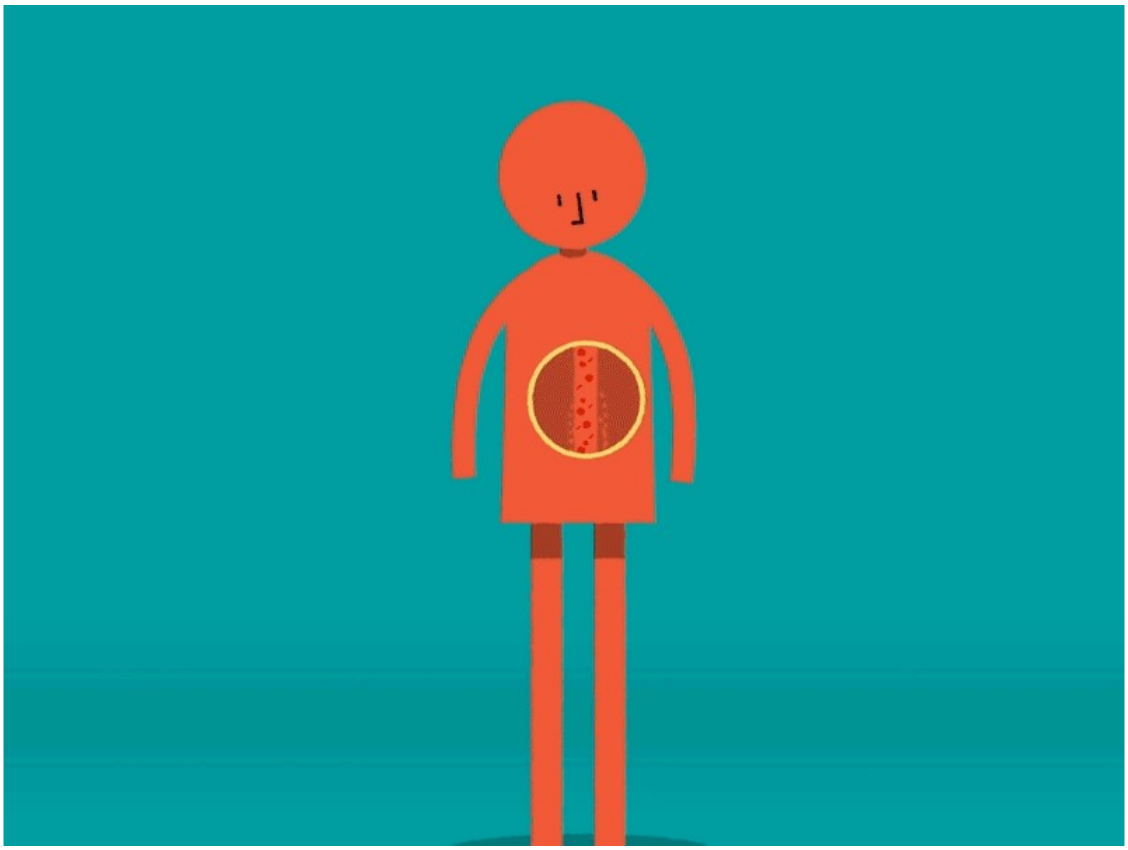
Student Name: Nguyen Dang Huynh Chau

Student ID: s3777214

Course: COSC2753 - Machine Learning

Lecturer: Dr. Nguyen Thien Bao



6. Conculsions

7. References

8. Appendix

✍ 1. Data Preparation

◌? 1.1 Introduction

? What will you get after this notebook?

◌? 1.2 Target question for insights

✴ 1.3 Importing Necessary Libraries and datasets

```
import sys
!{sys.executable} -m pip -q install missingno
!{sys.executable} -m pip -q install graphviz
!{sys.executable} -m pip -q install researchpy
!{sys.executable} -m pip -q install imbalanced-learn
```

```
# import libraries which are pandas and numpy
import pandas as pd
import numpy as np
import missingno as msno
```

```
#for plots
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"]= 15,10
```

```python
#Libraries for plotting
# Modules for data visualization
import seaborn as sns
import matplotlib.patches as mpatches
sns.set_theme(style="ticks", color_codes=True) #set theme in seaborn
# scatter matrix library
from pandas.plotting import scatter_matrix

#Libraries for feature scaling
from sklearn.preprocessing import StandardScaler

#Libraries for Validation
from sklearn.utils.multiclass import unique_labels
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score


from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

#Libraries for Training model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

Check numpy and pandas version

```python
# check the version of the packages
print("Numpy version: ", np.__version__)
print("Pandas version: ",pd.__version__)
! python --version
```

```
Numpy version:  1.20.3
Pandas version:  1.3.4
Python 3.9.7
```

------------> OBSERVATION

I want to check the numpy and pandas version since I want to make sure the verson is appropriate for my work load. Currently, it is still appropriate

? 1.4 Data Retrieving

Sample train Dataset

```
#Import data using functions of pandas
#Inside pandas.read_csv() method skipinitialspace parameter is use to skip initial space present in the dataframe.
#By default, it is False, so skipinitialspace must be True to skip the whitespace.
#data is imported by "read_csv() function of pandas"
train = pd.read_csv("Data/Paitients_Files_Train.csv", delimiter=',', skipinitialspace = True)

train.columns = train.columns.str.replace(' ', '') #strip the extra-whitespaces out

print("The shape of the ORGINAL data is (row, column):", str(train.shape))

# drop Unnamed, it is just a number given to identify each house
train.head(3)
```

```
The shape of the ORGINAL data is (row, column): (599, 11)
```

```
          ID  PRG   PL  PR  SK  TS   M11    BD2  Age  Insurance 
Sepssis
0  ICU200010    6  148  72  35   0  33.6  0.627   50          0 
Positive
1  ICU200011    1   85  66  29   0  26.6  0.351   31          0 
Negative
2  ICU200012    8  183  64   0   0  23.3  0.672   32          1 
Positive
```

Sample test Dataset

```
test = pd.read_csv("Data/Paitients_Files_Test.csv", delimiter=',', skipinitialspace = True)

test.columns = test.columns.str.replace(' ', '') #strip the extra-whitespaces out

print("The shape of the ORGINAL data is (row, column):", str(test.shape))

# drop Unnamed, it is just a number given to identify each house
test.head(3)
```

```
The shape of the ORGINAL data is (row, column): (169, 10)
```

```
          ID  PRG   PL  PR  SK   TS   M11    BD2  Age  Insurance
0  ICU200609    1  109  38  18  120  23.1  0.407   26          1
1  ICU200610    1  108  88  19    0  27.1  0.400   24          1
2  ICU200611    6   96   0   0    0  23.7  0.190   28          1
```

? 1.5 Data Information

I want to have an overall look on both of the train and test dataset, so I use .shape and .info() function in python to do that.

Sample train Dataset

```
print ("The shape of the train data is (row, column):"+
str(train.shape))
print (train.info())
```

```
The shape of the train data is (row, column):(599, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 599 entries, 0 to 598
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   ID         599 non-null    object
 1   PRG        599 non-null    int64
 2   PL         599 non-null    int64
 3   PR         599 non-null    int64
 4   SK         599 non-null    int64
 5   TS         599 non-null    int64
 6   M11        599 non-null    float64
 7   BD2        599 non-null    float64
 8   Age        599 non-null    int64
 9   Insurance  599 non-null    int64
 10  Sepssis    599 non-null    object
dtypes: float64(2), int64(7), object(2)
memory usage: 51.6+ KB
None
```

------------> OBSERVATION

From this, the information that I gained from the train dataset are the total paitent record is 599 with no missing and it has 11 columns with the target variable `Sepsis.

Sample test Dataset

```
print ("The shape of the test data is (row, column):"+
str(test.shape))
print (test.info())
```

```
The shape of the test data is (row, column):(169, 10)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ID         169 non-null    object 
 1   PRG        169 non-null    int64  
 2   PL         169 non-null    int64  
 3   PR         169 non-null    int64  
 4   SK         169 non-null    int64  
 5   TS         169 non-null    int64  
 6   M11        169 non-null    float64
 7   BD2        169 non-null    float64
 8   Age        169 non-null    int64  
 9   Insurance  169 non-null    int64  
dtypes: float64(2), int64(7), object(1)
memory usage: 13.3+ KB
None
```

------------> OBSERVATION

From this, the information that I gained from the test dataset are the total paitent record is 169 with no missing and it has 10 columns sine it does not have the target variable `Sepsis`.

2. Data Cleaning

? 2.1 About This Dataset

Categorical:

- **Dichotomous**(Nominal variable with only two categories)
  - **Sepsis** (Positive: if a patient in ICU will develop a sepsis , and Negative: otherwise) Negative Positive
- **Ordinal**(just like nominal datatype but can be ordered or ranked)
  - **Age** (Patients age (years)) : this could be numerical and ordinal since it can be ordered * Numeric:**
- **Discrete**
  - **PRG** (Plasma glucose)
- **Continous**

- **Age** (Patients age (years))
- **PL** (Platelets levels in the blood. Blood Work Result-1 (mu U/ml))
- **PR** (Pulse rate: Blood Pressure (mm Hg))
- **SK** (A sodium blood test. Blood Work Result-2 (mm)
- **TS** (Blood Work Result-3 (mu U/ml))
- **M11** (Body mass index (weight in kg/(height in m)^2)
- **BD2** (Blood Work Result-4 (mu U/ml))

✖ 2.2 Data preprocessing

2.2.1 Drop column `ID` and `Insurance`

```
patient_ID = train['ID']
train = train.drop(columns=['ID', 'Insurance'])
test = test.drop(columns=['ID', 'Insurance'])
```

2.2.2 Rename column `Sepssis`

```
train.rename(columns={"Sepssis": "Sepsis"}, inplace=True)
```

⌛ 2.2.3 Convert `Sepsis` in to binary number

```
train.loc[train['Sepsis'].isin(['Positive']), 'Sepsis'] = '1'
train.loc[train['Sepsis'].isin(['Negative']), 'Sepsis'] = '0'
```

⌛ 2.2.4 Drop duplicate

Although, there may be no duplicated value but I still want to drop duplicate.

Sample train Dataset

```
print ("The shape of the data set before dropping duplicated:"+
str(train.shape))
```

```
train = train.drop_duplicates()

print ("The shape of the data set after dropping duplicated:"+
str(train.shape))
```

```
The shape of the data set before dropping duplicated:(599, 9)
The shape of the data set after dropping duplicated:(599, 9)
```

Sample test Dataset

```
print ("The shape of the data set before dropping duplicated:"+
str(test.shape))

test = test.drop_duplicates()

print ("The shape of the data set after dropping duplicated:"+
str(test.shape))
```

```
The shape of the data set before dropping duplicated:(169, 8)
The shape of the data set after dropping duplicated:(169, 8)
```

2.2.5 Convert Data Type:

This step is also known as binary enconding, but I want to change values first sine I want to see the multicorrelation.

```
train['Sepsis'] = train['Sepsis'].astype('int')
```

✖ 2.3 Drop column

There maybe some irrelavant values, multicorrelation and data that may cause data lackage such as ID column. Since then, I want to drop these columns

2.3.1 Check correllation for dropping

I want to drop multi-correlation

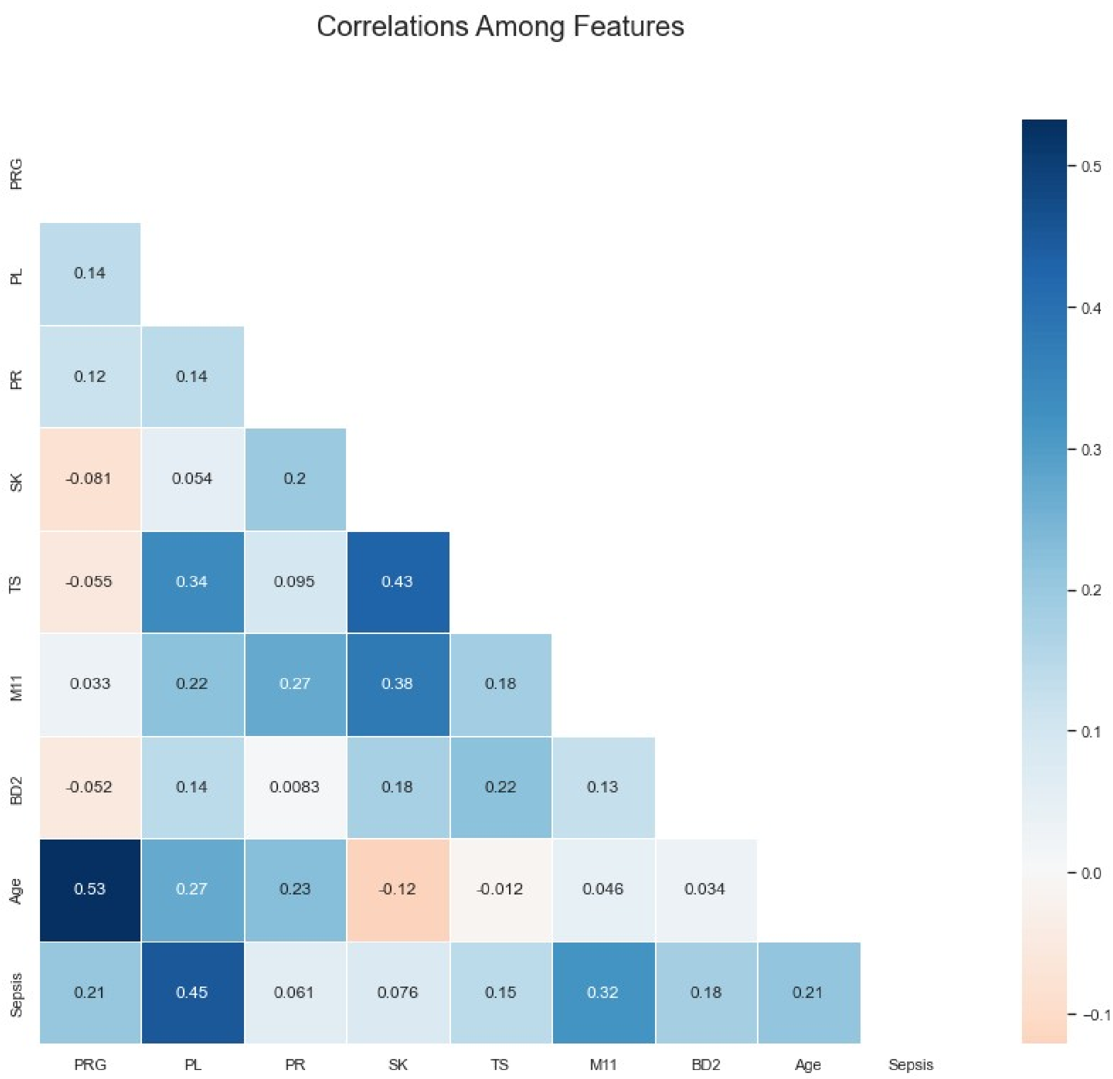
```
## get the most important variables.
corr = train.corr()**2
corr.Sepsis.sort_values(ascending=False)
```

```
Sepsis    1.000000
PL        0.202247
M11       0.099789
Age       0.044198
PRG       0.042897
BD2       0.032964
TS        0.021284
SK        0.005713
PR        0.003732
Name: Sepsis, dtype: float64
```

```
## heatmeap to see the correlation between features.
# Generate a mask for the upper triangle (taken from seaborn example
gallery)
mask = np.zeros_like(train.corr(), dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.set_style('whitegrid')
plt.subplots(figsize = (15,12))
sns.heatmap(train.corr(), annot=True, mask = mask,
            cmap = 'RdBu', ## in order to reverse the bar replace
"RdBu" with "RdBu_r"
            linewidths=.9, linecolor='white', fmt='.2g', center = 0,
square=True)
plt.title("Correlations Among Features", y = 1.03,fontsize = 20, pad =
40)
```

```
/var/folders/l5/0ygc5m0x66xc7d4v2qzjjv0h0000gn/T/
ipykernel_4048/2646670379.py:3: DeprecationWarning: `np.bool` is a
deprecated alias for the builtin `bool`. To silence this warning, use
`bool` by itself. Doing this will not modify any behavior and is safe.
If you specifically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  mask = np.zeros_like(train.corr(), dtype=np.bool)
```

```
Text(0.5, 1.03, 'Correlations Among Features')
```



------------> OBSERVATION

- Our target variable is `Sepsis`. So if there are any columns have high correlation (>= 0.5) I desire to drop them.
- There are no extremely high multicorrelation so I keep all of them.

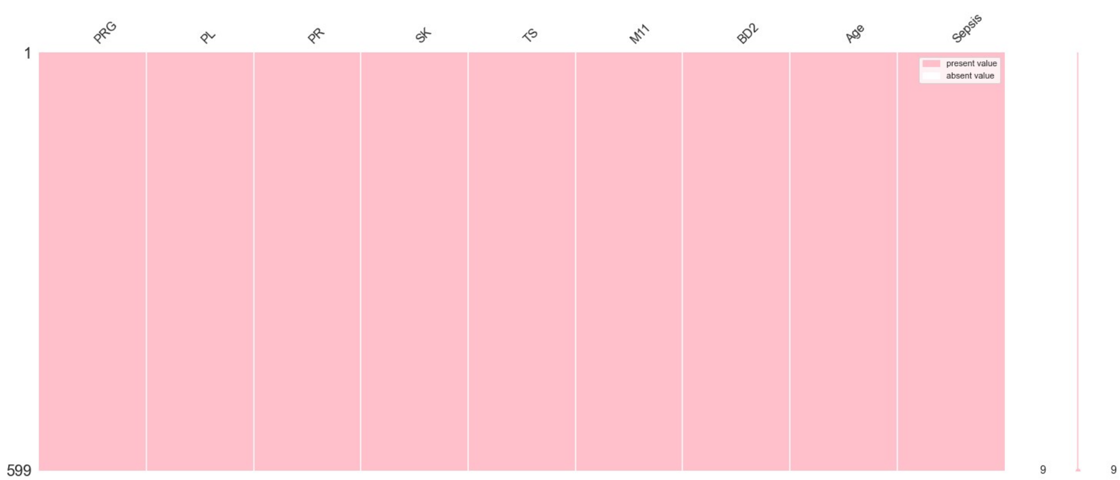
? 2.3.2 Check missing values for dropping

Sample train Dataset

```
def missing_percentage(df):
    """This function takes a DataFrame(df) as input and returns two
columns, total missing values and total missing values percentage"""
    total = df.isnull().sum().sort_values(ascending=False)
```

```
[df.isnull().sum().sort_values(ascending=False) != 0]
    percent = round(df.isnull().sum().sort_values(ascending=False) /
len(df) * 100, 2)[
        round(df.isnull().sum().sort_values(ascending=False) / len(df)
* 100, 2) != 0]
    return pd.concat([total, percent], axis=1, keys=['Total',
'Percent'])

# display missing values in descending
print("Missing values in the dataframe in descending: \n",
missing_percentage(train).sort_values(by='Total', ascending=False))

# visualize where the missing values are located
msno.matrix(train, color=(255 / 255, 192 / 255, 203 / 255))
pink_patch = mpatches.Patch(color='pink', label='present value')
white_patch = mpatches.Patch(color='white', label='absent value')
plt.legend(handles=[pink_patch, white_patch])
plt.show()
```

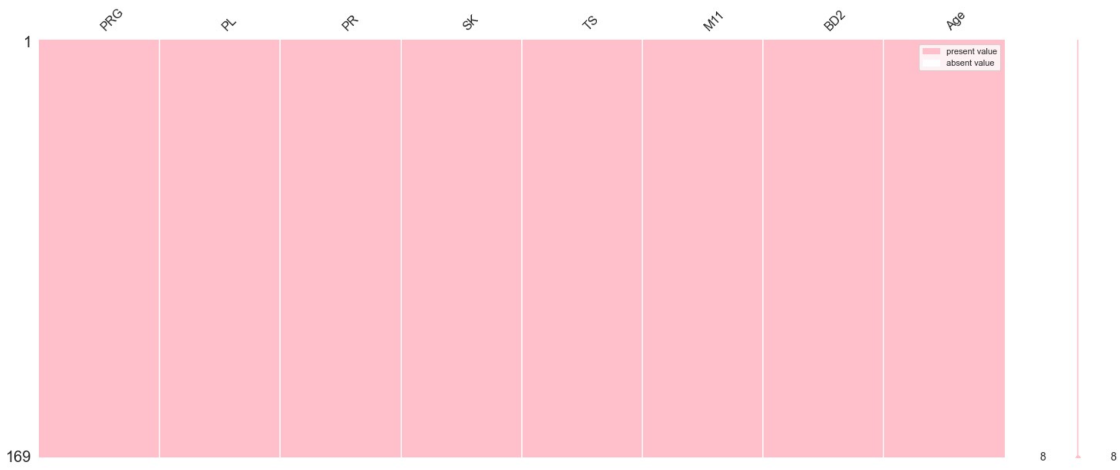
```
Missing values in the dataframe in descending:
 Empty DataFrame
Columns: [Total, Percent]
Index: []
```



Sample test Dataset

```
# display missing values in descending
print("Missing values in the dataframe in descending: \n",
missing_percentage(test).sort_values(by='Total', ascending=False))

# visualize where the missing values are located
msno.matrix(test, color=(255 / 255, 192 / 255, 203 / 255))
pink_patch = mpatches.Patch(color='pink', label='present value')
white_patch = mpatches.Patch(color='white', label='absent value')
plt.legend(handles=[pink_patch, white_patch])
plt.show()
```

```
Missing values in the dataframe in descending:
 Empty DataFrame
Columns: [Total, Percent]
Index: []
```



------> OBSERVATION

Suprisingly, there is no missing data in both of the dataset.

? 2.4 Upper Case the content

Sample train Dataset

```
# Cast all values inside the dataframe (except the columns' name) into
upper case.
train = train.applymap(lambda s: s.upper() if type(s) == str else s)
train.head(3)
```

	PRG	PL	PR	SK	TS	M11	BD2	Age	Sepsis
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

Sample test Dataset

```
# Cast all values inside the dataframe (except the columns' name) into
upper case.
test = test.applymap(lambda s: s.upper() if type(s) == str else s)
test.head(3)
```

	PRG	PL	PR	SK	TS	M11	BD2	Age
0	1	109	38	18	120	23.1	0.407	26

```
1     1  108  88  19    0  27.1  0.400   24
2     6   96   0   0    0  23.7  0.190   28
```

? 2.5 Extra-whitespaces:

```
def whitespace_remover(df):
    """
    The function will remove extra leading and trailing whitespace
from the data.
    """
    # iterating over the columns
    for i in df.columns:
        # checking datatype of each columns
        if df[i].dtype == 'object' or df[i].dtype == 'str':
            # applying strip function on column
            df[i] = df[i].map(str.strip)
        else:
            # if condition is False then it will do nothing.
            pass

# remove all the extra whitespace
whitespace_remover(train)
whitespace_remover(test)
```

? 2.6 Descriptive statistics for Central Tendency

I want to check and validate of the numerical columns:

1. Check overview statistics:
2. Check the scale
3. Check outliers

? 2.6.1 Overview statistics

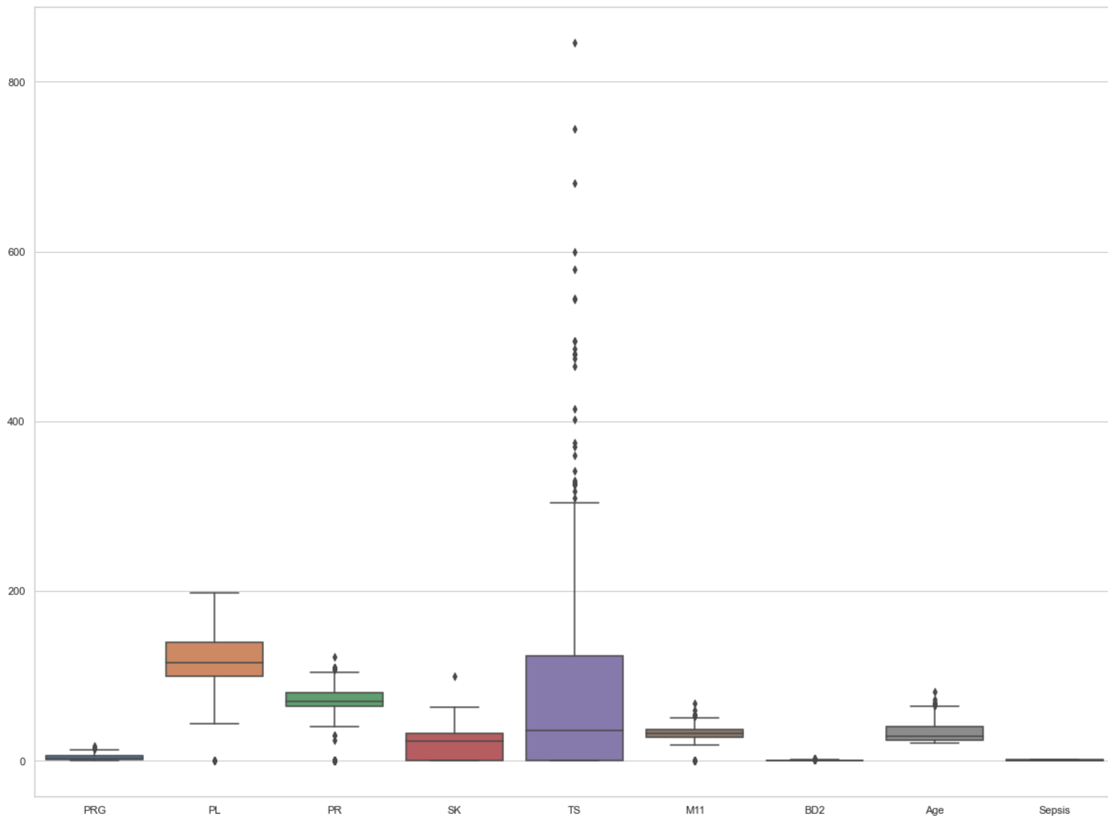
Sample train Dataset

```
# see the static of all numerical column
train.describe().T
```

```
        count        mean         std     min     25%      50%
75%  \
PRG     599.0    3.824708    3.362839   0.000   1.000    3.000
6.000
PL      599.0  120.153589   32.682364   0.000  99.000  116.000
140.000
PR      599.0   68.732888   19.335675   0.000  64.000   70.000
80.000
SK      599.0   20.562604   16.017622   0.000   0.000   23.000
32.000
TS      599.0   79.460768  116.576176   0.000   0.000   36.000
123.500
M11     599.0   31.920033    8.008227   0.000  27.100   32.000
36.550
BD2     599.0    0.481187    0.337552   0.078   0.248    0.383
0.647
Age     599.0   33.290484   11.828446  21.000  24.000   29.000
40.000
Sepsis  599.0    0.347245    0.476492   0.000   0.000    0.000
1.000

           max
PRG      17.00
PL      198.00
PR      122.00
SK       99.00
TS      846.00
M11      67.10
BD2       2.42
Age      81.00
Sepsis    1.00
```

```python
plt.rcParams['figure.figsize'] = [20, 15]
# plot the boxplot to see the outlier of each numerical column
sns.boxplot(data=train,orient="v")
```

```
<AxesSubplot:>
```



---------> OBSERVATION

The scale of this data set is considerably large so that I desire to have some of the domain knowlegde in order to detect the outliers.

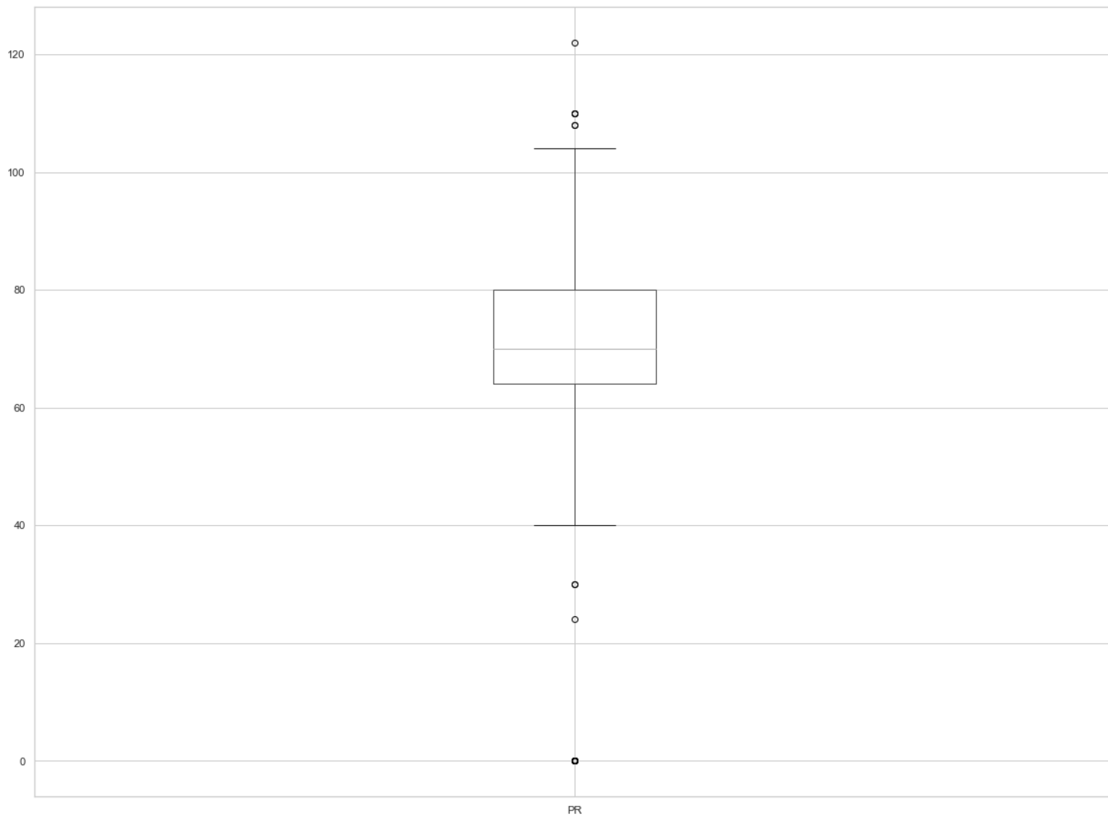
Box plot for numerical columns

```
train.boxplot('PL')
```

```
<AxesSubplot:>
```

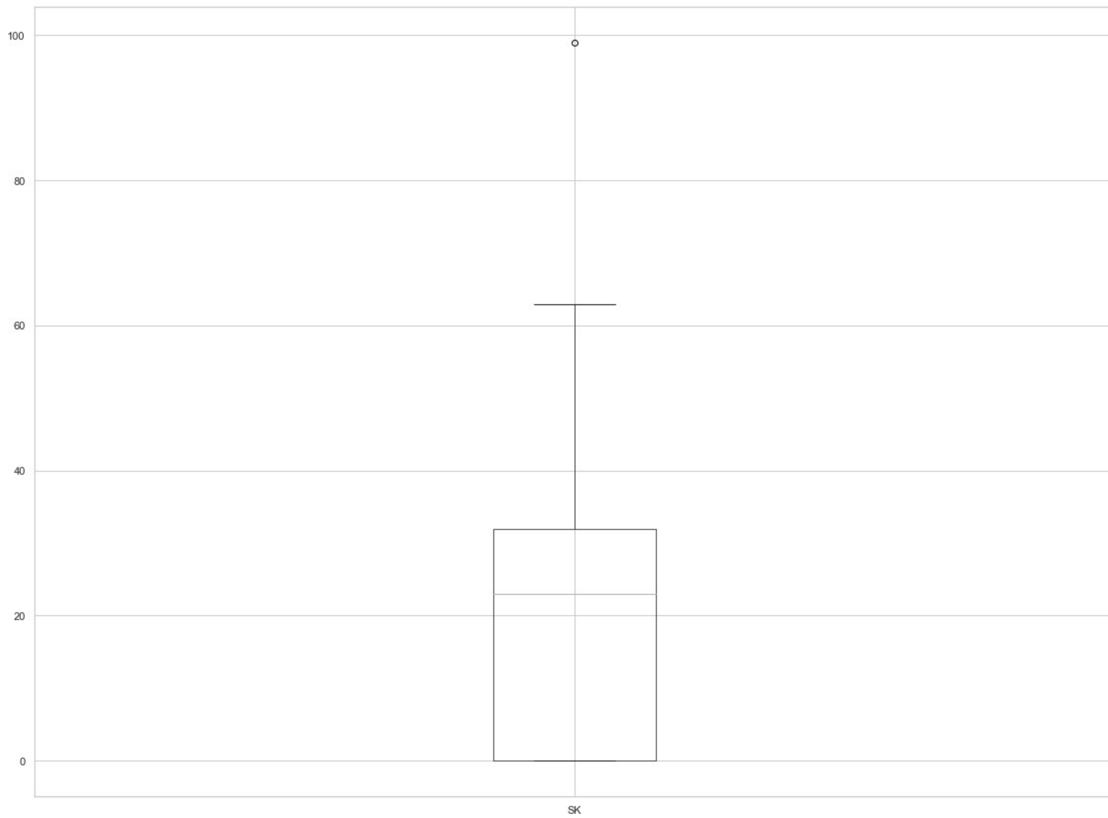

```
train.boxplot('PR')
```

```
<AxesSubplot:>
```



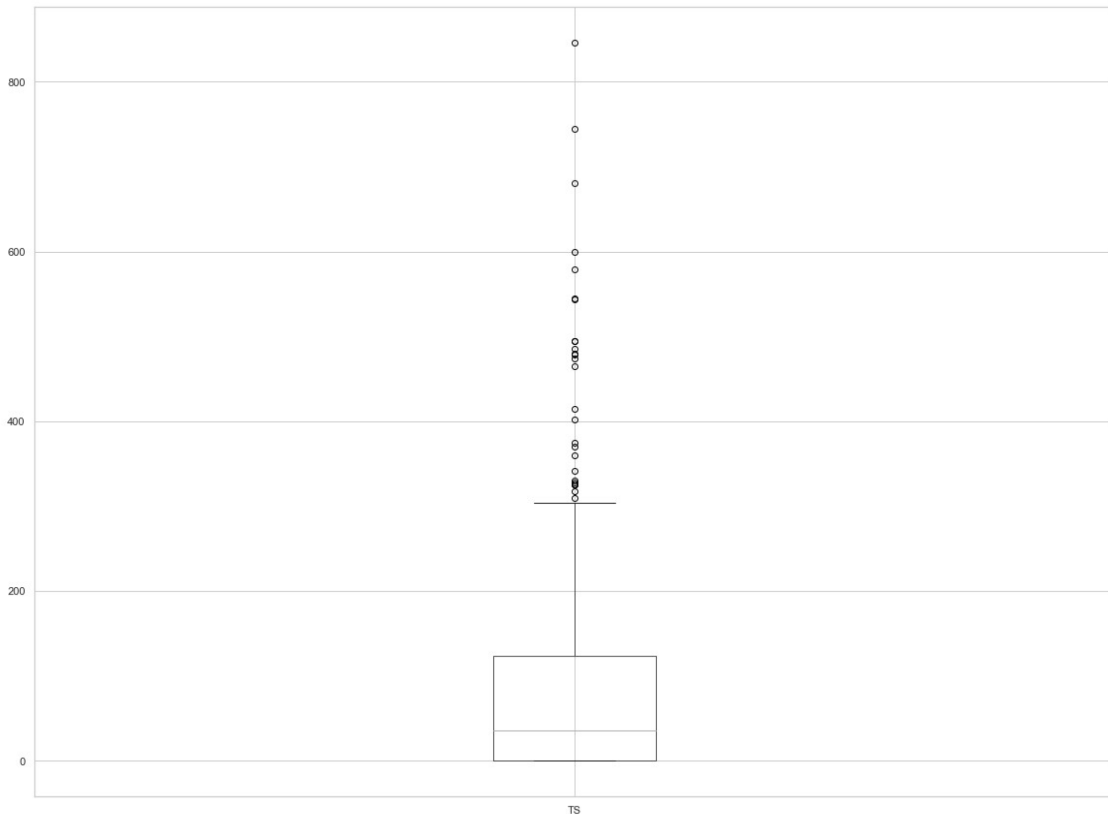

```
train.boxplot('SK')
```

```
<AxesSubplot:>
```



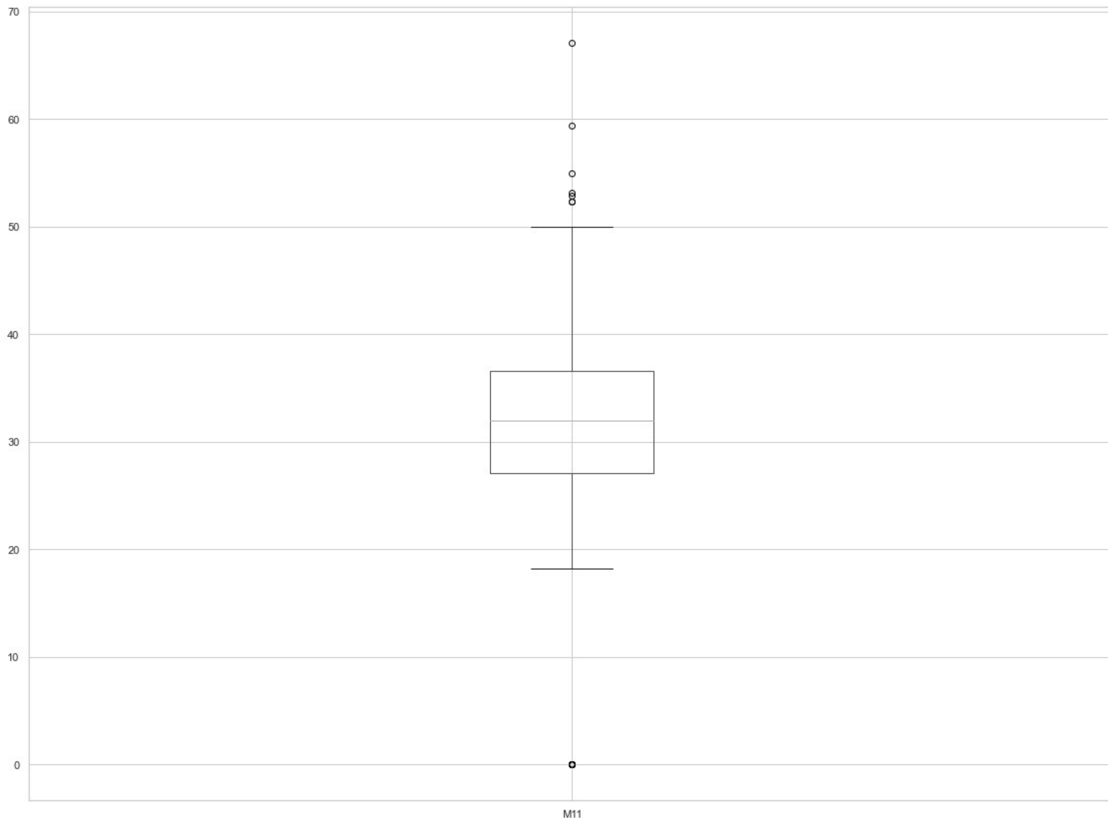

```
train.boxplot('TS')
```

```
<AxesSubplot:>
```



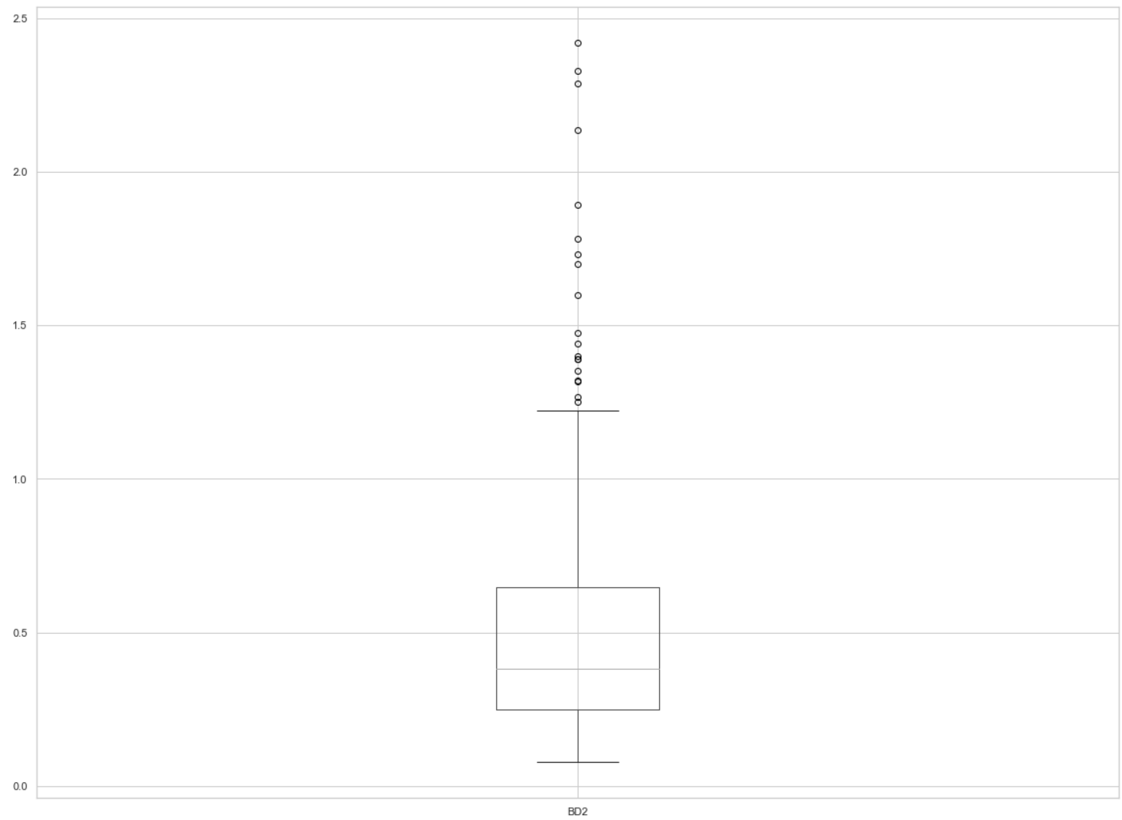

```
train.boxplot('M11')
```

```
<AxesSubplot:>
```




```
train.boxplot('BD2')
```

```
&lt;AxesSubplot:&gt;
```



---------> OBSERVATION

There are some outliers, but I still want to consider whether it is appropriate to drop them.

? 2.6.2 Domain Knowledge:

I want to try to explore the relationship between those columns and the normal range for each of them **** **PL** (platelet rate)

> A normal platelet count ranges from 150,000 to 450,000 platelets per microliter of blood. Having more than 450,000 platelets is a condition called thrombocytosis; having less than 150,000 is known as thrombocytopenia. [1]

PR (pulse rate)

> A normal resting heart rate for adults ranges from 60 to 100 beats per minute. [2]

SK

> Normally, serum potassium (SK) level is tightly maintained between 3.5 mmol/L and 5.5 mmol/L. [3]

TS

- The normal range of TSH levels is 0.4 to 4.0 milli-international units per liter. If you're already being treated for a thyroid disorder, the normal range is 0.5 to 3.0 milli-international units per liter.
- A value above the normal range usually indicates that the thyroid is underactive. This indicates hypothyroidism. When the thyroid isn't producing enough hormones, the pituitary gland releases more TSH to try to stimulate it.

M11

- Below 18.5 – you're in the underweight range [4]
- Between 18.5 and 24.9 – you're in the healthy weight range [4]
- Between 25 and 29.9 – you're in the overweight range [4]
- Between 30 and 39.9 – you're in the obese range [4]

PRG

BD2 (cannot search???)

However, I realised that whatever those values will be, the values which are higher or even lower, the higher chance that a patient will get a sepsis

? 2.6.3 Dectect outliers:

PL column:

Sample train Dataset

```
train_PL_q_low = train["PL"].quantile(0.01)
train_PL_q_hi  = train["PL"].quantile(0.99)

df_filtered = train[(train["PL"] > train_PL_q_hi) | (train["PL"] <
train_PL_q_low) | (train["PL"] == 0)]
print("Percentage compare to total: " + str(len(df_filtered)/
len(train) * 100))
df_filtered
```

```
Percentage compare to total: 1.8363939899833055
```

	PRG	PL	PR	SK	TS	M11	BD2	Age	Sepsis
8	2	197	70	45	543	30.5	0.158	53	1
62	5	44	62	0	0	25.0	0.587	36	0
75	1	0	48	20	0	24.7	0.140	22	0
182	1	0	74	20	23	27.7	0.299	21	0

```
228    4  197  70  39  744  36.7  2.329   31        0
342    1    0  68  35    0  32.0  0.389   22        0
349    5    0  80  32    0  41.0  0.346   37        1
408    8  197  74   0    0  25.9  1.191   39        1
502    6    0  68  41    0  39.0  0.727   41        1
561    0  198  66  32  274  41.3  0.502   28        1
579    2  197  70  99    0  34.7  0.575   62        1
```

Sample test Dataset

```
test_PL_q_low = test["PL"].quantile(0.01)
test_PL_q_hi  = test["PL"].quantile(0.99)

df_filtered = test[(test["PL"] > test_PL_q_hi) | (test["PL"] <
test_PL_q_low) | (test["PL"] == 0)]
print("Percentage compare to total: " + str(len(df_filtered)/
len(test) * 100))
df_filtered
```

```
Percentage compare to total: 2.366863905325444

     PRG   PL  PR  SK  TS   M11    BD2  Age
62     1  199  76  43   0  42.9  1.394   22
76     6  195  70   0   0  30.9  0.328   31
81     2   56  56  28  45  24.2  0.332   22
138    8   65  72  23   0  32.0  0.600   42
```

---------> OBSERVATION

- The outliers is insignificant, but, more people are in these special cases have sepsis. So I desire to keep them.

PR column:

Sample train Dataset

```
train_PR_q_low = train["PR"].quantile(0.01)
train_PR_q_hi  = train["PR"].quantile(0.99)

df_filtered = train[(train["PR"] > train_PR_q_hi) | (train["PR"] <
train_PR_q_low) | (train["PR"] == 0)]
print(len(df_filtered)/ len(train) * 100)
df_filtered
```

```
5.676126878130217

     PRG   PL   PR  SK   TS   M11    BD2  Age  Sepsis
7     10  115    0   0    0  35.3  0.134   29       0
15     7  100    0   0    0  30.0  0.484   32       1
43     9  171  110  24  240  45.4  0.721   54       1
49     7  105    0   0    0   0.0  0.305   24       0
60     2   84    0   0    0   0.0  0.304   21       0
```

```
78      0  131    0   0    0  43.2  0.270   26        1
81      2   74    0   0    0   0.0  0.102   22        0
84      5  137  108   0    0  48.8  0.227   37        1
106     1   96  122   0    0  22.4  0.207   27        0
172     2   87    0  23    0  28.9  0.773   25        0
177     0  129  110  46  130  67.1  0.319   26        1
193    11  135    0   0    0  52.3  0.578   40        1
222     7  119    0   0    0  25.2  0.209   37        0
261     3  141    0   0    0  30.0  0.761   27        1
266     0  138    0   0    0  36.3  0.933   25        1
269     2  146    0   0    0  27.5  0.240   28        1
300     0  167    0   0    0  32.3  0.839   30        1
332     1  180    0   0    0  43.3  0.282   41        1
336     0  117    0   0    0  33.8  0.932   44        0
347     3  116    0   0    0  23.5  0.187   23        0
357    13  129    0  30    0  39.9  0.569   44        1
362     5  103  108  37    0  39.2  0.305   65        0
426     0   94    0   0    0   0.0  0.256   25        0
430     2   99    0   0    0  22.2  0.108   23        0
435     0  141    0   0    0  42.4  0.205   29        1
453     2  119    0   0    0  19.6  0.832   72        0
468     8  120    0   0    0  30.0  0.183   38        1
484     0  145    0   0    0  44.2  0.630   31        1
494     3   80    0   0    0   0.0  0.174   22        0
522     6  114    0   0    0   0.0  0.189   26        0
533     6   91    0   0    0  29.8  0.501   31        0
535     4  132    0   0    0  32.9  0.302   23        1
549     4  189  110  31    0  28.5  0.680   37        0
589     0   73    0   0    0  21.1  0.342   25        0
```

Sample test Dataset

```
test_PR_q_low = test["PR"].quantile(0.01)
test_PR_q_hi  = test["PR"].quantile(0.99)

df_filtered = test[(test["PR"] > test_PR_q_hi) | (test["PR"] <
test_PR_q_low) | (test["PR"] == 0)]
print(len(df_filtered)/ len(test) * 100)
df_filtered
```

```
4.733727810650888
```

```
     PRG   PL   PR  SK  TS   M11    BD2  Age
2      6   96    0   0   0  23.7  0.190   28
5      4  183    0   0   0  28.4  0.212   36
20     0  119    0   0   0  32.4  0.141   24
44     4   90    0   0   0  28.0  0.610   31
92    13  158  114   0   0  42.3  0.257   44
98     0   99    0   0   0  25.0  0.253   22
104    2  129    0   0   0  38.5  0.304   41
107   10  115    0   0   0   0.0  0.261   30
```

---------> OBSERVATION

- The outliers is insignificant, and it is impossible that the pulse rate can be 0 and the patients are still alive.

```
test.loc[(test["PR"] == 0), 'PR'] = test["PR"].mean()
train.loc[(train["PR"] == 0), 'PR'] = train["PR"].mean()
```

SK column:

Sample train Dataset

```
train_SK_q_low = train["SK"].quantile(0.01)
train_SK_q_hi  = train["SK"].quantile(0.99)

df_filtered = train[(train["SK"] > train_SK_q_hi) | (train["SK"] <
train_SK_q_low) | (train["SK"] == 0)]
print(len(df_filtered)/ len(train) * 100)
df_filtered
```

```
30.217028380634392

     PRG   PL         PR  SK  TS   M11    BD2  Age  Sepsis
2      8  183  64.000000   0   0  23.3  0.672   32       1
5      5  116  74.000000   0   0  25.6  0.201   30       0
7     10  115  68.732888   0   0  35.3  0.134   29       0
9      8  125  96.000000   0   0   0.0  0.232   54       1
10     4  110  92.000000   0   0  37.6  0.191   30       0
..   ...  ...        ...  ..  ..   ...    ...  ...     ...
587    6  103  66.000000   0   0  24.3  0.249   29       0
589    0   73  68.732888   0   0  21.1  0.342   25       0
592    3  132  80.000000   0   0  34.4  0.402   44       1
596    0   67  76.000000   0   0  45.3  0.194   46       0
598    1  173  74.000000   0   0  36.8  0.088   38       1

[181 rows x 9 columns]
```

Sample test Dataset

```
test_SK_q_low = test["SK"].quantile(0.01)
test_SK_q_hi  = test["SK"].quantile(0.99)

df_filtered = test[(test["SK"] > test_SK_q_hi) | (test["SK"] <
test_SK_q_low) | (test["SK"] == 0)]
print(len(df_filtered)/ len(test) * 100)
df_filtered
```

```
31.360946745562128

     PRG   PL          PR  SK   TS   M11    BD2  Age
2      6   96   70.426036   0    0  23.7  0.190   28
5      4  183   70.426036   0    0  28.4  0.212   36
```

16	3	106	72.000000	0	0	25.8	0.207	27
17	6	117	96.000000	0	0	28.7	0.157	30
20	0	119	70.426036	0	0	32.4	0.141	24
23	6	183	94.000000	0	0	40.8	1.461	45
25	2	108	64.000000	0	0	30.8	0.158	21
27	0	125	68.000000	0	0	24.7	0.206	21
28	0	132	78.000000	0	0	32.4	0.393	21
29	5	128	80.000000	0	0	34.6	0.144	45
31	7	114	64.000000	0	0	27.4	0.732	34
33	2	111	60.000000	0	0	26.2	0.343	23
35	10	92	62.000000	0	0	25.9	0.167	31
36	13	104	72.000000	0	0	31.2	0.465	38
37	5	104	74.000000	0	0	28.8	0.153	48
42	4	128	70.000000	0	0	34.3	0.303	24
43	6	147	80.000000	0	0	29.5	0.178	50
44	4	90	70.426036	0	0	28.0	0.610	31
54	2	120	54.000000	0	0	26.8	0.455	27
59	11	127	106.000000	0	0	39.0	0.190	51
61	10	162	84.000000	0	0	27.7	0.182	54
75	8	91	82.000000	0	0	35.6	0.587	68
76	6	195	70.000000	0	0	30.9	0.328	31
77	9	156	86.000000	0	0	24.8	0.230	53
78	0	93	60.000000	0	0	35.3	0.263	25
79	3	121	52.000000	0	0	36.0	0.127	25
84	4	125	80.000000	0	0	32.3	0.536	27
85	5	136	82.000000	0	0	0.0	0.640	69
87	3	130	64.000000	0	0	23.1	0.314	22
91	8	107	80.000000	0	0	24.6	0.856	34
92	13	158	114.000000	0	0	42.3	0.257	44
94	7	129	68.000000	49	125	38.5	0.439	43
95	2	90	60.000000	0	0	23.5	0.191	25
98	0	99	70.426036	0	0	25.0	0.253	22
100	4	118	70.000000	0	0	44.5	0.904	26
104	2	129	70.426036	0	0	38.5	0.304	41
107	10	115	70.426036	0	0	0.0	0.261	30
109	9	164	78.000000	0	0	32.8	0.148	45
115	3	102	74.000000	0	0	29.5	0.121	32
125	1	111	94.000000	0	0	32.8	0.265	45
129	2	175	88.000000	0	0	22.9	0.326	22
130	2	92	52.000000	0	0	30.1	0.141	22
132	8	120	86.000000	0	0	28.4	0.259	22
135	2	105	75.000000	0	0	23.3	0.560	53
140	1	102	74.000000	0	0	39.5	0.293	42
144	9	140	94.000000	0	0	32.7	0.734	45
150	6	162	62.000000	0	0	24.3	0.178	50
151	4	136	70.000000	0	0	31.2	1.182	22
158	0	123	72.000000	0	0	36.3	0.258	52
159	1	106	76.000000	0	0	37.5	0.197	26
160	6	190	92.000000	0	0	35.5	0.278	66

```
163     9   89   62.000000   0     0  22.5  0.142   33
167     1  126   60.000000   0     0  30.1  0.349   47
```

---------> OBSERVATION

- The outliers is considerable I need to explore it before clear it.

TS column:

Sample train Dataset

```
train_TS_q_low = train["TS"].quantile(0.01)
train_TS_q_hi  = train["TS"].quantile(0.99)

df_filtered = train[(train["TS"] > train_TS_q_hi) | (train["TS"] <
train_TS_q_low) | (train["TS"] == 0)]
print(len(df_filtered)/ len(train) * 100)
df_filtered
```

```
49.248747913188645

     PRG   PL         PR  SK  TS   M11    BD2  Age  Sepsis
0      6  148  72.000000  35   0  33.6  0.627   50       1
1      1   85  66.000000  29   0  26.6  0.351   31       0
2      8  183  64.000000   0   0  23.3  0.672   32       1
5      5  116  74.000000   0   0  25.6  0.201   30       0
7     10  115  68.732888   0   0  35.3  0.134   29       0
..   ...  ...        ...  ..  ..   ...    ...  ...     ...
589    0   73  68.732888   0   0  21.1  0.342   25       0
590   11  111  84.000000  40   0  46.8  0.925   45       1
592    3  132  80.000000   0   0  34.4  0.402   44       1
596    0   67  76.000000   0   0  45.3  0.194   46       0
598    1  173  74.000000   0   0  36.8  0.088   38       1

[295 rows x 9 columns]
```

Sample test Dataset

```
test_TS_q_low = test["TS"].quantile(0.01)
test_TS_q_hi  = test["TS"].quantile(0.99)

df_filtered = test[(test["TS"] > test_TS_q_hi) | (test["TS"] <
test_TS_q_low) | (test["TS"] == 0)]
print(len(df_filtered)/ len(test) * 100)
df_filtered
```

```
51.4792899408284

     PRG   PL         PR  SK  TS   M11    BD2  Age
1      1  108  88.000000  19   0  27.1  0.400   24
2      6   96  70.426036   0   0  23.7  0.190   28
3      1  124  74.000000  36   0  27.8  0.100   30
```

```
5      4  183  70.426036   0   0  28.4  0.212   36
6      1  124  60.000000  32   0  35.8  0.514   21
..   ...  ...        ...  ..  ..   ...    ...  ...
162    9  170  74.000000  31   0  44.0  0.403   43
163    9   89  62.000000   0   0  22.5  0.142   33
165    2  122  70.000000  27   0  36.8  0.340   27
167    1  126  60.000000   0   0  30.1  0.349   47
168    1   93  70.000000  31   0  30.4  0.315   23

[87 rows x 8 columns]
```

---------> OBSERVATION

- The outliers is considerable I need to explore it before clear it.

M11 column:

Sample train Dataset

```
df_filtered = train[(train["M11"] == 0)]
print(len(df_filtered)/ len(train) * 100)
df_filtered
```

```
1.5025041736227045
```

	PRG	PL	PR	SK	TS	M11	BD2	Age	Sepsis
9	8	125	96.000000	0	0	0.0	0.232	54	1
49	7	105	68.732888	0	0	0.0	0.305	24	0
60	2	84	68.732888	0	0	0.0	0.304	21	0
81	2	74	68.732888	0	0	0.0	0.102	22	0
145	0	102	75.000000	23	0	0.0	0.572	21	0
371	0	118	64.000000	23	89	0.0	1.731	21	0
426	0	94	68.732888	0	0	0.0	0.256	25	0
494	3	80	68.732888	0	0	0.0	0.174	22	0
522	6	114	68.732888	0	0	0.0	0.189	26	0

---------> OBSERVATION

The number of impossible values accounts for just over 1.5 percent so I want to replace them with the group by age meadian since they are all

```
train.drop(train[(train["M11"] == 0)].index, inplace = True)
```

Sample test Dataset

```
df_filtered = test[(test["M11"] == 0)]
print(len(df_filtered)/ len(test) * 100)
df_filtered
```

```
1.183431952662722
```

```
     PRG   PL          PR  SK  TS  M11    BD2  Age
85     5  136  82.000000   0   0  0.0  0.640   69
107   10  115  70.426036   0   0  0.0  0.261   30
```

---------> OBSERVATION

The number of impossible values accounts for just over 1.5 percent so I want to replace them with the group by age meadian since they are all

```
test.drop(test[(test["M11"] == 0)].index, inplace = True)
```

BD2 column:

Sample train Dataset

```
train_BD2_q_low = train["BD2"].quantile(0.01)
train_BD2_q_hi  = train["BD2"].quantile(0.99)

df_filtered = train[(train["BD2"] > train_BD2_q_hi) | (train["BD2"] <
train_BD2_q_low) | (train["BD2"] == 0)]
print(len(df_filtered)/ len(train) * 100)
df_filtered
```

```
2.0338983050847457
```

```
     PRG   PL    PR  SK   TS   M11    BD2  Age  Sepsis
4      0  137  40.0  35  168  43.1  2.288   33       1
45     0  180  66.0  39    0  42.0  1.893   25       1
58     0  146  82.0   0    0  40.5  1.781   44       0
135    2  125  60.0  20  140  33.8  0.088   31       0
149    2   90  70.0  17    0  27.3  0.085   22       0
180    6   87  80.0   0    0  23.2  0.084   32       0
228    4  197  70.0  39  744  36.7  2.329   31       0
268    0  102  52.0   0    0  25.1  0.078   21       0
370    3  173  82.0  48  465  38.4  2.137   25       1
445    0  180  78.0  63   14  59.4  2.420   25       1
567    6   92  62.0  32  126  32.0  0.085   46       0
598    1  173  74.0   0    0  36.8  0.088   38       1
```

Sample test Dataset

```
test_BD2_q_low = test["BD2"].quantile(0.01)
test_BD2_q_hi  = test["BD2"].quantile(0.99)

df_filtered = test[(test["BD2"] > test_BD2_q_hi) | (test["BD2"] <
test_BD2_q_low) | (test["BD2"] == 0)]
print(len(df_filtered)/ len(test) * 100)
df_filtered
```

```
2.3952095808383236
```

```
    PRG   PL    PR  SK   TS   M11    BD2  Age
3     1  124  74.0  36    0  27.8  0.100   30
22    2   92  76.0  20    0  24.2  1.698   28
23    6  183  94.0   0    0  40.8  1.461   45
34    1  128  82.0  17  183  27.5  0.115   22
```

---------> OBSERVATION

- The outliers is considerable I need to explore it before clear it.

Check outliers

I want to query all the numberical columns to check if those medical values which is considered to be the outliers have the probability to have sepsis?

```
likely_sepsis = train[(train["PL"] > train_PL_q_hi) | (train["PL"] < train_PL_q_low) |
                      (train["PR"] > train_PR_q_hi) | (train["PR"] < train_PR_q_low) |
                      (train["SK"] > train_SK_q_hi) | (train["SK"] < train_SK_q_low) |
                      (train["TS"] > train_TS_q_hi) | (train["TS"] < train_TS_q_low) |
                      (train["BD2"] > train_BD2_q_hi) | (train["BD2"] < train_BD2_q_low)]

rslt_df = likely_sepsis[(likely_sepsis['Sepsis'] == 1)]

print(len(rslt_df)/ len(likely_sepsis) * 100)

48.64864864864865
```

---------> OBSERVATION

- As we can see, it is more than a half of the people having the abnormal statistic tend to have sepsis. Since then, I do not want to drop these outliers. Moreover, sepsis is a kind of illness, and having those abnormal statistic means likely to cause illness.

```
#I Want to save for later tree plotting
target_name = ['Sepsis']
feature_name = ['PRG', 'PL', 'PR', 'SK', 'TS', 'M11' 'BD2']
```

? 2.7 Save the Intermediate data

After the cleaning step, all data is saved to a csv file for visualisation step later in dash.

```
train.to_csv("Data/train_cleaned.csv", encoding='utf-8')
test.to_csv("Data/test_cleaned.csv", encoding='utf-8')
```

3. Data exploration (EDA)

Function for box plot visualization

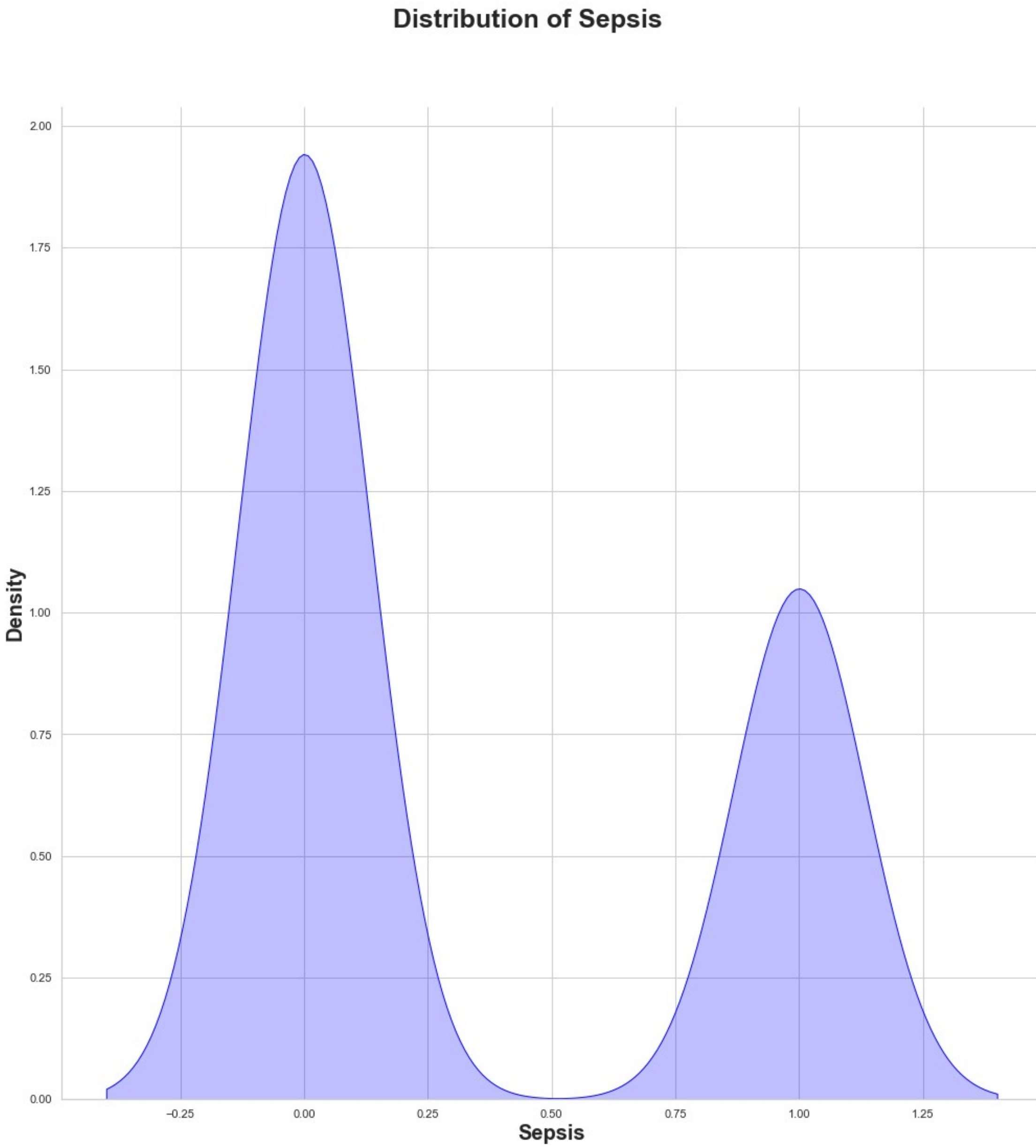
```
PROPS = {
    'boxprops':{'edgecolor':'black'},
    'medianprops':{'color':'black'},
    'whiskerprops':{'color':'black'},
    'capprops':{'color':'black'}
}
def plot_box(dataset, x, y, xlabel, ylabel, title, subtitle, color, title_position, subtitle_position, order=None):
    ax = sns.boxplot(data = dataset, y = y, x = x, order = order,
                     linewidth = 1.2, color = color, **PROPS,
                     flierprops = dict(marker = 'o', markeredgecolor = 'black', markersize = 6.5, linestyle = 'none', markerfacecolor = color, alpha = 0.9))

    plt.xlabel(xlabel, fontweight = 'bold', fontsize = 16)
    plt.ylabel(ylabel, fontweight = 'bold', fontsize = 16)
    ax.tick_params(labelsize = 14)
    ax.text(x = title_position, y = 1.07, s = title, fontsize = 22.5, weight = 'bold', ha = 'center', va = 'bottom', transform = ax.transAxes)
    ax.text(x = subtitle_position, y = 1.03, s = subtitle, fontsize = 16.5, alpha = 0.75, ha = 'center', va = 'bottom', transform = ax.transAxes)
    plt.show()
```

3.1 Overall look on target variable

3.1.1 Distribution of Sepsis

```
# sns.displot(train, x="Survived", hue="Pclass", kind="kde", fill=True)
plot = sns.displot(train, x="Sepsis", kind="kde", fill=True, color='blue', height= 14)

plot.fig.suptitle("Distribution of Sepsis", fontsize=25, y=1.08, fontweight = 'bold')
plot.set_xlabels("Sepsis", fontsize = 20, fontweight = 'bold' )
plot.set_ylabels("Density", fontsize = 20, fontweight = 'bold')
```

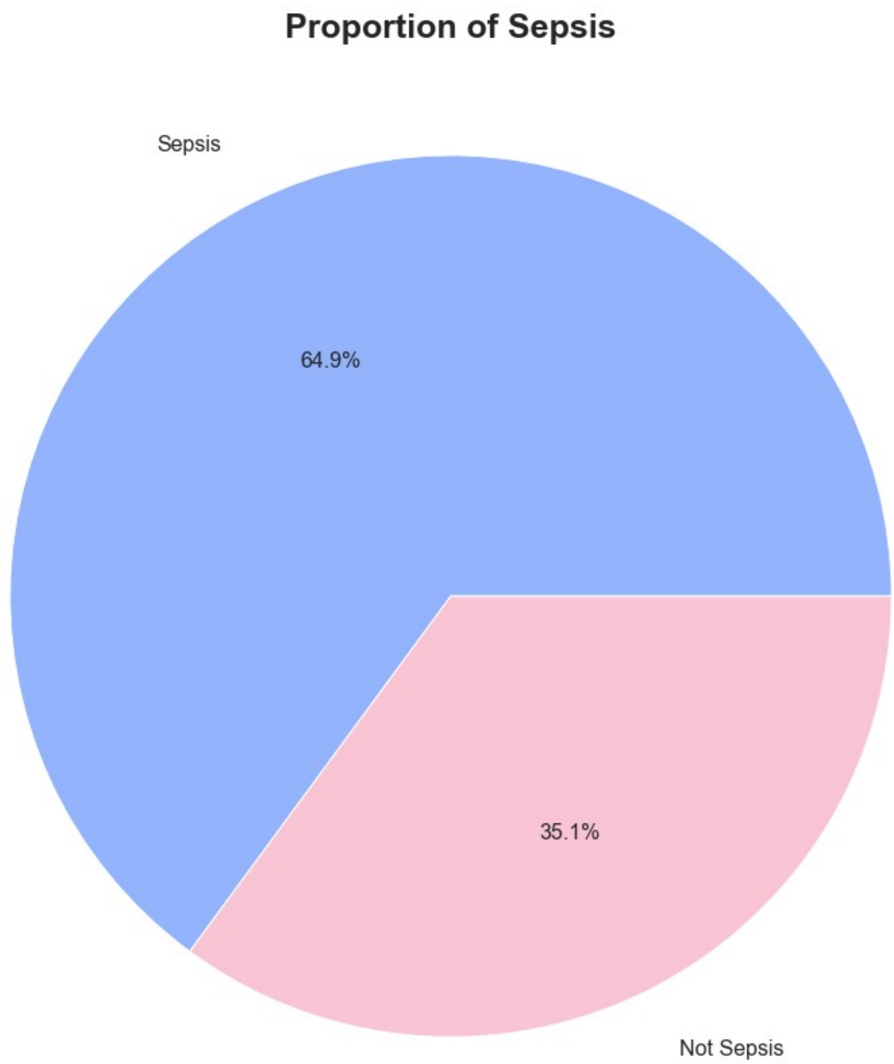
```
<seaborn.axisgrid.FacetGrid at 0x7fd2052d1610>
```



3.1.2 Proportion of Sepsis

```
# Pie chart
labels = ['Sepsis', 'Not Sepsis']
#colors
colors = ['#94B3FD', '#F9C5D5']
ax = plt.pie(train['Sepsis'].value_counts(), labeldistance=1.15,
labels=labels, colors=colors, autopct='%1.1f%%',
textprops={'fontsize': 16});
plt.title('Proportion of Sepsis', fontsize=25, fontweight = 'bold')
```

```
plt.rcParams['figure.figsize'] = [20, 15]
plt.show()
```

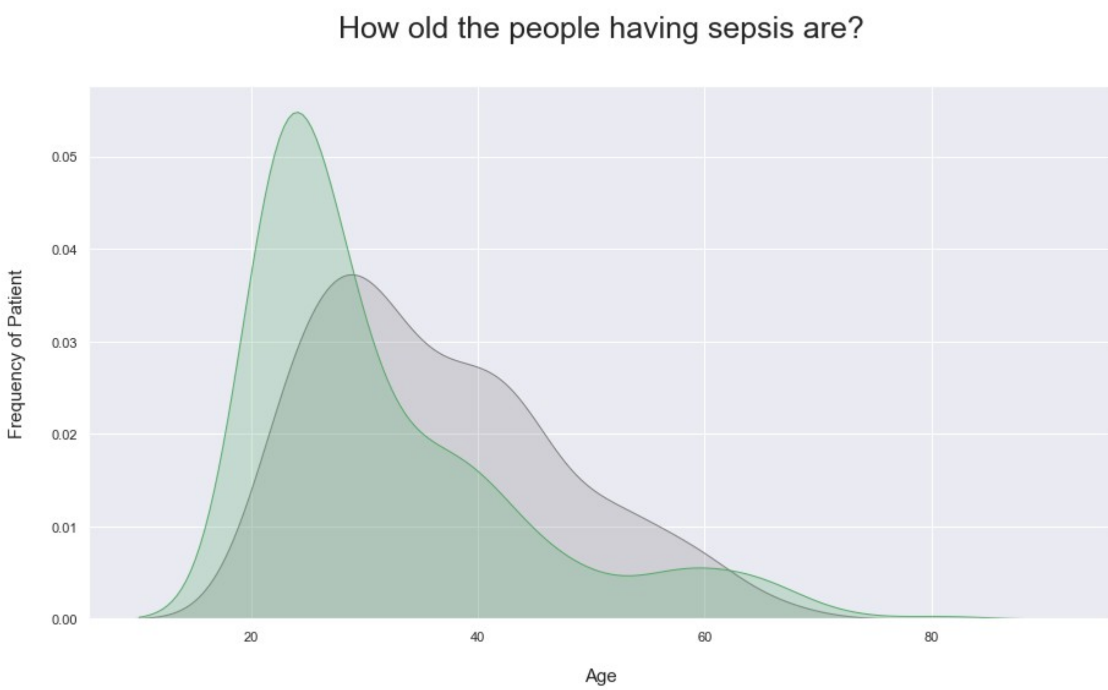


3.2 Frequency of each correspondiding Target variable type

3.2.1 How old are they?

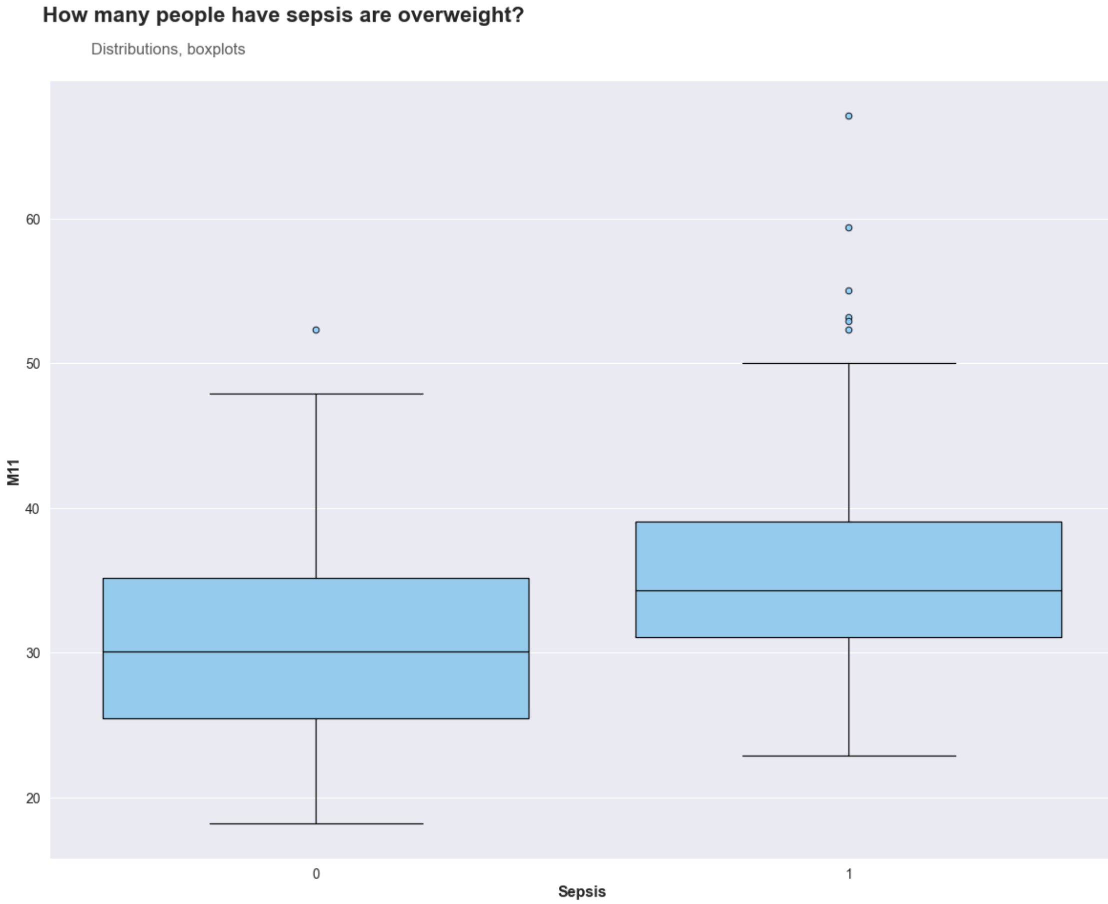
```
# Kernel Density Plot
fig = plt.figure(figsize=(15,8),)
sns.set(style="darkgrid")
ax=sns.kdeplot(train.loc[(train['Sepsis'] == 1),'Age'] ,
color='gray',shade=True)
ax=sns.kdeplot(train.loc[(train['Sepsis'] == 0),'Age'] ,
color='g',shade=True)
plt.title('How old the people having sepsis are?', fontsize = 25, pad
= 40)
plt.ylabel("Frequency of Patient", fontsize = 15, labelpad = 20)
plt.xlabel("Age", fontsize = 15, labelpad = 20)
```

```
Text(0.5, 0, 'Age')
```



3.2.2 How much they weight?

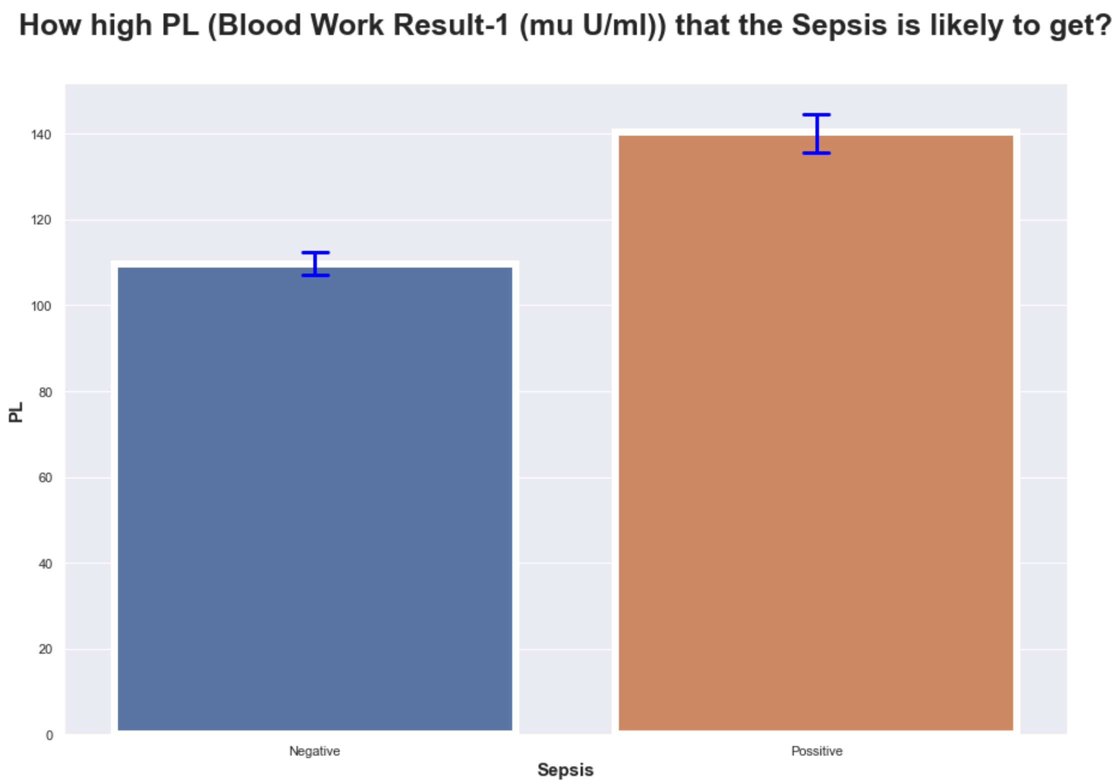
```
plot_box(train, x = "Sepsis", y = "M11", xlabel = 'Sepsis', ylabel =
"M11", title = "How many people have sepsis are overweight?",
         subtitle = "Distributions, boxplots", color = "lightskyblue",
title_position = 0.22, subtitle_position = 0.112)
```



3.2.3 How high PL (Blood Work Result-1 (mu U/ml)) that the Sepsis is likely to get?

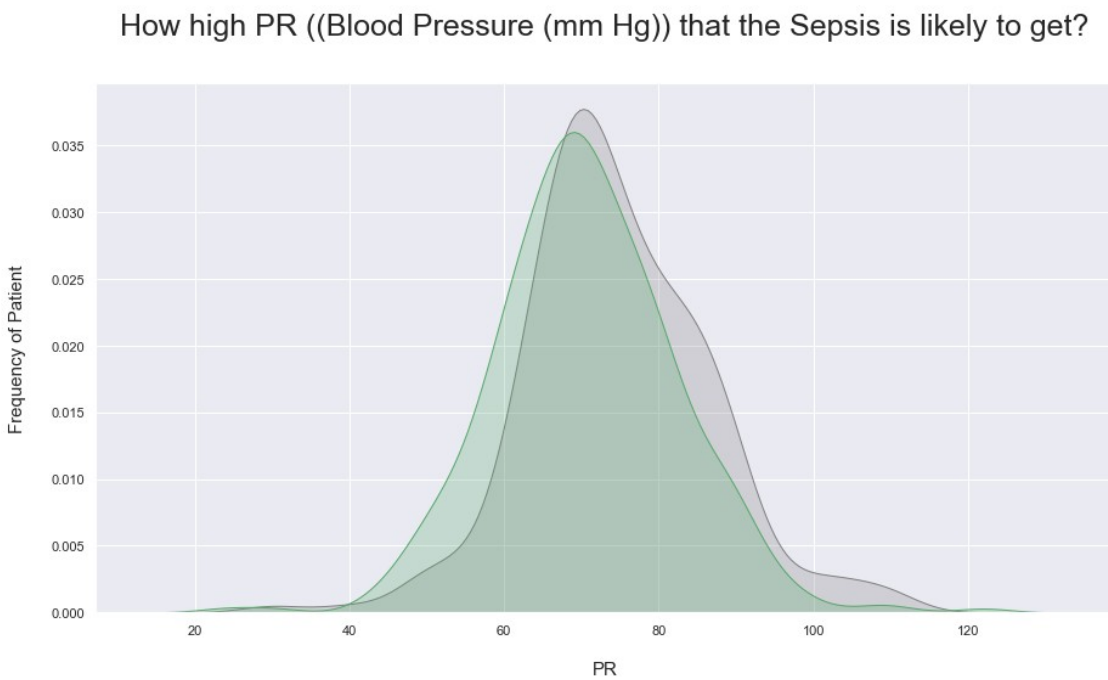
```
plt.subplots(figsize = (15,10))

sns.barplot(x = "Sepsis", y = "PL", data=train, linewidth=6, capsize =
.05, errcolor='blue', errwidth = 3)
plt.title("How high PL (Blood Work Result-1 (mu U/ml)) that the Sepsis
is likely to get?", fontsize = 25, fontweight = 'bold', pad=40)
plt.xlabel("Sepsis", fontsize = 15, fontweight = 'bold')
plt.ylabel("PL", fontsize = 15, fontweight = 'bold')
names = ['Negative', 'Possitive']
#val = sorted(train.Pclass.unique())
val = [0,1] ## this is just a temporary trick to get the label right.
plt.xticks(val, names)
```

```
([<matplotlib.axis.XTick at 0x7fd1eaa989a0>,
  <matplotlib.axis.XTick at 0x7fd1eaa982e0>],
 [Text(0, 0, 'Negative'), Text(1, 0, 'Possitive')])
```



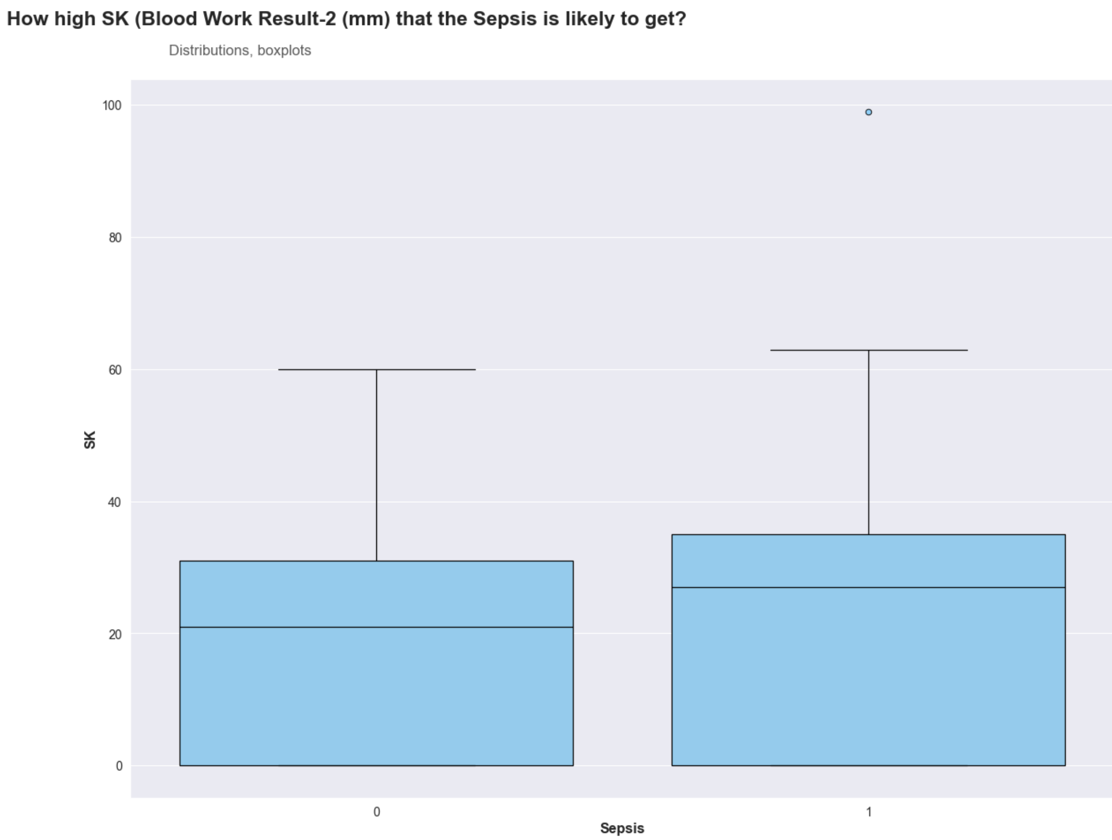
3.2.4 How high PR ((Blood Pressure (mm Hg)) that the Sepsis is likely to get?

```
# Kernel Density Plot
fig = plt.figure(figsize=(15,8),)
sns.set(style="darkgrid")
ax=sns.kdeplot(train.loc[(train['Sepsis'] == 1),'PR'] ,
color='gray',shade=True)
ax=sns.kdeplot(train.loc[(train['Sepsis'] == 0),'PR'] ,
color='g',shade=True)
plt.title('How high PR ((Blood Pressure (mm Hg)) that the Sepsis is
likely to get?', fontsize = 25, pad = 40)
plt.ylabel("Frequency of Patient", fontsize = 15, labelpad = 20)
plt.xlabel("PR", fontsize = 15, labelpad = 20)

Text(0.5, 0, 'PR')
```



3.2.5 How high SK (Blood Work Result-2 (mm) that the Sepsis is likely to get?

```
plot_box(train, x = "Sepsis", y = "SK", xlabel = 'Sepsis', ylabel =
"SK", title = "How high SK (Blood Work Result-2 (mm) that the Sepsis
is likely to get?",
         subtitle = "Distributions, boxplots", color = "lightskyblue",
title_position = 0.22, subtitle_position = 0.112)
```

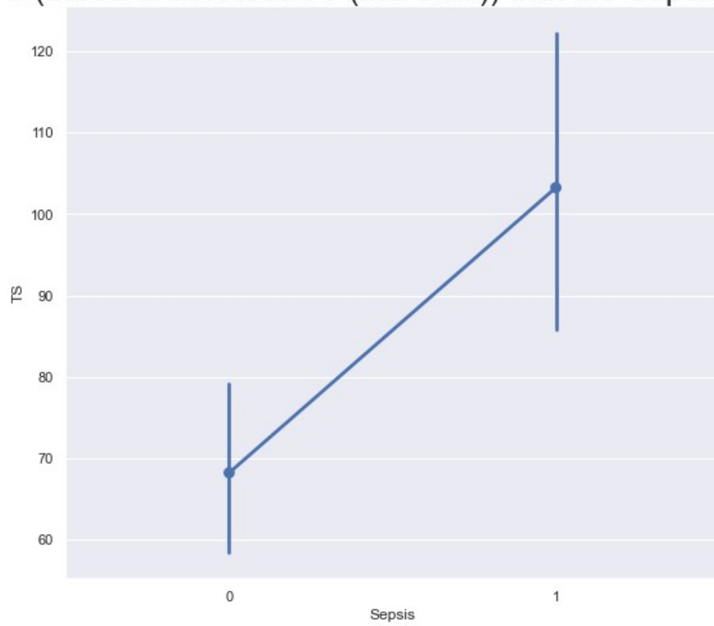


3.2.6 How high TS (Blood Work Result-3 (mu U/ml)) that the Sepsis is likely to get?

```
sns.factorplot(x =  "Sepsis", y = "TS", data = train,kind = 
"point",size = 8)
plt.title('How high TS (Blood Work Result-3 (mu U/ml)) that the Sepsis 
is likely to get?', fontsize = 25)
plt.subplots_adjust(top=0.85)
```

```
/Users/huynhchau/opt/anaconda3/lib/python3.9/site-packages/seaborn/
categorical.py:3717: UserWarning: The `factorplot` function has been 
renamed to `catplot`. The original name will be removed in a future 
release. Please update your code. Note that the default `kind` in 
`factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
/Users/huynhchau/opt/anaconda3/lib/python3.9/site-packages/seaborn/
categorical.py:3723: UserWarning: The `size` parameter has been 
renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

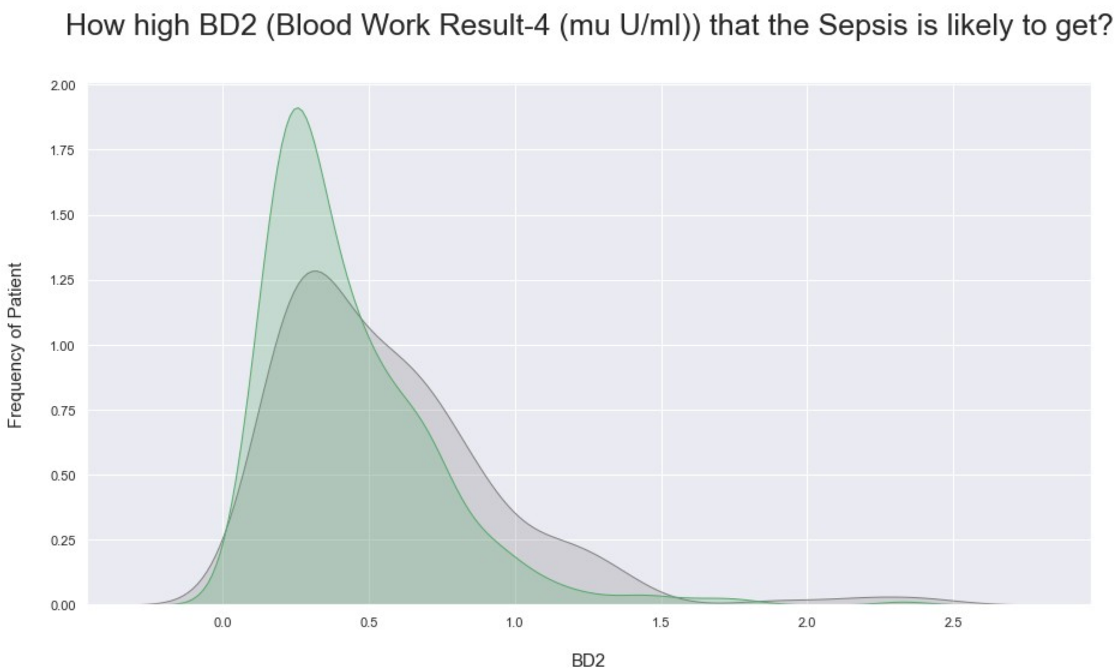
How high TS (Blood Work Result-3 (mu U/ml)) that the Sepsis is likely to get?



3.2.7 How high BD2 (Blood Work Result-4 (mu U/ml)) that the Sepsis is likely to get?

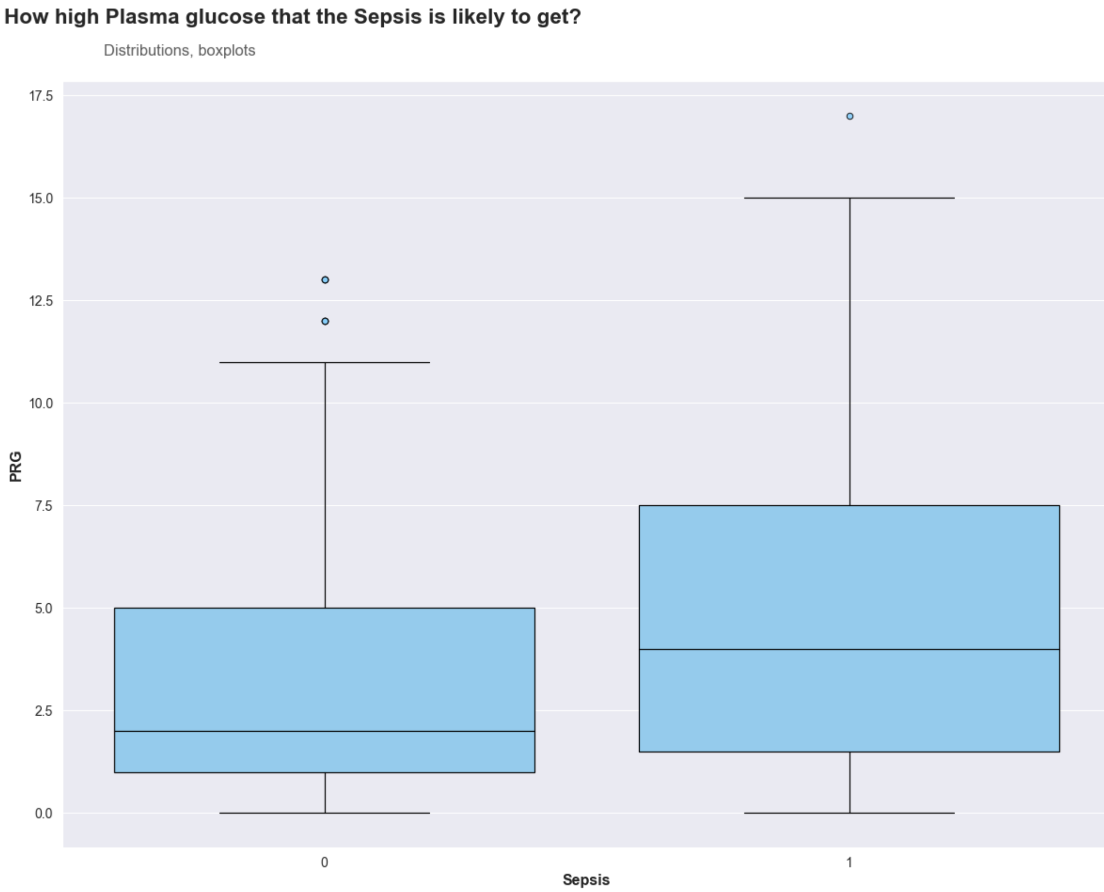
```
# Kernel Density Plot
fig = plt.figure(figsize=(15,8),)
sns.set(style="darkgrid")
ax=sns.kdeplot(train.loc[(train['Sepsis'] == 1),'BD2'] ,
color='gray',shade=True)
ax=sns.kdeplot(train.loc[(train['Sepsis'] == 0),'BD2'] ,
color='g',shade=True)
plt.title('How high BD2 (Blood Work Result-4 (mu U/ml)) that the
Sepsis is likely to get?', fontsize = 25, pad = 40)
plt.ylabel("Frequency of Patient", fontsize = 15, labelpad = 20)
plt.xlabel("BD2", fontsize = 15, labelpad = 20)
```

```
Text(0.5, 0, 'BD2')
```



3.2.8 How high BD2 (Blood Work Result-4 (mu U/ml)) that the Sepsis is likely to get?

```
plot_box(train, x = "Sepsis", y = "PRG", xlabel = 'Sepsis', ylabel =
"PRG", title = "How high Plasma glucose that the Sepsis is likely to
get?",
         subtitle = "Distributions, boxplots", color = "lightskyblue",
title_position = 0.22, subtitle_position = 0.112)
```



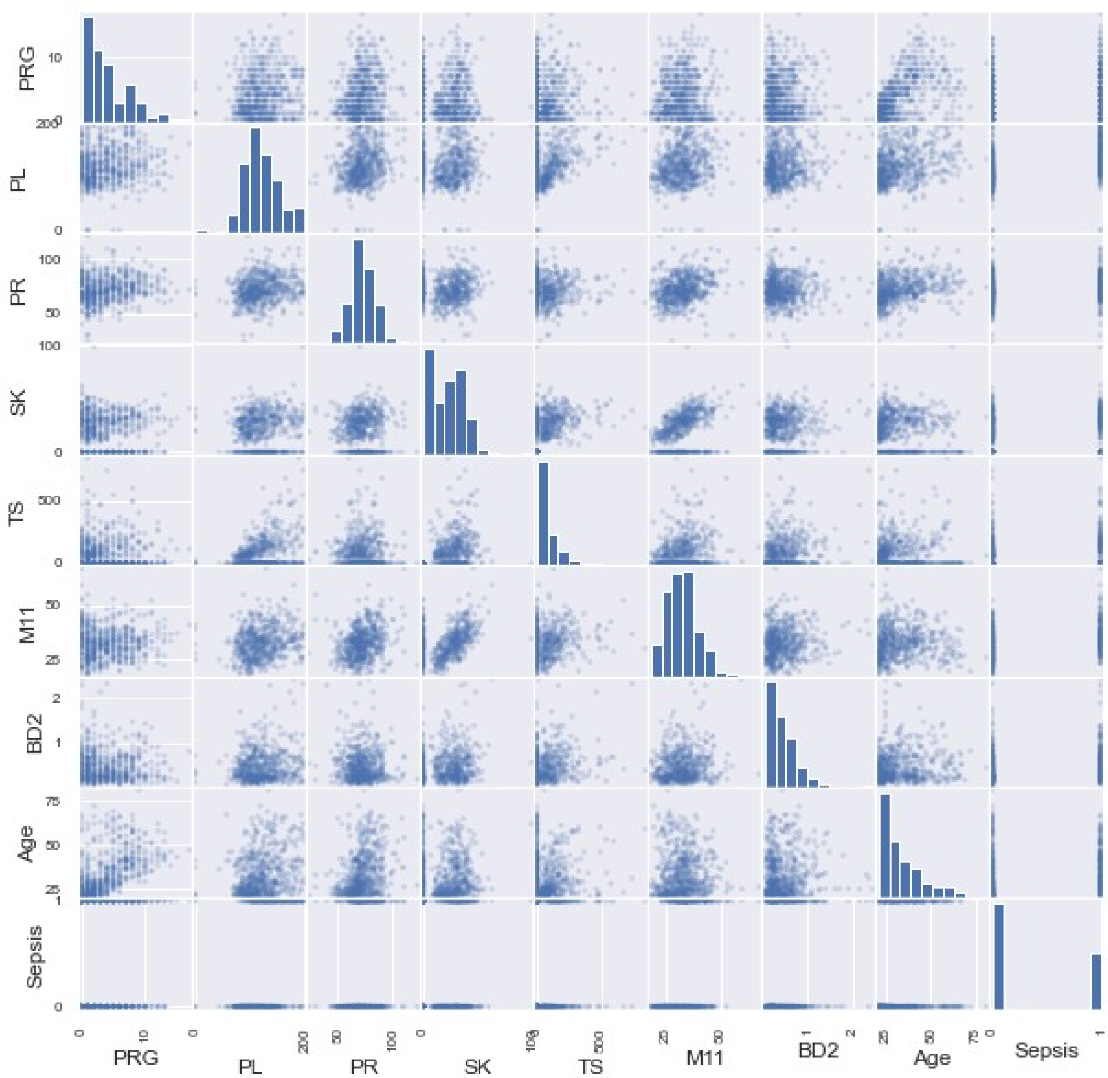
3.2.9 Scatter matrix

```
scatter_matrix(train,alpha=0.2,figsize=(10,10),diagonal='hist')
```

```
array([[<AxesSubplot:xlabel='PRG', ylabel='PRG'>,
        <AxesSubplot:xlabel='PL', ylabel='PRG'>,
        <AxesSubplot:xlabel='PR', ylabel='PRG'>,
        <AxesSubplot:xlabel='SK', ylabel='PRG'>,
        <AxesSubplot:xlabel='TS', ylabel='PRG'>,
        <AxesSubplot:xlabel='M11', ylabel='PRG'>,
        <AxesSubplot:xlabel='BD2', ylabel='PRG'>,
        <AxesSubplot:xlabel='Age', ylabel='PRG'>,
        <AxesSubplot:xlabel='Sepsis', ylabel='PRG'>],
       [<AxesSubplot:xlabel='PRG', ylabel='PL'>,
        <AxesSubplot:xlabel='PL', ylabel='PL'>,
        <AxesSubplot:xlabel='PR', ylabel='PL'>,
        <AxesSubplot:xlabel='SK', ylabel='PL'>,
        <AxesSubplot:xlabel='TS', ylabel='PL'>,
        <AxesSubplot:xlabel='M11', ylabel='PL'>,
        <AxesSubplot:xlabel='BD2', ylabel='PL'>,
```

```
 <AxesSubplot:xlabel='Age', ylabel='PL'>,
 <AxesSubplot:xlabel='Sepsis', ylabel='PL'>],
[<AxesSubplot:xlabel='PRG', ylabel='PR'>,
 <AxesSubplot:xlabel='PL', ylabel='PR'>,
 <AxesSubplot:xlabel='PR', ylabel='PR'>,
 <AxesSubplot:xlabel='SK', ylabel='PR'>,
 <AxesSubplot:xlabel='TS', ylabel='PR'>,
 <AxesSubplot:xlabel='M11', ylabel='PR'>,
 <AxesSubplot:xlabel='BD2', ylabel='PR'>,
 <AxesSubplot:xlabel='Age', ylabel='PR'>,
 <AxesSubplot:xlabel='Sepsis', ylabel='PR'>],
[<AxesSubplot:xlabel='PRG', ylabel='SK'>,
 <AxesSubplot:xlabel='PL', ylabel='SK'>,
 <AxesSubplot:xlabel='PR', ylabel='SK'>,
 <AxesSubplot:xlabel='SK', ylabel='SK'>,
 <AxesSubplot:xlabel='TS', ylabel='SK'>,
 <AxesSubplot:xlabel='M11', ylabel='SK'>,
 <AxesSubplot:xlabel='BD2', ylabel='SK'>,
 <AxesSubplot:xlabel='Age', ylabel='SK'>,
 <AxesSubplot:xlabel='Sepsis', ylabel='SK'>],
[<AxesSubplot:xlabel='PRG', ylabel='TS'>,
 <AxesSubplot:xlabel='PL', ylabel='TS'>,
 <AxesSubplot:xlabel='PR', ylabel='TS'>,
 <AxesSubplot:xlabel='SK', ylabel='TS'>,
 <AxesSubplot:xlabel='TS', ylabel='TS'>,
 <AxesSubplot:xlabel='M11', ylabel='TS'>,
 <AxesSubplot:xlabel='BD2', ylabel='TS'>,
 <AxesSubplot:xlabel='Age', ylabel='TS'>,
 <AxesSubplot:xlabel='Sepsis', ylabel='TS'>],
[<AxesSubplot:xlabel='PRG', ylabel='M11'>,
 <AxesSubplot:xlabel='PL', ylabel='M11'>,
 <AxesSubplot:xlabel='PR', ylabel='M11'>,
 <AxesSubplot:xlabel='SK', ylabel='M11'>,
 <AxesSubplot:xlabel='TS', ylabel='M11'>,
 <AxesSubplot:xlabel='M11', ylabel='M11'>,
 <AxesSubplot:xlabel='BD2', ylabel='M11'>,
 <AxesSubplot:xlabel='Age', ylabel='M11'>,
 <AxesSubplot:xlabel='Sepsis', ylabel='M11'>],
[<AxesSubplot:xlabel='PRG', ylabel='BD2'>,
 <AxesSubplot:xlabel='PL', ylabel='BD2'>,
 <AxesSubplot:xlabel='PR', ylabel='BD2'>,
 <AxesSubplot:xlabel='SK', ylabel='BD2'>,
 <AxesSubplot:xlabel='TS', ylabel='BD2'>,
 <AxesSubplot:xlabel='M11', ylabel='BD2'>,
 <AxesSubplot:xlabel='BD2', ylabel='BD2'>,
 <AxesSubplot:xlabel='Age', ylabel='BD2'>,
 <AxesSubplot:xlabel='Sepsis', ylabel='BD2'>],
[<AxesSubplot:xlabel='PRG', ylabel='Age'>,
 <AxesSubplot:xlabel='PL', ylabel='Age'>,
 <AxesSubplot:xlabel='PR', ylabel='Age'>,
```

```
        <AxesSubplot:xlabel='SK', ylabel='Age'>,
        <AxesSubplot:xlabel='TS', ylabel='Age'>,
        <AxesSubplot:xlabel='M11', ylabel='Age'>,
        <AxesSubplot:xlabel='BD2', ylabel='Age'>,
        <AxesSubplot:xlabel='Age', ylabel='Age'>,
        <AxesSubplot:xlabel='Sepsis', ylabel='Age'>],
       [<AxesSubplot:xlabel='PRG', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='PL', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='PR', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='SK', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='TS', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='M11', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='BD2', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='Age', ylabel='Sepsis'>,
        <AxesSubplot:xlabel='Sepsis', ylabel='Sepsis'>]],
dtype=object)
```



3.3 Statistical Test for Correlation

Null Hypothesis(H_0)**:** people having sepsis have equal medical statistic to people not having sepsis.
Alternative Hypothesis(H_A)**:** people having sepsis have higher medical statistic to people not having sepsis. ****

Select 2 sub dataset

```
sepsis = train[train['Sepsis'] == 1]
not_sepsis = train[train['Sepsis'] == 0]
```

Overall describe by group by

```
train.groupby('Sepsis').describe()
```

```
          PRG                                                      PL  \
        count      mean       std  min  25%  50%  75%   max  count
Sepsis
0       383.0  3.334204  3.024863  0.0  1.0  2.0  5.0  13.0  383.0
1       207.0  4.763285  3.758099  0.0  1.5  4.0  7.5  17.0  207.0

                     ...    BD2           Age
\
             mean    ...    75%    max  count       mean        std
min
Sepsis               ...

0       109.715405   ...  0.581  2.329  383.0  31.660574  11.966782
21.0
1       140.362319   ...  0.736  2.420  207.0  36.613527  10.864363
21.0


         25%   50%   75%   max
Sepsis
0       23.0  27.0  37.0  81.0
1       28.0  35.0  43.0  67.0

[2 rows x 64 columns]
```

---------> OBSERVATION

Overall, the people having sepsis tend to have higher medical statistic than people who do not have sepsis. However, I still want to have a statistical test for proving this is true.

Calculate P-values

```
import researchpy as rp
import scipy.stats as stats
```

```
stats.ttest_ind(train[['Age', 'PRG', 'PL', 'PR', 'SK', 'TS', 'M11',
'BD2']][train['Sepsis'] == 1],
                train[['Age', 'PRG', 'PL', 'PR', 'SK', 'TS', 'M11',
'BD2']][train['Sepsis'] == 0])
```

```
Ttest_indResult(statistic=array([ 4.95274199,  5.01946456,
12.10897277,  3.92209562,  1.71267205,
        3.49999657,  8.47792178,  4.59676448]),
pvalue=array([9.57648953e-07, 6.87703623e-07, 2.73728700e-30,
9.81387055e-05,
       8.73000762e-02, 5.00492364e-04, 1.85290130e-16, 5.25549932e-
06]))
```

---------> OBSERVATION

The p-values is not high enough so we can remove the null hypothesis (H_0). Since then, the higher the medical statistic the higher chance the paitent can get sepsis. That is also the reason why I do not want to drop outliers.

? Reason why I do not drop outliers:

- The higher the medical statistic is the higher chance the paitent can get sepsis.
- The my data set is extremely small, so, if I drop outliers then tha train process only have a considerably small dataset and it cannot "study" properly for every cases.

3.4 Summary

2. In the the 30s or older, people have higher probability to have Sepsis 3. The higher the body mass index is the higher chance that pantient can get Sepsis. The average of body mass index that people having Sepsis is over 35kg/m^2. 4. The higher the PL is the higher chance that pantient can get Sepsis. The average of PL (Blood Work Result-1 (mu U/ml)) that people having Sepsis is higher than 140 while people not have Sepsis have the average PL is around 100 5. The higher the PR is the higher chance that pantient can get Sepsis. The level of PR ((Blood Pressure (mm Hg) that people having Sepsis likely to have is higher than 80 6. The higher the SK is the higher chance that pantient can get Sepsis. The average of SK (Blood Work Result-2 (mm)) that people having Sepsis is higher than 30 while people not have Sepsis have the average SK is around 20 7. The higher the TS is the higher chance that pantient can get Sepsis. The average of TS (Blood Work Result-3 (mu U/ml)) that people having Sepsis is higher than 110 while people not have Sepsis have the average TS is lower than 70 8. The higher the TS is the higher chance that pantient can get Sepsis. The level of BD2 (Blood Work Result-4 (mu U/ml)) that people having Sepsis likely to have is higher than 0.5

4. Feature Engineering

4.1 Class Imbalancing

I want to normalise the target variable since if there is imbalance then the accuracy will be wrong.

```
train['Sepsis'].value_counts()
```

```
0    383
1    207
Name: Sepsis, dtype: int64
```

```
from sklearn.utils import resample
higher_value = train[train.Sepsis==0]
smaller_value = train[train.Sepsis==1]

# Rebalanced smaller class
balancing_values = resample(smaller_value, replace=True,
n_samples=381, random_state=123)

# Balance majority value with upsampled minority value
train = pd.concat([higher_value, balancing_values])

# Show new value in these classes
train.Sepsis.value_counts()
```

```
0    383
1    381
Name: Sepsis, dtype: int64
```

4.2 Splitting the training data

```
# separating our independent and dependent variable
X = train.drop(['Sepsis'], axis = 1)
#Target variable in y
y = train["Sepsis"]

X_train, X_test, y_train, y_test = train_test_split(X, y,test_size =
.2, random_state=42)

print("Length of X_train: " + str(len(X_train)))
print("Length of X_test: " + str(len(X_test)))
```

```
Length of X_train: 611
Length of X_test: 153
```

4.3 Feature Scaling

```
train.sample(5)
```

	PRG	PL	PR	SK	TS	M11	BD2	Age	Sepsis
286	5	155	84.0	44	545	38.7	0.619	34	0
176	6	85	78.0	0	0	31.2	0.382	42	0
142	2	108	52.0	26	63	32.5	0.318	22	0
439	6	107	88.0	0	0	36.8	0.727	31	0
79	2	112	66.0	22	0	25.0	0.307	24	0

```
headers = X_train.columns
X_train.head()
```

	PRG	PL	PR	SK	TS	M11	BD2	Age
532	1	86	66.0	52	65	41.3	0.917	29
276	7	106	60.0	24	0	26.5	0.296	29
472	0	119	66.0	27	0	38.8	0.259	22
523	9	130	70.0	0	0	34.2	0.652	45
147	2	106	64.0	35	119	30.5	1.400	34

------------> OBSERVATION

In this dataset, I realise that the scale of the dataset is quite large so I want to scale it. There are multiple ways to do feature scaling. [5] MinMaxScaler : it use min max to scale, if there is any negative values in the data, it scale them back between 0, and 1. StandardScaler: it makes mean = 0 and scales the data to unit variance. RobustScaler: nearly the same with StandardScaler but, it also use the median, and nterquertile range in order to remove outliers

---

I desire to use standardization scalers for this since I want to apply Logistic regression algorithm for my model, and Logistic regression often generates more reliable predictions with standardization scalers. Nevertheless, there are a lot of outliers in my dataset, so I decided to apply RobustScaler to gently remove those extreme outliers for a better model performance without affecting my dataset.

```
# Feature Scaling
## We will be using RobustScaler to transform
from sklearn.preprocessing import RobustScaler
```

```
scale = RobustScaler()

## transforming "train_x"
X_train = scale.fit_transform(X_train)
## transforming "test_x"
X_test = scale.fit_transform(X_test)

pd.DataFrame(X_train, columns=headers).head()
```

```
   PRG        PL     PR       SK        TS       M11       BD2     Age
0 -0.4 -0.782609 -0.250  0.90625  0.526316  1.058140  1.352718 -0.0625
1  0.8 -0.347826 -0.625  0.03125  0.000000 -0.662791 -0.217446 -0.0625
2 -0.6 -0.065217 -0.250  0.12500  0.000000  0.767442 -0.310999 -0.5000
3  1.2  0.173913  0.000 -0.71875  0.000000  0.232558  0.682680  0.9375
4 -0.2 -0.347826 -0.375  0.37500  0.963563 -0.197674  2.573957  0.2500
```

5. Model training

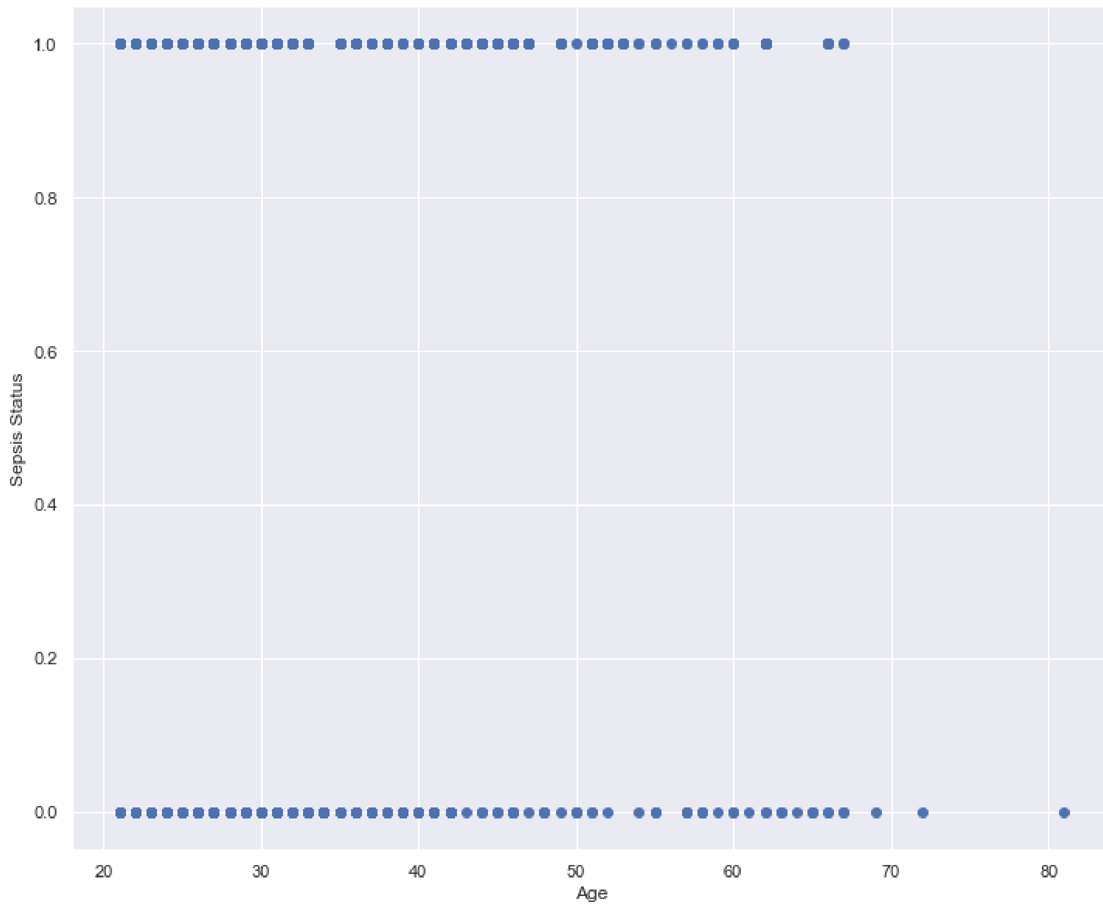
There are five machine learning model that I have learned so far which are Linear Regression, Lasso polynomial regression, Ridge Regression, Logistic Regression, Decision Tree and Random Forest. From that, I chose Logistic Regression, Decision Tree and Random Forest for my assignment 1. The reason will be demonstrated.

Major Keyword for this problem:

- Binary Classification

```
plt.subplots(figsize = (12,10))
plt.scatter(train.Age, train.Sepsis)
plt.xlabel("Age")
plt.ylabel('Sepsis Status')
```

```
Text(0, 0.5, 'Sepsis Status')
```



5.1 Logistic Regression:

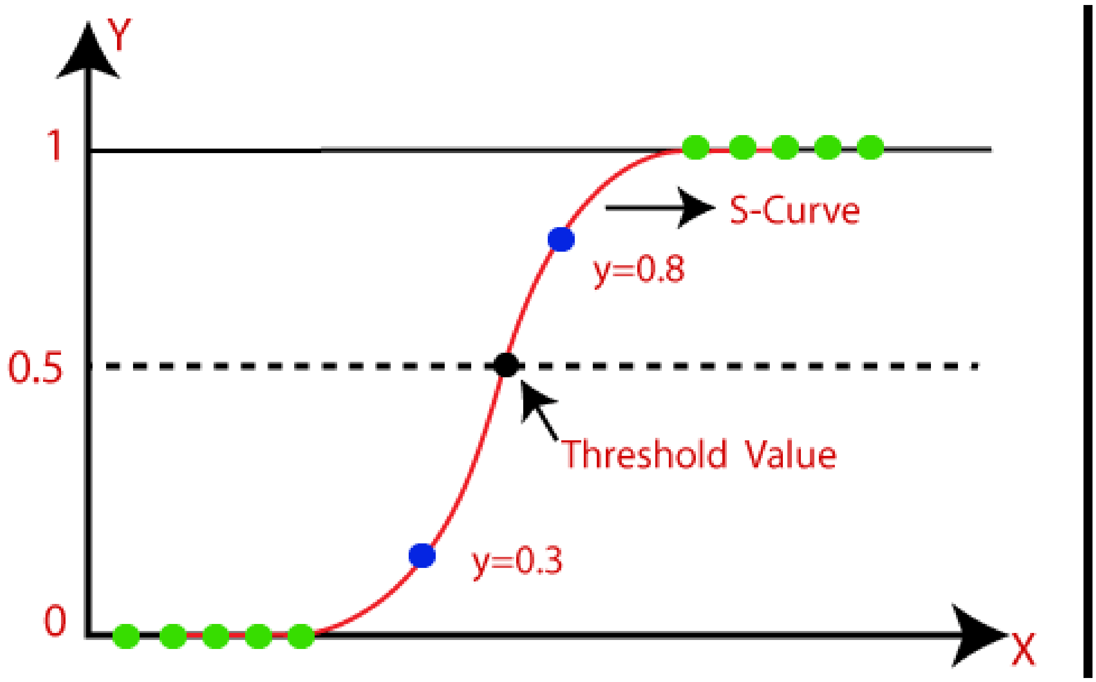
Logistic regression:

- It is a process for training the classification model having the discrete outcomes.
- It assumes there are linear relationships between the target variable (y) with the input features (x), hence, that y values can be determined by a linear combination of all the input features.
- It has the sigmoid function or the "S" curve to fit the data point.

$$ h = \theta_0 + \theta_1 x + \theta_2x + ... + \theta_nx \\ $$

This is the equation for a simple linear regression. here,

- h = Hypothesis h, with respect to weights θ.
- θ = weight of variable
- x = Features/attributes.



Why I chose Logistic regression:

- It does not need to assume there is any linearly seperable.
- It works best for binary classification.
- It works best for a linear problem.

5.1.1 Train Model

```
#Intialise model Logistic Regression:
logreg = LogisticRegression(solver='liblinear', # I use 'liblinear' for solver since this is just a simple prob.
                            penalty= 'l2', # I use 'l2' since it does not remove outliers.
                            intercept_scaling=1e3,
                            max_iter=100000000)

## Fit the model with "train_x" and "train_y"
logreg.fit(X_train,y_train)

## Use the "X_test" to predict the model outcome -> evaluate the outcome of model.
y_pred = logreg.predict(X_test)
## Use the "X_train" to predict the model outcome -> evaluate the outcome of model in train set.
y_pred_train = logreg.predict(X_train)
```

Overfitting

```
## Check if the model is overfitting or not
from sklearn.metrics import f1_score, accuracy_score
## Print F1 score and Accuracy Score:
print("Test F1 Score:" + str(f1_score(y_test, y_pred)))
print("Test Accuracy Score:" + str(accuracy_score (y_test, y_pred)))
print("--------------------------------------------------------")
print("Train F1 Score:" + str(f1_score (y_train, y_pred_train)))
print("Train Accuracy Score:" + str(accuracy_score (y_train,
y_pred_train)))
```

```
Test F1 Score:0.6861313868613139
Test Accuracy Score:0.7189542483660131
--------------------------------------------------------
Train F1 Score:0.7302631578947368
Train Accuracy Score:0.7315875613747954
```

------------> OBSERVATION

Overall, it is a quite good score, and there is no overfitting or underfitting.

5.1.2 Model Evaluation

There are many evaluation techniques for classification problem. There are:

- Confusion Matrix.
- ROC & AUC Curve

Since this is to predict the sepsis of the paintents, so it is neccessary to optimise all the values including accuracy, precision, recall and F1 score. However, I will concentrate mainly on the recall rate. I will explain below.

- Recal is calculated by $\frac{TP}{TP-FN}$. It is determined as the true positive.
- TP: True Positive
- FN: False Negative

--------> So that the recall high means the rate of missing really positive points is low. Meanwhile, the Precision high means that the accuracy of the points found is high. In medical, if the model mistakenly classified the people not having sepsis to have sepsis is not as dangerous as the classifying the people actually having sepsis to not having sepsis.

Confusion Matrix:

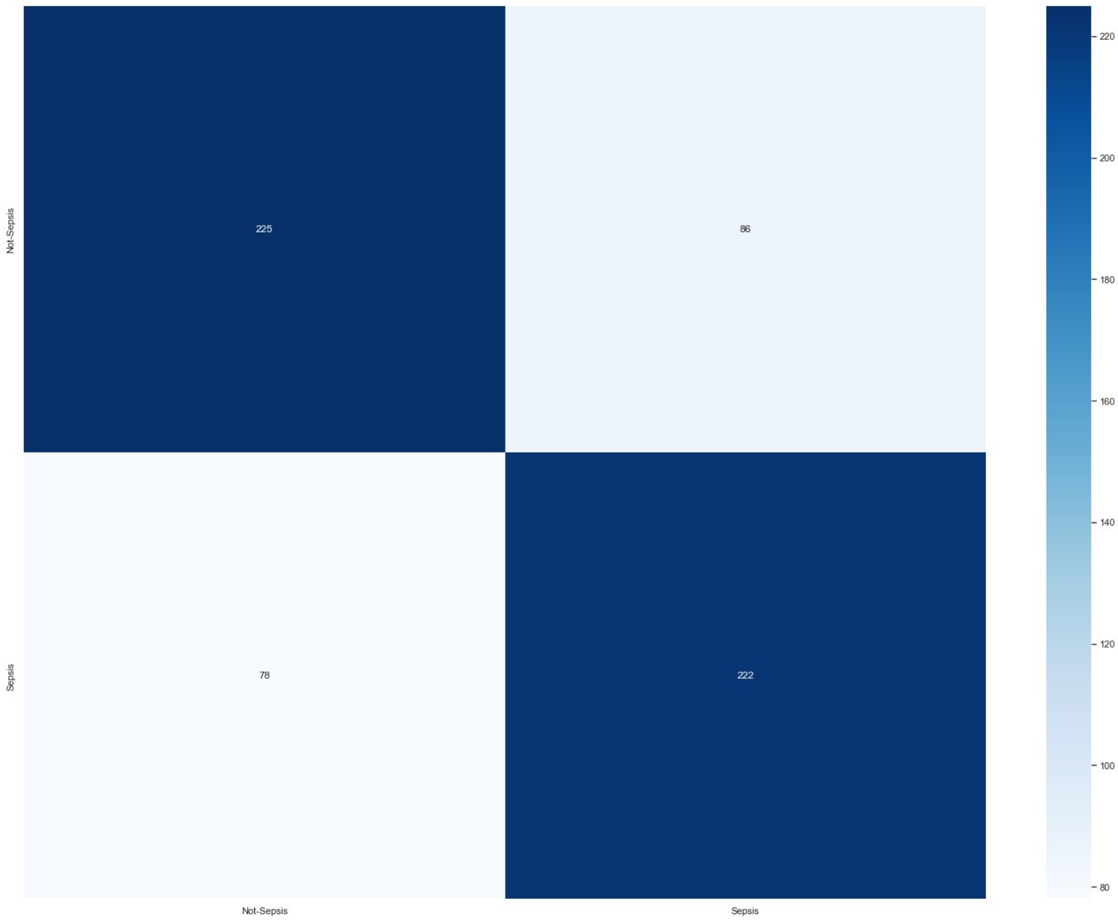
- It is used for calculating the percentage of data that is correctly classified, the class having the highest correct classified prediction, or misclassifies class. It cludes:
- **True Positive(TP)**
- **True Negative(TN)**
- **False Positive(FP)**
- **False Negative(FN)**

```
from sklearn.metrics import classification_report, confusion_matrix
# printing confision matrix
pd.DataFrame(confusion_matrix(y_test,y_pred),\
            columns=["Predicted Not-Sepsis", "Predicted Sepsis"],\
            index=["Not-Sepsis","Sepsis"])
```

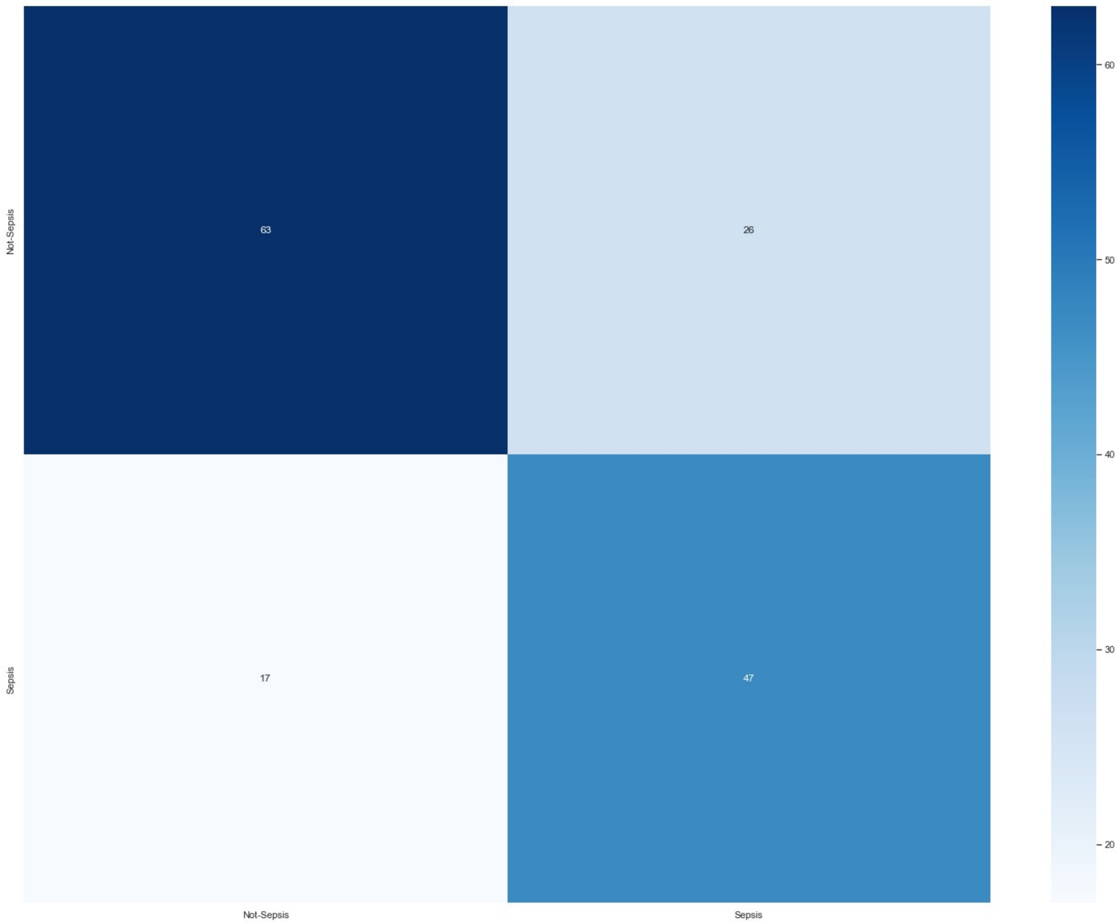
```
            Predicted Not-Sepsis  Predicted Sepsis
Not-Sepsis                    63                17
Sepsis                        26                47
```

```
classes = ['Not-Sepsis','Sepsis']

# helper function
def plot_confusionmatrix(y_train_pred,y_train,dom):
    print(f'{dom} Confusion matrix')
    cf = confusion_matrix(y_train_pred,y_train)
    sns.heatmap(cf,annot=True,yticklabels=classes
               ,xticklabels=classes,cmap='Blues', fmt='g')
    plt.tight_layout()
    plt.show()

print(f'Train score {accuracy_score(y_pred_train,y_train)}')
print(f'Test score {accuracy_score(y_pred,y_test)}')
plot_confusionmatrix(y_pred_train,y_train,dom='Train')
plot_confusionmatrix(y_pred,y_test,dom='Test')
```

```
Train score 0.7315875613747954
Test score 0.7189542483660131
Train Confusion matrix
```



Test Confusion matrix



------------> OBSERVATION

- From that confusion matrix I have:
- ***True Positive(TP)***: 22
- ***True Negative(TN)***: 6
- ***False Positive(FP)***: 18
- ***False Negative(FN)***: 77

Classification report:

```
from sklearn.metrics import classification_report, 
balanced_accuracy_score
print(classification_report(y_test, y_pred))
              precision    recall  f1-score   support

           0       0.71      0.79      0.75        80
           1       0.73      0.64      0.69        73
```

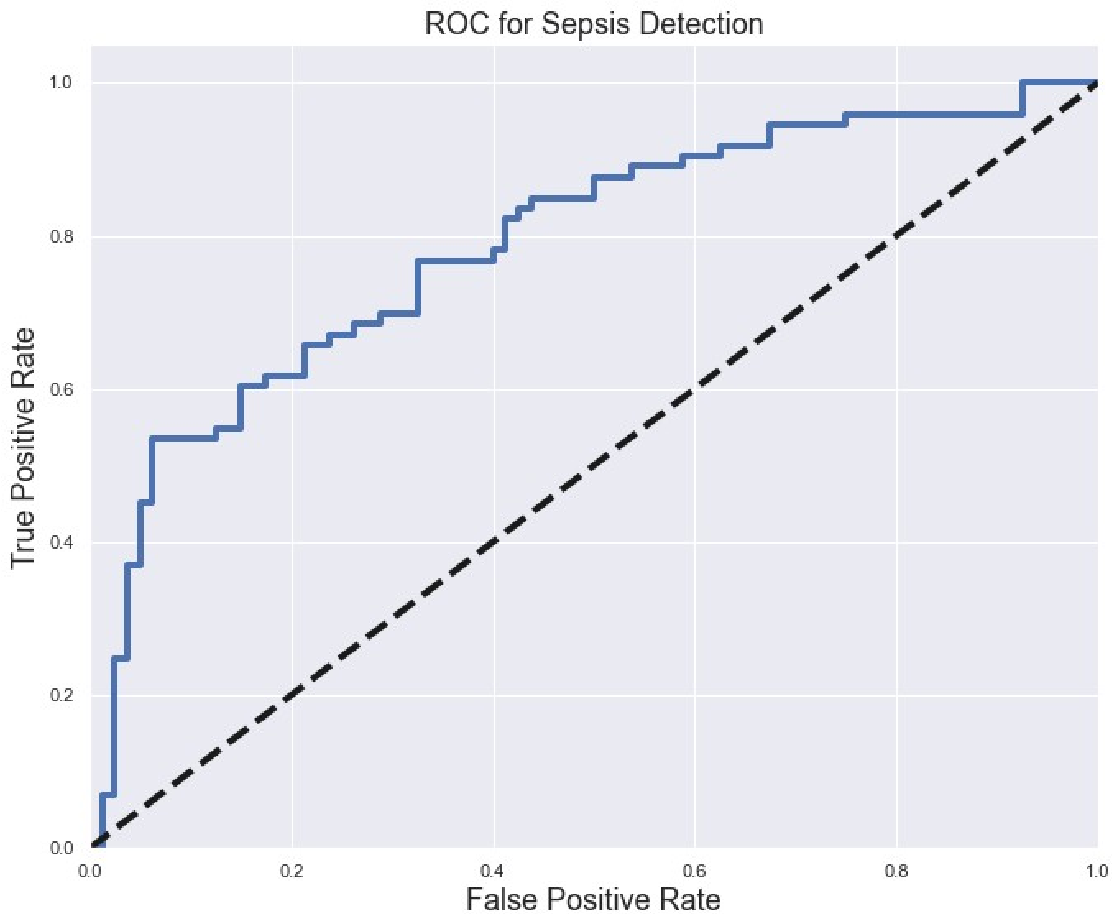
```
    accuracy                           0.72       153
   macro avg       0.72      0.72      0.72       153
weighted avg       0.72      0.72      0.72       153
```

------------> OBSERVATION

- ***Precision***: 0.72 means for each 100 predicted people to have sepsis, only 72 people actually have sepsis
- ***Recall***: 0.72 means for each 100 truly have sepsis, we have 72 people was labeled probably.
- ***F1-score***: It is a harmonic mean of the `Precision` and `Recall`.

AUC & ROC Curve

```
from sklearn.metrics import roc_curve, auc

y_score = logreg.decision_function(X_test)

FPR, TPR, _ = roc_curve(y_test, y_score)
ROC_AUC = auc(FPR, TPR)
print("AUC score: " + str(ROC_AUC))

plt.figure(figsize =[11,9])
plt.plot(FPR, TPR, label= 'ROC curve(area = %0.2f)'%ROC_AUC, 
linewidth= 4)
plt.plot([0,1],[0,1], 'k--', linewidth = 4)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate', fontsize = 18)
plt.ylabel('True Positive Rate', fontsize = 18)
plt.title('ROC for Sepsis Detection', fontsize= 18)
plt.show()

AUC score: 0.7880136986301368
```



------------> OBSERVATION

- In medical diagnosis, it is very important to have the model can have AUC reach 0.95 or higher.

Using Cross-validation:

```
accuracy = cross_val_score(logreg, X, y, scoring='accuracy', cv = 10)
print(accuracy)
#get the mean of each fold
print("Accuracy of Model with Cross Validation is:",accuracy.mean() *
100)
```

```
[0.55844156 0.81818182 0.72727273 0.76623377 0.73684211 0.72368421
 0.76315789 0.65789474 0.81578947 0.72368421]
Accuracy of Model with Cross Validation is: 72.91182501708818
```

5.1.3 Hypertuning parameter

Logistic regression have no critical parameter for hypertuning. I will hypertuning those parameter:

- solver
- penalty
- c values
- max_iter
- intercept_scaling

```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
#Parameters for Logistic Regression
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l2'] # Since Solver newton-cg and lbfgs support only 'l2'
or 'none' penalties.
lMaxIter = [1e4, 1e5, 1e6]
c_values = [1e4, 1e5, 1e6]
intercept_scaling= [1e4, 1e5, 1e6]

model = LogisticRegression()
random_state = [i for i in (0, 42)]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values,
max_iter=lMaxIter, intercept_scaling=intercept_scaling)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-
1, cv=cv, scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_,
grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.725239 using {'C': 100000.0, 'intercept_scaling': 100000.0,
'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 10000.0,
'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 10000.0,
'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 10000.0, 'intercept_scaling': 10000.0,
'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 10000.0,
'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 10000.0,
'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
```

0.722186 (0.052320) with: {'C': 10000.0, 'intercept_scaling': 10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 10000.0, 'intercept_scaling': 10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.723485 (0.050296) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.723485 (0.050296) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.723485 (0.050296) with: {'C': 10000.0, 'intercept_scaling': 100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.710401 (0.047841) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.710401 (0.047841) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling': 1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 10000.0, 'intercept_scaling':

```
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.710401 (0.047841) with: {'C': 10000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 100000.0, 'intercept_scaling':
10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.725239 (0.053282) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.725239 (0.053282) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.725239 (0.053282) with: {'C': 100000.0, 'intercept_scaling':
100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
```

```
1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.709097 (0.047346) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.709097 (0.047346) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.709097 (0.047346) with: {'C': 100000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 1000000.0, 'intercept_scaling':
10000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
```

```
100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.722186 (0.052320) with: {'C': 1000000.0, 'intercept_scaling':
100000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.708219 (0.046937) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 10000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.708219 (0.046937) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 100000.0, 'penalty': 'l2', 'solver':
'liblinear'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
0.722625 (0.052496) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver': 'lbfgs'}
0.708219 (0.046937) with: {'C': 1000000.0, 'intercept_scaling':
1000000.0, 'max_iter': 1000000.0, 'penalty': 'l2', 'solver':
'liblinear'}
```

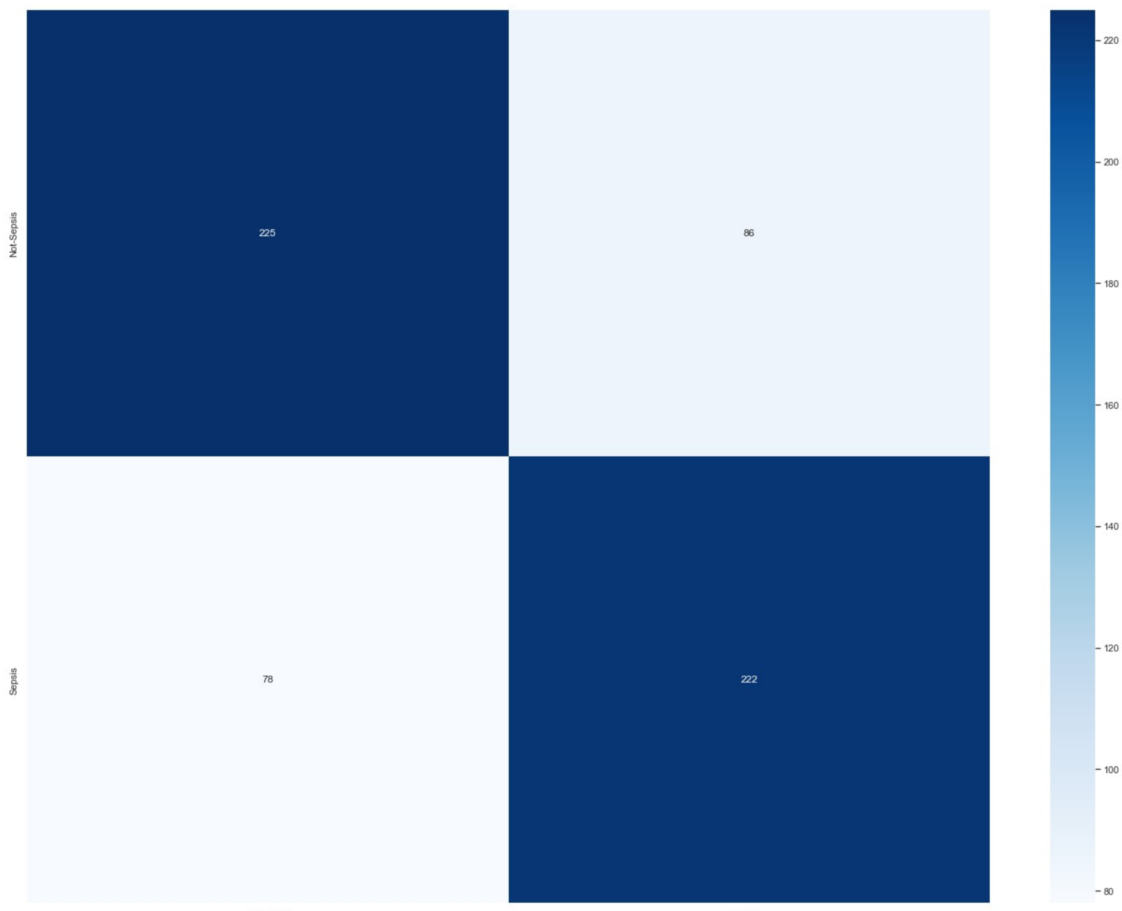
```
## Getting the best of score.
print (grid_search.best_score_)
## Getting the best of parameter.
print (grid_search.best_params_)
## Getting the best of estimator.
print(grid_search.best_estimator_)
```

```
0.7252392344497607
{'C': 100000.0, 'intercept_scaling': 100000.0, 'max_iter': 10000.0,
'penalty': 'l2', 'solver': 'liblinear'}
LogisticRegression(C=100000.0, intercept_scaling=100000.0,
max_iter=10000.0,
                   solver='liblinear')
```

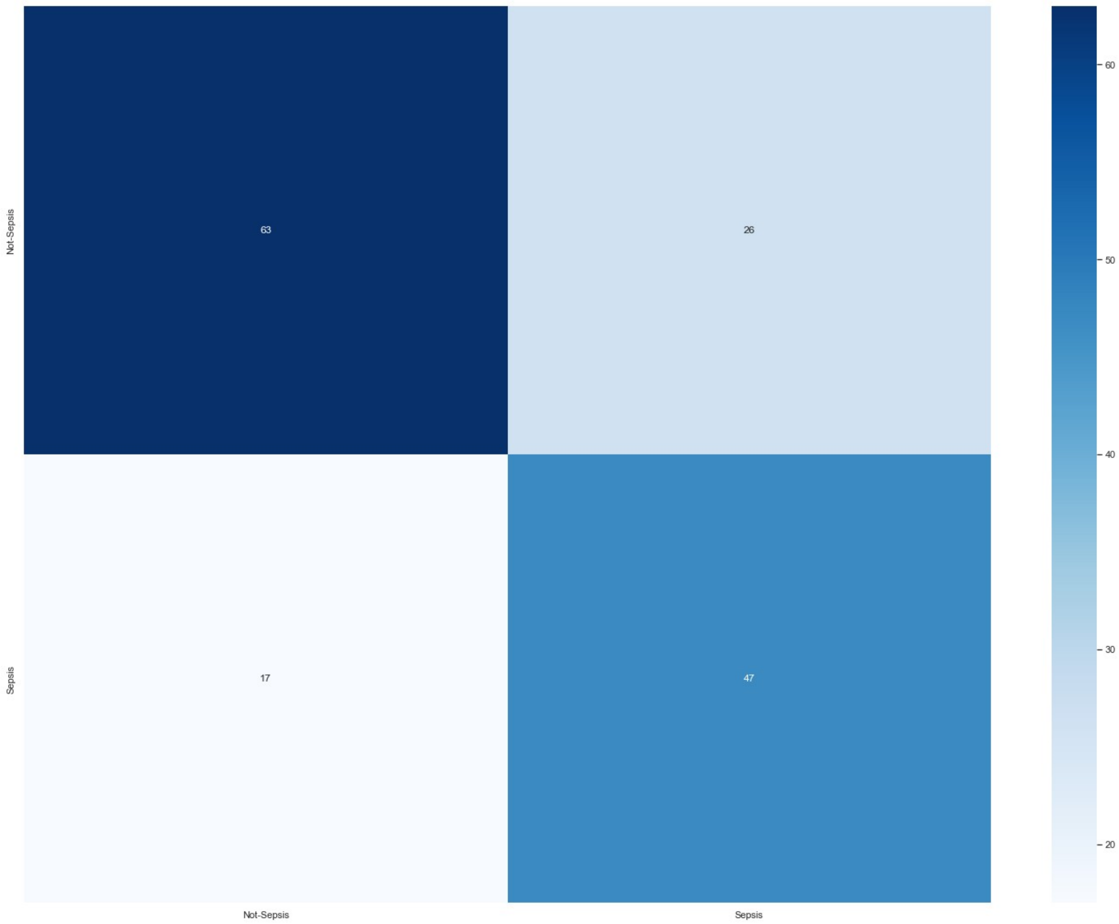
5.1.4 Retrain

```
model = LogisticRegression(C=10000.0, intercept_scaling=10000.0,
max_iter=10000.0, penalty='l2',
                   solver='newton-cg')
model.fit(X_train,y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
plot_confusionmatrix(y_train_pred,y_train,dom='Train')
plot_confusionmatrix(y_test_pred,y_test,dom='Test')
```

```
Train score 0.7315875613747954
Test score 0.7189542483660131
Train Confusion matrix
```




```
Test Confusion matrix
```




```
print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.71      0.79      0.75        80
           1       0.73      0.64      0.69        73

    accuracy                           0.72       153
   macro avg       0.72      0.72      0.72       153
weighted avg       0.72      0.72      0.72       153
```

5.1.5 Conclusion

- In order to valuate the Logistic Regression Model, I use some methods including the accuracy, AUC score, precision, recall and f1-score.
- Since the accuracy score only concentrates on the true and correct values, nevertheless, this is the medical problem so it also requires the wrong rate to be

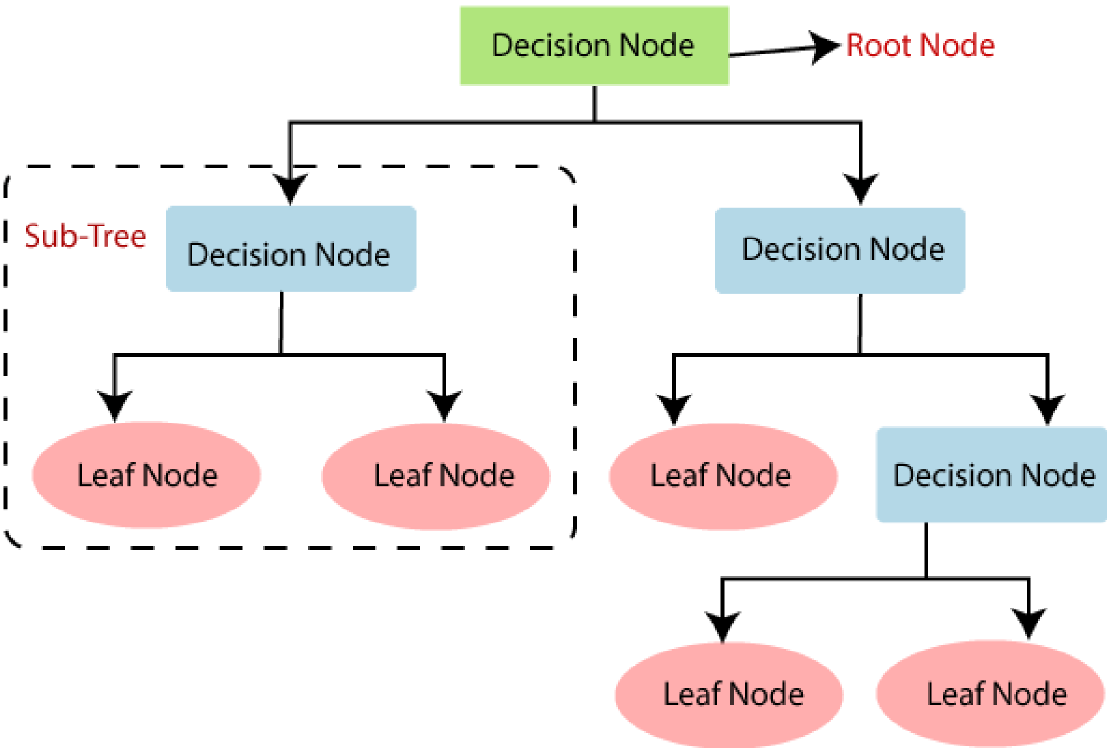
minimum as much as possible. Since then, the precision, recall and f1-score are good methods.

- In this model
- AUC score: 0.79 (The area under the ROC curve (AUC) results were considered excellent for AUC values between 0.9-1, good for AUC values between 0.8-0.9, fair for AUC values between 0.7-0.8, poor for AUC values between 0.6-0.7 and failed for AUC values between 0.5-0.6.) [6]
- The classification report demonstrates the precision, recall, and f1-score. Overall, the support columns reported the number of occurence of each class in my dataset. According to the report, the dataset is not well balanced, that is the reason why the scores is siginificantly different.
    - Precision classify the sepsis: 0.73
    - Recall classify the sepsis: 0.63
    - F1-score classify the sepsis: 0.68

5.2 Decision Tree Classifier:

Decision Tree Classifier:

- It is a supervised learning strategy that can be used for both classificaton and regression.
- It has the branch to represent a decision rule, and the outcomes are presented in each leafnode.
- It uses the Attribute Selection Measures(ASM) to split the leafnode.
- Some of the most popular selection measurement are Information Gain, Gain Ratio, and Gini Index.



Why I chose Decision Tree Classifier:

- It works best with classification.
- It is easy to interpret and visualize.
- I do not need to normalise the data.
- It has no assumptions about distribution because of the non-parametric nature of the algorithm.

5.2.1 Train Model:

```
# Initialise the model
clf=DecisionTreeClassifier(random_state=42)

# Fit the model with train set
clf.fit(X_train,y_train)

## Use the "X_train" to predict the model outcome -> evaluate the
outcome of model in train set.
y_train_predicted=clf.predict(X_train)

## Use the "X_test" to predict the model outcome -> evaluate the
outcome of model.
y_test_predicted=clf.predict(X_test)
```

Overfitting

```
## Check if the model is overfitting or not?
from sklearn.metrics import f1_score, accuracy_score
print("Test F1 Score:" + str(f1_score(y_test, y_test_predicted)))
print("Test Accuracy Score:" + str(accuracy_score (y_test,
y_test_predicted)))
print("---------------------------------------------------------")
print("Train F1 Score:" + str(f1_score (y_train, y_train_predicted)))
print("Train Accuracy Score:" + str(accuracy_score (y_train,
y_train_predicted)))
```

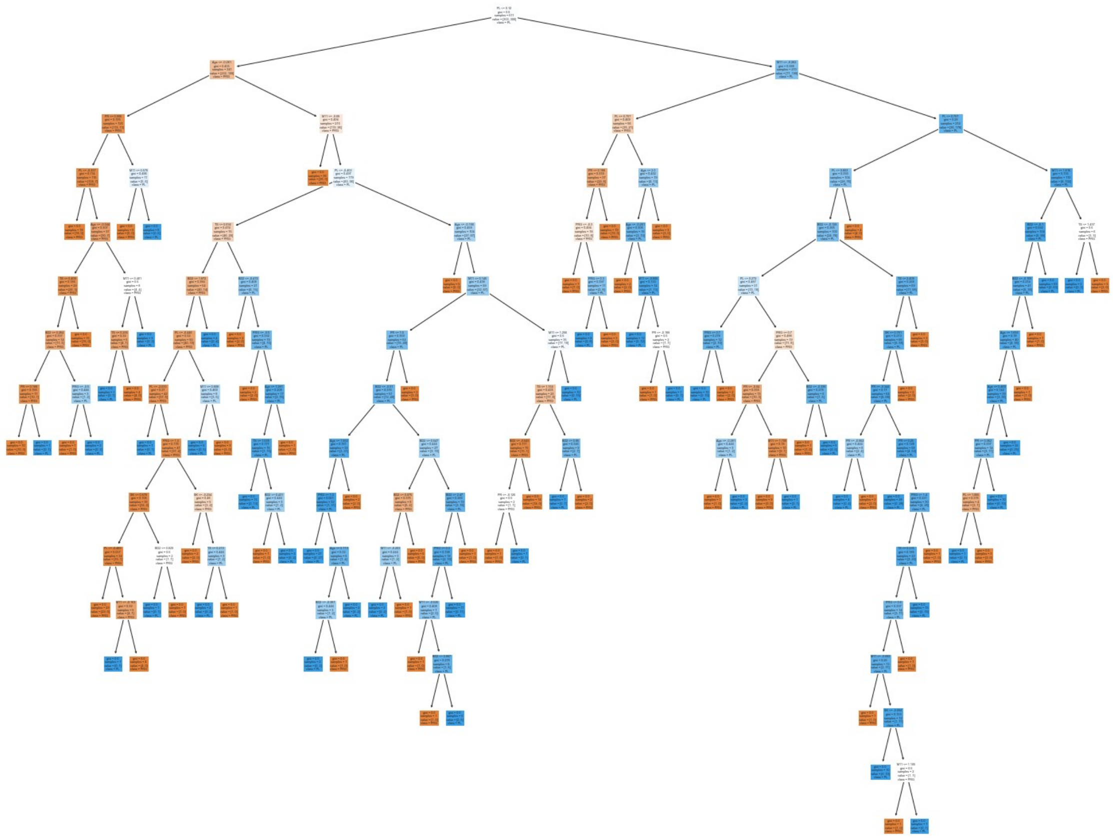
```
Test F1 Score:0.7671232876712328
Test Accuracy Score:0.7777777777777778
--------------------------------------------------------
Train F1 Score:1.0
Train Accuracy Score:1.0
```

------------> OBSERVATION

There is a large overfitting. However, the accuracy is not high as expected, so I will have pruning process and hyperparameter tuning process to make the accuracy equal.

Plot The Decision Tree

```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(clf,
                   feature_names=['PRG', 'PL', 'PR', 'SK', 'TS',
'M11', 'BD2', 'Age'],
                   class_names=['PRG', 'PL', 'PR', 'SK', 'TS', 'M11',
'BD2', 'Age'],
                   filled=True)
```



5.2.2 Hypertuning & Pruning:

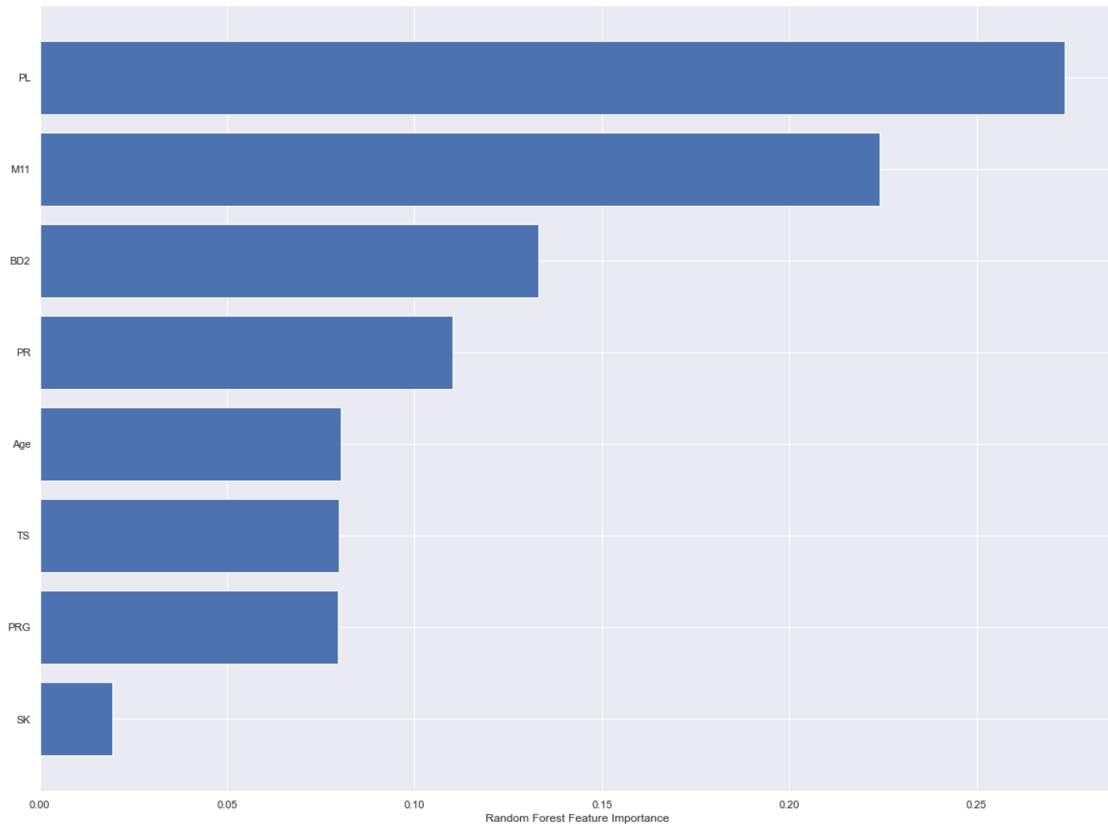
Feature importance for this decision tree:

```
column_names = X.columns
## feature importance
feature_importances = pd.DataFrame(clf.feature_importances_, index =
column_names, columns=['importance'])
feature_importances.sort_values(by='importance',
ascending=False).head(10)
```

```
     importance
PL     0.273419
M11    0.224177
Age    0.133194
BD2    0.110234
PR     0.080244
TS     0.079849
PRG    0.079496
SK     0.019387
```

```
sorted_idx = clf.feature_importances_.argsort()
plt.barh(['SK', 'PRG', 'TS', 'Age', 'PR', 'BD2', 'M11', 'PL'],
clf.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

```
Text(0.5, 0, 'Random Forest Feature Importance')
```



------------> OBSERVATION

In this decision tree, **PL, M11 and BD2** are three main important feature for classifying the sepsis and not sepsis patient.

5.2.2.a Post Pruning:

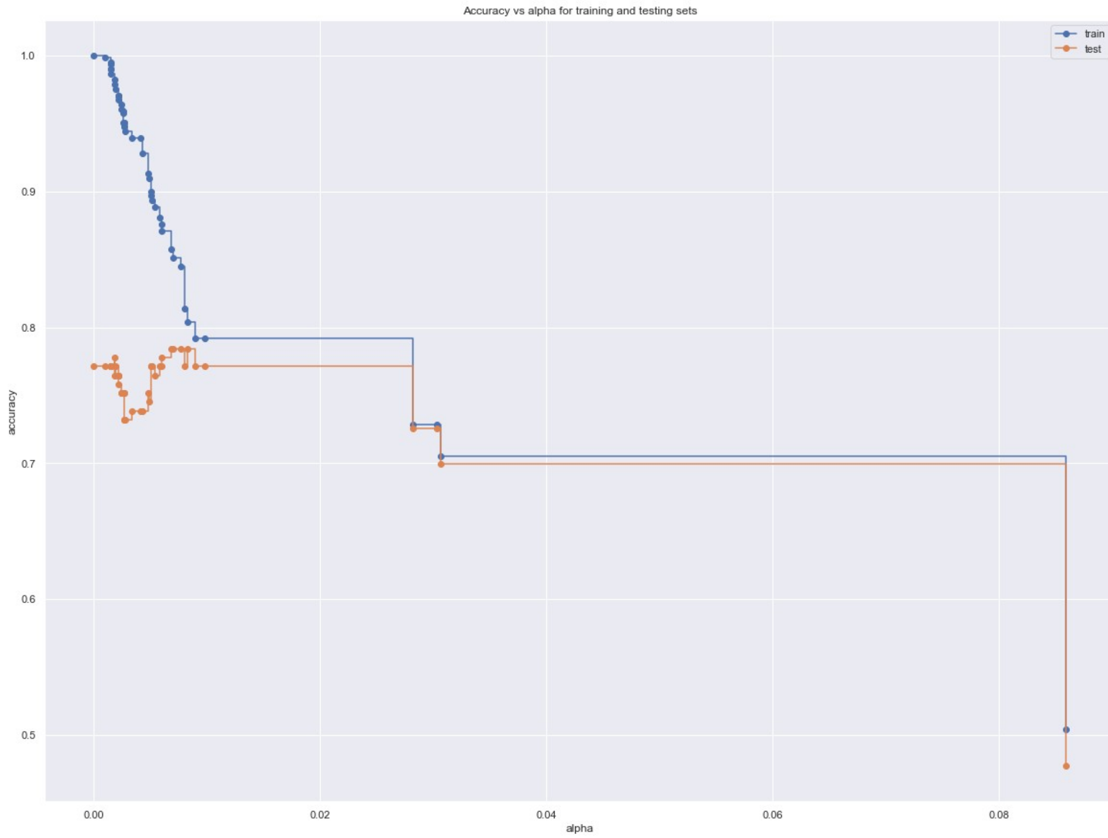
```
path=clf.cost_complexity_pruning_path(X_train,y_train)
#path variable gives two things ccp_alphas and impurities
ccp_alphas,impurities=path.ccp_alphas,path.impurities
print("ccp alpha will give list of values :",ccp_alphas)
print("***********************************************************")
print("Impurities in Decision Tree :",impurities)
```

```
ccp alpha will give list of values : [0.         0.00105701 0.00150027
0.00151976 0.00153437 0.00153437
 0.00154574 0.00183795 0.00187765 0.00193975 0.00218221 0.00218221
 0.00218221 0.0022073  0.00245021 0.00245499 0.00260231 0.00261866
 0.00262495 0.0026858  0.00270771 0.00272777 0.00272777 0.00280419
 0.00336526 0.00412604 0.00435117 0.00480891 0.00493921 0.00506741
 0.00509183 0.00521376 0.00545913 0.00588419 0.00599232 0.00603938
 0.00684154 0.00701431 0.00775899 0.00806801 0.00832212 0.00899989
 0.0098549  0.02821904 0.03031327 0.03065383 0.08593055]
***********************************************************
Impurities in Decision Tree : [0.         0.00317103 0.00617158
0.00921109 0.01227983 0.01534857
 0.02153151 0.03072127 0.03635422 0.04217346 0.04435568 0.04653789
 0.04872011 0.055342   0.05779221 0.0602472  0.06284951 0.06546817
 0.07859293 0.08396454 0.08667225 0.08940002 0.09212779 0.09773617
 0.10783194 0.11608402 0.12478636 0.13921308 0.14415229 0.15935451
 0.16444634 0.1696601  0.17511923 0.1868876  0.19887225 0.20491163
 0.22543625 0.23946487 0.24722385 0.27949589 0.29614014 0.30514003
 0.32484983 0.35306887 0.38338214 0.41403597 0.49996652]
```

```python
clfs=[]   #will store all the models here
for ccp_alpha in ccp_alphas:
    clf=DecisionTreeClassifier(random_state=0,ccp_alpha=ccp_alpha)
    clf.fit(X_train,y_train)
    clfs.append(clf)
print("Last node in Decision tree is {} and ccp_alpha for last node is
{}".format(clfs[-1].tree_.node_count,

ccp_alphas[-1]))
```

```
Last node in Decision tree is 1 and ccp_alpha for last node is
0.0859305514453022
```

```python
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o',
label="train",drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker='o',
label="test",drawstyle="steps-post")
ax.legend()
plt.show()
```



------------> OBSERVATION

According to this plot the plot, the ccp_alpha that has the nearly equal train and test set is at 0.00899989.

Retrain the model with ccp_alpha

```
# Initialise the model
clf=DecisionTreeClassifier(random_state=42,ccp_alpha=0.00899989)

# Fit the model with train set
clf.fit(X_train,y_train)

## Use the "X_train" to predict the model outcome -> evaluate the
outcome of model in train set.
y_train_predicted = clf.predict(X_train)

## Use the "X_test" to predict the model outcome -> evaluate the
outcome of model.
y_test_predicted = clf.predict(X_test)
```

```
## Check if the model is overfitting or not?
from sklearn.metrics import f1_score, accuracy_score
print("Test F1 Score:" + str(f1_score(y_test, y_test_predicted)))
print("Test Accuracy Score:" + str(accuracy_score (y_test,
y_test_predicted)))
print("---------------------------------------------------------")
print("Train F1 Score:" + str(f1_score (y_train, y_train_predicted)))
print("Train Accuracy Score:" + str(accuracy_score (y_train,
y_train_predicted)))
```

```
Test F1 Score:0.7785234899328859
Test Accuracy Score:0.7843137254901961
--------------------------------------------------------
Train F1 Score:0.8198198198198199
Train Accuracy Score:0.8036006546644845
```

5.2.2.b Pre Pruning:

```
grid_param={"criterion":["gini","entropy"],
            "splitter":["best","random"],
            "max_depth":range(2,400,50),
            "min_samples_leaf":np.arange(2,50,5)
           }
grid_search=GridSearchCV(estimator=clf,param_grid=grid_param,cv=5,n_jo
bs=-1)
grid_search.fit(X_train,y_train)
```

```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(ccp_alpha=0.00899989,
                                              random_state=42),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 400, 50),
                         'min_samples_leaf': array([ 2,  7, 12, 17,
22, 27, 32, 37, 42, 47]),
                         'splitter': ['best', 'random']})
```

```
pd.DataFrame(grid_search.cv_results_)
```

```
     mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0         0.002824      0.000360         0.000677        0.000126
1         0.001743      0.000238         0.000628        0.000129
2         0.002281      0.000260         0.000636        0.000020
3         0.001795      0.000120         0.000729        0.000157
4         0.002385      0.000108         0.000716        0.000082
..             ...           ...              ...             ...
315       0.001351      0.000079         0.000444        0.000063
316       0.002010      0.000245         0.000478        0.000122
```

```
317       0.001216      0.000098         0.000422        0.000100
318       0.001903      0.000180         0.000553        0.000086
319       0.001254      0.000119         0.000434        0.000100

    param_criterion param_max_depth param_min_samples_leaf
param_splitter  \
0              gini               2                      2
best
1              gini               2                      2
random
2              gini               2                      7
best
3              gini               2                      7
random
4              gini               2                     12
best
..              ...             ...                    ...
...
315         entropy             352                     37
random
316         entropy             352                     42
best
317         entropy             352                     42
random
318         entropy             352                     47
best
319         entropy             352                     47
random

                                                params
split0_test_score  \
0    {'criterion': 'gini', 'max_depth': 2, 'min_sam...
0.707317
1    {'criterion': 'gini', 'max_depth': 2, 'min_sam...
0.723577
2    {'criterion': 'gini', 'max_depth': 2, 'min_sam...
0.707317
3    {'criterion': 'gini', 'max_depth': 2, 'min_sam...
0.723577
4    {'criterion': 'gini', 'max_depth': 2, 'min_sam...
0.707317
..                                                 ...
...
315  {'criterion': 'entropy', 'max_depth': 352, 'mi...
0.658537
316  {'criterion': 'entropy', 'max_depth': 352, 'mi...
0.756098
317  {'criterion': 'entropy', 'max_depth': 352, 'mi...
0.707317
318  {'criterion': 'entropy', 'max_depth': 352, 'mi...
```

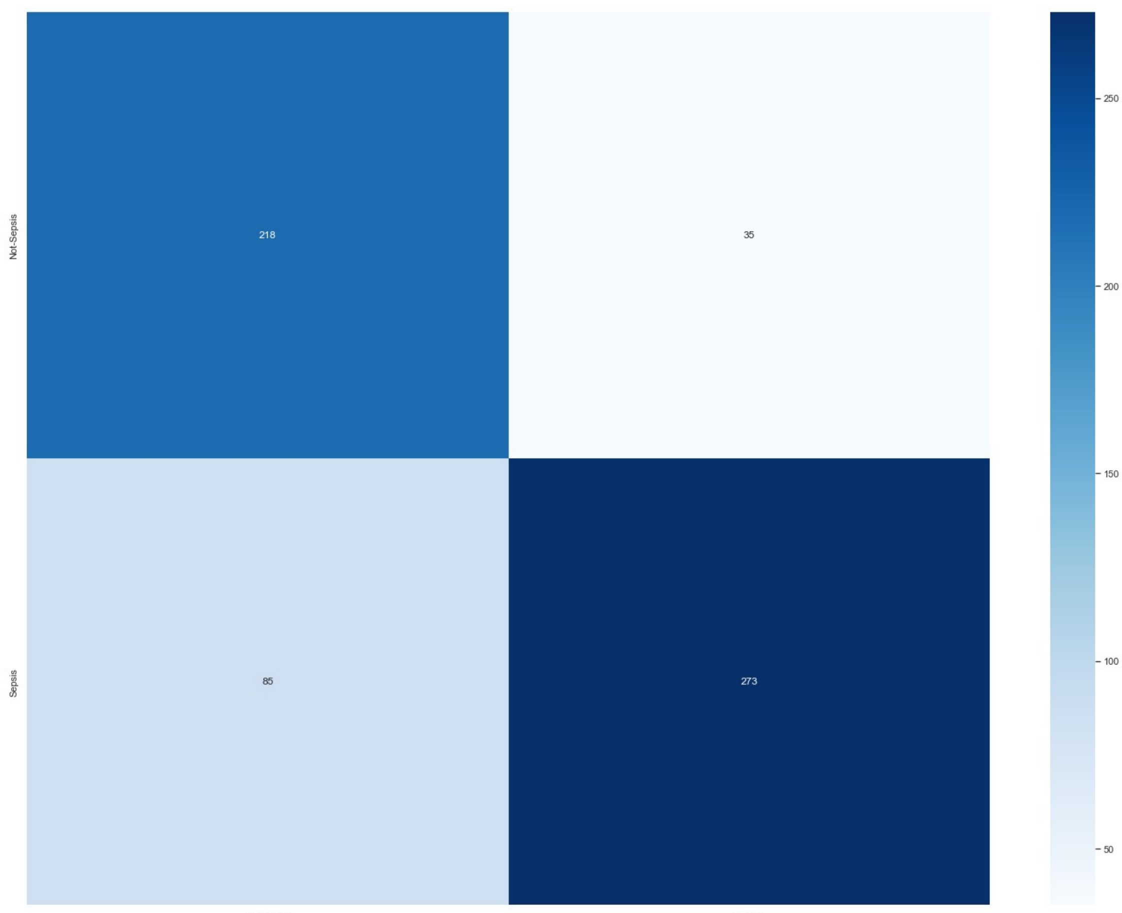
```
0.796748
319  {'criterion': 'entropy', 'max_depth': 352, 'mi...
0.658537

     split1_test_score  split2_test_score  split3_test_score  \
0             0.745902           0.704918           0.745902
1             0.696721           0.713115           0.713115
2             0.745902           0.704918           0.745902
3             0.696721           0.713115           0.713115
4             0.745902           0.704918           0.745902
..                 ...                ...                ...
315           0.704918           0.737705           0.688525
316           0.721311           0.704918           0.729508
317           0.704918           0.737705           0.696721
318           0.721311           0.688525           0.754098
319           0.704918           0.647541           0.663934

     split4_test_score  mean_test_score  std_test_score
rank_test_score
0             0.713115         0.723431        0.018540
141
1             0.688525         0.707011        0.012622
193
2             0.713115         0.723431        0.018540
141
3             0.688525         0.707011        0.012622
193
4             0.713115         0.723431        0.018540
141
..                 ...              ...             ...              .
..
315           0.688525         0.695642        0.025820
225
316           0.762295         0.734826        0.021506
113
317           0.688525         0.707037        0.016699
179
318           0.754098         0.742956        0.036258
92
319           0.672131         0.669412        0.019466
286

[320 rows x 17 columns]
```

```
## Getting the best of everything.
print (grid_search.best_score_)
print (grid_search.best_params_)
print(grid_search.best_estimator_)
```

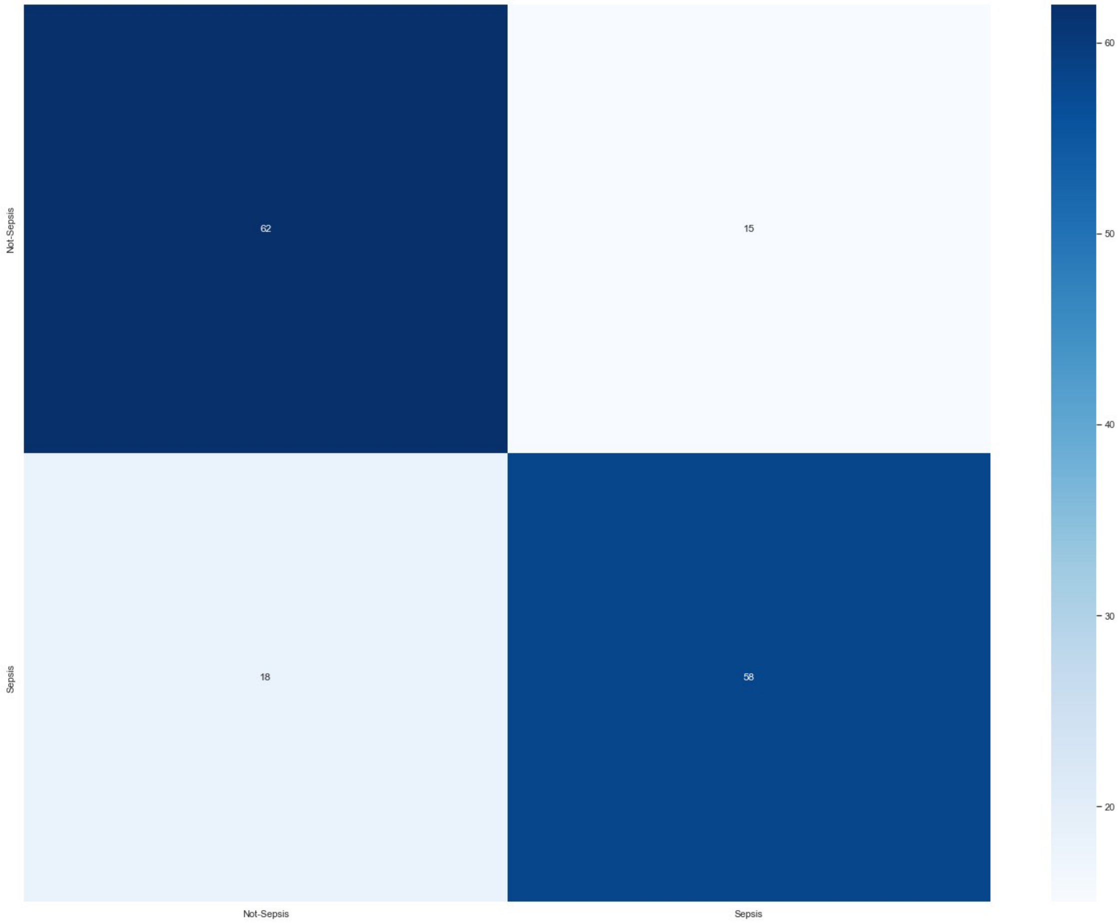
```
0.7937225109956018
{'criterion': 'entropy', 'max_depth': 52, 'min_samples_leaf': 2,
'splitter': 'best'}
DecisionTreeClassifier(ccp_alpha=0.00899989, criterion='entropy',
max_depth=52,
                       min_samples_leaf=2, random_state=42)
```

```
model = DecisionTreeClassifier(ccp_alpha=0.00899989, criterion='gini',
max_depth=52,
                               min_samples_leaf=2, random_state=42)
model.fit(X_train,y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
plot_confusionmatrix(y_train_pred,y_train,dom='Train')
plot_confusionmatrix(y_test_pred,y_test,dom='Test')
```

```
Train score 0.8036006546644845
Test score 0.7843137254901961
Train Confusion matrix
```



```
Test Confusion matrix
```



------------> OBSERVATION

No overfit anymore!

5.2.2.c Hypertuning parameter:

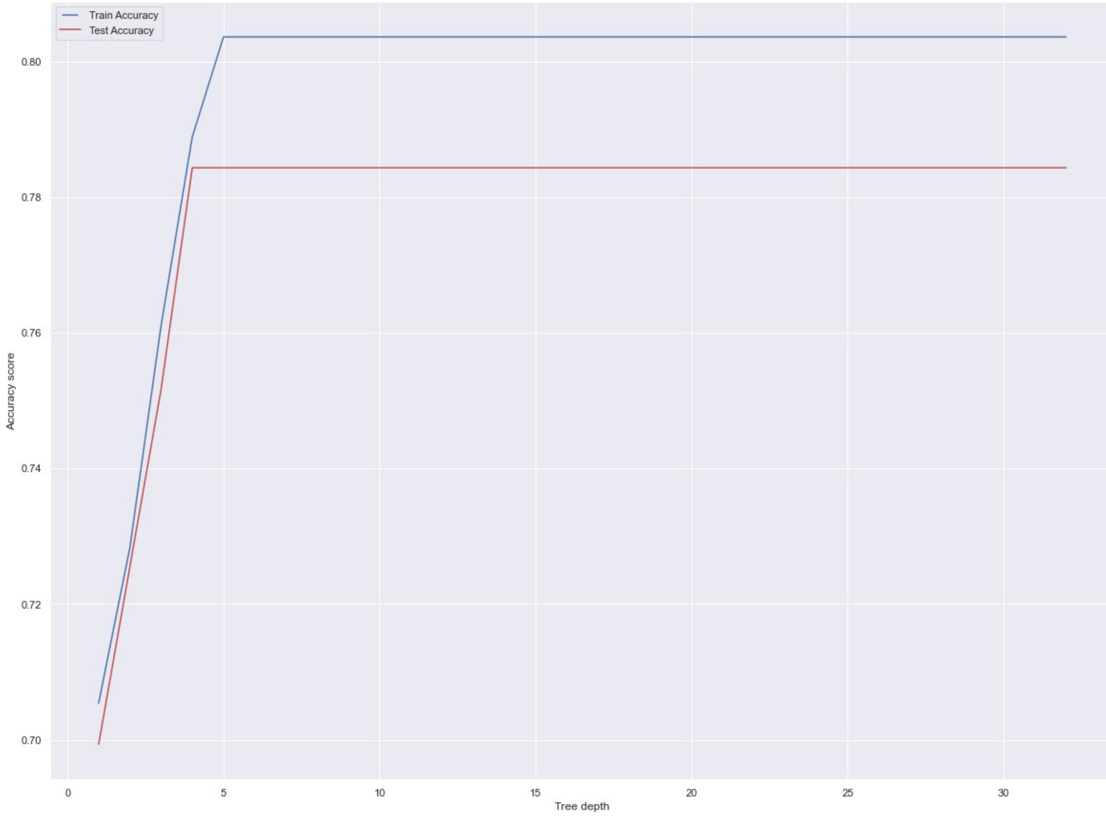
I actually use both of grid search for tuning, nevertheless, I also desire to hypertuning manually since the grid search just simply search for the higest accuracy score and do not concern for underfitting or overfitting. ***

Tuning max_depth:

```
max_depths = np.linspace(1, 32, 32, endpoint=True) # List of values
for tuning
train_results = [] # Store train accuracy results
test_results = []  # Store test accuracy results
for max_depth in max_depths:
```

```
    decisionTree = tree.DecisionTreeClassifier(ccp_alpha=0.00899989,
max_depth=max_depth, random_state=42,
                                              criterion='gini',
min_samples_leaf=2, splitter='best',
                                             )
    decisionTree.fit(X_train, y_train)
    train_pred = decisionTree.predict(X_train)
    train_acc = accuracy_score (y_train, train_pred)
    # Add accuracy score to previous train results
    train_results.append(train_acc)

    #test
    test_pred = decisionTree.predict(X_test)
    test_acc = accuracy_score (y_test, test_pred)
    # Add auc score to previous test results
    test_results.append(test_acc)

    print('The Training Accuracy for max_depth {}
is:'.format(max_depth), train_acc)
    print('The Test Accuracy for max_depth {} is:'.format(max_depth),
test_acc)

from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, "b", label='Train
Accuracy')
line2, = plt.plot(max_depths, test_results, "r", label='Test
Accuracy')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy score')
plt.xlabel('Tree depth')
plt.show()

The Training Accuracy for max_depth 1.0 is: 0.7054009819967266
The Test Accuracy for max_depth 1.0 is: 0.6993464052287581
The Training Accuracy for max_depth 2.0 is: 0.7283142389525368
The Test Accuracy for max_depth 2.0 is: 0.7254901960784313
The Training Accuracy for max_depth 3.0 is: 0.7610474631751227
The Test Accuracy for max_depth 3.0 is: 0.7516339869281046
The Training Accuracy for max_depth 4.0 is: 0.7888707037643208
The Test Accuracy for max_depth 4.0 is: 0.7843137254901961
The Training Accuracy for max_depth 5.0 is: 0.8036006546644845
The Test Accuracy for max_depth 5.0 is: 0.7843137254901961
The Training Accuracy for max_depth 6.0 is: 0.8036006546644845
The Test Accuracy for max_depth 6.0 is: 0.7843137254901961
The Training Accuracy for max_depth 7.0 is: 0.8036006546644845
The Test Accuracy for max_depth 7.0 is: 0.7843137254901961
The Training Accuracy for max_depth 8.0 is: 0.8036006546644845
The Test Accuracy for max_depth 8.0 is: 0.7843137254901961
The Training Accuracy for max_depth 9.0 is: 0.8036006546644845
The Test Accuracy for max_depth 9.0 is: 0.7843137254901961
```

```
The Training Accuracy for max_depth 10.0 is: 0.8036006546644845
The Test Accuracy for max_depth 10.0 is: 0.7843137254901961
The Training Accuracy for max_depth 11.0 is: 0.8036006546644845
The Test Accuracy for max_depth 11.0 is: 0.7843137254901961
The Training Accuracy for max_depth 12.0 is: 0.8036006546644845
The Test Accuracy for max_depth 12.0 is: 0.7843137254901961
The Training Accuracy for max_depth 13.0 is: 0.8036006546644845
The Test Accuracy for max_depth 13.0 is: 0.7843137254901961
The Training Accuracy for max_depth 14.0 is: 0.8036006546644845
The Test Accuracy for max_depth 14.0 is: 0.7843137254901961
The Training Accuracy for max_depth 15.0 is: 0.8036006546644845
The Test Accuracy for max_depth 15.0 is: 0.7843137254901961
The Training Accuracy for max_depth 16.0 is: 0.8036006546644845
The Test Accuracy for max_depth 16.0 is: 0.7843137254901961
The Training Accuracy for max_depth 17.0 is: 0.8036006546644845
The Test Accuracy for max_depth 17.0 is: 0.7843137254901961
The Training Accuracy for max_depth 18.0 is: 0.8036006546644845
The Test Accuracy for max_depth 18.0 is: 0.7843137254901961
The Training Accuracy for max_depth 19.0 is: 0.8036006546644845
The Test Accuracy for max_depth 19.0 is: 0.7843137254901961
The Training Accuracy for max_depth 20.0 is: 0.8036006546644845
The Test Accuracy for max_depth 20.0 is: 0.7843137254901961
The Training Accuracy for max_depth 21.0 is: 0.8036006546644845
The Test Accuracy for max_depth 21.0 is: 0.7843137254901961
The Training Accuracy for max_depth 22.0 is: 0.8036006546644845
The Test Accuracy for max_depth 22.0 is: 0.7843137254901961
The Training Accuracy for max_depth 23.0 is: 0.8036006546644845
The Test Accuracy for max_depth 23.0 is: 0.7843137254901961
The Training Accuracy for max_depth 24.0 is: 0.8036006546644845
The Test Accuracy for max_depth 24.0 is: 0.7843137254901961
The Training Accuracy for max_depth 25.0 is: 0.8036006546644845
The Test Accuracy for max_depth 25.0 is: 0.7843137254901961
The Training Accuracy for max_depth 26.0 is: 0.8036006546644845
The Test Accuracy for max_depth 26.0 is: 0.7843137254901961
The Training Accuracy for max_depth 27.0 is: 0.8036006546644845
The Test Accuracy for max_depth 27.0 is: 0.7843137254901961
The Training Accuracy for max_depth 28.0 is: 0.8036006546644845
The Test Accuracy for max_depth 28.0 is: 0.7843137254901961
The Training Accuracy for max_depth 29.0 is: 0.8036006546644845
The Test Accuracy for max_depth 29.0 is: 0.7843137254901961
The Training Accuracy for max_depth 30.0 is: 0.8036006546644845
The Test Accuracy for max_depth 30.0 is: 0.7843137254901961
The Training Accuracy for max_depth 31.0 is: 0.8036006546644845
The Test Accuracy for max_depth 31.0 is: 0.7843137254901961
The Training Accuracy for max_depth 32.0 is: 0.8036006546644845
The Test Accuracy for max_depth 32.0 is: 0.7843137254901961
```



------------> OBSERVATION

According to this plot, the max_depth=5 can reach the highest accuracy

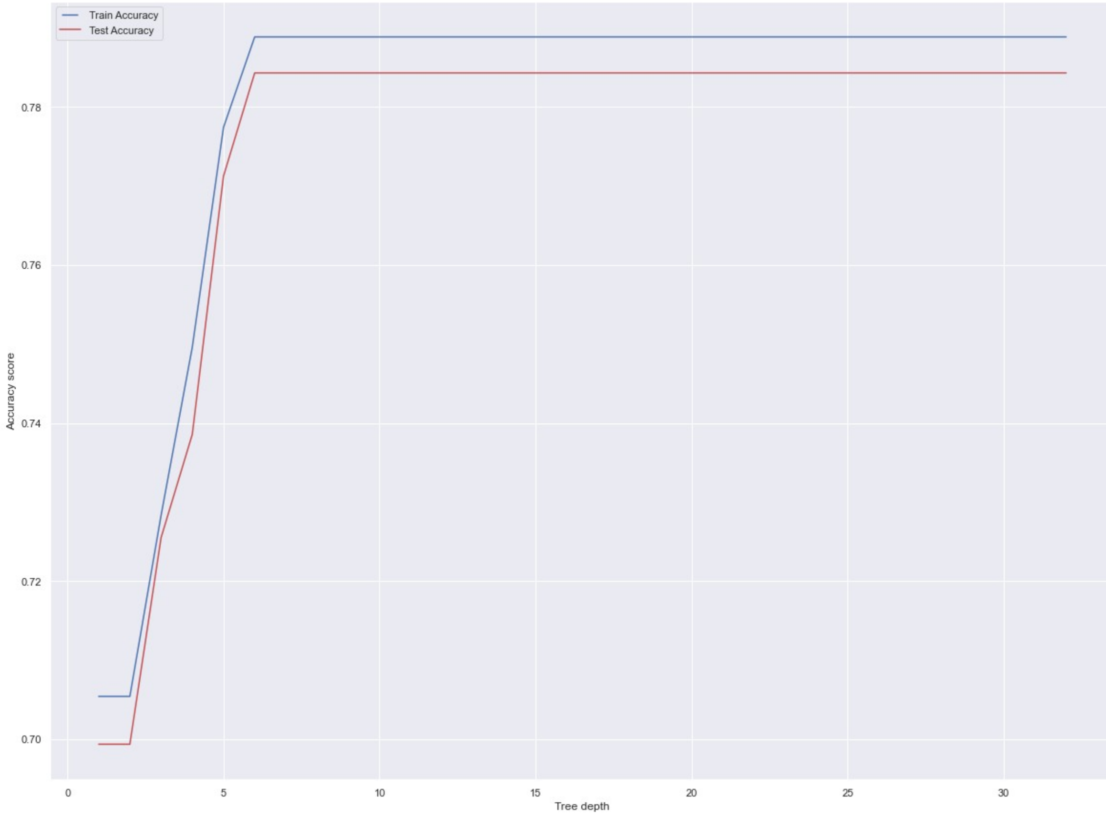
Tuning max_leaf_nodes:

```
train_results = [] # Store train accuracy results
test_results = []  # Store test accuracy results
for max_leaf_nodes in range(2,34):
   decisionTree = tree.DecisionTreeClassifier(ccp_alpha=0.00899989, max_depth=4, random_state=42,
                                              criterion='gini', min_samples_leaf=2, splitter='best',
                                              max_leaf_nodes=max_leaf_nodes
                                             )
   decisionTree.fit(X_train, y_train)
   train_pred = decisionTree.predict(X_train)
   train_acc = accuracy_score (y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(train_acc)

   #test
   test_pred = decisionTree.predict(X_test)
   test_acc = accuracy_score (y_test, test_pred)
```

```
    # Add auc score to previous test results
    test_results.append(test_acc)

    print('The Training Accuracy for max_leaf_nodes {}
is:'.format(max_leaf_nodes), train_acc)
    print('The Test Accuracy for max_leaf_nodes {}
is:'.format(max_leaf_nodes), test_acc)

from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, "b", label='Train
Accuracy')
line2, = plt.plot(max_depths, test_results, "r", label='Test
Accuracy')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy score')
plt.xlabel('Tree depth')
plt.show()
```

```
The Training Accuracy for max_leaf_nodes 2 is: 0.7054009819967266
The Test Accuracy for max_leaf_nodes 2 is: 0.6993464052287581
The Training Accuracy for max_leaf_nodes 3 is: 0.7054009819967266
The Test Accuracy for max_leaf_nodes 3 is: 0.6993464052287581
The Training Accuracy for max_leaf_nodes 4 is: 0.7283142389525368
The Test Accuracy for max_leaf_nodes 4 is: 0.7254901960784313
The Training Accuracy for max_leaf_nodes 5 is: 0.7495908346972177
The Test Accuracy for max_leaf_nodes 5 is: 0.738562091503268
The Training Accuracy for max_leaf_nodes 6 is: 0.7774140752864157
The Test Accuracy for max_leaf_nodes 6 is: 0.7712418300653595
The Training Accuracy for max_leaf_nodes 7 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 7 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 8 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 8 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 9 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 9 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 10 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 10 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 11 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 11 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 12 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 12 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 13 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 13 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 14 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 14 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 15 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 15 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 16 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 16 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 17 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 17 is: 0.7843137254901961
```

```
The Training Accuracy for max_leaf_nodes 18 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 18 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 19 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 19 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 20 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 20 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 21 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 21 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 22 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 22 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 23 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 23 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 24 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 24 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 25 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 25 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 26 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 26 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 27 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 27 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 28 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 28 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 29 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 29 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 30 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 30 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 31 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 31 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 32 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 32 is: 0.7843137254901961
The Training Accuracy for max_leaf_nodes 33 is: 0.7888707037643208
The Test Accuracy for max_leaf_nodes 33 is: 0.7843137254901961
```



------------> OBSERVATION

According to this plot, the max_leaf_nodes = 8 can reach the highest accuracy for train and test

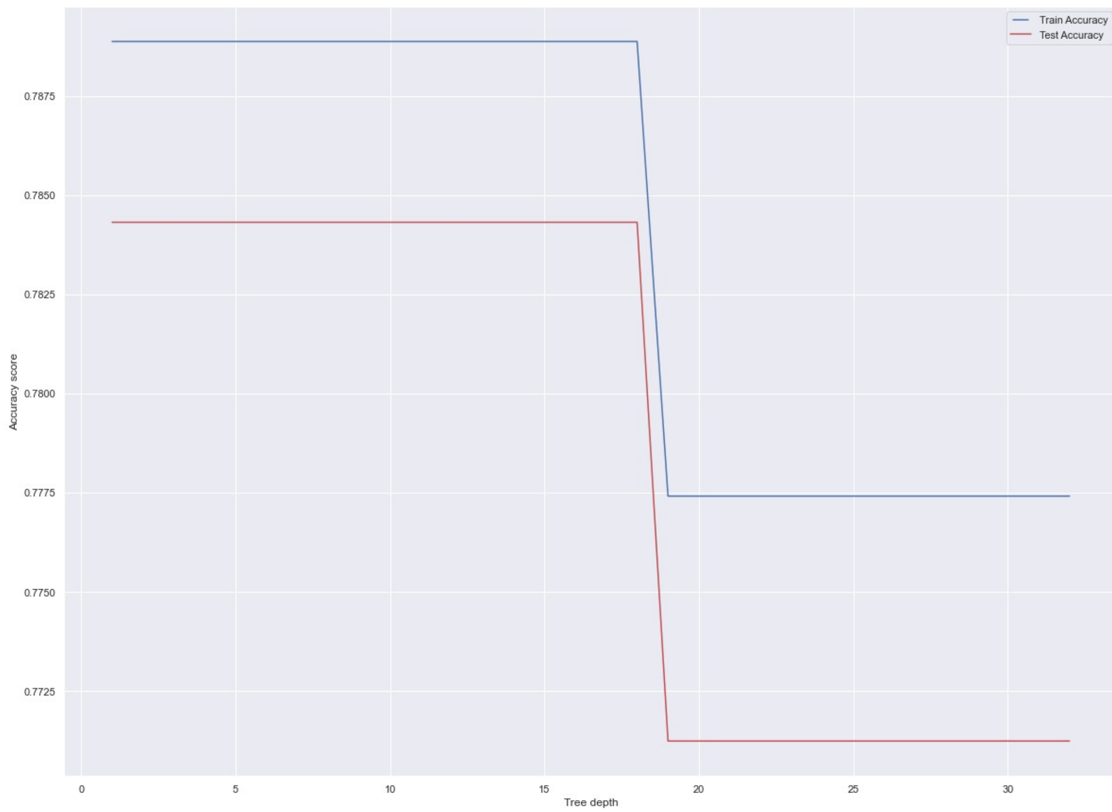
Tuning min_samples_leaf:

```
train_results = [] # Store train accuracy results
test_results = []  # Store test accuracy results
for min_samples_leaf in range(2,34):
   decisionTree = tree.DecisionTreeClassifier(ccp_alpha=0.00899989,
max_depth=4, random_state=42,
                                             criterion='gini',
min_samples_leaf=min_samples_leaf, splitter='best',
                                             max_leaf_nodes=8
                                            )
   decisionTree.fit(X_train, y_train)
   train_pred = decisionTree.predict(X_train)
   train_acc = accuracy_score (y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(train_acc)

   #test
   test_pred = decisionTree.predict(X_test)
   test_acc = accuracy_score (y_test, test_pred)
```

```
    # Add auc score to previous test results
    test_results.append(test_acc)

    print('The Training Accuracy for min_samples_leaf {} is:'.format(min_samples_leaf), train_acc)
    print('The Test Accuracy for min_samples_leaf {} is:'.format(min_samples_leaf), test_acc)

from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, "b", label='Train Accuracy')
line2, = plt.plot(max_depths, test_results, "r", label='Test Accuracy')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy score')
plt.xlabel('Tree depth')
plt.show()
```

```
The Training Accuracy for min_samples_leaf 2 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 2 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 3 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 3 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 4 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 4 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 5 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 5 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 6 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 6 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 7 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 7 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 8 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 8 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 9 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 9 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 10 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 10 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 11 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 11 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 12 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 12 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 13 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 13 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 14 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 14 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 15 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 15 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 16 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 16 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 17 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 17 is: 0.7843137254901961
```

```
The Training Accuracy for min_samples_leaf 18 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 18 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 19 is: 0.7888707037643208
The Test Accuracy for min_samples_leaf 19 is: 0.7843137254901961
The Training Accuracy for min_samples_leaf 20 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 20 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 21 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 21 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 22 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 22 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 23 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 23 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 24 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 24 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 25 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 25 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 26 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 26 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 27 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 27 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 28 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 28 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 29 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 29 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 30 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 30 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 31 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 31 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 32 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 32 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 33 is: 0.7774140752864157
The Test Accuracy for min_samples_leaf 33 is: 0.7712418300653595
```



------------> OBSERVATION

According to this plot, the min_samples_leaf = 2 can reach the highest accuracy for train and test.

Retrain

```
model = DecisionTreeClassifier(ccp_alpha=0.00899989, max_depth=4,
random_state=42,
                                              criterion='gini',
min_samples_leaf=2, splitter='best',
                                              max_leaf_nodes=8
                                             )
model.fit(X_train,y_train)

## Use the "X_test" to predict the model outcome -> evaluate the
outcome of model.
y_pred = model.predict(X_test)
## Use the "X_train" to predict the model outcome -> evaluate the
outcome of model in train set.
y_pred_train = model.predict(X_train)
```

```python
## Check if the model is overfitting or not?
print("Test F1 Score:" + str(f1_score(y_test, y_pred)))
print("Test Accuracy Score:" + str(accuracy_score (y_test, y_pred)))
print("--------------------------------------------------------")
print("Train F1 Score:" + str(f1_score (y_train, y_pred_train)))
print("Train Accuracy Score:" + str(accuracy_score (y_train,
y_pred_train)))
```

```
Test F1 Score:0.7755102040816327
Test Accuracy Score:0.7843137254901961
--------------------------------------------------------
Train F1 Score:0.8000000000000002
Train Accuracy Score:0.7888707037643208
```

```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.79        80
           1       0.77      0.78      0.78        73

    accuracy                           0.78       153
   macro avg       0.78      0.78      0.78       153
weighted avg       0.78      0.78      0.78       153
```

```python
false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)

print(roc_auc)
```

```
0.7841609589041095
```

```python
labels_df = pd.DataFrame(patient_ID, y_pred ,columns=["ID", "Sepsis"])
labels_df.to_csv(r'Sepsis_prediction.csv',index = False)
```

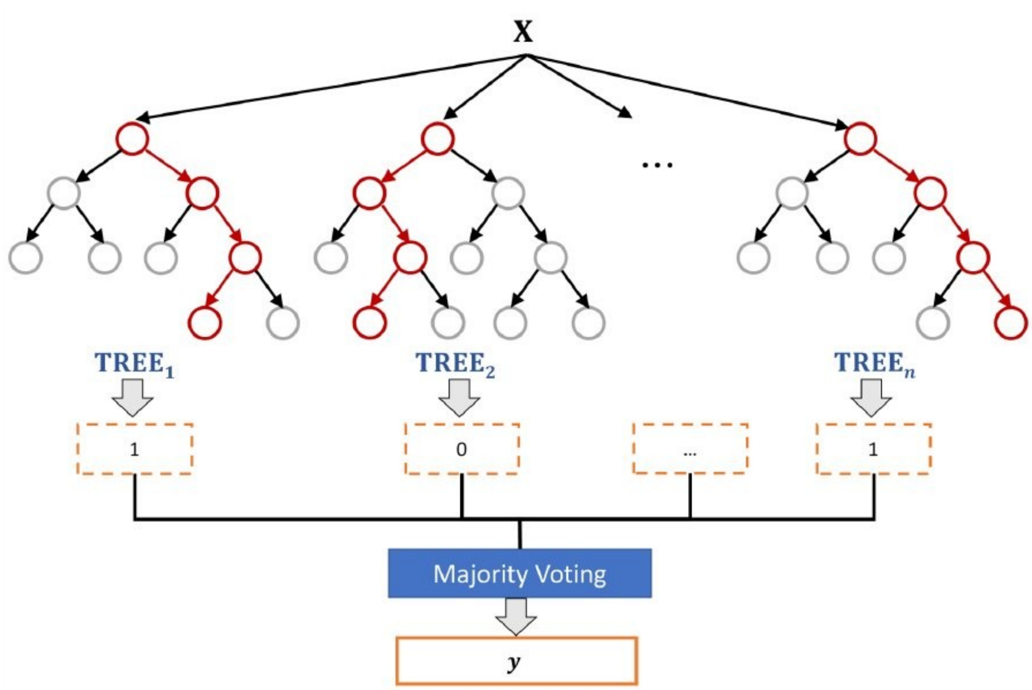
5.2.3 Conclusion

- In order to valuate the Decision Tree Model, I use some methods including the accuracy, AUC score, precision, recall and f1-score.
- Since the accuracy score only concentrates on the true and correct values, nevertheless, this is the medical problem so it also requires the wrong rate to be minimum as much as possible. Since then, the precision, recall and f1-score are good methods.
- In this model
- Accuracy: 0.80

- AUC score: 0.76 (The area under the ROC curve (AUC) results were considered excellent for AUC values between 0.9-1, good for AUC values between 0.8-0.9, fair for AUC values between 0.7-0.8, poor for AUC values between 0.6-0.7 and failed for AUC values between 0.5-0.6.) [6]
- Precision: 0.8
- Recall: 0.81
- F1-score: 0.80

5.3 Random Forest:

Random Forest:



- It is a supervised learning strategy that can be used for both classificaton and regression.
- It an ensemble method (collection of many decision tree).

Why I chose Random Forest:

- It works best with classification.

- It do not need to pruning like Decision Tree
- It has a lot of Decision Trees in order to generate the final output so it is more effective than Decision Tree.

5.3.1 Training model:

```
from sklearn.ensemble import RandomForestClassifier
# Initialise the model
random_forest = RandomForestClassifier(random_state=44)

# Fit the model with train set
random_forest.fit(X_train,y_train)

## Use the "X_train" to predict the model outcome -> evaluate the
outcome of model in train set.
y_train_predicted=random_forest.predict(X_train)

## Use the "X_test" to predict the model outcome -> evaluate the
outcome of model.
y_test_predicted=random_forest.predict(X_test)
```

Overfitting

```
## Check if the model is overfitting or not?
print("Test F1 Score:" + str(f1_score(y_test, y_pred)))
print("Test Accuracy Score:" + str(accuracy_score (y_test,
y_test_predicted)))
print("---------------------------------------------------------")
print("Train F1 Score:" + str(f1_score (y_train, y_pred_train)))
print("Train Accuracy Score:" + str(accuracy_score (y_train,
y_train_predicted)))
```

```
Test F1 Score:0.7755102040816327
Test Accuracy Score:0.8366013071895425
--------------------------------------------------------
Train F1 Score:0.8000000000000002
Train Accuracy Score:1.0
```

------------> OBSERVATION

According to the result, there is an extreme overfitting.

5.3.2 Hyperparameter tuning:

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold, 
StratifiedShuffleSplit
from sklearn.ensemble import RandomForestClassifier
n_estimators = [140,145,150,155,160]
max_depth = range(1,10)
criterions = ['gini', 'entropy']
cv = StratifiedShuffleSplit(n_splits=10, test_size=.30, 
random_state=44)


parameters = {'n_estimators':n_estimators,
              'max_depth':max_depth,
              'criterion': criterions

        }
grid = 
GridSearchCV(estimator=RandomForestClassifier(max_features='auto'),
                                 param_grid=parameters,
                                 cv=cv,
                                 n_jobs = -1)
grid.fit(X,y)
```

```
GridSearchCV(cv=StratifiedShuffleSplit(n_splits=10, random_state=44, 
test_size=0.3,
            train_size=None),
             estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 10),
                         'n_estimators': [140, 145, 150, 155, 160]})
```

```
print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
```

```
0.8478260869565217
{'criterion': 'gini', 'max_depth': 9, 'n_estimators': 145}
RandomForestClassifier(max_depth=9, n_estimators=145)
```

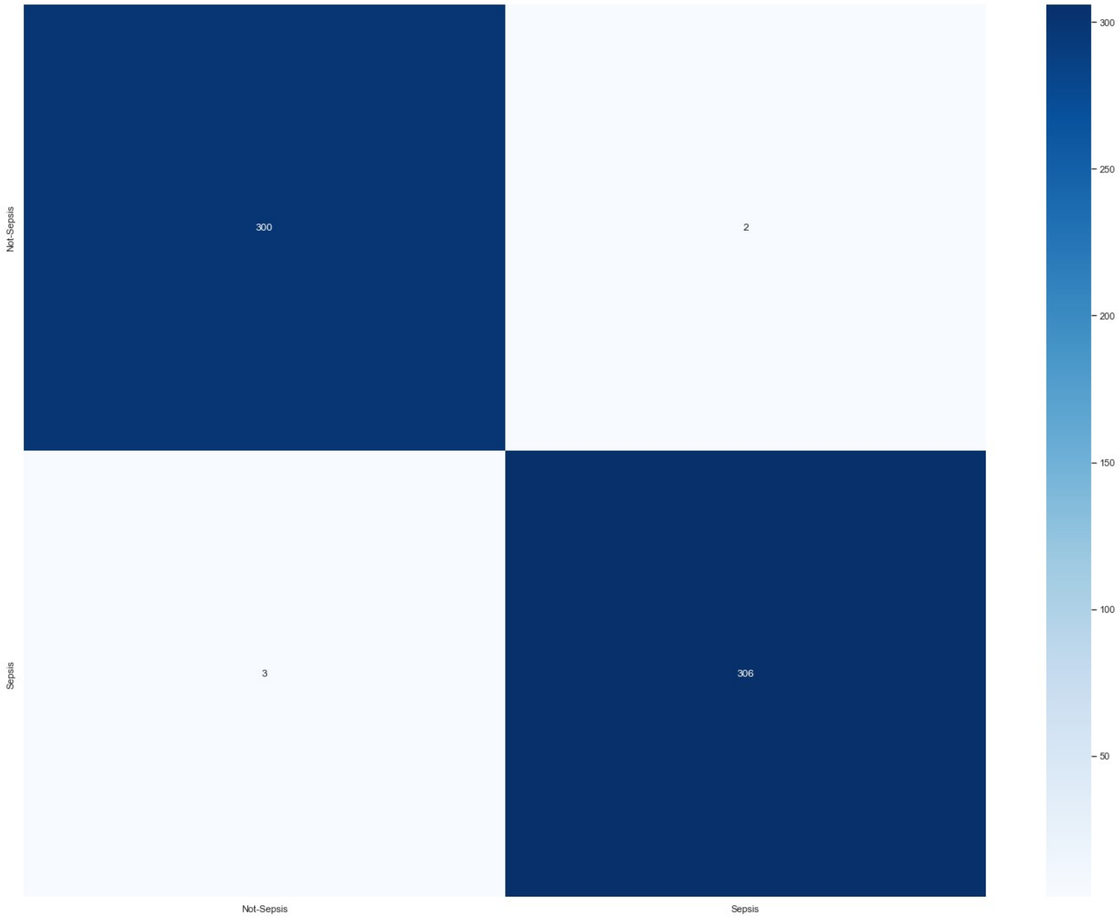
```
rf_grid = grid.best_estimator_
rf_grid.score(X,y)
```

```
0.9869109947643979
```

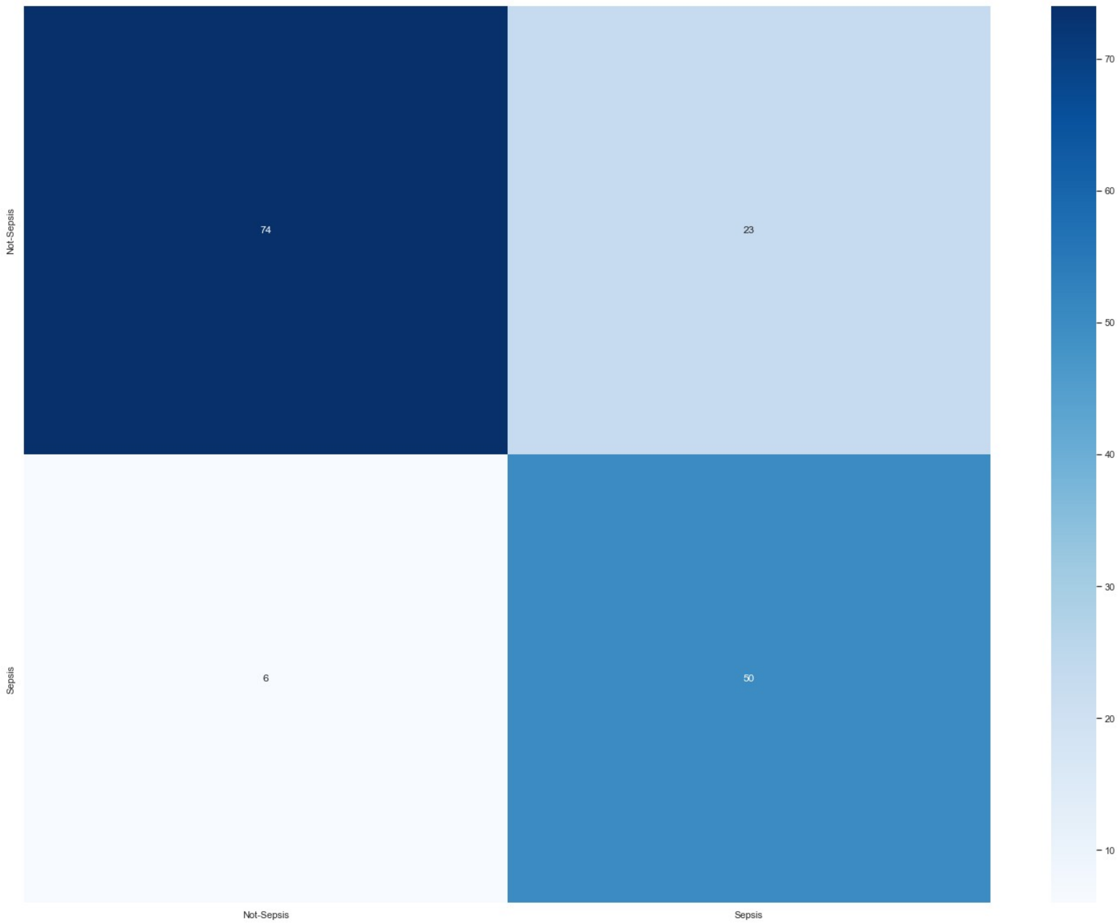
```
model = RandomForestClassifier(max_depth=9, n_estimators=155, 
random_state=44, criterion='gini')
model.fit(X_train,y_train)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print(f'Train score {accuracy_score(y_train_pred,y_train)}')
print(f'Test score {accuracy_score(y_test_pred,y_test)}')
```

```
plot_confusionmatrix(y_train_pred,y_train,dom='Train')
plot_confusionmatrix(y_test_pred,y_test,dom='Test')
```

```
Train score 0.9918166939443536
Test score 0.8104575163398693
Train Confusion matrix
```




```
Test Confusion matrix
```



------------> OBSERVATION

It is still overfit. I will try to manually tuning the parameters which are `max_depth`, `n_estimators`, `max_features`, `min_samples_split`, `min_samples_leaf`, `bootstrap`.

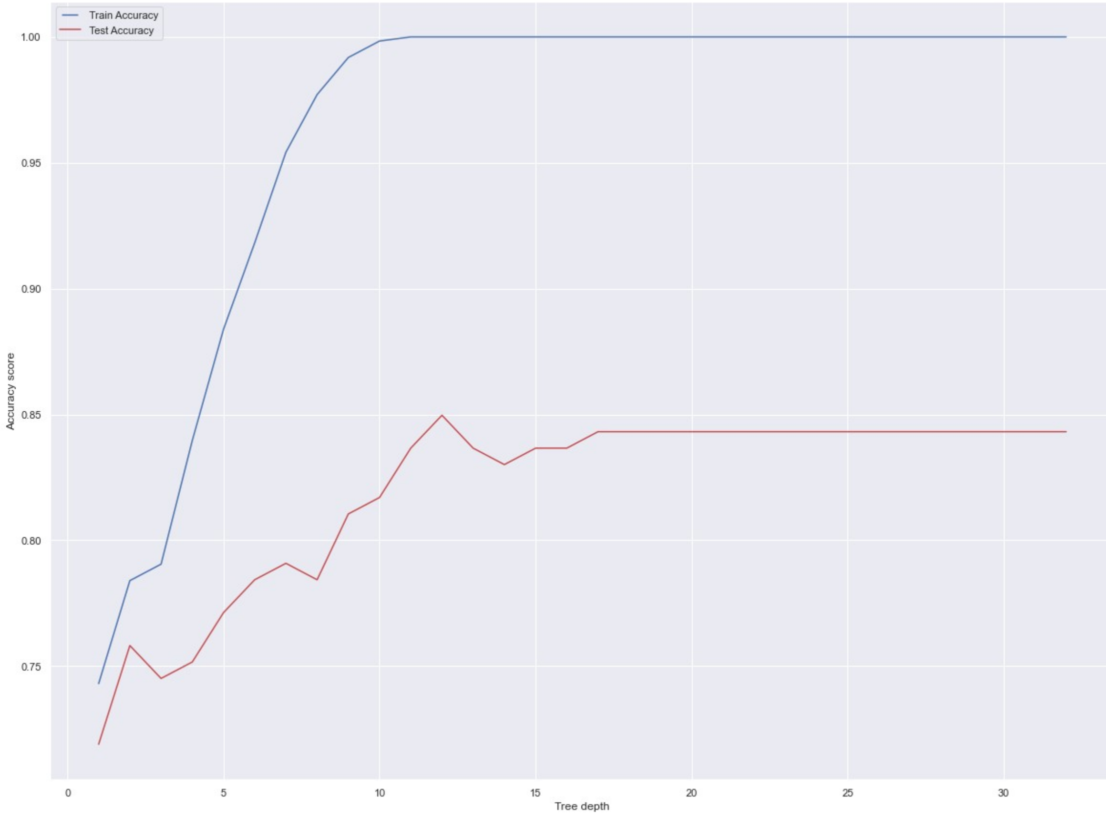
Tuning max_depth:

```
max_depths = np.linspace(1, 32, 32, endpoint=True) # List of values
for tuning
train_results = [] # Store train accuracy results
test_results = []  # Store test accuracy results
for max_depth in max_depths:
   decisionTree = RandomForestClassifier(max_depth=max_depth,
n_estimators=155, random_state=44)
   decisionTree.fit(X_train, y_train)
   train_pred = decisionTree.predict(X_train)
   train_acc = accuracy_score (y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(train_acc)
```

```
    #test
    test_pred = decisionTree.predict(X_test)
    test_acc = accuracy_score (y_test, test_pred)
    # Add auc score to previous test results
    test_results.append(test_acc)

    print('The Training Accuracy for max_depth {}
is:'.format(max_depth), train_acc)
    print('The Test Accuracy for max_depth {} is:'.format(max_depth),
test_acc)

from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, "b", label='Train
Accuracy')
line2, = plt.plot(max_depths, test_results, "r", label='Test
Accuracy')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy score')
plt.xlabel('Tree depth')
plt.show()
```

```
The Training Accuracy for max_depth 1.0 is: 0.7430441898527005
The Test Accuracy for max_depth 1.0 is: 0.7189542483660131
The Training Accuracy for max_depth 2.0 is: 0.7839607201309329
The Test Accuracy for max_depth 2.0 is: 0.7581699346405228
The Training Accuracy for max_depth 3.0 is: 0.79050736497545
The Test Accuracy for max_depth 3.0 is: 0.7450980392156863
The Training Accuracy for max_depth 4.0 is: 0.839607201309329
The Test Accuracy for max_depth 4.0 is: 0.7516339869281046
The Training Accuracy for max_depth 5.0 is: 0.88379705400982
The Test Accuracy for max_depth 5.0 is: 0.7712418300653595
The Training Accuracy for max_depth 6.0 is: 0.9181669394435352
The Test Accuracy for max_depth 6.0 is: 0.7843137254901961
The Training Accuracy for max_depth 7.0 is: 0.9541734860883797
The Test Accuracy for max_depth 7.0 is: 0.7908496732026143
The Training Accuracy for max_depth 8.0 is: 0.9770867430441899
The Test Accuracy for max_depth 8.0 is: 0.7843137254901961
The Training Accuracy for max_depth 9.0 is: 0.9918166939443536
The Test Accuracy for max_depth 9.0 is: 0.8104575163398693
The Training Accuracy for max_depth 10.0 is: 0.9983633387888707
The Test Accuracy for max_depth 10.0 is: 0.8169934640522876
The Training Accuracy for max_depth 11.0 is: 1.0
The Test Accuracy for max_depth 11.0 is: 0.8366013071895425
The Training Accuracy for max_depth 12.0 is: 1.0
The Test Accuracy for max_depth 12.0 is: 0.8496732026143791
The Training Accuracy for max_depth 13.0 is: 1.0
The Test Accuracy for max_depth 13.0 is: 0.8366013071895425
The Training Accuracy for max_depth 14.0 is: 1.0
The Test Accuracy for max_depth 14.0 is: 0.8300653594771242
The Training Accuracy for max_depth 15.0 is: 1.0
```

```
The Test Accuracy for max_depth 15.0 is: 0.8366013071895425
The Training Accuracy for max_depth 16.0 is: 1.0
The Test Accuracy for max_depth 16.0 is: 0.8366013071895425
The Training Accuracy for max_depth 17.0 is: 1.0
The Test Accuracy for max_depth 17.0 is: 0.8431372549019608
The Training Accuracy for max_depth 18.0 is: 1.0
The Test Accuracy for max_depth 18.0 is: 0.8431372549019608
The Training Accuracy for max_depth 19.0 is: 1.0
The Test Accuracy for max_depth 19.0 is: 0.8431372549019608
The Training Accuracy for max_depth 20.0 is: 1.0
The Test Accuracy for max_depth 20.0 is: 0.8431372549019608
The Training Accuracy for max_depth 21.0 is: 1.0
The Test Accuracy for max_depth 21.0 is: 0.8431372549019608
The Training Accuracy for max_depth 22.0 is: 1.0
The Test Accuracy for max_depth 22.0 is: 0.8431372549019608
The Training Accuracy for max_depth 23.0 is: 1.0
The Test Accuracy for max_depth 23.0 is: 0.8431372549019608
The Training Accuracy for max_depth 24.0 is: 1.0
The Test Accuracy for max_depth 24.0 is: 0.8431372549019608
The Training Accuracy for max_depth 25.0 is: 1.0
The Test Accuracy for max_depth 25.0 is: 0.8431372549019608
The Training Accuracy for max_depth 26.0 is: 1.0
The Test Accuracy for max_depth 26.0 is: 0.8431372549019608
The Training Accuracy for max_depth 27.0 is: 1.0
The Test Accuracy for max_depth 27.0 is: 0.8431372549019608
The Training Accuracy for max_depth 28.0 is: 1.0
The Test Accuracy for max_depth 28.0 is: 0.8431372549019608
The Training Accuracy for max_depth 29.0 is: 1.0
The Test Accuracy for max_depth 29.0 is: 0.8431372549019608
The Training Accuracy for max_depth 30.0 is: 1.0
The Test Accuracy for max_depth 30.0 is: 0.8431372549019608
The Training Accuracy for max_depth 31.0 is: 1.0
The Test Accuracy for max_depth 31.0 is: 0.8431372549019608
The Training Accuracy for max_depth 32.0 is: 1.0
The Test Accuracy for max_depth 32.0 is: 0.8431372549019608
```



------------> OBSERVATION

Best accuracy for the max_depth is 2

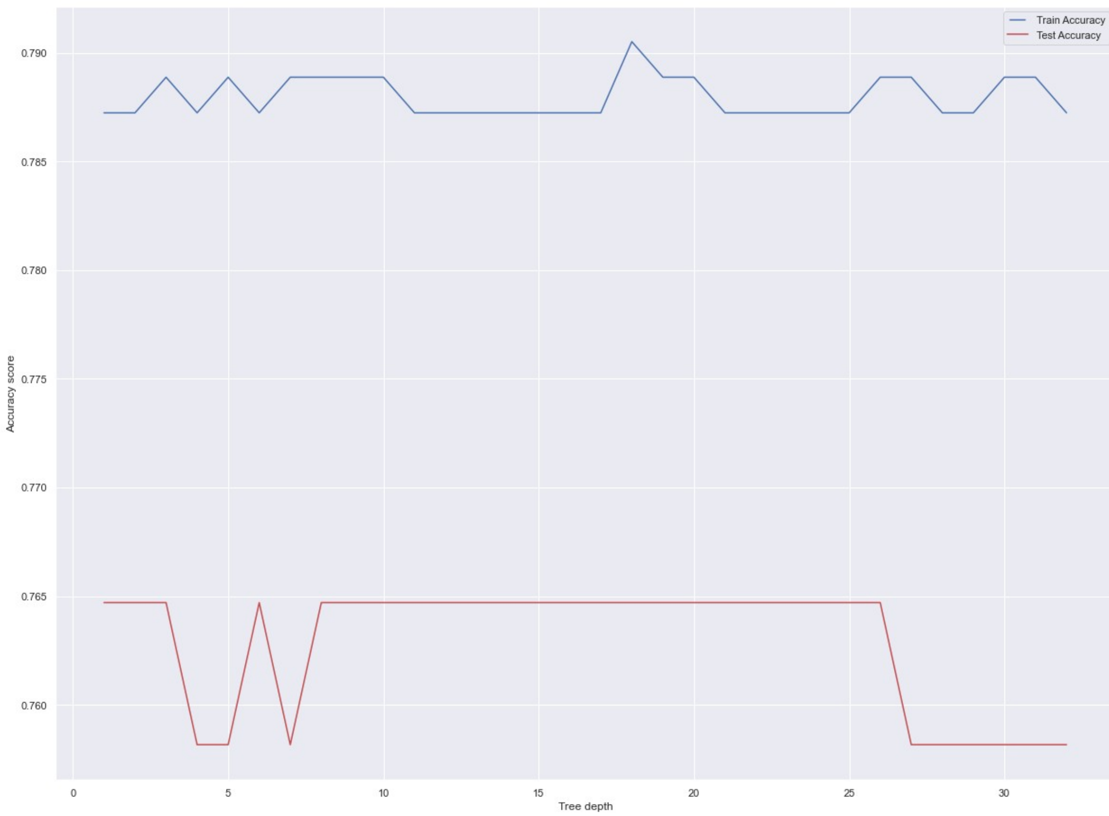
Tunning n_estimators:

```
train_results = [] # Store train accuracy results
test_results = []  # Store test accuracy results

for n_estimator in range(140,172):
   decisionTree = RandomForestClassifier(max_depth=2, 
n_estimators=n_estimator, random_state=42, criterion='gini')
   decisionTree.fit(X_train, y_train)
   train_pred = decisionTree.predict(X_train)
   train_acc = accuracy_score (y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(train_acc)

   #test
   test_pred = decisionTree.predict(X_test)
   test_acc = accuracy_score (y_test, test_pred)
   # Add auc score to previous test results
   test_results.append(test_acc)

   print('The Training Accuracy for n_estimator {}
```

```
is:'.format(n_estimator), train_acc)
   print('The Test Accuracy for n_estimator {}
is:'.format(n_estimator), test_acc)

from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, "b", label='Train
Accuracy')
line2, = plt.plot(max_depths, test_results, "r", label='Test
Accuracy')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy score')
plt.xlabel('Tree depth')
plt.show()
```

```
The Training Accuracy for n_estimator 140 is: 0.7872340425531915
The Test Accuracy for n_estimator 140 is: 0.7647058823529411
The Training Accuracy for n_estimator 141 is: 0.7872340425531915
The Test Accuracy for n_estimator 141 is: 0.7647058823529411
The Training Accuracy for n_estimator 142 is: 0.7888707037643208
The Test Accuracy for n_estimator 142 is: 0.7647058823529411
The Training Accuracy for n_estimator 143 is: 0.7872340425531915
The Test Accuracy for n_estimator 143 is: 0.7581699346405228
The Training Accuracy for n_estimator 144 is: 0.7888707037643208
The Test Accuracy for n_estimator 144 is: 0.7581699346405228
The Training Accuracy for n_estimator 145 is: 0.7872340425531915
The Test Accuracy for n_estimator 145 is: 0.7647058823529411
The Training Accuracy for n_estimator 146 is: 0.7888707037643208
The Test Accuracy for n_estimator 146 is: 0.7581699346405228
The Training Accuracy for n_estimator 147 is: 0.7888707037643208
The Test Accuracy for n_estimator 147 is: 0.7647058823529411
The Training Accuracy for n_estimator 148 is: 0.7888707037643208
The Test Accuracy for n_estimator 148 is: 0.7647058823529411
The Training Accuracy for n_estimator 149 is: 0.7888707037643208
The Test Accuracy for n_estimator 149 is: 0.7647058823529411
The Training Accuracy for n_estimator 150 is: 0.7872340425531915
The Test Accuracy for n_estimator 150 is: 0.7647058823529411
The Training Accuracy for n_estimator 151 is: 0.7872340425531915
The Test Accuracy for n_estimator 151 is: 0.7647058823529411
The Training Accuracy for n_estimator 152 is: 0.7872340425531915
The Test Accuracy for n_estimator 152 is: 0.7647058823529411
The Training Accuracy for n_estimator 153 is: 0.7872340425531915
The Test Accuracy for n_estimator 153 is: 0.7647058823529411
The Training Accuracy for n_estimator 154 is: 0.7872340425531915
The Test Accuracy for n_estimator 154 is: 0.7647058823529411
The Training Accuracy for n_estimator 155 is: 0.7872340425531915
The Test Accuracy for n_estimator 155 is: 0.7647058823529411
The Training Accuracy for n_estimator 156 is: 0.7872340425531915
The Test Accuracy for n_estimator 156 is: 0.7647058823529411
The Training Accuracy for n_estimator 157 is: 0.79050736497545
The Test Accuracy for n_estimator 157 is: 0.7647058823529411
```

```
The Training Accuracy for n_estimator 158 is: 0.7888707037643208
The Test Accuracy for n_estimator 158 is: 0.7647058823529411
The Training Accuracy for n_estimator 159 is: 0.7888707037643208
The Test Accuracy for n_estimator 159 is: 0.7647058823529411
The Training Accuracy for n_estimator 160 is: 0.7872340425531915
The Test Accuracy for n_estimator 160 is: 0.7647058823529411
The Training Accuracy for n_estimator 161 is: 0.7872340425531915
The Test Accuracy for n_estimator 161 is: 0.7647058823529411
The Training Accuracy for n_estimator 162 is: 0.7872340425531915
The Test Accuracy for n_estimator 162 is: 0.7647058823529411
The Training Accuracy for n_estimator 163 is: 0.7872340425531915
The Test Accuracy for n_estimator 163 is: 0.7647058823529411
The Training Accuracy for n_estimator 164 is: 0.7872340425531915
The Test Accuracy for n_estimator 164 is: 0.7647058823529411
The Training Accuracy for n_estimator 165 is: 0.7888707037643208
The Test Accuracy for n_estimator 165 is: 0.7647058823529411
The Training Accuracy for n_estimator 166 is: 0.7888707037643208
The Test Accuracy for n_estimator 166 is: 0.7581699346405228
The Training Accuracy for n_estimator 167 is: 0.7872340425531915
The Test Accuracy for n_estimator 167 is: 0.7581699346405228
The Training Accuracy for n_estimator 168 is: 0.7872340425531915
The Test Accuracy for n_estimator 168 is: 0.7581699346405228
The Training Accuracy for n_estimator 169 is: 0.7888707037643208
The Test Accuracy for n_estimator 169 is: 0.7581699346405228
The Training Accuracy for n_estimator 170 is: 0.7888707037643208
The Test Accuracy for n_estimator 170 is: 0.7581699346405228
The Training Accuracy for n_estimator 171 is: 0.7872340425531915
The Test Accuracy for n_estimator 171 is: 0.7581699346405228
```



------------> OBSERVATION

Best n_estimators is 140 max_depth=2

Tunning max_feature:

For max_feature I just take from 1 to 7 features that I currently have

```
for max_feature in range(1,8):
  model = RandomForestClassifier(max_depth=2, n_estimators=140,
random_state=42, criterion='gini',max_features=max_feature)
  model.fit(X_train,y_train)
  print('The Training Accuracy for max_features {}
is:'.format(max_feature), model.score(X_train,y_train))
  print('The Test Accuracy for max_features {}
is:'.format(max_feature), model.score(X_test,y_test))
  print('')
```

```
The Training Accuracy for max_features 1 is: 0.762684124386252
The Test Accuracy for max_features 1 is: 0.7124183006535948

The Training Accuracy for max_features 2 is: 0.7872340425531915
The Test Accuracy for max_features 2 is: 0.7647058823529411
```

```
The Training Accuracy for max_features 3 is: 0.7708674304418985
The Test Accuracy for max_features 3 is: 0.7450980392156863

The Training Accuracy for max_features 4 is: 0.7643207855973814
The Test Accuracy for max_features 4 is: 0.7581699346405228

The Training Accuracy for max_features 5 is: 0.7692307692307693
The Test Accuracy for max_features 5 is: 0.7647058823529411

The Training Accuracy for max_features 6 is: 0.7561374795417348
The Test Accuracy for max_features 6 is: 0.7320261437908496

The Training Accuracy for max_features 7 is: 0.7430441898527005
The Test Accuracy for max_features 7 is: 0.738562091503268
```

------------> OBSERVATION

Best n_estimators is 140 max_depth = 2 max_features=5

Tunning min_samples_leaf:

```
train_results = [] # Store train accuracy results
test_results = []  # Store test accuracy results
for min_samples_leaf in range(1,33):
   decisionTree = RandomForestClassifier(max_depth=2,
n_estimators=140, random_state=42, criterion='gini',
                                        max_features=5,
min_samples_leaf=min_samples_leaf
                                        )
   decisionTree.fit(X_train, y_train)
   train_pred = decisionTree.predict(X_train)
   train_acc = accuracy_score (y_train, train_pred)
   # Add accuracy score to previous train results
   train_results.append(train_acc)

   #test
   test_pred = decisionTree.predict(X_test)
   test_acc = accuracy_score (y_test, test_pred)
   # Add auc score to previous test results
   test_results.append(test_acc)

   print('The Training Accuracy for min_samples_leaf {}
is:'.format(min_samples_leaf), train_acc)
   print('The Test Accuracy for min_samples_leaf {}
is:'.format(min_samples_leaf), test_acc)

from matplotlib.legend_handler import HandlerLine2D
```

```
line1, = plt.plot(max_depths, train_results, "b", label='Train
Accuracy')
line2, = plt.plot(max_depths, test_results, "r", label='Test
Accuracy')
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('Accuracy score')
plt.xlabel('Tree depth')
plt.show()
```

```
The Training Accuracy for min_samples_leaf 1 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 1 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 2 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 2 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 3 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 3 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 4 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 4 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 5 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 5 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 6 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 6 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 7 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 7 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 8 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 8 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 9 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 9 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 10 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 10 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 11 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 11 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 12 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 12 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 13 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 13 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 14 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 14 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 15 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 15 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 16 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 16 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 17 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 17 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 18 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 18 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 19 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 19 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 20 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 20 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 21 is: 0.7692307692307693
```

```
The Test Accuracy for min_samples_leaf 21 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 22 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 22 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 23 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 23 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 24 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 24 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 25 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 25 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 26 is: 0.7675941080196399
The Test Accuracy for min_samples_leaf 26 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 27 is: 0.7675941080196399
The Test Accuracy for min_samples_leaf 27 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 28 is: 0.7675941080196399
The Test Accuracy for min_samples_leaf 28 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 29 is: 0.7675941080196399
The Test Accuracy for min_samples_leaf 29 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 30 is: 0.7692307692307693
The Test Accuracy for min_samples_leaf 30 is: 0.7712418300653595
The Training Accuracy for min_samples_leaf 31 is: 0.7675941080196399
The Test Accuracy for min_samples_leaf 31 is: 0.7647058823529411
The Training Accuracy for min_samples_leaf 32 is: 0.7675941080196399
The Test Accuracy for min_samples_leaf 32 is: 0.7647058823529411
```

------------> OBSERVATION

- Best n_estimators is 2
- max_depth=2
- max_features = 3
- min_samples_leaf = 2

Retrain

```
# Initialise the model
random_forest = RandomForestClassifier(max_depth=2, n_estimators=140,
random_state=44, criterion='gini',max_features=2,
#                                       min_samples_leaf=23

class_weight='balanced_subsample'
                                      )

# Fit the model with train set
random_forest.fit(X_train,y_train)

## Use the "X_train" to predict the model outcome -> evaluate the
outcome of model in train set.
y_train_predicted=random_forest.predict(X_train)

## Use the "X_test" to predict the model outcome -> evaluate the
outcome of model.
y_test_predicted=random_forest.predict(X_test)

## Check if the model is overfitting or not?
print("Test F1 Score:" + str(f1_score(y_test, y_pred)))
print("Test Accuracy Score:" + str(accuracy_score (y_test,
y_test_predicted)))
print("--------------------------------------------------------")
print("Train F1 Score:" + str(f1_score (y_train, y_pred_train)))
print("Train Accuracy Score:" + str(accuracy_score (y_train,
y_train_predicted)))
```

```
Test F1 Score:0.7755102040816327
Test Accuracy Score:0.7516339869281046
--------------------------------------------------------
Train F1 Score:0.8000000000000002
Train Accuracy Score:0.7823240589198036
```

```
print(classification_report(y_test, y_test_predicted))
```

```
              precision    recall  f1-score   support

           0       0.75      0.79      0.77        80
```

```
           1       0.75      0.71      0.73        73

    accuracy                           0.75       153
   macro avg       0.75      0.75      0.75       153
weighted avg       0.75      0.75      0.75       153
```

```
false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
print(roc_auc)
```

```
0.7841609589041095
```

5.2.3 Conclusion

- In order to valuate the Random Forest Model, I use some methods including the accuracy, AUC score, precision, recall and f1-score.
- Since the accuracy score only concentrates on the true and correct values, nevertheless, this is the medical problem so it also requires the wrong rate to be minimum as much as possible. Since then, the precision, recall and f1-score are good methods.
- In this model
- Accuracy: 0.80
- AUC score: 0.76 (The area under the ROC curve (AUC) results were considered excellent for AUC values between 0.9-1, good for AUC values between 0.8-0.9, fair for AUC values between 0.7-0.8, poor for AUC values between 0.6-0.7 and failed for AUC values between 0.5-0.6.) [6]
- Precision: 0.8
- Recall: 0.81
- F1-score: 0.80

6. Conclusion

In short, after the cleaning process, EDA, and t-test, the result is that the higher the statistical medical is, the higher chance for the paitent to have sepsis. Because of that, the outliers were not removed. Moreover, after traning all three models the Decision Tree has the highest score and the recall score is the highest so it is chosen for this project.

7. References

- [1] What Are Platelets and Why Are They Important?
- [2] What's a normal resting heart rate?
- [3] What Is a Potassium Blood Test?
- [3] What Is a Potassium Blood Test?
- [3] What Is a Potassium Blood Test?
- [3] What Is a Potassium Blood Test?
- [4] Assessing Your Weight
- [5] StandardScaler, MinMaxScaler and RobustScaler techniques – ML
- [6] The Relationship of Temporal Resolution to Diagnostic Performance for Dynamic Contrast Enhanced (DCE) MRI of the Breast