Chronical disease identification by using Machine Learning Classifiers

First Author#, Second Author\*, Third Author#

#First-Third Department, First-Third University  
Address

1first.author@first-third.edu

3third.author@first-third.edu

\*Second Company  
 Address Including Country Name

2second.author@second.com

Abstract— Hepatitis b is a chronic disease which can develop many long term health related issues including damage of liver and failure of liver and liver cancer and also prone to death in 2018 there are more than 1600 plus deaths are reported to CDC, but there were many other deaths which were not reported it so if a person is affected with hepatitis b and if he left with untreated his life mayor own and he made lose his life but the most important thing that needs to be remember every time is it is just a chronical condition where that can be successfully treated and managed if patient follows proper diet and needs to take care of his liver more and patient can expect long gun full life also if it is left as untreated then it will develop severe scaring on the liver and prone to liver cancer. Basically, there are three stage of this hepatitis b those are prodormal phase, icteric phase and convalescence phase. And this hepatitis b is known as a silent infection as it in fact people without knowing them and we can say it is more dangerous. But it can be identified with help of few medical conditions, and we are going to analyse how to avoid risk of getting hepatitis B in our current research paper.

Keywords— Machine learning, **Chronic disease**, Hepatitis B, Logistic regression, Random Forest classifier, Decision tree classifier.

1. Introduction

HBV is a DNA virus with a loosely circular, partly double-stranded genome and a distinct replication process. Because of the low fidelity reverse transcriptase and the fast replication rate, a single offspring virus in an infected cell may not be identical to the parent genome2. This results in the formation of a virus-related system made up of mutation flies or 'quasispecies,' which are populations of inherently separate, they are very closely having relation with viral variations. As a result, HBV quasispecies have a range of virus-related variations with varying fitness, allowing for fast adaptability to selection pressures such as host immunological factors and antiviral medicines. As a result, HBV variations have an influence on illness development, clinical progression, and responsiveness to treatment interventions. Estimates for infection prevalence (4%), absolute number of infected persons (257 million), and yearly mortality (887000) show the substantial worldwide illness problem owing to HBV. Viral hepatitis, along with hepatitis C virus (HCV), is most important reason of hepatocellular cancer. HBV infection is still a dangerous disease that need to be taken care around the globe. HBV has infected around 257 million people, while CHB affects more than 350 million people. It is widely established that HBsAg seroclearance is an essential prognostic marker during CHB therapy. In persistently HBV-infected individuals, the yearly rate of spontaneous HBsAg seroclearance ranged from 0.45% to 2.38%, demonstrating that HBsAg seroclearance is an uncommon occurrence. Previous research has revealed a link between natural or treatment stimulated HBsAg seroclearance and a good quality diagnosis, improved liver histology, a lower incidence of hepatocellular carcinoma (HCC), as well as longer longevity. As a result, HBsAg seroclearance is an essential objective for improving antiviral medication outcomes.

Previous research has found evidence of significant viral variables and host features of HBsAg seroclearance. Researchers looked at whether low blood HBsAg amounts only or in combination in conjunction with a small amount of blood serum HBV DNA load stayed relevant predictors of HBsAg seroclearance. Age is one of the most significant host factors for HBsAg seroconversion, followed by gender, fatty liver, cirrhosis either present at baseline or developed during follow-up, and baseline alanine aminotransferase (ALT) levels.. To fill a knowledge gap, we applied machine learning techniques rather than conventional models in this work to assess the relationship between available clinical factors and HBsAg seroclearance. In recent years, machine learning algorithms have received a lot of interest in the health area. It has been successfully applied to clinical datasets that are high-dimensional, correlated, nonlinear, and unbalanced in order to extract meaningful information and make precise diagnostic and prediction decisions. However, no current models have been found as having the highest performance for predicting HBsAg seroclearance. In this study, we built many acceptable machine learning models, including Random Forest classifier, Decision tree classifier, and LR, based on the dataset's features (highly dimensional and unbalanced), with the goal of identifying the best one. The primary goal of this project is to find the best machine learning model for predicting HBsAg seroclearance in a retrospective cohort of CHB patients.

.

1. Problem and Data set(s)

Our Objective is the binary classification problem statement. The objective of this task is to identify hepatitis a patient based on his history, and we are going to tell the life prediction of patient so here we are having a target variable called class and it is having a status either die or live.

1. Methods

For analysing the risk of Hepatitis B, we selected three different machine learning algorithms to solve given problem. Those algorithms are Random Forest classifier, Logistic regression classifier and Decision tree classifier.

## Decision tree:

It is a classifier with a tree-like structure, and all of the internal nodes or columns of the data set and branches describe the decisions that the decision tree of outcomes makes. In a decision tree, we will essentially have two nodes that represent the yield of the outcome node. The parent node is not have any additional nodes, only a decision node. A decision tree is a supervised learning approach that can handle both regression and classification tasks on data. Additionally, one of the most elegant algorithms, the decision tree, has a graphic depiction of every step of how it functions, and the decision tree can be visualised In the finished model, the branches from the root node are expanded to create this structure. We will use the cart algorithm, which combines classification and regression, to build the decision tree. The decision tree can include both numerical and category data. The rationale for adopting decision trees is that they typically function similarly to human decision-making processes and can be easily understood and seen due to their tree-like form. Due to the simplicity of decision tree operation, it is much simpler to comprehend how decision tree’s function. To make new decisions, we must first choose the given complete data set. Next, using an attribute selection measure, we must identify the best attribute within the set. Finally, we must split the data set into subsets that contain the best attribute's potential values. Finally, we must create the decision tree no, which contains the best attribute. We may utilise the Gini index or information gain to choose the root node, and we can prune to reduce the chance of overfitting.

## Logistic regression:

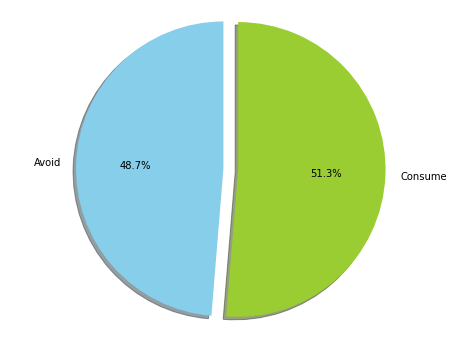
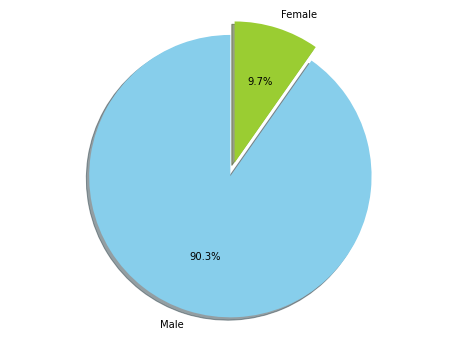
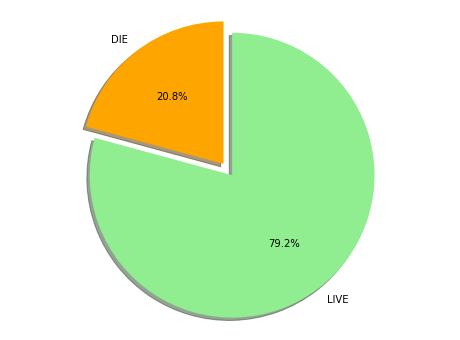
One of the most used machine learning techniques, logistic regression uses input label data to categorize the dependent variable. In essence, logistic regression is employed to address classification issues, and it will use several independent variables together with one dependent variable as the output for training and performing classification. Regression essentially forecasts the results of categorical dependent variables; therefore, the results will be expressed as either yes or no, 0 or 1, or binary output. Therefore, it will return a numerical value and, using the Sigmoid function, transform that value to either 0 or 1, rather than only returning 0 or 1. Except for the final layer, which consists of this probability layer that turns ordinal values into binary values, logistic regression is nearly identical to the linear regression employed in regression procedures. We will fit a sigmoid curve for logistic regression rather than a regression line since this will help us construct probabilities from regression values and explain the likelihood of our issue statement, such as whether the image is a cat or not or whether the patient is pregnant or not. Logistic regression assumes that the dependent variable must be categorical and that the independent variables must not be multicollinear. Logistic regression utilizes the concept of analytical modeling similar to linear regression, but it is utilized to categorize data so that it will considered as classification process. This is the sole distinction between the two. The main distinction between logistic regression and linear regression is that the latter employs the idea of predictive modelling, much like the former, while the former uses it to classify data to fall under a classification procedure.

## Random Forest classifier:

Because each algorithm will generate many decision trees, run over is simply the act of merging multiple classifiers to address a difficult problem and increase the model's performance. Random forest is a prominent machine learning method that belongs to the supervised machine learning branch and may be used in machine learning for both classification and regression issues. The more decision trees we build, the more accurate the classifier will be since random forest is described as a classifier that consists of varying numbers of decision trees that belong to various subsets of the provided data and averages out the predictive accuracy of the data set. We'll receive This random forest has a few assumptions, including that the future values of the data set's variables will have some actual values so that the random forest can provide correct results rather than random ones, and that each decision tree's forecasts will have the fewest fundamental relationships. Random forest was selected for this problem because it produces results with excellent accuracy even for bigger data sets and maintains output stability. The random forest algorithm operates as follows: first, random K points will be chosen for the training set and random K points will be chosen for the output.

There are a few presumptions associated with this random forest; they are that each decision tree's predictions will have the fewest core relationships and that there will be some actual values in the data set's future variables so that the random forest can produce accurate results rather than random ones. Even for larger data sets, the random forest's output is highly accurate, and its output stability is maintained, which are the reasons for choosing random forest classification algorithm for this problem. The functioning of random forest procedure is in this manner: First we select number of decision trees to be built and we will test out the trees that built. And after to that, model will increase the weights to reduce error and this process will be followed up till errors are reduced and by taking average of resultant from all models gives us final result.

1. Experimental setup

For Experimental setup, we are going to use Python as programming software and we are utilizing pandas, Matplotlib, Seaborn and scikit learn libraries for our analysis. When analysing exploratory data, firstly we are checking how many columns and Rows we are having and we are in our data set we are having 20 columns and 154 records for each column and real not having any missing values in our data set and now we have to convert all values into numeric and after converting into memory clear having few missing values in some columns and we treated that missing you with mode and median later we checked skewness for different columns that are available with the data set and all things are seems to be normal it distributed and left, right distributed their showing degree of skinners and for that we are going to apply long transfer to make it normally distributed and from histograms we can see that skewness is present in our data and mainly fixed and also we can observe that some of our variables patient to differentiate according to whether they belongs to class zero OR one distinction is not completely clear there is no near relationship between the variables plotted though some of them we can observed in a train to an interaction we can analyse the relationship between our candy circle variables and numerical variables for this part will take out advantage of pay grade of c burn that will allow us to plant little more freedom to choose X and y variables and here we are choosing swan plot a particular case of scatter plant which do not overlap the point and It is possible to observe that there is no difference in the variables plotted regarding the ANOREXIA status. This can be evidenced by the fact that not only patients from both levels of Class are distributed homogeneously but also there is not difference in the expression of the variables analysed regarding the levels of ANOREXIA. On the other hand, we can see a trend that patients with Class 0 tend to have ascites. However, there is no differences in how the variables are expressed regarding ASCITES status. The last thing that we will explore is if there is any strong correlation between the parameters. For this task, we will use the Pearson correlation coefficient because it is a good parameter to know the strength of the linear relationship between two variables. The importance of performing correlation analysis is our dataset lies on the fact that highly correlated variables can hurt some models or in other cases, could provide little extra information and considering them can be computational expensive without any real benefit. Also, knowing if our variables display a linear relationship can help us choose which machine learning algorithm is more suitable for our data. To perform the correlation analysis with all our variables, we first need to apply the function factorize to the columns containing non continuous variables to obtain a numeric representation of the categorical values contained in the dataset. Here are few visualizations that we applied for available dataset. 

1. Results

In this case, we chosen Logistic regression, Random Forest classifier and Decision tree classifier. Random forest is one of the most used machine learning algorithms Despite its simplicity, flexibility, and ease of use, it produces reliable results. To perform Random Forest and to evaluate the model afterwards, we will load the packages from scikit-learn. As logistic regression performs best at the first instance but no performance improvement after hyper parameter tuning, the model accuracy score is 0.80 i.e.., 80%. So, we elected next good performing algorithm that is Random Forest. As we can observe above, our basic model has an accuracy of 77.19% which tell us that it must be further improved. There are several ways to improve random forest model: gather more data, tune the hyperparameters of the model or choose other models. We will choose the second one, we will now tune the hyperparameters of our random forest classifier.

Model parameters are normally learned during training; however, hyperparameters must be set manually before training. In the case of random forest, hyperparameters include:

n\_estimators: number of trees in the forest

max\_features: maximum number of features in each tree

max\_depth: maximum splits for all trees

bootstrap: whether to implement bootstrap or not to build trees

criterion: assess stopping criteria for decision trees

Of course, when we implement basic random forest, Scikit-learn implements a set of default hyperparameters, but we are not sure if those parameters are the optimal for our problem.

In this point is when we need to consider two concepts: underfitting and overfitting. Underfitting occurs when the model is too simple, and it doesn't fit the data well: it has low variance but high bias. On the other hand, overfitting occurs when the model adjusts too well to the training set and performs poorly in new examples. If we tune the hyperparameters in the training dataset, we would then be prone to overfit our random forest classifier. So instead, we will go back to what was mentioned before: the cross validation.

We will use K-fold cross validation method to tune the hyperparameters: we will perform many iterations on the K-subset cross validation but using different model settings each time. Afterwards, we compare all models and select the best one; then, we will train the best model in the full training set and evaluate it on the testing set. We will take advantage of GridSearchCV package in Scikit-learn to perform this task.

So, after applying random forest to our dataset, we can conclude that our best model was able to predict survival from patients with hepatitis with an accuracy of 77% and a precision and recall of around 80%. This is not the best situation since we want our model to perform better, especially in this case that involves survival of patients. However, the moderate good results could be due to the small database and the large number of missing values.

The classification report of Random Forest Classification model is as follows:

Table

Description automatically generated

Fig: Classification report of Random Forest classifier.

Graphical user interface, application

Description automatically generated

Fig : Confusion matrix of Random Forest Classifier

and this model is having good balance between the recall and precision. The confusion matrix can be observed from the code submitted.

1. Conclusions

This work reveals unique viral variation patterns related with current HBeAg status, however it draws no conclusions about how this status was achieved (there is no previous clinical data) or what the model signifies in terms of patient outcomes (data was not part of a longitudinal study). The same applies to patient profiles that are not represented in the research population, such as HBeAg negative inactive carriers with low HBV DNA burdens, on which we are unable to comment. The goal of this work was not to establish an alternative diagnostic test; instead, the ML model was built based on the known HBeAg status as identified by conventional diagnostic methods. Even though we offer a classification model with excellent discriminative accuracy, this does not necessarily translate to modifications in clinical practise or decision-making. We need prospective trials with serial sampling to catch patients during the seroconversion process and track treatment groups to make such a model applicable. Furthermore, such a model must be calibrated to the target population. However, the general model's success in differentiating HBeAg status in the n = 37 patients undergoing treatment from the Dataset B cohort, which acted as an independent test group, gave us hope. It was able to develop a generic model for HBeAg classification with broad applicability to the clinical population by adding a diversified sample population for feature-selection. In addition, we describe the prevalence of mutants associated with resistance in naive patients. Previous research has demonstrated the existence of these mutants in naive patients.

1. References

[1]

E. Orito *et al.*, “Host-independent evolution and a genetic classification of the hepadnavirus family based on nucleotide sequences.,” *Proceedings of the National Academy of Sciences*, vol. 86, no. 18, pp. 7059–7062, Sep. 1989, doi: 10.1073/pnas.86.18.7059.

[2]

“Classification Model for Hepatitis B Disease Using Supervised Machine Learning Technique,” *Computer Engineering and Intelligent Systems*, May 2022, doi: 10.7176/ceis/13-3-01.

[3]

A. YUSUF and O. AKANDE, “HEPATITIS DISEASES PREDICTION USING MACHINE-LEARNING TECHNIQUES,” *FUDMA JOURNAL OF SCIENCES*, vol. 5, no. 3, pp. 1–8, Nov. 2021, doi: 10.33003/fjs-2021-0503-515.

[4]

A. Orooji and F. Kermani, “Machine Learning Based Methods for Handling Imbalanced Data in Hepatitis Diagnosis,” *Frontiers in Health Informatics*, vol. 10, no. 1, p. 57, Jan. 2021, doi: 10.30699/fhi.v10i1.259.

[5]

H. Chown, “A Comparison of Machine Learning Algorithms for the Prediction of Hepatitis C NS3 Protease Cleavage Sites,” *Journal of Proteomics & Bioinformatics*, vol. 12, no. 5, 2019, doi: 10.35248/0974-276x.19.12.501.

Appendix:

# Data DOWNLOAD : https://archive.ics.uci.edu/ml/Datas/Hepatitis

import pandas as pd

import numpy as np

from pandas import Series, DataFrame

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv('hep.csv')

# In[3]:

df.head()

df.columns = ['class', 'age', 'sex', 'steroid', 'antivirals', 'fatigue', 'malaise',

'anorexia', 'liver\_big', 'liver\_firm', 'spleen\_palable', 'spiders',

'ascites', 'varices', 'bilirubin', 'alk\_phosphate', 'sgot', 'albumin',

'protime', 'histology']

df.head()

df.describe()

df.info()

df['steroid'] = pd.to\_numeric(df['steroid'],errors='coerce')

df['fatigue'] = pd.to\_numeric(df['fatigue'],errors='coerce')

df['malaise'] = pd.to\_numeric(df['malaise'],errors='coerce')

df['anorexia'] = pd.to\_numeric(df['anorexia'],errors='coerce')

df['liver\_big'] = pd.to\_numeric(df['liver\_big'],errors='coerce')

df['liver\_firm'] = pd.to\_numeric(df['liver\_firm'],errors='coerce')

df['spleen\_palable'] = pd.to\_numeric(df['spleen\_palable'],errors='coerce')

df['spiders'] = pd.to\_numeric(df['spiders'],errors='coerce')

df['ascites'] = pd.to\_numeric(df['ascites'],errors='coerce')

df['varices'] = pd.to\_numeric(df['varices'],errors='coerce')

df['bilirubin'] = pd.to\_numeric(df['bilirubin'],errors='coerce')

df['alk\_phosphate'] = pd.to\_numeric(df['alk\_phosphate'],errors='coerce')

df['sgot'] = pd.to\_numeric(df['sgot'],errors='coerce')

df['albumin'] = pd.to\_numeric(df['albumin'],errors='coerce')

df['protime'] = pd.to\_numeric(df['protime'],errors='coerce')

df.info()

df["class"].replace((1,2),(0,1),inplace=True)

df["sex"].replace((1,2),(0,1),inplace=True)

df["age"].replace((1,2),(0,1),inplace=True)

df["steroid"].replace((1,2),(0,1),inplace=True)

df["antivirals"].replace((1,2),(0,1),inplace=True)

df["fatigue"].replace((1,2),(0,1),inplace=True)

df["malaise"].replace((1,2),(0,1),inplace=True)

df["anorexia"].replace((1,2),(0,1),inplace=True)

df["liver\_big"].replace((1,2),(0,1),inplace=True)

df["liver\_firm"].replace((1,2),(0,1),inplace=True)

df["spleen\_palable"].replace((1,2),(0,1),inplace=True)

df["spiders"].replace((1,2),(0,1),inplace=True)

df["ascites"].replace((1,2),(0,1),inplace=True)

df["varices"].replace((1,2),(0,1),inplace=True)

df["histology"].replace((1,2),(0,1),inplace=True)

# In[11]:

df.head()

df.isna().sum()

df['steroid'].mode()

df['steroid'].replace(to\_replace=np.nan,value=1,inplace=True)

df['fatigue'].mode()

df['fatigue'].replace(to\_replace=np.nan,value=0,inplace=True)

df['malaise'].mode()

df['malaise'].replace(to\_replace=np.nan,value=1,inplace=True)

df['anorexia'].mode()

df['anorexia'].replace(to\_replace=np.nan,value=1,inplace=True)

df['liver\_big'].mode()

df['liver\_big'].replace(to\_replace=np.nan,value=1,inplace=True)

df['liver\_firm'].mode()

df['liver\_firm'].replace(to\_replace=np.nan,value=1,inplace=True)

df['spleen\_palable'].mode()

df['spleen\_palable'].replace(to\_replace=np.nan,value=1,inplace=True)

df['spiders'].mode()

df['spiders'].replace(to\_replace=np.nan,value=1,inplace=True)

df['ascites'].mode()

df['ascites'].replace(to\_replace=np.nan,value=1,inplace=True)

df['varices'].mode()

df['varices'].replace(to\_replace=np.nan,value=1,inplace=True)

df['bilirubin'].skew(axis=0,skipna = True)

df['bilirubin'].median()

df['bilirubin'].replace(to\_replace=np.nan,value=1,inplace=True)

df['alk\_phosphate'].skew(axis=0,skipna = True)

df['alk\_phosphate'].median()

df['alk\_phosphate'].replace(to\_replace=np.nan,value=85,inplace=True)

df['sgot'].skew(axis=0,skipna = True)

df['sgot'].median()

df['sgot'].replace(to\_replace=np.nan,value=58,inplace=True)

df['albumin'].skew(axis=0,skipna = True)

df['albumin'].median()

df['albumin'].mean()

df['albumin'].replace(to\_replace=np.nan,value=4,inplace=True)

df['protime'].skew(axis=0,skipna = True)

print("Here skewness is near to symmetri, so we can check both mean and median")

df['protime'].median()

df['protime'].mean()

df['protime'].replace(to\_replace=np.nan,value=61,inplace=True)

print("Now we filled all the null values.")

# In[14]:

plt.figure(figsize=(6,3.5))

plt.subplot(1, 2, 1)

sns.distplot(df['sgot'],

kde\_kws={"color":"blue","lw":1.5,"alpha":0.8},

hist\_kws={"color":"green","alpha":0.3})

plt.subplot(1, 2, 2)

sns.distplot(df['alk\_phosphate'],

kde\_kws={"color":"red","lw":1.5,"alpha":0.8},

hist\_kws={"color":"pink","alpha":0.6})

sns.despine();

# In[15]:

plt.figure(figsize=(7,3.5))

plt.subplot(1, 2, 1)

sns.distplot(df['bilirubin'],

kde\_kws={"color":"green","lw":1.5,"alpha":0.8},

hist\_kws={"color":"lightblue","alpha":0.8})

sns.despine()

plt.subplot(1, 2, 2)

sns.distplot(df['albumin'],

kde\_kws={"color":"red","lw":1.5,"alpha":0.8},

hist\_kws={"color":"orange","alpha":0.3})

sns.despine();

g = sns.pairplot(df, x\_vars = ['bilirubin', 'protime', 'alk\_phosphate', 'sgot', 'albumin'],

y\_vars = ['bilirubin', 'protime', 'alk\_phosphate', 'sgot', 'albumin'],

hue = 'class',

kind= 'scatter',

palette = 'husl',

size = 2,

plot\_kws={"s": 35, "alpha": 0.8})

g.fig.get\_children()[-1].set\_bbox\_to\_anchor((0.05, 0.9, 0.18, 0.1));

graph = sns.PairGrid(df,

x\_vars=["anorexia", "ascites"],

y\_vars=['bilirubin', 'protime', 'alk\_phosphate', 'sgot', 'albumin'],

hue = 'class')

graph.map(sns.swarmplot, s = 6)

graph.add\_legend(frameon=True, bbox\_to\_anchor=(0.33, 0.96));

df.describe()

df.head(10)

die =len(df[df['class'] == 0])

live = len(df[df['class']== 1])

plt.figure(figsize=(8,6))

labels = 'DIE','LIVE'

sizes = [die,live]

colors = ['orange', 'lightgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

male =len(df[df['sex'] == 0])

female = len(df[df['sex']==1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'Male','Female'

sizes = [male,female]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

no =len(df[df['steroid'] == 0])

yes = len(df[df['steroid']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'Avoid','Consume'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'steroid')

plt.show()

no =len(df[df['antivirals'] == 0])

yes = len(df[df['antivirals']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'Avoid','Consume'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'antivirals',palette='GnBu')

plt.show()

no =len(df[df['fatigue'] == 0])

yes = len(df[df['fatigue']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'Never Exausted','Was Excausted'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'fatigue',palette='BrBG')

plt.show()

no =len(df[df['malaise'] == 0])

yes = len(df[df['malaise']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'Never in Discomfort','Was in Discomfort'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'malaise',palette='RdPu')

plt.show()

no =len(df[df['anorexia'] == 0])

yes = len(df[df['anorexia']== 1])

plt.figure(figsize=(8,6))

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'anorexia',palette='RdPu')

plt.show()

no =len(df[df['liver\_big'] == 0])

yes = len(df[df['liver\_big']== 1])

plt.figure(figsize=(8,6))

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'liver\_big',palette='RdPu')

plt.show()

no =len(df[df['liver\_firm'] == 0])

yes = len(df[df['liver\_firm']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'liver\_firm',palette='RdPu')

plt.show()

no =len(df[df['spleen\_palable'] == 0])

yes = len(df[df['spleen\_palable']== 1])

plt.figure(figsize=(8,6))

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'spleen\_palable',palette='RdPu')

plt.show()

no =len(df[df['spiders'] == 0])

yes = len(df[df['spiders']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'spiders',palette='RdPu')

plt.show()

no =len(df[df['ascites'] == 0])

yes = len(df[df['ascites']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'ascites',palette='RdPu')

plt.show()

no =len(df[df['varices'] == 0])

yes = len(df[df['varices']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'varices',palette='RdPu')

plt.show()

plt.figure(figsize=(10,8))

sns.scatterplot(x='age',y='bilirubin',data = df,hue = 'class')

plt.title('Bilirubin test values according to AGE')

plt.show()

plt.figure(figsize=(10,8))

sns.scatterplot(x='age',y='alk\_phosphate',data = df,hue = 'class')

plt.title('alk\_phosphate test values according to AGE')

plt.show()

plt.figure(figsize=(10,8))

sns.scatterplot(x='age',y='sgot',data = df,hue = 'class')

plt.title('sgot test values according to AGE')

plt.show()

plt.figure(figsize=(10,8))

sns.scatterplot(x='age',y='albumin',data = df,hue = 'class')

plt.title('albumin test values according to AGE')

plt.show()

plt.figure(figsize=(10,8))

sns.scatterplot(x='age',y='protime',data = df,hue = 'class')

plt.title('protime test values according to AGE')

plt.show()

no =len(df[df['histology'] == 0])

yes = len(df[df['histology']== 1])

plt.figure(figsize=(8,6))

# df to plot

labels = 'NO','YES'

sizes = [no,yes]

colors = ['skyblue', 'yellowgreen']

explode = (0.1, 0) # explode 1st slice

# Plot

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.show()

plt.figure(figsize=(15,6))

sns.countplot(x='age',data = df, hue = 'histology',palette='GnBu')

plt.show()

plt.figure(figsize=(15,12))

sns.heatmap(df.corr(), cmap='coolwarm',linewidths=.1,annot = True)

plt.show()

# In[52]:

# Models from Scikit-Learn

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import roc\_curve, auc, accuracy\_score

from sklearn.metrics import precision\_recall\_curve, confusion\_matrix

x = df.iloc[:, df.columns != 'class']

y = df.iloc[:, df.columns == 'class']

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(x, y, test\_size = 0.2,

random\_state = 42)

models = {"Logistic Regression": LogisticRegression(),

"KNN": KNeighborsClassifier(),

"Random Forest": RandomForestClassifier(),

"Decision Tree": DecisionTreeClassifier()}

# Create a function to fit and score models

def fit\_and\_score(models, X\_train, X\_test, y\_train, y\_test):

"""

Fits and evaluates given machine learning models.

models : a dict of differetn Scikit-Learn machine learning models

X\_train : training data (no labels)

X\_test : testing data (no labels)

y\_train : training labels

y\_test : test labels

"""

# Set random seed

np.random.seed(42)

# Make a dictionary to keep model scores

model\_scores = {}

# Loop through models

for name, model in models.items():

# Fit the model to the data

model.fit(X\_train, y\_train)

# Evaluate the model and append its score to model\_scores

model\_scores[name] = model.score(X\_test, y\_test)

return model\_scores

# In[56]:

model\_scores = fit\_and\_score(models=models,

X\_train=X\_train,

X\_test=X\_test,

y\_train=Y\_train,

y\_test=Y\_test)

model\_scores

# In[57]:

from warnings import simplefilter

from sklearn.exceptions import ConvergenceWarning

simplefilter("ignore", category=ConvergenceWarning)

param\_grid = [

{'penalty' : ['l1', 'l2', 'elasticnet'],

'C' : np.logspace(-4, 4, 20),

'solver' : ['lbfgs','newton-cg','liblinear'],

'max\_iter' : [500, 1000, 1500]

}

]

parameters\_optimize = {

'max\_features': ['auto', 'sqrt', 'log2', None],

'max\_depth': [2,3, 4],

'criterion': ['gini', 'entropy'],

'bootstrap': [True, False],

'n\_estimators': [2, 5, 10, 15, 20]

}

lr = LogisticRegression(random\_state=10)

model = lr.fit(X\_train, Y\_train)

# In[61]:

gsc = GridSearchCV(lr, param\_grid = param\_grid, cv = 10,verbose=True, n\_jobs=-1)

tuned\_model = gsc.fit(X\_train,Y\_train)

# In[62]:

tuned\_model.best\_estimator\_

# In[63]:

print (f'Accuracy - : {tuned\_model.score(X\_test,Y\_test):.3f}')

# In[64]:

y\_pred = tuned\_model.predict(X\_test)

print(accuracy\_score(Y\_test,y\_pred))

random\_forest\_hyp = RandomForestClassifier()

random\_forest\_search = GridSearchCV(random\_forest\_hyp,

cv = 20,

param\_grid = parameters\_optimize,

n\_jobs = 3)

random\_forest\_search.fit(X\_train, Y\_train)

print('The best parameteres after GridSearchCV', random\_forest\_search.best\_params\_)

# In[66]:

random\_forest\_hyp.set\_params(bootstrap = True,

criterion = 'entropy',

max\_depth = 4,

max\_features = 'sqrt',

n\_estimators = 15);

# In[67]:

random\_forest\_hyp.fit(X\_train, Y\_train);

y\_predicted\_grid = random\_forest\_hyp.predict(X\_test)

accuracy\_grid = accuracy\_score(Y\_test, y\_predicted\_grid)\*100

print(round(accuracy\_grid, 2), '%')

# In[68]:

plt.figure()

random\_confusion = confusion\_matrix(Y\_test, y\_predicted\_grid)

sns.heatmap(random\_confusion, annot = True);

# In[69]:

fpr, tpr, \_= roc\_curve(Y\_test, y\_predicted\_grid)

auc\_random\_grid = auc(fpr, tpr)

print(auc\_random\_grid)

# In[70]:

plt.figure()

plt.plot(fpr, tpr, color ='Green', linewidth = 1)

plt.title('ROC curve for Random Forest')

plt.plot([0,1], [0,1], 'k--', lw = 1)

plt.plot([0,0], [1,0], 'k--', lw = 1, color = 'black')

plt.plot([1,0], [1,1], 'k--', lw = 1, color = 'black')

plt.xlabel('False Positive rate')

plt.ylabel('True Positive rate');