Heart Deisease Prediction:

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn

%matplotlib inline

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

Reading Dataset:

In [2]:

```
data = pd.read_csv("framingham.csv")
```

In [3]:

```
data.shape
```

Out[3]:

```
(4240, 16)
```

There are 4240 records and 16 attributes

Displaying Columns:

In [4]:

```
data.columns
```

Out[4]:

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

In [5]:

```
data
```

Out[5]:

	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0
...	...	...	...	...	...	...	...	...	...	...	...
4235	0	48	2.0	1	20.0	NaN	0	0	0	248.0	131.0
4236	0	44	1.0	1	15.0	0.0	0	0	0	210.0	126.5

	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP
4237	0	52	2.0	0	0.0	0.0	0	0	0	269.0	133.5
4238	1	40	3.0	0	0.0	0.0	0	1	0	185.0	141.0
4239	0	39	3.0	1	30.0	0.0	0	0	0	196.0	133.0

4240 rows × 16 columns

Displaying top 5 rows:

In [6]:

```
data.head()
```

Out[6]:

	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	di
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	

Displaying Last 5 rows:

In [7]:

```
data.tail()
```

Out[7]:

	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP
4235	0	48	2.0	1	20.0	NaN	0	0	0	248.0	131.0
4236	0	44	1.0	1	15.0	0.0	0	0	0	210.0	126.5
4237	0	52	2.0	0	0.0	0.0	0	0	0	269.0	133.5
4238	1	40	3.0	0	0.0	0.0	0	1	0	185.0	141.0
4239	0	39	3.0	1	30.0	0.0	0	0	0	196.0	133.0

Checking Duplicate values:

In [8]:

```
duplicate_df = data[data.duplicated()]
duplicate_df
```

Out[8]:

Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	dia

As we see there are no Duplicate values

Handling Null Values:

In [9]:

```
data.isna().sum()
```

Out[9]:

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

In [10]:

```
# Percentage of null values present:

data.isna().sum()/data.shape[0]*100
```

Out[10]:

```
Gender             0.000000
age                0.000000
education          2.476415
currentSmoker      0.000000
cigsPerDay         0.683962
BPMeds             1.250000
prevalentStroke    0.000000
prevalentHyp       0.000000
diabetes           0.000000
totChol            1.179245
sysBP              0.000000
diaBP              0.000000
BMI                0.448113
heartRate          0.023585
glucose            9.150943
TenYearCHD         0.000000
dtype: float64
```

By the above data we can say that:

1. education has 2.47% of null values
2. cigsPerDay has 0.68% of null values
3. BPMeds has 1.25% of null values
4. totChol has 1.17% of null values
5. BMI has 0.44% of null values
6. heartRate has 0.02% of null values
7. glucose has 9.15% of null values

Total number of rows with missing values is 489.

since it is only 12 percent of the entire dataset the rows with missing values are excluded.

In [11]:

```
data.fillna(data.mean(), inplace=True)
data.isna().sum()
```

Out[11]:

```
Gender              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
```

Now there are no null values

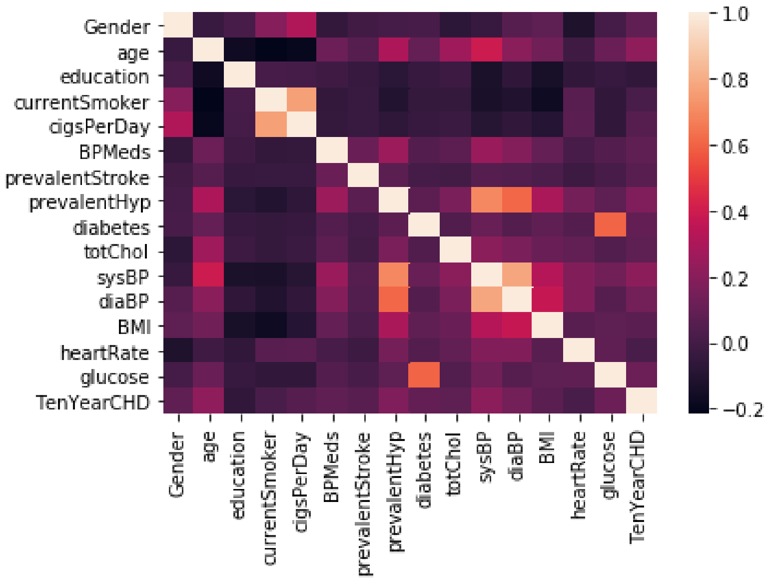
Outliers:

In [14]:

```
corr=data.corr()
sn.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns)
```

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23295c945c8>
```



In [17]:

```
columns = corr.index[abs(corr["TenYearCHD"])>0.1]
columns
```

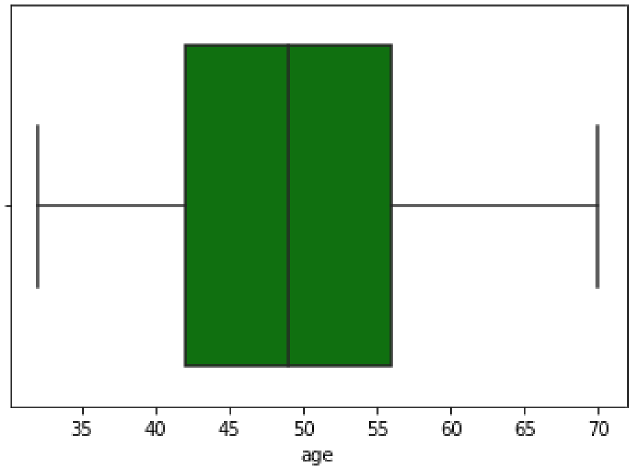
Out[17]:

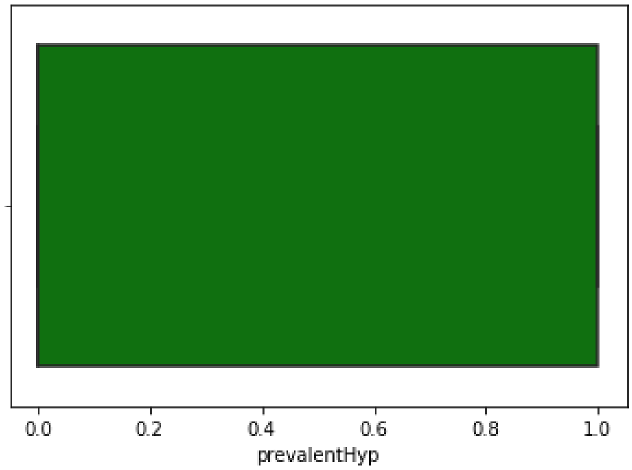
```
Index(['age', 'prevalentHyp', 'sysBP', 'diaBP', 'glucose', 'TenYearCHD'], dtype='object')
```

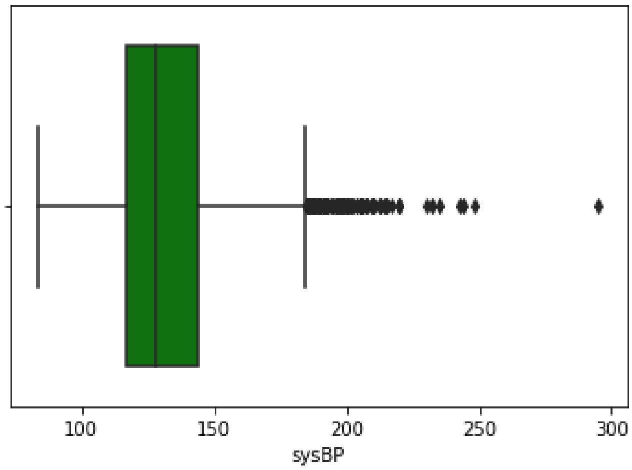
In [18]:

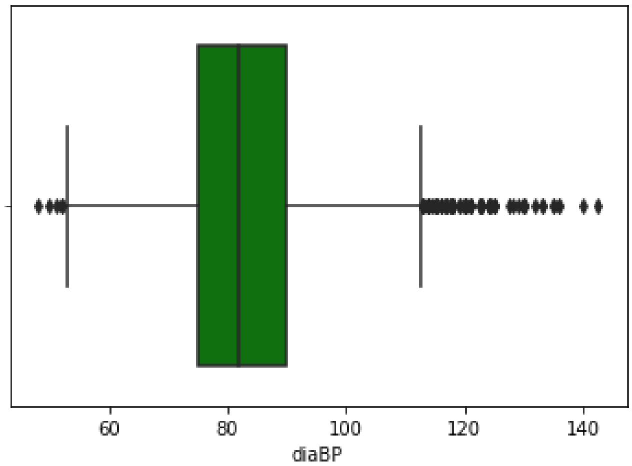
```
# Visualizing Outliers using Boxplot:

plt.title("Outliers")
for i in columns:
    sn.boxplot(data[i],color='green')
    plt.show()
```

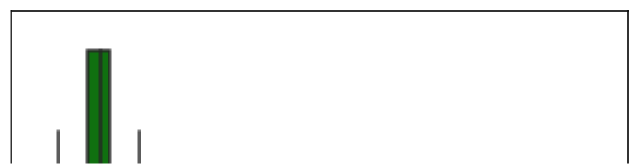
Outliers

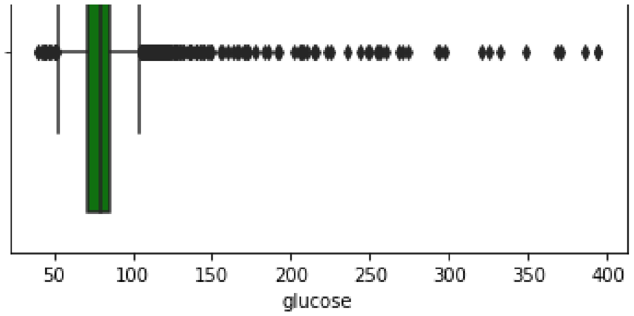

35 40 45 50 55 60 65 70
age

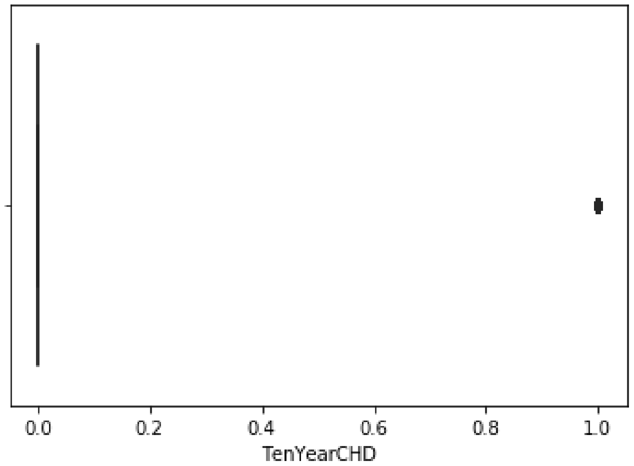

0.0 0.2 0.4 0.6 0.8 1.0
prevalentHyp


100 150 200 250 300
sysBP


60 80 100 120 140
diaBP







In [19]:

```
#Zscore Values

from scipy import stats

list1 = []

for i in columns:
    z = np.abs(stats.zscore(data[i]))
    list1.append(np.where(z>3)[0])
list1
```

Out[19]:

```
[array([], dtype=int64),
 array([], dtype=int64),
 array([  44,   87,  153,  249,  333,  369,  446,  481,  590,  664,  833,
         864,  903,  924,  932, 1003, 1079, 1189, 1520, 1567, 1588, 1751,
        1878, 1989, 2091, 2132, 2162, 2179, 2304, 2416, 2459, 2608, 2645,
        2683, 2909, 2930, 3062, 3214, 3489, 3554, 3616, 3675, 3844, 3981,
        4040, 4076, 4123, 4173], dtype=int64),
 array([  28,  158,  249,  407,  409,  423,  446,  481,  531,  833,  864,
        1189, 1608, 1614, 1751, 1760, 1808, 1878, 1989, 2088, 2093, 2179,
        2307, 2416, 2608, 2657, 2683, 2872, 3062, 3322, 3489, 3532, 3554,
        3635, 3953, 3981, 4040, 4075, 4076, 4173, 4228], dtype=int64),
 array([  37,   44,   66,  443,  763,  952,  965, 1068, 1238, 1268, 1363,
        1456, 1485, 1649, 1674, 1931, 1997, 2091, 2234, 2378, 2406, 2498,
        2503, 2649, 2801, 2855, 2891, 2893, 2909, 2961, 3002, 3112, 3327,
        3458, 3606, 3680, 3749, 3763, 3817, 3844, 3849, 3868, 3971, 3974,
        4042, 4064, 4076, 4084, 4096, 4228], dtype=int64),
 array([], dtype=int64)]
```

Removing Outliers using zscore method

In [20]:

```
data_final = data[(z < 3)].copy()
```

In [21]:

```
data_final.shape
```

Out[21]:

```
(4240, 16)
```

After Removing Outliers, the shape of the given dataset will be 4199 rows and 16 columns

Removing Unwanted Columns:

As we see that There is only one column that is not nessecary i.e. Education because it doesnot decide any person's heart disease

In [22]:

```
data_final.drop(['education'],axis=1,inplace=True)
```

In [23]:

```
data_final.describe()
```

Out[23]:

	Gender	age	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	t
count	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	4240.000000	4240.
mean	0.429245	49.580189	0.494104	9.005937	0.029615	0.005896	0.310613	0.025708	236.
std	0.495027	8.572942	0.500024	11.881610	0.168481	0.076569	0.462799	0.158280	44.
min	0.000000	32.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	107.
25%	0.000000	42.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	206.
50%	0.000000	49.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	234.
75%	1.000000	56.000000	1.000000	20.000000	0.000000	0.000000	1.000000	0.000000	262.
max	1.000000	70.000000	1.000000	70.000000	1.000000	1.000000	1.000000	1.000000	696.

In [24]:

```
data_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4240 entries, 0 to 4239
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Gender           4240 non-null   int64  
 1   age              4240 non-null   int64  
 2   currentSmoker    4240 non-null   int64  
 3   cigsPerDay       4240 non-null   float64
 4   BPMeds           4240 non-null   float64
 5   prevalentStroke  4240 non-null   int64  
 6   prevalentHyp     4240 non-null   int64  
 7   diabetes         4240 non-null   int64  
 8   totChol          4240 non-null   float64
 9   sysBP            4240 non-null   float64
 10  diaBP            4240 non-null   float64
 11  BMI              4240 non-null   float64
 12  heartRate        4240 non-null   float64
 13  glucose          4240 non-null   float64
 14  TenYearCHD       4240 non-null   int64  
dtypes: float64(8), int64(7)
memory usage: 530.0 KB
```

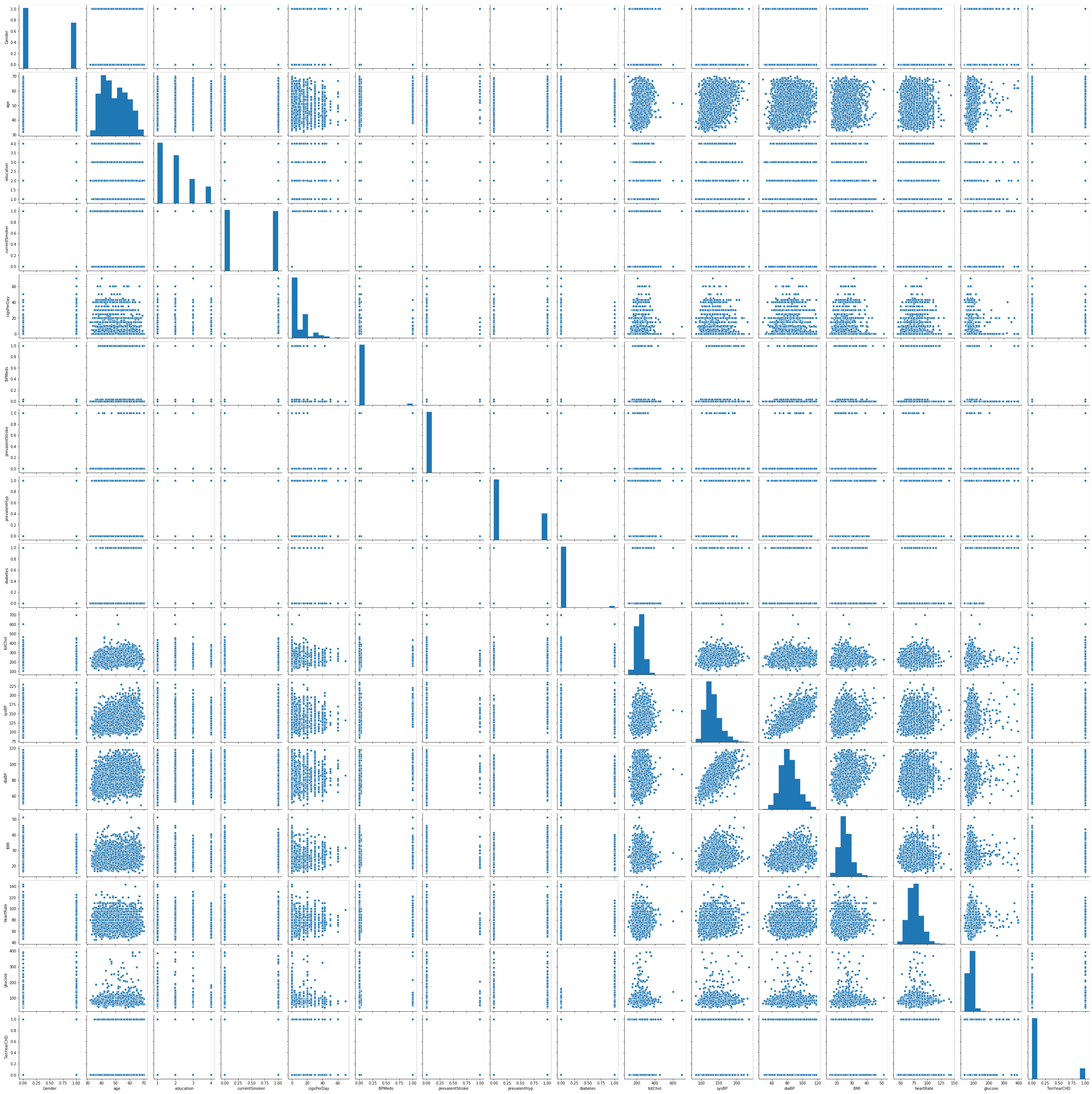
Exploratory Data Analysis:

In [93]:

```
sn.pairplot(data=data_final)
```

Out[93]:

```
<seaborn.axisgrid.PairGrid at 0x1d2c1e54d48>
```

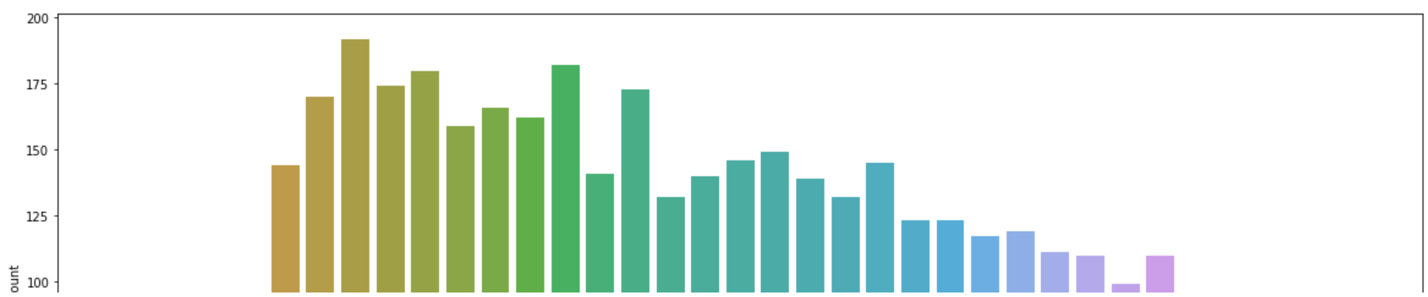


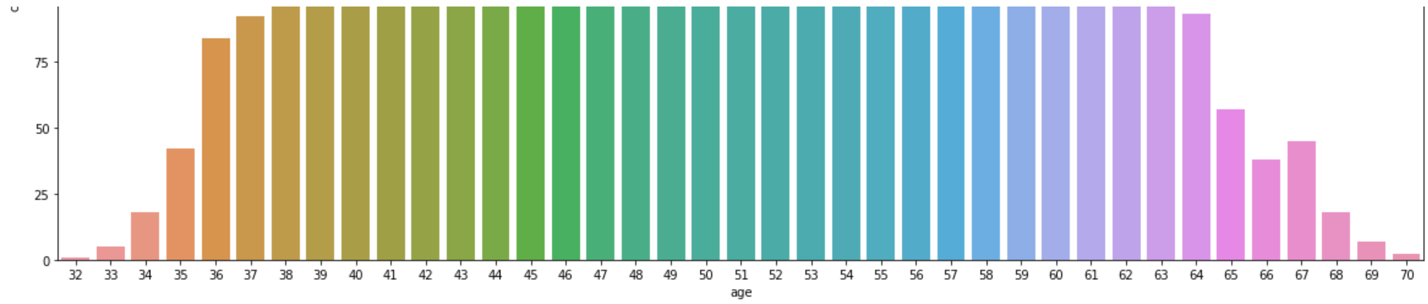
In [25]:

```
plt.rcParams["figure.figsize"] = (20,8)
sn.countplot(x='age',data=data_final)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23296568208>
```





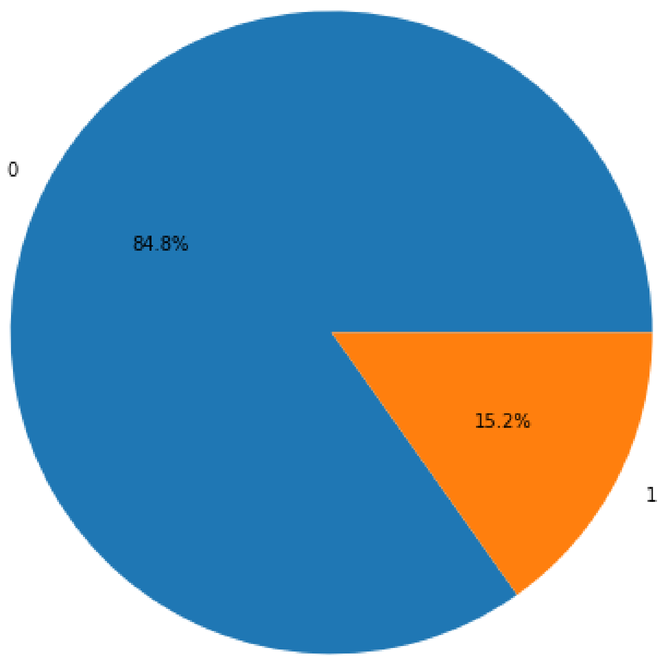
As you see most of the people are of age between 37 to 64 with a count of more than 100 people per age

In [26]:

```
labels= 0,1
plt.pie(data['TenYearCHD'].value_counts(), labels=labels,
        autopct='%1.1f%%')
```

Out[26]:

```
([<matplotlib.patches.Wedge at 0x232978dff88>,
  <matplotlib.patches.Wedge at 0x232978e75c8>],
 [Text(-0.9771297824712156, 0.5051904474628901, '0'),
  Text(0.9771297351718677, -0.5051905389483811, '1')],
 [Text(-0.5329798813479357, 0.2755584258888491, '84.8%'),
  Text(0.5329798555482914, -0.275558475790026, '15.2%')])
```



In [27]:

```
data.corr()
```

Out[27]:

	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes
Gender	1.000000	-0.029014	0.017188	0.197026	0.316023	-0.052203	-0.004550	0.005853	0.015693
age	-0.029014	1.000000	-0.164081	-0.213662	-0.192534	0.122036	0.057679	0.306799	0.101314
education	0.017188	-0.164081	1.000000	0.018297	0.008197	-0.010689	-0.035139	-0.080753	-0.038214

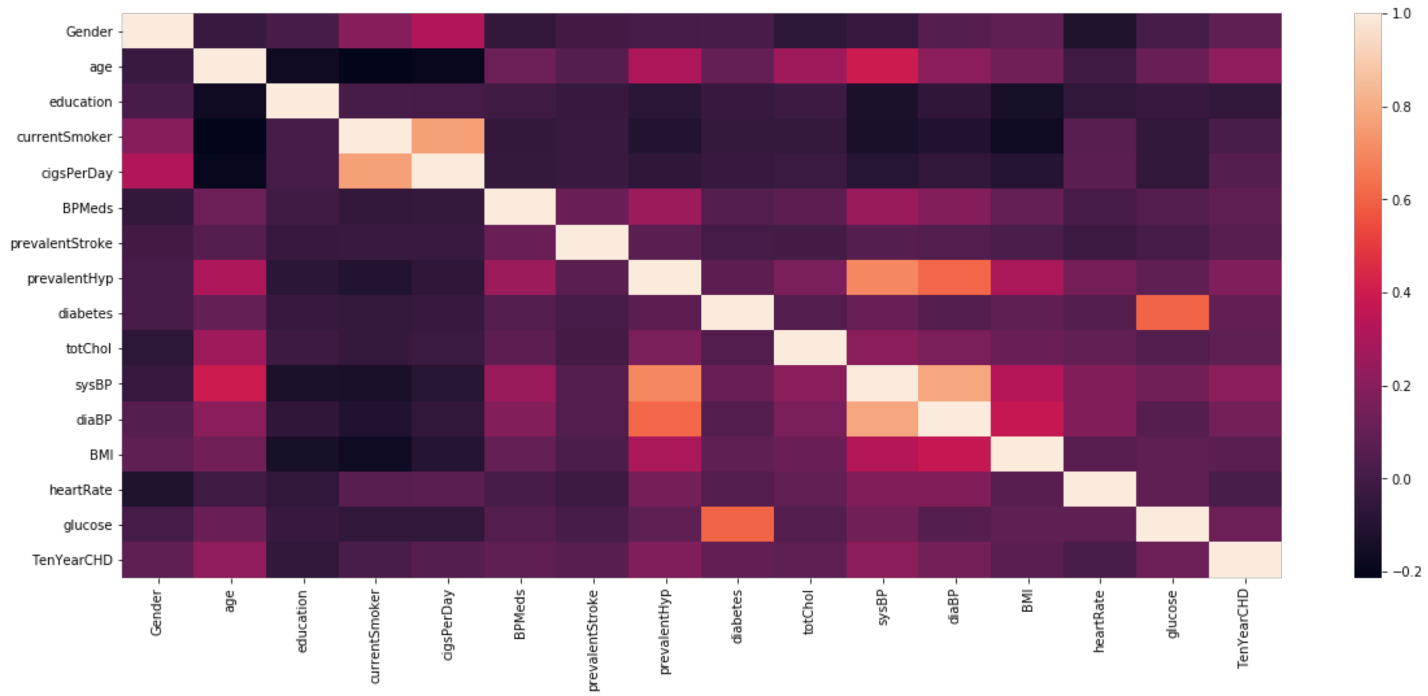
	Gender	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes
currentSmoker	0.197026	-0.2136[illegible]	0.018297	1.000000	0.767055	-0.0486[illegible]	-0.032980	-0.103710	-0.04[illegible]
cigsPerDay	0.316023	-0.192534	0.008197	0.767055	1.000000	-0.045847	-0.032711	-0.066444	-0.037086
BPMeds	-0.052203	0.122036	-0.010689	-0.048621	-0.045847	1.000000	0.115008	0.259125	0.051584
prevalentStroke	-0.004550	0.057679	-0.035139	-0.032980	-0.032711	0.115008	1.000000	0.074791	0.006955
prevalentHyp	0.005853	0.306799	-0.080753	-0.103710	-0.066444	0.259125	0.074791	1.000000	0.077752
diabetes	0.015693	0.101314	-0.038214	-0.044285	-0.037086	0.051584	0.006955	0.077752	1.000000
totChol	-0.070064	0.260691	-0.022993	-0.046211	-0.026182	0.078973	0.000105	0.162683	0.040161
sysBP	-0.035879	0.394053	-0.128126	-0.130281	-0.088523	0.252023	0.057000	0.696656	0.111265
diaBP	0.058199	0.205586	-0.061362	-0.107933	-0.056473	0.192387	0.045153	0.615840	0.050260
BMI	0.081705	0.135578	-0.135876	-0.167483	-0.092888	0.099586	0.024856	0.300599	0.086282
heartRate	-0.116913	-0.012839	-0.053603	0.062678	0.075257	0.015172	-0.017674	0.146777	0.048986
glucose	0.005718	0.116951	-0.033837	-0.054062	-0.056020	0.048925	0.018065	0.082757	0.605709
TenYearCHD	0.088374	0.225408	-0.053571	0.019448	0.057646	0.086805	0.061823	0.177458	0.097344

In [28]:

```
corr=data.corr()
sn.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns)
```

Out[28]:

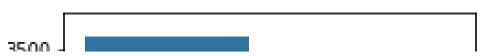
<matplotlib.axes._subplots.AxesSubplot at 0x23297903c48>

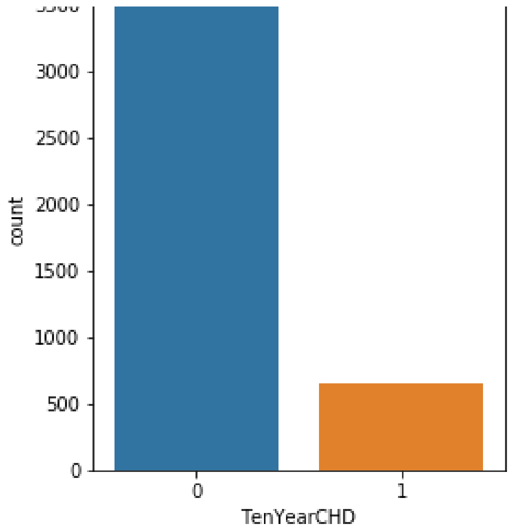


In [29]:

```
plt.rcParams["figure.figsize"] = (4,5)
sn.countplot(x='TenYearCHD',data=data_final)
print(data_final.TenYearCHD.value_counts())
```

```
0    3596
1     644
Name: TenYearCHD, dtype: int64
```





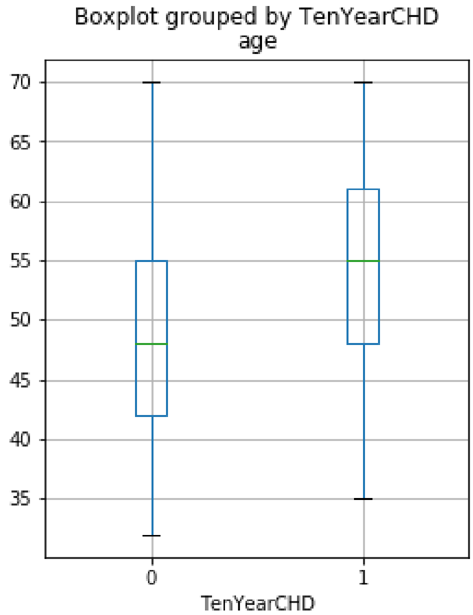
There are almost 3575 people who are not don't stand a chance to get heart disease in the next 10 years but only 624 have a chance to get

In [30]:

```
data.boxplot(column='age',by='TenYearCHD')
```

Out[30]:

<matplotlib.axes._subplots.AxesSubplot at 0x23297b6fd88>

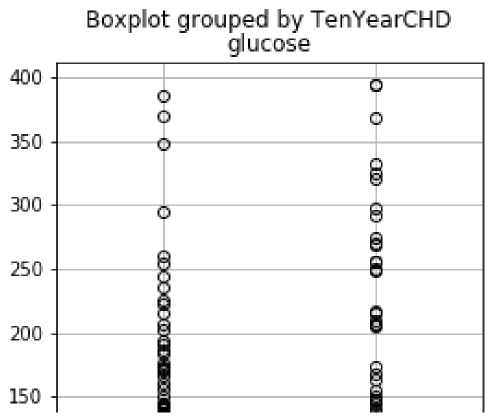


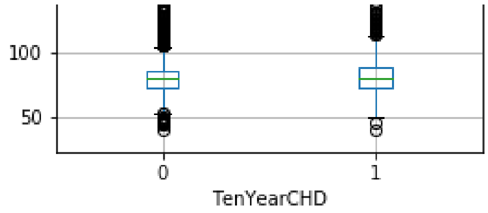
In [31]:

```
data.boxplot(column='glucose',by='TenYearCHD')
```

Out[31]:

<matplotlib.axes._subplots.AxesSubplot at 0x23297ae7a48>





Creating training and Testing Data:

In [32]:

```
from sklearn.model_selection import train_test_split
```

In [33]:

```
X = data_final.loc[:, data_final.columns != 'TenYearCHD']
y = data_final.loc[:, data_final.columns == 'TenYearCHD']
```

In [34]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
```

Building the Machine learning Models:

1. Linear Regression:

In [35]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score

from sklearn.linear_model import LogisticRegression
```

In [36]:

```
normalized_df_reg = LogisticRegression().fit(X_train, y_train)

normalized_df_reg_pred = normalized_df_reg.predict(X_test)

acc = accuracy_score(y_test, normalized_df_reg_pred)
print(f"The accuracy score for LogReg is: {round(acc,3)*100}%")

f1 = f1_score(y_test, normalized_df_reg_pred)
print(f"The f1 score for LogReg is: {round(f1,3)*100}%")

precision = precision_score(y_test, normalized_df_reg_pred)
print(f"The precision score for LogReg is: {round(precision,3)*100}%")

recall = recall_score(y_test, normalized_df_reg_pred)
print(f"The recall score for LogReg is: {round(recall,3)*100}%")
```

```
The accuracy score for LogReg is: 85.7%
The f1 score for LogReg is: 7.6%
The precision score for LogReg is: 76.9%
The recall score for LogReg is: 4.0%
```

From Above we got to know that by using Logistic regression we get f1 score as 7.6%

2. KNN Regression:

In [37]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [38]:

```
knn = KNeighborsClassifier(n_neighbors = 2)

knn.fit(X_train, y_train)

normalized_df_knn_pred = knn.predict(X_test)

acc = accuracy_score(y_test, normalized_df_knn_pred)
print(f"The accuracy score for KNN is: {round(acc,3)*100}%")

f1 = f1_score(y_test, normalized_df_knn_pred)
print(f"The f1 score for KNN is: {round(f1,3)*100}%")

precision = precision_score(y_test, normalized_df_knn_pred)
print(f"The precision score for KNN is: {round(precision,3)*100}%")

recall = recall_score(y_test, normalized_df_knn_pred)
print(f"The recall score for KNN is: {round(recall,3)*100}%")
```

```
The accuracy score for KNN is: 84.1%
The f1 score for KNN is: 10.0%
The precision score for KNN is: 30.0%
The recall score for KNN is: 6.0%
```

As we see that f1 score for test set using KNN classifier is 10.0% which is better than Linear Regression

3. Decision Tree:

In [39]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [40]:

```
dtc_up = DecisionTreeClassifier()

dtc_up.fit(X_train, y_train)

normalized_df_dtc_pred = dtc_up.predict(X_test)

acc = accuracy_score(y_test, normalized_df_dtc_pred)
print(f"The accuracy score for DTC is: {round(acc,3)*100}%")

f1 = f1_score(y_test, normalized_df_dtc_pred)
print(f"The f1 score for DTC is: {round(f1,3)*100}%")

precision = precision_score(y_test, normalized_df_dtc_pred)
print(f"The precision score for DTC is: {round(precision,3)*100}%")

recall = recall_score(y_test, normalized_df_dtc_pred)
print(f"The recall score for DTC is: {round(recall,3)*100}%")
```

```
The accuracy score for DTC is: 74.6%
The f1 score for DTC is: 23.200000000000003%
The precision score for DTC is: 20.9%
The recall score for DTC is: 26.0%
```

Here the f1 score for Decision Tree is 23.2%

Conclusion:

From the above three algorithms, the Decision tree algorithm gives the best f1 score of 21.2%. This is because:

1. **Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.**
2. **A decision tree does not require normalization of data.**
3. **A decision tree does not require scaling of data as well.**
4. **Missing values in the data also do NOT affect the process of building a decision tree to any considerable extent.**
5. **A Decision tree model is very intuitive and easy to explain to technical teams as well as stakeholders.**