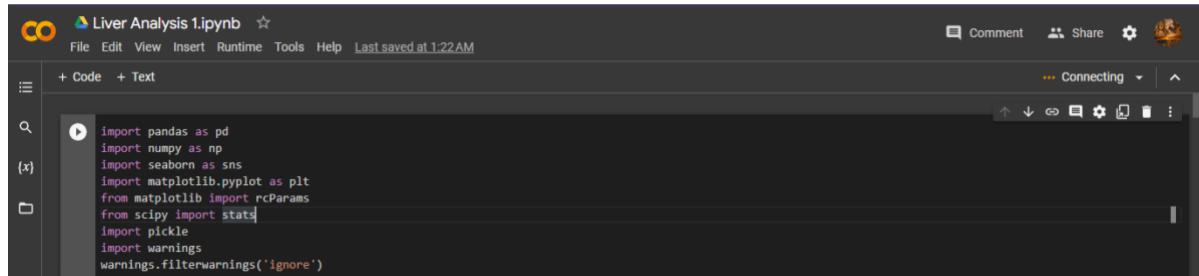


# Liver Patient Analysis Methods using Machine Learning

## Outputs:

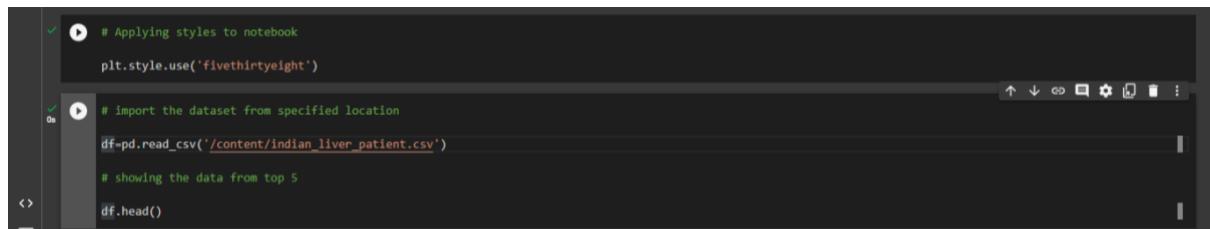
### Importing required libraries:



A screenshot of a Jupyter Notebook titled "Liver Analysis 1.ipynb". The code cell contains the following imports:

```
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 pickle
import warnings
warnings.filterwarnings('ignore')
```

### Read the csv file:



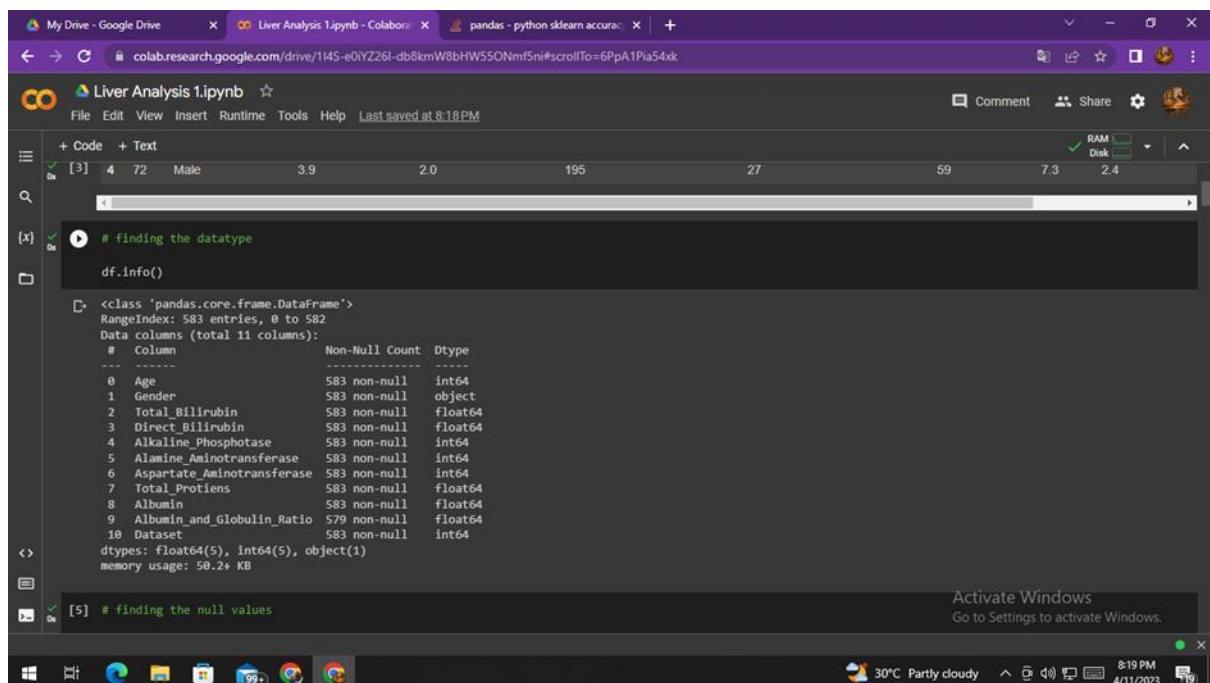
A screenshot of a Jupyter Notebook cell showing the execution of code to read a CSV file and display its head:

```
# Applying styles to notebook
plt.style.use('fivethirtyeight')

# import the dataset from specified location
df=pd.read_csv('/content/indian_liver_patient.csv')

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

### Getting the information using info() :



A screenshot of a Jupyter Notebook cell showing the execution of code to get information about the DataFrame using the `info()` method:

```
# finding the datatype
df.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_Phosphatase 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
```

## Finding sum of Null Values isnull().sum() function:

```
#finding the sum of null values in the data
df.isnull().sum()
```

{x} 1s

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
0	65	0	0.7	0.1	187	16	18	6.8	3.3		
1	62	1	10.9	5.5	699	64	100	7.5	3.2		
2	62	1	7.3	4.1	490	60	68	7.0	3.3		
3	58	1	1.0	0.4	182	14	20	6.8	3.4		
4	72	1	3.9	2.0	195	27	59	7.3	2.4		

## Handling and Filling the missing values:

```
# Handling and filling the missing value
df=df.fillna(0)
df.isnull().sum()
```

{x} 1s

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
0	65	0	0.7	0.1	187	16	18	6.8	3.3		
1	62	1	10.9	5.5	699	64	100	7.5	3.2		
2	62	1	7.3	4.1	490	60	68	7.0	3.3		
3	58	1	1.0	0.4	182	14	20	6.8	3.4		
4	72	1	3.9	2.0	195	27	59	7.3	2.4		

## Manual Encoding:

```
#Encoding(manual encoding)
df['Gender']=df['Gender'].replace({'Female':0,'Male':1})
df.head()
```

{x} 0s

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Dataset
0	65	0	0.7	0.1	187	16	18	6.8	3.3		
1	62	1	10.9	5.5	699	64	100	7.5	3.2		
2	62	1	7.3	4.1	490	60	68	7.0	3.3		
3	58	1	1.0	0.4	182	14	20	6.8	3.4		
4	72	1	3.9	2.0	195	27	59	7.3	2.4		

## **TYPES OF ANALYSIS (Descriptive Analysis):**

Liver Analysis 1.ipynb

```
File Edit View Insert Runtime Tools Help Last saved at 8:18PM
```

+ Code + Text

0a

```
"""
Types of Analysis
1. Univariate Analysis
2. Bivariate Analysis
3. Multivariate Analysis
4. Descriptive Analysis
"""

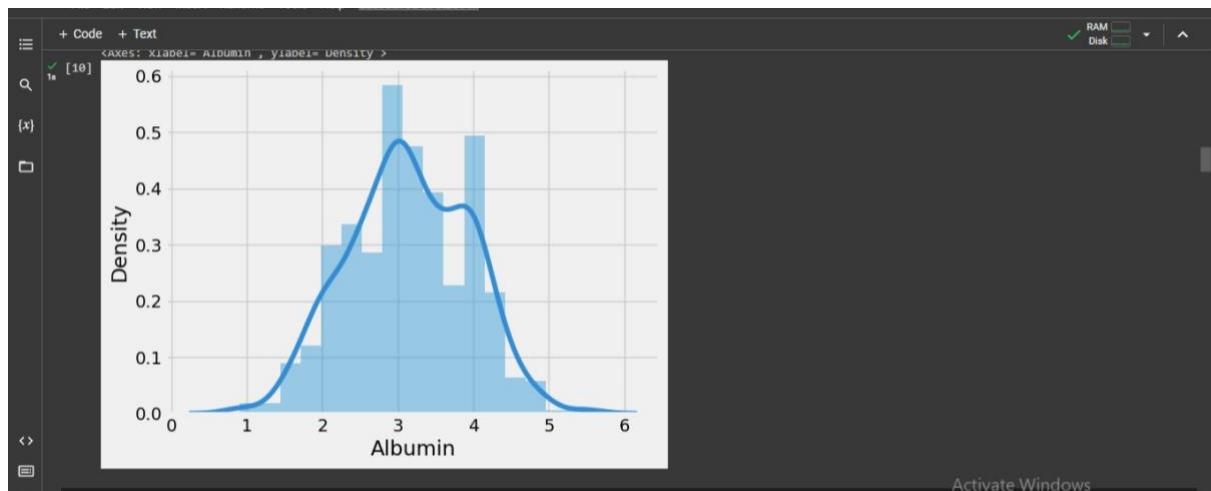
# Descriptive Analysis
df.describe()
```

Age Gender Total\_Bilirubin Direct\_Bilirubin Alkaline\_Phosphotase Alamine\_Aminotransferase Aspartate\_Aminotransferase Total\_Proteins Albu

	count	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
	std	16.189833	0.429603	6.209522	2.808498	242.937989	182.620356	288.918529	1.085451
	min	4.000000	0.000000	0.400000	0.100000	63.000000	10.000000	10.000000	2.700000
	25%	33.000000	1.000000	0.800000	0.200000	175.500000	23.000000	25.000000	5.800000
	50%	45.000000	1.000000	1.000000	0.300000	208.000000	35.000000	42.000000	6.600000
	75%	58.000000	1.000000	2.600000	1.300000	298.000000	60.500000	87.000000	7.200000
	max	90.000000	1.000000	75.000000	19.700000	2110.000000	2000.000000		5.5

### **i)Univariate Analysis:**

- ❖ Univariate Analysis is defined as carried out on only one variable to summarize or describe the variable.



## Splitting the data into dependent and independent variable:

Code in cell 1a:

```
#Splitting Dep & Indep variable
x=df.drop('Dataset',axis=1)
x.head()
```

Output of cell 1a:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_an
0	65	0	0.7	0.1	187	16	18	6.8	3.3	
1	62	1	10.9	5.5	699	64	100	7.5	3.2	
2	62	1	7.3	4.1	490	60	68	7.0	3.3	
3	58	1	1.0	0.4	182	14	20	6.8	3.4	
4	72	1	3.9	2.0	195	27	59	7.3	2.4	

Code in cell 1a:

```
[11] 4 72 1 3.9 2.0 195 27 59 7.3 2.4
```

Code in cell 1b:

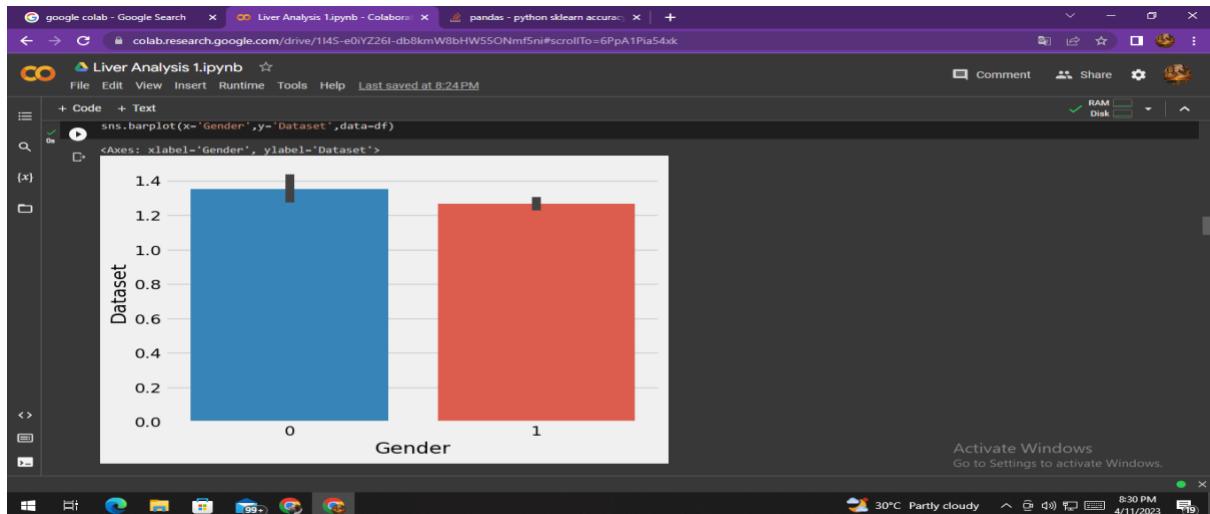
```
y=df['Dataset']
y
```

Output of cell 1b:

```
0    1
1    1
2    1
3    1
4    1
...
578   2
579   1
580   1
581   1
582   2
Name: Dataset, Length: 583, dtype: int64
```

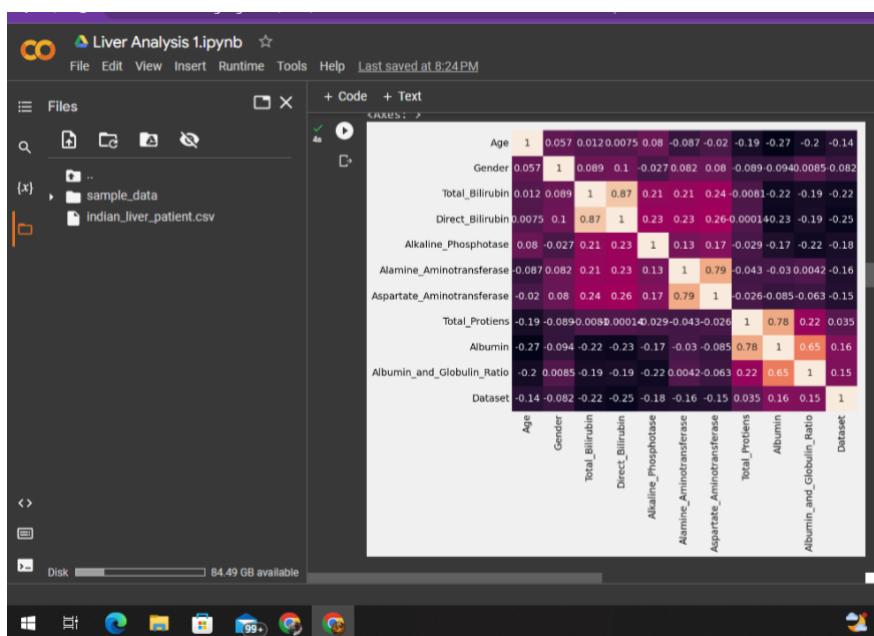
## ii) Bivariate Analysis:

- ❖ Bivariate Analysis refers to the analysis of two variable to determine relationship between them.



### iii) Multivariate Analysis:

- ❖ Multivariate Analysis is based in observation and analysis of more than one statistical outcome variable at a time.



## DATA PREPROCESSING:

### Finding the shape of the data

google colab - Google Search Liver Analysis 1.ipynb - Colaboratory pandas - python sklearn accuracy

Liver Analysis 1.ipynb

File Edit View Insert Runtime Tools Help Last saved at 8:24PM

+ Code + Text

Q

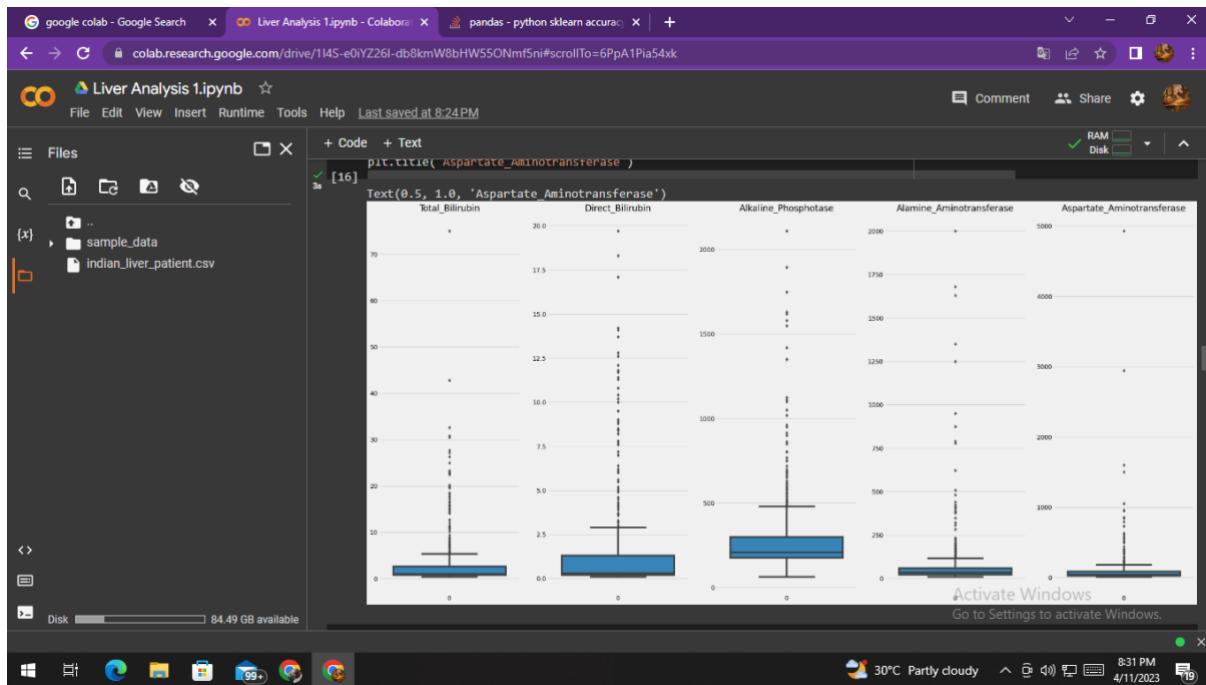
0s # Data preprocessing

{x} # Finding the shape of the data

□ df.shape

↳ (583, 11)

## Finding outliers:





## Count of Outliers:

google colab - Google Search | Liver Analysis 1.ipynb - Colaboratory | pandas - python sklearn accuracy | +

Liver Analysis 1.ipynb

```

print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))
print('Skewed data :',len(df[df['Total_Bilirubin']>upperBound]))

```

Q1 = 0.8  
Q3 = 2.6  
IQR value is 1.8  
The upperbound value is 5.300000000000001 & the lower bound value is -1.9000000000000001  
Skewed data : 84

Liver Analysis 1.ipynb

```

print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))
print('Skewed data :',len(df[df['Total_Bilirubin']>upperBound]))

```

Q1 = 0.8  
Q3 = 2.6  
IQR value is 1.8  
The upperbound value is 5.300000000000001 & the lower bound value is -1.9000000000000001  
Skewed data : 84

Liver Analysis 1.ipynb

```
IQR = q3-q1

print('IQR value is {}'.format(IQR))

upperBound = q3+(1.5*IQR)
lowerBound = q1-(1.5*IQR)

print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))

print('Skewed data :',len(df[df['Alkaline_Phosphotase']>upperBound]))
```

Q1 = 175.5  
Q3 = 298.0  
IQR value is 122.5  
The upperbound value is 481.75 & the lower bound value is -8.25  
Skewed data : 69

Liver Analysis 1.ipynb

```
IQR = q3-q1

print('IQR value is {}'.format(IQR))

upperBound = q3+(1.5*IQR)
lowerBound = q1-(1.5*IQR)

print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))

print('Skewed data :',len(df[df['Alanine_Aminotransferase']>upperBound]))
```

Q1 = 23.0  
Q3 = 60.5  
IQR value is 37.5  
The upperbound value is 116.75 & the lower bound value is -33.25  
Skewed data : 73

Liver Analysis 1.ipynb

```
Q1 = np.quantile(df['Total_Protiens'],0.25)
Q3 = np.quantile(df['Total_Protiens'],0.75)
IQR = Q3-Q1

print('IQR value is {}'.format(IQR))

upperBound = Q3+(1.5*IQR)
lowerBound = Q1-(1.5*IQR)

print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))

print('Skewed data :',len(df[df['Total_Protiens']>upperBound]))
```

Q1 = 5.8  
Q3 = 7.2  
IQR value is 1.4000000000000004  
The upperbound value is 9.3 & the lower bound value is 3.6999999999999993  
Skewed data : 2

Liver Analysis 1.ipynb

```
q3=np.quantile(df['Albumin_and_Globulin_Ratio'],0.75)

print('Q1 = {}'.format(q1))
print('Q3 = {}'.format(q3))

IQR = q3-q1

print('IQR value is {}'.format(IQR))

upperBound = q3+(1.5*IQR)
lowerBound = q1-(1.5*IQR)

print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))

print('Skewed data :',len(df[df['Albumin_and_Globulin_Ratio']>upperBound]))
```

Q1 = 0.7  
Q3 = 1.1  
IQR value is 0.4000000000000013  
The upperbound value is 1.7000000000000002 & the lower bound value is 0.0999999999999976  
Skewed data : 10

google colab - Google Search Liver Analysis 1.ipynb pandas - python sklearn accurac colab.research.google.com/drive/1I4S-e0YZ26I-db8kmW8bHW55ONmt5ni#scrollTo=6PpA1Pia54xk

Liver Analysis 1.ipynb

File Edit View Insert Runtime Tools Help Last saved at 8:24PM

Files

+ Code + Text

```
[ ] print('Q3 = {}'.format(q3))

IQR = q3-q1

print('IQR value is {}'.format(IQR))

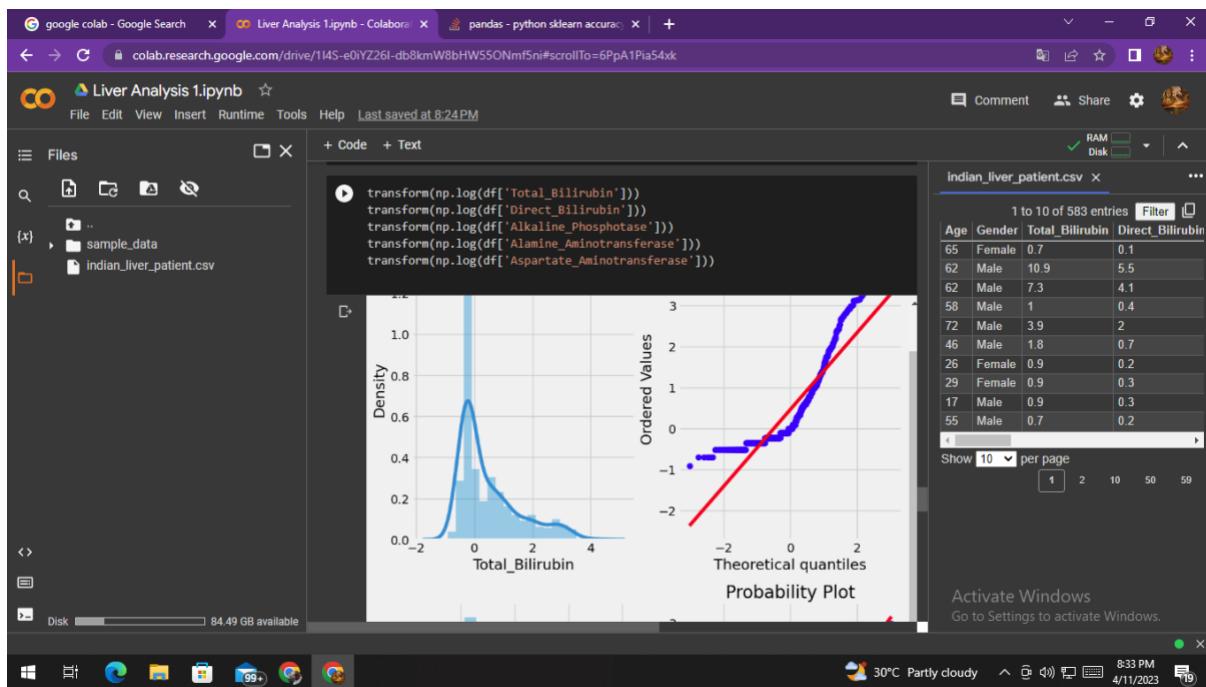
upperBound = q3+(1.5*IQR)
lowerBound = q1-(1.5*IQR)

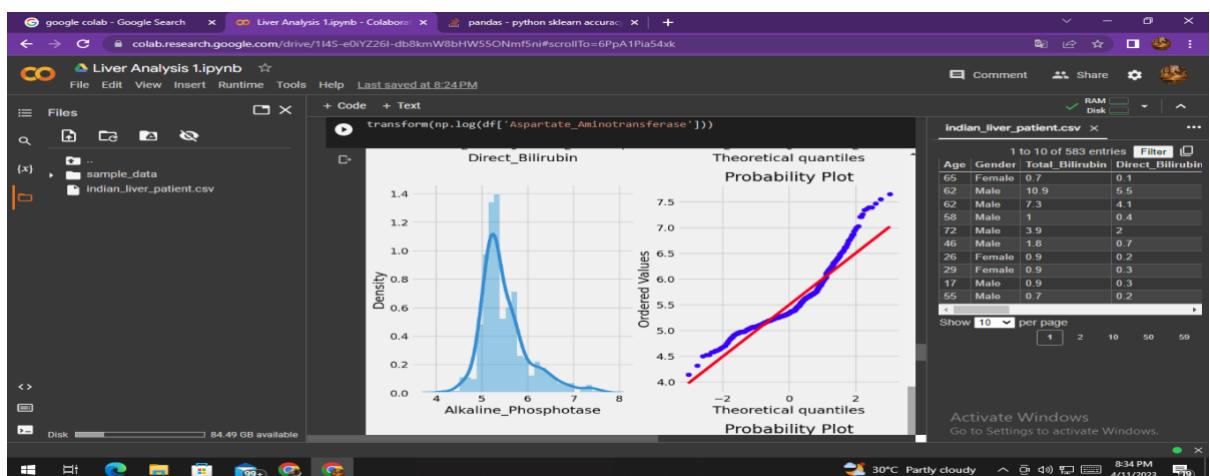
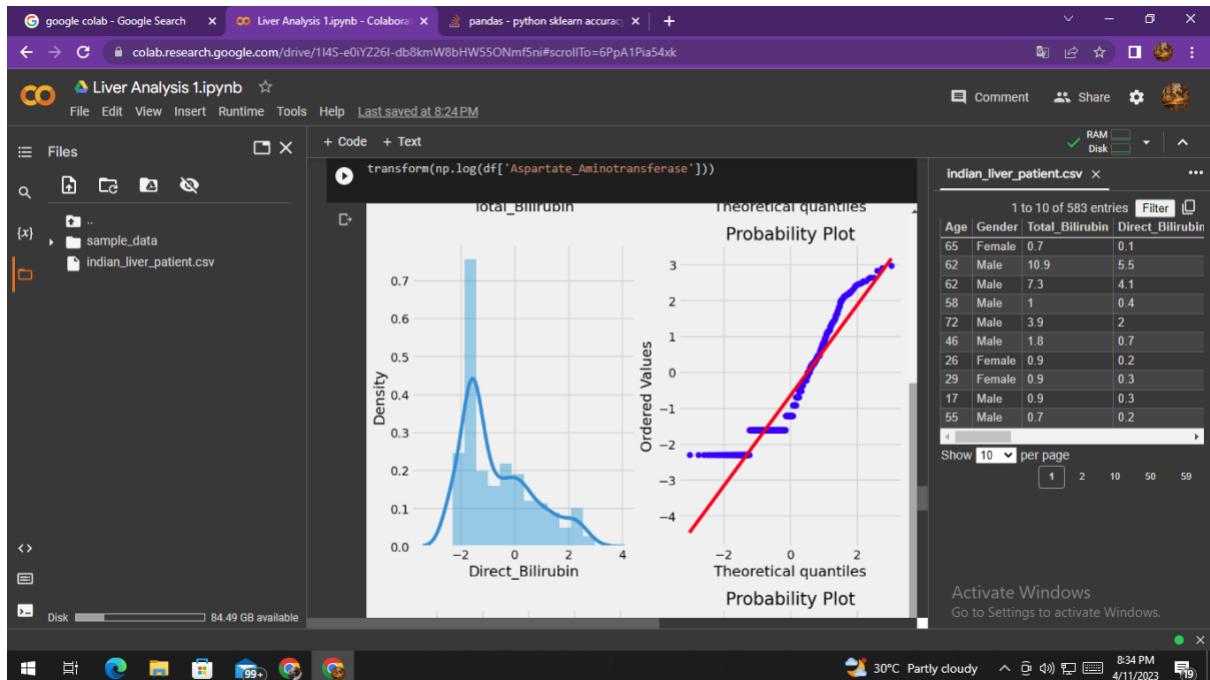
print('The upperbound value is {} & the lower bound value is {}'.format(upperBound,lowerBound))

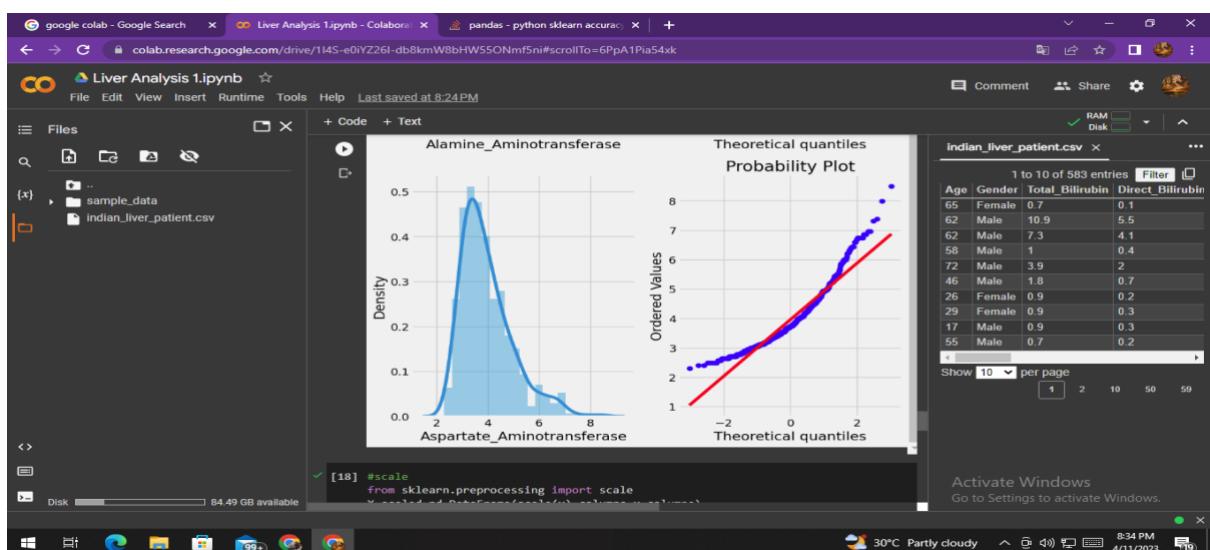
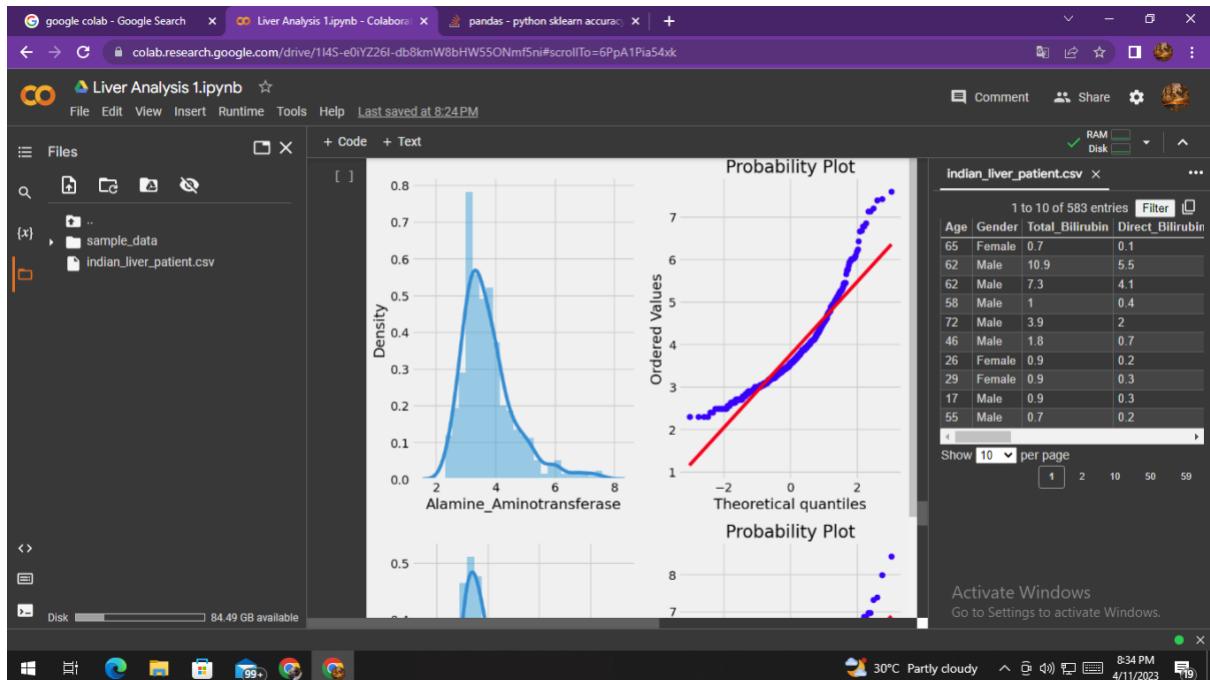
print('Skewed data :',len(df[df['Aspartate_Aminotransferase']>upperBound]))
```

Q1 = 25.0  
Q3 = 87.0  
IQR value is 62.0  
The upperbound value is 180.0 & the lower bound value is -68.0  
Skewed data : 66

## Transformation log:







## Scaling the Data:

The screenshot shows a Google Colab notebook titled "Liver Analysis 1.ipynb". The code cell [18] contains the command `X\_scaled=pd.DataFrame(scale(x),columns=x.columns)` followed by a call to `X\_scaled.head()`. The resulting output is a table showing scaled data for the first five rows of the dataset. The code cell [20] shows the splitting of the data into training and testing sets using `train\_test\_split` from `sklearn.model\_selection`. The code cell [21] imports `SMOTE` from `imblearn.over\_sampling`.

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	AI
0	1.252098	-1.762281	-0.418878	-0.493964	-0.426715	-0.354665	-0.318393	0.292120	0.198969	
1	1.066637	0.567446	1.225171	1.430423	1.682629	-0.091599	-0.034333	0.937566	0.073157	
2	1.066637	0.567446	0.644919	0.931508	0.821588	-0.113522	-0.145186	0.476533	0.198969	
3	0.819356	0.567446	-0.370523	-0.387054	-0.447314	-0.365626	-0.311465	0.292120	0.324781	
4	1.684839	0.567446	0.096902	0.183135	-0.393756	-0.294379	-0.176363	0.753153	-0.933340	

## Splitting the data into training and testing data:

The screenshot shows a Google Colab notebook titled "Liver Analysis 1.ipynb". The code cell [22] imports `SMOTE` from `imblearn.over\_sampling`. The code cell [23] shows the value counts for the target variable `y\_train`, which has two categories: 1 and 2. The code cell [24] uses `fit\_resample` from `SMOTE` to create balanced training data `x\_train\_smote` and `y\_train\_smote`. The code cell [25] shows the value counts for the resampled target variable `y\_train\_smote`, which now has equal counts for both categories. The code cell [26] defines a Random Forest classifier and fits it to the resampled training data, then predicts on the test set.

## MODEL BUILDING:

### Applying Algorithms

#### i) Random Forest Algorithm

The screenshot shows a Google Colab notebook titled "Liver Analysis 1.ipynb". The code cell contains Python code for a Random Forest classifier, including imports from `sklearn.ensemble` and `sklearn.metrics`, fitting the model, predicting on test data, calculating accuracy, and printing a classification report. The output cell displays the classification report table.

	precision	recall	f1-score	support
1	0.83	0.78	0.80	87
2	0.46	0.53	0.49	30
accuracy			0.72	117
macro avg	0.64	0.66	0.65	117
weighted avg	0.73	0.72	0.72	117

[32] #Decision tree model  
from sklearn.tree import DecisionTreeClassifier  
model4=DecisionTreeClassifier()  
model4.fit(x\_train\_smote,y\_train\_smote)  
y\_predict=model4.predict(x\_test)  
dct1=accuracy\_score(y\_test,y\_predict)  
dct1  
pd.crosstab(y\_test,y\_predict)

#### ii) Decision Tree

The screenshot shows a Google Colab notebook titled "Liver Analysis 1.ipynb". The code cell contains Python code for a Decision Tree classifier, including imports from `sklearn.tree`, fitting the model, predicting on test data, calculating accuracy, and printing a classification report. The output cell displays the classification report table.

	precision	recall	f1-score	support
1	0.81	0.76	0.79	87
2	0.42	0.50	0.45	30
accuracy			0.69	117
macro avg	0.62	0.63	0.62	117
weighted avg	0.71	0.69	0.70	117

#Decision tree model  
from sklearn.tree import DecisionTreeClassifier  
model4=DecisionTreeClassifier()  
model4.fit(x\_train\_smote,y\_train\_smote)  
y\_predict=model4.predict(x\_test)  
dct1=accuracy\_score(y\_test,y\_predict)  
dct1  
pd.crosstab(y\_test,y\_predict)  
print(classification\_report(y\_test,y\_predict))

[33] #KNN model  
from sklearn.neighbors import KNeighborsClassifier

### iii) K- Nearest Neighbour Algorithm

The screenshot shows a Google Colab notebook titled "Liver Analysis 1.ipynb". The code cell contains Python code for a KNN model:#KNN model
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))The output shows the accuracy score and a classification report:|  | precision | recall | f1-score | support |
| --- | --- | --- | --- | --- |
| 1 | 0.82 | 0.52 | 0.63 | 87 |
| 2 | 0.32 | 0.67 | 0.43 | 30 |

	accuracy			
accuracy	0.56			117
macro avg	0.57	0.59	0.53	117
weighted avg	0.69	0.56	0.58	117

```
[34] #Logistic Regression
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)
```

A status bar at the bottom indicates "Activate Windows Go to Settings to activate Windows." and shows the date and time as 4/11/2023 8:35 PM.

### iv) Logistic Regression

The screenshot shows a Google Colab notebook titled "Liver Analysis 1.ipynb". The code cell contains Python code for a Logistic Regression model:[33]
accuracy 0.56 117
macro avg 0.57 0.59 0.53 117
weighted avg 0.69 0.56 0.58 117

```
[34] #Logistic Regression
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))
```

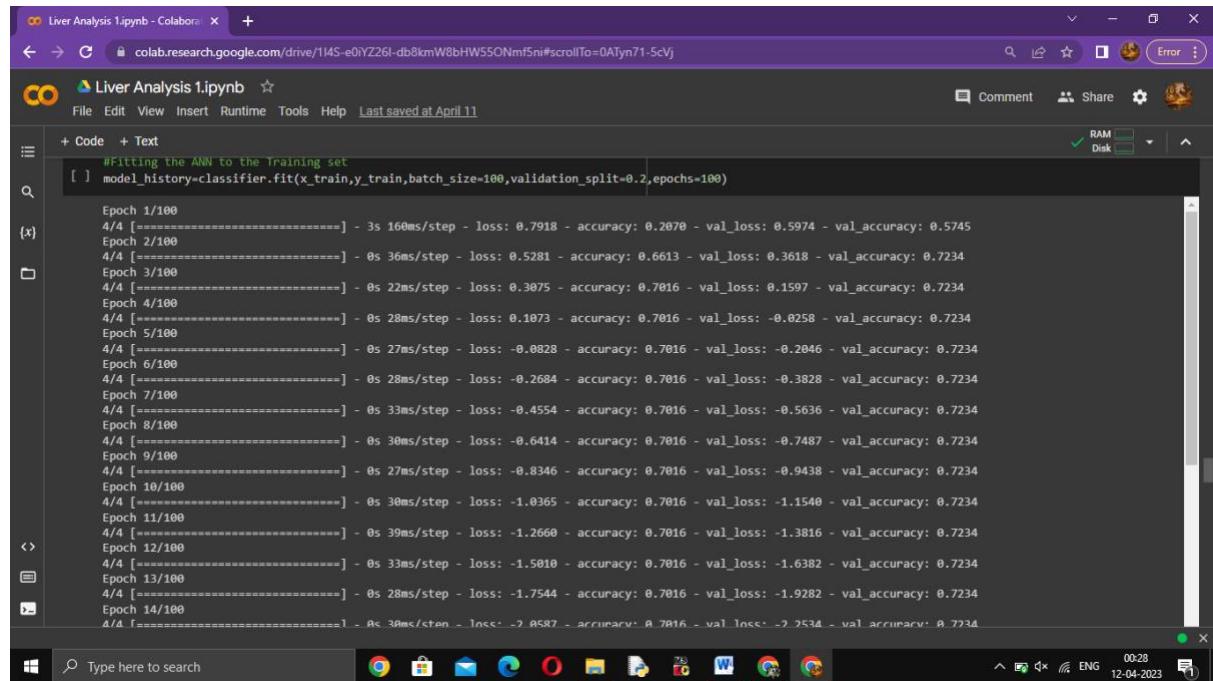
The output shows the accuracy score and a classification report:|  | precision | recall | f1-score | support |
| --- | --- | --- | --- | --- |
| 1 | 0.94 | 0.55 | 0.70 | 87 |
| 2 | 0.41 | 0.90 | 0.56 | 30 |

	accuracy			
accuracy	0.64			117
macro avg	0.68	0.73	0.63	117
weighted avg	0.80	0.64	0.66	117

```
[ ]
```

A status bar at the bottom indicates "Activate Windows Go to Settings to activate Windows." and shows the date and time as 4/11/2023 8:35 PM.

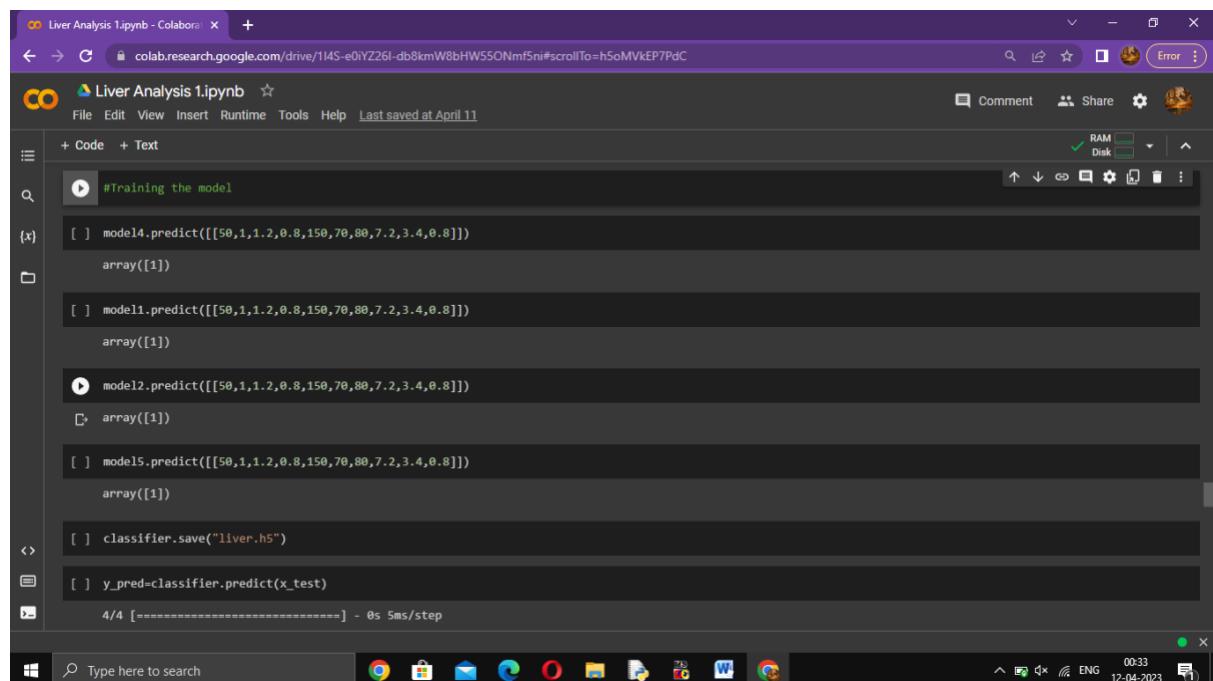
## Fitting the ANN to the training set:



```
#Fitting the ANN to the Training set
[ ] model_history=classifier.fit(x_train,y_train,batch_size=100,validation_split=0.2,epochs=100)

Epoch 1/100
4/4 [=====] - 3s 160ms/step - loss: 0.7918 - accuracy: 0.2070 - val_loss: 0.5974 - val_accuracy: 0.5745
Epoch 2/100
4/4 [=====] - 0s 36ms/step - loss: 0.5281 - accuracy: 0.6613 - val_loss: 0.3618 - val_accuracy: 0.7234
Epoch 3/100
4/4 [=====] - 0s 22ms/step - loss: 0.3075 - accuracy: 0.7016 - val_loss: 0.1597 - val_accuracy: 0.7234
Epoch 4/100
4/4 [=====] - 0s 28ms/step - loss: 0.1073 - accuracy: 0.7016 - val_loss: 0.0258 - val_accuracy: 0.7234
Epoch 5/100
4/4 [=====] - 0s 27ms/step - loss: 0.0828 - accuracy: 0.7016 - val_loss: 0.2046 - val_accuracy: 0.7234
Epoch 6/100
4/4 [=====] - 0s 28ms/step - loss: 0.2684 - accuracy: 0.7016 - val_loss: 0.3828 - val_accuracy: 0.7234
Epoch 7/100
4/4 [=====] - 0s 33ms/step - loss: 0.4554 - accuracy: 0.7016 - val_loss: 0.5636 - val_accuracy: 0.7234
Epoch 8/100
4/4 [=====] - 0s 30ms/step - loss: 0.6414 - accuracy: 0.7016 - val_loss: 0.7487 - val_accuracy: 0.7234
Epoch 9/100
4/4 [=====] - 0s 27ms/step - loss: 0.8346 - accuracy: 0.7016 - val_loss: 0.9438 - val_accuracy: 0.7234
Epoch 10/100
4/4 [=====] - 0s 30ms/step - loss: 1.0365 - accuracy: 0.7016 - val_loss: 1.1548 - val_accuracy: 0.7234
Epoch 11/100
4/4 [=====] - 0s 39ms/step - loss: 1.2668 - accuracy: 0.7016 - val_loss: 1.3816 - val_accuracy: 0.7234
Epoch 12/100
4/4 [=====] - 0s 33ms/step - loss: 1.5010 - accuracy: 0.7016 - val_loss: 1.6382 - val_accuracy: 0.7234
Epoch 13/100
4/4 [=====] - 0s 28ms/step - loss: 1.7544 - accuracy: 0.7016 - val_loss: 1.9282 - val_accuracy: 0.7234
Epoch 14/100
4/4 [=====] - 0s 30ms/step - loss: 2.057 - accuracy: 0.7016 - val_loss: 2.2534 - val_accuracy: 0.7234
```

## Testing the model:



```
#Training the model
[ ] model4.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]])
array([1])

[ ] model1.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]])
array([1])

[ ] model2.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]])
array([1])

[ ] model3.predict([[50,1,1.2,0.8,150,70,80,7.2,3.4,0.8]])
array([1])

[ ] classifier.save("liver.h5")
[ ] y_pred=classifier.predict(x_test)
4/4 [=====] - 0s 5ms/step
```

## Predicting the Test data:

```
def predict_exit(sample_value):
    #Convert list to numpy array
    sample_value=np.array(sample_value)
    #Reshape because sample_value contains only 1 record
    sample_value=sample_value.reshape(1,-1)
    #Feature scaling
    sample_value=scale(sample_value)
    return classifier.predict(sample_value)

#sample predict
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('Healthy')

1/1 [=====] - 0s 62ms/step
prediction:Liver Patient
```

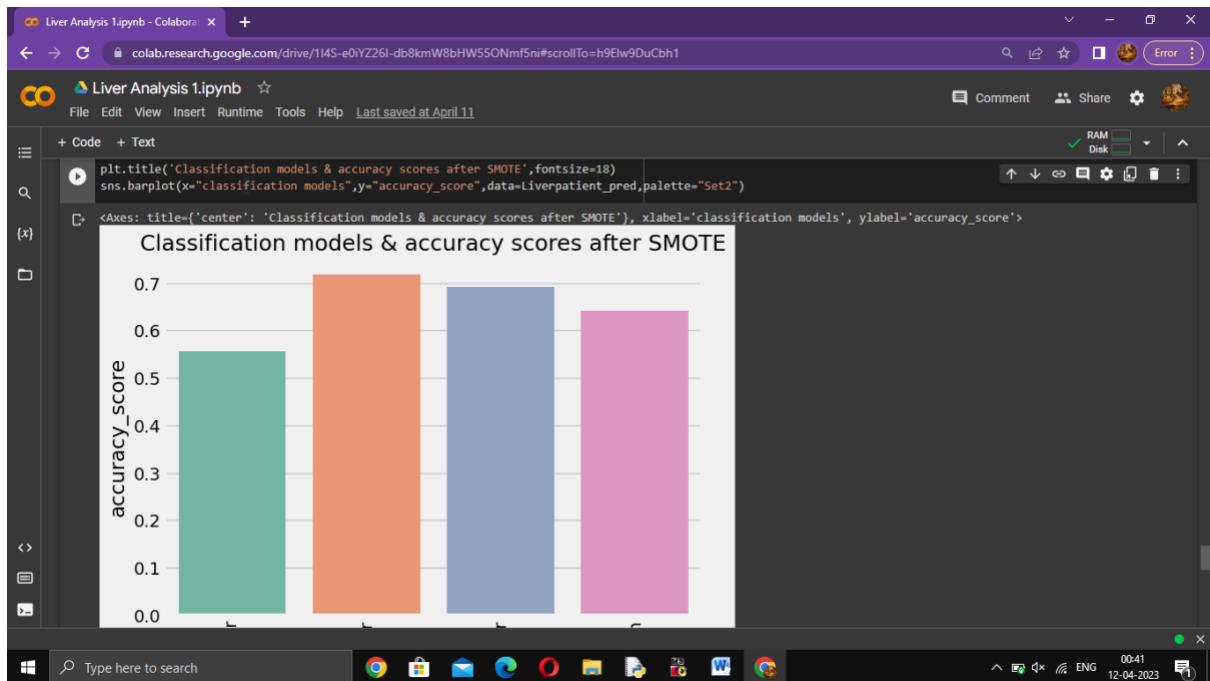
# PERFORMANCE TESTING AND HYPER PARAMETER TUNING

## i)comparing the model

Liver Analysis 1.ipynb - Colaboratory

```
#compare the model
acc_smote=[['KNN classifier',knn1],['RandomForestClassifier',rfc1],['DecisionTreeClassifier',dct1],['LogisticRegression',logit1]]
Liverpatient_pred=pd.DataFrame(acc_smote,columns=['classification models','accuracy_score'])
Liverpatient_pred
```

classification models	accuracy_score
KNN classifier	0.55556
RandomForestClassifier	0.717949
DecisionTreeClassifier	0.692308
LogisticRegression	0.641026



Liver Analysis 1.ipynb

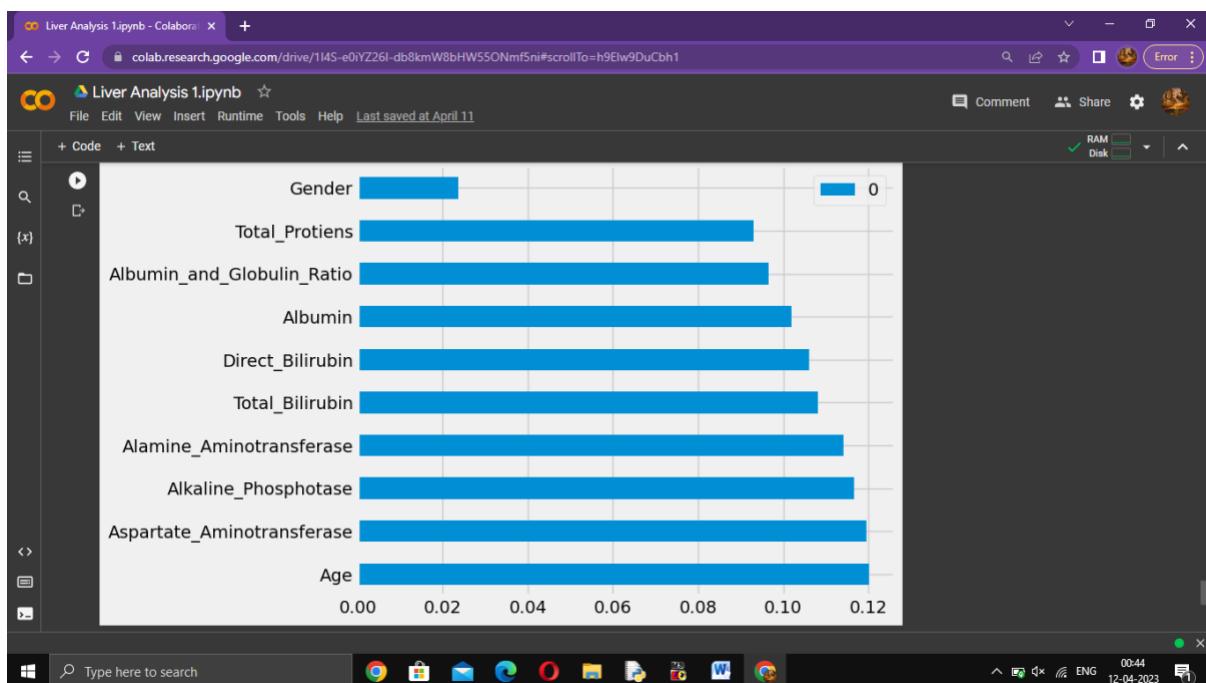
```
+ Code + Text
[ ] model.feature_importances_
array([0.12015134, 0.02356539, 0.10818847, 0.10612846, 0.11662611,
       0.11418247, 0.11955127, 0.09306392, 0.10194888, 0.09659369])
```

dd=pd.DataFrame(model.feature\_importances\_,index=x.columns).sort\_values(0,ascending=False)

dd

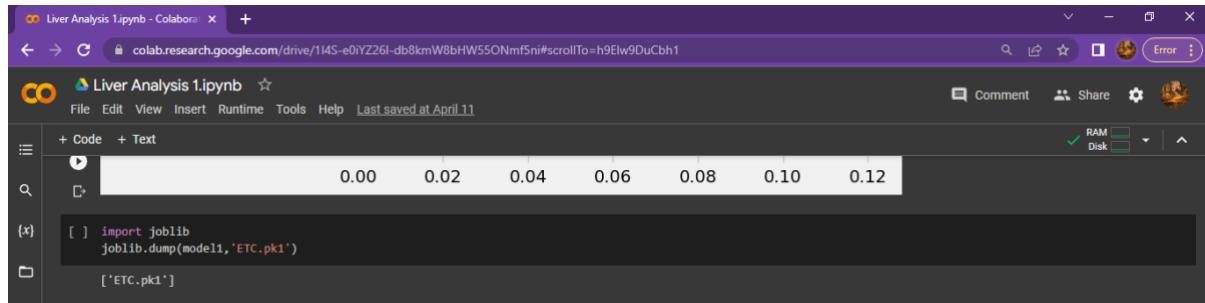
	0
Age	0.120151
Aspartate_Aminotransferase	0.119551
Alkaline_Phosphatase	0.116626
Alamine_Aminotransferase	0.114182
Total_Bilirubin	0.106188
Direct_Bilirubin	0.106128
Albumin	0.101949
Albumin_and_Globulin_Ratio	0.096594
Total_Protiens	0.093064
Gender	0.023565

## ii) Identifying important features



# Model Deployment

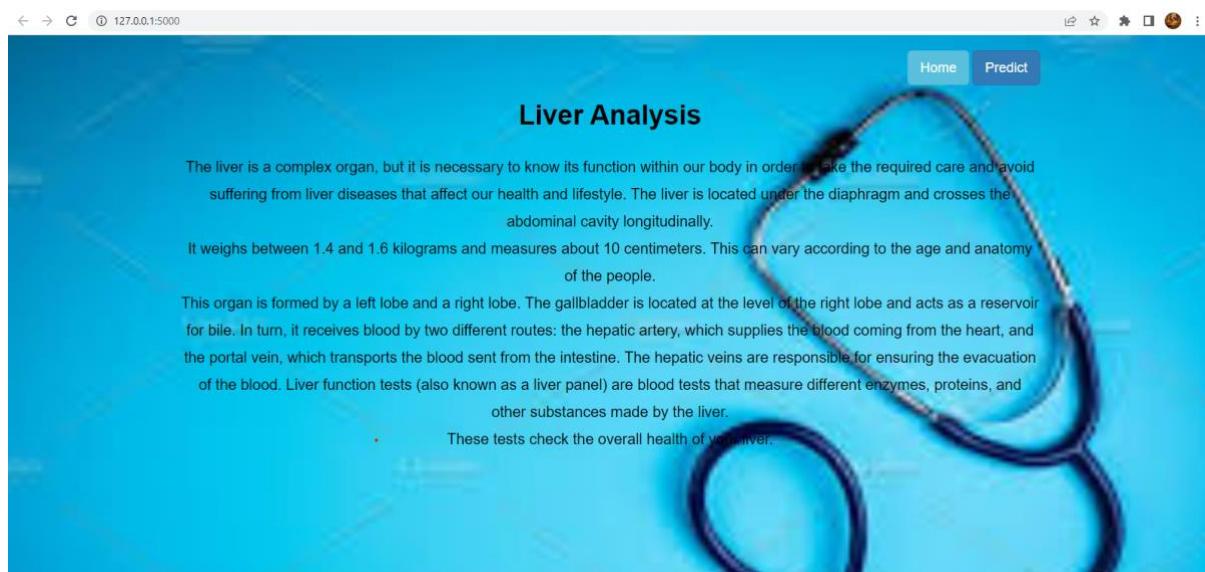
## i) Save the best model :



```
Liver Analysis 1.ipynb - Colaboratory + colab.research.google.com/drive/114S-e0lYZ26I-db8kmW8bHW55ONmf5ni#scrollTo=h9Elw9DuCbh1
File Edit View Insert Runtime Tools Help Last saved at April 11
Comment Share Settings Help
RAM Disk
[ ] import joblib
joblib.dump(model1,'ETC.pk1')
['ETC.pk1']
```

## Flask Outputs:

### Home page



## Predict Page

← → ⌛ 127.0.0.1:5000/predict

### Liver Patient analysis

Age  
18

Gender  
Male

Total\_bilirubin  
1

Direct\_bilirubin  
10

Alkaline\_Phosphotase  
56

Alamine\_Aminotransferase  
56

Aspartate\_Aminotransferase  
7

File Empathy map for L...pdf  
File brainstorming in li...pdf  
AUD-20230421-...opus  
LIVER DS - Copy.docx

Show all X

## Submit Page

