RISK FACTOR PREDICTION OF CARDIO-VASCULAR HEART DISEASE USING MACHINE LEARNING IN PYTHON

STUDENT MEMBERS:-

AGNISIS DUTTA

10800320053

ECE 3RD YEAR

SHILPI SEN

10800320070

ECE 3RD YEAR

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.

AGNISIS DUTTA

SHILPI SEN

PROJECT OVERVIEW

Cardiovascular diseases (CVDs) are a group of disorders of the heart and blood vessels and include coronary heart disease, cerebrovascular disease, rheumatic heart disease and other conditions. High blood pressure, high blood cholesterol, and smoking are key risk factors for heart disease. There were more than 523.2 million cases of cardiovascular disease in 2019, an increase of 26.6% compared with 2010.
The dataset is from an ongoing cardiovascular study on residents of the town of Framingham, Massachusetts. The dataset provides the patient's information. It includes over 4,000 records and 15 attributes.

We have trained the machine learning model using these dataset and understood the pattern from data. The model helps us for risk prediction of coronary heart disease in 10 year. With good accuracy rate we can able to predict the possibility of CVD.

PROJECT OBJECTIVE

The classification goal is to predict whether the patient has a 10-year risk of future coronary heart disease (CHD) considering the other features.
The objectives of the Framingham Study are to study the incidence and prevalence of cardiovascular disease (CVD) and its risk factors, trends in CVD incidence and its risk factors over time, and familial patterns of CVD and risk factors. Other important objectives include the estimation of incidence rates of disease and description of the natural history of cardiovascular disease, including the sequence of clinical signs and systems that precede the clinically recognizable syndrome and the consequences and course of clinically manifest disease.

Key Facts:

- Cardiovascular diseases (CVDs) are the leading cause of death globally, taking an estimated 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of these deaths, 85% were due to heart attack and stroke.
- More than four out of five CVD deaths are due to heart attacks and strokes, and one third of these deaths occur prematurely in people under 70 years of age.
- The most important behavioural risk factors of heart disease and stroke are unhealthy diet, physical inactivity, tobacco use and harmful use of alcohol.
- Over three quarters of CVD deaths take place in low- and middle-income countries.

In [1]:
```
pip install sklearn
```

```
Requirement already satisfied: sklearn in c:\users\agnisis\appdata\local\progra
ms\python\python310\lib\site-packages (0.0.post1)
Note: you may need to restart the kernel to use updated packages.
```

```
[notice] A new release of pip is available: 23.0 -> 23.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [78]:
```
#Importing data manipulation libraries.
import pandas as pd
import numpy as np

#Importing data visualization libraries.
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

Loading the Data

In this section, we will be importing our data and perform some usual data analysis on it later on.

In [79]:
```
#loading dataset
df = pd.read_csv('framingham_CVD_dataset.csv')

#Showing the dataframe.
df
```

Out[79]:

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

4240 rows × 16 columns

Variables

__Each attribute is a potential risk factor. There are both demographic, behavioral, and medical risk factors.__

Data Description

Demographic:

• *Sex:** male or female*

• *Age:** Age of the patient;(Continuous - Although the recorded ages have been truncated to whole numbers, the concept of age is continuous)*

Behavioral:

• *is_smoking:** whether or not the patient is a current smoker*

• *Cigs Per Day:** the number of cigarettes that the person smoked on average in one day. (can be considered continuous as one can have any number of cigarettes, even half a cigarette.)*

Medical(history):

• *BP Meds:** whether or not the patient was on blood pressure medication (Nominal)*

• *Prevalent Stroke:** whether or not the patient had previously had a stroke (Nominal)*

• *Prevalent Hyp:** whether or not the patient was hypertensive (Nominal)*

• *Diabetes:** whether or not the patient had diabetes (Nominal)*

Medical(current):

• *Tot Chol:** total cholesterol level (Continuous)*

• *Sys BP:** systolic blood pressure (Continuous)*

• *Dia BP:** diastolic blood pressure (Continuous)*

• *BMI:** Body Mass Index (Continuous)*

• *Heart Rate:** heart rate (Continuous - In medical research, variables such as heart rate though in fact discrete, yet are considered continuous because of large number of possible values.)*

• *Glucose:** glucose level (Continuous)*

Predict variable (desired target):

__• 10-year risk of coronary heart disease CHD(binary: “1”, means “Yes” , “0” means “No”) - DEPENDANT VARIABLE__

Data Inspection

Here, we will be performing basic and initial analysis on our raw data to check the data inside of each provided column, check the data type of each column and shape of our data.

In [80]:
```
#Checking the first 5 rows.
df.head()
```

Out[80]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	d
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	

In [81]:
```
#Checking the last 5 rows.
df.tail()
```

Out[81]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHy
4235	0	48	2.0	1	20.0	NaN	0	
4236	0	44	1.0	1	15.0	0.0	0	
4237	0	52	2.0	0	0.0	0.0	0	
4238	1	40	3.0	0	0.0	0.0	0	
4239	0	39	3.0	1	30.0	0.0	0	

In [82]:
```
#Checking the shape of the data.
df.shape
```

Out[82]:
```
(4240, 16)
```

In [83]:
```
#Checking for the title of all the columns.
df.columns
```

Out[83]:
```
Index(['sex', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
      dtype='object')
```

In [84]:
```
#Updated Dataset after renaming all the columns.
df.head(2)
```

Out[84]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	d
0	1	39	4.0	0	0.0	0.0	0	0	
1	0	46	2.0	0	0.0	0.0	0	0	

In [85]:

```
#Checking for null entries and data type of each column.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4240 entries, 0 to 4239
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   sex              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
```

It is seen that there are 4240 rows and 16 columns.

*There are some NULL values** present in **"education","cigsPerDay","BPMeds","totChol","BMI","heartRate"and "glucose"**.* *There are 9 float type** columns and **7 int type** columns **"10yearCHD" is our Target or Dependant variable.**

Statistical Analysis

In [86]:

```
#Statistical description of all the columns present in the dataset.
df.describe(include='all').T
```

Out[86]:

	count	mean	std	min	25%	50%	75%	max
sex	4240.0	0.429245	0.495027	0.00	0.00	0.0	1.00	1.0
age	4240.0	49.580189	8.572942	32.00	42.00	49.0	56.00	70.0
education	4135.0	1.979444	1.019791	1.00	1.00	2.0	3.00	4.0
currentSmoker	4240.0	0.494104	0.500024	0.00	0.00	0.0	1.00	1.0
cigsPerDay	4211.0	9.005937	11.922462	0.00	0.00	0.0	20.00	70.0
BPMeds	4187.0	0.029615	0.169544	0.00	0.00	0.0	0.00	1.0
prevalentStroke	4240.0	0.005896	0.076569	0.00	0.00	0.0	0.00	1.0
prevalentHyp	4240.0	0.310613	0.462799	0.00	0.00	0.0	1.00	1.0
diabetes	4240.0	0.025708	0.158280	0.00	0.00	0.0	0.00	1.0
totChol	4190.0	236.699523	44.591284	107.00	206.00	234.0	263.00	696.0
sysBP	4240.0	132.354599	22.033300	83.50	117.00	128.0	144.00	295.0
diaBP	4240.0	82.897759	11.910394	48.00	75.00	82.0	90.00	142.5
BMI	4221.0	25.800801	4.079840	15.54	23.07	25.4	28.04	56.8
heartRate	4239.0	75.878981	12.025348	44.00	68.00	75.0	83.00	143.0
glucose	3852.0	81.963655	23.954335	40.00	71.00	78.0	87.00	394.0
TenYearCHD	4240.0	0.151887	0.358953	0.00	0.00	0.0	0.00	1.0

Data Cleansing

Data cleansing, also referred to as data cleaning or data scrubbing, is the process of fixing incorrect, incomplete, duplicate or otherwise erroneous data in a data set. It involves identifying data errors and then changing, updating or removing data to correct them. Data cleansing improves data quality and helps provide more accurate, consistent and reliable information for decision-making in an organization.

Before proceeding further, let's rename our columns for a better understanding and better efficiency.

In [87]:

```
#Renaming the columns for a better view and understanding.
df.rename(columns={'sex':'Gender','currentSmoker':'smoking','cigsPerDay':'cigare
                    'prevalentStroke':'stroke','prevalentHyp':'hypertensive',
                    'totChol':'total_cholesterol','sysBP':'systolic_bp','diaBP':'
                    'BMI':'bmi','heartRate':'heart_rate','TenYearCHD':'10yearCHD'
          inplace = True)
```

In [88]:

```
#Checking the updated names of the columns.
df.columns
```

Out[88]:

```
Index(['Gender', 'age', 'education', 'smoking', 'cigarettes/day',
       'BP_meds', 'stroke', 'hypertensive', 'diabetes',
       'total_cholesterol', 'systolic_bp', 'diastolic_bp', 'bmi',
       'heart_rate', 'glucose', '10yearCHD'],
      dtype='object')
```

__Lets start checking the duplicates and null values with a suitable treatment afterwards.__

Duplicates:

In [89]:

```
#Checking for the duplicated entries in the dataset.
MissV = len(df[df.duplicated()])
print("There are",MissV, "duplicate values.")
```

```
There are 0 duplicate values.
```

__As we can see that there are no duplicates present in the dataset but let's check if we have any missing values.__

Missing values:

In [90]:

```
#Sum of all the null values present in each column.
for i in df.columns.tolist():
  print("Total missig values in",i,":",df[i].isna().sum())
```

```
Total missig values in Gender : 0
Total missig values in age : 0
Total missig values in education : 105
Total missig values in smoking : 0
Total missig values in cigarettes/day : 29
Total missig values in BP_meds : 53
Total missig values in stroke : 0
Total missig values in hypertensive : 0
Total missig values in diabetes : 0
Total missig values in total_cholesterol : 50
Total missig values in systolic_bp : 0
Total missig values in diastolic_bp : 0
Total missig values in bmi : 19
Total missig values in heart_rate : 1
Total missig values in glucose : 388
Total missig values in 10yearCHD : 0
```

In [91]:

```
#Sum of the null values overall.
print("Overall missing values are",df.isna().sum().sum())
```

```
Overall missing values are 645
```

We can be more clear on the exact portion, once we see a percentage distribution of null value counts and how much they'd affect our analysis.

In [92]:

```
#Calculating the percentage of NULL values in each column.
totalN = df.isna().sum().sort_values(ascending=False)
percentage = (df.isna().sum()/df.isna().count()).sort_values(ascending=False)
* missing_v = pd.concat([totalN, percentage], axis=1, keys=['Total',
'Percentage'] missing_v
```

Out[92]:

	Total	Percentage
glucose	388	9.150943
education	105	2.476415
BP_meds	53	1.250000
total_cholesterol	50	1.179245
cigarettes/day	29	0.683962
bmi	19	0.448113
heart_rate	1	0.023585
Gender	0	0.000000
age	0	0.000000
smoking	0	0.000000
stroke	0	0.000000
hypertensive	0	0.000000
diabetes	0	0.000000
systolic_bp	0	0.000000
diastolic_bp	0	0.000000
10yearCHD	0	0.000000

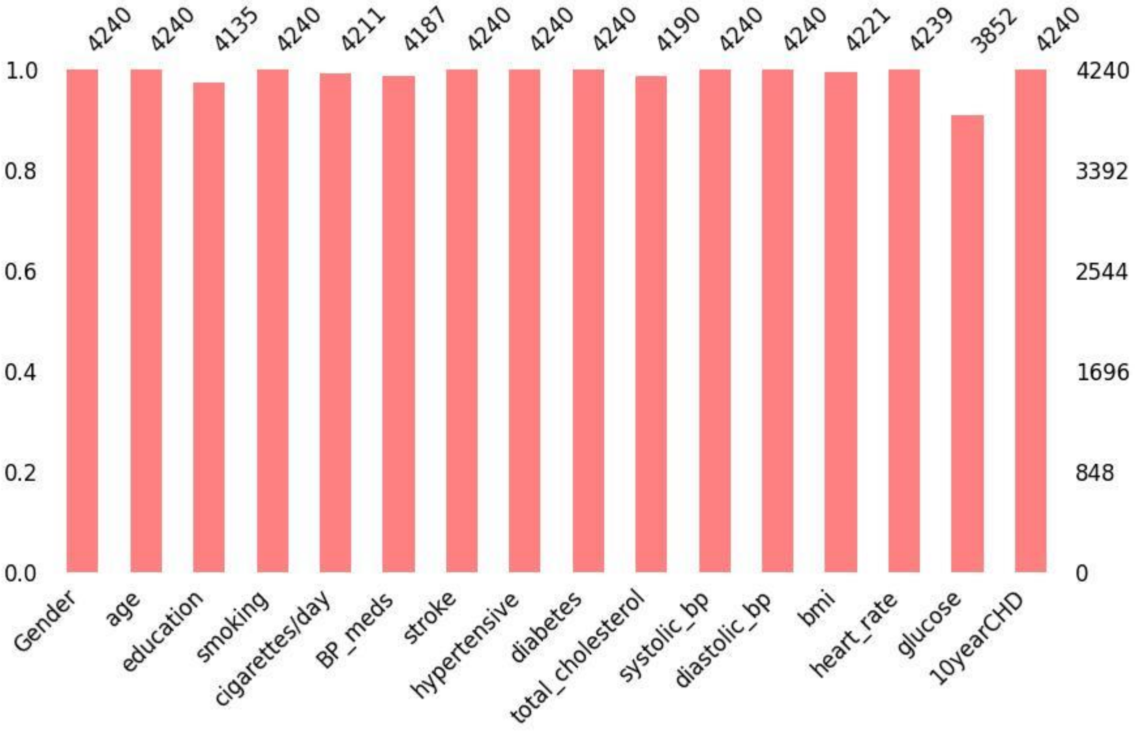
Let's Visualize the Null values for a clearer Picture.

In [93]:

```
#importing missingno for visualizing the null values.
import missingno as misno

# Visualizing the number of missing values as a bar chart
misno.bar(df,color=(1, 0.5, 0.5),figsize=(10,5), fontsize=12)
```

Out[93]:

```
<AxesSubplot: >
```



*This bar chart is clearly showing that glucose** and **education** have the most significant null values which can majorly affect our model building process and can also affect the predictions later.*

*But the major problem with these null values is that they cannot be estimated from other data inputs. This dataset is from medical domain, but the entries in this data are person-specific** and the values vary from person to person.The chances of two people sharing the same health statistic are very rare.So,we will be removing all the null values of all the features.*

It could have been imposed by using subtle techniques like KNNImputer, but that might not be as accurate because the other inputs are used to extrapolate null. What is being discussed is nevermore the ideal approach for this type of data.

Feature Engineering and EDA

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning. In order to make machine learning work well on new tasks, it might be necessary to design and train better features.

Feature engineering consists of various process like

- **Feature Creation**
- **Transformations**
- **Feature Extraction**

Feature Engineering is a very important step in machine learning. Feature engineering refers to the process of designing artificial features into an algorithm.

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.*

It’s a scientific approach to get the story of the data.It focuses more narrowly on checking assumptions required for model fitting and hypothesis testing. It also helps while handling missing values and making transformations of variables as needed.

Encoding:

We are encoding out Categorical feature values into binary values for a better model training and prediction. A Binary Data is a Data which uses two possible states or values i.e. 0 and 1.

- *The (0 and 1) also referred to as (true and false), (success and failure), (yes and no) etc.*
- *Binary Data is a discrete Data and also used in statistics.*

We don't need encoding as all values are numerical

Since, It's very clear by finding out all the unique values in the dataset that there are multiple continuous and categorical features present. we will be seperating categorical and numerical for better exploration and analysis.

Splitting the Data into Categorical and Numerical Variables

In [94]:

```
#Check Unique Values for each variable.
for i in df.columns.tolist():
  print(i,":",df[i].nunique())
```

```
Gender : 2
age : 39
education : 4
smoking : 2
cigarettes/day : 33
BP_meds : 2
stroke : 2
hypertensive : 2
diabetes : 2
total_cholesterol : 248
systolic_bp : 234
diastolic_bp : 146
bmi : 1364
heart_rate : 73
glucose : 143
10yearCHD : 2
```

Since, It's very clear by finding out all the unique values in the dataset that there are multiple continuous and categorical features present. we will be seperating categorical and numerical for better exploration and analysis.

In [95]:

```
#Extracting categorical features.

cat_features = ['education','Gender','smoking','BP_meds','stroke','hypertensive'
print(f'There are {len(cat_features)} Categorical Features.')
```

```
There are 7 Categorical Features.
```

In [96]:

```
#All the Categorical features.
cat_features
```

Out[96]:

```
['education',
 'Gender',
 'smoking',
 'BP_meds',
 'stroke',
 'hypertensive',
 'diabetes']
```

In [97]:

```
#head of the Categorical columns.
df[cat_features].head()
```

Out[97]:

	education	Gender	smoking	BP_meds	stroke	hypertensive	diabetes
0	4.0	1	0	0.0	0	0	0
1	2.0	0	0	0.0	0	0	0
2	1.0	1	1	0.0	0	0	0
3	3.0	0	1	0.0	0	1	0
4	3.0	0	1	0.0	0	0	0

In [98]:

```
#Extracting numerical features.

num_features = ['age','cigarettes/day','total_cholesterol','systolic_bp','diasto
print(f'There are {len(num_features)} Numerical Features.')
```

```
There are 8 Numerical Features.
```

In [99]:

```
#All the numerical features.
num_features
```

Out[99]:

```
['age',
 'cigarettes/day',
 'total_cholesterol',
 'systolic_bp',
 'diastolic_bp',
 'bmi',
 'heart_rate',
 'glucose']
```

In [100…

```
#first five rows of numerical columns.
df[num_features].head()
```

Out[100]:

	age	cigarettes/day	total_cholesterol	systolic_bp	diastolic_bp	bmi	heart_rate	glucose
0	39	0.0	195.0	106.0	70.0	26.97	80.0	77.0
1	46	0.0	250.0	121.0	81.0	28.73	95.0	76.0
2	48	20.0	245.0	127.5	80.0	25.34	75.0	70.0
3	61	30.0	225.0	150.0	95.0	28.58	65.0	103.0
4	46	23.0	285.0	130.0	84.0	23.10	85.0	85.0

[redacted]

Handling Missing Values

Handling the missing values is one of the greatest challenges faced by analysts, because making the right decision on how to handle it generates robust data models.

Let us look at different ways of imputing the missing values

1. **Deleting Rows**
2. **Replacing With Mean/Median/Mode**
3. **Assigning An Unique Category**

4.**Nearest Neighbors Imputations (KNNImputer)**

*We will be using the first(1)** and **Fourth(4)** way.*

Dropping All the Null Values:

Dropping all the missing values from the columns, It will not be a good idea to treat these columns with other popular techniques because of the nature and sensitivity of this dataset. It will not be quite justified the treatment. The best treatment is just dropping them.

Before

In [101…

```
#Checking the null count before removal of the null values.
df[['cigarettes/day','BP_meds','total_cholesterol','bmi','heart_rate','glucose']
```

Out[101]:

```
 cigarettes/day       29
BP_meds              53
total_cholesterol    50
bmi                  19
heart_rate            1
glucose             388
dtype: int64
```

In [102…

```
#Dropping the null values from the data.
df.dropna(subset=['cigarettes/day','BP_meds','total_cholesterol','bmi','heart_ra
```

After

In [103…

```
#Checking the null count after removal of the null values.
df[['cigarettes/day','BP_meds','total_cholesterol','bmi','heart_rate','glucose']
```

Out[103]:

```
 cigarettes/day      0
BP_meds              0
total_cholesterol    0
bmi                  0
heart_rate           0
glucose              0
dtype: int64
```

We have dropped education here beacause dropping only the missing values removes whole row and it will lead to shrink the data as we already have small dataset.So,We will be dropping the education column since it is not directly related with our target feature.

In [104…

```
# #Dropping Education from the dataset.
df.drop(['education'], axis=1, inplace=True)
```

In [105…

```
#Checking the count of null values in each column after treatment.
df.isna().sum()
```

Out[105]:

```
 Gender              0
age                  0
smoking              0
cigarettes/day       0
BP_meds              0
stroke               0
hypertensive         0
diabetes             0
total_cholesterol    0
systolic_bp          0
diastolic_bp         0
bmi                  0
heart_rate           0
glucose              0
10yearCHD            0
dtype: int64
```

Now, Our dataset is free of null/missing values.

EDA:

10yearCHD(Dependant Variable):

In [106…

```
# Checking the total people who have a risk of CHD(Coronary Heart
Disease). df["10yearCHD"].value_counts()
```

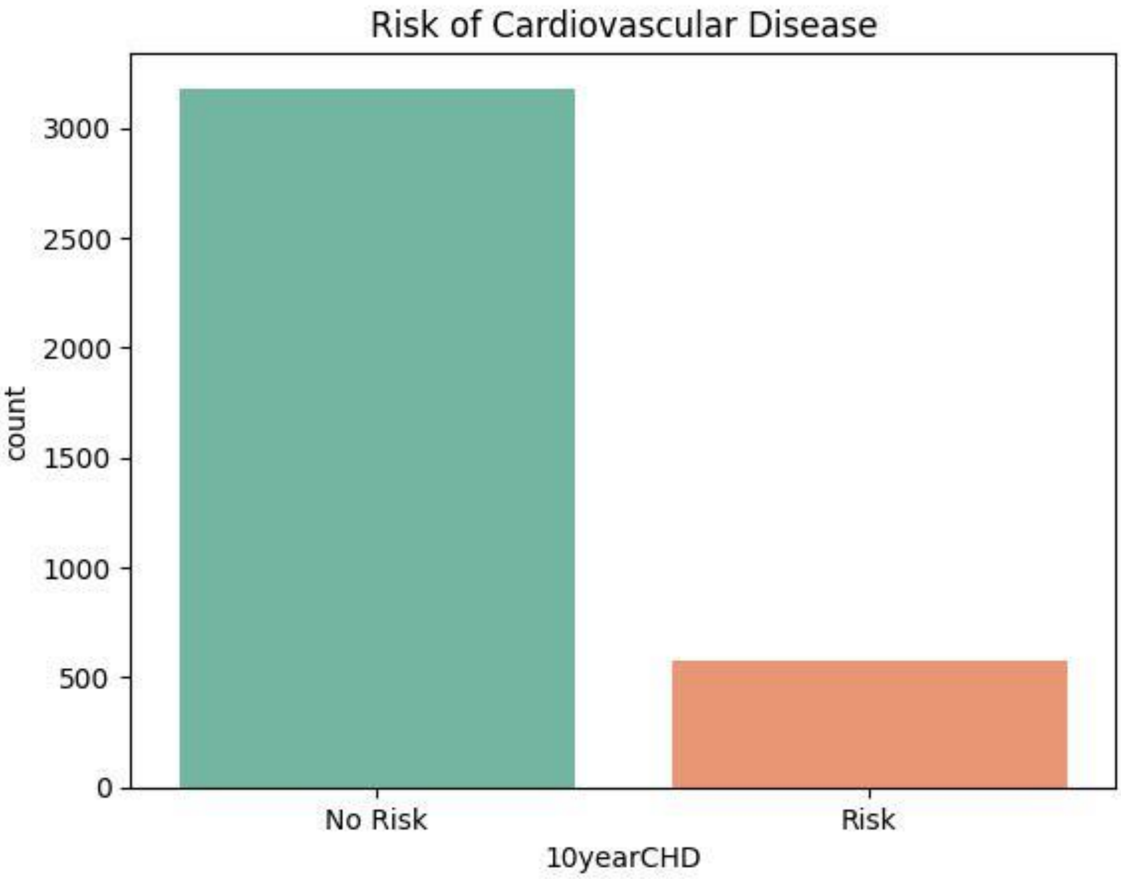
Out[106]:

```
0    3179
1572
Name: 10yearCHD, dtype: int64
```

Here, 10yearCHD signifies if the person has a risk of heart disease or not. It's a binary attribute(binary: “1”, means “Yes”, “0” means “No”) resembling the diagnosis results for *patients. We can use this attribute to see how many patients have a risk of CHD.*

In [107…

```
# plotting number of patients at risk of CHD vs those whose results are
normal. g = sns.countplot(data=df, x='10yearCHD',palette = "Set2")
g.set_xticklabels(['No Risk','Risk'])
g.set_title('Risk of Cardiovascular
Disease') plt.show()
```



Observation:

Looking at this count plot of target variable shows that the percentage/count of people with normal results are pretty high and this creates the problem of class imbalance. It could create problems for model to perform better because it will be overfitted with entries of normal patients and hence, It will become hard to predict for a person with a CHD because the model would be biased towards "No Risk".

So we have to convert this data into a balance class, we will treat this imbalance using SMOTE, during train/test split.

Analyzing Categorical features:

Gender

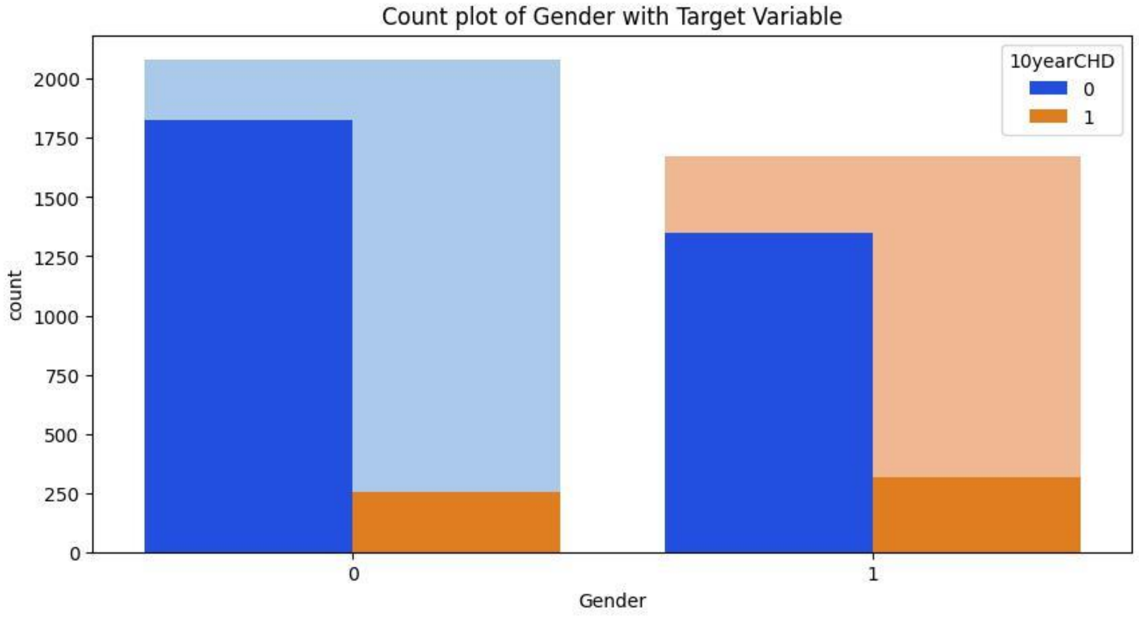
In [108…

```
# Counting the number of males and
females. df["Gender"].value_counts()
```

Out[108]:
```
 0    2081
 1    1670
Name: Gender, dtype: int64
```

In [109…
```
# Plotting the bar graph with number of males and females. fig,
ax = plt.subplots(figsize=(10,5)) sns.countplot(data=df,
x='Gender', ax=ax, palette='pastel')
sns.countplot(data=df, x='Gender', hue=df['10yearCHD'],ax=ax,
palette='bright') plt.xlabel('Gender')
plt.title('Count plot of Gender with Target Variable')
```

Out[109]:
```
Text(0.5, 1.0, 'Count plot of Gender with Target Variable')
```



Observation:

*We can see that number of female(0)** entries are more than **males(1)**. Since the data is a bit biased towards females, the model would be much more optimal to predict for a female patient.*

smoking

Smoking only about one cigarette per day carries a risk of developing coronary heart disease, around half that for people who smoke 20 per day. No safe level of smoking exists for cardiovascular disease.

In [110…
```
# Counting the number of smokers and non-smokers.(Yes = 1,No =
0) df["smoking"].value_counts()
```

Out[110]:
```
0    1919
11832
Name: smoking, dtype: int64
```

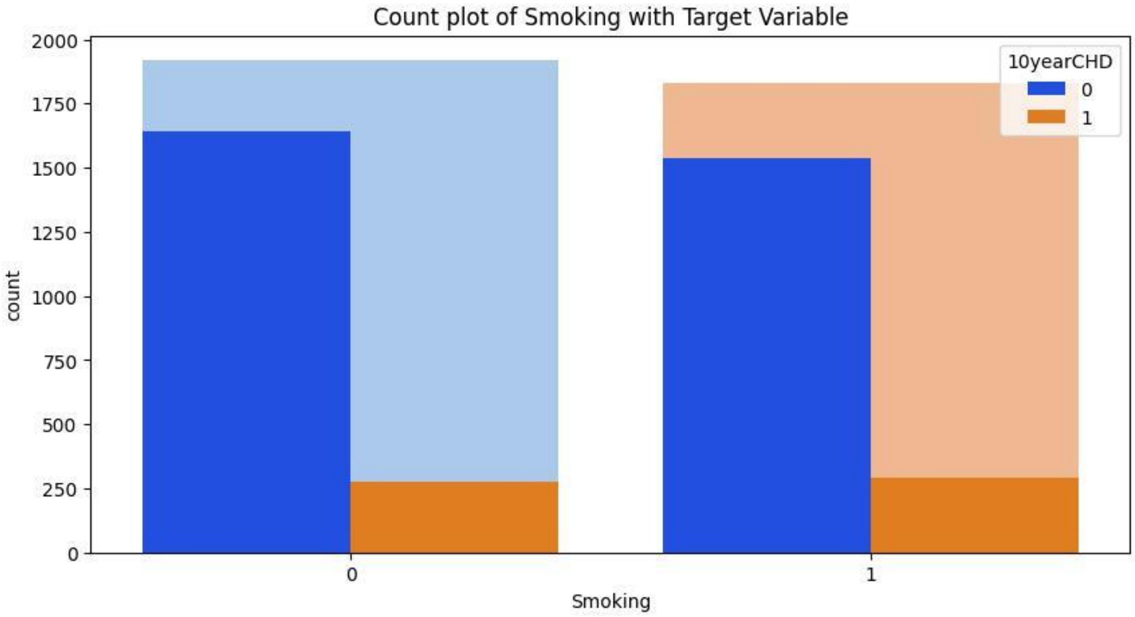
Almost the same number of entries by smokers and non- smokers. This feature is the ideal type since, it will not create any imbalance in class and it won't create any kind of bias

In [111…

```
# Plotting number of people smoking vs not
smoking. fig, ax = plt.subplots(figsize=(10,5))
sns.countplot(data=df, x='smoking', ax=ax, palette='pastel')
sns.countplot(data=df, x='smoking', hue=df['10yearCHD'],ax=ax,
palette='bright') plt.xlabel('Smoking')
plt.title('Count plot of Smoking with Target Variable')
```

Out[111]:

```
Text(0.5, 1.0, 'Count plot of Smoking with Target Variable')
```



Observation:

*From this plot we can conclude that, the non-smokers(0)** are at slightly lower risk of getting diagnosed by CHD in 10 years as compares to those who **smoke(1)**.*

Let us futher analyse how many males and females are smoking according to 'Gender' and 'smoking' features.

male smokers

In [112…

```
# Number of males who smokes cigarette.
male_smokers = df.loc[(df['smoking']==1) & (df['Gender']==1)]
```

In [113…

```
#Checking for the males who are smokers.
ms= male_smokers.shape[0]
print(f'There are total of {ms} male smokers who at least smoke one cigarette a
```

```
There are total of 1005 male smokers who at least smoke one cigarette a day.
```

female smokers

In [114…

```
# Number of females who smokes cigarette.
female_smokers = df.loc[(df['smoking']==1) & (df['Gender']==0)]
```

In [115…

```
#Checking for females who are smokers.
fs = female_smokers.shape[0]
print(f'There are total of {fs} female smokers who at least smoke one cigarette
There are total of 827 female smokers who at least smoke one cigarette a day.
```

From this individual analysis we can say that although the entries by females are higher than males but the number of smokers are more in males.

BP_meds:

Blood pressure is the pressure of circulating blood against the walls of blood vessels. Most of this pressure results from the heart pumping blood through the circulatory system. When used without qualification, the term "blood pressure" refers to the pressure in the large arteries.

In [116…

```
#Checking the value count of patients who have and have not experienced
stroke i df["BP_meds"].value_counts()
```

Out[116]:

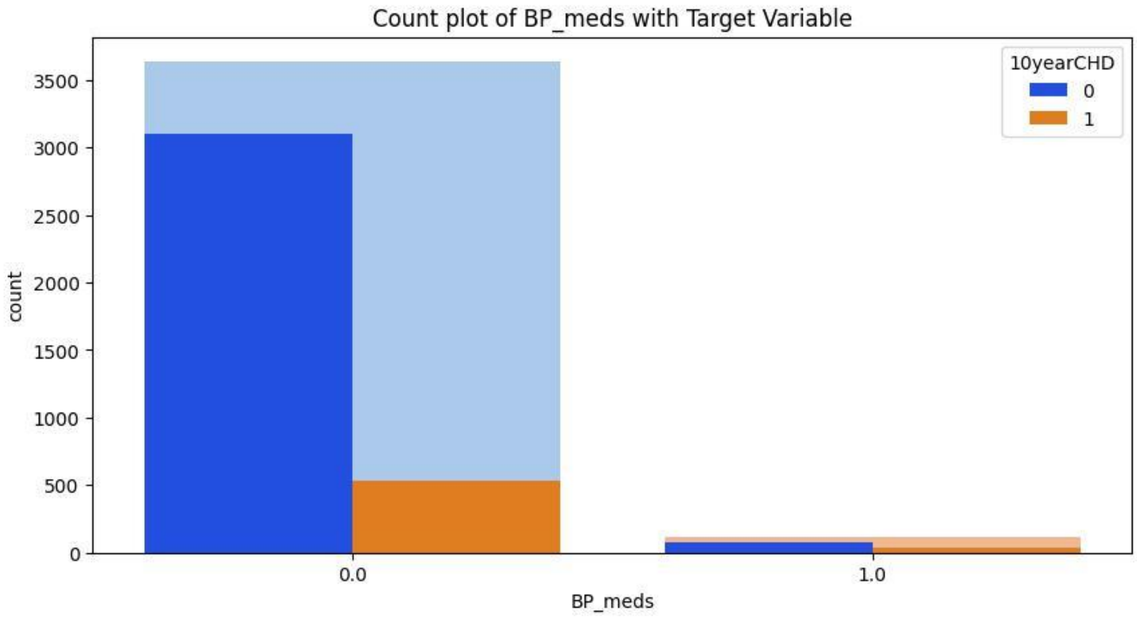
```
0.0    3637
1.0     114
Name: BP_meds, dtype: int64
```

In [117…

```
#Plotting the count plot for patients who take BP meds vs patients who do not
ta fig, ax = plt.subplots(figsize=(10,5))
sns.countplot(data=df, x='BP_meds', ax=ax, palette='pastel')
sns.countplot(data=df, x='BP_meds', hue=df['10yearCHD'],ax=ax, palette='bright')
plt.xlabel('BP_meds')
plt.title('Count plot of BP_meds with Target Variable')
```

Out[117]:

```
Text(0.5, 1.0, 'Count plot of BP_meds with Target Variable')
```



Observation:

From the above graph we can see that the count of patients who take BP meds are higher,and out of those maximum patients have less chance of CHD but only few of them are having risk of CHD.

stroke

A stroke, sometimes called a brain attack, occurs when something blocks blood supply to part of the brain or when a blood vessel in the brain bursts.

In [118…

```
#Checking the value count of patients who have and have not experienced
stroke i df["stroke"].value_counts()
```

Out[118]:

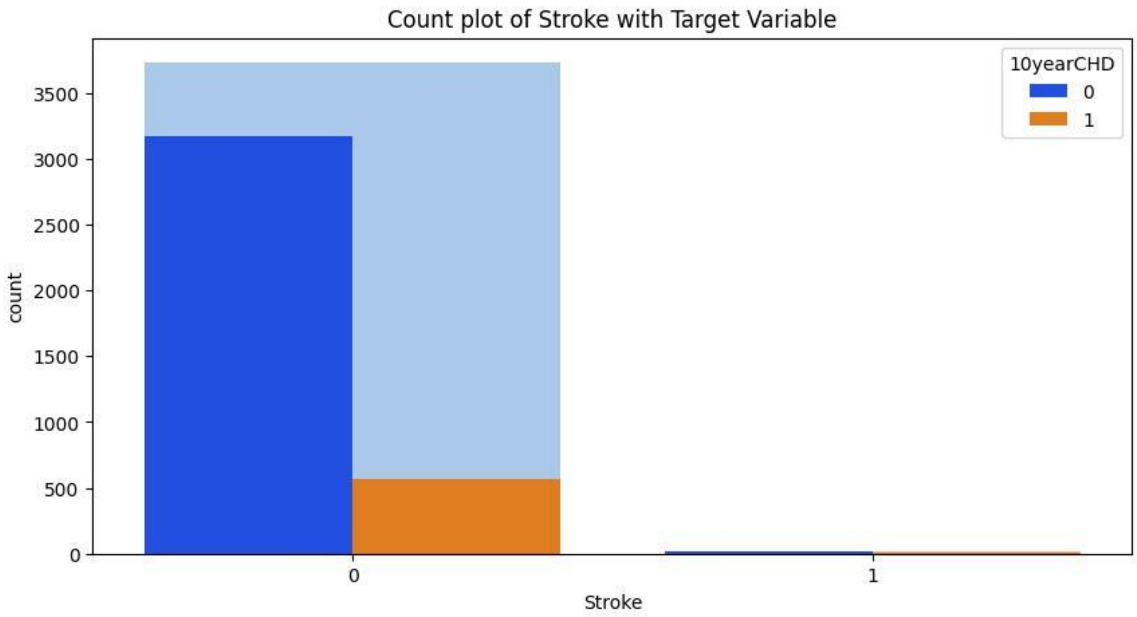
```
0    3730
121
Name: stroke, dtype: int64
```

In [119…

```
#Plotting the count plot for patients who have experienced stroke vs patients
wh fig, ax = plt.subplots(figsize=(10,5))
sns.countplot(data=df, x='stroke', ax=ax, palette='pastel')
sns.countplot(data=df, x='stroke', hue=df['10yearCHD'],ax=ax, palette='bright')
plt.xlabel('Stroke')
plt.title('Count plot of Stroke with Target Variable')
```

Out[119]:

```
Text(0.5, 1.0, 'Count plot of Stroke with Target Variable')
```



Observation:

From the value count above we can observe that the data have only 18 patients who have experienced stroke in the past, this will create a huge bias for the patients who have not experienced stroke in the past and the model will predict considering that.

hypertensive

Hypertension_or elevated blood pressure_is a erious s medical condition that significantly increases the risks of heart, brain, kidney and other diseases. An estimated 1.28 billion adults aged 30-79 years worldwide have hypertension, most (two-thirds) living in low- and middle-income countries.

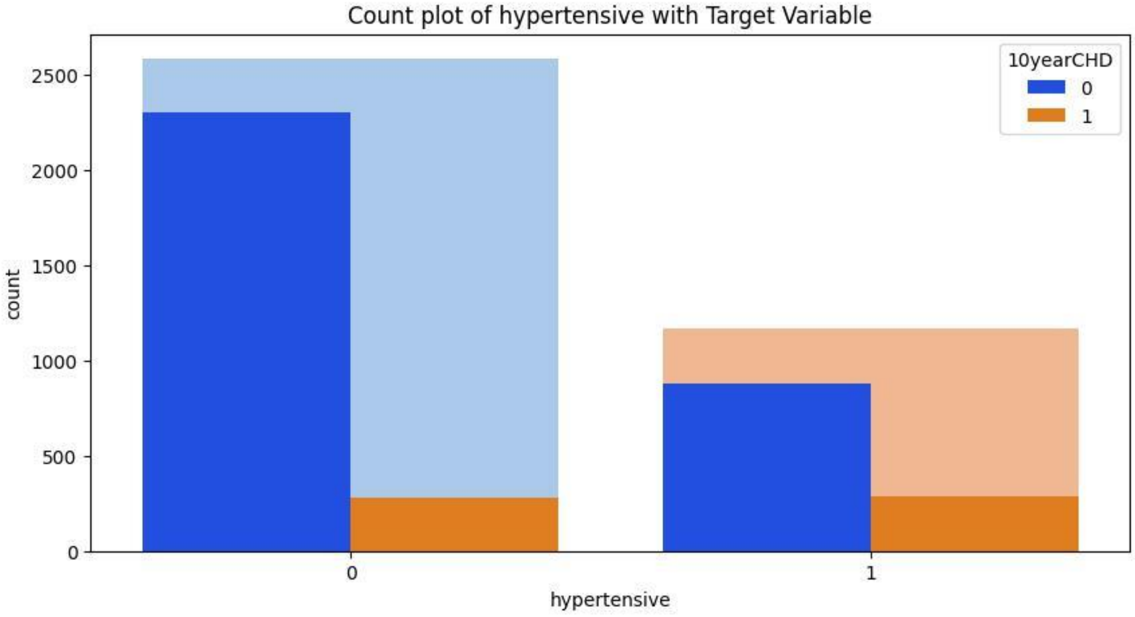
In [120…

```
#Checking for the value count of the hypertensive and non-hypertensive
patients. df["hypertensive"].value_counts()
```

```
Out[120]:  0    2581
           1    1170
          Name: hypertensive, dtype: int64
```

```
In [121…  #Plotting the count plot for hypertensive vs non-hypertensive with target
          varia fig, ax = plt.subplots(figsize=(10,5))
          sns.countplot(data=df, x='hypertensive', ax=ax, palette='pastel')
          sns.countplot(data=df, x='hypertensive', hue=df['10yearCHD'],ax=ax,
          palette='bri plt.xlabel('hypertensive')
          plt.title('Count plot of hypertensive with Target Variable')
```

```
Out[121]: Text(0.5, 1.0, 'Count plot of hypertensive with Target Variable')
```



Observation:

The plot shows the count of patients who are hypertensive is quite high than the non-hypertensive patients but by looking at both of the sides we can observe that the patients who are hypertensive are at high risk of CHD.

diabetes

Diabetes is a chronic disease that occurs either when the pancreas does not produce enough insulin or when the body cannot effectively use the insulin it produces. Insulin is a hormone that regulates blood glucose.

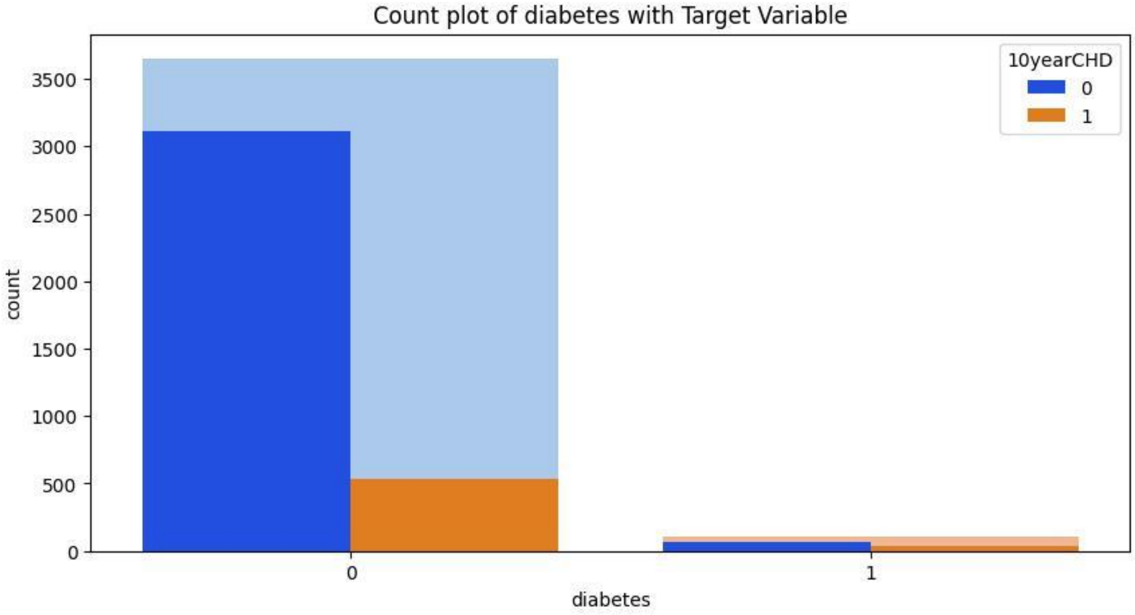
```
In [122…  #Checking the value count of diabetic and non-diabetic patients.
          df["diabetes"].value_counts()
```

```
Out[122]:  0    3649
          1102
          Name: diabetes, dtype: int64
```

```
In [123…  #Plotting the count plot for diabetic vs non-diabetic patients with target
          varia fig, ax = plt.subplots(figsize=(10,5))
          sns.countplot(data=df, x='diabetes', ax=ax, palette='pastel')
          sns.countplot(data=df, x='diabetes', hue=df['10yearCHD'],ax=ax, palette='bright'
          plt.xlabel('diabetes')
          plt.title('Count plot of diabetes with Target Variable')
```

Out[123]:

```
Text(0.5, 1.0, 'Count plot of diabetes with Target Variable')
```



Observation:

From the above graph we can see that there is a huge difference between the patients who are diabetic and non-diabetic.The dataset is biased towards the patients who are not diabetic and the number of diabetic people are very less.

Univariate Analysis

Univariate analysis explores variables (attributes) one by one. Variables could be either categorical or numerical."Uni" means "one", so in other words your data has only one variable. It doesn't deal with causes or relationships (unlike regression) and it's major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.There are different statistical and visualization techniques of investigation for each type of variable.Here we will be using distplots and boxplots.

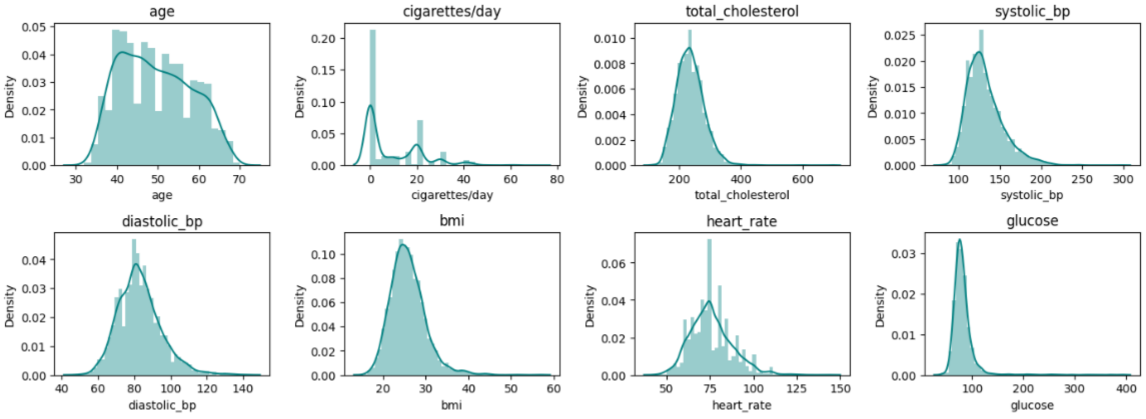
Univariate analysis for Numerical Features

In [124…

```
# Making distribution plot for Numerical features for checking the skewness.

n=1
plt.figure(figsize=(14,30))
for i in num_features:
  plt.subplot(12,4,n)
  n= n+1
  sns.distplot(df[i],color='teal')
  plt.title(i)
  plt.tight_layout()
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8:
UserWarnin g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8:
UserWarnin g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3266081622.py:8: UserWarnin
g:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please
see https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[i],color='teal')
```



Observation:

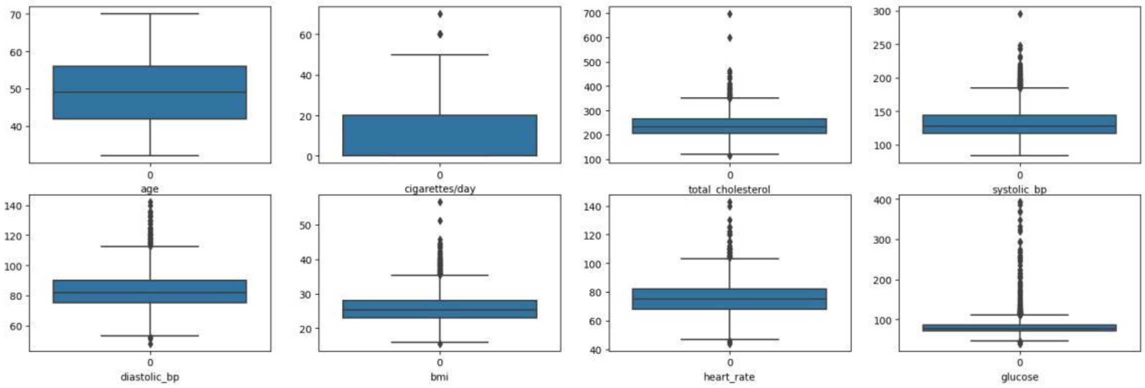
We observed from the above distribution plot of numerical features that the attributes cigarettes/day, total_cholesterol, systolic_bp, bmi, and glucose are slightly right skewed.

Outlier Detection

Outlier Analysis is a process that involves identifying the anomalous observation in the dataset. Outliers are extreme values that deviates from the other observations in the dataset.

In [125…

```
# plotting boxplot for each numerical feature to check for the outliers.

plt.figure(figsize=(20,40), facecolor='white')
plotnumber = 1
for numerical_feature in num_features:
    ax = plt.subplot(12,4,plotnumber)
    sns.boxplot(df[numerical_feature], )
    plt.xlabel(numerical_feature)
    plotnumber += 1
plt.show()
```



*We can see a lot of outliers in columns like, total_cholesterol, systolic_bp, diastolic_bp, bmi, glucose,** etc. As stated before we can't manipulate data in such way that we change the original patient stats, neither we can entirely drop those entries with outliers. This will lead to huge amount of data loss, We would lose meaningful data in order to achieve accurate predictions. The best solution to this could only be, to drop the rows with such outliers with minimal data loss.*

Removing the borderline outliers

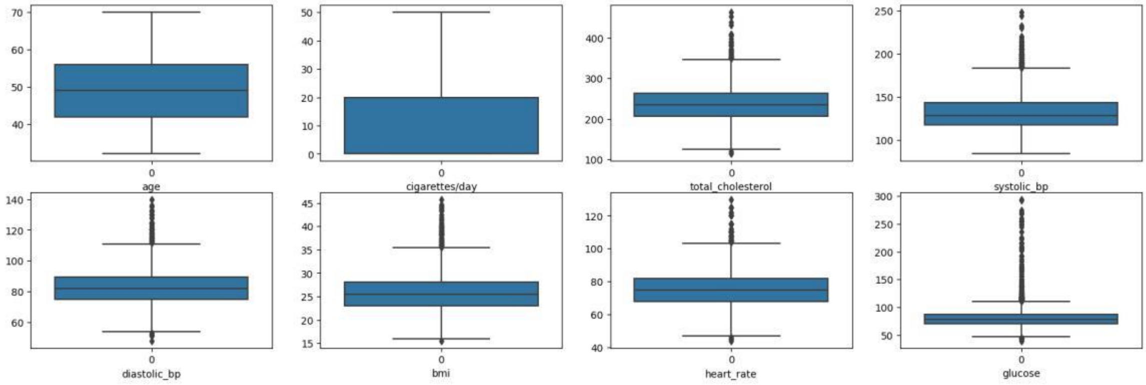
Removing the borderline outliers, We'll try to be considerate and only drop values that do not make any sense or unlikely to occur.

In [126…

```
# Removing values of Cigarette per day greater than
50. df = df[df["cigarettes/day"] <= 50]
# Removing values of DiaBp greater than 140.
df = df[df['diastolic_bp'] <= 140]
# Removing values of SysBP greater than
250. df = df[df['systolic_bp'] <= 250]
# Removing values of BMI greater than 50.
df = df[df['bmi'] <= 50]
# Removing values of heart rate greater than
130. df = df[df["heart_rate"] <= 130]
# Removing values of glucose greater than 300.
df = df[df["glucose"] <= 300]
# Removing values of total cholesterol greater than
500. df = df[df["total_cholesterol"] <= 500]
```

In [127…

```
# plotting boxplot for each numerical feature to check for the outliers.

plt.figure(figsize=(20,40), facecolor='white')
plotnumber = 1
for numerical_feature in num_features:
    ax = plt.subplot(12,4,plotnumber)
    sns.boxplot(df[numerical_feature], )
    plt.xlabel(numerical_feature)
    plotnumber += 1
plt.show()
```



Now,we again plotted boxplot after dropping rows with borderline outliers.

In [128…

```
#Information after removing the borderline outliers.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3723 entries, 0 to 4239
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             3723 non-null   int64  
 1   age                3723 non-null   int64  
 2   smoking            3723 non-null   int64  
 3   cigarettes/day     3723 non-null   float64
 4   BP_meds            3723 non-null   float64
 5   stroke             3723 non-null   int64  
 6   hypertensive       3723 non-null   int64  
 7   diabetes           3723 non-null   int64  
 8   total_cholesterol  3723 non-null   float64
 9   systolic_bp        3723 non-null   float64
 10  diastolic_bp       3723 non-null   float64
 11  bmi                3723 non-null   float64
 12  heart_rate         3723 non-null   float64
 13  glucose            3723 non-null   float64
 14  10yearCHD          3723 non-null   int64  
dtypes: float64(8), int64(7)
memory usage: 594.4 KB
```

Bivariate Analysis

Bivariate analysis is stated to be an analysis of any concurrent relation between two variables or attributes. This study explores the relationship of two variables as well as the *depth of this relationship to figure out if there are any discrepancies between two variables and any causes of this difference.*

*Here we have shown relationships of target variable** with all the **numerical variables**.*

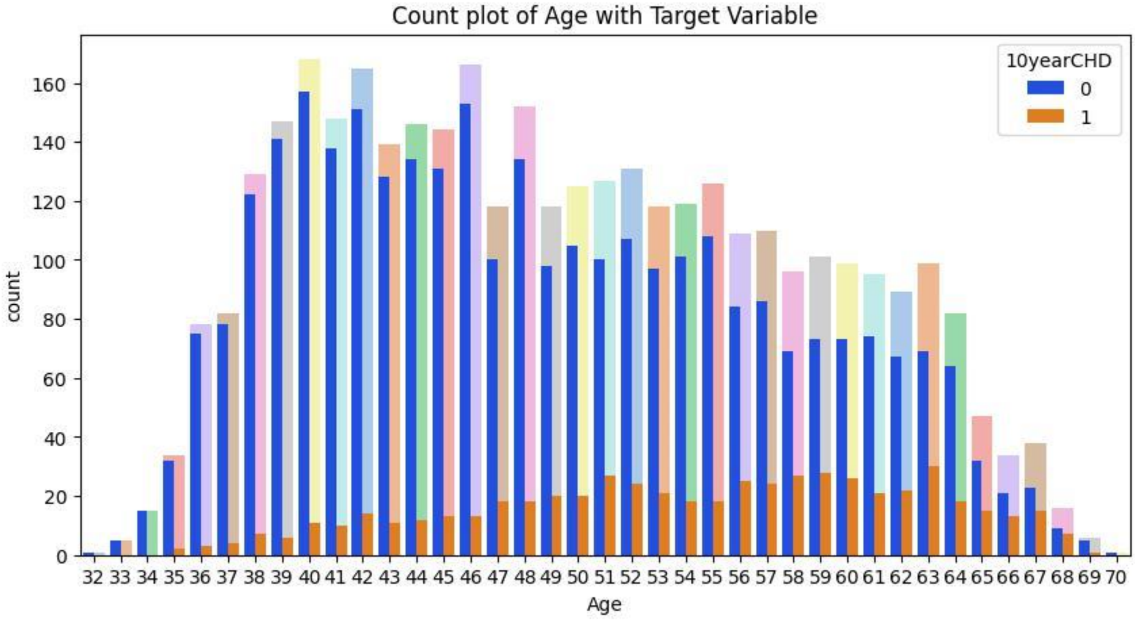
Age

In [129…

```
#Plotting count plot of age with target variable.
fig, ax = plt.subplots(figsize=(10,5))
sns.countplot(data=df, x='age', ax=ax, palette='pastel')
sns.countplot(data=df, x='age', hue=df['10yearCHD'],ax=ax, palette='bright')
plt.xlabel('Age')
plt.title('Count plot of Age with Target Variable')
```

Out[129]:

```
Text(0.5, 1.0, 'Count plot of Age with Target Variable')
```



Observation:

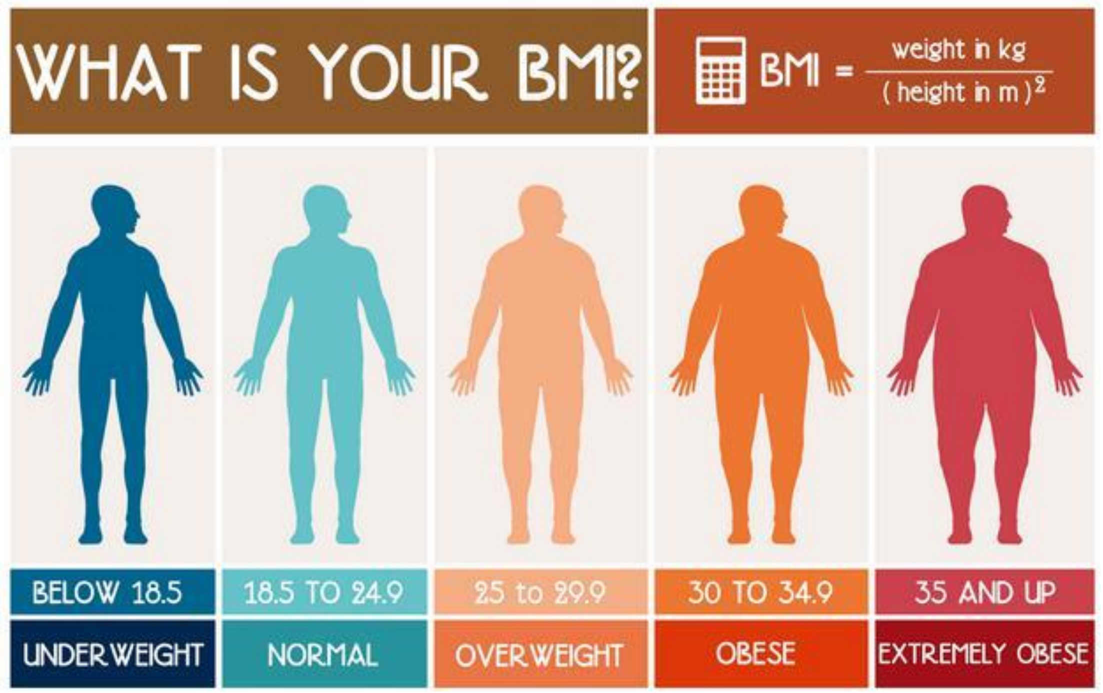
In this graph we can see that,between 32 to 40 there is less chance of CHDs.

After 40 there is an increase in CHDs and between 51 to 63 the graph shows highest chance of CHDs and after 63 the chance of CHDs is Decreasing as age increases.

BMI

The BMI is defined as the body mass divided by the square of the body height, and is expressed in units of kg/m^2.

Being overweight or obese increases your risk of developing diabetes and high blood pressure, both of which are risk factors for CVD.



If your BMI is:

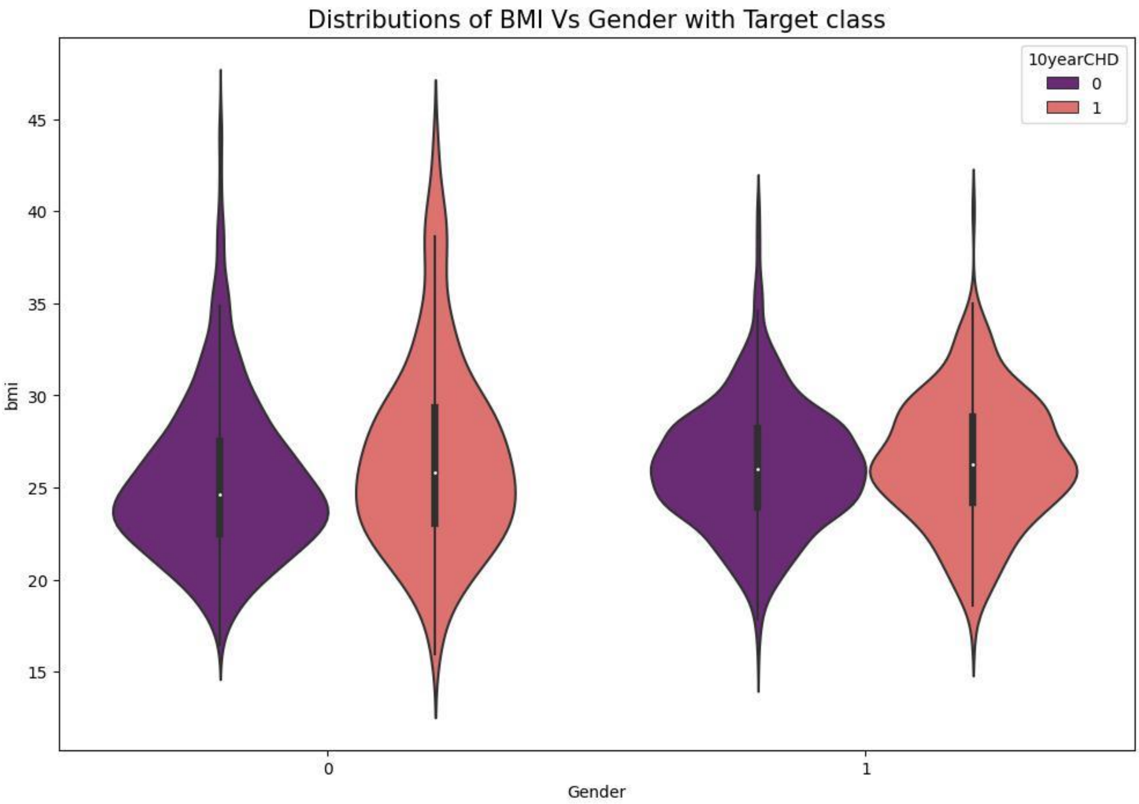
- below 18.5 – you're in the underweight range
- between 18.5 and 24.9 – you're in the healthy weight range
- between 25 and 29.9 – you're in the overweight range
- between 30 and 39.9 – you're in the obese range

In [130…

```
#plotting the boxplot between sex variable and BMI variable with target
class plt.figure(figsize=(12,8))
sns.violinplot(data=df,x="Gender", y='bmi',hue="10yearCHD",palette = 'magma')
plt.title("Distributions of BMI Vs Gender with Target class",fontsize=15)
```

Out[130]:

```
Text(0.5, 1.0, 'Distributions of BMI Vs Gender with Target class')
```



Observation:

In this above plot on x axis 0 represents female and 1 represents male, we can say that Female BMI is more than male BMI that leads to overweight.So,Female CHD is more than male CHD in our dataset.

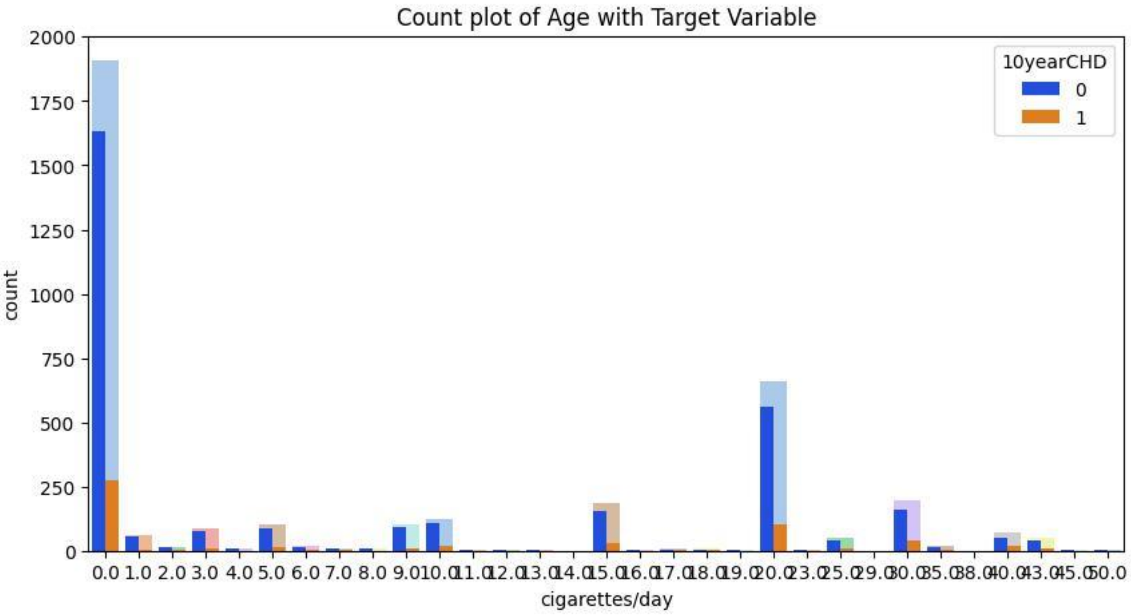
cigarettes/day

In [131…

```
#Plotting count plot of 'cigarettes/day' with target variable.
fig, ax = plt.subplots(figsize=(10,5))
sns.countplot(data=df, x='cigarettes/day', ax=ax, palette='pastel')
sns.countplot(data=df, x='cigarettes/day', hue=df['10yearCHD'],ax=ax,
palette='b plt.xlabel('cigarettes/day')
plt.title('Count plot of Age with Target Variable')
```

Out[131]:

```
Text(0.5, 1.0, 'Count plot of Age with Target Variable')
```



Observation:

Total_Cholesterol

Cholesterol is a fatty substance found in the blood. If you have high cholesterol, it can cause your blood vessels to narrow and increase your risk of developing a blood clot.

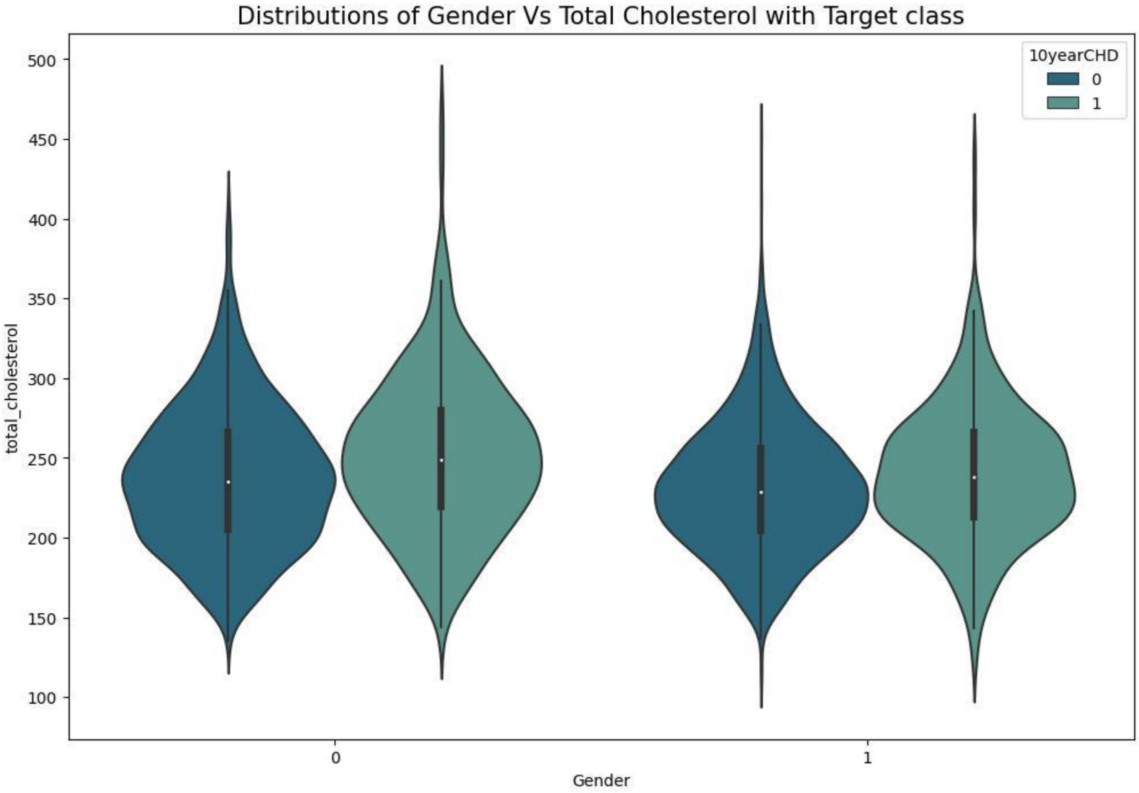
- High cholesterol is when you have too much of a fatty substance called cholesterol in your blood.
- It's mainly caused by eating fatty food, not exercising enough, being overweight, smoking and drinking alcohol. It can also run in families.
- You can lower your cholesterol by eating healthily and getting more exercise. Some people also need to take medicine.
- Too much cholesterol can block your blood vessels. It makes you more likely to have heart problems or a stroke.
- High cholesterol does not cause symptoms. You can only find out if you have it from a blood test.

In [132…

```
#plotting the boxplot between sex variable and totChol variable with target
clas plt.figure(figsize=(12,8))
sns.violinplot(data=df,x="Gender", y='total_cholesterol',hue="10yearCHD",palette
plt.title("Distributions of Gender Vs Total Cholesterol with Target class",fonts
```

Out[132]:

```
Text(0.5, 1.0, 'Distributions of Gender Vs Total Cholesterol with Target
clas s')
```

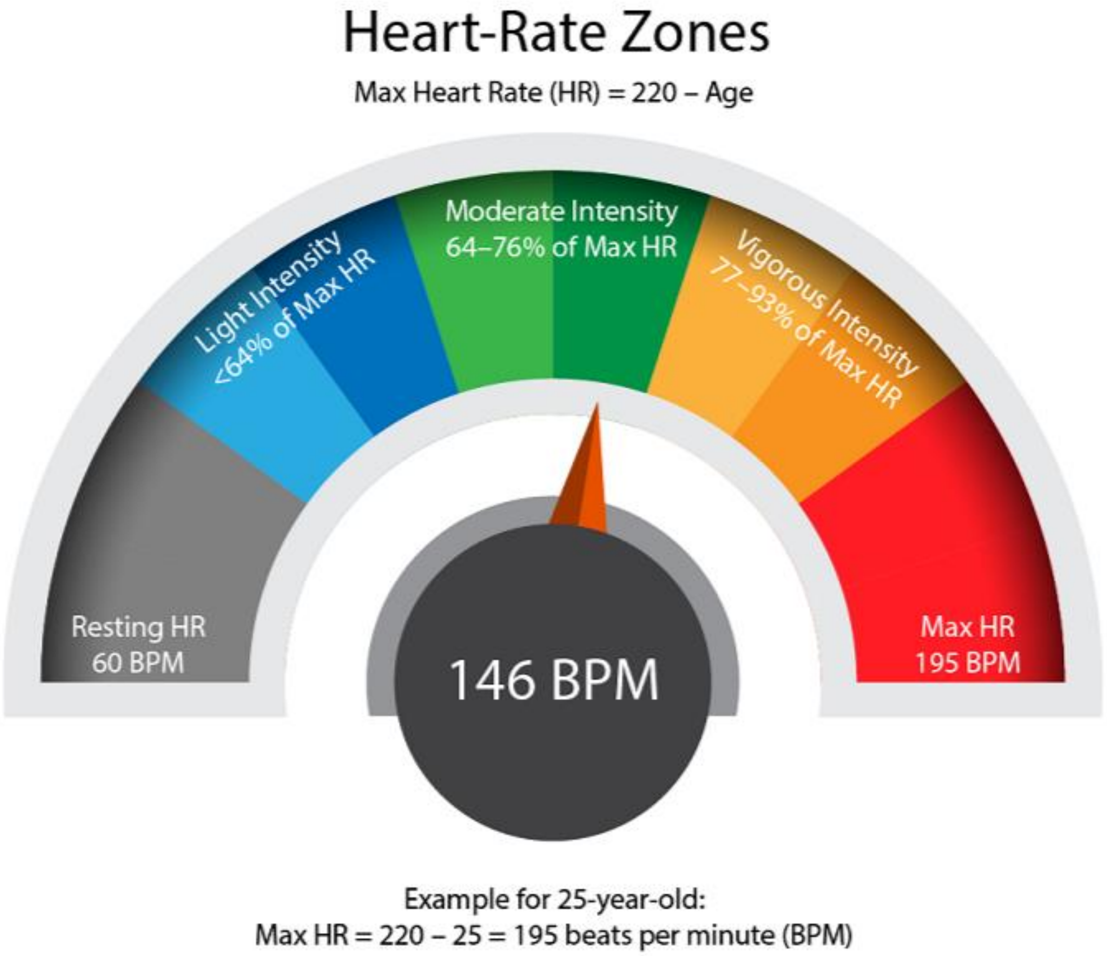


Observation:

Female has more cholesterol as compared to male. so, it can cause your blood vessels to narrow and increases your risk of developing a blood clot. In simple terms,more cholesterol leads to increase in CHD problems.

Heart_Rate

The number of heartbeats per unit of time, usually per minute. The heart rate is based on the number of contractions of the ventricles (the lower chambers of the heart). The heart rate may be too fast (tachycardia) or too slow (bradycardia). The pulse is a bulge of an artery from waves of blood that course through the blood vessels each time the heart beats. The pulse is often taken at the wrist to estimate the heart rate.

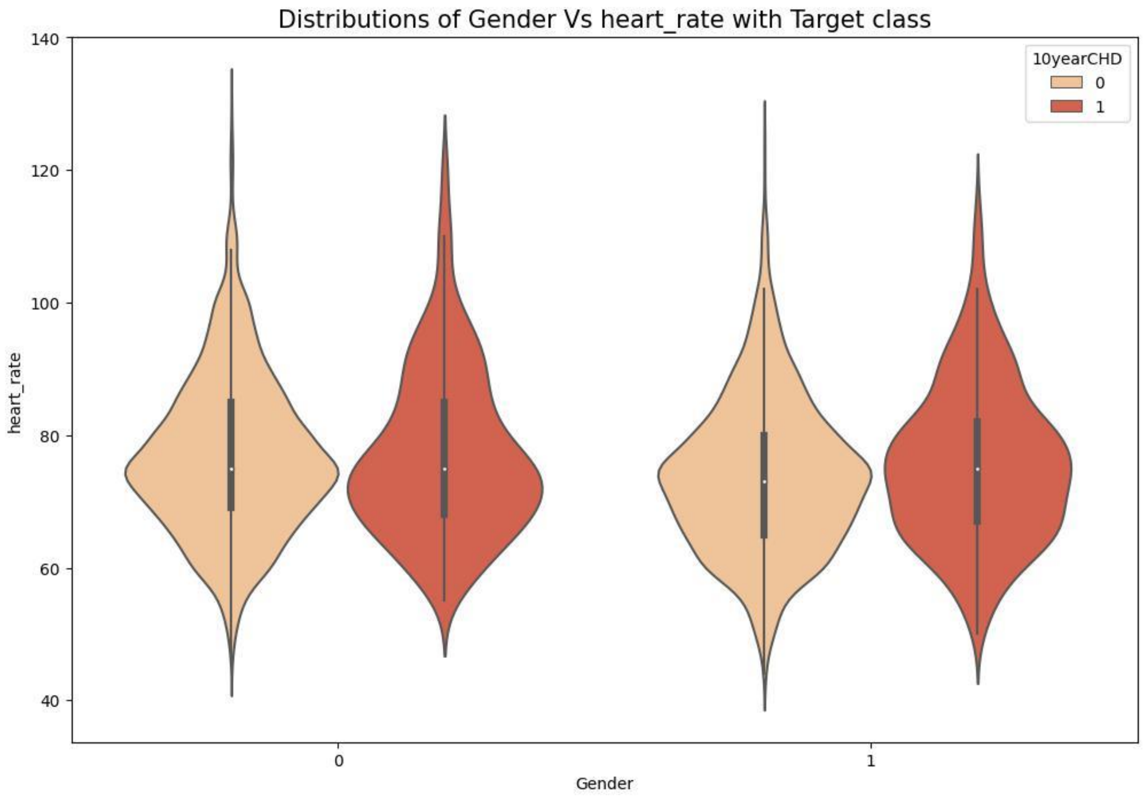


In [133…

```
#plotting the boxplot between Gender variable and heart_rate variable with
targe plt.figure(figsize=(12,8))
sns.violinplot(data=df,x="Gender",   y='heart_rate',hue="10yearCHD",palette   =   'OrR
plt.title("Distributions of Gender Vs heart_rate with Target class",fontsize=15)
```

Out[133]:

```
Text(0.5, 1.0, 'Distributions of Gender Vs heart_rate with Target class')
```



Observation:

As we can see the violin plot we can say that, Females have high heart rate as compared to males.

Glucose

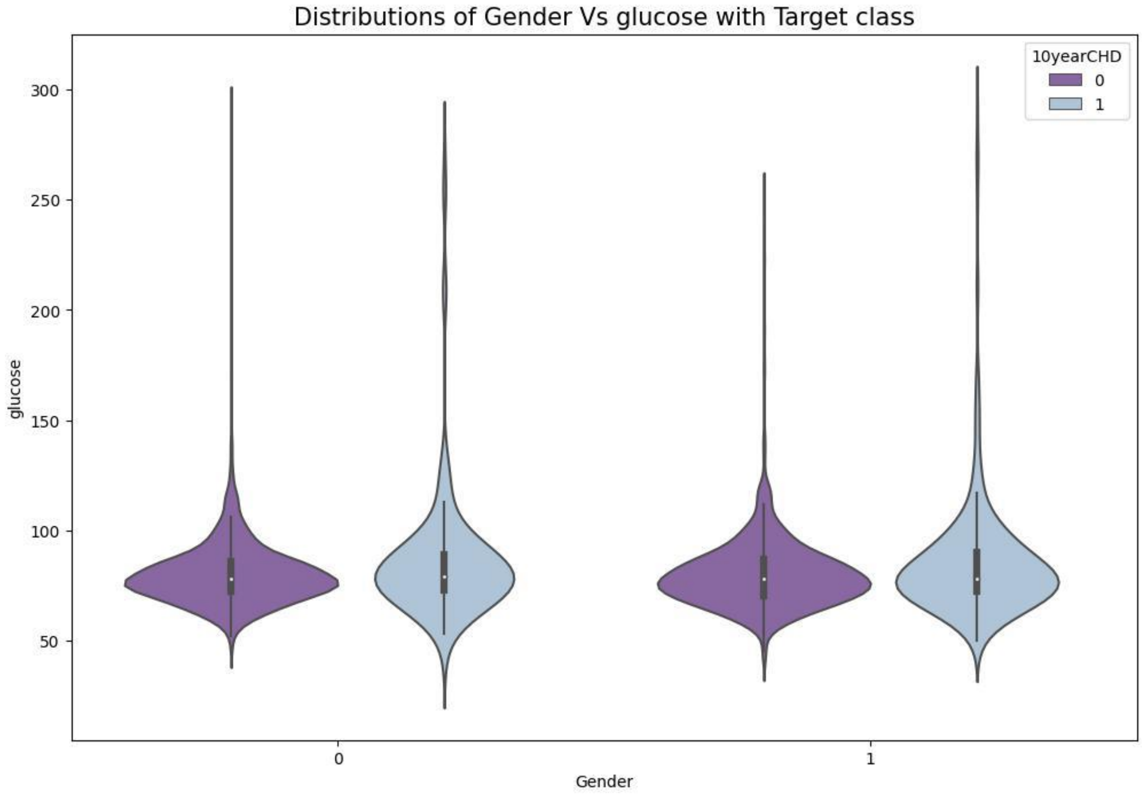
A fasting blood sugar level of 99 mg/dL or lower is normal, 100 to 125 mg/dL indicates you have prediabetes, and 126 mg/dL or higher indicates you have diabetes.

In [134…

```
#ploting the boxplot between sex variable and glucose variable with target
class plt.figure(figsize=(12,8))
sns.violinplot(data=df,x="Gender", y='glucose',hue="10yearCHD",palette = 'BuPu_r
plt.title("Distributions of Gender Vs glucose with Target class",fontsize=15)
```

Out[134]:

```
Text(0.5, 1.0, 'Distributions of Gender Vs glucose with Target class')
```



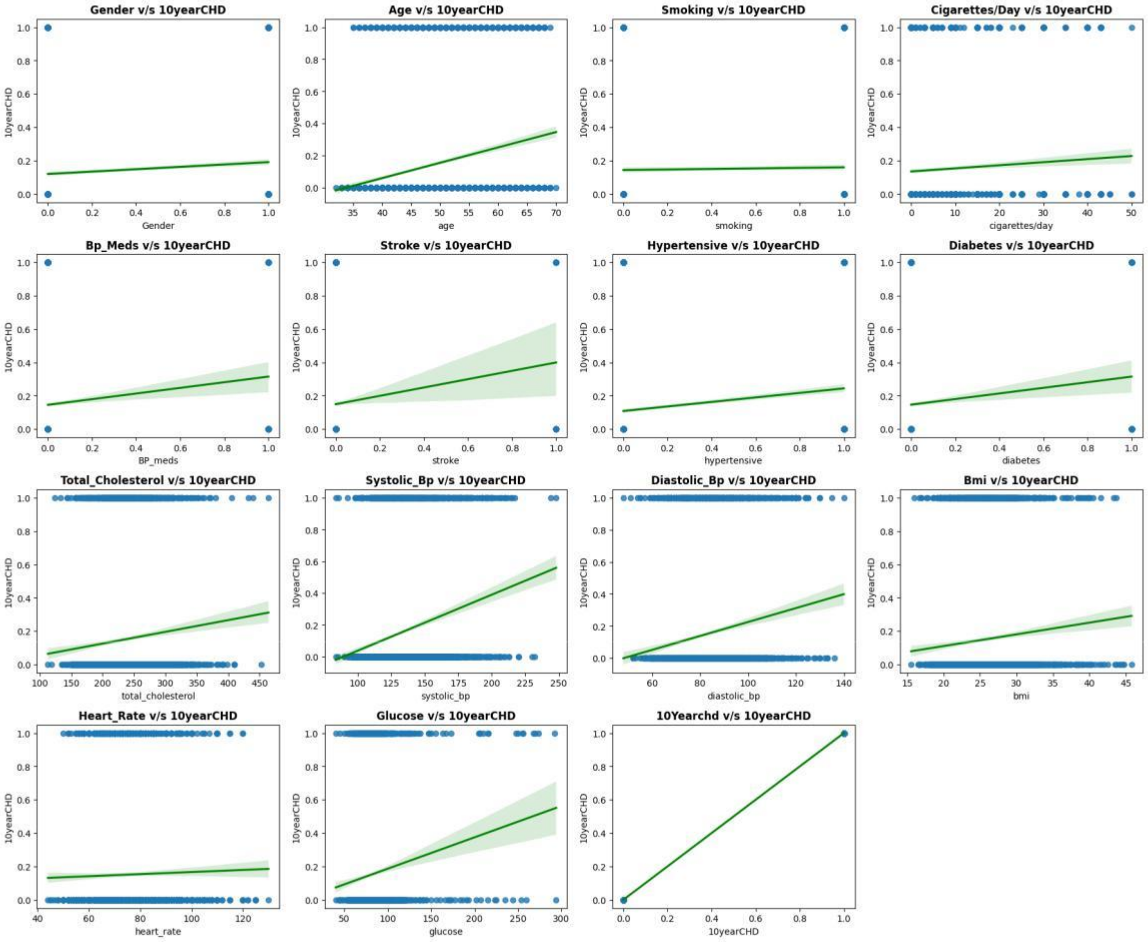
Observation:

In the above violin plot,we can see that males with CHD has more glucose level as compared to females with CHD.

Checking Linearity

In [135…

```
# Checking Linearity using Bivariate analysis.
# list of independent variables.
independent_variables = [i for i in df.columns if i not in ['10yearCHD']]

# defining figure.
plt.figure(figsize=(18,18))

# making subplots for all independent variables vs TenYearCHD(dependent
variable for n, column in enumerate(df.columns ):
  plt.subplot(5, 4, n+1)
  sns.regplot(x = df[column], y =df['10yearCHD'],line_kws={"color":
  "green"}) plt.title(f'{column.title()} v/s 10yearCHD',weight='bold')
  plt.tight_layout()
```



Observation:

From these subplots we observed that there is a Positive linearity in all of the variables in our dataset with our target variable i.e. 10yearCHD.

Multivariate Analysis

The purposes of multivariate data analysis is to study the relationships among the Provided attributes, classify the n collected samples into homogeneous groups, and make inferences about the underlying populations from the sample.

Checking for Multi-Collinearity

Correlation heatmaps are a type of plot that visualize the strength of relationships between numerical variables. Correlation plots are used to understand which variables are related to each other and the strength of this relationship.

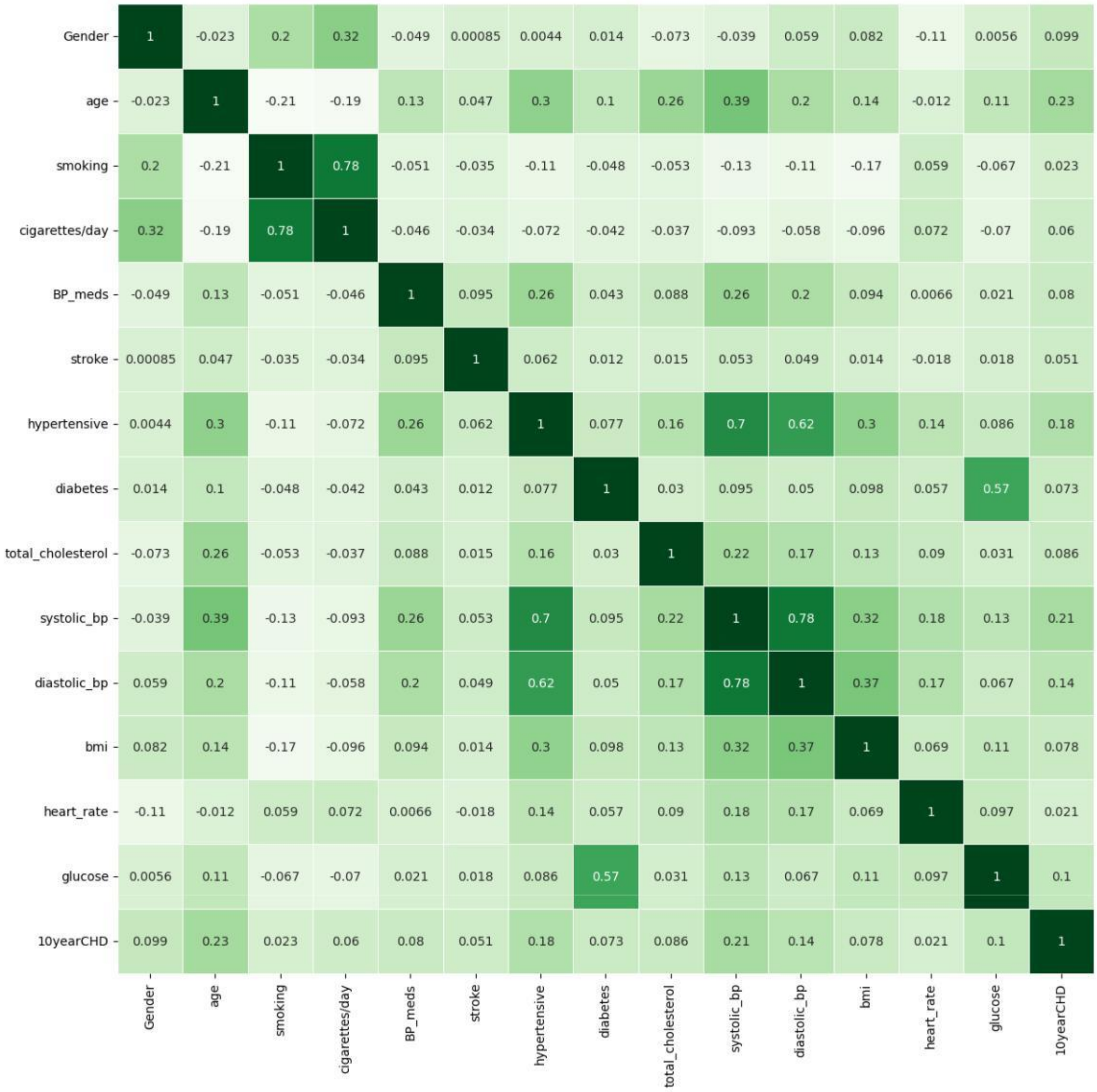
A correlation plot typically contains a number of numerical variables, with each variable represented by a column. The rows represent the relationship between each pair of variables. The values in the cells indicate the strength of the relationship, with positive values indicating a positive relationship and negative values indicating a negative relationship.

In [136…

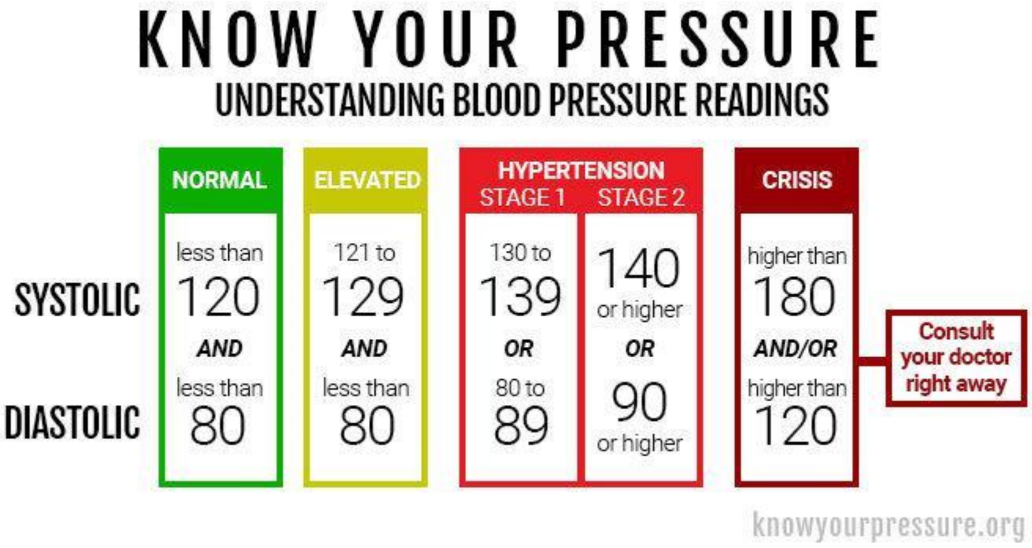
```
# Defining a Seaborn correlation
map(Heatmap). correlmap = df.corr()

# display the heatmap.
f, ax = plt.subplots(figsize=(14, 14))
sns.heatmap(correlmap,cmap= 'Greens', linewidths=.5,annot=True, ax = ax,cbar=Fal
```

Out[136]:

```
<AxesSubplot: >
```



Diastolic_bp and Systolic_bp



From the above Heatmap, We can see both of these columns are heavily correleted, there's some relationship we can establish with these two features further.

Also Elevation of systolic blood pressure predicts the risk of cardiovascular disease better than increases in diastolic blood pressure. Although associated with more variability in measurement, systolic blood pressure is easier to determine and allows more appropriate risk stratification than diastolic blood pressure.

We can combine these two features using the following formula:

MAP = (Systolic Blood Pressure + 2 x Diastolic Blood Pressure) / 3

Here, MAP signifies Mean Arterial Pressure

In [137…

```
# Calculating MAP using 'SysBP' and 'DiaBP'.
df["mean_art_pressure"] = (df["systolic_bp"] + 2 * df["diastolic_bp"])/3
```

In [138…

```
# Dropping the SysBP and DiaBp attributes, since they're both included in
MAP. df.drop(columns = ["systolic_bp", "diastolic_bp"], inplace = True)
```

Dropping Smoking

Since, the cigarette/day column is already having the information about smokers and non-smokers, where we can define by the number of cigarettes consumed by patients.So, we will be dropping the smoking column.

(As in cigarette/day column 0 represents patients who are non-smokers)

In [139…

```
#Dropping the smoking.
df.drop(columns = ["smoking"], inplace = True)
```

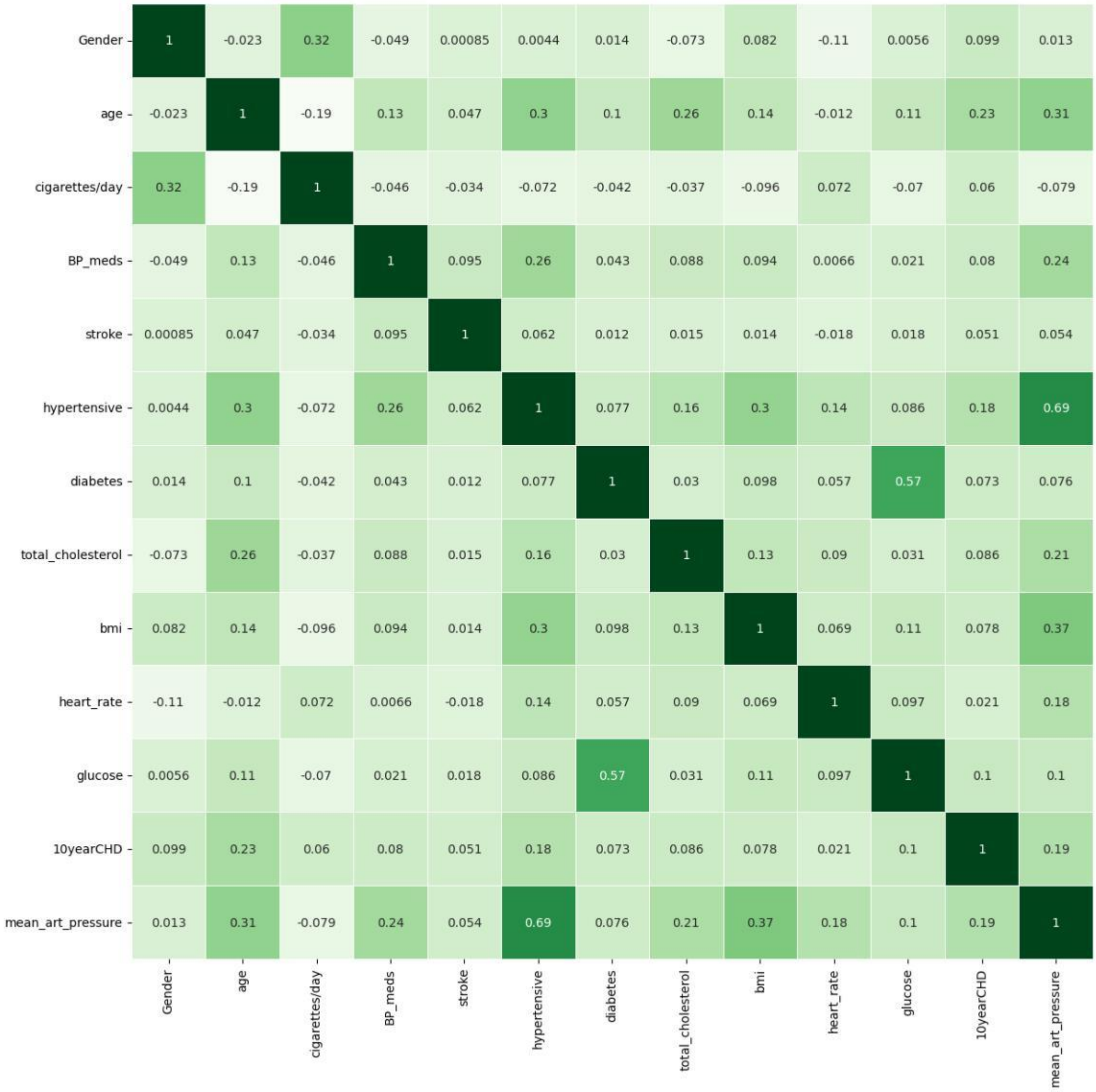
Checking Multi-collinearity after adjustments

In [140…

```
# Defining a Seaborn correlation
map(Heatmap). correlmap = df.corr()

# display the heatmap.
f, ax = plt.subplots(figsize=(14, 14))
sns.heatmap(correlmap,cmap= 'Greens', linewidths=.5,annot=True, ax = ax,cbar=Fal
```

Out[140]:

```
<AxesSubplot: >
```



One-Hot Encoding:

One hot encoding can be defined as the essential process of converting the categorical data variables to be provided to machine and deep learning algorithms which in turn improve predictions as well as classification accuracy of a model.

In [141…

```
#Creating dummy variables from categorical features.
dataset = pd.get_dummies(df,columns = ['Gender','BP_meds','stroke','hypertensive
```

In [142…

```
#Checking head after creating dummies.
dataset.head()
```

Out[142]:

	age	cigarettes/day	total_cholesterol	bmi	heart_rate	glucose	10yearCHD	mean_art_pres
0	39	0.0	195.0	26.97	80.0	77.0	0	82.00
1	46	0.0	250.0	28.73	95.0	76.0	0	94.33
2	48	20.0	245.0	25.34	75.0	70.0	0	95.83
3	61	30.0	225.0	28.58	65.0	103.0	1	113.33
4	46	23.0	285.0	23.10	85.0	85.0	0	99.33

Final Dataset

In [147…

```
# Checking the dataset after all the adjustments and
transformations. dataset
```

Out[147]:

	age	cigarettes/day	total_cholesterol	bmi	heart_rate	glucose	10yearCHD	mean_art_
0	39	0.0	195.0	26.97	80.0	77.0	0	8
1	46	0.0	250.0	28.73	95.0	76.0	0	9
2	48	20.0	245.0	25.34	75.0	70.0	0	9
3	61	30.0	225.0	28.58	65.0	103.0	1	11
4	46	23.0	285.0	23.10	85.0	85.0	0	9
...	...	...	...	...	...	...	...	
4233	50	1.0	313.0	25.97	66.0	86.0	1	12
4234	51	43.0	207.0	19.71	65.0	68.0	0	9
4237	52	0.0	269.0	21.47	80.0	107.0	0	9
4238	40	0.0	185.0	25.60	67.0	72.0	0	11
4239	39	30.0	196.0	20.91	85.0	80.0	0	10

3723 rows × 18 columns

In [144…

```
#Checking the shape of the dataset after all the transformations.

dataset.shape
```

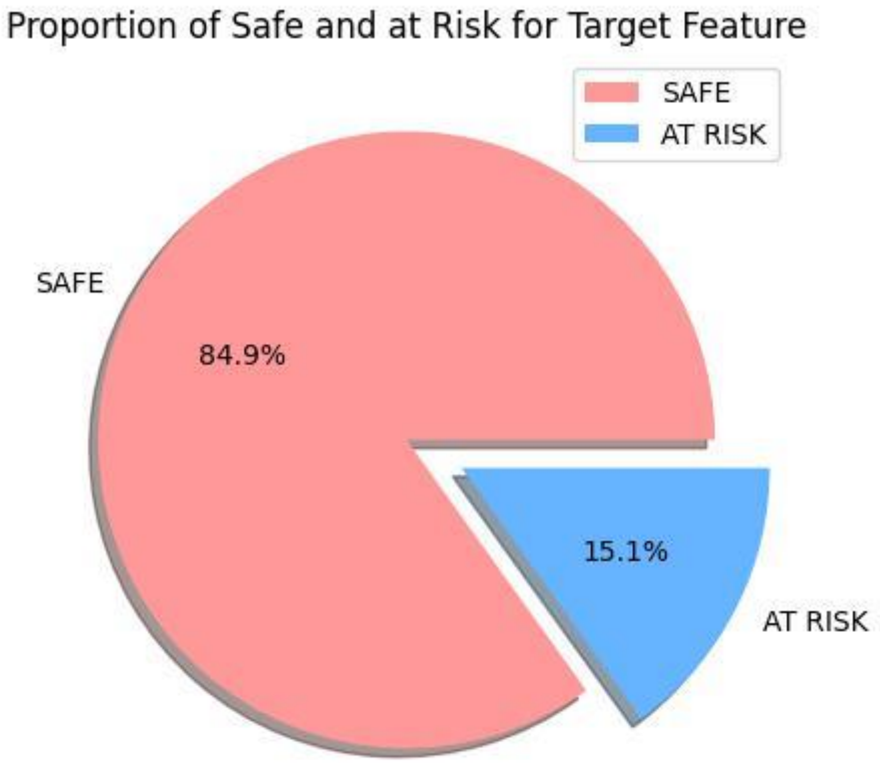
Out[144]:

```
(3723, 18)
```

Graph of Imbalance

In [148…

```
# Plotting the pie chart to check the balance in the dataset.

plt.figure(figsize=(7,5), dpi=100)
proportion = df['10yearCHD'].value_counts()
labels = ['SAFE','AT RISK']
plt.title('Proportion of Safe and at Risk for Target Feature')
plt.pie(proportion, explode=(0,0.2),labels=labels, shadow = True, autopct = '%1.
plt.legend()
plt.show()
```



As we can see that our target variable is highly imbalanced.

Majority of the data points belong to "SAFE"(no risk of CHD) class. Ratio of "SAFE" class to "AT RISK" class is 17:3.

In [149…

```
# Checking the count of the classes in the target variable.

df['10yearCHD'].groupby(df['10yearCHD']).count()
```

Out[149]:

```
10yearCHD
0    3161
1562
Name: 10yearCHD, dtype: int64
```

Scaling:

Scaling the Numerical Variables with StandardScaler.

StandardScaler is used to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1.

In [158…

```
#Applying normalization operation for numeric stability

from sklearn.preprocessing import
StandardScaler standardizer = StandardScaler()
columns_to_scale     =     ['age','cigarettes/day','total_cholesterol','mean_art_pressu
dataset[columns_to_scale] = standardizer.fit_transform(dataset[columns_to_scale]
## viewing dataset after
scalling dataset
```

Out[158]:

	age	cigarettes/day	total_cholesterol	bmi	heart_rate	glucose	10yearCHD
0	-1.231162	-0.764246	-0.956019	0.295785	0.368696	-0.214005	0
1	-0.413514	-0.764246	0.308249	0.735609	1.633892	-0.265009	0
2	-0.179901	0.959336	0.193315	-0.111553	-0.053037	-0.571036	0
3	1.338587	1.821126	-0.266418	0.698124	-0.896501	1.112111	1
4	-0.413514	1.217873	1.112783	-0.671330	0.790428	0.194031	0
...	...	...	...	...	...	...	...
4233	0.053713	-0.678067	1.756410	0.045884	-0.812154	0.245035	1
4234	0.170520	2.941455	-0.680179	-1.518492	-0.896501	-0.673045	0
4237	0.287326	-0.764246	0.744996	-1.078667	0.368696	1.316129	0
4238	-1.114355	-0.764246	-1.185886	-0.046579	-0.727808	-0.469027	0
4239	-1.231162	1.821126	-0.933032	-1.218611	0.790428	-0.060992	0

3723 rows × 18 columns

Splitting The Data

Train Test Split: *The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm.The procedure involves taking a dataset and dividing it into two subsets.*

In [159…

```
# Splitting the data into set of independent variables and a dependent
variable. X = dataset.drop('10yearCHD',axis=1).values
y = dataset['10yearCHD'].values
```

In [160…

```
#Train test split
from sklearn.model_selection import train_test_split
# Dividing the data in training and test dataset.
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.25, rando
# checking the shape of our train and test
data. print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2792, 17)
(931, 17)
(2792,)
(931,)
```

In [161…

```
print('Original dataset shape', len(X_train))
```

```
Original dataset shape 2792
```

In [162…

```
res_df=pd.DataFrame()
```

Performance Metrics

Different performance metrics are used to evaluate machine learning model. Based on our task we can choose our performance metrics. Since our task is of classification and that too binary class classification, whether client will or will not subscribe for deposits.

Here we will be using AUC ROC

*****ROC** also known as Receiver Operating Characteristics, shows the performance of binary class classifiers across the range of all possible thresholds plotting between true positive rate and 1-false positive rate.*

*****AUC** measures the likelihood of two given random points, one from positive and one from negative, the classifier will rank the positive points above negative points. AUC-ROC is popular classification metric that presents the advantage of being independent of false positive or negative points.*

*****Secondary Performance Metrics***

*****Macro-F1 Score:** F1 score is the harmonic mean between Precision and Recall.

Macro F1 score is used to know how our model works in overall dataset.*

*****Confusion Matrix:** This matrix gives the count of true negative, true positive,

false positive and false negative data points.*

In [ ]:

```
pip install sklearn
```

In [204…

```
#Importing Important libraries

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

from sklearn.model_selection import RepeatedStratifiedKFold from
sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix from
sklearn.metrics import classification_report
```

Models

Following models have been used for predictions:-

- Logistic Regression Classifier
- Decision Tree Classifier
- Random Forest Classifier
- K-Nearest Neighbors Classifier
- Gaussian Naive Bayes Classifier
- Support Vector Machine Classifier

Extra Null value removing by KNN Imputer

In [164…

```
from sklearn.impute import KNNImputer,SimpleImputer
```

In [165…

```
knn=KNNImputer()
x_train_trf=knn.fit_transform(X_train)
x_test_trf=knn.transform(X_test)
```

In [166…

```
pd.DataFrame(x_train_trf)
```

Out[166]:

	0	1	2	3	4	5	6	7	8	9
0	-1.581582	-0.764246	-0.634205	-1.048679	-0.053037	-0.418023	-1.025867	1.0	0.0	1.0
1	0.404133	-0.764246	0.974863	1.265398	1.212160	0.296040	1.660608	1.0	0.0	1.0
2	-1.347968	-0.764246	0.446169	-0.091561	-1.149540	0.041017	0.212247	0.0	1.0	1.0
3	0.988167	-0.333351	-0.496285	0.818076	-0.053037	0.857089	1.029870	1.0	0.0	1.0
4	-0.763934	-0.764246	-0.795112	-0.613853	0.031310	-0.418023	-1.084269	1.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...
2787	1.104974	-0.505709	0.078382	-0.096559	1.043467	-0.009987	-0.944105	1.0	0.0	1.0
2788	-0.063094	0.528440	1.664463	1.585270	-1.318233	-0.418023	0.726182	0.0	1.0	1.0
2789	0.871360	-0.764246	-1.691593	-0.781286	-0.474769	-0.163001	-1.025867	1.0	0.0	1.0
2790	-1.114355	0.959336	-0.657192	0.575673	-0.053037	-0.775054	-1.095949	0.0	1.0	1.0
2791	0.988167	0.528440	-1.415753	0.968016	1.633892	1.724164	-2.299022	0.0	1.0	1.0

2792 rows × 17 columns

Logistic Regression Classifier

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Logistic Regression is used when the dependent variable(target) is categorical. The model delivers a binary or dichotomous outcome limited to two possible outcomes: yes/no, 0/1, or true/false.

Logical regression analyzes the relationship between one or more independent variables and classifies data into discrete classes. It is extensively used in predictive modeling, where the model estimates the mathematical probability of whether an instance belongs to a specific category or not.

For example, 0 – represents a negative class; 1 – represents a positive class. Logistic regression is commonly used in binary classification problems where the outcome variable reveals either of the two categories (0 and 1).

Typical properties of the logistic regression equation include:

- *Logistic regression's dependent variable obeys 'Bernoulli distribution'*
- *Estimation/prediction is based on 'maximum likelihood.'*
- *Logistic regression does not evaluate the coefficient of determination (or R squared) as observed in linear regression'. Instead, the model's fitness is assessed through a concordance.*

In [167…

```
#Importing Logistic Regression from sklearn
from sklearn.linear_model import LogisticRegression
```

In [168…

```
# Creating model object for logistic regression.

clf = LogisticRegression(fit_intercept=True, max_iter=10000)
```

In [169…

```
# fit the model.

clf.fit(x_train_trf,y_train)
```

Out[169]:

```
▾ LogisticRegression
LogisticRegression(max_iter=10000)
```

In [170…

```
# Getting the predicted classes for training and testing set

train_class_preds = clf.predict(x_train_trf)
test_class_preds = clf.predict(x_test_trf)
```

In [181…

```
# Getting the accuracy scores for training and testing set.

train_accuracy_lg = accuracy_score(train_class_preds, y_train)
test_accuracy_lg = accuracy_score(test_class_preds, y_test)

# Display accuracies.

print("The accuracy on train data is ", train_accuracy_lg)
print("The accuracy on test data is ", test_accuracy_lg)
```

```
The accuracy on train data is 0.8556590257879656 The
accuracy on test data is 0.8549946294307197
```

In [182…

```
# Confusion Matrix for Logistic regression classifier.

cf_matrix = confusion_matrix(y_test,test_class_preds)
cf_matrix
```

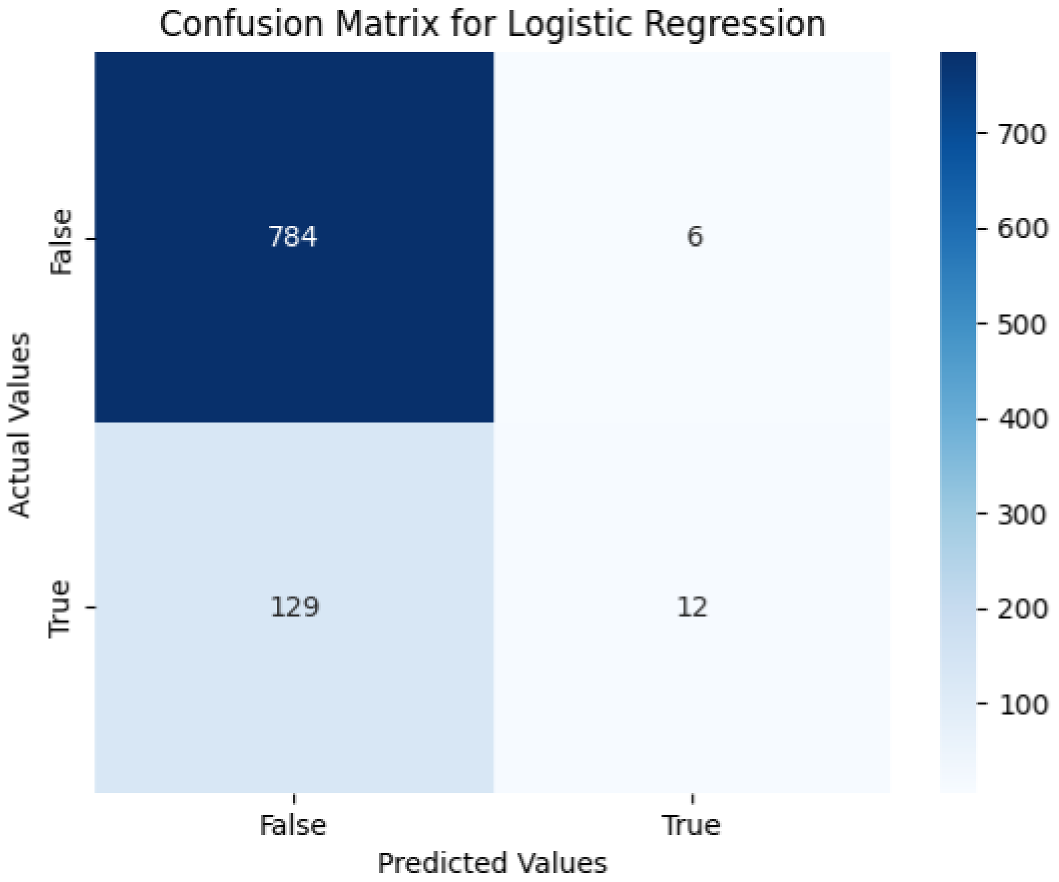
Out[182]:

```
array([[784,   6],
       [129,              12]], dtype=int64)
```

In [173…

```
#Plotting the cofusion matrix.
labels = ['784',' 6','129','12']

labels = np.asarray(labels).reshape(2,2)
```

```
ax = sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix for Logistic
Regression'); ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values');

# Ticket labels - List must be in alphabetical
order ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

# Display the visualization of the Confusion
Matrix. plt.show()
```



In [184…

```
# Predicted values.
y_pred_log_reg = clf.predict(x_test_trf)
```

In [183…

```
# Getting classification report.

dict_1 = classification_report(y_test, y_pred_log_reg, output_dict = True)
```

In [185…

```
#Adding results to model evaluation dataframe.
tempodf=pd.DataFrame(dict_1).transpose()
tempodf['Model'] = 'Logistic Regression Classifier'
res_df=res_df.append(tempodf[2:-2])
res_df
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\3833227129.py:4:
FutureWarni ng: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  res_df=res_df.append(tempodf[2:-2])
```

Out[185]:

	precision	recall	f1-score	support	Model
accuracy	0.854995	0.854995	0.854995	0.854995	Logistic Regression Classifier

Decision Tree Classifier

Decision trees can be used for classification as well as regression problems. The name itself suggests that it uses a flowchart like a tree structure to show the predictions that result from a series of feature-based splits. It starts with a root node and ends with a decision made by leaves.

Decision trees are upside down which means the root is at the top and then this root is split into various several nodes. Decision trees are nothing but a bunch of if-else statements in layman terms. It checks if the condition is true and if it is then it goes to the next node attached to that decision.

In a Decision Tree diagram, we have:

Root Node: *The first split which decides the entire population or sample data should further get divided into two or more homogeneous sets. In our case, the Outlook node.*

Splitting: *It is a process of dividing a node into two or more sub-nodes.*

Decision Node: *This node decides whether/when a sub-node splits into further sub-nodes or not. Here we have, Outlook node, Humidity node, and Windy node.*

Leaf: *Terminal Node that predicts the outcome (categorical or continuous value).*

The coloured nodes, i.e., Yes and No nodes, are the leaves.

In [186…

```
#Importing libraries for DecisionTreeClassifier model.
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

In [187…

```
# Creating model object for DecisionTreeClassifier.
dt_clf = DecisionTreeClassifier()
```

In [188…

```
# Storing the hyperparameters in Dict
parameters = {'max_depth' : [4,6,8,10],
              'min_samples_split' : [10,20,30,40,50],
              'min_samples_leaf' : [10,15,20]}
```

In [189…

```
# Applying GridSearchCV for hyperparameter tuning.
dt_clf = GridSearchCV(dt_clf, parameters, scoring='roc_auc', cv=5)
```

In [190…

```
# Fitting the model
dt_clf.fit(x_train_trf,y_train)
```

```
Out[190]: ▸            GridSearchCV
          ▸ estimator: DecisionTreeClassifier
                ▸ DecisionTreeClassifier
```

```
In [191…
# Checking the best parameters
dt_clf.best_estimator_
```

```
Out[191]: ▾                 DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6, min_samples_leaf=15, min_samples_sp
lit=50)
```

```
In [192…
# Getting the predicted classes for training and testing set

train_dt_prediction = dt_clf.predict(x_train_trf)
test_dt_prediction = dt_clf.predict(x_test_trf)
```

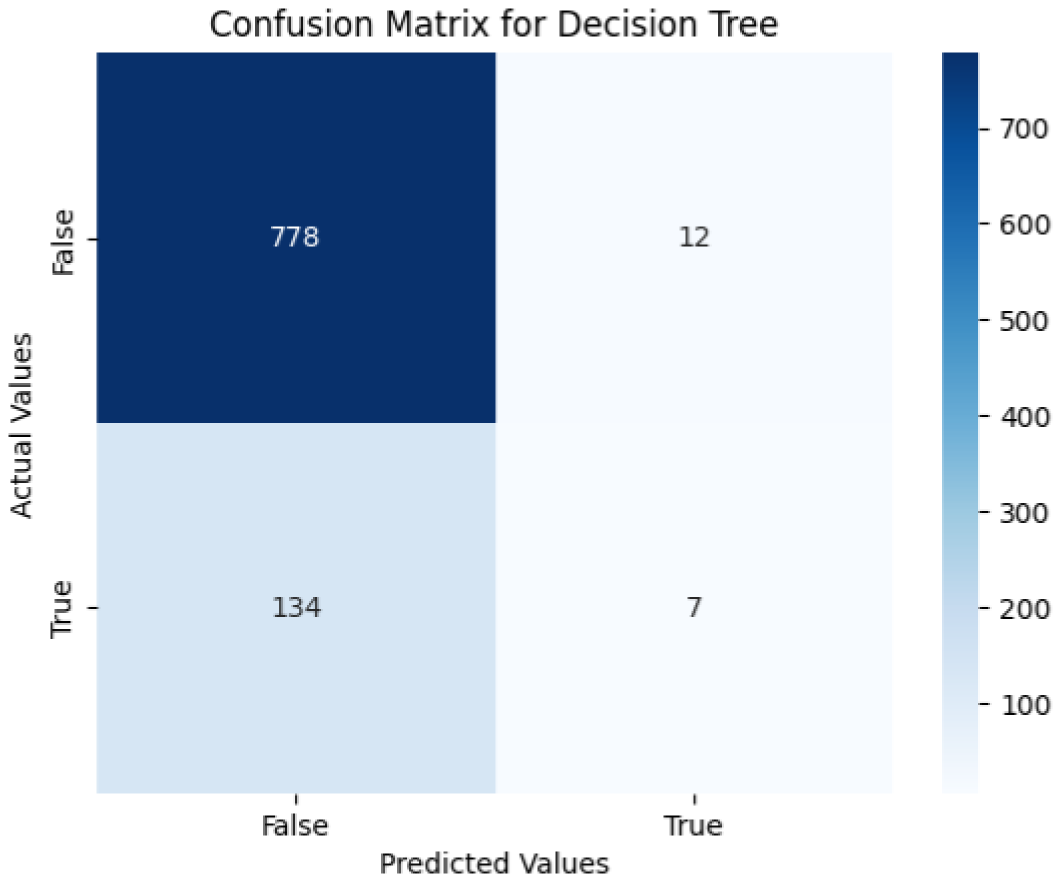
```
In [193…
# Getting the accuracy scores for training and testing set.

train_accuracy_dt = accuracy_score(train_dt_prediction, y_train)
test_accuracy_dt = accuracy_score(test_dt_prediction, y_test)

# Display accuracies.
print("The accuracy on train data is ", train_accuracy_dt)
print("The accuracy on test data is ", test_accuracy_dt)
```

```
The accuracy on train data is 0.8560171919770774 The
accuracy on test data is 0.8431793770139635
```

```
In [194…
# Confusion Matrix for random forest classifier.

dt_cf_matrix = confusion_matrix(y_test,test_dt_prediction)
dt_cf_matrix
```

```
Out[194]: array([[778,  12],
                 [134,   7]], dtype=int64)
```

```
In [195…
# Plotting the confusion matrix

labels = ['778','12','134','7']

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(dt_cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix for Decision
Tree'); ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values');

## Ticket labels - List must be in alphabetical
order ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion
Matrix. plt.show()
```



In [196…

```
# Predicted values.

y_pred_dt = dt_clf.predict(x_test_trf)
```

In [197…

```
# Getting classification report.

dict_2 = classification_report(y_test, y_pred_dt, output_dict = True)
```

In [198…

```
# Storing the scores in a dataframe
tempodf=pd.DataFrame(dict_2).transpose()
tempodf['Model'] = 'Decision Tree Classifier'
res_df=res_df.append(tempodf[2:-2])
res_df
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\756919794.py:4: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a
f uture version. Use pandas.concat instead.
  res_df=res_df.append(tempodf[2:-2])
```

Out[198]:

	precision	recall	f1-score	support	Model
accuracy	0.854995	0.854995	0.854995	0.854995	Logistic Regression Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	Decision Tree Classifier

Random Forest Classifier

Random Forest is a technique that uses ensemble learning, that combines many weak classifiers to provide solutions to complex problems.

As the name suggests random forest consists of many decision trees. Rather than depending on one tree it takes the prediction from each tree and based on the majority votes of predictions, predicts the final output.

Random forests use the bagging method. It creates a subset of the original dataset, and the final output is based on majority ranking and hence the problem of overfitting is taken care of.

In [205…

```
# Importing Necessary library

from sklearn.ensemble import RandomForestClassifier
```

In [206…

```
#Creating an instance for the random forest regressor.

rf_clf = RandomForestClassifier()
```

In [207…

```
# Storing the hyperparameters in Dict
params = {'n_estimators' : [750,
850], 'max_depth': [7,9],
          'max_features' : [7,8],
          'min_samples_leaf' : [2,3]}
```

In [208…

```
# Using GridSearchCV for hyperparameter tuning
cv = GridSearchCV(rf_clf, param_grid = params, scoring = 'roc_auc', cv =5)
```

In [209…

```
# Fitting the model
cv.fit(x_train_trf, y_train)
```

Out[209]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [210…

```
# Checking the best parameters
cv.best_estimator_
```

Out[210]:

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=7, max_features=8, min_samples_leaf=2,
                       n_estimators=850)
```

In [211…

```
# Getting the predicted classes for training and testing set

train_rf_prediction = cv.predict(x_train_trf)
test_rf_prediction = cv.predict(x_test_trf)
```

In [212…

```
# Getting the accuracy scores for training and testing set.

train_accuracy_rf = accuracy_score(train_rf_prediction, y_train)
test_accuracy_rf = accuracy_score(test_rf_prediction, y_test)

# Display accuracies.
```

```
print("The accuracy on train data is ", train_accuracy_rf)
print("The accuracy on test data is ", test_accuracy_rf)
```

```
The accuracy on train data is 0.8714183381088825 The
accuracy on test data is 0.8399570354457573
```

In [213…

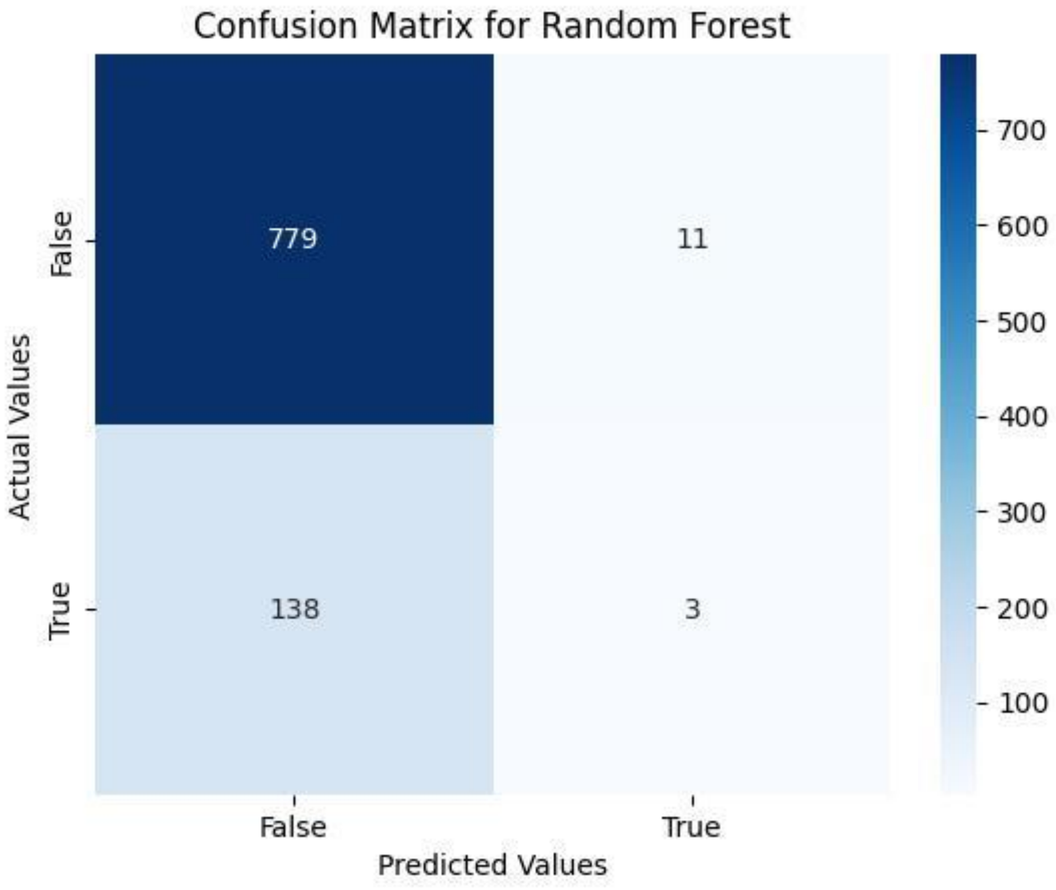
```
# Confusion Matrix for random forest classifier.

rf_cf_matrix = confusion_matrix(y_test,test_rf_prediction)
rf_cf_matrix
```

Out[213]:

```
array([[779,  11],
       [138,   3]], dtype=int64)
```

In [215…

```
# Plotting the confusion matrix

labels = ['779','11','138','3']

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(rf_cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix for Random
Forest'); ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values');

## Ticket labels - List must be in alphabetical
order ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion
Matrix. plt.show()
```



In [216…

```
# Predicted values.

y_pred_rf = cv.predict(x_test_trf)
```

In [217…

```
# Getting classification report.

dict_3 = classification_report(y_test, y_pred_rf, output_dict = True)
```

In [218…

```
# Storing the scores in a dataframe
tempodf=pd.DataFrame(dict_3).transpose()
tempodf['Model'] = 'Random Forest Classifier'
res_df=res_df.append(tempodf[2:-2])
res_df
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\530645543.py:4: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a
f uture version. Use pandas.concat instead.
  res_df=res_df.append(tempodf[2:-2])
```

Out[218]:

	precision	recall	f1-score	support	Model
accuracy	0.854995	0.854995	0.854995	0.854995	Logistic Regression Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	Decision Tree Classifier
accuracy	0.839957	0.839957	0.839957	0.839957	Random Forest Classifier

K-Nearest Neighbours Classifier

KNN which stands for K-Nearest Neighbours is a simple algorithm that is used for classification and regression problems in Machine Learning. KNN is also non-parametric which means the algorithm does not rely on strong assumptions instead tries to learn any functional form from the training data.

Unlike most of the algorithms with complex names, which are often confusing as to what they really mean, KNN is pretty straight forward. The algorithm considers the k nearest neighbours to predict the class or value of a data point.

The kNN 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.

In [219…

```
# Importing necessary library

from sklearn.neighbors import KNeighborsClassifier
```

In [220…

```
#Creating an instance for the KNN classifier.

KNN_clf = KNeighborsClassifier()
```

In [221…

```
# Using hyperparameter tuning to get the opimal value of n_neighbors

parameters = {'n_neighbors':np.arange(1,10)}
cv_knn = GridSearchCV(KNN_clf, cv = 5, param_grid = parameters)
```

In [222…

```
# Fitting the model

cv_knn.fit(x_train_trf, y_train)
```

Out[222]:

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

In [223…

```
# Checking the best parameter.

cv_knn.best_estimator_
```

Out[223]:

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=8)
```

In [224…

```
# Getting the predicted classes for training and testing set

train_knn_prediction = cv_knn.predict(x_train_trf)
test_knn_prediction = cv_knn.predict(x_test_trf)
```

In [225…

```
# Getting the accuracy scores for training and testing set.

train_accuracy_knn = accuracy_score(train_knn_prediction, y_train)
test_accuracy_knn = accuracy_score(test_knn_prediction, y_test)

# Display accuracies.

print("The accuracy on train data is ", train_accuracy_knn)
print("The accuracy on test data is ", test_accuracy_knn)
```

```
The accuracy on train data is 0.8553008595988538 The
accuracy on test data is 0.8431793770139635
```

In [226…

```
# Confusion Matrix for KNN classifier.

knn_cf_matrix = confusion_matrix(y_test,test_knn_prediction)
knn_cf_matrix
```

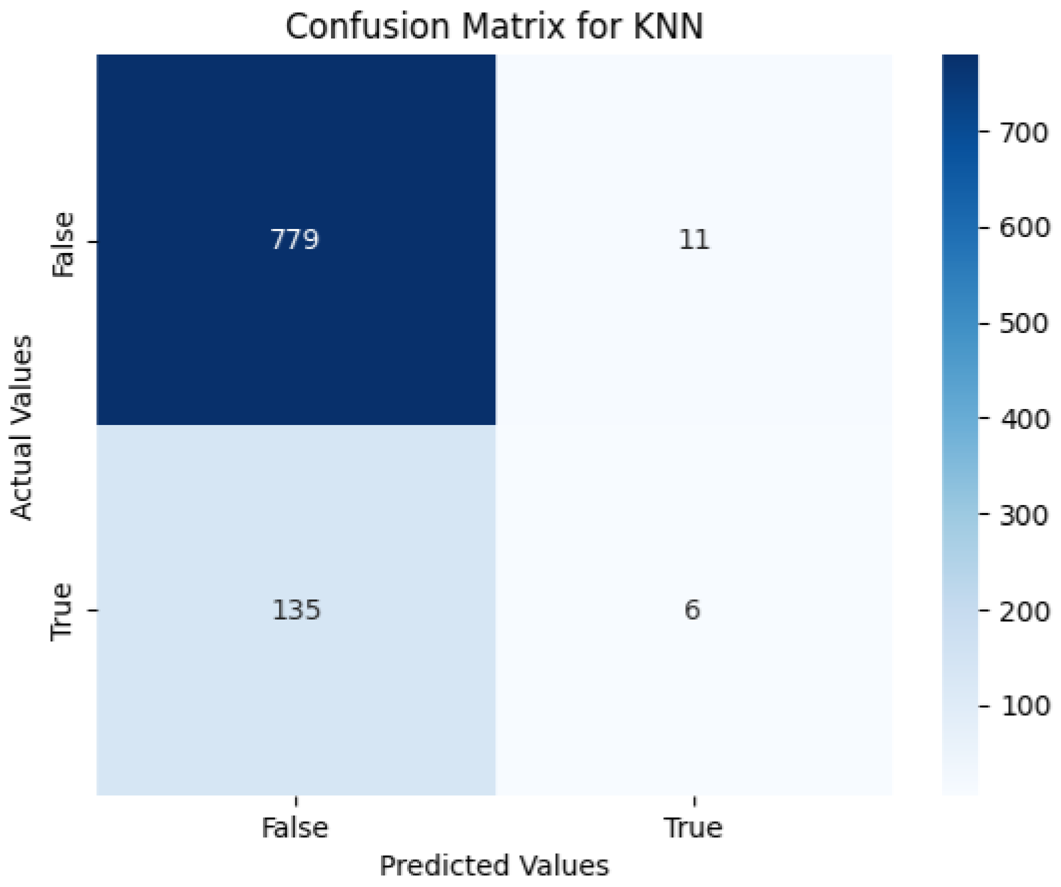
Out[226]:

```
array([[779,  11],
       [135,   6]], dtype=int64)
```

In [227…

```
# Plotting  the  confusion  matrix
labels = ['779','11','135','6']

labels = np.asarray(labels).reshape(2,2)
```

```
ax = sns.heatmap(knn_cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix for KNN');
ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical
order ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion
Matrix. plt.show()
```



In [228…

```
# Predicted values.

y_pred_KNN = cv_knn.predict(x_test_trf)
```

In [229…

```
# Getting classification report.

dict_4 = classification_report(y_test, y_pred_KNN, output_dict = True)
```

In [230…

```
# Storing the scores in a dataframe
tempodf=pd.DataFrame(dict_4).transpose()
tempodf['Model'] = 'K-Nearest Neighbours
Classifier' res_df=res_df.append(tempodf[2:-2])
res_df
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\1609648602.py:4:
FutureWarni ng: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  res_df=res_df.append(tempodf[2:-2])
```

Out[230]:

	precision	recall	f1-score	support	Model
accuracy	0.854995	0.854995	0.854995	0.854995	Logistic Regression Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	Decision Tree Classifier
accuracy	0.839957	0.839957	0.839957	0.839957	Random Forest Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	K-Nearest Neighbours Classifier

Gaussian Naive Bayes:

The Naïve Bayes algorithm is a classification technique based on the Bayes’ Theorem which assumes there is independence between the features. We interfere with applications utilizing this algorithm on a daily basis, for example it powers recommendation systems for streaming applications or adds on social media as well as many online retail websites.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Let’s have a look under the hood of this major classifier.

The simple form of the calculation for Bayes Theorem is as follows:

P(A|B) = P(B|A) * P(A) / P(B)

In [231…

```
# Necessary library

from sklearn.naive_bayes import GaussianNB
```

In [232…

```
# Creating model object for Naive Bayes Classifier

gnb = GaussianNB()
```

In [233…

```
# Fitting the model

gnb.fit(x_train_trf, y_train)
```

Out[233]:

```
▾ GaussianNB
GaussianNB()
```

In [234…

```
# Getting the predicted classes for training and testing set

train_class_preds_gnb = gnb.predict(x_train_trf)
test_class_preds_gnb = gnb.predict(x_test_trf)
```

In [235…

```
# Getting the accuracy scores for training and testing set.

train_accuracy_gnb = accuracy_score(train_class_preds_gnb, y_train)
test_accuracy_gnb = accuracy_score(test_class_preds_gnb, y_test)
```

```
# Display accuracies.

print("The accuracy on train data is ", train_accuracy_gnb)
print("The accuracy on test data is ", test_accuracy_gnb)
```

```
The accuracy on train data is 0.8277220630372493 The
accuracy on test data is 0.8195488721804511
```

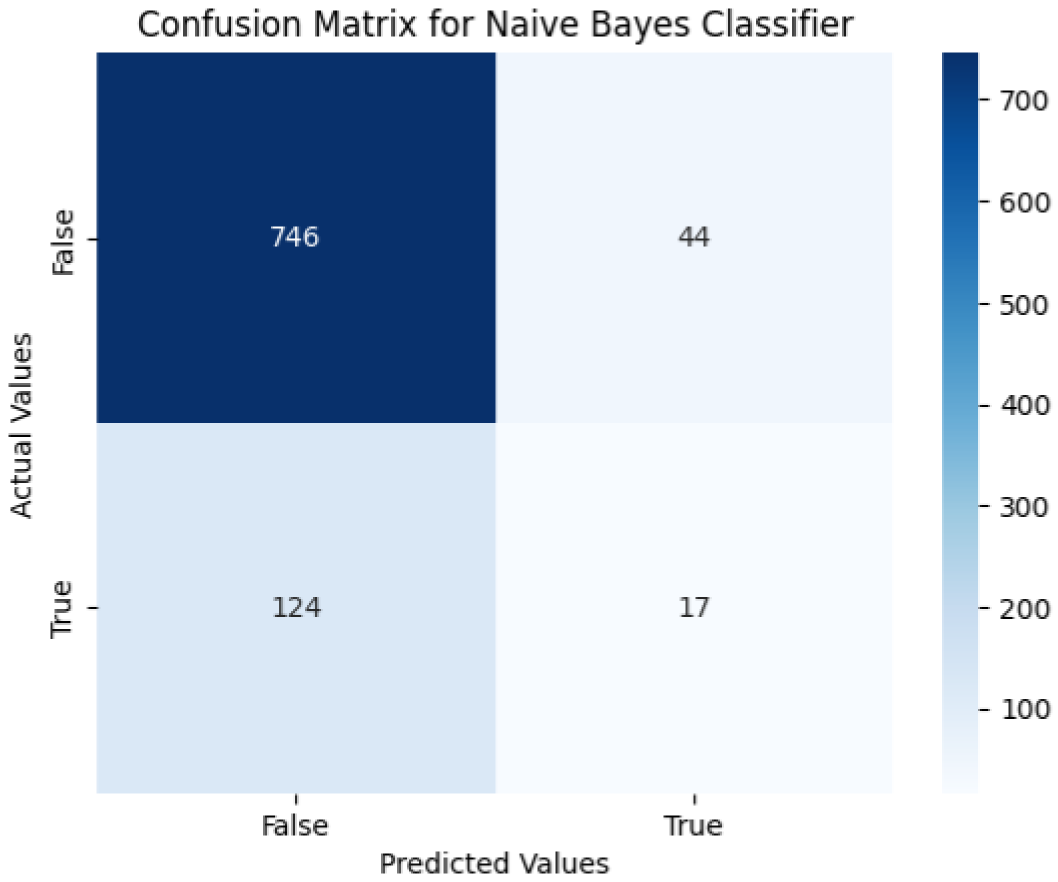
In [236…

```
# Confusion Matrix for logistic regression classifier.

cf_matrix_gnb = confusion_matrix(y_test,test_class_preds_gnb)
cf_matrix_gnb
```

```
Out[236]: array([[746,  44],
        [124,              17]], dtype=int64)
```

In [238…

```
# Plotting the confusion matrix
labels = ['746','44','124','17']

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(cf_matrix_gnb, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix for Naive Bayes
Classifier'); ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values');

## Ticket labels - List must be in alphabetical
order ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion
Matrix. plt.show()
```



In [239…

```
# Predicted values.

y_pred_gnb = gnb.predict(x_test_trf)
```

In [240…

```
# Getting classification report.

dict_5 = classification_report(y_test, y_pred_gnb, output_dict = True)
```

In [241…

```
# Storing the scores in a dataframe
tempodf=pd.DataFrame(dict_5).transpose()
tempodf['Model'] = 'Gaussian Naive Bayes
Classifier' res_df=res_df.append(tempodf[2:-2])
res_df
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\4273177268.py:4:
FutureWarni ng: The frame.append method is deprecated and will be removed
from pandas in a future version. Use pandas.concat instead.
  res_df=res_df.append(tempodf[2:-2])
```

Out[241]:

	precision	recall	f1-score	support	Model
accuracy	0.854995	0.854995	0.854995	0.854995	Logistic Regression Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	Decision Tree Classifier
accuracy	0.839957	0.839957	0.839957	0.839957	Random Forest Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	K-Nearest Neighbours Classifier
accuracy	0.819549	0.819549	0.819549	0.819549	Gaussian Naive Bayes Classifier

Support Vector Machine

SVM is a powerful supervised algorithm that works best on smaller datasets but on complex ones. Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks, but generally, they work best in classification problems.

Support Vectors: These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.

Margin: it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins hard margin and soft margin.

In [242…

```
# Importing Support vector machine Classifier.

from sklearn import svm
```

In [243…

```
# Create a svm Classifier

svmc = svm.SVC(kernel='poly')
```

In [244…

```
# Fitting the model with training set

svmc.fit(x_train_trf, y_train)
```

Out[244]:

```
▾ SVC
SVC(kernel='poly')
```

In [245…

```
# Getting the predicted classes for training and testing set

train_svm_prediction = svmc.predict(x_train_trf)
test_svm_prediction = svmc.predict(x_test_trf)
```

In [246…

```
# Getting the accuracy scores for training and testing set.

train_accuracy_svm = accuracy_score(train_svm_prediction, y_train)
test_accuracy_svm = accuracy_score(test_svm_prediction, y_test)

# Display accuracies.
print("The accuracy on train data is ", train_accuracy_svm)
print("The accuracy on test data is ", test_accuracy_svm)
```

```
The accuracy on train data is 0.8606733524355301 The
accuracy on test data is 0.8506981740064447
```

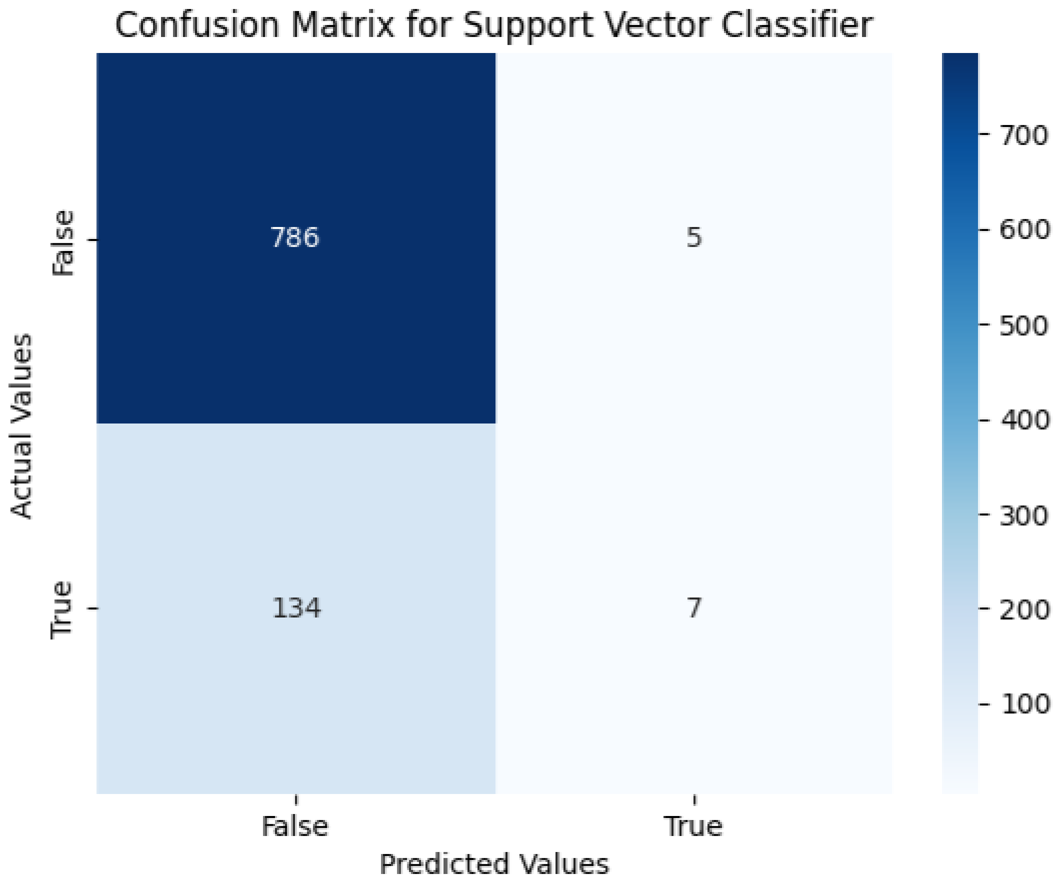
In [247…

```
# Confusion Matrix for random forest classifier.

svm_cf_matrix = confusion_matrix(y_test,test_svm_prediction)
svm_cf_matrix
```

```
Out[247]: array([[785,   5],
                 [134,   7]], dtype=int64)
```

In [252…

```
# Plotting the confusion matrix

labels = ['786','5','134','7']

labels = np.asarray(labels).reshape(2,2)

ax = sns.heatmap(svm_cf_matrix, annot=labels, fmt='', cmap='Blues')

ax.set_title('Confusion Matrix for Support Vector
Classifier'); ax.set_xlabel('Predicted Values')
ax.set_ylabel('Actual Values');

## Ticket labels - List must be in alphabetical
order ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion
Matrix. plt.show()
```



In [253…

```
# Predicted values.

y_pred_svm = svmc.predict(x_test_trf)
```

In [254…

```
# Getting classification report.

dict_7 = classification_report(y_test, y_pred_svm, output_dict = True)
```

In [255…

```
#Storing scores in a dataframe.
tempodf=pd.DataFrame(dict_7).transpose()
tempodf['Model'] = 'Support Vector Machine'
res_df=res_df.append(tempodf[2:-2])
res_df
```

```
C:\Users\agnisis\AppData\Local\Temp\ipykernel_7204\147926993.py:4: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a
f uture version. Use pandas.concat instead.
  res_df=res_df.append(tempodf[2:-2])
```

Out[255]:

	precision	recall	f1-score	support	Model
accuracy	0.854995	0.854995	0.854995	0.854995	Logistic Regression Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	Decision Tree Classifier
accuracy	0.839957	0.839957	0.839957	0.839957	Random Forest Classifier
accuracy	0.843179	0.843179	0.843179	0.843179	K-Nearest Neighbours Classifier
accuracy	0.819549	0.819549	0.819549	0.819549	Gaussian Naive Bayes Classifier
accuracy	0.850698	0.850698	0.850698	0.850698	Support Vector Machine
accuracy	0.850698	0.850698	0.850698	0.850698	Support Vector Machine

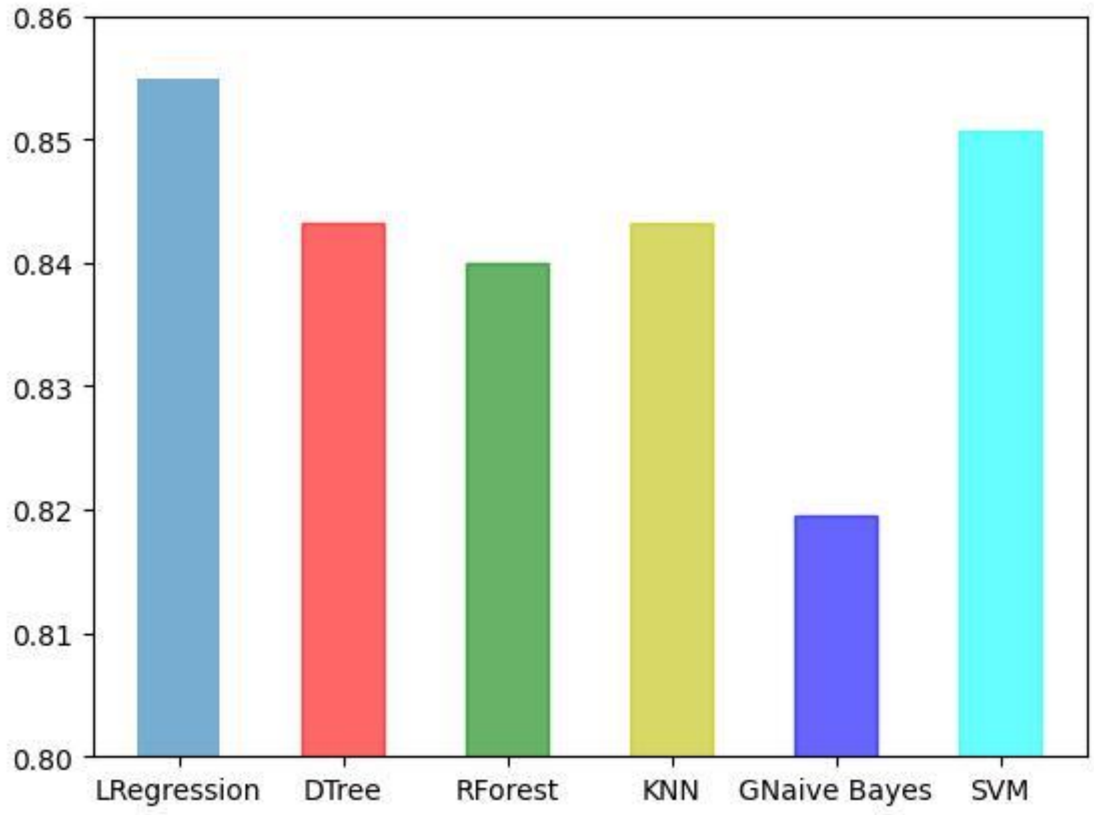
Comparing Models

In [ ]:

```
#comparing accuracies of six models
```

In [256…

```
models=["LRegression","DTree","RForest","KNN","GNaive Bayes","SVM"]
accuracies=[test_accuracy_lg,test_accuracy_dt,test_accuracy_rf,test_accuracy_knn

barlist=plt.bar(models,accuracies,width=0.5,alpha=0.6)

plt.ylim(0.8,0.86)

barlist[1].set_color('r')
barlist[2].set_color('g')
barlist[3].set_color('y')
barlist[4].set_color('b')
barlist[5].set_color('cyan')
```



Conclusions:

- *In conclusion, All the features provided in the dataset are extremely important and contribute towards the risk of getting CHDs. Although, we can conclude some majorly important features like:*
- *As age increases** the risk of getting diagnosed with **heart disease also increases**.*
- ***Cigarette consumption*** *is also a* ***major factor*** *that causes CHDs.*
- *Patients having Diabetes and cholesterol** problems show a **higher risk of CHDs**.*
- *Patients having high glucose levels** are **more prone to CHDs**.*
- *Patients with a history of “strokes”** have a **higher chance** of developing CHDs.*
- *Patients with high BMI(Body Mass Index)** are at **more risk of getting diagnosed with CHDs**.*
- *Finally we can say that, Logistic Regression Classifier** has performed best among all the models with the **accuracy** of **85%** & **f1-score** of **0.8527**.*

```
In [ ]:
```

Future Scope of Improvement

The field of AI-ML has already made significant contributions to stock market prediction, but there is still much room for improvement. Some potential areas of future development include:

- Improved data collection: More accurate and comprehensive data collection can help to improve the accuracy of AI-ML models. This could include incorporating alternative data sources such as social media sentiment analysis, news sentiment analysis, and satellite imagery.

- Enhanced algorithms: Advances in deep learning algorithms, reinforcement learning, and other machine learning techniques can improve the accuracy of stock market predictions. This could involve developing more advanced neural networks or using more complex modeling techniques.

- Better feature engineering: Feature engineering is the process of selecting the most relevant features or variables to include in a model. Better feature engineering can lead to more accurate predictions by identifying the most important factors that influence stock market movements.

- Explainability and interpretability: Stock market predictions generated by AI-ML models are often considered "black box" models, meaning it is difficult to understand how the model arrived at its prediction. Improving the explainability and interpretability of these models can increase their adoption by financial institutions and regulators.

- Incorporating market dynamics: Stock market predictions can be influenced by a variety of external factors, including global events, geopolitical tensions, and changes in interest rates. Future AI-ML models may be able to better incorporate these external factors into their predictions to provide more accurate forecasts.

- The automotive industry is one of the areas where Machine Learning is excelling by changing the definition of ‘safe’ driving. There are a few major companies such as Google, Tesla, Mercedes Benz, Nissan, etc. that have invested hugely in Machine Learning to come up with novel innovations. However, Tesla's self-driving car is the best in the industry. These self driving cars are built using Machine Learning, IoT sensors, high-definition cameras, voice recognition systems, etc.

- Robotics is one of the fields that always gain the interest of researchers as well as the common. In 1954, George Devol invented the first robot that was programmable and it was named Unimate. After that, in the 21st century, Hanson Robotics created the first AI-robot, Sophia. These inventions were possible with the help of Machine Learning and Artificial Intelligence.

- We are still at an infant state in the field of Machine Learning. There are a lot of advancements to achieve in this field. One of them that will take Machine Learning to the next level is Quantum Computing. It is a type of computing that uses the mechanical phenomena of quantum such as entanglement and superposition. By using the quantum phenomenon of superposition, we can create systems (quantum systems) that can exhibit multiple states at the same time. On the other hand, entanglement is the phenomenon where two different states can be referenced to each other. It helps in describing the correlation between the properties of a quantum system.

- As the name suggests computer vision gives a vision to a computer or a machine. Here comes into our minds what the Head of AI at Google, Jeff Dean, has once said, ' The progress we've made from 26% error in 2011 to 3% error in 2016 is hugely impactful. The way I like to think is, computers have now evolved eyes that work.

- The scope of Machine Learning in India, as well as in other parts of the world, is high in comparison to other career fields when it comes to job opportunities. According to Gartner, there will be 2.3 million jobs in the field of Artificial Intelligence and Machine Learning by 2022. Also, the salary of a Machine Learning Engineer is much higher than the salaries offered to other job profiles

CERTIFICATE

This is to certify that Mr. AGNISIS DUTTA of ASANSOL ENGINEERING COLLEGE, Roll no: 10800320053, ECE 3rd Year has successfully completed a project on Risk factor Prediction of Cardio-Vascular Heart Disease using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

CERTIFICATE

This is to certify that Miss. SHILPI SEN of

ASANSOL ENGINEERING COLLEGE, Roll no: 10800320070, ECE 3rd Year has successfully completed a project on Risk factor Prediction of Cardio-Vascular Heart Disease using Machine Learning with Python under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty