Heart Disease Predictor

Group Members

NAME: Subhodip Roy
REGISTRATION NO: 201080100110153
COLLEGE: Asansol Engineering College

NAME: Drippta Dutta
REGISTRATION NO: 201080100110135
COLLEGE: Asansol Engineering College

NAME: Arpita Maji
REGISTRATION NO: 201080100110141
COLLEGE: Asansol Engineering College

NAME: Moumita Maji
REGISTRATION NO: 201080100110143
COLLEGE: Asansol Engineering College

NAME: Snehashis Roy
REGISTRATION NO: 201080100110163
COLLEGE: Asansol Engineering College

Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, Prof. Arnab Chakraborty for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark. I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Subhodip Roy

Drippta Dutta

Arpita Maji

Moumita Maji

Snehashis Roy

Project Objective

In this project we have a shortened 'Heart Disease Prediction' Dataset from Framingham heart disease. In this dataset, the target attribute is the Ten year CHD which is in binary. So, in this project we need to do binary classification based on the attributes present in our dataset and predict whether a loan application should be approved or not.

Our objective in this project is to study the given dataset of 'Heart Disease Prediction'. We might need to pre-process the given dataset if we need to. Then, we would train 4 models viz. 'KNN classifier model', 'Logistic Regression classifier model', and 'Decision Tree classifier model'. After training the aforementioned models, we will need to find out the score, classification report, plot the Receiver Operating Characteristic graph and find out the Area Under Curve (AUC) for each of the models trained. Our next step would be to use the trained models to predict the outcomes using the given test dataset and compare the outcome of each model. We would then choose the best model based on the accuracy score and classification report.

Our methodology for solving the problems in the given project is described below:

- Load the required dataset.
- Study the dataset.
- Describe the dataset.
- Visualise the dataset.
- Find out if the dataset needs to be pre-processed.
    - ✧ It will be determined on the basis of whether the dataset has null values or outliers or any such discrepancy that might affect the output of the models to be trained.
- If the dataset is required to be pre-processed, take the necessary steps to pre-process the data.
- Find out the principal attributes for training.
- Split the given dataset for training and testing purpose.
- Fit the previously split train data in the aforementioned 4 models.
- Calculate the accuracy of the 4 models and find out the classification reports.
- Plot the necessary graphs.
- Use each trained model to predict the outcomes of the given test dataset.
- Choose the best model among the 4 trained models bases on the accuracy and classification reports.

Project Scope

Today, heart failure diseases affect more people worldwide than other autoimmune conditions. Cardiovascular Diseases (CVDs) affect the heart and obstruct blood flow through the blood vessels. Chronic ailments in CVD include heart disease (heart attack), cerebrovascular diseases (strokes), congestive heart failure, and many more pathologies. Worldwide, CVDs kill around 17 million a year, and death rates due to heart diseases have increased after the COVID-19 pandemic.

Several risk factors for manual heart disease prediction may include inactivity in a physical form, unhealthy eating habits, or even the consumption of alcohol. Preexisting conditions, age, chest pain level, blood test results, and several such factors can be ensembled together computationally for heart disease prediction. With such well-defined parameters and the rise of data science, a data-driven approach can surely help in heart disease prediction using machine learning technologies. Early identification of heart disease of improved diagnosis and high-risk individuals using a prediction model can be recommended for a fatality rate reduction, and decision-making is improved for further treatment and prevention.

Data Description

Source of the data: Framingham Heart Dataset by Github.

Data Description: The given train dataset has 4240 rows and 16 columns.

Columns	Attribute Name	Type	Description	Target Attribute
Male	male	Categorical	Gender of the patient (0/1)	No
Age	age	Non-Categorical	Age of the patient	No
Education	Education	Categorical	Education status of the patient (1/2/3/4)	No
Current Smoker	currentSmoker	Non-Categorical	The number of cigarettes that the person smoked on average in one day	No
BP Meds	BPMeds	Categorical	Whether or not the patient as on blood pressure medication (0/1)	No
Prevalent Stroke	prevalentStroke	Categorical	Whether or not the patient had previously had a stroke(0/1)	No
Prevalent Hypertensive	prevalentHyp	Categorical	Whether or not the patient was hypertensive(0/1)	No
Diabetes	diabetes	Categorical	Whether or not the patient had diabetes(0/1)	No
Total Cholestrol	totChol	Non-Categorical	Total cholesterol level	No
Systolic Blood Pressure	sysBP	Non-Categorical	Systolic blood pressure	No
Diastolic Blood Pressure	diaBP	Non-Categorical	Diastolic blood pressure	No
Body Mass Index	BMI	Non-Categorical	Body Mass Index	No
Heart Rate	heartRate	Non-Categorical	Heart Rate	No
Glucose	glucose	Non-Categorical	Glucose level	No
TenYearCHD	TenYearCHD	Non-Categorical	10 year risk of coronary heart disease CHD (binary: “1”, means “Yes”, “0” means “No”)	Yes

Table 1: Data Description

The following table shows the 16 number statistics of the given dataset:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
count	4240.000000	4240.000000	4135.000000	4240.000000	4211.000000	4187.000000	4240.000000	4240.000000	4240.000000	4190.000000	4240.000000	4240.000000	4221.000000	4239.000000	3852.000000	4240.000000
mean	0.429245	49.580189	1.979444	0.494104	9.005937	0.029615	0.005896	0.310613	0.025708	236.699523	132.354599	82.897759	25.800801	75.878981	81.963655	0.151887
std	0.495027	8.572942	1.019791	0.500024	11.922462	0.169544	0.076569	0.462799	0.158280	44.591284	22.033300	11.910394	4.079840	12.025348	23.954335	0.358953
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	107.000000	83.500000	48.000000	15.540000	44.000000	40.000000	0.000000
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	206.000000	117.000000	75.000000	23.070000	68.000000	71.000000	0.000000
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	234.000000	128.000000	82.000000	25.400000	75.000000	78.000000	0.000000
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.000000	0.000000	1.000000	0.000000	263.000000	144.000000	90.000000	28.040000	83.000000	87.000000	0.000000
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.000000	1.000000	1.000000	1.000000	696.000000	295.000000	142.500000	56.800000	143.000000	394.000000	1.000000

Table 2: 8 number statistics of the given dataset

Now we will pre-process the data. The methodology followed is given below:

Checking for null values.

- If null values are present, we will fill them or drop the row containing the null value based on the dataset.

Checking for outliers.

- If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

We searched for null values in our dataset and formed the following table:

Column Name	Count of Null Value
Male	0
Age	0
Education	105
Current Smoker	0
Cigs Per Day	29
BP Meds	53
Prevalent Stroke	0
Prevalent Hyp	0
Diabetes	0
Tot Chol	50
Sys BP	0
Dia BP	0
BMI	19
Heart Rate	1
Glucose	388
TenYearCHD	0

Table 3: Count of Null Values

To visualize the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:

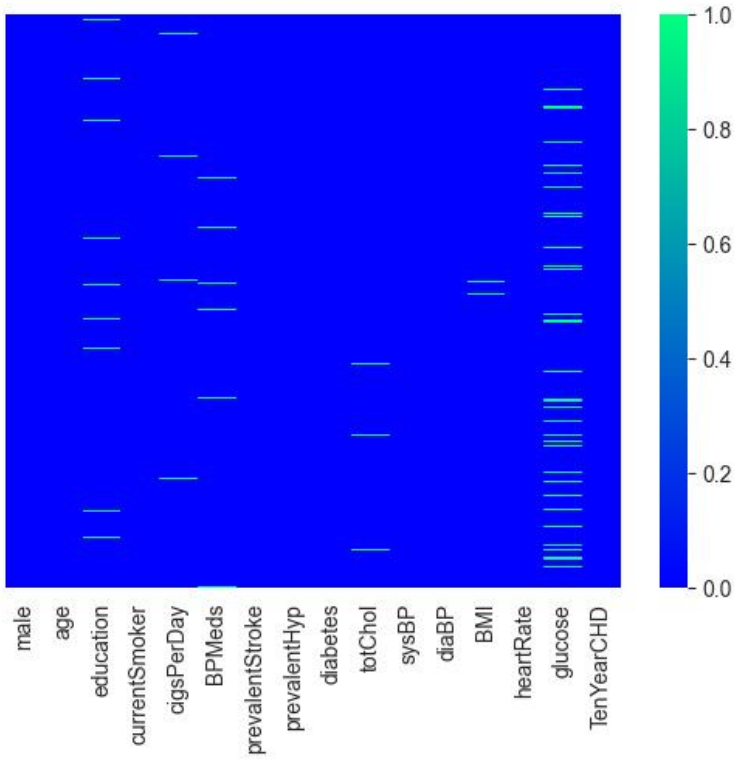


Fig 1: Heatmap of the given Data

The heatmap shows the dataset has null values

To remove the null values , we had the following methodology:

- Filling Null values with mode value. This was applied in columns “cigsPerDay” , “BPMeds” , “totChol” , “BMI” , “glucose” .
- Dropping the columns. This was applied in columns “education” , “currentSmoker” , “heartRate” .

After removing the null values, the following heatmap was obtained:

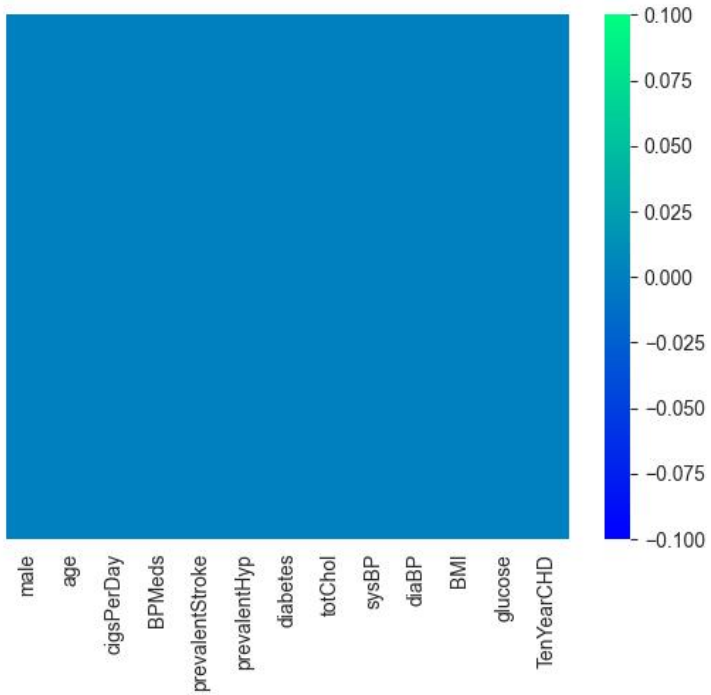


Fig 2: Heatmap implying absence of Null Values

Now we have successfully handled Null values. We dropped some of the columns with null values as our dataset has large number of entries (4240) , and the dropped columns were not so relatable with the target column (TenYearCHD).

So, we are moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. 'TenYearCHD' column in the dataset.

The following table gives the correlation value of each attribute with our target attribute i.e. 'TenYearCHD' :

Columns	**Correlation Value**
Male	0.0883735724112584
Age	0.22540774458750013
Education	-0.05424846659632095
Current Smoker	0.01944849847819047
Cigs Per Day	0.057755206928342934
BP Meds	0.0875194502387338
Prevalent Stroke	0.06182262818175444
Prevalent Hyp	0.17745756074899632
Diabetes	0.0973442355696611
Tot Chol	0.08236854386563609
Sys BP	0.21637382892923476
Dia BP	0.1451115926054232
BMI	0.07530032244325817
Heart Rate	0.02290660760589974
Glucose	0.12559035521913023

Table 4: Correlation Values with Target Attribute

Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

Most common causes of outliers on a data set:

• Data entry errors (human errors)

• Measurement errors (instrument errors)

• Experimental errors (data extraction or experiment planning/executing errors)

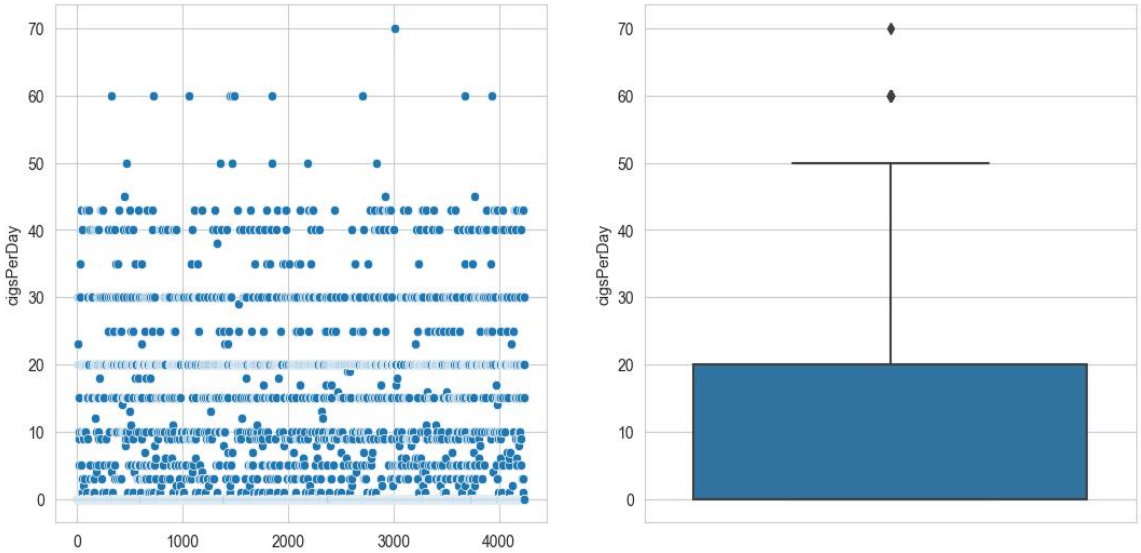
• Intentional (dummy outliers made to test detection methods)

• Data processing errors (data manipulation or data set unintended mutations)

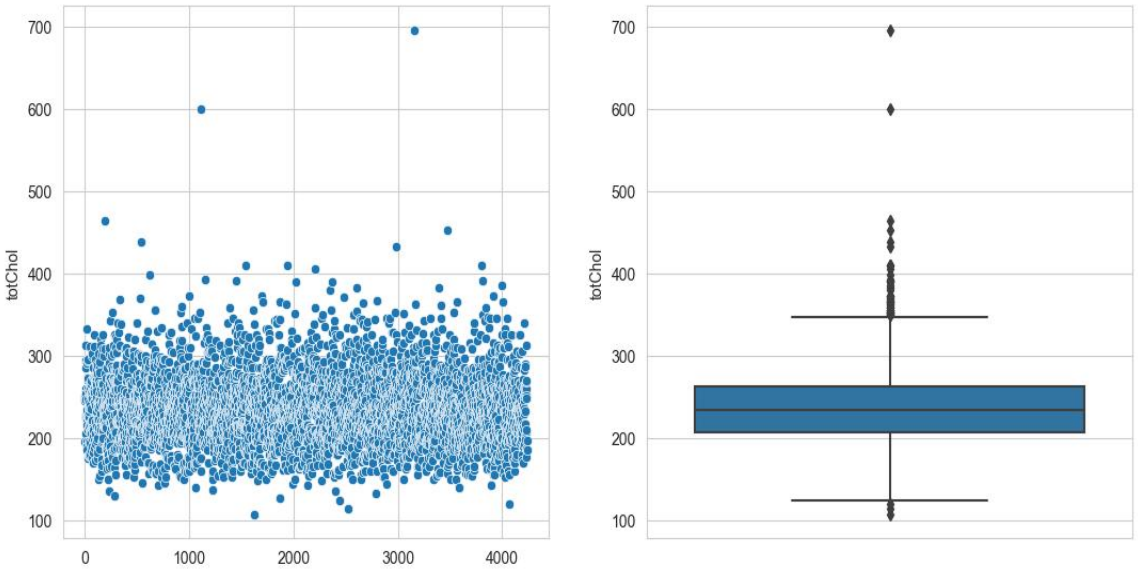
• Sampling errors (extracting or mixing data from wrong or various sources)

• Natural (not an error, novelties in data)

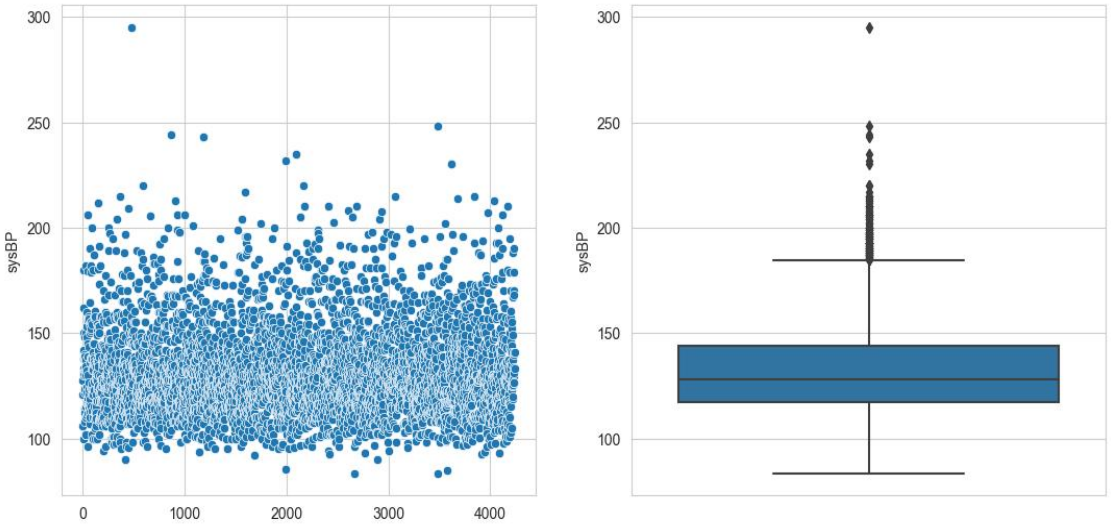
We plot distribution graph and boxplot to visualise the outliers. The plots are given below



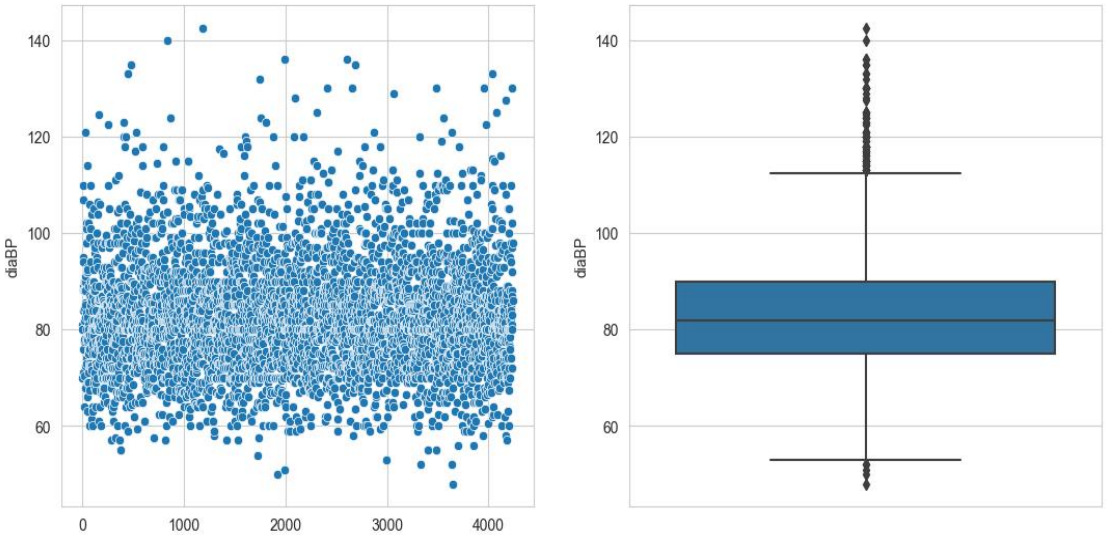
Scatter plot and boxplot of Cigs per day showing possible outliers



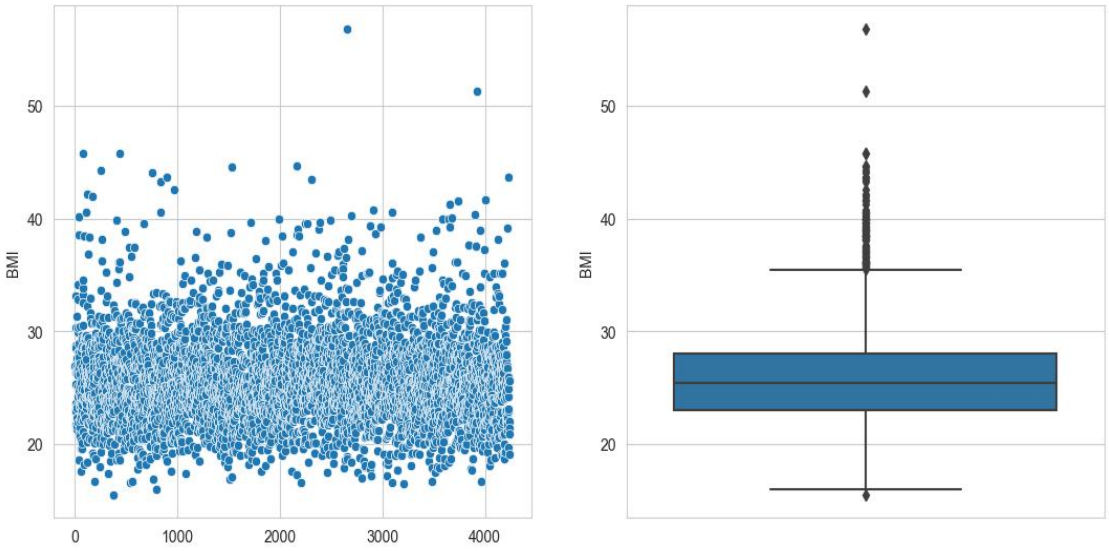
Scatter plot and boxplot of Tot Chol showing possible outliers



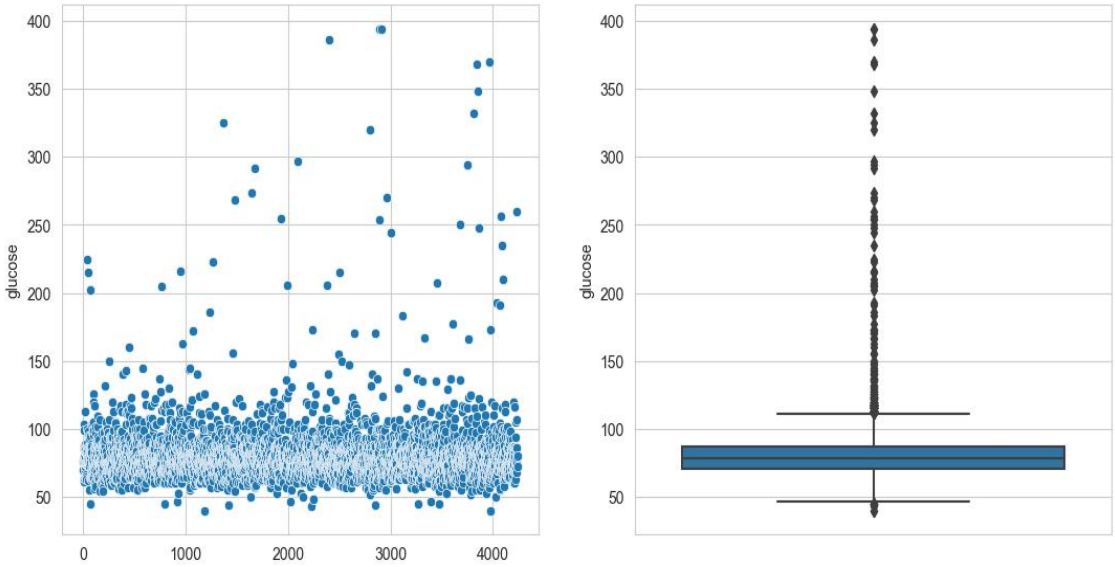
Scatter plot and boxplot of Sys BP showing possible outliers



Scatter plot and boxplot of Día BP showing possible outliers



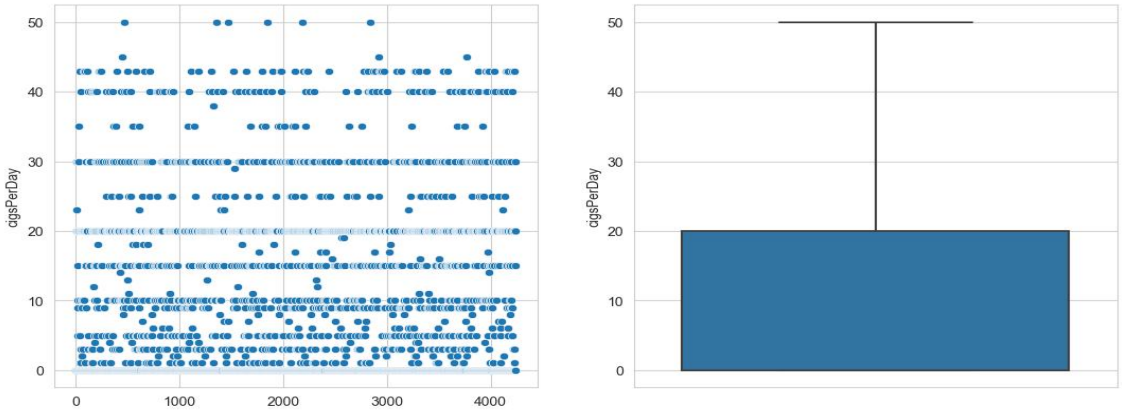
Scatter plot and boxplot of BMI showing possible outliers



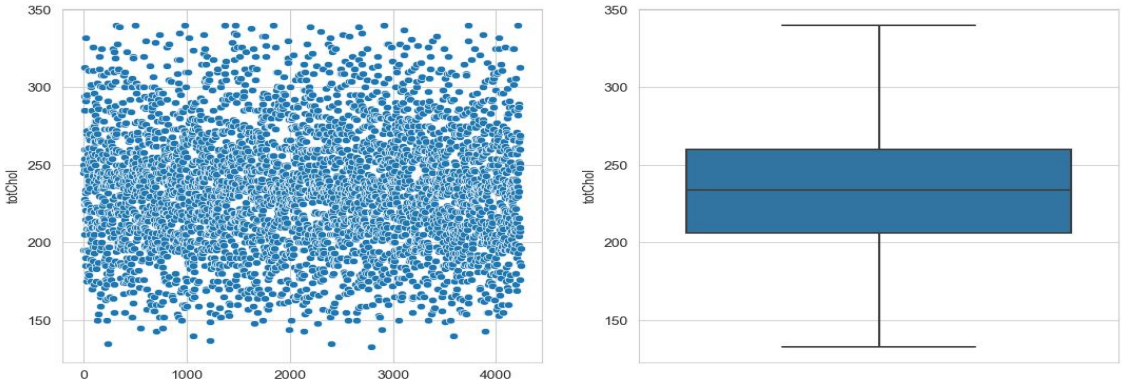
Scatter plot and boxplot of Glucose showing possible outliers

From the distribution plots and boxplots, we can see that there are possible outliers in the columns 'Cigs per day', 'Tot chol',’Sys BP’,’Dia BP’, ‘BMI’ and 'Glucose'. We have not checked for outliers. ‘Male’, ‘Age’, ‘BP medes’,’Current smoker’, ’Prevalent Hyp’, ‘Prevalent Stroke’, ’Diabetes’, ’Ten-year CHD’ as they are columns having values in binaries and also initially. We have not checked for outliers in. We have not checked for outliers in the column 'Education’, ‘Current Smoker’, ‘Heart Rate’ because there corelation is very less So, we drop these 3 columns. Now, we had to handle the outliers. We handled the outlier’s statistics method. We used Median Absolute Deviation or MAD to handle the outliers. In this method, data outside the range 3 * mad is excluded and replaced with median of the data.

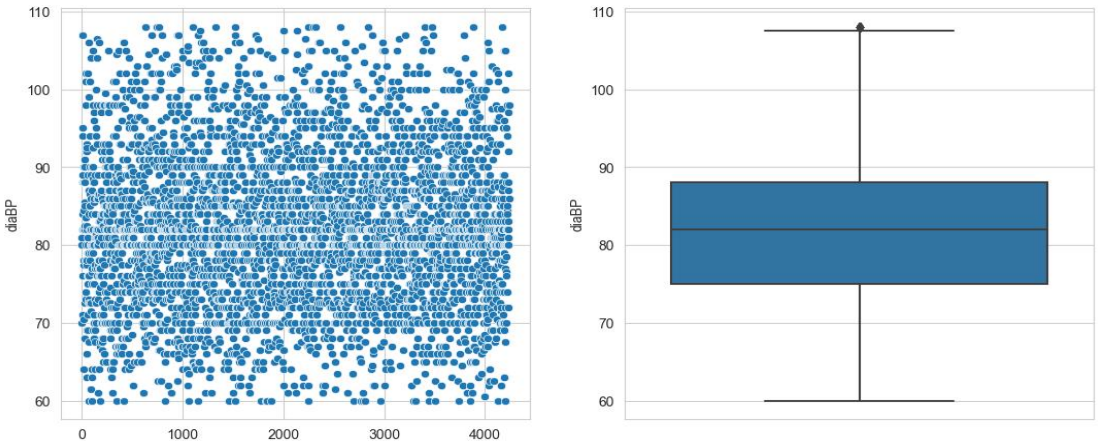
After handling the data, we replot the same distribution and boxplot but this time using the modified data. We obtain the following plots:



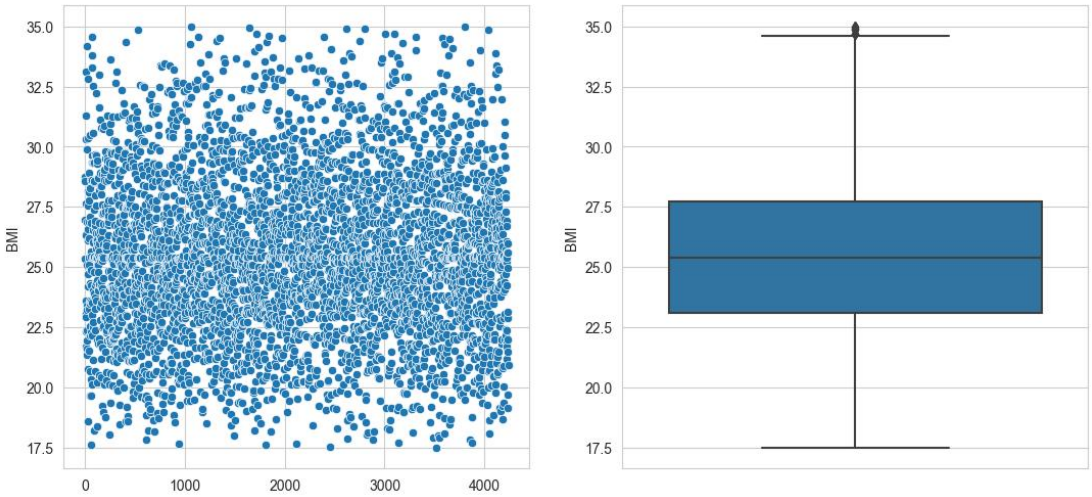
Scatter plot and boxplot of CigsPerDay after removing outliers



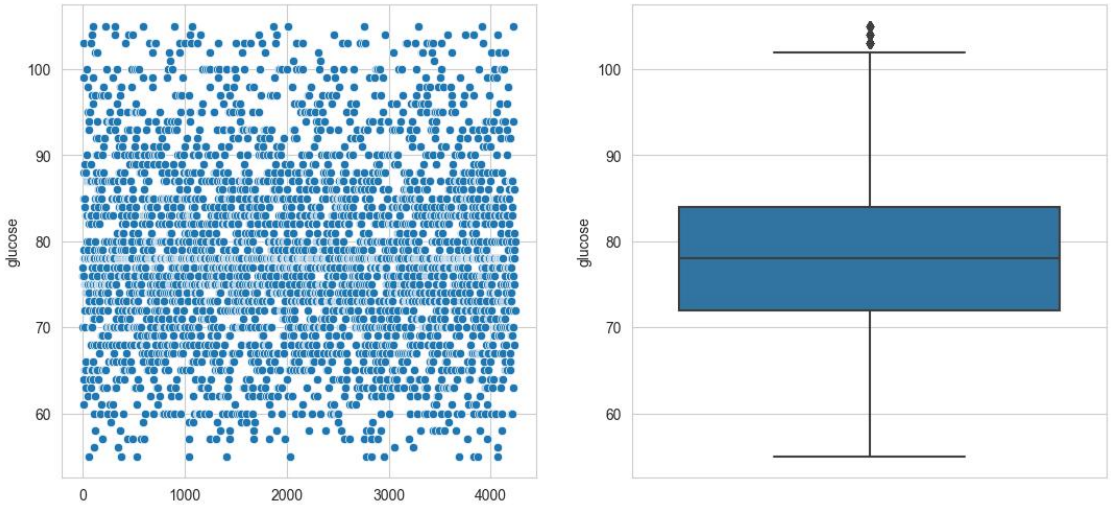
Scatter plot and boxplot of Tot chol after removing outliers



Scatter plot and boxplot of Día BP after removing outliers

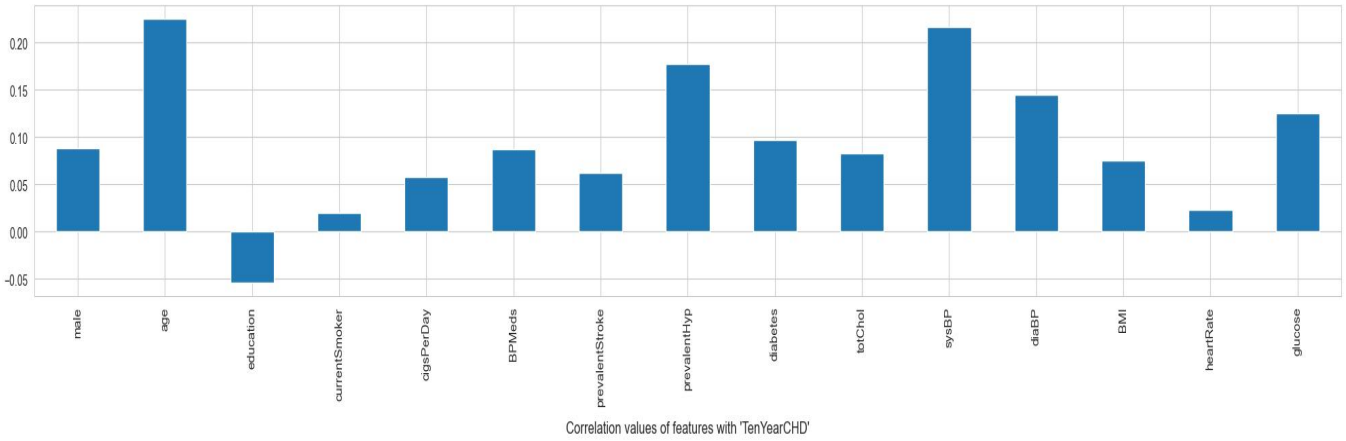


Scatter plot and boxplot of BMI after removing outlier

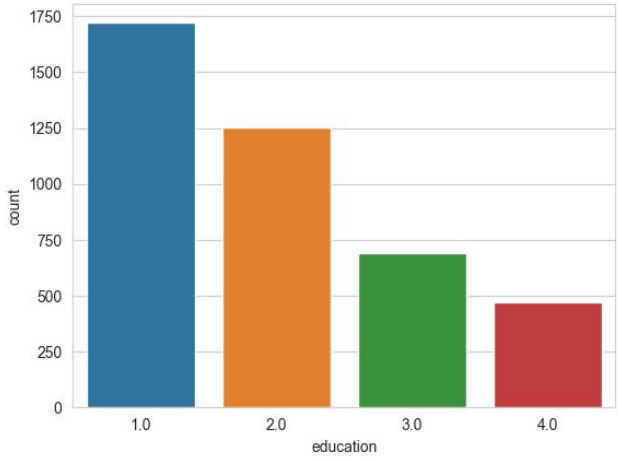


Scatter plot and boxplot of Glucose removing possible outliers

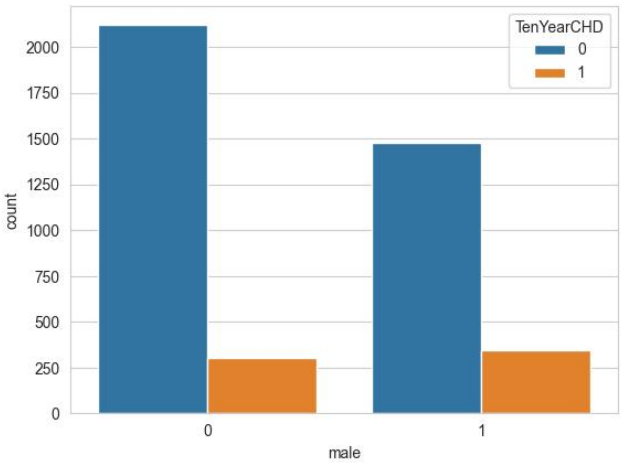
From the distribution plots and boxplots shown above, we can see that a lot of outliers have been handled. We will now visualise other attributes with respect to our target column i.e. 'TenYearCHD'. We have used bar plot , boxplot , scatter plot , histogram to visualise other attributes against different columns.



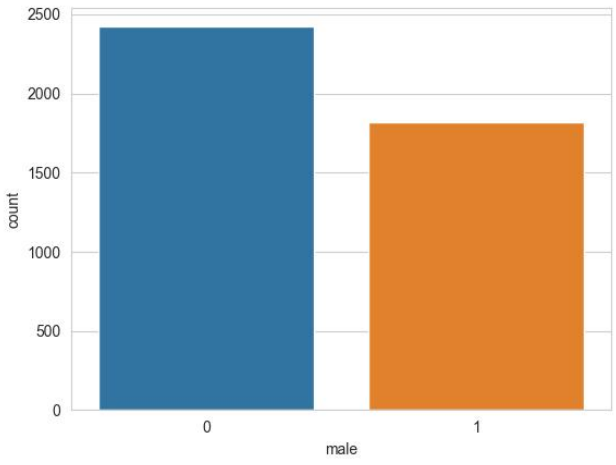
Correlation values of features with 'TenYearCHD'



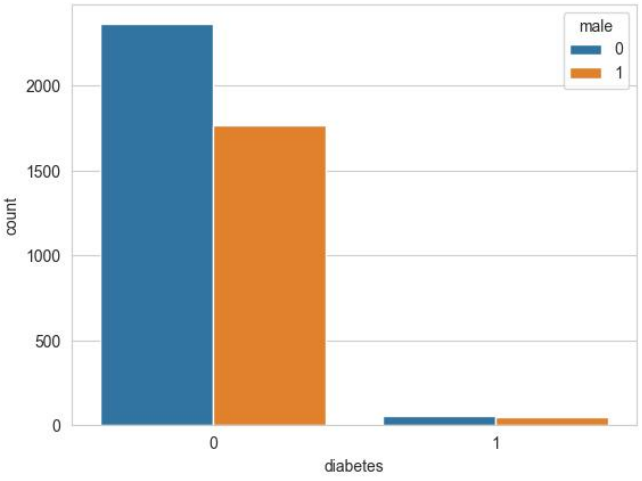
Count of types of education



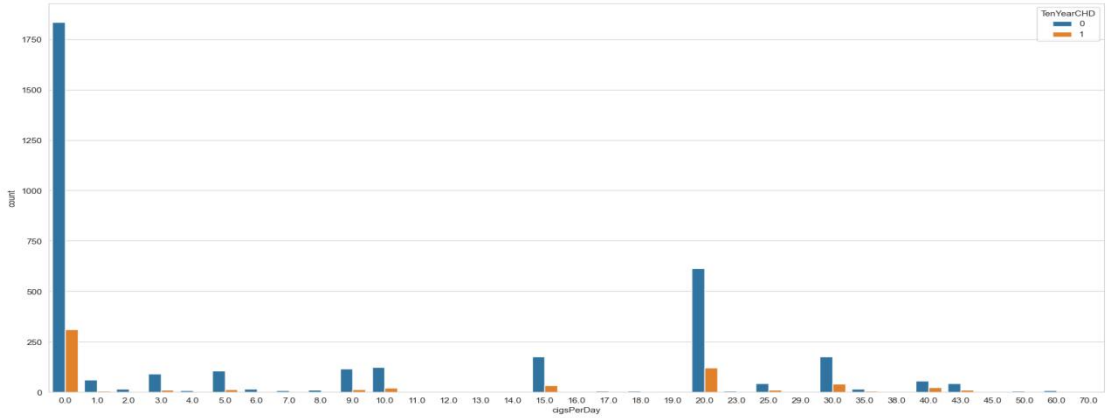
Gender vs TenYearCHD



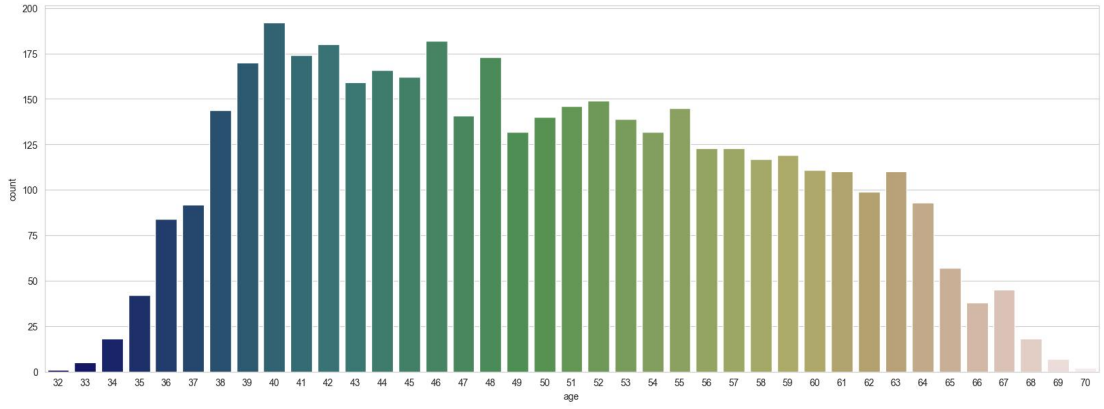
Male and female Count



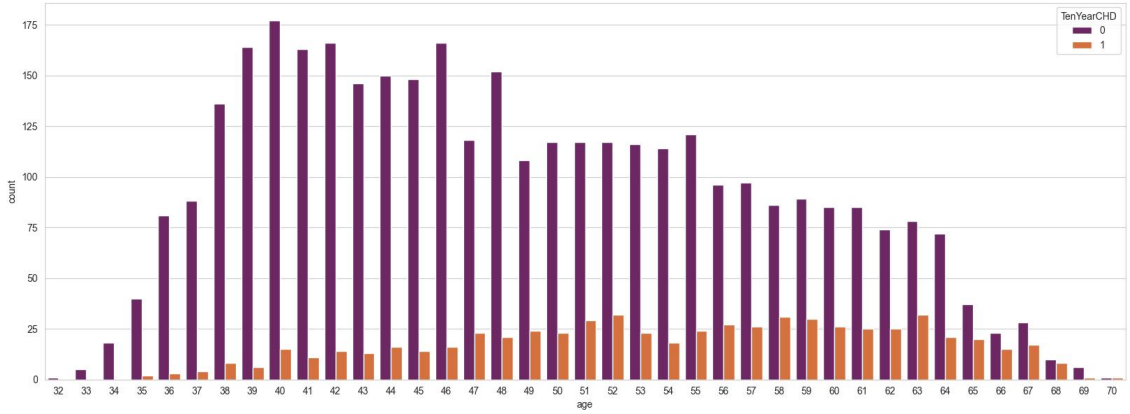
No of people having diabetes



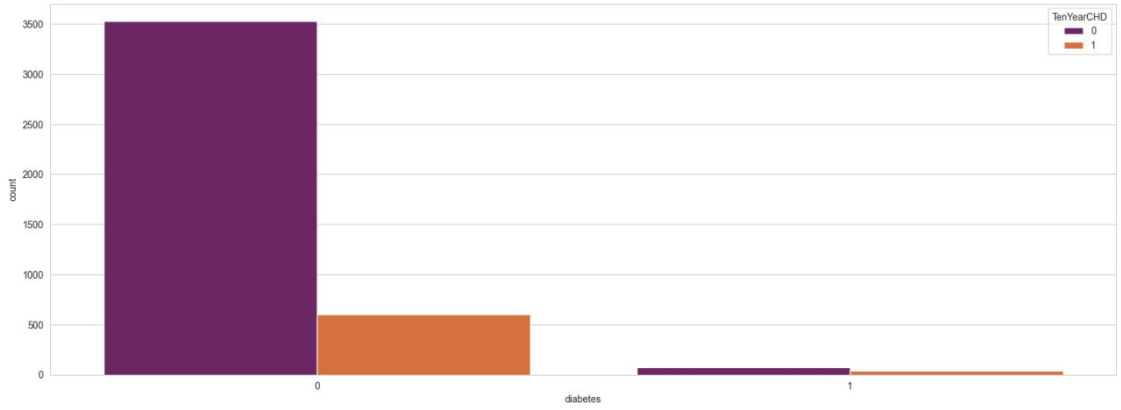
CigsPerDay vs. TenYearCHD



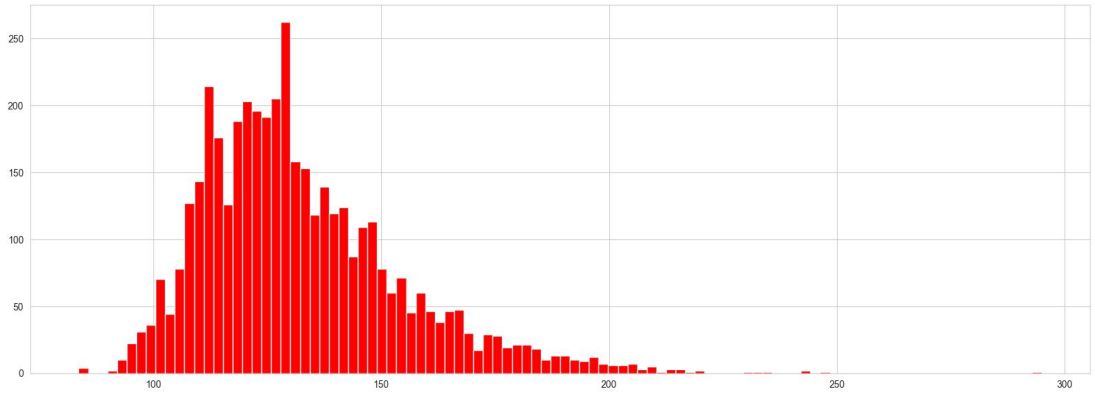
People count by age



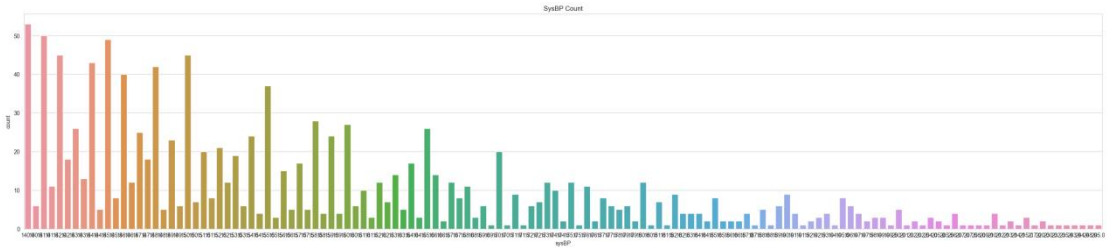
Age vs. TenYearCHD

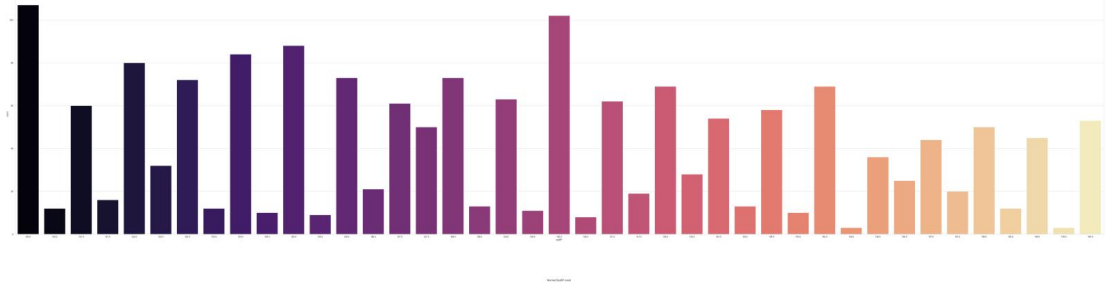


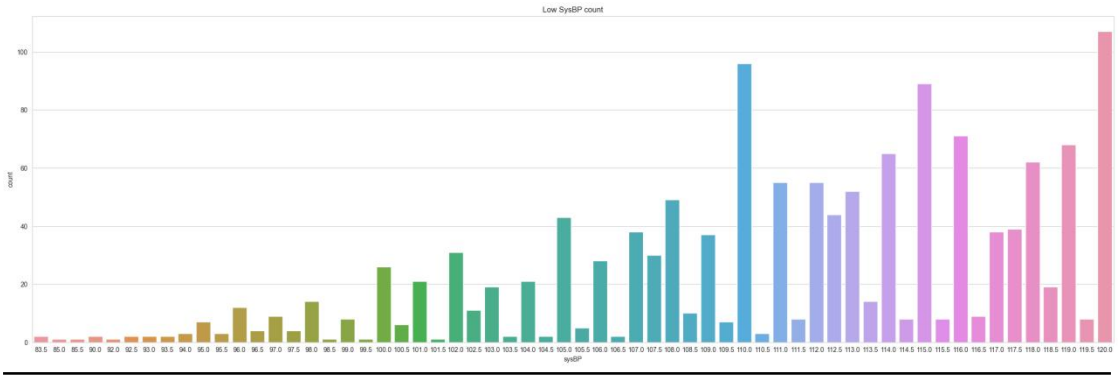
Diabetes vs TenYearCHD

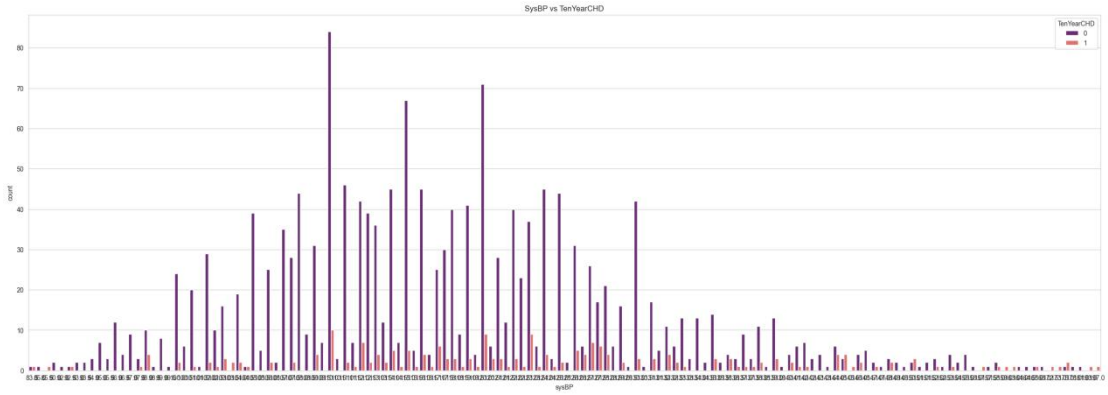


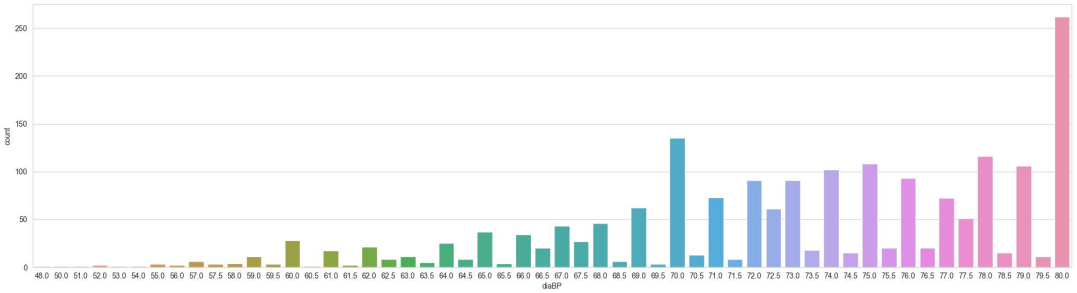
SysBP Histogram

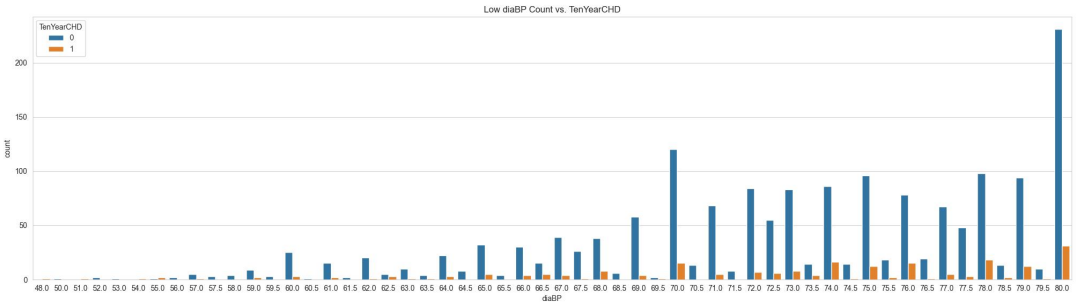


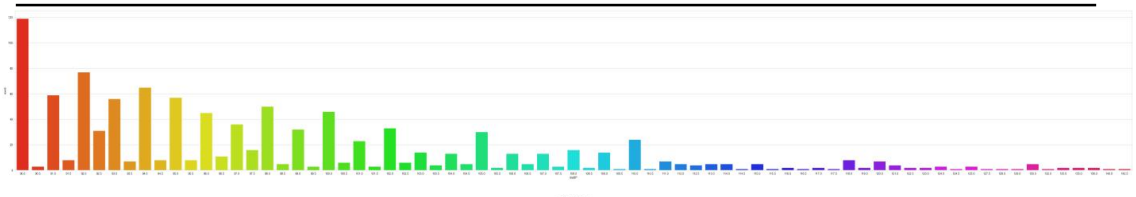


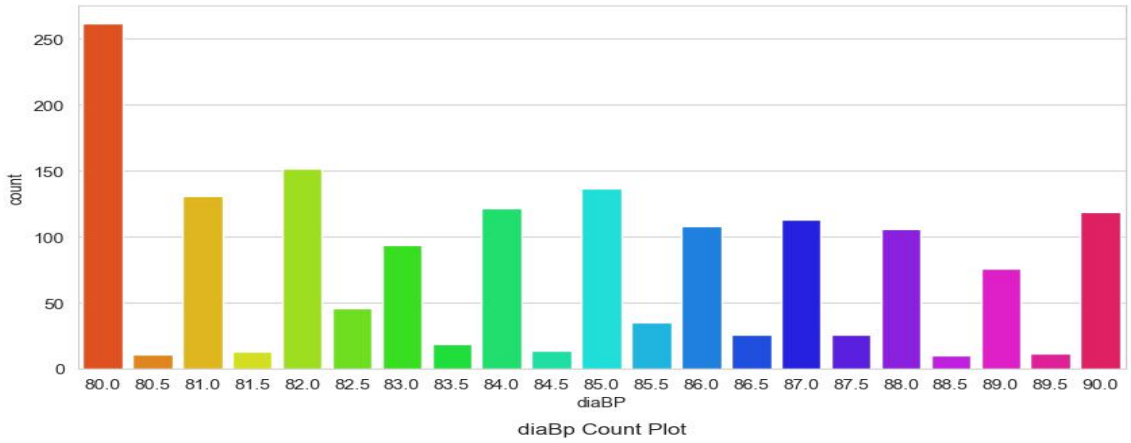

Low SysBP count
count
sysBP


SysBP vs TenYearCHD
TenYearCHD
0
1
count
sysBP

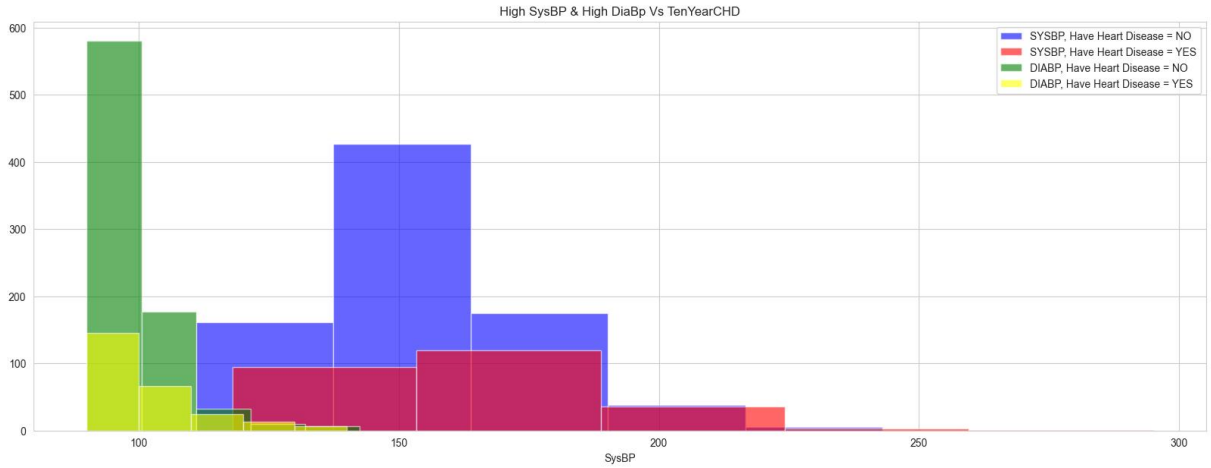

count
daBP

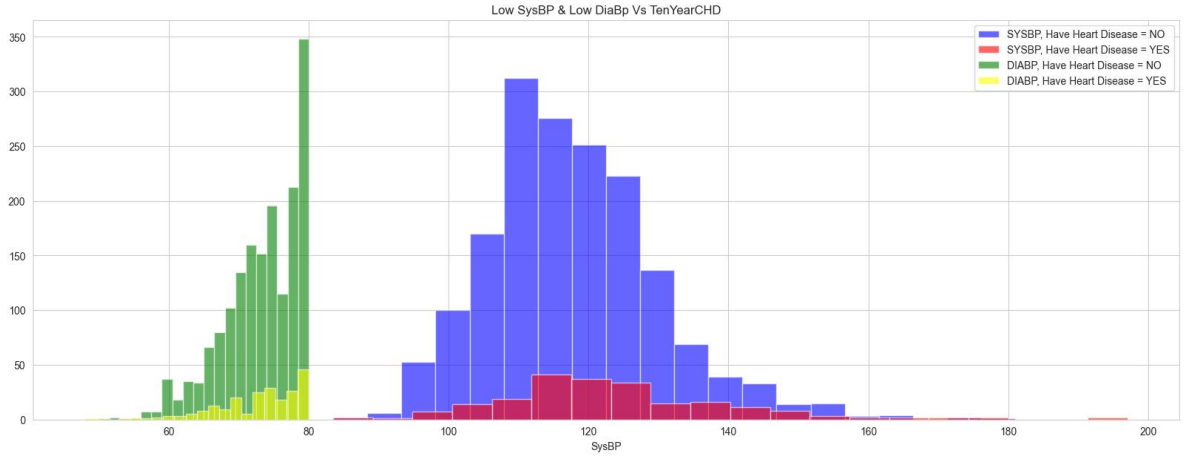

Low diaBP Count vs. TenYearCHD
TenYearCHD
0
1
count
daBP

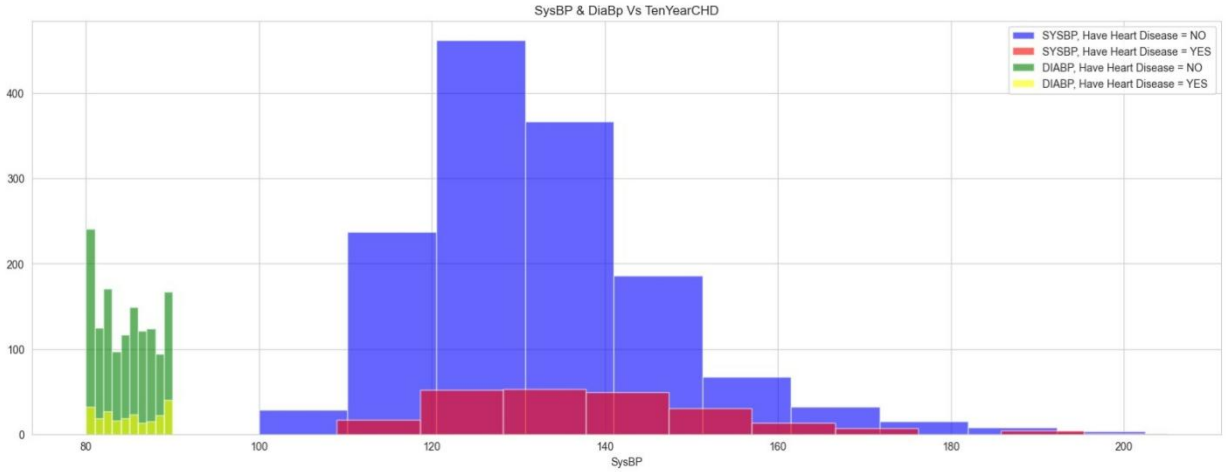


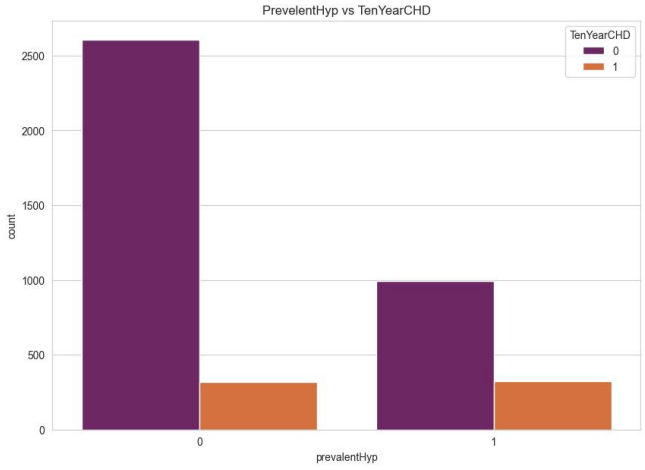


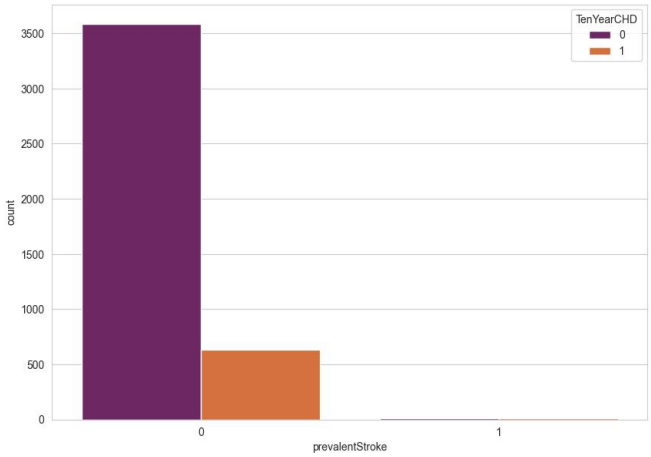
diaBp Count Plot











PreveletStroke vs TenYearCHD

Model Building

Splitting data for training and testing purpose

We split the given train dataset into two parts for training and testing purpose. The split ratio we used is 0.30 which indicates we used 70% data for training purpose and 30% data for testing purpose. We will be using the same split ratio for all the models trained.

Now we will be training our required models. Our project goal requires us to train specific 4 classifier models viz.

I. Regression (As the target attribute is binary, we will be using Logistic Regression Classifier)

II. Random Forest Classifier

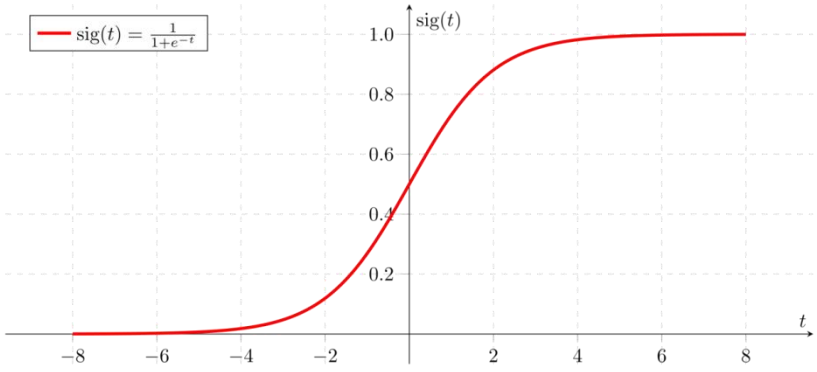
III. Decision Tree Classifier

IV. KNN Classifier

We will be using the final dataset obtained after pre-processing the given train dataset to train our required models.

Logistic Regression

Logistic regression is a type of regression analysis.Regression analysis is a type of predictive modelling technique which is used to find the relationship between a dependent variable (usually known as the "Y" variable) and either one independent variable (the "X" variable) or a series of independent variables. When two or more independent variables are used to predict or explain the outcome of the dependent variable, this is known as multiple regression.Logistic Regression models the data using sigmoid function.



Based on the number of categories, Logistic Regression can be classified as:

I. **Binomial :** Target variable can have only 2 possible types: "0" or "1" which may represent 'win" vs "loss" , "pass" vs "fail" , "dead" vs "alive" ,etc.

II. **Multinomial :** Target variable can have 3 or more possible types which not ordered (i.e. types have no quantitive significance) like "disease A" vs "disease B" vs "disease C".

III. **Ordinal :** It deals with target variable with ordered categories. For example, a test score can be categorized as: "very poor" , "poor" , "good" , "very good". Here , each category can be given a score like 0,1,2,3.

The object description of the Logistic Regression used is given below:

Object Name	**Logistic Regression**
Parameters	**Value**
solver	liblinear
class_weight	balanced

dual	True
fit_intercept	True
penalty	12
intercept_scaling	1
max_iter	100000

Table 5: Object Parameter Table for Logistic Regression

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted TenYearCHD / Actual TenYearCHD	0	1
0	785	288
1	87	112

Table 6: Confusion Matrix Of Logistic Regression

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Logistic Regression model.

The ROC curve is given below:

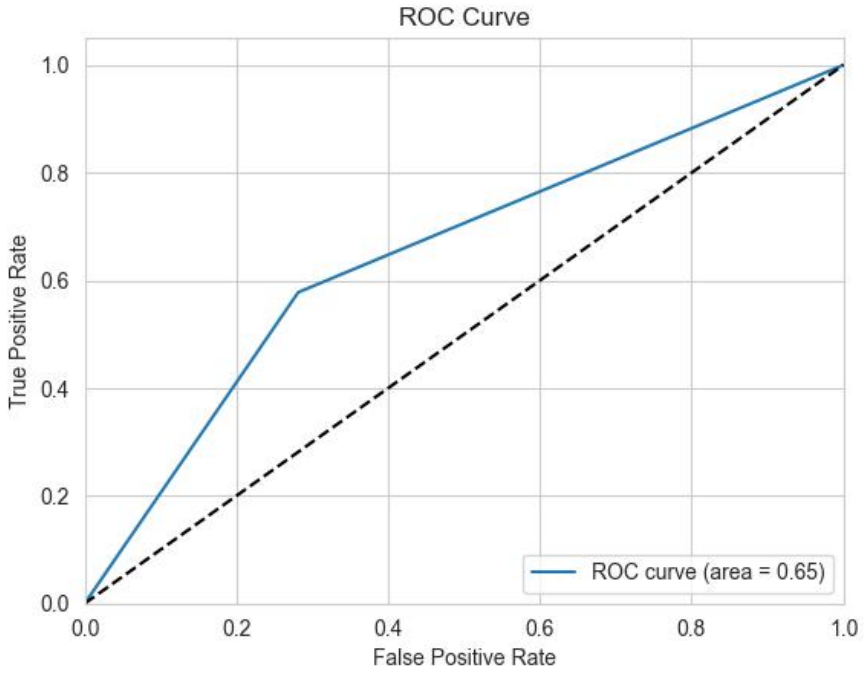


Fig 3: ROC curve of Logistic Regression Model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Logistic Regression Model is 0.65

AUC value of Logistic Regression Model = 0.65

Now we will be preparing the classification report of our Logistic Regression Model.

Accuracy		**0.71**		
	Precision	**recall**	**fl-score**	**support**
0	0.90	0.73	0.81	1073
1	0.28	0.56	0.37	199

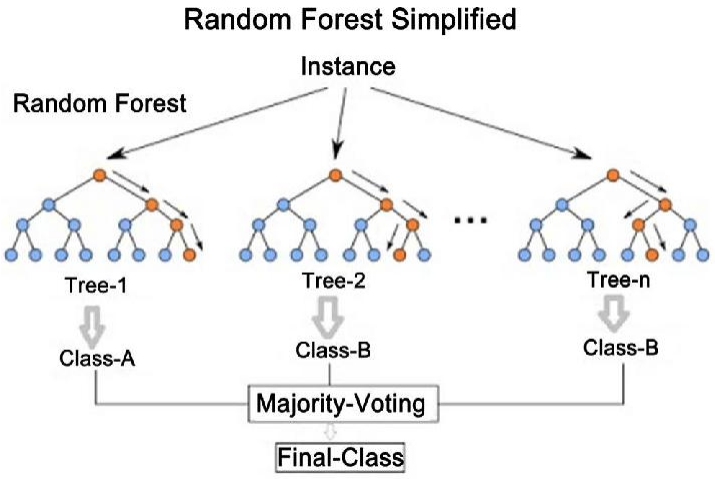
Table 7: Classification Report Table Of Logistic Regression

Random Forest Classifier

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

It consists of multiple decision trees just as a forest has many trees. On top of that, it uses randomness to enhance its accuracy and combat over fitting, which can be a huge issue for such a sophisticated algorithm. These algorithms make decision trees based on a random selection of data samples and get predictions from every tree. After that, they select the best viable solution through votes.

The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. Feature randomness, also known as feature bagging or "the random subspace method", generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features.



As our dataset is not balanced we are using Balanced Random Forest Classifier.

The object description of the Balanced Random Forest Classifier used is given below:

Object Name	**Balanced Random Forest Classifier**
Parameters	**Value**
max_depth	2
random_state	0

Table 8: Object Parameter Table for Random Forest Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted TenYearCHD / **Actual TenYearCHD**	**0**	**1**
0	706	367
1	81	118

Table 9: Confusion Matrix Of Random Forest Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Random Forest Classifier model.

The ROC curve is given below:

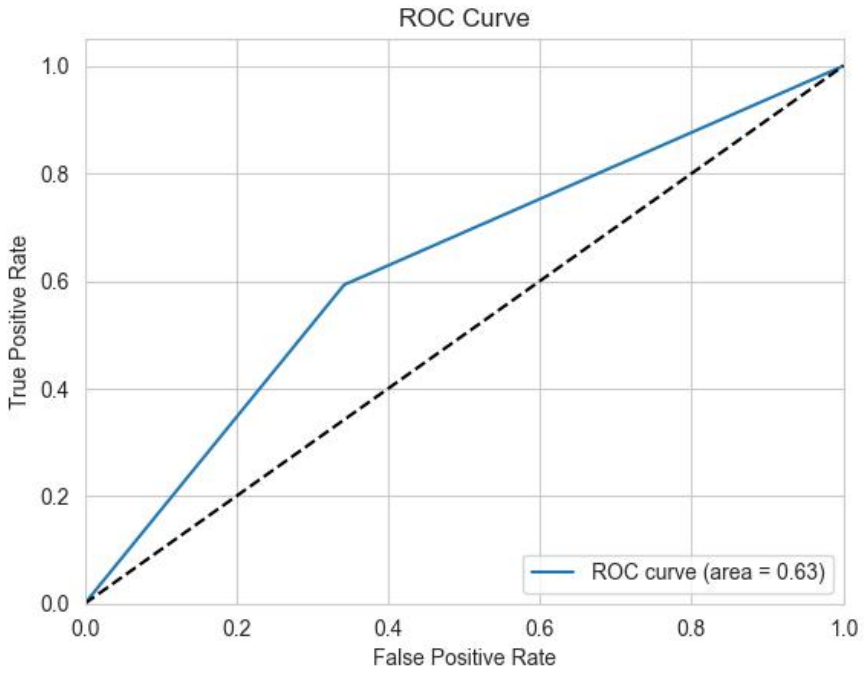


Fig 4: ROC curve of Random Forest Classifier Model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Random Forest Classifier is 0.63

AUC value of Random Forest Classifier = 0.63

Now we will be preparing the classification report of our Random Forest Classifier Model.

Accuracy		**0.65**		
	Precision	**recall**	**fl-score**	**support**
0	0.90	0.66	0.76	1073
1	0.24	0.59	0.35	199

Table 10: Classification Report Table Of Random Forest Classifier

KNN Classifier

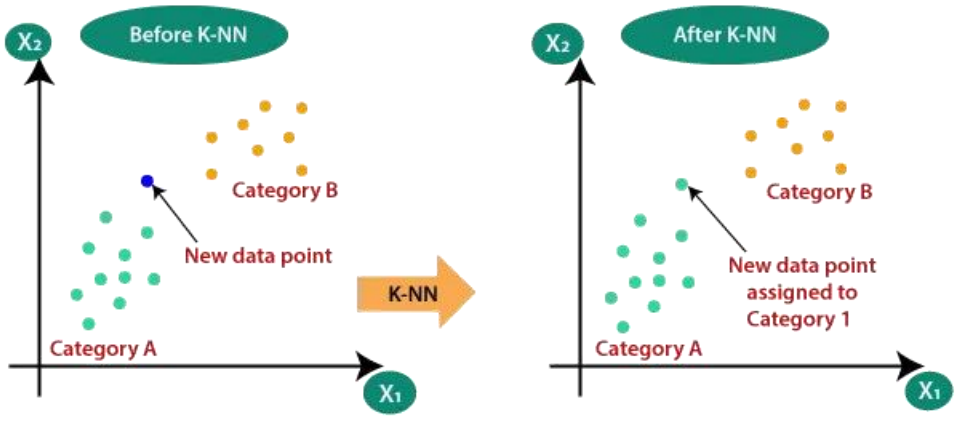
K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closet to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

The K-NN working can be explained on the basis of the below algorithm:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.

Suppose we have a new data point and we need to put it in the required category.

- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:
- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:
- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.



How to choose the optimal value of K:

- There are no pre-defined statistical methods to find the most favorable value of K.
- Initialize a random K value and start computing.
- Choosing a small value of K leads to unstable decision boundaries.
- The substantial K value is better for classification as it leads to smoothening the decision boundaries.
- Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate

.

We used a GridSearchCV object to find the best optimum value of k. Our test results gave the value of k = 5. The object description of the k-NN Classifier used is given below:

Object Name	KNeighborsClassifier
Parameters	**Value**
solver	liblinear
class_weight	balanced
dual	True
fit_intercept	True
penalty	l2
intercept_scaling	1
max_iter	200000

Table 11: Object Parameter Table for KNN Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted TenYearCHD	**0**	**1**
Actual TenYearCHD		
0	1021	52
1	181	18

Table 12: Confusion Matrix Of KNN Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained k-NN model. The ROC curve is given below:

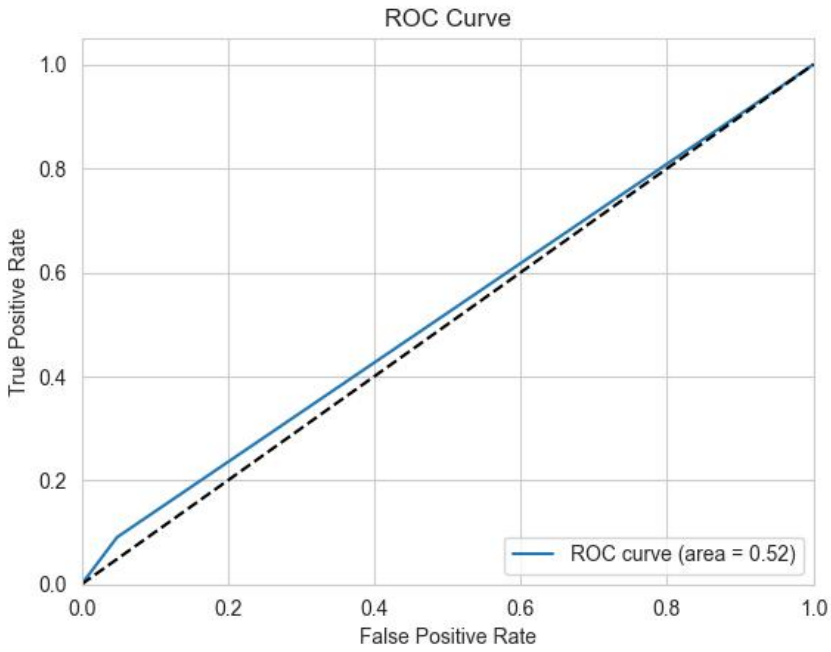


Table 5: ROC curve of KNN Classifier

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our KNN classifier model is 0.52

AUC value of KNN = 0.52

Now we will be preparing the classification report of our K-NN model.

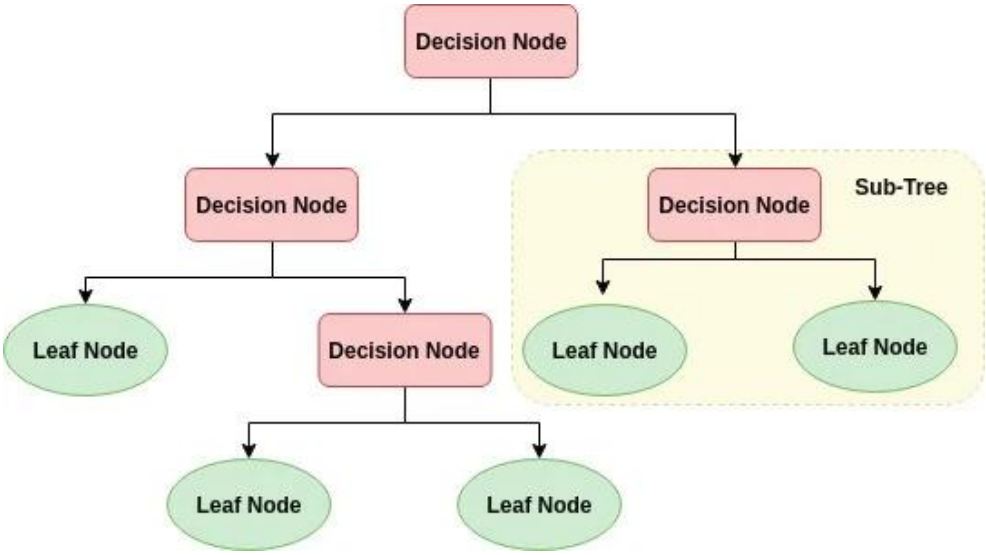
Classification Report of KNN Classifier Model				
Accuracy		**0.86**		
	Precision	**recall**	**fl-score**	**support**
0	0.85	0.95	0.90	1073
1	0.26	0.09	0.13	199

Table 13: Classification Report Table Of KNN Classifier Model

Decision Tree

A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome.

The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



The object description of the Decision Tree Classifier used is given below:

Object Name	DecisionTreeClassifier
Parameters	**Value**
criterion	entropy
max_depth	10
max_features	sqrt
min_samples_leaf	4
min_samples_split	2
min_weight_fraction_leaf	0
splitter	random

Table 14: Object Parameter Table for Decision Tree

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted TenYearCHD / **Actual TenYearCHD**	**0**	**1**
0	1063	15
1	187	7

Table 15: Confusion Matrix Of Decision Tree

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Decision Tree model. The ROC curve is given below:

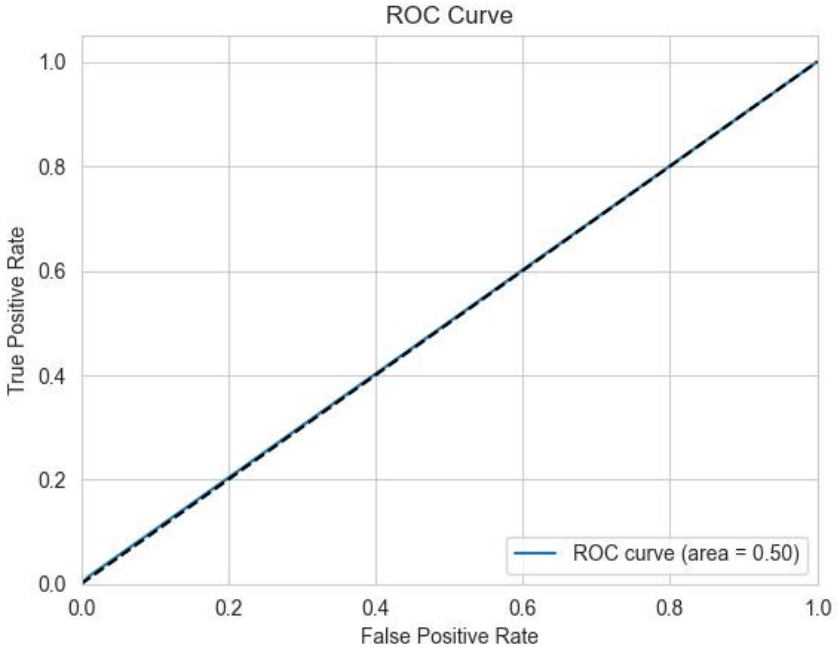


Table 6: ROC curve of Decision Tree

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Decision Tree model is 0.50.

AUC value of Logistic Regression = 0.50

Now we will be preparing the classification report of our Logistic Regression model.

Classification Report of Decision Tree Model				
Accuracy		**0.86**		
	Precision	**recall**	**fl-score**	**support**
0	0.84	0.99	0.91	1073
1	0.20	0.02	0.03	199

Table 16: Classification Report Table Of Decision Tree Model

Comparison of the Models Trained

We have trained 4 models using the 4 four algorithms viz.

1. Logistic Regression
2. Random Forest Classifier
3. K-Nearest Neighbour
4. Decision Tree

The 4 models had different accuracy. The comparison of the accuracies of the models are given below:

Logistic Regression	**0.705**
Random forest	**0.647**
K-Nearest Neighbour	**0.816**
Decision Tree	**0.841**

Table 17: Accuracy Comparison Table

The following bar graph shows the accuracy comparison in graphical way:

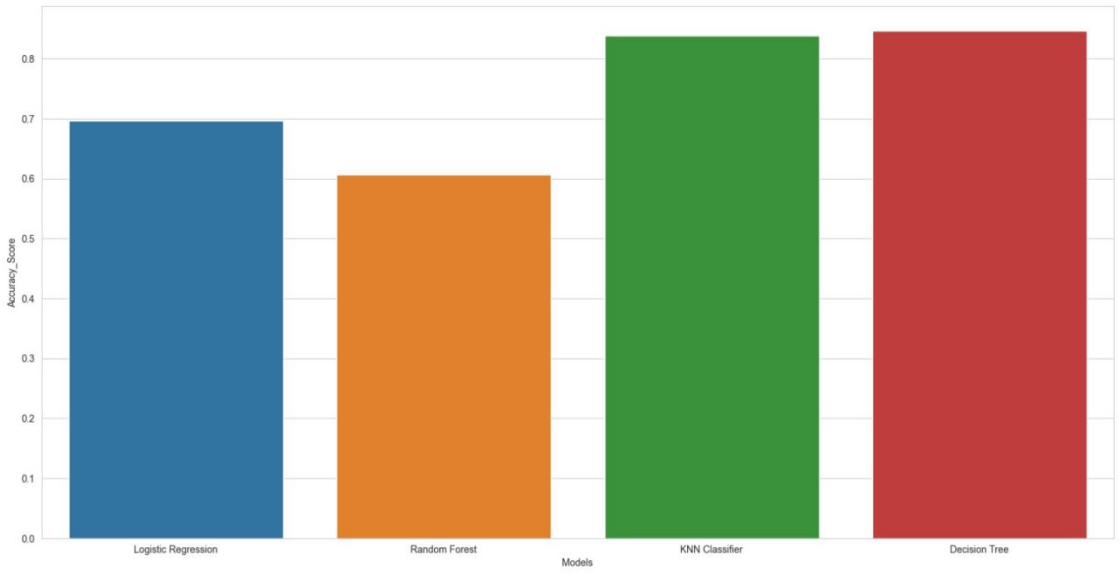


Fig 7: Comparison of accuracy of the 4 different models trained

Thus, from the above comparison we can see that though Decision Tree classifier has the highest accuracy its recall score for 1 (i.e. Having heart disease) is 0.04 which is quite low, that means our model may predict that someone will not have heart disease where as actually they may have. That may be fatal as because if one actually having a chance of heart disease is predicted with no heart disease, precautions may not be taken. So, we will not be choosing KNN and Decision tree and among Logistic Regression and Random Forest (which are giving a balanced result) we choose Logistic Regression because it is giving a more accuracy. So, our selected model is Logistic Regression.

Test Dataset

Testing The given Test Dataset

We were given a test dataset for this Heart Disease Prediction problem. We pre-process the given test dataset in similar way we pre-processed our train dataset.

The methodology followed is given below:

- Checking for null values.
    - If null values are present, we will fill them or drop the row containing the null value based on the dataset.
- Checking for outliers.
    - If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

N.B.: After pre-processing of the test dataset had been done, we retrained the models with 100% of the train dataset given and used those trained models to predict the outcome for each input in the test dataset. There was a total of 1272 inputs in the test dataset. The correct outcomes obtained are tabulated as below:

Classifier Model	**Correct Outcomes**	
	0 (No)	**1(Yes)**
Logistic Regression	785	112
Random Forest	706	118
K-Nearest Neighbour	1021	18
Decision Tree	1061	12

Table 18: Outcomes of 4 different classifier models on the test dataset.

Code

Heart Disease Predictor

About Dataset

Data Description

The dataset is from an ongoing cardiovascular study on residents of the town of Framingham, Massachusetts. The classification goal is to predict whether the patient has 10-year risk of future coronary heart disease (CHD).The dataset provides the patients' information. It includes over 4,000 records and 15 attributes. Each attribute is a potential risk factor. There are both demographic, behavioral and medical risk factors.

Features:

```
01. Male            | Male or Female        ( binary: "1", means
"Yes", "0" means "No" )                              |
02. Age             | Age of the patient
|
03. Cigs Per Day    | The number of cigarettes that the person
smoked on average in one day                          |
04. BP Meds         | Whether or not the patient as on blood
pressure medication                                    |
05. PrevalentStroke | Whether or not the patient had previously
had a stroke                                          |
06. Prevalent Hyp   | Whether or not the patient was hypertensive
|
07. Diabetes        | Whether or not the patient had diabetes
|
08. Tot Chol        | Total cholesterol level
|
09. Sys BP          | Systolic blood pressure
|
10. Dia BP          | Diastolic blood pressure
|
11. BMI             | Body Mass Index
|
12. Heart Rate      | Heart rate
|
13. Glucose         | Glucose level
|
14. TenYearCHD      | 10 year risk of coronary heart disease CHD
(binary: "1", means "Yes", "0" means "No") - TARGET |
```

All of the dataset values were collected at the moment of medical examination.

In [329…

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
```

```
df_master=pd.read_csv("./framingham_heart_disease.csv")
df=df_master.copy()
```

In [330…
```
df
```

Out[330]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp
0	1	39	4.0	0	0.0	0.0	0	0
1	0	46	2.0	0	0.0	0.0	0	0
2	1	48	1.0	1	20.0	0.0	0	0
3	0	61	3.0	1	30.0	0.0	0	1
4	0	46	3.0	1	23.0	0.0	0	0
...	...	...	...	...	...	...	...	...
4235	0	48	2.0	1	20.0	NaN	0	0
4236	0	44	1.0	1	15.0	0.0	0	0
4237	0	52	2.0	0	0.0	0.0	0	0
4238	1	40	3.0	0	0.0	0.0	0	1
4239	0	39	3.0	1	30.0	0.0	0	0

4240 rows × 16 columns

In [331…
```
#Number of (Rows , Columns)
df.shape
```

Out[331]: (4240, 16)

In [332…
```
#Information about DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   male             4240 non-null   int64  
 1   age              4240 non-null   int64  
 2   education        4135 non-null   float64
 3   currentSmoker    4240 non-null   int64  
 4   cigsPerDay       4211 non-null   float64
 5   BPMeds           4187 non-null   float64
 6   prevalentStroke  4240 non-null   int64  
 7   prevalentHyp     4240 non-null   int64  
 8   diabetes         4240 non-null   int64  
 9   totChol          4190 non-null   float64
 10  sysBP            4240 non-null   float64
 11  diaBP            4240 non-null   float64
 12  BMI              4221 non-null   float64
 13  heartRate        4239 non-null   float64
 14  glucose          3852 non-null   float64
 15  TenYearCHD       4240 non-null   int64  
dtypes: float64(9), int64(7)
memory usage: 530.1 KB
```

In [333…
```
#Description about DataFrame
df.describe()
```

Out[333]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalent
count	4240.000000	4240.000000	4135.000000	4240.000000	4211.000000	4187.000000	4240.0
mean	0.429245	49.580189	1.979444	0.494104	9.005937	0.029615	0.0
std	0.495027	8.572942	1.019791	0.500024	11.922462	0.169544	0.0
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.000000	0.0
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.000000	0.0
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.000000	0.0
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.000000	1.0

In [334…
```
#Counting total number of Null Values in particular row
df.isna().sum()
```

Out[334]:
```
male                0
age                 0
education         105
currentSmoker       0
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
heartRate           1
glucose           388
TenYearCHD          0
dtype: int64
```

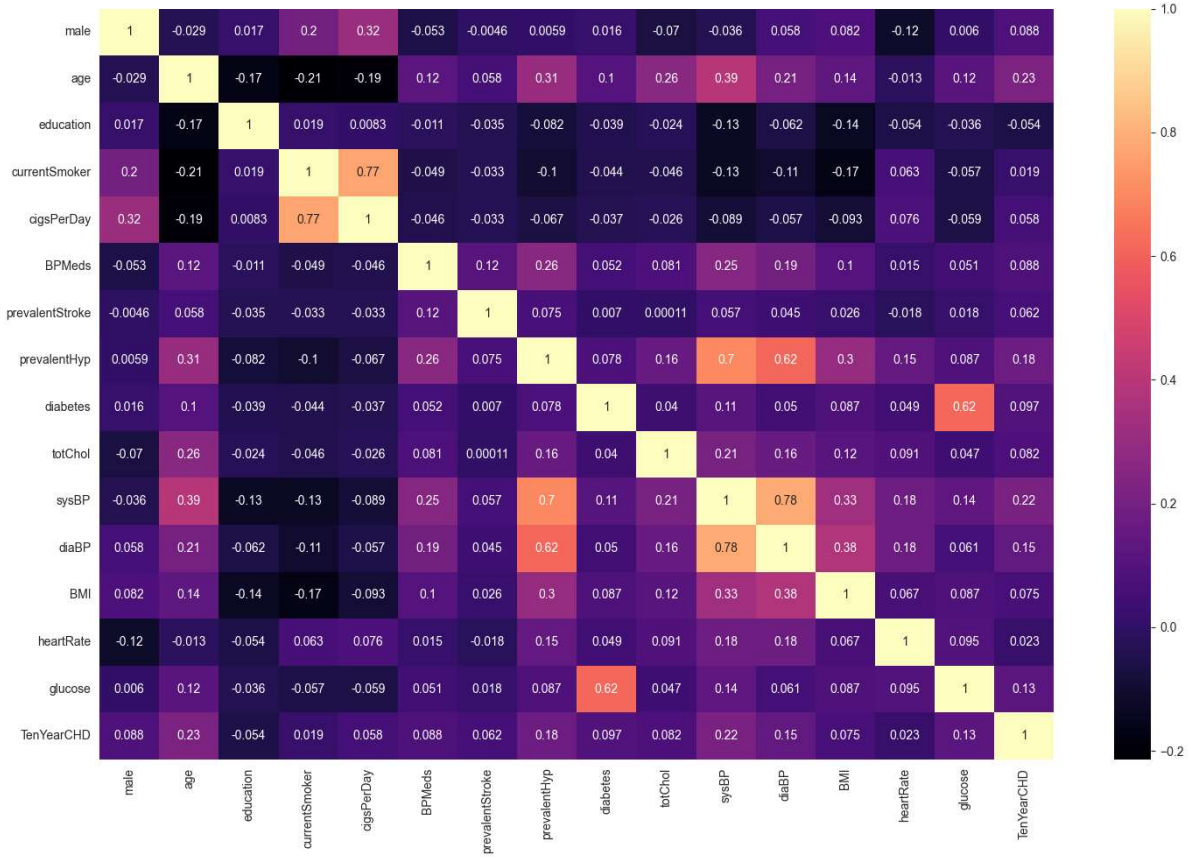
In [335…
```
#Correlation with every attributes
df.corr()
```

Out[335]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalent
male	1.000000	-0.029014	0.017415	0.197026	0.317143	-0.052504	-0.0
age	-0.029014	1.000000	-0.166356	-0.213662	-0.192959	0.123052	0.0
education	0.017415	-0.166356	1.000000	0.018528	0.008323	-0.010898	-0.0
currentSmoker	0.197026	-0.213662	0.018528	1.000000	0.769774	-0.048927	-0.0
cigsPerDay	0.317143	-0.192959	0.008323	0.769774	1.000000	-0.046155	-0.0
BPMeds	-0.052504	0.123052	-0.010898	-0.048927	-0.046155	1.000000	0.1
prevalentStroke	-0.004550	0.057679	-0.035142	-0.032980	-0.032711	0.117370	1.0
prevalentHyp	0.005853	0.306799	-0.081726	-0.103710	-0.066645	0.261067	0.0
diabetes	0.015693	0.101314	-0.038749	-0.044285	-0.037089	0.052060	0.0
totChol	-0.070413	0.262554	-0.023613	-0.046488	-0.026479	0.080623	0.0
sysBP	-0.035879	0.394053	-0.129496	-0.130281	-0.088797	0.254194	0.0
diaBP	0.058199	0.205586	-0.061919	-0.107933	-0.056715	0.194122	0.0
BMI	0.081871	0.136096	-0.137747	-0.167857	-0.093293	0.100702	0.0
heartRate	-0.116932	-0.012843	-0.054182	0.062686	0.075564	0.015230	-0.0
glucose	0.005979	0.122356	-0.035843	-0.056726	-0.058886	0.051197	0.0
TenYearCHD	0.088374	0.225408	-0.054248	0.019448	0.057755	0.087519	0.0

In [336…

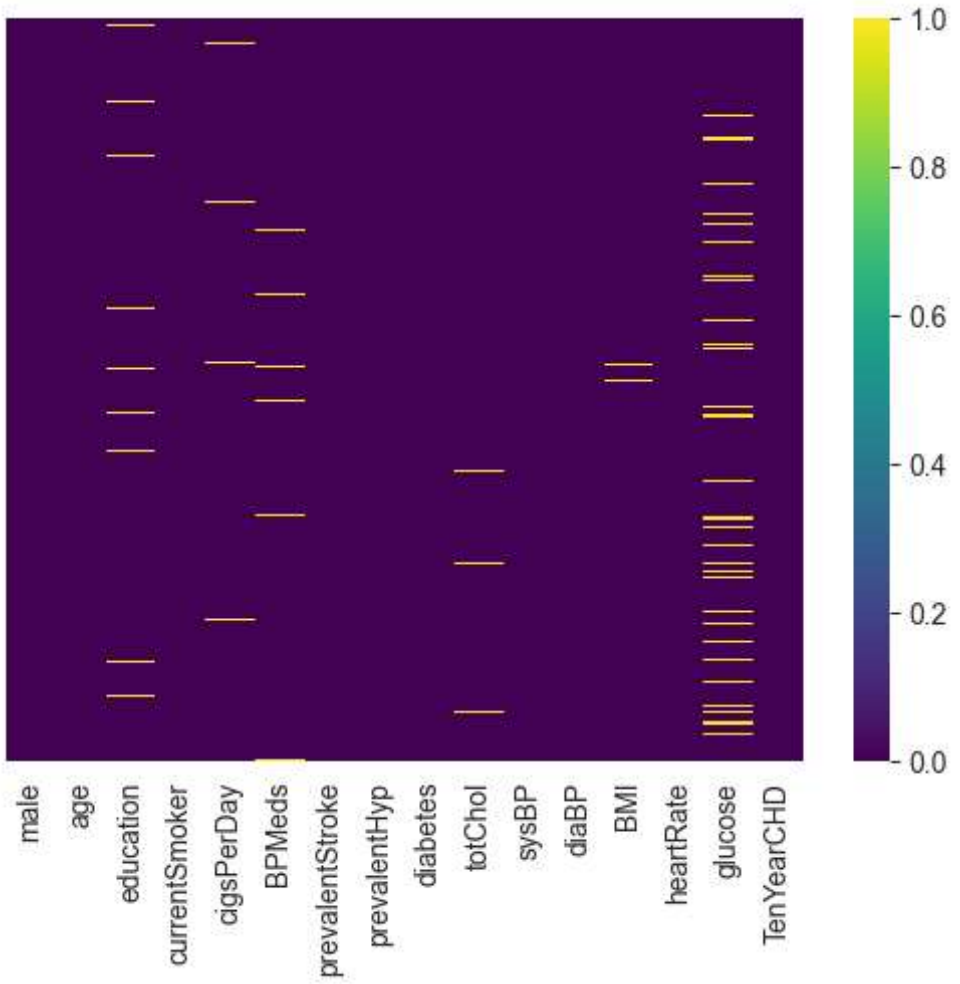
```
#Heatmap of the Correlation Values
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),cmap="magma", annot = True, annot_kws = {"size":10})
plt.tight_layout()
```



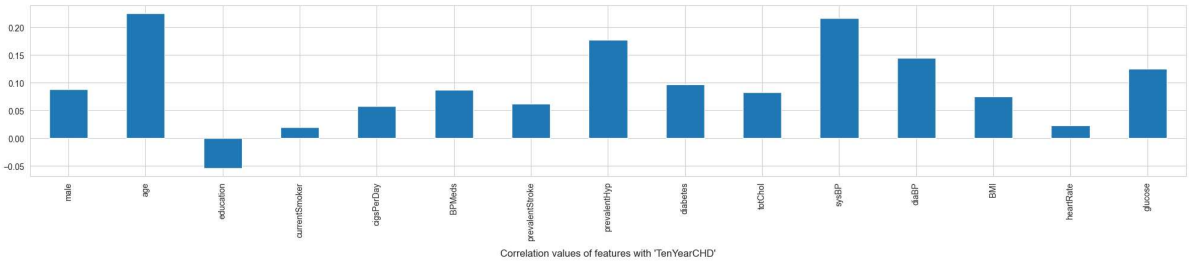
In [337…

```
#Heatmap of Null Values
sns.heatmap(df.isnull(), yticklabels = False, cbar = True, cmap = 'viridis')
```

Out[337]: <AxesSubplot:>

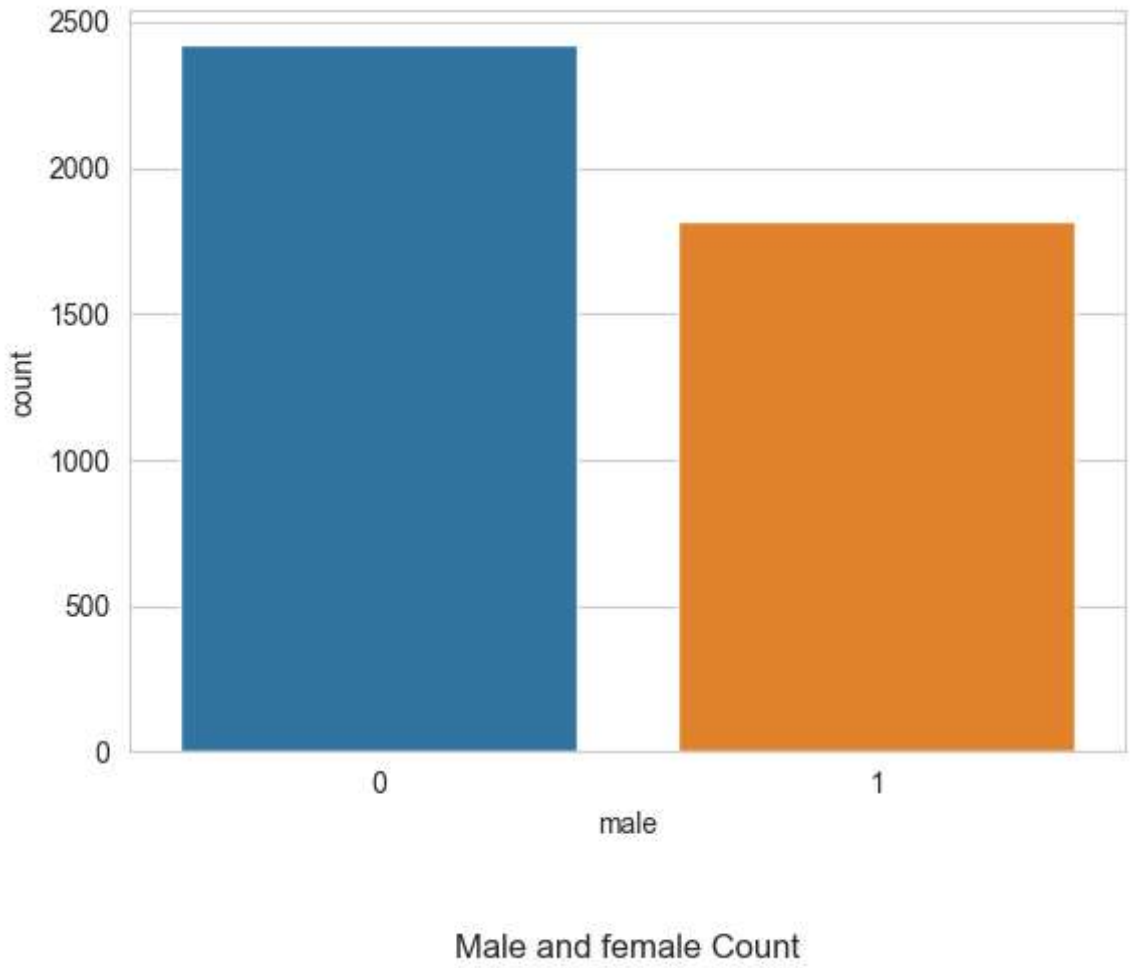


In [338…
```
#Correlation values of every columns with 'TenYearCHD'(Target Value)
plt.figure(figsize=(20,5))
df.drop('TenYearCHD', axis=1,inplace=False).corrwith(df.TenYearCHD).plot(kind='bar
plt.title("Correlation values of features with 'TenYearCHD'",y=-0.5)
plt.tight_layout()
```



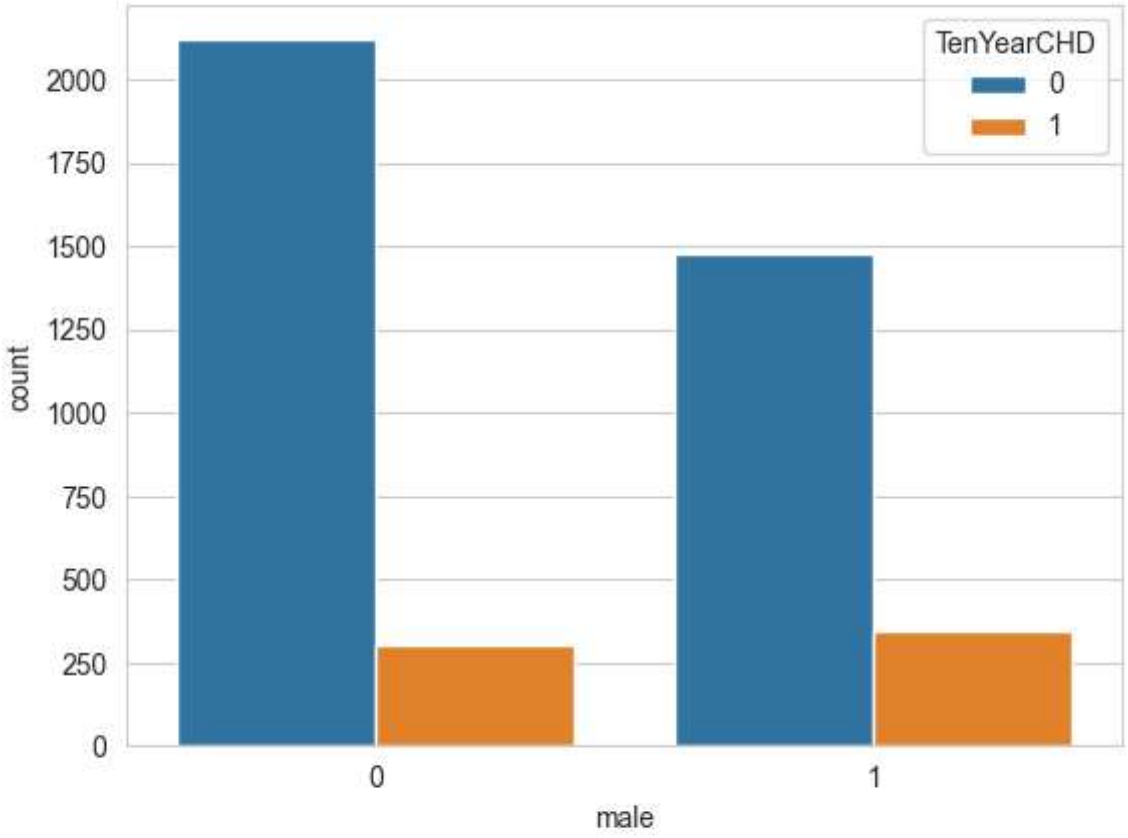
In [339…
```
#Total count of people
sns.countplot(x=df.male)
plt.title("Male and female Count",y=-0.3)
```

Out[339]: Text(0.5, -0.3, 'Male and female Count')



In [340…
```
#Count of Male and Female having TenYearCHD or not
sns.countplot(x=df.male,hue=df.TenYearCHD)
plt.title("Gender vs TenYearCHD",y=-0.3)
```

Out[340]: Text(0.5, -0.3, 'Gender vs TenYearCHD')



Gender vs TenYearCHD

In [341…

```
#Percentage of Female having TenYearCHD in Framingham

femaleCHD = df[ (df["male"]==0) & (df["TenYearCHD"]==1)].shape[0]
print("Total number of Female having TenYearCHD:",femaleCHD)

tot = df["male"].shape[0]
print("Total number of people in Framington:",tot)

femaleCHDpercent = ( femaleCHD/tot )*100
print("Percentage of Female having Heart Disease", femaleCHDpercent)
```

```
Total number of Female having TenYearCHD: 301
Total number of people in Framington: 4240
Percentage of Female having Heart Disease 7.099056603773585
```

In [342…

```
#Percentage of Male having TenYearCHD in Framingham

maleCHD = df[ (df["male"]==1) & (df["TenYearCHD"]==1)].shape[0]
print("Total number of Male having TenYearCHD:",maleCHD)

tot = df["male"].shape[0]
print("Total number of people in Framington:",tot)

maleCHDpercent = ( maleCHD/tot )*100
print("Percentage of Female having Heart Disease", maleCHDpercent)
```

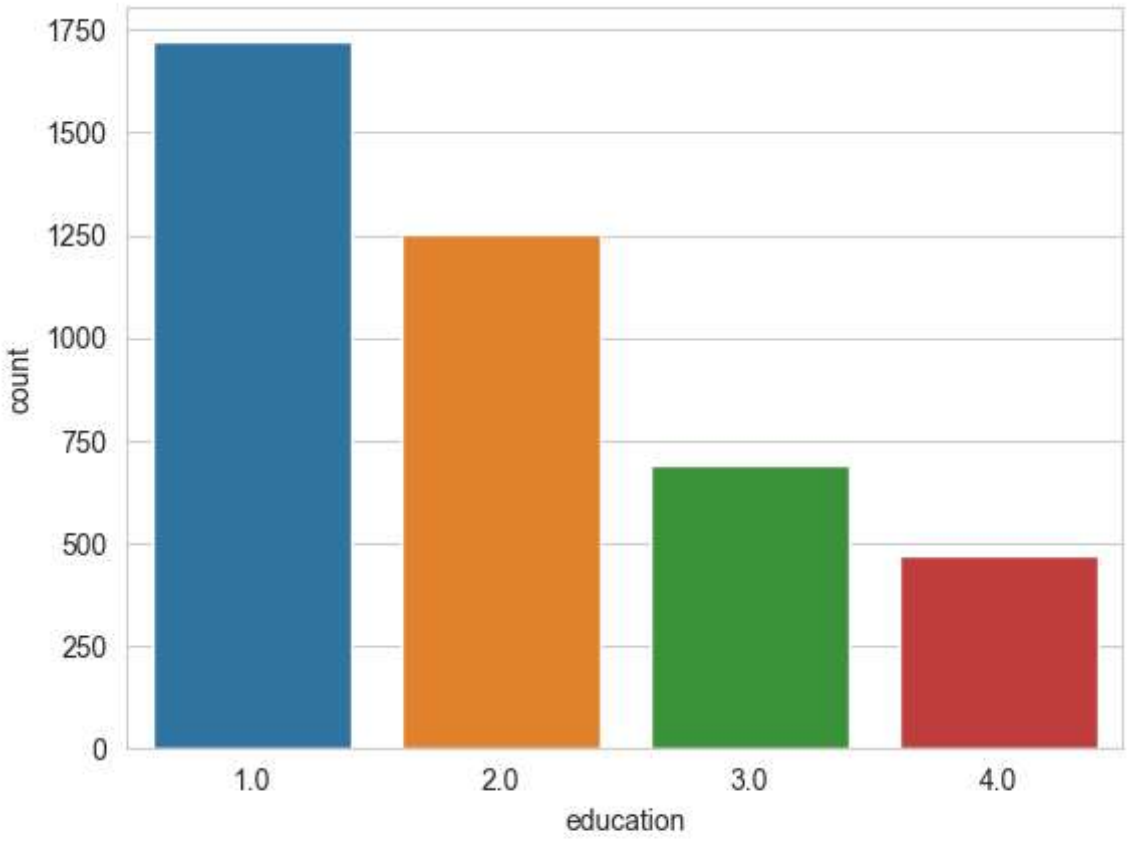
```
Total number of Male having TenYearCHD: 343
Total number of people in Framington: 4240
Percentage of Female having Heart Disease 8.089622641509434
```

In [343…

```
#Count of levels of education
sns.countplot(x=df.education)
```

```
plt.title("Count of types of education",y=-0.3)
```

Out[343]:
```
Text(0.5, -0.3, 'Count of types of education')
```



Count of types of education

In [344…
```
#Correlation of Education with every columns
df.corr().education
```

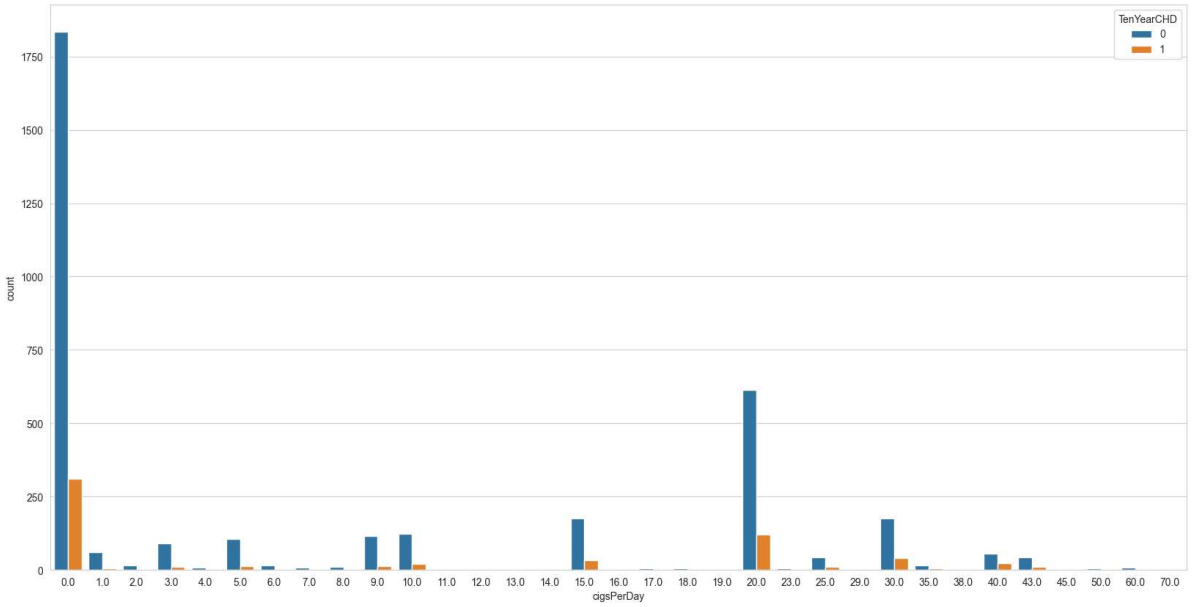
Out[344]:
```
male              0.017415
age              -0.166356
education         1.000000
currentSmoker     0.018528
cigsPerDay        0.008323
BPMeds           -0.010898
prevalentStroke  -0.035142
prevalentHyp     -0.081726
diabetes         -0.038749
totChol          -0.023613
sysBP            -0.129496
diaBP            -0.061919
BMI              -0.137747
heartRate        -0.054182
glucose          -0.035843
TenYearCHD       -0.054248
Name: education, dtype: float64
```

In [345…
```
#Number of male
x = df['male'].value_counts()
print(x)
print("0 = Female ")
print("1 = Male ")
```

```
0    2420
1    1820
Name: male, dtype: int64
0 = Female
1 = Male
```

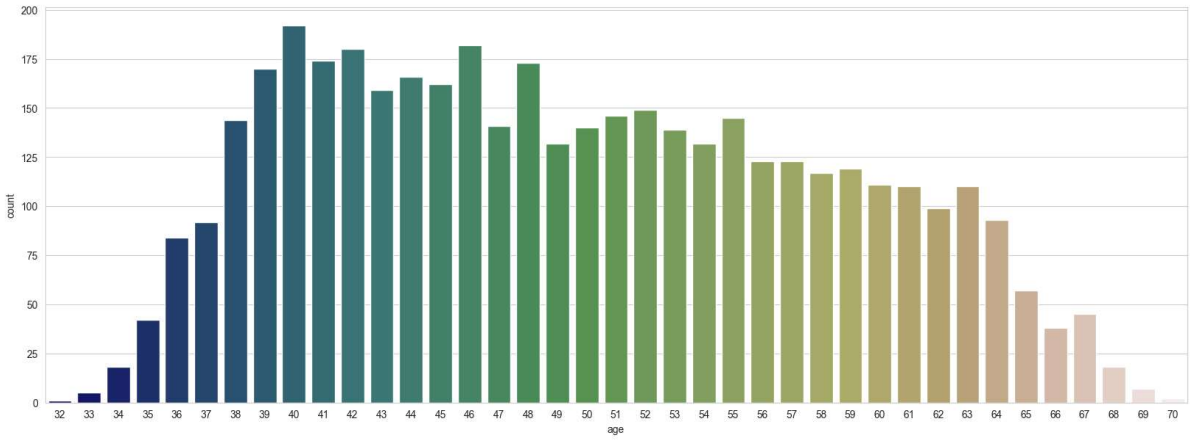
In [346…
```
#Dropping columns not required
df.drop(columns=['education','currentSmoker','heartRate'],inplace=True)
```

In [347…
```
# plot fot TenYearCHD vs. CigsPerDay
plt.figure(figsize=(20,10))
sns.countplot(x=df.cigsPerDay,hue=df.TenYearCHD)
plt.title("CigsPerDay vs. TenYearCHD",y=-0.3)
plt.show()
```



CigsPerDay vs. TenYearCHD

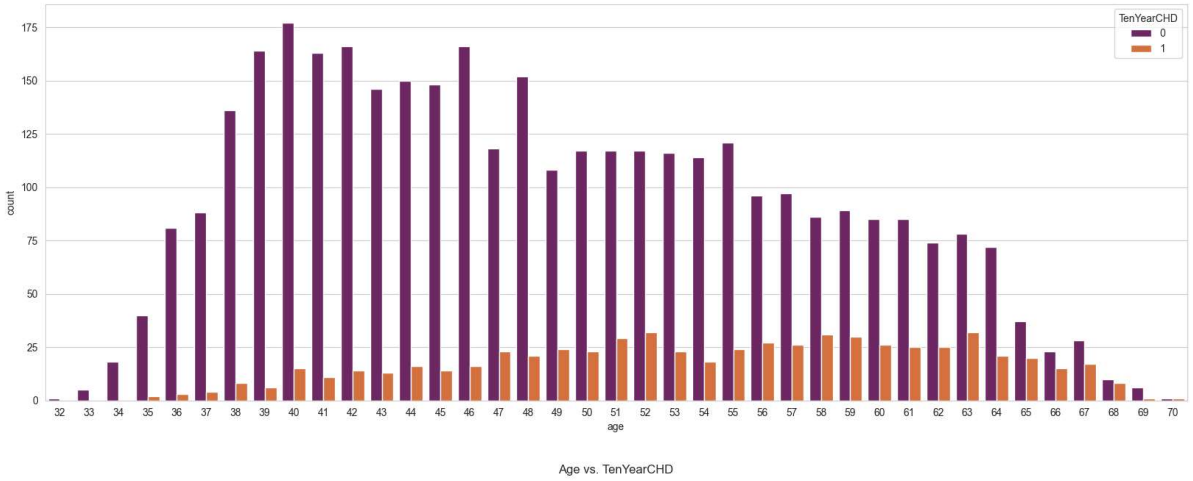
In [348…
```
#count by age
plt.figure(figsize=(20,7))
sns.countplot(x=df.age,palette='gist_earth')
plt.title('People count by age',y=-0.2);
```



People count by age

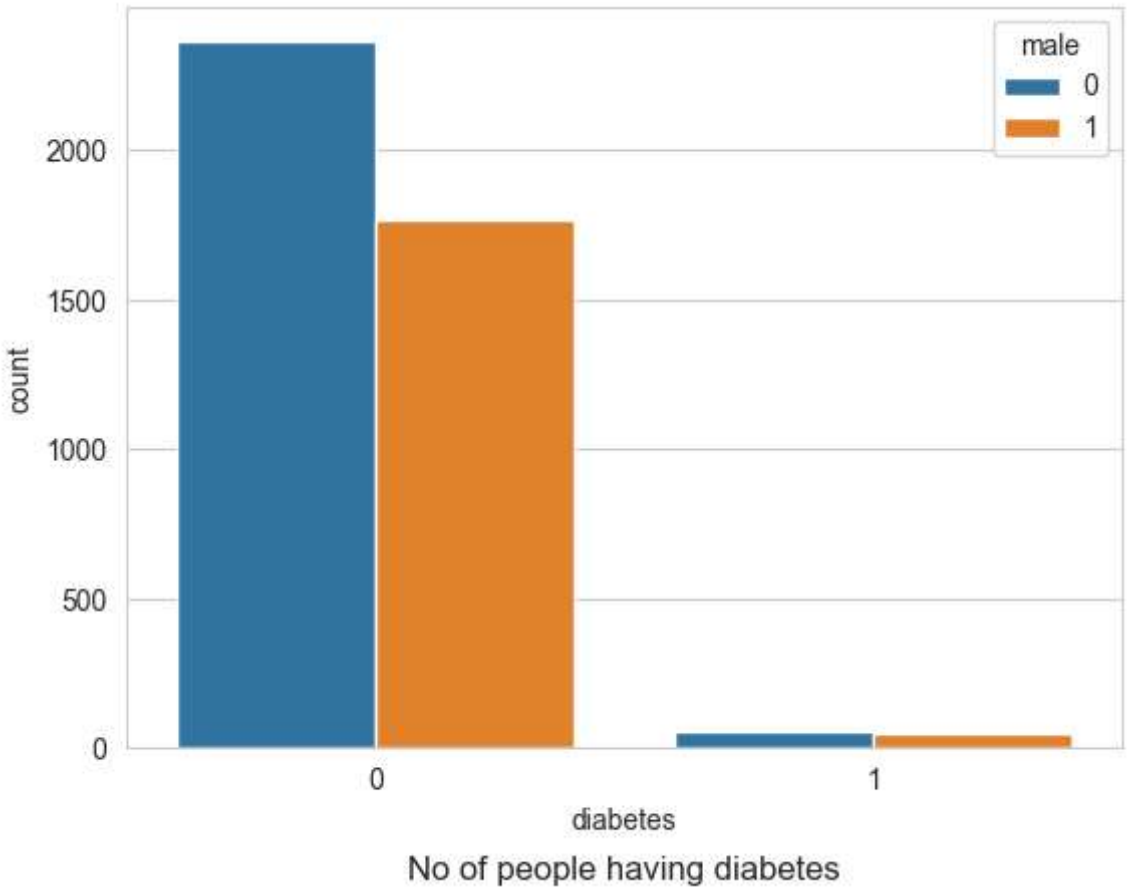
In [349…
```
#age vs TenYearCHD
plt.figure(figsize=(20,7))
sns.set_style('whitegrid')
```

```
sns.countplot(x = df.age ,hue=df.TenYearCHD, palette = 'inferno')
plt.title("Age vs. TenYearCHD",y=-0.2)
plt.show()
```



Age vs. TenYearCHD

In [350…
```
#Count of people having diabetes vs. TenYearCHD
sns.countplot(x=df.diabetes,hue=df.male)
plt.title("No of people having diabetes",y=-0.2)
plt.show()
```

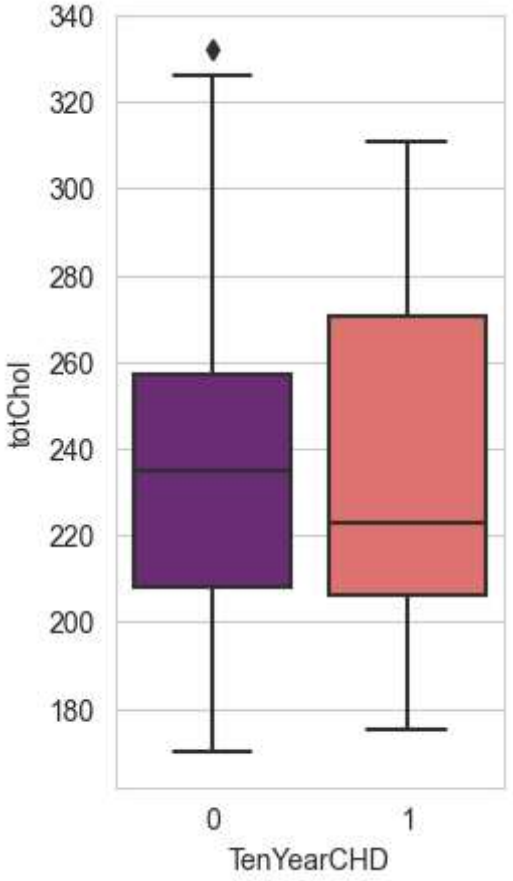


No of people having diabetes

In [351…
```
#count of poeple having diabetes
df[df.diabetes==1].diabetes.count()
```

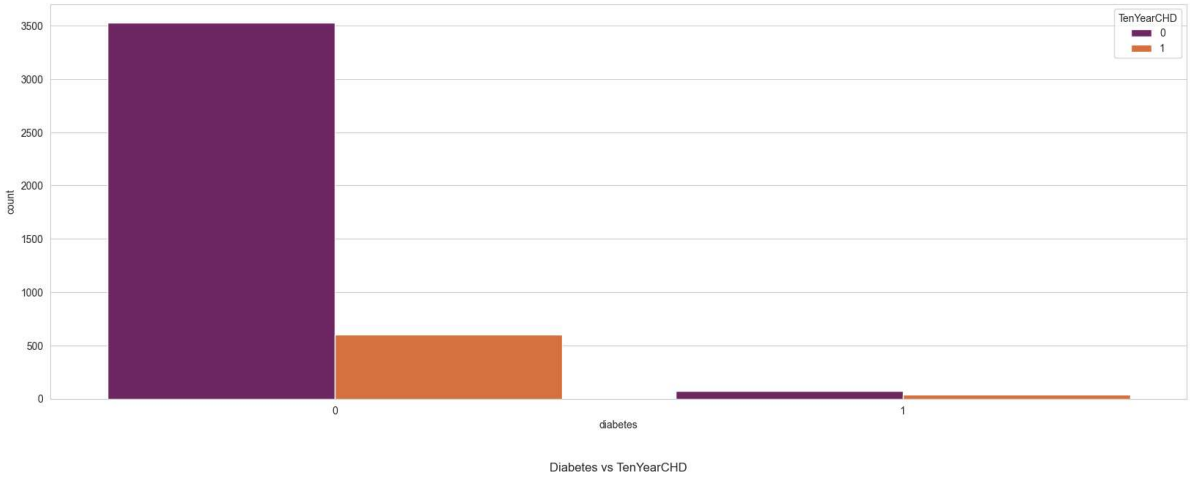
Out[351]: 109

In [352…
```
#Bar plot of TenYearCHD vs totChol
plt.figure(figsize=(2.5,5))
sns.boxplot(x= df.TenYearCHD, y = df.totChol[0:100], palette='magma')
plt.show()
```



In [353…

```
#Diabetes vs. TenYearCHD
plt.figure(figsize=(20,7))
sns.set_style('whitegrid')
sns.countplot(x = df.diabetes ,hue=df.TenYearCHD, palette = 'inferno')
plt.title("Diabetes vs TenYearCHD",y=-0.2)
plt.show()
```



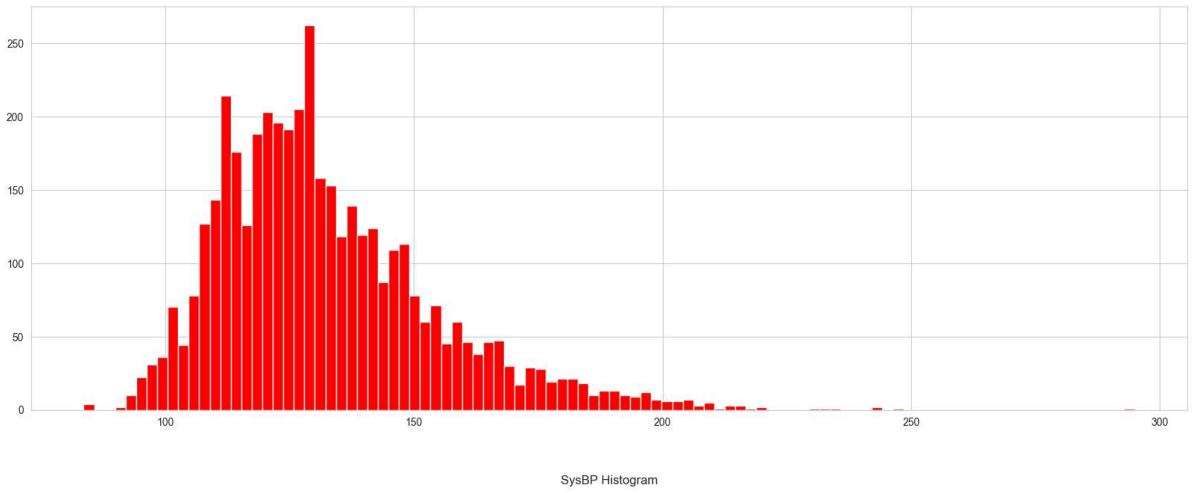
In [354…

```
#Count of unique values in every column
for i in df.columns:
    print(i,len(df[i].unique()))
```

```
male 2
age 39
cigsPerDay 34
BPMeds 3
prevalentStroke 2
prevalentHyp 2
diabetes 2
totChol 249
sysBP 234
diaBP 146
BMI 1365
glucose 144
TenYearCHD 2
```

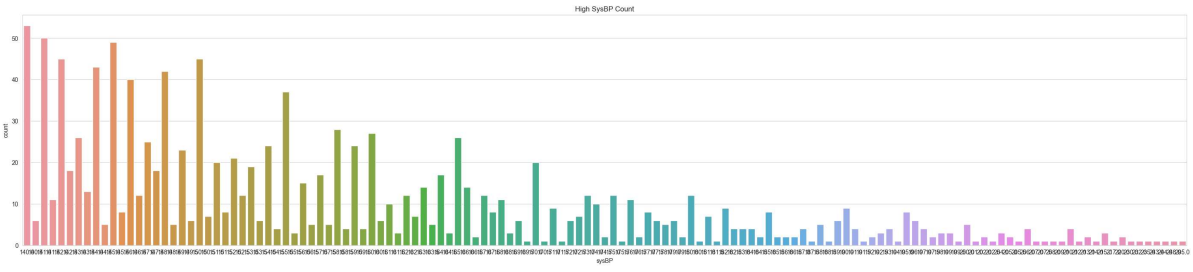
In [355…

```
#Histogram for sysBP
df['sysBP'].hist(color='red',bins=100,figsize=(20,7))
plt.title("SysBP Histogram",y=-0.2)
plt.show()
```



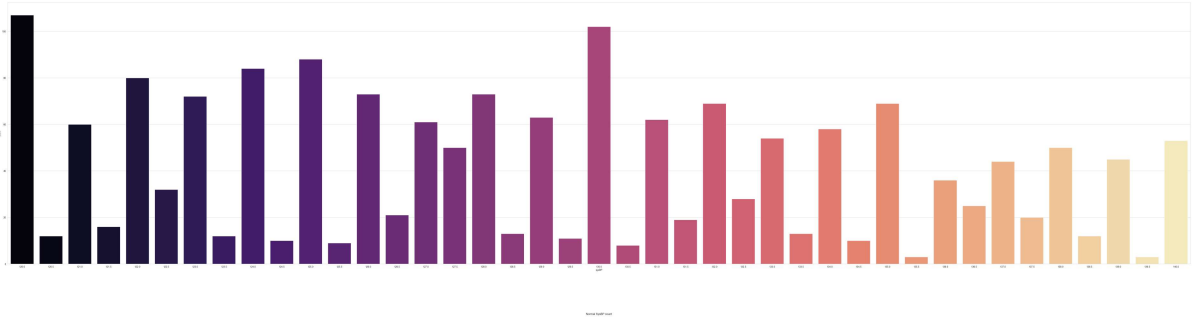
In [356…

```
#High SysBP Count
plt.figure(figsize=(35,7))
dfHighBp = df[df.sysBP>=140]
dfHighBp
plt.title("High SysBP Count")
sns.countplot(x=dfHighBp.sysBP)
plt.show()
```



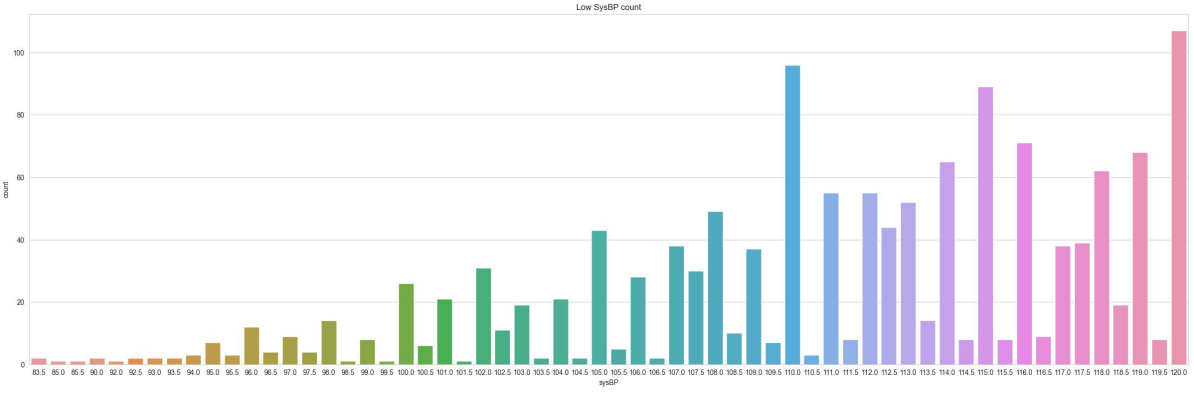
In [357…

```
#Normal SysBp Count
plt.figure(figsize=(90, 20))
dfNBp = df[(df.sysBP>=120)&(df.sysBP<=140)]
dfNBp

sns.countplot(x=dfNBp.sysBP,palette='magma')
plt.title("Normal SysBP count",y=-0.2)
plt.show()
```



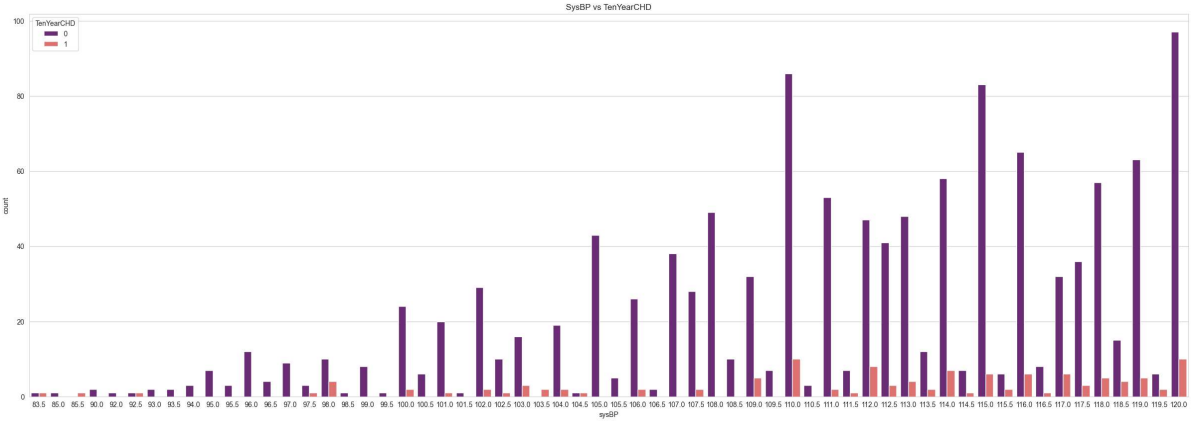
In [358…

```
#Low SysBP Count
plt.figure(figsize=(30, 20))
dfLowBp = df[df.sysBP<=120]
dfLowBp
plt.subplot(2,1,1)
sns.countplot(x=dfLowBp.sysBP)
plt.title("Low SysBP count")
plt.show()
```



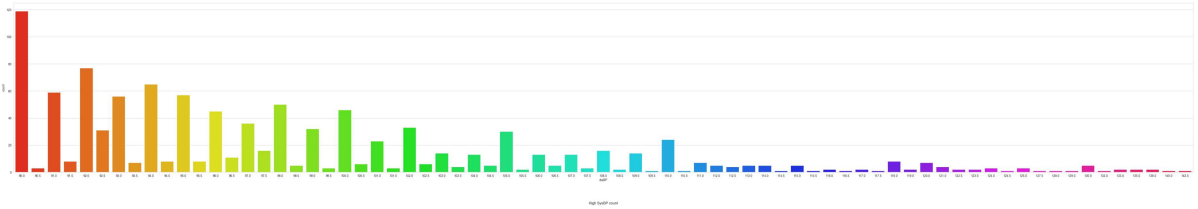
In [359…

```
#Low SysBP vs TenYearCHD
plt.figure(figsize=(30, 10))
sns.countplot(x=dfLowBp.sysBP, hue=df.TenYearCHD, palette='magma')
plt.title("SysBP vs TenYearCHD")
plt.show()
```



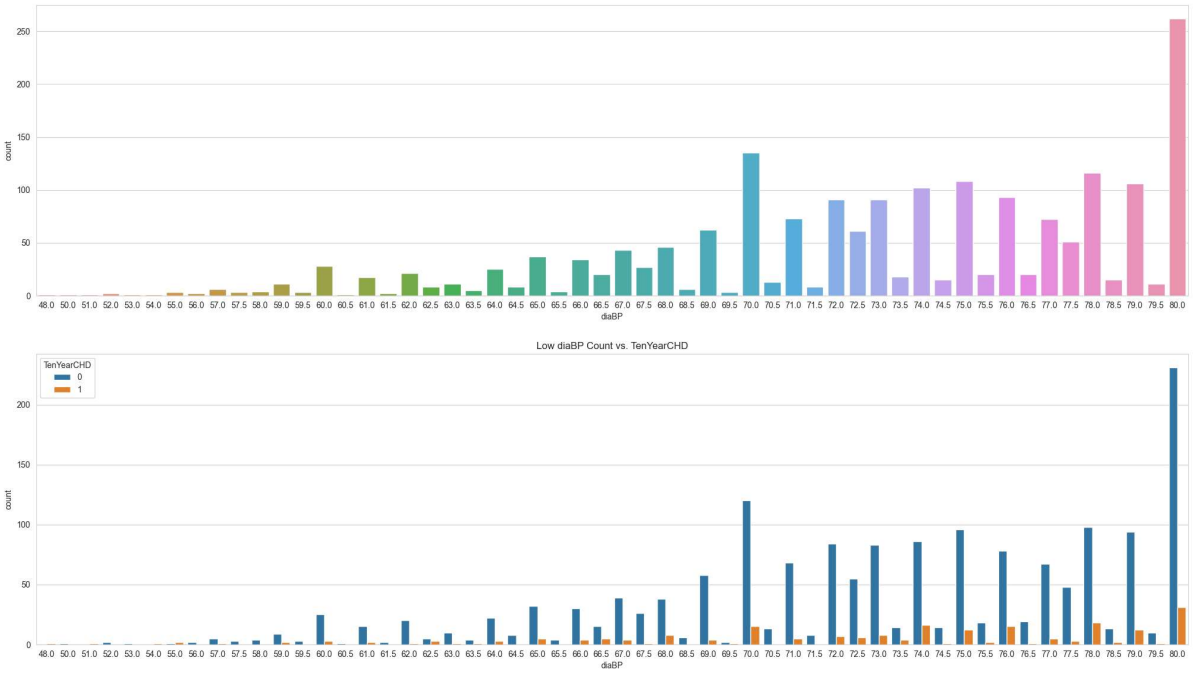
In [360…

```
#High DiaBP Count
plt.figure(figsize=(70,10))
dfHighBp = df[df.diaBP>=90]
dfHighBp
sns.countplot(x=dfHighBp.diaBP, palette='hsv')
plt.title("High SysBP count",y=-0.2)
plt.show()
```



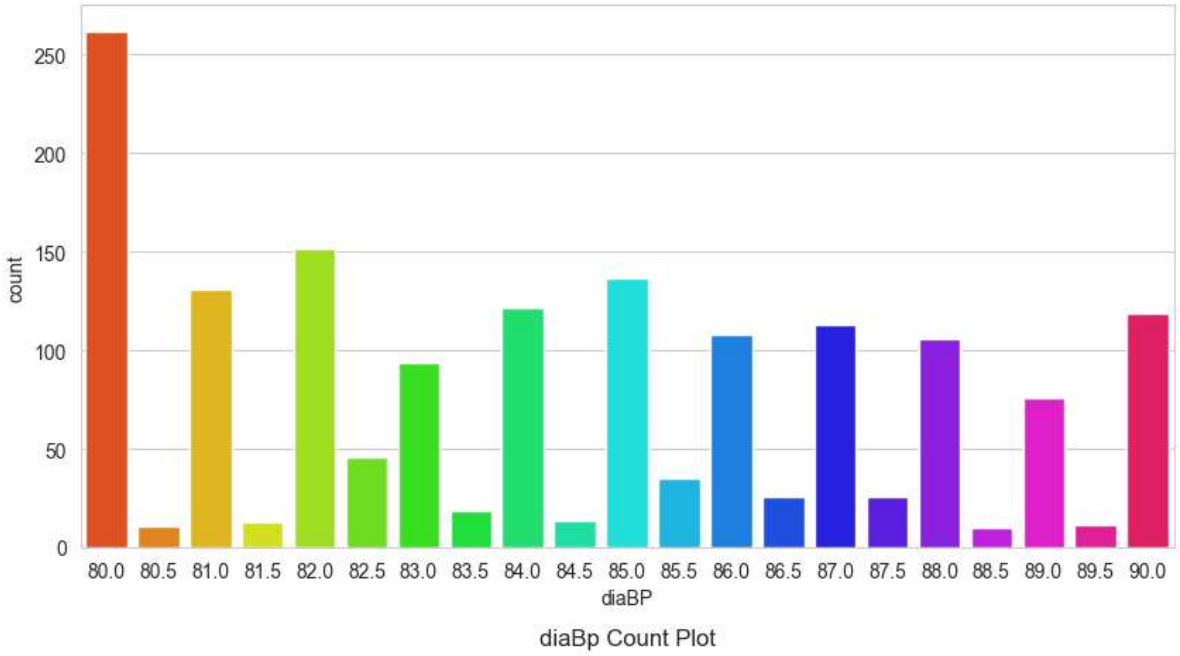
In [361…

```
#Low DiaBP Count
plt.figure(figsize=(25,14))
sns.set_style('whitegrid')
dfLowBp = df[df.diaBP<=80]
dfLowBp
plt.subplot(2,1,1)
sns.countplot(x=dfLowBp.diaBP)
plt.title("Low diaBP Count vs. TenYearCHD",y=-0.2)
plt.subplot(2,1,2)
sns.countplot(x=dfLowBp.diaBP,hue=df.TenYearCHD)
plt.show()
```



In [362…

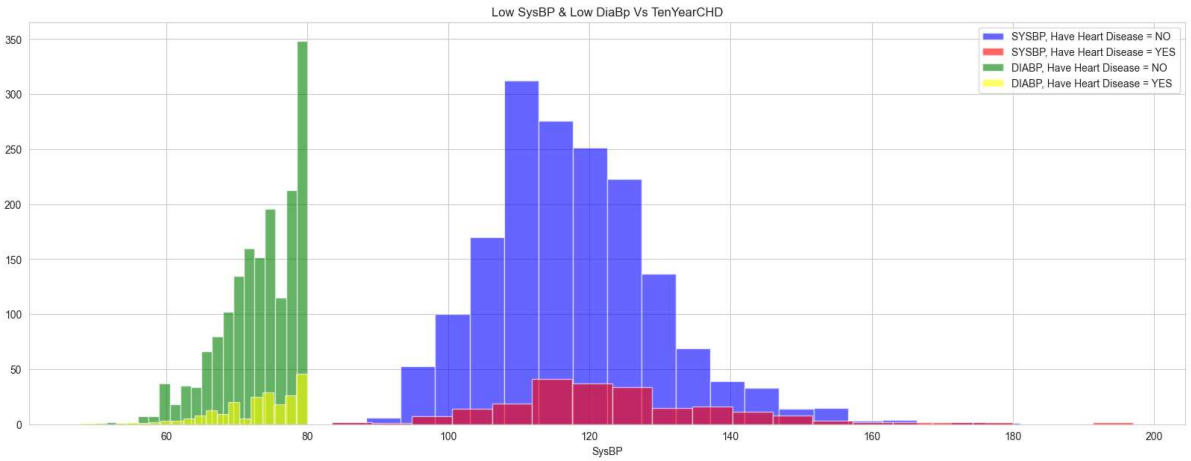
```
#Normal DiaBP COunt
plt.figure(figsize=(10,5))
dfNBp = df[(df.diaBP>=80) & (df.diaBP<=90)]
dfNBp
plt.title("diaBp Count Plot",y=-0.2)
sns.countplot(x=dfNBp.diaBP, palette='hsv')

plt.show()
```



diaBp Count Plot

In [363…

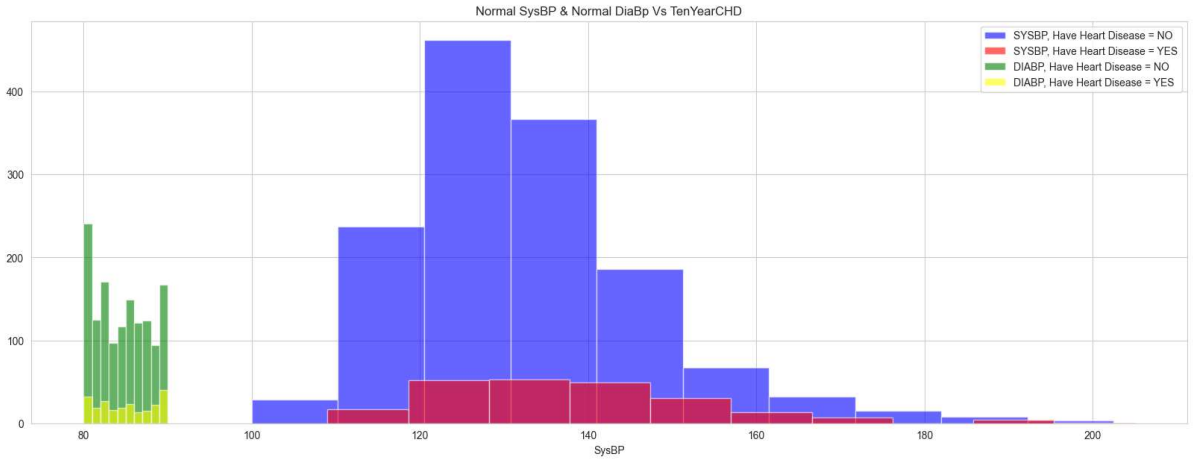
```
#Low SysBP & Low DiaBp Vs TenYearCHD
plt.figure(figsize=(20,7))
# sns.barplot(x=['Ten Year CHD',"fgrg","uiu"],y=[len(dfLowBp.TenYearCHD),len(dfNBp.

# sns.barplot(x=['Low BP',"High BP","Normal BP"], y=[len(dfLowBp.sysBP),len(dfNBp.s
dfLowBp[dfLowBp["TenYearCHD"]==0]["sysBP"].hist(bins=20, color='blue', label='SYSBF
dfLowBp[dfLowBp["TenYearCHD"]==1]["sysBP"].hist(bins=20, color='red', label='SYSBP,
plt.xlabel("SysBP")
dfLowBp[dfLowBp["TenYearCHD"]==0]["diaBP"].hist(bins=20, color='green', label='DIAE
dfLowBp[dfLowBp["TenYearCHD"]==1]["diaBP"].hist(bins=20, color='yellow', label='DIA
plt.xlabel("SysBP")
plt.title("Low SysBP & Low DiaBp Vs TenYearCHD",y=1)
plt.legend()
plt.show()
```



In [364…

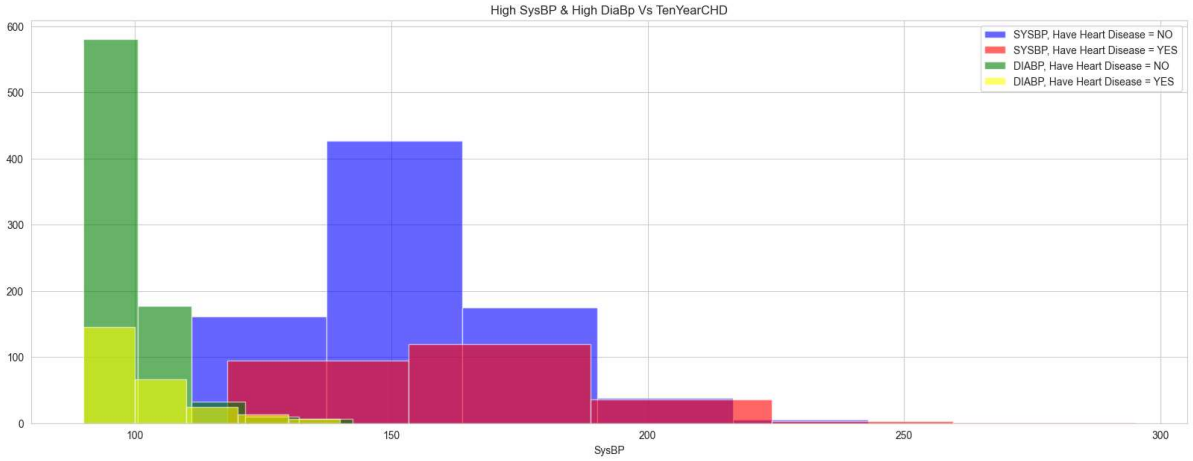
```
#Normal SysBP & Normal DiaBp Vs TenYearCHD
plt.figure(figsize=(20,7))
dfNBp[dfNBp["TenYearCHD"]==0]["sysBP"].hist(bins=10, color='blue', label='SYSBP, Ha
dfNBp[dfNBp["TenYearCHD"]==1]["sysBP"].hist(bins=10, color='red', label='SYSBP, Hav
plt.xlabel("SysBP")
dfNBp[dfNBp["TenYearCHD"]==0]["diaBP"].hist(bins=10, color='green', label='DIABP, H
dfNBp[dfNBp["TenYearCHD"]==1]["diaBP"].hist(bins=10, color='yellow', label='DIABP,
plt.xlabel("SysBP")
plt.title("Normal SysBP & Normal DiaBp Vs TenYearCHD",y=1)
```

```
plt.legend()
plt.show()
```



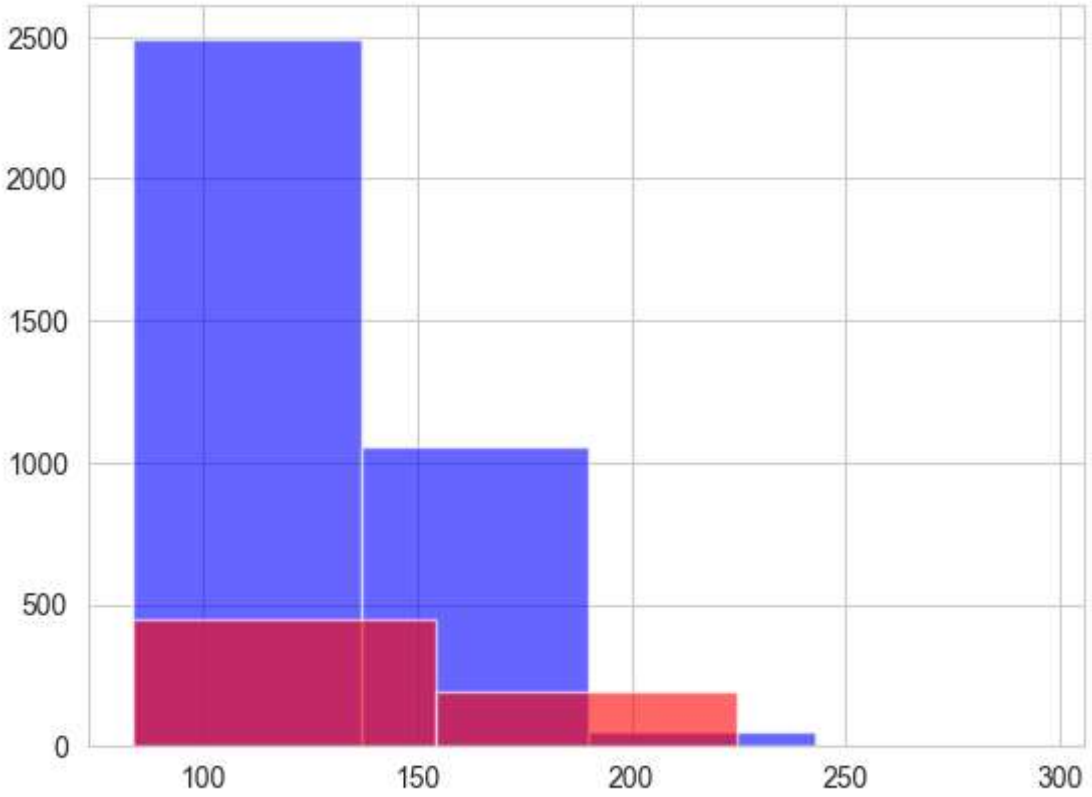
In [365…

```
#High SysBP & High DiaBp Vs TenYearCHD
plt.figure(figsize=(20,7))
dfHighBp[dfHighBp["TenYearCHD"]==0]["sysBP"].hist(bins=5, color='blue', label='SYSE
dfHighBp[dfHighBp["TenYearCHD"]==1]["sysBP"].hist(bins=5, color='red', label='SYSBP
plt.xlabel("SysBP")
dfHighBp[dfHighBp["TenYearCHD"]==0]["diaBP"].hist(bins=5, color='green', label='DIA
dfHighBp[dfHighBp["TenYearCHD"]==1]["diaBP"].hist(bins=5, color='yellow', label='DI
plt.xlabel("SysBP")
plt.title("High SysBP & High DiaBp Vs TenYearCHD",y=1)
plt.legend()
plt.show()
```



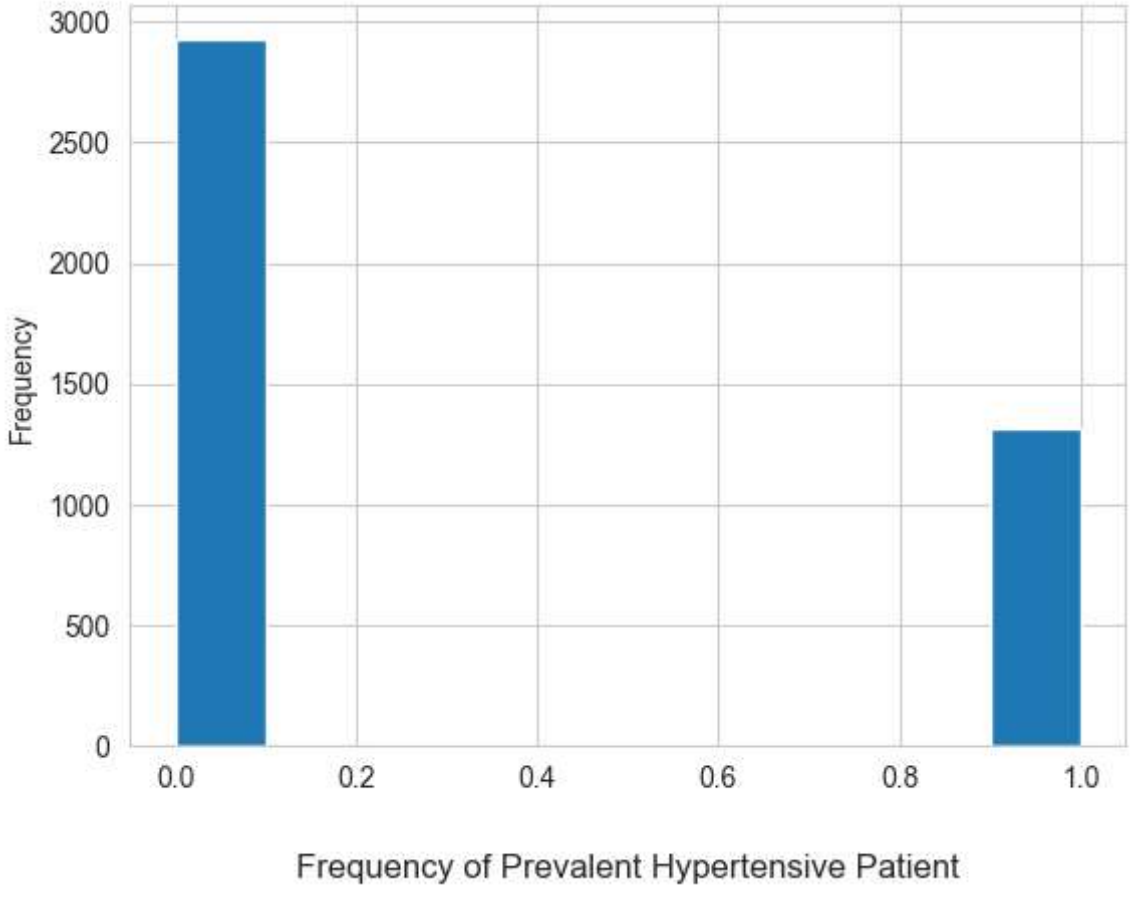
In [366…

```
df[df["TenYearCHD"]==0]["sysBP"].hist(bins=3,color="blue",alpha=0.6 , label='')
df[df["TenYearCHD"]==1]['sysBP'].hist(bins=3,color="red",alpha=0.6  , label='')
plt.title("",y=-0.2)
plt.show()
```



In [367…

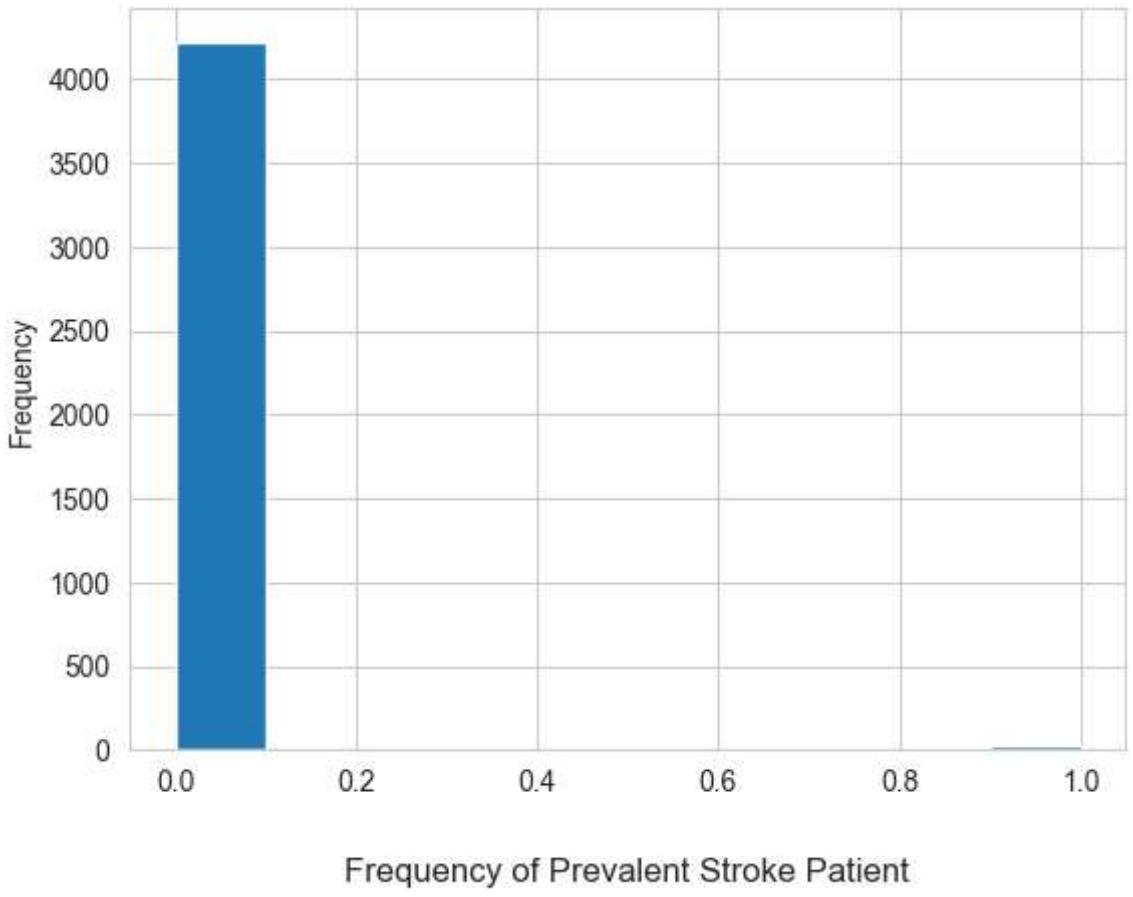
```
#Frequency of Prevalent Hypertensive Patient
df.     prevalentHyp.plot.hist()
plt.title("Frequency of Prevalent Hypertensive Patient",y=-0.2)
plt.show()
```



In [368…

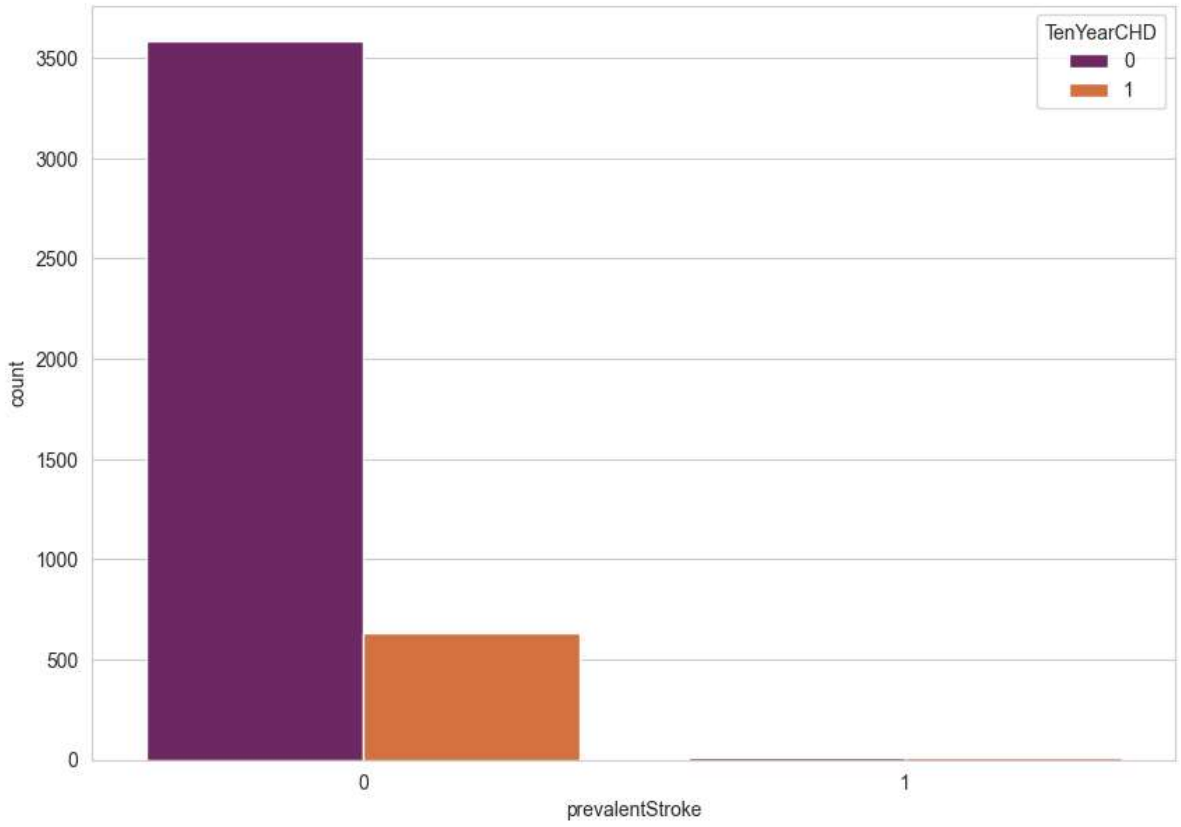
```
#Frequency of Prevalent Stroke Patient
df.     prevalentStroke.plot.hist()
```

```
plt.title("Frequency of Prevalent Stroke Patient",y=-0.2)
plt.show()
```



In [369…
```
#PreveletStroke vs TenYearCHD
plt.figure(figsize=(10,7))
sns.set_style('whitegrid')
sns.countplot(x = df.prevalentStroke ,hue=df.TenYearCHD, palette = 'inferno')
plt.title("PreveletStroke vs TenYearCHD",y=-0.2)
```

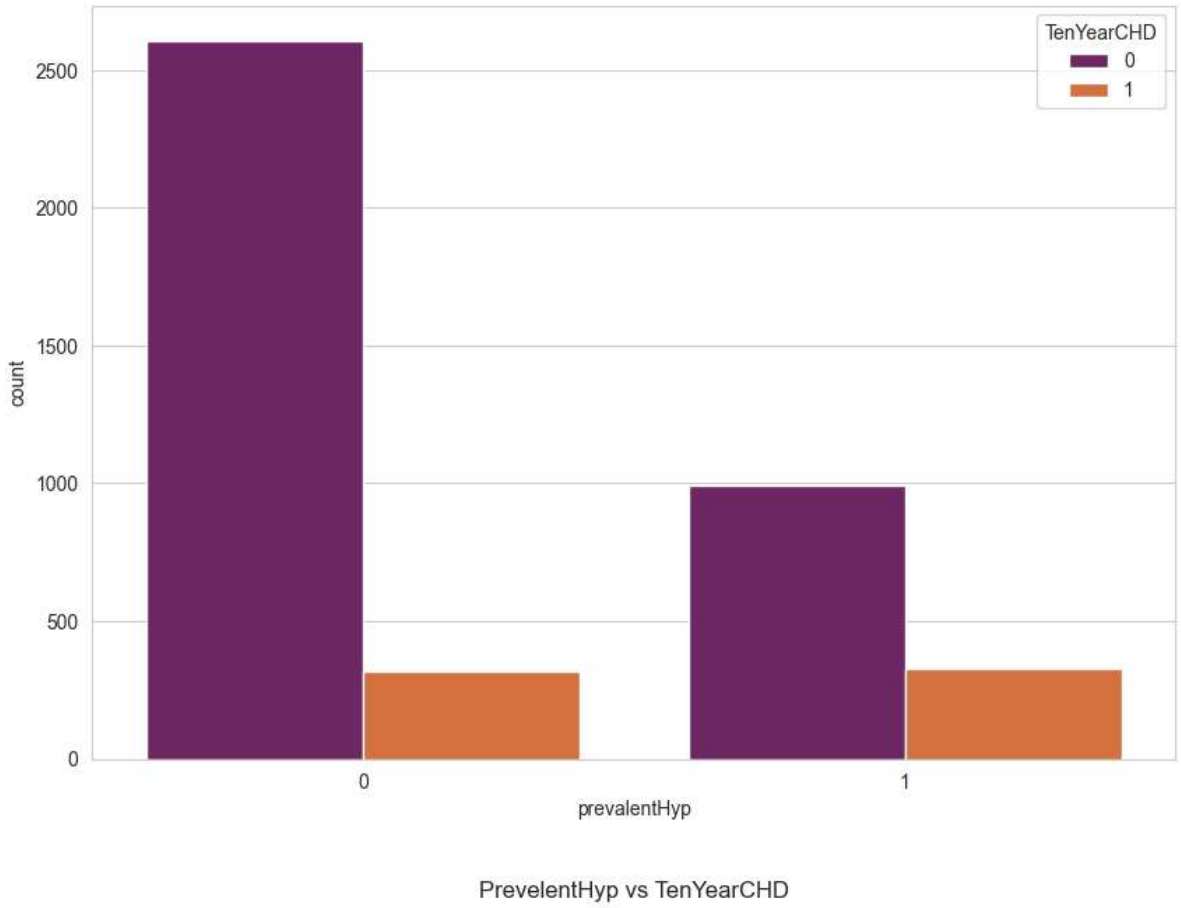
Out[369]:
```
Text(0.5, -0.2, 'PreveletStroke vs TenYearCHD')
```



PreveletStroke vs TenYearCHD

In [370…
```
plt.figure(figsize=(10,7))
sns.set_style('whitegrid')
sns.countplot(x = df.prevalentHyp ,hue=df.TenYearCHD , palette = 'inferno')
plt.title("PrevelentHyp vs TenYearCHD",y=-0.2)
```

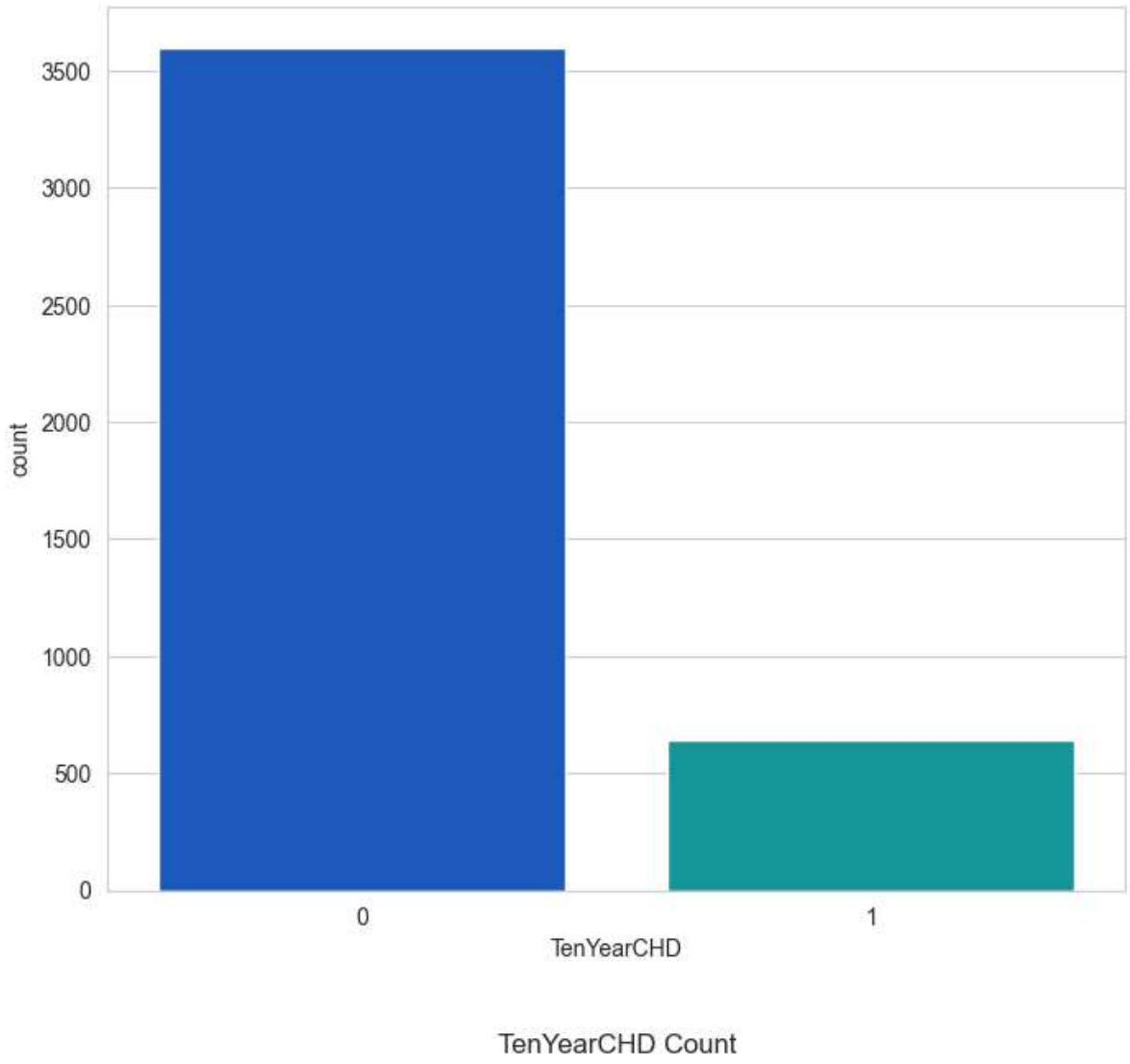
Out[370]:
```
Text(0.5, -0.2, 'PrevelentHyp vs TenYearCHD')
```



PrevelentHyp vs TenYearCHD

In [371…

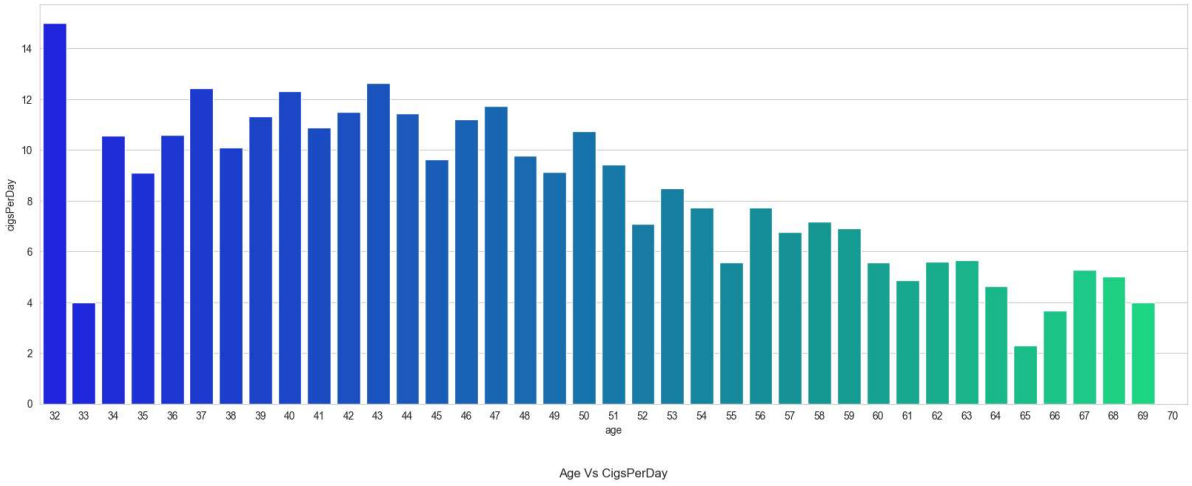
```
#TenYearCHD Count
plt.figure(figsize=(8,7))
sns.set_style('whitegrid')
sns.countplot(x = 'TenYearCHD', data = df, palette = 'winter')
plt.title("TenYearCHD Count",y=-0.2)
plt.show()
```



TenYearCHD Count

In [372…
```
# sns.set_style('whitegrid')
# sns.countplot(x = 'TenYearCHD', hue = 'male', data = df, palette = 'winter')
```

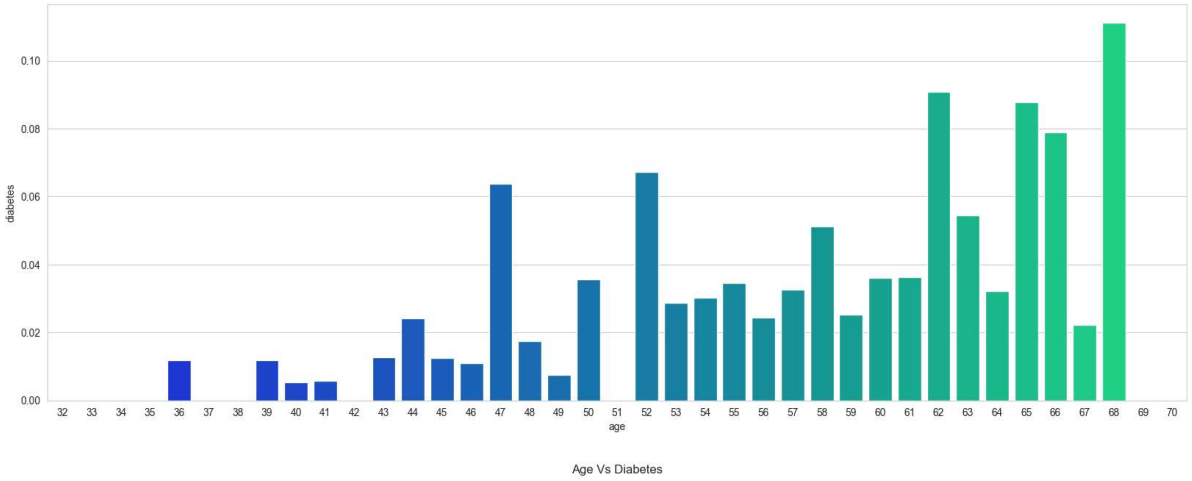
In [373…
```
#Age Vs CigsPerDay
plt.figure(figsize=(20,7))
sns.barplot(x= 'age', y = 'cigsPerDay', data = df, palette = 'winter', errorbar=Nor
plt.title("Age Vs CigsPerDay",y=-0.2)
plt.show()
```



Age Vs CigsPerDay

In [374…
```
#Age vs diabetes
plt.figure(figsize=(20,7))
```

```
sns.barplot(x= 'age', y = 'diabetes', data = df, palette = 'winter', errorbar=None)
plt.title("Age Vs Diabetes",y=-0.2)
plt.show()
```



end

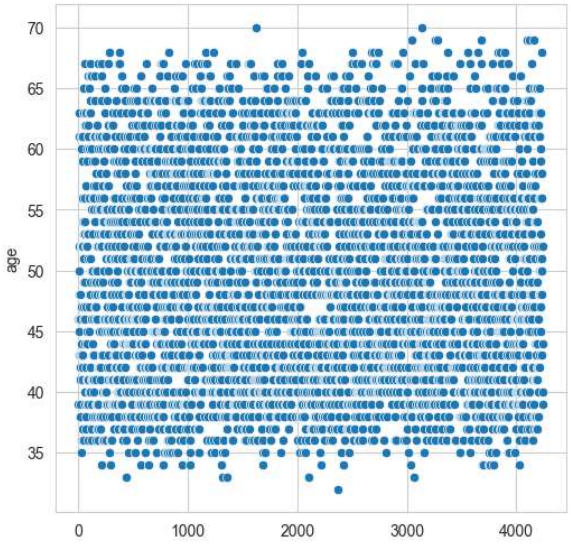
Removing Outliers

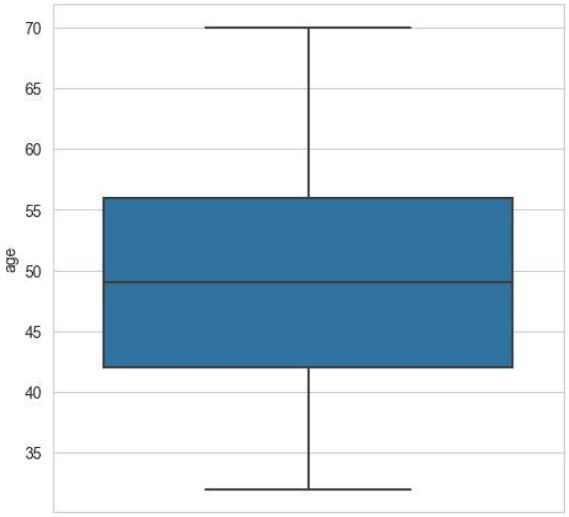
In [375…
```
#Checking heads before removing outliers
df.head()
```

Out[375]:

	male	age	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	dia
0	1	39	0.0	0.0	0	0	0	195.0	106.0	7(
1	0	46	0.0	0.0	0	0	0	250.0	121.0	8
2	1	48	20.0	0.0	0	0	0	245.0	127.5	8(
3	0	61	30.0	0.0	0	1	0	225.0	150.0	9:
4	0	46	23.0	0.0	0	0	0	285.0	130.0	8-

In [376…
```
#Checking for outliers in Age
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.age,)
plt.subplot(3,3,2)
sns.scatterplot(df["age"])
plt.show()
```





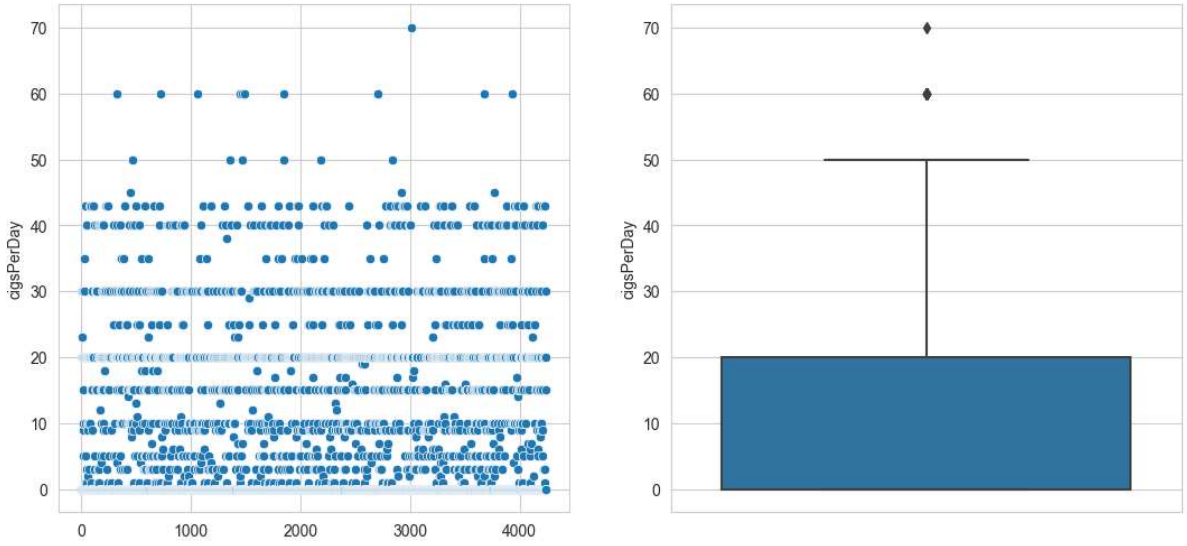
In [377…

```
#finding out upper and lower limit of age
data=np.array(df.age)
data.sort()
q1=np.percentile(data,25)
q2=np.percentile(data,50)
q3=np.percentile(data,75)
print("First Quartile =",q1)
print("Second Quartile =",q2)
print("Third Quartile =",q3)
iqr=q3-q1
print("Inter Quartile range =",iqr)
lower_lim=(q1-1.5*iqr)
upper_lim=(q3+1.5*iqr)
print("Lower Limit fot age =",lower_lim)
print("Upper Limit for age =",upper_lim)
```

```
First Quartile = 42.0
Second Quartile = 49.0
Third Quartile = 56.0
Inter Quartile range = 14.0
Lower Limit fot age = 21.0
Upper Limit for age = 77.0
```

In [378…

```
#Scatter and boxplot to check for outliers in CigsPerDay
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.cigsPerDay)
plt.subplot(3,3,2)
sns.scatterplot(df.cigsPerDay)

plt.show()
```

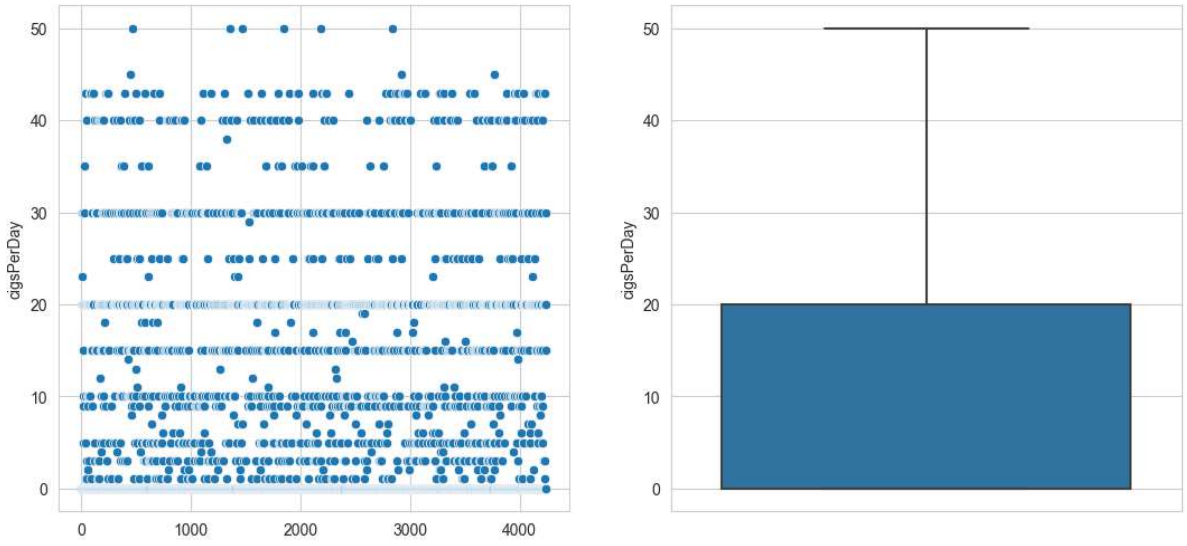


In [379…

```
#Replacing outliers in cigsPerDay witn median of the column.
tempCigsPerDay=df.cigsPerDay

for i in df.cigsPerDay:
    if(i>50):
        tempCigsPerDay.replace(i,tempCigsPerDay.median(),inplace=True)

df.cigsPerDay=tempCigsPerDay
```

In [380…

```
#Scatter and barplot to check the outliers.
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.cigsPerDay)
plt.subplot(3,3,2)
sns.scatterplot(df.cigsPerDay)

plt.show()
```



In [381…

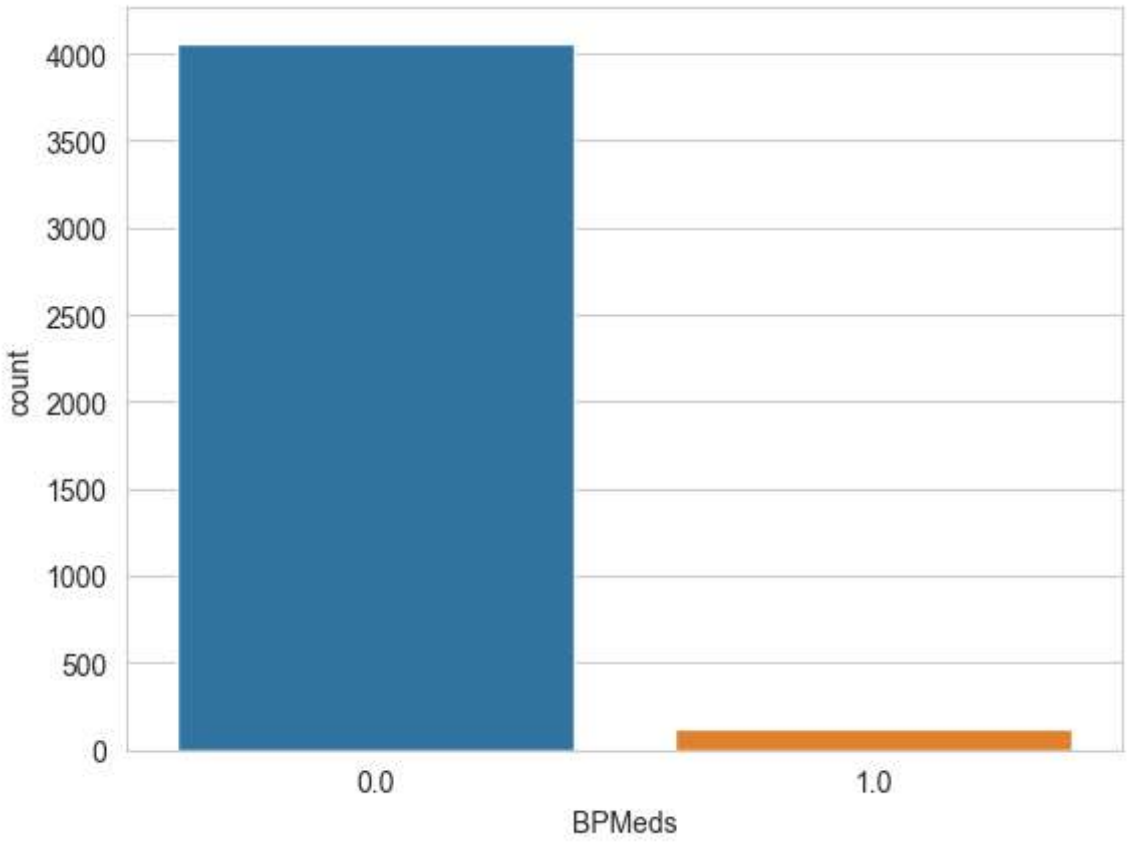
```
#Finding out upper and lower limit of cigsPerDay
data=np.array(df.cigsPerDay)
data.sort()
q1=np.percentile(data,25)
q2=np.percentile(data,50)
q3=np.percentile(data,75)
print("First Quartile =",q1)
print("Second Quartile =",q2)
```

```
print("Third Quartile =",q3)
iqr=q3-q1
print("Inter Quartile range =",iqr)
lower_lim=(q1-1.5*iqr)
upper_lim=(q3+1.5*iqr)
print("Lower Limit fot cigsPerDay =",lower_lim)
print("Upper Limit for cigsPerDay =",upper_lim)
```

```
First Quartile = nan
Second Quartile = nan
Third Quartile = nan
Inter Quartile range = nan
Lower Limit fot cigsPerDay = nan
Upper Limit for cigsPerDay = nan
```

In [382…

```
#Count of BPMeds
sns.countplot(x=df.BPMeds)
plt.show()
```



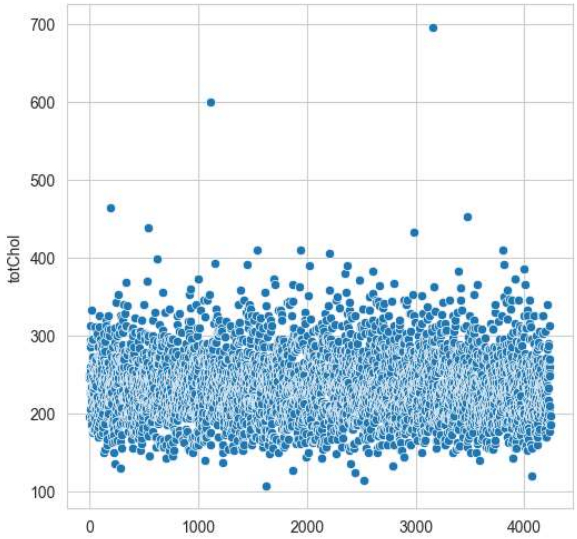
In [383…

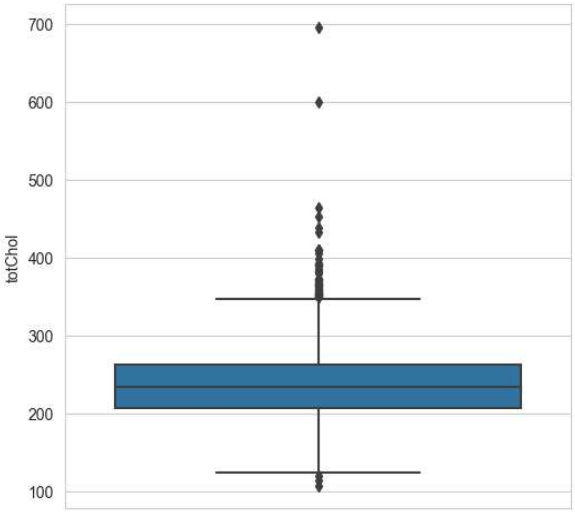
```
df[df.BPMeds!=0].BPMeds.count()
```

Out[383]: 124

In [384…

```
#scatter and box plot to check the outliers in totChol.
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.totChol)
plt.subplot(3,3,2)
sns.scatterplot(df.totChol)
plt.show()
```



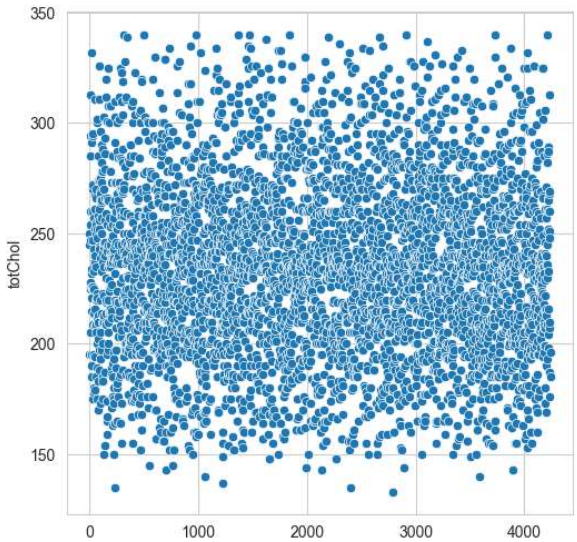


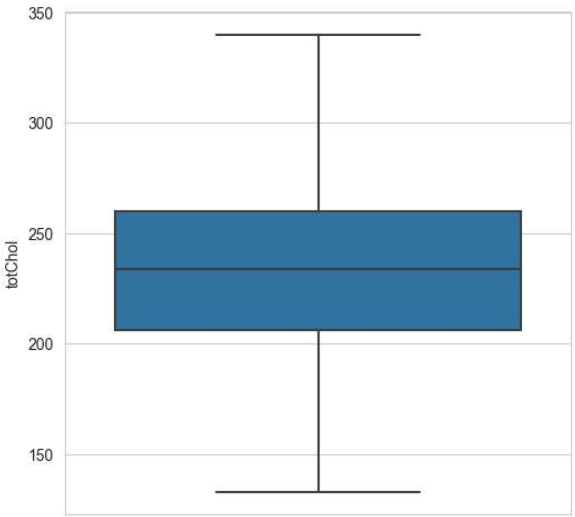
In [385…

```
#replacing outliers in totChol with mean.
temp2=df.totChol
j=0
for i in df.totChol:
    j=j+1
    if(i>340 or i<130):
        temp2.replace(i,temp2.mean(),inplace=True)
        # df.drop(j)
df.totChol=temp2
```

In [386…

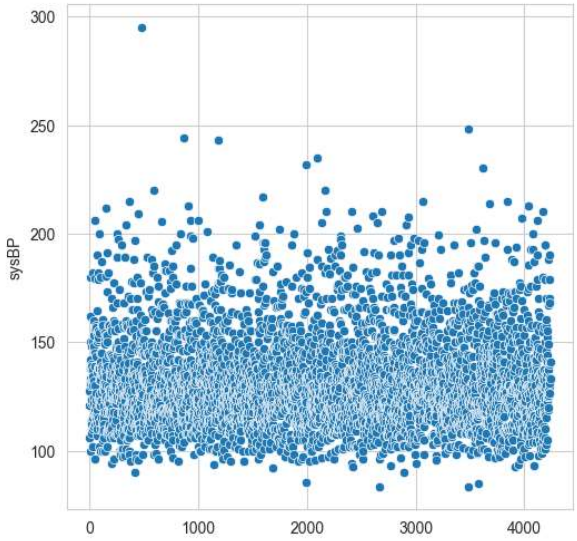
```
#Scatter and barplot after removing the outliers of totChol.
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.totChol)
plt.subplot(3,3,2)
sns.scatterplot(df.totChol)
plt.show()
```

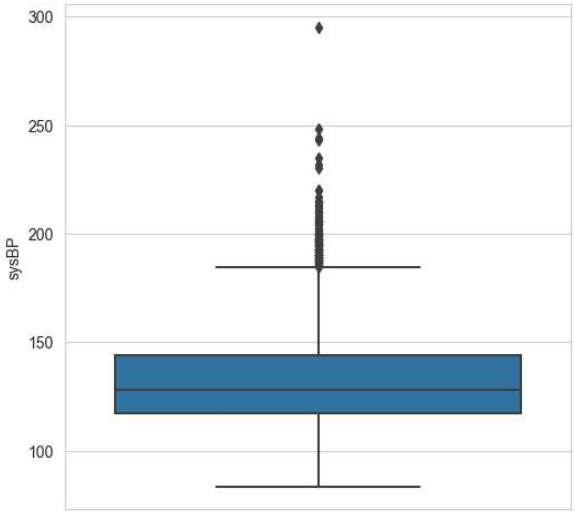




In [387…

```
# Scatter and bar plot to check outliers in SysBP
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.sysBP)
plt.subplot(3,3,2)
sns.scatterplot(df.sysBP)
plt.show()
```



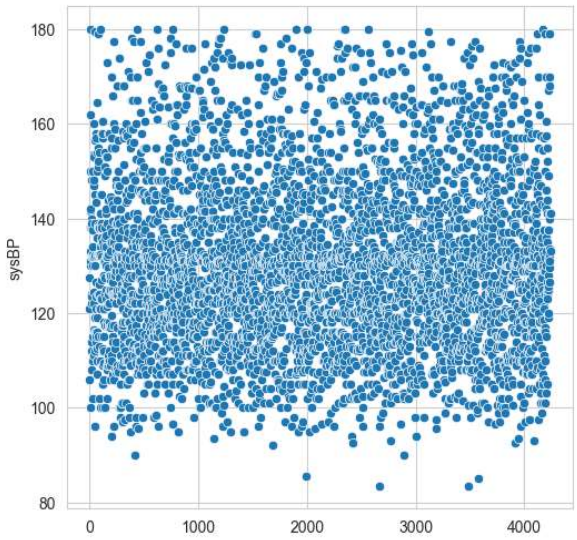


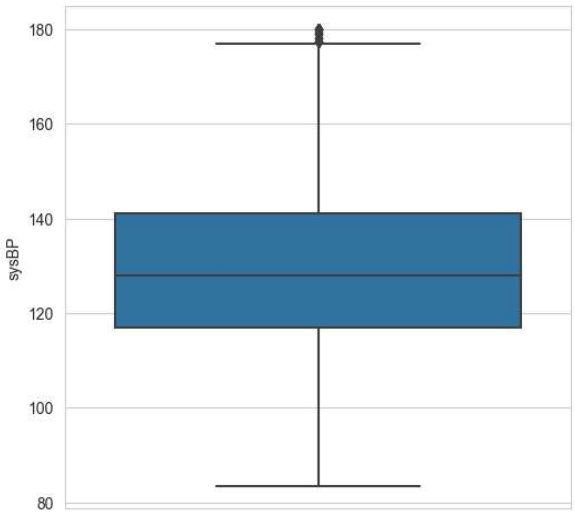
In [388…

```
#replacing outliers in sysBP with mean
temp2=df.sysBP
j=0
for i in df.sysBP:
    j=j+1
    if(i>180):
        temp2.replace(i,temp2.mean(),inplace=True)
        # df.drop(j)
df.sysBP=temp2
```

In [389…

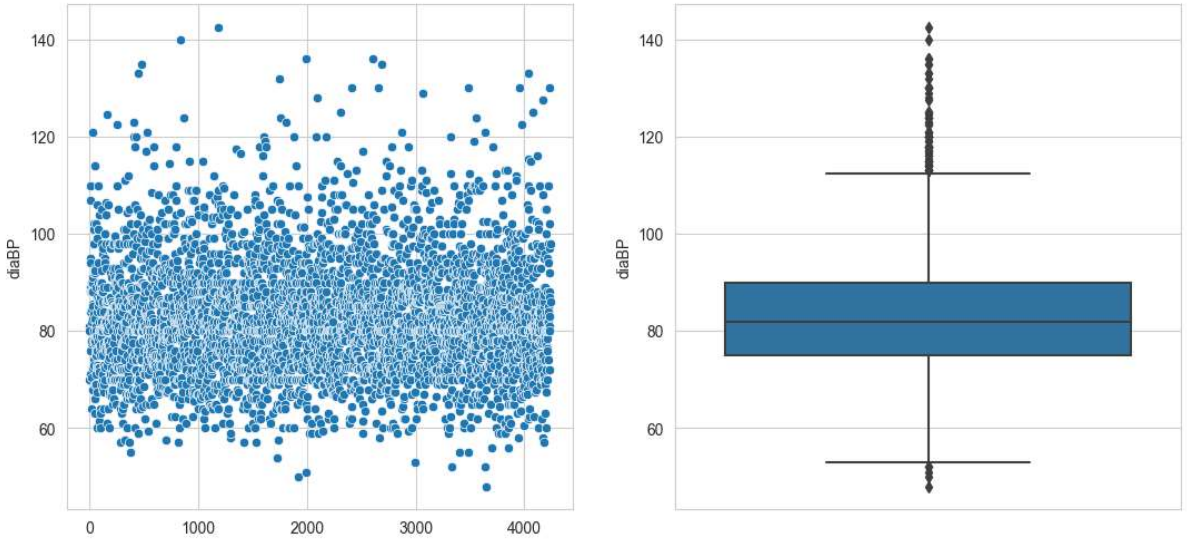
```
#Scatter and bar plot after removing outliers of sysBP
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.sysBP)
plt.subplot(3,3,2)
sns.scatterplot(df.sysBP)
plt.show()
```





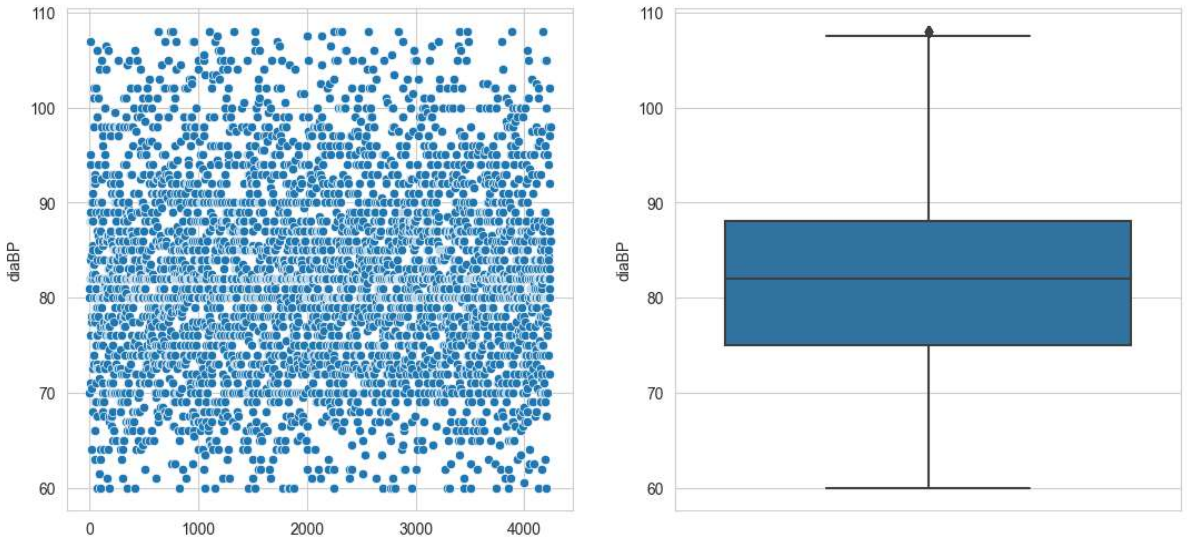
In [390…

```
#scatter and bar plot for checking outliers in diaBP
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.diaBP)
plt.subplot(3,3,2)
sns.scatterplot(df.diaBP)
plt.show()
```

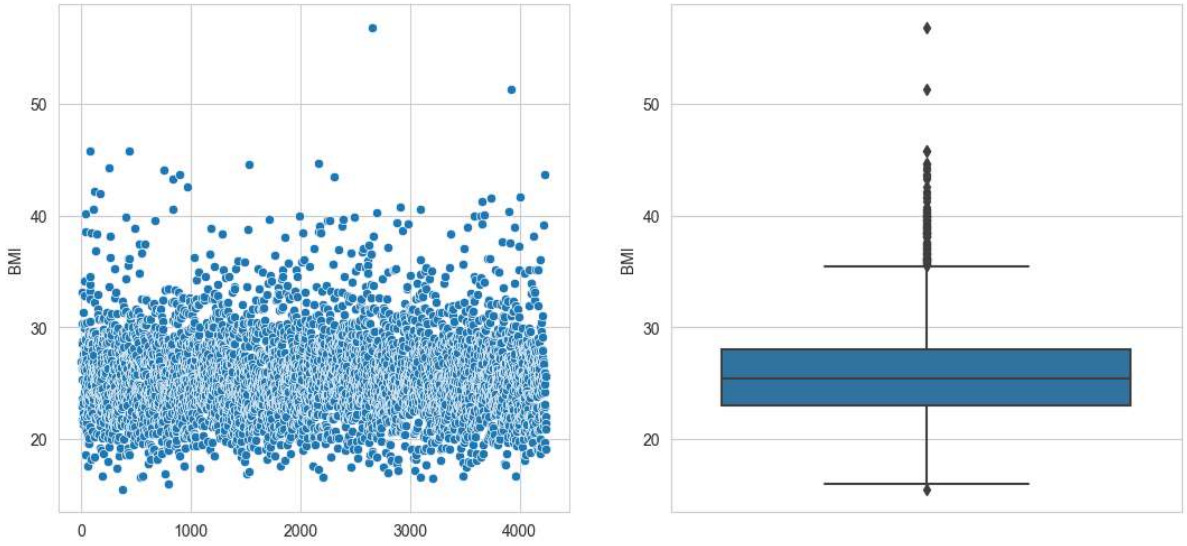


In [391…
```
#replacing outliers in diaBP with median.
tempDiaBP=df.diaBP
j=0
for i in df.diaBP:
    j=j+1
    if(i>108 or i<60):
        tempDiaBP.replace(i,tempDiaBP.median(),inplace=True)
        # df.drop(j,axis=0)
df.diaBP=tempDiaBP
```

In [392…
```
#Scatter and bar plot after removing ouliers of diaBP
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.diaBP)
plt.subplot(3,3,2)
sns.scatterplot(df.diaBP)
plt.show()
```



In [393…
```
#scatter and bar plot to check outliers in BMI
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.BMI)
plt.subplot(3,3,2)
sns.scatterplot(df.BMI)
plt.show()
```

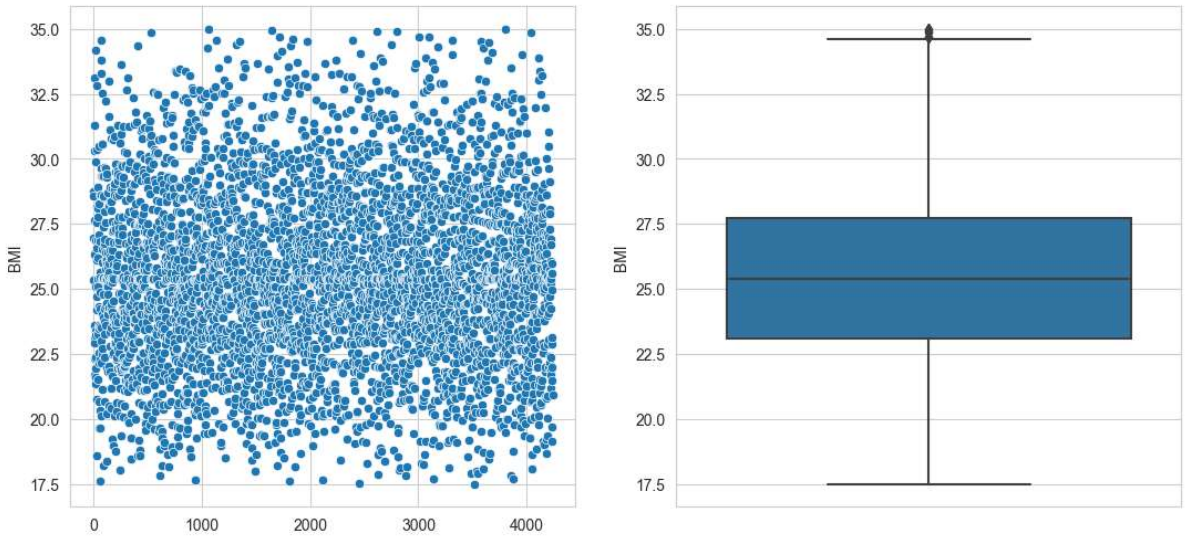


In [394…

```
#replacing outliers in BMI with median
tempBMI=df.BMI
j=0
for i in df.BMI:
    if(i>35 or i<17.5):
        j=j+1
        tempBMI.replace(i,tempBMI.median(),inplace=True)
        # df.drop(j)
df.BMI=tempBMI
# df
```

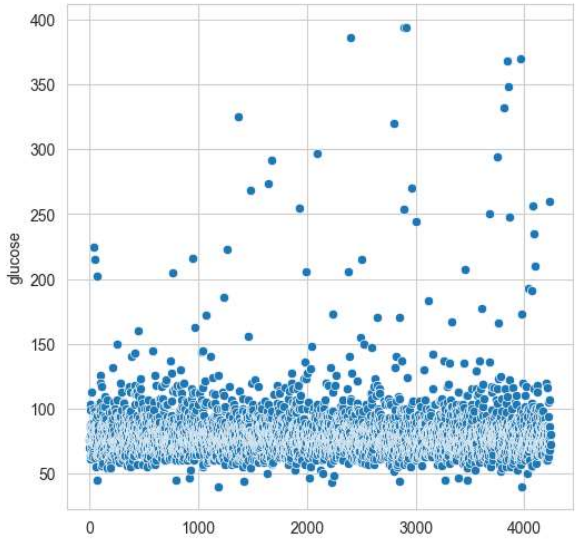
In [395…

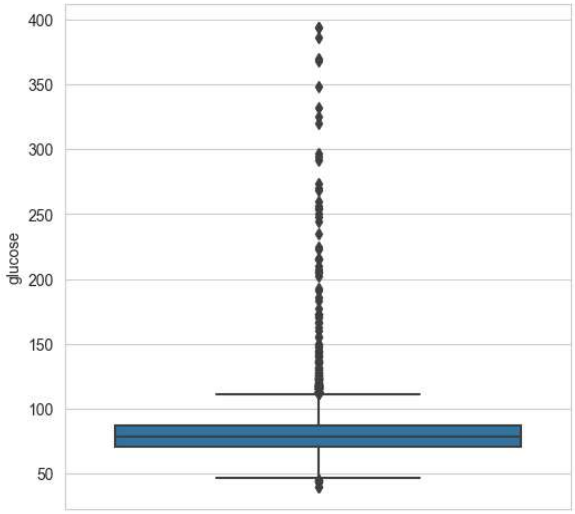
```
#Scatter and bar plot after removing outlers in BMI
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.BMI)
plt.subplot(3,3,2)
sns.scatterplot(df.BMI)
plt.show()
```



In [396…

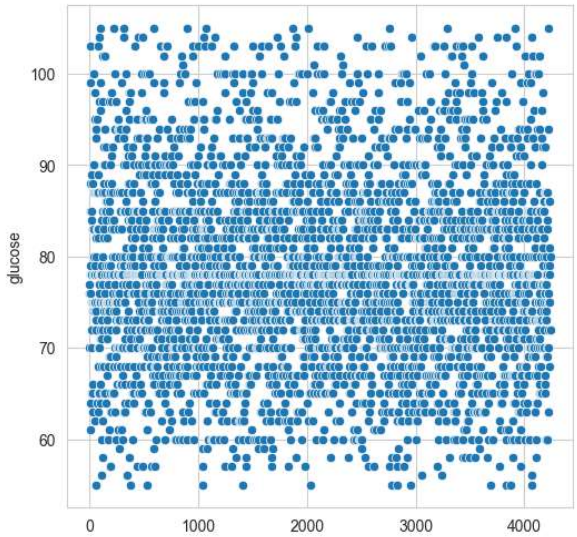
```
#Scatter and bar plot to check outlers in glucose.
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.glucose)
plt.subplot(3,3,2)
sns.scatterplot(df.glucose)
plt.show()
```

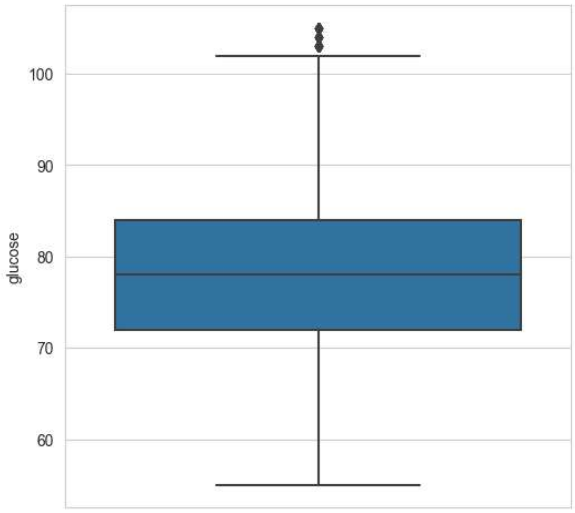




In [397…
```
#Replacing ouliers in glucose with median.
tempGlucose=df.glucose
for i in df.glucose:
    if(i>105 or i<55):
        tempGlucose.replace(i,tempGlucose.median(),inplace=True)
df.glucose=tempGlucose
```

In [398…
```
#Scatter and bar plot after removing outliers in glucose.
plt.figure(figsize=(20,20))
plt.subplot(3,3,3)
sns.boxplot(y=df.glucose)
plt.subplot(3,3,2)
sns.scatterplot(df.glucose)
plt.show()
```





Null Value Removal

In [399…
```
#Checking for Null values.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   male             4240 non-null   int64  
 1   age              4240 non-null   int64  
 2   cigsPerDay       4211 non-null   float64
 3   BPMeds           4187 non-null   float64
 4   prevalentStroke  4240 non-null   int64  
 5   prevalentHyp     4240 non-null   int64  
 6   diabetes         4240 non-null   int64  
 7   totChol          4190 non-null   float64
 8   sysBP            4240 non-null   float64
 9   diaBP            4240 non-null   float64
 10  BMI              4221 non-null   float64
 11  glucose          3852 non-null   float64
 12  TenYearCHD       4240 non-null   int64  
dtypes: float64(7), int64(6)
memory usage: 430.8 KB
```

In [400…

```
#Counting number of Null values in each column
df.isna().sum()
```

Out[400]:

```
male                0
age                 0
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
glucose           388
TenYearCHD          0
dtype: int64
```

In [401…

```
# z= df.education.mode()
# print(z)
# df.education.replace(to_replace=np.nan, value = z[0],inplace=True )
# print(df.loc[131]['cigsPerDay'])
```

In [402…

```
#Replacing null values in cigsPerDay with mode.
z= df.cigsPerDay.mode()
print(z)
df.cigsPerDay.replace(to_replace=np.nan, value = z[0],inplace=True )
print(df.loc[131]['cigsPerDay'])
```

```
0    0.0
Name: cigsPerDay, dtype: float64
0.0
```

In [403…

```
#Null values in cigsPerDay has been removed.
df.isna().sum()
```

Out[403]:
```
male                0
age                 0
cigsPerDay          0
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
glucose           388
TenYearCHD          0
dtype: int64
```

In [404…
```
#Replacing Null values in BPMeds with mode.
y= df.BPMeds.mode()
print(y)
df.BPMeds.replace(to_replace=np.nan, value = y[0],inplace=True )
```

```
0    0.0
Name: BPMeds, dtype: float64
```

In [405…
```
#Null values in BPMeds has been removed.
df.isna().sum()
```

Out[405]:
```
male                0
age                 0
cigsPerDay          0
BPMeds              0
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
glucose           388
TenYearCHD          0
dtype: int64
```

In [406…
```
#Replacing Null values in totChol with mode.
x=df.totChol.mode()
print(x)
df.totChol.replace(to_replace=np.nan, value= x[0], inplace= True)
```

```
0    240.0
Name: totChol, dtype: float64
```

In [407…
```
#Null values in totChol has been removed
df.isna().sum()
```

Out[407]:
```
male                0
age                 0
cigsPerDay          0
BPMeds              0
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol             0
sysBP               0
diaBP               0
BMI                19
glucose           388
TenYearCHD          0
dtype: int64
```

In [408…
```
#Replacing Null values in BMI with mode.
y=df.BMI.mode()
print(y)
df.BMI.replace(to_replace=np.nan, value=y[0], inplace=True)
```

```
0    25.4
Name: BMI, dtype: float64
```

In [409…
```
#Null values in BMI has been removed.
df.isna().sum()
```

Out[409]:
```
male                0
age                 0
cigsPerDay          0
BPMeds              0
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol             0
sysBP               0
diaBP               0
BMI                 0
glucose           388
TenYearCHD          0
dtype: int64
```

In [410…
```
#Replacing Null values in glucose with mode.
z=df.glucose.mode()
print(z)
df.glucose.replace(to_replace=np.nan, value=z[0], inplace=True)
```

```
0    78.0
Name: glucose, dtype: float64
```

In [411…
```
#Null values in glucose has been removed.
df.isna().sum()
```

Out[411]:
```
male               0
age                0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
glucose            0
TenYearCHD         0
dtype: int64
```

In [412…
```
# z=df.heartRate.mode()
# print(z)
# df.heartRate.replace(to_replace=np.nan, value=z[0], inplace=True)
```

In [413…
```
#All Null values have been removed.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   male             4240 non-null   int64  
 1   age              4240 non-null   int64  
 2   cigsPerDay       4240 non-null   float64
 3   BPMeds           4240 non-null   float64
 4   prevalentStroke  4240 non-null   int64  
 5   prevalentHyp     4240 non-null   int64  
 6   diabetes         4240 non-null   int64  
 7   totChol          4240 non-null   float64
 8   sysBP            4240 non-null   float64
 9   diaBP            4240 non-null   float64
 10  BMI              4240 non-null   float64
 11  glucose          4240 non-null   float64
 12  TenYearCHD       4240 non-null   int64  
dtypes: float64(7), int64(6)
memory usage: 430.8 KB
```

Train Test Split

In [414…
```
#Splitting dataset in to train dataset and test dataset using SKlearn train test sp
X_train,X_test,y_train,y_test=train_test_split(df.drop('TenYearCHD',axis=1),df['Ten
print("No. of Train rows -> ",y_train.shape, X_train.shape)
print("No. of Test rows -> ", y_test.shape, X_test.shape)
```

```
No. of Train rows ->  (2968,) (2968, 12)
No. of Test rows ->  (1272,) (1272, 12)
```

In [415…
```
#X training dataset
X_train
```

Out[415]:

	male	age	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP
829	0	53	0.0	0.0	0	0	0	240.0	133.5
3342	0	63	0.0	0.0	0	0	0	297.0	133.5
1821	0	66	0.0	0.0	0	1	0	275.0	132.5
2215	1	60	20.0	0.0	0	0	0	260.0	127.5
49	1	36	20.0	0.0	0	1	0	194.0	139.0
...	...	...	...	...	...	...	...	...	...
1146	1	57	30.0	0.0	0	0	0	225.0	140.0
1259	0	43	0.0	0.0	0	0	0	204.0	132.0
3264	0	51	2.0	0.0	0	0	0	261.0	127.0
399	1	66	0.0	0.0	0	1	0	276.0	159.0
2532	0	41	0.0	0.0	0	1	0	317.0	149.5

2968 rows × 12 columns

In [416…

```
#Y Taining Dataset.
y_train
```

Out[416]:

```
829     0
3342    0
1821    1
2215    1
49      0
       ..
1146    0
1259    0
3264    0
399     0
2532    0
Name: TenYearCHD, Length: 2968, dtype: int64
```

In [417…

```
#X Testing Dataset
X_test
```

Out[417]:

	male	age	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sy
3756	0	46	15.0	0.0	0	1	0	274.0	158.000
2145	1	62	0.0	0.0	0	0	0	202.0	149.500
30	1	36	35.0	0.0	0	0	0	295.0	102.000
1685	0	53	0.0	0.0	0	0	0	225.0	92.000
1983	0	63	0.0	0.0	0	1	0	283.0	164.000
...	...	...	...	...	...	...	...	...	
4121	0	55	0.0	0.0	0	0	0	240.0	145.000
1637	0	36	5.0	0.0	0	0	0	200.0	121.500
720	0	39	20.0	0.0	0	0	0	240.0	120.000
4006	1	40	20.0	0.0	0	0	0	242.0	115.000
3729	1	63	0.0	0.0	0	1	0	161.0	130.575

1272 rows × 12 columns

In [418…

```
#Y testing Dataset
y_test
```

Out[418]:

```
3756    0
2145    0
30      0
1685    0
1983    0
       ..
4121    0
1637    0
720     0
4006    0
3729    0
Name: TenYearCHD, Length: 1272, dtype: int64
```

In [419…

```
#Standarinsing the X Test
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)

# y_train_std = scaler.fit_transform(np.reshape(y_train,(1,-1)))
# print ("X Train Transform",X_train_std,"Y Train Transform",y_train_std)
# print(type(y_train_std[0][0]))
```

In [420…

```
print(X_train_std)
X_train_std=pd.DataFrame(X_train_std,columns=X_train.columns)
X_train_std
```

```
[[-0.85004523  0.39779183 -0.76665135 ...  0.03309737 -0.03265677
  -0.01435914]
 [-0.85004523  1.57046381 -0.76665135 ...  0.98730342 -0.12558896
  -0.43841805]
 [-0.85004523  1.9222654  -0.76665135 ...  0.28420423  2.65338352
  -0.01435914]
 ...
 [-0.85004523  0.16325743 -0.59417296 ... -0.11756674 -1.57952818
   1.89390598]
 [ 1.17640798  1.9222654  -0.76665135 ... -0.017124    1.77202655
  -1.39255061]
 [-0.85004523 -1.00941454 -0.76665135 ...  1.08774616 -0.03265677
   0.93977342]]
```

Out[420]:

	male	age	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	tot
0	-0.850045	0.397792	-0.766651	-0.181842	-0.082367	-0.674993	-0.157676	0.13
1	-0.850045	1.570464	-0.766651	-0.181842	-0.082367	-0.674993	-0.157676	1.57
2	-0.850045	1.922265	-0.766651	-0.181842	-0.082367	1.481497	-0.157676	1.01
3	1.176408	1.218662	0.958133	-0.181842	-0.082367	-0.674993	-0.157676	0.63
4	1.176408	-1.595751	0.958133	-0.181842	-0.082367	1.481497	-0.157676	-1.03
...	...	...	...	...	...	...	...	
2963	1.176408	0.866861	1.820525	-0.181842	-0.082367	-0.674993	-0.157676	-0.24
2964	-0.850045	-0.774880	-0.766651	-0.181842	-0.082367	-0.674993	-0.157676	-0.77
2965	-0.850045	0.163257	-0.594173	-0.181842	-0.082367	-0.674993	-0.157676	0.66
2966	1.176408	1.922265	-0.766651	-0.181842	-0.082367	1.481497	-0.157676	1.04
2967	-0.850045	-1.009415	-0.766651	-0.181842	-0.082367	1.481497	-0.157676	2.08

2968 rows × 12 columns

Logistic Regression

In [421…
```
#Importing Logistic regression
from sklearn.linear_model import LogisticRegression
```

In [422…
```
#Initializing Logistic Regression
lreg_model = LogisticRegression(max_iter=10000)
lreg_model
```

Out[422]:

LogisticRegression

LogisticRegression(max_iter=10000)

In [423…
```
#Training Logistic Regression.
lreg_trained_model=lreg_model.fit(X_train, y_train)
```

In [424…
```
#predicting values with X_test.
predictions = lreg_trained_model.predict(X_test)
```

```
print (predictions.shape)
predictions
```

```
(1272,)
```

Out[424]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [425…
```
#importing classicfication report
from sklearn.metrics import classification_report
```

In [426…
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      1073
           1       0.78      0.07      0.13       199

    accuracy                           0.85      1272
   macro avg       0.82      0.53      0.52      1272
weighted avg       0.84      0.85      0.80      1272
```

In [427…
```
#importing confusion matrix and accuracy score and printing the confusion matrix.
from sklearn.metrics import confusion_matrix,accuracy_score
confusion_df = pd.DataFrame(confusion_matrix(y_test, predictions))
print ('Accuracy Score :', accuracy_score(y_test, predictions))
confusion_df
```

```
Accuracy Score : 0.8514150943396226
```

Out[427]:

	0	1
0	1069	4
1	185	14

In [428…
```
#Importing GridSearchCV to find the best parameters.
from sklearn.model_selection import GridSearchCV
```

In [429…
```
##Finding the best parameters using GridSearchCV
#parameters=[{"penalty":["l1","elasticnet",'l2',],"dual":[True,False],"C":[1.0,2.0,
# grid_search=GridSearchCV(estimator=lreg_model,param_grid=parameters,scoring='accu
# grid_search=grid_search.fit(X_train_std,y_train)
```

In [430…
```
##Accuracy using GridSearchCV
# accuracy=grid_search.best_score_
# accuracy
```

In [431…
```
#Printing The Best parameters found by GridSearchCV
#grid_search.best_params_
```

In [432…
```
##Logistic Regression with class_weight
lreg_cw = LogisticRegression(solver='liblinear', class_weight='balanced',dual= True
lreg_trained_cw=lreg_cw.fit(X_train, y_train)
predictions = lreg_trained_cw.predict(X_test)
print (predictions.shape)
predictions
```

```
(1272,)
```

```
c:\Users\Subhodip\AppData\Local\Programs\Python\Python310\lib\site-packages\sklear
n\svm\_base.py:1225: ConvergenceWarning: Liblinear failed to converge, increase th
e number of iterations.
  warnings.warn(
```

Out[432]:
```
array([0, 1, 0, ..., 0, 0, 1], dtype=int64)
```

In [433…
```
#Printing classification report of Logistic Regression after using parameters found
print(classification_report(y_test,predictions))
lr_accuracy_score=accuracy_score(y_test, predictions)
print ('Accuracy Score :',lr_accuracy_score)
```

```
              precision    recall  f1-score   support

           0       0.90      0.73      0.81      1073
           1       0.28      0.56      0.37       199

    accuracy                           0.70      1272
   macro avg       0.59      0.65      0.59      1272
weighted avg       0.80      0.70      0.74      1272

Accuracy Score : 0.7028301886792453
```

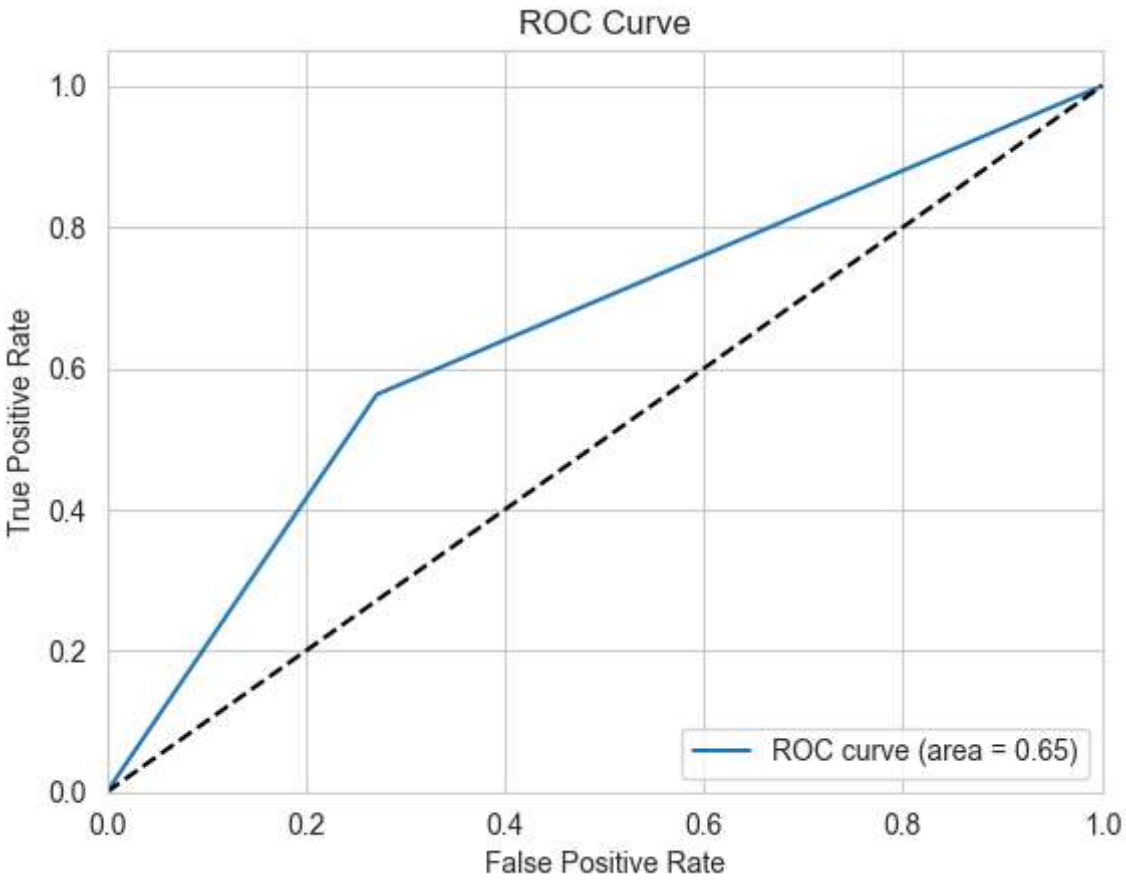
In [434…
```
#Confusion matrix of Logistic regression after using the parameters found by GridSe
from sklearn.metrics import confusion_matrix, accuracy_score
confusion_df = pd.DataFrame(confusion_matrix(y_test, predictions))
print ('Accuracy Score :', accuracy_score(y_test, predictions))
confusion_df
```

```
Accuracy Score : 0.7028301886792453
```

Out[434]:

	0	1
0	782	291
1	87	112

In [435…
```
#ROC Curve of Logistic Regression.
from sklearn.metrics import roc_curve,auc
fpr,tpr,_=roc_curve(y_test,predictions)
roc_auc=auc(fpr,tpr)
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



Random Forest

In [436…
```
# X_train,X_test,y_train,y_test=train_test_split(df.drop('TenYearCHD',axis=1),df['T
# print("No. of Train rows -> ",y_train.shape, X_train.shape)
# print("No. of Test rows -> ", y_test.shape, X_test.shape)
```

In [437…
```
#Importing Random Forest.
from sklearn.ensemble import RandomForestClassifier
```

In [438…
```
#Fitting Random forest classifier.
clf = RandomForestClassifier(n_jobs=2, random_state=0,n_estimators=10)
clf.fit(X_train, y_train)
```

Out[438]:
```
RandomForestClassifier(n_estimators=10, n_jobs=2, random_state=0)
```

In [439…
```
#Testing Random Forest Classifier.
predictions=clf.predict(X_test)
print (predictions.shape)
predictions
```

```
(1272,)
```

Out[439]:
```
array([0, 1, 0, ..., 0, 0, 0], dtype=int64)
```

In [440…
```
#Classification report of Random Forest.
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.85      0.98      0.91      1073
           1       0.31      0.05      0.08       199

    accuracy                           0.83      1272
   macro avg       0.58      0.51      0.49      1272
weighted avg       0.76      0.83      0.78      1272
```

In [441…
```
#Confusion matrix of random forest classifier.
from sklearn.metrics import confusion_matrix
confusion_df = pd.DataFrame(confusion_matrix(y_test, predictions))
print("Percentage of 0 Predicted correctly = ",(confusion_df[0][0]/(confusion_df[0]
print("Percentage of 1 Predicted correctly = ",(confusion_df[1][1]/(confusion_df[1]
confusion_df
```

```
Percentage of 0 Predicted correctly =  98.13606710158435
Percentage of 1 Predicted correctly =  4.522613065326634
```

Out[441]:

	0	1
0	1053	20
1	190	9

In [442…
```
# View a list of the features and their importance scores
list(zip(X_train, clf.feature_importances_))
```

Out[442]:
```
[('male', 0.02545478371143646),
 ('age', 0.13687209051287266),
 ('cigsPerDay', 0.07228464848659573),
 ('BPMeds', 0.007769191432079697),
 ('prevalentStroke', 0.0027158825547201264),
 ('prevalentHyp', 0.02798745314647995),
 ('diabetes', 0.013342161832965285),
 ('totChol', 0.1556181050991587),
 ('sysBP', 0.15254591278150137),
 ('diaBP', 0.13131539905173154),
 ('BMI', 0.15102884930568222),
 ('glucose', 0.1230655220847761)]
```

In [443…
```
##Importing Balanced Random forest classifier.
from imblearn.ensemble import BalancedRandomForestClassifier
```

In [444…
```
#Training Balanced Random Forest Classifier.
clf_balanced=BalancedRandomForestClassifier(max_depth=2, random_state=0)
clf_balanced.fit(X_train,y_train)
```

Out[444]:
```
BalancedRandomForestClassifier(max_depth=2, random_state=0)
```

In [445…
```
#Testing Random Forest Classifier.
predictions=clf_balanced.predict(X_test)
print (predictions.shape)
predictions
```

```
(1272,)
```

Out[445]:
```
array([1, 1, 0, ..., 0, 0, 1], dtype=int64)
```

In [446…
```
#Printing classification report of Balanced Random Forest Classifier.
print(classification_report(y_test,predictions))
rf_accuracy_score=accuracy_score(y_test, predictions)
print ('Accuracy Score :',rf_accuracy_score)
```

```
              precision    recall  f1-score   support

           0       0.90      0.66      0.76      1073
           1       0.24      0.59      0.35       199

    accuracy                           0.65      1272
   macro avg       0.57      0.63      0.55      1272
weighted avg       0.79      0.65      0.69      1272

Accuracy Score : 0.6477987421383647
```

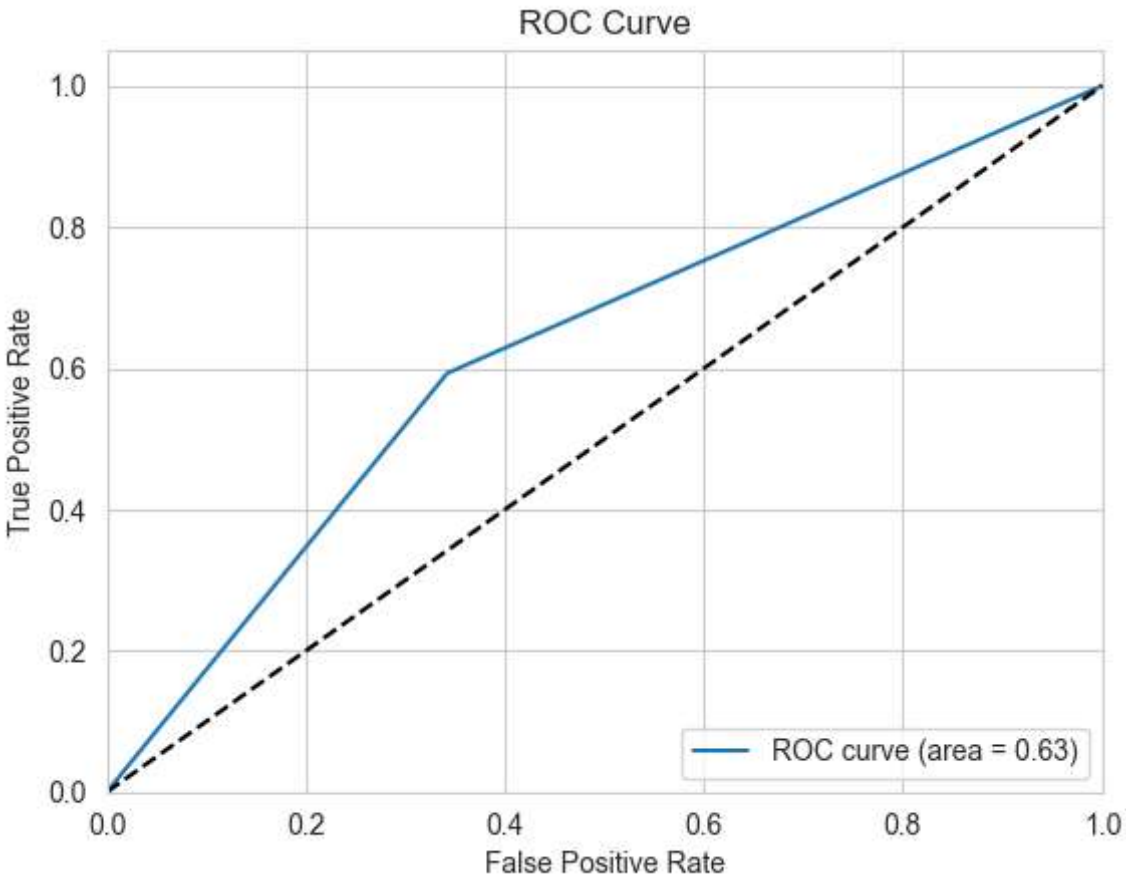
In [447…
```
#Printing confusion matrix of Balanced Random Forest Classifier.
from sklearn.metrics import confusion_matrix
cfm=confusion_matrix(y_test,predictions)
confusion_df = pd.DataFrame(confusion_matrix(y_test, predictions))
print("Percentage of 0 Predicted correctly = ",(confusion_df[0][0]/(confusion_df[0]
print("Percentage of 1 Predicted correctly = ",(confusion_df[1][1]/(confusion_df[1]
confusion_df
```

```
Percentage of 0 Predicted correctly =  65.79683131407269
Percentage of 1 Predicted correctly =  59.2964824120603
```

Out[447]:

	0	1
0	706	367
1	81	118

In [448…
```
#Roc curve for balanced random forest classifier.
from sklearn.metrics import roc_curve,auc
fpr,tpr,_=roc_curve(y_test,predictions)
roc_auc=auc(fpr,tpr)
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



K-Nearest Neighbour

In [449…
```
#Importing KNN classifier.
from sklearn.neighbors import KNeighborsClassifier
```

In [450…
```
# X_train,X_test,y_train,y_test=train_test_split(df.drop('TenYearCHD',axis=1),df['T
# print("No. of Train rows -> ",y_train.shape, X_train.shape)
# print("No. of Test rows -> ", y_test.shape, X_test.shape)
# X_train_std=scaler.fit_transform(X_train)
# print(X_train)
```

In [451…
```
#Initializing KNN Classifier.
knn = KNeighborsClassifier(n_neighbors=5, metric='manhattan', n_jobs=-1)
```

In [452…
```
# parameters=[{"n_neighbors":[1,2,3,4,5],"weights":["uniform","distance"],"algorith
# grid_search=GridSearchCV(estimator=knn,param_grid=parameters,scoring='accuracy',c
# grid_search=grid_search.fit(X_train_std,y_train)
```

In [453…
```
# accuracy=grid_search.best_score_
# accuracy
```

In [454…
```
# grid_search.best_params_
```

In [455…
```
#Training KNN Classifier.
knn.fit(X_train_std, y_train)
```

Out[455]:

KNeighborsClassifier

```
KNeighborsClassifier(metric='manhattan', n_jobs=-1)
```

In [456…

```
#Testing KNN classifier.
X_test_std = scaler.fit_transform(X_test)
predictions=knn.predict(X_test_std)
print (predictions.shape)
predictions
```

```
(1272,)
c:\Users\Subhodip\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
  warnings.warn(
```

Out[456]:

```
array([0, 0, 0, ..., 0, 0, 1], dtype=int64)
```

In [457…

```
from sklearn.metrics import classification_report
```

In [458…

```
#Printing classification report of KNN
print(classification_report(y_test,predictions))
knn_accuracy_score=accuracy_score(y_test, predictions)
print ('Accuracy Score :',rf_accuracy_score)
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      1073
           1       0.26      0.09      0.13       199

    accuracy                           0.82      1272
   macro avg       0.55      0.52      0.52      1272
weighted avg       0.76      0.82      0.78      1272

Accuracy Score : 0.6477987421383647
```

In [459…

```
#Printing confusion matrix of KNN
from sklearn.metrics import confusion_matrix
confusion_df = pd.DataFrame(confusion_matrix(y_test, predictions))
confusion_df
```

Out[459]:

	0	1
0	1021	52
1	181	18

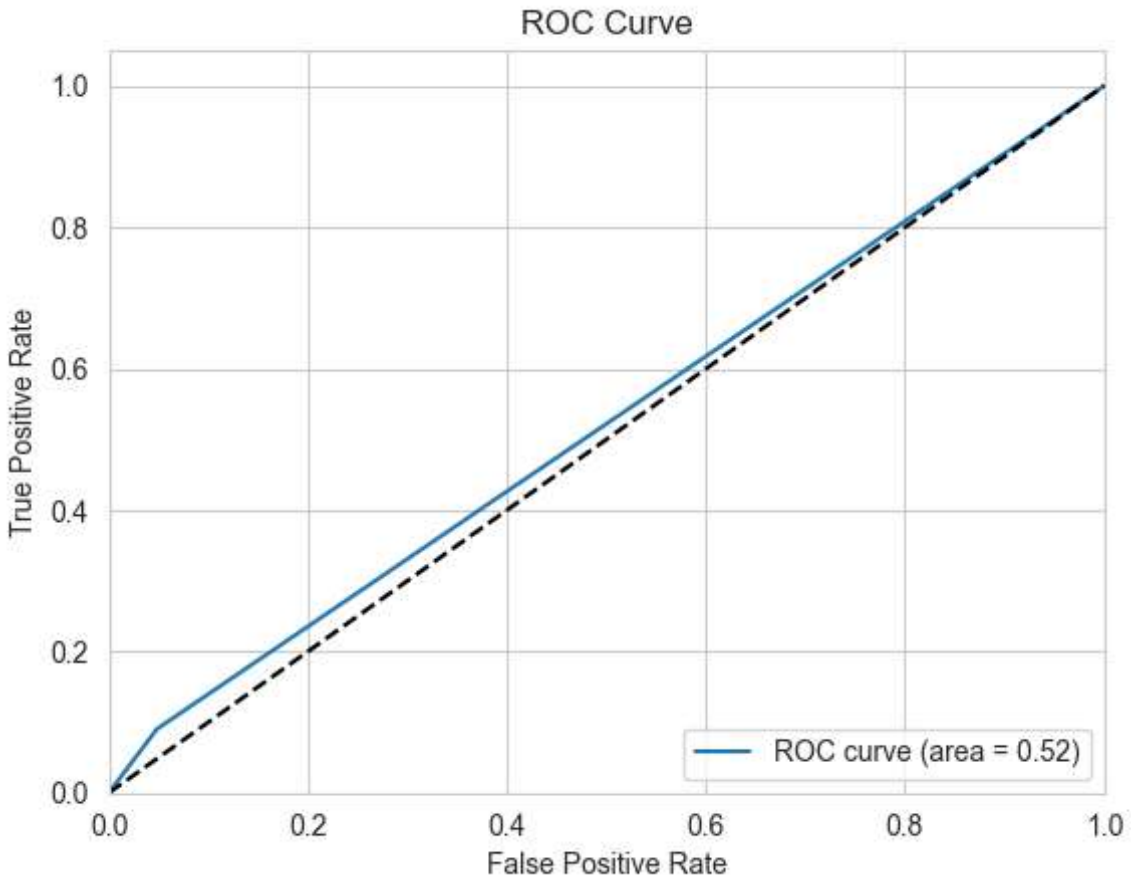
In [460…

```
j=0
k=0
for i in y_train:
    if i==0:
        j=j+1
    else:
        k=k+1
print("No. of 1 =",k)
print("Percentage of 1 =",(k/len(y_train))*100)
print("No. of 0 =",j)
print("Percentage of 0 =",j/len(y_train)*100)
```

```
No. of 1 = 445
Percentage of 1 = 14.993261455525605
No. of 0 = 2523
Percentage of 0 = 85.00673854447439
```

In [461…

```
#Roc of KNN
from sklearn.metrics import roc_curve,auc
fpr,tpr,_=roc_curve(y_test,predictions)
roc_auc=auc(fpr,tpr)
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



Decision Tree

In [462…

```
#Importing decision tree
from sklearn.tree import DecisionTreeClassifier
```

In [463…

```
# X_train,X_test,y_train,y_test=train_test_split(df.drop('TenYearCHD',axis=1),df['T
# print("No. of Train rows -> ",y_train.shape, X_train.shape)
# print("No. of Test rows -> ", y_test.shape, X_test.shape)
# X_train_std=scaler.fit_transform(X_train)
# print(X_train)
```

In [464…
```
#initializing decision tree.
dt_train_gini = DecisionTreeClassifier()
```

In [465…
```
# parameters=[{"criterion":["gini", "entropy", "log_loss"],"splitter":["best","rand
# grid_search=GridSearchCV(estimator=dt_train_gini,param_grid=parameters,scoring='a
# grid_search=grid_search.fit(X_train_std,y_train)
```

In [466…
```
# accuracy=grid_search.best_score_
# accuracy
```

In [467…
```
# grid_search.best_params_
```

In [468…
```
#Initializing decision tree with best parameters
dt_train_gini=DecisionTreeClassifier(criterion='entropy',max_depth=10,max_features=
```

In [469…
```
#training decision tree.
dt_trained=dt_train_gini.fit(X_train, y_train)
```

In [470…
```
#testing decision tree.
predictions=dt_trained.predict(X_test)
predictions
```

Out[470]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [471…
```
print(dt_train_gini.predict_proba(X_train)[:10])
```

```
[[0.89568345 0.10431655]
 [0.89568345 0.10431655]
 [0.5        0.5       ]
 [0.81308411 0.18691589]
 [0.8902439  0.1097561 ]
 [0.875      0.125     ]
 [0.75510204 0.24489796]
 [0.95744681 0.04255319]
 [0.89568345 0.10431655]
 [0.55555556 0.44444444]]
```

In [472…
```
print(dt_train_gini.predict_proba(X_train)[-10:])
```

```
[[0.73015873 0.26984127]
 [0.94       0.06      ]
 [1.         0.        ]
 [0.65384615 0.34615385]
 [0.8        0.2       ]
 [0.85       0.15      ]
 [0.95394737 0.04605263]
 [0.89568345 0.10431655]
 [0.73015873 0.26984127]
 [0.94       0.06      ]]
```

In [473…
```
#Printing classification report pf Decision tree.
print(classification_report(y_test,predictions))
dct_accuracy_score=accuracy_score(y_test, predictions)
print ('Accuracy Score :',rf_accuracy_score)
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.91      1073
           1       0.29      0.03      0.05       199

    accuracy                           0.84      1272
   macro avg       0.57      0.51      0.48      1272
weighted avg       0.76      0.84      0.78      1272

Accuracy Score : 0.6477987421383647
```

In [474…

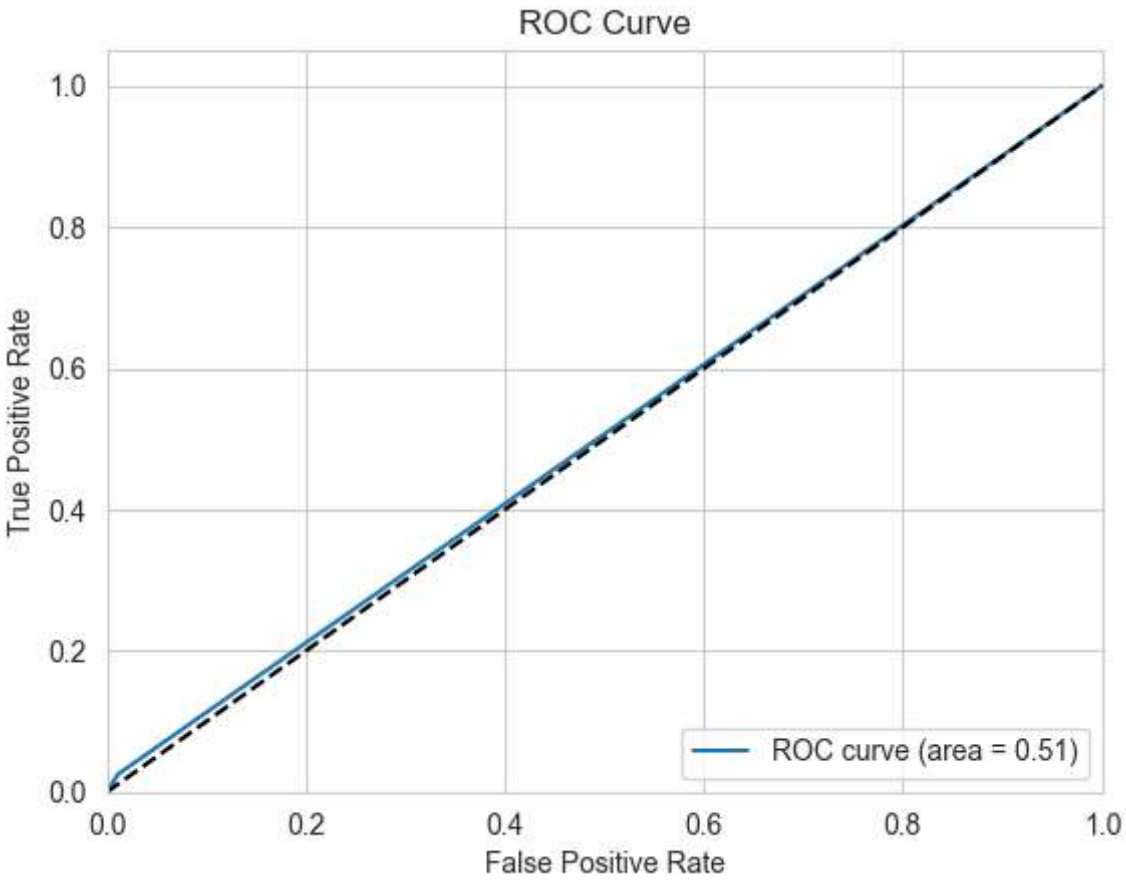
```
#printing confusion matrix for decision tree
from sklearn.metrics import confusion_matrix
confusion_df = pd.DataFrame(confusion_matrix(y_test, predictions))
confusion_df
```

Out[474]:

	0	1
0	1061	12
1	194	5

In [475…

```
#ROC Curve for decision tree.
from sklearn.metrics import roc_curve,auc
fpr,tpr,_=roc_curve(y_test,predictions)
roc_auc=auc(fpr,tpr)
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



Comparision Of Models

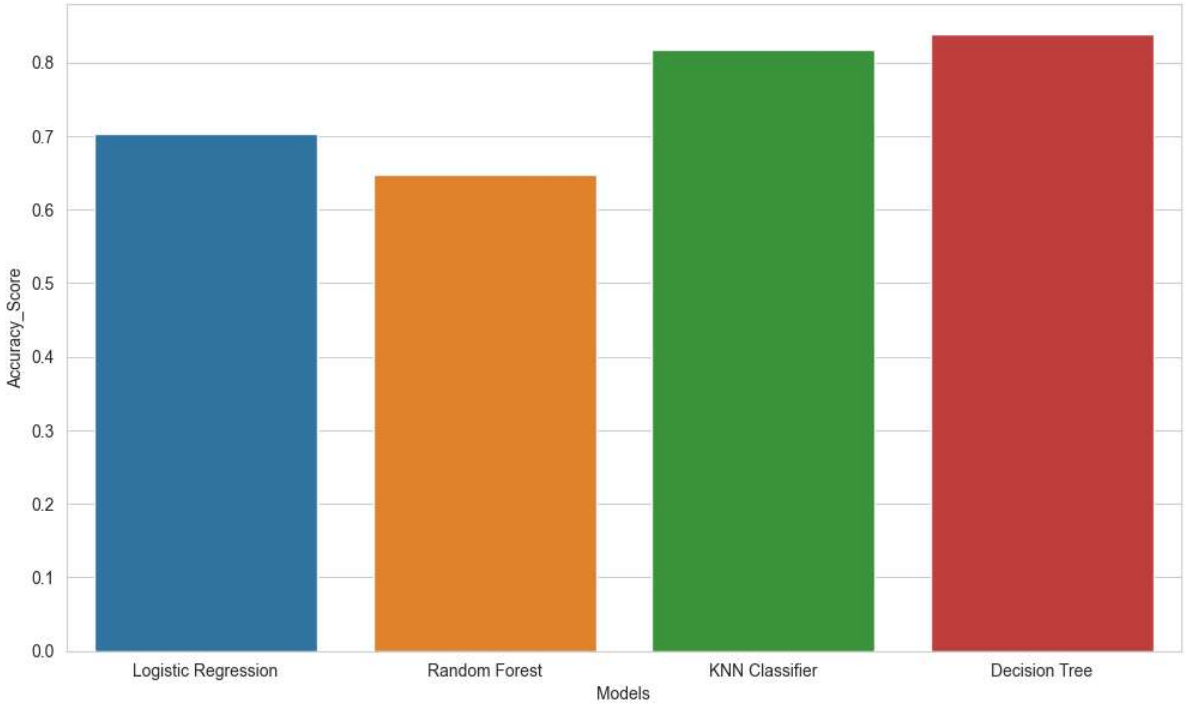
In [476…
```
#Printing accuracy scores.
accuracy_score_dict={"Models":['Logistic Regression','Random Forest','KNN Classifie
accuracy_score_df=pd.DataFrame.from_dict(accuracy_score_dict)
accuracy_score_df
```

Out[476]:

	Models	Accuracy_Score
0	Logistic Regression	0.702830
1	Random Forest	0.647799
2	KNN Classifier	0.816824
3	Decision Tree	0.838050

In [477…
```
#Accuracy comparision graph.
plt.figure(figsize=(12,7))
sns.barplot(x = 'Models' , y = 'Accuracy_Score' , data = accuracy_score_df )
```

Out[477]: <AxesSubplot:xlabel='Models', ylabel='Accuracy_Score'>


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Accuracy_Score
Logistic Regression
Random Forest
KNN Classifier
Decision Tree
Models

Future Scope

- User who are intended to check their chances of heart disease in recent 10 years can use our trained model to check whether they will have a coronary heart disease or not. This trained models can be implemented on graphical user interface to make it easy to use.

- In the future, the work could be improved by creating a web application premised on the logistic regression algorithm and by using a larger dataset than the one used in this study, which would help to provide better outcomes and aid health professionals in predicting heart disease efficiently and effectively.

- Various hospital can use this model and modify them according to there needs.To predict the heart health of the patient

- In future an intelligent system may be developed that can lead to selection of proper treatment methods for a patient diagnosed with heart disease. A lot of work has been done already in making models that can predict whether a patient is likely to develop heart disease or not. There are several treatment methods for a patient once diagnosed with a particular form of heart disease. Data mining can be of very good help in deciding the line of treatment to be followed by extracting knowledge from such suitable databases.

Certificate of Appreciation

Subhodip Roy

This is to certify that Mr. SUBHODIP ROY of Asansol Engineering , registration number: 201080100110153, has successfully completed a project on HEART DEASEAS PREDICTATION using machine learning with python project was developed under the guidance of Prof. ARNAB CHAKRABORTY

Mr. Subhodip Roy
Student

Prof. Arnab Chakraborty
Mentor

Certificate of Appreciation

Drippta Dutta

This is to certify that Mr. DRIPPTA DUTTA of Asansol Engineering , registration number: 201080100110135, has successfully completed a project on HEART DEASEAS PREDICTATION using machine learning with python project was developed under the guidance of Prof. ARNAB CHAKRABORTY

Mr. Drippta Dutta
Student

Prof. Arnab Chakraborty
Mentor

Certificate of Appreciation

Snehasis Roy

This is to certify that Mr. SNEHASIS ROY of Asansol Engineering , registration number: 201080100110163 , has successfully completed a project on HEART DEASEAS PREDICTATION using machine learning with python project was developed under the guidance of Prof. ARNAB CHAKRABORTY

Mr. Snehasis Roy
Student

Prof. Arnab Chakraborty
Mentor

Certificate of Appreciation

Moumita Maji

This is to certify that Ms. MOUMITA MAJI of Asansol Engineering , registration number: 201080100110143 , has successfully completed a project on HEART DEASEAS PREDICTATION using machine learning with python project was developed under the guidance of Prof. ARNAB CHAKRABORTY

Ms. Moumita Maji
Student

Prof. Arnab Chakraborty
Mentor

Certificate of Appreciation

Arpita Maji

This is to certify that Ms. ARPITA MAJI of Asansol Engineering , registration number: 201080100110141 , has successfully completed a project on HEART DEASEAS PREDICTATION using machine learning with python project was developed under the guidance of
Prof. ARNAB CHAKRABORTY

Ms. Arpita Maji
Student

Prof. Arnab Chakraborty
Mentor