APRIL 30, 2022

CAPSTONE PROJECT

ML Foundation Course

Januka Dharmapriya

266

janukad@janashakthi.com

Capstone Project
ML Foundation Course

Explanation of used dataset

The used data set for the project was collected from North East of Andhra Pradesh, India which was published in online portals. This dataset has 894 samples and the Training Dataset consists of 583 samples and the Testing Dataset consists of 311 samples.

The Training Dataset set contains 416 liver patient records and 167 non-liver patient records. The given dataset has only two (2) classes as "Yes" for liver patient and "No" for non-liver patient. There are eleven (11) attributes available in the dataset and 'Gender' and 'Class' attributes are nominal attribute while all the others are numerical attributes. The last attribute is a class field used to divide the dataset into two groups as liver patient or not. This dataset contains several missing (unavailable) attribute values, denoted by "blank" values.

Class distribution of the Training and the Test datasets are as follows.

	Training Dataset	Test Dataset
Yes	416	221
No	167	90
	583	311

Attribute Information:

1. Age - Age of the patient
2. Gender - Gender of the patient
3. TB - Total Bilirubin
4. DB - Direct Bilirubin
5. ALK - Alkaline Phosphatase
6. SGPT - Alamine Aminotransferase
7. SGOT - Aspartate Aminotransferase
8. TP - Total Proteins
9. ALB - Albumin
10. AG_Ratio - Albumin and Globulin Ratio
11. Class - Used to split the data into two sets (labeled by the experts)

Using above dataset, a supervised machine learning model was built to classified the data into the “Yes” or “No” classes.

A. Screenshots with an explanation of the tool used for the above training process and its outputs.

The training process was completed by using the “Keras” from TensorFlow while following steps listed in below.

1. Loading the Training and Testing datasets (Using Python “Pandas” framework).
2. Describing Training dataset.
3. Describing Testing dataset.
4. Visualizations of the Training and Testing datasets.
5. Data pre-processing.
6. Feature selections for the training process.
7. Building the model (Using “Keras” from TensorFlow).
8. Training the model.
9. Evaluating the built-in model.

1. Loading the Training and Testing datasets.

The provided CSV file contained with two datasets for training and testing in two separate sheets. I have used Python “Pandas” module [1] and read_CSV method to read the data from the CSV file.

```
train_df = pd.read_csv('https://raw.githubusercontent.com/JanukaD/Capstone-Project/main/datasets/1/train.csv')
test_df = pd.read_csv('https://raw.githubusercontent.com/JanukaD/Capstone-Project/main/datasets/1/test.csv')
```

Figure 1: "Pandas" module for read the data

2. Describing Training dataset

The training dataset consists of a total of 583 patient records distributed under 11 attributes. Also, the dataset contains several missing attribute values under some columns. From the available attributes, “Gender” and “Class” attributes are nominal attributes, and the rest of the others are numerical attributes. The following figure [2] shows the attributes, their data ty pes, and total available values under each attribute.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        583 non-null    int64
 1   Age       583 non-null    int64
 2   Gender    583 non-null    object
 3   TB        581 non-null    float64
 4   DB        579 non-null    float64
 5   ALK       581 non-null    float64
 6   SGPT      582 non-null    float64
 7   SGOT      582 non-null    float64
 8   TP        581 non-null    float64
 9   ALB       581 non-null    float64
 10  AG_Ratio  582 non-null    float64
 11  Class     583 non-null    object
dtypes: float64(8), int64(2), object(2)
memory usage: 54.8+ KB
```

```
train_df.isnull().sum()

Age         0
Gender      0
TB          2
DB          4
ALK         2
SGPT        1
SGOT        1
TP          2
ALB         2
AG_Ratio    1
Class       0
dtype: int64
```

Figure 2: Attributes and data types - Training dataset

The following figure [3] displays the overall summary of the Training dataset.

	Age	TB	DB	ALK	SGPT	SGOT	TP	ALB	AG_Ratio
count	583.000000	581.000000	579.000000	581.000000	582.000000	582.000000	581.000000	581.000000	582.000000
mean	44.746141	3.307573	1.486701	291.063683	80.780069	110.073883	6.486231	3.138382	0.946306
std	16.189833	6.218411	2.817115	243.206230	182.770380	289.140205	1.085508	0.794631	0.318994
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000	0.300000
25%	33.000000	0.800000	0.200000	176.000000	23.000000	25.000000	5.800000	2.600000	0.700000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000	0.925000
75%	58.000000	2.600000	1.300000	298.000000	60.750000	87.000000	7.200000	3.800000	1.100000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000	2.800000

Figure 3: Summary of the Training dataset

3. Describing Testing dataset

The testing dataset consists of a total of 311 patient records distributed under 11 attributes. Also, the dataset contains several missing attribute values under some columns. From the available attributes, “Gender” and “Class” attributes are nominal attributes, and the rest of the others are numerical attributes. The following figure [4] shows the attributes, their data type s, and total available values under each attribute.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 311 entries, 0 to 310
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        311 non-null    int64
 1   Age       311 non-null    int64
 2   Gender    311 non-null    object
 3   TB        307 non-null    float64
 4   DB        308 non-null    float64
 5   ALK       309 non-null    float64
 6   SGPT      308 non-null    float64
 7   SGOT      309 non-null    float64
 8   TP        310 non-null    float64
 9   ALB       310 non-null    float64
 10  AG_Ratio  309 non-null    float64
 11  Class     311 non-null    object
dtypes: float64(8), int64(2), object(2)
memory usage: 29.3+ KB
```

```
test_df.isnull().sum()

Age         0
Gender      0
TB          4
DB          3
ALK         2
SGPT        3
SGOT        2
TP          1
ALB         1
AG_Ratio    2
Class       0
dtype: int64
```

Figure 4: Attributes and Data types - Testing dataset

The following figure [5] displays the overall summary of the Testing dataset.

	Age	TB	DB	ALK	SGPT	SGOT	TP	ALB	AG_Ratio
count	311.000000	307.000000	308.000000	309.000000	308.000000	309.000000	310.000000	310.000000	309.000000
mean	45.372990	3.836482	1.726299	277.812298	77.844156	103.734628	6.634516	3.199032	0.937735
std	16.474294	7.554519	3.269869	194.084457	171.754394	227.543019	1.094412	0.811546	0.323404
min	4.000000	0.500000	0.100000	63.000000	10.000000	11.000000	2.700000	0.900000	0.300000
25%	33.000000	0.800000	0.200000	180.000000	22.000000	25.000000	5.925000	2.700000	0.700000
50%	46.000000	1.000000	0.300000	210.000000	33.000000	40.000000	6.800000	3.200000	0.960000
75%	59.000000	2.700000	1.300000	298.000000	60.000000	79.000000	7.300000	3.900000	1.100000
max	90.000000	75.000000	19.700000	1630.000000	2000.000000	2946.000000	9.600000	5.500000	2.800000

Figure 5: Summary of the Testing dataset

4. Visualizations of the Training and Testing datasets.

To visualize the “Class” distribution of the Training and Testing datasets I have used the “Seaborn” python data visualization library which is based on matplotlib. The reason for choosing “Seaborn” library for that is it can easily use for making statistical graphics in Python while it is closely integrated with the Pandas data structures.

The following figure [6] shows the “Class” distribution of the Training dataset.

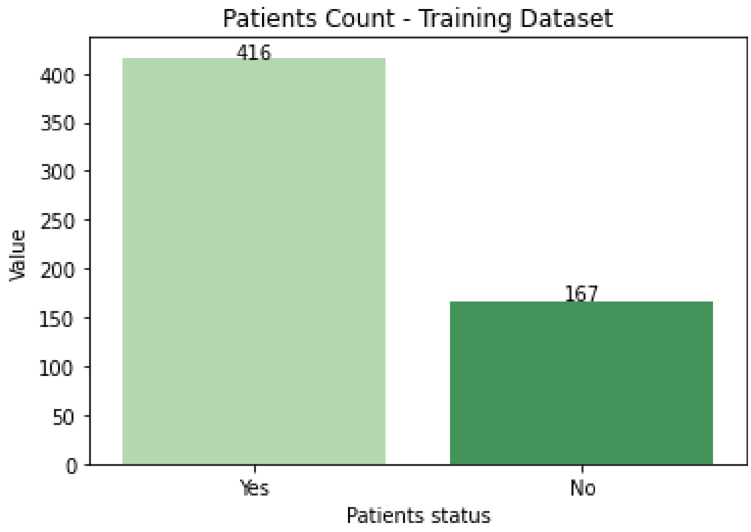


Figure 6: "Class" distribution of the Training dataset

The following figure [7] shows the “Class” distribution of the Testing dataset.

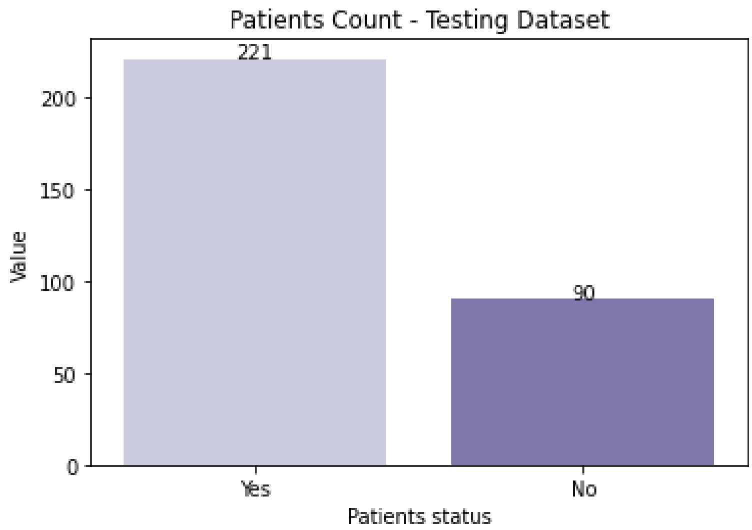


Figure 7: "Class" distribution of the Testing dataset

5. Data pre-processing.

First of all, I have used “pandas.get_dummies” method to convert the categorical variables (Gender, Class) in the both datasets into dummy variables. Following figure [8] displays the status of first 06 rows in the Training dataset after this process.

```
    Age    TB   DB    ALK  SGPT   SGOT   TP  ALB  AG_Ratio  Gender_Male  \
ID
1    65   0.7  0.1  187.0  16.0   18.0  6.8  3.3      0.90            0
2    62  10.9  5.5  699.0  64.0  100.0  7.5  3.2      0.74            1
3    62   7.3  4.1  490.0  60.0   68.0  7.0  3.3      0.89            1
4    58   1.0  0.4  182.0  14.0   20.0  6.8  3.4      1.00            1
5    72   3.9  2.0  195.0  27.0   59.0  7.3  2.4      0.40            1
6    30   0.9  0.3  202.0  15.0   11.0  6.7  3.1      1.10            0

    Class_Yes
ID
1           1
2           1
3           1
4           1
5           1
6           1
```

Figure 8: Converting categorical variables into dummy variables

Since some of the attributes in both datasets had missing values, I have used “Scikit-learn.SimpleImputer” imputation transformer for completing the missing values with the mean of all the attributes in both datasets. The following figure shows the status of attributes in the Testing dataset after the data pre-processing step. As shown in the figure [9] null count is zero for all the attributes.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           583 non-null    float64
 1   Age          583 non-null    float64
 2   TB           583 non-null    float64
 3   DB           583 non-null    float64
 4   ALK          583 non-null    float64
 5   SGPT         583 non-null    float64
 6   SGOT         583 non-null    float64
 7   TP           583 non-null    float64
 8   ALB          583 non-null    float64
 9   AG_Ratio     583 non-null    float64
 10  Gender_Male  583 non-null    float64
 11  Class_Yes    583 non-null    float64
dtypes: float64(12)
memory usage: 54.8 KB
```

Figure 9: Completing the missing values

6. Feature Selection for the training process.

In the feature selection process, I have used the correlation heatmap matrix to understand how the features are related. The following figure [10] shows the correlations between the attributes of the Training dataset. The closer the heatmap is to 1.00, the more directly correlated the attributes are which could point to multicollinearity.

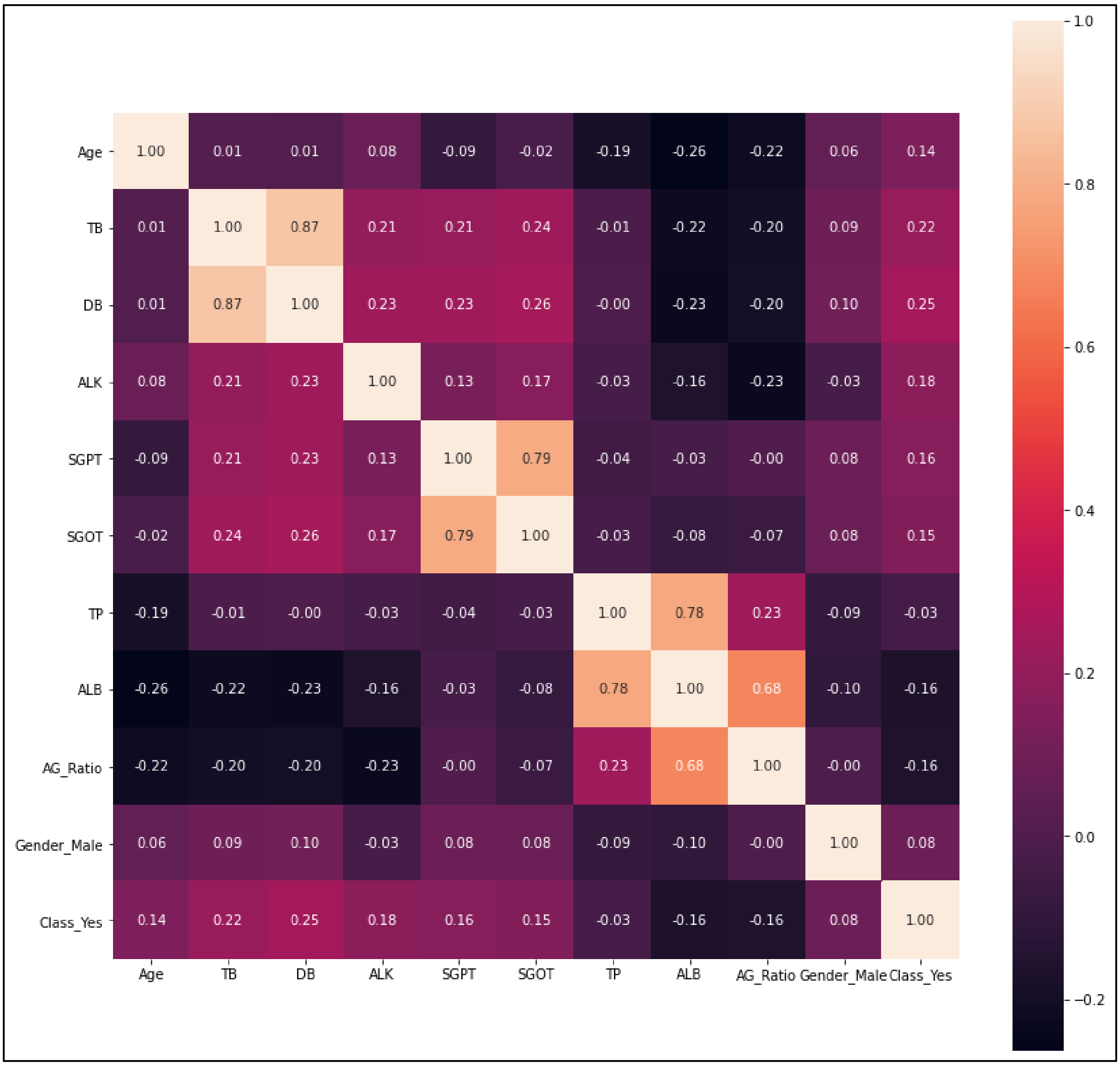


Figure 10: Correlation between the attributes of the Training dataset

From the below pair plot [11], I get a good understanding of how the attributes on the Training dataset are related to each other. Focusing on the relationship to the “Class_Yes” attribute;

- It indicates that there can not be found any linear separability within each two attributes.

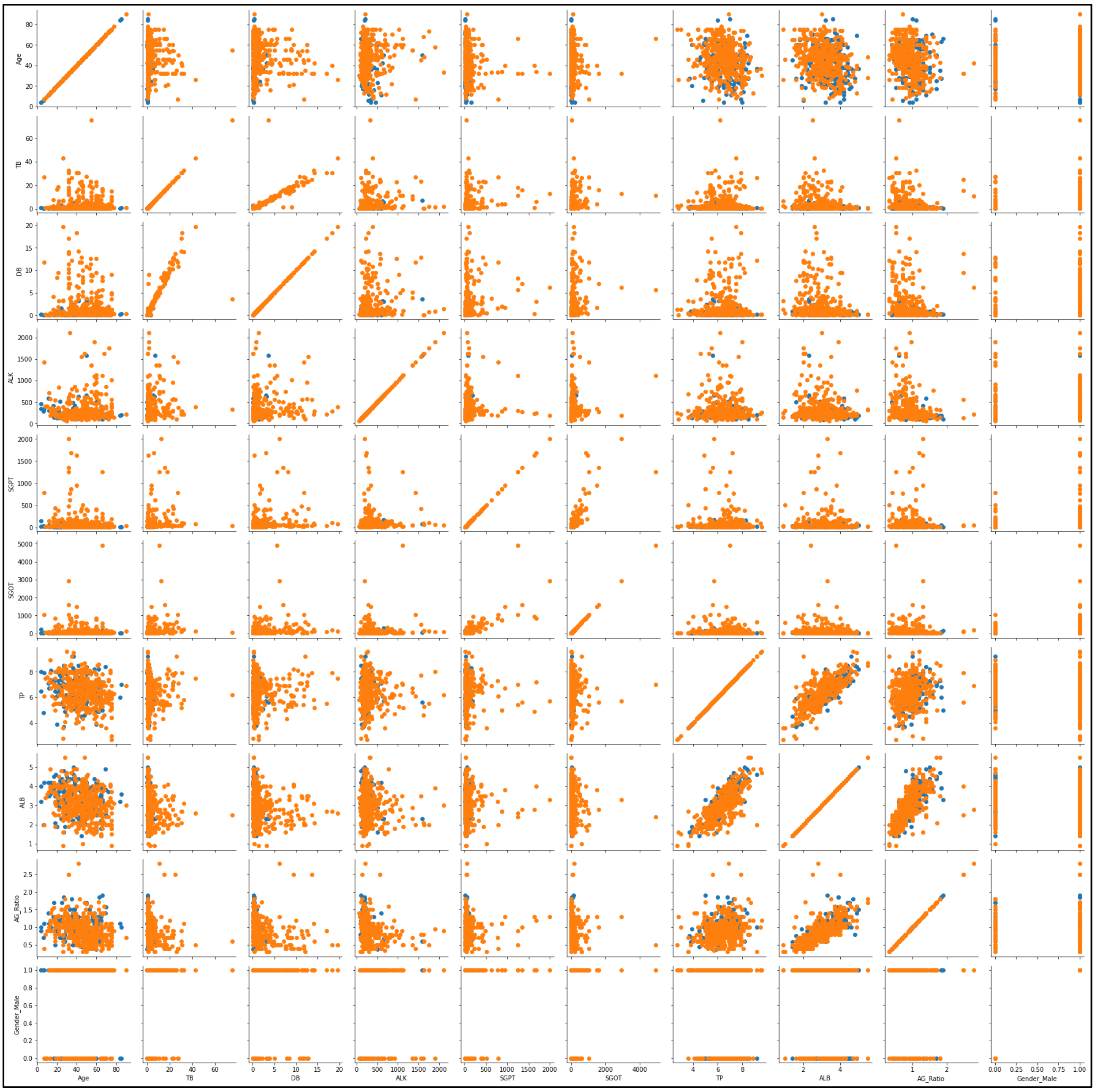


Figure 11: Pair plot for correlations

Since I cannot find any direct linear separability between attributes, I have used ANOVA (Analysis of Variance) for feature selection. The following figure [12] displays the ANOVA-F values I got for the attributes in the Training dataset.

	Feature_name	F_Score
3	DB	37.513824
2	TB	29.447427
4	ALK	20.248643
8	ALB	16.183672
5	SGPT	15.988828
9	AG_Ratio	15.776049
6	SGOT	13.566804
1	Age	11.171429
10	Gender_Male	3.973363
7	TP	0.644499
0	ID	0.209905

Figure 12: F values

According to these values, most of the attributes have F-score which is higher than 1. The "TP" attribute has the lowest value that is 0.644499 (ID value was neglected). So that, I have decided to choose only those for which the F-score is higher than 1.00. Because of that, I have selected the following features for the training process.

- DB
- TB
- ALK
- ALB
- SGPT
- AG_Ratio
- SGOT
- Age
- Gender_Male

7. Building the model.

For building the supervised learning model, I have used Keras from the TensorFlow. Keras act as a wrapper for the TensorFlow. It can easily be used for building machine learning models with having only a few lines of code.

Keras

Figure 13: Keras for python deep learning

Since the planned model has exactly one input tensor and one output tensor, I have used “Sequential model” in Keras for this step. The built model consists with a total of 04 layers which including an input layer, an output layer, and two hidden layers. As mentioned previously, I have selected 09 attributes as the features. So that I have used the Dense layer with 128 nodes with an input shape of about 9 in the input layer. “Sigmoid” function is used as a layer activation function in the output layer. Furthermore, to prevent the model from overfitting I have used “Dropout” layers under the input and hidden layers. Since there are only two label classes (positive - 1, negative - 0) I have used “BinaryCrossentropy” class as the probabilistic lose in the model compilation layer. The following figure [14] shows the summary of built model.

Figure 14: Keras Model summary

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1280
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                4128
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 21,953
Trainable params: 21,953
Non-trainable params: 0
_________________________________________________________________
```

8. Training the model.

To train the built model, I have used 1000 epochs, and the followings [15] are the evaluation metric values for the model.

```
loss: 0.19979175925254822
truepositives: 397.0
falsepositives: 26.0
truenegatives: 141.0
falsenegatives: 19.0
accuracy: 0.9228130578994751
precision: 0.9385342597961426
recall: 0.9543269276618958
auc: 0.9828636050224304
```

Figure 15: Results of the model evaluation

The following figure [16] displays the accuracy against the epochs. According to this accuracy of the model was gradually increased with the number of epochs. Also, there is still a chance

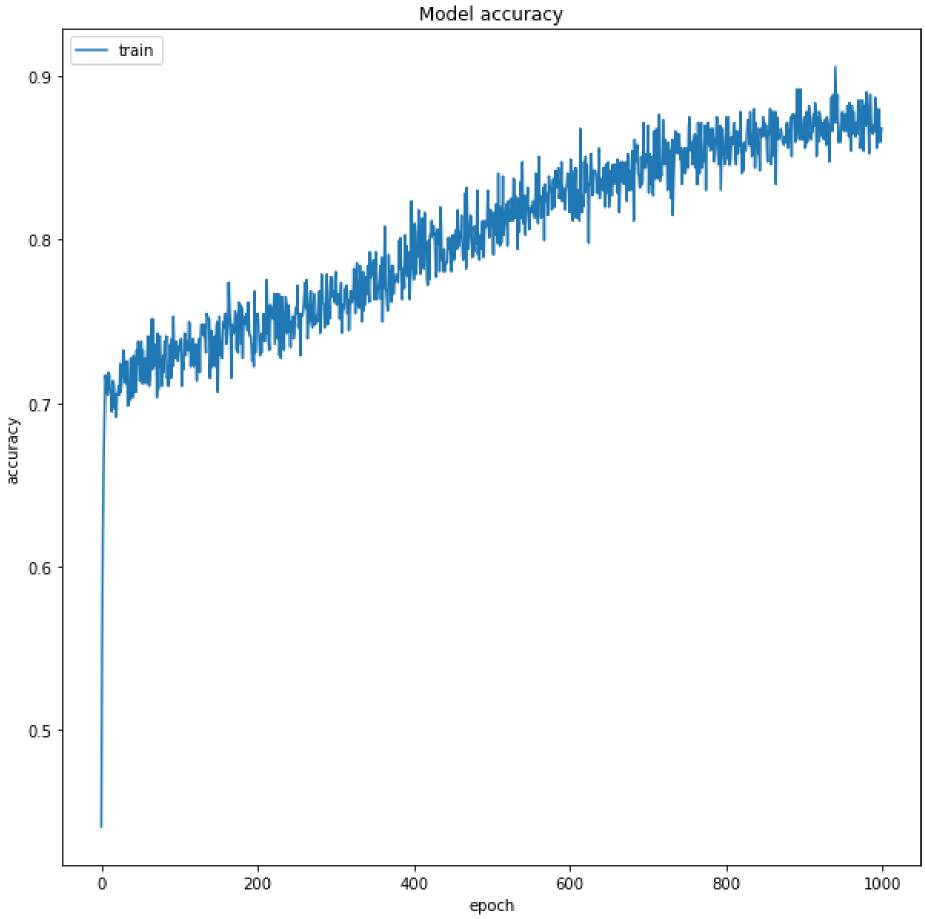


Figure 16: Model accuracy against the epochs

to train a little more in the model since the trend for the accuracy of the Training dataset is still rising for the last few epochs.

The following figure [17] displays the loss against the epochs. From the plot of loss, can see that the model has good performance on the dataset.

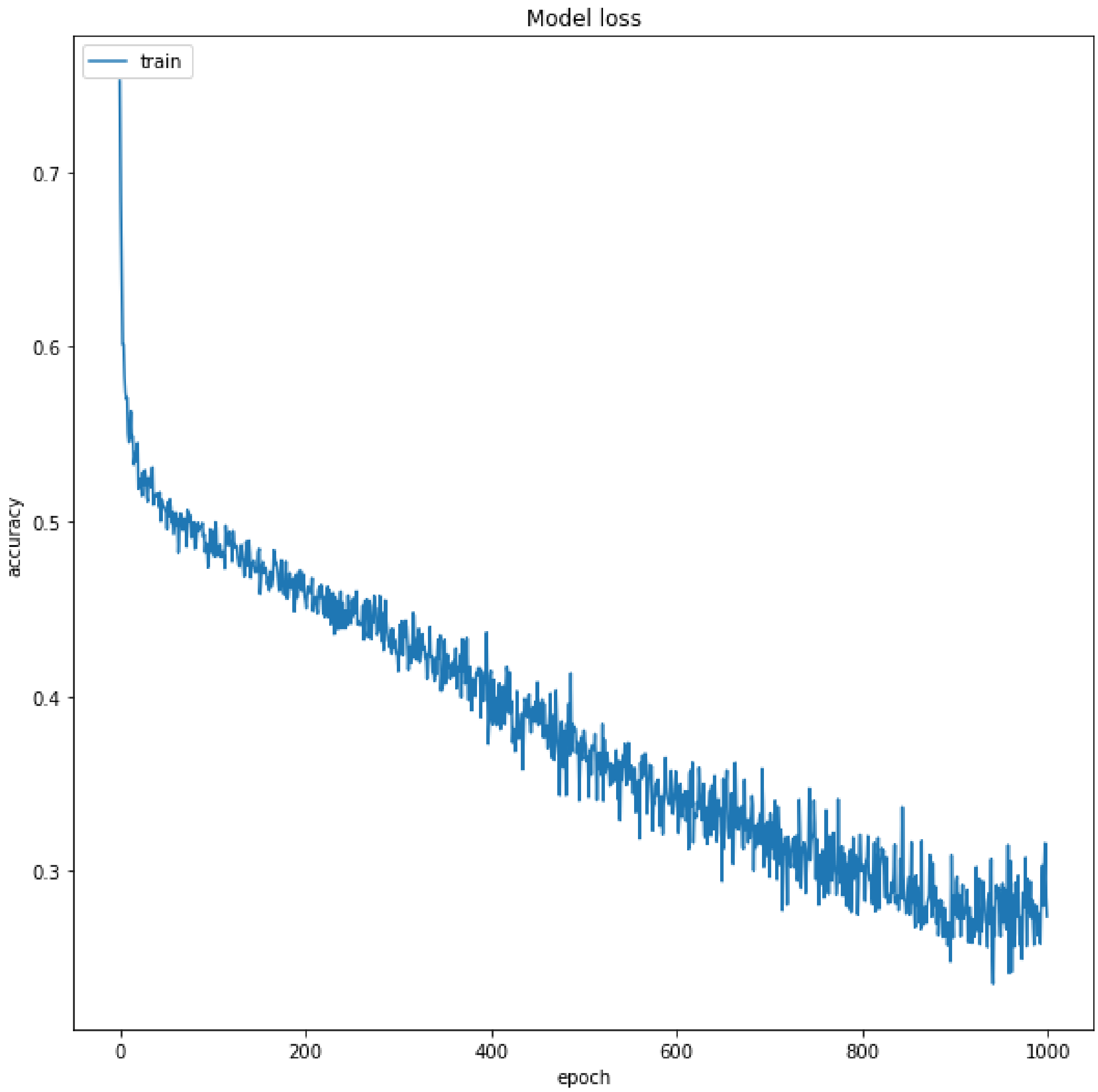


Figure 17: Model loss against the epochs

B. Confusion matrix for the Test dataset after completing your training process for the Test dataset.

- ✓ Correctly identified negative patients (True Negatives): 78
- ✓ Incorrectly identified negative patients (False Positives): 12
- ✓ Incorrectly identified positive patients:(False Negatives): 11
- ✓ Correctly identified positive patients (True Positives): 210
- ✓ Total Positive patients: 221
- ✓ Total Negative Patients: 90

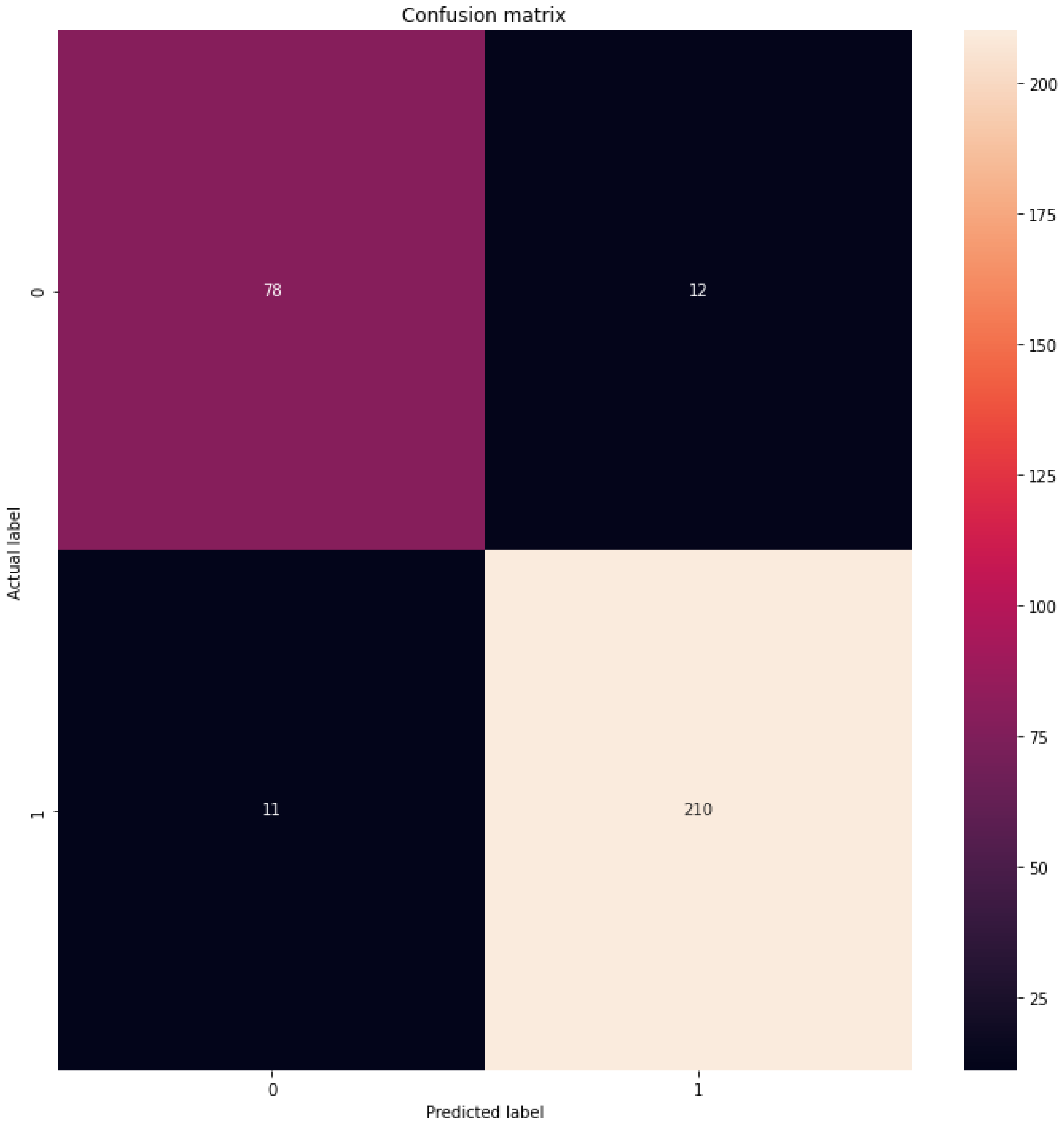


Figure 18: Confusion matric for Test dataset

c. Calculate the measures for the Test dataset.

(TP - True Positive, TN - True Negative, FP - False Positive, FN - False Negative)

1. **Accuracy:**

Accuracy $= \frac{TP + TN}{TP+TN+FN+FP}$

$= \frac{210 + 78}{311}$

$=$ 0.9260

2. **Precision:**

Precision $= \frac{TP}{TP+FP}$

$= \frac{210}{210 + 12}$

$=$ 0.9459

3. **Sensitivity:**

Sensitivity $= \frac{TP}{TP + FN}$

$= \frac{210}{210 + 11}$

$=$ 0.9502

4. **Specificity:**

Specificity: $= \frac{TN}{TN+ FP}$

$= \frac{78}{78 + 12}$

$=$ 0.8666

5. **Error Rate:**

Error Rate $= \frac{FP + FN}{TP+TN+FN+FP}$

$= \frac{12 + 11}{311}$

$=$ 0.0739

C. List of Appendix

1. CoLab Notebook (Code lines)

https://colab.research.google.com/drive/1UNVkWT1Nzgifl8K68gYqHY6Uk39aJwWZ?usp=sharing

Capstone Project - Januka Dharmapriya (266)

```
import numpy as np
import pandas as pd
import os

from sklearn import preprocessing
from scipy.stats import pearsonr

# machine learning  - supervised
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import accuracy_score
```

Loading the Training and Testing datasets

- Read two seperate work sheets in the same Excel file using pandas.

```
train_df = pd.read_csv('https://raw.githubusercontent.com/JanukaD/Capstone-Project/main/datas
test_df = pd.read_csv('https://raw.githubusercontent.com/JanukaD/Capstone-Project/main/datase
```

- Describing the Training dataset

- First five rows of the training dataset.

```
train_df.head()
```

	ID	Age	Gender	TB	DB	ALK	SGPT	SGOT	TP	ALB	AG_Ratio	Class
0	1	65	Female	0.7	0.1	187.0	16.0	18.0	6.8	3.3	0.90	Yes
1	2	62	Male	10.9	5.5	699.0	64.0	100.0	7.5	3.2	0.74	Yes
2	3	62	Male	7.3	4.1	490.0	60.0	68.0	7.0	3.3	0.89	Yes
3	4	58	Male	1.0	0.4	182.0	14.0	20.0	6.8	3.4	1.00	Yes
4	5	72	Male	3.9	2.0	195.0	27.0	59.0	7.3	2.4	0.40	Yes

- Check the columns that contains null values in the training dataset.

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        583 non-null    int64
 1   Age       583 non-null    int64
 2   Gender    583 non-null    object
 3   TB        581 non-null    float64
 4   DB        579 non-null    float64
 5   ALK       581 non-null    float64
 6   SGPT      582 non-null    float64
 7   SGOT      582 non-null    float64
 8   TP        581 non-null    float64
 9   ALB       581 non-null    float64
 10  AG_Ratio  582 non-null    float64
 11  Class     583 non-null    object
dtypes: float64(8), int64(2), object(2)
memory usage: 54.8+ KB
```

- Count of the null values in each column in the training dataset.

```
train_df.isnull().sum()
```

```
ID          0
Age         0
Gender      0
TB          2
DB          4
ALK         2
SGPT        1
SGOT        1
TP          2
ALB         2
AG_Ratio    1
Class       0
dtype: int64
```

- Describing the Training dataset

```
train_df.describe()
```

	ID	Age	TB	DB	ALK	SGPT	SG
count	583.000000	583.000000	581.000000	579.000000	581.000000	582.000000	582.0000
mean	292.000000	44.746141	3.307573	1.486701	291.063683	80.780069	110.0738
std	168.441879	16.189833	6.218411	2.817115	243.206230	182.770380	289.1402
min	1.000000	4.000000	0.400000	0.100000	63.000000	10.000000	10.0000
25%	146.500000	33.000000	0.800000	0.200000	176.000000	23.000000	25.0000
50%	292.000000	45.000000	1.000000	0.300000	208.000000	35.000000	42.0000
75%	437.500000	58.000000	2.600000	1.300000	298.000000	60.750000	87.0000

- Describing the Testing dataset

- First five rows of the testing dataset.

```
test_df.head()
```

	ID	Age	Gender	TB	DB	ALK	SGPT	SGOT	TP	ALB	AG_Ratio	Class
0	1	65	Female	0.7	0.1	187.0	16.0	18.0	6.8	3.3	0.90	Yes
1	2	62	Male	10.9	5.5	699.0	64.0	100.0	7.5	3.2	0.74	Yes
2	3	62	Male	7.3	4.1	490.0	60.0	68.0	7.0	3.3	0.89	Yes
3	4	58	Male	1.0	0.4	182.0	14.0	20.0	6.8	3.4	1.00	Yes
4	5	72	Male	3.9	2.0	195.0	27.0	59.0	7.3	2.4	0.40	Yes

- Check the columns that contains null values in the testing dataset.

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 311 entries, 0 to 310
Data columns (total 12 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   ID        311 non-null    int64
 1   Age       311 non-null    int64
 2   Gender    311 non-null    object
 3   TB        307 non-null    float64
 4   DB        308 non-null    float64
 5   ALK       309 non-null    float64
 6   SGPT      308 non-null    float64
 7   SGOT      309 non-null    float64
 8   TP        310 non-null    float64
```

```
 9   ALB       310 non-null    float64
 10  AG_Ratio  309 non-null    float64
 11  Class     311 non-null    object
dtypes: float64(8), int64(2), object(2)
memory usage: 29.3+ KB
```

- Count of the null values in each column in the testing dataset.

```
test_df.isnull().sum()
```

```
ID          0
Age         0
Gender      0
TB          4
DB          3
ALK         2
SGPT        3
SGOT        2
TP          1
ALB         1
AG_Ratio    2
Class       0
dtype: int64
```

- Describing the Testing dataset

```
test_df.describe()
```

	ID	Age	TB	DB	ALK	SGPT	SG
count	311.000000	311.000000	307.000000	308.000000	309.000000	308.000000	309.0000
mean	156.000000	45.372990	3.836482	1.726299	277.812298	77.844156	103.7346
std	89.922189	16.474294	7.554519	3.269869	194.084457	171.754394	227.5430
min	1.000000	4.000000	0.500000	0.100000	63.000000	10.000000	11.0000
25%	78.500000	33.000000	0.800000	0.200000	180.000000	22.000000	25.0000
50%	156.000000	46.000000	1.000000	0.300000	210.000000	33.000000	40.0000
75%	233.500000	59.000000	2.700000	1.300000	298.000000	60.000000	79.0000
max	311.000000	90.000000	75.000000	19.700000	1630.000000	2000.000000	2946.0000

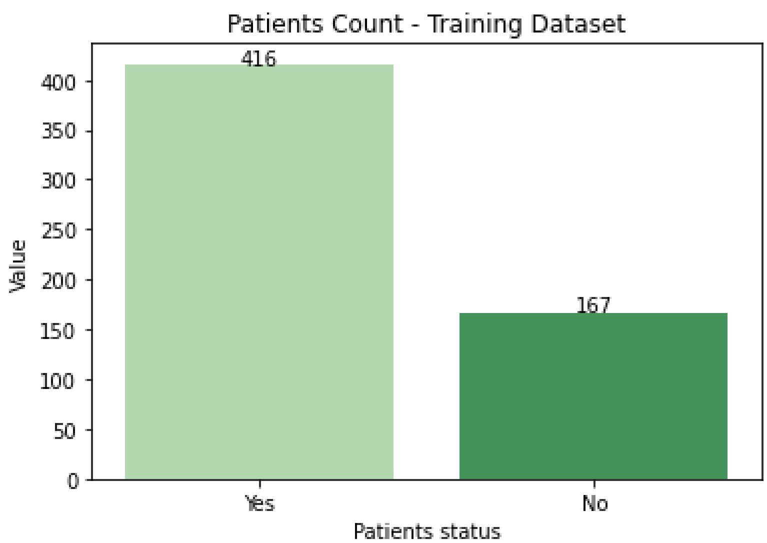
Visualization of the datasets

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

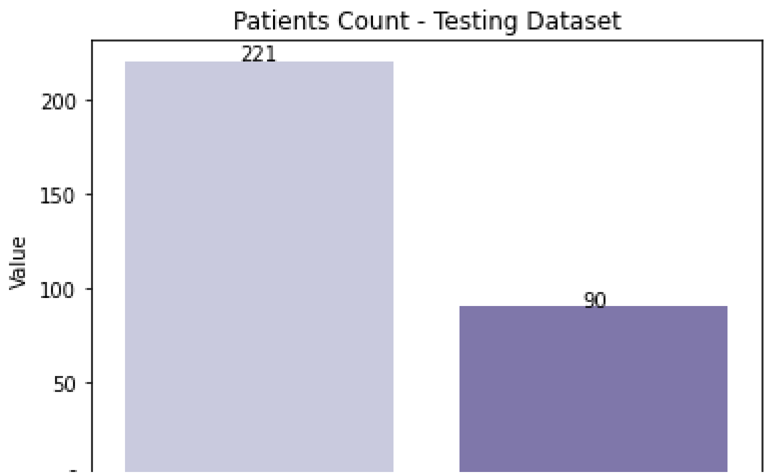
- Class distribution of the Training dataset

```
graph = sns.countplot(data = train_df, x = "Class", label = "Count", palette="Greens")
plt.title('Patients Count - Training Dataset')
plt.xlabel('Patients status')
plt.ylabel('Value')

i=0
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1,
        train_df['Class'].value_counts()[i],ha="center")
    i += 1
```



- Class distribution of the Test dataset

```
graph = sns.countplot(data = test_df, x = "Class", label = "Count", palette="Purples")
plt.title('Patients Count - Testing Dataset')
plt.xlabel('Patients status')
plt.ylabel('Value')

i=0
for p in graph.patches:
    height = p.get_height()
    graph.text(p.get_x()+p.get_width()/2., height + 0.1,
        test_df['Class'].value_counts()[i],ha="center")
    i += 1
```



Data pre-processing

- Convert categorical variables (Gender & Class) to the dummy variables - Training dataset

```
train_df_dummy = pd.get_dummies(train_df, columns=["Gender","Class"], drop_first=True)
print(train_df_dummy.head(6))
```

```
   ID  Age    TB   DB    ALK  SGPT   SGOT   TP  ALB  AG_Ratio  Gender_Male  \
0   1   65   0.7  0.1  187.0  16.0   18.0  6.8  3.3      0.90            0
1   2   62  10.9  5.5  699.0  64.0  100.0  7.5  3.2      0.74            1
2   3   62   7.3  4.1  490.0  60.0   68.0  7.0  3.3      0.89            1
3   4   58   1.0  0.4  182.0  14.0   20.0  6.8  3.4      1.00            1
4   5   72   3.9  2.0  195.0  27.0   59.0  7.3  2.4      0.40            1
5   6   46   1.8  0.7  208.0  19.0   14.0  7.6  4.4      1.30            1

   Class_Yes
0          1
1          1
2          1
3          1
4          1
5          1
```

- Convert categorical variables (Gender & class) to the dummy variables - Testing dataset

```
test_df_dummy = pd.get_dummies(test_df, columns=["Gender","Class"], drop_first=True)
print(test_df_dummy.head(6))
```

```
   ID  Age    TB   DB    ALK  SGPT   SGOT   TP  ALB  AG_Ratio  Gender_Male  \
0   1   65   0.7  0.1  187.0  16.0   18.0  6.8  3.3      0.90            0
1   2   62  10.9  5.5  699.0  64.0  100.0  7.5  3.2      0.74            1
2   3   62   7.3  4.1  490.0  60.0   68.0  7.0  3.3      0.89            1
3   4   58   1.0  0.4  182.0  14.0   20.0  6.8  3.4      1.00            1
4   5   72   3.9  2.0  195.0  27.0   59.0  7.3  2.4      0.40            1
5   6   30   0.9  0.3  202.0  15.0   11.0  6.7  3.1      1.10            0
```

```
   Class_Yes
0          1
1          1
2          1
3          1
4          1
5          1
```

- Fill missing values in column with mean - Training dataset

```
from sklearn.impute import SimpleImputer

imp=SimpleImputer(missing_values=np.NaN, strategy = 'mean')

train_df_imputed = pd.DataFrame(imp.fit_transform(train_df_dummy))
train_df_imputed.columns=train_df_dummy.columns
train_df_imputed.index=train_df_dummy.index
```

```
train_df_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   ID           583 non-null    float64
 1   Age          583 non-null    float64
 2   TB           583 non-null    float64
 3   DB           583 non-null    float64
 4   ALK          583 non-null    float64
 5   SGPT         583 non-null    float64
 6   SGOT         583 non-null    float64
 7   TP           583 non-null    float64
 8   ALB          583 non-null    float64
 9   AG_Ratio     583 non-null    float64
 10  Gender_Male  583 non-null    float64
 11  Class_Yes    583 non-null    float64
dtypes: float64(12)
memory usage: 54.8 KB
```

```
train_df_imputed.isnull().sum()
```

```
ID             0
Age            0
TB             0
DB             0
ALK            0
SGPT           0
SGOT           0
TP             0
```

```
ALB            0
AG_Ratio       0
Gender_Male    0
Class_Yes      0
dtype: int64
```

- Fill missing values in column with mean - Testing dataset

```
test_df_imputed = pd.DataFrame(imp.fit_transform(test_df_dummy))
test_df_imputed.columns=test_df_dummy.columns
test_df_imputed.index=test_df_dummy.index
```

```
test_df_imputed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 311 entries, 0 to 310
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   ID           311 non-null    float64
 1   Age          311 non-null    float64
 2   TB           311 non-null    float64
 3   DB           311 non-null    float64
 4   ALK          311 non-null    float64
 5   SGPT         311 non-null    float64
 6   SGOT         311 non-null    float64
 7   TP           311 non-null    float64
 8   ALB          311 non-null    float64
 9   AG_Ratio     311 non-null    float64
 10  Gender_Male  311 non-null    float64
 11  Class_Yes    311 non-null    float64
dtypes: float64(12)
memory usage: 29.3 KB
```

```
test_df_imputed.isnull().sum()
```

```
ID             0
Age            0
TB             0
DB             0
ALK            0
SGPT           0
SGOT           0
TP             0
ALB            0
AG_Ratio       0
Gender_Male    0
Class_Yes      0
dtype: int64
```

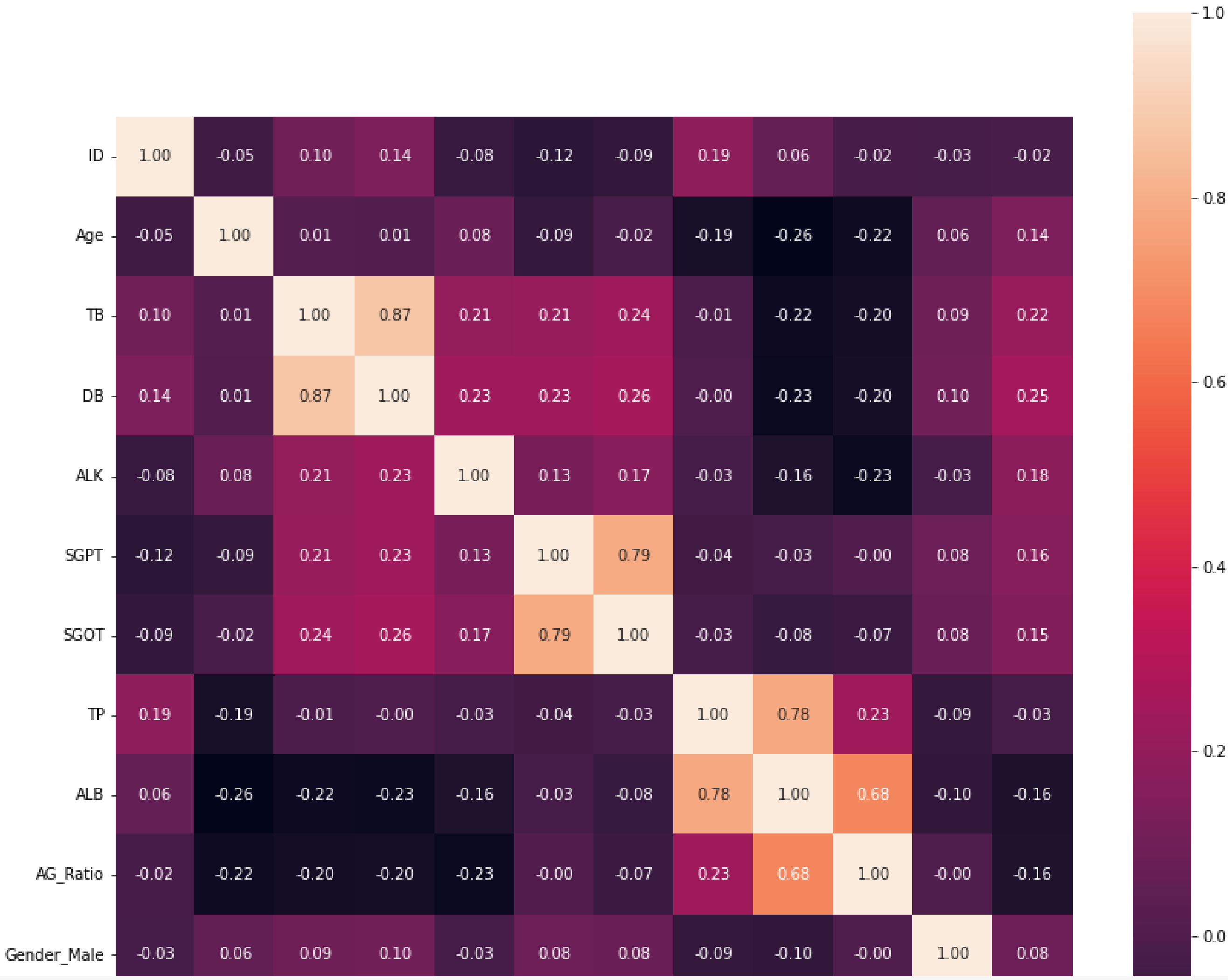
Feature Selection for training process

- Correlations in the training dataset

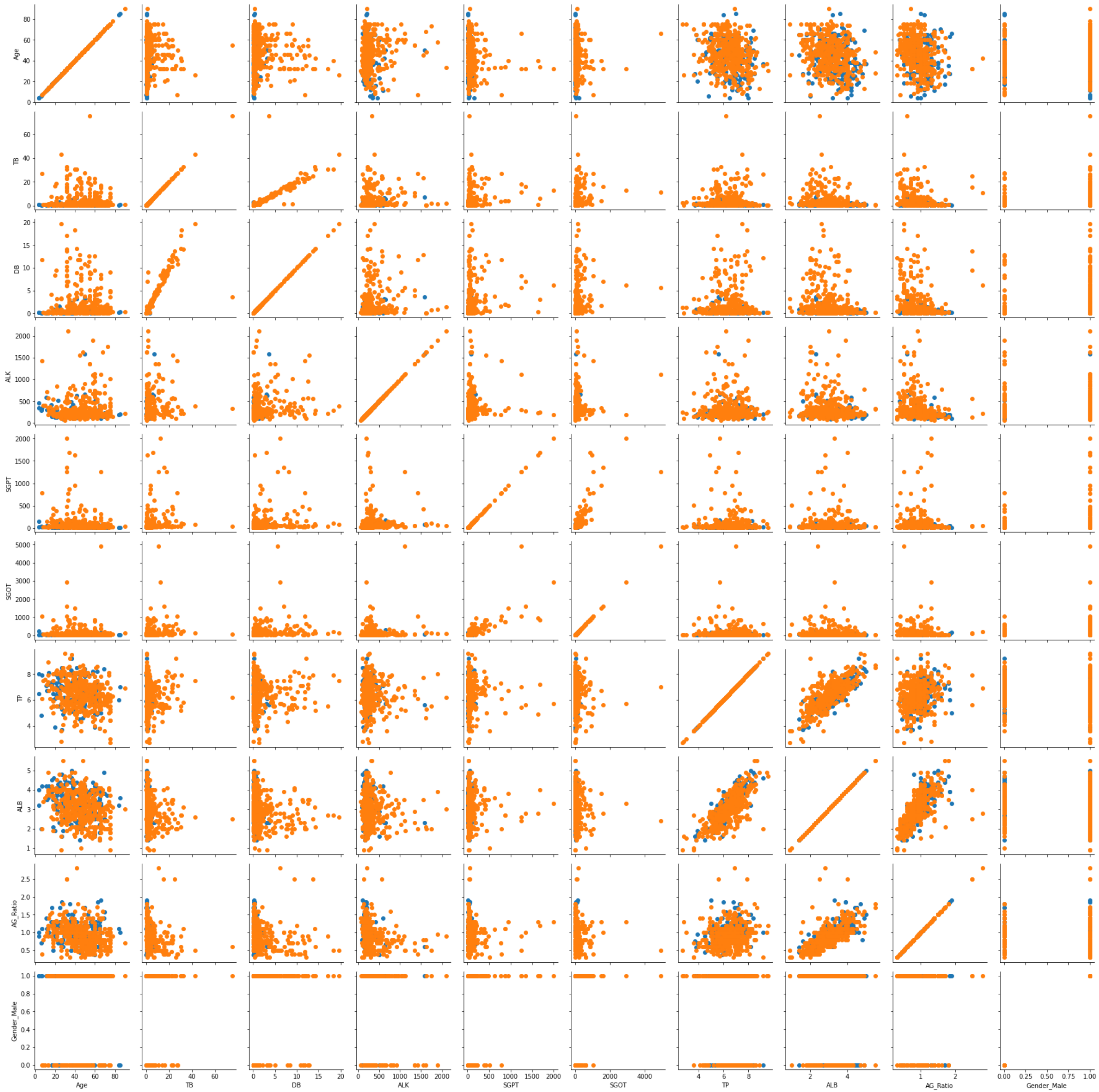
```
train_df_imputed.corr()
```

	ID	Age	TB	DB	ALK	SGPT	SGOT	
ID	1.000000	-0.052385	0.102097	0.137979	-0.079053	-0.124809	-0.094242	0.18
Age	-0.052385	1.000000	0.013671	0.007469	0.081128	-0.087106	-0.020252	-0.18
TB	0.102097	0.013671	1.000000	0.873826	0.205340	0.213492	0.237244	-0.00
DB	0.137979	0.007469	0.873826	1.000000	0.232494	0.233465	0.257226	-0.00
ALK	-0.079053	0.081128	0.205340	0.232494	1.000000	0.125071	0.166413	-0.03
SGPT	-0.124809	-0.087106	0.213492	0.233465	0.125071	1.000000	0.791756	-0.04
SGOT	-0.094242	-0.020252	0.237244	0.257226	0.166413	0.791756	1.000000	-0.02
TP	0.189367	-0.188979	-0.009687	-0.001903	-0.030051	-0.043292	-0.026929	1.00
ALB	0.062698	-0.263220	-0.221134	-0.228159	-0.163532	-0.028336	-0.084926	0.78
AG_Ratio	-0.024613	-0.215796	-0.203780	-0.198986	-0.232344	-0.001605	-0.069384	0.23
Gender_Male	-0.029633	0.056560	0.088439	0.099636	-0.028170	0.082542	0.079348	-0.08
Class_Yes	-0.019004	0.137351	0.219634	0.246275	0.183515	0.163653	0.151056	-0.03

```
correlations = train_df_imputed.corr()
plt.figure(figsize=(14,14))
g = sns.heatmap(correlations,cbar = True, square = True, annot=True, fmt= '.2f', annot_kws={'
```




```
g = sns.PairGrid(train_df_imputed, hue = "Class_Yes", vars=['Age','TB','DB','ALK', 'SGPT', 'S
                                                                'AG_Ratio', 'Gender_Male'])
g.map(plt.scatter)
plt.show()
```



Age
TB
DB
ALK
SGPT
SGOT
TP
ALB
AG_Ratio
Gender_Male

- Getting "ANOVA F" measures

```
from sklearn.feature_selection import SelectKBest,f_classif

fvalue_selector = SelectKBest(score_func=f_classif, k="all")

X = train_df_imputed[train_df_imputed.columns.drop("Class_Yes")]
y = (train_df_imputed["Class_Yes"])

fvalue_selector.fit(X, y)
names = X.columns.values[fvalue_selector.get_support()]
scores = fvalue_selector.scores_[fvalue_selector.get_support()]
names_scores = list(zip(names, scores))
ns_df = pd.DataFrame(data = names_scores, columns= ['Feature_name','F_Score'])
ns_df_sorted = ns_df.sort_values(['F_Score','Feature_name'], ascending = [False, True])
print(ns_df_sorted)
```

```
   Feature_name    F_Score
3            DB  37.513824
2            TB  29.447427
4           ALK  20.248643
8           ALB  16.183672
5          SGPT  15.988828
9      AG_Ratio  15.776049
6          SGOT  13.566804
1           Age  11.171429
10  Gender_Male   3.973363
7            TP   0.644499
0            ID   0.209905
```

Building the Model

```
import tensorflow as tf

train_df_labels = np.array(train_df_imputed.pop("Class_Yes"))
test_df_labels = np.array(test_df_imputed.pop("Class_Yes"))
```

- Training the selected features (except the Gender)

```
selected_feature_columns = ['Age', 'Gender_Male','TB','DB','ALK', 'SGPT', 'SGOT', 'ALB','AG_R

train_features = np.array(train_df_imputed[selected_feature_columns])
test_features = np.array(test_df_imputed[selected_feature_columns])
```

- Standardizing the selected features using skleran

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)

train_features = np.clip(train_features, -3, 3)
test_features = np.clip(test_features, -3, 3)
```

- Creating a Keras model (Sequential model)

```
from tensorflow import keras
from tensorflow.keras import layers

maxnorm = tf.keras.constraints.max_norm

model = keras.Sequential(
    [
        layers.Dense(128, activation="relu", input_shape=(9,), kernel_constraint=maxnorm(3)),
        layers.Dropout(0.5),

        layers.Dense(128, activation="relu", kernel_constraint=maxnorm(3)),
        layers.Dropout(0.5),

        layers.Dense(32, kernel_constraint=maxnorm(3)),
        layers.Dropout(0.2),

        layers.Dense(1, activation='sigmoid'), #output layer
    ]
)

#Model compilation
model.compile(optimizer='Nadam',
                 loss="binary_crossentropy",
                 metrics=[tf.keras.metrics.TruePositives(name='truepositives'),
                          tf.keras.metrics.FalsePositives(name='falsepositives'),
                          tf.keras.metrics.TrueNegatives(name='truenegatives'),
                          tf.keras.metrics.FalseNegatives(name='falsenegatives'),
                          tf.keras.metrics.BinaryAccuracy(name='accuracy'),
                          tf.keras.metrics.Precision(name='precision'),
                          tf.keras.metrics.Recall(name='recall'),
```

```
                              tf.keras.metrics.AUC(name='auc')])

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 128)               1280

 dropout_9 (Dropout)         (None, 128)               0

 dense_13 (Dense)            (None, 128)               16512

 dropout_10 (Dropout)        (None, 128)               0

 dense_14 (Dense)            (None, 32)                4128

 dropout_11 (Dropout)        (None, 32)                0

 dense_15 (Dense)            (None, 1)                 33

=================================================================
Total params: 21,953
Trainable params: 21,953
Non-trainable params: 0
_________________________________________________________________
```

```
model.get_config()
```

```
         'bias_initializer': {'class_name': 'Zeros', 'config': {}},
         'bias_regularizer': None,
         'dtype': 'float32',
         'kernel_constraint': {'class_name': 'MaxNorm',
          'config': {'axis': 0, 'max_value': 3}},
         'kernel_initializer': {'class_name': 'GlorotUniform',
          'config': {'seed': None}},
         'kernel_regularizer': None,
         'name': 'dense_13',
         'trainable': True,
         'units': 128,
         'use_bias': True}},
       {'class_name': 'Dropout',
        'config': {'dtype': 'float32',
         'name': 'dropout_10',
         'noise_shape': None,
         'rate': 0.5,
         'seed': None,
         'trainable': True}},
       {'class_name': 'Dense',
        'config': {'activation': 'linear',
         'activity_regularizer': None,
         'bias_constraint': None,
         'bias_initializer': {'class_name': 'Zeros', 'config': {}},
         'bias_regularizer': None,
```

```
        'dtype': 'float32',
        'kernel_constraint': {'class_name': 'MaxNorm',
         'config': {'axis': 0, 'max_value': 3}},
        'kernel_initializer': {'class_name': 'GlorotUniform',
         'config': {'seed': None}},
        'kernel_regularizer': None,
        'name': 'dense_14',
        'trainable': True,
        'units': 32,
        'use_bias': True}},
      {'class_name': 'Dropout',
       'config': {'dtype': 'float32',
        'name': 'dropout_11',
        'noise_shape': None,
        'rate': 0.2,
        'seed': None,
        'trainable': True}},
      {'class_name': 'Dense',
       'config': {'activation': 'sigmoid',
        'activity_regularizer': None,
        'bias_constraint': None,
        'bias_initializer': {'class_name': 'Zeros', 'config': {}},
        'bias_regularizer': None,
        'dtype': 'float32',
        'kernel_constraint': None,
        'kernel_initializer': {'class_name': 'GlorotUniform',
         'config': {'seed': None}},
        'kernel_regularizer': None,
        'name': 'dense_15',
        'trainable': True,
        'units': 1,
        'use_bias': True}}],
     'name': 'sequential_3'}
```


Training the model

- Training the build model using dataset

```
model_trained = model.fit(
    train_features,
    train_df_labels,
    batch_size=320,
    epochs=1000,
    verbose=1,)
```

```
Epoch 973/1000
2/2 [==============================] - 0s 12ms/step - loss: 0.2905 - truepositives: 3
Epoch 974/1000
2/2 [==============================] - 0s 12ms/step - loss: 0.2752 - truepositives: 3
Epoch 975/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2735 - truepositives: 3
```

```
2/2 [==============================] - 0s 13ms/step - loss: 0.2735 - truepositives: 3
Epoch 976/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.3018 - truepositives: 3
Epoch 977/1000
2/2 [==============================] - 0s 14ms/step - loss: 0.2988 - truepositives: 3
Epoch 978/1000
2/2 [==============================] - 0s 17ms/step - loss: 0.2869 - truepositives: 3
Epoch 979/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2831 - truepositives: 3
Epoch 980/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2766 - truepositives: 3
Epoch 981/1000
2/2 [==============================] - 0s 10ms/step - loss: 0.2947 - truepositives: 3
Epoch 982/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2981 - truepositives: 3
Epoch 983/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2842 - truepositives: 3
Epoch 984/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2752 - truepositives: 3
Epoch 985/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2757 - truepositives: 3
Epoch 986/1000
2/2 [==============================] - 0s 15ms/step - loss: 0.2921 - truepositives: 3
Epoch 987/1000
2/2 [==============================] - 0s 17ms/step - loss: 0.2666 - truepositives: 3
Epoch 988/1000
2/2 [==============================] - 0s 14ms/step - loss: 0.2576 - truepositives: 3
Epoch 989/1000
2/2 [==============================] - 0s 14ms/step - loss: 0.2935 - truepositives: 3
Epoch 990/1000
2/2 [==============================] - 0s 14ms/step - loss: 0.2910 - truepositives: 3
Epoch 991/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2732 - truepositives: 3
Epoch 992/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2727 - truepositives: 3
Epoch 993/1000
2/2 [==============================] - 0s 16ms/step - loss: 0.2777 - truepositives: 3
Epoch 994/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2996 - truepositives: 3
Epoch 995/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2810 - truepositives: 3
Epoch 996/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2941 - truepositives: 3
Epoch 997/1000
2/2 [==============================] - 0s 12ms/step - loss: 0.2813 - truepositives: 3
Epoch 998/1000
2/2 [==============================] - 0s 11ms/step - loss: 0.2787 - truepositives: 3
Epoch 999/1000
2/2 [==============================] - 0s 14ms/step - loss: 0.3121 - truepositives: 3
Epoch 1000/1000
2/2 [==============================] - 0s 13ms/step - loss: 0.2999 - truepositives: 3
```

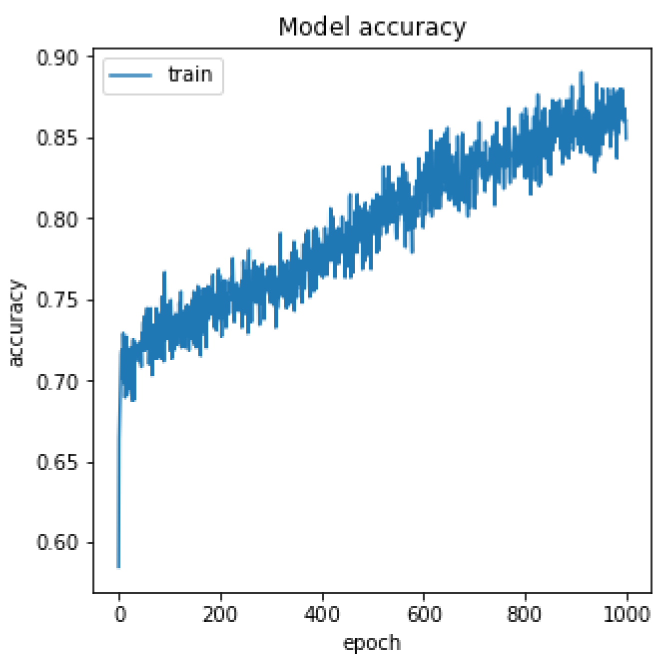
- Evaluate the built model

```
results = model.evaluate(train_features, train_df_labels, batch_size=32, verbose=0)

for x in range(len(results)):
    print(f"{model.metrics_names[x]}: {results[x]}")
```

```
loss: 0.20478622615337372
truepositives: 392.0
falsepositives: 16.0
truenegatives: 151.0
falsenegatives: 24.0
accuracy: 0.9313893914222717
precision: 0.9607843160629272
recall: 0.942307710647583
auc: 0.9842382669448853
```

- summarize history for Accuracy & Loss

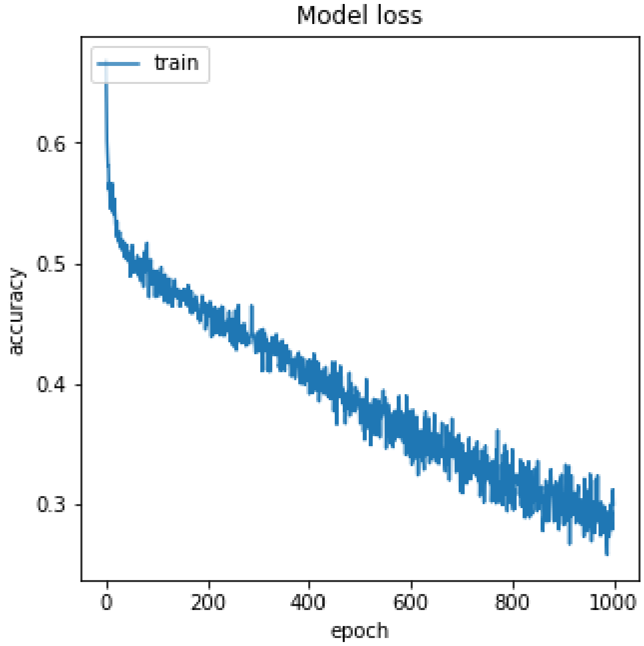
```
# summarize history for accuracy
plt.figure(figsize=(5,5))
plt.plot(model_trained.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
# summarize history for loss
plt.figure(figsize=(5,5))
plt.plot(model_trained.history['loss'])
```

```
plt.title('Model loss')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7f63f40fd090>
```



Evaluating the model

```
train_predictions_baseline = model.predict(train_features, batch_size=32)
test_predictions_baseline = model.predict(test_features, batch_size=32)
```

- Confusion Matrix

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(test_df_labels, test_predictions_baseline >0.5)
plt.figure(figsize=(12,12))
sns.heatmap(cm, annot=True, fmt="d",xticklabels=True, yticklabels=True,)

plt.title('Confusion matrix')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

print('Correctly identified negative patients (True Negatives): ', cm[0][0])
print('Incorrectly identified negative patients (False Positives): ', cm[0][1])
print('Incorrectly identified positive patients:(False Negatives): ', cm[1][0])
print('Correctly identified positive patients (True Positives): ', cm[1][1])
```

```
print('Total Positive patients: ', np.sum(cm[1]))
print('Total Negative Patients: ', np.sum(cm[0]))
```

```
Correctly identified negative patients (True Negatives):  78
Incorrectly identified negative patients (False Positives):  12
Incorrectly identified positive patients:(False Negatives):  11
Correctly identified positive patients (True Positives):  210
Total Positive patients:  221
Total Negative Patients:  90
```