Prediction of Heart Disease in a patient by using Machine Learning Algorithms.

By: Mahesh Ganta Pilli

UID: U01064382

Email: gantapilli.2@wright.edu

Abstract

The healthcare sector is constantly changing. Healthcare personnel may find it challenging to keep up with the continual development of new technologies and therapies. Machine learning technology is one of the biggest buzzwords in healthcare in recent years. But what is it, why is it crucial for patient data, and what are some advantages of machine learning in the medical field?

Machine learning is a specific type of artificial intelligence that allows systems to learn from data and detect patterns without much human intervention. Instead of being told what to do, computers using machine learning are shown patterns and data that then allow them to come to their own conclusions. Computers use machine learning systems to automatically scan emails, find spam, recognize things in images, and process big data.

The field of machine learning techniques is expanding and has a wide range of possible applications. Machine learning technology will be more and more crucial for healthcare practitioners and healthcare systems to interpret medical data as patient data becomes more readily available.

Introduction

The human way of life has now been harmed by chronic stress, lethargic work environments, and hectic routines in the fast-paced society. According to surveys, smoking, lifestyle modifications, and obesity are the main causes of cardiovascular diseases (CVDs) in both men and women. A significant healthcare challenge is the detection of heart disease in aged individuals as the morbidity rate associated with the condition has risen as a result of heart failure and related operations.

In improving patient medical outcomes, it is crucial to apply cutting-edge approaches and techniques in bio-medical data research to evaluate and diagnose heart disease at an early stage.

Problem Statement

So, we have decided to work with a Heart Disease detector using the heart disease dataset. Based on various components such as:
'Age', 'Gender', 'Height', 'Weight', 'Systolic_BP', 'Diastolic_BP', 'Cholesterol', 'Glucose', 'Smoking', 'Alco Intake', 'Physical Exer'.

Our intention is to create a supervised machine learning model that, by combining different elements, can aid in predicting a person's likelihood of contracting heart disease. This project helps us in finding out which model will predict more accurately for these types of features.

Background

There have been several publications and prototypes pertinent to this work in the past. These models are created using a variety of methods, including deep learning models, probabilistic models created using fundamental machine learning, etc.

I looked at a variety of articles and earlier studies to comprehend the fundamental functionality. We came to the conclusion that these procedures don't have obvious pre-processing and tuning mechanism which outcomes best parameters of the model.

Dataset

The dataset can be found https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset.
This dataset contains a few attributes such as

'Age', 'Gender', 'Height', 'Weight', 'Systolic_BP', 'Diastolic_BP', 'Cholesterol', 'Glucose', 'Smoking', 'Alco Intake', 'Physical Exer'

And the Label(Target): **“Heart Disease”**

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10003 entries, 0 to 10002
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             10003 non-null  int64  
 1   Gender          10003 non-null  object 
 2   Height          10003 non-null  int64  
 3   Weight          9973 non-null   float64
 4   Systolic_BP     9838 non-null   float64
 5   Diastolic_BP    9897 non-null   float64
 6   Cholesterol     10003 non-null  int64  
 7   Glucose         10003 non-null  int64  
 8   Smoking         10003 non-null  object 
 9   Alco Intake     10003 non-null  int64  
 10  Physical Exer   10003 non-null  object 
 11  HeartDisease    10003 non-null  int64  
dtypes: float64(3), int64(6), object(3)
memory usage: 937.9+ KB
```

The shape of the dataset is **10003 rows × 12** columns.

A glance at the dataset:

```
data
```

	Age	Gender	Height	Weight	Systolic_BP	Diastolic_BP	Cholesterol	Glucose	Smoking	Alco Intake	Physical Exer	HeartDisease
0	18393	F	168	62.0	110.0	80.0	1	1	N	0	Y	0
1	20228	M	156	85.0	140.0	90.0	3	1	N	0	Y	1
2	18857	M	165	64.0	NaN	70.0	3	1	N	0	N	1
3	17623	F	169	82.0	150.0	100.0	1	1	N	0	Y	1
4	17474	M	156	56.0	100.0	60.0	1	1	N	0	N	0
...	...	...	...	...	...	...	...	...	...	...	...	...
9998	18328	M	168	70.0	160.0	80.0	3	1	N	0	Y	1
9999	19702	F	163	68.0	140.0	90.0	1	1	N	0	Y	1
10000	23421	F	170	74.0	120.0	80.0	1	1	N	0	Y	0
10001	15527	F	158	61.0	120.0	80.0	1	1	N	0	Y	0
10002	21279	M	160	83.0	140.0	90.0	1	1	N	0	Y	1

10003 rows × 12 columns

Changes done to the original dataset:

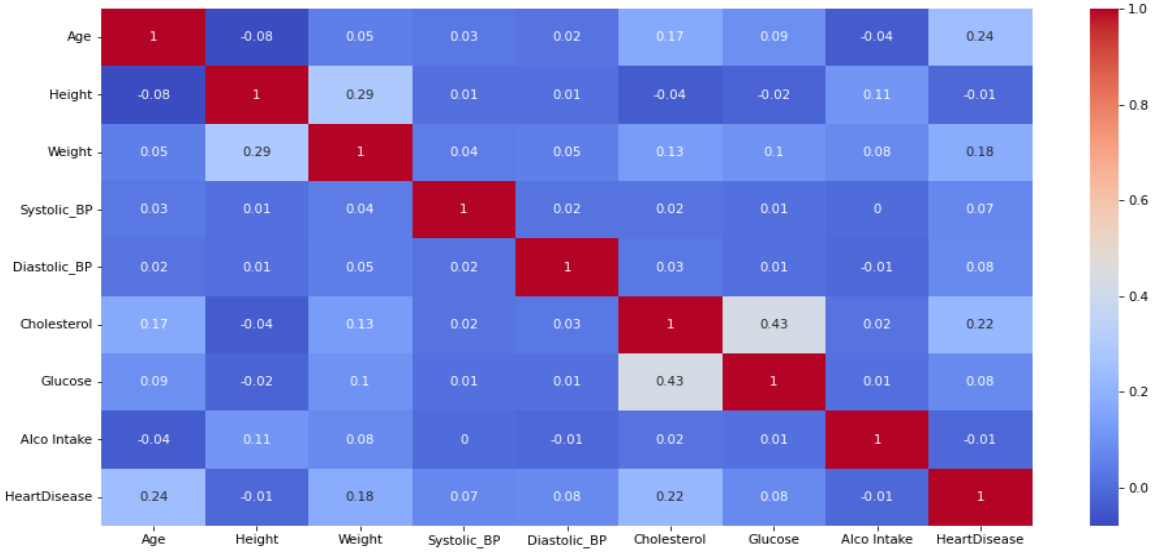
Before the dataset was completely numerical so I had converted columns named Smoking, Physical Exer to categorical 0 -> N, 1-> Y. And deleted some records of individual columns like Weight', 'Systolic_BP', 'Diastolic_BP' for data processing steps.

One Drive Link: https://raidermailwright-my.sharepoint.com/:x:/g/personal/gantapilli_2_wright_edu/EUiIqwM4xjFEs8ga6uswCHIBo6Uugk7PLzChIQO7FJqIig?e=S46PRh

Detailed design of Features

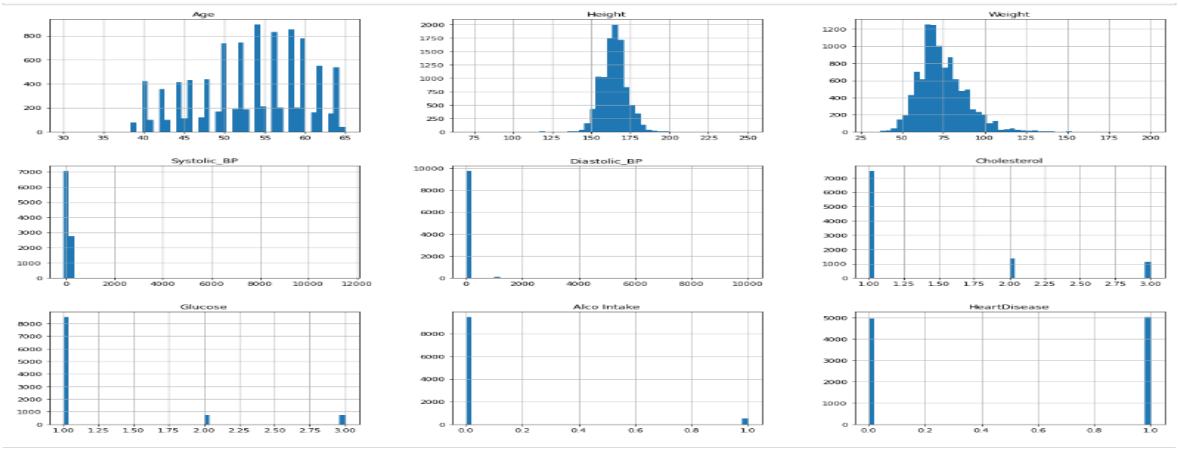
To Find the correlation between various components of the dataset. We had used below code snippet and plotted corelation matrix.

```
def func1():
    mplt.figure(figsize=(16, 8))
    corela_matrix = data.corr().round(2)
    sns.heatmap(data=corela_matrix,cmap='coolwarm',annot=True)

func1()
```



From the above figure we can find out which feature has strongest co-relation with Target (HeartDisease). We can define Age and Cholesterol has strong positive corelation and Alco_Intake has negative corelation with Target feature.

The frequency plot describes the frequency of various data points in the dataset.



Data Pre-Processing and Exploratory Data Analysis (EDA):

Pre- Processing the dataset. So that we are able to feed the dataset to the ML model.

These consist of:

1. Converting age columns from number of days to age in years.

```
data['Age'] = nump.round(data['Age'] / 365.25)
data.head()
```

	Age	Gender	Height	Weight	Systolic_BP	Diastolic_BP	Cholesterol	Glucose	Smoking	Alco Intake	Physical Exer	HeartDisease
0	50.0	F	168	62.0	110.0	80.0	1	1	N	0	Y	0
1	55.0	M	156	85.0	140.0	90.0	3	1	N	0	Y	1
2	52.0	M	165	64.0	NaN	70.0	3	1	N	0	N	1
3	48.0	F	169	82.0	150.0	100.0	1	1	N	0	Y	1
4	48.0	M	156	56.0	100.0	60.0	1	1	N	0	N	0

2. Using a **one-hot encoding meth**od to convert categorical data into numeric values.
   a. We are using one-hot encoding method to convert our categorical data in dataset to binary representation where this approach will create extra columns according to the unique categorical data in the specified feature.
   b. In our case we have 2 categories Y and N so it will create new colums for binary representation.

Before Encoding the data:

	Age	Gender	Height	Weight	Systolic_BP	Diastolic_BP	Cholesterol	Glucose	Smoking	Alco Intake	Physical Exer	HeartDisease
0	18393	F	168	62.0	110.0	80.0	1	1	N	0	Y	0
1	20228	M	156	85.0	140.0	90.0	3	1	N	0	Y	1
2	18857	M	165	64.0	NaN	70.0	3	1	N	0	N	1
3	17623	F	169	82.0	150.0	100.0	1	1	N	0	Y	1
4	17474	M	156	56.0	100.0	60.0	1	1	N	0	N	0

After Encoding the data:

```
data.head()
```

	Age	Height	Weight	Systolic_BP	Diastolic_BP	Cholesterol	Glucose	Alco Intake	HeartDisease	Gender_F	Gender_M	Smoking_N	Smoking_Y	Physical Exer_N	Physi Exer
0	50.0	168	62.0	110.0	80.0	1	1	0	0	1	0	1	0	0	
1	55.0	156	85.0	140.0	90.0	3	1	0	1	0	1	1	0	0	
2	52.0	165	64.0	120.0	70.0	3	1	0	1	0	1	1	0	1	
3	48.0	169	82.0	150.0	100.0	1	1	0	1	1	0	1	0	0	
4	48.0	156	56.0	100.0	60.0	1	1	0	0	0	1	1	0	1	

3. Using the **Standard scaler normalization** method to scale data.

   Normalization is a common technique for organizing data for machine learning. Changing the values of numerical columns in a data collection to a similar scale without distorting variations in value ranges is the aim of normalization. Not all machine learning datasets necessitate normalization. It is only essential when the functions have diverse scopes.

   It's interesting to note that the term "standardization" describes setting the mean to 0 and the standard deviation to 1.

```
In [13]: from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         col_scale = ['Age','Height', 'Weight','Systolic_BP','Diastolic_BP']
         data[col_scale] = sc.fit_transform(data[col_scale])
```

```
In [14]: X = data.drop(['HeartDisease'], axis=1)
         y = data['HeartDisease']
         data.head()
```

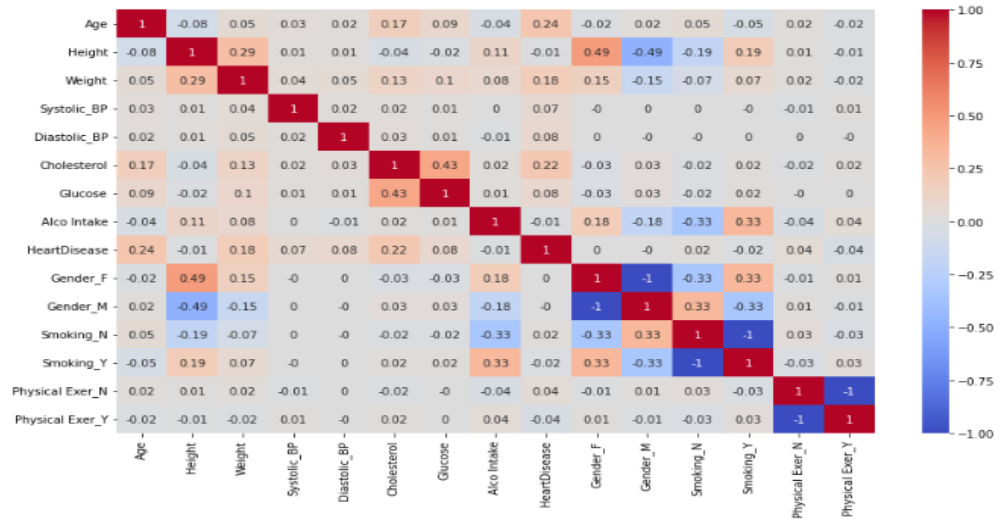
Out[14]:

	Age	Height	Weight	Systolic_BP	Diastolic_BP	Cholesterol	Glucose	Alco Intake	HeartDisease	Gender_F	Gender_M	Smoking_N	Smoking_Y	Physi Exer
0	-0.479034	0.451507	-0.844563	-0.154500	-0.086569	1	1	0	0	1	0	1	0	
1	0.256962	-1.015896	0.749050	0.102349	-0.038249	3	1	0	1	0	1	1	0	
2	-0.184635	0.084656	-0.705988	-0.068883	-0.134889	3	1	0	1	0	1	1	0	
3	-0.773432	0.573790	0.541187	0.187966	0.010071	1	1	0	1	1	0	1	0	
4	-0.773432	-1.015896	-1.260288	-0.240116	-0.183209	1	1	0	0	0	1	1	0	

   We can clearly observe that the values are in the range of -1 to 1.

4. Determining **correlations** between independent and dependent variables.

```
[15]: mplt.figure(figsize=(14, 8))
      corela_matrix = data.corr().round(2)
      sns.heatmap(data=corela_matrix,cmap='coolwarm',annot=True)
t[15]: <AxesSubplot:>
```



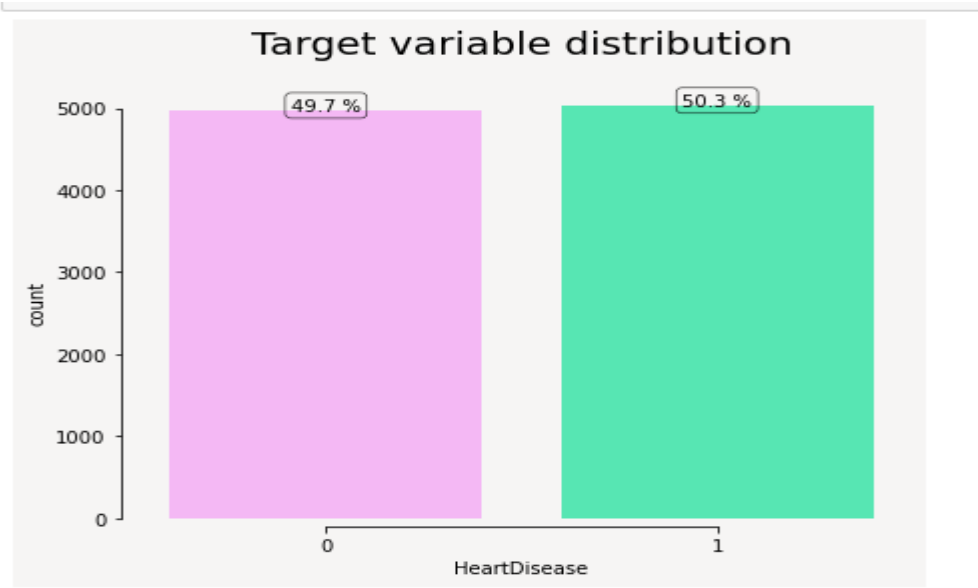
5. We had spitted our data into 70% training data and 30% testing data of shape 7002 is of train data and 3001 testing data with random_state = 42 which makes our model to split data similarly in every run.

```
In [18]: from sklearn.model_selection import train_test_split
         x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=42)

In [19]: x_train.shape
Out[19]: (7002, 14)

In [20]: x_test.shape
Out[20]: (3001, 14)

In [21]: y_train.shape
Out[21]: (7002,)
```

Conclusion:

As from this we can clearly conclude that only as few features from the above plot seems related to each other. So, taking all the features for training the model seems redundant.
Which may even cause loss of useful information while training.

Target Distribution:



Feature selection:

For selection of features, I had implemented RFE. It is necessary **for Recursive Feature Elimination (RFE)** to conserve a specific number of features, yet it is typically unclear in advance how many valid features there are. RFE efficiently rejects features until the required number of features are left during the search for a subset of the training dataset's features. We are using logistic regression as an algorithm used to choose features and specify the count as 10.

```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE

X = x_train
y = y_train
# Build a logreg and compute the feature importances
logi_model = LogisticRegression(random_state=42)
rfe = RFE(logi_model,n_features_to_select=10)
rfe = rfe.fit(X, y)
# Final selected attributes
print('Selected features: %s' % list(X.columns[rfe.support_]))
```

This helped us finding the best fit features for the model:

```
Selected features: ['Age', 'Weight', 'Systolic_BP', 'Diastolic_BP', 'Cholesterol', 'Glucose', 'Smoking_N', 'Smoking_Y', 'Physic
al Exer_N', 'Physical Exer_Y']
```

Models Implemented:

To solve this problem, we have used the following ML algorithms:

1. Logistic Regression
2. K Nearest Neighbours
3. Decision Tree Classifier
4. Random Forest Classifier
5. Extreme Gradient Boosting (XGB)
6. Stochastic gradient descent (SGD)

Logistic Regression:

Logistic regression is the popular ML model which predicts the outcome as either 0 or 1 based upon the features we provide as input. In our case we had implemented the basic logistic regression model.

Implementation Steps:

1. In this model implementation we are using basic logistic regression model with random state 42 which make our data split similarly in each run. We fit our model using x_tarin and y_train.

```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression

logis_r = LogisticRegression(random_state = 42)
logis_r.fit(x_train,y_train)
```

```
LogisticRegression(random_state=42)
```

2. We made our model to train on our training data and performing prediction on test data(x_test) after fitting our model.

```
y_pred_logi_r = logis_r.predict(x_test)
```

```
y_pred_logi_r.shape
```

```
(3001,)
```

3. At last, we are calculating accuracy of the model using accuracy_score function imported from sklearn.metrics and plotting classification report of it along with confusion matrix.

```
score_logr = round(accuracy_score(y_test,y_pred_logi_r)*100,2)
print(f'The Logistic Regression Accuracy Score is : {score_logr}%')
```

```
The Logistic Regression Accuracy Score is : 72.18%
```

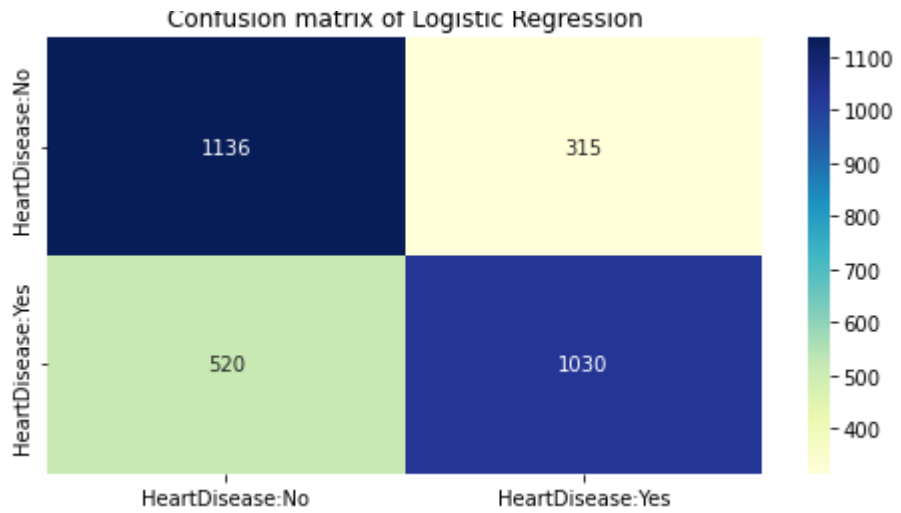
Classification report:

F1 score of logistic regression is 71.15%.

```
: print('Logistic Regression Classification Report: \n', classification_report(y_test,y_pred_logi_r),'\n')
  print('Logistic Regression Confusion Matrix: \n', confusion_matrix(y_test,y_pred_logi_r,))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.78      0.73      1451
           1       0.77      0.66      0.71      1550

    accuracy                           0.72      3001
   macro avg       0.73      0.72      0.72      3001
weighted avg       0.73      0.72      0.72      3001
```

Confusion matrix:



K Nearest Neighbours (KNN):

K Nearest Neighbour (KNN) is an easy-to-use supervised machine learning (ML) technique. It is a lazy learner ML algorithm.

In our case we had implemented KNN Classifier with hyper parameter tuning mechanism as GridSearchCV where we try out different algorithms and to get best unique **k-value** which is very important in KNN.

Implementation Steps:

1. In this model implementation I had use KNeighborsClassifier() to train the model.

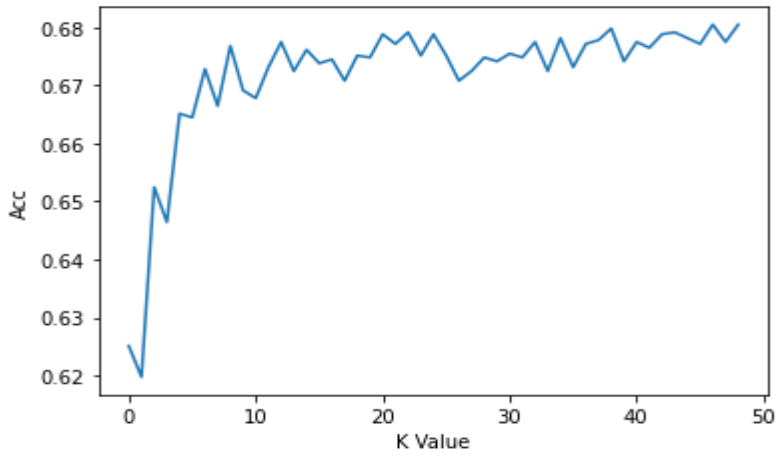
```
kNeigh_clas = KNeighborsClassifier()
kNeigh_clas.fit(x_train,y_train)
```

```
KNeighborsClassifier()
```

2. Main goal in KNN is to find the k-value, for which I had implemented 2 procedures:
    a. Firstly, I had used traditional for loop approach where we try out each value between the range of 1-50(I considered) and plotted the graph using matplotlib to find at which point of k the accuracy is more. But, by seeing this plot it is very difficult to find out which k-value as good accuracy.

```
score_auck= []
for k in range(1,50):
    kNeigh_clas = KNeighborsClassifier(n_neighbors=k)
    kNeigh_clas.fit(x_train,y_train)
    y_pred=kNeigh_clas.predict(x_test)
    score_auck.append(accuracy_score(y_test,y_pred))
```

```
mplt.plot(score_auck)
mplt.xlabel("K Value")
mplt.ylabel("Acc")
mplt.show()
```



b. So, to overcome above difficulty I had used GridSearchCV method which is one of the hyper-parameters tuning method where we going to try all possible sets of given param_gsvknn and will finalize the one which yields better results on our dataset using best parameters of the run.

```
param_gsvknn = [{
    'n_neighbors': n_neigh_list,
    'p': p_l,
    'weights': weights_list,
    'algorithm': algorithms_list,
}]

gri_se = GridSearchCV(kNeigh_clas, param_gsvknn,
                      scoring='accuracy',
                      refit=True,
                      cv=kf,
                      verbose=1,
                      n_jobs=4)
gri_se.fit(x_train, y_train)
print('Best Parameters: ', gri_se.best_params_)
```

Best Parameters:

```
: print(gri_se.best_estimator_)

  KNeighborsClassifier(algorithm='kd_tree', n_neighbors=41, p=1)
```

3. With the help of above outcome parameters, we can generate a model with high accuracy when compared with other parameters within.

```
kNeigh_clas = KNeighborsClassifier(n_neighbors=gri_se.best_estimator_.n_neighbors,algorithm=gri_se.best_estimator_.algorithm,
kNeigh_clas.fit(x_train,y_train)
y_pred_knc=kNeigh_clas.predict(x_test)
y_pred_knc.shape
```

4. At last, we are calculating accuracy of the model and plotting classification report of it along with confusion matrix.

```
knn_auc_score = round(accuracy_score(y_test,y_pred_knc)*100,2)
print(f'The KNN Classification Accuracy Score is : {knn_auc_score}%')
```

```
The KNN Classification Accuracy Score is : 68.58%
```

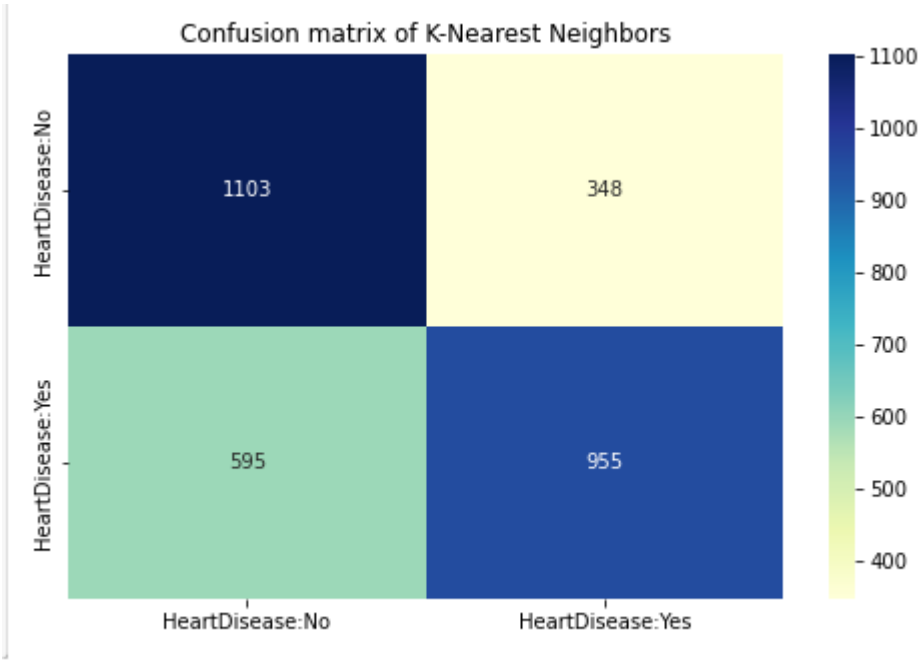
Classification report:

The F1 score of KNN Classification is 66.94%.

```
print('KNN Classification Classification Report: \n', classification_report(y_test,y_pred_knc),'\n')
print('KNN Classification Confusion Matrix: \n', confusion_matrix(y_test,y_pred_knc),'\n')
```

```
KNN Classification Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.76      0.70      1451
           1       0.73      0.62      0.67      1550

    accuracy                           0.69      3001
   macro avg       0.69      0.69      0.69      3001
weighted avg       0.69      0.69      0.68      3001
```

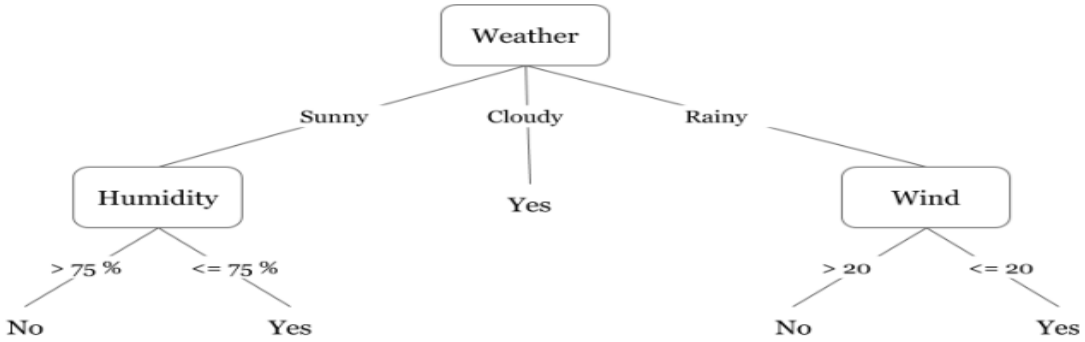
Confusion matrix:



Decision Tree:

A supervised machine learning approach called a decision tree can be used to solve classification and regression issues. In this instance, I used a decision tree classifier to assess the dataset.

Below is the intuition of Decision tree algorithm on the backend how it creates tree like structures when we train our model.

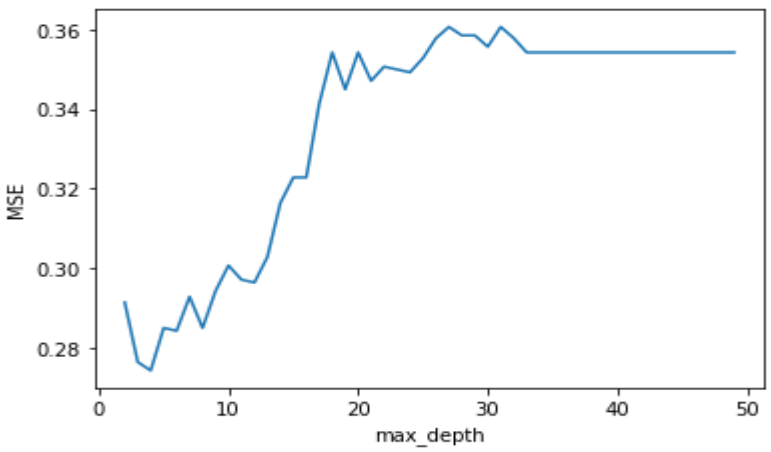


Implementation Steps:

1. In the Decision Forest implementation, I am converting our data into array like structures for easy and faster calculation.

```
: X = x_train.to_numpy()
  y = y_train.to_numpy()
```

2. I am finding Mean Squared Error (MSE) to calculate the depth at which the MSE value is close to zero which helps us in getting better accuracy than other depth values. In our case, I got the point at which MSE is close to zero is 4.



Max depth at which MSE error is close to zero is 4.

```
best_depth = max_depth_range[nump.array(max_depth).argmin()]
print(f'The better choice of max_depth: {best_depth}')
```

```
The better choice of max_depth: 4
```

3. Later I am utilizing that depth to calculate the accuracy and plot classification report of the model along with confusion matrix.

```
from sklearn.tree import export_graphviz
deci_tr_c = DecisionTreeClassifier(max_depth= best_depth, random_state=42)
deci_tr_c.fit(x_train, y_train)

Y_pred_deci_tr_c = deci_tr_c.predict(x_test)
Y_pred_deci_tr_c.shape

export_graphviz(deci_tr_c, out_file='tree.dot')
```

```
deci_tr_c_aucscore = round(accuracy_score(y_test,Y_pred_deci_tr_c)*100,2)
print(f'The Decision Tree Classifiaction Accuracy Score is : {str(deci_tr_c_aucscore)}%')
```

```
The Decision Tree Classifiaction Accuracy Score is : 72.57%
```

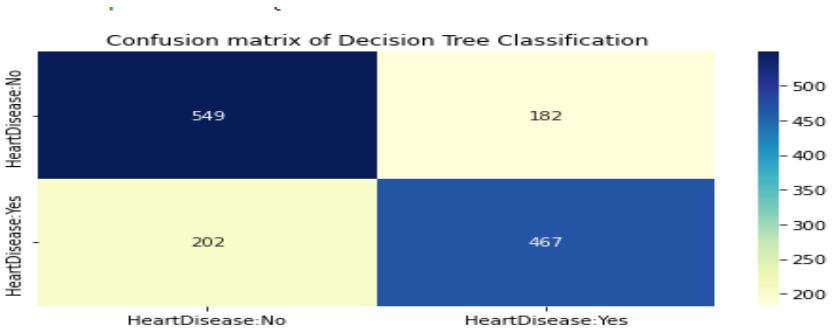
Classification report:

The F1 score of Decision tree classification is 70.86%.

```
print('Decision Tree Classifier Classification Report: \n', classification_report(y_test,Y_pred_deci_tr_c),'\n')
print('Decision Tree Classification Confusion Matrix: \n', confusion_matrix(y_test,Y_pred_deci_tr_c),'\n')
```

```
Decision Tree Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.75      0.74       731
           1       0.72      0.70      0.71       669

    accuracy                           0.73      1400
   macro avg       0.73      0.72      0.72      1400
weighted avg       0.73      0.73      0.73      1400
```

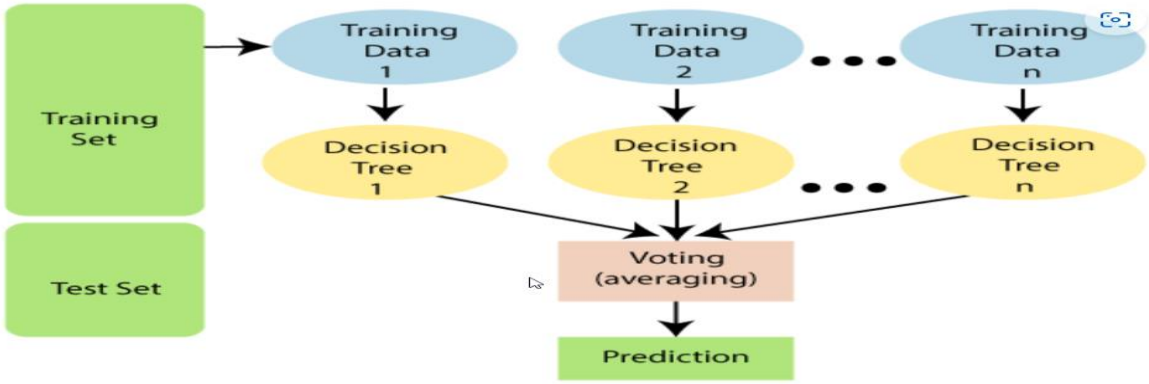
Confusion Matrix:



Random Forest Algorithm:

The Random Forest Algorithm implements the bagging approach by default and is based on decision trees. It constructs multiple decision tree and selects the most voted decision tree as the final prediction. More trees in the forest leads to higher accuracy and avoids replanting problems.

Sample Intuition of Random Forest:



In our case I had implemented RandomForestClassifier with hyper-parameter tuning methods as RandomizedSearchCV and GridSearchCV for better evaluation of the model.

Implementation Steps:

1. In Random forest implementation, I had used RandomizedSearchCV and GridSearchCV which helps in hyper-parameters tuning.

2. First, I had implemented RandomizedSearchCV to narrow down our search in the Grid model as Randomized search will select random subsets from the specified parameters and produce the best parameters for the passed parameters.

```
RandomizedSearchCV(cv=KFold(n_splits=3, random_state=42, shuffle=True),
                   estimator=RandomForestClassifier(random_state=42), n_iter=50,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [2, 7, 12, 17, 22, 27, 32,
                                                      37, 42, 47, 52, 57, 62,
                                                      67, 72, 77, 82, 87, 92,
                                                      97, None],
                                        'max_features': ['sqrt', 'log2',
                                                         'auto'],
                                        'min_samples_leaf': [0.005, 0.015,
                                                             0.025, 0.035,
                                                             0.045, 0.055],
                                        'min_samples_split': [0.005, 0.015,
                                                              0.025, 0.035,
                                                              0.045, 0.055],
                                        'n_estimators': [10, 15, 20, 25, 30, 35,
                                                         40, 45, 50, 55, 60, 65,
                                                         70, 75, 80, 85, 90, 95,
                                                         100, 105, 110, 115,
                                                         120, 125, 130, 135,
                                                         140, 145, 150, 155, ...]},
                   scoring='accuracy')
```

Best Parameters of Randomized Search:

```
randsecv_model.best_params_
```

```
{'n_estimators': 225,
 'min_samples_split': 0.005,
 'min_samples_leaf': 0.005,
 'max_features': 'sqrt',
 'max_depth': 67,
 'criterion': 'gini'}
```

3. After RandomizedSearchCV I am calculating accuracy using above parameters.

```
rand_fore_cl = RandomForestClassifier(n_estimators = randsecv_model.best_estimator_.n_estimators, criterio
rand_fore_cl.fit(x_train, y_train)
y_pred_rand_fore_rsv = rand_fore_cl.predict(x_test)
y_pred_rand_fore_rsv.shape
#score_randscv = round(accuracy_score(y_pred_rand_fore_rsv.round(), y_test)*100,2)
score_randscv= round(accuracy_score(y_test,y_pred_rand_fore_rsv)*100,2)
print(f'The Random Forest Classification uisng RandomizedSearch hyperparameter tuning Accuracy Score is :
```

```
The Random Forest Classification uisng RandomizedSearch hyperparameter tuning Accuracy Score is : 72.5%
```

4. Later, on the basis of RandomSearchCV results I implement GridSearchCV as it tries out each possible subset of the given parameters which takes lot of time to compute, s o to narrow down our running time I selected the output of RSV on different runs and provided them as input parameters to GridSearch CV.

```
model = GridSearchCV(rand_fore_cl, param_gscv, cv=kf, scoring='accuracy', n_jobs=4)
model.fit(x_train, y_train)

model.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 67,
 'max_features': 'sqrt',
 'min_samples_leaf': 0.005,
 'min_samples_split': 0.005,
 'n_estimators': 60}
```

5. Later I had utilized the best parameters of Grid Search to calculate the accuracy and comparing with RandomizedSearchCV accuracy and later plotting classification report of the model along with confusion matrix.

```
rand_for_gsv = RandomForestClassifier(n_estimators = model.best_estimator_.n_estimators, criterion=mode
rand_for_gsv.fit(x_train, y_train)
y_pred_rand_for_gsv = rand_for_gsv.predict(x_test)
y_pred_rand_for_gsv.shape
score_rand_gsv= round(accuracy_score(y_test,y_pred_rand_for_gsv)*100,2)
print(f'The Random Forest Classification uisng GridSearch hyperparameter tuning Accuracy Score is : {sc
```

```
The Random Forest Classification uisng GridSearch hyperparameter tuning Accuracy Score is : 73.36%
```

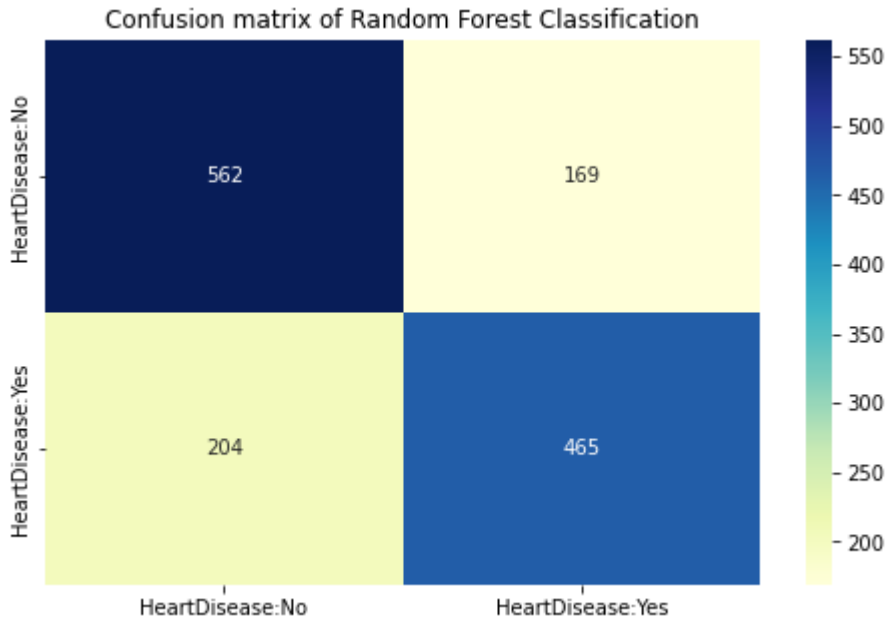
```
The Random Forest Classification Accuracy Score is : 73.36%
```

Classification report:

The F1 score of Random Forest Classification report is 71.3%

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.77      0.75       731
           1       0.73      0.70      0.71       669

    accuracy                           0.73      1400
   macro avg       0.73      0.73      0.73      1400
weighted avg       0.73      0.73      0.73      1400
```

Confusion Matrix:



XGB Classifier:

Extreme Gradient Boosting, or XGB, is a decision-tree-based ensemble machine learning technique that develops our model using a gradient boosting framework. This model is similar to the Random Forest model which uses the bagging technique where all of the decision tree forecasts are summed to provide the final prediction.

But XGB uses a collection of shallow decision trees that are repeatedly trained by GBDTs, using the error residuals from the earlier models to match the new model with each iteration. The weighted average of all the tree predictions represents the final projection.

Implementation Steps:

1. In XGB Classifier implementation, I had imported XGB Classifier using statement. "**from xgboost import XGBClassifier** ".

2. I had used parameters max_depth from the decision tree to narrow down my run time (tried other value too but the accuracy at those points is a bit less) and colsample_by tree whose value ranges from 0.5-1.

3. I had defined a function to classify and fit our model on our training data with depth equal to 4.

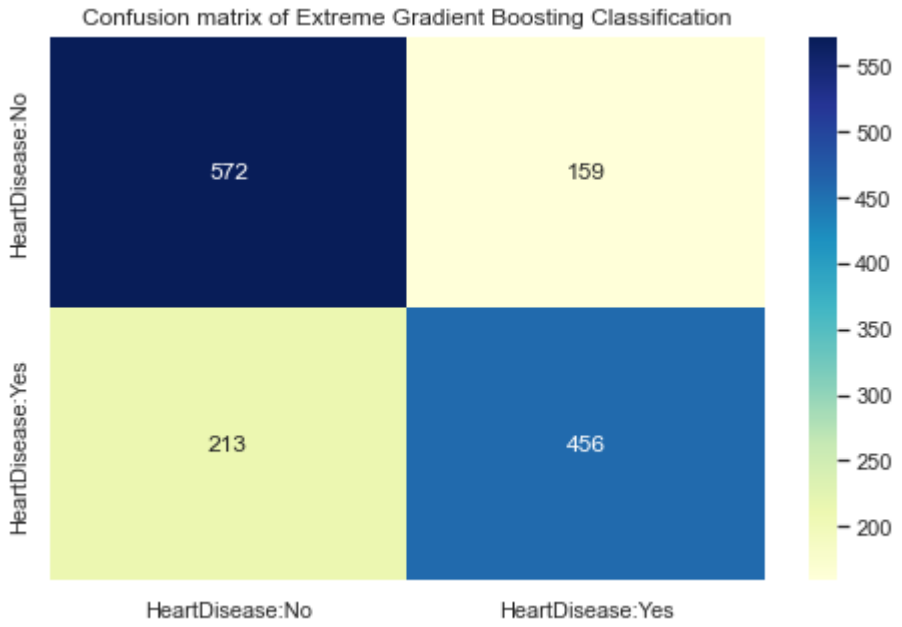
```
xgb_class = XGBClassifier(n_estimators=19,random_state=42,max_depth=4,colsample_bytree=0.8)

def train_xgclassi(classifier,x_train,y_train,x_test,y_test):

    classifier.fit(x_train,y_train)
    y_pred = classifier.predict(x_test)
    accuracy = accuracy_score(y_test,y_pred)
    return y_pred,accuracy

auc_scr = []
confu_mat={}

y_pred_xgb,current_accuracy= train_xgclassi(xgb_class, x_train, y_train, x_test, y_test)
print("For ",xgb_class)
print("Accuracy - ",current_accuracy)
```

4. Later I had calculated accuracy and confusion matrix for this model.

```
XGB_score=round(auc_scr[0]*100,2)
print(f'The XGB  Classifier Accuracy Score is : {XGB_score}%')
```

```
The XGB  Classifier Accuracy Score is : 73.43%
```

Confusion Matrix:



Classification report:

The F1 score of XGB Classification is 71.42%.

XGB Classification Report:

	precision	recall	f1-score	support
0	0.73	0.78	0.75	731
1	0.74	0.68	0.71	669
accuracy			0.73	1400
macro avg	0.74	0.73	0.73	1400
weighted avg	0.73	0.73	0.73	1400

Stochastic gradient descent (SGD):

SGD is a type of gradient descent approach where it uses single data point to minimize the loss instead of using the whole dataset as traditional GD does. In my implementation of SGD, I had used parfit as a optimizer.

Reason for getting low accuracy than logistic regression: Logistic regression by default use Vanila optimizer in which whole dataset is fed into the model. But SGD divides the data into small pieces and fed into the model in which only small amount data is fed because of which the accuracy is a bit low.

Implementation Steps:

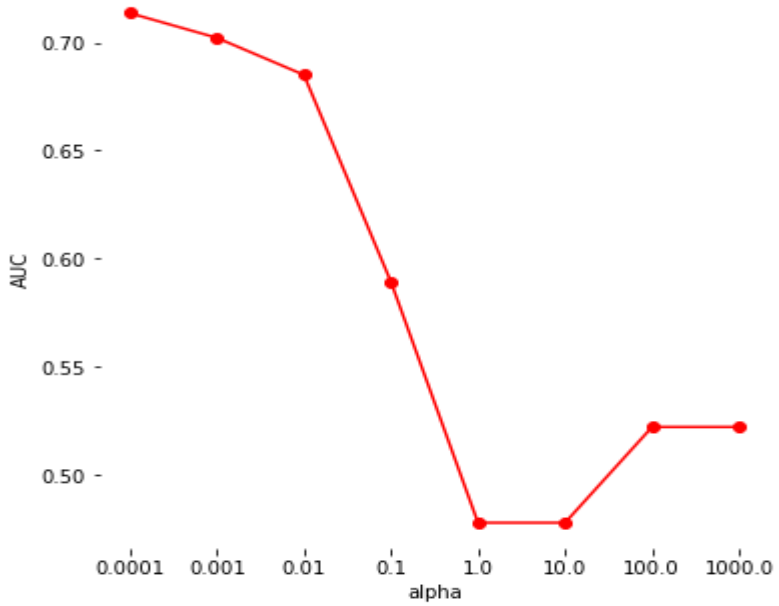
1. In SGD implementation, I had used parfit as the optimizer mechanism.

2. Used loss function as logistic loss which gives logistic regression.

3. Specified the values of learning rate as:

```
grid = {
    'alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3], # Learning rate
    'loss': ['log'],
    'penalty': ['l1'],
    'n_jobs': [4]
}
paramGrid = ParameterGrid(grid)
sgd=SGDClassifier()
bestModel, bestScore, allModels, allScores = pf.bestFit(sgd, paramGrid,x_train, y_train, x_test, y_test, metric = accuracy_score,
bestScore*=100
print(bestModel, bestScore)
```

4. I plotted the graph among learning rate and accuracy to get at which point accuracy is high.



5. After getting the best params I had used to calculate the accuracy and plotted confusion matrix and classification report of the model trained.

The SGD Classifier Accuracy Score is: 71.5%

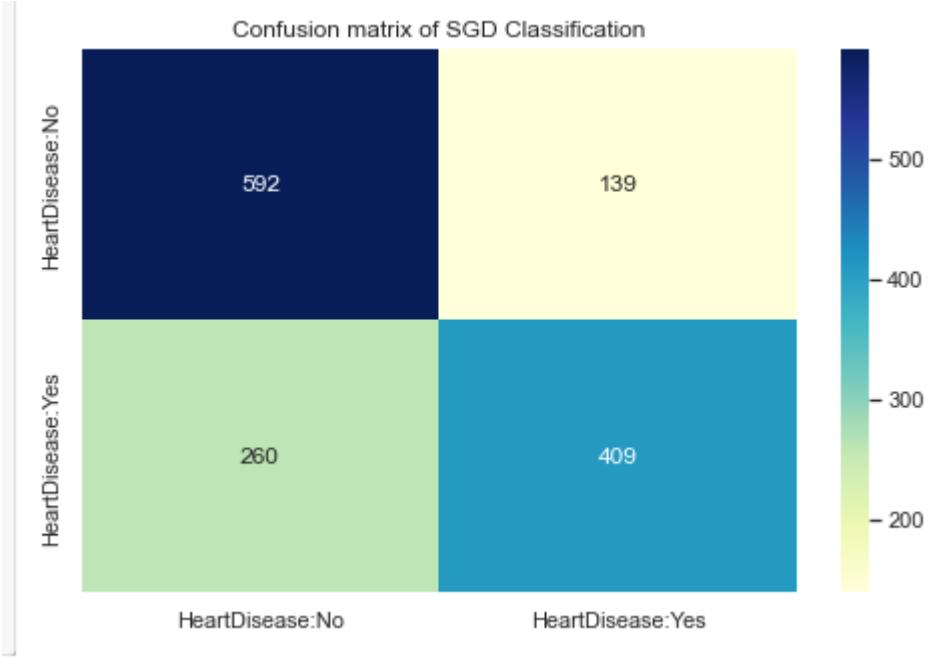
Classification report:

The F1 score of SGD classification is 67.21%.

SGD Classification Report:

	precision	recall	f1-score	support
0	0.69	0.81	0.75	731
1	0.75	0.61	0.67	669
accuracy			0.71	1400
macro avg	0.72	0.71	0.71	1400
weighted avg	0.72	0.71	0.71	1400

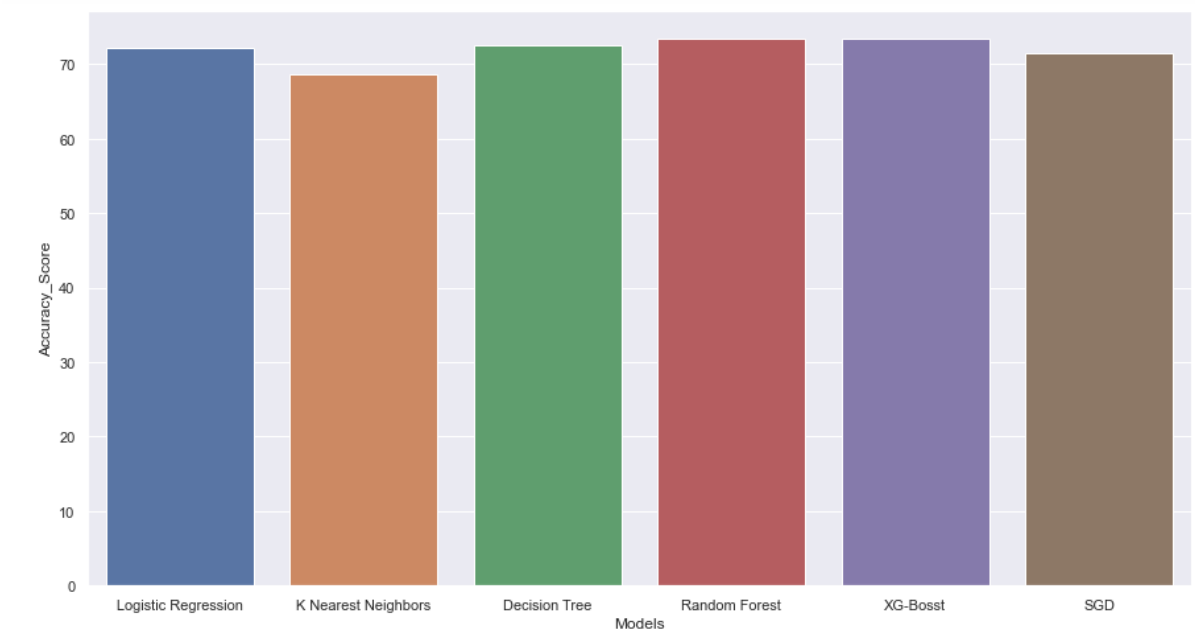
Confusion matrix:



Experimental Results:

The Analysis of all the models trained.

	Models	Accuracy_Score
4	XG-Bosst	73.43
3	Random Forest	73.36
2	Decision Tree	72.57
0	Logistic Regression	72.18
5	SGD	71.50
1	K Nearest Neighbors	68.58



Discussion:

From the above analysis using the various metrics for evaluation of various models.
I used:

1) Accuracy
2) Confusion matrix
3) Other metric such as F1 score, precision, recall etc.

All models performed well at some point but the XG-boost has outperformed every other model implemented because of its inbuilt implementation of gradient boosting decision tree algorithm.

Challenges:

I had gone through various models and implementations of the classifiers which took a great deal of time.
But still there is lot of room for improvement in terms of pre-processing and EDA.

Even I tried various models the accuracy is the major challenge. According to the work I have done

1) Pre-processing
2) EDA
3) Perfect parameters for the model
4) Hyper parameter tuning
5) Various evaluation metrics

These things can help us improve us with improvement in accuracy and also instead of using only accuracy and F1-score we can try out other classification metrics like r2 value, roc accuracy score to get better of our model.

Conclusion:

At last, I would like to conclude that with a well refined dataset and various experimentations using that data in machine learning. Can cause a major impact on the lives.

This was a minor implementation of results that the everyday data can help us predict the user from CVA (Cardio vascular attacks).

References:

https://blog.mlreview.com/parfit-hyper-parameter-optimization-77253e7e175e
https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/
https://machinelearningmastery.com/rfe-feature-selection-in-python/
https://scikit-learn.org/stable/modules/grid_search.html
https://www.projectpro.io/recipes/optimize-hyper-parameters-of-decisiontree-model-using-grid-search-in-python