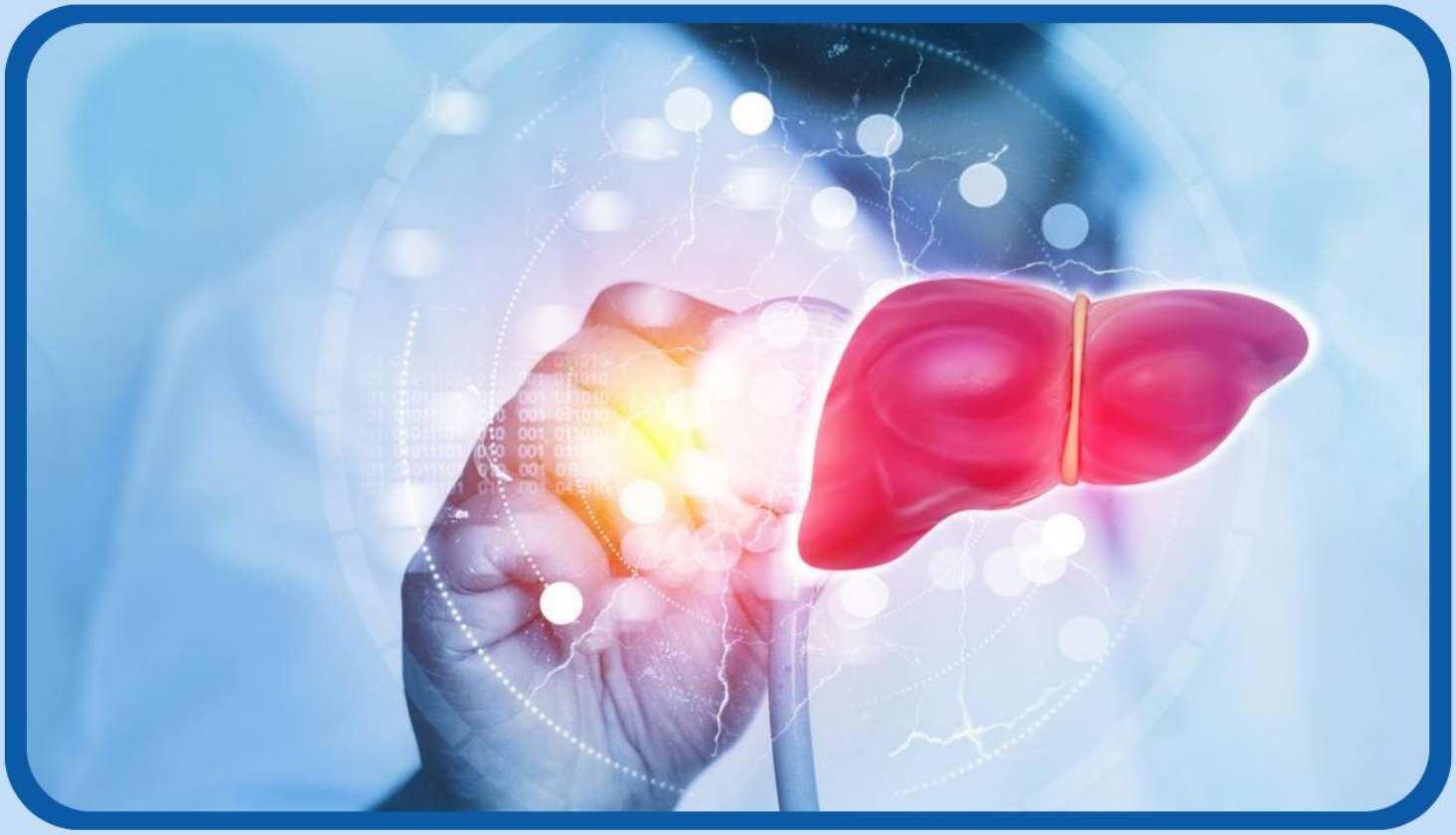
ADVANCED ANALYSIS OF NON ALCOHOLIC FATTY LIVER PATIENTS



GROUP 05

SANUJI DEVASURENDRA -S14810
SAHAN MADHUSHANKA - S14835
ISINI ASALYA - S15023

Abstract

Non-Alcoholic Fatty Liver Disease (NAFLD) is one of the most common chronic liver diseases, affecting approximately 25% of the global population. This report brings out the advanced analysis of the Non-alcoholic Fatty Liver dataset sourced from the Kaggle website. this exploratory analysis brings out a study on predictors and the response, which is the status of a patient, with the objective of predicting whether a NAFLD patient may experience mortality. Therefore, to accomplish the task first k-protype clustering was used to find out the observation clusters and there were no any observation clusters. Then several machine learning models were fitted and the confusion matrix, f1 score, accuracy, recall, precision score were used to assess the models. Logistic regression classifier, logistic ridge regression classifier, random forest classifier, XG Boosting classifier, Gradient Boosting classifier, Naïve Bayes classifier and support vector machine classifier models were fitted, and Gradient Boosting was identified as the best model with the best class wise accuracy. Then the most important variables were identified for the best model using the variable importance and the partial dependency plots were used to discuss the associations of the important variables.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

NAFLD (Non-Alcoholic Fatty Liver Disease), FLD (Fatty Liver Disease), HDL (High-Density Lipoprotein), LDL (Low-Density Lipoprotein)

1. Introduction

Non-Alcoholic Fatty Liver Disease (NAFLD) has become a major health concern due to its prevalence, difficulties in diagnosis, complex pathogenesis, and lack of approved therapies. It has affected millions of people worldwide. As the name suggests, this complexity develops from a build-up of fat in the liver.

Since this disease is a spectrum of diseases ranging from simple hepatic steatosis to liver cancer, there are four main stages.

1. Simple fatty liver (steatosis) - a largely harmless build-up of fat in the liver cells that may only be diagnosed during tests carried out for another reason.
2. Non-Alcoholic Steatohepatitis (NASH) – a more serious form of NAFLD where the liver has become inflamed.
3. Fibrosis - a persistent inflammation that causes scar tissue around the liver and nearby blood vessels, but the liver is still able to function normally.

4. Cirrhosis - the most severe stage, occurring after years of inflammation, where the liver shrinks and becomes scarred and lumpy. This damage is permanent and can lead to liver failure (where your liver stops working properly) and liver cancer.

According to the study of Golabi et al. (2022), it was identified that among solid cancers, liver cancer was the leading cause of cancer-related mortality, where cirrhosis increases the risk of developing liver cancer. Hence, proper attention to this disease is essentially required due to its widespread distribution across the world, which could lead to mortality in the worst case. Also, it should be taken seriously, as proper diagnosis is not specifically introduced for NAFLD.

2. Description of the Question

Even though NAFLD does not cause mortality at the earliest stage, the problem is that there are still no approved diagnostic procedures to cure this disease completely. As a result of that, unless you control lifestyle modifications by having a healthy diet, regular exercise, taking medications to treat cholesterol, high blood pressure, and diabetes, losing weight, and avoiding alcohol, the situation could be worse.

Hence, at present, multiple researchers around the world are trying to come up with an advanced therapy for this disease in order to minimize the fatality rate. Hence, as a part of that journey, identifying which complications are affecting mortality plays a crucial role.

Therefore, this exploratory analysis aims **to predict whether a NAFLD patient may experience mortality** and the sub-objectives are to,

1. Perform an exploratory data analysis to identify the most influential predictors of the response.
2. Develop a predictive model that incorporates the most significant factors to forecast the patient's status.

3. Description of the Dataset

The Non-Alcoholic Fatty Liver Disease (NAFLD) dataset is a Kaggle sourced dataset that contains three sub datasets where all the data has been collected from a population cohort study of NAFLD patients. However, to accomplish the objectives of this study, the "nafld1" and "nafld2" datasets were utilized as they have a meaningful interpretation towards the research objectives.

Description of "nafld1" dataset

This dataset contained 17549 observations, 9 variables, and one unlabeled variable that had no meaningful interpretation for our analysis. Hence, that variable was dropped in further studies. Therefore, the description of the dataset is as follows.

No.	Variable Name	Description
1	id	Subject identifier
2	age	Age at entry to the study
3	male	0 = female; 1 = male

4	weight	Weight in kg
5	height	Height in cm
6	bmi	Body Mass Index of the person
7	case.id	The ID of the NAFLD case to which this subject is matched
8	futime	The time to death or last follow up
9	Status (The Response)	0 = alive at last follow up; 1 = dead

Table 3.1 Description of "nafld1" dataset

Description of "nafld2" dataset

This dataset carried 400124 observations, which is a collection of time dependent repeated measures on 15205 individuals for two tests: HDL and cholesterol. Furthermore, it contained four variables, as described below.

No.	Variable Name	Description
1	id	Subject identifier
2	days	Days since index date
3	test	The type of value recorded
4	value	The numeric value

Table 3.2 Description of "nafld2"

3.1 Data Pre-processing

In the data pre-processing, the two datasets (nafld1 and nafld2) were initially merged together using the variable "id" to obtain more meaningful results. Since the nafld2 contained multiple time dependent observations for each ID with respect to the index date for total cholesterol and High Density lipoprotein (HDL), we categorized the data as "pre data" and "post data" with respect to the index date while data frames were created for each test. Then the average values were taken for each ID for their two sets of tests. Afterwards, the two datasets were merged using the common ID.

Next, the problem was handling the missing values. There were basically two types of missing values related to variables.

1. Missing values related to weight, height, and BMI.
2. Missing values related to values of two tests.

3.1.1 Treatment for missing values related to weight, height and BMI.

First, we tried to identify the factors that significantly affect the data on weight and height by performing an exploratory analysis of their distribution. Then it was observed that there is no significant association between weight, height, and age.

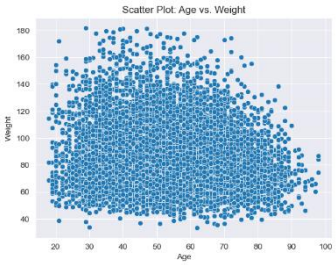


Figure 3.1.1 Scatterplot of Weight by Age

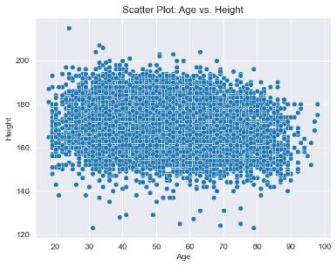


Figure 3.1.2 Scatterplot of Height by Age

However, when gender was considered, as illustrated by Figures 3.1.3 and 3.1.4, a clear variation could be observed for both weight and height based on whether the individual is male or female. Hence, missing values for weight and height were grouped by using the average weight and height for each gender.

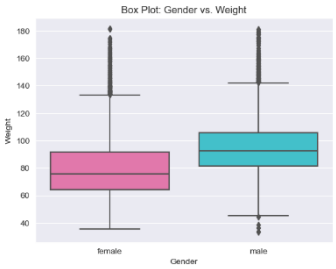


Figure 3.1.3 Boxplot of Weight by Gender

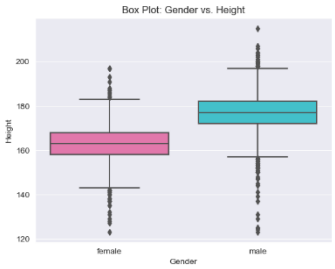


Figure 3.1.4 Boxplot of Height by Gender

After adjusting for missing values for weight and height, the distributions were as in Figures 3.1.5 and 3.1.6. Therefore, we can clearly observe that the two distributions of height and weight are not the same for males and females.

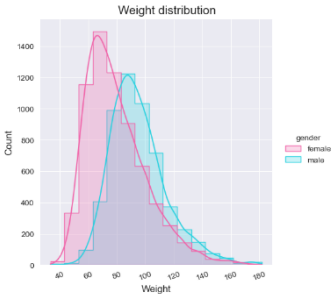


Figure 3.1.5 Distribution of Weight by Gender

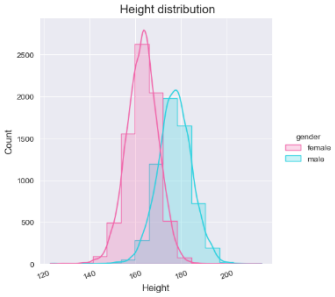


Figure 3.1.6 Distribution of Height by Gender

Additionally, several new variables were created utilizing nafld2," which contained repeated time dependent values for two tests, Cholesterol and HDL, per individual based on index date as follows:

• Average HDL level before the index date	• Average HDL level after the index date	• Average HDL level throughout the follow up
• Average cholesterol level before the index date	• Average cholesterol level after the index date	• Average cholesterol level throughout the follow up

Table 3.1.1 Average Levels of Cholesterol and HDL based on Index Date

3.1.2 Treatment for missing values related to values of two tests (Cholesterol and HDL)

However, the problem raised after creating the variables in Table 3.1.1 is that there were multiple rows (observations) that became missing values, as initially there were no records or follow ups after the index date. Hence, those such observations were deleted, as if we used them to be imputed using the means of each distribution, it could mislead the whole analysis. Moreover, since the dataset carried a fair number of total observations, the decision to drop the missing values was taken.

Therefore, after the process of data pre-processing was carried out, a dataset with 11543 observations and 14 variables was obtained.

4. Important Results from the descriptive analysis

As for the objective of developing a predictive model, it is essential to understand how an individual's mortality due to NAFLD is associated with its determining factors.

- The response "status" predicts whether the person is alive or not at the end of a 20-year community study. According to Figure 4.1, it can be observed that the majority of the patients were alive at the end of the study. This indicates that there is a class imbalance in the dataset.
- Following that, it was identified that the age of a patient has a significant impact on death.

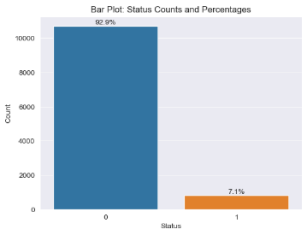


Figure 4.1: Bar Plot of Status

- Also, it was identified that, a person's obesity and weight have no impact on the disease's severity.
- Considering the clinical data, the graphs demonstrated no change in the mean levels of pre or post total cholesterol and HDL cholesterol levels.

- According to Figure 4.2, it was observed that within a shorter period of time from the index date, the majority of patients had died.
- In the survival analysis, it was identified from the Kaplan-Miere survival curve in Figure 4.3 that even after 7000 days (approximately 19 years of time), an estimated 70% of the NAFLD population is still able to survive. Further, it was observed through the moderate steepness of the survival curve, which suggests a considerable rate of mortality.
- Since there are no observational clusters, according to figure 4.5, score plot of the partial least squares, cluster analysis can be disregarded. Inaddition, percentage variation explained by first 2 principal components are 0.0797.
- Further, correlations among the variables can be identified through the loading plot of partial least squares. Additionally, the existence of variable clusters can be explained by the loading plot.

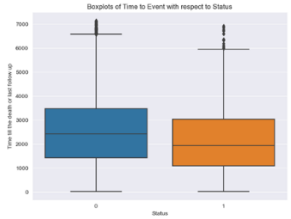


Figure 4.2: Boxplot of Time to Event (Death/ Last follow-up) by Status

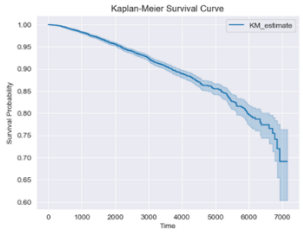


Figure 4.3 Kaplan-Meier Survival Curve

- Furthermore, as shown in Figure 4.7, it was identified that there are high correlations between the clinical variables and also between the BMI variable and the height_cm and weight_kg variables, while other correlations are negligible.

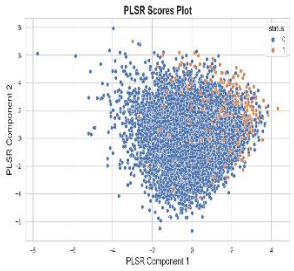


Figure 4.5: Score plot using PLS

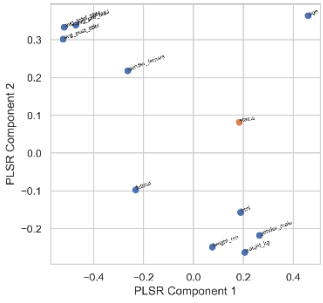


Figure 4.6: Loadings plot of PLS

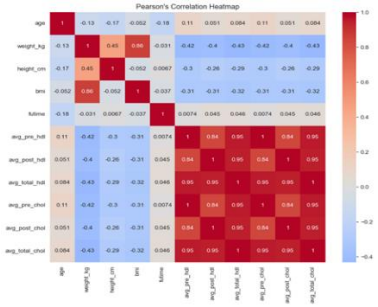


Figure 4.7: Correlations of numerical variables

- Moreover, considering the following Table 4.1 and Figure 4.7, it was identified that the age variable shows a somewhat strong association with the response variable, while all the other numerical variables don't show a high correlation with the response variable. All the clinical test variables are negatively correlated with the response, along with the follow up time, height, and weight variables. Among the qualitative variables, it was identified that, under the 5% level of significance, there is a significant association between the two variables, gender and status.

Variables	Spearman's Correlation Coefficient	Variables	Spearman's Correlation Coefficient
Age and Status	0.25868	avg_pre_hdl and status	-0.04559
weight_kg and status	-0.00496	avg_post_hdl and status	-0.07593
height_cm and status	-0.03787	avg_total_hdl and status	-0.05906
bmi and status	0.014366	avg_pre_chol and status	-0.04559
futime and status	-0.07704	avg_post_chol and status	-0.075934
		avg_total_chol and status	-0.059061

Table 4.1 Spearman's Correlation Coefficients between the numerical predictors and the Response variable

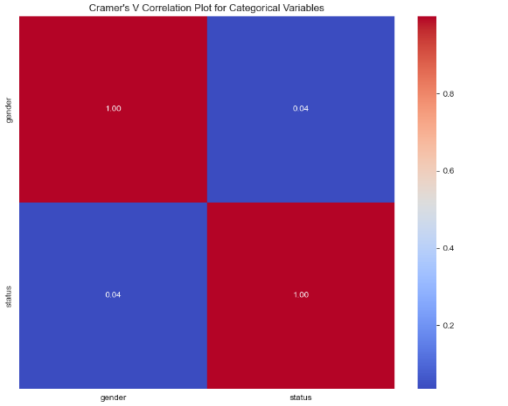


Figure 4.7: Cramer's V correlation heatmap between status and gender variables

5. Important Results from the advanced analysis

5.1 Cluster Analysis

In the descriptive analysis, from the score plot, it was identified that there are no observation clusters in the dataset. However, we further analyzed the dataset for observation clusters as the percentage variation explained by first 2 principal components are 0.0797 which is obviously less than 85%. Hence, the K-prototype clustering technique was utilized to test for observational clusters, which can handle mixed data

types (both numerical and categorical). According to Silhouette score, perfectly well separated clusters are given by scores which are closer to 1.

However, according to our dataset, Figure 5.1.1 shows that silhouette values are even less than 0.5, indicating overlapping clusters, and performing a cluster analysis might not help increase the model's accuracy. Therefore, it has been identified that there are no observation clusters in our dataset.

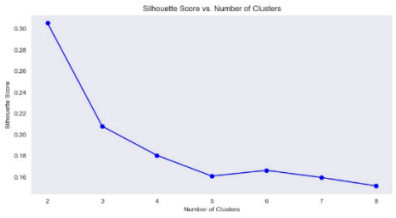


Figure 5.1.1: Silhouette score vs no of Clusters in the cluster analysis

No. of Clusters	2	3	4	5	6
Silhouette score	0.30523	0.20773	0.18030	0.160778	0.16617

Table 5.1: Silhouette Scores obtained from the k-prototype clustering

5.2. Model Fitting

Since our study focuses on classifying the NAFLD patients' status (1- dead, 0- alive) binary classification models were applied to the dataset.

5.2.1 Logistic Ridge Classifier

Through the descriptive analysis, it was observed the presence of the multicollinearity in the dataset. Hence, as a remedy to this issue the shrinkage technique of Ridge was performed. According to the results of Table 5.2.1.1, SMOTE improves in identifying the death of a patient more accurately, where both classes have a recall of more than 0.5, even with some degree of loss of accuracy.

With no resampling technique			SMOTE		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.93	1.00	0.03	0.83	0.85	0.59

Table 5.2.1.1: Table of overall accuracy and class wise recall results of Logistic Ridge Regression Classifier

5.2.2 Random Forest Classifier

Random forest is an ensemble learning technique that combines multiple decision trees to improve the accuracy and robustness of the prediction. which is relatively easy to use and interpret. It seems like SMOTE improves the model according to table 5.2.2.1, but death of patient is not sufficiently explained in both techniques.

With no resampling technique			SMOTE		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.93	1.00	0.06	0.88	0.92	0.32

Table 5.2.2.1: Table of overall accuracy and class wise recall results of Random Forest Classifier

5.2.3 Support Vector Machine Classifier

SVM is a vastly considered classification technique that categorizes data by determining the best decision boundary that best divides the data points into classes. The following results were obtained for our dataset.

With no resampling technique			SMOTE		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.93	0.99	0.11	0.86	0.91	0.20

Table 5.2.3.1: Table of overall accuracy and class wise recall results of Random Forest Classifier

5.2.4. Naive Bayes Classifier

Naive Bayes Classifier is one of the simple and most effective Classification algorithms which predicts on the basis of the probability. The following results were obtained for our dataset. Here, it could observe a drop in correctly identifying surviving patients by the SMOTE technique, however, which is not less than 0.5, according to Table 5.2.4.1.

With no resampling technique			SMOTE		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.92	0.97	0.15	0.62	0.62	0.71

Table 5.2.4.1: Table of overall accuracy and class wise recall results of Naïve Bayes Classifier

5.2.5. Gradient Boosting Classifier

Gradient Boosting is a functional gradient algorithm which combines several weak learning models to produce a powerful predicting model. The following results were obtained for our dataset.

With no resampling technique			SMOTE		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.93	1.00	0.01	0.85	0.87	0.55

Table 5.2.5.1: Table of overall accuracy and class wise recall results of Gradient Boosting Classifier

5.2.6. XG Boosting Classifier

The XG Boosting classifier is a machine learning technique for structured and tabular data which is an implementation of gradient boosted decision trees aimed for speed and performance. The following results were obtained for our dataset.

With no resampling technique			SMOTE		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.94	1.00	0.09	0.88	0.92	0.33

Table 5.2.6.1: Table of overall accuracy and class wise recall results of Gradient Boosting Classifier

6. Discussion and Conclusions

6.1. Pre-processing for advanced analysis

In the descriptive analysis, it was identified that our response variable is imbalanced. Therefore, the Synthetic Minority Oversampling Technique (SMOTE) was used as a balancing technique. In the model

fitting, first it was tried without SMOTE, and then SMOTE was applied in order to improve the accuracy of the predictions. In addition, from the existing research studies, it was identified that BMI has a significant impact on NAFLD. Hence, it was considered as a new variable created using height and weight values.

6.2. The Best Model

According to Section 5.2, it could be identified that some models less precisely predict the class of death due to NAFLD. However, considering the models with more than 50% ability to correctly identify the death, the logistic ridge classifier with SMOTE is considered the best model.

Accuracy	Recall	
	Class 0	Class 1
0.83	0.85	0.59

Table 6.2.1: Table of overall accuracy and class wise results of Best Model – Logistic Ridge Classifier

However, to further analyze what factors cause to death due to NAFLD, the gradient boosting classifier was utilized, and Figure 6.2.1 shows the major key factors that affect the mortality of NAFLD patients.

1. Age
2. Gender
3. Follow up time.
4. Average post HDL
5. BMI

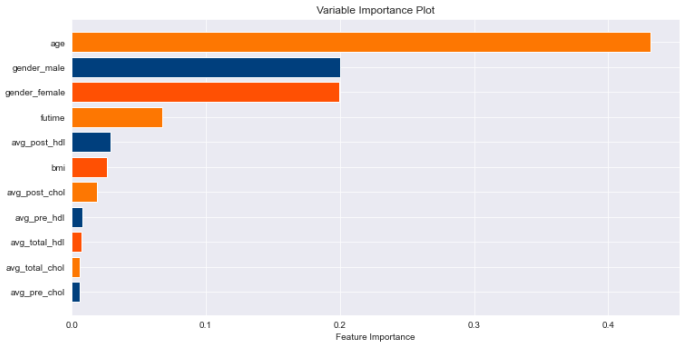


Figure 6.2.1: Variable Importance Plot of Gradient Boosting

6.3. Key Factors that Affect the death of the NAFLD patients

6.3.1 Age

According to a study on established and emerging factors affecting the progression of NAFLD undertaken by Linköping University in Sweden older age increases the impact of mortality. In general, also, we know that age becomes a crucial factor in determining whether a patient dies from any disease or not. Therefore, the same norm could be applied to NAFLD, as it is natural for older people to experience serious illnesses and pass away since the liver experiences significant structural and functional changes with aging.

This can be further explained by the PD plot in Figure 6.3.1, where the higher the age, the higher the risk of mortality.

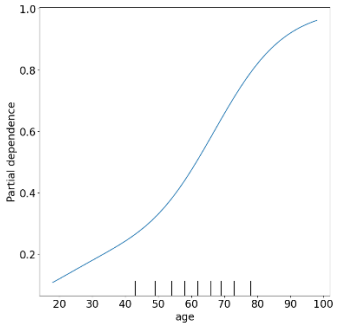


Figure 6.3.1.1: PD Plot of Age

6.3.2 Gender

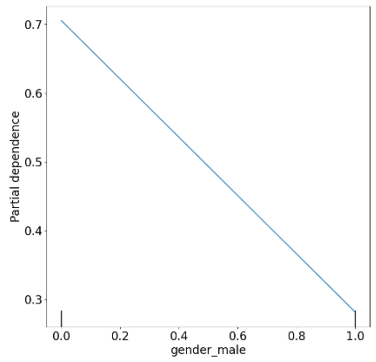


Figure 6.3.2.1: PD Plot of Gender_male

According to a cohort study carried in Japan for 12 years, it was noted that there is two times prevalence for men when compared to female patients with NAFLD. However, another study suggests that, in presence of cardiovascular events, there is a high chance of female NAFLD patients to at risk of death rather than male patients, which can used to explain the higher risk for female patients than the male patients in the PD plot of Figure 6.3.2.1.

6.3.3 Follow-up Time

Follow up time refers to the following up patients after identifying that they are NAFLD patients.

The resources state that the result of not having adequate time to get properly treated for NAFLD, as untreated FLD can progress to cirrhosis or even liver cancer. Therefore, late identification which makes less follow up times may cause a patient to die.

Figure 6.3.3 clearly explain this situation indicating that there is a high probability to die when the follow-up time is less and probability to die reduces when the follow up time increases.

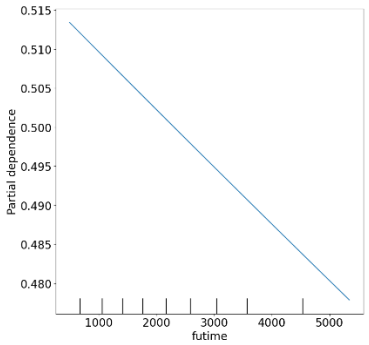


Figure 6.3.3.1: PD Plot of Follow-up Time

6.3.4 Average post HDL

The average post HDL means average HDL cholesterol level of the patient after the index date.

Medical researchers have identified that high cholesterol can also turn fatty liver disease (steatosis) into a more serious and sometimes fatal condition known as nonalcoholic steatohepatitis (NASH). Thus, the Cholesterol level plays a major role in controlling NAFLD. Basically, there are two types of cholesterol.

1. High Density lipoprotein (HDL) cholesterol
2. Low-density lipoprotein (LDL) cholesterol

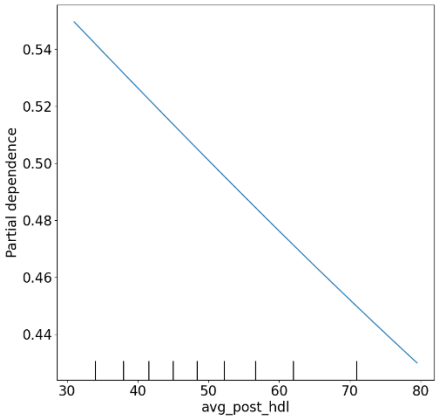


Figure 6.3.4.1: PD Plot of avg_post_HDL

Among these two HDL cholesterols, sometimes known as "good cholesterol," removes cholesterol from the blood and transports it to the liver. Previous studies have shown that higher HDL cholesterol is an independent protective factor for NAFLD, total cholesterol (TC) is associated with an increased risk of NAFLD.

The above fact is confirmed by Figure 6.3.4 that when the post HDL level increase, the probability of a patient to die decreases.

6.3.5 BMI

Obesity is defined as the condition of being overweight. The BMI of a person is typically used to assess such a situation. If the measurement is above the range for a particular person's height and weight, then that person is considered overweight.

The problem with being overweight is that your body gains fat, especially excess fat, which will be stored around your tummy area, which is a common symptom of NAFLD. Then these untreated conditions will lead to cirrhosis or liver cancer, which will cause death.

Hence, this could be a reasonable explanation for the partial dependency plot, whereas the higher the BMI, the greater the risk of death due to NAFLD that can be observed.

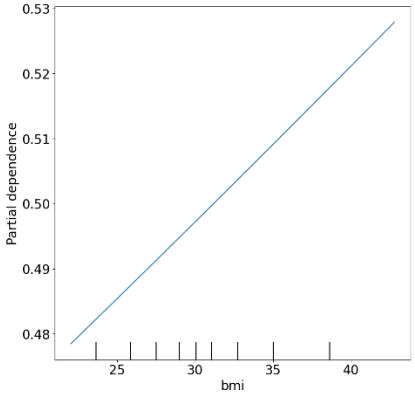


Figure 6.3.5.1: PD Plot of BMI

7. Issues encountered and proposed solutions

SOMTE was utilized to address the class imbalance problem. Next, for both no-resampling technique models and SMOTE-fitted models, low F1 scores were obtained for the overall model. Hence, even though class-wise accuracy was considered, to select the best model, further improvement is required. Even tuning the models did not improve them.

To address this issue, stacking and multi-layer perception (MLP) models were utilized based on suggestions from existing studies.

Stacking			MLP		
Accuracy	Recall		Accuracy	Recall	
	Class 0	Class 1		Class 0	Class 1
0.90	0.95	0.17	0.93	1.00	0.00

Table 7.1: Table of overall accuracy and class wise results of Stacking and MLP

According to Table 7.1, it was noted that they do not support overcoming this problem. However, ensemble techniques could be utilized to address this problem in future studies.

Appendix

```
#Grradient boosting
# Import models and utility functions
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_digits

# Setting SEED for reproducibility
SEED = 23
# Instantiate Gradient Boosting Regressor
gbc = GradientBoostingClassifier(n_estimators=300,
                                 learning_rate=0.05,
                                 random_state=100,
                                 max_features=5 )
# Fit to training set
gbc.fit(x_train, y_train)

# Predict on test set
pred_y = gbc.predict(x_test)

# accuracy
acc = accuracy_score(y_test, pred_y)
print("Gradient Boosting Classifier accuracy is :
{:.2f}".format(acc))
from sklearn.metrics import classification_report
f1 = f1_score(y_test, pred_y)
accuracy = accuracy_score(y_test, pred_y)

print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
print(classification_report(y_test, pred_y))
#hyperparameter tuning
from sklearn.model_selection import GridSearchCV
parameters ={{"learning_rate":[1, 0.5, 0.25, 0.1, 0.05, 0.01],
"max_depth":[1,2,3,4,5,6,7,8,9,10],
"min_samples_split":[0.1,0.2,0.3],
"min_samples_leaf":[0.1,0.2,0.3,0.4],
"max_features":[1,2,3,4,5,6,7]}}
grid_search= GridSearchCV(estimator=gbc,param_
grid=parameters,scoring='accuracy',cv=3,n_jobs=-
1,verbose=10)
grid_search = grid_search.fit(x_train, y_train)

accuracy = grid_search.best_score_ accuracy
Grid_search.best_params_
gbc_tuned =
GradientBoostingClassifier(n_estimators=300,learning_rate=
0.05,
random_state=100, max_features=6,
 min_samples_leaf=0.1,
 min_samples_split=0.1, max_depth=2)
gbc_tuned.fit(x_train, y_train)
pred_y = gbc_tuned.predict(x_test)
f1 = f1_score(y_test, pred_y)
accuracy = accuracy_score(y_test, pred_y)

print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
print(classification_report(y_test, pred_y))
#smote
#Performing SMOTE
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
x_train_resampled, y_train_resampled =
smote.fit_resample(x_train, y_train)
# Create and train a Random Forest classifier on the
resampled data
gb_classifier_resampled =
GradientBoostingClassifier(n_estimators=300,
                                 learning_rate=0.05,
                                 random_state=100,
                                 max_features=5 )


gb_classifier_resampled.fit(x_train_resampled,
y_train_resampled)
# Make predictions on the original test data
y_pred = gb_classifier_resampled.predict(x_test)

# Calculate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
classification_rep = classification_report(y_test, y_pred)
print('Classification Report:\n', classification_rep)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
roc_auc
gb_classifier_resampled.fit(x_train_resampled, y_train_resampled)
# Make predictions on the original test data
y_pred = gb_classifier_resampled.predict(x_test)

# Calculate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
classification_rep = classification_report(y_test, y_pred)
print('Classification Report:\n', classification_rep)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# Get feature importances
feature_importances =
gb_classifier_resampled.feature_importances_
# Match feature importances to feature names (assuming you have
a feature list)
feature_names = x_train.columns  # Replace with your feature
names
importance_dict = dict(zip(feature_names, feature_importances))

# Sort features by importance
sorted_importance = sorted(importance_dict.items(), key=lambda x:
x[1], reverse=True)

# Print feature importance
for feature, importance in sorted_importance:
    print(f'{feature}: {importance:.4f}')

#VIP Plot
#VIP
import matplotlib.pyplot as plt
feature_importance =
gb_classifier_resampled.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12, 6))
plt.barh(pos, feature_importance[sorted_idx],
align='center',color=['#003F7D','#FD7702','#FF5003'])
plt.yticks(pos,
np.array(x_test.columns.values.tolist())[sorted_idx])
plt.xlabel('Feature Importance')
plt.title('Variable Importance Plot')
plt.show()
features = [0]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator
(gb_classifier_resampled, x_train_resampled, features,ax=ax)
plt.savefig('0.png',dpi=300)

features = [1]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('1.png',dpi=300)
features = [2]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('2.png',dpi=300)
features = [3]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('3.png',dpi=300)
features = [4]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('4.png',dpi=300)
features = [5]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('5.png',dpi=300)
features = [6]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('6.png',dpi=300)
```

```
features = [7]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('7.png',dpi=300)
features = [8]
fig, ax = plt.subplots(figsize=(10,10))
plt.rc('font', size=20)
one=PartialDependenceDisplay.from_estimator(gb_classifier_resa
mpled, x_train_resampled, features,ax=ax)
plt.savefig('8.png',dpi=300)
#random forest
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100,
max_depth=10, random_state=42)
rf=rf_classifier.fit(x_train, y_train)
y_pred = rf_classifier.predict(x_test)
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
cm = confusion_matrix(y_test, y_pred)
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
f1 = f1_score(y_test, y_pred)
print(f'F1 Score: {f1:.2f}')
# Generate a classification report
classification_rep = classification_report(y_test, y_pred)
print('Classification Report:\n', classification_rep)
rf_confusion = confusion_matrix(y_test,y_pred)
rf_class_wise_accuracy =
rf_confusion.diagonal()/rf_confusion.sum(axis=1)
for class_idx, acc in enumerate(rf_class_wise_accuracy):
    print(f"Class {class_idx}: Accuracy = {acc * 100:.2f}%")
#Hyper parameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    # Add more hyperparameters to tune as needed
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(x_train, y_train)
best_rf = grid_search.best_estimator_
y_pred_tuned_rf = best_rf.predict(x_test)

#Performing SMOTE
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
x_train_resampled, y_train_resampled =
smote.fit_resample(x_train, y_train)
rf_classifier_resampled =
RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier_resampled.fit(x_train_resampled, y_train_resampled)
#Hyper parameter tuning_SMOTE model
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    # Add more hyperparameters to tune as needed
}
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier,
param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(x_train_resampled, y_train_resampled)
best_rf_smote_tune = grid_search.best_estimator_
y_pred_tuned_rf_smote_tune = best_rf_smote_tune.predict(x_test)

#Cluster Analysis
numerical_columns = ['age', 'weight_kg', 'height_cm', 'bmi', 'futime',
       'avg_pre_hdl', 'avg_post_hdl', 'avg_total_hdl', 'avg_pre_chol',
       'avg_post_chol', 'avg_total_chol']
categorical_columns = ['gender_female', 'gender_male']
x_copy = x_train
scaler = StandardScaler()
x_copy[numerical_columns] =
scaler.fit_transform(x_copy[numerical_columns])
X = x_copy.values
min_clusters = 2
max_clusters = 8
silhouette_scores = []
#perform K-prototype
for n_clusters in range(min_clusters, max_clusters + 1):
    kproto = KPrototypes(n_clusters=n_clusters, verbose=2)
#max_iter=100)
    clusters = kproto.fit_predict(X,
categorical=list(range(len(categorical_columns))))
# Calculate silhouette score for this clustering
    silhouette_avg = silhouette_score(X, clusters)
    silhouette_scores.append(silhouette_avg)
```

```
    print(f"Number of Clusters: {n_clusters}, Silhouette Score:
{silhouette_avg}")
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 5))
plt.plot(range(min_clusters, max_clusters + 1),
silhouette_scores, marker='o', linestyle='-', color='b')
plt.title('Silhouette Score vs. Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.grid()
plt.show()
#svm
kernel = "poly"
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel=kernel) # Linear Kernel

#Train the model using the training sets
clf.fit(x_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(x_test)
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

#Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
#F1 Score

from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
# Generate a classification report
classification_rep = classification_report(y_test, y_pred)
print('Classification Report:\n', classification_rep)
svm_classifier_tuned =
svm.SVC(kernel="rbf",C=10,gamma=0.0001)
svm_classifier_tuned.fit(x_train, y_train)
# Make predictions on the original test data
y_pred = svm_classifier_tuned.predict(x_test)

# Calculate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
#Performing SMOTE
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
x_train_resampled, y_train_resampled =
smote.fit_resample(x_train, y_train)
# Create and train a Random Forest
classifier on the resampled data
svm_classifier_resampled = svm.SVC(kernel=kernel)
svm_classifier_resampled.fit(x_train_resampled,
y_train_resampled)
# Make predictions on the original test data
y_pred = svm_classifier_resampled.predict(x_test)
```

```
# Calculate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
classification_rep = classification_report(y_test, y_pred)
print('Classification Report:\n', classification_rep)
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf','linear']}
grid_search =
GridSearchCV(svm_classifier_resampled,param_grid,cv=4,n_jobs=-
1,verbose=10)
grid_search.fit(x_train_resampled, y_train_resampled)
svm_classifier_resampled_tuned =
svm.SVC(kernel="rbf",C=10,gamma=0.001)
svm_classifier_resampled_tuned.fit(x_train_resampled,
y_train_resampled)
# Make predictions on the original test data
y_pred = svm_classifier_resampled_tuned.predict(x_test)

# Calculate accuracy and F1 score
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'F1 Score: {f1:.2f}')

from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, f1_score
trainx = train.drop(columns=['status','Unnamed: 0','weight_kg','height_cm'])
trainy = pd.DataFrame(train['status'])

testx = test.drop(columns=['status','Unnamed: 0','weight_kg','height_cm'])
testy = pd.DataFrame(test['status'])
#smote
oversample = SMOTE(random_state=0)
trainx , trainy = oversample.fit_resample(trainx , trainy)
# Display the class distribution after SMOTE
print(trainy.value_counts())

#logistic regression
logi_model = LogisticRegression()
# Fit the model to the training data
logi_model.fit(trainx , trainy)
# Make predictions on the test data
logi_pred = logi_model.predict(testx)
# Calculate accuracy
logi_accuracy = accuracy_score(testy, logi_pred)
print(f"Accuracy: {logi_accuracy * 100:.2f}%")
#confusion matrix
print("\nConfusion Matrix of logistic regression::")
print(confusion_matrix(testy, logi_pred))
logi_confusion = confusion_matrix(testy, logi_pred)
logi_class_wise_accuracy = logi_confusion.diagonal() /
logi_confusion.sum(axis=1)
for class_idx, acc in enumerate(logi_class_wise_accuracy):
    print(f"Class {class_idx}: Accuracy = {acc * 100:.2f}%")
f1_logi = f1_score(testy, logi_pred)
f1_logi
# Classification report
print("Classification Report:")
print(classification_report(testy, logi_pred))
```

```
 #logistic ridge
lridge_model = LogisticRegression(penalty='l2', C=1.0,
solver='liblinear')
# Fit the model to the training data
lridge_model.fit(trainx, trainy)
# Make predictions on the test data
lridge_pred = lridge_model.predict(testx)
# Calculate and display accuracy
lridge_accuracy = accuracy_score(testy, lridge_pred)
print(f"Accuracy: {lridge_accuracy * 100:.2f}%")

#naive bayes
NB_model = GaussianNB()
# Create a Multinomial Naive Bayes Classifier (for
discrete data)
# model = MultinomialNB()
# Fit the model to the training data
NB_model.fit(trainx, trainy)
# Make predictions on the test data
NB__pred = NB_model.predict(testx)
# Calculate and display accuracy
NB_accuracy = accuracy_score(testy, NB__pred)
print(f"Accuracy: {NB_accuracy * 100:.2f}%")

#XG Boosting
xgb_model = XGBClassifier()
# Train the classifier on the training data
xgb_model.fit(trainx, trainy)
xgb_pred = xgb_model.predict(testx)
# Evaluate the model's accuracy
xgb_accuracy = accuracy_score(testy,xgb_pred )
print(f"Accuracy: {xgb_accuracy * 100:.2f}%")

#hyperparameter tunning
xgb_classifier = XGBClassifier()
# Define a parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 500, 1000],
    'max_depth': [5,7,3],
    'learning_rate': [0.1, 0.01, 0.001],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}
# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=xgb_classifier,
param_grid=param_grid, scoring='accuracy', cv=5)
# Fit the grid search to the data
  grid_search.fit(trainx, trainy)
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
# Get the best model
best_model = grid_search.best_estimator_
# Make predictions on the test data using the best
model
y_pred = best_model.predict(testx)
# Evaluate the best model's accuracy
# Calculate and display accuracy
y_accuracy = accuracy_score(testy, y_pred)
print(f"Accuracy: {y_accuracy * 100:.2f}%")
# Classification report
print("Classification Report:")
print(classification_report(testy, y_pred))
y_confusion = confusion_matrix(testy, y_pred)
y_class_wise_accuracy = y_confusion.diagonal() /
y_confusion.sum(axis=1)
for class_idx, acc in
enumerate(y_class_wise_accuracy):
    print(f"Class {class_idx}: Accuracy = {acc *
100:.2f}%")
f1_y = f1_score(testy, y_pred)
f1_y
```