Data Dictionary:

The dataset, which is from an ongoing cardiovascular study on citizens of the town of Framingham, Massachusetts. It is openly accessible on the Kaggle website. Determining the patients' 10-year risk of developing Coronary Heart Disease (CHD) is the aim of classification.The patients' information is available in the dataset. It has sixteen features (including the target feature) and more than 4,000 records. Risk factors include behavioural, medical, and demographic aspects.

Attributes:

Demographic:

- Sex: Male or Female (binary)
- Age: Age of the patient: (Continuous - Although converted to whole numbers, but the concept of age is continuous)
- Education: No explicit information provided

Behavioral:

- Current Smoker: 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. (Considered continuous as one can have any number of cigarettes, even half.)

Information on medical history:

- BP Meds: Whether or not the patient was on blood pressure medication
- Prevalent Stroke: Whether or not the patient had previously had a stroke
- Prevalent Hyp: Whether or not the patient was hypertensive
- Diabetes: Whether or not the patient had diabetes

Information on current medical condition:

- 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)

Target variable:

- 10 year risk of Coronary Heart Disease (CHD) - (binary: “1” means “Yes”, “0” means “No”)

Import Necessary Libraries

In [1]:
```
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from scipy.stats import chi2_contingency
from scipy import stats
from sklearn.feature_selection import SelectKBest, f_classif, mutual_info_classif
from sklearn.metrics import (accuracy_score, classification_report,confusion_matrix, Co
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.tree import plot_tree
from sklearn.ensemble import AdaBoostClassifier
import matplotlib.image as mpimg
from sklearn.ensemble import BaggingClassifier
from mlxtend.feature_selection import SequentialFeatureSelector
from plotly import graph_objects as go
from sklearn.model_selection import RandomizedSearchCV
from plotly.subplots import make_subplots
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
```

Read CSV Dataset

In [2]:
```
# Read the file using the file path
dg = pd.read_csv(r"C:\Users\HP\Desktop\framingham.csv")
# print the first five lines (the head)
dg.head()
```

Out[2]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes
0	1	39	4.0	0	0.0	0.0	0	0	0
1	0	46	2.0	0	0.0	0.0	0	0	0
2	1	48	1.0	1	20.0	0.0	0	0	0

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes
3	0	61	3.0	1	30.0	0.0	0	1	0
4	0	46	3.0	1	23.0	0.0	0	0	0

In [3]:
```
# Get the shape of the dataset
dg.shape
```

Out[3]:
```
(4240, 16)
```

In [4]:
```
#Get info about the dataset- the datatype
dg.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 [5]:
```
# Check for missing values
dg.isna().sum()
```

Out[5]:
```
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
```

Data Preprocessing

Cleaning the Dataset using Interpolation, Forward and Backward Filling Methods

Cleaning the Education Column

In [6]:
```
# Check the unique value of education feature
dg["education"].unique()
```

Out[6]:
```
array([ 4.,  2.,  1.,  3., nan])
```

In [7]:
```
# Check the sum of null values
dg["education"].isnull().sum()
```

Out[7]:
```
105
```

In [8]:
```
# Check for empty strings
dg[dg["education"]== ("" or " ")]
```

Out[8]:

male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes

In [9]:
```
# Fill the missing values using forward filling method
dg["education"] = dg["education"].fillna(method='ffill')
```

Cleaning the "cigsPerDay" Column

In [10]:
```
#Check the unique value of cigsPerDay column
dg["cigsPerDay"].unique()
```

Out[10]:
```
array([ 0., 20., 30., 23., 15.,  9., 10.,  5., 35., 43.,  1., 40.,  3.,
        2., nan, 12.,  4., 18., 25., 60., 14., 45.,  8., 50., 13., 11.,
        7.,  6., 38., 29., 17., 16., 19., 70.])
```

In [11]:
```
#Check the sum of null values
dg["cigsPerDay"].isnull().sum()
```

Out[11]:
```
29
```

In [12]:
```
#Fill the missing values by the interpolation method
dg["cigsPerDay"].interpolate(method="linear", limit_direction= "both",inplace= True)
```

Cleaning the "BPMeds" Column

In [13]:
```
#Check the unique value of BPMeds column
dg["BPMeds"].unique()
```

Out[13]:
```
array([ 0.,  1., nan])
```

In [14]:
```
#Check the sum of null values
dg["BPMeds"].isnull().sum()
```

Out[14]:
```
53
```

In [15]:
```
#Fill the missing values using backward method
dg["BPMeds"] = dg["BPMeds"].fillna(method='bfill')
```

Cleaning the 'totChol' Column

In [16]:
```
dg['totChol'].unique()
```

Out[16]:
```
array([195., 250., 245., 225., 285., 228., 205., 313., 260., 254., 247.,
       294., 332., 226., 221., 232., 291., 190., 185., 234., 215., 270.,
       272., 295., 209., 175., 214., 257., 178., 233., 180., 243., 237.,
        nan, 311., 208., 252., 261., 179., 194., 267., 216., 240., 266.,
       255., 220., 235., 212., 223., 300., 302., 248., 200., 189., 258.,
       202., 213., 183., 274., 170., 210., 197., 326., 188., 256., 244.,
       193., 239., 296., 269., 275., 268., 265., 173., 273., 290., 278.,
       264., 282., 241., 288., 222., 303., 246., 150., 187., 286., 154.,
       279., 293., 259., 219., 230., 320., 312., 165., 159., 174., 242.,
       301., 167., 308., 325., 229., 236., 224., 253., 464., 171., 186.,
       227., 249., 176., 163., 191., 263., 196., 310., 164., 135., 238.,
       207., 342., 287., 182., 352., 284., 217., 203., 262., 129., 155.,
       323., 206., 283., 319., 304., 340., 328., 280., 368., 218., 276.,
       339., 231., 198., 177., 201., 277., 184., 199., 168., 292., 305.,
       306., 152., 161., 181., 251., 271., 370., 439., 145., 330., 157.,
       398., 162., 314., 166., 160., 281., 289., 355., 307., 156., 329.,
       143., 211., 298., 334., 192., 204., 318., 309., 353., 360., 335.,
       158., 372., 346., 169., 140., 324., 600., 315., 392., 322., 149.,
       137., 172., 317., 358., 153., 345., 391., 410., 297., 356., 338.,
       107., 148., 366., 333., 327., 344., 126., 365., 362., 316., 144.,
```

```
       351., 390., 321., 405., 359., 350., 336., 380., 299., 124., 371.,
       113., 354., 382., 364., 341., 133., 367., 432., 337., 696., 363.,
       331., 361., 453., 347., 373., 385., 119.])
```

In [17]:
```
#Check the sum of null values
dg['totChol'].isnull().sum()
```

Out[17]: 50

In [18]:
```
#Fill the missing values using backward method
dg['totChol'] = dg['totChol'].fillna(method='bfill')
```

Cleaning the "BMI" Column

In [19]:
```
dg['BMI'].unique()
```

Out[19]: array([26.97, 28.73, 25.34, ..., 26.7 , 43.67, 20.91])

In [20]:
```
#Check the sum of null values
dg['BMI'].isnull().sum()
```

Out[20]: 19

In [21]:
```
#Fill the missing values using interpolation method
dg['BMI'].interpolate(method="linear", limit_direction= "both", inplace= True)
```

Cleaning the "Glucose" Column

In [22]:
```
dg['glucose'].unique()
```

Out[22]:
```
array([ 77.,  76.,  70., 103.,  85.,  99.,  78.,  79.,  88.,  61.,  64.,
        84.,  nan,  72.,  89.,  65., 113.,  75.,  83.,  66.,  74.,  63.,
        87., 225.,  90.,  80., 100., 215.,  98.,  62.,  95.,  94.,  55.,
        82.,  93.,  73.,  45., 202.,  68.,  97., 104.,  96., 126., 120.,
       105.,  71.,  56.,  60., 117., 102.,  58.,  92., 109.,  86., 107.,
        54.,  67.,  69.,  57.,  91., 132., 150.,  59.,  81., 115., 140.,
       112., 118., 143., 114., 160., 110., 123., 108., 145., 122., 137.,
       106., 127., 205., 130., 101.,  47.,  53., 216., 163., 144., 116.,
       121., 172., 124., 111.,  40., 186., 223., 325.,  44., 156., 268.,
        50., 274., 292., 255., 136., 206., 131., 148., 297.,  43., 173.,
        48., 386., 155., 147., 170.,  52., 320., 254., 394., 270., 244.,
       183., 142., 119., 135., 167., 207., 129., 177., 250., 294., 166.,
       125., 332., 368., 348., 248., 370., 193., 191., 256., 235., 210.,
       260.])
```

In [23]:
```
#Check the sum of null values
dg['glucose'].isnull().sum()
```

Out[23]: 388

In [24]:
```
#Fill the missing values using forward fill method
dg['glucose'] = dg['glucose'].fillna(method='ffill')
```

Cleaning the "HeartRate" Column

In [25]:
```
dg['heartRate'].unique()
```

Out[25]:
```
array([ 80.,  95.,  75.,  65.,  85.,  77.,  60.,  79.,  76.,  93.,  72.,
        98.,  64.,  70.,  71.,  62.,  73.,  90.,  96.,  68.,  63.,  88.,
        78.,  83., 100.,  67.,  84.,  57.,  50.,  74.,  86.,  55.,  92.,
        66.,  87., 110.,  81.,  56.,  89.,  82.,  48., 105.,  61.,  54.,
        69.,  52.,  94., 140., 130.,  58., 108., 104.,  91.,  53.,  nan,
       106.,  59.,  51., 102., 107., 112., 125., 103.,  44.,  47.,  45.,
        97., 122., 120.,  99., 115., 143., 101.,  46.])
```

In [26]:
```
#Check the sum of null values
dg['heartRate'].isnull().sum()
```

Out[26]:
```
1
```

In [27]:
```
#Fill the missing values using mean method
mean = dg['heartRate'].mean()
dg['heartRate'] = dg['heartRate'].fillna(dg['heartRate'].mean())
```

Check again for Null Values

In [28]:
```
dg.isnull().sum()
```

Out[28]:
```
male               0
age                0
education          0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
dtype: int64
```

Rename Column 'Male' with 'Sex'

In [29]:
```
# Rename the feature 'male' with 'sex'
dg = dg.rename(columns={'male': 'sex'})
dg.columns
```

Out[29]:
```
Index(['sex', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD'],
      dtype='object')
```

In [30]:
```
# Copy the cleaned dataset
df = dg.copy()
dd = dg.copy()
dt = dg.copy()
ds = dg.copy()
```

Description of the Features in Dataset

In [31]:
```
# Description of the dataset
dg.describe()
```

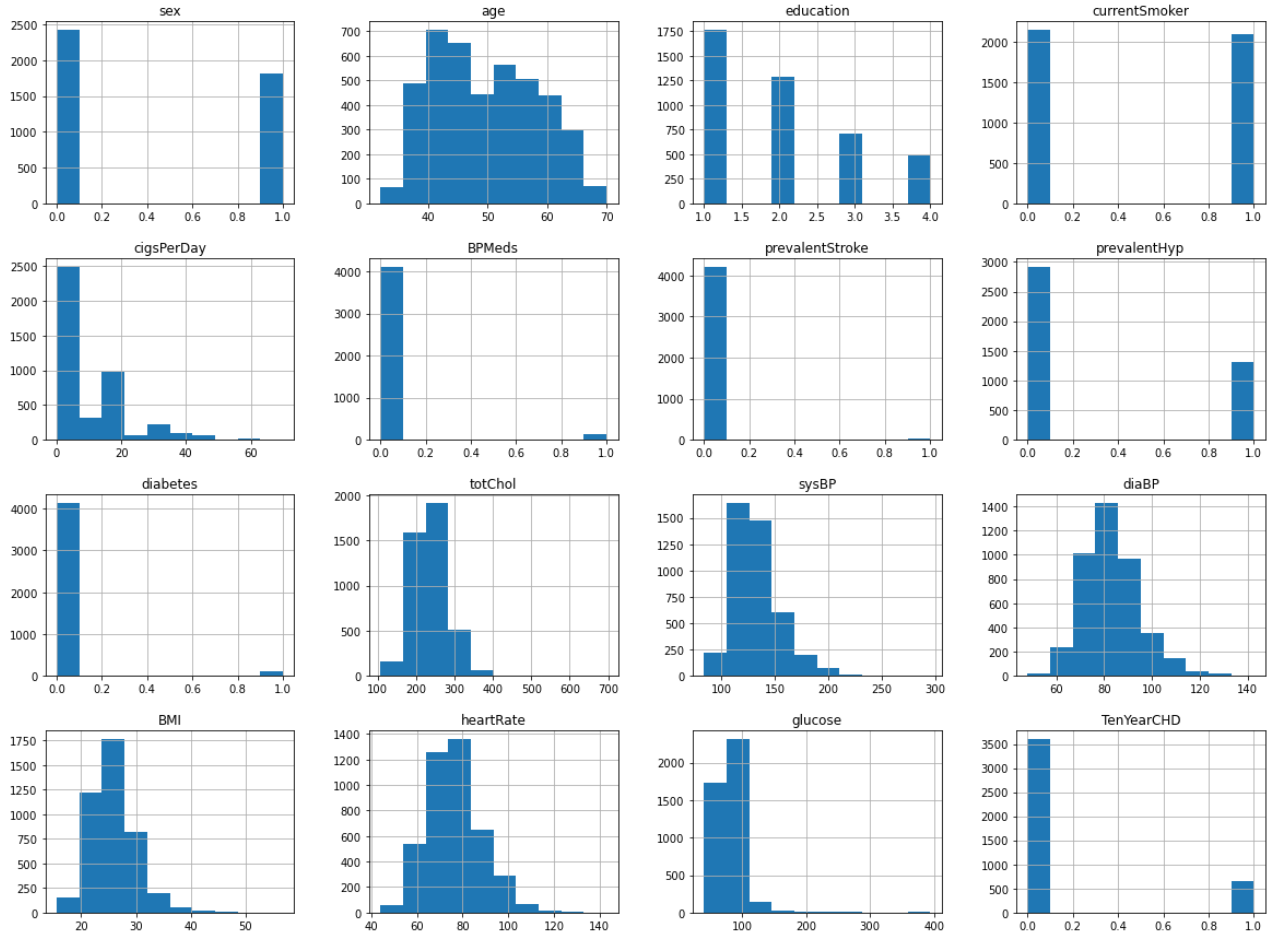
Out[31]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke
count	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000
mean	0.429245	49.580189	1.981368	0.494104	9.009316	0.029717	0.005896
std	0.495027	8.572942	1.020378	0.500024	11.916453	0.169825	0.076569
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.000000	0.000000
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.000000	1.000000

Distribution of the Features in the Dataset

In [32]:
```
plt.figure(dpi = 120)
dg.hist(figsize = (20, 15))
plt.show()
```

```
<Figure size 720x480 with 0 Axes>
```



Ratio of the Percentage of Males and Females

In [33]:
```
#Count the number of female and male patients
count_female = len(dg[dg['sex'] == 0])
count_male = len(dg[dg['sex'] == 1])

# Calculate the percentage of female and male patients
total_patients = len(dg)
percentage_female = (count_female / total_patients) * 100
percentage_male = (count_male / total_patients) * 100

# Print the percentages
print("Percentage of Females: {:.2f}%".format(percentage_female))
print("Percentage of Males: {:.2f}%".format(percentage_male))
```

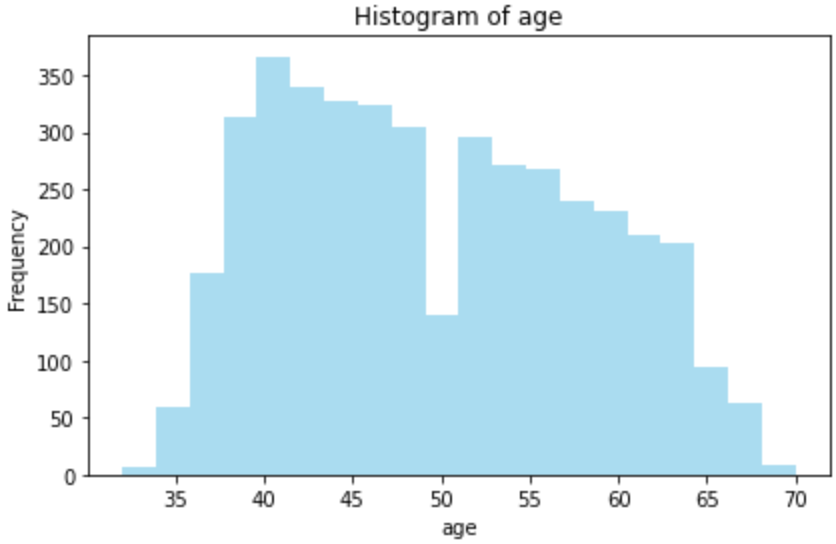
```
Percentage of Females: 57.08%
Percentage of Males: 42.92%
```

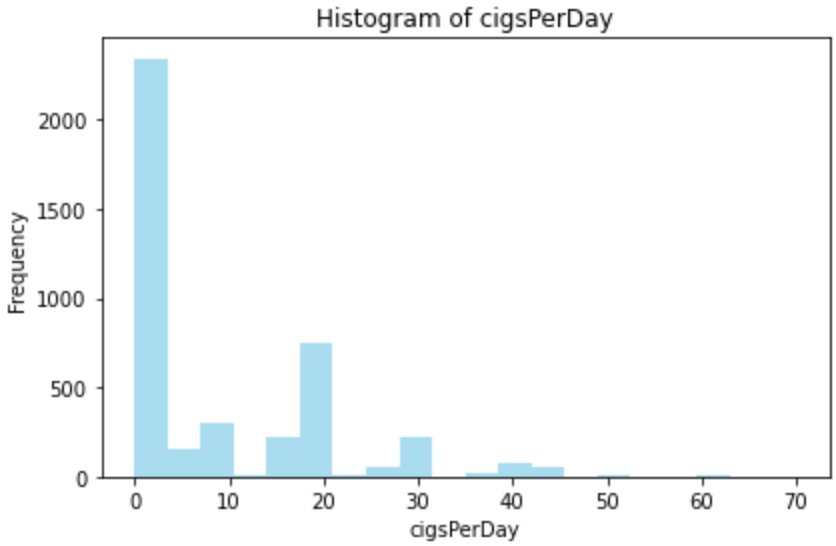
Taking a Closer Look into the Non-Binary Features: Histogram

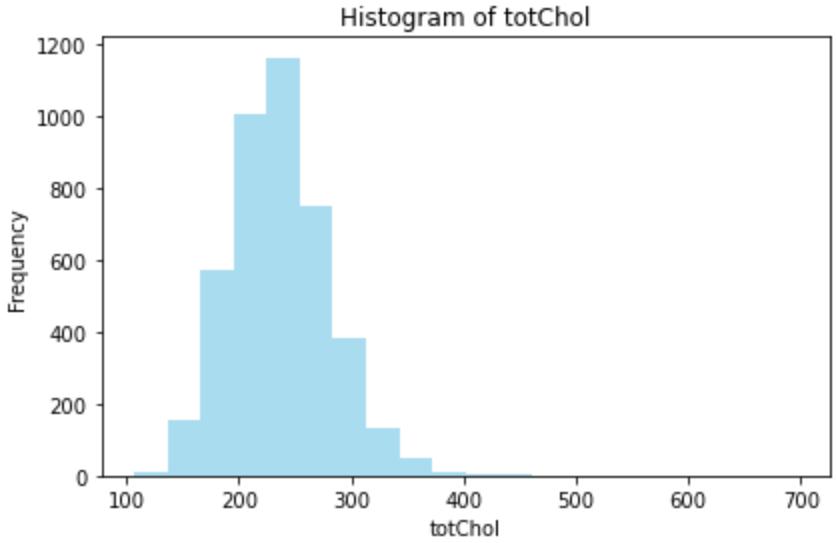
For the purpose of Data Transformation

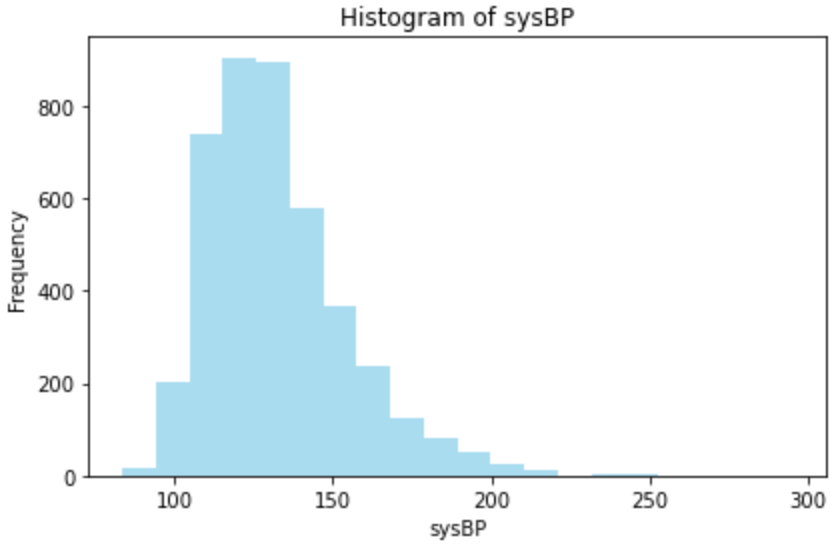
In [34]:
```
# create a list for the Non-Binary Features
columns = ['age', 'cigsPerDay', 'totChol', 'sysBP', 'diaBP', 'BMI', 'glucose']
```

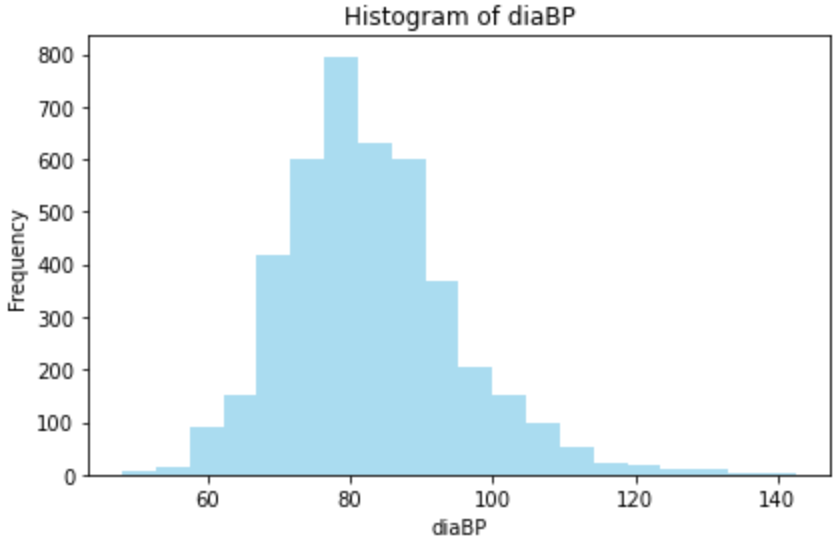
```
# Create histogram chart for the features
for column in columns:
    plt.figure(figsize=(6, 4))
    plt.hist(dg[column], bins=20, color='skyblue', alpha=0.7)
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```

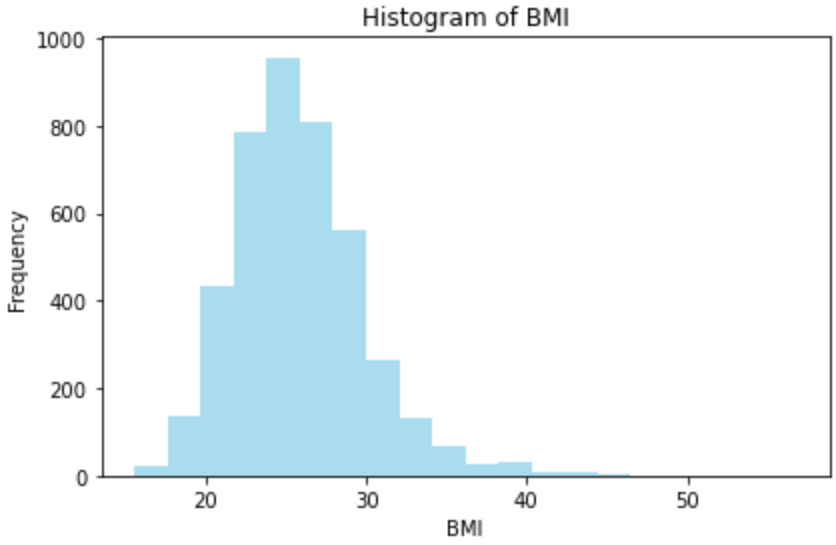


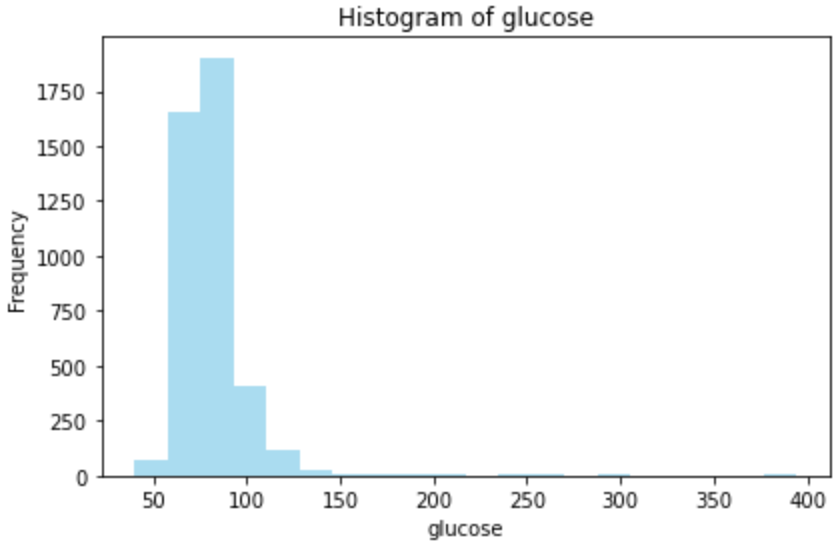



Histogram of totChol
Frequency
totChol
0
200
400
600
800
1000
1200
100
200
300
400
500
600
700


Histogram of sysBP
Frequency
sysBP
0
200
400
600
800
100
150
200
250
300


Histogram of diaBP
Frequency
diaBP
0
100
200
300
400
500
600
700
800
60
80
100
120
140





Data Transformation - for Continuous Features

Age

In [35]:
```
# Create bin for age feature
age_bins = [32, 37, 42, 47, 52, 57, 61, 70]
age_intervals = ['32-36', '37-41', '42-46', '47-51', '52-56', '57-61', '62+']
dg["age_range"] = pd.cut(dg["age"], bins = age_bins, labels = age_intervals)
```

Cigarretes Per Day

In [36]:
```
# Create bin for cigarretes feature
cig_bins = [1, 6, 12, 17, 22, 27, 31, 37, 40, 42]
cig_intervals = ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30', '30-35', '36-40', '
dg["cig_per_day_range"] = pd.cut(dg["cigsPerDay"], bins = cig_bins, labels = cig_interv
```

Total Cholesterol

In [37]:
```
# Create bin for total cholesterol feature
chol_bins = [100, 151, 201, 251, 301, 351, 401, 405]
chol_intervals = ['100-150', '151-200', '201-250', '251-300', '301-350', '351-400', '40
dg["totchol_range"] = pd.cut(dg["totChol"], bins = chol_bins, labels = chol_intervals)
```

Systolic Blood Pressure

In [38]:
```
# Create bin for systolic blood pressure feature
sys_bins = [80, 106, 126, 151, 176, 200, 201]
sys_intervals = ['80-105', '106-125', '126-150', '151-175', '176-200', '201+']
dg["sysBP_range"] = pd.cut(dg["sysBP"], bins = chol_bins, labels = chol_intervals)
```

BMI

In [39]:
```
# Create bin for bmi feature
bmi_bins = [0, 22, 27, 30, 35]
bmi_intervals = ['0-20', '21-25', '26-30', '30+']
dg["bmi_range"] = pd.cut(dg["BMI"], bins = bmi_bins, labels = bmi_intervals)
```

Glucose

In [40]:
```
# Create bin for glucose feature
glu_bins = [42, 57, 67, 77, 101, 102]
glu_intervals = ['40-55', '56-65', '66-75', '76-100', '101+']
dg["glu_range"] = pd.cut(dg["glucose"], bins = glu_bins, labels = glu_intervals)
```

Diastolic Blood Pressure

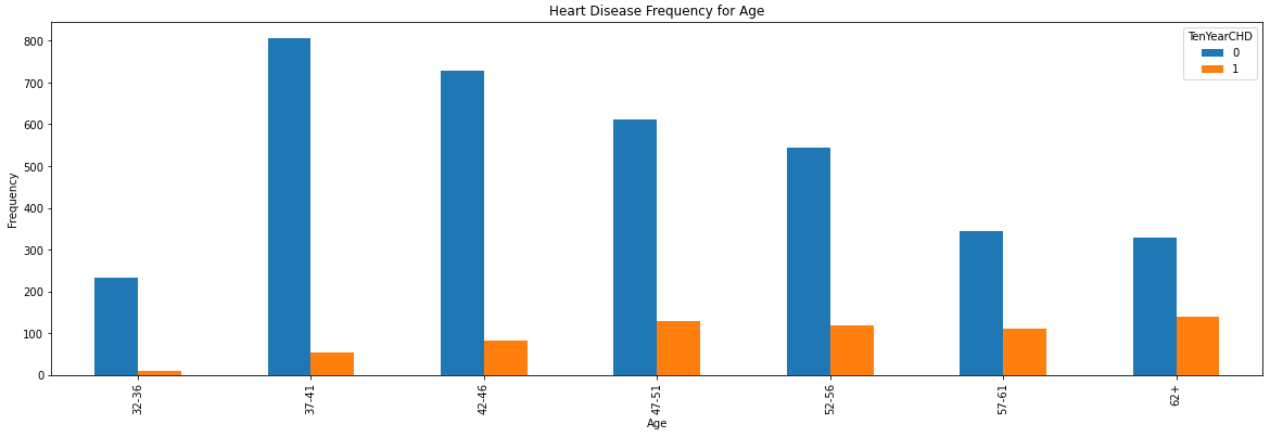
In [41]:
```
# Create bin for diastolic blood pressure feature
dia_bins = [55, 65, 75, 85, 95, 105, 150]
dia_intervals = ['48-60', '61-70', '71-80', '81-90', '91-100', '101+']
dg["diaBP_range"] = pd.cut(dg["diaBP"], bins = dia_bins, labels = dia_intervals)
```

Relationship between Target and Individual Features

Age Range

In [42]:
```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.age_range, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6))

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Age')
plt.xlabel('Age')
```

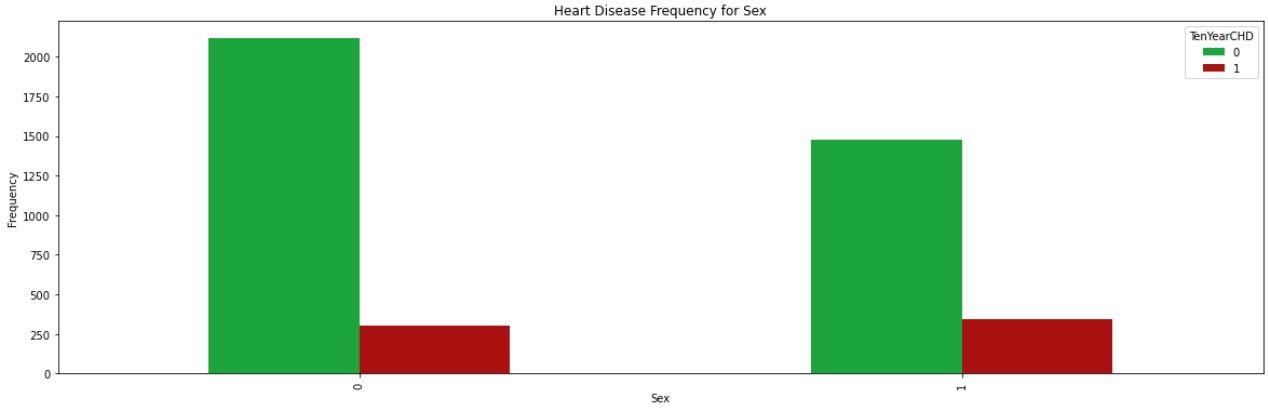
```
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndAges.png')

# Display the plot
plt.show()
```



Sex

In [43]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.sex, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6), color=['#1CA53B','

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Sex')
plt.xlabel('Sex')
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndSex.png')

# Display the plot
plt.show()
```

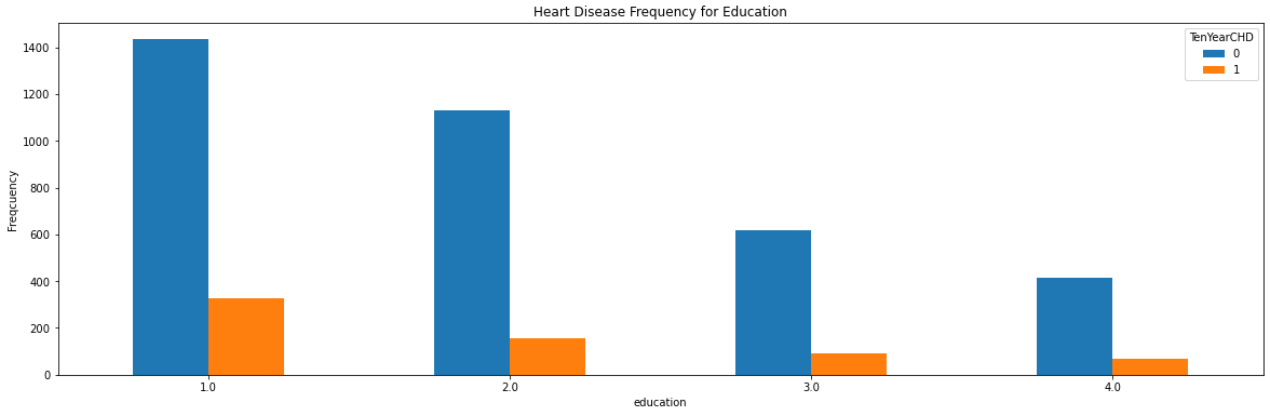


Education

In [44]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.education, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6))
```

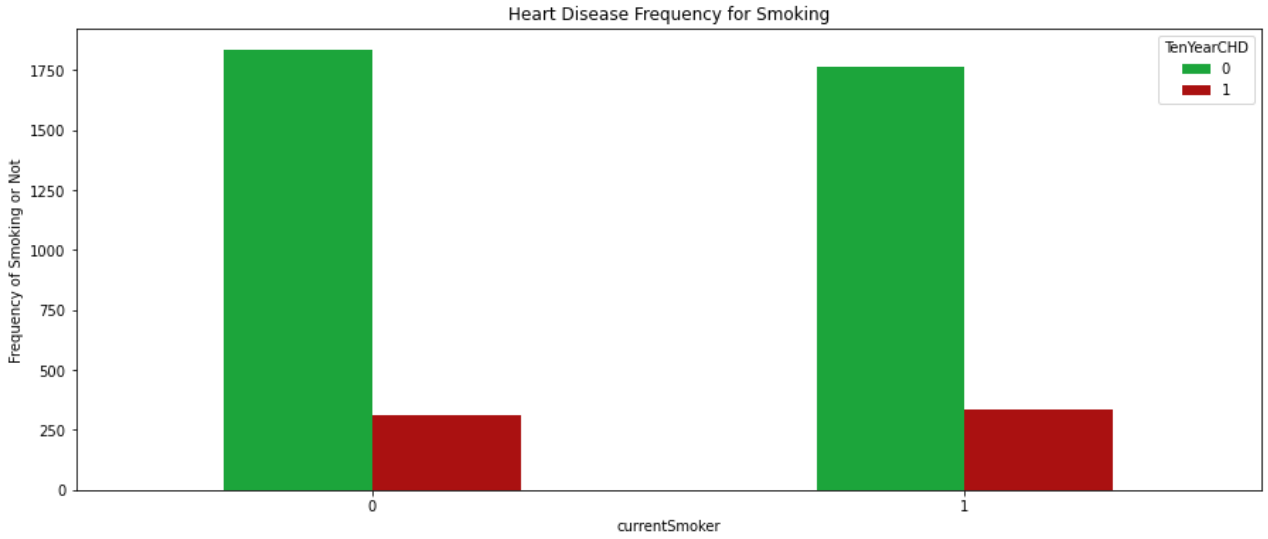
```
# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Education')
plt.xticks(rotation = 0)
plt.ylabel('Freqcuency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndEducation.png')

# Display the plot
plt.show()
```



Current Smoker

In [45]:

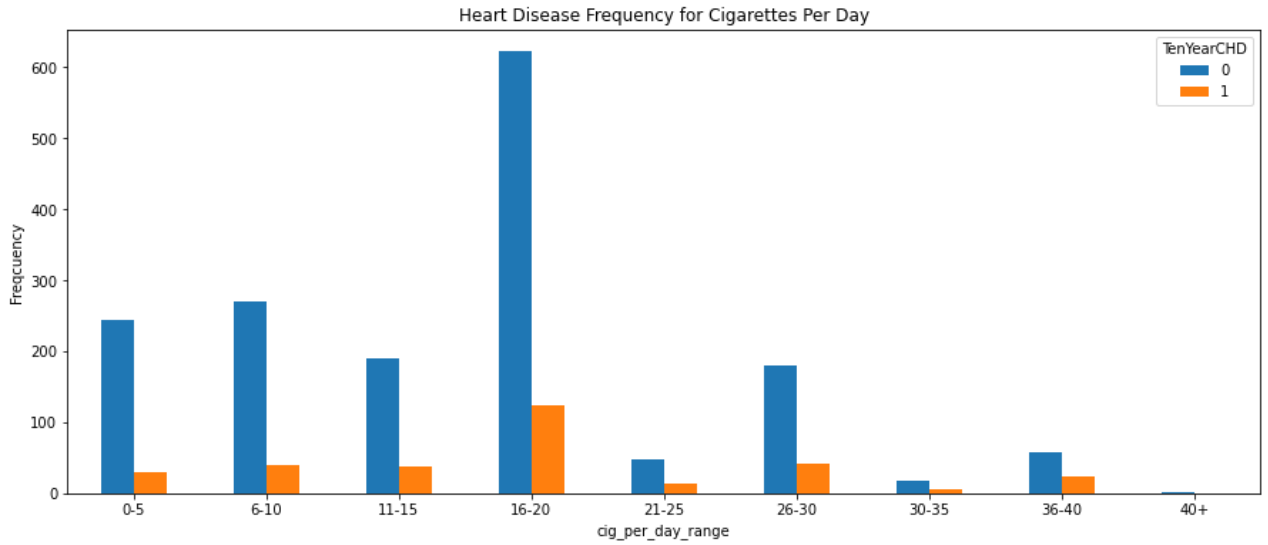
```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.currentSmoker, dg.TenYearCHD).plot(kind="bar",figsize=(15,6),color=['#1C

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Smoking')
plt.xticks(rotation = 0)
plt.ylabel('Frequency of Smoking or Not')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndCurrentSmoker.png')

# Display the plot
plt.show()
```



Cigarettes Per Day

In [46]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.cig_per_day_range, dg.TenYearCHD).plot(kind="bar",figsize=(15,6))

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Cigarettes Per Day')
plt.xticks(rotation = 0)
plt.ylabel('Freqcuency')

# Save the plot as an image file
#plt.savefig('cig_per_day')

# Display the plot
plt.show()
```

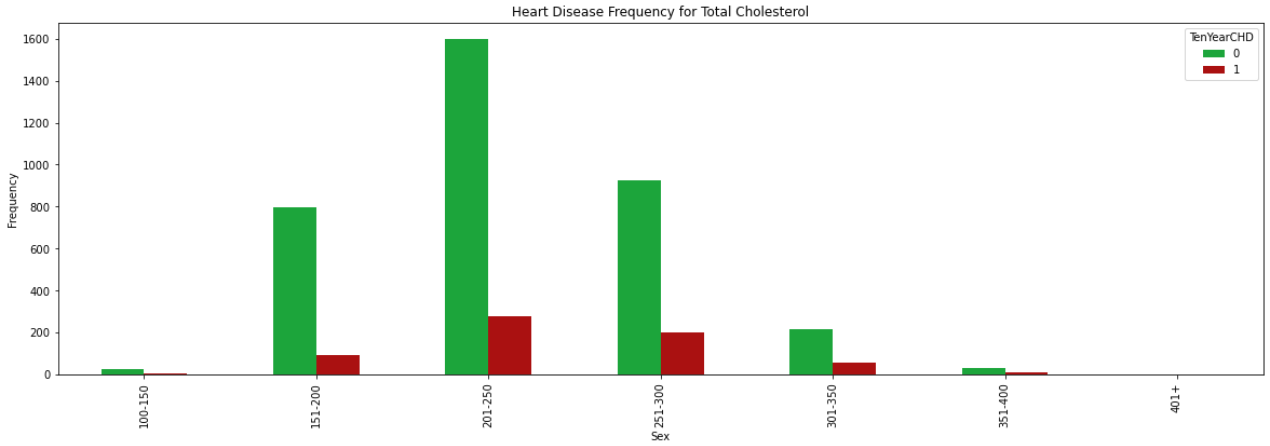


Total Cholesterol

In [47]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.totchol_range, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6), color=['
```

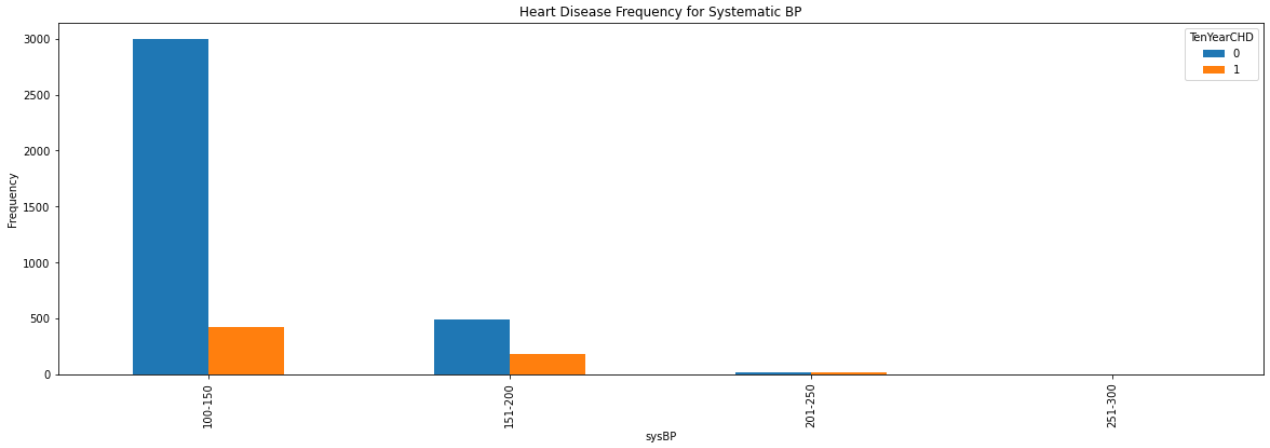
```
# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Total Cholesterol')
plt.xlabel('Sex')
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndTotalCholesterol.png')

# Display the plot
plt.show()
```



Systematic Blood Pressure

In [48]:

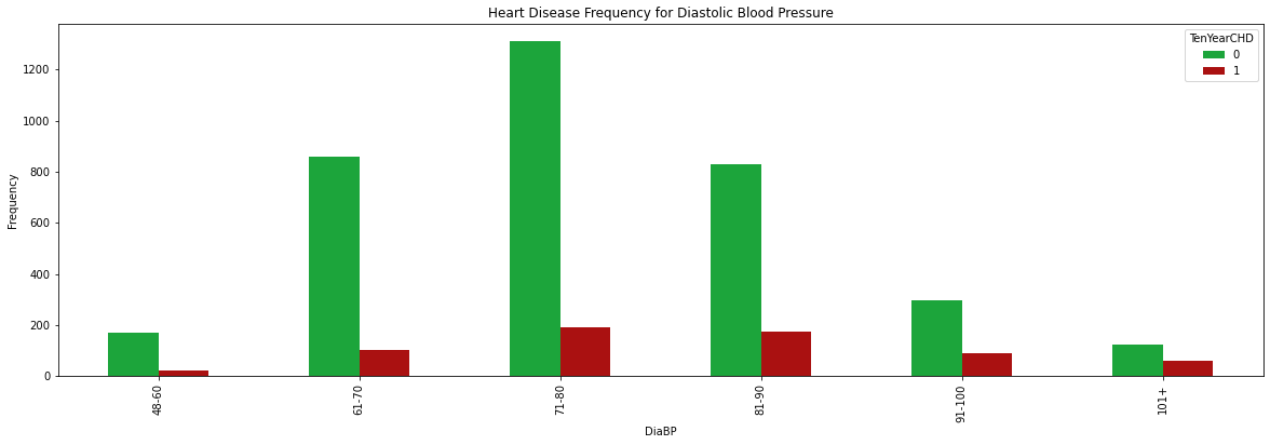
```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.sysBP_range, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6))

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Systematic BP')
plt.xlabel('sysBP')
plt.ylabel('Frequency')

# Save the plot as an image file (optional)
#plt.savefig('heartDiseaseAndSystolicBP.png')

# Display the plot
plt.show()
```



Diastolic Blood Pressure

In [49]:

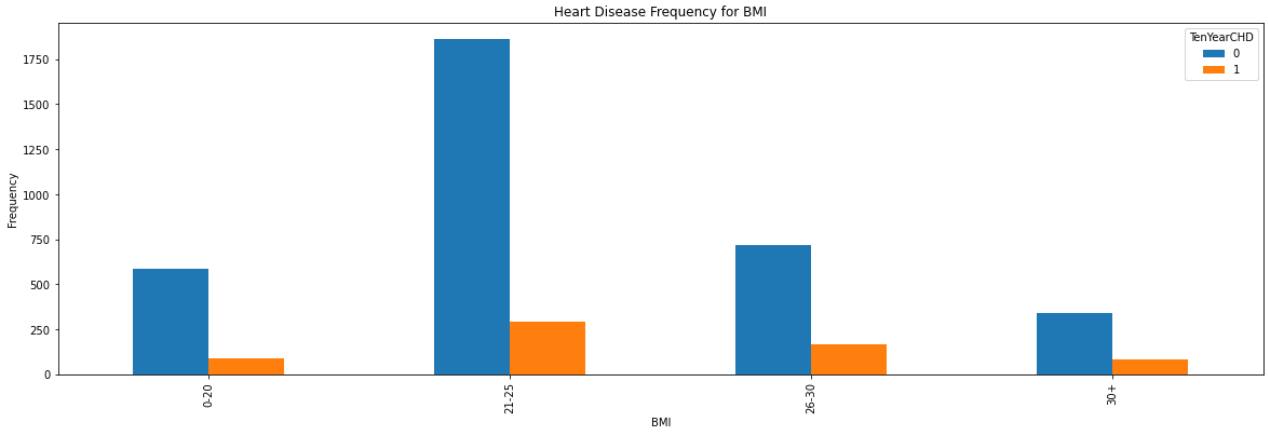
```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.diaBP_range, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6), color=['#1

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Diastolic Blood Pressure')
plt.xlabel('DiaBP')
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndDiastolicBP.png')

# Display the plot
plt.show()
```



BMI

In [50]:

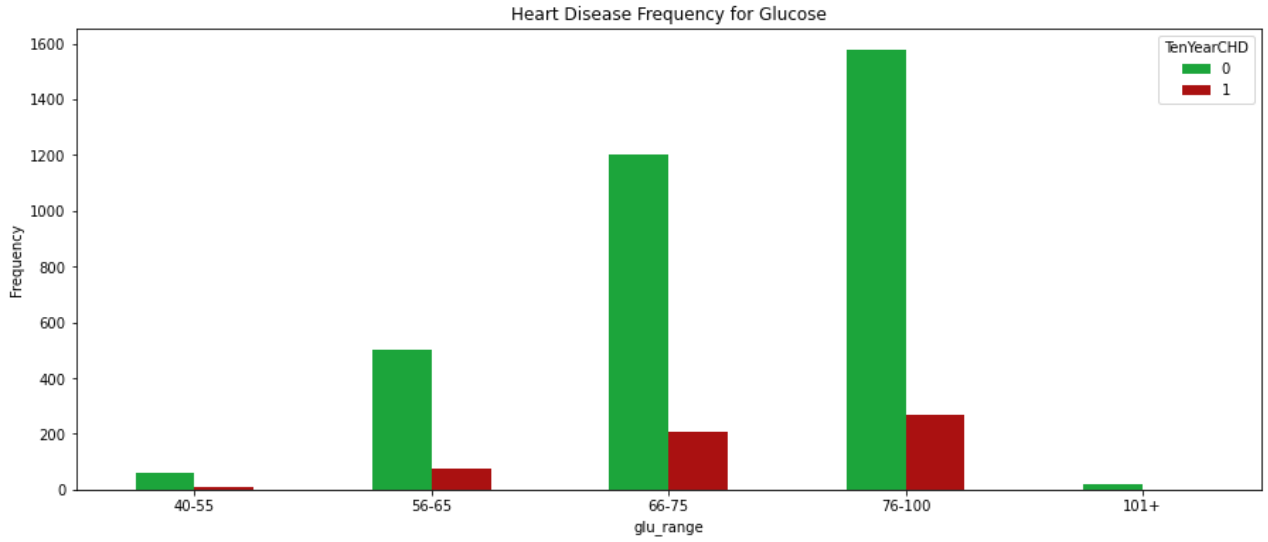
```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.bmi_range, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6))

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for BMI')
plt.xlabel('BMI')
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndBMI.png')

# Display the plot
plt.show()
```



Glucose

In [51]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.glu_range, dg.TenYearCHD).plot(kind="bar",figsize=(15,6),  color=['#1CA5

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Glucose')
plt.xticks(rotation = 0)
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndGlucose.png')

# Display the plot
plt.show()
```

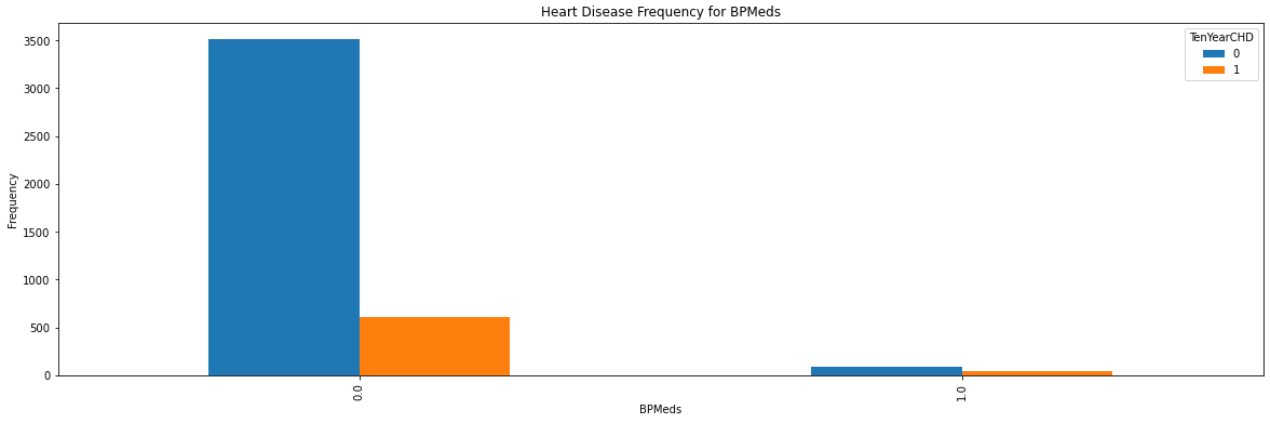


Blood Pressure Medications

In [52]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.BPMeds, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6))

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for BPMeds')
plt.xlabel('BPMeds')
```

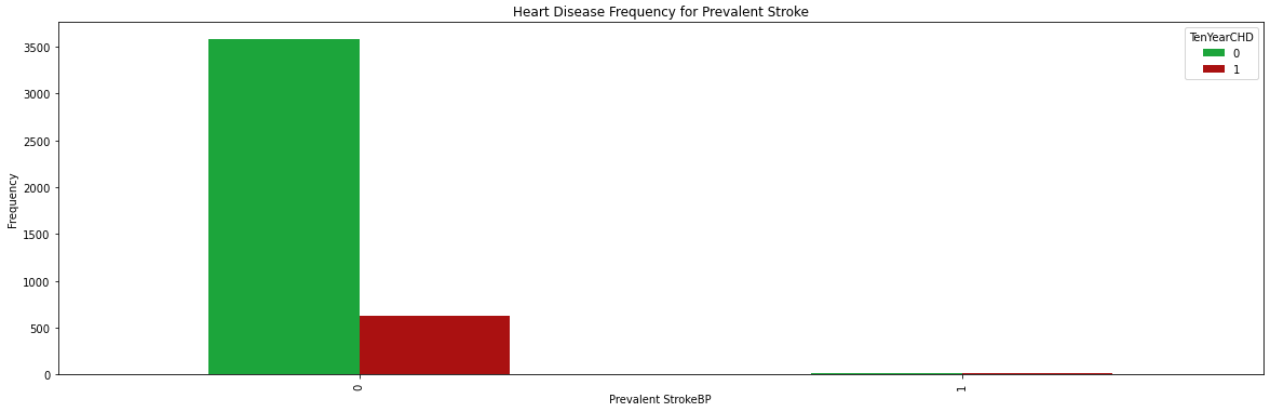
```
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndBPMeds.png')

# Display the plot
plt.show()
```



Prevalent Stroke

In [53]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.prevalentStroke, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6), color=

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Prevalent Stroke')
plt.xlabel('Prevalent StrokeBP')
plt.ylabel('Frequency')

# Save the plot as an image file
#plt.savefig('heartDiseaseAndPrevStroke.png')

# Display the plot
plt.show()
```

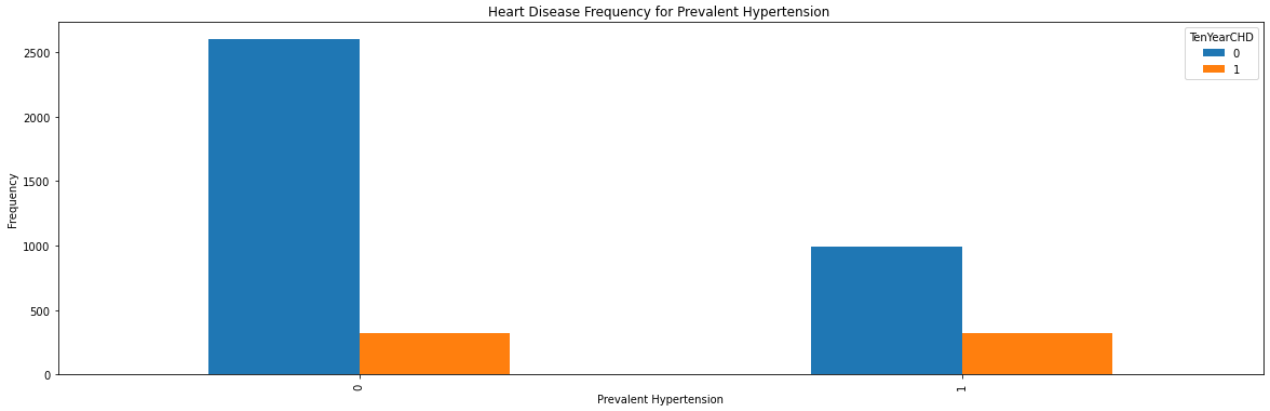


Prevalent Hypertension

In [54]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.prevalentHyp, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6))
```

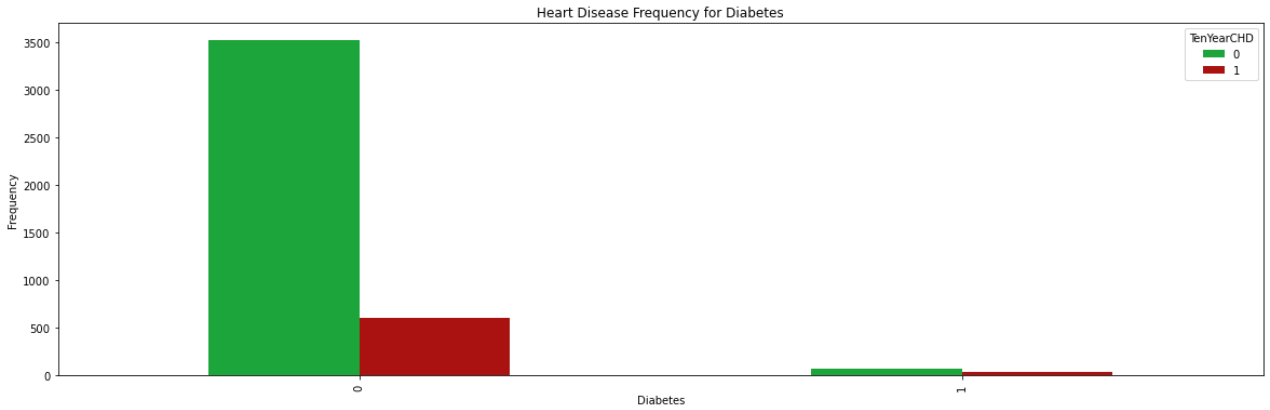
```
# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Prevalent Hypertension ')
plt.xlabel('Prevalent Hypertension')
plt.ylabel('Frequency')

# Save the plot as an image file (optional)
#plt.savefig('heartDiseaseAndPrevHyp.png')

# Display the plot
plt.show()
```



Diabetes

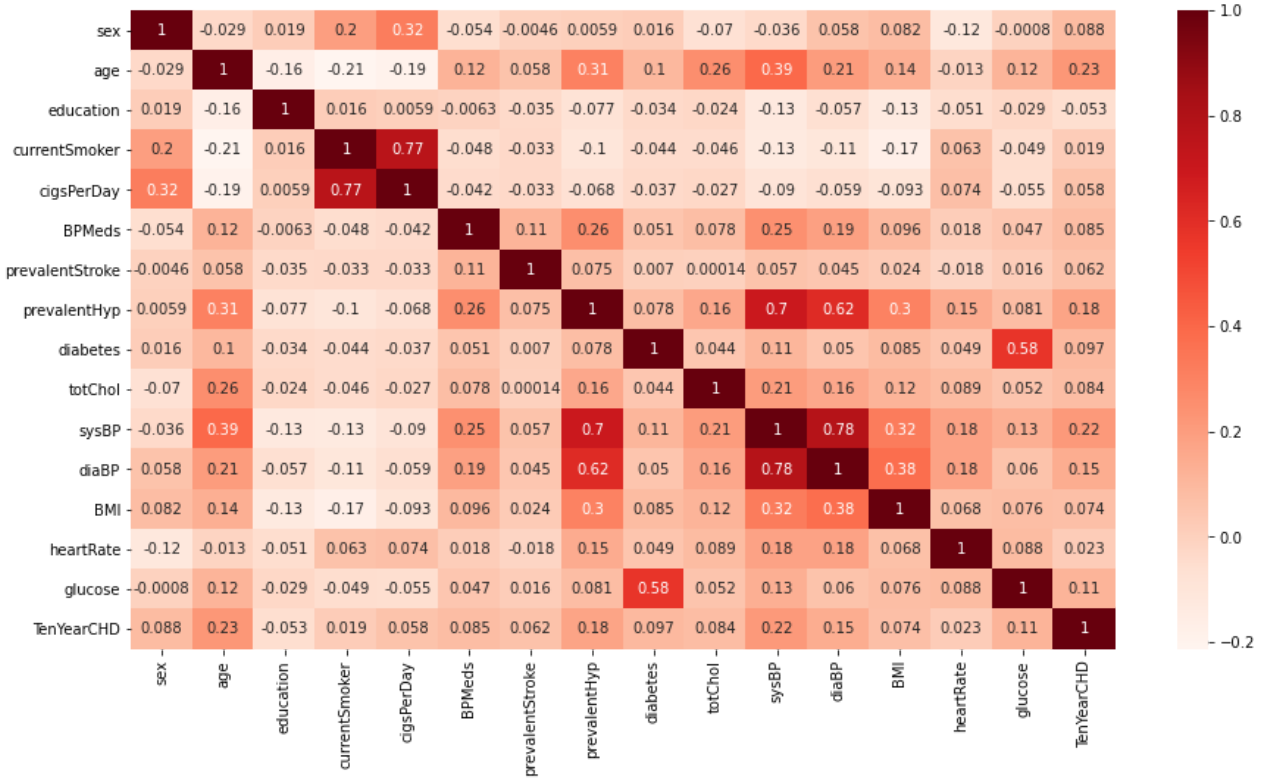
In [55]:

```
# Create a cross-tabulation (contingency table) and plot it as a bar chart
pd.crosstab(dg.diabetes, dg.TenYearCHD).plot(kind="bar", figsize=(20, 6), color=['#1CA5

# Set the plot title, xlabel, and ylabel
plt.title('Heart Disease Frequency for Diabetes')
plt.xlabel('Diabetes')
plt.ylabel('Frequency')

# Save the plot as an image file (optional)
#plt.savefig('heartDiseaseAndSex.png')

# Display the plot
plt.show()
```



Correlation of the Features

In [56]:
```
# Using Pearson Correlation
plt.figure(figsize=(15,8))
cor = dg.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)

# Display the plot
plt.show()
```



Relevant Correlation of Features with Target Feature

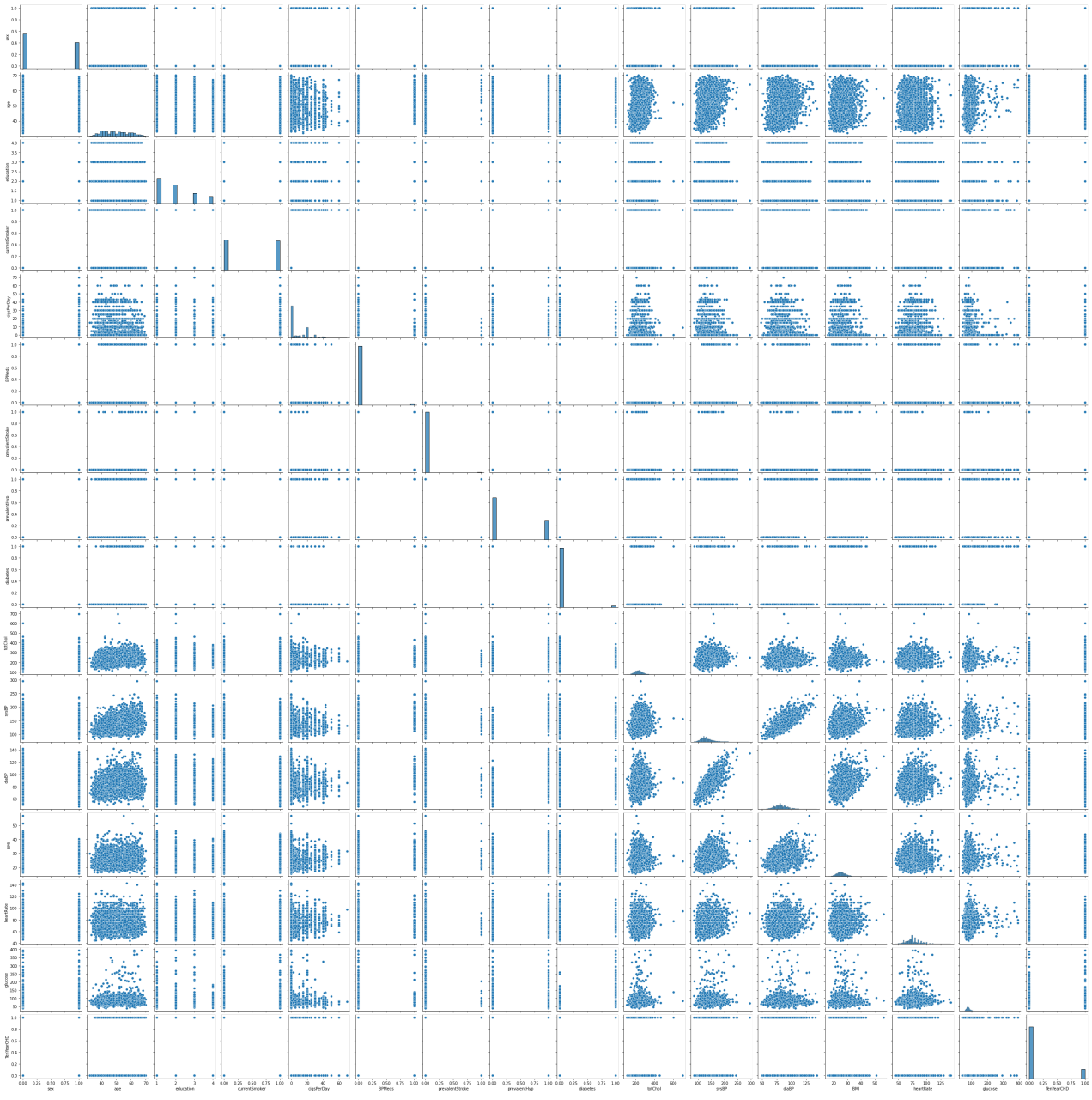
In [57]:
```
#Correlation with output variable
cor_target = abs(cor["TenYearCHD"])

#Selecting correlated features
relevant_features = cor_target[cor_target>0.035]
relevant_features
```

Out[57]:
```
sex                0.088374
age                0.225408
education          0.053459
cigsPerDay         0.057743
BPMeds             0.084604
prevalentStroke    0.061823
prevalentHyp       0.177458
diabetes           0.097344
totChol            0.084200
sysBP              0.216374
diaBP              0.145112
BMI                0.073778
glucose            0.113933
TenYearCHD         1.000000
Name: TenYearCHD, dtype: float64
```

Distribution of the entire Dataset for Outliers Purpose

In [58]:
```
# Display the distribution
sns.pairplot(dg)
```

Out[58]: <seaborn.axisgrid.PairGrid at 0x299b09656d0>



Closer Look into Features with likely Outliers

Cholesterol - Using Multiple Interquartile Range and Box Plot

In [59]:
```
# convert totchol to numpy
a = dg.totChol
a = a.to_numpy()
a
```

Out[59]:
```
array([195., 250., 245., ..., 269., 185., 196.])
```

In [60]:
```
q75, q25 = np.percentile(a, [75, 25])
iqr = q75 - q25
print (iqr)
```

```
57.0
```

In [61]:
```
# upper and lower quartile of outliers
print (q25 - 1.5 * iqr, q75 + 1.5 * iqr)
```

```
120.5 348.5
```

In [62]:
```
# far outliers
print (q25 - 3 * iqr, q75 + 3 * iqr)
```

```
35.0 434.0
```

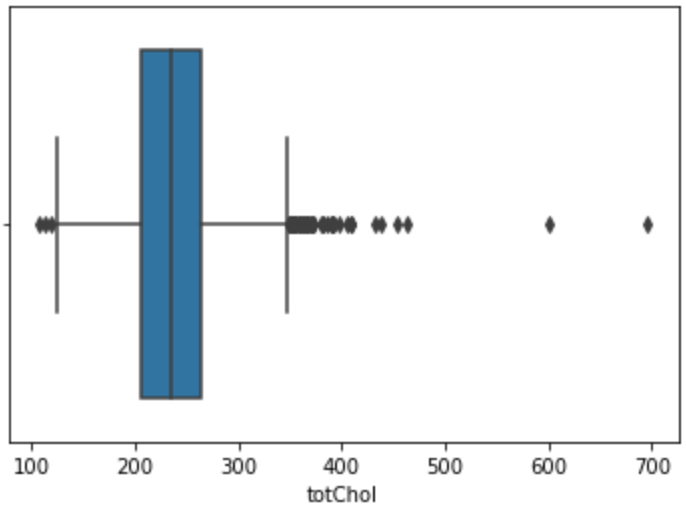
In [63]:
```
#checking each feature with likely outliers
#Cholesterol
sns.boxplot(x = dg.totChol)
outliers = dg[(dg['totChol'] > 434)]

#outliers
# Display the outliers
print("Data points with 'totChol' greater than 500:")
outliers
```

```
Data points with 'totChol' greater than 500:
```

Out[63]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabete
194	0	42	3.0	0	0.0	0.0	0	0	
543	1	47	2.0	1	18.0	0.0	0	1	
1111	0	52	2.0	0	0.0	0.0	0	1	
3160	1	51	2.0	1	9.0	0.0	0	1	
3474	1	42	2.0	1	15.0	0.0	0	1	

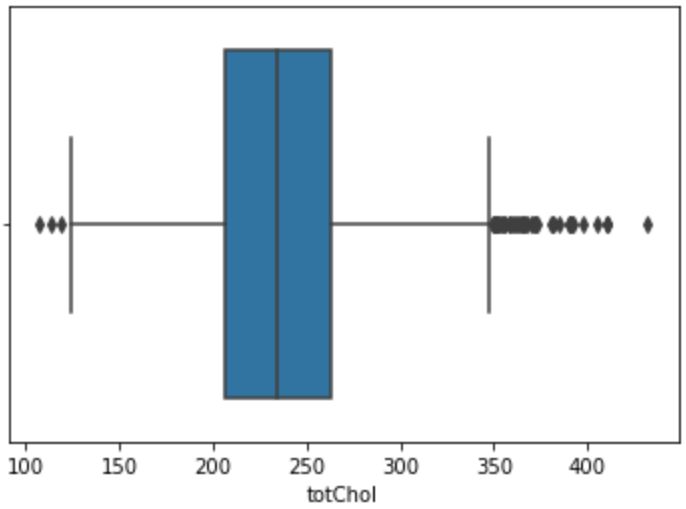
5 rows × 23 columns



Removing Outliers - for Cholesterol

In [64]:
```
# Dropping 2 outliers in cholesterin
dg = dg.drop(dg[dg.totChol > 434].index)
sns.boxplot(dg.totChol)
```

Out[64]: <AxesSubplot:xlabel='totChol'>



SysBP Outliers - Using Multiple IQR

In [65]:
```
# convert sysbp to numpy
b = dg.sysBP
b = b.to_numpy()
b
```

Out[65]: array([106. , 121. , 127.5, ..., 133.5, 141. , 133.])

In [66]:
```
q75, q25 = np.percentile(b, [75, 25])
iqr = q75 - q25
print (iqr)
```

```
26.5
```

In [67]:
```
# upper and lower quartile of outliers
print (q25 - 1.5 * iqr, q75 + 1.5 * iqr)
```

```
77.25 183.25
```

In [68]:
```
# far outliers
print (q25 - 3 * iqr, q75 + 3 * iqr)
```

```
37.5 223.0
```

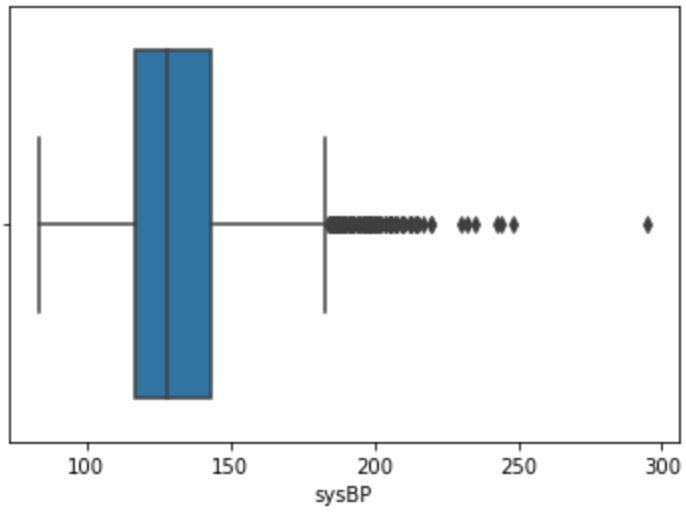
In [69]:
```
#checking each feature with likely outliers
#sysBP
sns.boxplot(x = dg.sysBP)
outliers = dg[(dg['sysBP'] >223)]

#outliers
# Display the outliers
print("Data points with 'sysBP' greater than 223:")
outliers
```

```
Data points with 'sysBP' greater than 223:
```

Out[69]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabete
481	0	64	1.0	0	0.0	0.0	0	1	
864	0	59	2.0	0	0.0	1.0	0	1	
1189	0	48	1.0	0	0.0	0.0	0	1	
1989	1	61	1.0	0	0.0	0.0	0	1	
2091	1	65	1.0	0	0.0	0.0	0	1	
3489	0	62	2.0	0	0.0	1.0	0	1	
3616	0	54	2.0	1	15.0	0.0	0	1	

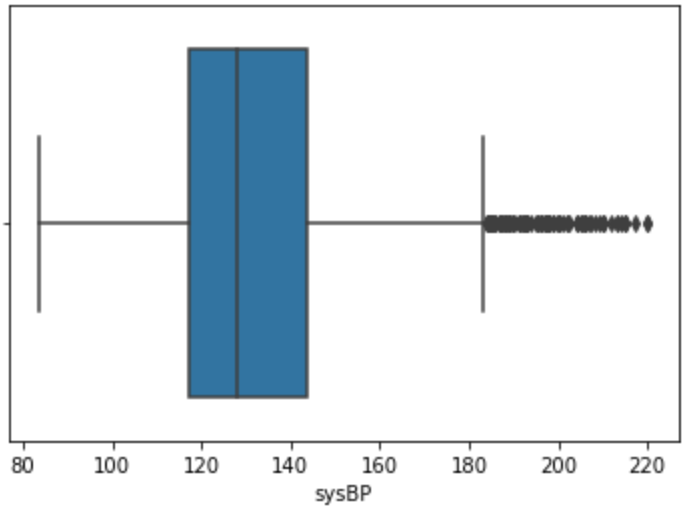
7 rows × 23 columns



Removing Outliers - SysBP

In [70]:
```
# Dropping 2 outliers in cholesterin
dg = dg.drop(dg[dg.sysBP > 223].index)
sns.boxplot(dg.sysBP)
```

Out[70]: <AxesSubplot:xlabel='sysBP'>



BMI Outliers - Using Multiple IQR

In [71]:
```
# convert bmi to numpy
c = dg.BMI
c = c.to_numpy()
c
```

Out[71]: array([26.97, 28.73, 25.34, ..., 21.47, 25.6 , 20.91])

In [72]:
```
q75, q25 = np.percentile(c, [75, 25])
iqr = q75 - q25
print (iqr)
```

4.952500000000001

In [73]:
```
# upper and lower quartile of outliers
print (q25 - 1.5 * iqr, q75 + 1.5 * iqr)
```

15.64125 35.45125

In [74]:
```
# far outliers
print (q25 - 3 * iqr, q75 + 3 * iqr)
```

8.212499999999999 42.88

In [75]:
```
#checking each feature with likely outliers
#BMI
sns.boxplot(x = dg.BMI)
```

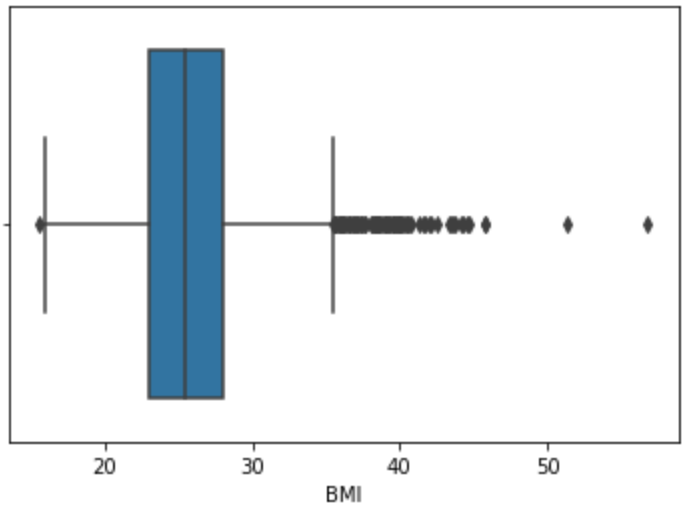
```
outliers = dg[(dg['BMI'] > 43)]
#outliers

# Display the outliers
print("Data points with 'BMI' greater than 50:")
outliers
```

```
Data points with 'BMI' greater than 50:
```

Out[75]:

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabete
78	0	45	1.0	0	0.0	0.0	0	1	
249	0	60	1.0	1	20.0	0.0	0	0	
433	0	45	2.0	0	0.0	0.0	0	1	
750	0	67	1.0	0	0.0	1.0	0	1	
833	0	53	1.0	0	0.0	0.0	0	1	
894	0	42	1.0	1	20.0	0.0	0	1	
1525	0	44	2.0	0	0.0	0.0	0	1	
2162	0	66	1.0	0	0.0	0.0	0	1	
2307	0	37	1.0	1	1.0	0.0	0	1	
2657	0	55	1.0	0	0.0	0.0	0	1	
3927	0	61	1.0	0	0.0	1.0	1	1	
4228	0	50	1.0	0	0.0	0.0	0	1	

12 rows × 23 columns

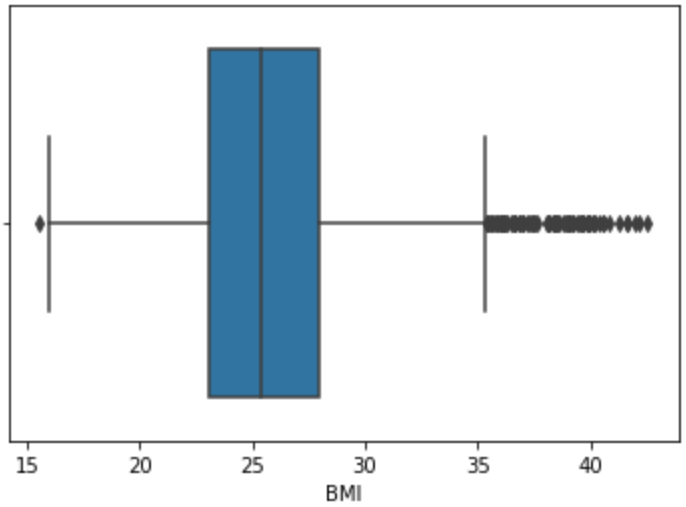


Removing Outliers- BMI

In [76]:

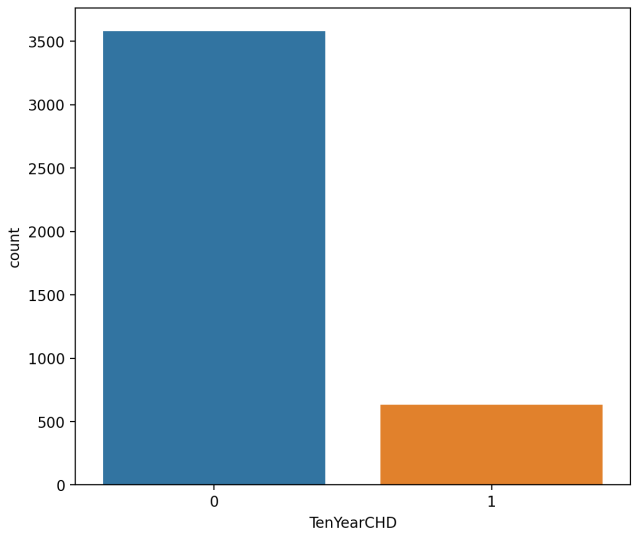
```
# Dropping 2 outliers in cholesterin
dg = dg.drop(dg[dg.BMI > 43].index)
sns.boxplot(dg.BMI)
```

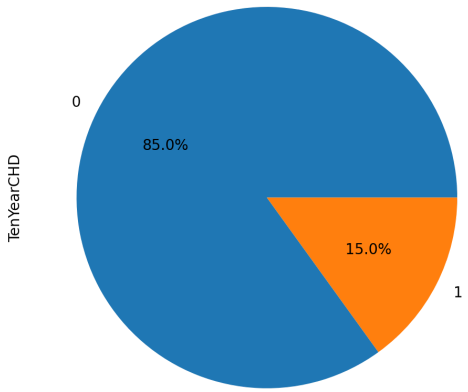
Out[76]: <AxesSubplot:xlabel='BMI'>



Checking if the Target Feature of the Dataset is Balanced

In [77]:
```
#Checking how imbalance the data/target variable using seaborn
fig, ax_position = plt.subplots(1, 2,figsize = (15, 6), dpi=200)
a = sns.countplot (x = 'TenYearCHD', data = dg, ax = ax_position[0])
a = dg['TenYearCHD'].value_counts().plot.pie(autopct = "%1.1f%%", ax = ax_position[1])
```





In [78]:
```
dt.isnull().sum()
```

Out[78]:
```
sex              0
age              0
education        0
currentSmoker    0
cigsPerDay       0
BPMeds           0
prevalentStroke  0
prevalentHyp     0
diabetes         0
totChol          0
sysBP            0
diaBP            0
```

```
BMI               0
heartRate         0
glucose           0
TenYearCHD        0
dtype: int64
```

Building the first Model with the Unbalanced Target Feature

Scaling the Dataset for a Common Scale

Using Four Classifiers:

1. XGBoost
2. Random Forest
3. Logistic Regression
4. Support Vector Machine

In [79]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    classification = classification_report(y_test, y_pred)

    print(classification)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.90       716
           1       0.37      0.14      0.20       132

    accuracy                           0.83       848
   macro avg       0.61      0.55      0.55       848
weighted avg       0.78      0.83      0.79       848


Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.85      0.99      0.91       716
           1       0.33      0.03      0.06       132

    accuracy                           0.84       848
   macro avg       0.59      0.51      0.48       848
weighted avg       0.77      0.84      0.78       848


Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       716
           1       0.75      0.05      0.09       132

    accuracy                           0.85       848
   macro avg       0.80      0.52      0.50       848
weighted avg       0.83      0.85      0.79       848


Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       716
           1       0.75      0.02      0.04       132

    accuracy                           0.85       848
   macro avg       0.80      0.51      0.48       848
weighted avg       0.83      0.85      0.78       848
```

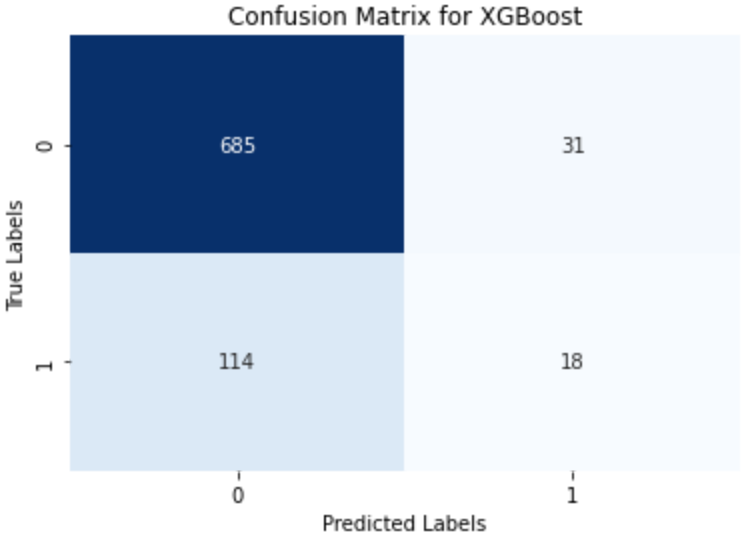
Confusion Matrix for the four classifiers

In [80]:

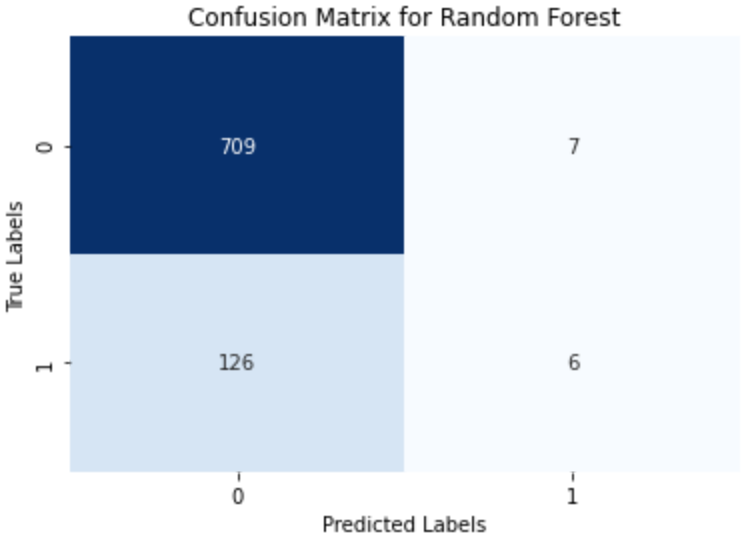
```
# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
```

```
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

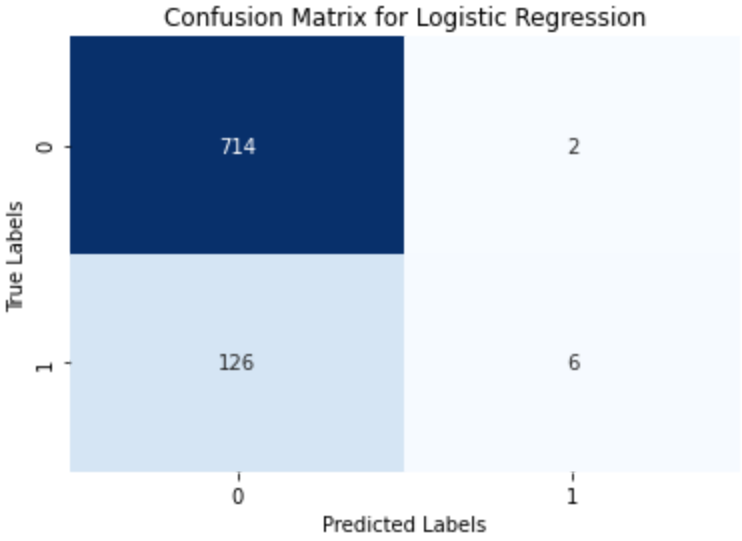
```
Classifier: XGBoost
```



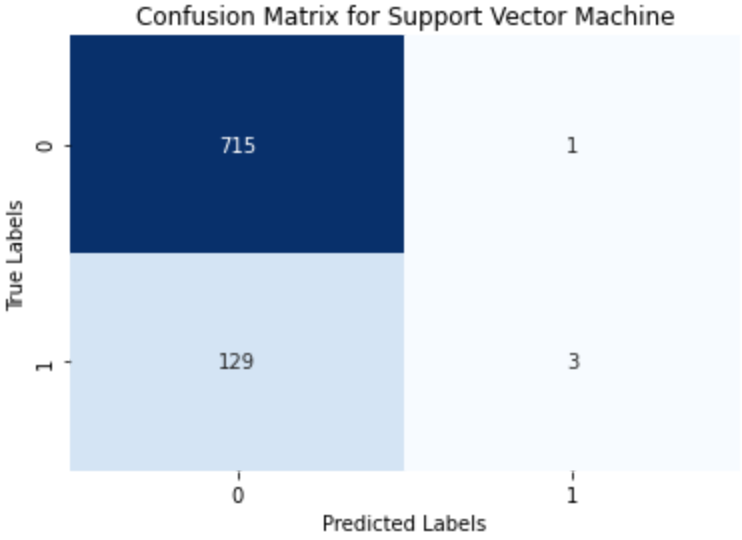

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



Classifier: Support Vector Machine

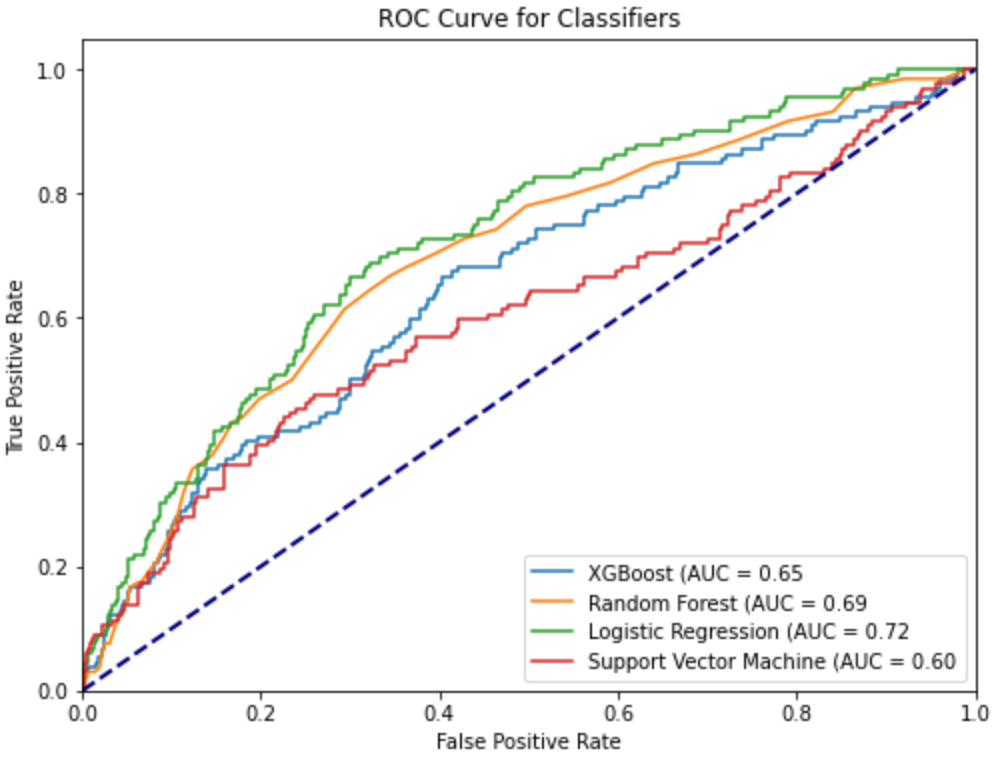


Area Under Curve - ROC Curves for the Classifiers

In [81]:

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))
```

```
for clf_name, y_prob in y_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Cross Validation, MAE, RMSE, r2-Score

In [82]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD'])
y = dt['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Perform k-fold cross-validation for each classifier
```

```
k_folds = 5  # You can adjust the number of folds as needed

for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Perform k-fold cross-validation and get predictions
    y_pred = cross_val_predict(clf, X_scaled, y, cv=k_folds)

    # Calculate evaluation metrics for each fold
    mae = mean_absolute_error(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    #mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # Print evaluation metrics for each fold
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R2 Score: {r2}')
    #print(f'MSE: {mse}')

    # Calculate and print mean accuracy
    scores = cross_val_score(clf, X_scaled, y, cv=k_folds, scoring='accuracy')
    print(f"Cross-Validation Scores (Accuracy): {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print("\n")
```

```
Classifier: XGBoost
MAE: 0.16650943396226414
RMSE: 0.4080556750766544
R2 Score: -0.2926025466529407
Cross-Validation Scores (Accuracy): [0.83254717 0.82900943 0.83136792 0.8384434  0.83608
491]
Mean Accuracy: 0.8334905660377359


Classifier: Random Forest
MAE: 0.15188679245283018
RMSE: 0.38972656113335435
R2 Score: -0.17908787541713012
Cross-Validation Scores (Accuracy): [0.84787736 0.85141509 0.84551887 0.85495283 0.84787
736]
Mean Accuracy: 0.8495283018867925


Classifier: Logistic Regression
MAE: 0.14669811320754716
RMSE: 0.38301189695301524
R2 Score: -0.1388084759463586
Cross-Validation Scores (Accuracy): [0.85259434 0.85613208 0.85023585 0.85731132 0.85023
585]
Mean Accuracy: 0.8533018867924529


Classifier: Support Vector Machine
MAE: 0.15212264150943397
RMSE: 0.39002902649602117
R2 Score: -0.18091875721125605
Cross-Validation Scores (Accuracy): [0.84787736 0.8490566  0.8490566  0.84551887 0.84787
736]
Mean Accuracy: 0.847877358490566
```

Selecting Important Features

Using KBest (Filter Method)

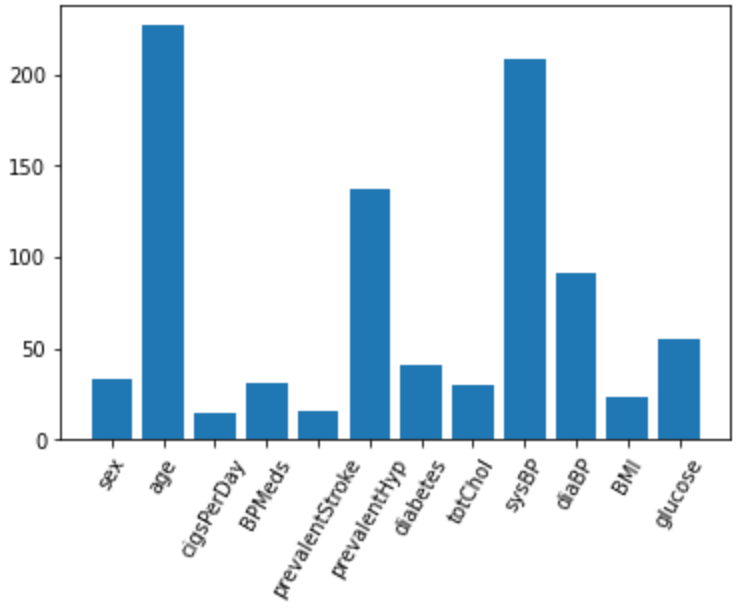
In [83]:
```
# building KBest model and using k as 15
selector = SelectKBest(f_classif, k=13)
selector.fit(dt, dt.TenYearCHD)
```

Out[83]:
```
SelectKBest

SelectKBest(k=13)
```

In [84]:
```
# getting the most important features by kbest
mask = selector.get_support()
names = selector.feature_names_in_[mask]
scores = selector.scores_[mask]
names
```

Out[84]:
```
array(['sex', 'age', 'cigsPerDay', 'BPMeds', 'prevalentStroke',
       'prevalentHyp', 'diabetes', 'totChol', 'sysBP', 'diaBP', 'BMI',
       'glucose', 'TenYearCHD'], dtype=object)
```

In [85]:
```
# visualizing the important features
plt.bar(names, scores)
plt.xticks(rotation = 60)
plt.savefig("import_feature_kbest")
```



Building Model for the Important Features Gotten from KBest

In [86]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education', 'currentSmoker', 'heartRate'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Print the classification report
    report = classification_report(y_test, y_pred)
    print(report)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      1077
           1       0.38      0.13      0.20       195

    accuracy                           0.83      1272
   macro avg       0.62      0.55      0.55      1272
weighted avg       0.79      0.83      0.80      1272



Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1077
           1       0.61      0.07      0.13       195

    accuracy                           0.85      1272
   macro avg       0.73      0.53      0.52      1272
weighted avg       0.82      0.85      0.80      1272



Classifier: Logistic Regression
              precision    recall  f1-score   support
```

```
              0       0.86      0.99      0.92      1077
              1       0.67      0.09      0.16       195

    accuracy                            0.85      1272
   macro avg       0.76      0.54      0.54      1272
weighted avg       0.83      0.85      0.80      1272


Classifier: Support Vector Machine
              precision    recall  f1-score   support

              0       0.85      1.00      0.92      1077
              1       0.67      0.01      0.02       195

    accuracy                            0.85      1272
   macro avg       0.76      0.50      0.47      1272
weighted avg       0.82      0.85      0.78      1272
```

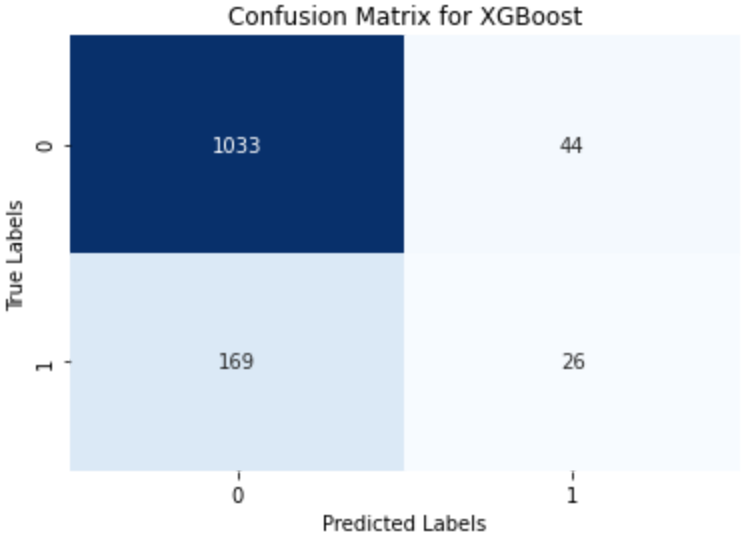
Confusion Matrix of the KBest Features' Model

In [87]:

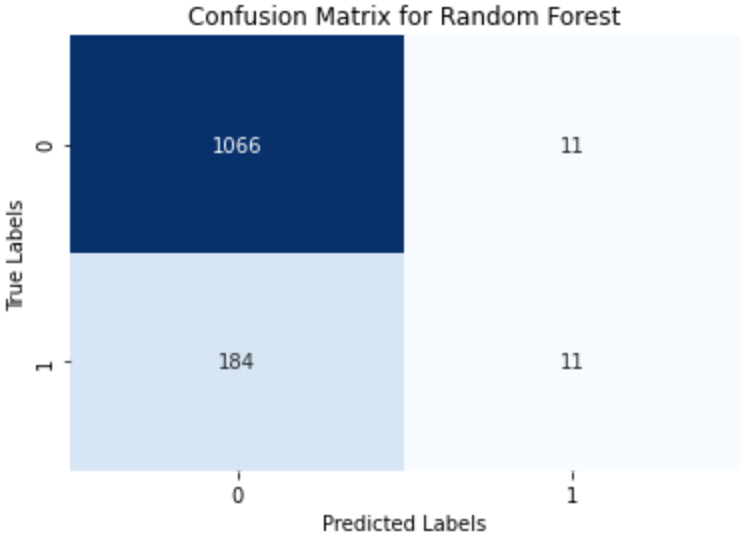
```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education', 'prevalentStroke', 'glucose'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
```

```
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

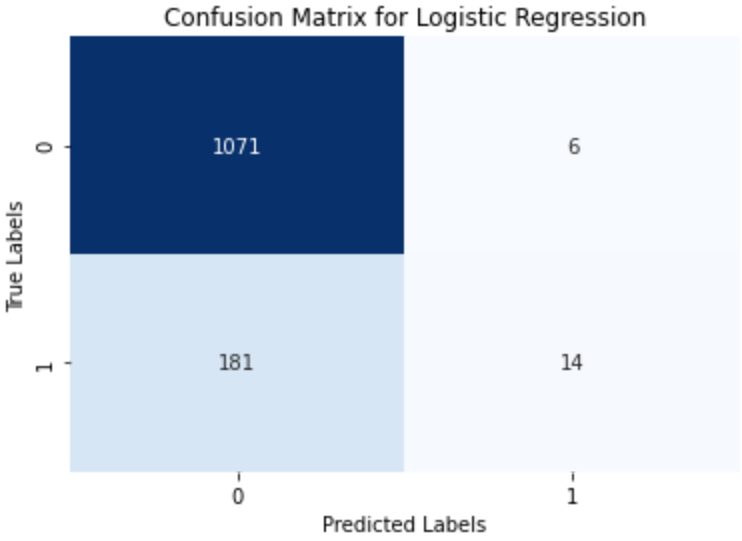
```
Classifier: XGBoost
```



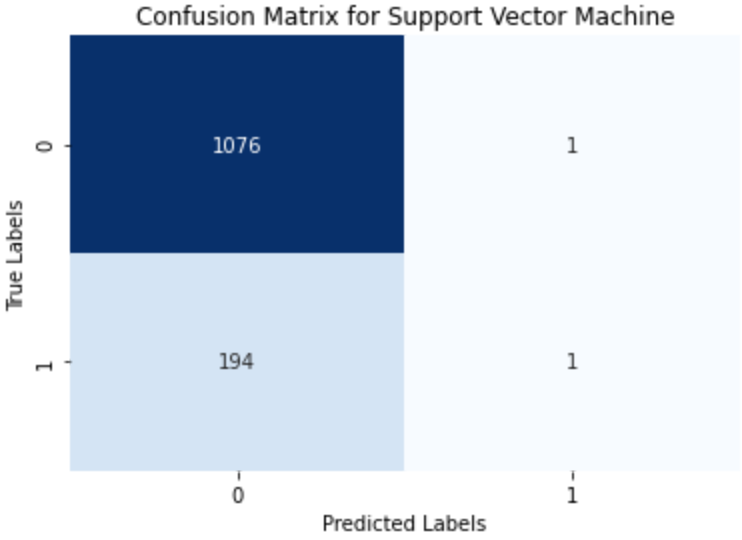

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



Classifier: Support Vector Machine

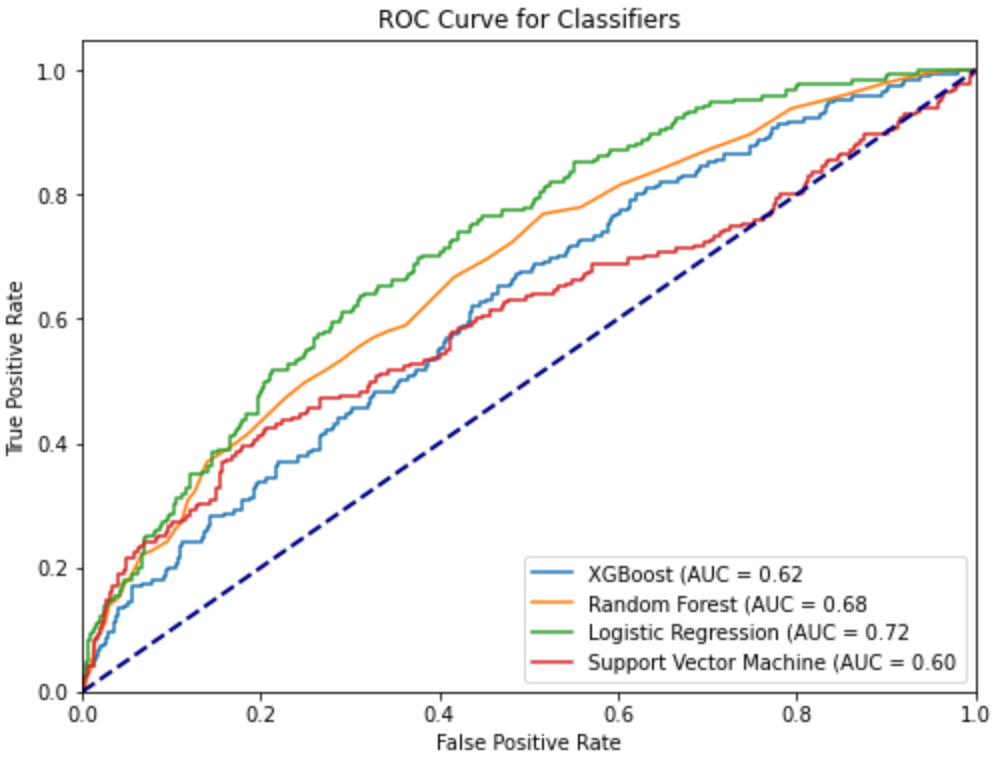


ROC Curve

In [88]:

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in y_probs.items():
```

```
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Cross Validation, MSE, R2 Score and RMSE

In [89]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education', 'prevalentStroke', 'glucose'])
y = dt['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Perform k-fold cross-validation for each classifier
k_folds = 5  # You can adjust the number of folds as needed
```

```
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Perform k-fold cross-validation and get predictions
    y_pred = cross_val_predict(clf, X_scaled, y, cv=k_folds)

    # Calculate evaluation metrics for each fold
    mae = mean_absolute_error(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    #mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # Print evaluation metrics for each fold
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R2 Score: {r2}')
    #print(f'MSE: {mse}')

    # Calculate and print mean accuracy
    scores = cross_val_score(clf, X_scaled, y, cv=k_folds, scoring='accuracy')
    print(f"Cross-Validation Scores (Accuracy): {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print("\n")
```

```
Classifier: XGBoost
MAE: 0.16721698113207548
RMSE: 0.4089217298360109
R2 Score: -0.29809519203531876
Cross-Validation Scores (Accuracy): [0.83608491 0.83018868 0.83136792 0.83608491 0.83018
868]
Mean Accuracy: 0.8327830188679245


Classifier: Random Forest
MAE: 0.15424528301886792
RMSE: 0.3927407325690422
R2 Score: -0.19739669335838994
Cross-Validation Scores (Accuracy): [0.84551887 0.8490566  0.84551887 0.84669811 0.84198
113]
Mean Accuracy: 0.8457547169811322


Classifier: Logistic Regression
MAE: 0.14882075471698114
RMSE: 0.38577293155038905
R2 Score: -0.15528641209349248
Cross-Validation Scores (Accuracy): [0.85259434 0.85495283 0.84551887 0.85259434 0.85023
585]
Mean Accuracy: 0.8511792452830189


Classifier: Support Vector Machine
MAE: 0.15259433962264152
RMSE: 0.39063325462976334
R2 Score: -0.18458052079950815
Cross-Validation Scores (Accuracy): [0.84787736 0.84787736 0.84787736 0.84787736 0.84551
887]
Mean Accuracy: 0.8474056603773585
```

Using Domain Knowledge to remove 'Education' from the features because it does not really contribute or influence anyone having heart disease

In [90]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Print the classification report
    report = classification_report(y_test, y_pred)
    print(report)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.86      0.96      0.90      1077
           1       0.33      0.12      0.18       195

    accuracy                           0.83      1272
   macro avg       0.60      0.54      0.54      1272
weighted avg       0.78      0.83      0.79      1272



Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.85      0.99      0.92      1077
           1       0.57      0.07      0.12       195

    accuracy                           0.85      1272
```

```
   macro avg       0.71      0.53      0.52      1272
weighted avg       0.81      0.85      0.80      1272


Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1077
           1       0.70      0.11      0.19       195

    accuracy                           0.86      1272
   macro avg       0.78      0.55      0.55      1272
weighted avg       0.84      0.86      0.81      1272


Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      1077
           1       0.60      0.02      0.03       195

    accuracy                           0.85      1272
   macro avg       0.72      0.51      0.47      1272
weighted avg       0.81      0.85      0.78      1272
```

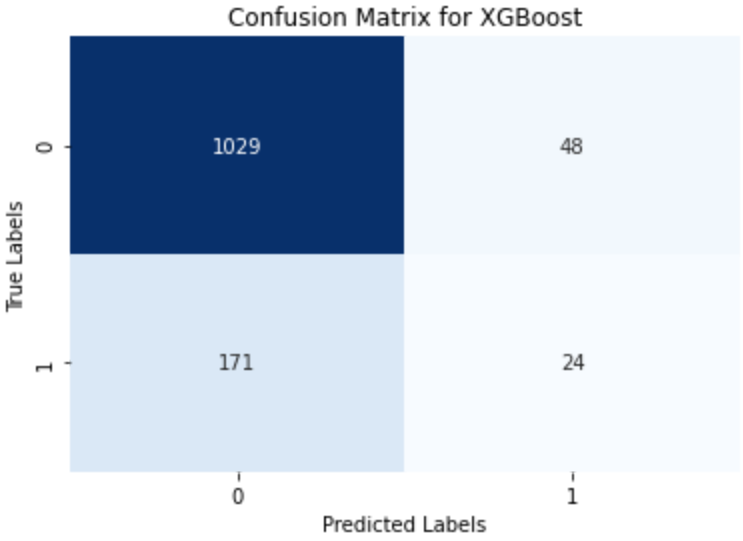
Confusion Matrix of the Domain Features' Model

In [91]:

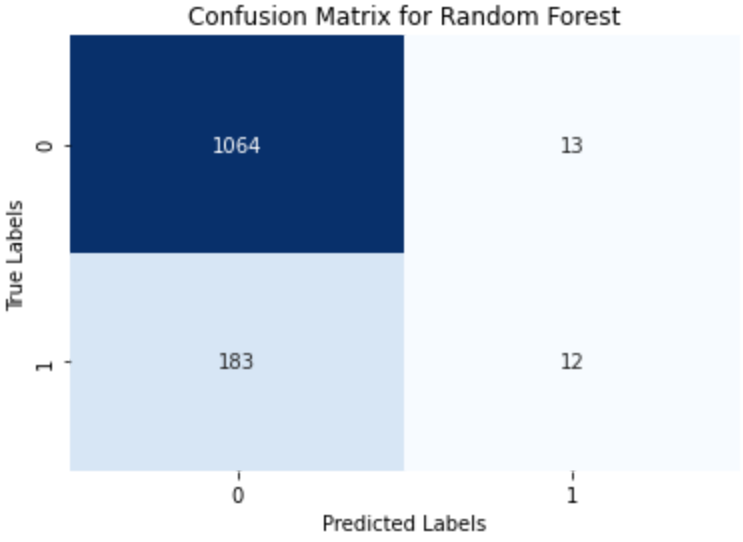
```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)
```

```
    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

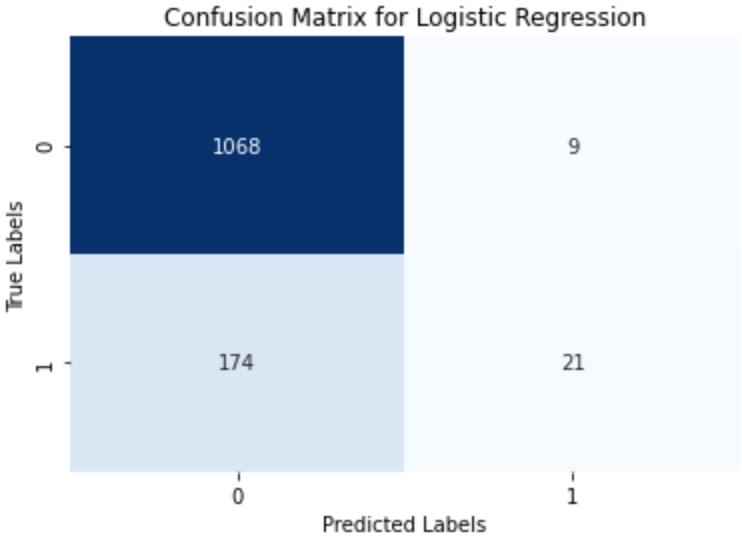
```
Classifier: XGBoost
```



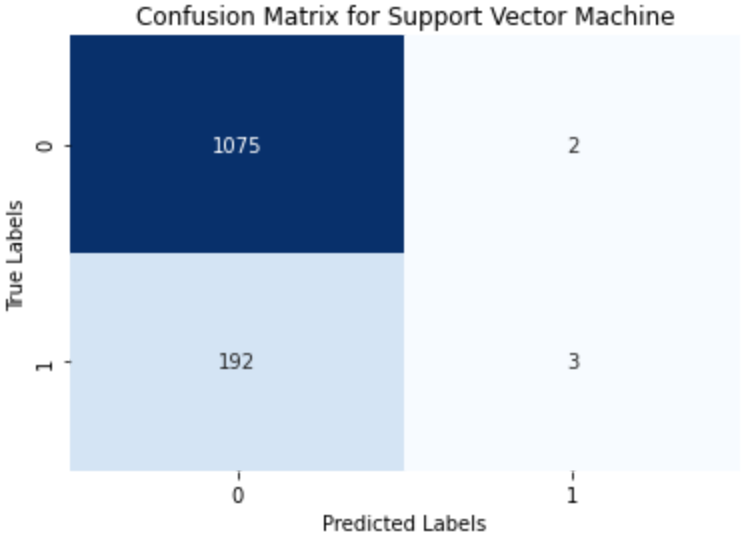

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



Classifier: Support Vector Machine

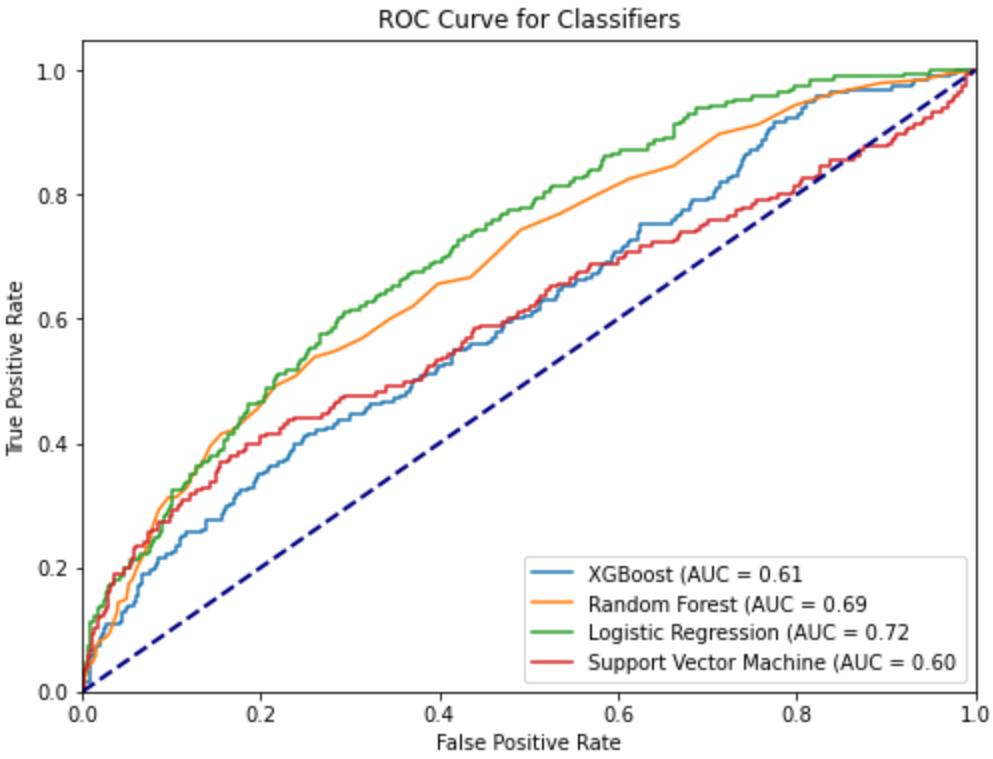


ROC Curve

In [92]:

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in y_probs.items():
```

```
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Cross Validation, MSE, R2 Score and RMSE

In [93]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education'])
y = dt['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Perform k-fold cross-validation for each classifier
k_folds = 5  # You can adjust the number of folds as needed
```

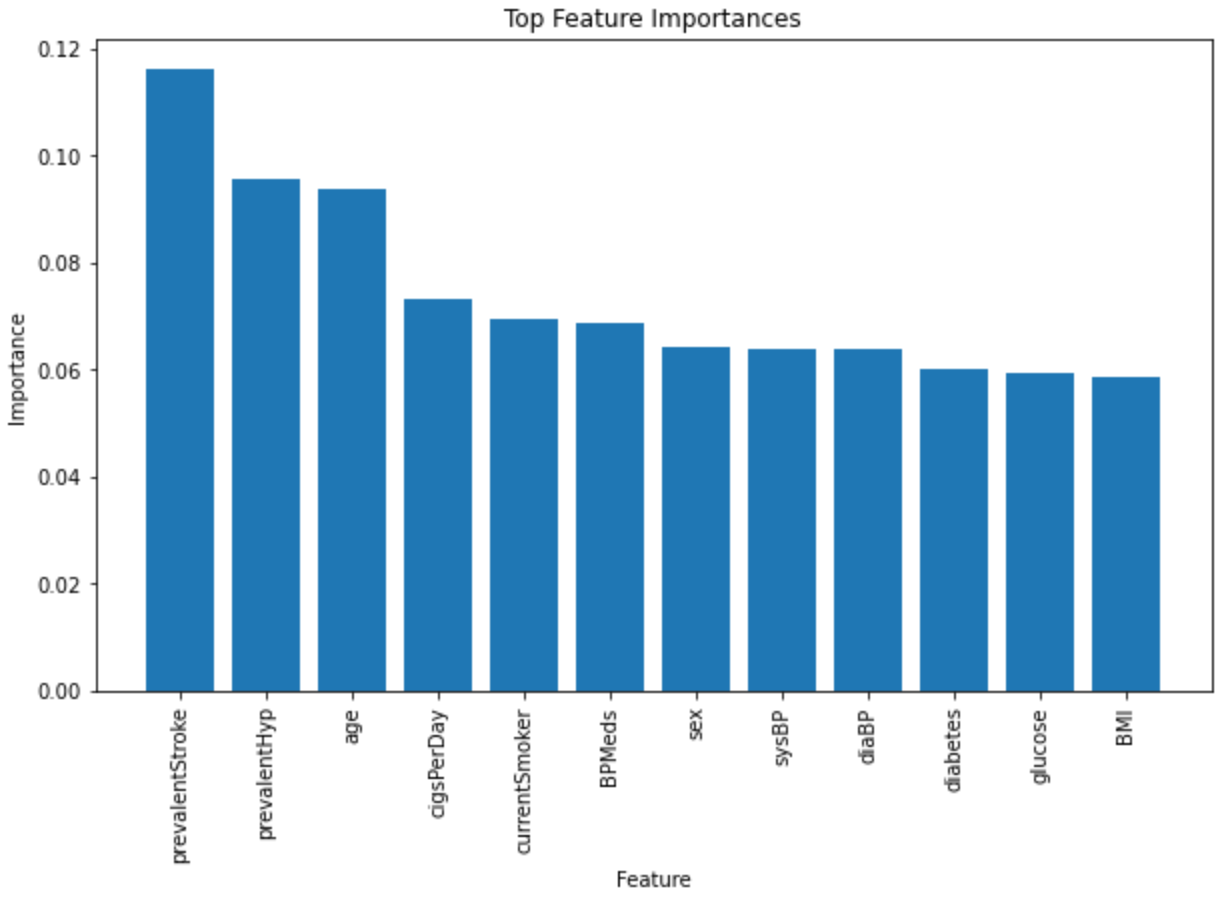
```
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Perform k-fold cross-validation and get predictions
    y_pred = cross_val_predict(clf, X_scaled, y, cv=k_folds)

    # Calculate evaluation metrics for each fold
    mae = mean_absolute_error(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    #mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # Print evaluation metrics for each fold
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R2 Score: {r2}')
    #print(f'MSE: {mse}')

    # Calculate and print mean accuracy
    scores = cross_val_score(clf, X_scaled, y, cv=k_folds, scoring='accuracy')
    print(f"Cross-Validation Scores (Accuracy): {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print("\n")
```

```
Classifier: XGBoost
MAE: 0.16627358490566038
RMSE: 0.40776658139879535
R2 Score: -0.2907716648588148
Cross-Validation Scores (Accuracy): [0.83490566 0.83254717 0.83608491 0.84080189 0.82429
245]
Mean Accuracy: 0.8337264150943395


Classifier: Random Forest
MAE: 0.15235849056603773
RMSE: 0.3903312574801533
R2 Score: -0.18274963900538221
Cross-Validation Scores (Accuracy): [0.84669811 0.84787736 0.84551887 0.85377358 0.84787
736]
Mean Accuracy: 0.8483490566037736


Classifier: Logistic Regression
MAE: 0.1474056603773585
RMSE: 0.3839344480212195
R2 Score: -0.14430112132873663
Cross-Validation Scores (Accuracy): [0.85259434 0.85731132 0.85023585 0.85613208 0.84669
811]
Mean Accuracy: 0.8525943396226415


Classifier: Support Vector Machine
MAE: 0.15235849056603773
RMSE: 0.3903312574801533
R2 Score: -0.18274963900538221
Cross-Validation Scores (Accuracy): [0.84787736 0.85023585 0.84669811 0.84551887 0.84787
736]
Mean Accuracy: 0.8476415094339623
```

Selecting Important Features Using XGBoost Classifier (Wrapper Method)

In [94]:

```
# Create an instance of the XGBoost classifier
xgb_classifier = XGBClassifier()

# Fit the model to your training data
xgb_classifier.fit(X_train, y_train)

# Get feature importances
importances = xgb_classifier.feature_importances_

# Get the names of the features
feature_names = X.columns

# Sort feature importances in descending order
indices = importances.argsort()[::-1]

# Plot the top N feature importances
top_n = 12  # Adjust the number of top features to display
plt.figure(figsize=(10, 6))
plt.title("Top Feature Importances")
plt.bar(range(top_n), importances[indices][:top_n], align="center")
plt.xticks(range(top_n), feature_names[indices][:top_n], rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.show()
```



Building Models for the Important Features Gotten from XGBoost

In [95]:
```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education', 'heartRate', 'totChol'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Print the classification report
    report = classification_report(y_test, y_pred)
    print(report)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      1077
           1       0.37      0.12      0.18       195

    accuracy                           0.83      1272
   macro avg       0.61      0.54      0.54      1272
weighted avg       0.78      0.83      0.80      1272



Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.86      0.98      0.91      1077
           1       0.46      0.08      0.14       195

    accuracy                           0.84      1272
   macro avg       0.66      0.53      0.53      1272
weighted avg       0.79      0.84      0.80      1272
```

```
Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.86      0.99      0.92      1077
           1       0.64      0.08      0.15       195

    accuracy                           0.85      1272
   macro avg       0.75      0.54      0.53      1272
weighted avg       0.82      0.85      0.80      1272


Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      1077
           1       0.67      0.02      0.04       195

    accuracy                           0.85      1272
   macro avg       0.76      0.51      0.48      1272
weighted avg       0.82      0.85      0.78      1272
```

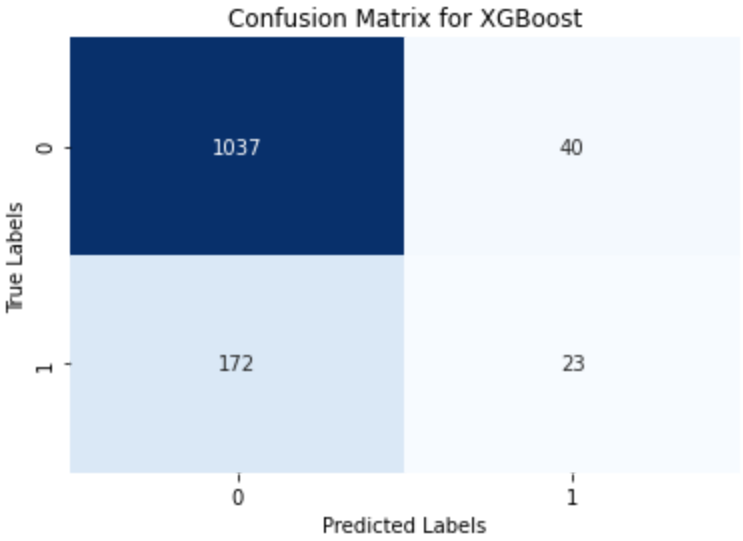
Confusion Matrix for XGBoost Classifier Important Features Selected

In [96]:

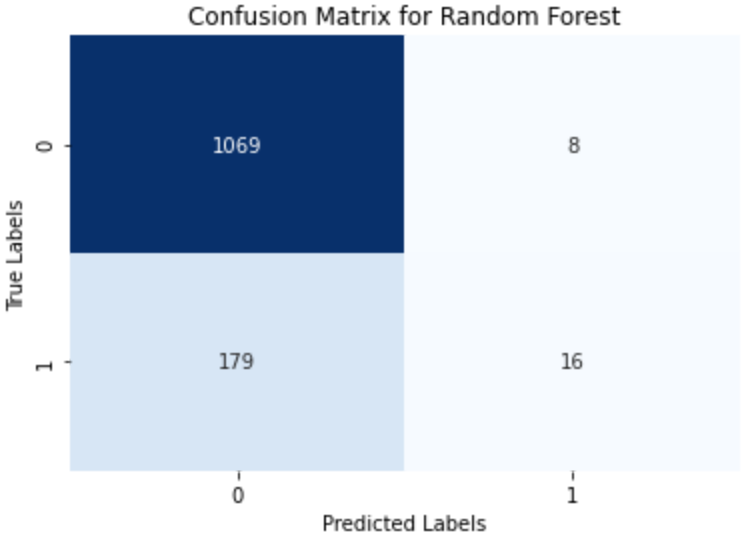
```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education', 'heartRate', 'totChol'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)
```

```
    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

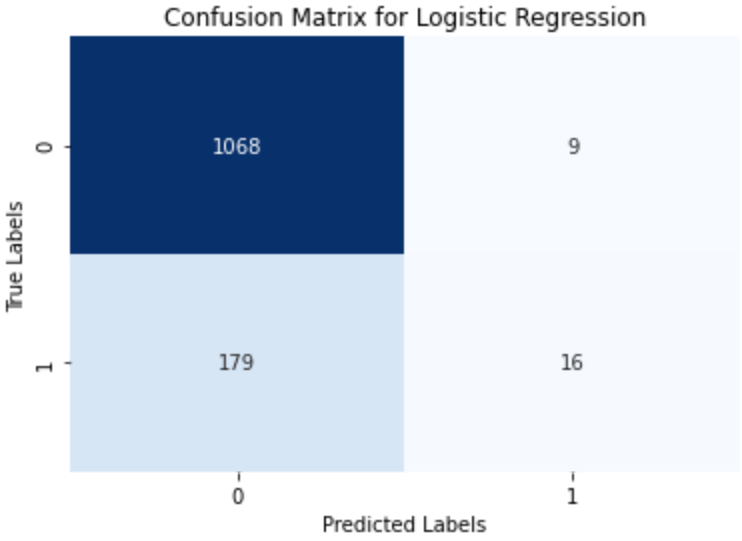
```
Classifier: XGBoost
```



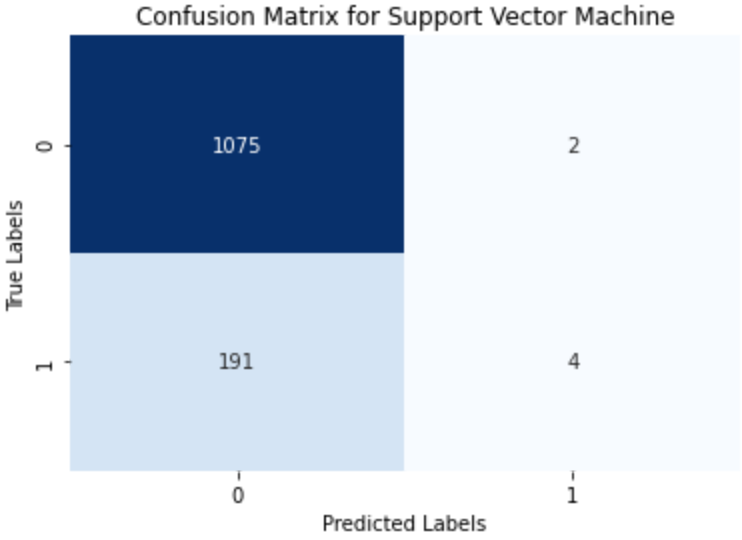

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



Classifier: Support Vector Machine

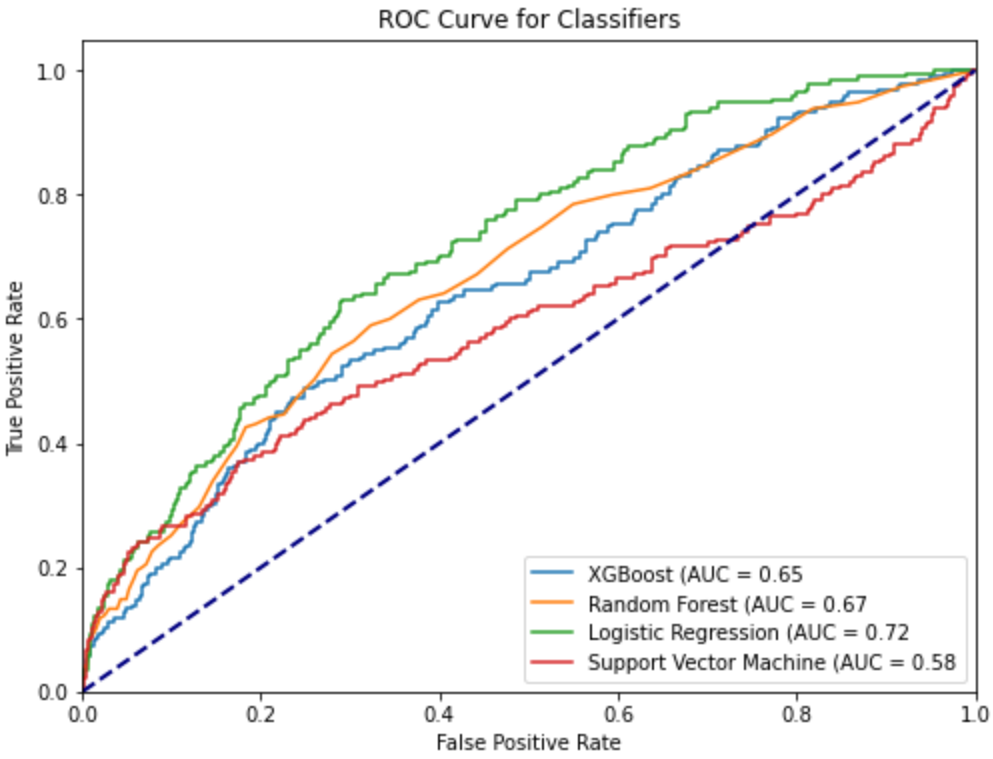


ROC Curve

In [97]:

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in y_probs.items():
```

```
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Cross Validation, MSE, R2 Score and RMSE

In [98]:

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'education', 'heartRate', 'totChol'])
y = dt['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Perform k-fold cross-validation for each classifier
```

```
k_folds = 5  # You can adjust the number of folds as needed

for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Perform k-fold cross-validation and get predictions
    y_pred = cross_val_predict(clf, X_scaled, y, cv=k_folds)

    # Calculate evaluation metrics for each fold
    mae = mean_absolute_error(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    #mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # Print evaluation metrics for each fold
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R2 Score: {r2}')
    #print(f'MSE: {mse}')

    # Calculate and print mean accuracy
    scores = cross_val_score(clf, X_scaled, y, cv=k_folds, scoring='accuracy')
    print(f"Cross-Validation Scores (Accuracy): {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print("\n")
```

```
Classifier: XGBoost
MAE: 0.16863207547169812
RMSE: 0.4106483598794693
R2 Score: -0.3090804828000746
Cross-Validation Scores (Accuracy): [0.82900943 0.82665094 0.83490566 0.83962264 0.82665
094]
Mean Accuracy: 0.8313679245283019


Classifier: Random Forest
MAE: 0.1570754716981132
RMSE: 0.39632748037212007
R2 Score: -0.21936727488790164
Cross-Validation Scores (Accuracy): [0.8384434  0.84316038 0.85023585 0.85023585 0.84080
189]
Mean Accuracy: 0.8445754716981133


Classifier: Logistic Regression
MAE: 0.14764150943396226
RMSE: 0.3842414728188021
R2 Score: -0.14613200312286256
Cross-Validation Scores (Accuracy): [0.85259434 0.85731132 0.84551887 0.85849057 0.84787
736]
Mean Accuracy: 0.8523584905660379


Classifier: Support Vector Machine
MAE: 0.15188679245283018
RMSE: 0.38972656113335435
R2 Score: -0.17908787541713012
Cross-Validation Scores (Accuracy): [0.84787736 0.85141509 0.84669811 0.84787736 0.84669
811]
Mean Accuracy: 0.8481132075471699
```

Forward Feature Selection from Sequential Feature Selector (SFS) - Wrapper Method

In [99]:
```
# Create an instance of the classifier you want to use for feature selection
classifier_for_feature_selection = RandomForestClassifier()

# Create a list of feature names (replace 'feature_names' with your actual list of featu
feature_names = ['sex', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose']

# Initialize the SFS model for forward feature selection
sfs = SequentialFeatureSelector(classifier_for_feature_selection, k_features=10, forwar

# Fit SFS to the training data
sfs.fit(X_train, y_train)

# Get the indices of the selected features
selected_feature_indices = list(sfs.k_feature_idx_)

# Get the corresponding feature names from your list of feature names
selected_feature_names = [feature_names[i] for i in selected_feature_indices]

# Print the selected feature names
print("Selected Features:")
selected_feature_names
```

```
[2023-12-15 15:51:56] Features: 1/10 -- score: 0.8487199141499311
[2023-12-15 15:52:10] Features: 2/10 -- score: 0.8487199141499311
[2023-12-15 15:52:25] Features: 3/10 -- score: 0.8487199141499311
[2023-12-15 15:52:38] Features: 4/10 -- score: 0.8487199141499311
[2023-12-15 15:52:51] Features: 5/10 -- score: 0.8493933148233317
[2023-12-15 15:53:04] Features: 6/10 -- score: 0.8477092453483683
[2023-12-15 15:53:17] Features: 7/10 -- score: 0.8413039898706003
[2023-12-15 15:53:28] Features: 8/10 -- score: 0.8241265947842675
[2023-12-15 15:53:38] Features: 9/10 -- score: 0.8258123676336154
Selected Features:
[2023-12-15 15:53:46] Features: 10/10 -- score: 0.8389510620539287
```

Out[99]:
```
['sex',
 'age',
 'education',
 'currentSmoker',
 'cigsPerDay',
 'BPMeds',
 'prevalentStroke',
 'prevalentHyp',
 'diabetes',
 'totChol']
```

Model Built on the Features Selected by SFS

In [100…
```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'sex', 'age', 'heartRate', 'diaBP', 'BMI'])
y = dt['TenYearCHD']

# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Print the classification report
    report = classification_report(y_test, y_pred)
    print(report)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      1077
           1       0.22      0.06      0.10       195

    accuracy                           0.82      1272
   macro avg       0.54      0.51      0.50      1272
weighted avg       0.75      0.82      0.78      1272



Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.85      0.98      0.91      1077
           1       0.31      0.06      0.10       195

    accuracy                           0.84      1272
   macro avg       0.58      0.52      0.50      1272
weighted avg       0.77      0.84      0.79      1272



Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.85      0.99      0.92      1077
           1       0.59      0.05      0.09       195

    accuracy                           0.85      1272
   macro avg       0.72      0.52      0.51      1272
weighted avg       0.81      0.85      0.79      1272
```

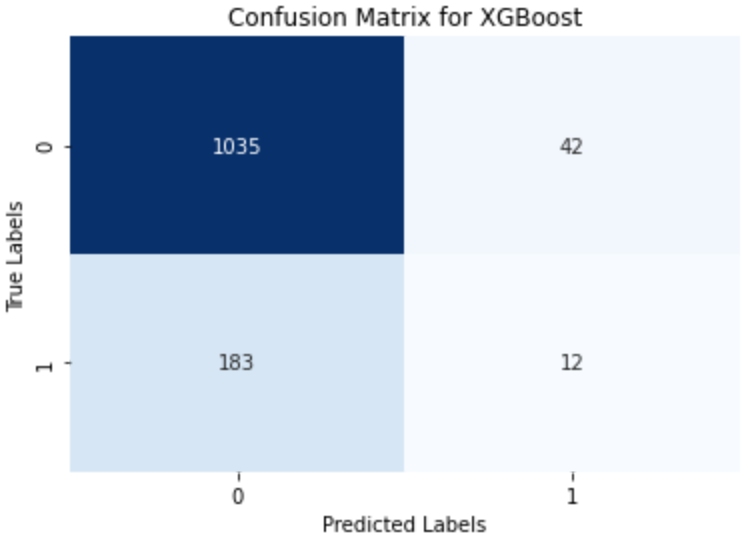
```
Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      1077
           1       0.67      0.01      0.02       195

    accuracy                           0.85      1272
   macro avg       0.76      0.50      0.47      1272
weighted avg       0.82      0.85      0.78      1272
```

Confusion Matrix for SFS

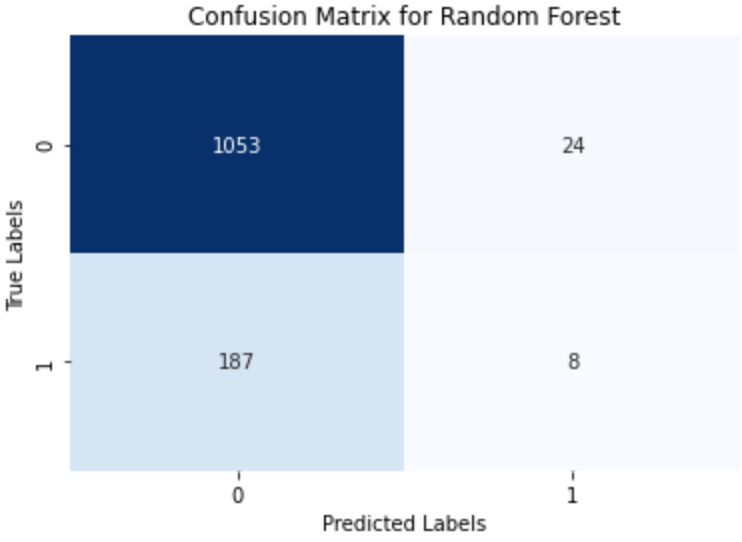
In [101…

```
# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

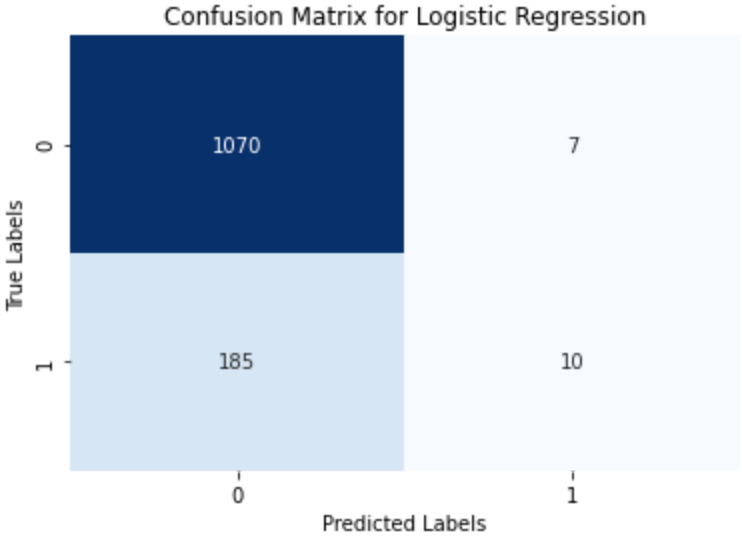
```
Classifier: XGBoost
```



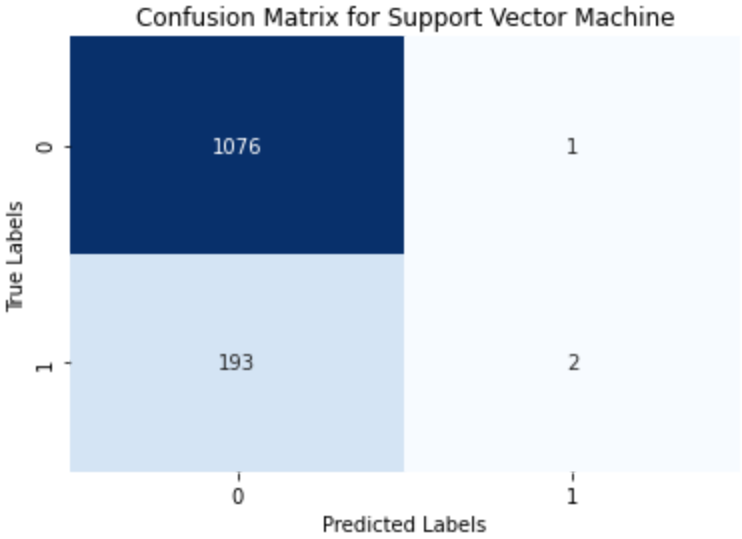

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



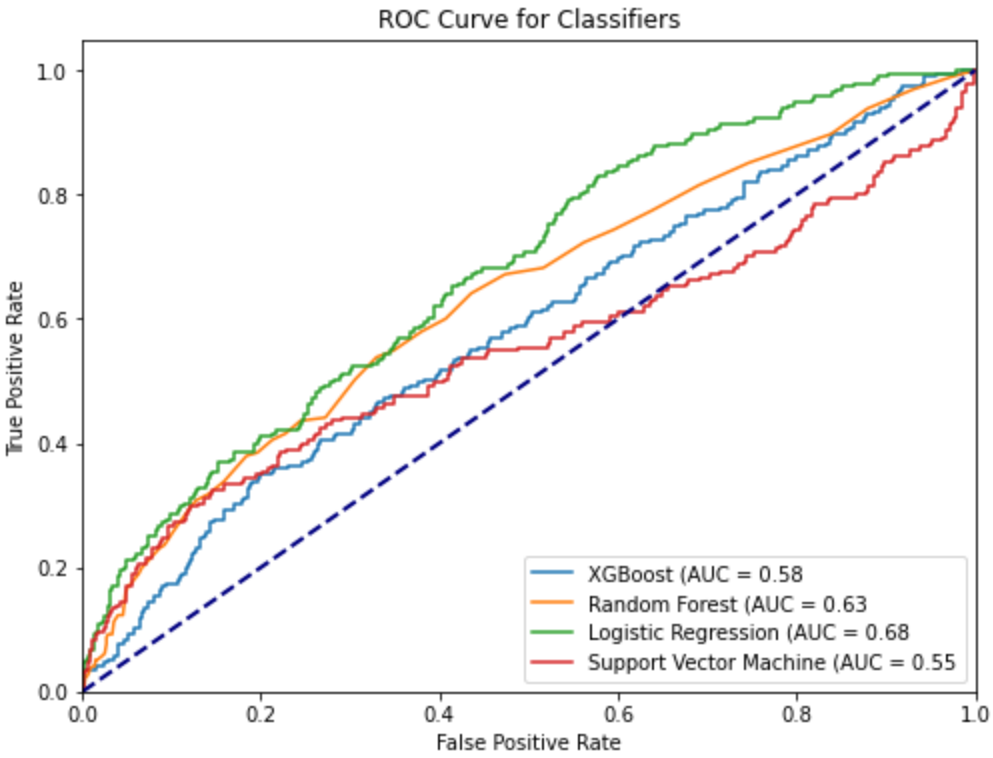

```
Classifier: Support Vector Machine
```



ROC Curve

In [102…

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in y_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Cross Validation, MSE, R2 Score and RMSE

In [103…

```
# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD', 'heartRate', 'totChol', 'glucose', 'diaBP', 'BMI'])
y = dt['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Perform k-fold cross-validation for each classifier
k_folds = 5  # You can adjust the number of folds as needed

for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Perform k-fold cross-validation and get predictions
    y_pred = cross_val_predict(clf, X_scaled, y, cv=k_folds)

    # Calculate evaluation metrics for each fold
    mae = mean_absolute_error(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    #mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # Print evaluation metrics for each fold
```

```
    print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R2 Score: {r2}')
    #print(f'MSE: {mse}')

    # Calculate and print mean accuracy
    scores = cross_val_score(clf, X_scaled, y, cv=k_folds, scoring='accuracy')
    print(f"Cross-Validation Scores (Accuracy): {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print("\n")
```

```
Classifier: XGBoost
MAE: 0.16627358490566038
RMSE: 0.40776658139879535
R2 Score: -0.2907716648588148
Cross-Validation Scores (Accuracy): [0.84316038 0.83726415 0.82429245 0.83490566 0.82900
943]
Mean Accuracy: 0.8337264150943395


Classifier: Random Forest
MAE: 0.16839622641509433
RMSE: 0.4103610927160302
R2 Score: -0.30724960100594867
Cross-Validation Scores (Accuracy): [0.83726415 0.8254717  0.82783019 0.82783019 0.83490
566]
Mean Accuracy: 0.8306603773584905


Classifier: Logistic Regression
MAE: 0.14858490566037735
RMSE: 0.38546712656253496
R2 Score: -0.15345553029936654
Cross-Validation Scores (Accuracy): [0.85023585 0.85495283 0.8490566  0.85613208 0.84669
811]
Mean Accuracy: 0.8514150943396226


Classifier: Support Vector Machine
MAE: 0.15141509433962264
RMSE: 0.3891209250857921
R2 Score: -0.17542611182887824
Cross-Validation Scores (Accuracy): [0.85023585 0.85023585 0.84787736 0.84551887 0.84905
66 ]
Mean Accuracy: 0.8485849056603774
```

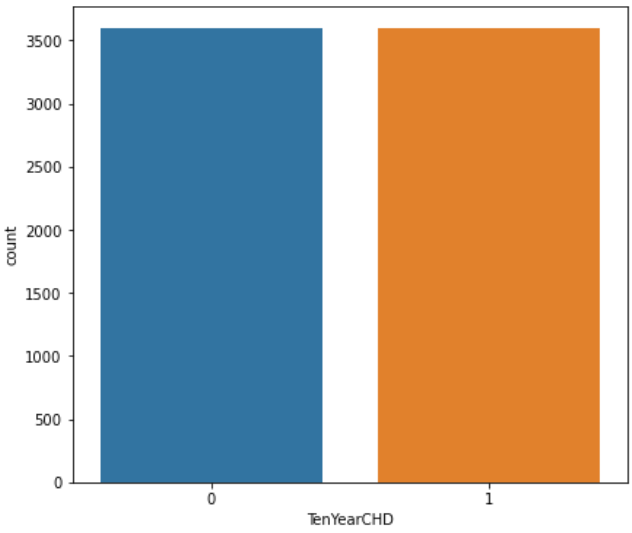
Balancing the Dataset

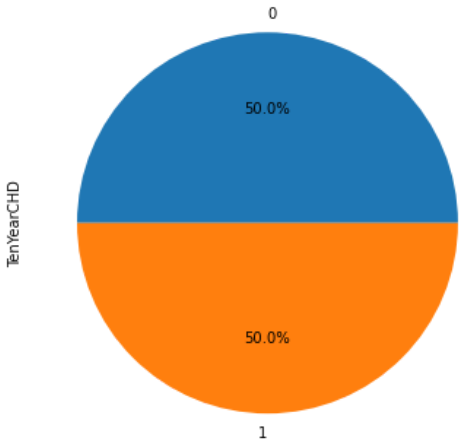
In [104…

```
# import SMOTE
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42, k_neighbors = 2)

# Split the data into features (X) and the target variable (y)
X = dt.drop(columns=['TenYearCHD'])
y = dt['TenYearCHD']

# apply SMOTE to resample the dataset
X_res, y_res = sm.fit_resample(X, y) # The object is applied
X, y = X_res, y_res # reassigning the balanced dataset to X,y
```

In [105…

```
# Plot of the dataset
dj = pd.concat([X_res,y_res], axis = 1) # creating a dataframe for the balanced dat
fig, ax=plt.subplots(1,2,figsize=(15,6)) # creating the axis shell for subplot
a = sns.countplot(x='TenYearCHD',data=dj, ax=ax[0]) # assigning each of the plot to th
a= dj['TenYearCHD'].value_counts().plot.pie(autopct="%1.1f%%", ax=ax[1])
```





Building Model for the Balanced Dataset

In [106…

```
# Split the data into features (X) and the target variable (y)
X = dj.drop(columns=['TenYearCHD'])
y = dj['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)
```

```
    # Print the classification report
    report = classification_report(y_test, y_pred)
    print(report)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       745
           1       0.92      0.87      0.89       694

    accuracy                           0.90      1439
   macro avg       0.90      0.90      0.90      1439
weighted avg       0.90      0.90      0.90      1439



Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       745
           1       0.93      0.92      0.93       694

    accuracy                           0.93      1439
   macro avg       0.93      0.93      0.93      1439
weighted avg       0.93      0.93      0.93      1439



Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       745
           1       0.66      0.68      0.67       694

    accuracy                           0.67      1439
   macro avg       0.67      0.68      0.67      1439
weighted avg       0.68      0.67      0.67      1439



Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.77      0.70      0.73       745
           1       0.71      0.77      0.74       694

    accuracy                           0.73      1439
   macro avg       0.74      0.74      0.73      1439
weighted avg       0.74      0.73      0.73      1439
```

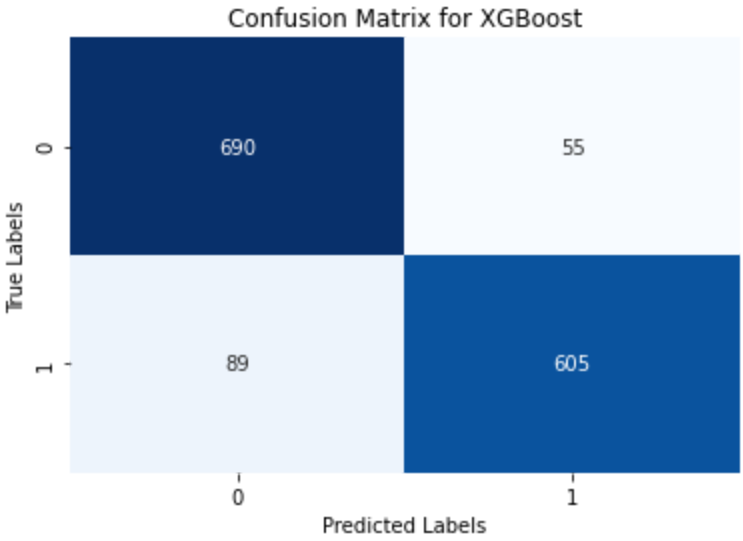
Confusion Matrix

In [107…

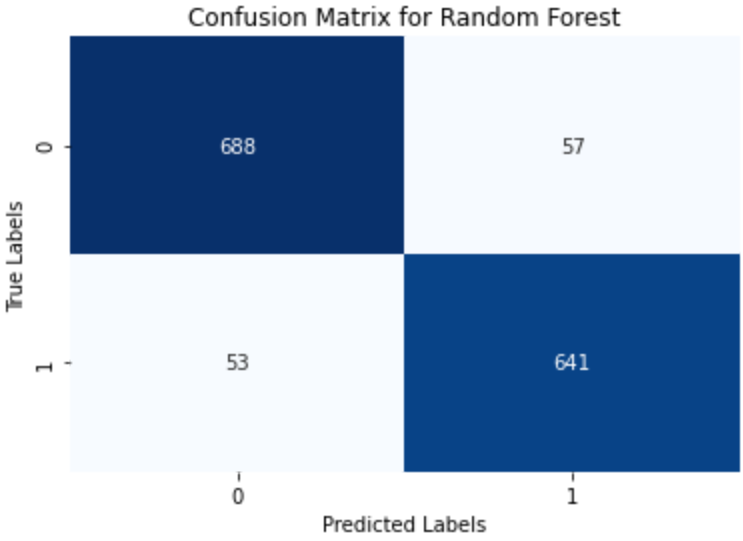
```
# Split the data into features (X) and the target variable (y)
X = dj.drop(columns=['TenYearCHD'])
y = dj['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

```
# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

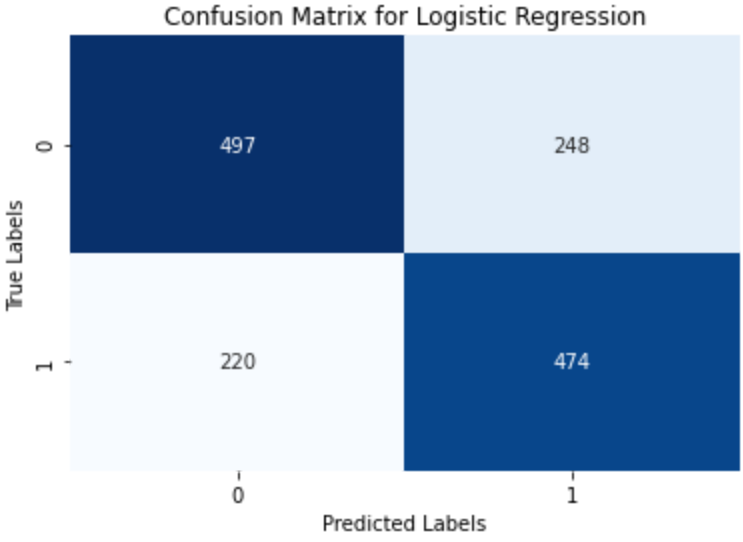
```
Classifier: XGBoost
```



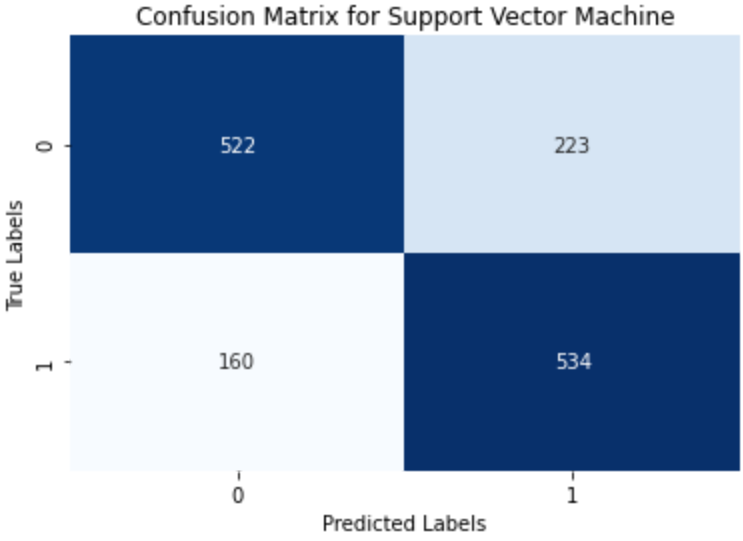

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



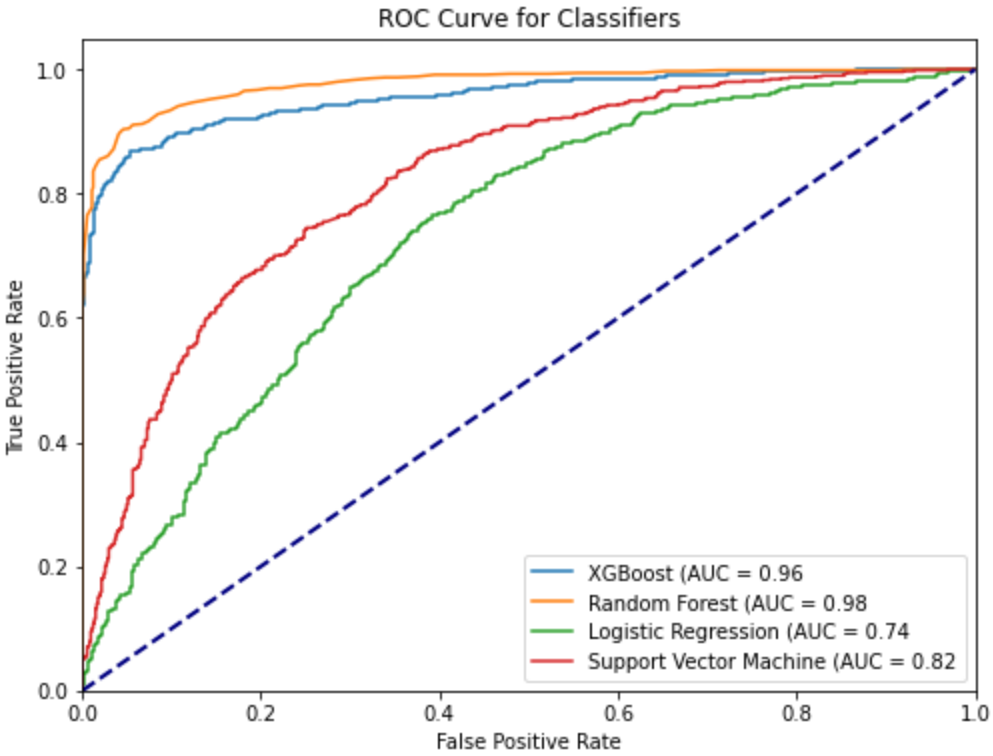

```
Classifier: Support Vector Machine
```



ROC Curve

In [108…

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in y_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Cross Validation, Mean, MAE, R2_Score, RMSE

In [109…

```
# Split the data into features (X) and the target variable (y)
X = dj.drop(columns=['TenYearCHD'])
y = dj['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Perform k-fold cross-validation for each classifier
k_folds = 5  # You can adjust the number of folds as needed

for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Perform k-fold cross-validation and get predictions
    y_pred = cross_val_predict(clf, X_scaled, y, cv=k_folds)

    # Calculate evaluation metrics for each fold
    #mae = mean_absolute_error(y, y_pred)
    rmse = np.sqrt(mean_squared_error(y, y_pred))
    mse = mean_squared_error(y, y_pred)
    r2 = r2_score(y, y_pred)

    # Print evaluation metrics for each fold
    #print(f'MAE: {mae}')
    print(f'RMSE: {rmse}')
    print(f'R2 Score: {r2}')
    print(f'MAE: {mse}')

    # Calculate and print mean accuracy
    scores = cross_val_score(clf, X_scaled, y, cv=k_folds, scoring='accuracy')
    print(f"Cross-Validation Scores (Accuracy): {scores}")
    print(f"Mean Accuracy: {scores.mean()}")
    print("\n")
```

```
Classifier: XGBoost
RMSE: 0.3549271808001541
R2 Score: 0.4961067853170189
MAE: 0.12597330367074527
Cross-Validation Scores (Accuracy): [0.57539958 0.95135511 0.94228095 0.95897079 0.94228
095]
Mean Accuracy: 0.8740574750082397


Classifier: Random Forest
RMSE: 0.2990065121816413
R2 Score: 0.6423804226918799
MAE: 0.08940489432703003
Cross-Validation Scores (Accuracy): [0.73870744 0.94927033 0.95062587 0.96105702 0.94993
```

```
046]
Mean Accuracy: 0.9099182228425127


Classifier: Logistic Regression
RMSE: 0.5664910631094455
R2 Score: -0.2836484983314793
MAE: 0.32091212458286983
Cross-Validation Scores (Accuracy): [0.61917999 0.68797776 0.70236439 0.69054242 0.69541
029]
Mean Accuracy: 0.6790949711059198


Classifier: Support Vector Machine
RMSE: 0.5063556018754131
R2 Score: -0.025583982202447064
MAE: 0.25639599555061177
Cross-Validation Scores (Accuracy): [0.65601112 0.76441974 0.7705146  0.77329624 0.75382
476]
Mean Accuracy: 0.7436132919534408
```

Spliting the whole dataset into Train and Test

In [110…

```
# Split the data into features (X) and target (y)
X = dj.drop(columns=['TenYearCHD'])
y = dj['TenYearCHD']

# Split the data into training and testing sets
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, y, test_size=0.2, random_sta
```

In [111…

```
dj.columns
```

Out[111…

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

Concatenate the Train Dataset

In [112…

```
df_train = pd.concat([X_train1, y_train1], axis = 1)
df_train.head()
```

Out[112…

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabete
6256	0	50	4.000000	0	0.000000	0.374349	0	1	
4668	0	60	1.000000	0	0.961139	0.000000	0	1	
940	0	53	2.000000	0	0.000000	0.000000	0	1	
1511	0	38	2.000000	0	0.000000	0.000000	0	0	
6034	1	59	1.016243	1	40.000000	0.000000	0	0	

Concatenate the Test Dataset

In [113…

```
df_test = pd.concat([X_test1, y_test1], axis = 1)
df_test.head()
```

Out[113…

	sex	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabete
4054	0	48	3.000000	0	0.000000	0.0	0	0	
3813	0	44	3.000000	0	0.000000	0.0	0	0	
5065	0	47	2.072889	0	0.218668	0.0	0	0	
4339	0	51	1.446289	1	22.768557	0.0	0	0	
1615	0	49	2.000000	0	0.000000	0.0	0	0	

Hyperparameter Tuning Using Randomized Search CV

In [114…

```
X = dj.drop(columns = 'TenYearCHD')
y = dj['TenYearCHD']

# Define a list of classifiers along with their hyperparameter grids
classifiers = [
    ("XGBoost", XGBClassifier(), {
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 4, 5],
        'learning_rate': [0.1, 0.01, 0.001],
        'subsample': [0.7, 0.8, 0.9]
    }),
    ("Random Forest", RandomForestClassifier(), {
        'n_estimators': [100, 200, 300],
        'max_depth': [10, 20, 30]
    }),
    ("Logistic Regression", LogisticRegression(), {
        'C': [0.1, 1, 10],
        'penalty': ['l1', 'l2']
    }),
    ("Support Vector Machine", SVC(), {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
```

```
        'gamma': ['scale', 'auto']
    })
]

# Iterate through the classifiers and perform RandomizedSearchCV
for clf_name, clf, param_grid in classifiers:
    print(f"Classifier: {clf_name}")

    # Create a RandomizedSearchCV object
    random_search = RandomizedSearchCV(clf, param_distributions=param_grid, n_iter=10,

    # Perform RandomizedSearchCV on the training data
    random_search.fit(X_train, y_train)

    # Get the best estimator and its performance
    best_estimator = random_search.best_estimator_
    best_score = random_search.best_score_

    # Make predictions on the testing data using the best estimator
    y_pred = best_estimator.predict(X_test)

    # Print the classification report
    report = classification_report(y_test, y_pred)

    # Print best hyperparameters, best cross-validation score, and the classification r
    print("Best Hyperparameters:", random_search.best_params_)
    print("Best Cross-Validation Score:", best_score)
    print(report)
    print("\n")
```

```
Classifier: XGBoost
Best Hyperparameters: {'subsample': 0.8, 'n_estimators': 200, 'max_depth': 5, 'learning_
rate': 0.1}
Best Cross-Validation Score: 0.8866682280058928
              precision    recall  f1-score   support

           0       0.88      0.94      0.91       745
           1       0.93      0.86      0.89       694

    accuracy                           0.90      1439
   macro avg       0.90      0.90      0.90      1439
weighted avg       0.90      0.90      0.90      1439



Classifier: Random Forest
Best Hyperparameters: {'n_estimators': 200, 'max_depth': 30}
Best Cross-Validation Score: 0.9129141389340083
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       745
           1       0.92      0.92      0.92       694

    accuracy                           0.92      1439
   macro avg       0.92      0.92      0.92      1439
weighted avg       0.92      0.92      0.92      1439



Classifier: Logistic Regression
Best Hyperparameters: {'penalty': 'l2', 'C': 10}
Best Cross-Validation Score: 0.6839947115929437
              precision    recall  f1-score   support
```

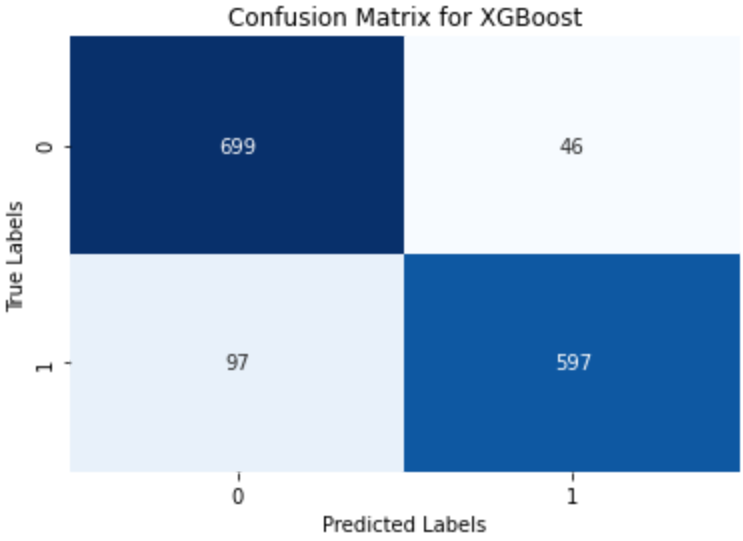
```
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       745
           1       0.66      0.68      0.67       694

    accuracy                           0.67      1439
   macro avg       0.67      0.68      0.67      1439
weighted avg       0.68      0.67      0.67      1439


Classifier: Support Vector Machine
Best Hyperparameters: {'kernel': 'rbf', 'gamma': 'scale', 'C': 10}
Best Cross-Validation Score: 0.780462206776716
              precision    recall  f1-score   support

           0       0.83      0.75      0.79       745
           1       0.76      0.83      0.79       694

    accuracy                           0.79      1439
   macro avg       0.79      0.79      0.79      1439
weighted avg       0.80      0.79      0.79      1439
```

Confusion Matrix

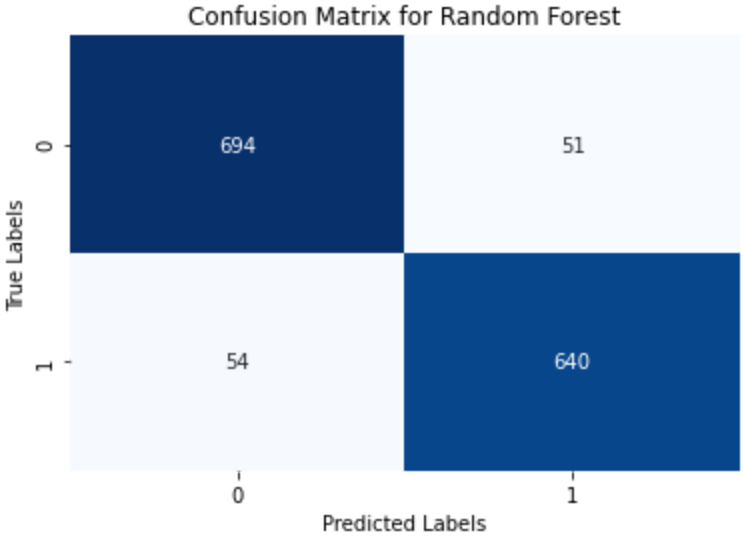
In [115…

```python
# Iterate through the classifiers and perform RandomizedSearchCV
for clf_name, clf, param_grid in classifiers:
    print(f"Classifier: {clf_name}")

    # Create a RandomizedSearchCV object
    random_search = RandomizedSearchCV(clf, param_distributions=param_grid, n_iter=10,

    # Perform RandomizedSearchCV on the training data
    random_search.fit(X_train, y_train)

    # Get the best estimator and its performance
    best_estimator = random_search.best_estimator_
    best_score = random_search.best_score_

    # Make predictions on the testing data using the best estimator
    y_pred = best_estimator.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

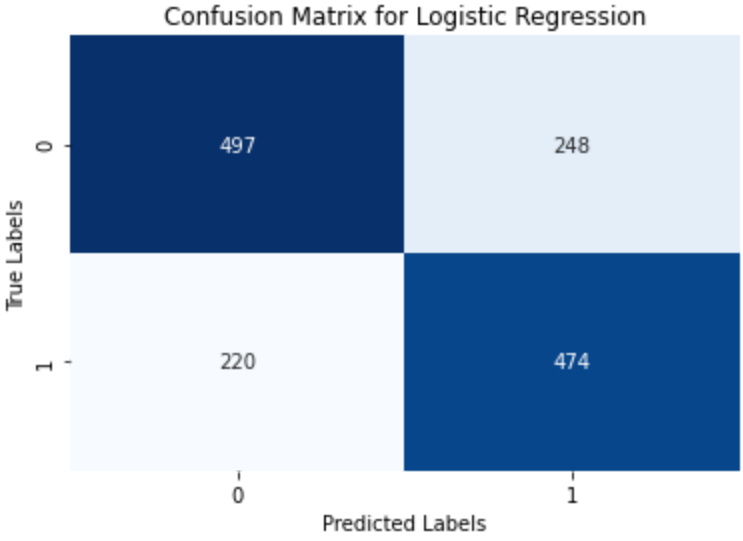
```
Classifier: XGBoost
```



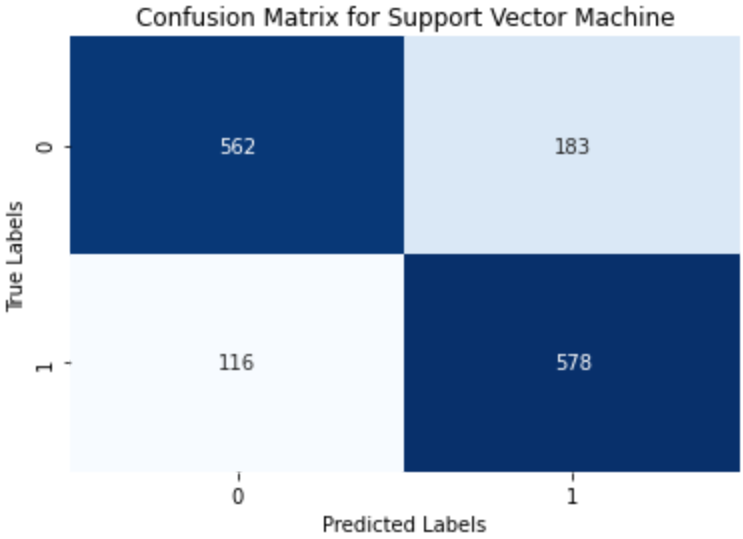

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



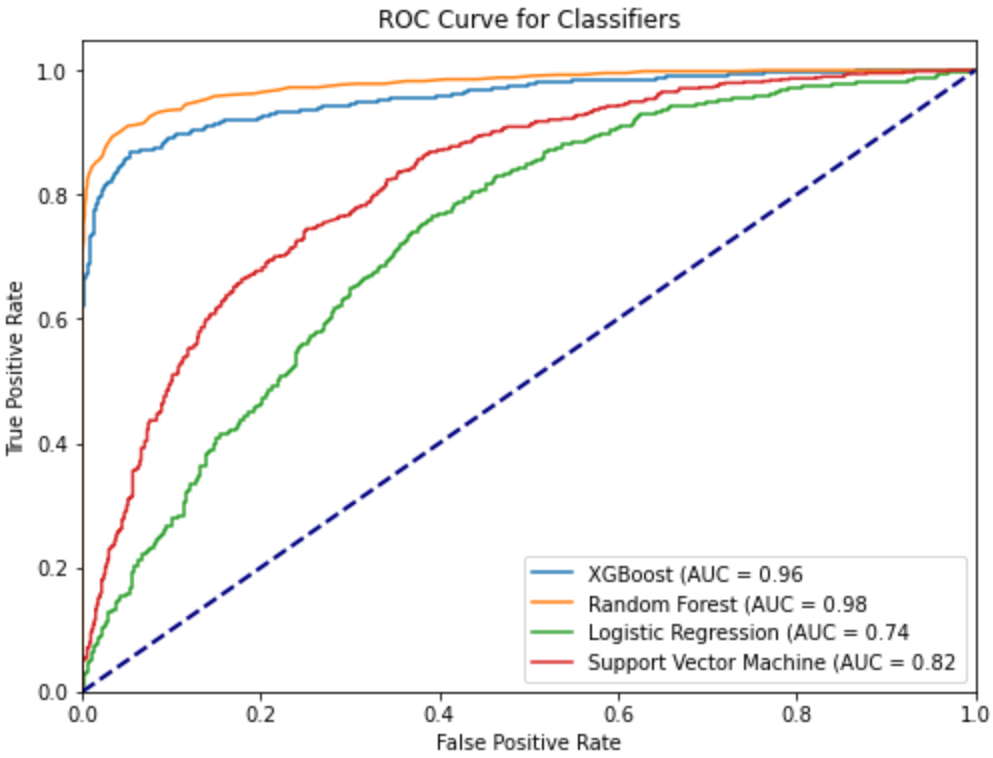

```
Classifier: Support Vector Machine
```



ROC Curve

In [116…

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in y_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Using Bagging (Ensemble Technique)

In [117…

```
# Define the base classifiers
base_classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Iterate through the base classifiers and apply Bagging
for clf_name, base_classifier in base_classifiers:
    print(f"Classifier: {clf_name}")

    # Create the BaggingClassifier
    bagging_classifier = BaggingClassifier(
        base_classifier,
        n_estimators=100,  # Number of base classifiers (adjust as needed)
        max_samples=0.8,   # Fraction of samples to use for each base classifier
        random_state=42    # Set a random seed for reproducibility
    )

    # Fit the BaggingClassifier on the training data
    bagging_classifier.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = bagging_classifier.predict(X_test)

    # Evaluate the BaggingClassifier
    classification = classification_report(y_test, y_pred)

    print(classification)
    print("\n")
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       745
           1       0.93      0.87      0.90       694

    accuracy                           0.91      1439
   macro avg       0.91      0.90      0.91      1439
weighted avg       0.91      0.91      0.91      1439


Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       745
           1       0.89      0.91      0.90       694

    accuracy                           0.90      1439
   macro avg       0.90      0.91      0.90      1439
weighted avg       0.91      0.90      0.90      1439


Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       745
           1       0.66      0.68      0.67       694

    accuracy                           0.67      1439
   macro avg       0.67      0.67      0.67      1439
weighted avg       0.67      0.67      0.67      1439


Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.77      0.70      0.73       745
           1       0.70      0.77      0.74       694

    accuracy                           0.73      1439
   macro avg       0.74      0.74      0.73      1439
weighted avg       0.74      0.73      0.73      1439
```

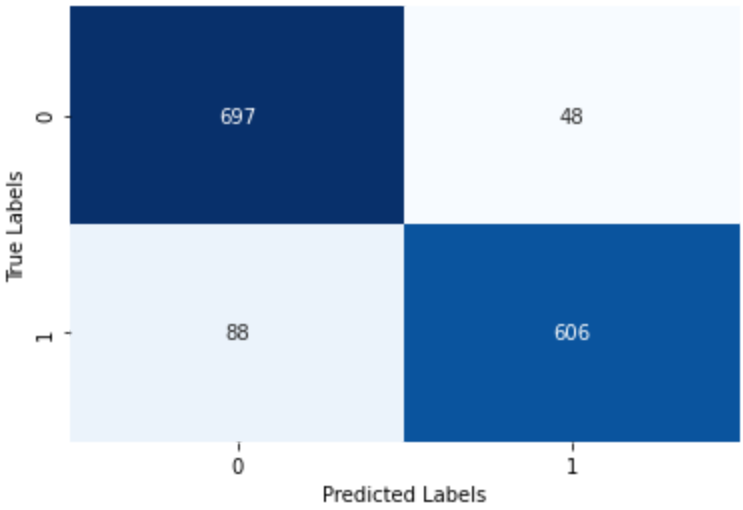
Confusion Matrix

In [118…

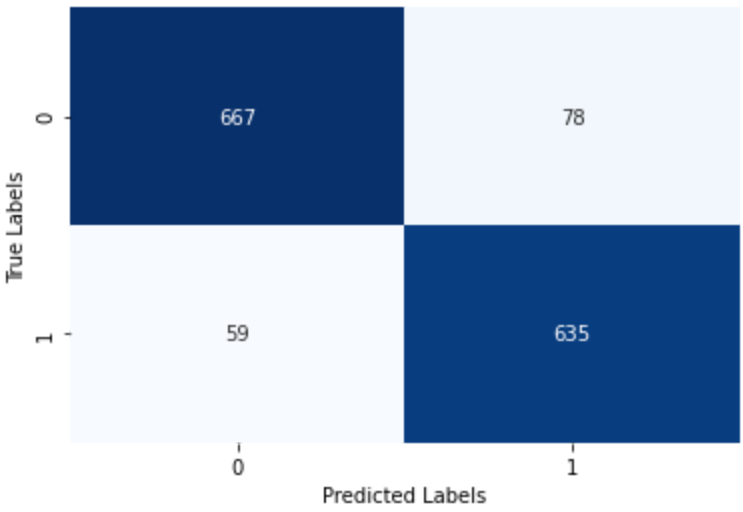
```python
# Iterate through the base classifiers and apply Bagging
for clf_name, base_classifier in base_classifiers:
    print(f"Classifier: {clf_name}")

    # Create the BaggingClassifier
    bagging_classifier = BaggingClassifier(
        base_classifier,
        n_estimators=100,
        max_samples=0.8,
        random_state=42
```

```
    )

    # Fit the BaggingClassifier on the training data
    bagging_classifier.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = bagging_classifier.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Print the confusion matrix
    print("Confusion Matrix:")

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    #plt.title(f'Confusion Matrix for {conf_matrix}')
    plt.show()
    print("\n")
```

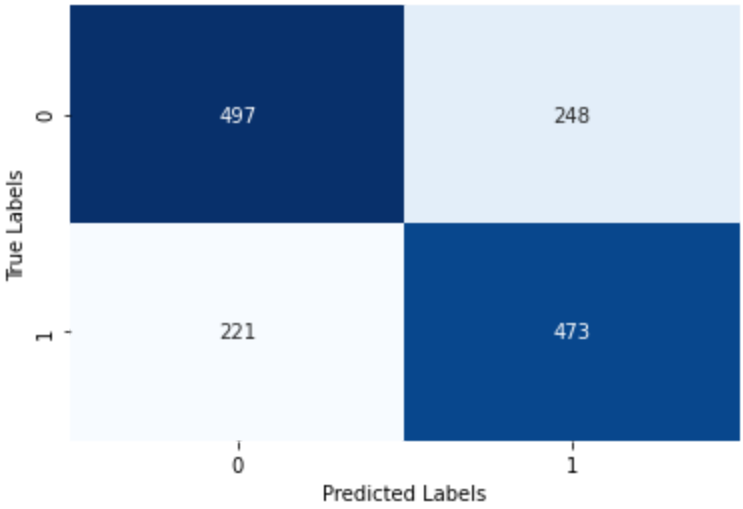
```
Classifier: XGBoost
Confusion Matrix:
```



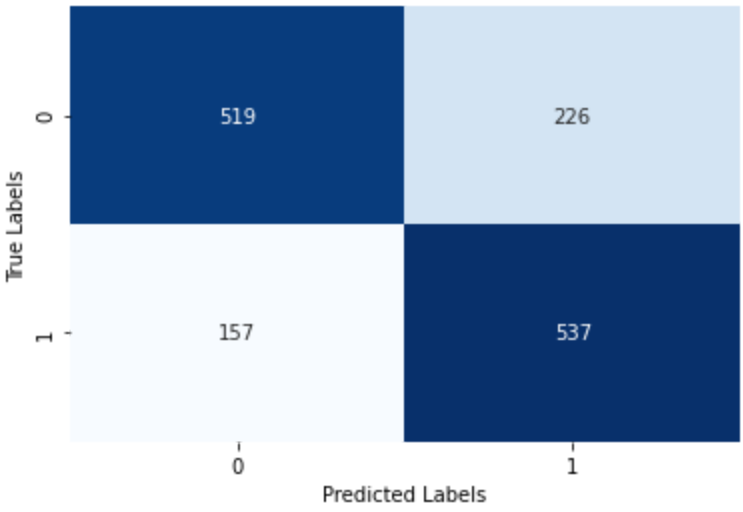

```
Classifier: Random Forest
Confusion Matrix:
```



```
Classifier: Logistic Regression
Confusion Matrix:
```




```
Classifier: Support Vector Machine
Confusion Matrix:
```



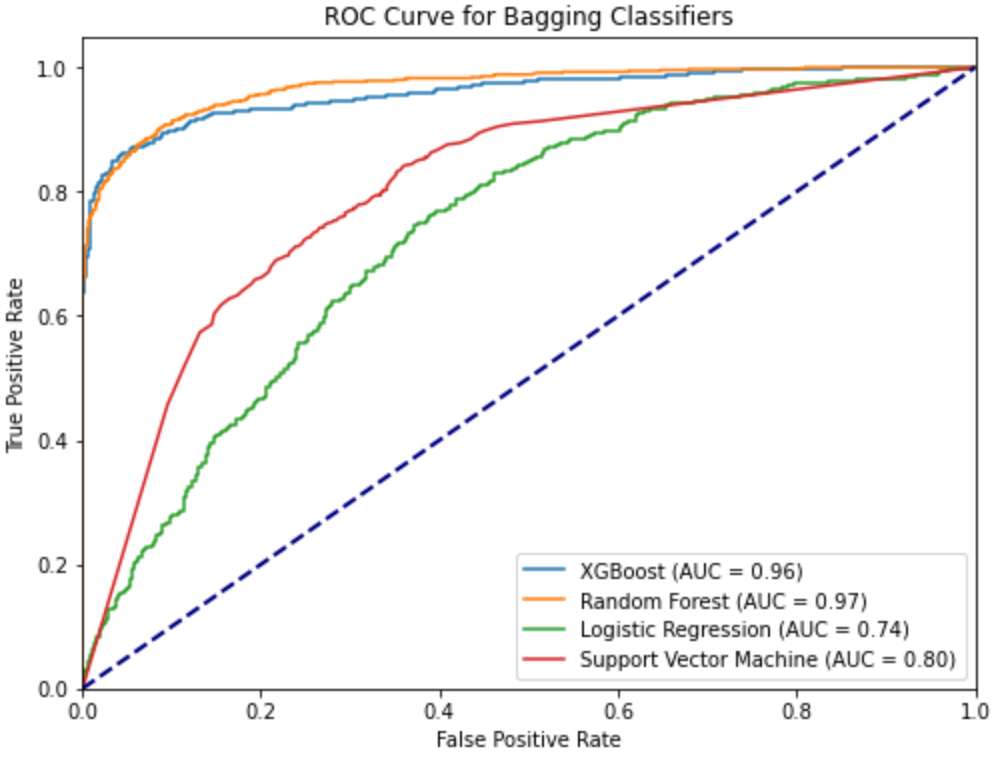
ROC Curve

In [119…

```
# Create an empty dictionary to store predicted probabilities for each BaggingClassifie
bagging_probs = {}

# Iterate through the base classifiers and apply Bagging
for clf_name, base_classifier in base_classifiers:
    print(f"Classifier: {clf_name}")

    # Create the BaggingClassifier
    bagging_classifier = BaggingClassifier(
        base_classifier,
        n_estimators=100,  # Number of base classifiers
        max_samples=0.8,   # Fraction of samples to use for each base classifier
        random_state=42    # Set a random seed for reproducibility
    )

    # Fit the BaggingClassifier on the training data
    bagging_classifier.fit(X_train, y_train)
```

```
    # Calculate the probability estimates for the positive class
    y_prob = bagging_classifier.predict_proba(X_test)[:, 1]

    # Store the predicted probabilities in the dictionary
    bagging_probs[clf_name] = y_prob

# Plot ROC curves for each BaggingClassifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in bagging_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Bagging Classifiers')
plt.legend(loc="lower right")
plt.show()
```

```
Classifier: XGBoost
Classifier: Random Forest
Classifier: Logistic Regression
Classifier: Support Vector Machine
```



Using AdaBoost

In [120…

```
X = dj.drop(columns = 'TenYearCHD')
y = dj['TenYearCHD']

# Feature scaling
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression())
]

# Iterate through the classifiers
for clf_name, base_classifier in classifiers:
    print(f"Classifier: {clf_name}")

    # Create the AdaBoostClassifier
    adaboost_classifier = AdaBoostClassifier(
        base_classifier,
        n_estimators=40,
        learning_rate=0.5,
        algorithm='SAMME.R',
        random_state=42
    )

    # Cross-validated scores
    cv_scores = cross_val_score(adaboost_classifier, X_train_scaled, y_train, cv=5, sco
    print(f"Cross-Validated Scores: {cv_scores}")
    print(f"Average Cross-Validated Score: {np.mean(cv_scores):.4f}")

    # Make cross-validated predictions
    y_cv_pred = cross_val_predict(adaboost_classifier, X_train_scaled, y_train, cv=5)

    # Calculate Mean Absolute Error (MAE)
    mae = mean_absolute_error(y_train, y_cv_pred)
    print(f"Mean Absolute Error (MAE): {mae:.4f}")

    # Calculate Root Mean Squared Error (RMSE)
    rmse = np.sqrt(mean_squared_error(y_train, y_cv_pred))
    print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

    # Calculate R^2 Score
    r2 = r2_score(y_train, y_cv_pred)
    print(f"R^2 Score: {r2:.4f}")

    # Fit the AdaBoostClassifier on the scaled training data
    adaboost_classifier.fit(X_train_scaled, y_train)

    # Make predictions on the scaled test data
    y_pred = adaboost_classifier.predict(X_test_scaled)

    # Evaluate the AdaBoostClassifier
    classification = classification_report(y_test, y_pred)
    print("Classification Report:")
    print(classification)

    print("\n")
```

```
Classifier: XGBoost
Cross-Validated Scores: [0.49522155 0.49522155 0.49609036 0.49565217 0.49565217]
Average Cross-Validated Score: 0.4956
```

```
Mean Absolute Error (MAE): 0.5044
Root Mean Squared Error (RMSE): 0.7102
R^2 Score: -1.0179
Classification Report:
              precision    recall  f1-score   support

           0       0.52      1.00      0.68       745
           1       0.00      0.00      0.00       694

    accuracy                           0.52      1439
   macro avg       0.26      0.50      0.34      1439
weighted avg       0.27      0.52      0.35      1439



Classifier: Random Forest
Cross-Validated Scores: [0.91398784 0.8931364  0.9192007  0.89826087 0.91565217]
Average Cross-Validated Score: 0.9080
Mean Absolute Error (MAE): 0.0920
Root Mean Squared Error (RMSE): 0.3032
R^2 Score: 0.6322
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       745
           1       0.93      0.92      0.92       694

    accuracy                           0.93      1439
   macro avg       0.93      0.93      0.93      1439
weighted avg       0.93      0.93      0.93      1439



Classifier: Logistic Regression
Cross-Validated Scores: [0.6602954  0.6698523  0.69939183 0.67565217 0.69478261]
Average Cross-Validated Score: 0.6800
Mean Absolute Error (MAE): 0.3200
Root Mean Squared Error (RMSE): 0.5657
R^2 Score: -0.2801
Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.68      0.69       745
           1       0.66      0.67      0.67       694

    accuracy                           0.68      1439
   macro avg       0.68      0.68      0.68      1439
weighted avg       0.68      0.68      0.68      1439
```

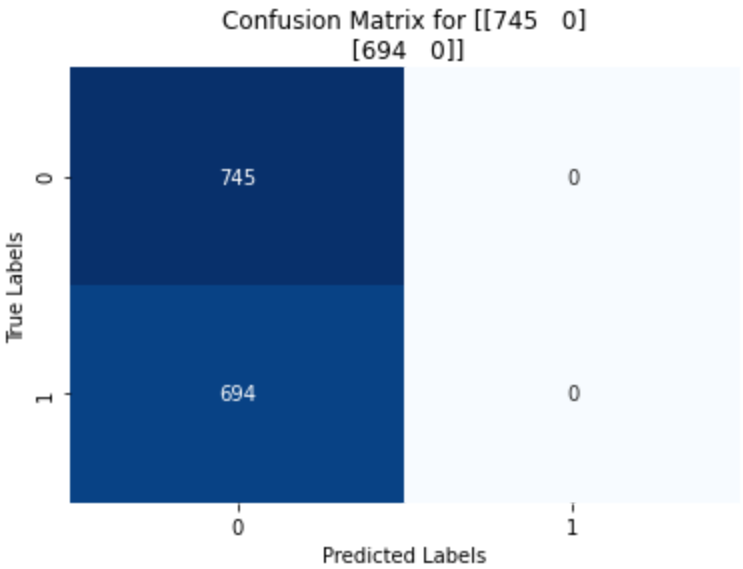
Confusion Matrix

In [121…

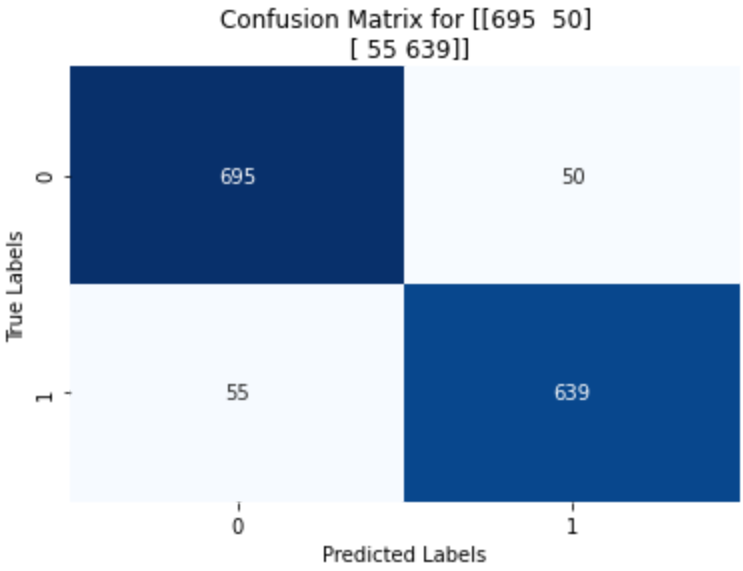
```
# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression())
]

# Iterate through the classifiers and apply AdaBoost
```

```
for clf_name, base_classifier in classifiers:
    print(f"Classifier: {clf_name}")

    # Create the AdaBoostClassifier
    adaboost_classifier = AdaBoostClassifier(
        base_classifier,
        n_estimators=50,  # Number of base classifiers (adjust as needed)
        learning_rate=1.0,
        algorithm='SAMME.R',
        random_state=42
    )

    # Fit the AdaBoostClassifier on the training data
    adaboost_classifier.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = adaboost_classifier.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test, y_pred)

    # Print the confusion matrix
    print("Confusion Matrix:")

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {conf_matrix}')
    plt.show()
    print("\n")
```

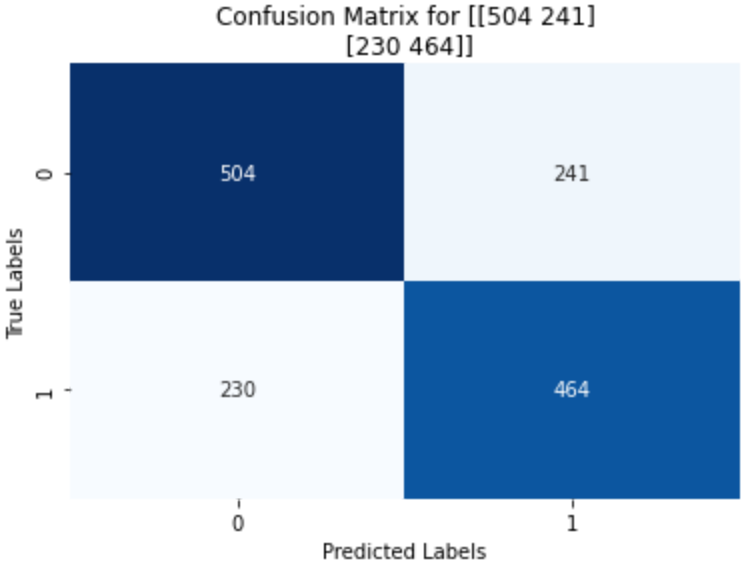
```
Classifier: XGBoost
Confusion Matrix:
```




```
Classifier: Random Forest
Confusion Matrix:
```



Classifier: Logistic Regression
Confusion Matrix:



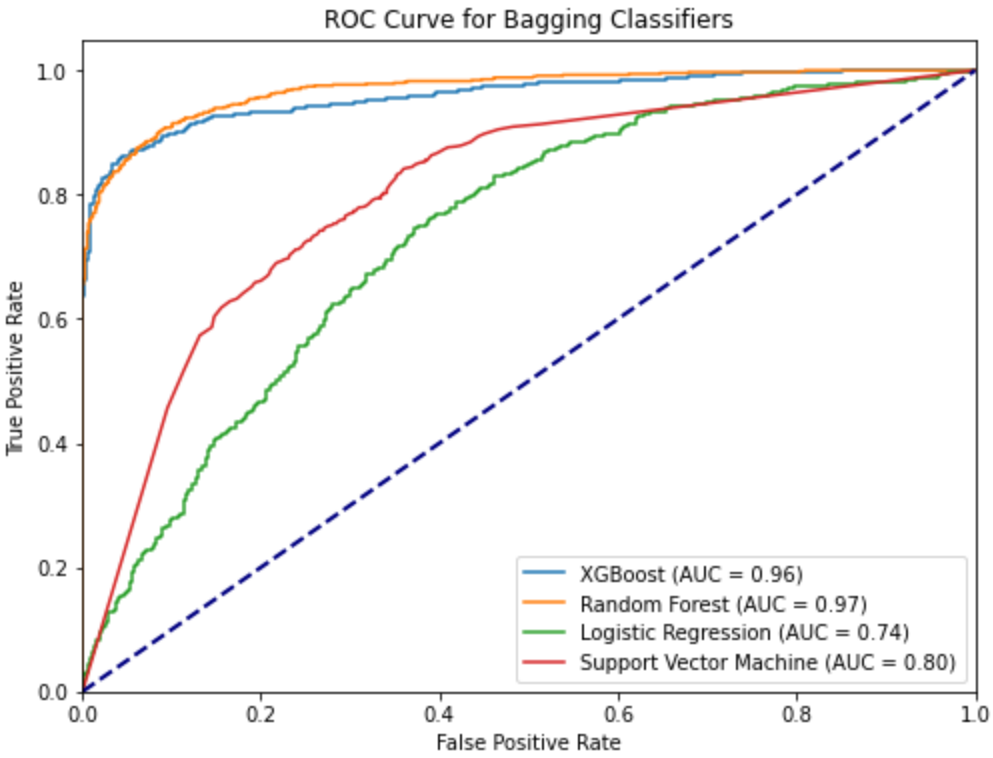
ROC Curve

In [122…

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression())
]

# Iterate through the classifiers and apply AdaBoost
for clf_name, base_classifier in classifiers:
    print(f"Classifier: {clf_name}")

    # Create the AdaBoostClassifier
    adaboost_classifier = AdaBoostClassifier(
```

```
        base_classifier,
        n_estimators=50,  # Number of base classifiers (adjust as needed)
        learning_rate=1.0,
        algorithm='SAMME.R',
        random_state=42
    )

    # Fit the AdaBoostClassifier on the training data
    adaboost_classifier.fit(X_train, y_train)

    # Calculate the probability estimates for the positive class
    y_prob = adaboost_classifier.predict_proba(X_test)[:, 1]

    # Compute ROC curve and ROC area
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

# Plot ROC curves for each BaggingClassifier
plt.figure(figsize=(8, 6))

for clf_name, y_prob in bagging_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    auc = roc_auc_score(y_test, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Bagging Classifiers')
plt.legend(loc="lower right")
plt.show()
```

```
Classifier: XGBoost
Classifier: Random Forest
Classifier: Logistic Regression
```



Using Grid Search CV

In [123…

```
X = dj.drop(columns = 'TenYearCHD')
y = dj['TenYearCHD']

# Define a list of classifiers along with their hyperparameter grids
classifiers = [
    ("XGBoost", XGBClassifier(), {
        'n_estimators': [100, 200, 300],
        'max_depth': [3, 4, 5],
        'learning_rate': [0.1, 0.01, 0.001],
        'subsample': [0.7, 0.8, 0.9]
    }),
    ("Random Forest", RandomForestClassifier(), {
        'n_estimators': [100, 200, 300],
        'max_depth': [10, 20, 30]
    }),
    ("Logistic Regression", LogisticRegression(), {
        'C': [0.1, 1, 10],
        'penalty': ['l1', 'l2']
    }),
    ("Support Vector Machine", SVC(), {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    })
]

# Iterate through the classifiers and perform GridSearchCV
for clf_name, clf, param_grid in classifiers:
    print(f"Classifier: {clf_name}")

    # Create a GridSearchCV object
```

```
    grid_search = GridSearchCV(clf, param_grid, scoring='accuracy', cv=5, n_jobs=-1)

    # Perform GridSearchCV on the training data
    grid_search.fit(X_train, y_train)

    # Get the best estimator and its performance
    best_estimator = grid_search.best_estimator_
    best_score = grid_search.best_score_

    # Make predictions on the testing data using the best estimator
    y_pred = best_estimator.predict(X_test)

    # Calculate evaluation metrics
    classification = classification_report (y_test, y_pred)

    # Print evaluation metrics and best hyperparameters
    print("Best Hyperparameters:", grid_search.best_params_)
    print("Best Cross-Validation Score:", best_score)
    print (classification)
    print("\n")
```

```
Classifier: XGBoost
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300, 'subsa
mple': 0.7}
Best Cross-Validation Score: 0.8930999886676991
              precision    recall  f1-score   support

           0       0.89      0.92      0.91       745
           1       0.92      0.88      0.90       694

    accuracy                           0.90      1439
   macro avg       0.90      0.90      0.90      1439
weighted avg       0.90      0.90      0.90      1439



Classifier: Random Forest
Best Hyperparameters: {'max_depth': 30, 'n_estimators': 300}
Best Cross-Validation Score: 0.9108283911910249
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       745
           1       0.93      0.93      0.93       694

    accuracy                           0.93      1439
   macro avg       0.93      0.93      0.93      1439
weighted avg       0.93      0.93      0.93      1439



Classifier: Logistic Regression
Best Hyperparameters: {'C': 10, 'penalty': 'l2'}
Best Cross-Validation Score: 0.6839947115929437
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       745
           1       0.66      0.68      0.67       694

    accuracy                           0.67      1439
   macro avg       0.67      0.68      0.67      1439
weighted avg       0.68      0.67      0.67      1439
```

```
Classifier: Support Vector Machine
Best Hyperparameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best Cross-Validation Score: 0.780462206776716
              precision    recall  f1-score   support

           0       0.83      0.75      0.79       745
           1       0.76      0.83      0.79       694

    accuracy                           0.79      1439
   macro avg       0.79      0.79      0.79      1439
weighted avg       0.80      0.79      0.79      1439
```

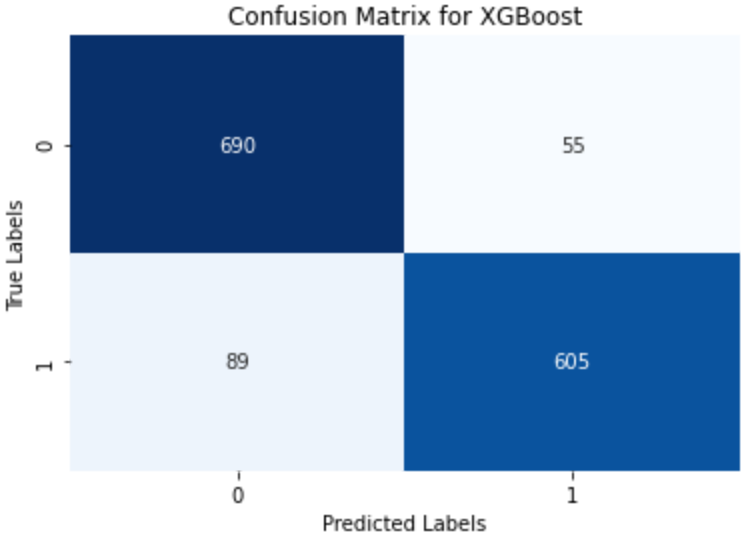
Confusion Matrix for Grid Search CV

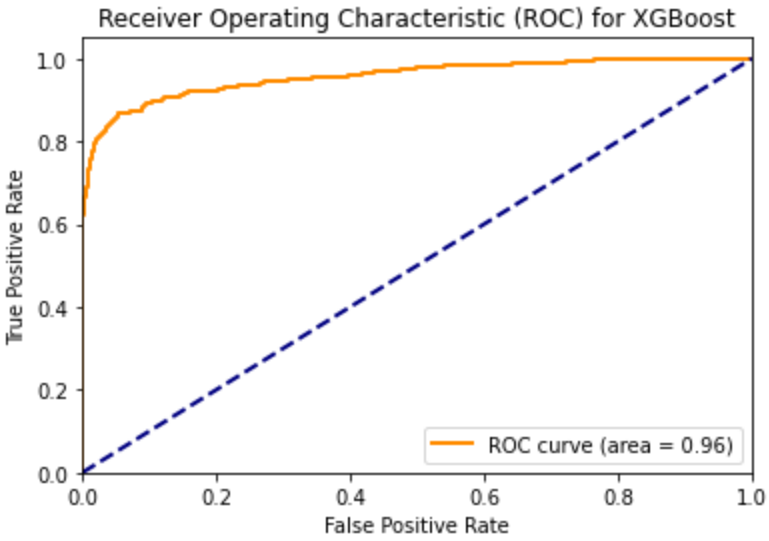
In [124…

```
base_classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))
]

for clf_name, classifier in base_classifiers:
    print(f"Classifier: {clf_name}")

    # Fit the classifier on the training data
    classifier.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = classifier.predict(X_test)

    # Calculate and display the confusion matrix
    confusion = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()

    # Calculate class probabilities
    if hasattr(classifier, "predict_proba"):
        y_prob = classifier.predict_proba(X_test)[:, 1]
    else:
        y_prob = classifier.decision_function(X_test)

    # Calculate and display the ROC curve
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = roc_auc_score(y_test, y_prob)
    plt.figure(figsize=(6, 4))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.form
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'Receiver Operating Characteristic (ROC) for {clf_name}')
```

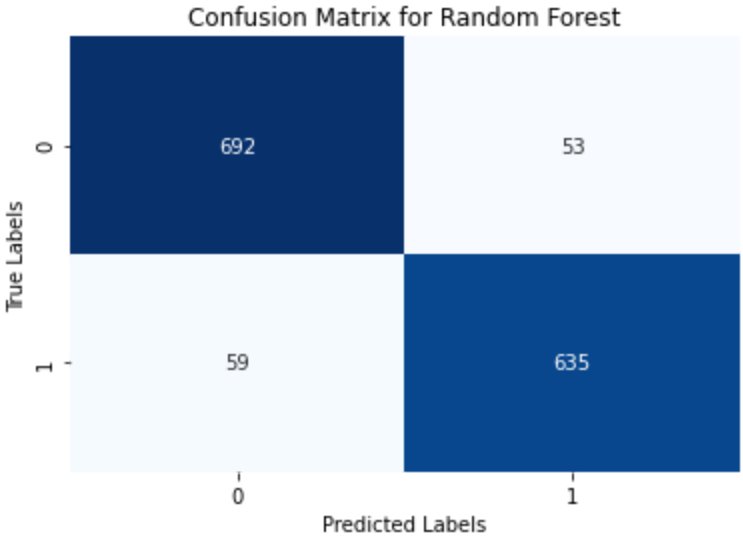
```
    plt.legend(loc='lower right')
    plt.show()
```

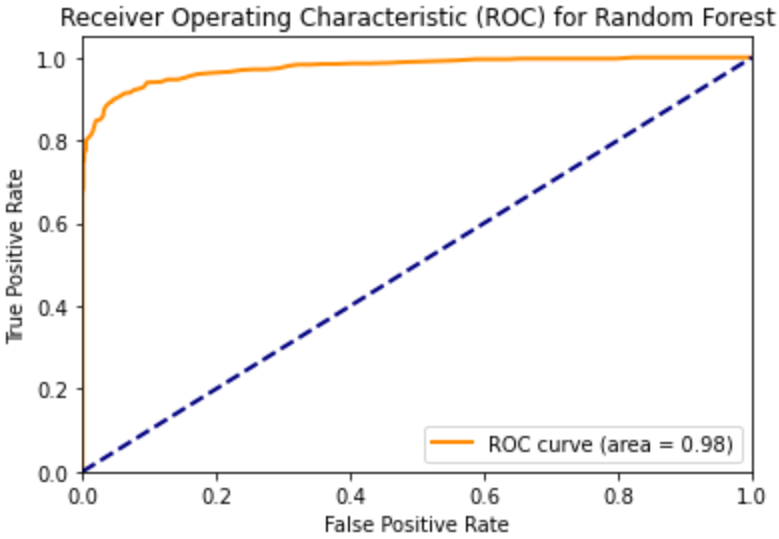
```
Classifier: XGBoost
```



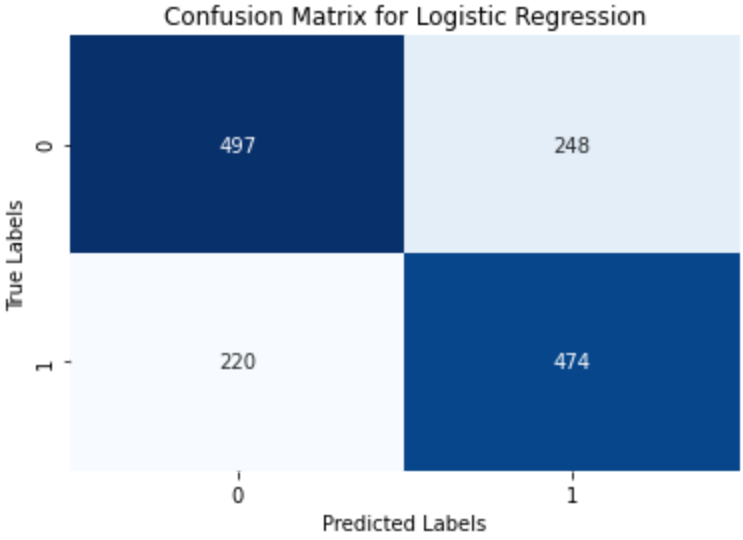


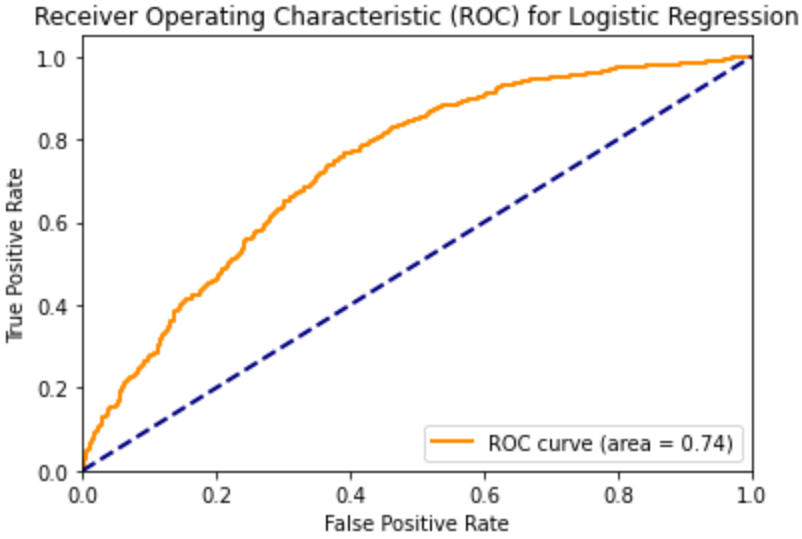

```
Classifier: Random Forest
```



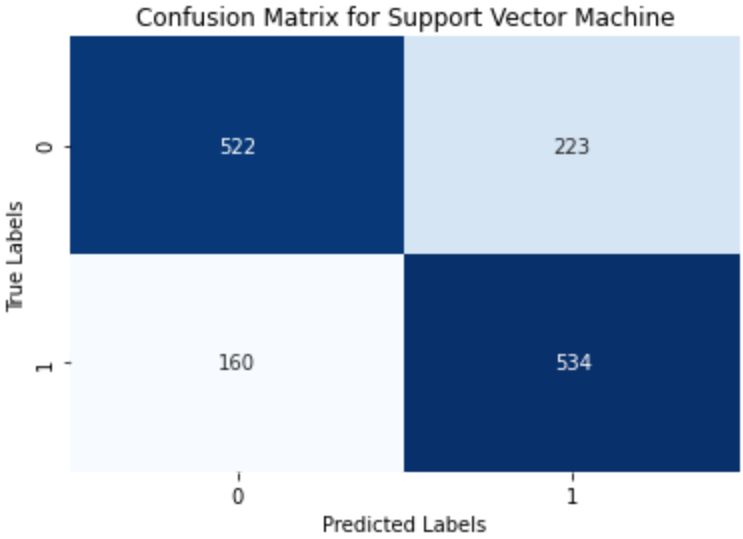


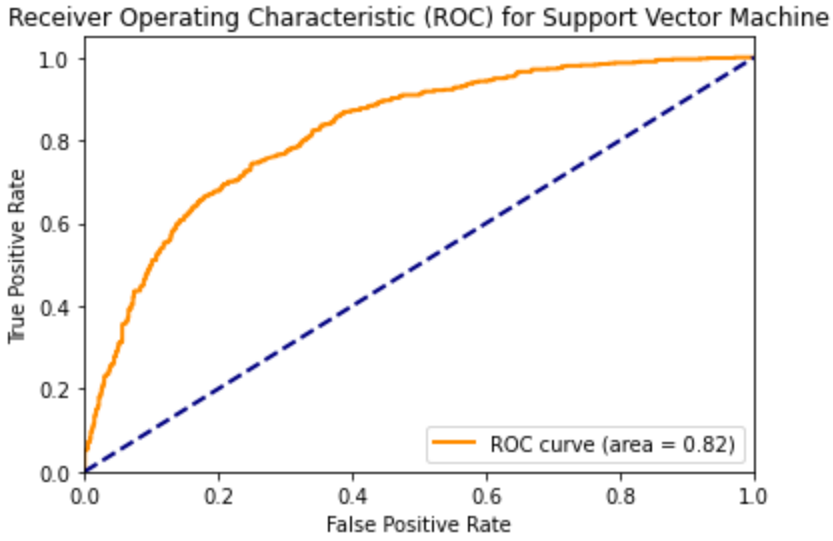

```
Classifier: Logistic Regression
```






```
Classifier: Support Vector Machine
```





Data Transformation for Train and Test Dataset

Age

In [125…

```
# Create bin for age feature for train dataset
age_bins = [32, 37, 42, 47, 52, 57, 61, 70]
age_intervals = ['32-36', '37-41', '42-46', '47-51', '52-56', '57-61', '62+']
df_train['age_range'] = pd.cut(df_train["age"], bins = age_bins, labels = age_intervals
```

In [126…

```
# Create bin for age feature for test dataset
age_bins = [32, 37, 42, 47, 52, 57, 61, 70]
age_intervals = ['32-36', '37-41', '42-46', '47-51', '52-56', '57-61', '62+']
df_test['age_range'] = pd.cut(df_test["age"], bins = age_bins, labels = age_intervals)
```

Cigarette Per Day

In [127…
```
# Create bin for cigarette per day feature for train dataset
cig_bins = [1, 6, 12, 17, 22, 27, 31, 37, 39, 40]
cig_intervals = ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30', '30-35', '36-40', '
df_train["cig_per_day_range"] = pd.cut(df_train["cigsPerDay"], bins = cig_bins, labels
```

In [128…
```
# Create bin for cigarette per day feature for test dataset
cig_bins = [1, 6, 12, 17, 22, 27, 31, 33, 36]
cig_intervals = ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30', '30-35', '36+']
df_test["cig_per_day_range"] = pd.cut(df_test["cigsPerDay"], bins = cig_bins, labels =
```

Total Cholesterol

In [129…
```
# Create bin for total cholesterol feature for train dataset
chol_bins = [100, 151, 201, 251, 301, 351, 401, 405]
chol_intervals = ['100-150', '151-200', '201-250', '251-300', '301-350', '351-400', '40
df_train["totchol_range"] = pd.cut(df_train["totChol"], bins = chol_bins, labels = chol
```

In [130…
```
# Create bin for total cholesterol feature for test dataset
chol_bins = [100, 151, 201, 251, 301, 351, 401, 405]
chol_intervals = ['100-150', '151-200', '201-250', '251-300', '301-350', '351-400', '40
df_test["totchol_range"] = pd.cut(df_test["totChol"], bins = chol_bins, labels = chol_i
```

BMI

In [131…
```
# Create bin for bmi feature for train dataset
bmi_bins = [0, 22, 27, 30, 35]
bmi_intervals = ['0-20', '21-25', '26-30', '30+']
df_train["bmi_range"] = pd.cut(df_train["BMI"], bins = bmi_bins, labels = bmi_intervals
```

In [132…
```
# Create bin for bmi feature for test dataset
bmi_bins = [0, 22, 27, 30, 35]
bmi_intervals = ['0-20', '21-25', '26-30', '30+']
df_test["bmi_range"] = pd.cut(df_test["BMI"], bins = bmi_bins, labels = bmi_intervals)
```

Glucose

In [133…
```
# Create bin for glucose feature for train dataset
glu_bins = [42, 57, 67, 77, 101, 102]
glu_intervals = ['40-55', '56-65', '66-75', '76-100', '101+']
df_train["glu_range"] = pd.cut(df_train["glucose"], bins = glu_bins, labels = glu_inter
```

In [134…
```
# Create bin for glucose feature for test dataset
glu_bins = [42, 57, 67, 77, 101, 102]
glu_intervals = ['40-55', '56-65', '66-75', '76-100', '101+']
df_test["glu_range"] = pd.cut(df_test["glucose"], bins = glu_bins, labels = glu_interva
```

In [135…
```
df_test['heartRate'].max()
```

Out[135…
```
140.0
```

Heart Rate

In [136…
```
# Create bin for heart rate feature for train dataset
HR_bins = [45, 65, 75, 85, 95, 105, 140]
hr_intervals = ['45-60', '61-70', '71-80', '81-90', '91-100', '101+']
df_train["hr_range"] = pd.cut(df_train['heartRate'], bins = HR_bins, labels = dia_inter
```

In [137…
```
# Create bin for heart rate feature for test dataset
HR_bins = [45, 65, 75, 85, 95, 105, 140]
hr_intervals = ['45-60', '61-70', '71-80', '81-90', '91-100', '101+']
df_test["hr_range"] = pd.cut(df_test['heartRate'], bins = HR_bins, labels = dia_interva
```

Diastolic Blood Pressure

In [138…
```
# Create bin for diastolic BP feature for train dataset
dia_bins = [55, 65, 75, 85, 95, 105, 150]
dia_intervals = ['48-60', '61-70', '71-80', '81-90', '91-100', '101+']
df_train["diaBP_range"] = pd.cut(df_train["diaBP"], bins = dia_bins, labels = dia_inter
```

In [139…
```
# Create bin for diastolic BP feature for test dataset
dia_bins = [55, 65, 75, 85, 95, 105, 150]
dia_intervals = ['48-60', '61-70', '71-80', '81-90', '91-100', '101+']
df_test["diaBP_range"] = pd.cut(df_test["diaBP"], bins = dia_bins, labels = dia_interva
```

In [140…
```
ar = df_train.groupby(["age_range"], as_index = False).agg({'TenYearCHD':'mean'})
ma2 = df_train.groupby(['age_range'], as_index = False).agg({'TenYearCHD':'size'})
mat = ma2.rename({'TenYearCHD': 'count'}, axis=1)
am = pd.merge(ar, mat, on = 'age_range')
am
```

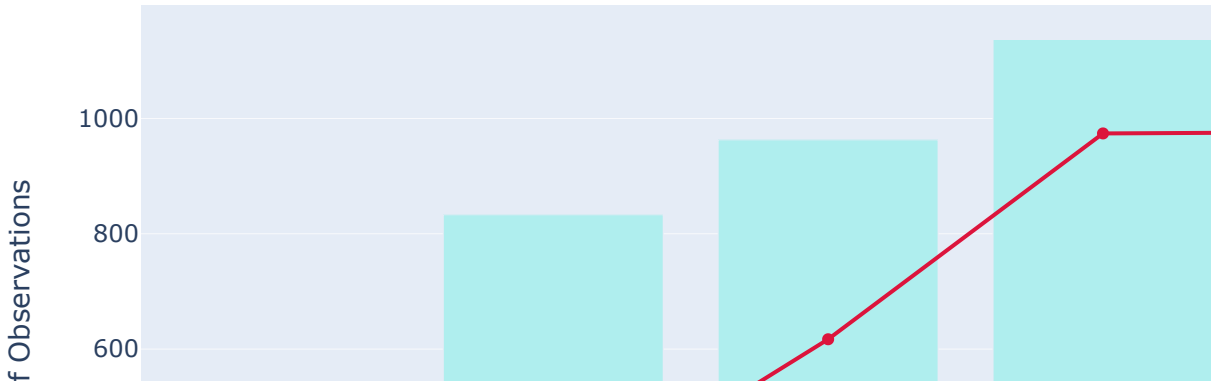
Out[140…

	age_range	TenYearCHD	count
0	32-36	0.107317	205
1	37-41	0.237695	833
2	42-46	0.393562	963
3	47-51	0.581354	1137
4	52-56	0.582692	1040
5	57-61	0.660218	827
6	62+	0.655957	747

Relationship between Actual Age Range and Average Target Variable - For Train Dataset

In [141…

```
x = am['age_range']
y1 = am['count']
y2 = am['TenYearCHD']
#y3 = am['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Age Range vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Group Age Range and TenYearCHD

In [142…

```
ar = df_train.groupby(["age_range"], as_index = False).agg({'TenYearCHD':'mean'})
ma2 = df_train.groupby(['age_range'], as_index = False).agg({'TenYearCHD':'size'})
mat = ma2.rename({'TenYearCHD': 'count'}, axis=1)
am = pd.merge(ar, mat, on = 'age_range')
am
```

Out[142…

	age_range	TenYearCHD	count
0	32-36	0.107317	205
1	37-41	0.237695	833
2	42-46	0.393562	963
3	47-51	0.581354	1137
4	52-56	0.582692	1040
5	57-61	0.660218	827
6	62+	0.655957	747

Group Cigarettes Per Day Range and TenYearCHD

In [143…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
cpd = df_train.groupby(["cig_per_day_range"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cp = df_train.groupby(["cig_per_day_range"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cp = cp.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
pc = pd.merge(cpd, cp, on="cig_per_day_range")
pc
```

Out[143…

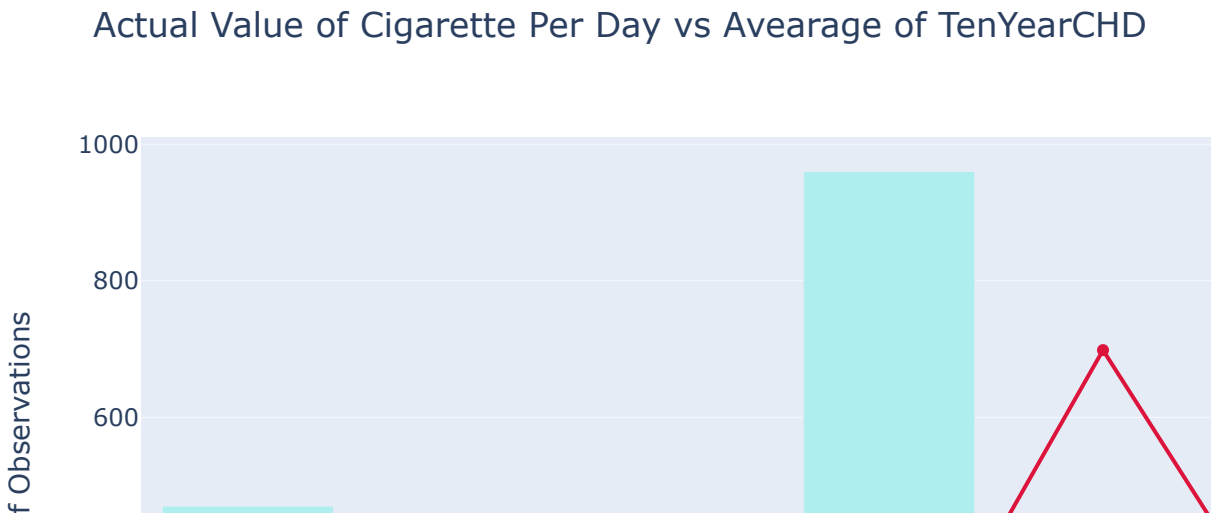
	cig_per_day_range	TenYearCHD	count
0	0-5	0.580851	470
1	6-10	0.437018	389
2	11-15	0.563889	360
3	16-20	0.485417	960
4	21-25	0.789773	176
5	26-30	0.518395	299
6	30-35	0.808989	89
7	36-40	0.928571	14
8	40+	0.543689	103

Relationship between Average Cigarette Per Day Range and Target Variable - For Train Dataset

In [144…

```
x = pc['cig_per_day_range']
y1 = pc['count']
y2 = pc['TenYearCHD']
#y3 = am['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
```

```
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Cigarette Per Day vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Group Total Cholesterol Range and TenYearCHD

In [145…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
tr = df_train.groupby(["totchol_range"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
tcr = df_train.groupby(["totchol_range"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
tcr = tcr.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
st = pd.merge(tr, tcr, on="totchol_range")
st
```

Out[145…

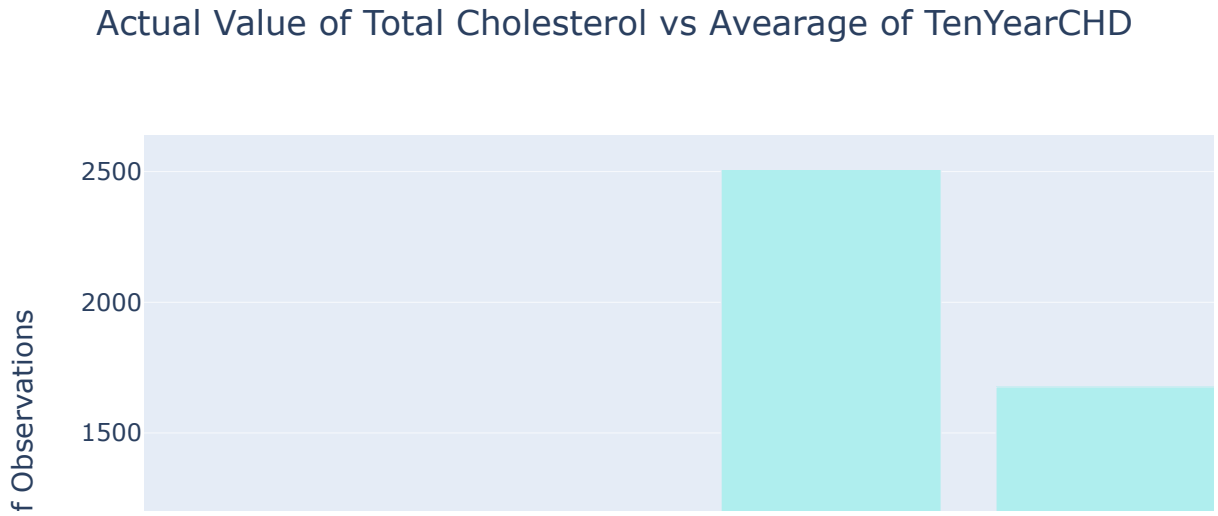
	totchol_range	TenYearCHD	count
0	100-150	0.523810	42
1	151-200	0.380157	1018
2	201-250	0.494817	2508
3	251-300	0.558473	1676
4	301-350	0.609700	433
5	351-400	0.588235	51
6	401+	1.000000	2

Relationship between Average Total Cholesterol Range and Target Variable - For Train Dataset

In [146…

```
x = st['totchol_range']
y1 = st['count']
y2 = st['TenYearCHD']
#y3 = am['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)
```

```
fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Total Cholesterol vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



In [147…
```
df_train.columns
```

Out[147…
```
Index(['sex', 'age', 'education', 'currentSmoker', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
       'diaBP', 'BMI', 'heartRate', 'glucose', 'TenYearCHD', 'age_range',
       'cig_per_day_range', 'totchol_range', 'bmi_range', 'glu_range',
       'hr_range', 'diaBP_range'],
      dtype='object')
```

In [148…
```
df_train.education = df_train.education.astype(int)
```

Group Education Range and Predicted TenYearCHD

In [149…
```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
cd = df_train.groupby(["education"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
dc = df_train.groupby(["education"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
hr = dc.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
rh = pd.merge(cd, hr, on="education")
rh
```

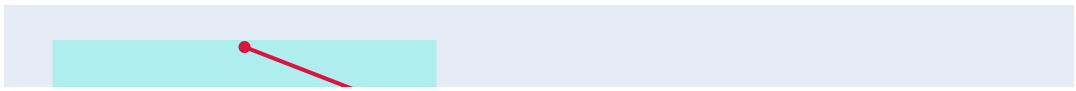
Out[149…

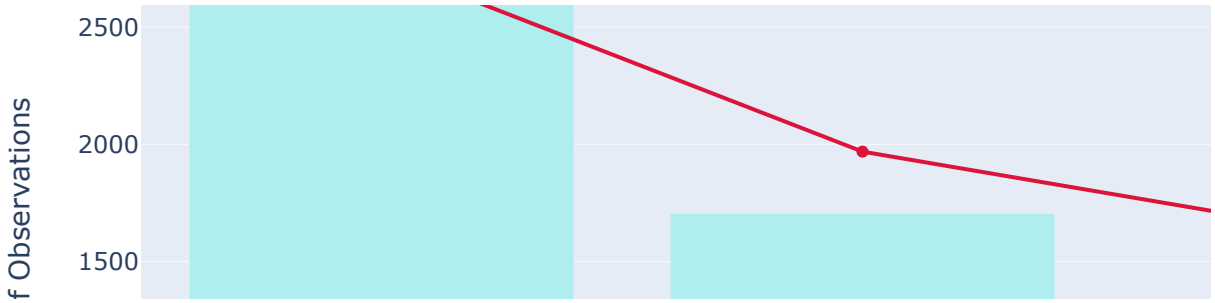
	education	TenYearCHD	count
0	1	0.592420	2797
1	2	0.473900	1705
2	3	0.422007	827
3	4	0.207547	424

Relationship between Education and Average Target Variable - For Train Dataset

In [150…
```
x = rh['education']
y1 = rh['count']
```

```
y2 = rh['TenYearCHD']
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Education vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```





Group BMI Range and Predicted TenYearCHD

In [151…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
bm = df_train.groupby(["bmi_range"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
bm2 = df_train.groupby(["bmi_range"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
mb = bm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
bi = pd.merge(bm, mb, on="bmi_range")
bi
```

Out[151…

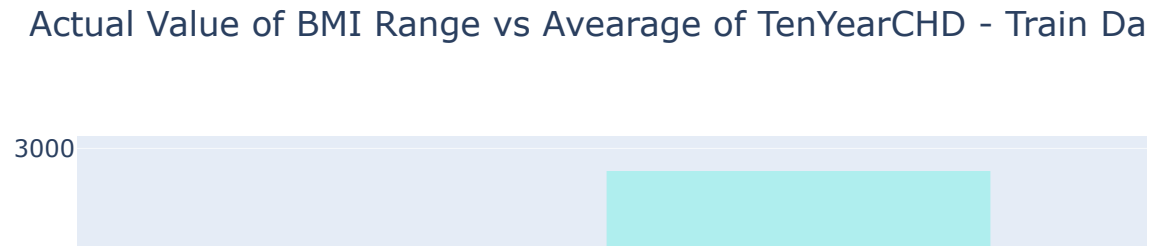
	bmi_range	TenYearCHD	count
0	0-20	0.378378	740
1	21-25	0.494142	2902
2	26-30	0.584548	1372
3	30+	0.529412	578

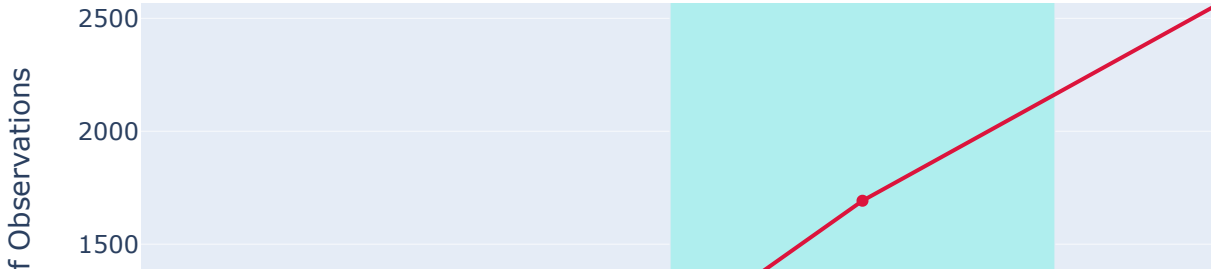
Relationship between BMI Range and Average Target Variable - For Train Dataset

In [152…

```
x = bi['bmi_range']
y1 = bi['count']
y2 = bi['TenYearCHD']
```

```
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of BMI Range vs Avearage of TenYearCHD - Train Dataset",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```





Group Current Smoker Range and Predicted TenYearCHD

In [153…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["currentSmoker"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["currentSmoker"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="currentSmoker")
cs
```

Out[153…

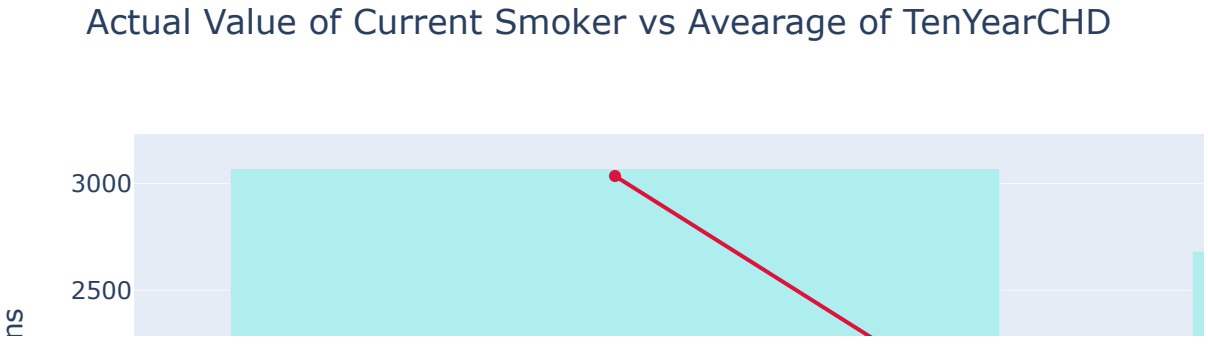
	currentSmoker	TenYearCHD	count
0	0	0.532248	3070
1	1	0.472605	2683

Relationship between Current Smoker and Average Target Variable - For Train Dataset

In [154…

```
x = cs['currentSmoker']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']
```

```
fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Current Smoker vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



In [155…

```
df_train["BPMeds"] = df_train["BPMeds"].astype(int)
```

Group Blood Pressure Medicines and Predicted TenYearCHD

In [156…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["BPMeds"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["BPMeds"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="BPMeds")
cs
```

Out[156…

	BPMeds	TenYearCHD	count
0	0	0.506385	5638
1	1	0.408696	115

Relationship between BPMeds and Average Target Variable - For Train Dataset

In [157…

```
x = cs['BPMeds']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']
```

```
fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of BPMeds vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```

Group Prevalent Stroke and Predicted TenYearCHD

In [158…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["prevalentStroke"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["prevalentStroke"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="prevalentStroke")
cs
```

Out[158…

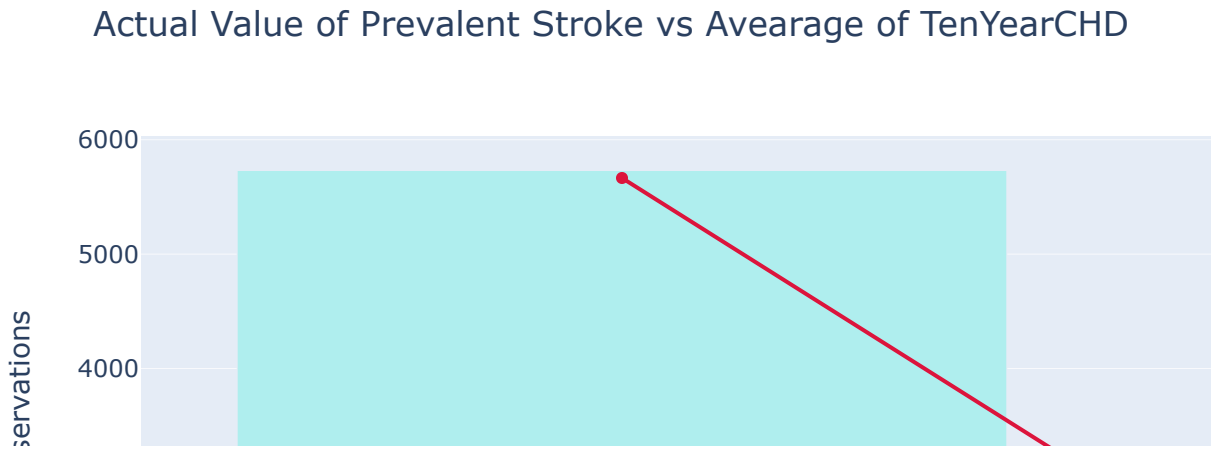
	prevalentStroke	TenYearCHD	count
0	0	0.504973	5731
1	1	0.363636	22

Relationship between Prevalent Stroke and Average Target Variable - For Train Dataset

In [159…

```
x = cs['prevalentStroke']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
```

```
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Prevalent Stroke vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Group Prevalent Hypertension and Predicted TenYearCHD

In [160…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["prevalentHyp"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["prevalentHyp"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="prevalentHyp")
cs
```

Out[160…

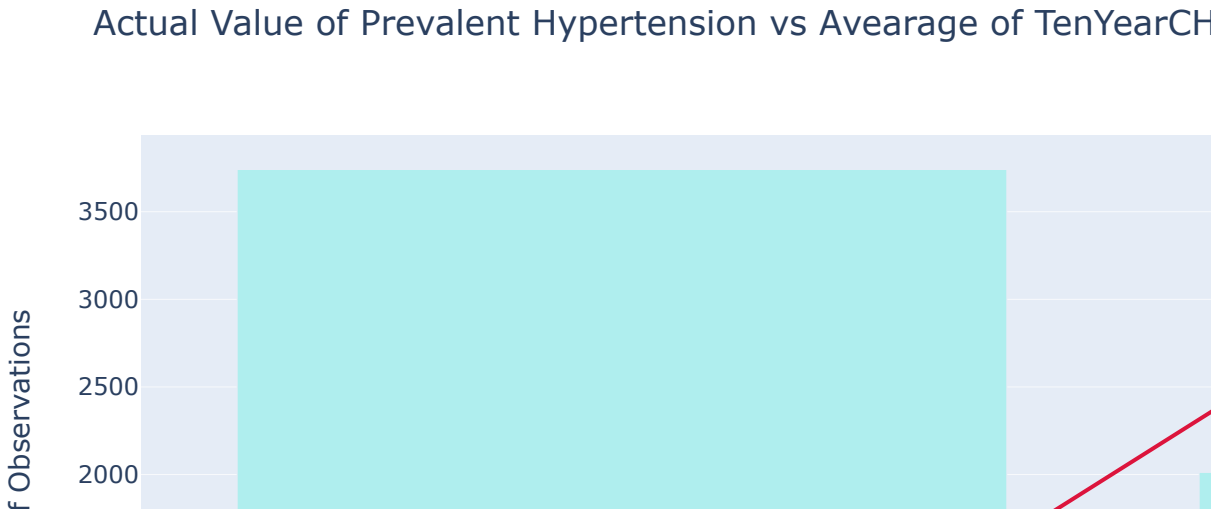
	prevalentHyp	TenYearCHD	count
0	0	0.443197	3741
1	1	0.618290	2012

Relationship between Prevalent Hypotension and Average Target Variable - For Train Dataset

In [161…

```
x = cs['prevalentHyp']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
```

```
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Prevalent Hypertension vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Group Glucose Range and Predicted TenYearCHD

In [162…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["glu_range"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["glu_range"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="glu_range")
cs
```

Out[162…

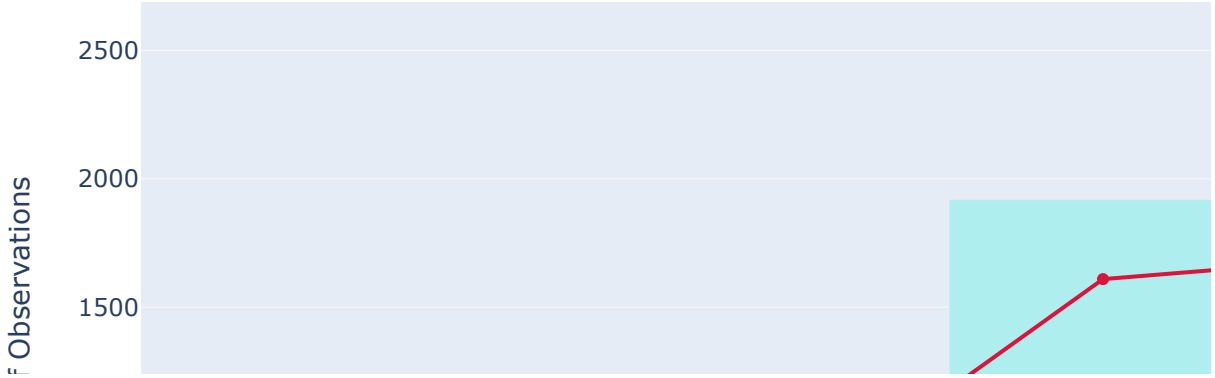
	glu_range	TenYearCHD	count
0	40-55	0.360656	61
1	56-65	0.396084	664
2	66-75	0.498697	1919
3	76-100	0.510372	2555
4	101+	0.586207	29

Relationship between Glucose Range and Average Target Variable - For Train Dataset

In [163…

```
x = cs['glu_range']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
```

```
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Glucose Range vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Group Diastolic BP and Predicted TenYearCHD

In [164…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["diaBP_range"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["diaBP_range"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="diaBP_range")
cs
```

Out[164…

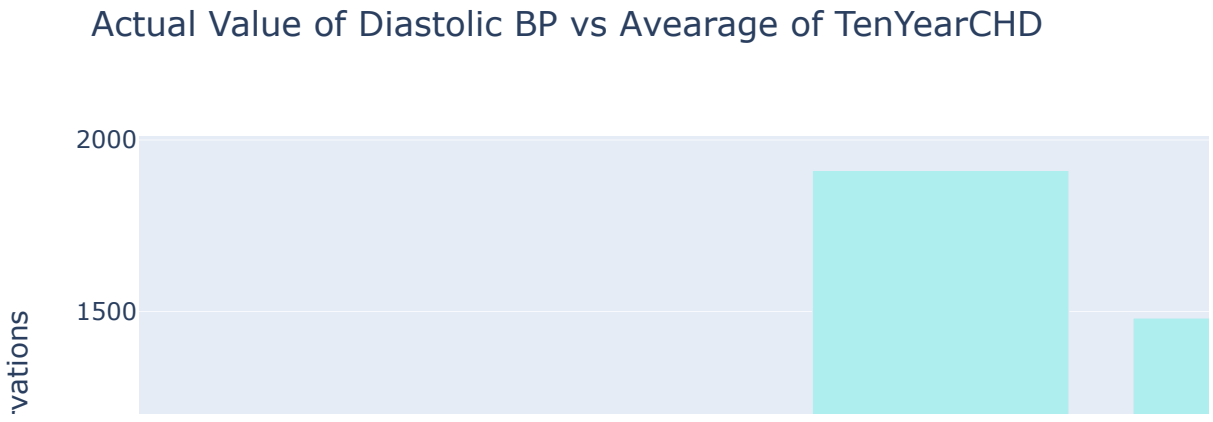
	diaBP_range	TenYearCHD	count
0	48-60	0.303191	188
1	61-70	0.377477	1110
2	71-80	0.455782	1911
3	81-90	0.562458	1481
4	91-100	0.648609	683
5	101+	0.735135	370

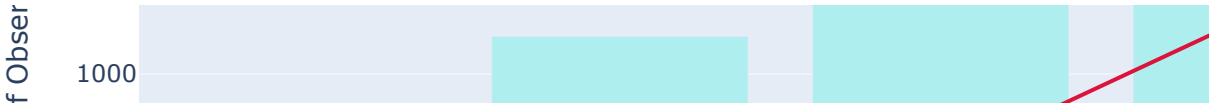
Relationship between Diastolic BP Range and Average Target Variable - For Train Dataset

In [165…

```
x = cs['diaBP_range']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
```

```
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Diastolic BP vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```





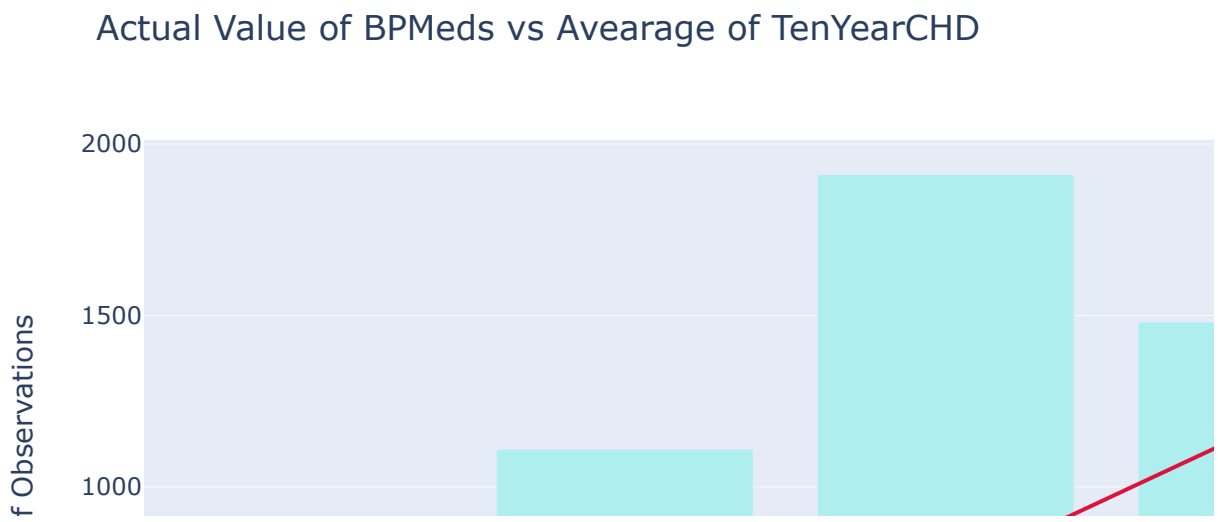
Relationship between DiaBP and Average Target Variable - For Train Dataset

Group Diabetes and Predicted TenYearCHD

In [166…

```
x = cs['diaBP_range']
y1 = cs['count']
y2 = cs['TenYearCHD']

fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.update_layout(
    title_text = "Actual Value of BPMeds vs Avearage of TenYearCHD",
    #title_text="Diastolic Blood Pressure",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
```

```
        side="right",
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



In [167…

```
# Group by "cig_per_day_range" and calculate the mean of "TenYearCHD"
csm = df_train.groupby(["diabetes"], as_index=False)["TenYearCHD"].mean()

# Group by "cig_per_day_range" and calculate the count of records in each group
cm2 = df_train.groupby(["diabetes"], as_index=False)["TenYearCHD"].size()

# Rename the "TenYearCHD" column to "count"
cm = cm2.rename(columns={'size': 'count'})

# Merge the two DataFrames on "cig_per_day_range"
cs = pd.merge(csm, cm, on="diabetes")
cs
```

Out[167…

	diabetes	TenYearCHD	count
0	0	0.497578	5573

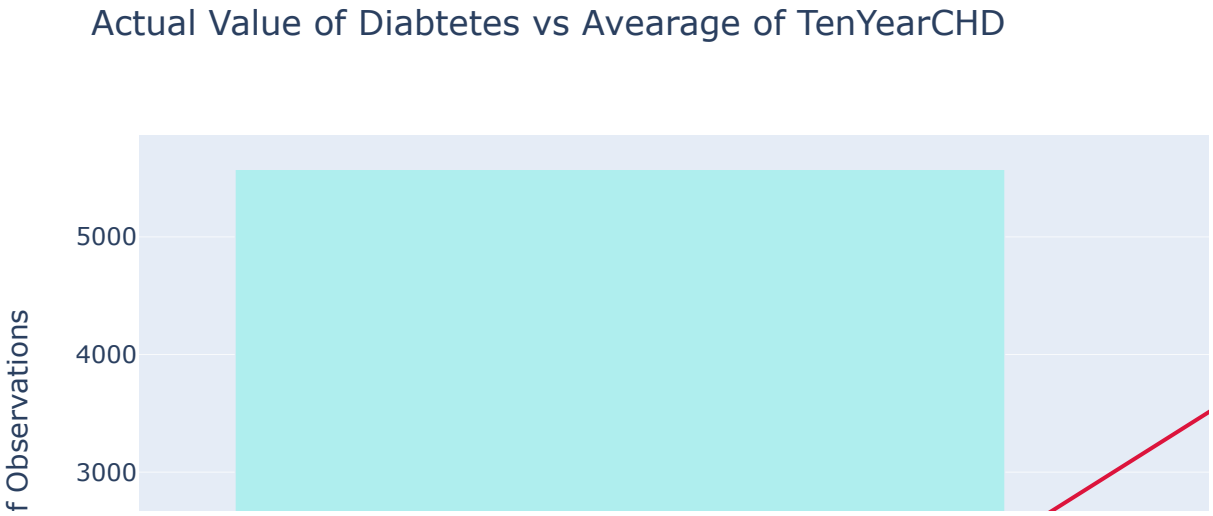
	diabetes	TenYearCHD	count
1	1	0.716667	180

Relationship between Diabetes and Average Target Variable - For Train Dataset

In [168…

```
x = cs['diabetes']
y1 = cs['count']
y2 = cs['TenYearCHD']
#y3 = bi['predicted_y']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

# fig.add_trace(
#     go.Scatter(
#         x=x,
#         y=y3,
#         yaxis="y2",
#         name="predicted value",
#         marker=dict(color="blue"),
#     )
# )

fig.update_layout(
    title_text = "Actual Value of Diabtetes vs Avearage of TenYearCHD",
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
```

```
)

fig.show()
```



Building Model on the train dataset and testing with the test dataset

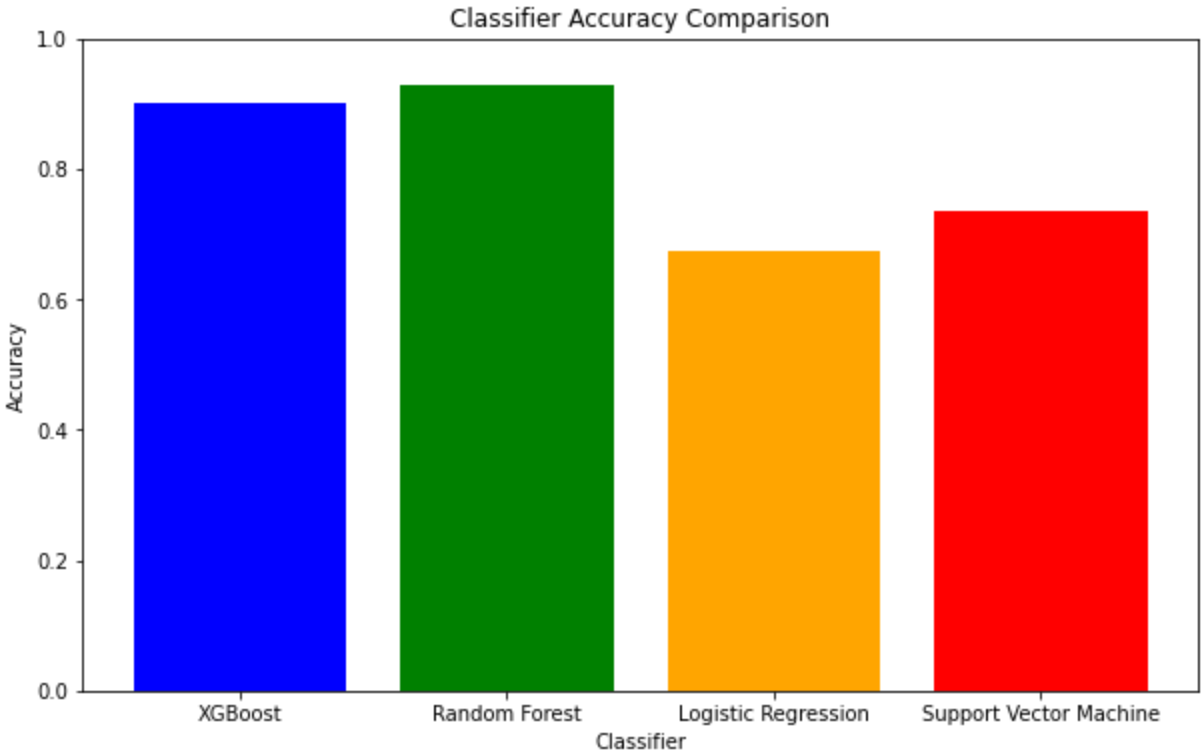
In [170…

```
X = df_train.drop(columns = 'TenYearCHD')
y = df_train['TenYearCHD']

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Initialize lists to store accuracy scores and classifier names
accuracy_scores = []
```

```
classifier_names = []

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train1)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate accuracy and append to the lists
    accuracy = np.mean(y_pred == y_test1)
    accuracy_scores.append(accuracy)
    classifier_names.append(clf_name)

    # Print the classification report
    report = classification_report(y_test1, y_pred)
    print(report)
    print("\n")

# Visualize the accuracy scores
plt.figure(figsize=(10, 6))
plt.bar(classifier_names, accuracy_scores, color=['blue', 'green', 'orange', 'red'])
plt.xlabel('Classifier')
plt.ylabel('Accuracy')
plt.title('Classifier Accuracy Comparison')
plt.ylim(0, 1)
plt.show()
```

```
Classifier: XGBoost
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       745
           1       0.92      0.87      0.89       694

    accuracy                           0.90      1439
   macro avg       0.90      0.90      0.90      1439
weighted avg       0.90      0.90      0.90      1439


Classifier: Random Forest
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       745
           1       0.93      0.93      0.93       694

    accuracy                           0.93      1439
   macro avg       0.93      0.93      0.93      1439
weighted avg       0.93      0.93      0.93      1439


Classifier: Logistic Regression
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       745
           1       0.66      0.68      0.67       694

    accuracy                           0.67      1439
```

```
   macro avg       0.67      0.68      0.67      1439
weighted avg       0.68      0.67      0.67      1439


Classifier: Support Vector Machine
              precision    recall  f1-score   support

           0       0.77      0.70      0.73       745
           1       0.71      0.77      0.74       694

    accuracy                           0.73      1439
   macro avg       0.74      0.74      0.73      1439
weighted avg       0.74      0.73      0.73      1439
```



Cross Validation Score, MAE, MSE, R2_Score

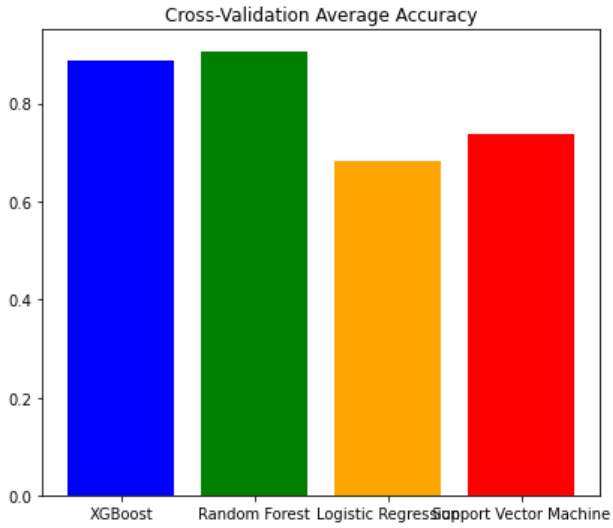
In [173…

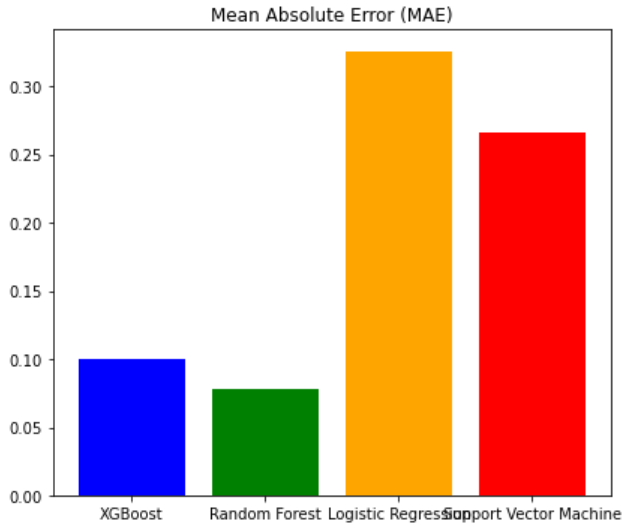
```
# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

# Initialize lists to store metric values and classifier names
average_scores = []
mae_values = []
```

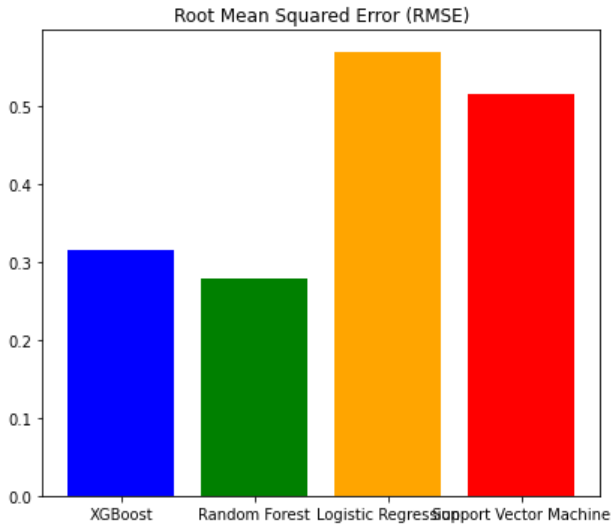
```
rmse_values = []
r2_values = []
classifier_names = []

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")

    # Train the model on the training data
    clf.fit(X_train, y_train1)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate cross-validated scores
    cv_scores = cross_val_score(clf, X_train, y_train1, cv=5, scoring='accuracy')
    average_score = np.mean(cv_scores)

    # Calculate mean absolute error (MAE)
    mae = mean_absolute_error(y_test1, y_pred)

    # Calculate root mean squared error (RMSE)
    rmse = np.sqrt(mean_squared_error(y_test1, y_pred))

    # Calculate R^2 score
    r2 = r2_score(y_test1, y_pred)

    # Append metric values and classifier name to lists
    average_scores.append(average_score)
    mae_values.append(mae)
    rmse_values.append(rmse)
    r2_values.append(r2)
    classifier_names.append(clf_name)

    print(f"Cross-Validation Average Accuracy: {average_score:.4f}")
    print(f"Mean Absolute Error (MAE): {mae:.4f}")
    print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
    print(f"R^2 Score: {r2:.4f}")
    print("\n")

# Visualize the metrics
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Bar chart for Cross-Validation Accuracy
axes[0, 0].bar(classifier_names, average_scores, color=['blue', 'green', 'orange', 'red
axes[0, 0].set_title('Cross-Validation Average Accuracy')

# Bar chart for Mean Absolute Error (MAE)
axes[0, 1].bar(classifier_names, mae_values, color=['blue', 'green', 'orange', 'red'])
axes[0, 1].set_title('Mean Absolute Error (MAE)')

# Bar chart for Root Mean Squared Error (RMSE)
axes[1, 0].bar(classifier_names, rmse_values, color=['blue', 'green', 'orange', 'red'])
axes[1, 0].set_title('Root Mean Squared Error (RMSE)')

# Bar chart for R^2 Score
axes[1, 1].bar(classifier_names, r2_values, color=['blue', 'green', 'orange', 'red'])
axes[1, 1].set_title('R^2 Score')
```

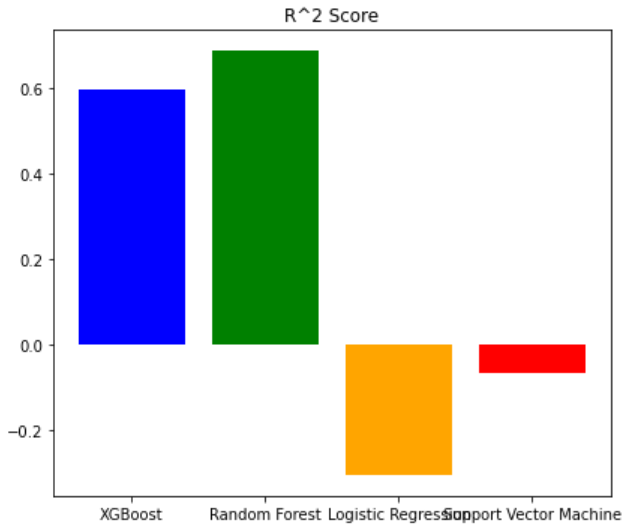
```
plt.tight_layout()
plt.show()
```

```
Classifier: XGBoost
Cross-Validation Average Accuracy: 0.8891
Mean Absolute Error (MAE): 0.1001
Root Mean Squared Error (RMSE): 0.3163
R^2 Score: 0.5992


Classifier: Random Forest
Cross-Validation Average Accuracy: 0.9054
Mean Absolute Error (MAE): 0.0778
Root Mean Squared Error (RMSE): 0.2790
R^2 Score: 0.6883


Classifier: Logistic Regression
Cross-Validation Average Accuracy: 0.6838
Mean Absolute Error (MAE): 0.3252
Root Mean Squared Error (RMSE): 0.5703
R^2 Score: -0.3025


Classifier: Support Vector Machine
Cross-Validation Average Accuracy: 0.7386
Mean Absolute Error (MAE): 0.2662
Root Mean Squared Error (RMSE): 0.5159
R^2 Score: -0.0660
```









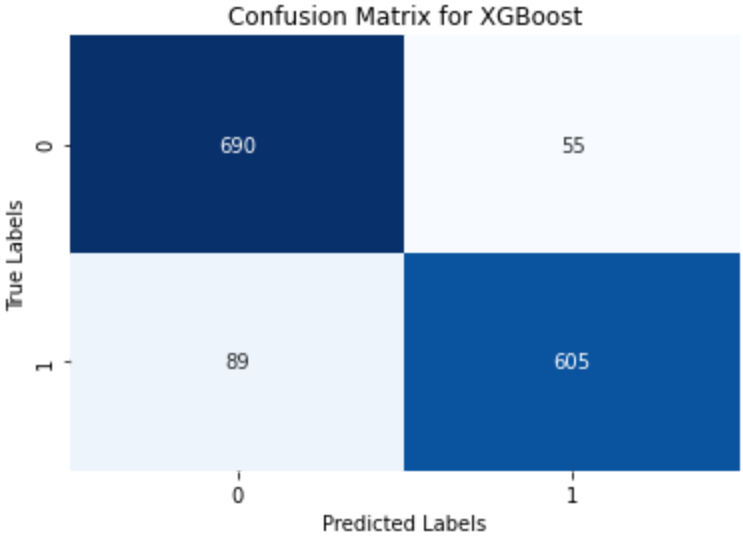
Confusion Matrix

In [174…

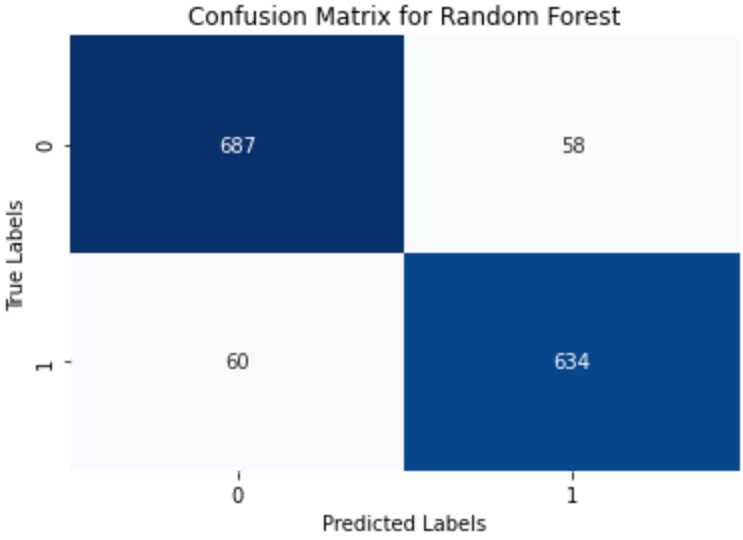
```
X = df_train.drop(columns = 'TenYearCHD')
y = df_train['TenYearCHD']

# Feature scaling (if needed)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train1)
X_test = scaler.transform(X_test1)

# Define a list of classifiers
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC())
]

#classifier.fit(X, y)

# Iterate through the classifiers
for clf_name, clf in classifiers:
    print(f"Classifier: {clf_name}")
```

```
    # Train the model on the training data
    clf.fit(X_train, y_train1)

    # Make predictions on the testing data
    y_pred = clf.predict(X_test)

    # Calculate the confusion matrix
    conf_matrix = confusion_matrix(y_test1, y_pred)

    # Create a heatmap of the confusion matrix
    plt.figure(figsize=(6, 4))  # Adjust the figure size as needed
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title(f'Confusion Matrix for {clf_name}')
    plt.show()
    print("\n")
```

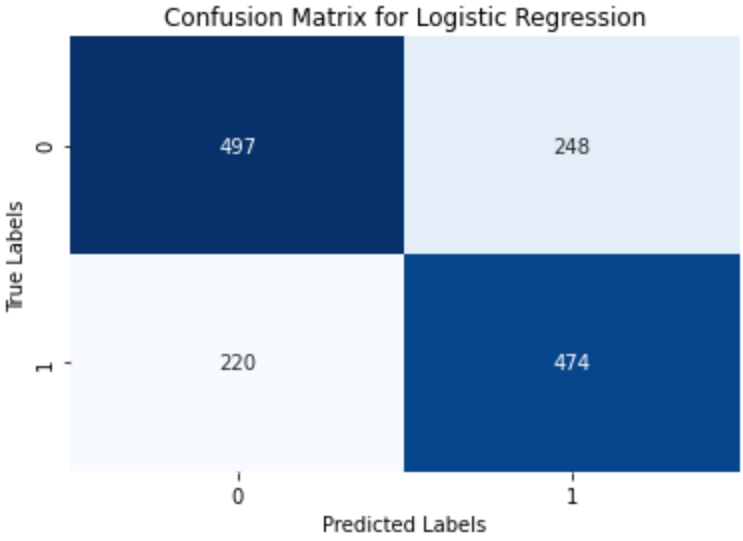
```
Classifier: XGBoost
```



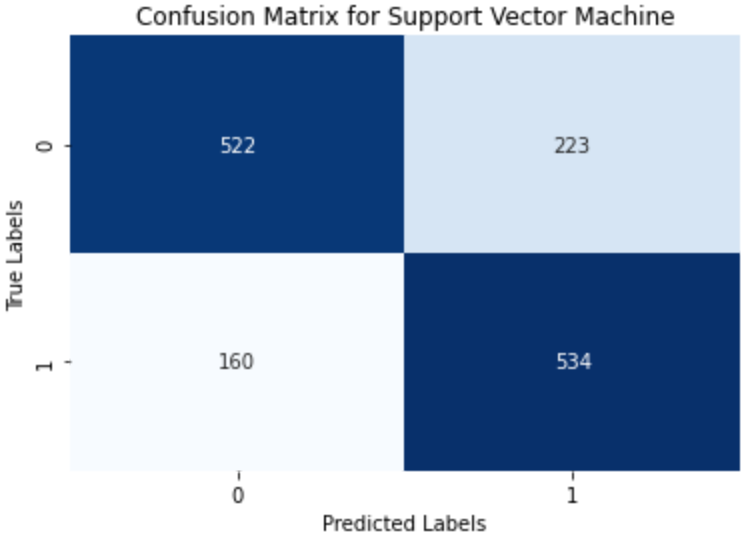

```
Classifier: Random Forest
```




```
Classifier: Logistic Regression
```



Classifier: Support Vector Machine

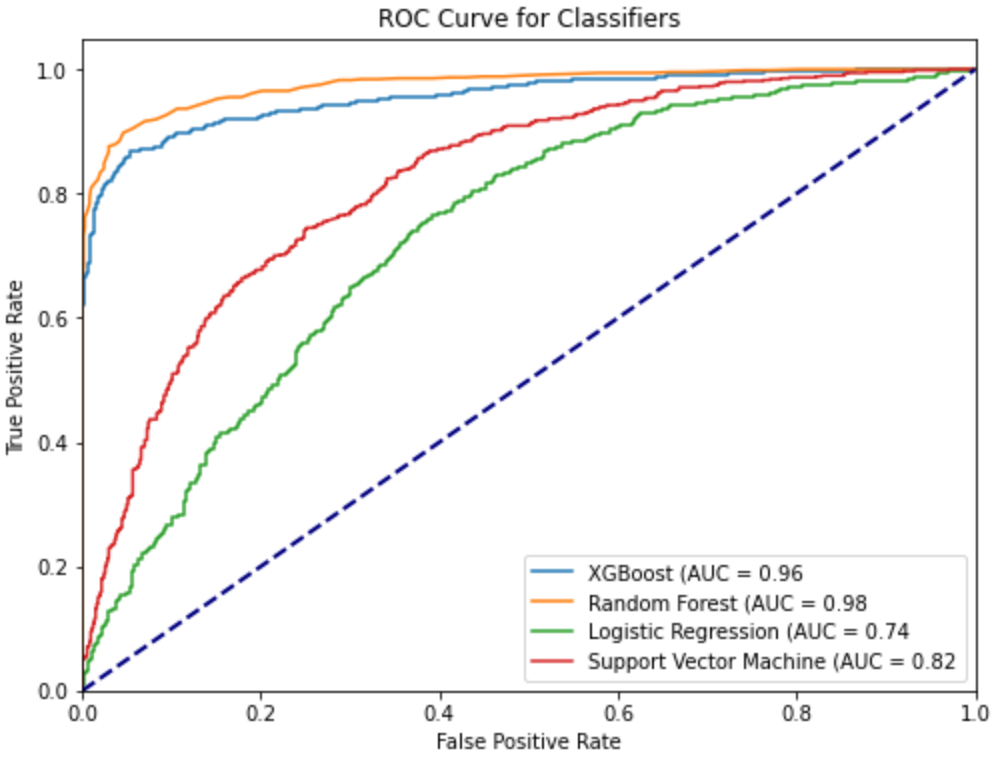


ROC Curve

In [175…

```
# Create a list of classifiers and their corresponding names
classifiers = [
    ("XGBoost", XGBClassifier()),
    ("Random Forest", RandomForestClassifier()),
    ("Logistic Regression", LogisticRegression()),
    ("Support Vector Machine", SVC(probability=True))  # Enable probability estimation
]

# Create an empty dictionary to store predicted probabilities
y_probs = {}

# Train and predict for each classifier and store the predicted probabilities
for clf_name, clf in classifiers:
    clf.fit(X_train, y_train1)
    y_probs[clf_name] = clf.predict_proba(X_test)[:, 1]

# Plot ROC curves for each classifier
plt.figure(figsize=(8, 6))
```

```
for clf_name, y_prob in y_probs.items():
    fpr, tpr, thresholds = roc_curve(y_test1, y_prob)
    auc = roc_auc_score(y_test1, y_prob)

    plt.plot(fpr, tpr, label=f'{clf_name} (AUC = {auc:.2f}')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
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 for Classifiers')
plt.legend(loc="lower right")
plt.show()
```



Model for Predicting the Target Variable

In [177…

```
# Split the data into features (X) and target (y)
X = dj.drop(columns=['TenYearCHD'])
y = dj['TenYearCHD']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Create and train the Logistic Regression classifier
model = LogisticRegression(solver='liblinear', random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Add predicted values to the original DataFrame
dj['pred_TenYearCHD'] = model.predict(X)
```

In [178…

```
df_test['pred_TenYearCHD'] = dj['pred_TenYearCHD']
```

In [179…

```
df_test['education'] = df_test['education'].astype(int)
```

Relationship between Education and Average Target Variable - For Test Dataset

In [180…

```
eg = df_test.groupby(['education'], as_index = False).agg({'TenYearCHD':'mean', 'pred_T
egt = df_test.groupby(['education'], as_index = False).agg({'TenYearCHD':'size'})
eg2 = egt.rename({'TenYearCHD': 'count'}, axis=1)
egy = pd.merge(eg, eg2, on = 'education')
egy
```

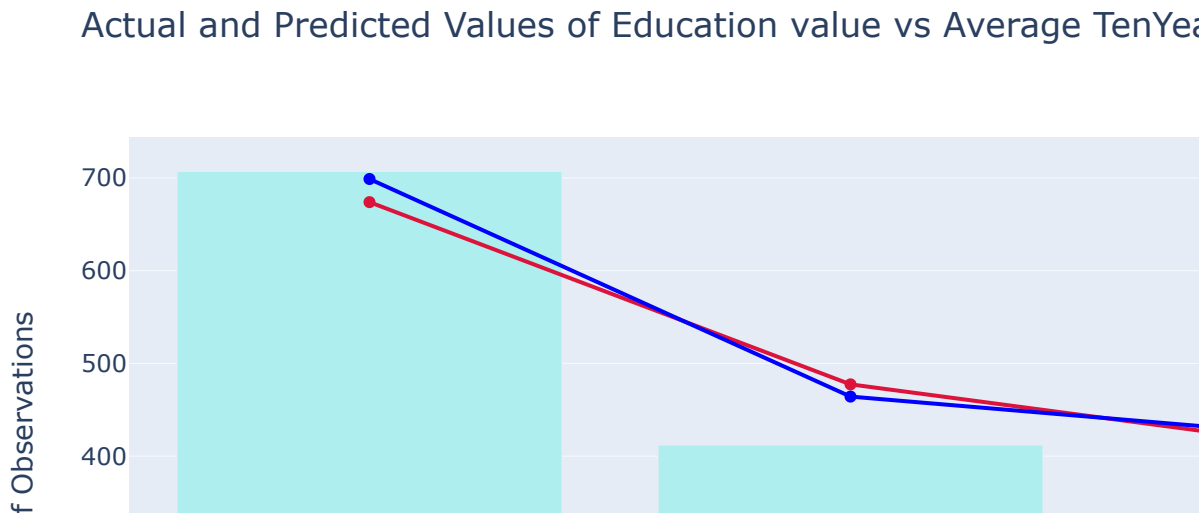
Out[180…

	education	TenYearCHD	pred_TenYearCHD	count
0	1	0.584158	0.602546	707
1	2	0.439320	0.429612	412
2	3	0.388646	0.397380	229
3	4	0.120879	0.274725	91

In [181…

```
x = egy['education']
y1 = egy['count']
y2 = egy['TenYearCHD']
y3 = egy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
```

```
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Education value vs Average TenYearCHD
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Age Range and Average Target Variable - For Test Dataset

In [182…

```
ag = df_test.groupby(['age_range'], as_index = False).agg({'TenYearCHD':'mean', 'pred_T
agt2 = df_test.groupby(['age_range'], as_index = False).agg({'TenYearCHD':'size'})
ag2 = agt2.rename({'TenYearCHD': 'count'}, axis=1)
agy = pd.merge(ag, ag2, on = 'age_range')
agy
```

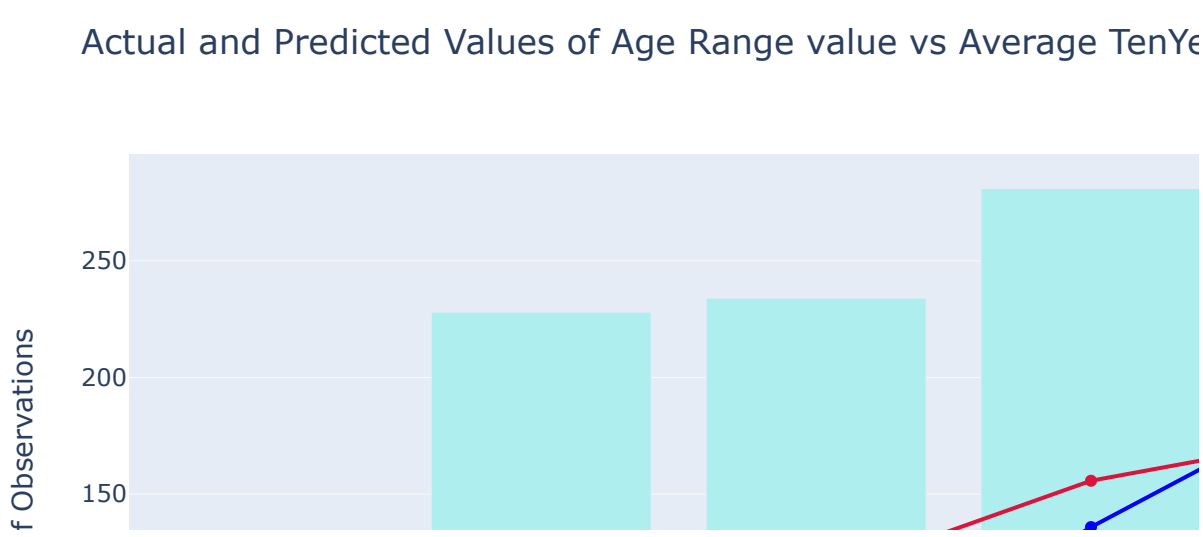
Out[182…

	age_range	TenYearCHD	pred_TenYearCHD	count
0	32-36	0.075472	0.075472	53
1	37-41	0.250000	0.092105	228
2	42-46	0.384615	0.222222	234
3	47-51	0.519573	0.455516	281
4	52-56	0.589552	0.656716	268
5	57-61	0.642458	0.888268	179
6	62+	0.632653	0.913265	196

In [183…

```
x = agy['age_range']
y1 = agy['count']
y2 = agy['TenYearCHD']
y3 = agy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
```

```
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Age Range value vs Average TenYearCHD
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Sex and Average Target Variable - For Test Dataset

In [184…

```
gtt = df_test.groupby(['sex'], as_index = False).agg({'TenYearCHD':'mean', 'pred_TenYea
gt2 = df_test.groupby(['sex'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'sex')
gy
```

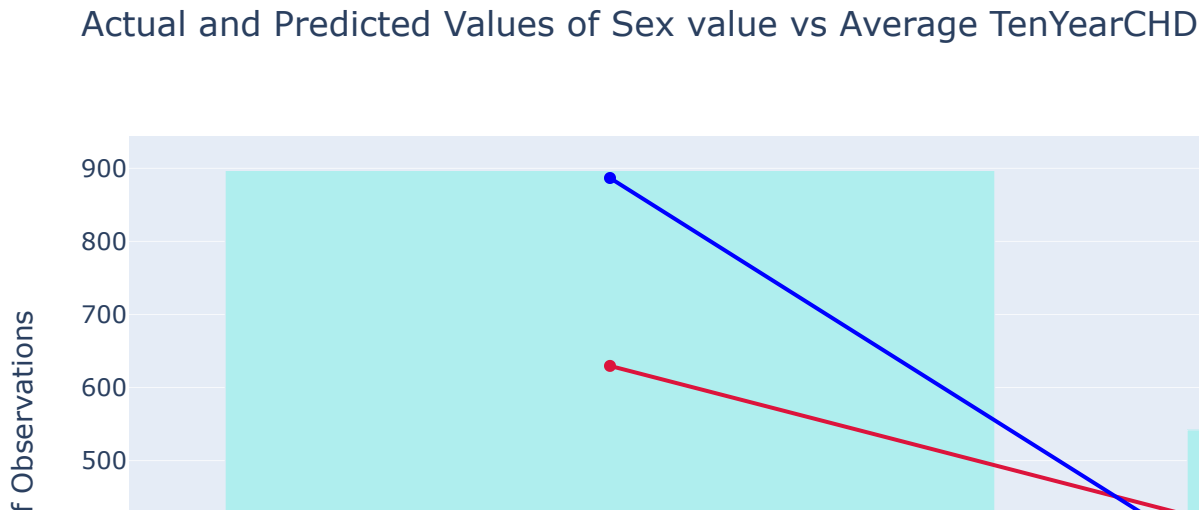
Out[184…

	sex	TenYearCHD	pred_TenYearCHD	count
0	0	0.512821	0.574136	897
1	1	0.431734	0.376384	542

In [185…

```
x = gy['sex']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
```

```
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Sex value vs Average TenYearCHD - Test
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between BMI Range and Average Target Variable - For Test Dataset

In [186…

```
gtt = df_test.groupby(['bmi_range'], as_index = False).agg({'TenYearCHD':'mean', 'pred_
gt2 = df_test.groupby(['bmi_range'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'bmi_range')
gy
```

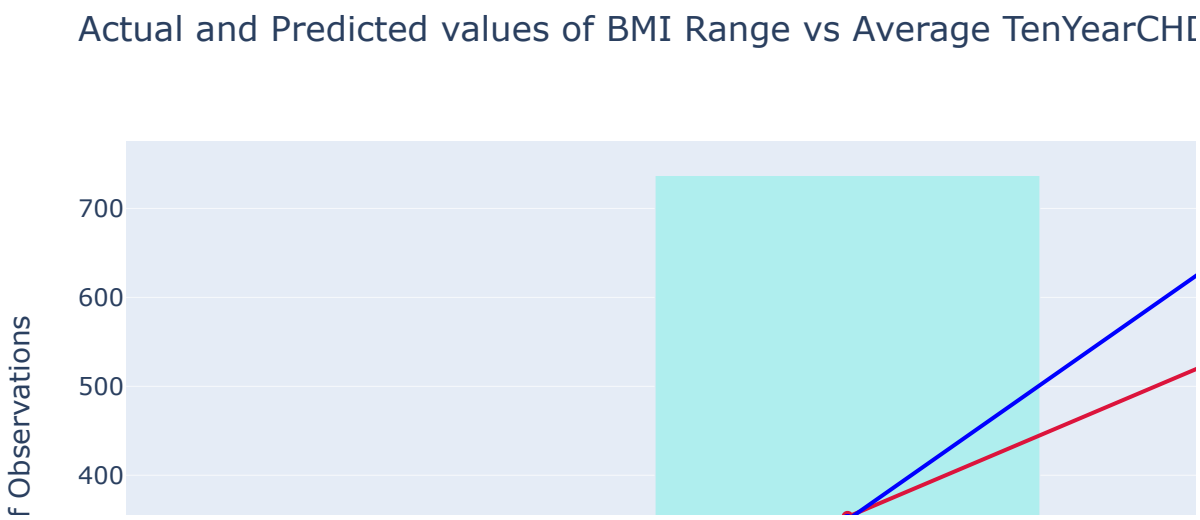
Out[186…

	bmi_range	TenYearCHD	pred_TenYearCHD	count
0	0-20	0.343750	0.348958	192
1	21-25	0.465400	0.464043	737
2	26-30	0.556575	0.614679	327
3	30+	0.565789	0.565789	152

In [187…

```
x = gy['bmi_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = "Actual and Predicted values of BMI Range vs Average TenYearCHD - Test
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
```

```
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Cigarettes Per Day and Average Target Variable - For Test Dataset

In [188…

```
df_test['cigsPerDay'] = df_test['cigsPerDay'].astype(int)
```

In [189…

```
gtt = df_test.groupby(['cig_per_day_range'], as_index = False).agg({'TenYearCHD':'mean'
gt2 = df_test.groupby(['cig_per_day_range'], as_index = False).agg({'TenYearCHD':'size'
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'cig_per_day_range')
gy
```

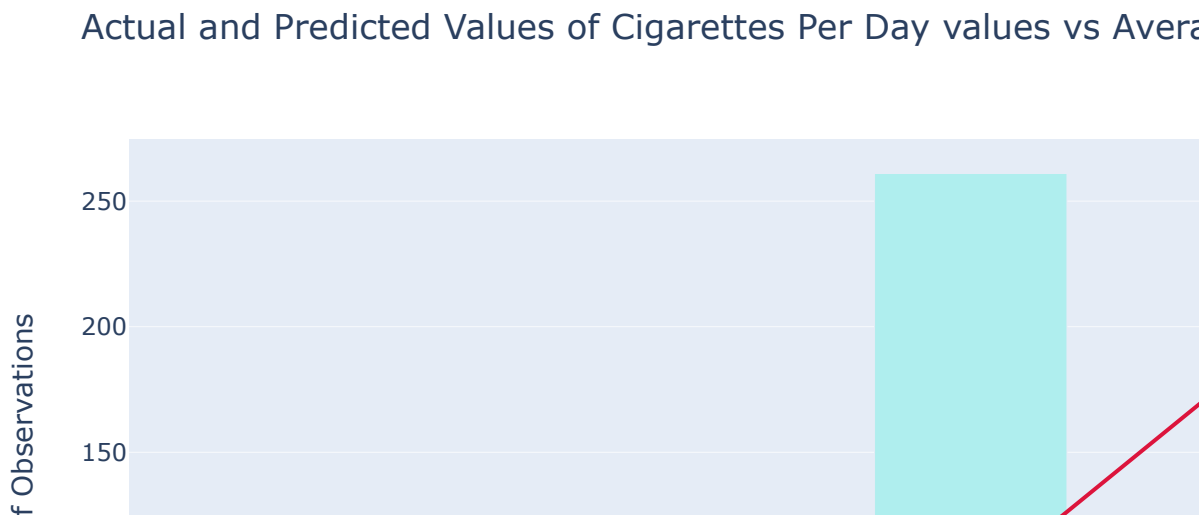
Out[189…

	cig_per_day_range	TenYearCHD	pred_TenYearCHD	count
0	0-5	0.556604	0.273585	106
1	6-10	0.490196	0.362745	102
2	11-15	0.575000	0.387500	80
3	16-20	0.509579	0.375479	261
4	21-25	0.744186	0.581395	43
5	26-30	0.379310	0.724138	58
6	30-35	1.000000	1.000000	3
7	36+	0.888889	0.666667	9

In [190…

```
x = gy['cig_per_day_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Cigarettes Per Day values vs Average T
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
```

```
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



In [191…

```
gtt = df_test.groupby(['bmi_range'], as_index = False).agg({'TenYearCHD':'mean', 'pred_
gt2 = df_test.groupby(['bmi_range'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'bmi_range')
gy
```

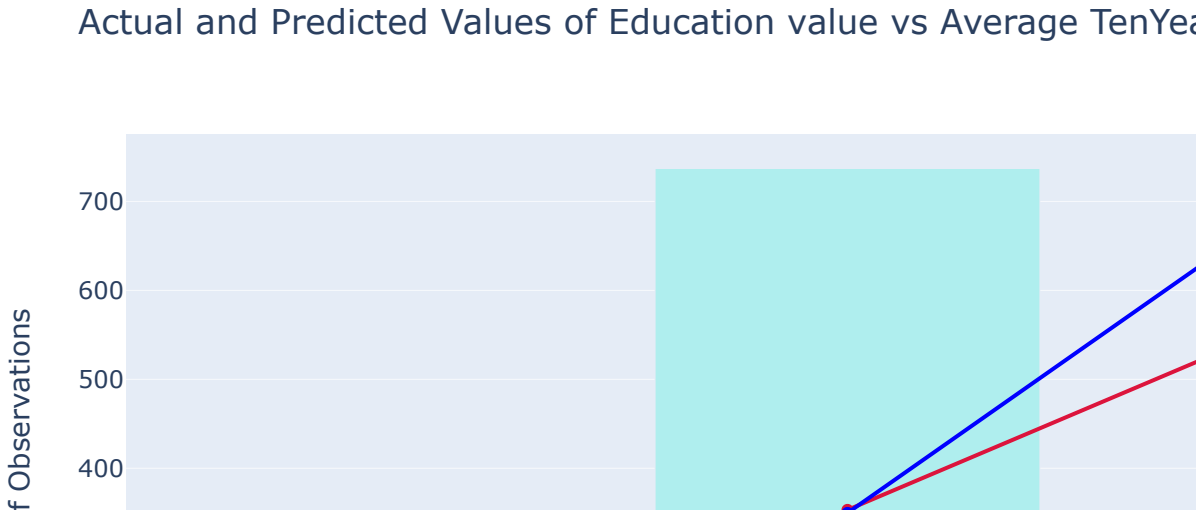
Out[191…

	bmi_range	TenYearCHD	pred_TenYearCHD	count
0	0-20	0.343750	0.348958	192
1	21-25	0.465400	0.464043	737
2	26-30	0.556575	0.614679	327

	bmi_range	TenYearCHD	pred_TenYearCHD	count
3	30+	0.565789	0.565789	152

In [192…

```
x = gy['bmi_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Education value vs Average TenYearCHD
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Total Cholesterol and Average Target Variable - For Test Dataset

In [193…

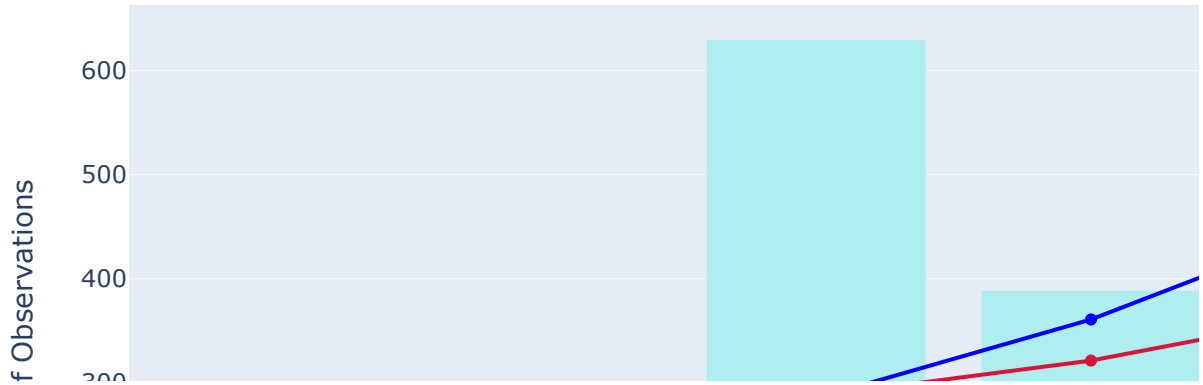
```
gtt = df_test.groupby(['totchol_range'], as_index = False).agg({'TenYearCHD':'mean', 'p
gt2 = df_test.groupby(['totchol_range'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'totchol_range')
gy
```

Out[193…

	totchol_range	TenYearCHD	pred_TenYearCHD	count
0	100-150	0.363636	0.090909	11
1	151-200	0.352713	0.286822	258
2	201-250	0.476190	0.473016	630
3	251-300	0.529563	0.591260	389
4	301-350	0.608333	0.750000	120
5	351-400	0.608696	0.869565	23
6	401+	1.000000	1.000000	1

In [194…

```
x = gy['totchol_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Total CHolesterol values vs Average Te
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Glucose Range and Average Target Variable - For Test Dataset

In [195…

```
gtt = df_test.groupby(['glu_range'], as_index = False).agg({'TenYearCHD':'mean', 'pred_
gt2 = df_test.groupby(['glu_range'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'glu_range')
gy
```

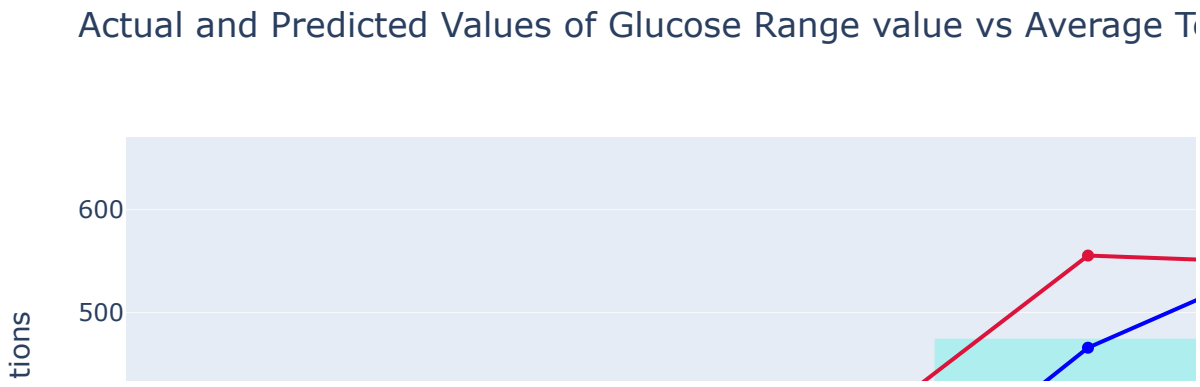
Out[195…

	glu_range	TenYearCHD	pred_TenYearCHD	count
0	40-55	0.230769	0.269231	26
1	56-65	0.348387	0.296774	155
2	66-75	0.496842	0.450526	475
3	76-100	0.489796	0.535322	637
4	101+	0.300000	0.300000	10

In [196…

```
x = gy['glu_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
```

```
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Glucose Range value vs Average TenYear
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Diastolic BP and Average Target Variable - For Test Dataset

In [197…

```
gtt = df_test.groupby(['diaBP_range'], as_index = False).agg({'TenYearCHD':'mean', 'pre
gt2 = df_test.groupby(['diaBP_range'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'diaBP_range')
gy
```

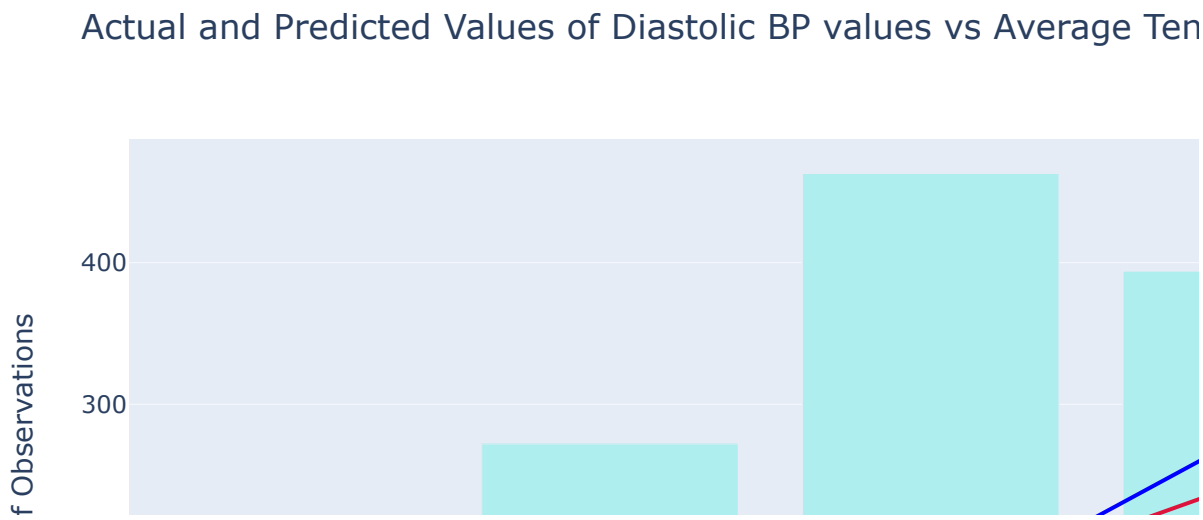
Out[197…

	diaBP_range	TenYearCHD	pred_TenYearCHD	count
0	48-60	0.409091	0.196970	66
1	61-70	0.382353	0.316176	272
2	71-80	0.412527	0.399568	463
3	81-90	0.540609	0.593909	394
4	91-100	0.620915	0.803922	153
5	101+	0.719101	0.876404	89

In [198…

```
x = gy['diaBP_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)
```

```
fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Diastolic BP values vs Average TenYear(
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Heart Rate Range and Average Target Variable - For Test Dataset

In [199…

```
df_test['heartRate'] = df_test['heartRate'].astype(int)
```

In [200…

```
gtt = df_test.groupby(['hr_range'], as_index = False).agg({'TenYearCHD':'mean', 'pred_T
gt2 = df_test.groupby(['hr_range'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'hr_range')
gy
```

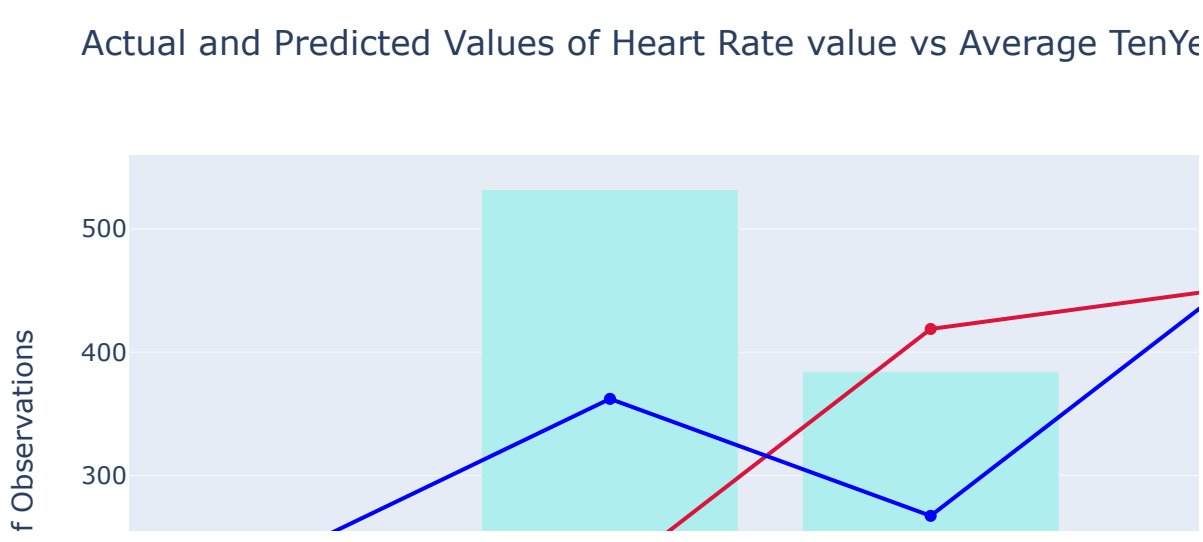
Out[200…

	hr_range	TenYearCHD	pred_TenYearCHD	count
0	48-60	0.391129	0.467742	248
1	61-70	0.460526	0.515038	532
2	71-80	0.536458	0.479167	384
3	81-90	0.549738	0.554974	191
4	91-100	0.456140	0.473684	57
5	101+	0.576923	0.461538	26

In [201…

```
x = gy['hr_range']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
```

```
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Heart Rate value vs Average TenYearCHD
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Prevalent Stroke and Average Target Variable - For Test Dataset

In [202…

```
gtt = df_test.groupby(['prevalentStroke'], as_index = False).agg({'TenYearCHD':'mean',
gt2 = df_test.groupby(['prevalentStroke'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'prevalentStroke')
gy
```

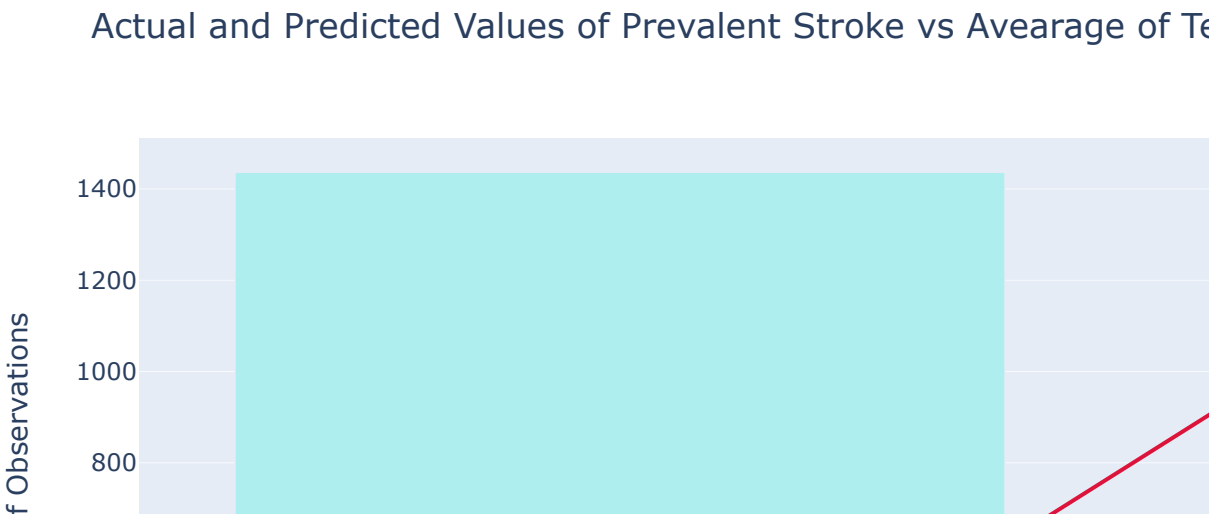
Out[202…

	prevalentStroke	TenYearCHD	pred_TenYearCHD	count
0	0	0.481198	0.499304	1436
1	1	1.000000	0.666667	3

In [203…

```
x = gy['prevalentStroke']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
```

```
    )
)

fig.update_layout(
    title_text = "Actual and Predicted Values of Prevalent Stroke vs Avearage of TenYea
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Prevalent Hypertension and Average Target Variable - For Test Dataset

In [204…

```
gtt = df_test.groupby(['prevalentHyp'], as_index = False).agg({'TenYearCHD':'mean', 'pr
gt2 = df_test.groupby(['prevalentHyp'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'prevalentHyp')
gy
```

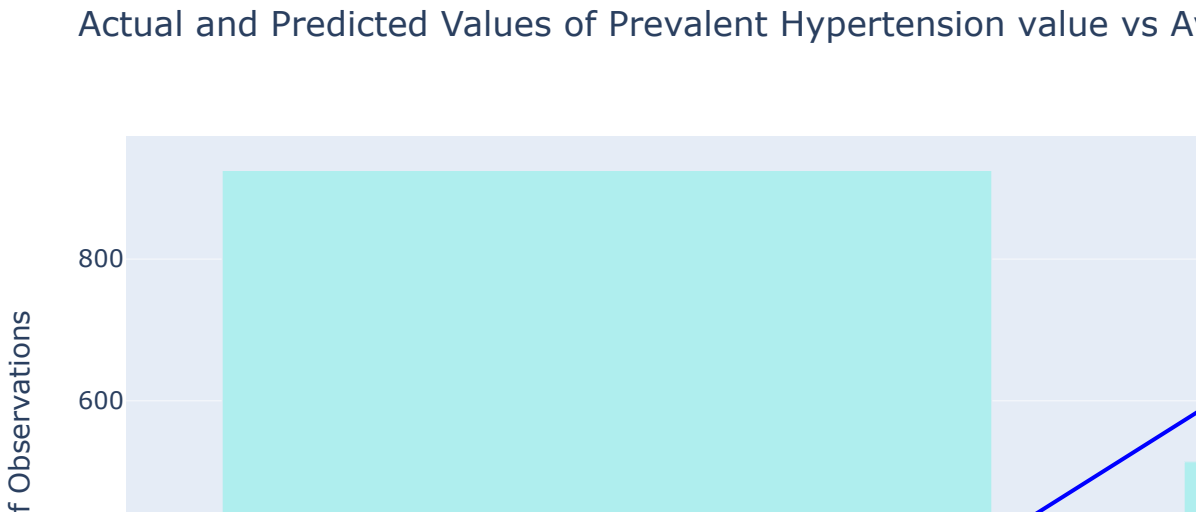
Out[204…

	prevalentHyp	TenYearCHD	pred_TenYearCHD	count
0	0	0.436757	0.363243	925
1	1	0.564202	0.745136	514

In [205…

```
x = gy['prevalentHyp']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Prevalent Hypertension value vs Averag
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
```

```
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Current Smoker and Average Target Variable - For Test Dataset

In [206…

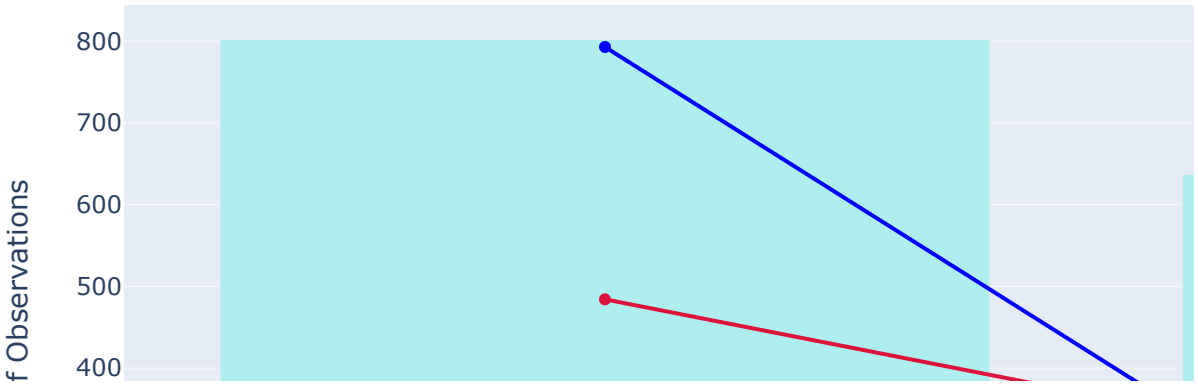
```
gtt = df_test.groupby(['currentSmoker'], as_index = False).agg({'TenYearCHD':'mean', 'p
gt2 = df_test.groupby(['currentSmoker'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'currentSmoker')
gy
```

Out[206…

	currentSmoker	TenYearCHD	pred_TenYearCHD	count
0	0	0.503741	0.568579	802
1	1	0.455259	0.412873	637

In [207…

```
x = gy['currentSmoker']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Current Smoker value vs Average TenYea
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Relationship between Diabetes and Average Target Variable - For Test Dataset

In [208…

```
gtt = df_test.groupby(['diabetes'], as_index = False).agg({'TenYearCHD':'mean', 'pred_T
gt2 = df_test.groupby(['diabetes'], as_index = False).agg({'TenYearCHD':'size'})
gt2 = gt2.rename({'TenYearCHD': 'count'}, axis=1)
gy = pd.merge(gtt, gt2, on = 'diabetes')
gy
```

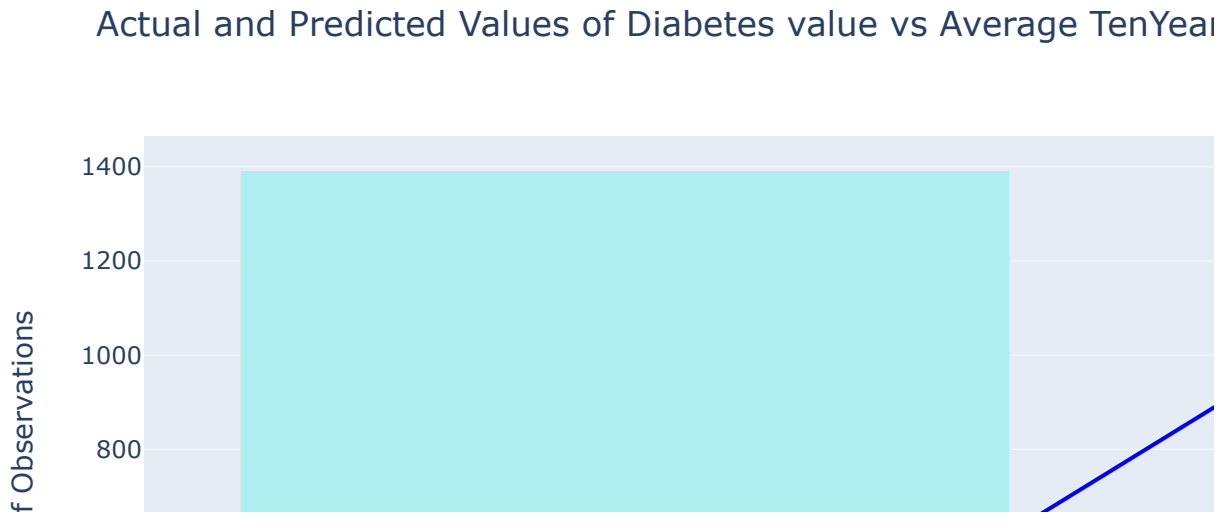
Out[208…

	diabetes	TenYearCHD	pred_TenYearCHD	count
0	0	0.477730	0.487069	1392
1	1	0.617021	0.872340	47

In [209…

```
x = gy['diabetes']
y1 = gy['count']
y2 = gy['TenYearCHD']
y3 = gy['pred_TenYearCHD']


fig = go.Figure(
    data=go.Bar(
        x=x,
        y=y1,
        name="Number of Observations",
        marker=dict(color="paleturquoise"),
    )
```

```
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y2,
        yaxis="y2",
        name="actual value",
        marker=dict(color="crimson"),
    )
)

fig.add_trace(
    go.Scatter(
        x=x,
        y=y3,
        yaxis="y2",
        name="predicted value",
        marker=dict(color="blue"),
    )
)

fig.update_layout(
    title_text = 'Actual and Predicted Values of Diabetes value vs Average TenYearCHD -
    legend=dict(orientation="h"),
    yaxis=dict(
        title=dict(text="Number of Observations"),
        side="left",
        #range=[0, 500],
    ),
    yaxis2=dict(
        title=dict(text="Average of y"),
        side="right",
        #range=[0, 1],
        overlaying="y",
        tickmode="sync",
    ),
)

fig.show()
```



Hypothesis Testing Using P-Value

In [210…

```
# Sample data for two variables in a DataFrame
x = dt.drop('TenYearCHD', axis=1)
y = dt['TenYearCHD']

# Calculate Spearman's rank correlation
correlation, p_value = stats.spearmanr(x, y)

# Print the result
print("Spearman's Rank Correlation:")
print(correlation)
print('\n')
print("P-value:")
print(p_value)
```

```
Spearman's Rank Correlation:
[[ 1.         -0.0296442  -0.00374437  0.19702562  0.28108471 -0.05355439
  -0.0045504   0.00585284  0.01569307 -0.06433362 -0.00551057  0.06735472
   0.14101454 -0.11512255 -0.01262974  0.08837357]
 [-0.0296442   1.         -0.18582197 -0.21145092 -0.21353162  0.11663118
   0.05508612  0.30529306  0.10085484  0.28266308  0.39097795  0.20796485
   0.14454074 -0.01456725  0.11125817  0.22295345]
 [-0.00374437 -0.18582197  1.          0.02229362  0.01562028 -0.00640304
  -0.03400489 -0.08359411 -0.03815147 -0.02926679 -0.1306784  -0.06701049
  -0.13940792 -0.03685502 -0.01715714 -0.06384619]
 [ 0.19702562 -0.21145092  0.02229362  1.          0.92869082 -0.04794132
  -0.03298039 -0.1037103  -0.0442853  -0.04971856 -0.13066083 -0.11442958
  -0.17282516  0.07033363 -0.07426864  0.0194485 ]
 [ 0.28108471 -0.21353162  0.01562028  0.92869082  1.         -0.047879
  -0.03540561 -0.09404115 -0.04107199 -0.04140202 -0.11294143 -0.09179921
  -0.13965475  0.07689313 -0.08596211  0.04337426]
 [-0.05355439  0.11663118 -0.00640304 -0.04794132 -0.047879    1.
   0.11351442  0.25771866  0.05055849  0.07477784  0.19937109  0.17385717
   0.08288984  0.00359689  0.01080933  0.08460398]
 [-0.0045504   0.05508612 -0.03400489 -0.03298039 -0.03540561  0.11351442
   1.          0.07479113  0.00695509  0.00711035  0.0592879   0.04625165
   0.00794945 -0.01361601  0.00218933  0.06182263]
 [ 0.00585284  0.30529306 -0.08359411 -0.1037103  -0.09404115  0.25771866
   0.07479113  1.          0.07775205  0.16080269  0.69453917  0.61471116
   0.28403929  0.13634544  0.07610466  0.17745756]
 [ 0.01569307  0.10085484 -0.03815147 -0.0442853  -0.04107199  0.05055849
   0.00695509  0.07775205  1.          0.03160615  0.09045386  0.04678374
   0.06904262  0.04895513  0.20939978  0.09734424]
 [-0.06433362  0.28266308 -0.02926679 -0.04971856 -0.04140202  0.07477784
   0.00711035  0.16080269  0.03160615  1.          0.22006918  0.1824307
```

```
  0.14690923  0.08663443  0.03342172  0.08153297]
 [-0.00551057  0.39097795 -0.1306784  -0.13066083 -0.11294143  0.19937109
  0.0592879   0.69453917  0.09045386  0.22006918  1.          0.77761502
  0.32272698  0.17130895  0.10803424  0.1943749 ]
 [ 0.06735472  0.20796485 -0.06701049 -0.11442958 -0.09179921  0.17385717
  0.04625165  0.61471116  0.04678374  0.1824307   0.77761502  1.
  0.37390053  0.17849453  0.04220837  0.13150104]
 [ 0.14101454  0.14454074 -0.13940792 -0.17282516 -0.13965475  0.08288984
  0.00794945  0.28403929  0.06904262  0.14690923  0.32272698  0.37390053
  1.          0.05667904  0.06661801  0.07233278]
 [-0.11512255 -0.01456725 -0.03685502  0.07033363  0.07689313  0.00359689
  -0.01361601  0.13634544  0.04895513  0.08663443  0.17130895  0.17849453
  0.05667904  1.          0.09145743  0.01815041]
 [-0.01262974  0.11125817 -0.01715714 -0.07426864 -0.08596211  0.01080933
  0.00218933  0.07610466  0.20939978  0.03342172  0.10803424  0.04220837
  0.06661801  0.09145743  1.          0.05265576]
 [ 0.08837357  0.22295345 -0.06384619  0.0194485   0.04337426  0.08460398
  0.06182263  0.17745756  0.09734424  0.08153297  0.1943749   0.13150104
  0.07233278  0.01815041  0.05265576  1.        ]]


P-value:
[[0.00000000e+000 5.35887798e-002 8.07428822e-001 2.26668624e-038
  7.86790094e-078 4.85403440e-004 7.67066112e-001 7.03203267e-001
  3.06960495e-001 2.76250677e-005 7.19804960e-001 1.13604366e-005
  2.81906948e-020 5.51809185e-014 4.10974851e-001 8.21173518e-009]
 [5.35887798e-002 0.00000000e+000 3.00667959e-034 4.63533249e-044
  6.45107449e-045 2.57149162e-014 3.32438719e-004 3.67799322e-092
  4.64026659e-011 1.00903251e-078 6.73870204e-155 1.20410045e-042
  3.11662096e-021 3.42966014e-001 3.72940310e-013 6.56275271e-049]
 [8.07428822e-001 3.00667959e-034 0.00000000e+000 1.46666451e-001
  3.09210145e-001 6.76812115e-001 2.68132908e-002 5.00332336e-008
  1.29764141e-002 5.67071465e-002 1.30431742e-017 1.25947743e-005
  7.54909875e-020 1.63980718e-002 2.64017885e-001 3.17715635e-005]
 [2.26668624e-038 4.63533249e-044 1.46666451e-001 0.00000000e+000
  0.00000000e+000 1.79268792e-003 3.17552401e-002 1.29322447e-011
  3.92391505e-003 1.20165882e-003 1.31746576e-017 7.81032317e-014
  8.73938299e-030 4.55899771e-006 1.29002584e-006 2.05462672e-001]
 [7.86790094e-078 6.45107449e-045 3.09210145e-001 0.00000000e+000
  0.00000000e+000 1.81759499e-003 2.11391791e-002 8.49927443e-010
  7.47840793e-003 7.01217259e-003 1.63540857e-013 2.11893836e-009
  6.49375336e-020 5.36258090e-007 2.06845464e-008 4.73066245e-003]
 [4.85403440e-004 2.57149162e-014 6.76812115e-001 1.79268792e-003
  1.81759499e-003 0.00000000e+000 1.23179748e-013 2.67884564e-065
  9.90308163e-004 1.09045492e-006 2.88215230e-039 3.97632667e-030
  6.47697405e-008 8.14872928e-001 4.81640985e-001 3.44283434e-008]
 [7.67066112e-001 3.32438719e-004 2.68132908e-002 3.17552401e-002
  2.11391791e-002 1.23179748e-013 0.00000000e+000 1.08566912e-006
  6.50725285e-001 6.43463538e-001 1.12091006e-004 2.59179310e-003
  6.04816569e-001 3.75407329e-001 8.86671553e-001 5.61901411e-005]
 [7.03203267e-001 3.67799322e-092 5.00332336e-008 1.29322447e-011
  8.49927443e-010 2.67884564e-065 1.08566912e-006 0.00000000e+000
  3.99851370e-007 5.87919294e-026 0.00000000e+000 0.00000000e+000
  1.66508411e-079 4.78117410e-019 7.00188848e-007 2.45259083e-031]
 [3.06960495e-001 4.64026659e-011 1.29764141e-002 3.92391505e-003
  7.47840793e-003 9.90308163e-004 6.50725285e-001 3.99851370e-007
  0.00000000e+000 3.95948784e-002 3.62862909e-009 2.31055527e-003
  6.80257627e-006 1.42914698e-003 3.17302734e-043 2.12785214e-010]
 [2.76250677e-005 1.00903251e-078 5.67071465e-002 1.20165882e-003
  7.01217259e-003 1.09045492e-006 6.43463538e-001 5.87919294e-026
  3.95948784e-002 0.00000000e+000 1.14627080e-047 4.73791105e-033
  6.88228759e-022 1.60272778e-008 2.95379997e-002 1.05881416e-007]
 [7.19804960e-001 6.73870204e-155 1.30431742e-017 1.31746576e-017
  1.63540857e-013 2.88215230e-039 1.12091006e-004 0.00000000e+000
  3.62862909e-009 1.14627080e-047 0.00000000e+000 0.00000000e+000
```

```
 2.30377777e-103 2.75473945e-029 1.74777253e-012 2.25945330e-037]
[1.13604366e-005 1.20410045e-042 1.25947743e-005 7.81032317e-014
 2.11893836e-009 3.97632667e-030 2.59179310e-003 0.00000000e+000
 2.31055527e-003 4.73791105e-033 0.00000000e+000 0.00000000e+000
 8.50037894e-141 1.08727273e-031 5.98093888e-003 8.14279017e-018]
[2.81906948e-020 3.11662096e-021 7.54909875e-020 8.73938299e-030
 6.49375336e-020 6.47697405e-008 6.04816569e-001 1.66508411e-079
 6.80257627e-006 6.88228759e-022 2.30377777e-103 8.50037894e-141
 0.00000000e+000 2.22015227e-004 1.41581328e-005 2.41999066e-006]
[5.51809185e-014 3.42966014e-001 1.63980718e-002 4.55899771e-006
 5.36258090e-007 8.14872928e-001 3.75407329e-001 4.78117410e-019
 1.42914698e-003 1.60272778e-008 2.75473945e-029 1.08727273e-031
 2.22015227e-004 0.00000000e+000 2.43100209e-009 2.37357348e-001]
[4.10974851e-001 3.72940310e-013 2.64017885e-001 1.29002584e-006
 2.06845464e-008 4.81640985e-001 8.86671553e-001 7.00188848e-007
 3.17302734e-043 2.95379997e-002 1.74777253e-012 5.98093888e-003
 1.41581328e-005 2.43100209e-009 0.00000000e+000 6.03440147e-004]
[8.21173518e-009 6.56275271e-049 3.17715635e-005 2.05462672e-001
 4.73066245e-003 3.44283434e-008 5.61901411e-005 2.45259083e-031
 2.12785214e-010 1.05881416e-007 2.25945330e-037 8.14279017e-018
 2.41999066e-006 2.37357348e-001 6.03440147e-004 0.00000000e+000]]
```

Checking if Model is Overfitting

In [211…

```
# Assuming X and y are your features and target variable
X = dt.drop(columns=['TenYearCHD'])
y = dt['TenYearCHD']

# Step 1: Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4

# Step 2: Train the XGBoost model on the training set
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

# Step 3: Make predictions on the training set
train_predictions = lr_model.predict(X_train)

# Evaluate performance on the training set
train_accuracy = accuracy_score(y_train, train_predictions)
print(f'Training Accuracy: {train_accuracy:.2f}')

# Make predictions on the test set
test_predictions = lr_model.predict(X_test)

# Evaluate performance on the test set
test_accuracy = accuracy_score(y_test, test_predictions)
print(f'Test Accuracy: {test_accuracy:.2f}')

# Check for overfitting
if train_accuracy > test_accuracy:
    print("Warning: The model may be overfitting.")
else:
    print("The model does not appear to be overfitting.")
```

```
Training Accuracy: 0.85
Test Accuracy: 0.86
The model does not appear to be overfitting.
```

In [ ]: