ARIGNAR ANNA GOVERNMENT ARTS COLLEGE

VILLUPURAM – 605 602.

DEPARTMENT OF COMPUTER APPLICATIONS

MACHINE LEARNING WITH PYTHON

Project Title : A Review of Liver Patient Analysis Method using Machine Learning

TeamID : NM2023TMID16843

Team Leader: ARUNRAJ A (132AA232BB4D2DB9F1344A2366F01B31)

Team member: ARUN KUMAR S (E09B34A726432B9A8A349974C6611EB10)

Team member: ABINESH S (664363E0911F43692F54FC5F528149B3)

Team member: SARAVANAN P (1DB64E8C83FF627BCDF43FC85F1D3519)55

Abstract

Around a million deaths occur due to liver diseases globally. There are several traditional methods to diagnose liver diseases, but they are expensive. Early prediction of liver disease would benefit all individuals prone to liver diseases by providing early treatment. As technology is growing in health care, machine learning significantly affects health care for predicting conditions at early stages. This study finds how accurate machine learning is in predicting liver disease.

This present study introduces the liver disease prediction (LDP) method in predicting liver disease that can be utilised by health professionals, stakeholders, students and researchers. Five algorithms, namely Support Vector Machine (SVM), Naïve Bayes, K-Nearest Neighbors (K-NN), Linear Discriminant Analysis (LDA), and Classification and Regression Trees (CART), are selected. The accuracy is compared to uncover the best classification method for predicting liver disease using R and Python. From the results, K-NN obtains the best accuracy with 91.7%, and the autoencoder network achieved 92.1% accuracy, which is above the acceptable level of accuracy and can be considered for liver disease prediction.

Introduction

In this project we will analyse the parameters of various classification algorithms and compare their predictive accuracies so as

to find out the best classifier for determining the liver disease. This project compares various classification algorithms such as Random Forest, Logistic Regression, KNN and ANN Algorithm with an aim to identify the best technique. Based on this study, Random Forest with the highest accuracy outperformed the other algorithms and can be further utilised in the prediction of liver disease and can be recommended to the Liver.According to World Health Organization (WHO) report in 2018, the number of deaths due to liver diseases is around one million and ranked 11th in the world with a critical number of fatalities (World Total Deaths, n.d.). Unnoticed at the initial stages, these symptoms are only visible when the disease turns chronic. However, even though the liver is partially infected, it can still function (Devikanniga et al., 2020).

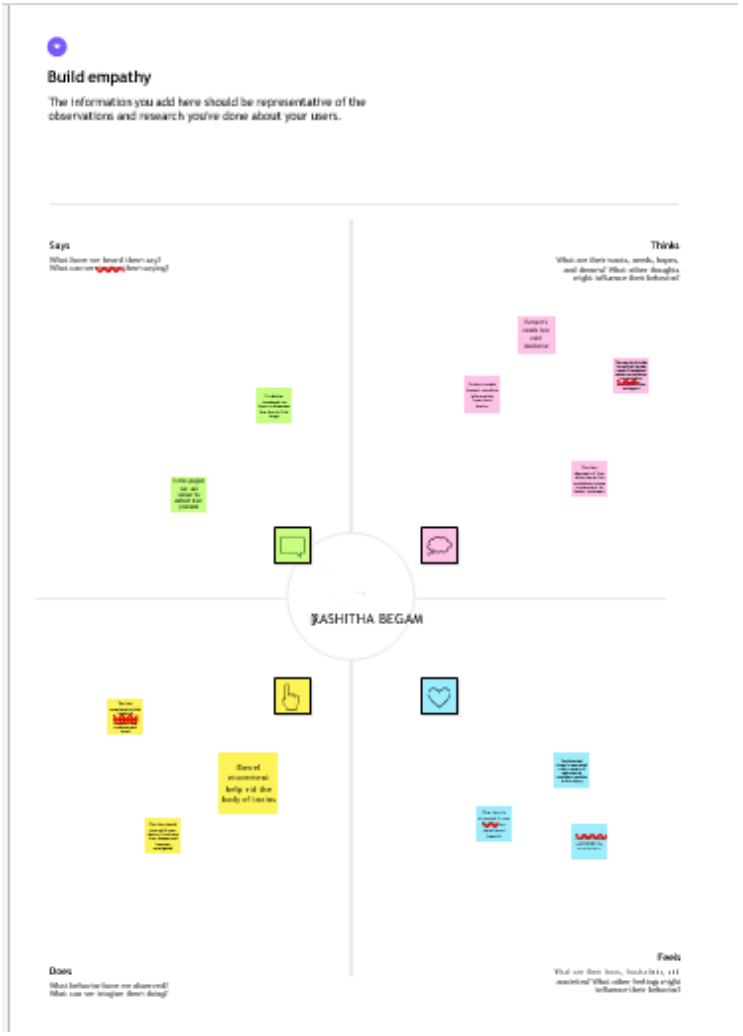
Diagnosis of liver diseases can be divided into three stages i.e., the first stage is liver inflammation, the second is liver scarring (cirrhosis), and the final stage is liver cancer or failure. Since these scenarios are present in liver disease, early prediction is significant to provide better health for New Zealanders. If liver disease is diagnosed early, there will be a chance of early treatment and control of deaths due to liver diseases (Arbain & Balakrishnan, 2019). But when the liver fails to function, few treatments are available except liver transplantation (Shaheamlunget al., 2020), which is very expensive, particularly in New Zealand (Hepatitis C, 2021). Apparently, in New Zealand, 35 40% of thepopulation are not diagnosed with Hepatitis C at the early stages because of the asymptomatic behaviour of liver disease. Unfortunately,most of these individuals do not know the risks linked to liver disease. Due to the asymptomatic behaviour and higher costs of liver disease treatment, it is essential to prevent or diagnose early for better treatment.

Purpose

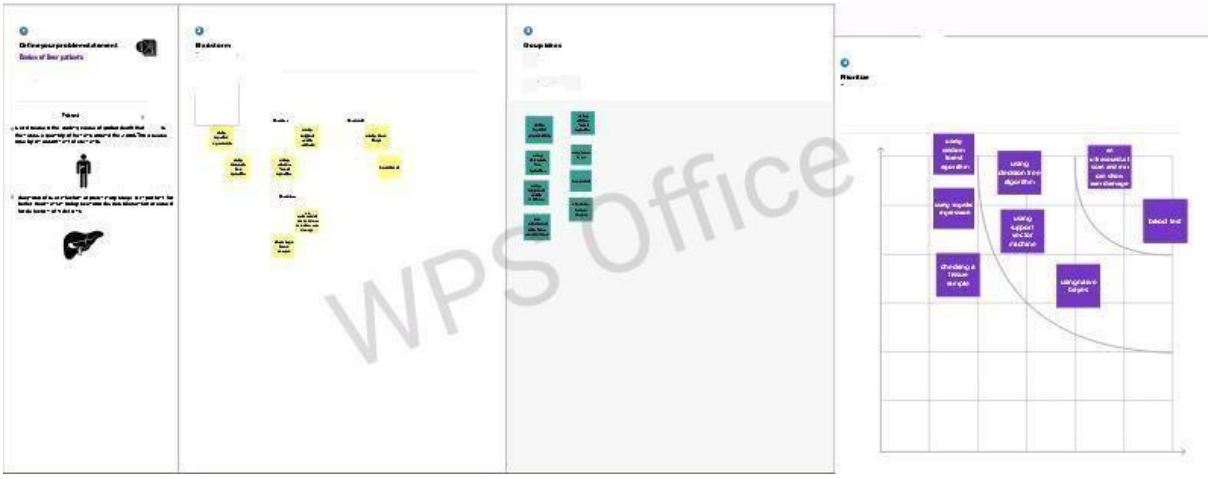
Liver function tests can be used to: Screen for liver infections, such as hepatitis. Monitor the progression of a disease, such as viral or alcoholic hepatitis, and determine how well a treatment is working. Measure the severity of a disease, particularly scarring of the liver (cirrhosis)

Problem Definetion & Design Thinking

Empathy map


Build empathy
The information you add here should be representative of the observations and research you've done about your users.
Says
Thinks
JIASHITHA BEGAM
Does
Feels

Ideation & brainstorming map



Result :

Liver Patients analysis using machine learning can provide accurate and realiable results for the diagnosis, prognosis, and treatment of liver disease. Machine learning algorithm can analyse large amounts of patients data and identify patterns that may be difficult for human experts to detect.

Home.html Source code

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>Bootstrap Example</title>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width,initial-scale=1">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">

<style type="text/css">

 body{

 background-color:#ffcf0059;

}

nav{
```

```
background-color:#ad38c2;

height:60px;

}

.navbar-brand{

color:white

}

</style>

</head>

<body>


<nav class="navbar">

<div class="containe-field">

<div class="navbar-header">


<a class="navbar-brand"><h1>Liver Patient Analysis</h1></a></nav>

</div>


 <h2>Introduction</h2>


  <p>Liver diseases averts the normal function of the liver. This disease is caused by an assortment of elements

that harm the liver. Diagnosis of liver infection at the preliminary stage is important for better treatment. In today’s scenario devices like sensors are used for detection of infections. Accurate classification

techniques are required for automatic identification of disease samples. This disease diagnosis is very

costly and complicated. Therefore, the goal of this work is to evaluate the performance of different Machine
```

Learning algorithms in order to reduce the high cost of liver disease diagnosis. Early prediction of liver disease using classification algorithms is an efficacious task that can help the doctors to diagnose the disease within a short duration of time. In this project we will analyse the parameters of various

classification algorithms and compare their predictive accuracies so as to find out the best classifier for determining the liver disease. This project compares various classification algorithms such as Random Forest, Logistic Regression, KNN and ANN Algorithm with an aim to identify the best technique. Based on this study, Random Forest with the highest accuracy outperformed the other algorithms and can be further utilised in the prediction of liver disease and can be recommended to the user.

Technical Architecture:

Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

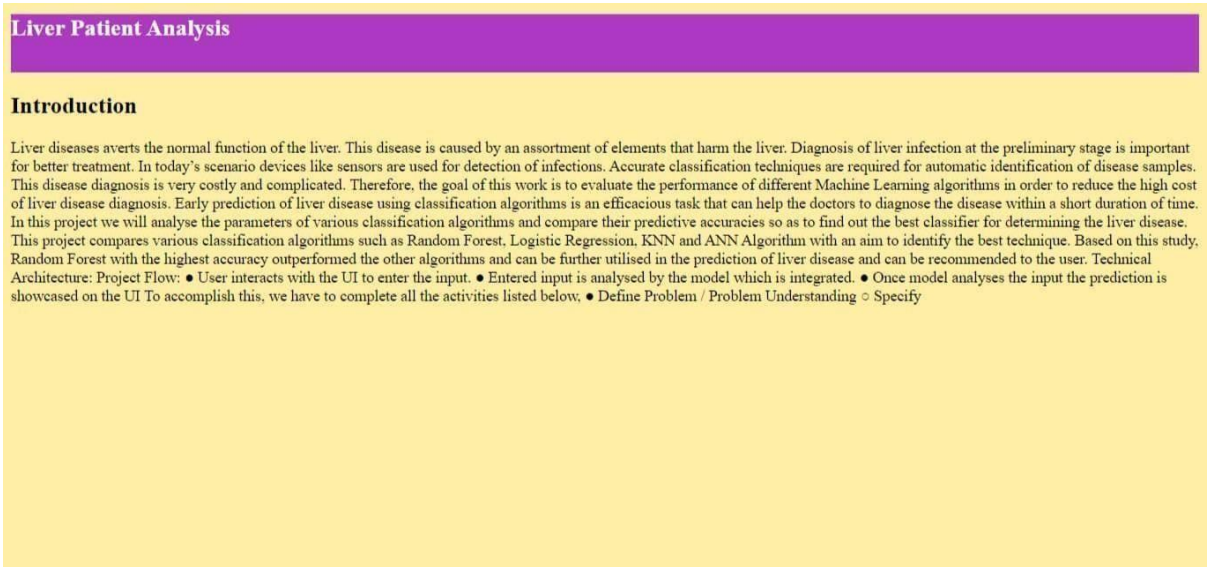
To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
  - Specify

</p>

</html>

Output :



Index.html Source code

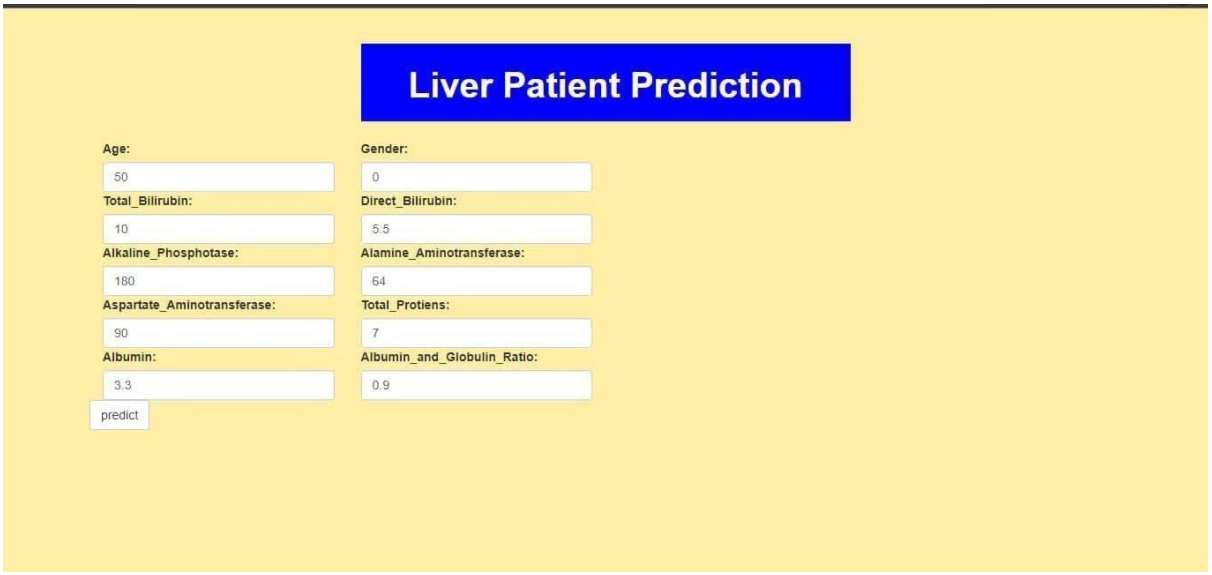
```
<!DOCTYPE html>

<html>

<head>

<title>Liver Patient Analysis</title>

<!-- Latest Compiled and minified CSS -->

<Link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<style type="text/css">

body{ background-color:

#ffcf0059;

  }

.page-header{

background-color:

blue; width: 100%;

height: auto; text-align:

center; padding-top:

5px; color: #fff

}

h1{ font-size:

  40px; font-

  weight: bold;

  }

</style>

</head>

</body>

  <div class="container">

  <div class="row">

  <div class="col-md-3"></div>
```

```
<div class="col-md-6">
<div class="page-header">
<h1>Liver Patient Prediction</h1>
</div>
</div>
</div>
</div>
  <div class="container">
  <div class="row">
  <div class="col-mid-3"></div>
  <div class="col-md-6">
  <form action="/data_predict" method="post">
  <div class="row">
  <div class="col-md-6">
  <div class="form-gorup">
  <label for="Age">Age:</label>
  <input type="text" class="form-control" id="age" name="age">
  </div>
  </div>
  <div class="col-md-6">
  <div class="form-gorup">
  <label for="gender">Gender:</label>
  <input type="text" class="form-control" id="gender" name="gender">
  </div>
  </div>
  <div class="col-md-6">
  <div class="form-gorup">
  <label for="tb">Total_Bilirubin:</label>
```

```
<input type="text" class="form-control" id="tb" name="tb">
</div>
</div>
</div>
<div class="col-md-6">
<div class="form-gorup">
<label for="db">Direct_Bilirubin:</label>
<input type="text" class="form-control" id="db" name="db">
</div>
</div>
<div class="col-md-6">
<div class="form-gorup">
<label for="ap">Alkaline_Phosphotase:</label>
<input type="text" class="form-control" id="ap" name="ap">
</div>
</div>
<div class="col-md-6">
<div class="form-gorup">
<label for="aa1">Alamine_Aminotransferase:</label>
<input type="text" class="form-control" id="aa1" name="aa1">
</div>
</div>
<div class="col-md-6">
<div class="form-gorup">
<label for="aa2">Aspartate_Aminotransferase:</label>
<input type="text" class="form-control" id="aa2" name="aa2">
</div>
</div>
```

```
<div class="col-md-6">
<div class="form-gorup">
<label for="tp">Total_Protiens:</label>
<input type="text" class="form-control" id="tp" name="tp">
</div>
</div>
<div class="col-md-6">
<div class="form-gorup">
<label for="a">Albumin:</label>
<input type="text" class="form-control" id="a" name="a">
</div>
</div>
<div class="col-md-6">
<div class="form-gorup">
<label for="agr">Albumin_and_Globulin_Ratio:</label>
<input type="text" class="form-control" id="agr" name="agr">
</div>
<button type="submit" class="btn btn-default" > predict</button>
</form>
</div>
</div>
</div>
<!-- Latest Compiled and minified javascript -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</body>
</html>
```

Output :

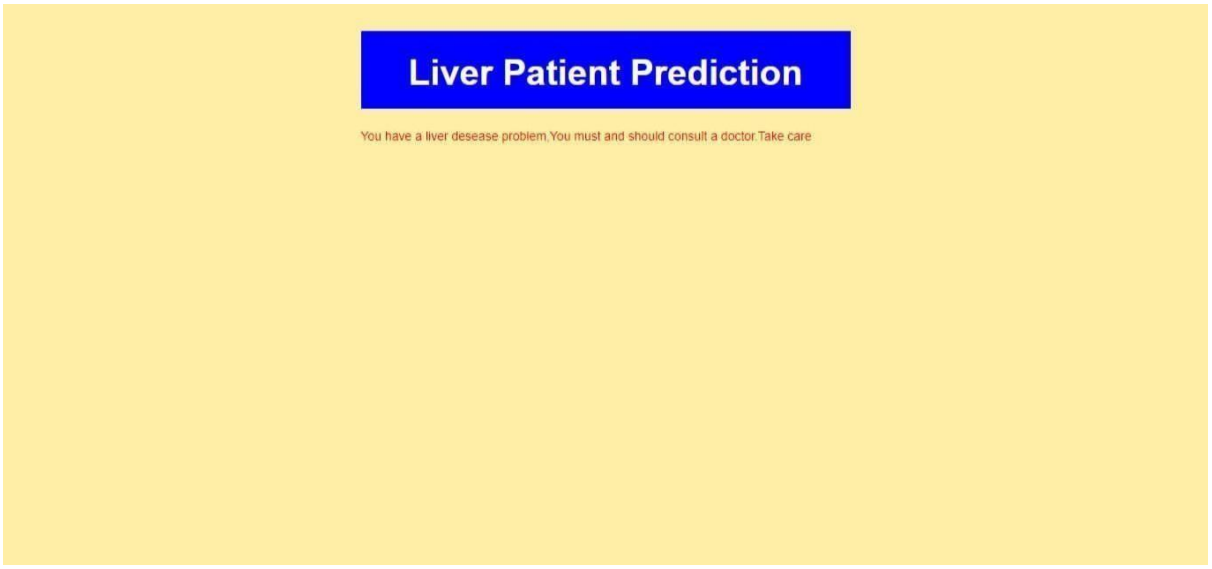


No chance.html source code

```
<!DOCTYPE html>

<html>

<head>

  <title>Liver Patient Analysis</title>

  <!-- Latest Compiled and minified CSS -->

  <Link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">


  <style type="text/css">

    body{

      background-color: #ffcf0059;

    }

    .page-header{

      background-color: blue;

      width: 100%;

      height: auto;

      text-align: center;

      padding-top: 5px;
```

```
        color: #fff;
          }
          h1{ font-size:
        40px; font-
        weight: bold;
    }
  </style>
  </head>
  </body>


    <div class="container">
    <div class="row">
    <div class="col-md-3"></div>
    <div class="col-md-6">
    <div class="page-header">
    <h1>Liver Patient Prediction</h1></div>
    <p style="color:red;">
    You have a liver desease problem,You must and should consult a doctor.Take care</h3></p>
    </style>
    </div>
    </div>
    </div>
    </div>
  <!-- Latest Compiled and minified javascript -->
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
</html>
```

Output:



Advantages

General

Diagnoses, grades and stages:

Hepatitis C

Hepatitis B

Steatohepatitis

Autoimmune hepatitis

Evalutes abnormal liver function tests

Identifies hepatotoxicity

Clarifies uncertain diagnoses

Confirms etiology of liver masses

Defines extent of necroinflammatory activity

Differentiates fiabrosis from cirrhosis

Liver transplant

Identifies acute cellular rejection

Defines recurrence of original disease

Identifies progressive fibrosis

Diagnoses other liver processes.

Disadvantage

General

Invasive

Accessibility to the procedure

Need for training

Cost

Sample

Sampling error

Intraobserver and interobserver variations in

Interpretatooin

Specimen length and width

Patient

Site pain

Shoulder pain

Neuralgia

Hypotension

Bleeding

Hemothorax

Hemoblilia

Application :

- Diagnosis: Liver patients analysis can be used to diagnose liver diseases such as cirrhosis, hepatitis, and Livercancer,by analysing various biomarkers such as liver enyzmes, bilirubin, and albumin, doctors can determine the health of the liver and diagnose any underlying diseases.

- Treatment: Liver patients analysis can also to monitor the effectiveness of treatmentsfor liver diseases. By regulary analysing liver function tests and other biomarkers, doctors can achieve optimal results

Conclusion

- pertinent Since the liver disease is not easy to diagnose, given the delicate nature of its signs, this research is in determining the algorithms that have better accuracy in predicting this dreadful disease.
- Once the dataset is selected, the preprocessing step is conducted by replacing the missing values and balancing the dataset.

- After that, using R, five different supervised learning methods are applied (i.e., SVM, Naïve Bayes, K-NN, LDA, and CART), and the accuracy with confusion matrix metrics are recorded.
- In this study, the autoencoder with 3-layers achieved an accuracy of 92.1%, slightly higher than K-NN due to its ability to ascertain overlapping features better than conventional K-NNs. Most of the algorithms are more than the acceptable level of accuracy, which is 75%.

- The results from this study would be able to assist health professionals and relevant stakeholders in the early detection of liver disease.

Future scope

- In this paper ,we proposed and built a machine learning based on a hybrid classifier to be used as a classification model for liver diseases diagnosis to improve performance and experrts to identify the chances of disease and conscious orescription of further treatment healthcare and examination .

- In future work, the use of fast datasets technique like apache hadoop or spark can be incorporated with this technique. In addition to this, we can use distributed refined algorithm like forest tree implement in apache hadoop to increase scalability and efficieny.

Appenix

Source code

Mliestone 2:

```
import pandas as pd

import numpy as np import seaborn as sns import

matplotlib.pyplot as plt from matplotlib import rcParams from

scipy import stats import warnings

warnings.filterwarnings('ignore') from sklearn.tree import

DecisionTreeClassifier from sklearn.ensemble import

RandomForestClassifier from sklearn.model_selection import

train_test_split from sklearn.metrics import

classification_report,confusion_matrix
```

```
data=pd.read_csv("indian_liver_patient.csv")
```

```
data.head() data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   Age                         583 non-null    int64  
 1   Gender                      583 non-null    object 
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
```

5	Alamine_Aminotransferase	583 non-null	int64
6	Aspartate_Aminotransferase	583 non-null	int64
7	Total_Protiens	583 non-null	float64
8	Albumin	583 non-null	float64
9	Albumin_and_Globulin_Ratio	579 non-null	float64
10	Dataset	583 non-null	int64

dtypes: float64(5), int64(5), object(1)

memory usage: 50.2+ KB

data.isnull().any()

Age	False
Gender	False
Total_Bilirubin	False
Direct_Bilirubin	False
Alkaline_Phosphotase	False
Alamine_Aminotransferase	False
Aspartate_Aminotransferase	False
Total_Protiens	False
Albumin	False
Albumin_and_Globulin_Ratio	True

Dataset False

dtype: bool

data.isnull().sum()

Age 0

Gender 0

Total_Bilirubin 0

Direct_Bilirubin 0

Alkaline_Phosphotase 0

Alamine_Aminotransferase 0

Aspartate_Aminotransferase 0

Total_Protiens 0

Albumin 0

Albumin_and_Globulin_Ratio 4

Dataset 0

dtype: int64

data['Albumin_and_Globulin_Ratio'] =
data['Albumin_and_Globulin_Ratio'].fillna(data['Albumin_and_Globulin_Ratio'].mode()[0])

data.isnull().sum()

Age 0

Gender 0

Total_Bilirubin 0 Direct_Bilirubin 0

Alkaline_Phosphotase 0

Alamine_Aminotransferase 0

Aspartate_Aminotransferase 0

Total_Protiens 0

Albumin 0

Albumin_and_Globulin_Ratio 0

Dataset 0

dtype: int64

```
from sklearn.preprocessing import
LabelEncoder lc = LabelEncoder()
data['Gender'] =
lc.fit_transform(data['Gender'])
```

Milestone 3:

```
data.describe()
```

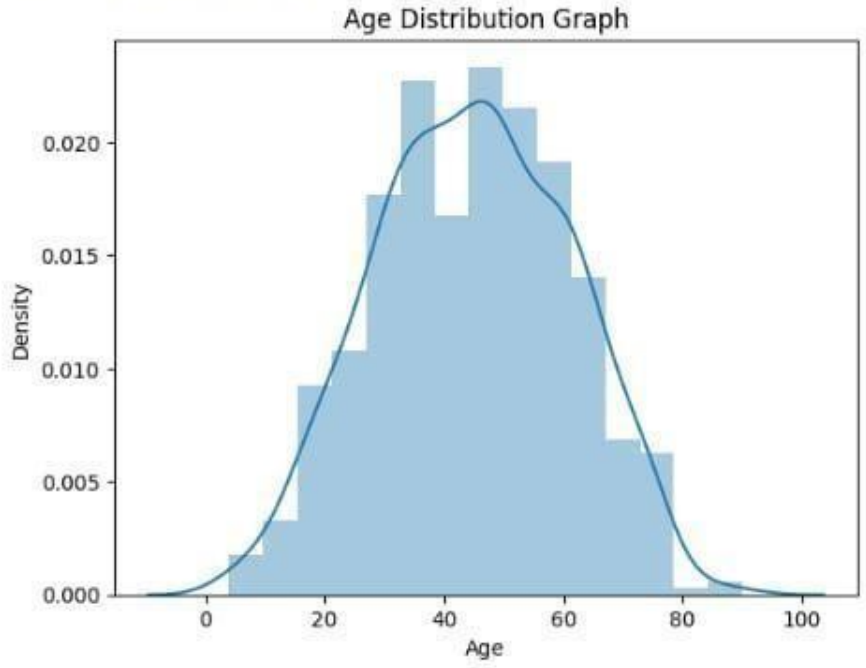
	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000
mean	44.746141	0.756432	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.141852	0.947427	1.286449
std	16.189833	0.429603	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.795519	0.318522	0.452490
min	4.000000	0.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000	0.300000	1.000000
25%	33.000000	1.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.600000	0.700000	1.000000
50%	45.000000	1.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000	0.950000	1.000000
75%	58.000000	1.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.800000	1.100000	2.000000
max	90.000000	1.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000	2.800000	2.000000

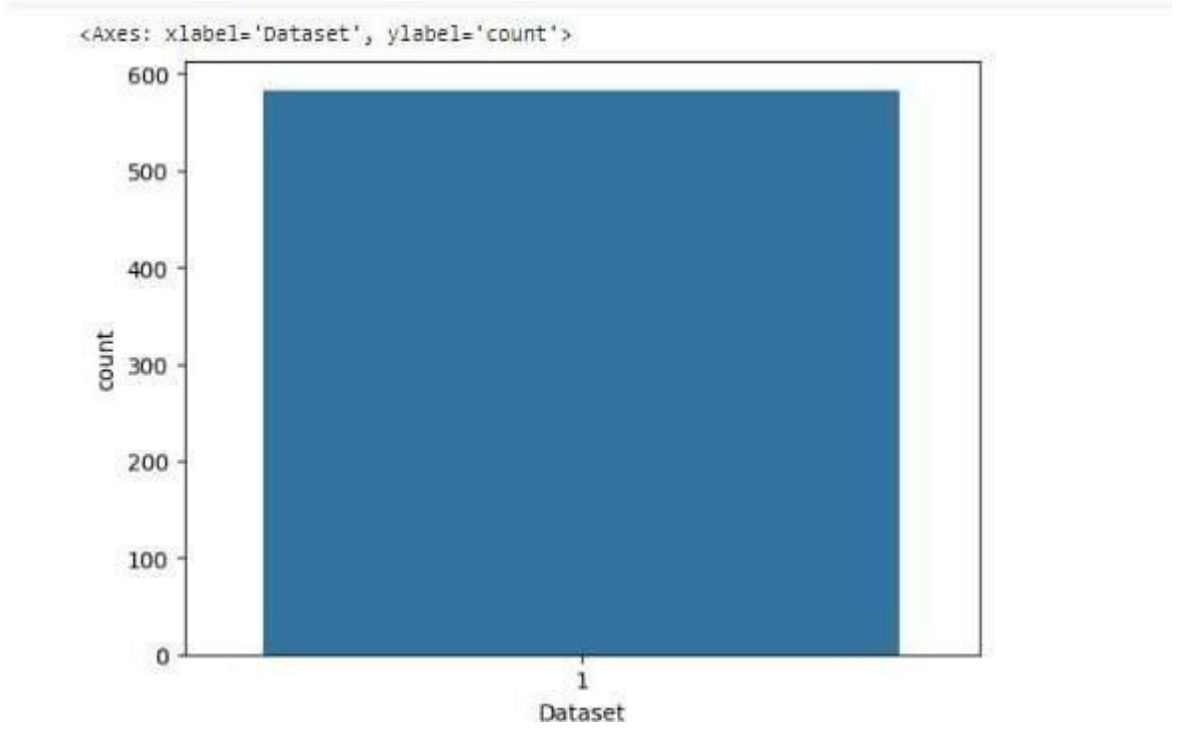
sns.distplot(data['Age'])

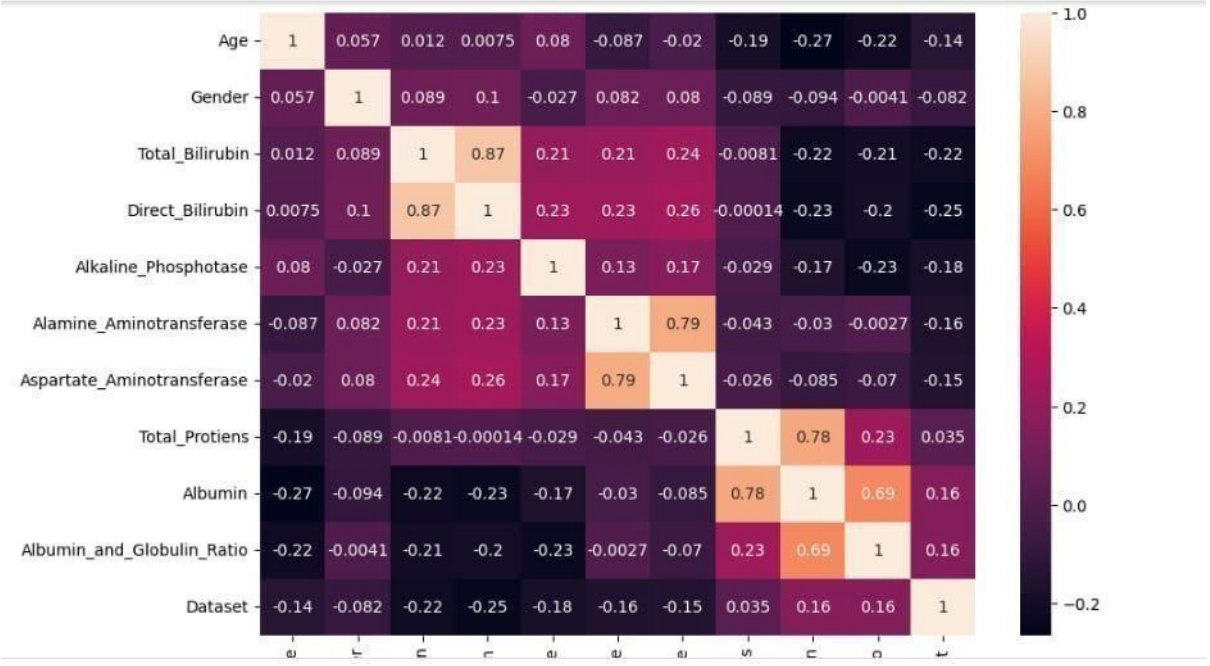
plt.title('Age Distribution Graph')

plt.show()

```
]
sns.distplot(data['Age'])
```








```
sns.countplot(data['Dataset'],x=data['Gender'])

plt.figure(figsize=(10,7))

sns.heatmap(data.corr(),annot=True)
```

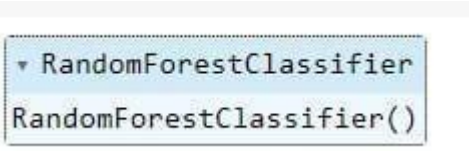
```
from sklearn.preprocessing import scale
x=pd.DataFrame (scale(x),columns=x.columns)
x_scaled.head()
```

	age	gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase
0	1.252098	-1.762281	-0.418878	-0.493964	-0.426715
1	1.066637	0.567446	1.225171	1.430423	1.682629
2	1.066637	0.567446	0.644919	0.931508	0.821588
3	0.819356	0.567446	-0.370523	-0.387054	-0.447314
4	1.684839	0.567446	0.096902	0.183135	-0.393756

```
x=data.iloc[:,:-1]
y=data.Dataset

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)

from imblearn.over_sampling import SMOTE
smote = SMOTE()
y_train.value_counts()
  1  329
  2  137
Name: Dataset, dtype: int64
x_train_smote, y_train_smote = smote.fit_resample(x_train,y_train)
y_train_smote.value_counts()
1  329
2  329
Name: Dataset, dtype: int64
```

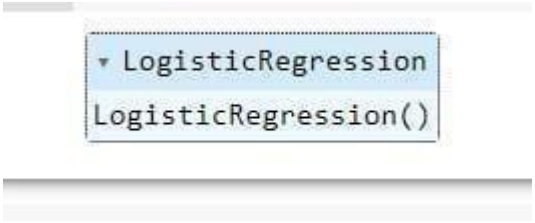
Milestone 4:

```
from sklearn.ensemble import RandomForestClassifier

RFmodel=RandomForestClassifier()

RFmodel.fit(x_train,y_train)
```




```
RFpred=RFmodel.predict(x_test)

RFaccuracy=accuracy_score(RFpred,y_test)

RFaccuracy

0.7521367521367521

RFcm=confusion_matrix(RFpred,y_test)

RFcm

array([[75, 17],

       [12, 13]])

from sklearn.neighbors import KNeighborsClassifier

KNN=KNeighborsClassifier()

KNN.fit(x_train,y_train)

KNNpred=KNN.predict(x_test)

KNNaccuracy=accuracy_score(KNNpred,y_test)

KNNaccuracy

0.6837606837606838

KNNcm=confusion_matrix(KNNpred,y_test)

KNNcm

array([[69, 19],

       [18, 11]])

from sklearn.tree import DecisionTreeClassifier
```

```
DTC=DecisionTreeClassifier()
DTC.fit(x_train,y_train)
```




```
DTCpred=DTC.predict(x_test)
DTCaccuracy=accuracy_score(DTCpred,y_test)
DTCaccuracy
0.717948717948718
DTCcm=confusion_matrix(DTCpred,y_test)
DTCcm
array([[66, 12],
       [21, 18]])
from sklearn.linear_model import LogisticRegression
LR=LogisticRegression()

LRpred=LR.predict(x_test)
LRaccuracy=accuracy_score(LRpred,y_test)
LRaccuracy
0.7435897435897436
LRcm=confusion_matrix(LRpred,y_test)
LRcm
array([[82, 25],
       [ 5,  5]])
import tensorflow.keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
classifier = Sequential()

classifier.add(Dense(units=100,activation='relu',input_dim=10))

classifier.add(Dense(units=50,activation='relu'))

classifier.add(Dense(units=1,activation='sigmoid'))

classifier.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])

RFmodel_history=classifier.fit(x_train,y_train,batch_size=100,validation_split=0.2,epochs=100)

Epoch 1/100

4/4 [==============================] - 0s 80ms/step - loss: -658367.8125 - accuracy: 0.7016 -
val_loss:
-593204.0625 - val_accuracy: 0.7234

Epoch 2/100

4/4 [==============================] - 0s 37ms/step - loss: -667815.0000 - accuracy: 0.7016 -
val_loss:
-601997.3125 - val_accuracy: 0.7234

Epoch 3/100

4/4 [==============================] - 0s 53ms/step - loss: -677944.8750 - accuracy: 0.7016 -
val_loss:
-610722.8125 - val_accuracy: 0.7234

Epoch 4/100

4/4 [==============================] - 0s 51ms/step - loss: -687855.5625 - accuracy: 0.7016 -
val_loss:
-619532.7500 - val_accuracy: 0.7234

Epoch 5/100

4/4 [==============================] - 0s 43ms/step - loss: -697299.9375 - accuracy: 0.7016 -
val_loss:
-628574.0000 - val_accuracy: 0.7234

Epoch 6/100

4/4 [==============================] - 0s 41ms/step - loss: -707700.5000 - accuracy: 0.7016 -
val_loss:
-637505.2500 - val_accuracy: 0.7234

Epoch 7/100
```

4/4 [==============================] - 0s 41ms/step - loss: -717873.1875 - accuracy: 0.7016 - val_loss: -646466.5625 - val_accuracy: 0.7234

Epoch 8/100

4/4 [==============================] - 0s 52ms/step - loss: -727852.1250 - accuracy: 0.7016 - val_loss: -655572.5625 - val_accuracy: 0.7234

Epoch 9/100

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 - val_loss: - val_accuracy: 0.7234

[==============================] - 0s - loss: - accuracy: 0.7016 - 29ms/step -738061.7500 -664791.8125

Epoch 10/100

4/4 24ms/step -748137.7500 val_loss: -674201.8125 - val_accuracy: 0.7234

Epoch 11/100

4/4 [==============================] - 0s 37ms/step - loss: -759247.3125 - accuracy: 0.7016 - val_loss: -683482.2500 - val_accuracy: 0.7234

Epoch 12/100

4/4 [==============================] - 0s 26ms/step - loss: -769670.8750 - accuracy: 0.7016 - val_loss: -692927.8125 - val_accuracy: 0.7234

Epoch 13/100

4/4 [==============================] - 0s 19ms/step - loss: -780058.7500 - accuracy: 0.7016 - val_loss: -702565.5625 - val_accuracy: 0.7234

Epoch 14/100

4/4 [==============================] - 0s 36ms/step - loss: -790572.3125 - accuracy: 0.7016 - val_loss: -712313.8125 - val_accuracy: 0.7234

Epoch 15/100

4/4 [==============================] - 0s 25ms/step - loss: -801802.5000 - accuracy: 0.7016 - val_loss: -721993.8125 - val_accuracy: 0.7234

Epoch 16/100

- 0s - loss: - accuracy: 0.7016 -

Epoch

4/4 [==========] - 0s - loss: - accuracy: 0.7016 - val_loss: - val_accuracy: 0.7234

[==========] - 0s - loss: - accuracy: 0.7016 -
4/4 [==========] - 0s 30ms/step - loss: -812847.3750 - accuracy: 0.7016 - val_loss: -731739.5625 - val_accuracy: 0.7234

Epoch 17/100

4/4 [==========] - 0s 42ms/step - loss: -823909.3125 - accuracy: 0.7016 - val_loss: -741573.6875 - val_accuracy: 0.7234

Epoch 18/100

4/4 [==========] - 0s 43ms/step - loss: -834615.9375 - accuracy: 0.7016 - val_loss: -751650.8125 - val_accuracy: 0.7234

Epoch 19/100

4/4 [==========] 20ms/step -846041.3125 val_loss: -761723.0000 - val_accuracy: 0.7234

20/100

27ms/step -857278.9375
-771933.6875

Epoch 21/100

4/4 20ms/step -868970.2500 val_loss: -782164.2500 - val_accuracy: 0.7234

Epoch 22/100

4/4 [==========] - 0s 33ms/step - loss: -880613.0000 - accuracy: 0.7016 - val_loss: -792485.2500 - val_accuracy: 0.7234

Epoch 23/100

4/4 [==========] - 0s 67ms/step - loss: -891858.9375 - accuracy: 0.7016 - val_loss: -803042.0625 - val_accuracy: 0.7234

Epoch 24/100

4/4 [==========] - 0s 57ms/step - loss: -903902.8750 - accuracy: 0.7016 - val_loss: -813523.7500 - val_accuracy: 0.7234

- 0s - loss: - accuracy: 0.7016 -

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 - val_loss: - val_accuracy: 0.7234

[==============================] - 0s - loss: - accuracy: 0.7016 -

Epoch 25/100

4/4 [==============================] - 0s 56ms/step - loss: -915529.8750 - accuracy: 0.7016 - val_loss:
-824129.3750 - val_accuracy: 0.7234

Epoch 26/100

4/4 [==============================] - 0s 55ms/step - loss: -927790.4375 - accuracy: 0.7016 - val_loss:
-834718.0625 - val_accuracy: 0.7234

Epoch 27/100

4/4 [==============================] - 0s 53ms/step - loss: -939390.9375 - accuracy: 0.7016 - val_loss:
-845578.3750 - val_accuracy: 0.7234

Epoch 28/100

4/4 [==============================] - 0s 20ms/step - loss: -951746.6875 - accuracy: 0.7016 - val_loss:
-856461.6250 - val_accuracy: 0.7234

Epoch 29/100

4/4 [==============================] - 0s 13ms/step - loss: -964017.0625 - accuracy: 0.7016 - val_loss:
-867335.2500 - val_accuracy: 0.7234

Epoch 30/100

4/4 [==============================] 14ms/step -976092.1875 val_loss:
-878283.5625 - val_accuracy: 0.7234

31/100

18ms/step -988350.1875

val_loss:

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 - val_loss: - val_accuracy: 0.7234

[==============================] - 0s - loss: - accuracy: 0.7016 - -889272.2500

Epoch 32/100

4/4 16ms/step -1000706.9375 val_loss: -900362.7500 - val_accuracy: 0.7234

Epoch 33/100

4/4 [==============================] - 0s 19ms/step - loss: -1013206.9375 - accuracy: 0.7016 val_loss: -911565.0000 - val_accuracy: 0.7234

Epoch 34/100

4/4 [==============================] - 0s 21ms/step - loss: -1026137.4375 - accuracy: 0.7016 val_loss: -922823.0000 - val_accuracy: 0.7234

Epoch 35/100

4/4 [==============================] - 0s 19ms/step - loss: -1037931.1875 - accuracy: 0.7016 val_loss: -934481.6250 - val_accuracy: 0.7234

Epoch 36/100

4/4 [==============================] - 0s 14ms/step - loss: -1051233.7500 - accuracy: 0.7016 val_loss: -945996.4375 - val_accuracy: 0.7234

Epoch 37/100

4/4 [==============================] - 0s 19ms/step - loss: -1064053.8750 - accuracy: 0.7016 val_loss: -957543.7500 - val_accuracy: 0.7234

Epoch 38/100

4/4 [==============================] - 0s 19ms/step - loss: -1077281.3750 - accuracy: 0.7016 val_loss: -969030.5625 - val_accuracy: 0.7234

Epoch 39/100

4/4 [==============================] - 0s 19ms/step - loss: -1090125.3750 - accuracy: 0.7016 val_loss: -980694.3125 - val_accuracy: 0.7234

Epoch 40/100

- 0s - loss: - accuracy: 0.7016 -

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 - val_loss:
- val_accuracy: 0.7234

[==============================] - 0s - loss: - accuracy: 0.7016 -
4/4 [==============================] - 0s 25ms/step - loss: -1103089.1250 - accuracy: 0.7016 -
-992526.8125 - val_accuracy: 0.7234

41/100

19ms/step -1116665.6250
-1004365.9375

42/100

val_loss:

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy:
0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

14ms/step -1129659.7500 -1016444.1875

43/100

13ms/step -1143343.8750 -1028518.8125

Epoch 44/100

4/4 [==============================] - 0s 20ms/step - loss: -1156914.8750 - accuracy: 0.7016 val_loss: -1040690.8750 - val_accuracy: 0.7234

Epoch 45/100

4/4 [==============================] - 0s 17ms/step - loss: -1170839.7500 - accuracy: 0.7016 val_loss: -1052845.2500 - val_accuracy: 0.7234

Epoch 46/100

4/4 [==============================] - 0s 20ms/step - loss: -1184115.1250 - accuracy: 0.7016 val_loss: -1065299.1250 - val_accuracy: 0.7234

Epoch 47/100

4/4 [==============================] - 0s 19ms/step - loss: -1198001.7500 - accuracy: 0.7016 val_loss: -1077800.1250 - val_accuracy: 0.7234

Epoch 48/100

4/4 [==============================] - 0s 19ms/step - loss: -1212449.0000 - accuracy: 0.7016 val_loss: -1090211.3750 - val_accuracy: 0.7234

Epoch 49/100

4/4 [==============================] - 0s 19ms/step - loss: -1226363.3750 - accuracy: 0.7016 val_loss: -1102746.2500 - val_accuracy: 0.7234

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234
Epoch 50/100

4/4 [==============================] - 0s 20ms/step - loss: -1240148.2500 - accuracy: 0.7016 val_loss: -1115452.1250 - val_accuracy: 0.7234

Epoch 51/100

4/4 [==============================] - 0s 15ms/step - loss: -1254514.6250 - accuracy: 0.7016 - -1128213.5000

52/100

19ms/step -1268405.0000 -1141175.3750

53/100

13ms/step -1283378.7500 -1153975.8750

54/100

13ms/step -1297842.6250 -1166885.2500

Epoch 55/100

4/4 [==============================] - 0s 19ms/step - loss: -1313219.6250 - accuracy: 0.7016 val_loss: -1179737.3750 - val_accuracy: 0.7234

Epoch 56/100

4/4 [==============================] - 0s 14ms/step - loss: -1327222.6250 - accuracy: 0.7016 val_loss: -1193090.7500 - val_accuracy: 0.7234

Epoch 57/100

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

4/4 [==============================] - 0s 19ms/step - loss: -1341815.2500 - accuracy: 0.7016 val_loss: -1206648.1250 - val_accuracy: 0.7234

Epoch 58/100

4/4 [==============================] - 0s 19ms/step - loss: -1356829.8750 - accuracy: 0.7016 val_loss: -1220226.7500 - val_accuracy: 0.7234

Epoch 59/100

4/4 [==============================] - 0s 14ms/step - loss: -1372226.3750 - accuracy: 0.7016 val_loss: -1233714.5000 - val_accuracy: 0.7234

Epoch 60/100

4/4 [==============================] - 0s 13ms/step - loss: -1387510.3750 - accuracy: 0.7016 val_loss: -1247228.3750 - val_accuracy: 0.7234

Epoch 61/100

4/4 [==============================] - 0s 14ms/step - loss: -1402555.0000 - accuracy: 0.7016 val_loss: -1260871.1250 - val_accuracy: 0.7234

Epoch 62/100

4/4 [==============================] - 0s 23ms/step - loss: -1417701.6250 - accuracy: 0.7016 -

-1274690.7500

63/100

14ms/step -1433544.1250

-1288476.2500

64/100

19ms/step -1449153.0000

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234
-1302399.7500

65/100

13ms/step -1464224.3750
-1316549.5000

Epoch 66/100

4/4 [==============================] - 0s 12ms/step - loss: -1479801.0000 - accuracy: 0.7016 val_loss: -1330783.8750 - val_accuracy: 0.7234

Epoch 67/100

4/4 [==============================] - 0s 13ms/step - loss: -1496706.6250 - accuracy: 0.7016 val_loss: -1344791.1250 - val_accuracy: 0.7234

Epoch 68/100

4/4 [==============================] - 0s 14ms/step - loss: -1512144.8750 - accuracy: 0.7016 val_loss: -1359089.2500 - val_accuracy: 0.7234

Epoch 69/100

4/4 [==============================] - 0s 14ms/step - loss: -1528065.8750 - accuracy: 0.7016 val_loss: -1373541.0000 - val_accuracy: 0.7234

Epoch 70/100

4/4 [==============================] - 0s 19ms/step - loss: -1544517.3750 - accuracy: 0.7016 val_loss: -1388008.6250 - val_accuracy: 0.7234

Epoch 71/100

4/4 [==============================] - 0s 13ms/step - loss: -1561332.0000 - accuracy: 0.7016 val_loss: -1402453.5000 - val_accuracy: 0.7234

Epoch 72/100

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

4/4 [==============================] - 0s 19ms/step - loss: -1577190.0000 - accuracy: 0.7016 val_loss: -1417268.3750 - val_accuracy: 0.7234

Epoch 73/100

4/4 [==============================] - 0s 14ms/step - loss: -1594071.0000 - accuracy: 0.7016 - -1432073.5000

74/100

13ms/step -1610046.1250 -1447196.8750

75/100

20ms/step -1627160.6250 -1462295.5000

76/100

14ms/step -1644354.2500 -1477444.3750

Epoch 77/100

4/4 [==============================] - 0s 19ms/step - loss: -1661460.2500 - accuracy: 0.7016 val_loss: -1492718.0000 - val_accuracy: 0.7234

Epoch 78/100

4/4 [==============================] - 0s 19ms/step - loss: -1679104.5000 - accuracy: 0.7016 val_loss: -1508000.1250 - val_accuracy: 0.7234

Epoch 79/100

4/4 [==============================] - 0s 14ms/step - loss: -1695154.1250 - accuracy: 0.7016 val_loss: -1523874.7500 - val_accuracy: 0.7234

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch 80/100

4/4 [==============================] - 0s 19ms/step - loss: -1713723.8750 - accuracy: 0.7016 val_loss: -1539401.7500 - val_accuracy: 0.7234

Epoch 81/100

4/4 [==============================] - 0s 20ms/step - loss: -1730503.8750 - accuracy: 0.7016 val_loss: -1555277.7500 - val_accuracy: 0.7234

Epoch 82/100

4/4 [==============================] - 0s 20ms/step - loss: -1748175.6250 - accuracy: 0.7016 val_loss: -1571173.1250 - val_accuracy: 0.7234

Epoch 83/100

4/4 [==============================] - 0s 22ms/step - loss: -1766677.6250 - accuracy: 0.7016 val_loss: -1586805.2500 - val_accuracy: 0.7234

Epoch 84/100

4/4 [==============================] - 0s 14ms/step - loss: -1784156.8750 - accuracy: 0.7016 -

-1602683.8750

85/100

14ms/step -1801669.3750

-1618731.8750

86/100

20ms/step -1819357.3750

-1634839.8750

87/100

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

20ms/step -1838195.5000 -1650720.6250

Epoch 88/100

4/4 [==============================] - 0s 14ms/step - loss: -1855447.3750 - accuracy: 0.7016 val_loss: -1667040.0000 - val_accuracy: 0.7234

Epoch 89/100

4/4 [==============================] - 0s 20ms/step - loss: -1874518.3750 - accuracy: 0.7016 val_loss: -1683222.5000 - val_accuracy: 0.7234

Epoch 90/100

4/4 [==============================] - 0s 18ms/step - loss: -1891730.3750 - accuracy: 0.7016 val_loss: -1699956.3750 - val_accuracy: 0.7234

Epoch 91/100

4/4 [==============================] - 0s 19ms/step - loss: -1910361.5000 - accuracy: 0.7016 val_loss: -1716632.1250 - val_accuracy: 0.7234

Epoch 92/100

4/4 [==============================] - 0s 20ms/step - loss: -1929668.5000 - accuracy: 0.7016 val_loss: -1733106.0000 - val_accuracy: 0.7234

Epoch 93/100

4/4 [==============================] - 0s 13ms/step - loss: -1948571.8750 - accuracy: 0.7016 val_loss: -1749683.3750 - val_accuracy: 0.7234

Epoch 94/100

4/4 [==============================] - 0s 14ms/step - loss: -1966975.2500 - accuracy: 0.7016 val_loss: -1766571.7500 - val_accuracy: 0.7234

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch 95/100

4/4 [==============================] - 0s 14ms/step - loss: -1985814.0000 - accuracy: 0.7016 - -1783625.3750

96/100

19ms/step -2004937.6250 -1800817.8750

97/100

val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

Epoch

4/4 [==============================] - 0s - loss: - accuracy: 0.7016 val_loss: - val_accuracy: 0.7234

13ms/step -2024367.1250 -1818068.7500

98/100

14ms/step -2043511.3750 -1835457.7500

Epoch 99/100

4/4 [==============================] - 0s 19ms/step - loss: -2062323.6250 - accuracy: 0.7016 val_loss: -1853025.7500 - val_accuracy: 0.7234

Epoch 100/100

4/4 [==============================] - 0s 19ms/step - loss: -2082436.8750 - accuracy: 0.7016 val_loss: -1870389.7500 - val_accuracy: 0.7234

DTC.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]])

array([2])

```
RFmodel.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]
```

]) array([1]) classifier.save("liver.h5") y_test

=(y_test>0.5) y_test

355 True

407 True

90 True

402 True

268 True

...

516 True

305 True

167 True

312 True

329 True

Name: Dataset, Length: 117, dtype: bool

def predict_exit(sample_value):

sample_value =np.array(sample_value)

sample_value =sample_value.reshape(1,-

1) sample_value =scale(sample_value)

return classifier.predict(sample_value)

sample_value=[[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]]

if predict_exit(sample_value)>0.5:

print('Prediction: Liver

Patient') else: print('Prediction:

Healthy')

1/1 [==============================] - 0s 105ms/step

Prediction: Liver Patient

1/1 [==============================] - 0s 24ms/step

Prediction: Liver Patient

Milestone 5:

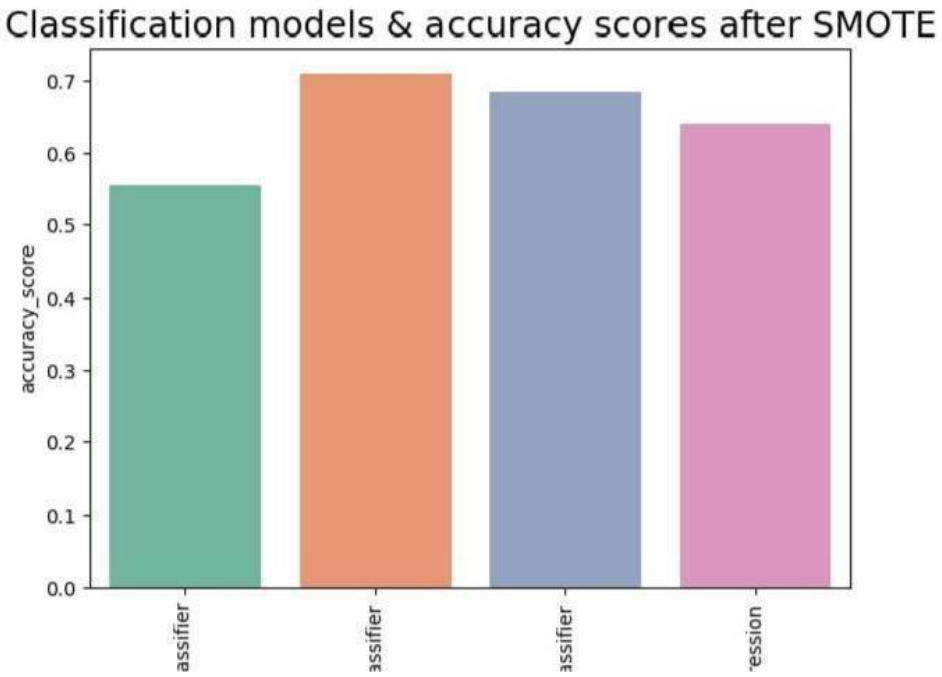
acc_smote=[['KNN Classifier',KNN],['RandomForestClassifier',RFaccuracy],['DecisionTreeClassifier',DTCaccuracy],['LogisticRegression',LRaccuracy]]

Liverpatient_pred=pd.DataFrame(acc_smote,columns=['classification models','accuracy_score'])

Liverpatient_pred

	classification models	accuracy_score
0	KNN Classifier	0.555556
1	RandomForestClassifier	0.709402
2	DecisionTreeClassifier	0.683761
3	LogisticRegression	0.641026

```
plt.figure(figsize=(7,5))
plt.xticks(rotation=90)
plt.title('Classification models & accuracy scores after SMOTE',fontsize=18)
sns.barplot(x="classification models",y="accuracy_score",data=Liverpatient_pred,palette="Set2")
```




```
from sklearn.ensemble import ExtraTreesClassifier
model1=ExtraTreesClassifier()
model1.fit(x,y)
```

model1.feature_importances

array([0.13287739, 0.01991026, 0.09463777, 0.08566328, 0.14577484,

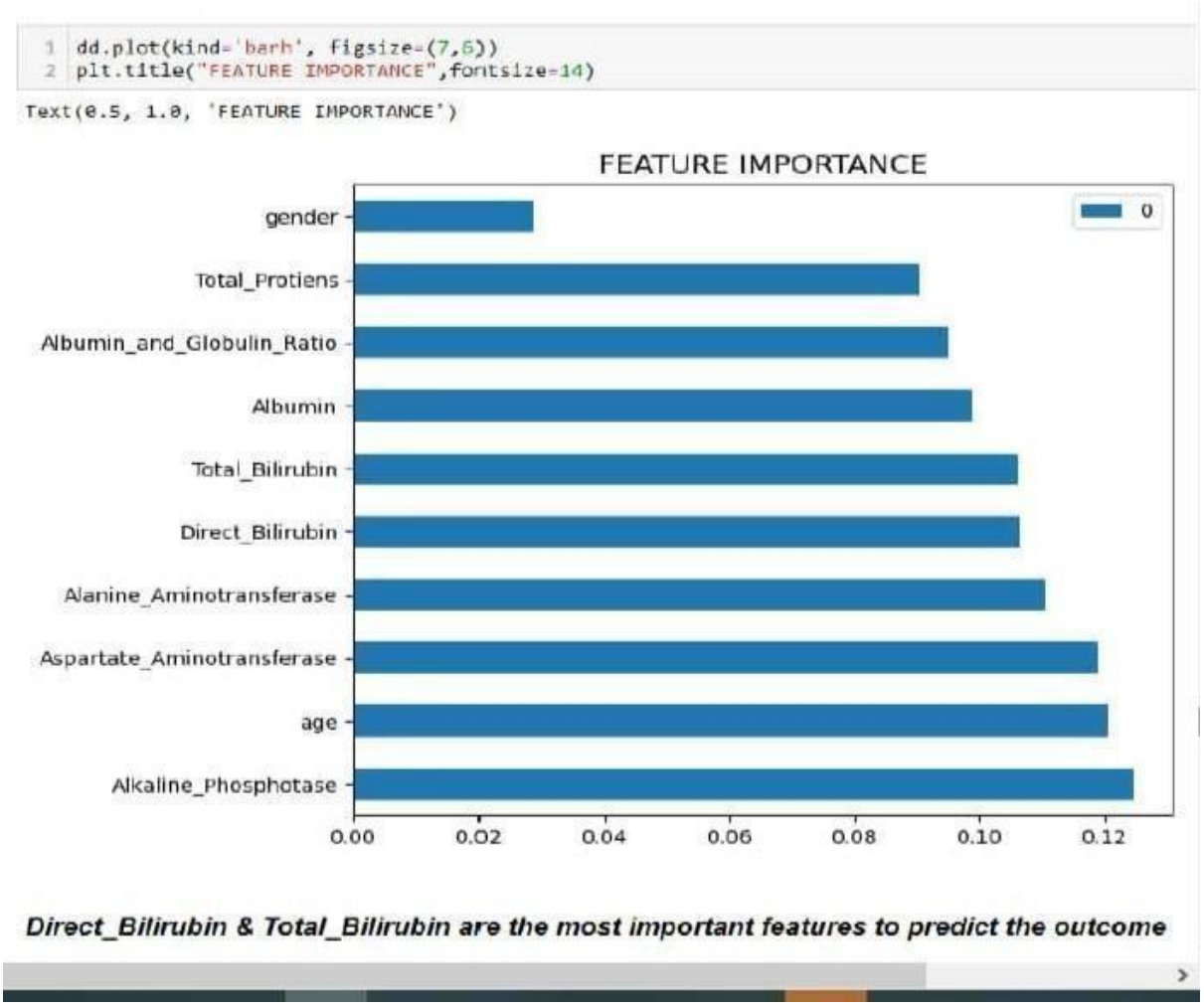
0.13219089, 0.12830574, 0.09378994, 0.09210753, 0.07474237])

dd.pd.DataFrame(model1.feature_importances_,index=x.columns).sort_values(0,ascending=False)

dd

data.plot(kind='barh',figsize=(7,6))

plt.title("FEATURE IMPORTANCE",fontsize=14)



import joblib

joblib.dump(RFmodel,'liver_analysis_1.pkl')

['liver_analysis_1.pkl']

data.plot(kind='barh',figsize=(6,5))

plt.title("FEATURE IMPORTANCE",fontsize=14)

Milestone6

import numpy as np

import pickle import

os

```
app=Flask(_name____
) @app.route('/') def
home():
  return
render_template('home.html')
@app.route('/predict') def index():
return render_template("index.html")
@app.route('/data_predict',methods=['POST'])
def predict():
    age = request.form['age']
    gender=request.form['gender'
    ] tb = request.form['tb'] db =
    request.form['db'] ap =
    request.form['ap'] aa1 =
    request.form['aa1']

    aa2 = request.form['aa2'] tp = request.form['tp'] a = request.form['a'] agr = request.form['agr']
    data =
    [[float(age),float(gender),float(db),float(ap),float(aa1),float(aa2),float(tp),float(a),float(agr)]
    model=pickle.load(open(os.path.join('c:Users/91630/Desktop/liver patient/Liver Patient Analysis/Flask
```

app,pkl_objects','liver_analysis_1.pkl'),'rb'

```
    )) prediction= model.predict(data)[0] if

    (prediction == 1):

      return render_template('noChance.html',prediction='You have a liver desease problem,You must and should consult a doctor.Take care')

    else:

      return render_template('chance.html', prediction='You dont have a liver desease

problem') if __name___=='_main_': app.run(debug=True)
```