
3  3.0  0
4  3.0  0
```

```
data.shape
```

```
(302, 14)
```

```
data.isnull().sum()
```

```
63.0     0
1.0      0
1.0.1    0
145.0    0
233.0    0
1.0.2    0
2.0      0
150.0    0
0.0      0
```

```
2.3        0
3.0        0
0.0.1      0
6.0        0
0          0
dtype: int64
```

EDA

```
#@title Checing for NaN values in all columns
nan_check = data.isna()  # or df.isnull()
nan_counts = nan_check.sum()
print(nan_counts)
```

```
63.0       0
1.0        0
1.0.1      0
145.0      0
233.0      0
1.0.2      0
2.0        0
150.0      0
0.0        0
2.3        0
3.0        0
0.0.1      4
6.0        2
0          0
dtype: int64
```

```
#@title replacing nan with mean values
import pandas as pd
import numpy as np

# Replace "?" with NaN in the specified columns
data['0.0.1'].replace('?', np.nan, inplace=True)
data['6.0'].replace('?', np.nan, inplace=True)

# Converting the columns to numeric
data['0.0.1'] = pd.to_numeric(data['0.0.1'], errors='coerce')
data['6.0'] = pd.to_numeric(data['6.0'], errors='coerce')

# Calculating the mean for each column
mean_0_0_1 = data['0.0.1'].mean()
mean_6_0 = data['6.0'].mean()

# Replacing NaN values with the respective means
data['0.0.1'].fillna(mean_0_0_1, inplace=True)
data['6.0'].fillna(mean_6_0, inplace=True)
```

```python
nan_check = data.isna()  # or df.isnull()
nan_counts = nan_check.sum()
print(nan_counts)
```

```
63.0     0
1.0      0
1.0.1    0
145.0    0
233.0    0
1.0.2    0
2.0      0
150.0    0
0.0      0
2.3      0
3.0      0
0.0.1    0
6.0      0
0        0
dtype: int64
```

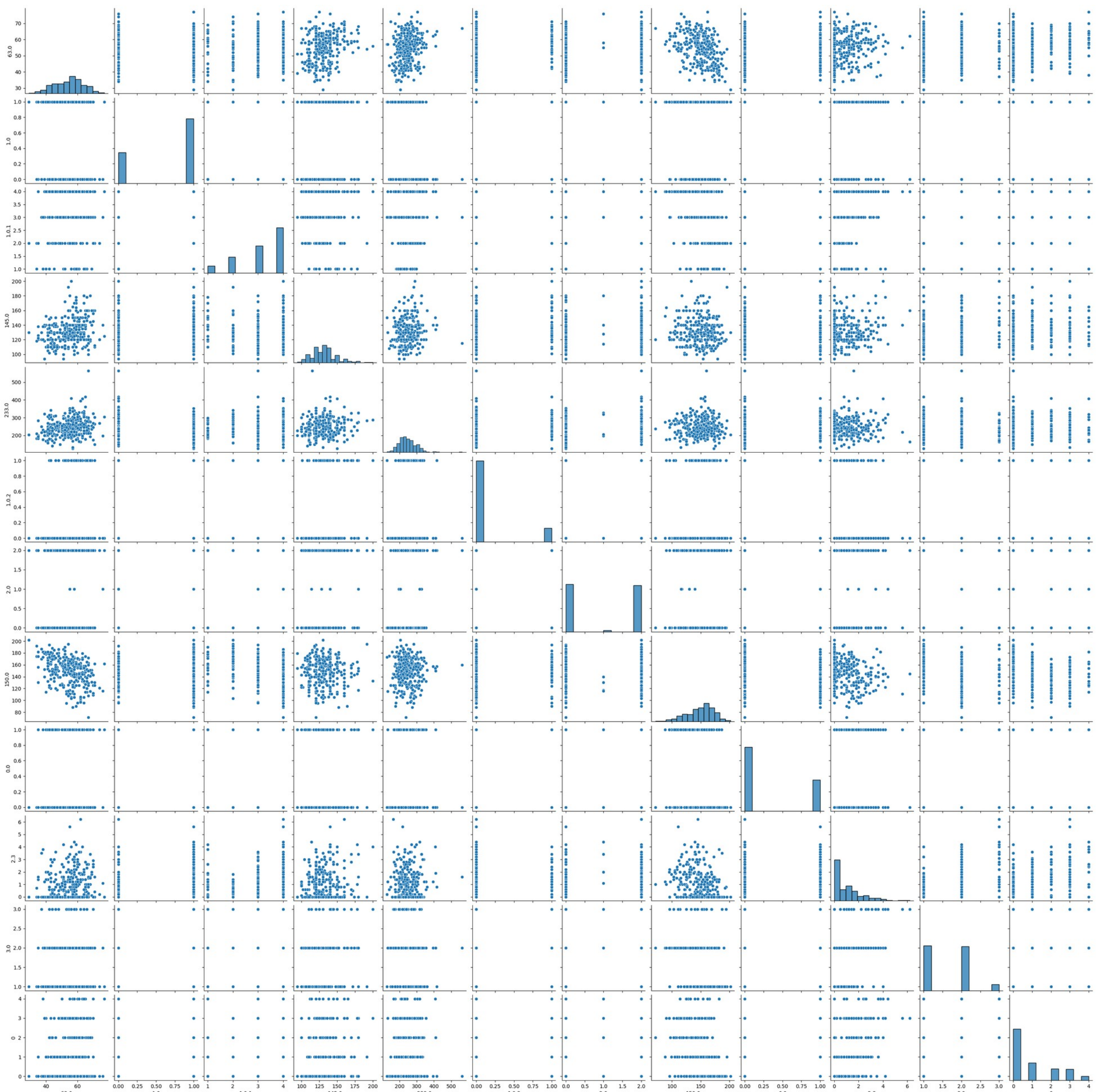
```python
data.dtypes
```

```
63.0     float64
1.0      float64
1.0.1    float64
145.0    float64
233.0    float64
1.0.2    float64
2.0      float64
150.0    float64
0.0      float64
2.3      float64
3.0      float64
0.0.1    float64
6.0      float64
0          int64
dtype: object
```

```python
import seaborn as sns
```

```python
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x7b31d237b9d0>
```



```
#@title b. and c.

# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

X = data.iloc[:, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]]
y = data.iloc[:, 13]

# Spliting the dataset into train and test sets in an 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```python
test_size=0.2, random_state=42)

accuracyScores = []

# Training decision trees using 'entropy' and 'gini' impurity as
splitting criteria
for criterion in ['entropy', 'gini']:

    clf = DecisionTreeClassifier(criterion=criterion, random_state=42)

    # Train the classifier on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = clf.predict(X_test)

    # Calculate the accuracy score for this criterion
    accuracy = accuracy_score(y_test, y_pred)

    # Append the accuracy score to the list
    accuracyScores.append((criterion, accuracy))

# Determine the best criterion for attribute selection based on
accuracy scores
bestCriterion, bestAccuracy = max(accuracyScores, key=lambda x: x[1])

print("--------------------------------------------------------------
---")
print("Accuracy Scores: ")
print("Entropy:", accuracyScores[0][1])
print("Gini:", accuracyScores[1][1])
print(f"The best criterion for attribute selection is
'{bestCriterion}' with an accuracy of {bestAccuracy:.4f}")
print("--------------------------------------------------------------
---")
```

```
-----------------------------------------------------------------
Accuracy Scores: 
Entropy: 0.45901639344262296
Gini: 0.4918032786885246
The best criterion for attribute selection is 'gini' with an accuracy
of 0.4918
-----------------------------------------------------------------
```

```python
#@title d.

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import warnings

# Creating a DecisionTreeClassifier
```

```python
dt_classifier = DecisionTreeClassifier(criterion='gini',
max_features='sqrt')  # Since 'gini' was found as the best criteria

# Defining the hyperparameter grid to search
param_grid = {
    'min_samples_split': [2, 5, 10, 20, 25],
    'max_features': ['auto', 'sqrt', 'log2', 'none']
}

# Creating GridSearch
grid_search = GridSearchCV(estimator=dt_classifier,
param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the model with different hyperparameter combinations
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_min_samples_split = grid_search.best_params_['min_samples_split']
best_max_features = grid_search.best_params_['max_features']

# Get the best accuracy score
best_accuracy = grid_search.best_score_

print(f'Best min_samples_split: {best_min_samples_split}')
print(f'Best max_features: {best_max_features}')
print(f'Best accuracy: {best_accuracy}')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
```

```
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
```

```
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/tree/_classes.py:269:
```

```
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and
will be removed in 1.3. To keep the past behaviour, explicitly set
`max_features='sqrt'`.
  warnings.warn(

Best min_samples_split: 25
Best max_features: sqrt
Best accuracy: 0.5853741496598639
```

```
#@title e.

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

# Creating a Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Defining the hyperparameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10, 20, 30],
    'min_samples_split': [2, 5, 10, 20, 25]
}

# Creating GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid, scoring='accuracy', cv=5)

# Fitting the model with different hyperparameter combinations
grid_search.fit(X_train, y_train)

# Getting the best hyperparameters
best_n_estimators = grid_search.best_params_['n_estimators']
best_max_depth = grid_search.best_params_['max_depth']
best_min_samples_split = grid_search.best_params_['min_samples_split']

# Training the Random Forest classifier with best hyperparameters
best_rf_classifier =
RandomForestClassifier(n_estimators=best_n_estimators,
max_depth=best_max_depth, min_samples_split=best_min_samples_split,
random_state=42)
best_rf_classifier.fit(X_train, y_train)

# predictions on the test data
y_pred = best_rf_classifier.predict(X_test)

# Generating a classification report
class_report = classification_report(y_test, y_pred, zero_division=0)

print(f'Best n_estimators: {best_n_estimators}')
```

```
print(f'Best max_depth: {best_max_depth}')
print(f'Best min_samples_split: {best_min_samples_split}')
print(f'Classification Report:\n{class_report}')
```

```
Best n_estimators: 300
Best max_depth: None
Best min_samples_split: 10
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.91      0.78        32
           1       0.00      0.00      0.00         9
           2       0.00      0.00      0.00         8
           3       0.00      0.00      0.00         9
           4       0.00      0.00      0.00         3

    accuracy                           0.48        61
   macro avg       0.14      0.18      0.16        61
weighted avg       0.36      0.48      0.41        61
```