PROJECT INTERIM REPORT

Batch details	PGPDSE-FT Bengaluru Aug 2022
Team members	Harshad Bhole Kunal Patil Prajwal Salutagi Rahul C Reji Shreyasi Kundu
Domain of Project	Healthcare
Proposed project title	Cardiovascular Disease Predictiom
Group Number	Group 5
Team Leader	Kunal Patil
Mentor Name	Dr. Debasish Roy

Date: 27-01-2023

Signature of the Mentor
(Dr. Debashish Roy)

Signature of the Team Leader
(Kunal Patil)

INDEX

1.BUSINESS UNDERSTANDING:

1.1.1 BUSINESS PROBLEM STATEMENT:

Cardiovascular disease (CVD) is one of the world's biggest preventable killers and has become an emerging problem throughout the developing world. CVD and its associated diseases are costly and can greatly impact the workforce and its productivity. The mortality or high-risk rate can be reduced if an early detection system for cardiovascular disease is introduced and for that we need to study the data carefully. A proper and careful study of regarding the previous data can be carried out to extract some important and interesting insight that may help out reduce cost of healthcare.

1.2 Business Objective: Our goal is to identify the parameters influencing an individual patient and encourage them to adapting healthy Life-Style. We will try to early identification of cardiovascular diseases using various ML classification algorithms. Hopefully can be of great help for early detection and management of cardiovascular diseases.

1.2TOPIC SURVEY:

1.2.1Problem understanding:

Cardiovascular diseases cases are rising quickly every day, thus it's crucial and worrisome to predict any potential illnesses in advance. This diagnosis is a challenging task that requires accuracy and efficiency. The primary focus of the research paper is on which patients, given certain medical characteristics, are more likely to suffer cardiovascular disease. Using the patient's medical history, we will develop a system to determine if a cvd disease diagnosis is likely or not for the patient. To forecast and categorise the patient with cardiovascular disease, we will employ a variety of machine learning methods, including KNN and logistic regression. To limit how the model can be utilised, a very helpful way is the regulation to increase the precision of cvds prediction in any individual will be done. Thus, utilizing the model to determine the likelihood that the classifier can correctly and accurately identify cvds illness will relieved quite a bit of pressure. The mentioned technique for predicting cardiac disease will improve patient treatment while costing less

1.2.2Current solution to the problem:

In India, huge mortality occurs due to cardiovascular diseases (CVDs) as these diseases are not diagnosed in early stages. Machine learning (ML) algorithms can be used to build efficient and economical prediction system for early diagnosis of CVDs in India. . This initiative will provide us with important information that can help us predict the patients with cardiovascular disease and It is implemented on the.pynb format.

1.2.3Proposed solution to the problem:

A total of 70000 entries with 14 independent variables and and cardio as target Variable. anonymized medical records are collected from which Seventy percent of the collected data will be used to train the prediction system. Five state-of-the-art ML algorithms (k-Nearest Neighbours, Naïve Bayes, Logistic Regression, AdaBoost and Random Forest [RF]) will be applied using Python programming language to develop the prediction system. The performance will be

evaluated over remaining 30% of the data. Classification – Random Forest, Decision Tree, Naive Bayes, KNN, XGBoost, Logistic Regression.

1.3 CRITICAL ASSESSMENT OF TOPIC SURVEY:

Cardiovascular disease (CVD) is the leading cause of death and a major cause of disability worldwide. CVDs are the leading cause of death globally. An estimated 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of those deaths, 85% were due to heart attack and stroke. Most cardiovascular diseases can be prevented by addressing behavioral risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol. As the leading cause of death globally, it is important to detect cardiovascular disease as early as possible so that management with counselling and medicines can begin. This project try a better way of improving the timely prediction of cardiovascular diseases in suspected patients.

[[**References**: 1. Aadar Pandita, Siddharth Vashisht, Aryan Tyagi, Prof. Sarita Yadav.Review Paper on Prediction of Heart Disease using Machine Learning Algorithms .

2. Armin Yazdani, Kasturi Dewi Varathan, Yin Kia Chiam, Asad Waqar Malik & Wan Azman Wan Ahmad *BMC Medical Informatics and Decision Making* volume. A novel approach for heart disease prediction using strength scores with significant predictors.

3. Harshit Jindal[1], Sarthak Agrawal[1], Rishabh Khera[1], Rachna Jain[2] and Preeti Nagrath[2] Published under licence by IOP Publishing Ltd. Heart disease prediction using machine learning algorithms.]]

2.DATA UNDERSTANDING:

2.1 DATA DICTIONARY:

INDEPENDENT VARIABLES:

S.No	Feature Name	Feature Description
1.	id	ID of the patient
2.	age in days	The age of patient in days
3.	age in years	The age of patient in years
4.	height	The height of patients
5.	weight	The weight of patients
6.	gender	The gender of patients
7.	ap_hi	systolic blood pressure
8.	ap_lo	diastolic blood pressure
9.	cholesterol	Cholesterol level of patients
10.	glucose	Glucose level of patients
11.	smoking	Patient is habitual to smoking or not
12.	alcohol	Patient is habitual to alcohol intake or not
13.	active	Active life or passive life

DEPENDENT VARIABLE (TARGET VARIABLE):

cardio--This is the target variable of the dataset that indicates whether or not the patient has a cardiovascular disease

2.2 VARIABLE CATEGORIZATION:

Independent variables:

Numerical column: 7

Categorical column: 7

Target variable:

Categorical - 1

Total columns: 15

3.DATA PREPROCESSING:

3.1DISTRIBUTION OF VARIABLES:

The data given to us is in shape a 70000 * 15 data frame. The data consists of Numerical and Categorical data. While further analyzing the data, we realize that the categorical data is mostly Binary flagged as 0 or 1. The id and unnamed column has a unique value for each row, so it is the index column and should be dropped. And also, age_days

Data info:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Unnamed: 0   70000 non-null  int64  
 1   id           70000 non-null  int64  
 2   age_days     70000 non-null  int64  
 3   age_year     70000 non-null  float64
 4   gender       70000 non-null  int64  
 5   height       70000 non-null  int64  
 6   weight       70000 non-null  float64
 7   ap_hi        70000 non-null  int64  
 8   ap_lo        70000 non-null  int64  
 9   cholesterol  70000 non-null  int64  
 10  gluc         70000 non-null  int64  
 11  smoke        70000 non-null  int64  
 12  alco         70000 non-null  int64  
 13  active       70000 non-null  int64  
 14  cardio       70000 non-null  int64  
dtypes: float64(2), int64(13)
memory usage: 8.0 MB
```

3.2 NULL VALUE TREATMENT:

Null value treatment is essential to building most of the commonly used machine learning classification models such as logistic regression, decision tree, KNN, and others. To infer that we have used isnull() function the null values from the dataset.

```
: df.isnull().sum()

: Unnamed: 0     0
  id             0
  age_days       0
  age_year       0
  gender         0
  height         0
  weight         0
  ap_hi          0
  ap_lo          0
  cholesterol    0
  gluc           0
  smoke          0
  alco           0
  active         0
  cardio         0
  dtype: int64
```

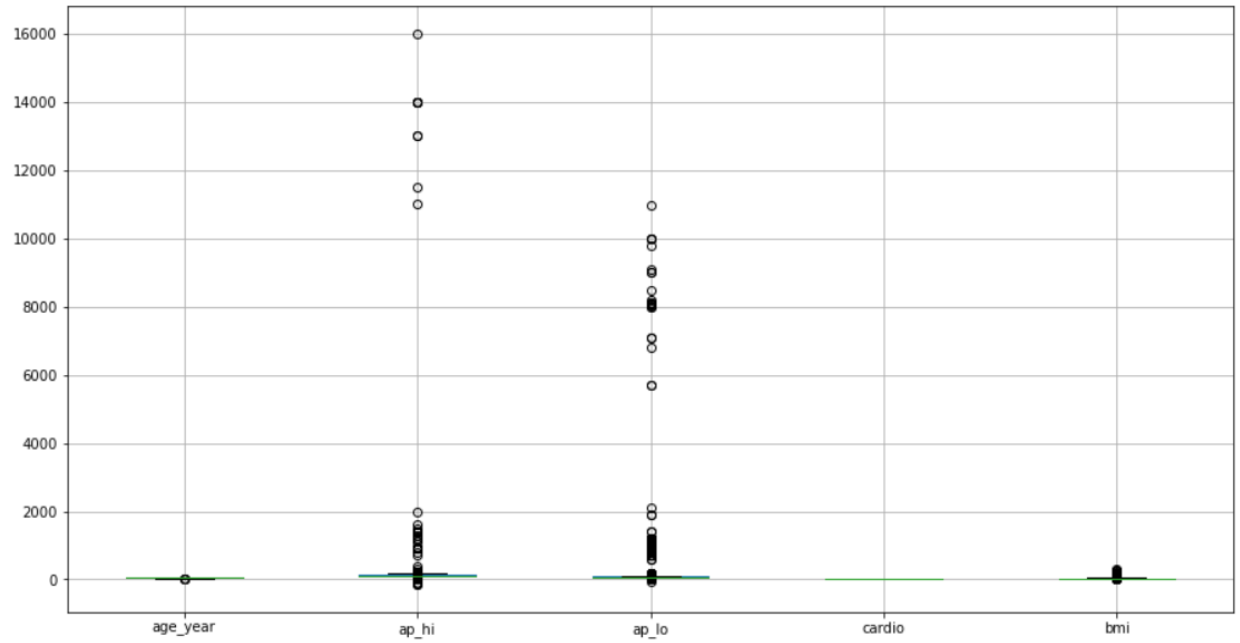
From the above figure, it is evident that there is no missing value in the dataframe.

3.3 PRESENCE OF OUTLIERS AND TREATMENT:

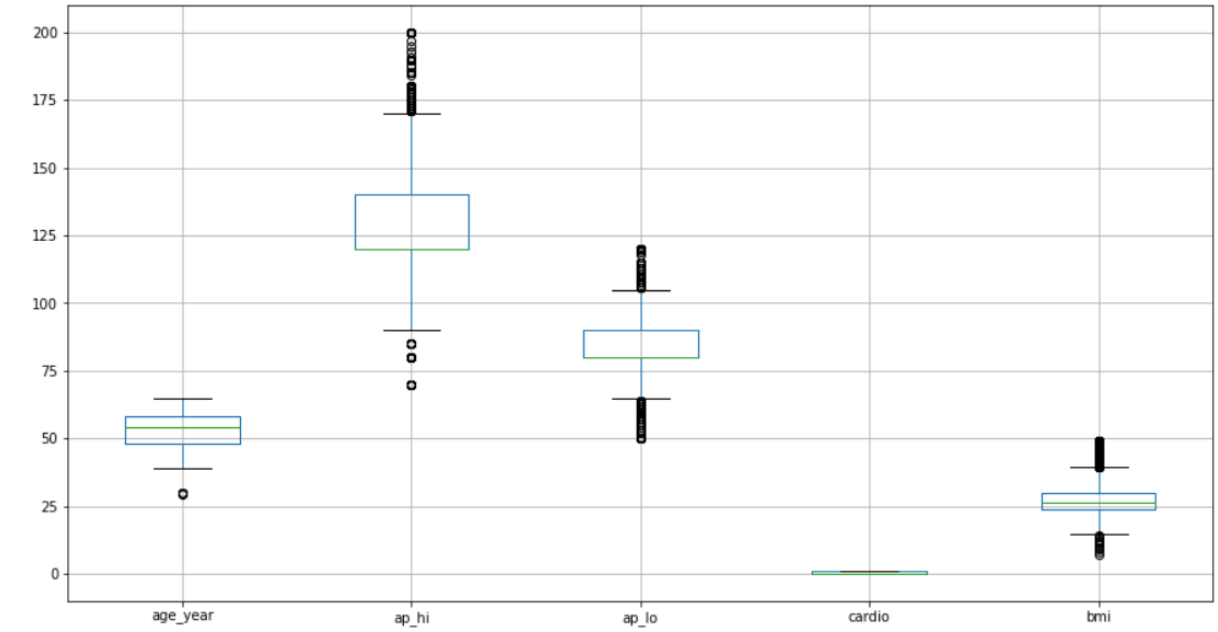
There are about 14 numerical variables on which the presence of outliers is to be determined. A distribution was assumed to be skewed when the skewness is outside the range of ±0.5 as it is quite impossible to have a real dataset with skewness of each variable or at least one of the variables with a perfect zero skewness. It is also understood that as the value of skewness increases, the farther the outlier is. Out of the 14 numerical variables, it was found that all numerical variables were outside the range of acceptable skewness. The skewness values for those 14 numerical variables are shown below:

```
df_num.skew()

age_year    -0.307055
height      -0.642187
weight       1.012070
ap_hi       85.296214
ap_lo       32.114083
dtype: float64
```

Outlier Detection



From the boxplot we can see that age_year, height, weight, ap_hi, ap_lo is having outliers. Once there are outliers outside the acceptable range, it has to be treated. Dropping the rows with outliers are recommended as not more than 5% the dataset is getting dropped. After dropping of outliers, the result is as follows:



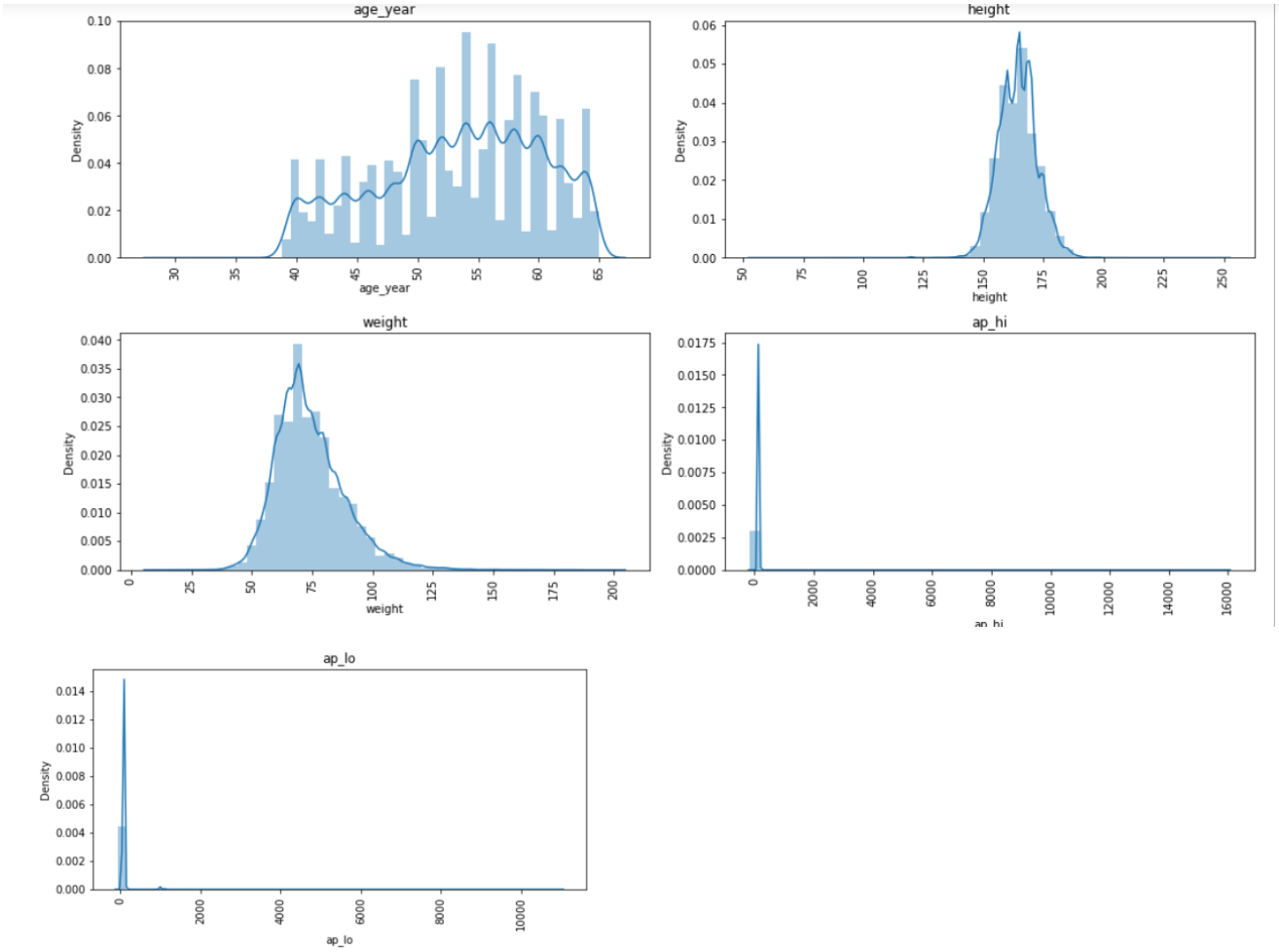
Here we remove outlier by using IQR method with outer fence as limit, because outer fence gives more flexibility to the data as well as less leakage of the data. Formulae as follows:

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3-Q1

upperfense = Q3+3*IQR
lowerfense = Q1-3*IQR
```

4.Exploratory Data Analysis

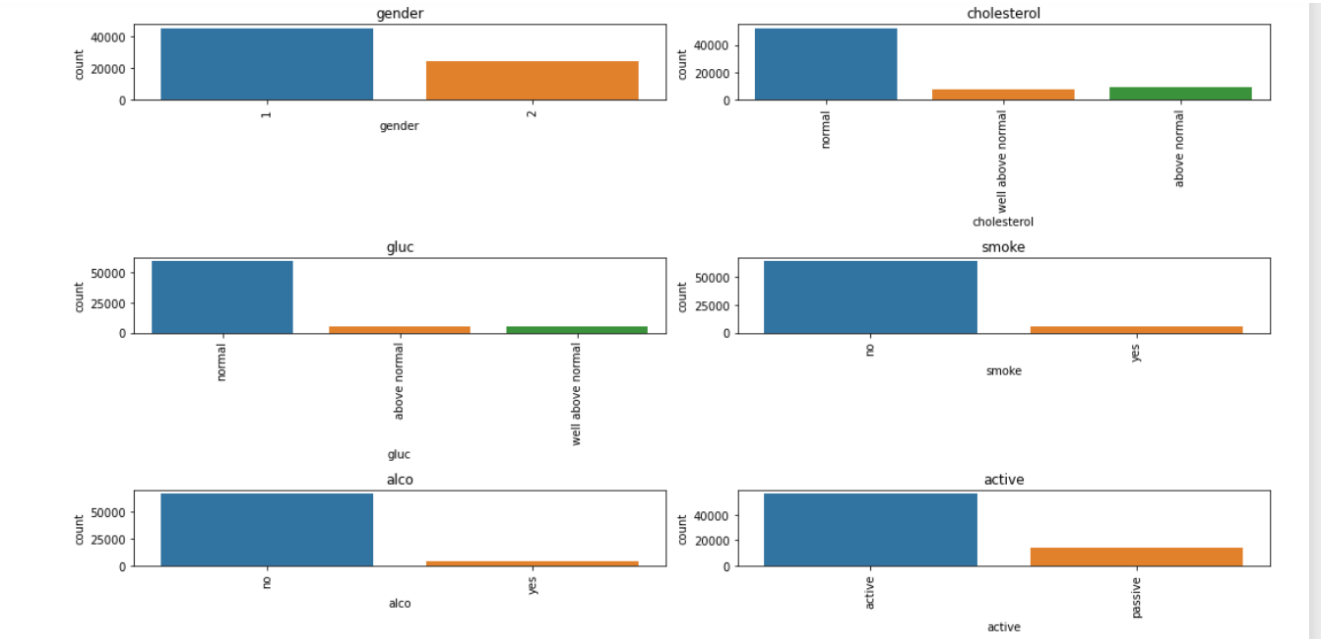
4.1 Univariate and bivariate analysis

Univariate Analysis of Numerical variable:



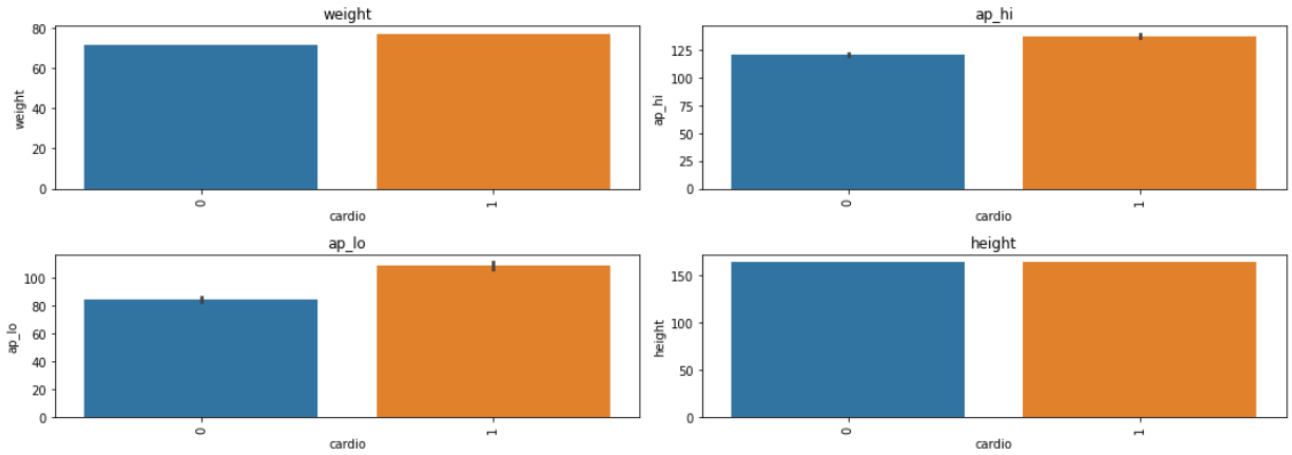
- age_year: This columns are looking uniformly distributed with multiple modes. For this columns we can do binning to make them simple to understand. But at the same time both are giving same information, so we can treat age_days as unnecessary column and drop it.
- height and weight: This columns are looking nearly normally distributed.
- ap_hi and ap_lo: This columns have very low range and have some outliers. Thats why they are looking skewed. In adults, blood pressure is considered to be normal under a systolic value of 140 mmHg and under a diastolic value of 90 mm

Univariate Analysis of Categorical variable:



- In Cholesterol their are most of the patients are in the normal condition & Near about 10k patients are well above normal & 12k patients are in above normal condition.
- In Gluc near about sixtiy thousand patients have normal condition & Five thousand patients are above normal & Five thousand patients are in well above normal condition.
- In Smoke column there are above sixty thousand patients who don't somke.
- In Alco column there are above sixty thousand patients who don't Drink alcho.
- In active columns there are near about Fifty thousand people who are active.

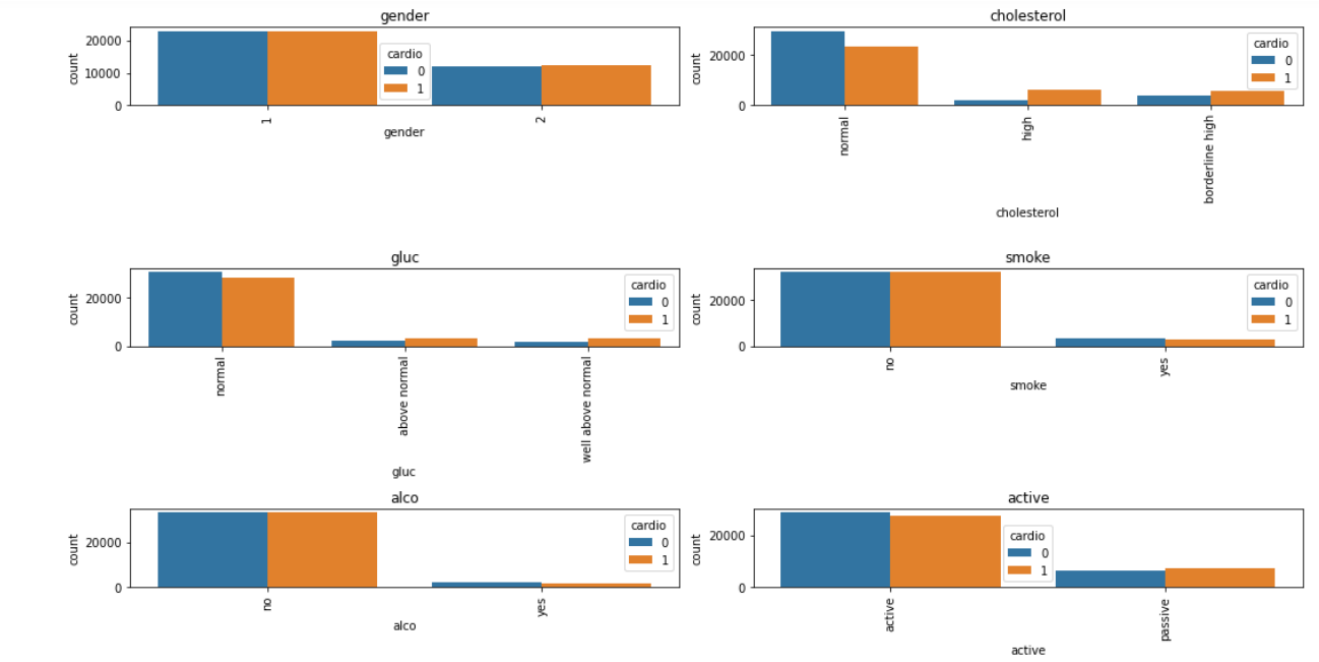
Bivariate analysis of numeric data:



- Weight is showing minor correlation with the target variable, as we can see that increase in weight have high chances of cardiovascular disease.
- From ap_hi and ap_lo we can say that, as ones systolic or diastolic blood pressure increases their chances of getting cardiovascular disease also increases.
- As weight and height have no significant impact on cardiovascular disease so we can

make new feature out of them in the form of bmi column.

Bivariate analysis of categorical data:



- As cholesterol is getting towards high level, the chances of cardiovascular disease is increasing.
- Also as glucose is increasing above normal level, the chances of getting cardiovascular disease is increasing.
- As passive lifestyle is increasing the chances of getting cardiovascular disease is increasing.
- Cardiovascular disease are not based on the gender of the person.

4.2 Multivariate Analysis:

- The age_year, ap_hi and ap_lo are correlated with target variable than any other columns but still they are not significantly related.
- As a reason, we cannot use linear models .

4.3 Feature Engineering:

REDUNDANT FEATURES :

age_days-Since these feature is showing the age of patient in days , we are dropping it because the other feature age_year is giving same information of patients.

Unnamed, id- These are the unique IDs of the respective features, which would not be considered for analysis, hence it can be dropped.

height and weight- since there is a column for BMI which is calculated using height and weight these columns will have severe multicollinearity, so we drop them.

gender- since it is showing no relation with the gender of the patient

Before dropping redundant features:

- Age_days and age_year are showing very high correlation as both are showing the same information.
- Weight and height is showing not that much correlation with target variable.

After dropping redundant features and outlier treatment the results we got:

- Marginal improvement in the relation between bmi(which replaced weight and height) and target variable.
- Increase in the relation between ap_hi and ap_lo with target variable.
- Also Increase in the relation between age_year with target variable.

Scaling the data:

	age_year	ap_hi	ap_lo	bmi
0	-0.433435	-1.002596	-0.136398	-1.075423
1	0.309894	0.818762	0.936638	1.506656
2	-0.245476	0.211643	-1.209435	-0.768479
3	-0.745349	1.425882	2.009675	0.268029
4	-0.805707	-1.609715	-2.282472	-0.867420
5	0.992866	-0.395476	-0.136398	0.402346
6	1.073478	0.211643	-0.136398	2.064896
7	1.264272	0.211643	0.936638	0.521665
8	-0.727121	-1.002596	-1.209435	0.214332
9	0.150291	-1.002596	-2.282472	-0.414900

Numerical variables after scaling

StandardScaler standardizes a feature by subtracting the mean and then scaling to unit variance. Unit variance means dividing all the values by the standard deviation.

$$z = \frac{x - \mu}{\sigma}$$

x = Observed value

μ = Mean

σ = Standard deviation

StandardScaler results in a distribution with a standard deviation equal to 1. The variance is equal to 1 also, because variance = standard deviation squared. And 1 squared = 1. StandardScaler makes the mean of the distribution 0. About 68% of the values will lie be between -1 and 1.

Categorical Encoding:

Label Encoding:

This approach is very simple and it involves converting each value in a column to a number. Consider a dataset of bridges having a column names bridge-types having below values. Though there will be many more columns in the dataset, to understand label-encoding, we will focus on one categorical column only

5 BASELINE MODEL BUILDING:

DATA SPLITTING: For model building, we split the data in train and test. For validation purpose, we reserve 30% data as a test data and remaining 70% for training the model.

```
: 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=4)
  print ('Train set:', X_train.shape,  y_train.shape)
  print ('Test set:', X_test.shape,  y_test.shape)

  Train set: (47845, 9) (47845,)
  Test set: (20506, 9) (20506,)
```

Criterias for model evaluation:

We have used Decision tree classification models as base model and the metrics that we used to validate our model is

- **f1 score**: The **F1-score** combines the precision and recall of a classifier into a single metric by taking their harmonic mean. It is primarily used to compare the performance of two classifiers.
- **Accuracy score** : One of the widely used metrics that computes the performance of classification models is accuracy. The percentage of labels that our model successfully predicted is represented by accuracy. For instance, if our model accurately classified 80 of 100 labels, its accuracy would be 0.80.

Other criterias:

Kappa score: The Cohen Kappa Score is used to compare the predicted labels from a model with the actual labels in the data. The score ranges from -1 (worst possible performance) to 1 (best possible performance). A Cohen Kappa Score of 0 means that the model is no better than random guessing, and a score of 1 means that the model is perfect.

5.1 Decision Tree Classifier:

A decision tree is a supervised learning approach used to solve classification and regression issues, but it is mostly used to solve classification issues. It is a classifier organized by the tree structure, where the internal nodes are the data variables, the branches are the decision rules, and each node is the output. It consists of two nodes. One of them is a decision node used for decision-making, and it has various branches. The second node is a leaf node, which represents the result of these decisions.

A decision node is a node on which a decision of split is to be made. A node that cannot be split further is known as the terminal/leaf node. A leaf node represents the decision. A decision tree can work with both numerical and categorical variables.

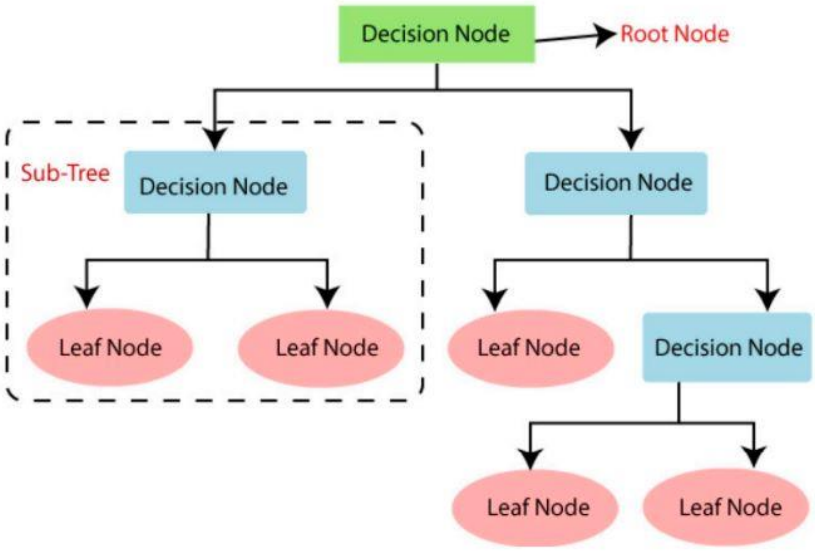


Fig.Decision Tree model

```
DecisionTree
[[7995 2377]
 [3198 6936]]
Accurancy: 73%
f1_score 71.33233917827943
Classification report train data
              precision    recall  f1-score   support

           0       0.74      0.79      0.76     24243
           1       0.76      0.71      0.74     23602

    accuracy                           0.75     47845
   macro avg       0.75      0.75      0.75     47845
weighted avg       0.75      0.75      0.75     47845

Classification report
              precision    recall  f1-score   support

           0       0.71      0.77      0.74     10372
           1       0.74      0.68      0.71     10134

    accuracy                           0.73     20506
   macro avg       0.73      0.73      0.73     20506
weighted avg       0.73      0.73      0.73     20506
```

- **Hyperparameter tuning of decision tree:**

After Base model, we try to do Hyperparameter tuning of decision tree by giving following parameters. We got best parameters for decision tree classifier as follows:

```
tuned_paramaters = [{'criterion': ['entropy', 'gini'],
                     'max_depth': range(2, 15),
                    'max_features': ["sqrt", "log2"],
                     'min_samples_split': range(2,10),
                     'min_samples_leaf': range(1,10)
                     #'max_leaf_nodes': range(1, 10)
}]
decision_tree_classification=DecisionTreeClassifier(random_state = 10)
tree_grid = GridSearchCV(estimator = decision_tree_classification, param_grid = tuned_paramaters, cv =20)
tree_grid_model = tree_grid.fit(X_train, y_train)
print('Best parameters for decision tree classifier: ', tree_grid_model.best_params_, '\n')
```

```
Best parameters for decision tree classifier:  {'criterion': 'gini', 'max_depth': 8, 'max_features': 'sqrt', 'min_samples_lea
f': 2, 'min_samples_split': 6}
```

After finding best parameters, we built another tuned model of decision tree.

- **Decision tree Tuned model:**

```
dt_tuned=DecisionTreeClassifier(criterion="gini",max_depth=8,max_features="sqrt",min_samples_split=6,min_samples_leaf=2)
dt_tuned.fit(X_train,y_train)
y_pred=dt_tuned.predict(X_test)
metrics(y_test,y_pred,"DT")
```

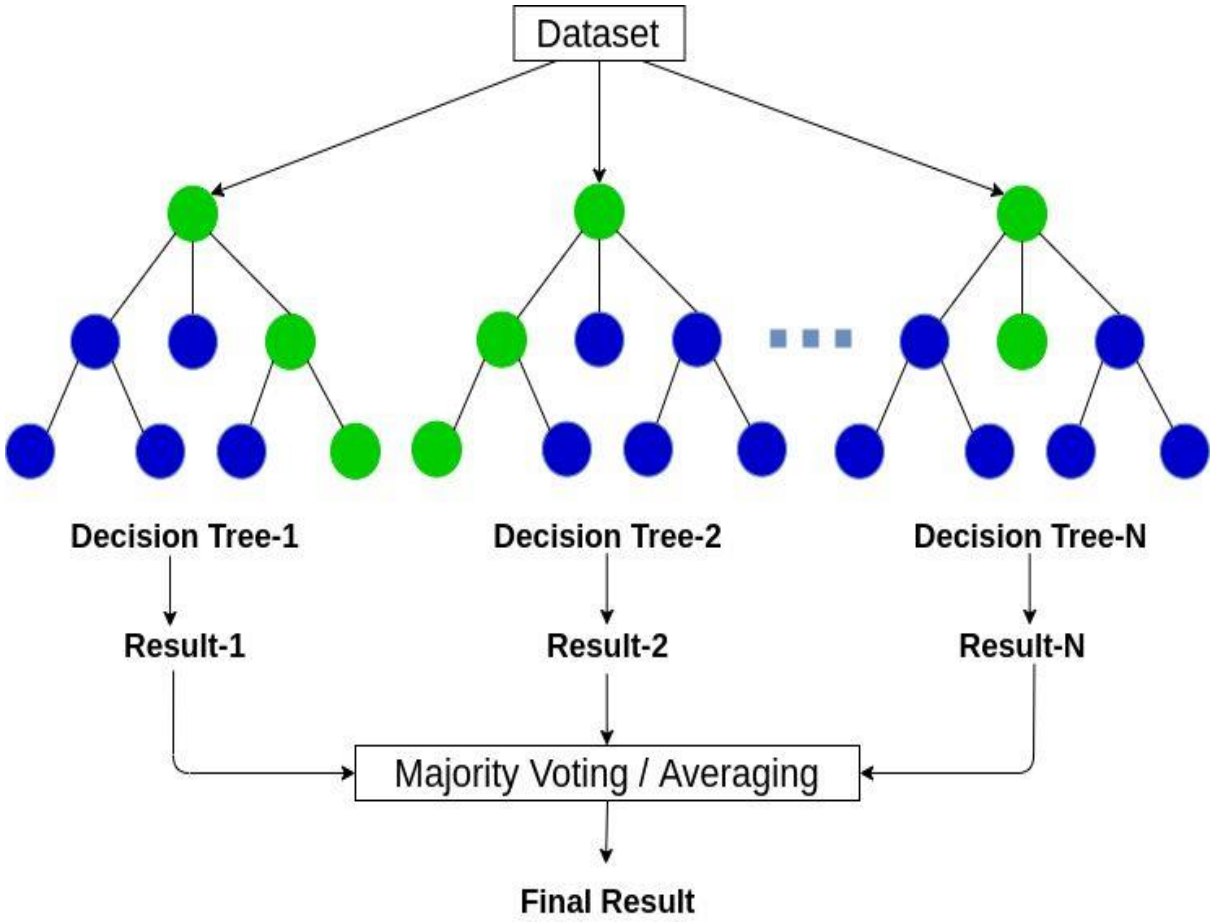
```
The metrics for the model DT

Accuracy score: 0.7219811826646517

recall score: 0.6764356435643565

precision score: 0.7372396676378548

f1 score: 0.7055300253007695

roc auc score: 0.7212966655351793
Classification Report
              precision    recall  f1-score   support

           0       0.71      0.77      0.74     10413
           1       0.74      0.68      0.71     10100

    accuracy                           0.72     20513
   macro avg       0.72      0.72      0.72     20513
weighted avg       0.72      0.72      0.72     20513
```

After performing hyperparameter tuning, the values for recall is same for both 0 and 1 but accuracy score declined.

5.2 RANDOM FOREST :

It is the method of constructing multiple decision trees on randomly selected data samples. We can use the bootstrap sampling method to select the random samples of the same size from the dataset to construct multiple trees. This method is used for both regression and classification analysis.

The random forest returns the prediction based on all the individual decision trees prediction. It avoids the over-fitting problem as it considers a random data sample to construct a decision tree.



```
Random Forest
[[7548 2824]
 [3216 6918]]
Accurancy: 71%
f1_score 69.61159186959146
Classification report train data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     24243
           1       1.00      1.00      1.00     23602

    accuracy                           1.00     47845
   macro avg       1.00      1.00      1.00     47845
weighted avg       1.00      1.00      1.00     47845

Classification report
              precision    recall  f1-score   support

           0       0.70      0.73      0.71     10372
           1       0.71      0.68      0.70     10134

    accuracy                           0.71     20506
   macro avg       0.71      0.71      0.71     20506
weighted avg       0.71      0.71      0.71     20506
```

- **Random forest tuning parameters:**

After Base model, we try to do Hyperparameter tuning of Random forest by giving following parameters. We got best parameters for Random forest classifier as follows:

```
cv.best_estimator_
```

```
RandomForestClassifier(max_features='log2', min_samples_split=8,
                       n_estimators=200)
```

After finding best parameters, we built another tuned model of Random Forest.

- **RFC tuned model:**

```
rfc_tuned=RandomForestClassifier(n_estimators=200,criterion="gini",max_features="sqrt",min_samples_split=8)
rfc_tuned.fit(X_train,y_train)
y_pred=rfc_tuned.predict(X_test)
metrics(y_test,y_pred,"RFC")
```

```
The metrics for the model RFC

Accuracy score: 0.7270023887290986

recall score: 0.6988118811881188

precision score: 0.7339850249584027

f1 score: 0.7159667275309393

roc auc score: 0.7265787054072737
Classification Report
              precision    recall  f1-score   support

           0       0.72      0.75      0.74     10413
           1       0.73      0.70      0.72     10100

    accuracy                           0.73     20513
   macro avg       0.73      0.73      0.73     20513
weighted avg       0.73      0.73      0.73     20513
```

After performing hyperparameter tuning, the values for recall are optimum for both 0 and 1 also accuracy score remained same.

5.3 NAÏVE BAYES

In statistics, **naive Bayes classifiers** are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features (see Bayes classifier). They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve high accuracy levels.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

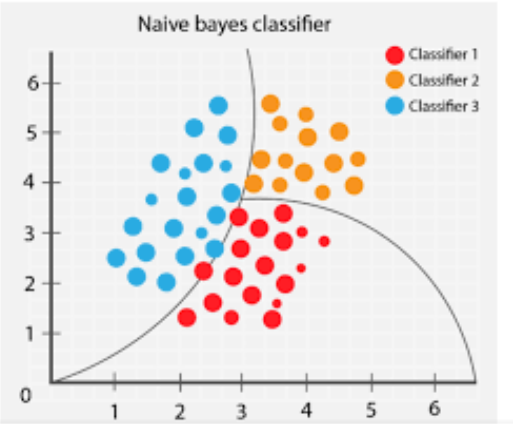
Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some *n* features (independent variables), it assigns to this instance probabilities $p(C_k \mid x_1, \ldots, x_n)$ for each of *K* possible outcomes or *classes* .

The problem with the above formulation is that if the number of features *n* is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. The model must therefore be reformulated to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In plain English, using Bayesian probability terminology, the above equation can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$



```
Naive Bayes
[[7548 2824]
 [3216 6918]]
Accurancy: 72%
f1_score 69.61159186959146
Classification report train data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     24243
           1       1.00      1.00      1.00     23602

    accuracy                           1.00     47845
   macro avg       1.00      1.00      1.00     47845
weighted avg       1.00      1.00      1.00     47845

Classification report
              precision    recall  f1-score   support

           0       0.70      0.73      0.71     10372
           1       0.71      0.68      0.70     10134

    accuracy                           0.71     20506
   macro avg       0.71      0.71      0.71     20506
weighted avg       0.71      0.71      0.71     20506
```

- **Naïve Bayes After Binning**

After Base model, we try to bin independent variables ap_hi, ap_lo, age_year and bmi and built new model. We got following results:

```
gnb = GaussianNB()
gnb_model = gnb.fit(X_train, y_train)
test_report = get_test_report(gnb_model, test_data=X_test)
print(test_report)
## After binning we can see that naive bias has improved some
```

	precision	recall	f1-score	support
0	0.69	0.79	0.74	10413
1	0.75	0.63	0.69	10100
accuracy			0.71	20513
macro avg	0.72	0.71	0.71	20513
weighted avg	0.72	0.71	0.71	20513

The accuracy score for the base model and bin model is same but notable changes can be seen in recall values - The recall value for 0 improved from 0.73 to 0.78 but at the same time for 1 it declined from 0.68 to 0.63.

5.4KNN Base Model:

KNeighborsClassifier is a classifier implemented in the scikit-learn library in python that is based on the k-nearest neighbors algorithm. The k-nearest neighbors algorithm is a simple, non-parametric method that is used for both classification and regression tasks.

In the classification task, the KNeighborsClassifier class finds the k-nearest samples in the training data for each point in the test data, and assigns the most common class label among those k-nearest samples to the test point. The number of k-nearest samples to consider is controlled by the hyperparameter k, which is an integer value.

```
|: ### Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("KNeighborsClassifier")
print(cm)
print('Accurancy: {:.0f}%'.format(model.score(X_test, y_test)*100))
```

```
KNeighborsClassifier
[[8708 1705]
 [5636 4464]]
Accurancy: 64%
```

- **KNN After binning (using MinMax Scaler)**

```
knn_classification = KNeighborsClassifier(n_neighbors = 3)

# fit the model using fit() on train data
knn_model_3 = knn_classification.fit(X_train_scaled, y_train)
test_report=get_test_report(knn_model_3,X_test_scaled)
print(test_report)
```

	precision	recall	f1-score	support
0	0.64	0.54	0.59	10413
1	0.59	0.69	0.64	10100
accuracy			0.61	20513
macro avg	0.62	0.62	0.61	20513
weighted avg	0.62	0.61	0.61	20513

After binning and using minmax scaler the accuracy score decreased from 0.64 to 0.61.

- **KNN After binning (using STD Scaler)**

```
knn_classification = KNeighborsClassifier(n_neighbors = 3)

# fit the model using fit() on train data
knn_model_3 = knn_classification.fit(X_train_std, y_train)
test_report=get_test_report(knn_model_3,X_test_std)
print(test_report)
## here we can see that metrix values improved and hence we can tune it
```

	precision	recall	f1-score	support
0	0.68	0.62	0.65	10413
1	0.64	0.70	0.67	10100
accuracy			0.66	20513
macro avg	0.66	0.66	0.66	20513
weighted avg	0.66	0.66	0.66	20513

After binnig and use of standard scaler knn model accuracy improved from 0.64 to 0.66.

- **KNN After binning (using STD Scaler)+tuning:**

```
print('Best parameters for KNN Classifier: ', knn_grid.best_params_, '\n')
```

```
Best parameters for KNN Classifier:  {'metric': 'manhattan', 'n_neighbors': 9}
```

```
knn_classification=KNeighborsClassifier(n_neighbors = 9,metric='manhattan')
knn_model_4 = knn_classification.fit(X_train_std, y_train)
```

```
def get_test_report(model, test_data):
    test_pred = model.predict(test_data)

    return(classification_report(y_test, test_pred))
test_report=get_test_report(knn_model_4,X_test_std)
print(test_report)
```

```
C:\Users\lenovo\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py
havior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this be
he statistic is taken will be eliminated, and the value None will no longer be acce
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

	precision	recall	f1-score	support
0	0.70	0.65	0.67	10413
1	0.66	0.72	0.69	10100
accuracy			0.68	20513
macro avg	0.68	0.68	0.68	20513
weighted avg	0.68	0.68	0.68	20513

After binning , use of standard scaler and model tuning the accuracy and recall of the knn model increased . But still its not satisfactory so we will make another model.

5.5 Logistic Regression Base Model:

The base model of logistic regression is giving accuracy of 0.73 and recall for 0 is 0.79 and recall for 1 is 0.67. We will try to find threshold to get optimize model.

	precision	recall	f1-score	support
0	0.71	0.79	0.75	10413
1	0.76	0.67	0.71	10100
accuracy			0.73	20513
macro avg	0.73	0.73	0.73	20513
eighted avg	0.73	0.73	0.73	20513

- **Threshold with youden's index:**
  - **Youden's index**: Youden's Index is the classification cut-off probability for which the (Sensitivity + Specificity - 1) is maximized.

Youden's Index = max(Sensitivity + Specificity - 1) = max(TPR + TNR - 1) = max(TPR - FPR)

i.e. select the cut-off probability for which the (TPR - FPR) is maximum.

With the help of Youden's index we got the threshold 0.480969, whatever predicted value below 0.48 will be classify as 0 and the value above 0.48 will be classify as 1.

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
```

```
youdens_table = pd.DataFrame({'TPR': tpr,
                              'FPR': fpr,
                              'Threshold': thresholds})
youdens_table['Difference'] = youdens_table.TPR - youdens_table.FPR
youdens_table = youdens_table.sort_values('Difference', ascending = False).reset_index(drop = True)
youdens_table.head(1) #first value will give best value
```

	TPR	FPR	Threshold	Difference
0	0.693366	0.230385	0.480969	0.462981

- **Youden's threshold model:**

The model after taking threshold 0.48 have same accuracy but different recall values as compared to base model. The recall value for 0 is reduced to 0.77 and for 1 increase to 0.69.

```
acc_table = classification_report(y_test, y_pred_youden)

# print the table
print(acc_table)
```

```
              precision    recall  f1-score   support

           0       0.72      0.77      0.74     10413
           1       0.74      0.69      0.72     10100

    accuracy                           0.73     20513
   macro avg       0.73      0.73      0.73     20513
weighted avg       0.73      0.73      0.73     20513
```

- **RFE:**

RFE stands for Recursive Feature Elimination. It is a feature selection technique that is used in machine learning and statistics to select a subset of features from a given dataset. The goal of RFE is to select the most informative features that contribute the most to the prediction of the target variable, while removing the less informative features.

The RFE algorithm works by recursively removing the least important feature and building the model again, until the desired number of features is reached. The feature importance is usually determined by the model's performance or by a feature ranking algorithm such as mutual information or chi-squared.

```
logreg = LogisticRegression()
rfe_model = RFE(estimator = logreg, n_features_to_select = 3)

# fit the RFE model on the train dataset using fit()
rfe_model = rfe_model.fit(X_train, y_train)
feat_index = pd.Series(data = rfe_model.ranking_, index = X_train.columns)
signi_feat_rfe = feat_index[feat_index==1].index
print(signi_feat_rfe)
```

- **Output of RFE:**

After performing RFE we got 3 significant parameters on which we will built the model.

```
Index(['cholesterol', 'smoke', 'active'], dtype='object')
```

- **RFE Model :**

```
print(classification_report(y_test, y_pred_rfe))
```

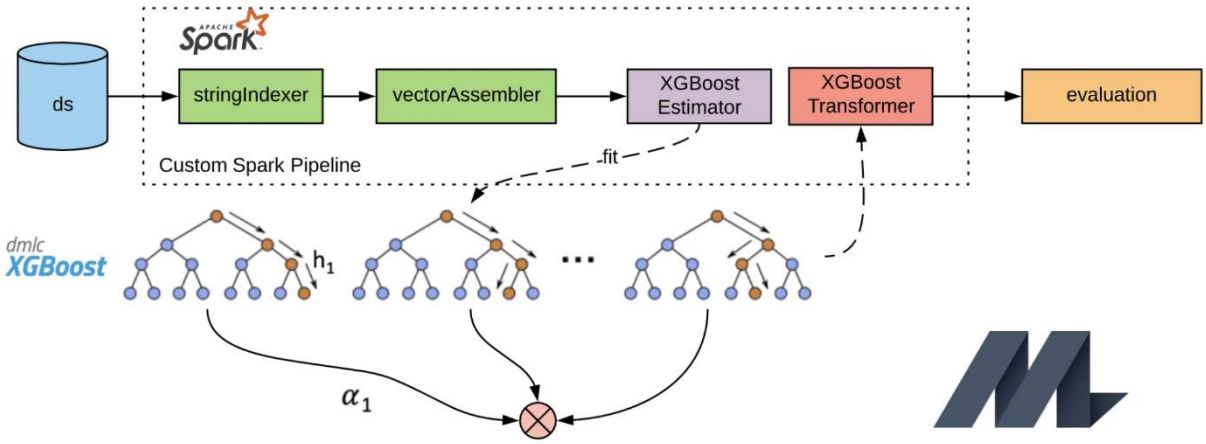
	precision	recall	f1-score	support
0	0.57	0.84	0.68	10413
1	0.67	0.34	0.45	10100
accuracy			0.59	20513
macro avg	0.62	0.59	0.56	20513
weighted avg	0.62	0.59	0.56	20513

After using the significant variables, the accuracy of the model declined to 0.59 from 0.73. And also there is improved in recall value of 0 which is 0.84 but declined in recall value of 1 which is 0.34.

5.6 XGBoost:

XG Boost is an optimized version of gradient boosting that uses a more efficient algorithm and data structure to train the models.

One of the main advantages of XG Boost is its ability to handle large datasets with a large number of features. It also provides a variety of options for tuning the model such as regularization, learning rate, and tree-specific parameters.




```
The metrics for the model XG

Accuracy score: 0.731682347779457

recall score: 0.6927722772277227

precision score: 0.744520110661843

f1 score: 0.7177146373987076

roc auc score: 0.7310975570331449
Classification Report
              precision    recall  f1-score   support

           0       0.72      0.77      0.74     10413
           1       0.74      0.69      0.72     10100

    accuracy                           0.73     20513
   macro avg       0.73      0.73      0.73     20513
weighted avg       0.73      0.73      0.73     20513
```

- **XGboost tuned:**

Following are the best parameters we got after tuning the model:

```
tuning_parameters={"n_estimators": range(1,100),
                   "learning rate":[0.1, 0.2, 0.3, 0.4, 0.5, 0.6],
                   "max_depth": range(3,18),
                   "gamma": [0, 1, 2, 3, 4] }
xgb_model=XGBClassifier()
xgb_grid=GridSearchCV(estimator=xgb_model,param_grid=tuning_parameters,cv=20,scoring='recall')
xgb_grid.fit(X_train,y_train)
```

```
print('Best parameters for XGboost clasifier',xgb_grid.best_params_)
```

```
Best parameters for XGboost clasifier {'gamma': 0, 'learning rate': 0.1, 'max_depth': 3, 'n_estimators': 1}
```

Best parameters for XGboost clasifier {'gamma': 0, 'learning rate': 0.1, 'max_depth': 3, 'n_estimators': 1}

After that we built a model on best parameters which gave us following results:

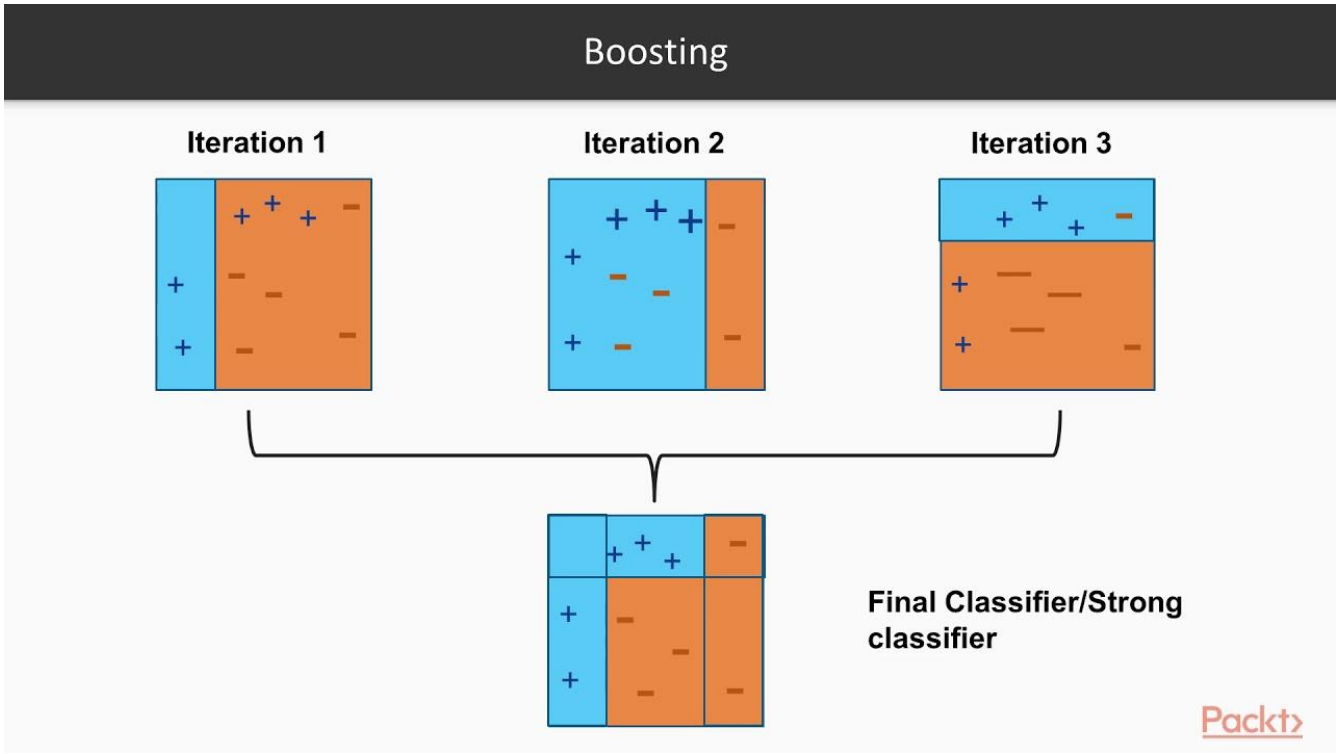
```
test_report = get_test_report(xgb_model)

# print the performance measures
print(test_report)
```

	precision	recall	f1-score	support
0	0.71	0.78	0.74	10413
1	0.75	0.67	0.71	10100
accuracy			0.73	20513
macro avg	0.73	0.73	0.72	20513
weighted avg	0.73	0.73	0.72	20513

The value of accuracy is 0.73 and the value of recall for 0 is 0.78 and for 1 is 0.67.

5.7 AdaBoost:

The model creates several stumps (decision tree with only a single decision node and two leaf nodes) on the train set and predicts the class based on these weak learners (stumps). For the first model, it assigns equal weights to each sample. It assigns the higher weight for the wrongly predicted samples and lower weight for the correctly predicted samples. This method continues till all the observations are correctly classified or the predefined number of stumps is created.



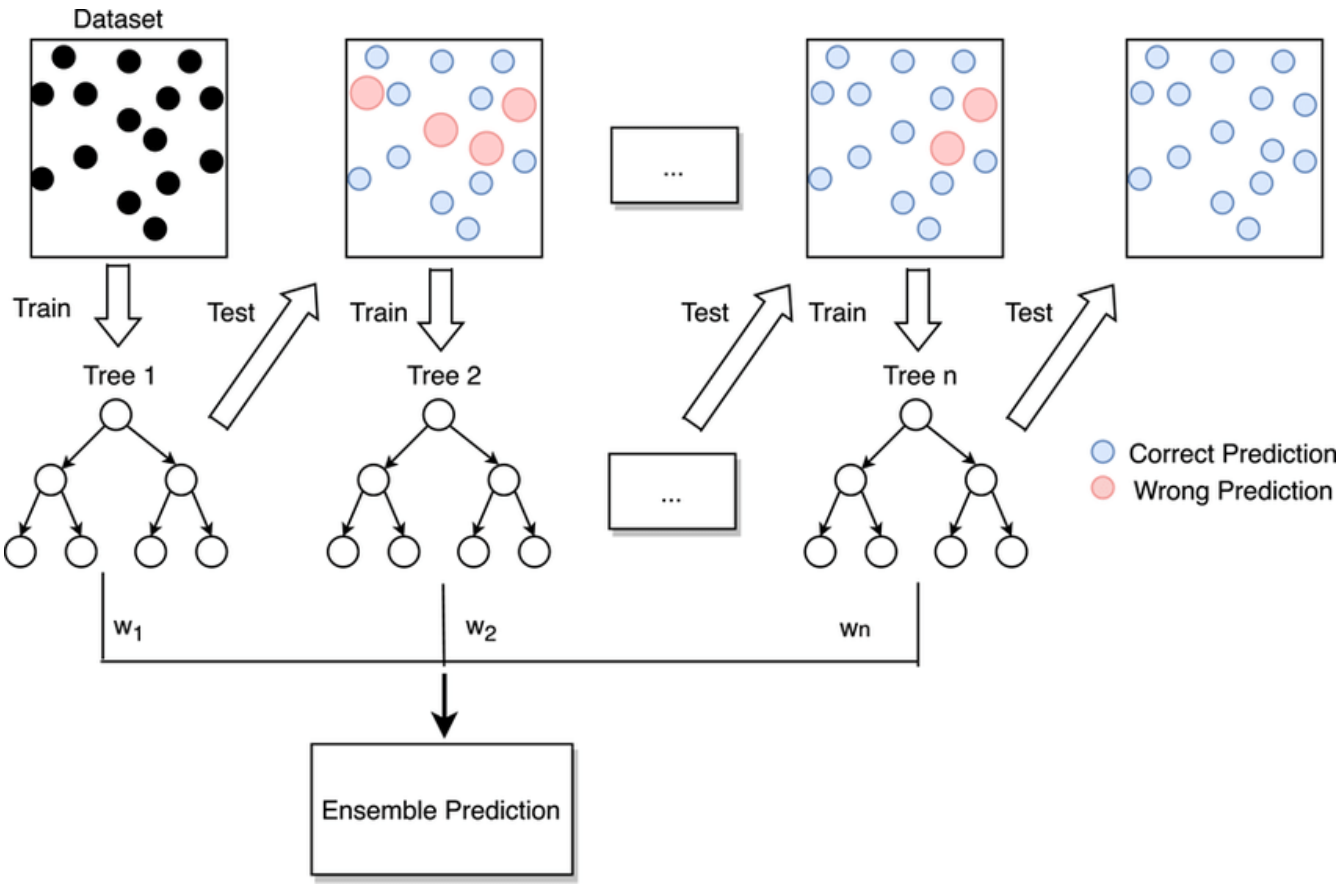
After applying the AdaBoost classifier, we got the following results:

	precision	recall	f1-score	support
0	0.71	0.80	0.75	10413
1	0.76	0.66	0.71	10100
accuracy			0.73	20513
macro avg	0.74	0.73	0.73	20513
weighted avg	0.74	0.73	0.73	20513

The value of accuracy is 0.73 and the value of recall for 0 is 0.80 and for 1 is 0.66

5.8 Gradient Booster:

This method optimizes the differentiable loss function by building the number of weak learners (decision trees) sequentially. It considers the residuals from the previous model and fits the next model to the residuals. The algorithm uses a gradient descent method to minimize the error.



After applying the Gradient boost classifier, we got following results:

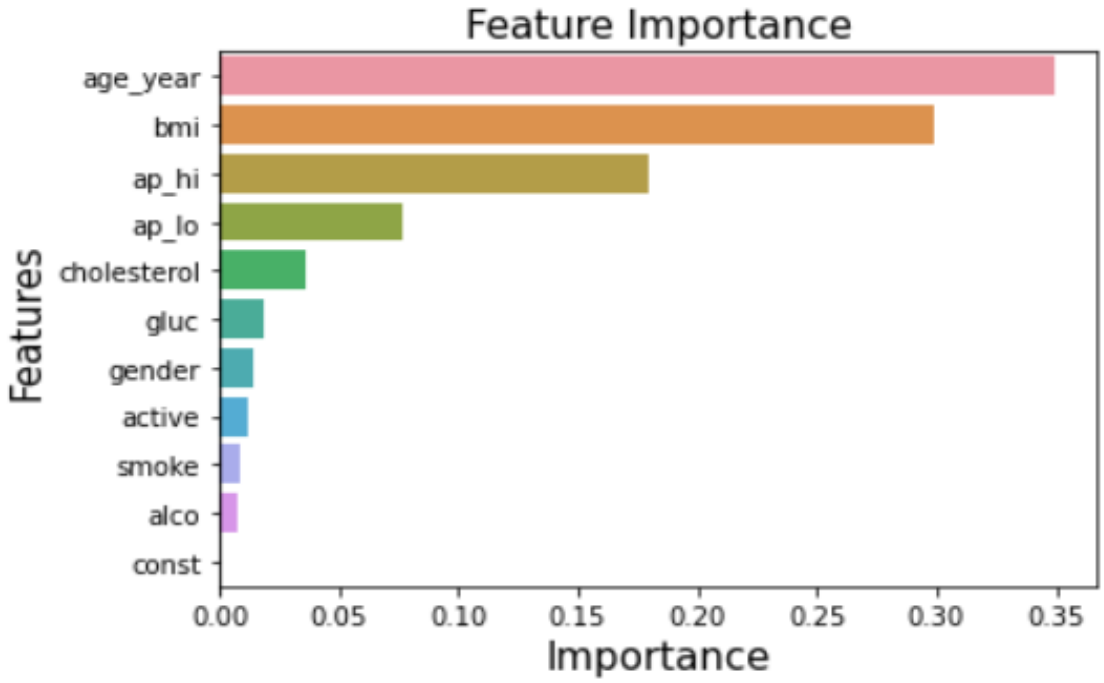
```
print(classification_report(y_test,y_pred_test))
```

	precision	recall	f1-score	support
0	0.71	0.77	0.74	10413
1	0.74	0.68	0.71	10100
accuracy			0.73	20513
macro avg	0.73	0.73	0.73	20513
weighted avg	0.73	0.73	0.73	20513

The value of accuracy is 0.73 and the value of recall for 0 is 0.77 and for 1 is 0.68.

6.Feature Importance

Random forest consists of a number of decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. The measure based on which the (locally) optimal condition is chosen is called impurity. When training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure. This is the feature importance measure exposed in sklearn's Random Forest implementation.



The significant features as per the random forest model are age_year followed by bmi, ap_hi, ap_lo and cholesterol.

7. Comparison of Metrics for all Algorithms :

We have tabulated all the metrics from different algorithms and a dataframe out of it.

	model name	accuracy	recall	precision	roc auc	f1 score
9	Random Forest tuned	0.720031	0.696931	0.724102	0.719684	0.710257
7	Gradient Boost	0.735582	0.694752	0.74984	0.734969	0.721246
8	xgboost	0.731195	0.690297	0.745031	0.73058	0.71662
5	Random Forest	0.706625	0.69	0.707082	0.706375	0.698437
0	Logreg	0.726466	0.689307	0.737891	0.725908	0.712772
1	KNN	0.698971	0.68495	0.698012	0.698761	0.69142
10	XGB Tuned	0.725881	0.668119	0.748198	0.725013	0.705895
6	Ada boost	0.731634	0.65703	0.76478	0.730512	0.706822
4	Decision Tree Tuned	0.726027	0.655743	0.755533	0.724971	0.70211
2	Naive Beyes	0.715449	0.652772	0.738877	0.714507	0.693161
3	Decision Tree	0.638912	0.634158	0.633093	0.63884	0.633625

8. Score Card :

We have collated all the results from the various algorithms and made two score cards One with accuracy as the metric and the other with recall as the metric.

8.1 Accuracy:

In Machine Learning, the Accuracy score (or just Accuracy) is a Classification metric featuring a fraction of the predictions that a model got right. The metric is prevalent as it is easy to calculate and interpret. Also, it measures the model's performance with a single value.

$$Accuracy = \frac{TrueNegatives + TruePositive}{TruePositive + FalsePositive + TrueNegative + FalseNegative}$$

---

	model name	accuracy
7	Gradient Boost	0.735582
6	Ada boost	0.731634
8	xgboost	0.731195
4	Decision Tree Tuned	0.729635
0	Logreg	0.726466
10	XGB Tuned	0.725881
9	Random Forest tuned	0.721689
2	Naive Beyes	0.715449
5	Random Forest	0.705455
1	KNN	0.698971
3	Decision Tree	0.638083

8.2 Recall :

The recall is calculated as the ratio between the number of *Positive* samples correctly classified as *Positive* to the total number of *Positive* samples. The recall measures the model's ability to detect *Positive* samples. The higher the recall, the more positive samples detected.

$$Recall = \frac{True_{positive}}{True_{positive} + False_{negative}}$$

Recall is very important in domains such as healthcare (e.g., identifying cardivovascular diseasses), where **you really want to minimize the chance of missing positive cases** (predicting false negatives). These are typically cases where missing a positive case has a much bigger cost than wrongly classifying something as positive.

	model name	recall
9	Random Forest tuned	0.697723
7	Gradient Boost	0.694752
8	xgboost	0.690297
0	Logreg	0.689307
5	Random Forest	0.688317
1	KNN	0.68495
10	XGB Tuned	0.668119
6	Ada boost	0.65703
2	Naive Beyes	0.652772
3	Decision Tree	0.63604
4	Decision Tree Tuned	0.629307

9. Practical Importance of model :

9.1 From patient's point of view: Early detection of cardiovascular disease will help patients to seek the proper help as early as possible. So that they can try to improve their lifestyle and help themselves to come over it.

9.2 From Hospital's point of view: The hospitals can track their patients behaviour and will suggest the proper measures to take as soon as possible. Also, they can suggest some precautionary measures to patients who have bad lifestyle.

9.3 From company's point of view: The company can identify healthy workforce by considering the behaviour pattern of the worker.

9.4 From Insurance provider's point of view: The insurance company can track the behaviour pattern of their customers and as per their health condition they can charge health premium. Also, they can make aware their customers about upcoming risk of disease which will be beneficial for both customer and insurance provider.

9.5 From Societal and Country's perspective: Cardiovascular diseases (CVDs), the world's leading cause of death, claim as many as 17.9 million lives each year, globally. According to a study by the Indian Council of Medical Research and Registrar General of India, **India accounts for approximately 60 per cent of the world's heart disease burden**.

So, with early detection of the cardiovascular diseases, we can also focus on vulnerable sections which are getting affected by them. And government can motivate them to adopt healthy lifestyle as well as by providing them proper guidance on timely basis.

10. Conclusion:

- It was a great learning experience working on a Healthcare dataset.
- Our dataset consist of categorical and numerical features.
- Weight and height were important parameters but it cannot be used for prediction as we made a new feature named bmi with the help of of weight and height columns and use it in the model.
- When visualized age in groups, it is found that clients with age more than 30 and less than 60 are more susceptible to the cardiovascular diseases.
- The dataset don't have the problem of imbalance and it is nearly balance on target column.
- Different machine learning models are trained and tested on the dataset.
- Different models are summarized in table above.
- If accuracy was taken as the metric, Gradient boosting model and Adaboost are showing best performances.
- If recall was taken as the metric, Random forest with tuned parameters and Gradient boost model are showing Best performances
- The Tuning hyperparameters helped to optimize the recall score on both 0 and 1.
- Based on the Subject Matter Expert we can decide on which metric to be used Accuracy or Recall and try to improve that single metric to improve the model performance in further studies.