**Problem Statement:**

The given dataset is related to Indian patients who have been tested for a liver disease. Based on chemical compounds (bilrubin,albumin,protiens,alkaline phosphatase) present in human body and tests like SGOT, SGPT the outcome mentioned is whether person is a patient i.e, whether he needs to be diagnosed further or not.

**Coding :**

# -\*- coding: utf-8 -\*-

"""

Created on Mon Feb 4 11:31:25 2019

@author: Admin

"""

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

#import the dataset

data = pd.read\_excel('C:/Users/Admin/Desktop/sahilpy/indian\_patients.xlsx')

data.head()

data.rename(columns={"Albumin.1":"Albumin\_and\_Globulin\_Ratio"},inplace=True)

data.rename(columns = {"Globulin\_Ratio,Class": "Class"},inplace = True)

data.head()

#check for the na values

data.isnull().sum()

data= pd.DataFrame(data)

data.isna().sum()

#sns.pairplot(data)##visualizing the data using pairplot

print(data.columns)

##creating the boxplots , it is a univariate data analysis , below we are checking the outliers in the data

sns.boxplot(data['Age']) #Not contain Outliers

#sns.boxplot(data['Unnamed: 0'])#not contain outliers

sns.boxplot(data['Total\_Bilirubin'])

sns.boxplot(data['Direct\_Bilirubin'])

sns.boxplot(data['Alkaline\_Phosphotase'])

sns.boxplot(data['Alamine\_Aminotransferase'])

sns.boxplot(data['Aspartate\_Aminotransferase'])

sns.boxplot(data['Albumin'])#not contain Outliers

sns.boxplot(data[ 'Total\_Protiens'])

sns.boxplot(data[ 'Albumin\_and\_Globulin\_Ratio'])

##here we are checking the distribution of the data

plt.hist(data['Total\_Protiens'])

plt.hist(data['Albumin\_and\_Globulin\_Ratio'])

plt.hist(data['Total\_Bilirubin'])

plt.hist(data['Direct\_Bilirubin'])

plt.hist(data['Alkaline\_Phosphotase'])

plt.hist(data['Alamine\_Aminotransferase'])

plt.hist(data['Aspartate\_Aminotransferase'])

plt.hist(data['Albumin'])

##here we are imputing the na values with mean and mode

mean\_value=data['Total\_Protiens'].mean()

data['Total\_Protiens']=data['Total\_Protiens'].fillna(mean\_value)

mean\_value1=data['Albumin\_and\_Globulin\_Ratio'].mean()

data['Albumin\_and\_Globulin\_Ratio']=data['Albumin\_and\_Globulin\_Ratio'].fillna(mean\_value1)

data.Gender.mode()

data['Gender']=data['Gender'].fillna('Male')

#we are applying log on the data , it reduces the variability in data it reduces the outliers, the data become compact or dense

data['Albumin\_and\_Globulin\_Ratio']=np.log(data['Albumin\_and\_Globulin\_Ratio'])

data['Total\_Bilirubin']=np.log(data['Total\_Bilirubin'])

data['Direct\_Bilirubin']=np.log(data['Direct\_Bilirubin'])

data['Alkaline\_Phosphotase']=np.log(data['Alkaline\_Phosphotase'])

data['Alamine\_Aminotransferase']=np.log(data['Alamine\_Aminotransferase'])

data['Aspartate\_Aminotransferase']=np.log(data['Aspartate\_Aminotransferase'])

##in below code , we are finding the outliers and remove the outliers

Q1 = data['Total\_Bilirubin'].quantile(0.25)

Q3 = data['Total\_Bilirubin'].quantile(0.75)

IQR = Q3 - Q1

print(IQR)

lower\_limit = Q1 - 1.5 \* IQR

upper\_limit = Q3 + 1.5 \* IQR

print (lower\_limit)

print (upper\_limit)

print (data.Total\_Bilirubin[data.Total\_Bilirubin <= lower\_limit].shape)

print (data.Total\_Bilirubin[data.Total\_Bilirubin >= upper\_limit].shape)

data['Total\_Bilirubin'] = data.query('(@Q1 - 1.5 \* @IQR) <= Total\_Bilirubin <= (@Q3 + 1.5 \* @IQR)')

#mean\_value3=data['Total\_Bilirubin'].median()

#data['Total\_Bilirubin']=data['Total\_Bilirubin'].fillna(mean\_value3)

#data=data.dropna(how = 'any')

#data.dropna(['Total\_Bilirubin'])

Q4 = data['Direct\_Bilirubin'].quantile(0.25)

Q5 = data['Direct\_Bilirubin'].quantile(0.75)

IQR1 = Q5 - Q4

print(IQR1)

lower\_limit1 = Q4 - 1.5 \* IQR1

upper\_limit1 = Q5 + 1.5 \* IQR1

print (lower\_limit1)

print (upper\_limit1)

print (data.Direct\_Bilirubin[data.Direct\_Bilirubin <= lower\_limit1].shape)

print (data.Direct\_Bilirubin[data.Direct\_Bilirubin >= upper\_limit1].shape)

data['Direct\_Bilirubin'] = data.query('(@Q4 - 1.5 \* @IQR1) <= Direct\_Bilirubin <= (@Q5 + 1.5 \* @IQR1)')

#mean\_value4=data['Direct\_Bilirubin'].median()

#data['Direct\_Bilirubin']=data['Direct\_Bilirubin'].fillna(mean\_value4)

#data=data.dropna(how = 'any')

Q5 = data['Alkaline\_Phosphotase'].quantile(0.25)

Q6 = data['Alkaline\_Phosphotase'].quantile(0.75)

IQR2 = Q6 - Q5

print(IQR2)

lower\_limit2 = Q5 - 1.5 \* IQR2

upper\_limit2 = Q6 + 1.5 \* IQR2

print (lower\_limit2)

print (upper\_limit2)

print (data.Alkaline\_Phosphotase[data.Alkaline\_Phosphotase <= lower\_limit2].shape)

print (data.Alkaline\_Phosphotase[data.Alkaline\_Phosphotase >= upper\_limit2].shape)

data['Alkaline\_Phosphotase'] = data.query('(@Q5 - 1.5 \* @IQR2) <= Alkaline\_Phosphotase <= (@Q6 + 1.5 \* @IQR2)')

#mean\_value5=data['Alkaline\_Phosphotase'].median()

#data['Alkaline\_Phosphotase']=data['Alkaline\_Phosphotase'].fillna(mean\_value5)

#data=data.dropna(how = 'any')

Q7 = data['Alamine\_Aminotransferase'].quantile(0.25)

Q8 = data['Alamine\_Aminotransferase'].quantile(0.75)

IQR3 = Q8 - Q7

print(IQR3)

lower\_limit3 = Q7 - 1.5 \* IQR3

upper\_limit3 = Q8 + 1.5 \* IQR3

print (lower\_limit3)

print (upper\_limit3)

print (data.Alamine\_Aminotransferase[data.Alamine\_Aminotransferase <= lower\_limit3].shape)

print (data.Alamine\_Aminotransferase[data.Alamine\_Aminotransferase >= upper\_limit3].shape)

data['Alamine\_Aminotransferase'] = data.query('(@Q7 - 1.5 \* @IQR3) <= Alamine\_Aminotransferase <= (@Q8 + 1.5 \* @IQR3)')

#mean\_value6=data['Alamine\_Aminotransferase'].median()

#data['Alamine\_Aminotransferase']=data['Alamine\_Aminotransferase'].fillna(mean\_value6)

#data=data.dropna(how = 'any')

#data.shape

Q9 = data['Aspartate\_Aminotransferase'].quantile(0.25)

Q10 = data['Aspartate\_Aminotransferase'].quantile(0.75)

IQR4 = Q10 - Q9

print(IQR4)

lower\_limit4 = Q9 - 1.5 \* IQR4

upper\_limit4 = Q10 + 1.5 \* IQR4

print (lower\_limit4)

print (upper\_limit4)

print (data.Aspartate\_Aminotransferase[data.Aspartate\_Aminotransferase <= lower\_limit4].shape)

print (data.Aspartate\_Aminotransferase[data.Aspartate\_Aminotransferase >= upper\_limit4].shape)

data['Aspartate\_Aminotransferase'] = data.query('(@Q9 - 1.5 \* @IQR4) <= Aspartate\_Aminotransferase<= (@Q10 + 1.5 \* @IQR4)')

#mean\_value7=data['Aspartate\_Aminotransferase'].median()

#data['Aspartate\_Aminotransferase']=data['Aspartate\_Aminotransferase'].fillna(mean\_value7)

Q11 = data['Total\_Protiens'].quantile(0.25)

Q12 = data['Total\_Protiens'].quantile(0.75)

IQR5 = Q12 - Q11

print(IQR5)

lower\_limit5 = Q11 - 1.5 \* IQR5

upper\_limit5 = Q12 + 1.5 \* IQR5

print (lower\_limit5)

print (upper\_limit5)

print (data.Total\_Protiens[data.Total\_Protiens <= lower\_limit5].shape)

print (data.Total\_Protiens[data.Total\_Protiens >= upper\_limit5].shape)

data['Total\_Protiens'] = data.query('(@Q11 - 1.5 \* @IQR5) <= Total\_Protiens <= (@Q12 + 1.5 \* @IQR5)')

#mean\_value=data['Total\_Protiens'].mean()

#data['Total\_Protiens']=data['Total\_Protiens'].fillna(mean\_value)

data=data.dropna(how = 'any')

data.shape

X = data.iloc[:,0:-1].values

X=pd.DataFrame(X)

X.head()

Y = data.iloc[:,-1].values

from sklearn.preprocessing import LabelEncoder

labelencoder\_y = LabelEncoder()

Y = labelencoder\_y.fit\_transform(Y)

Y=pd.DataFrame(Y)

Y.head()

##creating the dummies of the data

dum=pd.get\_dummies(X[1])

dum.head()

X = pd.concat([X,dum],axis=1)

X.head()

X = X.drop([1],axis=1)

X.head()

#splitting the data in train and test

from sklearn.model\_selection import train\_test\_split

Xtrain,Xtest,Ytrain,Ytest = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 0)

#we are scaling the data, scaling is important if we have very smaller values and higher values in data , so model gives more preference or weightage to the higher values so it is a good practice to scale the data

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

Xtrain = sc.fit\_transform(Xtrain)

Xtest = sc.transform(Xtest)

'''

#Applying PCA

from sklearn.decomposition import PCA

pca = PCA(n\_components = 3)

Xtrain = pca.fit\_transform(Xtrain)

Xtest = pca.transform(Xtest)

explained\_variance = pca.explained\_variance\_ratio\_

'''

# Applying the logistic regression model

#I built logistic , random forest , xgboost , svm models, but the logistic regression gives more better results among all other models

from sklearn.linear\_model import LogisticRegression

cla = LogisticRegression(random\_state=50)

cla.fit(Xtrain,Ytrain)

# Predicting the Test set results

y\_pred = cla.predict(Xtest)

#creating the confusion matrix and accuracy score to evaluate the model

from sklearn.metrics import confusion\_matrix,accuracy\_score

cm=confusion\_matrix(Ytest,y\_pred)

print(cm)

print(accuracy\_score(Ytest, y\_pred))

plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)

classNames = ['Negative','Positive']

plt.title('Patient Has Liver Disease or Not Has a Disease Confusion Matrix - Test Data')

plt.ylabel('True label')

plt.xlabel('Predicted label')

tick\_marks = np.arange(len(classNames))

plt.xticks(tick\_marks, classNames, rotation=45)

plt.yticks(tick\_marks, classNames)

s = [['TN','FP'], ['FN', 'TP']]

for i in range(2):

for j in range(2):

plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))

plt.show()

'''

from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(n\_estimators=1000,random\_state=1)

#Train the model using the training sets y\_pred=clf.predict(X\_test)

clf.fit(Xtrain,Ytrain)

y\_pred1=clf.predict(Xtest)

print(confusion\_matrix(Ytest,y\_pred1))

print(accuracy\_score(Ytest, y\_pred1))

importances\_rf = pd.Series(clf.feature\_importances\_,index=Xtrain.columns)

sorted\_importances\_rf = importances\_rf.sort\_values(ascending=False)

sorted\_importances\_rf .plot(kind = 'barh',color = 'lightgreen')

plt.show()

Xtrain=pd.DataFrame(Xtrain)

Xtest=pd.DataFrame(Xtest)

Xtrain=Xtrain.iloc[:,3:6]

Xtest=Xtest.iloc[:,3:6]

from xgboost import XGBClassifier

classifier = XGBClassifier()

classifier.fit(Xtrain,Ytrain)

y\_pred2=classifier.predict(Xtest)

print(confusion\_matrix(Ytest,y\_pred2))

print(accuracy\_score(Ytest, y\_pred2))

from sklearn.svm import SVC

classifier = SVC(kernel = 'rbf', random\_state = 0)

classifier.fit(Xtrain, Ytrain)

y\_pred3=classifier.predict(Xtest)

print(confusion\_matrix(Ytest,y\_pred3))

print(accuracy\_score(Ytest, y\_pred3))

'''

**Project Methodology:**

1. We have a liver patient datset , and which is in xlsx format so we imported that dataset using pandas library
2. After that we checked for the na values values in the data , so we had na values in some variables in the dataset , so we imputer that na values with the mode and mean ,
3. After That we made boxplots for detect the outliers in the dataset , so there are several outliers present in the dataset , boxplot is a univariate analysis,
4. After that we applied log onto the dataset, we are applying log on the data , it reduces the variability in data it reduces the outliers, the data become compact or dense
5. Then we treted the outliers , nothing but we removed the outliers from the data
6. After that we divided the data into training and testing for evaluation , we created the dummies of some of the variables,
7. After that we scaled the data, scaling is important if we have very smaller values and higher values in data , so model gives more preference or weightage to the higher values so it is a good practice to scale the data
8. Then we have built multiple models in the like logistic , random forest, svm , xgboost also we applied the pca for dimentionality reduction , but the logistic regression gave the better accuracy among all other models
9. So then we build a confusion matrix , it is used to evaluate the model performance , basically confusion matrix is used in all the classification algorithms , and we have got the 73 percentage of accuracy

[68 0]

[26 4]]

1. 0.7346938775510204

