

NAANMUDHALVAN PROJECT

**A REVIEW OF LIVER PATIENT ANALYSIS METHODS USING
MACHINE LEARNING**

**BACHELOR OF SCIENCE IN COMPUTER SCIENCE
TO THE**

**THIRUVALLUVAR UNIVERSITY, SERKKADU,
VELLORE-632115**

BY

DINESH G -- 35620U18007

LOKESH P -- 35620U18019

SRINIVASAN J -- 35620U18044

YUVARAJ S -- 35620U18053



APRIL-2023

GOVERNMENT ARTS AND SCIENCE COLLEGE

ARAKKONAM - 631051

(AFFILIATED TO THIRUVALLUVAR UNIVERSITY)

GUIDED AND HEAD OF DEPARTMENT

Dr.S.Selvakani

S.NO	CONTENT
1.	INTRODUCTION
2.	<p>MILESTONE 1: DEFINE PROBLEM /PROBLEM UNDERSTANDING</p> <ul style="list-style-type: none">*Specify the business problem*Business requirements*Literature Survey*Social or Business Impact
3.	<p>MILESTONE 2: DATA COLLECTION & PREPARATION</p> <ul style="list-style-type: none">*Collect The Dataset* Importing the libraries* Read The Dataset* Data Preparation* Handling MissingValues* Handling Categorical Values
4.	<p>MILESTONE 3:EXPLORATORY DATA ANALYSIS</p> <ul style="list-style-type: none">* Descriptive Statistical* Visual Analysis* Univariate Analysis* Bivariate Analysis* Multivariate Analysis* Handling Imbalance Data

5.	<p>MILESTONE 4:MODEL BUILDING</p> <ul style="list-style-type: none">*Training The Model In Multiple*Random Forest Model*Decision Tree Model*KNN Model*Logistic Regression Model*ANN Model*Testing The Model
6.	<p>MILESTONE 5: PERFORMANCE TESTING & HYPER PARAMETER TUNING</p> <ul style="list-style-type: none">*Testing Model With Multiple Metrics*Compare The Model*Identifying Important Features
7.	<p>MILESTONE 5: MODEL DEPLOYMENT</p> <ul style="list-style-type: none">*Save the best model*Integrate with Web Framework*Building Html Pages*Build Python Code*Run The Web Application

8.	<p>MILESTONE 6: PROJECT DEMONSTRATION & DOCUMENTATION</p> <ul style="list-style-type: none">*Record Explanation Video For Project End To End Solution*Project Documentation-Step By Step Project Development Procedure
----	---

INTRODUCTION

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.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirement

Patients with Liver disease have been continuously increasing because of excessive consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles and drugs and other factors. This dataset was used to evaluate prediction algorithms in an effort to reduce burden on doctors. Use these patient records to build a prediction model that will predict which patients have liver disease and which ones do not.

Activity 3: Literature Survey (Student Will Write)

With a growing trend of sedentary and lack of physical activities, diseases related to liver have become a common encounter nowadays. In rural areas the intensity is still manageable, but in urban areas, and especially metropolitan areas the liver disease is a very common sighting nowadays. Problems with liver patients are not easily discovered in an early stage as it will be functioning normally even when it is partially damaged. An early diagnosis of liver problems will increase patients survival rate. There are various algorithms that have been used with varying levels of success. Logistic regression, decision tree, random forest, and neural networks have all been used and have been able to accurately predict liver disease.

Activity 4: Social or Business Impact.

Social Impact:- Today almost everybody above the age of 12 years has smartphones with them, and so we can incorporate these solutions into an android app or ios app. Also it can be incorporated into a website and these app and website will be highly beneficial for a large section of society.

Business Model/Impact:- Its now more feasible Blood test centers to give the result. As for this model user don't need to have any deep knowledge of medical science and liver diseases. User need to do pass the details being asked, which are already present in the blood test report(some like age, gender are already known) and then user will get the results of prediction

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg:
kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/uciml/indian-liver-patient-records>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

```
1
2 import pandas as pd
3 import numpy as np
4 import seaborn as sns
5 import matplotlib.pyplot as plt
6 from matplotlib import rcParams
7 from scipy import stats
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
#import the dataset from specified location
data = pd.read_csv('E:/Datascience/Datasets/indian_liver_patient.csv')
```

```
# showing the data from top 5
data.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferas
0	65	Female	0.7	0.1	187	16	18
1	62	Male	10.9	5.5	699	64	100
2	62	Male	7.3	4.1	490	60	68
3	58	Male	1.0	0.4	182	14	20
4	72	Male	3.9	2.0	195	27	59

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

- Let's find the shape of our dataset first. To find the shape of our data, the `df.shape` method is used. To find the data type, `df.info()` function is used.

```
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
```

- For checking the null values, `df.isnull()` function is used. To sum those null values we use `.sum()` function.

```
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
```

We can see that there are null values in the Albumin_and_Globulin_Ration Column.

Let us check how many numbers of null records present in the

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

Age                0
Gender             0
Total_Bilirubin    0
Direct_Bilirubin   0
Alkaline_Phosphatase 0
Alamine_Aminotransferase 0
Aspartate_Aminotransferase 0
Total_Protiens     0
Albumin            0
Albumin_and_Globulin_Ratio 4
Dataset            0
dtype: int64
```

Closing Value column using sum() function.

- From the above code of analysis, we can infer that columns such as Albumin and Globulin Ratio is having the missing values, we need to treat them in a required way.

```
#checking for the missing data after cleaning data
data['Albumin_and_Globulin_Ratio'] = data.fillna(data['Albumin_and_Globulin_Ratio'].mode()[0])
data.isnull().sum()

Age                0
Gender             0
Total_Bilirubin    0
Direct_Bilirubin   0
Alkaline_Phosphatase 0
Alamine_Aminotransferase 0
Aspartate_Aminotransferase 0
Total_Protiens     0
Albumin            0
Albumin_and_Globulin_Ratio 0
Dataset            0
dtype: int64
```

- We will fill in the missing values in the numeric data type using the meanvalue of that particular column and categorical data type using the most repeated value.

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.

- In our project, for Gender, encoding is done.

```
1 from sklearn.preprocessing import LabelEncoder
2 lc = LabelEncoder()
3 data['gender']= lc.fit_transform(data['gender'])
```

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the

2
3 data.describe()

	age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alanine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	6.483190	3.141852
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451	0.795519
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000	0.900000
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000	2.600000
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000	3.100000
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000	3.800000
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	9.600000	5.500000

statistical process. Here pandas has a worthy function called describe.

With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

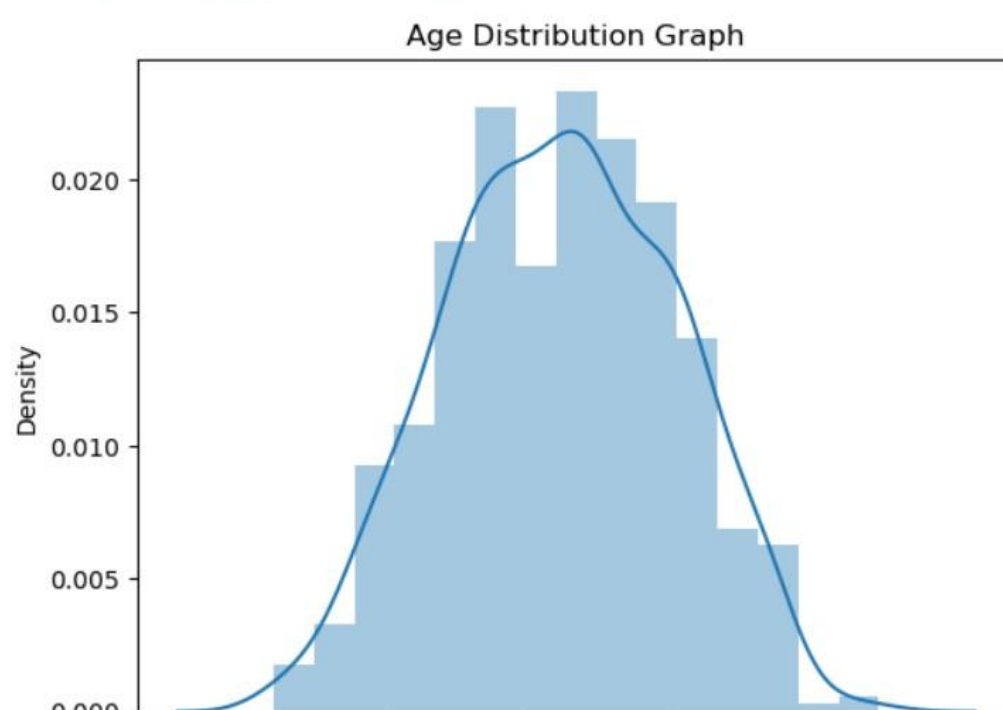
Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

- The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
1 sns.distplot(data['age'])
2 plt.title('Age Distribution Graph')
3 plt.show()
4
```

D:\Anaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is deprecated and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function) or `histplot` (an axes-level function for histograms).
warnings.warn(msg, FutureWarning)



- In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features.

Countplot:-

A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for `barplot()`, so you can compare counts across nested

variables.

Activity 2.2: Bivariate analysis

```
1 sns.countplot(data['outcome'], hue=data['gender'])
2
```

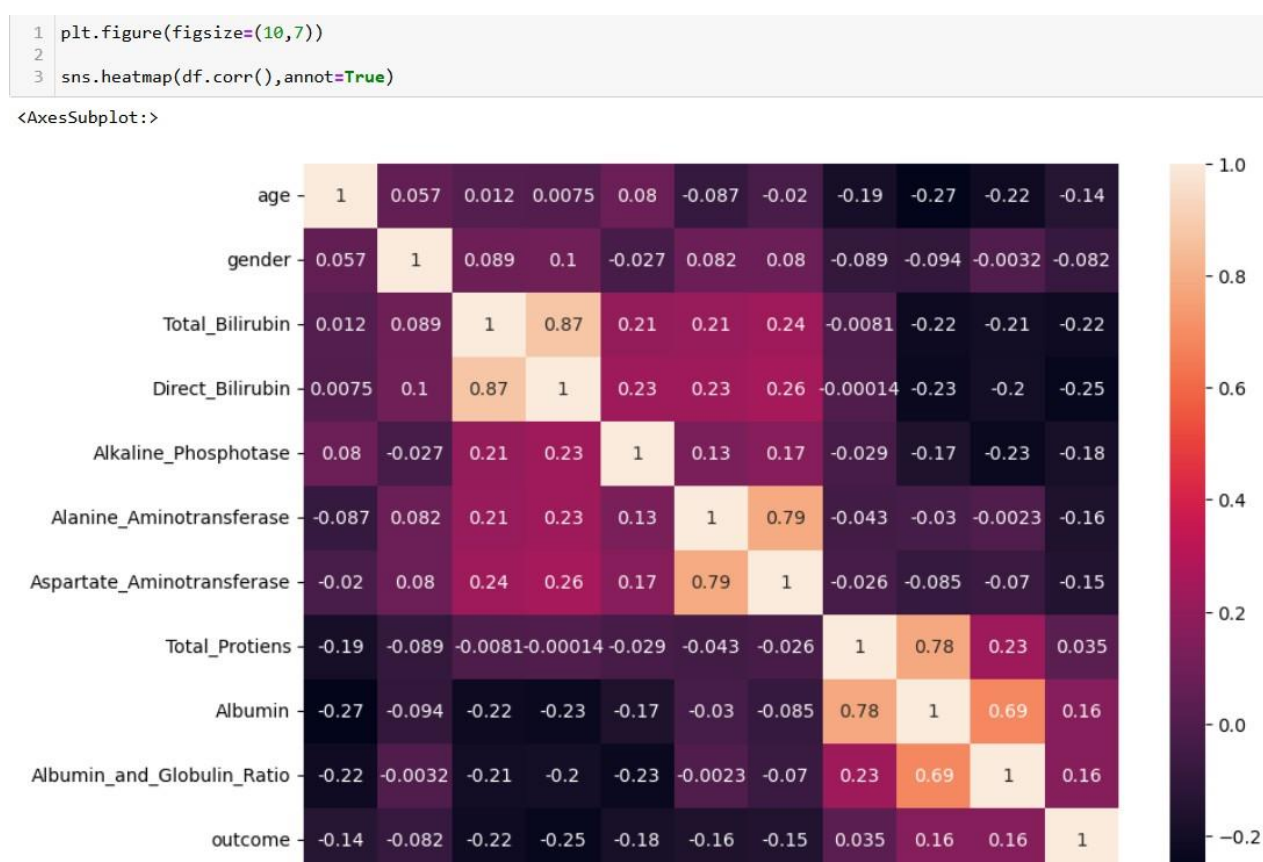
```
D:\Anaconda\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pas
ersion 0.12, the only valid positional argument will be `data`, and passing
sult in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='outcome', ylabel='count'>
```

From the graph we can infer that , gender and outcome is a categorical variables with 2 categories , from gender column we can infer that 1-category is having more weightage than category-0, and outcome with 0,it means healthy is a underclass when compared with category -1, which means liver patient .

Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between



multiple features. Here we have used a heat plot from the seaborn package.

Now, the code would be normalising the data by scaling it to have a similar range of values, and then splitting that data into a training set and a test set for training the model and testing its performance, respectively.

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept.

```
1 from sklearn.preprocessing import scale
2 X_scaled=pd.DataFrame (scale(X), columns=X.columns)

1 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

We will perform scaling only on the input values.Once the dataset is scaled, it will be converted into an array and we need to convert it back to a dataframe.

Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

```
1 X=data.iloc[:, :-1]
2 y=data.outcome
```

Here x and y variables are created. On x variable, df is passed with

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(X_scaled,y, test_size=0.2, random_state=42)
```

dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

Activity 2.4: Handling Imbalance Data

Data Balancing is one of the most important step, which need to be performed for classification models, because when we train our model on imbalanced dataset, we will get biased results, which means our model is able to predict only one class element

For balancing the data we are using the SMOTE Method.

SMOTE: Synthetic minority over sampling technique, which will create new synthetic data points for under class as per the requirements given by us using KNN method.

```
1 pip install imblearn

Requirement already satisfied: imblearn in d:\anaconda\lib\site-packages
Requirement already satisfied: imbalanced-learn in d:\anaconda\lib\site-
Requirement already satisfied: numpy>=1.17.3 in d:\anaconda\lib\site-pac
Requirement already satisfied: joblib>=1.1.1 in d:\anaconda\lib\site-pac
Requirement already satisfied: scikit-learn>=1.0.2 in d:\anaconda\lib\si
0.2)
Requirement already satisfied: scipy>=1.3.2 in d:\anaconda\lib\site-pack
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda\lib\s
2.0)
Note: you may need to restart the kernel to use updated packages.

1 from imblearn.over_sampling import SMOTE
2 smote = SMOTE()

1 y_train.value_counts()

1    329
0    137
Name: outcome, dtype: int64

1 X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

1 y_train_smote.value_counts()

1    329
0    329
Name: outcome, dtype: int64
```

From the above picture, we can infer that,previously our dataset had 329 class 1, and 132 class items, after applying smote technique on the dataset the size has become equal.

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.

Activity 1.1: Random Forest model

A function named RandomForestClassifier is imported and train and test data are passed as the parameters. Inside the function, RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
1 from sklearn.ensemble import RandomForestClassifier
2 model1=RandomForestClassifier()
3 model1.fit(X_train_smote, y_train_smote)
4 y_predict=model1.predict(X_test)
5 rfc1=accuracy_score(y_test,y_predict)
6 rfc1
7 pd.crosstab(y_test, y_predict)
8 print(classification_report(y_test, y_predict))
```

Activity 1.2: Decision tree model

A function named DecisionTreeClassifier is imported and train and test data are passed as the parameters. Inside the function, DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, a confusion matrix and classification report is done.

```
from sklearn.tree import DecisionTreeClassifier
model4=DecisionTreeClassifier()
model4.fit(X_train_smote, y_train_smote)
y_predict=model4.predict(X_test)
dtt1=accuracy_score(y_test,y_predict)
dtt1
pd.crosstab(y_test,y_predict)
print(classification_report(y_test, y_predict))
```

Activity 1.3: KNN model

A function named K KNeighborsClassifier is imported and train and test data are passed as the parameters. Inside the function, KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict()

function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```

from sklearn.neighbors import KNeighborsClassifier
model2=KNeighborsClassifier()
model2.fit(X_train_smote, y_train_smote)
y_predict = model2.predict(X_test)
knn1=(accuracy_score(y_test, y_predict))
knn1
pd.crosstab(y_test,y_predict)
print(classification_report(y_test, y_predict))

```

Activity 1.4: Logistic Regression model

A function named Logistic Regression is imported and train and test data are passed as the parameters. Inside the function, Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```

from sklearn.linear_model import LogisticRegression
model5=LogisticRegression()
model5.fit(X_train_smote, y_train_smote)
y_predict=model5.predict(X_test)
logi1=accuracy_score(y_test, y_predict)
logi1
pd.crosstab(y_test,y_predict)
print(classification_report(y_test, y_predict))

```

Activity 1.5: ANN model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20%

validation split, and 100 epochs.

```
1 import tensorflow.keras
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense
4
1 # Initialising the ANN
2 classifier = Sequential()
1 # Adding the input layer and the first hidden layer
2 classifier.add(Dense(units=100, activation='relu', input_dim=10))
1 # Adding the second hidden layer
2 classifier.add(Dense(units=50, activation='relu'))
1 # Adding the output layer
2 classifier.add(Dense(units=1, activation='sigmoid'))
1 # Compiling the ANN
2 classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
1 # Fitting the ANN to the Training set
2 model_history = classifier.fit(X_train, y_train, batch_size=100, validation_split=0.2, epochs=100)
Epoch 1/100
4/4 [=====] - 2s 133ms/step - loss: 0.6497 - accuracy: 0.6532 - val_loss: 0.6394 - val_accuracy: 0.7234
Epoch 2/100
4/4 [=====] - 0s 21ms/step - loss: 0.6112 - accuracy: 0.7070 - val_loss: 0.6062 - val_accuracy: 0.7234
Epoch 3/100
4/4 [=====] - 0s 20ms/step - loss: 0.5901 - accuracy: 0.7016 - val_loss: 0.5800 - val_accuracy: 0.7234
Epoch 4/100
4/4 [=====] - 0s 20ms/step - loss: 0.5743 - accuracy: 0.7016 - val_loss: 0.5592 - val_accuracy: 0.7234
Epoch 5/100
4/4 [=====] - 0s 21ms/step - loss: 0.5619 - accuracy: 0.7016 - val_loss: 0.5437 - val_accuracy: 0.7234
```

Activity 2: Testing the model

```
1 #Age→Gender→Total_Bilirubin→Direct_Bilirubin→Alkaline_Phosphatase→Alanin_Aminotransferase→Asparate_Aminotransferase
2 model1.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]]) : 0.77
D:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
warnings.warn(
array([1], dtype=int64) : 0.76
1 #Age→Gender→Total_Bilirubin→Direct_Bilirubin→Alkaline_Phosphatase→Alanin_Aminotransferase→Asparate_Aminotransferase
2 model1.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]]) : 0.77
D:\Anaconda\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
warnings.warn(
array([1], dtype=int64) : 0.77
```

In ANN we first have to save the model to test the inputs

```
1 classifier.save("liver.h5")

1 y_pred = classifier.predict(X_test)
4/4 [=====] - 0s 2ms/step

1 y_pred

y_pred = (y_pred > 0.5)
y_pred
y([[ True],
 [ True],
 [ True],
 [ True]])
```

This code defines a function named "predict_exit" which takes in a sample_value as an input. The function then converts the input sample_value from a list to a numpy array. It reshapes the sample_value array as it contains only one record. Then, it applies feature scaling to the reshaped sample_value array using a scaler object 'scale' that should have been previously defined and fitted. Finally, the function returns the prediction of the classifier on the scaled sample_value.

```
1 def predict_exit(sample_value):
2
3     # Convert list to numpy array
4     sample_value = np.array(sample_value)
5
6     # Reshape because sample_value contains only 1 record
7     sample_value = sample_value.reshape(1, -1)
8
9     # Feature Scaling
10    sample_value = scale(sample_value)
11
12    return classifier.predict(sample_value)

1 #Age→Gender→Total_Bilirubin→Direct_Bilirubin→Alkaline_Phosphatase→,
2 sample_value = [[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]]
3 if predict_exit(sample_value)>0.5:
4     print('Prediction: Liver Patient')
5 else:
6     print('Prediction: Healthy ')

1/1 [=====] - 0s 105ms/step
Prediction: Liver Patient
```


Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using accuracy, score to compare between models

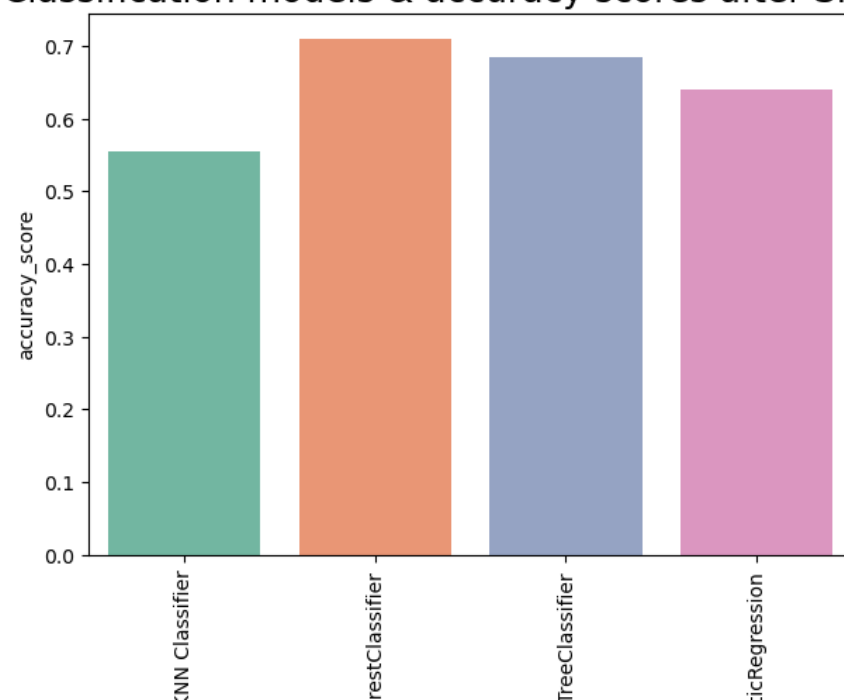
Activity 1.1: Compare the model

For comparing the above four models, the Accuracy function is defined.

```
1 plt.figure(figsize=(7,5))
2 plt.xticks(rotation=90)
3 plt.title('Classification models & accuracy scores after SMOTE',fontsize=18)
4 sns.barplot(x="classification models", y="accuracy_score", data=Liverpatient_pred,palette = "Set2") '1)
```

<AxesSubplot:title={'center':'Classification models & accuracy scores after SMOTE'}, xlabel='classification
l='accuracy_score'>

Classification models & accuracy scores after SMOTE

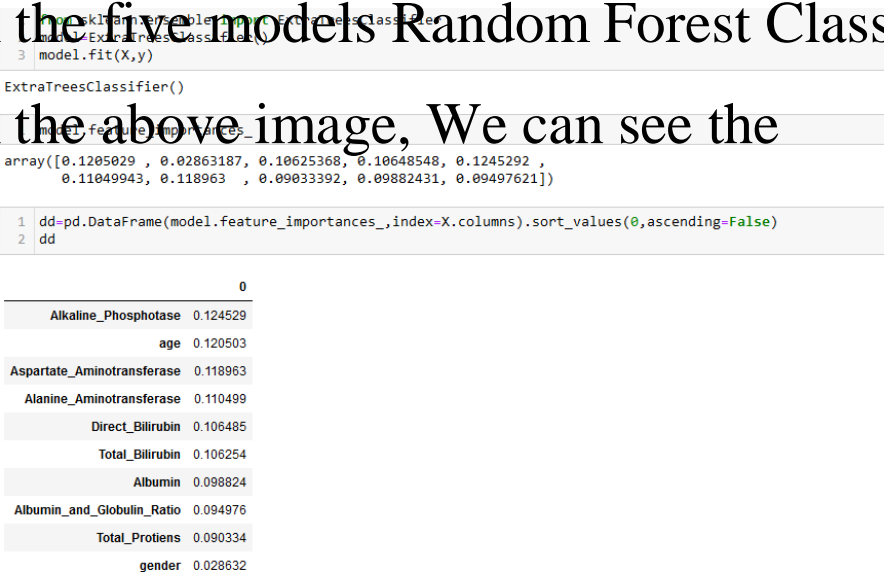


```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
model.fit(X,y)
```

After calling the function, the results of models are displayed as output.

From the five models Random Forest Classifier is performing well.

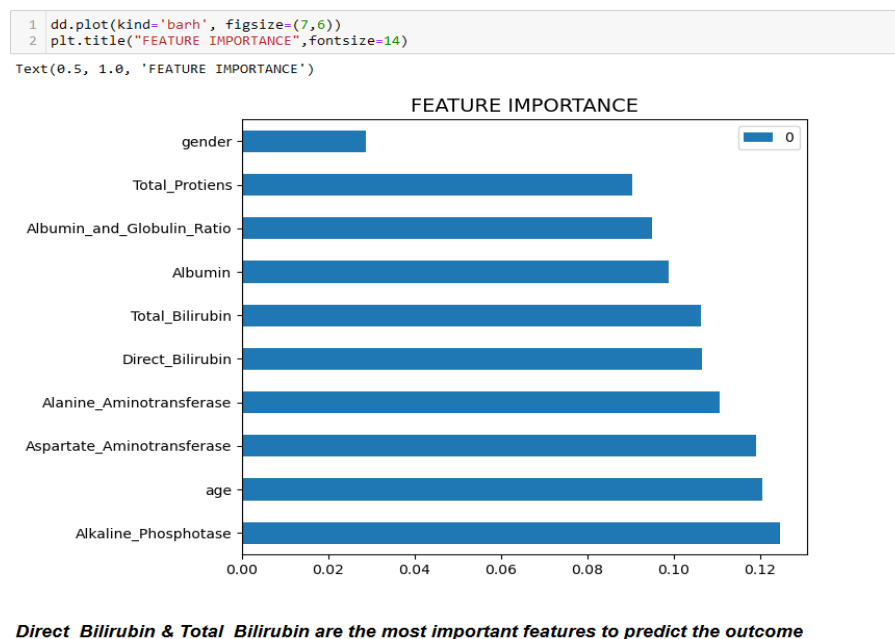
From the above image, We can see the



accuracy of the model. Random Forest Classifier is giving the accuracy of 70 percent.

Activity 2: Identifying Important Features

10 attributes are passed to predict the actual outcome, Its necessary to identify the l important feature to determin the output. Here we are using function called feature_importance to identify the important features among the available attributes and understand with a visualization.



Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```

1 import joblib
2 joblib.dump(model1, 'ETC.pkl')
['ETC.pkl']

```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project create two HTML files namely

- home.html
- predict.html

and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (`_name_`) as argument. And render HTML page:

```
app=Flask(__name__) # our flask app

@app.route('/') # rendering the html template
def home():
    return render_template('home.html')

@app.route('/predict') # rendering the html template
def index() :
    return render_template("index.html")
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with the home.html function.

Hence, when the home page of the web server is opened in the browser, the html page will be rendered.

Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/data_predict', methods=['POST']) # route for our prediction
def predict():
    age = request.form['age'] # requesting for age data
    gender = request.form['gender'] # requesting for gender data
    tb = request.form['tb'] # requesting for Total_Bilirubin data
    db = request.form['db'] # requesting for Direct_Bilirubin data
    ap = request.form['ap'] # requesting for Alkaline_Phosphotase data
    aa1 = request.form['aa1'] # requesting for Alamine_Aminotransferase data
    aa2 = request.form['aa2'] # requesting for Aspartate_Aminotransferase data
    tp = request.form['tp'] # requesting for Total_Protiens data
    a = request.form['a'] # requesting for Albumin data
    agr = request.form['agr'] # requesting for Albumin_and_Globulin_Ratio data

    # converting data into float format
    data = [[float(age), float(gender), float(tb), float(db), float(ap), float(aa1), float(aa2), float(tp),
    float(a), float(agr))]

    # Loading model which we saved
    model = pickle.load(open('liver_analysis.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 :')
    else:
        return render_template('chance.html', prediction='You dont have a liver desease problem')

if __name__ == '__main__':
    app.run()
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
if __name__ == '__main__':
    app.run()
```

Activity 2.3: Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on

the submit button, and see the result/prediction on the web.

```
base) D:\TheSmartBridge\Projects\2. DrugClassification\Drug c
* Serving Flask app "app" (lazy loading)
* Environment: production
* WARNING: This is a development server. Do not use it in a p
* Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now,Go the web browser and write the localhost url
(http://127.0.0.1:5000) to get the below result



Now,when you click Go to predict the button from the banner you will get
redirected to the prediction page.

Liver Patient Prediction

Age:

Gender:

Enter 0 as male, 1 as female

Total_Bilirubin:

Direct_Bilirubin:

Alkaline_Phosphatase:

Alamine_Aminotransferase:

Aspartate_Aminotransferase:

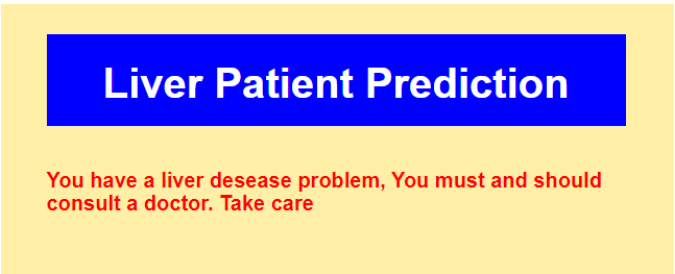
Total_Protiens:

Albumin:

Albumin_and_Globulin_Ratio:

Predict

Inputs- Now, the user will give inputs to get the predicted page after giving details user has to click on Predict Button to get the result.



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project developmentprocedure

Create document as per the template provided.

