PROJECT-1 REPORT

Predicting Mortality caused by Heart Failure

Submitted towards the partial fulfillment of the criteria for award of Post Graduate Program In Analytics And Artificial Intelligence by Imarticus

Submitted By:

SANDEEP REDDY S

Course and Batch: PGA -JAN-2021

ABSTRACT

Cardiovascular disease (CVD's) are the No.1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts 31% of all deaths worldwide. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyper-lipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Acknowledgements

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.
Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guided us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank, all the faculties, as this project utilized knowledge gained from every course that formed the PGAA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: Aug 15, 2021 S SANDEEP REDDY
Place: Chennai

Certificate of Completion

I hereby certify that the project titled **"Heart Failure detection with Machine learning model project"** was undertaken and completed under my supervision by S SANDEEP REDDY from the batch of PGAA-1 (Jan 2021)

Date: Aug 15, 2021
Place: Chennai

Table of Contents

CHAPTER 1 : INTRODUCTION

1.1 Title & Objective of the study

"Heart failure prediction data project", is the project we are working upon fall under the health care domain. The Excel file contain complete medical records of patients having heart failure released by Ahmad and colleagues in July 2017. The primary purpose of working on this project is to predict the probability of heart failure, whether the patient is in high risk of getting affected by cardiovascular disease or not using the past data. That means, given set of new predictor variables, we need to predict the target variable as 1 ->high risk of cardiovascular disease , 0-> not in risk.

1.2 Need of the study

In this project, the main purpose is to predict whether a patient is at risk of cardiovascular disease or not, so that, early detection and management will be helpful for giving a required and better treatment. Machine learning applied to medical records, in particular, can be an effective tool both to predict the survival of each patient having heart failure symptoms, and to detect the most important clinical features (or risk factors) that may lead to heart failure.

1.3 Business or Enterprise under study

BMC Medical informatics and decision making is under study. Data of medical records of patients having heart failure issued by BMC in 2020. The data contains the indicator of heart failure , ejection fraction, serum creatinine and etc.

1.4 Business Model of Enterprise

Selecting the relevant variables from the dataset and arranging their values in order of importance to create a models to predict the probability of heart failure of an individual in the future by performing different types of algorithms on the data.

1.5 Data Sources

In Kaggle by BMC Medical informatics and decision making - Data contains the information about the patient medical reports. The dataset contains the information like age, sex, anaemia, ejection fraction, etc.

Dataset Description:
Contains 299 rows and 13 columns

The response variable is 'DEATH_EVENT' with '0' for Not in risk of CVD and '1' in risk of CVD.

1.7 Tools & Techniques

Tools: Spyder

Techniques: Logistic regression , Decision tree , Random forest classifier, Gradient boosting classifier

CHAPTER 2: DATA PREPARATION AND UNDERSTANDING

One of the first steps we engaged in was to outline the sequence of steps that we will be following for our project. Each of these steps are elaborated below
After importing the required libraries, a sequence of steps were followed to perform data pre-processing.

2.1 Phase I – Data Extraction and Cleaning:

- **Primary Analysis**

  In this primary analysis we get know about the each column's. it provide a separate Notebook file with the following information.

  i. missing values – Missing values in a particular column
  ii. dtype - Datatype of the column
  iii. unique values - no. of unique values in a column

code:

```
df_h = df.head()
  with open(dtype_file, "w") as f:
    for c in df_h:
      line1 = df_h[c]
      line2 = df[c].nunique()
      line3 = df[c].isnull().sum()
      f.write(str(line1) + "\n" + "Unique: " + str(line2) +
          ", missing: " + str(line3)
      + "\n\n" + "-----------------"+"\n")
  if classification == 0:
    plt.boxplot(df[y_name]); plt.show()
```

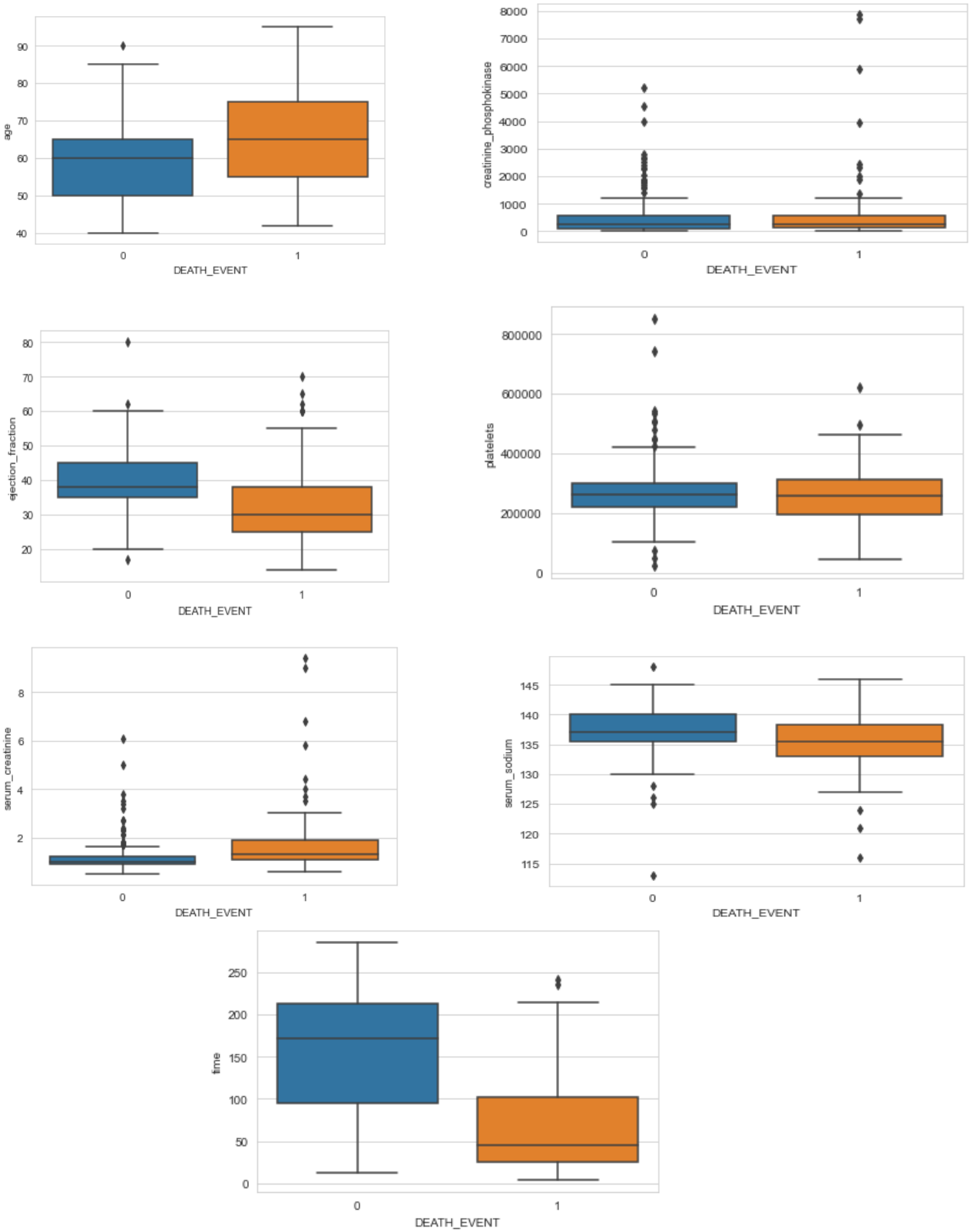
example:

```
0   75.0
1   55.0
2   65.0
3   50.0
4   65.0
Name: age, dtype: float64
Unique: 47, missing: 0
```

- There are no missing values in the data set ,so no need to perform missing values treatment.
- And all my features are numerical, so no need for label encode or one hot encoding.
- There are 6 binary columns(0,1) and 7 numerical columns.

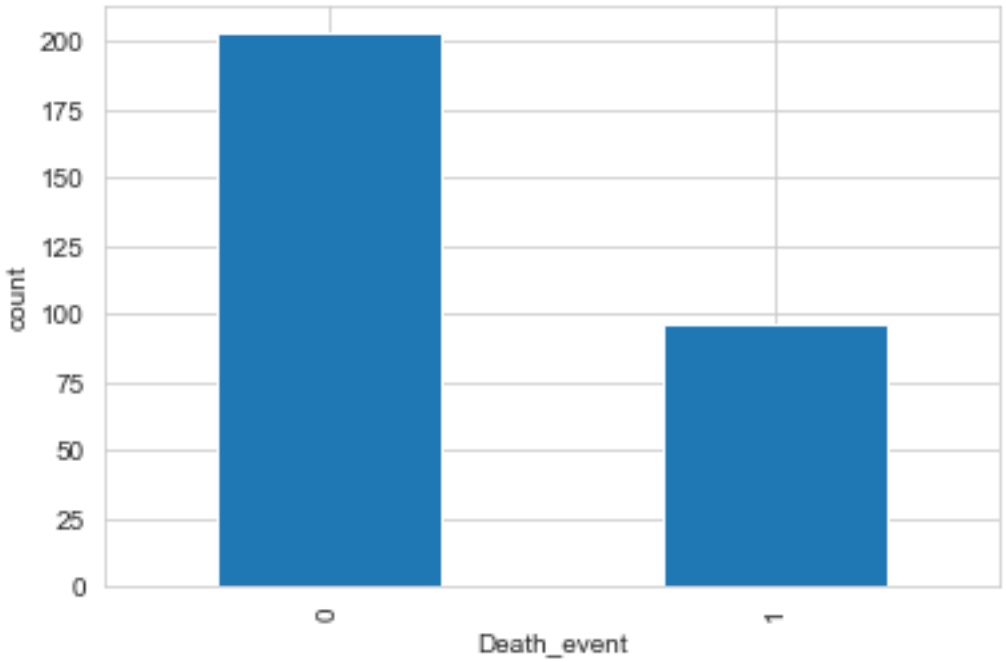
2.2 Phase II– Exploratory Data Analysis:

For visual analysis , we use boxplot from seaborn library, Here each continuous column in the data set is plotted against the Y (“DEATH_EVENT”)



- My plot reveals that most columns behaves almost same and they have many outliers .

- Age vs DEATH_EVENT plot shows that, above age 65 there is more chance of getting risk of heart failure.
- And in time vs DEATH_EVENT shows that , there is high chance of surviving heart failure if a patient lasts >100 follow-ups , upto hundred follow-ups there is a chance of getting heart failure.
- Also there is high skew and low skew in other columns.



- **This bar plot shows that, how many samples available in each class.**
- **203 samples for '0' and 96 samples for '1',so there is almost 35% sample difference between 0 and 1.**

Since my features are numerical, planned to build a model without any preprocessing steps. Based on the model performances , will do preprocessing steps one by one.

CHAPTER 3: FITTING MODELS TO DATA

3.1 Data Partition

The data is divided into, 70% of data falls in training data set and 30% of the data falls in testing data set.(This just for the base model building).

Train and Test data split:

x_train,x_test,y_train,y_test=ms.train_test_split(x,y,test_size=0.3, random_state=rstate)
model_fit = model.fit(x_train,y_train)
where,
'x' is independent feature.
'y' is dependent feature.
"test_size" is size of the testing data given.

3.2 Models Comparison:

Here we compare various model with fixed random state in a loop to get a best model.

Code:

```
models['LogR'] = LogisticRegression()
models['KNN'] = KNeighborsClassifier()
models['DTC'] = DecisionTreeClassifier()
models['NBC'] = GaussianNB()
models['ABC'] = AdaBoostClassifier()
models['SVC'] = SVC()
models['RFC'] = RandomForestClassifier()
models['GBC'] = GradientBoostingClassifier()
models['XGB'] = XGBClassifier()
models['MLP'] = MLPClassifier()
mmd = cl.compare_models(x,y,rstate)
for k, v in mmd.items():
print(k, ":", v)
```

output:

best_model : ('RFC', 0.8568)

models_data : {'LogR': {'mean': 0.752, 'std': 0.0512}, 'KNN': {'mean': 0.483, 'std': 0.094}, 'DTC': {'mean': 0.791, 'std': 0.0828}, 'NBC': {'mean': 0.7978, 'std': 0.0358}, 'ABC': {'mean': 0.8108, 'std': 0.0509}, 'SVC': {'mean': 0.5163, 'std': 0.0565}, 'RFC': {'mean': 0.8568, 'std': 0.0474}, 'GBC': {'mean': 0.8366, 'std': 0.0848}, 'XGB': {'mean': 0.8497, 'std': 0.0603}, 'MLP': {'mean': 0.4761, 'std': 0.0695}}

We got RandomForestClassifier as a best model.

3.3 Model Building:(without preprocessing**)**

Model 1

Here we use **RandomForestClassifier** because ,after comparing with other models ,RFC is the best performing model.

Code:

```
model1 = cl.RandomForestClassifier(random_state=169,)
print(model); print("")
x_train,x_test,y_train,y_test=ms.train_test_split(x,y,test_size=0.3, random_state=rstate)
model_fit = model.fit(x_train,y_train)
joblib.dump(model_fit, cpath+"/models"+str(model_id))
train_pred = model_fit.predict(x_train)
print("Train" , y_train.value_counts()/len(y_train))
print("Test" , y_test.value_counts()/len(y_test))
if pred_th != None:
    test_pred = threshold_prediction(model_fit, x_test, y, pred_th)
else:
    test_pred = model_fit.predict(x_test)

print("Training:")
print(classification_report(y_train, train_pred))
#   print("roc_auc:", roc_auc_score(y_train, train_pred))
print("")
print("% of Unknown classe @ threshold = "+str(pred_th), " is ",
round(len(test_pred[test_pred==-1])/len(test_pred),3))
print("Testing:")
print(classification_report(y_test, test_pred))
```

output:

Training:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	144
1	1.00	1.00	1.00	65
accuracy			1.00	209
macro avg	1.00	1.00	1.00	209
weighted avg	1.00	1.00	1.00	209

% of Unknown classe @ threshold = None is 0.0

Testing:

	precision	recall	f1-score	support
0	0.83	0.93	0.88	59
1	0.83	0.65	0.73	31
accuracy			0.83	90
macro avg	0.83	0.79	0.80	90
weighted avg	0.83	0.83	0.83	90

In the above report ,the training set got overfitted and my testing set was better.

Model 2:

Now we try adjusting parameter **max_depth=2** to overcome overfitting. This will help increasing the depth of the trees in the model.

Code:

```
model1 = cl.RandomForestClassifier(random_state=169,max_depth=2)
 print(model); print("")
x_train,x_test,y_train,y_test=ms.train_test_split(x,y,test_size=0.3, random_state=rstate)
model_fit = model.fit(x_train,y_train)
joblib.dump(model_fit, cpath+"/models"+str(model_id))
train_pred = model_fit.predict(x_train)
print("Train" , y_train.value_counts()/len(y_train))
print("Test" , y_test.value_counts()/len(y_test))
if pred_th != None:
   test_pred = threshold_prediction(model_fit, x_test, y, pred_th)
else:
   test_pred = model_fit.predict(x_test)

print("Training:")
print(classification_report(y_train, train_pred))
#   print("roc_auc:", roc_auc_score(y_train, train_pred))
print("")
print("% of Unknown classe @ threshold = "+str(pred_th), " is ",
round(len(test_pred[test_pred==-1])/len(test_pred),3))
print("Testing:")
        print(classification_report(y_test, test_pred))
```

output

Training:

	precision	recall	f1-score	support
0	0.82	0.98	0.90	144
1	0.92	0.54	0.68	65
accuracy			0.84	209
macro avg	0.87	0.76	0.79	209
weighted avg	0.85	0.84	0.83	209

% of Unknown classe @ threshold = None is 0.0

Testing:

	precision	recall	f1-score	support
0	0.76	0.98	0.86	59
1	0.93	0.42	0.58	31
accuracy			0.79	90
macro avg	0.85	0.70	0.72	90
weighted avg	0.82	0.79	0.76	90

In the above report , we just came out of overfitting , but my performance of the model gone down. So now try max_depth=4

Model 3

Code:

```
  model1 = cl.RandomForestClassifier(random_state=169,max_depth=4)
   print(model); print("")
   x_train,x_test,y_train,y_test=ms.train_test_split(x,y,test_size=0.3, random_state=rstate)
   model_fit = model.fit(x_train,y_train)
   joblib.dump(model_fit, cpath+"/models"+str(model_id))
   train_pred = model_fit.predict(x_train)
   print("Train" , y_train.value_counts()/len(y_train))
   print("Test" , y_test.value_counts()/len(y_test))
   if pred_th != None:
      test_pred = threshold_prediction(model_fit, x_test, y, pred_th)
   else:
      test_pred = model_fit.predict(x_test)

   print("Training:")
   print(classification_report(y_train, train_pred))
#   print("roc_auc:", roc_auc_score(y_train, train_pred))
   print("")
   print("% of Unknown classe @ threshold = "+str(pred_th), " is ",
round(len(test_pred[test_pred==-1])/len(test_pred),3))
   print("Testing:")
            print(classification_report(y_test, test_pred))
```

output

Training:

	precision	recall	f1-score	support
0	0.92	0.98	0.95	144
1	0.95	0.80	0.87	65
accuracy			0.92	209
macro avg	0.93	0.89	0.91	209
weighted avg	0.92	0.92	0.92	209

% of Unknown classe @ threshold = None is 0.0

Testing:

	precision	recall	f1-score	support
0	0.84	0.97	0.90	59
1	0.91	0.65	0.75	31
accuracy			0.86	90
macro avg	0.87	0.81	0.83	90
weighted avg	0.86	0.86	0.85	90

in the above model , the training set is good with 92% accuracy and precision is also good with 0.92 for 0 and 0.95 for 1, but recall for 1 affected due to class imbalance.

Model 4

Here ,now we try adjusting class_weight parameter for stabilizing the model

Code:

```
  model1=cl.RandomForestClassifier(random_state=169,max_depth=4,
                                   class_weight={0:0.75,1:1})
    print(model); print("")
   x_train,x_test,y_train,y_test=ms.train_test_split(x,y,test_size=0.3, random_state=rstate)
   model_fit = model.fit(x_train,y_train)
   joblib.dump(model_fit, cpath+"/models"+str(model_id))
   train_pred = model_fit.predict(x_train)
   print("Train" , y_train.value_counts()/len(y_train))
   print("Test" , y_test.value_counts()/len(y_test))
   if pred_th != None:
      test_pred = threshold_prediction(model_fit, x_test, y, pred_th)
   else:
      test_pred = model_fit.predict(x_test)

   print("Training:")
   print(classification_report(y_train, train_pred))
#   print("roc_auc:", roc_auc_score(y_train, train_pred))
   print("")
   print("% of Unknown classe @ threshold = "+str(pred_th), " is ",
round(len(test_pred[test_pred==-1])/len(test_pred),3))
   print("Testing:")
   print(classification_report(y_test, test_pred))
```

output:

```
Training:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       144
           1       0.95      0.80      0.87        65

    accuracy                           0.92       209
   macro avg       0.93      0.89      0.91       209
weighted avg       0.92      0.92      0.92       209


% of Unknown classe @ threshold = None  is  0.0
Testing:
              precision    recall  f1-score   support

           0       0.85      0.95      0.90        59
           1       0.88      0.68      0.76        31

    accuracy                           0.86        90
   macro avg       0.86      0.81      0.83        90
weighted avg       0.86      0.86      0.85        90
```

here my model is stable , but recall of class 1 is too poor. I can understand that my weak learner is class 1.

CHAPTER 4 Model Fitting(With pre-processing steps)

4.1 Scaling:

The continues features like **creatinine_phosphokinase, ejection_fraction, platelets, serum_creatinine, serum_sodium, time**, are scaled using minmax scalar

Code:

```
df_cat=x[['age','anaemia','sex','diabetes','high_blood_pressure','smoking']]
   df_num= x.drop(['age','anaemia','sex','diabetes','high_blood_pressure','smoking'],axis=1)
   df_num=pl.minmax_normalization(df_num)
   x = pl.join_num_cat(df_num,df_cat)
```

output

	0	1	2	3	4	5
0	0.0713	0.0909	0.2908	0.1573	0.4857	0.0
1	1.0	0.3636	0.2888	0.0676	0.6577	0.0071
2	0.0156	0.0909	0.1659	0.0892	0.4575	0.0106

After scaling , my number will be scaled between 0-1

4.2 Feature Transformation:

In scaling 'age' column is separated as categorical , because here, the age column will be transformed to age groups as, 1,2,3,4,5.

Code:

```
list_=df['age']
 age_grp = {1:range(1,21),2:range(21,41),3:range(41,60),4:range(60,101)}
 x['age']= [key for v in list_
       for key,val in age_grp.items() if v in val]
```

output:

before:		after:
	age	age
0	75	4
1	55	3
2	65	4
3	50	3

4.3 Stratified Sampling:

Since class 1 is a weak learner , stratified sampling will give a equal no.of samples for both classes 0 and 1.

Function:

```
def stratified_sample(df, col, n_samples):
    n = min(n_samples, df[col].value_counts().min())
    df_ = df.groupby(col).apply(lambda x: x.sample(n))
    df_.index = df_.index.droplevel(0)
    return df_
```

Code:

```
n_samples = 5000
 df_temp = pd.concat((x,y),axis=1)
 df_sample = pl.stratified_sample(df_temp, y_name, n_samples)
 print("stratified sample:"); print(df_sample[y_name].value_counts())

 y_new=df_sample[y_name]
 x_new=df_sample.drop([y_name],axis=1)
```

output:

```
stratified sample:
0   96
1   96
```

After stratified sampling , we got 96 samples for each class 0 and 1 .

4.4 K-Fold-cross validation Analysis:

Here we do k-fold for stratified sample with GradientBoostingClassifier. This gives how the each fold of a data performs.. I will fold the data set horizontally into 5 folds, accuracy is calculated for each fold and finally mean performance is calculated.

Function:

```
def kfold_cross_validate(model,X,Y,rstate):
    feature_folds = ms.KFold(n_splits=5, shuffle = True, random_state=rstate)
    cv_estimate = ms.cross_val_score(model, X, Y, cv = feature_folds)
    print('Mean performance metric = ', np.mean(cv_estimate))
    print('SDT of the metric       = ', np.std(cv_estimate))
    print('Outcomes by cv fold')
    for i, x in enumerate(cv_estimate):
        print('Fold ', (i+1, x))
    print("")
```

code:

```
best_model = cl.GradientBoostingClassifier()
 pl.kfold_cross_validate(best_model, x, y,171)
```

output:

```
Mean performance metric =  0.8528248587570623
SDT of the metric       =  0.019549981940275758
Outcomes by cv fold
Fold  (1, 0.8333333333333334)
Fold  (2, 0.8833333333333333)
Fold  (3, 0.8666666666666667)
Fold  (4, 0.8333333333333334)
Fold  (5, 0.847457627118644)
```

We can observe that there is not much difference between folds. Only 0.03-0.05(3%-5%) is the difference. And mean accuracy is 0.85 or 85%.

4.5 Model fitting:

Here we going to use four different models with above steps including.

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- Gradient Boosting Classifier

Logistic Regression(with adjusted parameters**):**

```
LogisticRegression(class_weight={0: 0.7, 1: 1}, random_state=169)
Training:
              precision    recall  f1-score   support

           0       0.82      0.66      0.73        71
           1       0.69      0.84      0.76        63

    accuracy                           0.75       134
   macro avg       0.76      0.75      0.75       134
weighted avg       0.76      0.75      0.75       134


% of Unknown classe @ threshold = None  is  0.0
Testing:
              precision    recall  f1-score   support

           0       0.90      0.72      0.80        25
           1       0.82      0.94      0.87        33

    accuracy                           0.84       58
   macro avg       0.86      0.83      0.84       58
weighted avg       0.85      0.84      0.84       58

roc_auc: 0.8296969696969696
```

In the reports , we can see that LOGR has a poor training and testing performances than the previous model without any preprocessing.

Decision Tree Classifier:
DecisionTreeClassifier(random_state=169,max_depth=2,max_leaf_nodes=3)

Training:

	precision	recall	f1-score	support
0	0.87	0.85	0.86	71
1	0.83	0.86	0.84	63
accuracy			0.85	134
macro avg	0.85	0.85	0.85	134
weighted avg	0.85	0.85	0.85	134

% of Unknown classe @ threshold = None is 0.0
Testing:

	precision	recall	f1-score	support
0	0.84	0.84	0.84	25
1	0.88	0.88	0.88	33
accuracy			0.86	58
macro avg	0.86	0.86	0.86	58
weighted avg	0.86	0.86	0.86	58

roc_auc: 0.8593939393939394

In decision tree , the model is well stable with good precision and recall values, and accuracy also improved than the previous one.
ROC_AUC score also printed to evaluate the model.
Code:

```
from sklearn.metrics import classification_report, roc_auc_score
print("roc_auc:", roc_auc_score(y_test, test_pred))
```

This actually evaluates the actual y_test value with predicted value.

Random Forest Classifier :

RandomForestClassifier(random_state=169,max_depth=3,class_weight={0:0.7,1:1},)

Training:

	precision	recall	f1-score	support
0	0.90	0.93	0.92	71
1	0.92	0.89	0.90	63
accuracy			0.91	134
macro avg	0.91	0.91	0.91	134
weighted avg	0.91	0.91	0.91	134

% of Unknown classe @ threshold = None is 0.0

Testing:

	precision	recall	f1-score	support
0	0.96	0.88	0.92	25
1	0.91	0.97	0.94	33
accuracy			0.93	58
macro avg	0.94	0.92	0.93	58
weighted avg	0.93	0.93	0.93	58

roc_auc: 0.9248484848484849

The Random forest classifier performance Is better that the previous models ,because of multiple probabilities of decision trees were built, but one thing is not improved , which is recall of class 0 is gone down to 0.88. other-than roc auc score also looks good with 0.92

Gradient Boosting Classifier :

GB builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n_classes_ regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

```
model1 = cl.GradientBoostingClassifier(random_state=169,max_depth=1,learning_rate=0.2,)
Training:
              precision    recall  f1-score   support

           0       0.93      0.94      0.94        71
           1       0.94      0.92      0.93        63

    accuracy                           0.93       134
   macro avg       0.93      0.93      0.93       134
weighted avg       0.93      0.93      0.93       134


% of Unknown classe @ threshold = None  is  0.0
Testing:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        25
           1       0.94      0.94      0.94        33

    accuracy                           0.93        58
   macro avg       0.93      0.93      0.93        58
weighted avg       0.93      0.93      0.93        58

roc_auc: 0.9296969696969698
```

We can clearly observe that the model I stable and have performance is than the previous model where recall is bit low but here it well performed. Even Roc_auc score is increased by 0.5%

CHAPTER 5 : CONCLUSION

Out of all the algorithms used, Gradient boosting classifier gave us the least type I error and type II error with 93% accuracy. Gradient boosting with tuning was preferred, were evaluation technique roc_auc score is 0.9294 ,almost 93% .RFC also gave better results but class 1 got only 0.89 precision and recall. So the best suited model Gradient Boosting Classifier (GBC).

BEST MODEL CLASSIFICATION REPORT

model1 = cl.GradientBoostingClassifier(random_state=169,max_depth=1,learning_rate=0.2,)
Training:

	precision	recall	f1-score	support
0	0.93	0.94	0.94	71
1	0.94	0.92	0.93	63
accuracy			0.93	134
macro avg	0.93	0.93	0.93	134
weighted avg	0.93	0.93	0.93	134

% of Unknown classe @ threshold = None is 0.0
Testing:

	precision	recall	f1-score	support
0	0.92	0.92	0.92	25
1	0.94	0.94	0.94	33
accuracy			0.93	58
macro avg	0.93	0.93	0.93	58
weighted avg	0.93	0.93	0.93	58

roc_auc: 0.9296969696969698