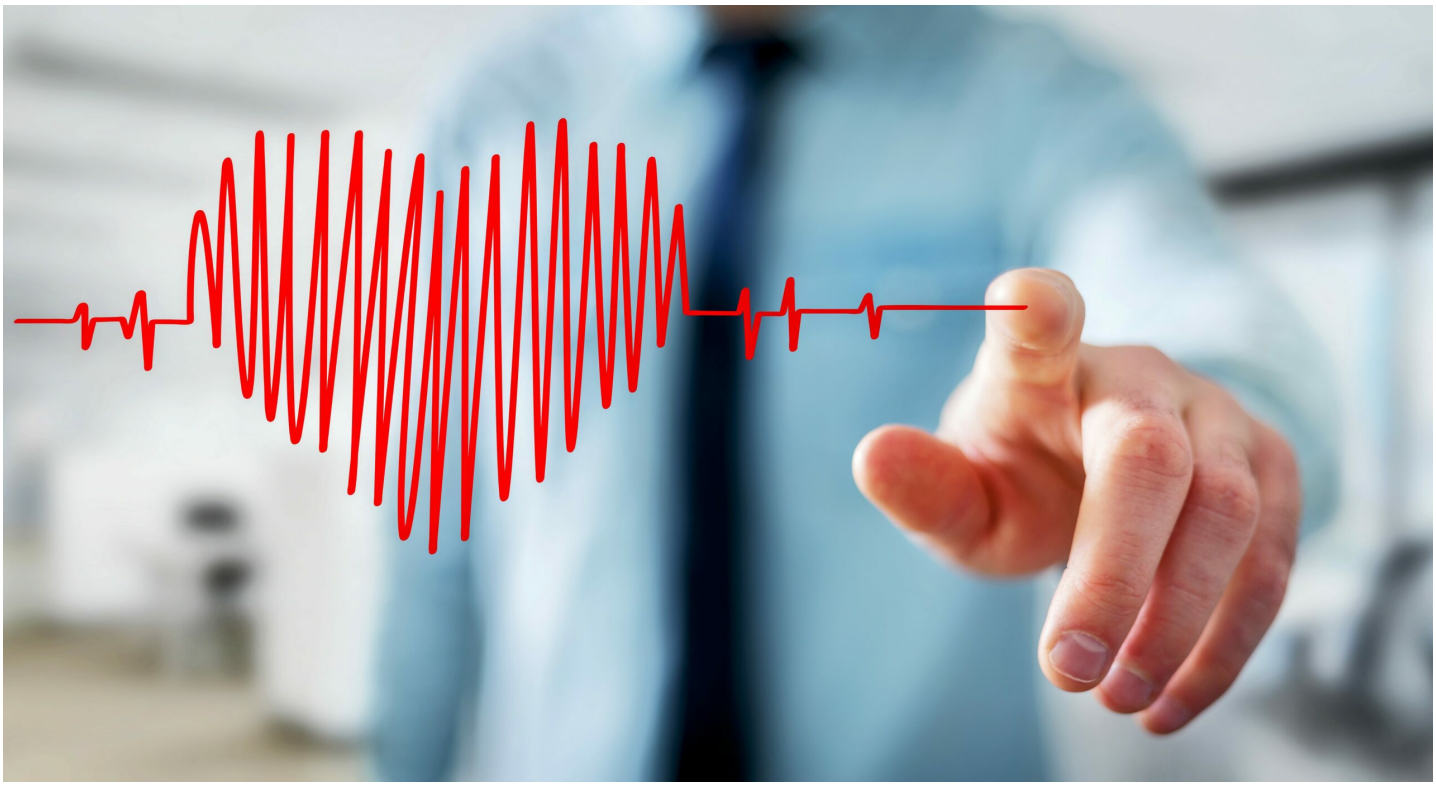
Open in Colab

Cardiovascular Risk Prediction



About Project:

> **The dataset is from an ongoing cardiovascular study on residents of the town of Framingham, Massachusetts. The classification goal is to predict whether the patient has a 10-year risk of future coronary heart disease (CHD). The dataset provides the patients' information. It includes over 4,000 records and 15 attributes.**

Variables:

- **Each attribute is a potential risk factor. There are both demographic, behavioral, and medical risk factors.**

Objective:

> **Predict the overall risk of heart disease using Classification regression**

Data Description

Demographic

1. **Sex:** male or female("M" or "F")
2. **Age:** Age of the patient;(Continuous - Although the recorded ages have been truncated to whole numbers, the concept of age is continuous)

Behavioral

1. **is_smoking:** whether or not the patient is a current smoker ("YES" or "NO")
2. **Cigs Per Day:** the number of cigarettes that the person smoked on average in one day.(can be considered continuous as one can have any number of cigarettes, even half a cigarette.)

Medical(history)

1. **BP Meds:** whether or not the patient was on blood pressure medication (Nominal)
2. **Prevalent Stroke:** whether or not the patient had previously had a stroke (Nominal)
3. **Prevalent Hyp:** whether or not the patient was hypertensive (Nominal)
4. **Diabetes:** whether or not the patient had diabetes (Nominal)

Medical(current)

1. **Tot Chol:** total cholesterol level (Continuous)
2. **Sys BP:** systolic blood pressure (Continuous)
3. **Dia BP:** diastolic blood pressure (Continuous)
4. **BMI:** Body Mass Index (Continuous)
5. **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.)
6. **Glucose:** glucose level (Continuous)

Predict variable (desired target)

1. **TenYearCHD:** 10-year risk of coronary heart disease CHD(binary: 1 means “Yes”, 0 means “No”) - DV

What is Cardiovascular Disease?

Cardiovascular disease can refer to a number of conditions:

Heart disease

Heart and blood vessel disease (also called heart disease) includes numerous problems, many of which are related to a process called atherosclerosis.

Atherosclerosis is a condition that develops when a substance called plaque builds up in the walls of the arteries. This buildup narrows the arteries, making it harder for blood to flow through. If a blood clot forms, it can block the blood flow. This can cause a heart attack or stroke.

Heart attack

A heart attack occurs when the blood flow to a part of the heart is blocked by a blood clot. If this clot cuts off the blood flow completely, the part of the heart muscle supplied by that artery begins to die.

Most people survive their first heart attack and return to their normal lives, enjoying many more years of productive activity. But experiencing a heart attack does mean that you need to make some changes.

The medications and lifestyle changes that your doctor recommends may vary according to how badly your heart was damaged, and to what degree of heart disease caused the heart attack.

Learn more about heart attack.

Stroke

An ischemic stroke (the most common type of stroke) occurs when a blood vessel that feeds the brain gets blocked, usually from a blood clot.

When the blood supply to a part of the brain is cut off, some brain cells will begin to die. This can result in the loss of functions controlled by that part of the brain, such as walking or talking.

A hemorrhagic stroke occurs when a blood vessel within the brain bursts. This is most often caused by uncontrolled hypertension (high blood pressure).

Some effects of stroke are permanent if too many brain cells die after being starved of oxygen. These cells are never replaced.

The good news is that sometimes brain cells don't die during stroke — instead, the damage is temporary. Over time, as injured cells repair themselves, previously impaired function improves. (In other cases, undamaged brain cells nearby may take over for the areas of the brain that were injured.)

Either way, strength may return, speech may get better and memory may improve. This recovery process is what stroke rehabilitation is all about.

When it comes to spotting stroke and getting help, the faster, the better. That's because prompt treatment may make the difference between life and death — or the difference between a full recovery and long-term disability. Use the letters in F.A.S.T to spot a stroke. F is for face drooping. A is for arm weakness. S is for speech difficulty. T is for time to call 911.

Learn more about stroke.

Heart failure

Heart failure, sometimes called congestive heart failure, means the heart isn't pumping blood as well as it should. Heart failure does not mean that the heart stops beating — that's a common misperception. Instead, the heart keeps working, but the body's need for blood and oxygen isn't being met.

Heart failure can get worse if left untreated. If your loved one has heart failure, it's very important to follow the doctor's orders.

Learn more about heart failure.

Arrhythmia

Arrhythmia refers to an abnormal heart rhythm. There are various types of arrhythmias. The heart can beat too slow, too fast or irregularly.

Bradycardia, or a heart rate that's too slow, is when the heart rate is less than 60 beats per minute. Tachycardia, or a heart rate that's too fast, refers to a heart rate of more than 100 beats per minute.

An arrhythmia can affect how well your heart works. With an irregular heartbeat, your heart may not be able to pump enough blood to meet your body's needs.

Learn more about arrhythmia.

Heart valve problems

When heart valves don't open enough to allow the blood to flow through as it should, a condition called stenosis results. When the heart valves don't close properly and thus allow blood to leak through, it's called regurgitation. If the valve leaflets bulge or prolapse back into the upper chamber, it's a condition called prolapse. Discover more about the roles your heart valves play in healthy circulation.

Learn more about heart valve disease.

Data preparation phase

Importing required libraries

We imported several libraries for the project:

1. **numpy:** To work with arrays
2. **pandas:** To work with csv files and dataframes
3. **matplotlib:** To create charts using `pyplot`, define parameters using `rcParams` and color them with `cm.rainbow`
4. **warnings:** To ignore all warnings which might be showing up in the notebook due to past/future depreciation of a feature
5. **train_test_split:** To split the dataset into training and testing data
6. **StandardScaler:** To scale all the features, so that the Machine Learning model better adapts to the dataset

Next, we imported all the necessary Machine Learning algorithms.

In [1]:

```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
rcParams['figure.figsize'] = (10, 6)
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn import ensemble
```

Loading data from csv file to dataframe

In [2]:

```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [3]:

```
path = './data_cardiovascular_risk.csv'
```

In [4]:

```
# using pandas library and 'read_csv' function to read csv file
dataset = pd.read_csv(path)

# Make a copy of data
df = dataset.copy()
```

Data Exploration

In [5]:

```
#number of rows and columns
df.shape
```

Out[5]:

```
(3390, 17)
```

- **This Dataset has 3390 observations in it with 17 columns(features)**

In [6]:

```
df.columns.tolist()
```

Out[6]:

```
['id',
 'age',
 'education',
 'sex',
 'is_smoking',
 'cigsPerDay',
 'BPMeds',
 'prevalentStroke',
 'prevalentHyp',
 'diabetes',
 'totChol',
 'sysBP',
 'diaBP',
 'BMI',
 'heartRate',
 'glucose',
 'TenYearCHD']
```

In [7]:

```
# Here showing in Dataset the First Five rows by using head() method
df.head()
```

Out[7]:

	id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP
0	0	64	2.0	F	YES	3.0	0.0	0	0	0	221.0	148.0	85.0
1	1	36	4.0	M	NO	0.0	0.0	0	1	0	212.0	168.0	98.0
2	2	46	1.0	F	YES	10.0	0.0	0	0	0	250.0	116.0	71.0
3	3	50	1.0	M	YES	20.0	0.0	0	1	0	233.0	158.0	88.0
4	4	64	1.0	F	YES	30.0	0.0	0	0	0	241.0	136.5	85.0

In [8]:

```
# Here showing in Dataset the Last Five rows by using tail() method
df.tail()
```

Out[8]:

	id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP
3385	3385	60	1.0	F	NO	0.0	0.0	0	0	0	261.0	123.5
3386	3386	46	1.0	F	NO	0.0	0.0	0	0	0	199.0	102.0
3387	3387	44	3.0	M	YES	3.0	0.0	0	1	0	352.0	164.0
3388	3388	60	1.0	M	NO	0.0	NaN	0	1	0	191.0	167.0
3389	3389	54	3.0	F	NO	0.0	0.0	0	0	0	288.0	124.0

> **Before any analysis, we just wanted to take a look at the data. So, we used the `info()` method.**

In [9]:

```
#information of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3390 entries, 0 to 3389
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   id               3390 non-null   int64  
 1   age              3390 non-null   int64  
 2   education        3303 non-null   float64
 3   sex              3390 non-null   object 
 4   is_smoking       3390 non-null   object 
 5   cigsPerDay       3368 non-null   float64
 6   BPMeds           3346 non-null   float64
 7   prevalentStroke  3390 non-null   int64  
 8   prevalentHyp     3390 non-null   int64  
 9   diabetes         3390 non-null   int64  
 10  totChol          3352 non-null   float64
 11  sysBP            3390 non-null   float64
 12  diaBP            3390 non-null   float64
 13  BMI              3376 non-null   float64
 14  heartRate        3389 non-null   float64
 15  glucose          3086 non-null   float64
 16  TenYearCHD       3390 non-null   int64  
dtypes: float64(9), int64(6), object(2)
memory usage: 450.4+ KB
```

As you can see from the output above, there are a total of 16 features and 1 target variable. Also, there are some missing values so we need to take care of null values. Next, we used `describe()` method.

In [10]:

```
#description of the data in the dataset
df.describe(include='all')
```

Out[10]:

	id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	
count	3390.000000	3390.000000	3303.000000	3390	3390	3368.000000	3346.000000	3390.000000	3390.000000	3
unique	NaN	NaN	NaN	2	2	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	F	NO	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	1923	1703	NaN	NaN	NaN	NaN	
mean	1694.500000	49.542183	1.970936	NaN	NaN	9.069477	0.029886	0.006490	0.315339	
std	978.753033	8.592878	1.019081	NaN	NaN	11.879078	0.170299	0.080309	0.464719	
min	0.000000	32.000000	1.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	
25%	847.250000	42.000000	1.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	
50%	1694.500000	49.000000	2.000000	NaN	NaN	0.000000	0.000000	0.000000	0.000000	
75%	2541.750000	56.000000	3.000000	NaN	NaN	20.000000	0.000000	0.000000	1.000000	
max	3389.000000	70.000000	4.000000	NaN	NaN	70.000000	1.000000	1.000000	1.000000	

The method revealed that the range of each variable is different. The maximum value of age is 70 but for chol it is 696. Thus, feature scaling must be performed on the dataset.

Data Preprocessing and Basic EDA

> **We will drop the `education` and `id` columns because it has no correlation with heart disease.**

In [11]:

```
# Removing columns such as 'id' & 'education'
df.drop(['id','education'],axis=1,inplace=True)
```

In [12]:

```
df.head()
```

Out[12]:

	age	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartR
0	64	F	YES	3.0	0.0	0	0	0	221.0	148.0	85.0	NaN	9
1	36	M	NO	0.0	0.0	0	1	0	212.0	168.0	98.0	29.77	7
2	46	F	YES	10.0	0.0	0	0	0	250.0	116.0	71.0	20.35	8
3	50	M	YES	20.0	0.0	0	1	0	233.0	158.0	88.0	28.26	6
4	64	F	YES	30.0	0.0	0	0	0	241.0	136.5	85.0	26.42	7

Before we go ahead, an important step to do is to convert our string feature into an integer.

We will name

- **In sex feature M will be converted to 1 and F will be converted to 0.**
- **In is_smoking feature YES will be converted to 1 and NO will be converted to 0.**

In [13]:

```
# Applying function to convert string data to an integer
df['sex'] = df['sex'].apply(lambda x : 1 if x == "M" else 0)
df['is_smoking'] = df['is_smoking'].apply(lambda x : 1 if x == "YES" else 0)
```

In [14]:

```
# checking the distribution
df['sex'].value_counts()
```

Out[14]:

```
sex
0    1923
1    1467
Name: count, dtype: int64
```

In [15]:

```
# checking the distribution
df['is_smoking'].value_counts()
```

Out[15]:

```
is_smoking
0    1703
1    1687
Name: count, dtype: int64
```

In [16]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3390 entries, 0 to 3389
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   age              3390 non-null   int64  
 1   sex              3390 non-null   int64  
 2   is_smoking       3390 non-null   int64  
 3   cigsPerDay       3368 non-null   float64
 4   BPMeds           3346 non-null   float64
 5   prevalentStroke  3390 non-null   int64  
 6   prevalentHyp     3390 non-null   int64  
 7   diabetes         3390 non-null   int64  
 8   totChol          3352 non-null   float64
 9   sysBP            3390 non-null   float64
 10  diaBP            3390 non-null   float64
 11  BMI              3376 non-null   float64
 12  heartRate        3389 non-null   float64
 13  glucose          3086 non-null   float64
 14  TenYearCHD       3390 non-null   int64  
dtypes: float64(8), int64(7)
memory usage: 397.4 KB
```

Missing Value Analysis

Handling missing data is important as many machine learning algorithms do not support data with missing values.

In [17]:

```
# checking for missing values
df.isnull().sum()
```

Out[17]:

```
age                 0
sex                 0
is_smoking          0
cigsPerDay         22
BPMeds             44
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            38
sysBP               0
diaBP               0
BMI                14
heartRate           1
glucose           304
TenYearCHD          0
dtype: int64
```

Visualize missing values (NaN) values using [Missingno Library]

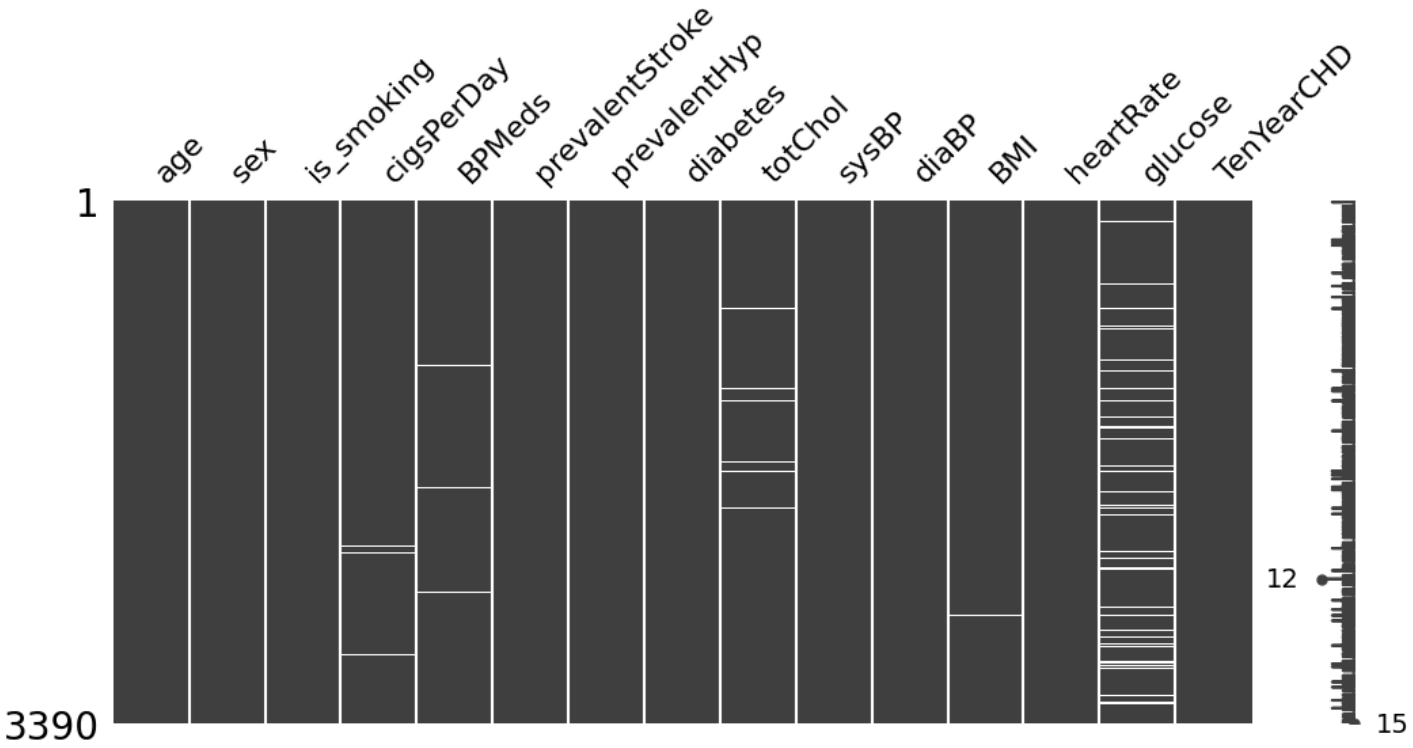
> **The next single-line code will visualize the location of missing values.**

In [18]:

```
# Program to visualize missing values in dataset

# Importing the  Missingno libraries
import missingno as msno

# Visualize missing values as a matrix
msno.matrix(df,figsize=(12, 5))
```

Out[18]:

<Axes: >



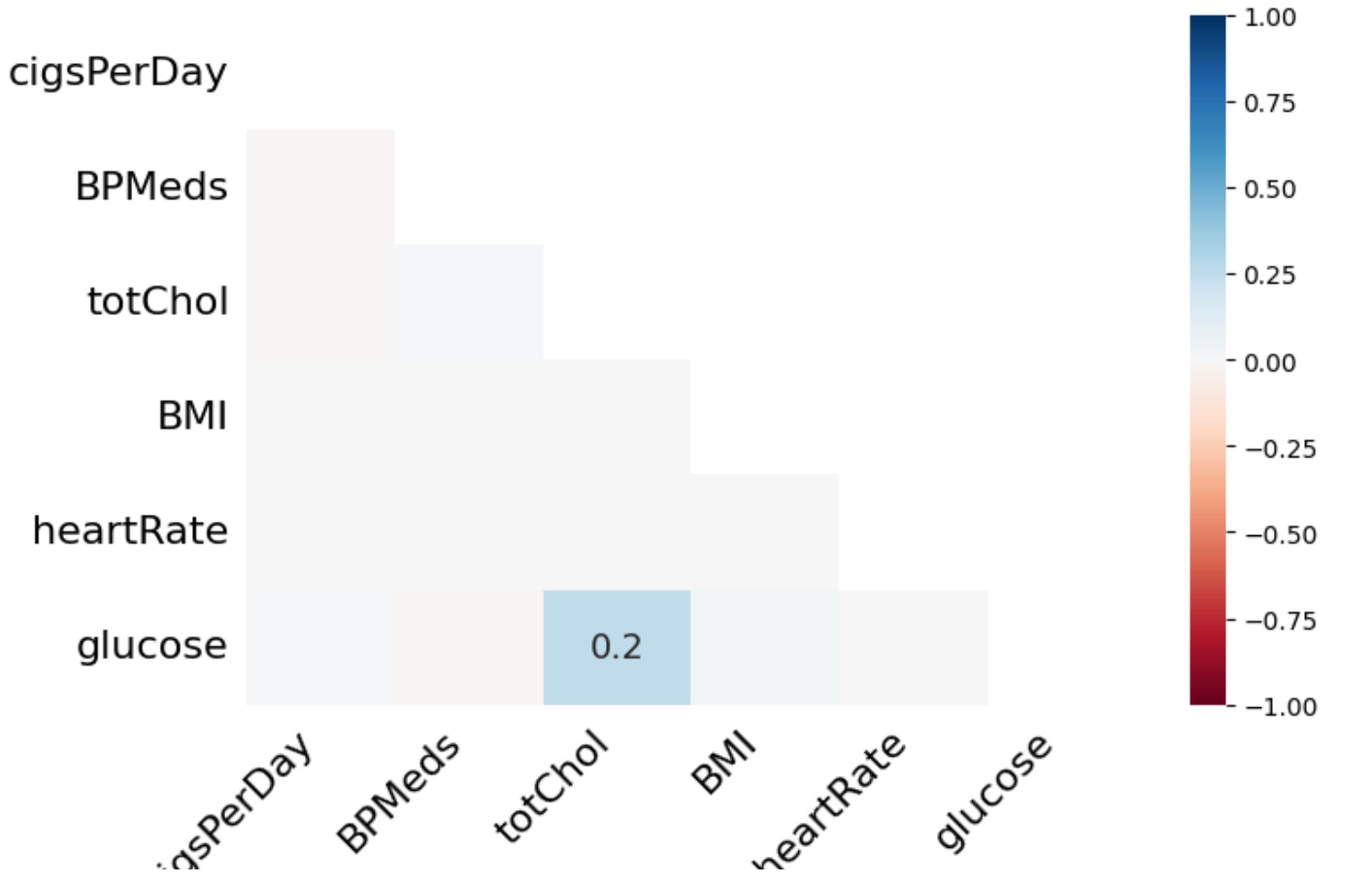
These trends give an idea about how the features are correlated with one another. But to get a better idea about correlations we need to use heatmaps.

In [19]:

```
# Visualize the correlation between the number of
# missing values in different columns as a heatmap
msno.heatmap(df,figsize=(8, 5))
```

Out[19]:

<Axes: >



- **There is no strong Correlation between any features.**

Let's check percentage of missing data of each features

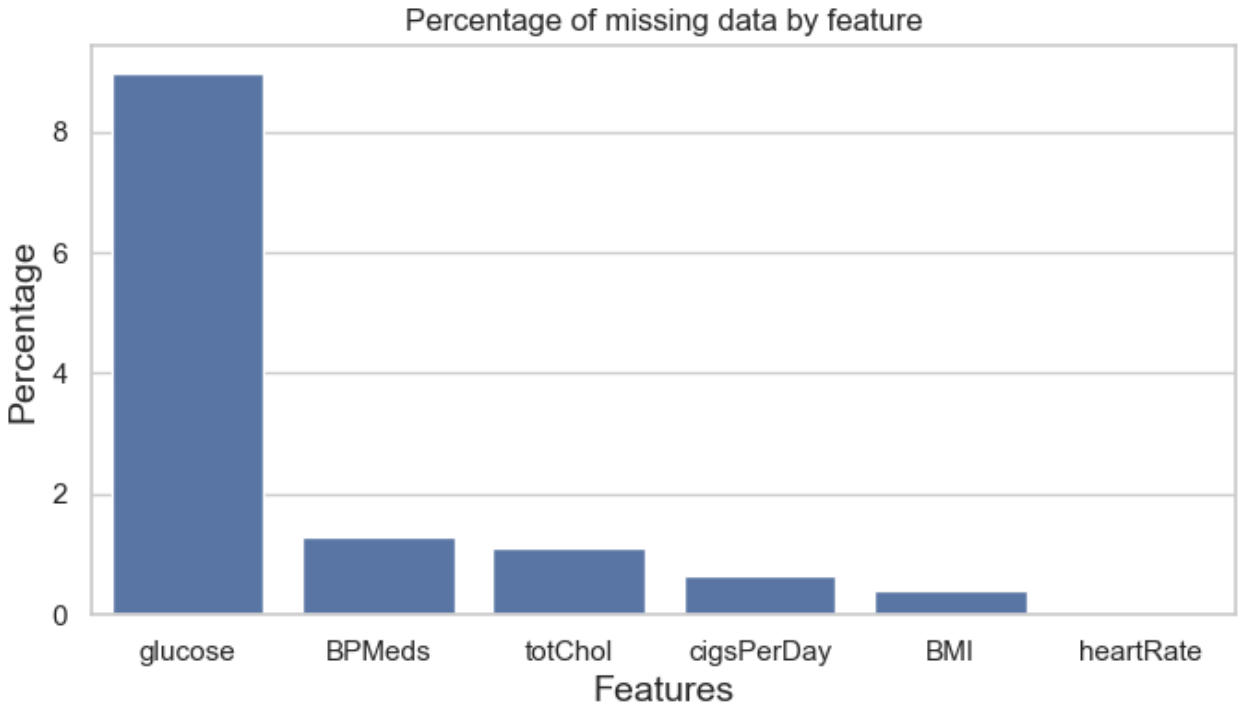
In [20]:

```
# percentage of missing data per category
total = df.isnull().sum().sort_values(ascending=False)
percent_total = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)*100
missing = pd.concat([total, percent_total], axis=1, keys=["Total", "Percentage"])
missingdf = missing[missing['Total']>0]
print(missingdf)
```

```
             Total  Percentage
glucose        304    8.967552
BPMeds          44    1.297935
totChol         38    1.120944
cigsPerDay      22    0.648968
BMI             14    0.412979
heartRate        1    0.029499
```

In [21]:

```
# Visualizing the percentage of missing data
plt.figure(figsize=(8,4))
sns.set(style="whitegrid")
sns.barplot(x=missingdf.index, y=missingdf['Percentage'], data = missingdf)
plt.title('Percentage of missing data by feature')
plt.xlabel('Features', fontsize=14)
plt.ylabel('Percentage', fontsize=14)
plt.show()
```



In [22]:

```
# let's count the all rows which are having missing values
count=0
for i in df.isnull().sum(axis=1):
    if i>0:
        count=count+1
print('Total number of rows with missing values is ', count)
# checking missing value percentage
print('since it is only',round((count/len(df.index))*100), 'percent of the entire dataset
```

```
the rows with missing values are excluded.')
```

```
Total number of rows with missing values is  386
since it is only 11 percent of the entire dataset the rows with missing values are exclud
ed.
```

At 8.97% , the blood glucose entry has the highest percentage of missing data. The otherfeatures have very few missing entries. Since the missing entries account for only `11%` of the total data so, we can drop these entries without losing alot of data.

In [23]:

```
# Excluding the missing values
df.dropna(axis=0,inplace=True)
```

In [24]:

```
# Now checking for missing values
df.isnull().any()
```

Out[24]:

```
age                False
sex                False
is_smoking         False
cigsPerDay         False
BPMeds             False
prevalentStroke    False
prevalentHyp       False
diabetes           False
totChol            False
sysBP              False
diaBP              False
BMI                False
heartRate          False
glucose            False
TenYearCHD         False
dtype: bool
```

In [25]:

```
# Checking for any duplicate values
len(df[df.duplicated()])
```

Out[25]:

```
0
```

In [26]:

```
# statistical measures about the data
df.describe()
```

Out[26]:

	age	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totC
count	3004.000000	3004.000000	3004.000000	3004.000000	3004.000000	3004.000000	3004.000000	3004.000000	3004.000
mean	49.521305	0.448402	0.492676	9.078562	0.030293	0.005992	0.314913	0.027297	237.222
std	8.595076	0.497413	0.500030	11.890855	0.171421	0.077189	0.464559	0.162974	45.267
min	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	113.000
25%	42.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	206.000
50%	49.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	234.000
75%	56.000000	1.000000	1.000000	20.000000	0.000000	0.000000	1.000000	0.000000	265.000
max	70.000000	1.000000	1.000000	70.000000	1.000000	1.000000	1.000000	1.000000	696.000

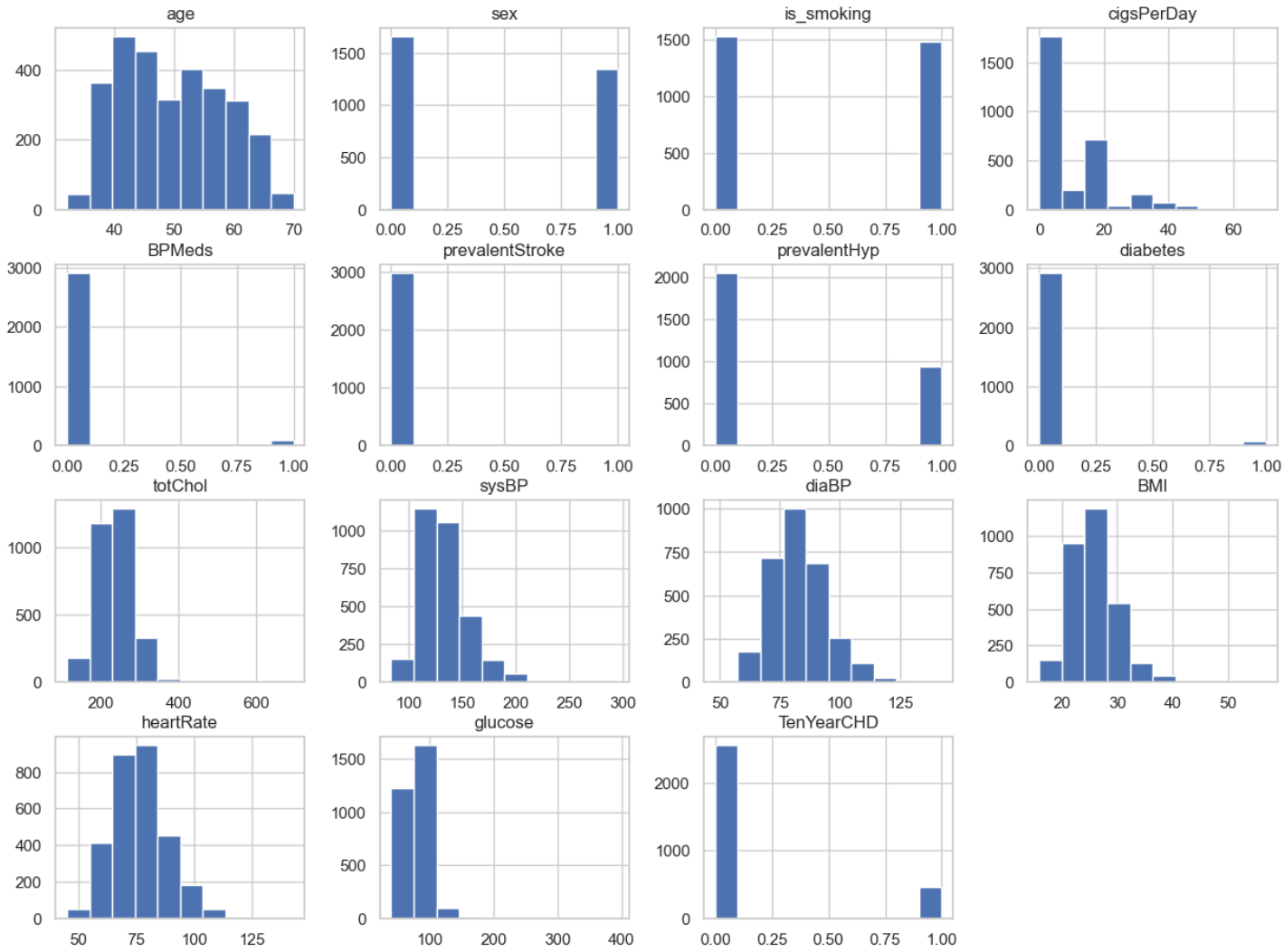
The features described in the above data set are:

- **Count:** tells us the number of NoN-empty rows in a feature.
- **Mean:** tells us the mean value of that feature.
- **Std:** tells us the Standard Deviation Value of that feature.
- **Min:** tells us the minimum value of that feature.
- **25%, 50%, and 75%:** are the percentile/quartile of each features.
- **Max:** tells us the maximum value of that feature.

Now, Let's visualise Data Distribution

In [27]:

```
# plot histogram to see the distribution of the data
fig = plt.figure(figsize = (15,11))
ax = fig.gca()
df.hist(ax = ax)
plt.show()
```



- **From above distribution plot we can say that the data on the prevalent stroke, diabetes, and blood pressure meds(BPMeds) are poorly balanced.**

Analysing features

Target Variable Analysis:

In [28]:

```
# checking the distribution of Target Variable
df.TenYearCHD.value_counts()
```

Out[28]:

```
TenYearCHD
0    2547
1     457
Name: count, dtype: int64
```

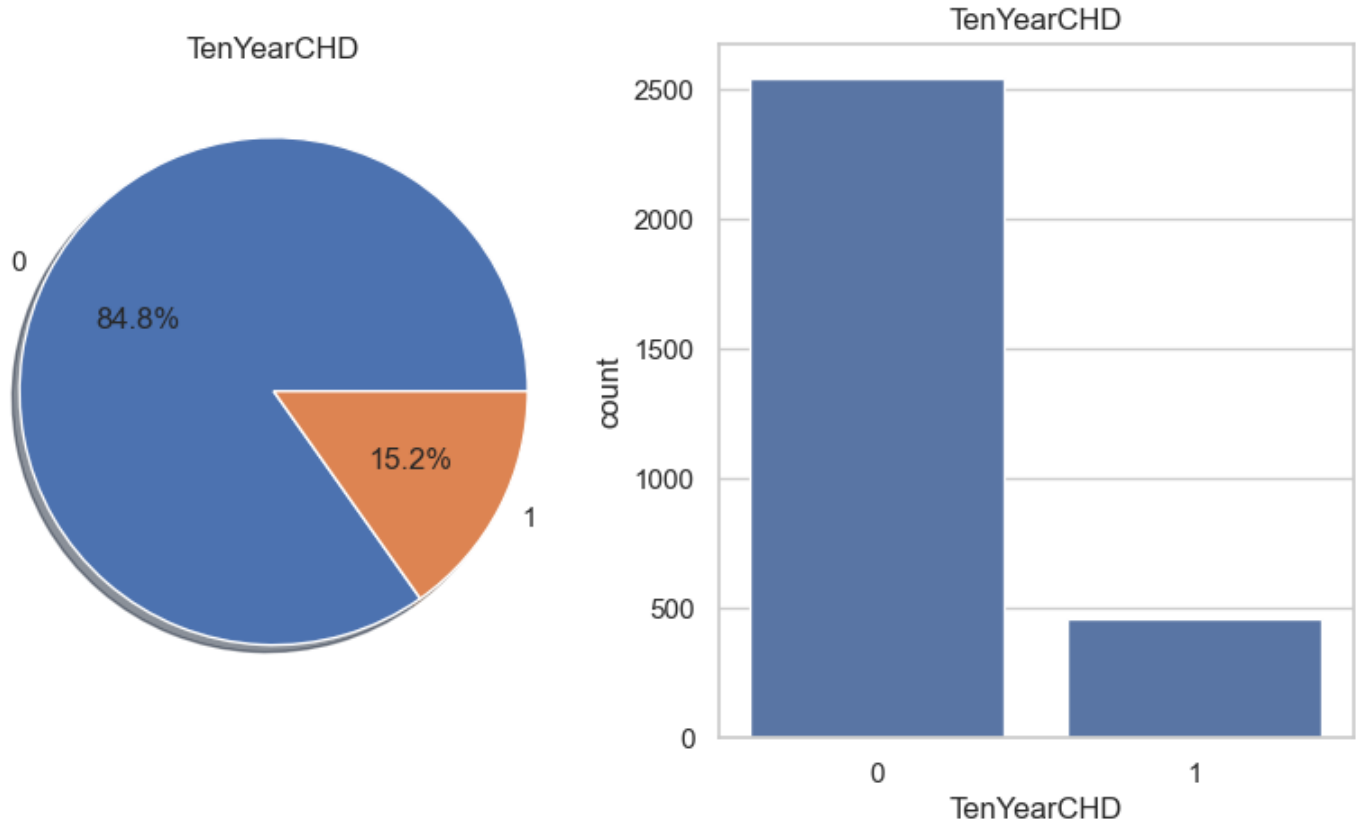
0 --> Person do not have risk of coronary heart disease

1 --> Person have risk of coronary heart disease

There are 2547 patients without heart disease and 457 patients with the disease.

In [29]:

```
#Plotting pie chart of target variable
f,ax=plt.subplots(1,2,figsize=(10,5))
df['TenYearCHD'].value_counts().plot.pie(autopct='%1.1f%%',ax=ax[0],shadow=True)
ax[0].set_title('TenYearCHD')
ax[0].set_ylabel('')
sns.countplot(x='TenYearCHD',data=df,ax=ax[1])
ax[1].set_title('TenYearCHD')
plt.show()
```



- **We can see above that we have the imbalanced data set as the number of people without the disease greatly exceeds the number of people with the disease.**

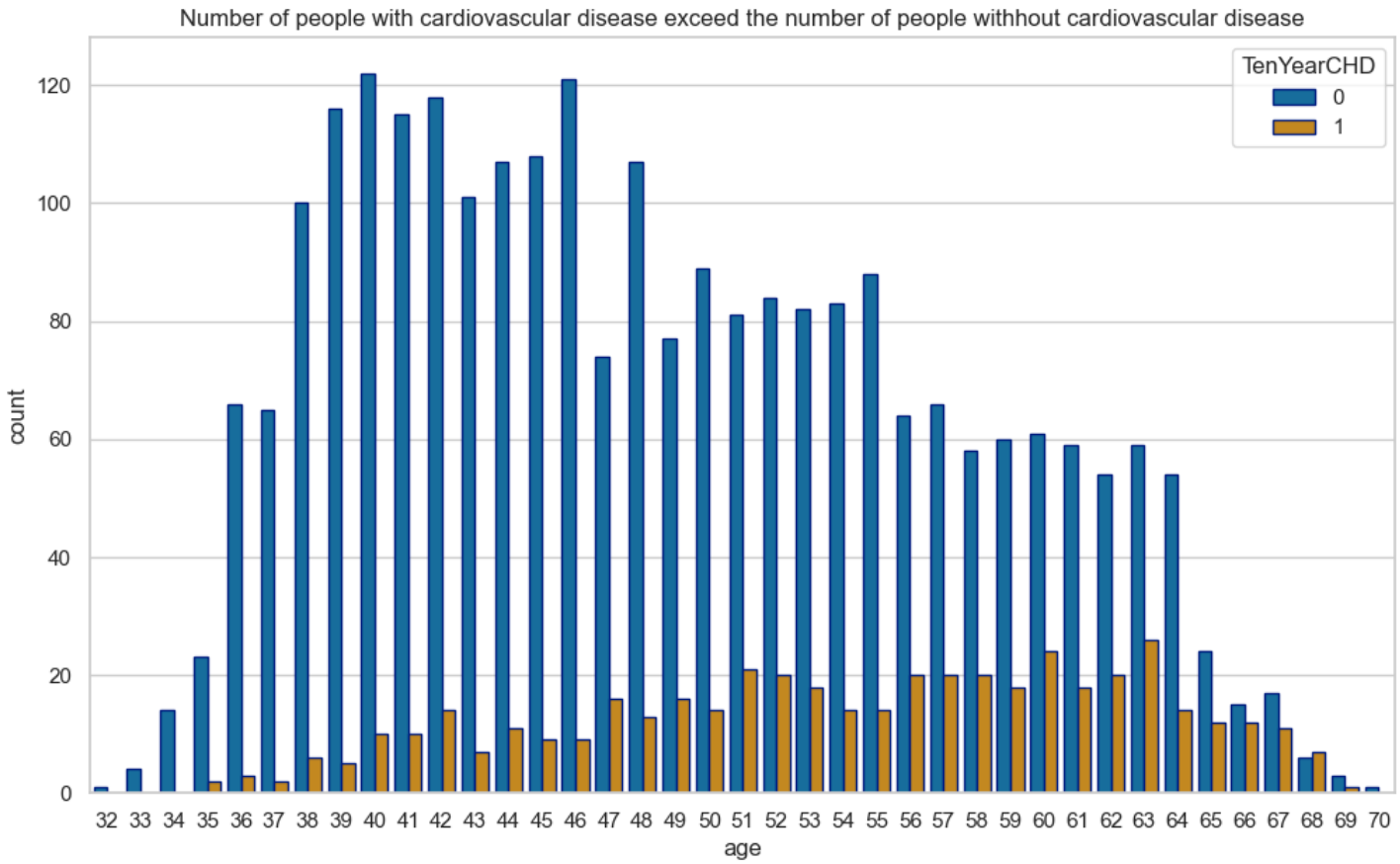
Let's look at the number of people with cardiovascular disease exceed the number of people withhout cardiovascular disease respect to age.

In [30]:

```
# Visualising the target and age variable
plt.figure(figsize=[12,7])
plt.title("Number of people with cardiovascular disease exceed the number of people withh
out cardiovascular disease")
sns.countplot(x='age', hue='TenYearCHD',data=df, palette='colorblind', edgecolor=sns.col
or_palette('dark', n_colors=1)[0])
```

Out[30]:

```
<Axes: title={'center': 'Number of people with cardiovascular disease exceed the number o
f people withhout cardiovascular disease'}, xlabel='age', ylabel='count'>
```



- **As we can see in above plot The people with the highest risk of developing heart disease are betwwen the ages of 51 and 63.**
- **Because the number of sick people generally increases with age.**

Categorical variable comparisons with Target Variable(TenYearCHD):

We will use Stacked Bar chart for Comparison between Categorical variable and Target Variable.

Stacked Bar Chart: ***A stacked bar graph (or stacked bar chart) is a chart that uses bars to show comparisons between categories of data, but with ability to break down and compare parts of a whole. Each bar in the chart represents a whole, and segments in the bar represent different parts or categories of that whole.***

In [31]:

```
from operator import add
def stacked_barchart(data, title = None, ylabel = None, xlabel = None):
  # Function to plot stacked bar chart
    default_colors = ['#006400', '#FF0000', '#228B22']
    # From raw value to percentage
    totals = data.sum(axis=1)
    bars = ((data.T / totals) * 100).T
    r = list(range(data.index.size))

    # Plot
    barWidth = 0.95
    names = data.index.tolist()
    bottom = [0] * bars.shape[0]

    # Create bars
    color_index = 0
    plots = []
    for bar in bars.columns:
```

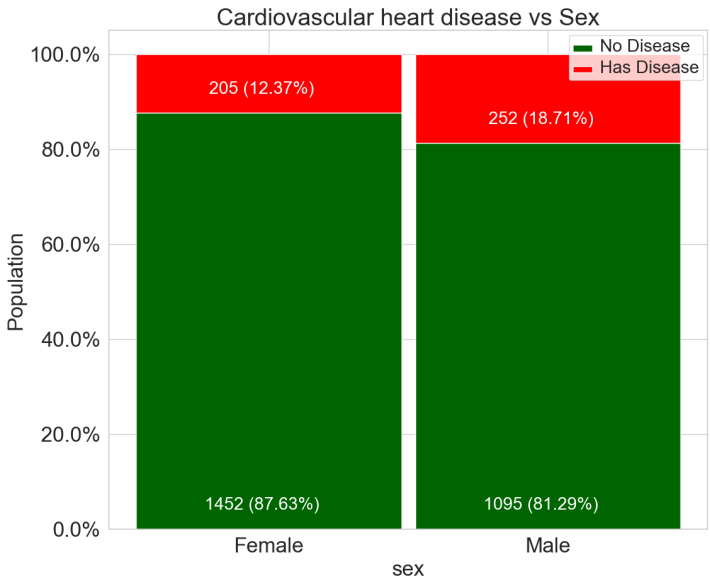
```
        plots.append(plt.bar(r, bars[bar], bottom=bottom, color=default_colors[color_ind
ex], edgecolor='white', width=barWidth))
        bottom = list(map(add, bottom, bars[bar]))
        color_index = 0 if color_index >= len(default_colors) else color_index + 1

    # Custom x axis
    plt.title(title)
    plt.xticks(r, names)
    plt.xlabel(data.index.name if xlabel is None else xlabel)
    plt.ylabel(data.columns.name if ylabel is None else ylabel)
    ax = plt.gca()

    y_labels = ax.get_yticks()
    ax.set_yticklabels([str(y) + '%' for y in y_labels])

    flat_list = [item for sublist in data.T.values for item in sublist]
    for i, d in zip(ax.patches, flat_list):
        data_label = str(d) + " (" + str(round(i.get_height(), 2)) + "%)"
        ax.text(i.get_x() + 0.45, i.get_y() + 5, data_label, horizontalalignment='center
', verticalalignment='center', fontdict = dict(color = 'white', size = 20))

    for item in ([ax.title]):
        item.set_fontsize(27)

    for item in ([ax.xaxis.label, ax.yaxis.label] + ax.get_xticklabels() + ax.get_yticklabels()):
        item.set_fontsize(24)

    legend = ax.legend(plots, bars.columns.tolist(), fancybox=True)
    plt.setp(legend.get_texts(), fontsize='20')
```

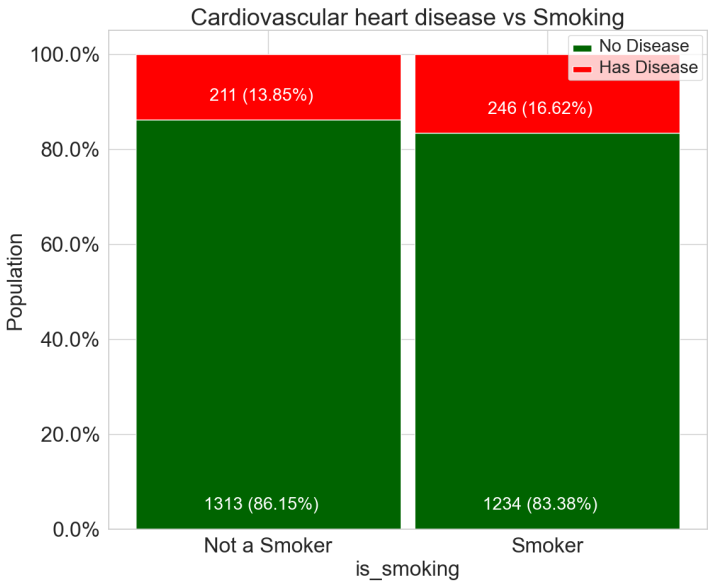
Now, Let's Visualzing each category with respect to target variable

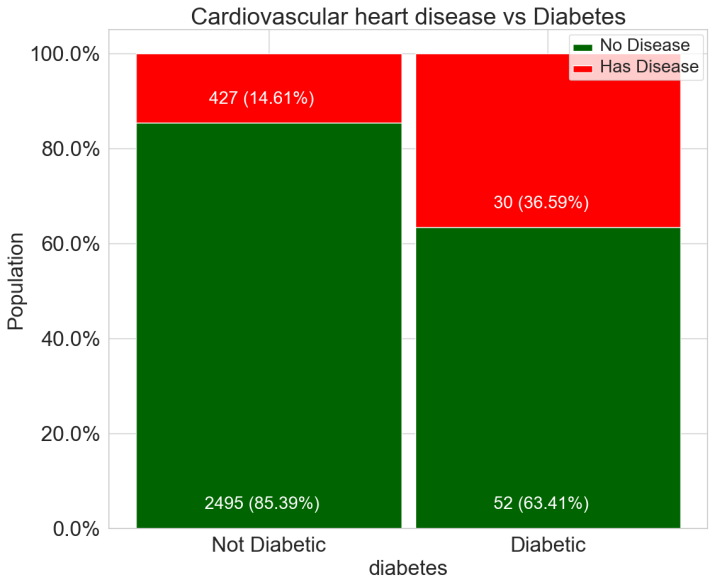
In [32]:

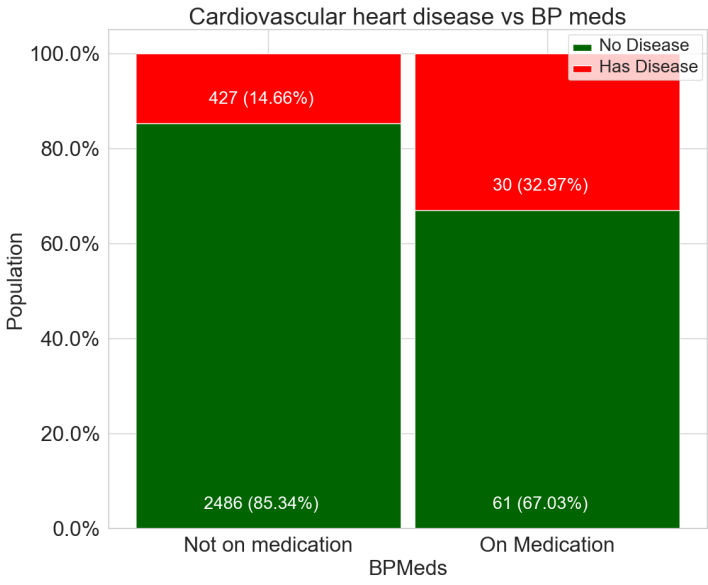
```
# Visualzing each category with respect to target variable
fig = plt.gcf()
fig.set_size_inches(27, 35)
grid_rows = 3
grid_cols = 2

#Plot sex vs disease outcome
plt.subplot(grid_rows, grid_cols, 1)
temp = df[['sex','TenYearCHD']].groupby(['sex','TenYearCHD']).size().unstack('TenYearCHD
')
temp.rename(index={0:'Female', 1:'Male'}, columns={0:'No Disease', 1:'Has Disease'}, inp
lace = True)
stacked_barchart(temp, title = 'Cardiovascular heart disease vs Sex', ylabel = 'Populatio
n')

#Plot smoking satus vs disease outcome
plt.subplot(grid_rows, grid_cols, 2)
temp = df[['is_smoking','TenYearCHD']].groupby(['is_smoking','TenYearCHD']).size().unsta
ck('TenYearCHD')
temp.rename(index={0:'Not a Smoker', 1:'Smoker'}, columns={0:'No Disease', 1:'Has Diseas
e'}, inplace = True)
stacked_barchart(temp, title = 'Cardiovascular heart disease vs Smoking', ylabel = 'Popul
ation')

#Plot diabetes vs disease outcome
plt.subplot(grid_rows, grid_cols, 3)
temp = df[['diabetes','TenYearCHD']].groupby(['diabetes','TenYearCHD']).size().unstack('
TenYearCHD')
temp.rename(index={0:'Not Diabetic', 1:'Diabetic'}, columns={0:'No Disease', 1:'Has Dise
ase'}, inplace = True)
stacked_barchart(temp, title = 'Cardiovascular heart disease vs Diabetes', ylabel = 'Popu
lation')

#Plot BP meds vs disease outcome
plt.subplot(grid_rows, grid_cols, 4)
temp = df[['BPMeds','TenYearCHD']].groupby(['BPMeds','TenYearCHD']).size().unstack('TenY
```

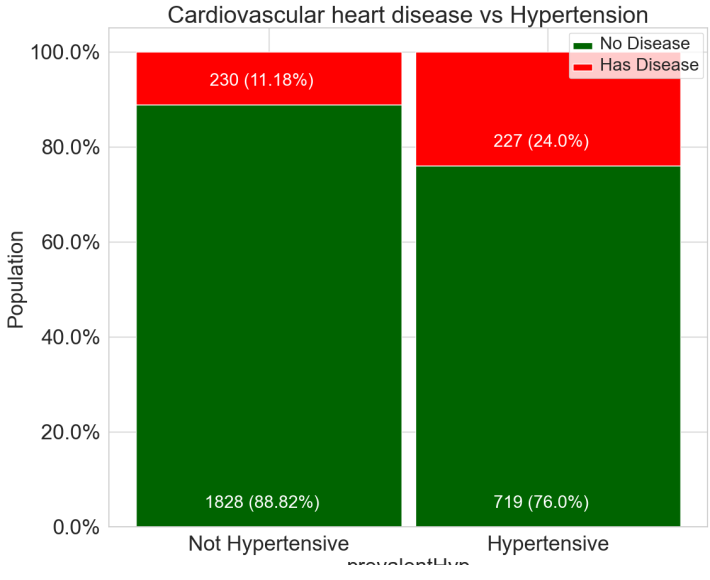
```
earCHD')
temp.rename(index={0:'Not on medication', 1:'On Medication'}, columns={0:'No Disease', 1
:'Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'Cardiovascular heart disease vs BP meds', ylabel = 'Popul
ation')

#Plot Hypertension vs disease outcome
plt.subplot(grid_rows, grid_cols, 5)
temp = df[['prevalentHyp','TenYearCHD']].groupby(['prevalentHyp','TenYearCHD']).size().u
nstack('TenYearCHD')
temp.rename(index={0:'Not Hypertensive', 1:'Hypertensive'}, columns={0:'No Disease', 1:'
Has Disease'}, inplace = True)
stacked_barchart(temp, title = 'Cardiovascular heart disease vs Hypertension', ylabel = '
Population')
```











prevalentHyp

From the above categorical variables comparison plot we can conclude that,

- **Slightly more males are suffering from Cardiovascular heart disease than females.**
- **The people who have Cardiovascular heart disease is almost equal between smokers and non smokers.**
- **The percentage of people who have Cardiovascular heart disease is higher among the diabetic patients and also those patients with prevalent hypertension have more risk of Cardiovascular heart disease compare to those who don't have hypertensive problem.**
- **The percentage of people who are on medication of blood pressure have more risk of Cardiovascular heart disease compare to those who are not on medication.**

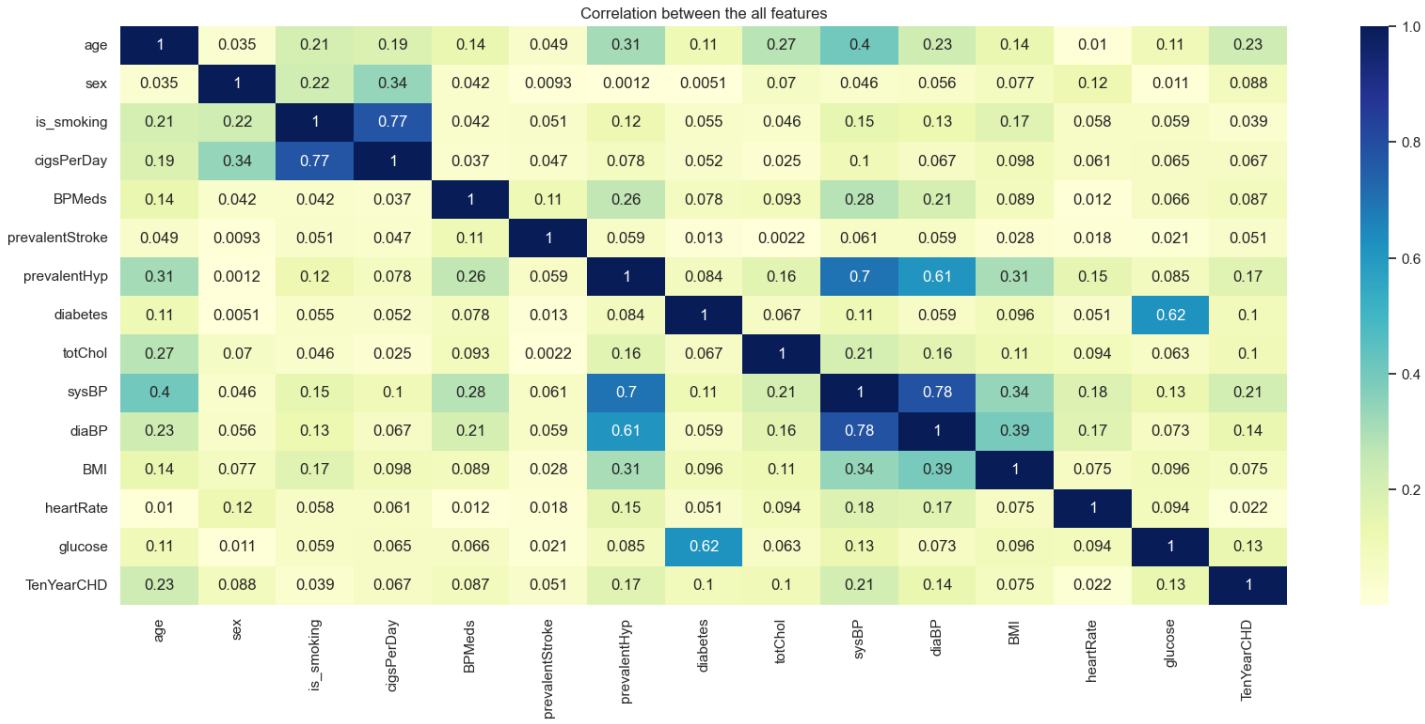
Now, Let's see the Correlation between the all features

In [33]:

```
# let's find all feature's correlation with Heatmap
# correlation
plt.figure(figsize=(20,8))
correlation = df.corr()
sns.heatmap(abs(correlation), annot = True, cmap='YlGnBu')
plt.title('Correlation between the all features')
```

Out[33]:

```
Text(0.5, 1.0, 'Correlation between the all features')
```



From the above correlation plot we can conclude that ,

- **There are no features with more than 0.2 correlation with the Ten year risk of developing CHD and this shows that the features a poor predictors. However the features with the highest correlations are age, prevalent hypertension(prevalentHyp) and systolic blood pressure(sysBP).**
- **Also there are a couple of features that are highly correlated with each other and it makes no sense to use both of them in building a machine learning model.**

These includes:

- **Blood glucose and diabetes;**
- **systolic and diastolic blood pressures;**
- **cigarette smoking and the number of cigarretes smoked per day.**

Therefore we need to carry out feature selection to pick the best features.

Feature Engineering/Selection

Tree-based: SelectFromModel

SelectFromModel is an Embedded method. Embedded methods use algorithms that have built-in feature selection methods.

Here,

We have used RandomForest() to select features based on feature importance. We calculate feature importance using node impurities in each decision tree.

In Random forest, the final feature importance is the average of all decision tree feature importance.

In [34]:

```
#define the features
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [35]:

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier
# define SelectFromModel feature selection method
embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100), max_featu
res=14)
embeded_rf_selector.fit(X, y)

embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features')
```

```
7 selected features
```

In [36]:

```
# Important or top Features
embeded_rf_feature
```

Out[36]:

```
['age', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose']
```

The top features are:

1. **Age**
2. **Total cholesterol**
3. **Systolic blood pressure**
4. **Diastolic blood pressure**
5. **BMI**
6. **Heart rate**
7. **Blood glucose**

Statistics on the top feature:

In [37]:

```
# Importing statsmodels
import statsmodels.api as sm
```

Statsmodels is a Python module that provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration.

In [38]:

```
# Splitting the dependent and independent variables
top_features = df[embeded_rf_feature]
y = df['TenYearCHD']
```

In [39]:

```
# Fit the data
result = sm.Logit(y,top_features).fit()
# The summary table below, gives us a descriptive summary about the regression results
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.415336
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:              TenYearCHD   No. Observations:                 3004
Model:                           Logit   Df Residuals:                     2997
Method:                            MLE   Df Model:                            6
Date:                 Fri, 19 Apr 2024   Pseudo R-squ.:                 0.02592
Time:                         15:46:54   Log-Likelihood:                -1247.7
converged:                       True   LL-Null:                       -1280.9
Covariance Type:            nonrobust   LLR p-value:                 2.236e-12
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
age            0.0226      0.006      3.584      0.000       0.010       0.035
totChol       -0.0018      0.001     -1.554      0.120      -0.004       0.000
sysBP          0.0245      0.004      6.722      0.000       0.017       0.032
diaBP         -0.0297      0.006     -4.601      0.000      -0.042      -0.017
BMI           -0.0544      0.013     -4.082      0.000      -0.081      -0.028
heartRate     -0.0301      0.004     -7.336      0.000      -0.038      -0.022
glucose        0.0055      0.002      3.060      0.002       0.002       0.009
==============================================================================
```

In the output, Iterations refer to the number of times the model iterates over the data, trying to optimise the model.

Explanation of some of the terms in the summary table:

- **coef : the coefficients of the independent variables in the regression equation.**
- **Log-Likelihood : the natural logarithm of the Maximum Likelihood Estimation(MLE) function. MLE is the optimisation process of finding the set of parameters which result in best fit.**
- **LL-Null : the value of log-likelihood of the model when no independent variable is included(only an intercept is included).**
- **Pseudo R-squ. : a substitute for the R-squared value in Least Squares linear regression. It is the ratio of the log-likelihood of the null model to that of the full model.**

In [40]:

```
# Checking the odds ratio of top features
params = result.params
conf = result.conf_int()
conf['Odds Ratio'] = params
conf.columns = ['5%', '95%', 'Odds Ratio']
print(np.exp(conf))
```

```
                5%       95%  Odds Ratio
age       1.010284  1.035552    1.022840
totChol   0.995854  1.000481    0.998165
sysBP     1.017473  1.032083    1.024752
diaBP     0.958523  0.983091    0.970729
BMI       0.922652  0.972119    0.947062
heartRate 0.962550  0.978167    0.970327
glucose   1.001974  1.009041    1.005501
```

From above table we can conclude that the odds of getting cardiovascular heart disease risk increases with

about 2% for every increase in age and systolic blood pressure(sysBP).

The other factors show no significant positive odds.

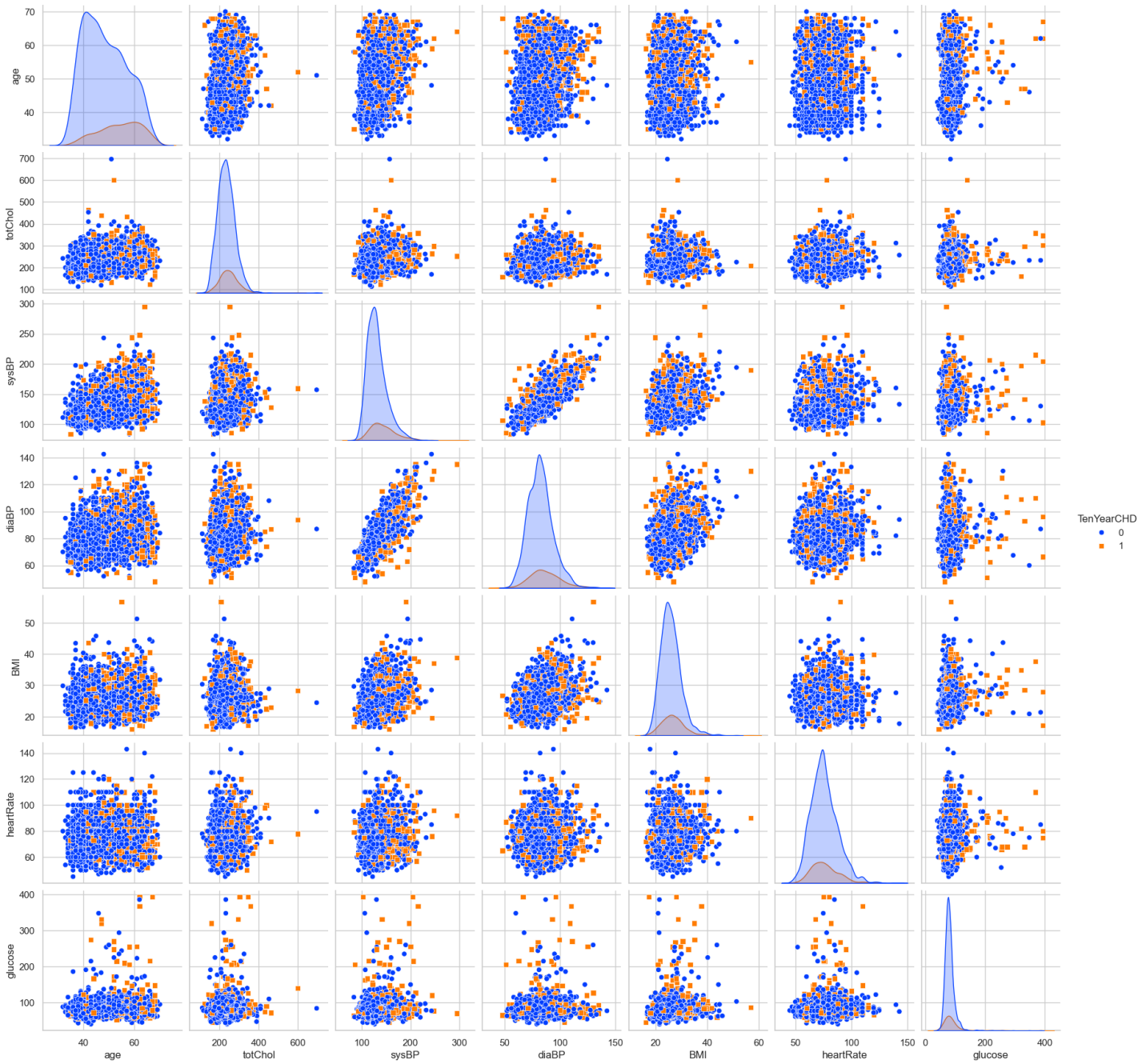
Pair plots

In [41]:

```
# Plotting pairplot of top features vs target variable
sns.pairplot(df, hue = 'TenYearCHD', markers=["o", "s"], vars = embeded_rf_feature, palette='bright')
```

Out[41]:

```
<seaborn.axisgrid.PairGrid at 0x1e0f933cb60>
```



Modelling and predicting with Machine Learning

Since our dataset is imbalanced i.e for every positive case there are about 5-6 negative cases. We may end up with a classifier that is biased to the negative cases. The classifier may have a high accuracy but poor a precision and recall.

To handle this problem we will balance the dataset using the Synthetic Minority Oversampling Technique (SMOTE)

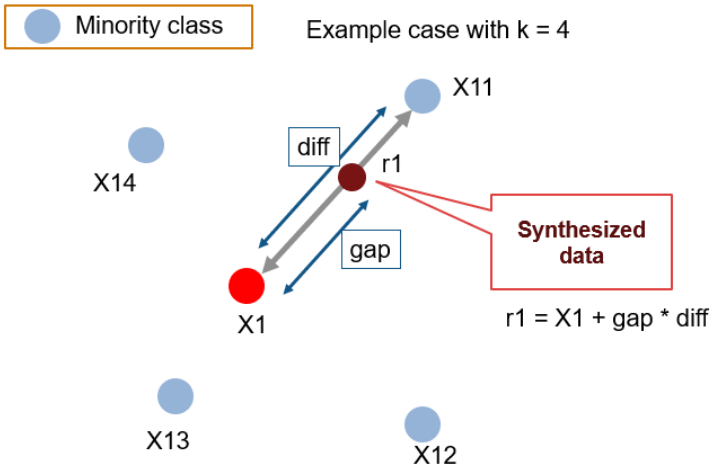
(SMOTE).

SMOTE :Synthetic Minority Oversampling Technique

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. This algorithm helps to overcome the overfitting problem posed by random oversampling. It focuses on the feature space to generate new instances with the help of interpolation between the positive instances that lie together.

Working Procedure:

> **At first the total no. of oversampling observations, N is set up. Generally, it is selected such that the binary class distribution is 1:1. But that could be tuned down based on need. Then the iteration starts by first selecting a positive class instance at random. Next, the KNN's (by default 5) for that instance is obtained. At last, N of these K instances is chosen to interpolate new synthetic instances. To do that, using any distance metric the difference in distance between the feature vector and its neighbors is calculated. Now, this difference is multiplied by any random value in (0,1] and is added to the previous feature vector. This is pictorially represented below:**



SMOTE algorithm works in 4 simple steps:

- **Choose a minority class as the input vector**
- **Find its k nearest neighbors (k_neighbors is specified as an argument in the SMOTE() function)**
- **Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor**
- **Repeat the steps until data is balanced**

In [42]:

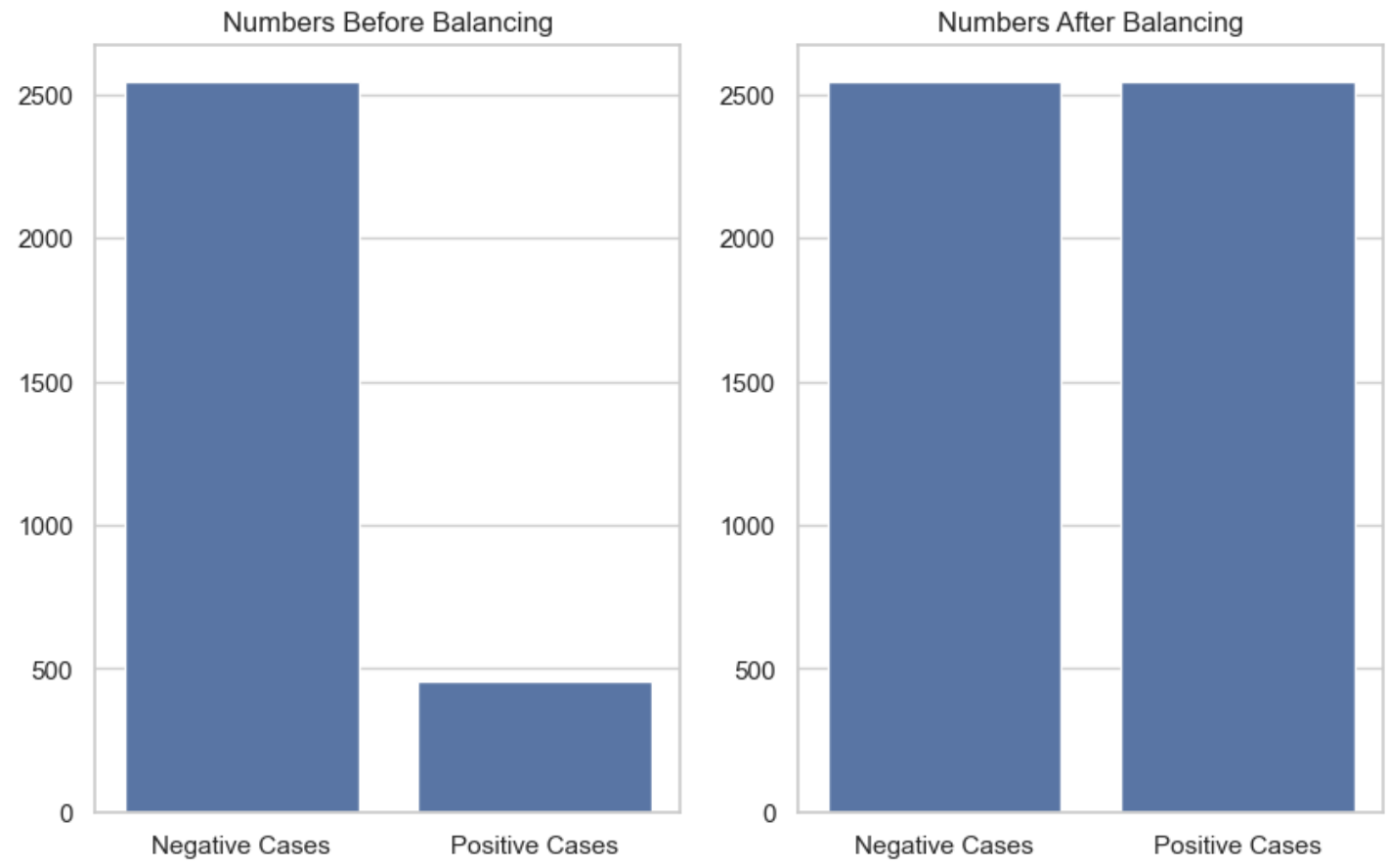
```
from imblearn.over_sampling import SMOTE

smote = SMOTE()

X = df[embeded_rf_feature]
y = df.iloc[:,-1]
# fit predictor and target variable
x_smote, y_smote = smote.fit_resample(X, y)

print('Original dataset shape', len(df))
print('Resampled dataset shape', len(y_smote))
```

```
Original dataset shape 3004
Resampled dataset shape 5094
```

In [43]:

```
from collections import Counter
labels = ["Negative Cases","Positive Cases"]
plt.figure(figsize=(10,6))
plt.subplot(1,2,1)
sns.barplot(x=labels, y=list(dict(Counter(y)).values()))
```

```
plt.title("Numbers Before Balancing")
plt.subplot(1,2,2)
sns.barplot(x=labels,y= list(dict(Counter(y_smote)).values()))
plt.title("Numbers After Balancing")
plt.show()
```



As seen after applying SMOTE, the new dataset is much more balanced.

Splitting the data to Training and Testing set

In [44]:

```
# First let's create our new dataset
df_new = pd.concat([pd.DataFrame(x_smote), pd.DataFrame(y_smote)], axis=1)
df_new.columns = ['age', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose','Ten
YearCHD']
df_new.head()
```

Out[44]:

	age	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
0	36	212.0	168.0	98.0	29.77	72.0	75.0	0
1	46	250.0	116.0	71.0	20.35	88.0	94.0	0
2	50	233.0	158.0	88.0	28.26	68.0	94.0	1
3	64	241.0	136.5	85.0	26.42	70.0	77.0	0
4	61	272.0	182.0	121.0	32.80	85.0	65.0	1

In [45]:

```
X_new = df_new[embeded_rf_feature]
y_new = df_new["TenYearCHD"]

X_train,X_test,Y_train,Y_test = train_test_split(X_new,y_new,test_size=0.2,random_state=
42)
print("Training features have {0} records and Testing features have {1} records.".\
      format(X_train.shape[0], X_test.shape[0]))
```

```
Training features have 4075 records and Testing features have 1019 records.
```

Models:

The four algorithms that we will be using are:

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

Here, we will be using GridsearchCV search algorithm for above algorithms

1. **Logistic Regression**

Logistic regression aims to measure the relationship between a categorical dependent variable and one or more independent variables (usually continuous) by plotting the dependent variables' probability scores.

In [46]:

```
# Importing required libraries
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC


from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score,precision_score,classification_report,roc_auc_sc
ore,roc_curve
```

In [47]:

```
# search for optimun parameters using gridsearch
params = {'penalty':['l1','l2'],
          'C' : [1e-6,1e-5,1e-4,1e-3,1e-2,1e-1,1,10,100,1e-3,1e+4,1e+5,1e+6],
          'class_weight':['balanced',None]}
logistic_clf = GridSearchCV(LogisticRegression(),param_grid=params,cv=10, scoring='roc_a
uc')
```

In [48]:

```
#training the classifier
logistic_clf.fit(X_train,Y_train)

logistic_clf.best_params_
```

Out[48]:

```
{'C': 0.0001, 'class_weight': 'balanced', 'penalty': 'l2'}
```

In [49]:

```
#making predictions
logistic_predict = logistic_clf.predict(X_test)
```

In [50]:

```
logistic_accuracy = accuracy_score(Y_test,logistic_predict)
print(f"Using logistic regression we get an accuracy of {round(logistic_accuracy*100,2)}%
")
```

```
Using logistic regression we get an accuracy of 67.71%
```

In [51]:

```
print('Train ROC-AUC score : ', logistic_clf.best_estimator_.score(X_train,Y_train))
print('Test ROC-AUC score : ', logistic_clf.best_estimator_.score(X_test,Y_test))
```

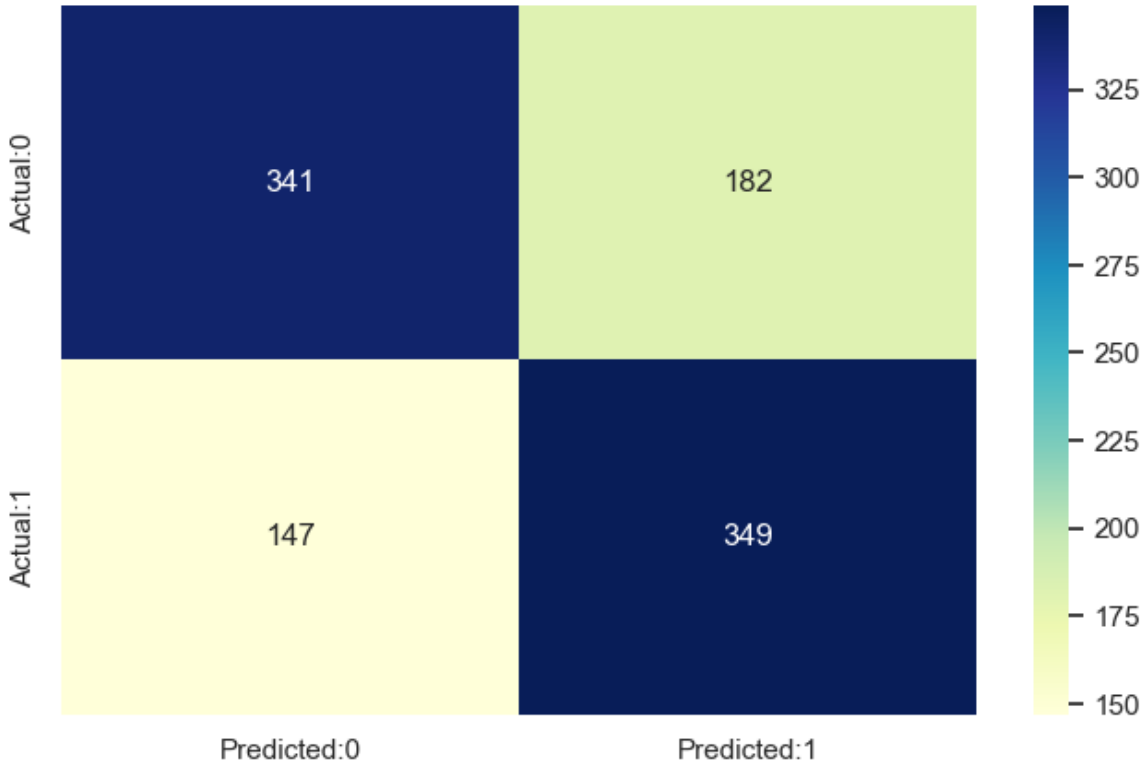
```
Train ROC-AUC score :  0.6633128834355828
Test ROC-AUC score :  0.677134445534838
```

In [52]:

```
# confusion matrix of Logistic Model
cm=confusion_matrix(Y_test,logistic_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0'
,'Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[52]:

```
<Axes: >
```



In [53]:

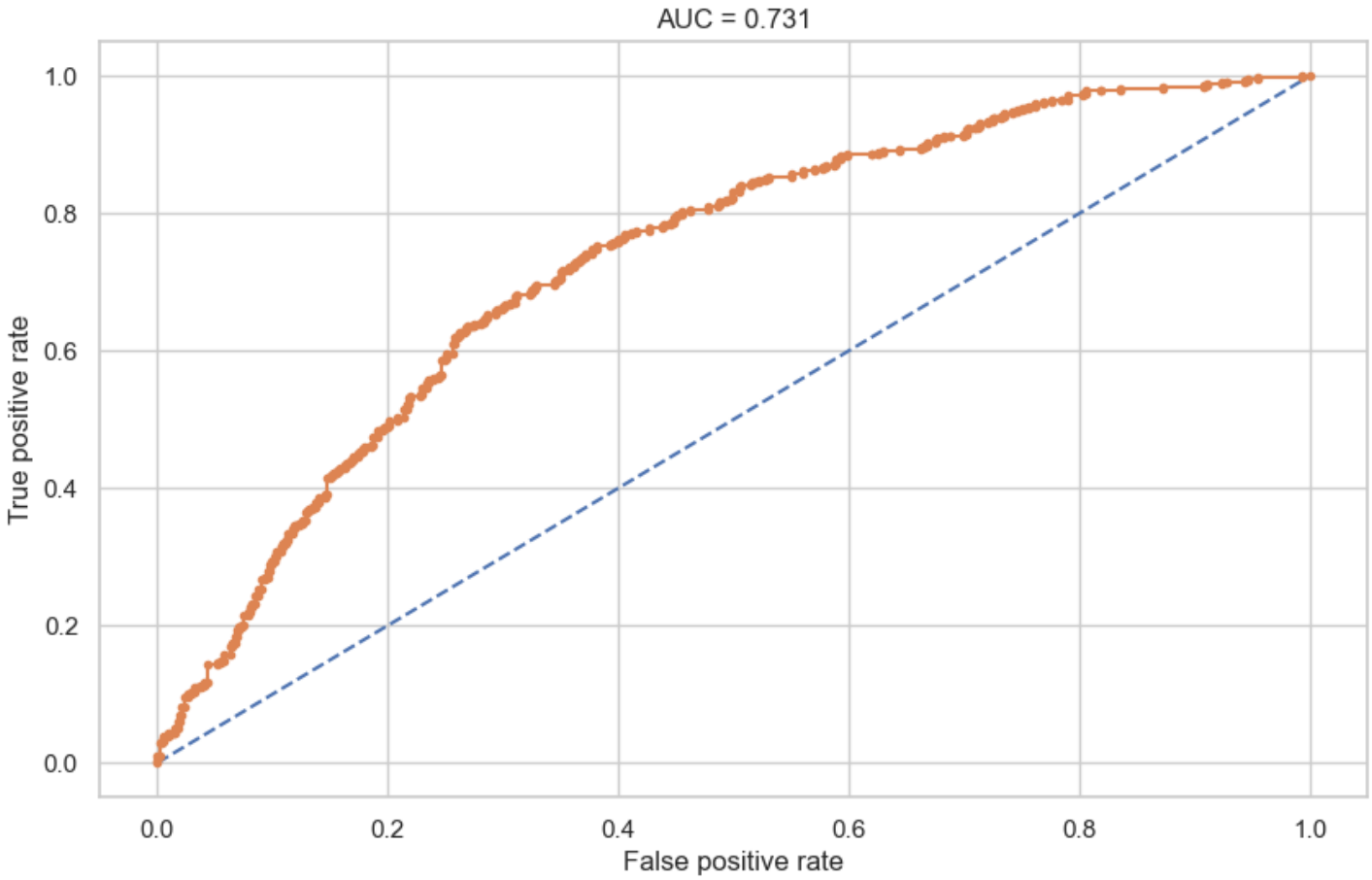
```
print(classification_report(Y_test,logistic_predict))
```

```
              precision    recall  f1-score   support

           0       0.70      0.65      0.67       523
           1       0.66      0.70      0.68       496

    accuracy                           0.68      1019
   macro avg       0.68      0.68      0.68      1019
weighted avg       0.68      0.68      0.68      1019
```

In [54]:

```
# ROC curve and AUC
probs = logistic_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]
# calculate AUC
log_auc = roc_auc_score(Y_test, probs)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(Y_test, probs)
```

```
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(log_auc,3)}")
plt.show()
```



1. **Random Forest**

Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

In [55]:

```
# search for optimun parameters using gridsearch
params_rf = {
    'max_depth': [4, 6, 8],
    'min_samples_leaf': [40, 50],
    'min_samples_split': [50, 100, 150],
    'n_estimators': [50, 80, 100]
  }

random_clf = GridSearchCV(RandomForestClassifier(),param_grid=params_rf,cv=10, scoring='
roc_auc')
```

In [56]:

```
#training the classifier
random_clf.fit(X_train,Y_train)

random_clf.best_params_
```

Out[56]:

```
{'max_depth': 8,
 'min_samples_leaf': 40,
```

```
 'min_samples_split': 50,
 'n_estimators': 100}
```

In [57]:

```
#making predictions
random_predict = random_clf.predict(X_test)
```

In [58]:

```
random_accuracy = accuracy_score(Y_test,random_predict)
print(f"Using Random Forest we get an accuracy of {round(random_accuracy*100,2)}%")
```

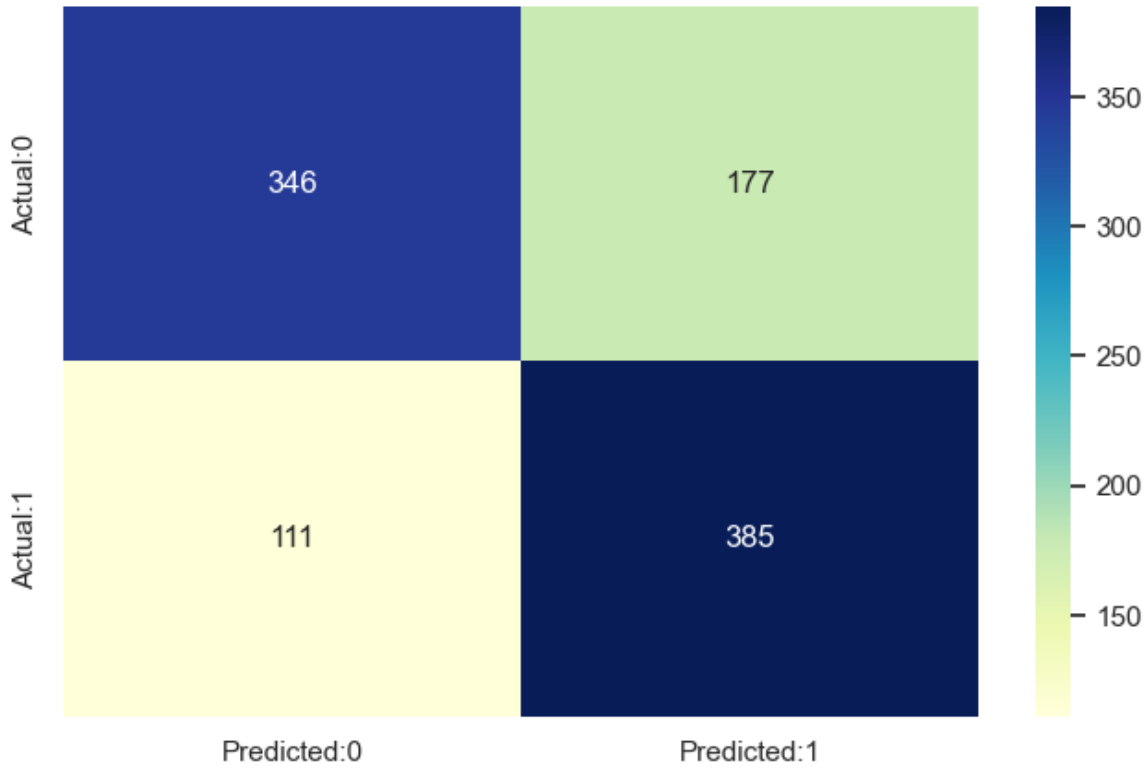
```
Using Random Forest we get an accuracy of 71.74%
```

In [59]:

```
# confusion matrix of Random Forest
cm=confusion_matrix(Y_test,random_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0'
,'Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[59]:

```
<Axes: >
```



In [60]:

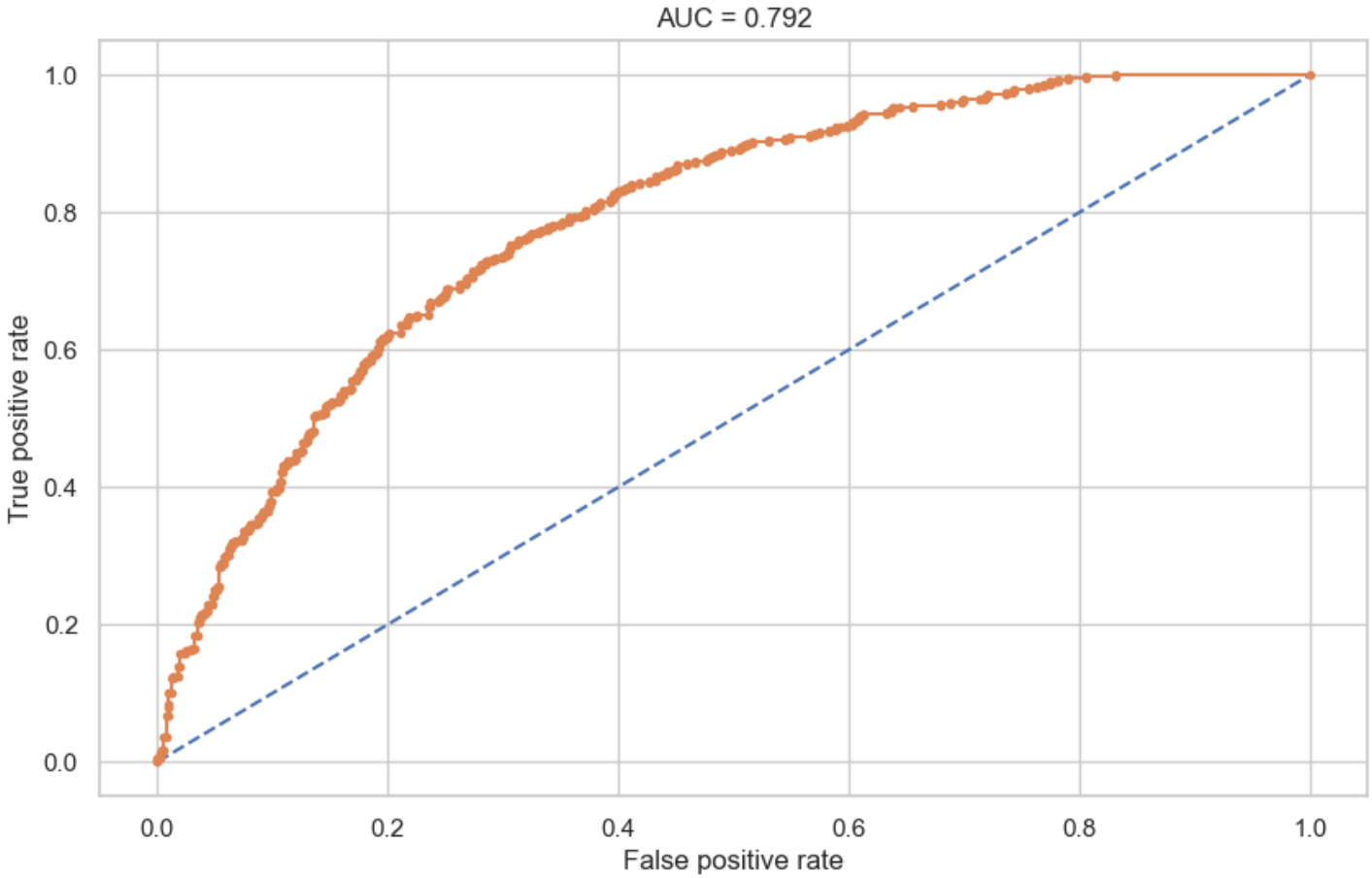
```
print(classification_report(Y_test,random_predict))
```

```
              precision    recall  f1-score   support

           0       0.76      0.66      0.71       523
           1       0.69      0.78      0.73       496

    accuracy                           0.72      1019
   macro avg       0.72      0.72      0.72      1019
weighted avg       0.72      0.72      0.72      1019
```

In [61]:

```
# ROC curve and AUC
```

```
probs1 = random_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs1 = probs1[:, 1]
# calculate AUC
ran_auc = roc_auc_score(Y_test, probs1)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(Y_test, probs1)
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(ran_auc,3)}")
plt.show()
```



1. **XGBoost**

XGBoost stands for eXtreme Gradient Boosting. The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms

In [62]:

```
params_xgb = {
    'max_depth': range (2, 12, 1),
    'n_estimators': range(60, 220, 20),
    'learning_rate': [0.1, 0.05, 0.01, 0.005]
  }

xgb_clf = GridSearchCV(XGBClassifier(), param_grid = params_xgb, cv = 10, scoring='roc_auc')
```

In [63]:

```
#training the classifier
xgb_clf.fit(X_train,Y_train)

xgb_clf.best_params_
```

Out[63]:

```
{'learning_rate': 0.1, 'max_depth': 11, 'n_estimators': 200}
```

In [64]:

```
#making predictions
xgb_predict = xgb_clf.predict(X_test)
```

In [65]:

```
xgb_accuracy = accuracy_score(Y_test,xgb_predict)
print(f"Using XG boost we get an accuracy of {round(xgb_accuracy*100,2)}%")
```

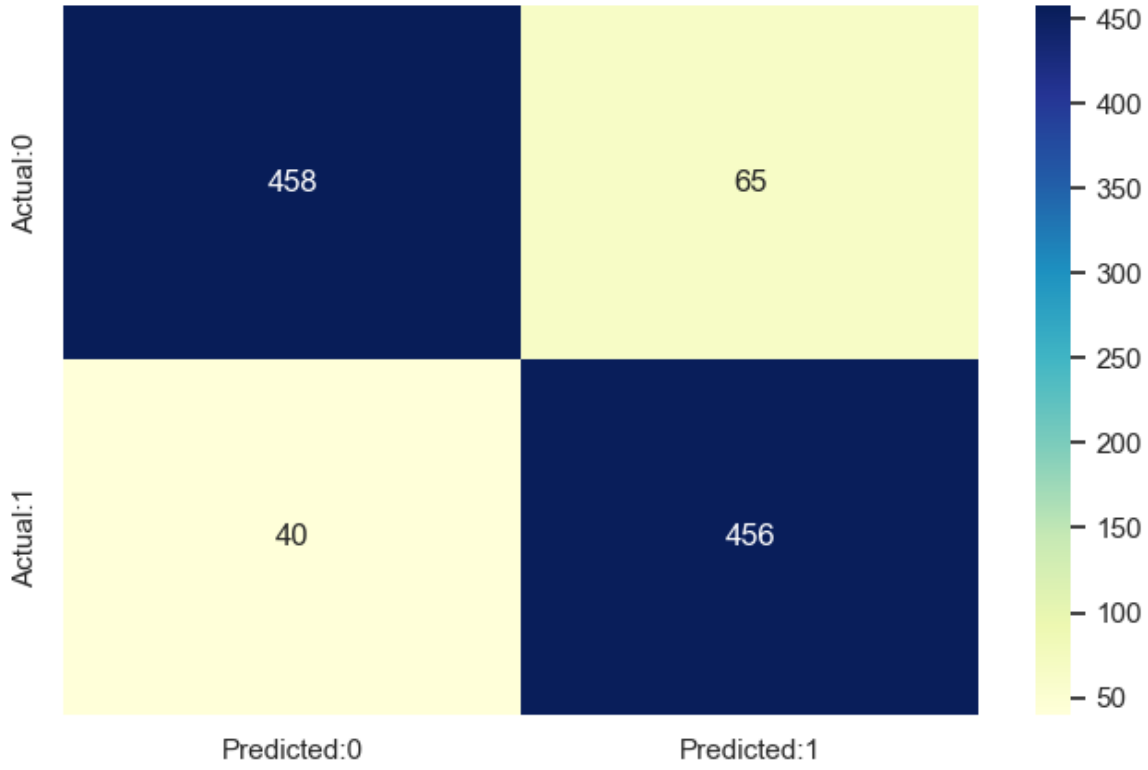
```
Using XG boost we get an accuracy of 89.7%
```

In [66]:

```
# confusion matrix of XG boost Classifier
cm=confusion_matrix(Y_test,xgb_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0'
,'Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[66]:

```
<Axes: >
```



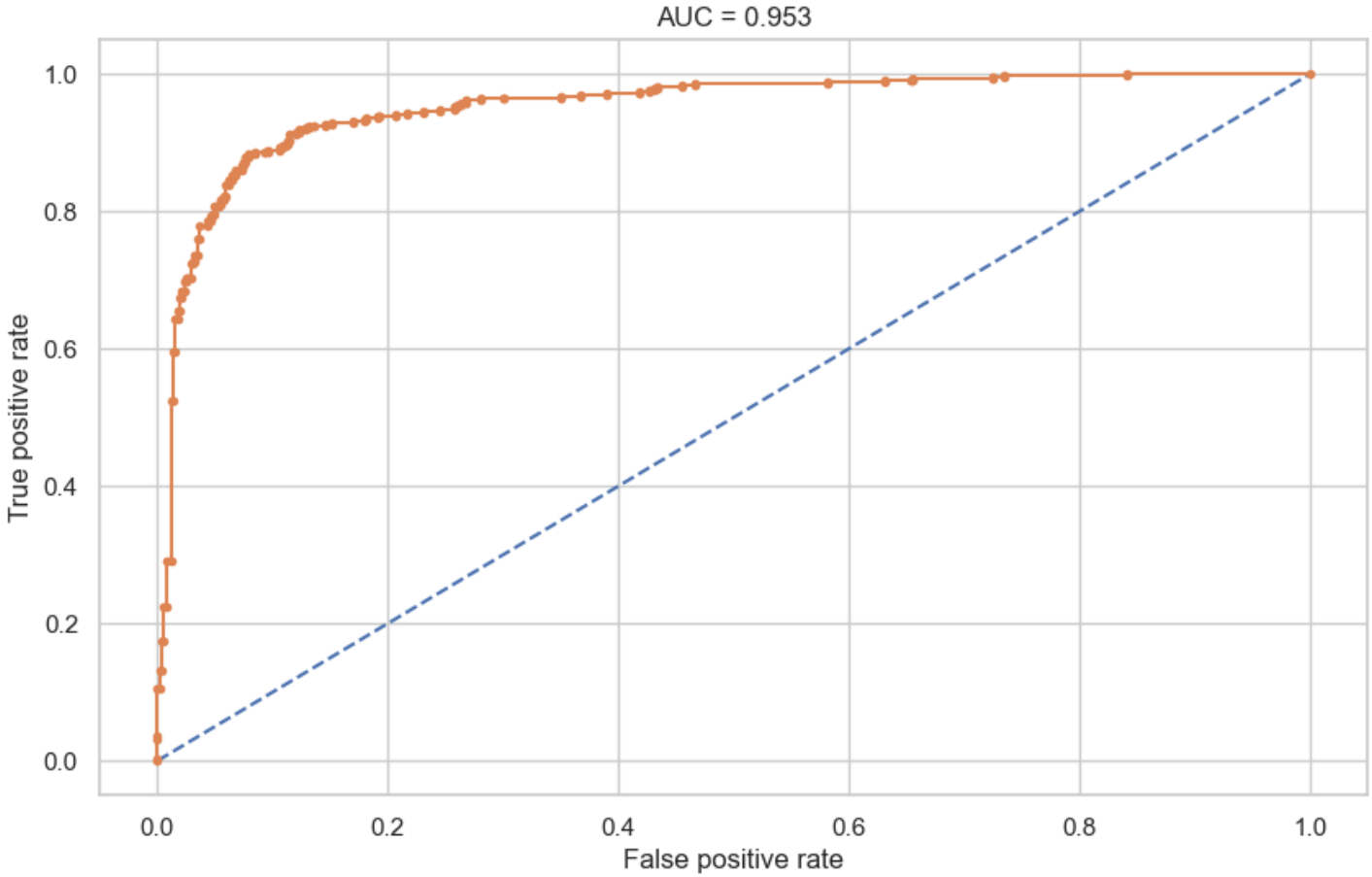
In [67]:

```
print(classification_report(Y_test, xgb_predict))
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90       523
           1       0.88      0.92      0.90       496

    accuracy                           0.90      1019
   macro avg       0.90      0.90      0.90      1019
weighted avg       0.90      0.90      0.90      1019
```

In [68]:

```
# ROC curve and AUC
probs2 = xgb_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs2 = probs2[:, 1]
# calculate AUC
xgb_auc = roc_auc_score(Y_test, probs2)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(Y_test, probs2)
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(xgb_auc,3)}")
plt.show()
```



1. **Support Vector Machine**

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems.

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

In [69]:

```
# Grid search for optimum parameters
Cs = [0.001, 0.01, 0.1, 1, 10]
gammas = [0.001, 0.01, 0.1, 1]
param_grid = {'C': Cs, 'gamma' : gammas}
svm_clf = GridSearchCV(SVC(kernel='rbf', probability=True), param_grid, cv=10)
```

In [70]:

```
#training the classifier
svm_clf.fit(X_train,Y_train)

svm_clf.best_params_
```

Out[70]:

```
{'C': 10, 'gamma': 0.01}
```

In [71]:

```
#making predictions
svm_predict = svm_clf.predict(X_test)
```

In [72]:

```
svm_accuracy = accuracy_score(Y_test,svm_predict)
print(f"Using Support Vector Machine we get an accuracy of {round(svm_accuracy*100,2)}%")
```

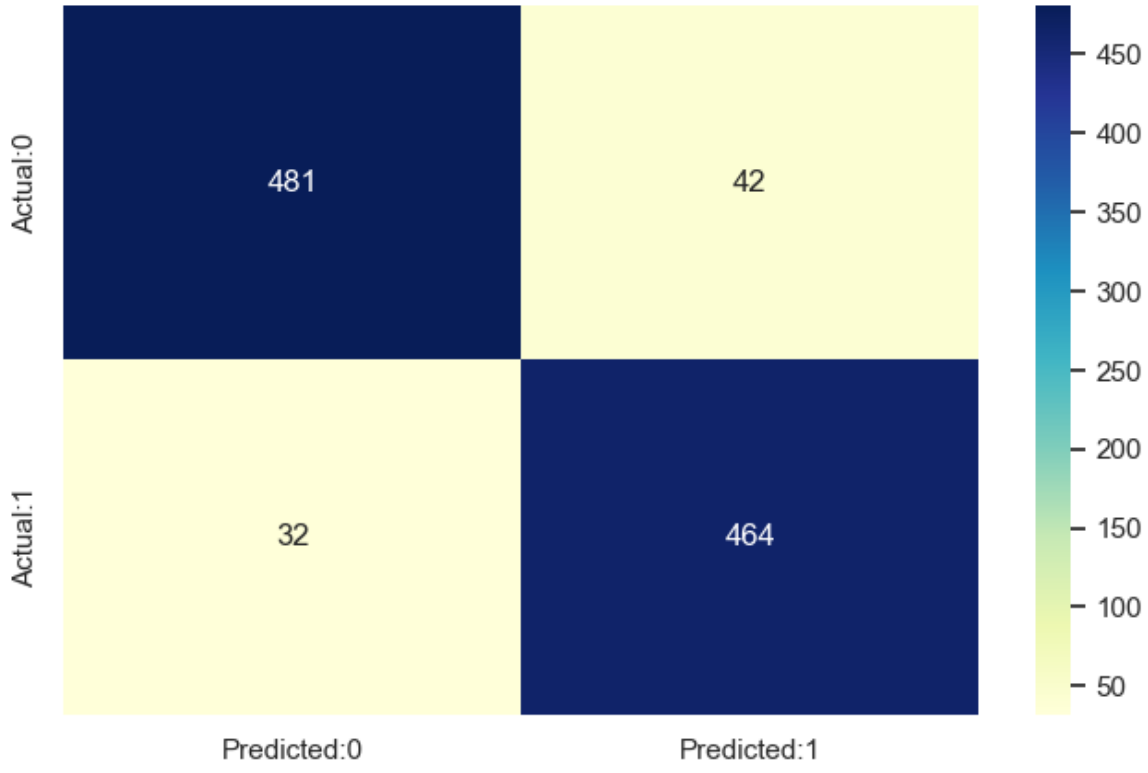
```
Using Support Vector Machine we get an accuracy of 92.74%
```

In [73]:

```
# confusion matrix of SVM
cm=confusion_matrix(Y_test,svm_predict)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0'
,'Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[73]:

```
<Axes: >
```



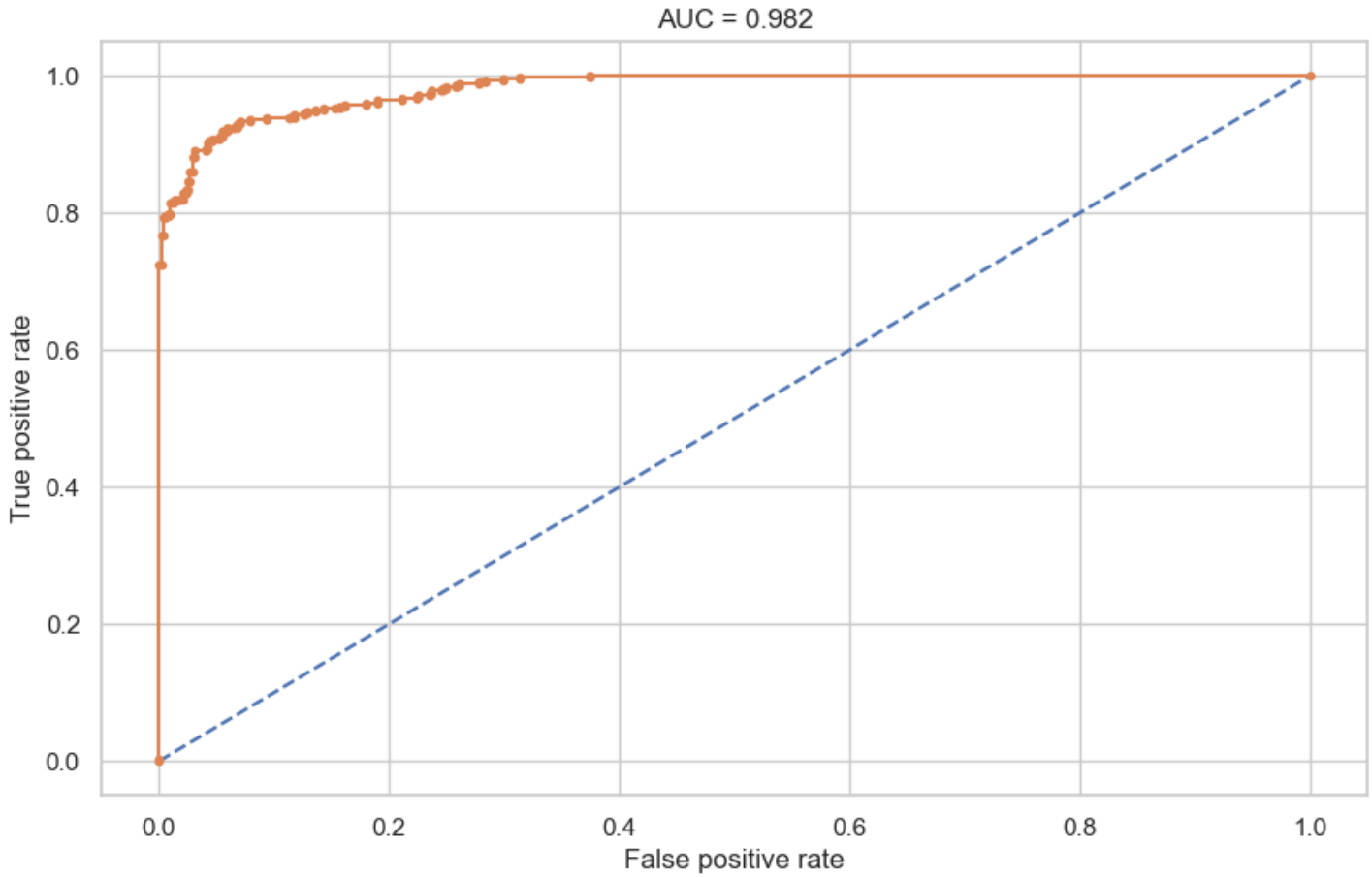
In [74]:

```
print(classification_report(Y_test, svm_predict))
```

```
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       523
           1       0.92      0.94      0.93       496

    accuracy                           0.93      1019
   macro avg       0.93      0.93      0.93      1019
weighted avg       0.93      0.93      0.93      1019
```

In [75]:

```
# ROC curve and AUC
probs3 = svm_clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs3 = probs3[:, 1]
# calculate AUC
svc_auc = roc_auc_score(Y_test, probs3)

# calculate roc curve
fpr, tpr, thresholds = roc_curve(Y_test, probs3)
# plot curve
sns.set_style('whitegrid')
plt.figure(figsize=(10,6))
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(fpr, tpr, marker='.')
plt.ylabel('True positive rate')
plt.xlabel('False positive rate')
plt.title(f"AUC = {round(svc_auc,3)}")
plt.show()
```



Lets collect all our best models

In [76]:

```
# Creating dataframe which shows the performance metrics of each model
Performance_df = pd.DataFrame({
    "Logistic regression":{'Test Accuracy':round(logistic_accuracy, 2),'Precision': roun
d(precision_score(Y_test, logistic_predict), 2),'Recall': round(recall_score(Y_test, log
istic_predict), 2),'F1 Score': round(f1_score(Y_test, logistic_predict), 2), 'AUC':round
(log_auc, 2)},
    "Random Forest":{'Test Accuracy':round(random_accuracy, 2),'Precision': round(precis
ion_score(Y_test, random_predict), 2),'Recall': round(recall_score(Y_test, random_predic
t), 2),'F1 Score': round(f1_score(Y_test, random_predict), 2), 'AUC':round(ran_auc, 2)},
    "XG Boost":{'Test Accuracy':round(xgb_accuracy, 2),'Precision': round(precision_scor
e(Y_test, xgb_predict), 2),'Recall': round(recall_score(Y_test, xgb_predict), 2),'F1 Sco
re': round(f1_score(Y_test, xgb_predict), 2), 'AUC':round(xgb_auc, 2)},
    "Support vector machine":{'Test Accuracy':round(svm_accuracy, 2),'Precision': round(
```

```
precision_score(Y_test, svm_predict), 2),'Recall': round(recall_score(Y_test, svm_predic
t), 2),'F1 Score': round(f1_score(Y_test, svm_predict), 2), 'AUC':round(svc_auc, 2)}
}).T
```

In [77]:

```
Performance_df
```

Out[77]:

	Test Accuracy	Precision	Recall	F1 Score	AUC
Logistic regression	0.68	0.66	0.70	0.68	0.73
Random Forest	0.72	0.69	0.78	0.73	0.79
XG Boost	0.90	0.88	0.92	0.90	0.95
Support vector machine	0.93	0.92	0.94	0.93	0.98

Observation from above table:

- **XG Boost, Support vector machine** gives highest Accuracy, Recall, Precision and AUC score.
- Highest recall is given by **Support vector machine**
- Highest AUC is given by **Support vector machine**

Overall we can say that Support vector machine is the best model that can be used for the risk prediction of Cardiovasular heart disease.

Let's Plot the Accuracy and AUC score graph of each algorithm

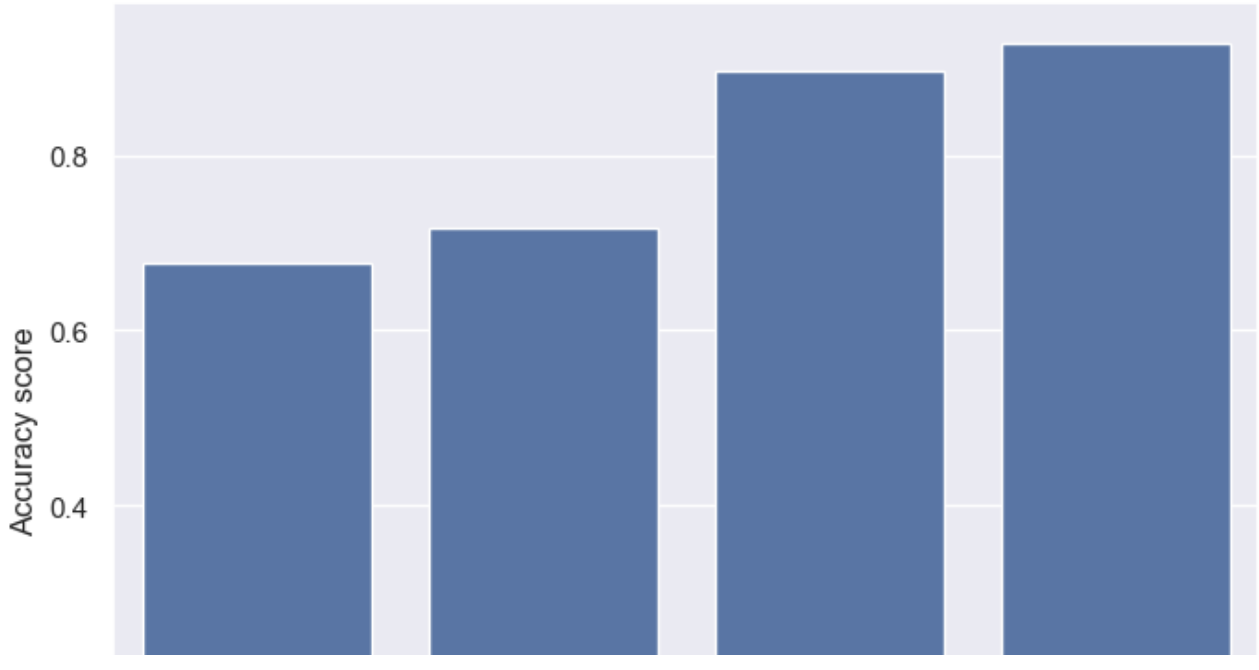
Accuracy Score plot

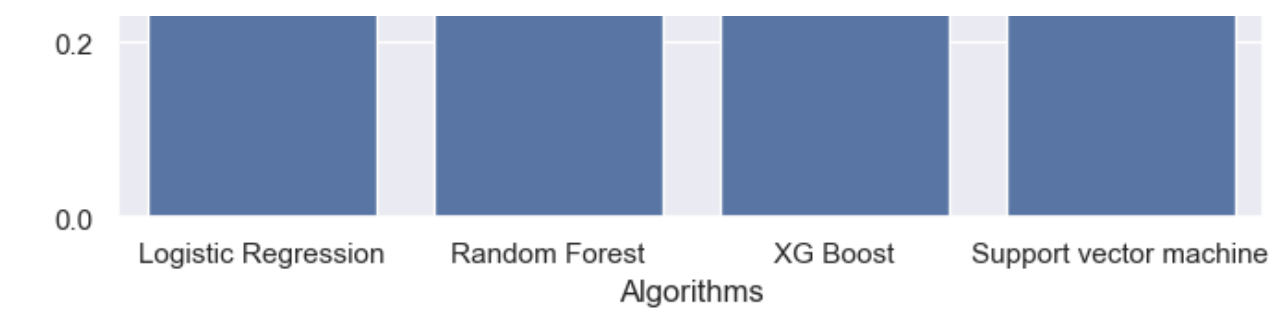
In [78]:

```
# Storing accuracies of each algorithm in a list
scores = [logistic_accuracy,random_accuracy,xgb_accuracy,svm_accuracy]
# Naming the algorithms and storing in a list
algorithms = ["Logistic Regression","Random Forest","XG Boost","Support vector machine"]
# Visualize the algorithms
sns.set(rc={'figure.figsize':(8,6)})
plt.xlabel("Algorithms")
plt.ylabel("Accuracy score")
sns.barplot(x=algorithms,y=scores)
```

Out[78]:

```
<Axes: xlabel='Algorithms', ylabel='Accuracy score'>
```





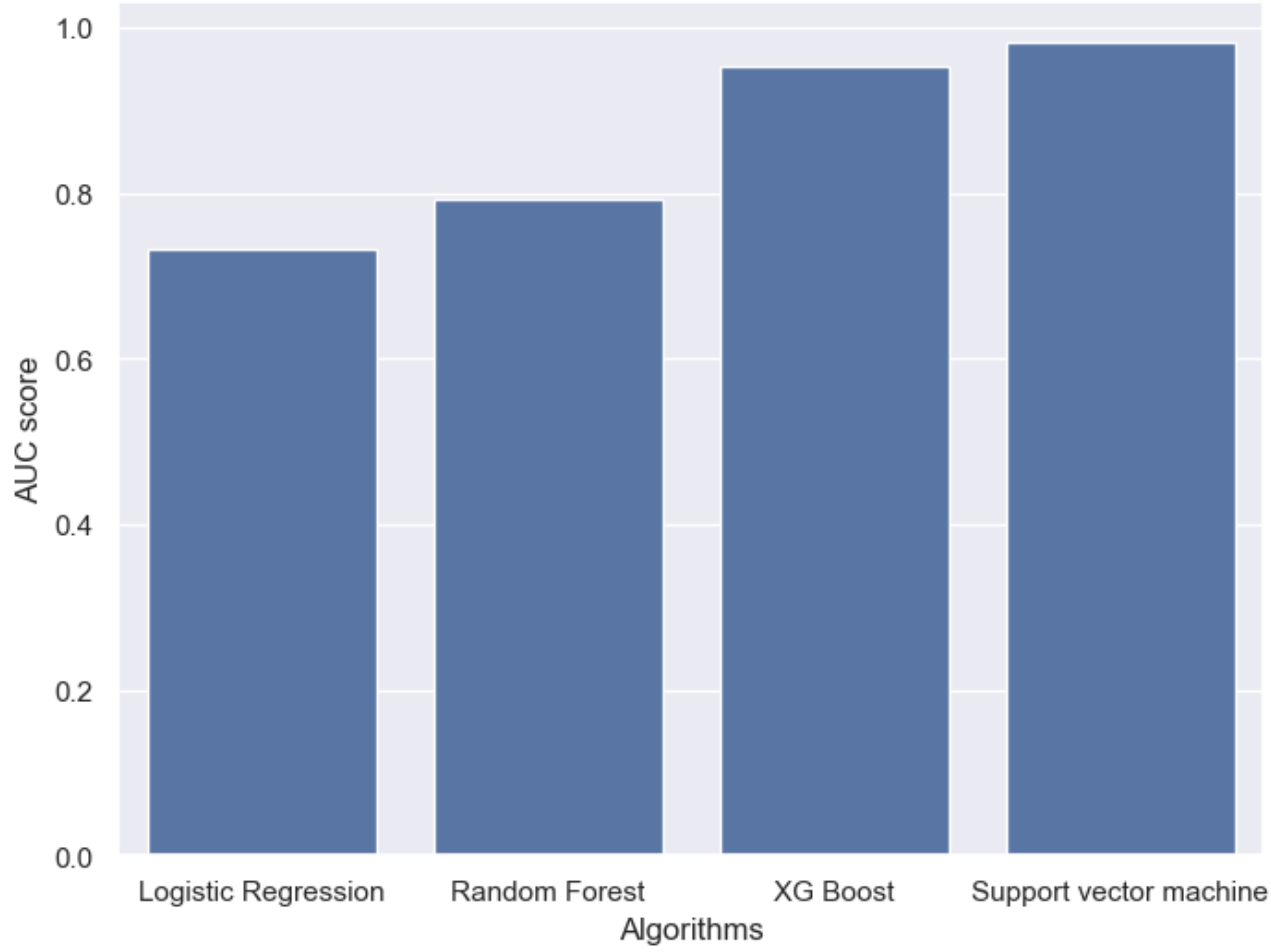
AUC Score plot

In [79]:

```
# Storing AUC score of each algorithm in a list
auc_scores = [log_auc,ran_auc,xgb_auc,svc_auc]
# Naming the algorithms and storing in a list
algorithms = ["Logistic Regression","Random Forest","XG Boost","Support vector machine"]
# Visualize the algorithms
sns.set(rc={'figure.figsize':(8,6)})
plt.xlabel("Algorithms")
plt.ylabel("AUC score")
sns.barplot(x=algorithms,y=auc_scores)
```

Out[79]:

```
<Axes: xlabel='Algorithms', ylabel='AUC score'>
```



From both the graphs we can say that the best performing model is Support Vector Machine algorithm.

Conclusion:

- **The people who have Cardiovascular heart disease is almost equal between smokers and non smokers.**
- **The top features in predicting the ten year risk of developing Cardiovasular Heart Disease are 'age', 'totChol', 'sysBP', 'diaBP', 'BMI', 'heartRate', 'glucose'.**

- **The Support vector machine with the radial kernel is the best performing model in terms of accuracy and the F1 score and Its high AUC-score shows that it has a high true positive rate.**
- **Balancing the dataset by using the SMOTE technique helped in improving the models' sensitivity.**
- **With more data(especially that of the minority class) better models can be built.**

In [ ]:

In [ ]:

In [ ]:

In [ ]: