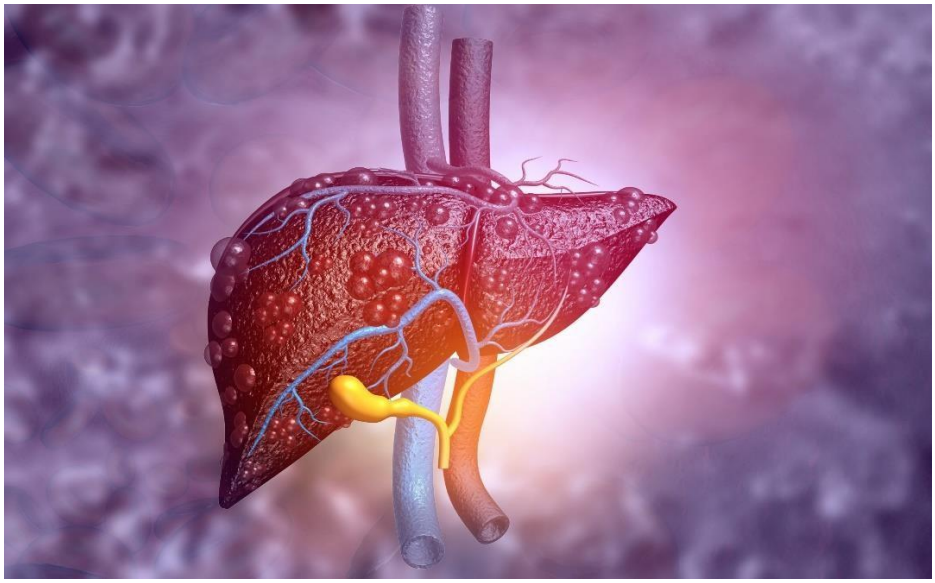
Detecting Liver Patients in India using Hypothesis Testing and Machine Learning Methods



SUBMITTED BY: PALLAVI MAZUMDAR

Motivation behind the Project:

The aim of this project is to understand the factors behind the cause of various liver diseases in India and gain various insights on how it is affecting different age groups and/or genders through hypothesis testing and to see if we can predict the presence of this disease using Machine learning models.

Introduction:

Liver diseases are fast being recognized as public health priorities in India. The burden of liver disease in India is significant because it alone contributed to 18.3% of the two million global liver disease–related deaths in 2015. Contribution of cirrhosis and its complications, collectively chronic liver diseases (CLDs), as causes of mortality in India have been increasing progressively since 1980, compared with China, the other country in Asia with a large population, where it remains stationary and is even showing downward trends.

In India, similar to many developing nations, there are limitations in the quality of the available epidemiological data resources in liver disease in the context of diagnostic precision and clinical phenotyping, uniformity of reporting, and non-existent electronic databases across the country. Despite this, a fair body of evidence is available that suggests the increasing impact of liver diseases on the country's economy and health care resources, apart from being a cause of premature death and disability.

Methodology:

- ## Data Source & Description

  This data set is taken from Kaggle and contains 416 liver patient records and 167 non liver patient records. The data set was collected from test samples in North East of Andhra Pradesh, India. 'is_patient' is a class label used to divide into groups (liver patient or not). This data set

contains 441 male patient records and 142 female patient records. Any patient whose age exceeded 89 is listed as being of age "90".

1. ***Age*** - Age of the patient
2. ***Gender*** **-** Gender of the patient
3. ***tot_bilirubin*** - Total Bilirubin
4. ***direct_bilirubin*** - Direct Bilirubin
5. ***alkphos*** - Alkaline Phosphatase
6. ***sgpt*** - Alanine Aminotransferase
7. ***sgot*** **-** Aspartate Aminotransferase
8. ***tot_proteins*** **-** Total Proteins
9. ***albumin*** - Albumin
10. ***ag_ratio*** - Albumin and Globulin Ratio
11. ***is_patient*** - Selector field used to split the data into two sets

- Tools used
  - Python (Jupyter)
  - Microsoft Word

Hypothesis Testing:

Z- Proportionality Test

H0: The proportion of patients is equal for both Male and Female

H1: The proportion of patients is not equal for both the Genders

```
In [5]: dataf['is_patient'].value_counts()

Out[5]: 1    416
        2    167
        Name: is_patient, dtype: int64
```

In [6]:
```
import numpy as np

from statsmodels.stats.proportion import proportions_ztest

Female = dataf[dataf['Gender'] == 'Female']
Male = dataf[dataf['Gender'] == 'Male']
n1 = len(Female)
n2 = len(Male)
print(n1,n2)

x1=len(Female[Female['is_patient'] == 1])
x2=len(Male[Male['is_patient'] == 1])
print(x1)
print(x2)
```

```
142 441
92
324
```

In [7]:
```
# Output the p-value of the test statistic (two-tailed test)
count = np.array([x1,x2])
lens = np.array([n1,n2])
proportions_ztest(count, lens)
```

Out[7]:
```
(-1.9899646317394697, 0.0465948317274726)
```

Here, our Z-test statistic is -1.989 and p value is 0.04

Since p < alpha(0.05), we reject the null hypothesis

Therefore the proportion of the liver disease is not equal for Male and Female

ANOVA test

H0 : mean1 = mean2 = mean3 = mean4

H1 : Atleast one of the mean is not equal

Comparing the mean of the 'Direct_Bilirubin' among different age groups

In [8]:
```
patient_df = dataf[dataf['is_patient'] == 1]
patient_df
```

Out[8]:

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Al
0	65	Female	0.7	0.1	187	
1	62	Male	10.9	5.5	699	
2	62	Male	7.3	4.1	490	

In [9]:
```
age_group_0to20 = patient_df[patient_df['Age'] <= 20]['Direct_Bilirubin']
age_group_0to20
```

Out[9]:
```
70     0.2
85     0.5
88     0.2
98     0.1
99     0.1
102    0.2
134    0.7
137    0.2
138    0.2
139    0.2
```

In [29]:
```
age_group_21to40 = patient_df[(patient_df['Age'] > 20) & (patient_df['Age'] < 41)]['Direct_Bilirubin']
age_group_21to40
```

Out[29]:
```
6       0.2
7       0.3
16      0.8
18      0.3
19      0.3
       ...
575    13.7
576     8.2
577     8.4
579     0.1
581     0.5
Name: Direct_Bilirubin, Length: 135, dtype: float64
```

In [30]:
```
age_group_41to60 = patient_df[(patient_df['Age'] > 40) & (patient_df['Age'] < 61)]['Direct_Bilirubin']
age_group_41to60
```

Out[30]:
```
3       0.4
5       0.7
9       0.2
10      0.1
20      1.0
       ...
557     1.2
558     2.5
565    11.8
567     1.4
580     0.2
Name: Direct_Bilirubin, Length: 186, dtype: float64
```

In [31]:
```
age_group_61to90 = patient_df[(patient_df['Age'] > 60) & (patient_df['Age'] < 91)]['Direct_Bilirubin']
age_group_61to90
```

Out[31]:
```
0      0.1
1      5.5
2      4.1
4      2.0
11     1.3
      ...
560    7.7
561    7.6
562    8.5
563    0.5
571    0.3
Name: Direct_Bilirubin, Length: 72, dtype: float64
```

```
In [32]: from scipy.stats import f_oneway

         f_oneway(age_group_0to20, age_group_21to40, age_group_41to60, age_group_61to90)

Out[32]: F_onewayResult(statistic=1.0512558757248254, pvalue=0.36972189437217495)
```

Here, our f statistic is 1.051 and p value is 0.3697

Since p > alpha, we do not reject the null hypothesis

Therefore mean1 = mean2 = mean3 = mean4

Machine Learning models:

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase
Age	1.000000	0.011763	0.007529	0.080425	-0.086883
Total_Bilirubin	0.011763	1.000000	0.874618	0.206669	0.214065
Direct_Bilirubin	0.007529	0.874618	1.000000	0.234939	0.233894
Alkaline_Phosphotase	0.080425	0.206669	0.234939	1.000000	0.125680
Alamine_Aminotransferase	-0.086883	0.214065	0.233894	0.125680	1.000000
Aspartate_Aminotransferase	-0.019910	0.237831	0.257544	0.167196	0.791966
Total_Protiens	-0.187461	-0.008099	-0.000139	-0.028514	-0.042518
Albumin	-0.265924	-0.222250	-0.228531	-0.165453	-0.029742
Albumin_and_Globulin_Ratio	-0.216408	-0.206267	-0.200125	-0.234166	-0.002375
is_patient	-0.137351	-0.220208	-0.246046	-0.184866	-0.163416

Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	is_patient
-0.019910	-0.187461	-0.265924	-0.216408	-0.137351
0.237831	-0.008099	-0.222250	-0.206267	-0.220208
0.257544	-0.000139	-0.228531	-0.200125	-0.246046
0.167196	-0.028514	-0.165453	-0.234166	-0.184866
0.791966	-0.042518	-0.029742	-0.002375	-0.163416
1.000000	-0.025645	-0.085290	-0.070040	-0.151934
-0.025645	1.000000	0.784053	0.234887	0.035008
-0.085290	0.784053	1.000000	0.689632	0.161388
-0.070040	0.234887	0.689632	1.000000	0.163131
-0.151934	0.035008	0.161388	0.163131	1.000000

Importing the models

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score
```

```
X = data.iloc[:, :10]
y = data['is_patient']
```

Scaling using MinMaxScaler

```
scaler=MinMaxScaler()
scaled_values=scaler.fit_transform(X)
X.loc[:,:]=scaled_values
```

Train-Test split

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size =0.3, random_state = 123)
```

```
print(f'Shape of X:{X_train.shape}, Shape of y: {y_train.shape}')
```

```
Shape of X:(408, 10), Shape of y: (408,)
```

```
print(f'Shape of X:{X_test.shape}, Shape of y: {y_test.shape}')
```

```
Shape of X:(175, 10), Shape of y: (175,)
```

```
dt = DecisionTreeClassifier(max_depth=4,min_samples_leaf=0.14,random_state=1)
lr = LogisticRegression(solver='lbfgs')
KNN_1 = KNN(n_neighbors = 10)
rf = RandomForestClassifier(n_estimators=25, random_state=2)
```

Decision Tree

```
dt.fit(X_train, y_train)
dt_y_pred = dt.predict(X_test)
dt_acc_test = accuracy_score(y_test, dt_y_pred)
print('Test set accuracy of dt: {:.2f}'.format(dt_acc_test))
```

```
Test set accuracy of dt: 0.70
```

Logistic Regression

```
lr.fit(X_train, y_train)
lr_y_pred = lr.predict(X_test)
lr_acc_test = accuracy_score(y_test, lr_y_pred)
print('Test set accuracy of dt: {:.2f}'.format(lr_acc_test))
```

```
Test set accuracy of dt: 0.73
```

KNN

```
KNN_1.fit(X_train, y_train)
knn_y_pred = KNN_1.predict(X_test)
knn_acc_test = accuracy_score(y_test, knn_y_pred)
print('Test set accuracy of dt: {:.2f}'.format(knn_acc_test))
```

```
Test set accuracy of dt: 0.69
```

Random Forest

```
rf.fit(X_train, y_train)
rf_y_pred = rf.predict(X_test)
rf_acc_test = accuracy_score(y_test, rf_y_pred)
print('Test set accuracy of dt: {:.2f}'.format(rf_acc_test))
```

```
Test set accuracy of dt: 0.73
```

Predict Proba Scores

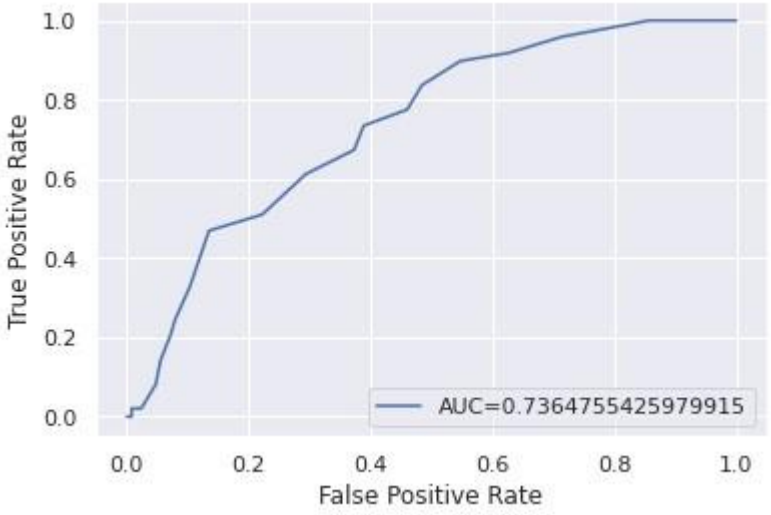
```
y_score1 = dt.predict_proba(X_test)[:,1]
y_score2 = lr.predict_proba(X_test)[:,1]
y_score3 = KNN_1.predict_proba(X_test)[:,1]
y_score4 = rf.predict_proba(X_test)[:,1]
```

```
print('roc_auc_score for DecisionTree: ', roc_auc_score(y_test, y_score1))
print('roc_auc_score for Logistic Regression: ', roc_auc_score(y_test, y_score2))
print('roc_auc_score for KNearest Neighbors: ', roc_auc_score(y_test, y_score3))
print('roc_auc_score for Random Forest: ', roc_auc_score(y_test, y_score4))
```

```
roc_auc_score for DecisionTree:  0.7021379980563655
roc_auc_score for Logistic Regression:  0.7048914804016846
roc_auc_score for KNearest Neighbors:  0.6484450923226434
roc_auc_score for Random Forest:  0.7364755425979915
```

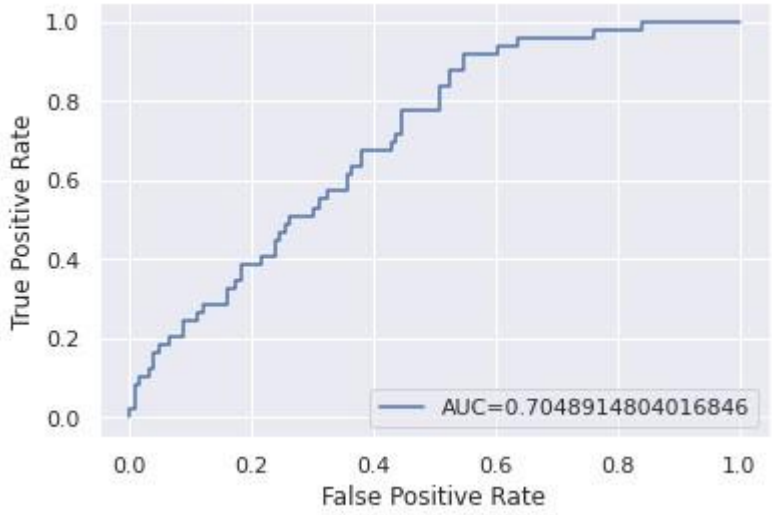
Random Forest ROC Curve

```
#define metrics
y_pred_proba_rf_auc = rf.predict_proba(X_test)[::,1]
fpr_rf, tpr_rf, _ = roc_curve(y_test,  y_pred_proba_rf_auc, pos_label = 2)
rf_auc = roc_auc_score(y_test, y_score4)


#create ROC curve
plt.plot(fpr_rf,tpr_rf,label="AUC="+str(rf_auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



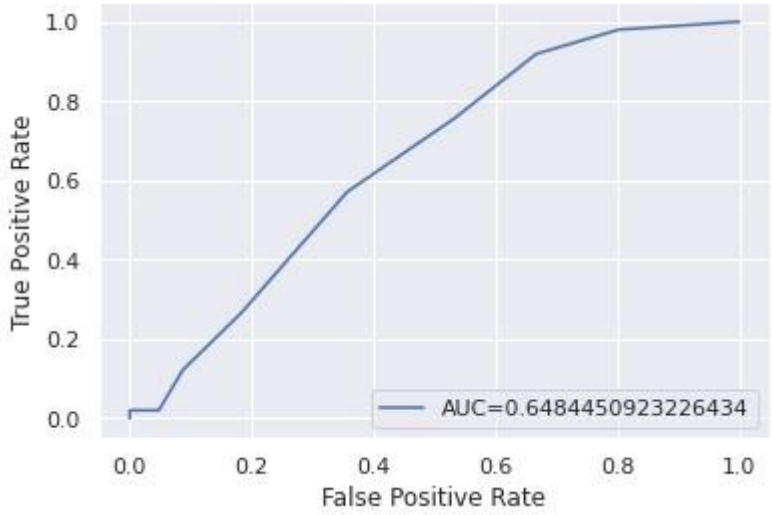
Logistic Regression ROC Curve

```
#define metrics
y_pred_proba_auc_lr = lr.predict_proba(X_test)[::,1]
fpr_lr, tpr_lr, _ = roc_curve(y_test,  y_pred_proba_auc_lr, pos_label = 2)
lr_auc = roc_auc_score(y_test, y_score2)


#create ROC curve
plt.plot(fpr_lr,tpr_lr,label="AUC="+str(lr_auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc =4)
plt.show()
```



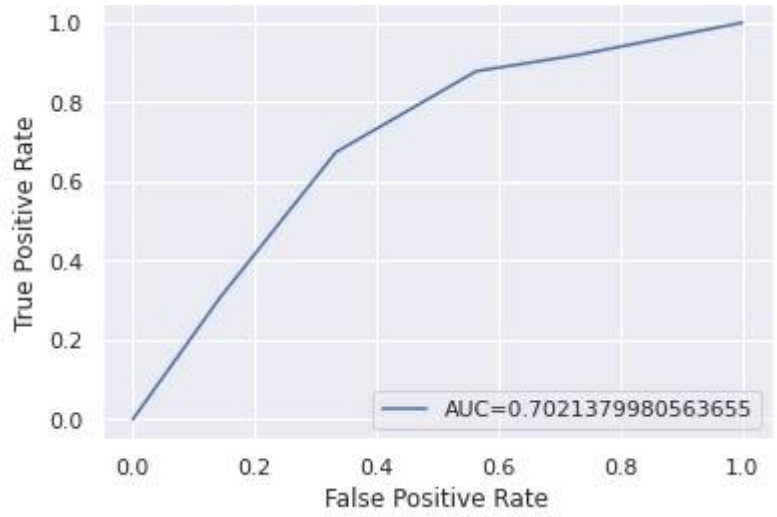
KNN ROC Curve

```
#define metrics
y_pred_proba_auc_knn = KNN_1.predict_proba(X_test)[::,1]
fpr_knn, tpr_knn, _ = roc_curve(y_test,  y_pred_proba_auc_knn, pos_label = 2)
knn_auc = roc_auc_score(y_test, y_score3)

#create ROC curve
plt.plot(fpr_knn,tpr_knn,label="AUC="+str(knn_auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc =4)
plt.show()
```



Decision Tree ROC Curve

```
#define metrics
y_pred_proba_auc_dt = dt.predict_proba(X_test)[::,1]
fpr_dt, tpr_dt, _ = roc_curve(y_test,  y_pred_proba_auc_dt, pos_label = 2)
dt_auc = roc_auc_score(y_test, y_score1)


#create ROC curve
plt.plot(fpr_dt,tpr_dt,label="AUC="+str(dt_auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc =4)
plt.show()
```




```
results = pd.DataFrame({'Model': ['Decision Tree','Logistic Regression',
                                  'KNN','Random Forest Classifier'],
                        'Roc_Score': [roc_auc_score(y_test, y_score1),
                                      roc_auc_score(y_test, y_score2),
                                      roc_auc_score(y_test, y_score3),
                                      roc_auc_score(y_test, y_score4)],
                       'Accuracy_Score': [dt_acc_test, lr_acc_test,knn_acc_test, rf_acc_test]})
df_results = results.sort_values(by='Roc_Score', ascending=False)
df_results
```

	Model	Roc_Score	Accuracy_Score
3	Random Forest Classifier	0.736476	0.731429
1	Logistic Regression	0.704891	0.725714
0	Decision Tree	0.702138	0.702857
2	KNN	0.648445	0.691429