In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, confusion_matrix, accuracy_score, roc_cu
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [2]:
```
df=pd.read_csv('train.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diab
0	0	64	2.0	F	YES	3.0	0.0	0	0	
1	1	36	4.0	M	NO	0.0	0.0	0	1	
2	2	46	1.0	F	YES	10.0	0.0	0	0	
3	3	50	1.0	M	YES	20.0	0.0	0	1	
4	4	64	1.0	F	YES	30.0	0.0	0	0	

In [4]:
```
df.tail()
```

Out[4]:

	id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp
3385	3385	60	1.0	F	NO	0.0	0.0	0	0
3386	3386	46	1.0	F	NO	0.0	0.0	0	0
3387	3387	44	3.0	M	YES	3.0	0.0	0	1
3388	3388	60	1.0	M	NO	0.0	NaN	0	1
3389	3389	54	3.0	F	NO	0.0	0.0	0	0

In [5]:
```
df.shape
```

Out[5]:
```
(3390, 17)
```

In [6]:
```
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
```

In [7]:
```
df.describe(include='all').T
```

Out[7]:

	count	unique	top	freq	mean	std	min	25%	50%	75%
id	3390.0	NaN	NaN	NaN	1694.5	978.753033	0.0	847.25	1694.5	2541.7
age	3390.0	NaN	NaN	NaN	49.542183	8.592878	32.0	42.0	49.0	56.(
education	3303.0	NaN	NaN	NaN	1.970936	1.019081	1.0	1.0	2.0	3.(
sex	3390	2	F	1923	NaN	NaN	NaN	NaN	NaN	NaN
is_smoking	3390	2	NO	1703	NaN	NaN	NaN	NaN	NaN	NaN
cigsPerDay	3368.0	NaN	NaN	NaN	9.069477	11.879078	0.0	0.0	0.0	20.(
BPMeds	3346.0	NaN	NaN	NaN	0.029886	0.170299	0.0	0.0	0.0	0.(
prevalentStroke	3390.0	NaN	NaN	NaN	0.00649	0.080309	0.0	0.0	0.0	0.(
prevalentHyp	3390.0	NaN	NaN	NaN	0.315339	0.464719	0.0	0.0	0.0	1.(
diabetes	3390.0	NaN	NaN	NaN	0.025664	0.158153	0.0	0.0	0.0	0.(
totChol	3352.0	NaN	NaN	NaN	237.074284	45.24743	107.0	206.0	234.0	264.(
sysBP	3390.0	NaN	NaN	NaN	132.60118	22.29203	83.5	117.0	128.5	144.(
diaBP	3390.0	NaN	NaN	NaN	82.883038	12.023581	48.0	74.5	82.0	90.(
BMI	3376.0	NaN	NaN	NaN	25.794964	4.115449	15.96	23.02	25.38	28.0
heartRate	3389.0	NaN	NaN	NaN	75.977279	11.971868	45.0	68.0	75.0	83.(
glucose	3086.0	NaN	NaN	NaN	82.08652	24.244753	40.0	71.0	78.0	87.(
TenYearCHD	3390.0	NaN	NaN	NaN	0.150737	0.357846	0.0	0.0	0.0	0.(

Checking For duplication of Data.

In [8]:
```
df[df.duplicated()]
```

Out[8]:

id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabe

Handling Missing Values

Checking for Missing Values

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
id                  0
age                 0
education          87
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
```

In [10]:
```
# Before Altering the Data copying orinal data
df_copy=df.copy()
```

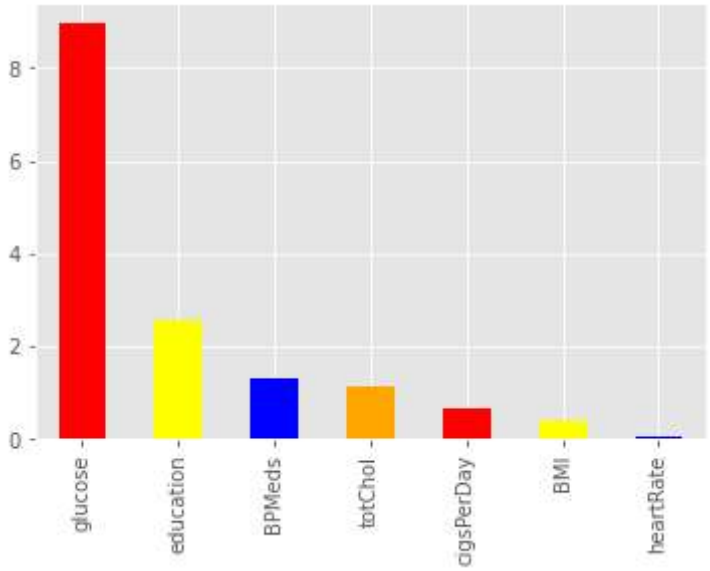
Hear total Missing data was less than 13% insted of delecting data we replacing missing values with approxmate values

In [11]:
```
# Missing Value Count Function
def show_missing():
    missing = df_copy.columns[df_copy.isnull().any()].tolist()
    return missing

# Missing data counts and percentage
print('Missing Data Count')
print(df_copy[show_missing()].isnull().sum().sort_values(ascending = False))
print('--'*50)
print('Missing Data Percentage')
print(round(df_copy[show_missing()].isnull().sum().sort_values(ascending = False)/1
```

```
Missing Data Count
glucose       304
education      87
BPMeds         44
totChol        38
cigsPerDay     22
BMI            14
heartRate       1
dtype: int64
--------------------------------------------------------------------------------
-----------------
Missing Data Percentage
glucose       8.97
education     2.57
BPMeds        1.30
totChol       1.12
cigsPerDay    0.65
BMI           0.41
heartRate     0.03
dtype: float64
```

In [12]:
```
round(df_copy[show_missing()].isnull().sum().sort_values(ascending = False)/len(df_
```

Out[12]:
```
<AxesSubplot:>
```



Glucose

In the following column mean and median are nearby. we have fill the missing values so i am using median values for filling the missing values.

In [13]:
```
df['glucose'].describe()
```

Out[13]:
```
count    3086.000000
mean       82.086520
std        24.244753
min        40.000000
25%        71.000000
50%        78.000000
75%        87.000000
max       394.000000
Name: glucose, dtype: float64
```

In [14]:
```
print('Glucose Feature Missing Before')
print(df_copy[['glucose']].isnull().sum())
```

```
print('--'*50)
df_copy['glucose']=df_copy['glucose'].fillna(df['glucose'].median())
print('Glucose Feature Missing After')
print(df_copy[['glucose']].isnull().sum())
print('--'*50)
```

```
Glucose Feature Missing Before
glucose    304
dtype: int64
----------------------------------------------------------------------------------
-----------------
Glucose Feature Missing After
glucose    0
dtype: int64
----------------------------------------------------------------------------------
-----------------
```

Education

In [15]:
```
df['education'].describe()
```

Out[15]:
```
count    3303.000000
mean        1.970936
std         1.019081
min         1.000000
25%         1.000000
50%         2.000000
75%         3.000000
max         4.000000
Name: education, dtype: float64
```

In [16]:
```
df['education'].unique()
```

Out[16]:
```
array([ 2.,  4.,  1.,  3., nan])
```

Education feature is not a continues variable so we using Mode for filling the missing values.

In [17]:
```
print('Education Feature Missing Before')
print(df_copy[['education']].isnull().sum())
print('--'*50)
df_copy['education']=df_copy['education'].fillna(df['education'].mode()[0])
print('Education Feature Missing After')
print(df_copy[['education']].isnull().sum())
print('--'*50)
```

```
Education Feature Missing Before
education    87
dtype: int64
----------------------------------------------------------------------------------
-----------------
Education Feature Missing After
education    0
dtype: int64
----------------------------------------------------------------------------------
-----------------
```

BPMeds

In [18]:
```
df['BPMeds'].describe()
```

Out[18]:
```
count    3346.000000
mean        0.029886
std         0.170299
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
Name: BPMeds, dtype: float64
```

In [19]:
```
df['BPMeds'].unique()
```

Out[19]:
```
array([ 0., nan,  1.])
```

In [20]:
```
print('BPMeds Feature Missing Before')
print(df_copy[['BPMeds']].isnull().sum())
print('--'*50)
df_copy['BPMeds']=df_copy['BPMeds'].fillna(df['BPMeds'].mode()[0])
print('BPMeds Feature Missing After')
print(df_copy[['BPMeds']].isnull().sum())
print('--'*50)
```

```
BPMeds Feature Missing Before
BPMeds    44
dtype: int64
----------------------------------------------------------------------------------
-----------------
BPMeds Feature Missing After
BPMeds    0
dtype: int64
----------------------------------------------------------------------------------
-----------------
```

Total Cholostral

In [21]:
```
df['totChol'].describe()
```

Out[21]:
```
count    3352.000000
mean      237.074284
std        45.247430
min       107.000000
25%       206.000000
50%       234.000000
75%       264.000000
max       696.000000
Name: totChol, dtype: float64
```

In [22]:
```
print('Total colostrol Feature Missing Before')
print(df_copy[['totChol']].isnull().sum())
print('--'*50)
df_copy['totChol']=df_copy['totChol'].fillna(df['totChol'].median())
print('Total colostrol Feature Missing After')
print(df_copy[['totChol']].isnull().sum())
print('--'*50)
```

```
Total colostrol Feature Missing Before
totChol    38
dtype: int64
--------------------------------------------------------------------------------
-----------------
Total colostrol Feature Missing After
totChol    0
dtype: int64
--------------------------------------------------------------------------------
-----------------
```

Cigrates per Day

In [23]:
```
df['cigsPerDay'].describe()
```

Out[23]:
```
count    3368.000000
mean        9.069477
std        11.879078
min         0.000000
25%         0.000000
50%         0.000000
75%        20.000000
max        70.000000
Name: cigsPerDay, dtype: float64
```

In [24]:
```
print('Cigars per day Feature Missing Before')
print(df_copy[['cigsPerDay']].isnull().sum())
print('--'*50)
df_copy['cigsPerDay']=df_copy['cigsPerDay'].fillna(df['cigsPerDay'].median())
print('Cigars per day Feature Missing After')
print(df_copy[['cigsPerDay']].isnull().sum())
print('--'*50)
```

```
Cigars per day Feature Missing Before
cigsPerDay    22
dtype: int64
--------------------------------------------------------------------------------
-----------------
Cigars per day Feature Missing After
cigsPerDay    0
dtype: int64
--------------------------------------------------------------------------------
-----------------
```

Body Mass Index(BMI)

In [25]:
```
df['BMI'].describe()
```

Out[25]:
```
count    3376.000000
mean       25.794964
std         4.115449
min        15.960000
25%        23.020000
50%        25.380000
75%        28.040000
max        56.800000
Name: BMI, dtype: float64
```

In [26]:
```
print('BMI Feature Missing Before')
print(df_copy[['BMI']].isnull().sum())
print('--'*50)
df_copy['BMI']=df_copy['BMI'].fillna(df['BMI'].median())
print('BMI Feature Missing After')
```

```
print(df_copy[['BMI']].isnull().sum())
print('--'*50)
```

```
BMI Feature Missing Before
BMI    14
dtype: int64
------------------------------------------------------------------------------
-----------------
BMI Feature Missing After
BMI    0
dtype: int64
------------------------------------------------------------------------------
-----------------
```

Heart Rate

In [27]:
```
df['heartRate'].describe()
```

Out[27]:
```
count    3389.000000
mean       75.977279
std        11.971868
min        45.000000
25%        68.000000
50%        75.000000
75%        83.000000
max       143.000000
Name: heartRate, dtype: float64
```

In [28]:
```
print('Heart Rate Feature Missing Before')
print(df_copy[['heartRate']].isnull().sum())
print('--'*50)
df_copy['heartRate']=df_copy['heartRate'].fillna(df['heartRate'].median())
print('Heart Rate Feature Missing After')
print(df_copy[['heartRate']].isnull().sum())
print('--'*50)
```

```
Heart Rate Feature Missing Before
heartRate    1
dtype: int64
------------------------------------------------------------------------------
-----------------
Heart Rate Feature Missing After
heartRate    0
dtype: int64
------------------------------------------------------------------------------
-----------------
```

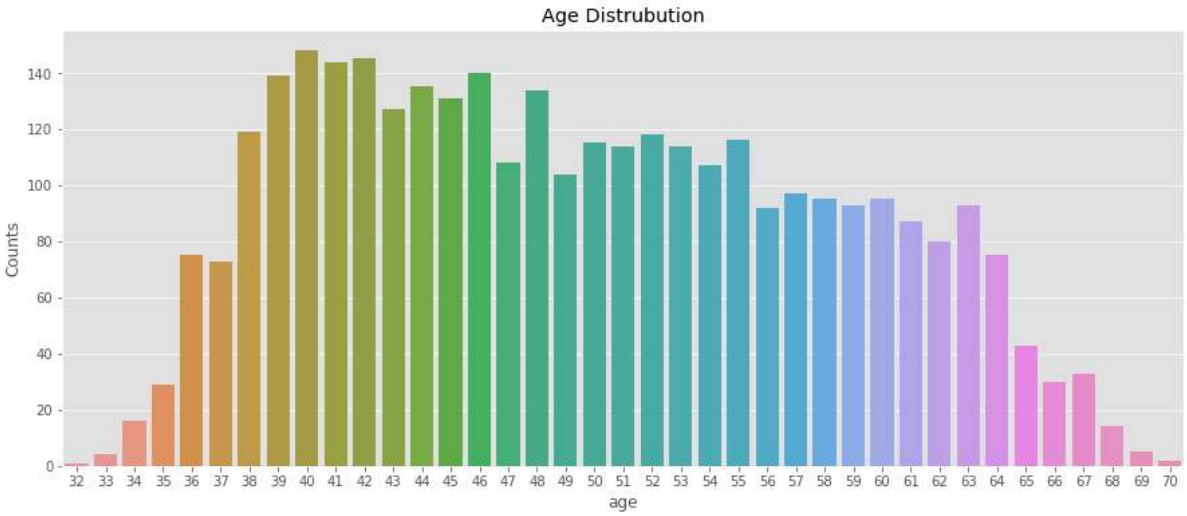
EDA

Age

Data contains people of age from 32-70 years. People are effected to cardivasucular Desises from 35

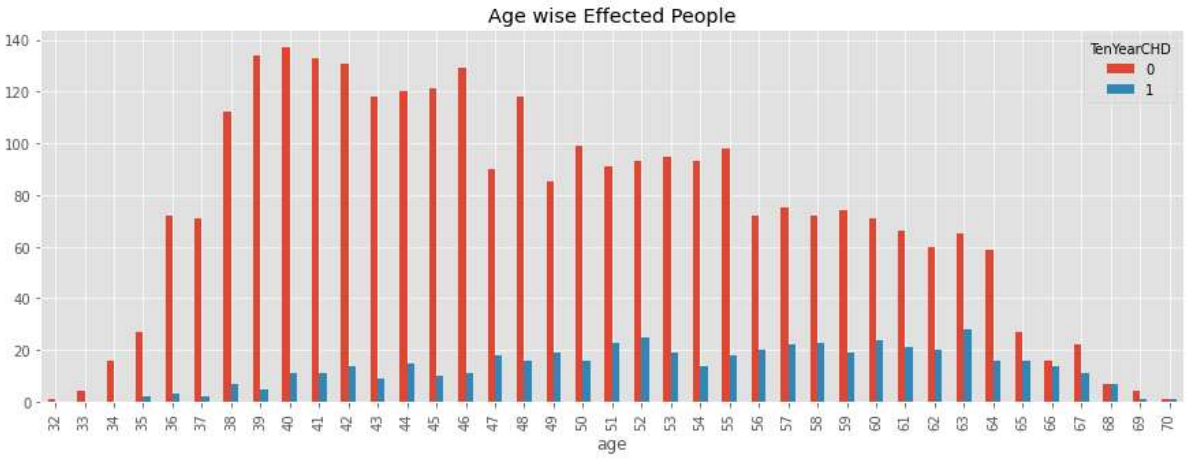
In [29]:
```
fig, ax = plt.subplots(figsize=(15,6))
age_dis=pd.DataFrame(df.groupby(['age'])['id'].count())
sns.barplot(x=age_dis.index,y=age_dis['id'])
plt.ylabel('Counts')
plt.title('Age Distrubution')
```

Out[29]:
```
Text(0.5, 1.0, 'Age Distrubution')
```



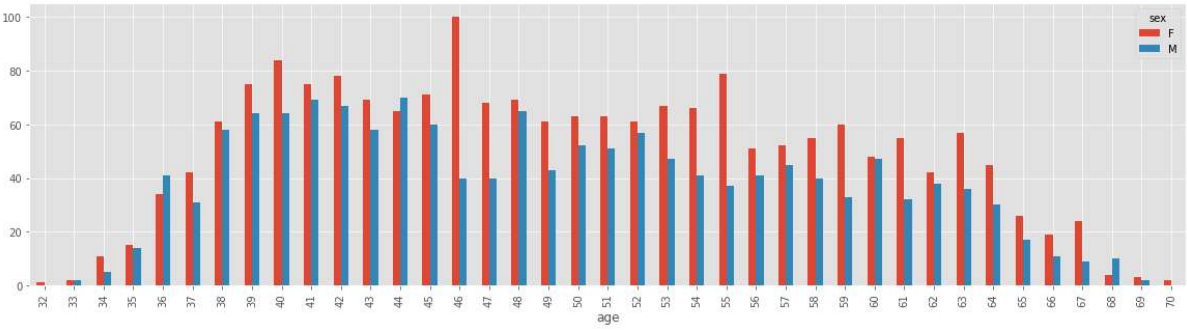
In [30]:
```
plt.rcParams['figure.figsize'] = (15, 5)
df.groupby(['age','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('Age wise Effected People')
```

Out[30]:
```
Text(0.5, 1.0, 'Age wise Effected People')
```



In [31]:
```
plt.rcParams['figure.figsize'] = (20, 5)
df.groupby(['age','sex'])['TenYearCHD'].count().unstack().plot(kind='bar')
```

Out[31]:
```
<AxesSubplot:xlabel='age'>
```



Education

In [32]:
```
df['education'].unique()
```

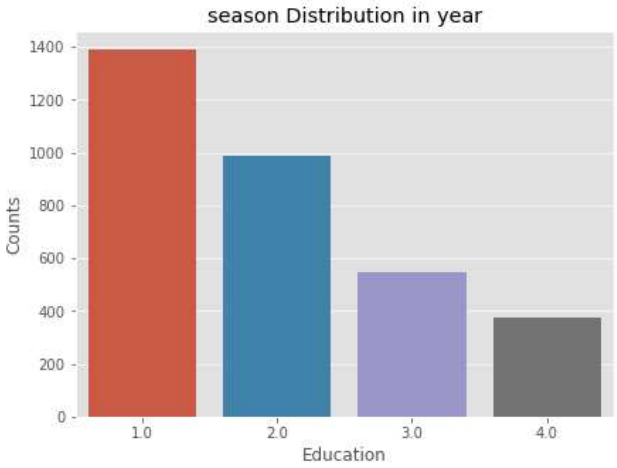
Out[32]:
```
array([ 2.,  4.,  1.,  3., nan])
```

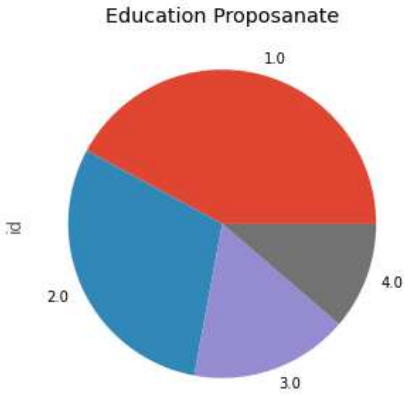
In [33]:
```
df.groupby(['education'])['id'].count()
```

Out[33]:
```
education
1.0    1391
2.0     990
3.0     549
4.0     373
Name: id, dtype: int64
```

In [34]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['education'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('Education')
plt.ylabel('Counts')
plt.title('season Distribution in year')
ax2=plt.subplot(1,2,2)
df.groupby(['education'])['id'].count().plot(kind='pie')
plt.title('Education Proposanate')
```

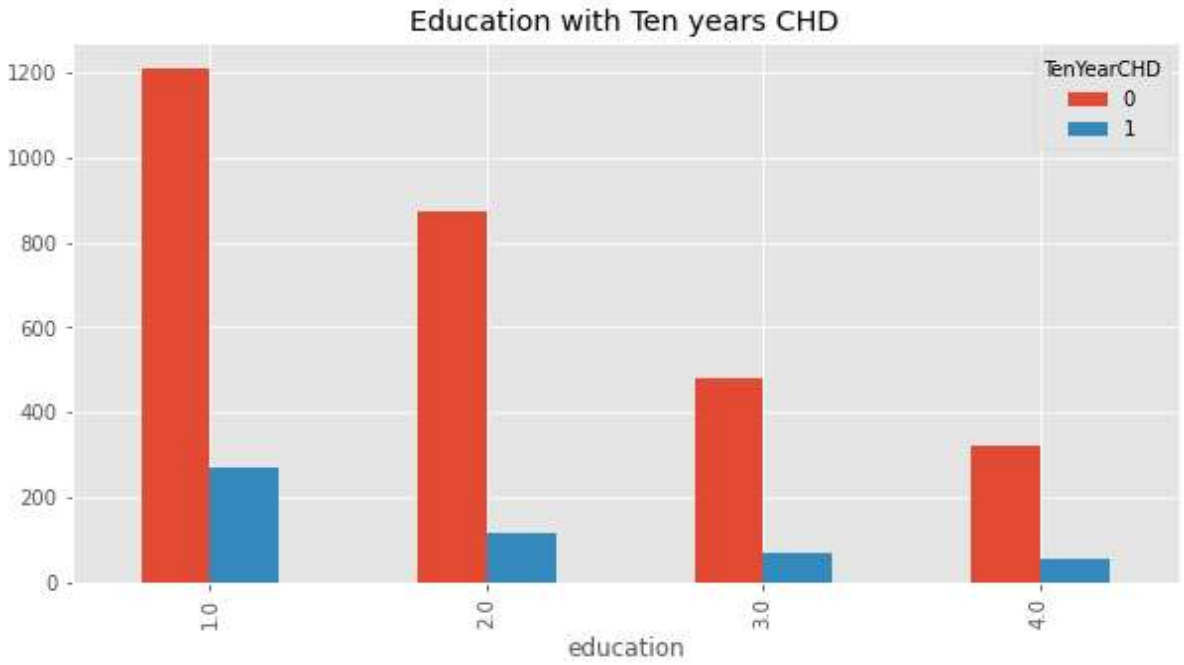
Out[34]:
```
Text(0.5, 1.0, 'Education Proposanate')
```





In [35]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['education','TenYearCHD'])['id'].count().unstack().plot(kind='bar'
plt.title('Education with Ten years CHD')
```

Out[35]:
```
Text(0.5, 1.0, 'Education with Ten years CHD')
```



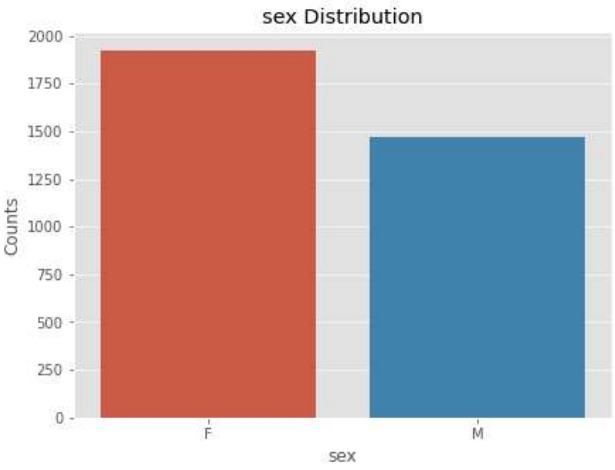
SEX

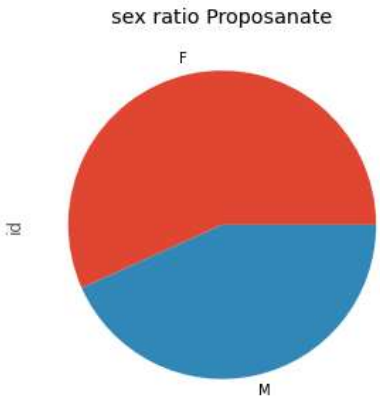
In [36]:
```
df.groupby(['sex'])['id'].count()
```

Out[36]:
```
sex
F    1923
M    1467
Name: id, dtype: int64
```

In [37]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['sex'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('sex')
plt.ylabel('Counts')
plt.title('sex Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['sex'])['id'].count().plot(kind='pie')
plt.title('sex ratio Proposanate')
```

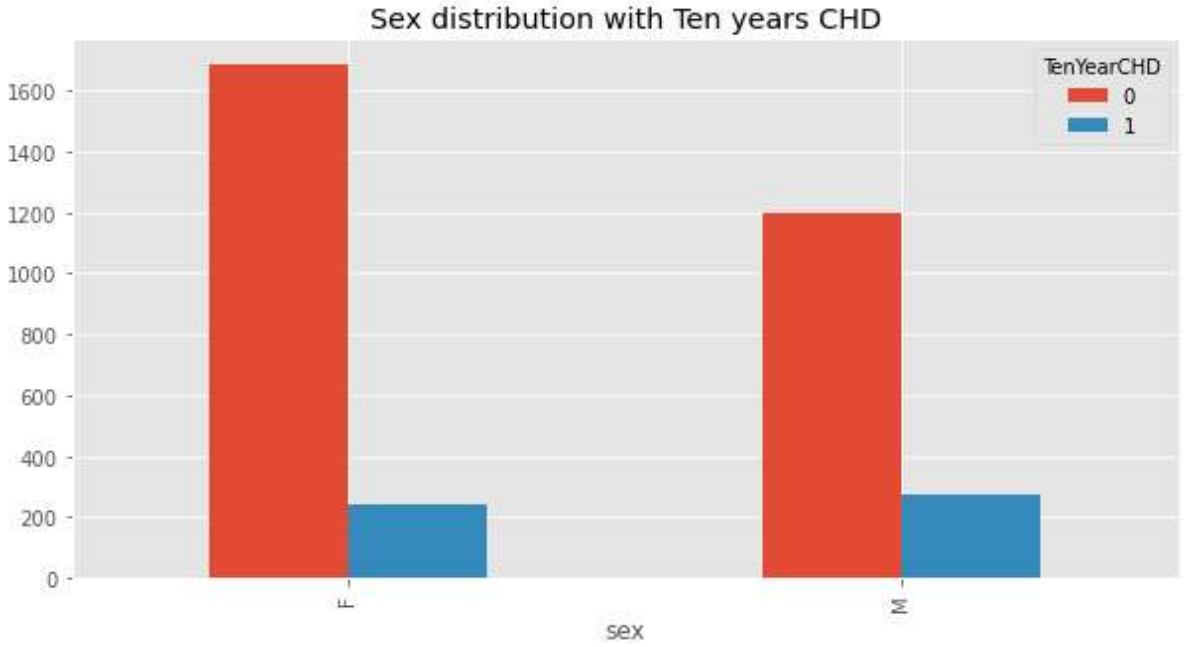
Out[37]:
```
Text(0.5, 1.0, 'sex ratio Proposanate')
```





In [38]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['sex','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('Sex distribution with Ten years CHD')
```

Out[38]:
```
Text(0.5, 1.0, 'Sex distribution with Ten years CHD')
```



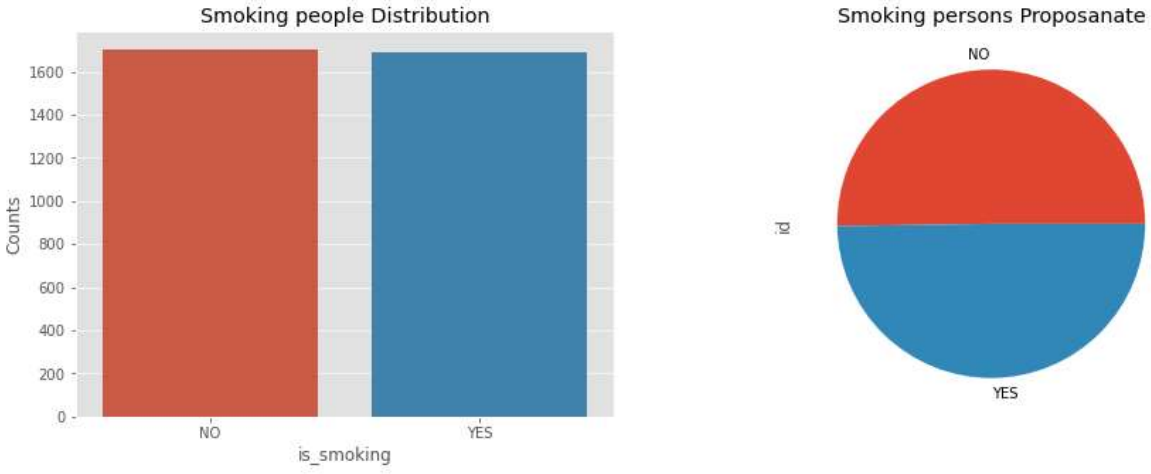
Smoking Data

In [39]:
```
df.groupby(['is_smoking'])['id'].count()
```

Out[39]:
```
is_smoking
NO     1703
YES    1687
Name: id, dtype: int64
```

In [40]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['is_smoking'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('is_smoking')
plt.ylabel('Counts')
plt.title('Smoking people Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['is_smoking'])['id'].count().plot(kind='pie')
plt.title('Smoking persons Proposanate')
```

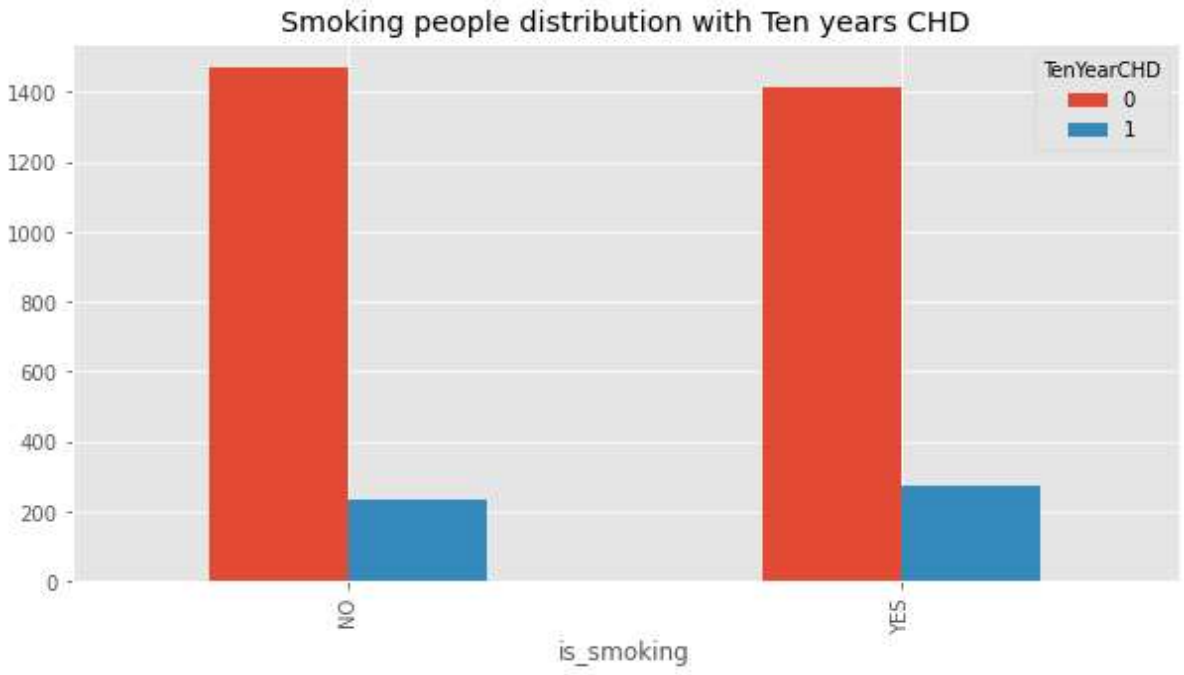
Out[40]:
```
Text(0.5, 1.0, 'Smoking persons Proposanate')
```



In [41]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['is_smoking','TenYearCHD'])['id'].count().unstack().plot(kind='bar
```

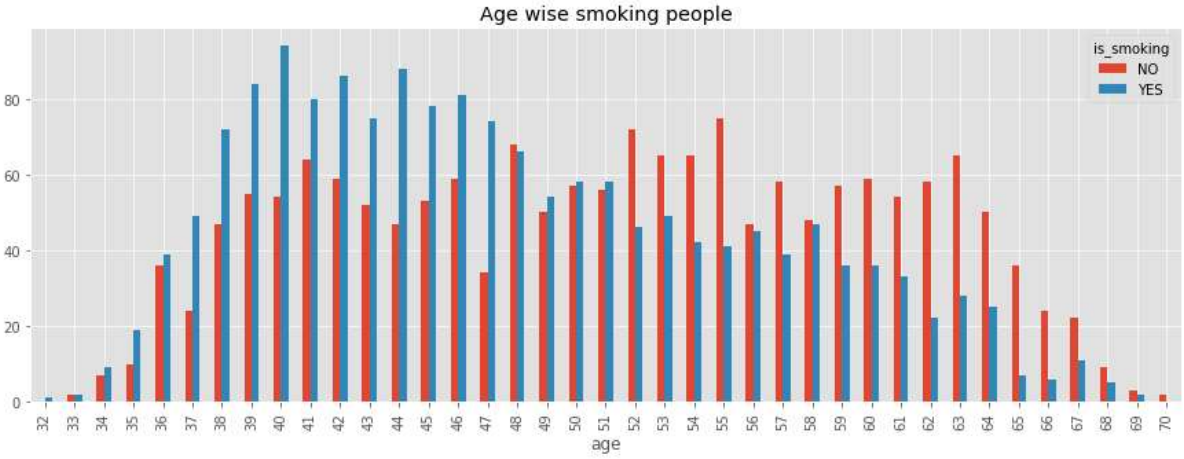
```
plt.title('Smoking people distribution with Ten years CHD')
```

Out[41]:
```
Text(0.5, 1.0, 'Smoking people distribution with Ten years CHD')
```



In [42]:
```
plt.rcParams['figure.figsize'] = (15, 5)
df.groupby(['age','is_smoking'])['id'].count().unstack().plot(kind='bar')
plt.title('Age wise smoking people')
```

Out[42]:
```
Text(0.5, 1.0, 'Age wise smoking people')
```



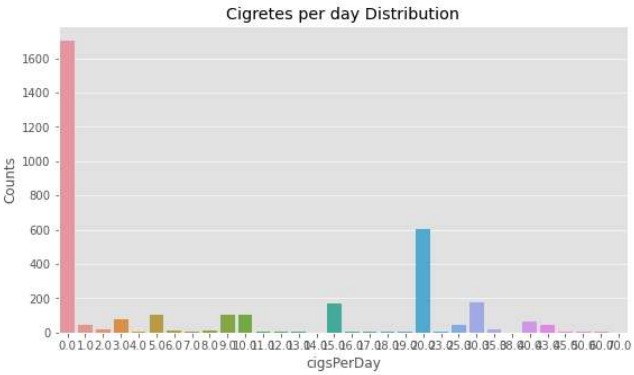
Cigretes per Day

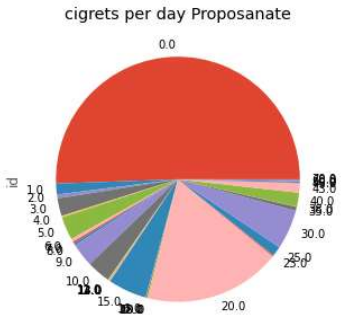
In [43]:
```
df.groupby(['cigsPerDay'])['id'].count()
```

Out[43]:
```
cigsPerDay
0.0     1703
1.0       48
2.0       17
3.0       79
4.0        7
5.0      103
6.0       14
7.0        8
8.0       10
9.0      104
10.0     106
11.0       4
12.0       3
13.0       3
14.0       1
15.0     172
16.0       2
17.0       5
18.0       7
19.0       2
20.0     606
23.0       5
25.0      44
30.0     176
35.0      17
38.0       1
40.0      62
43.0      42
45.0       2
50.0       6
60.0       8
70.0       1
Name: id, dtype: int64
```

In [44]:
```
fig, ax = plt.subplots(1,2,figsize=(20,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['cigsPerDay'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('cigsPerDay')
plt.ylabel('Counts')
plt.title('Cigretes per day Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['cigsPerDay'])['id'].count().plot(kind='pie')
plt.title('cigrets per day Proposanate')
```

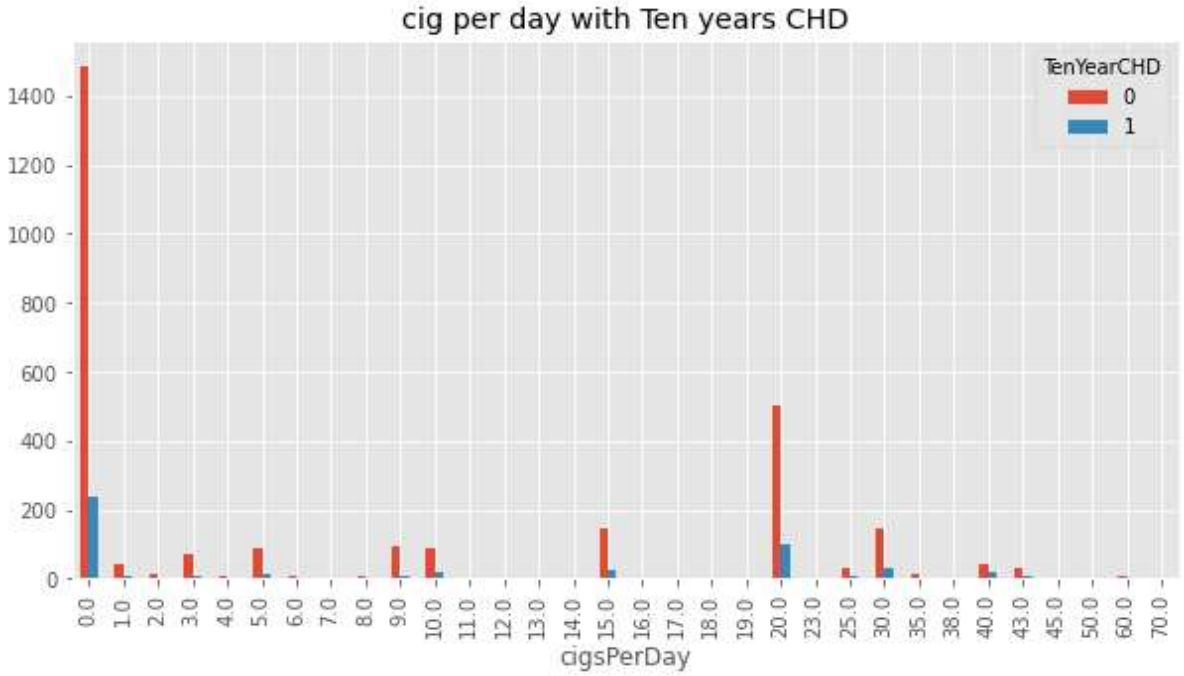
Out[44]:
```
Text(0.5, 1.0, 'cigrets per day Proposanate')
```





In [45]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['cigsPerDay','TenYearCHD'])['id'].count().unstack().plot(kind='bar
plt.title('cig per day with Ten years CHD')
```

Out[45]:
```
Text(0.5, 1.0, 'cig per day with Ten years CHD')
```



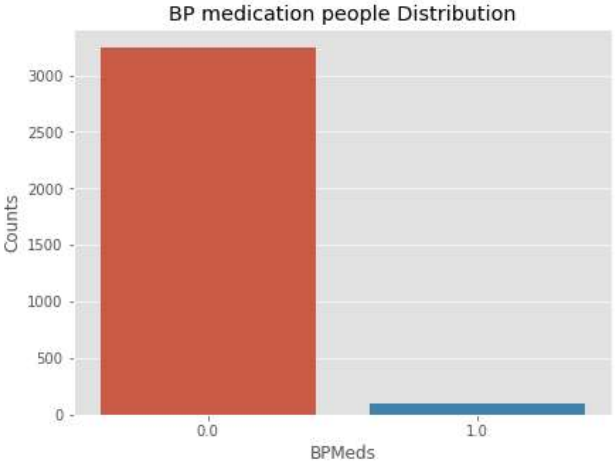
BP Medication

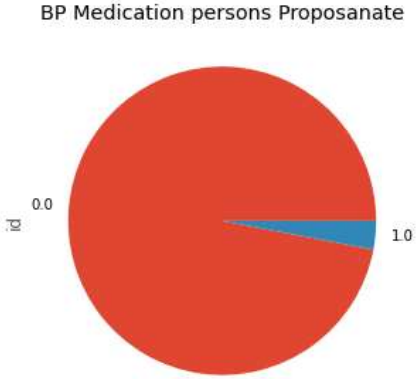
In [46]:
```
df.groupby(['BPMeds'])['id'].count()
```

Out[46]:
```
BPMeds
0.0    3246
1.0     100
Name: id, dtype: int64
```

In [47]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['BPMeds'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('BPMeds')
plt.ylabel('Counts')
plt.title('BP medication people Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['BPMeds'])['id'].count().plot(kind='pie')
plt.title('BP Medication persons Proposanate')
```

Out[47]:
```
Text(0.5, 1.0, 'BP Medication persons Proposanate')
```

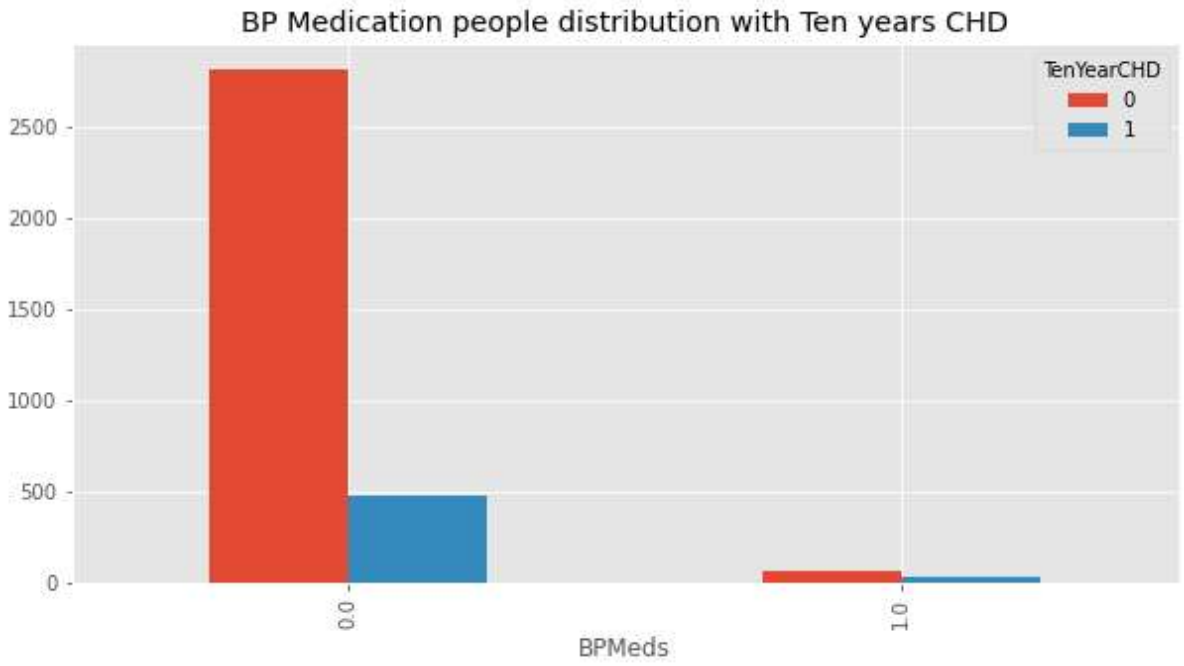




In [48]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['BPMeds','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
```

```
plt.title('BP Medication people distribution with Ten years CHD')
```

Out[48]:
```
Text(0.5, 1.0, 'BP Medication people distribution with Ten years CHD')
```



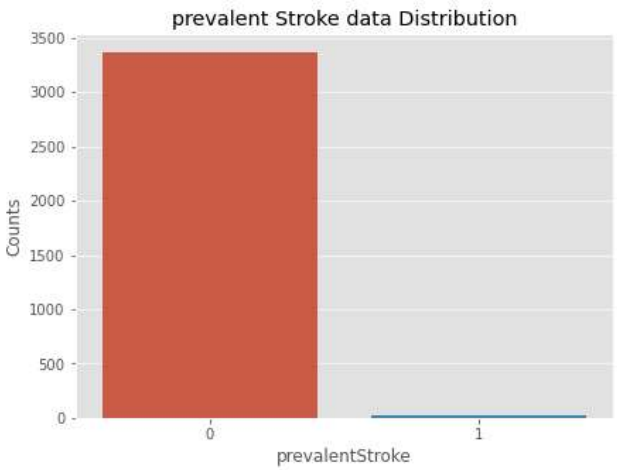
Prevalent Stroke

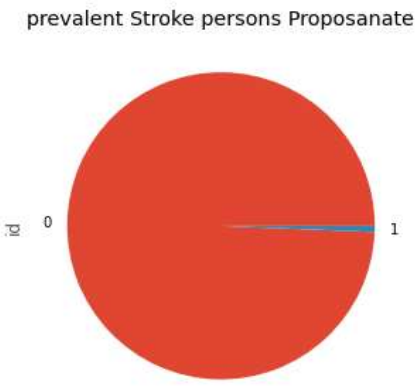
In [49]:
```
df.groupby(['prevalentStroke'])['id'].count()
```

Out[49]:
```
prevalentStroke
0    3368
1      22
Name: id, dtype: int64
```

In [50]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['prevalentStroke'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('prevalentStroke')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['prevalentStroke'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

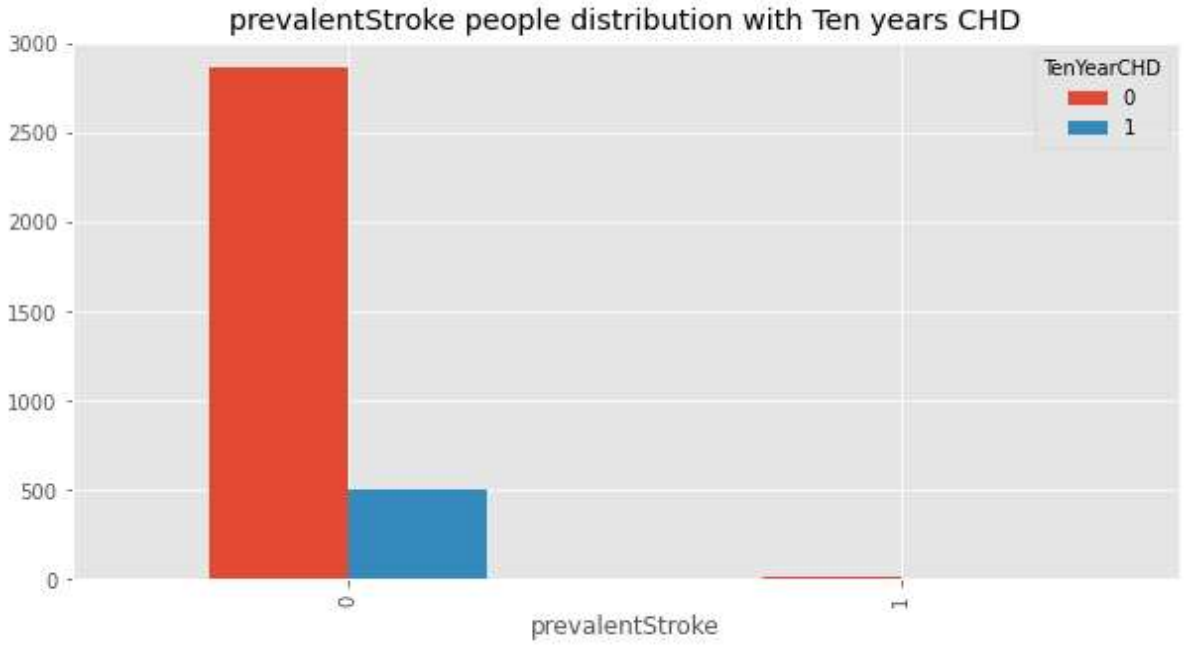
Out[50]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [51]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['prevalentStroke','TenYearCHD'])['id'].count().unstack().plot(kind
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[51]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



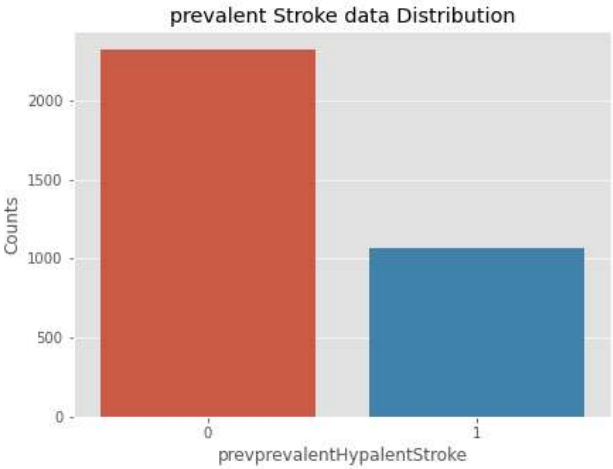
Hypertension

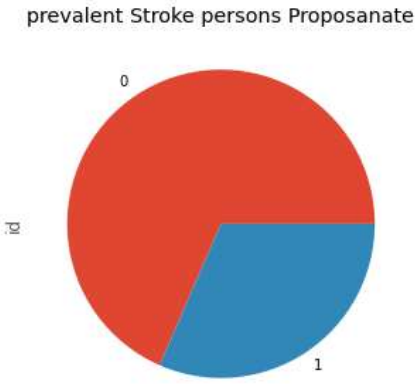
In [52]:
```
df.groupby(['prevalentHyp'])['id'].count()
```

Out[52]:
```
prevalentHyp
0    2321
1    1069
Name: id, dtype: int64
```

In [53]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['prevalentHyp'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('prevprevalentHypalentStroke')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['prevalentHyp'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

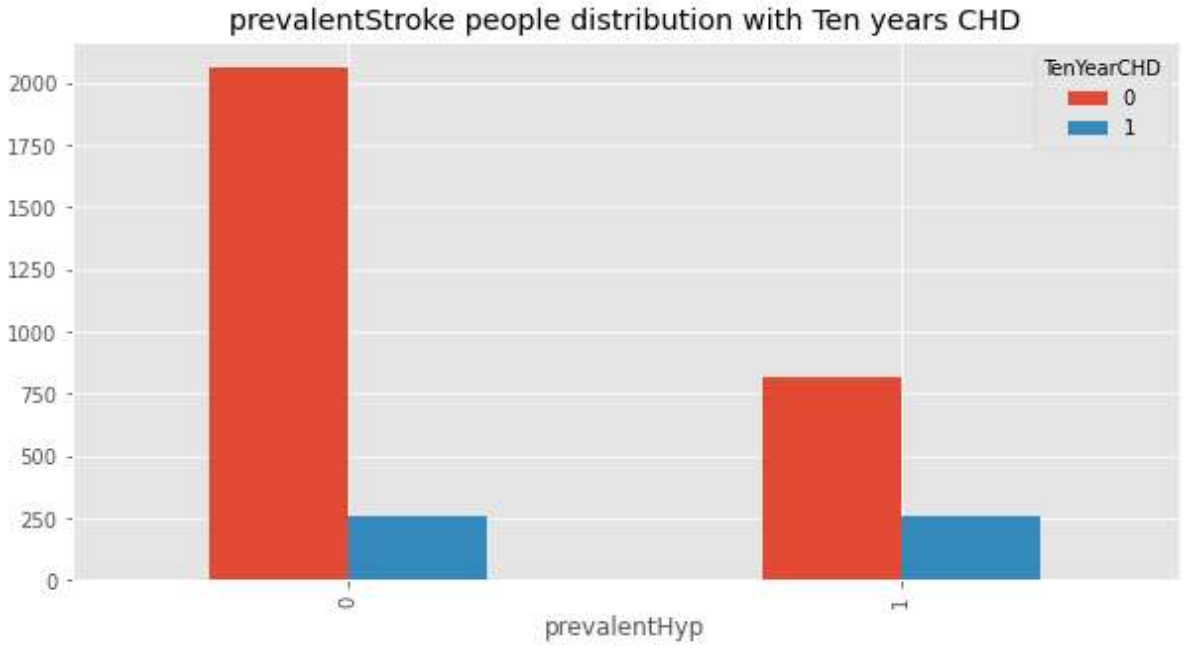
Out[53]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [54]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['prevalentHyp','TenYearCHD'])['id'].count().unstack().plot(kind='b
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[54]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



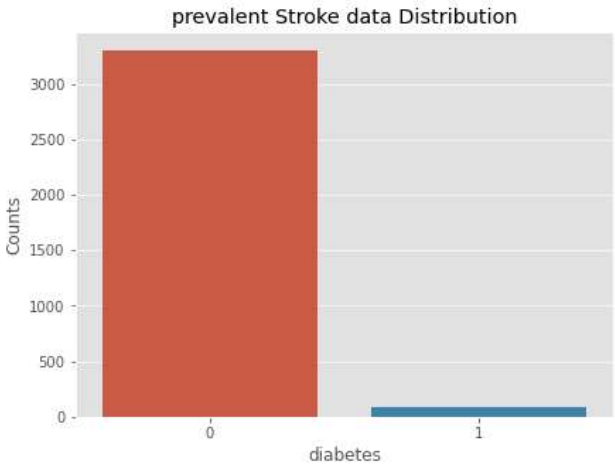
diabetes

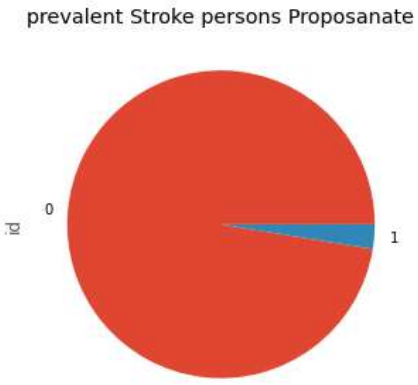
In [55]:
```
df.groupby(['diabetes'])['id'].count()
```

Out[55]:
```
diabetes
0    3303
1      87
Name: id, dtype: int64
```

In [56]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['diabetes'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('diabetes')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['diabetes'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

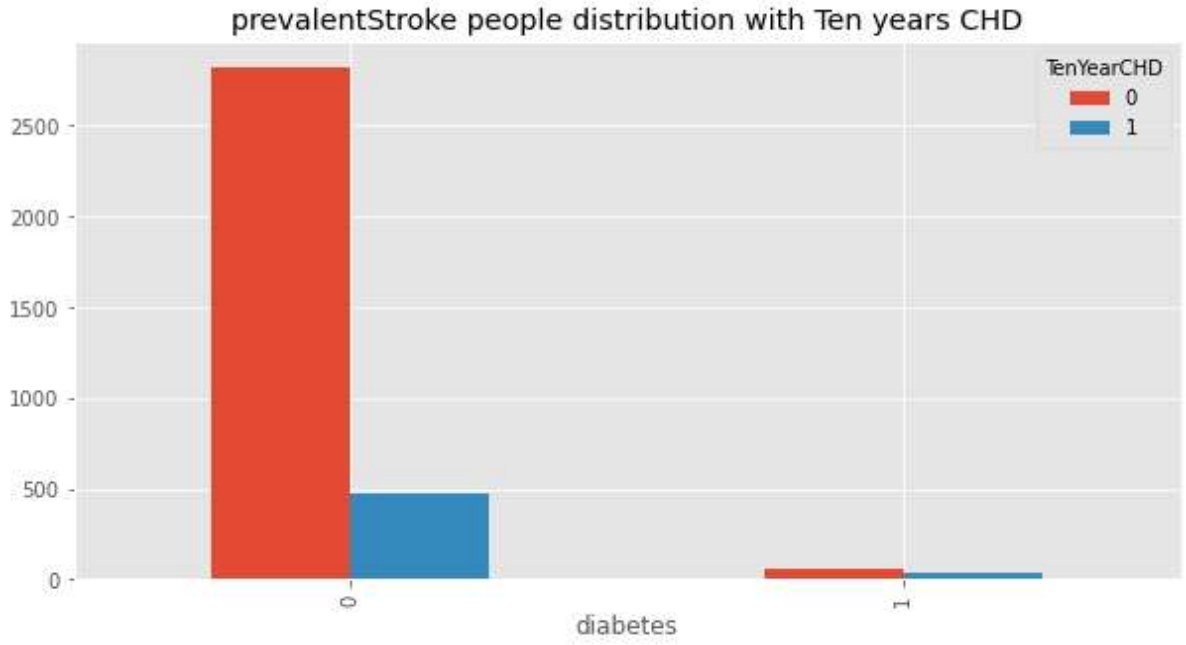
Out[56]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [57]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['diabetes','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[57]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



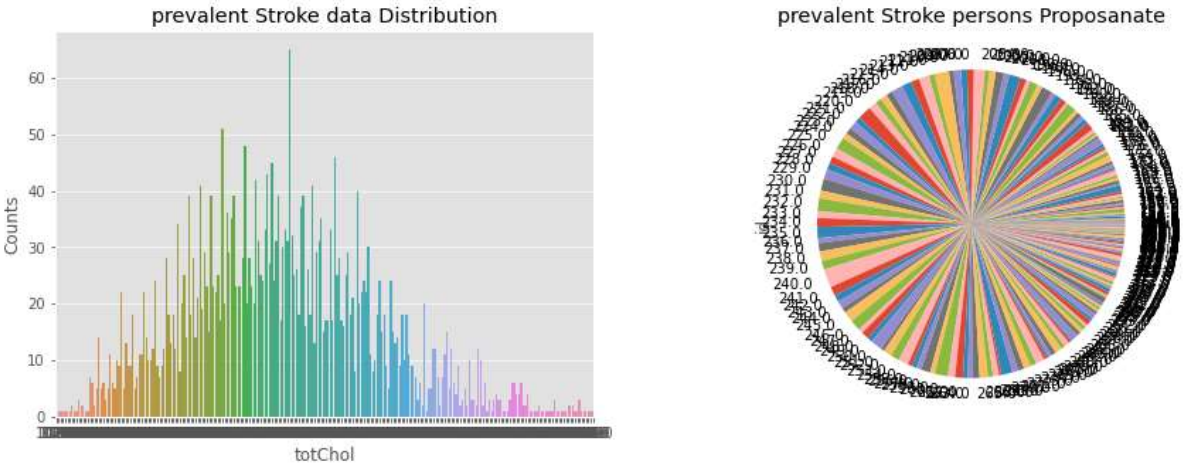
totChol

In [58]:
```
df.groupby(['totChol'])['id'].count()
```

Out[58]:
```
totChol
107.0    1
113.0    1
119.0    1
124.0    1
126.0    1
        ..
439.0    1
453.0    1
464.0    1
600.0    1
696.0    1
Name: id, Length: 240, dtype: int64
```

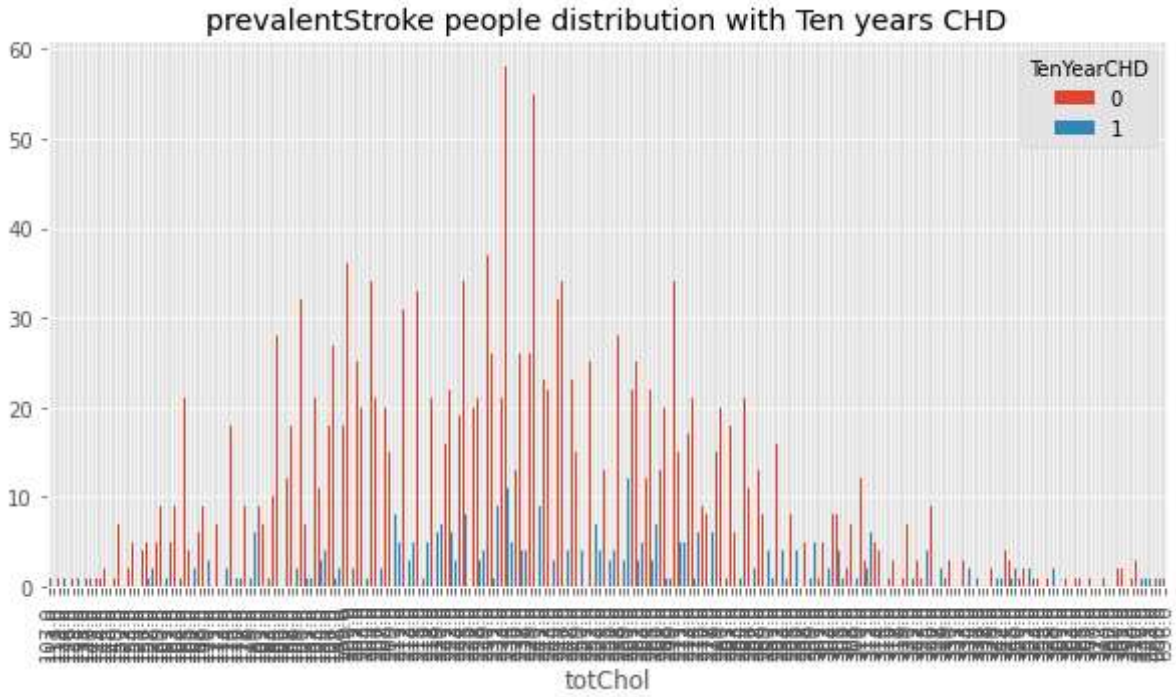
In [59]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['totChol'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('totChol')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['totChol'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

Out[59]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```



In [60]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['totChol','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[60]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



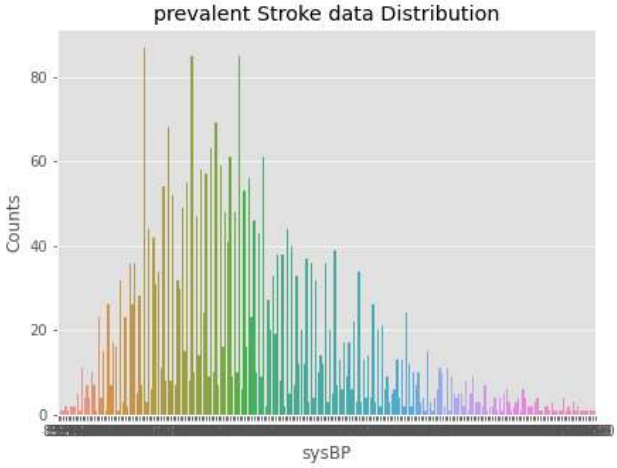
sysBP

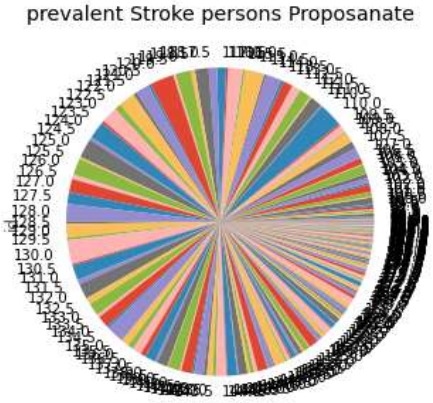
In [61]:
```
df.groupby(['sysBP'])['id'].count()
```

Out[61]:
```
sysBP
83.5     2
85.0     1
85.5     1
90.0     2
92.5     1
        ..
235.0    1
243.0    1
244.0    1
248.0    1
295.0    1
Name: id, Length: 226, dtype: int64
```

In [62]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['sysBP'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('sysBP')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['sysBP'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

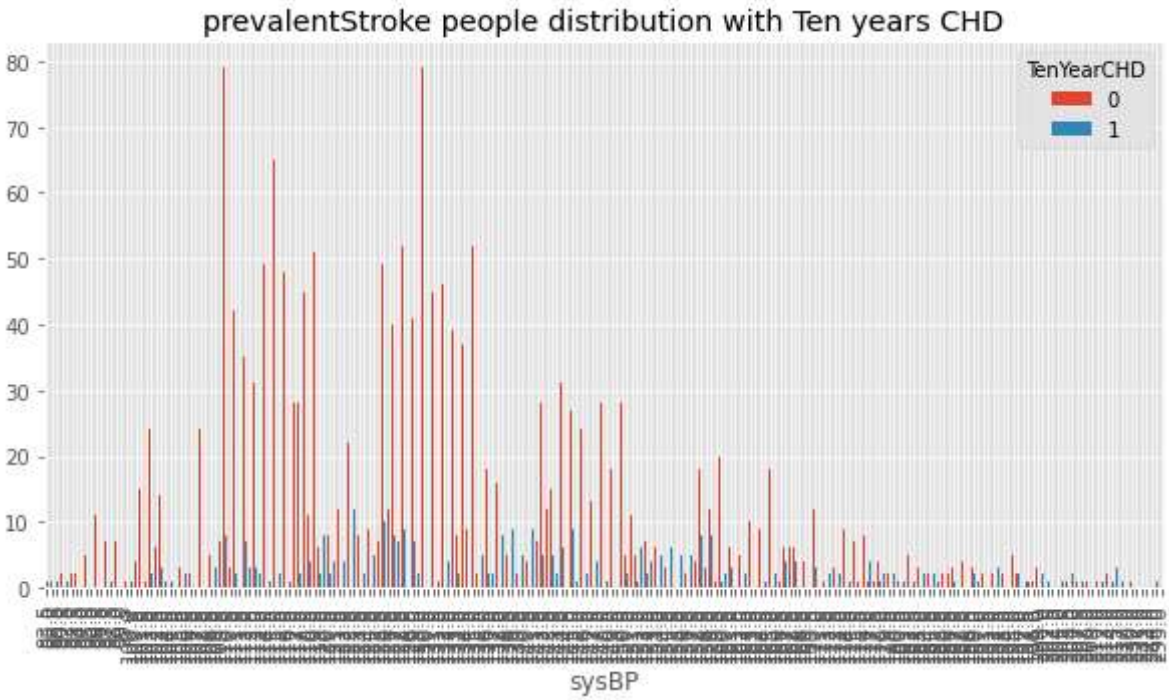
Out[62]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [63]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['sysBP','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[63]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



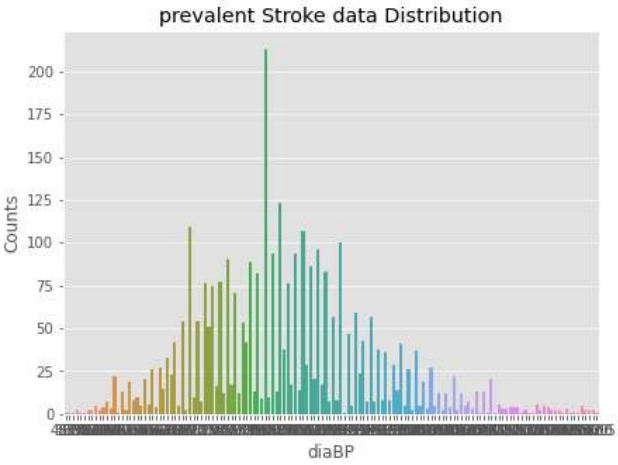
diaBP

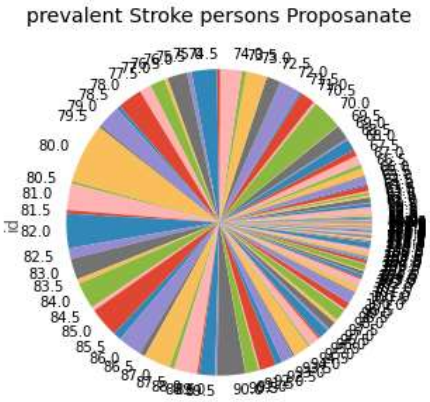
In [64]:
```
df.groupby(['diaBP'])['id'].count()
```

Out[64]:
```
diaBP
48.0     1
50.0     1
51.0     1
52.0     2
53.0     1
        ..
130.0    5
133.0    2
135.0    2
136.0    2
142.5    1
Name: id, Length: 142, dtype: int64
```

In [65]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['diaBP'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('diaBP')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['diaBP'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

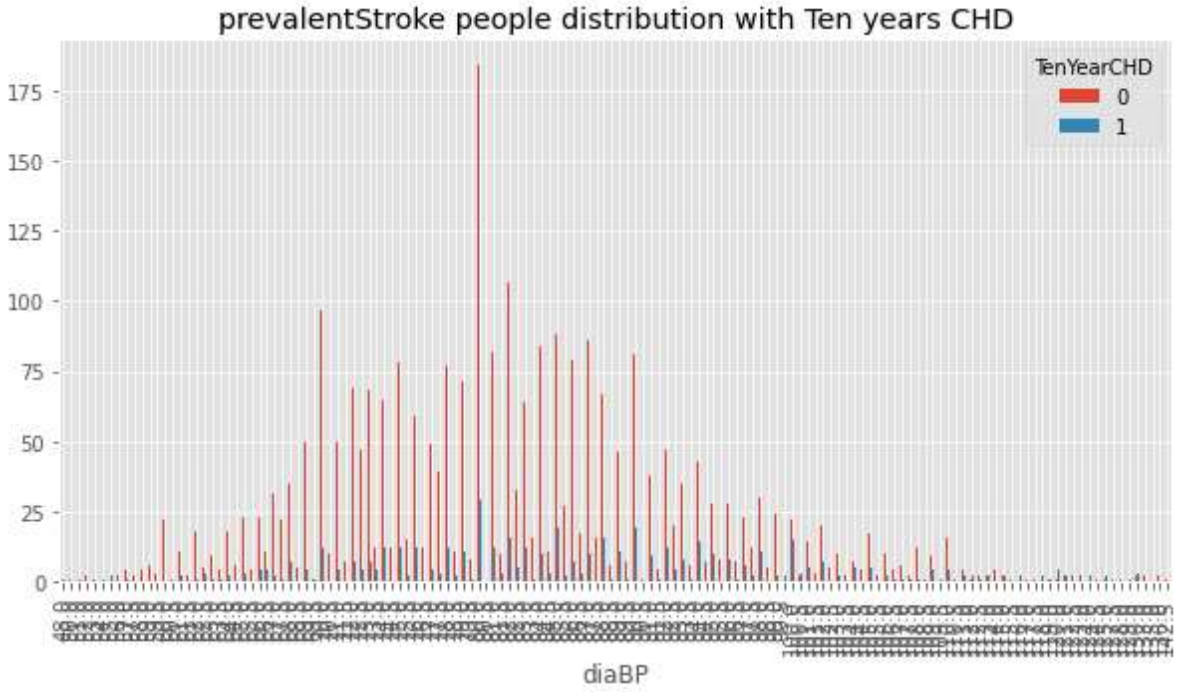
Out[65]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [66]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['diaBP','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[66]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



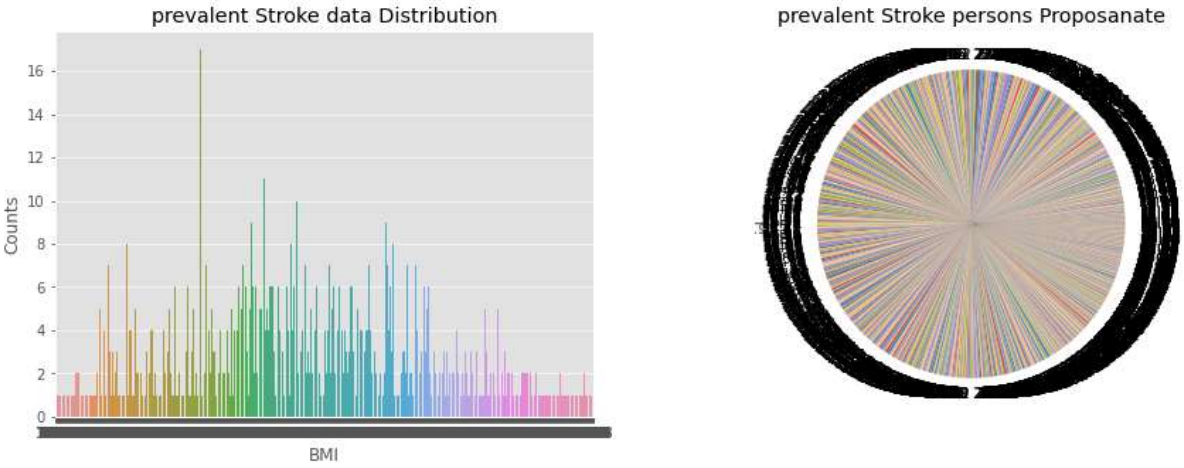
BMI

In [67]:
```
df.groupby(['BMI'])['id'].count()
```

Out[67]:
```
BMI
15.96    1
16.48    1
16.59    1
16.61    1
16.69    1
        ..
44.71    1
45.79    1
45.80    1
51.28    1
56.80    1
Name: id, Length: 1259, dtype: int64
```

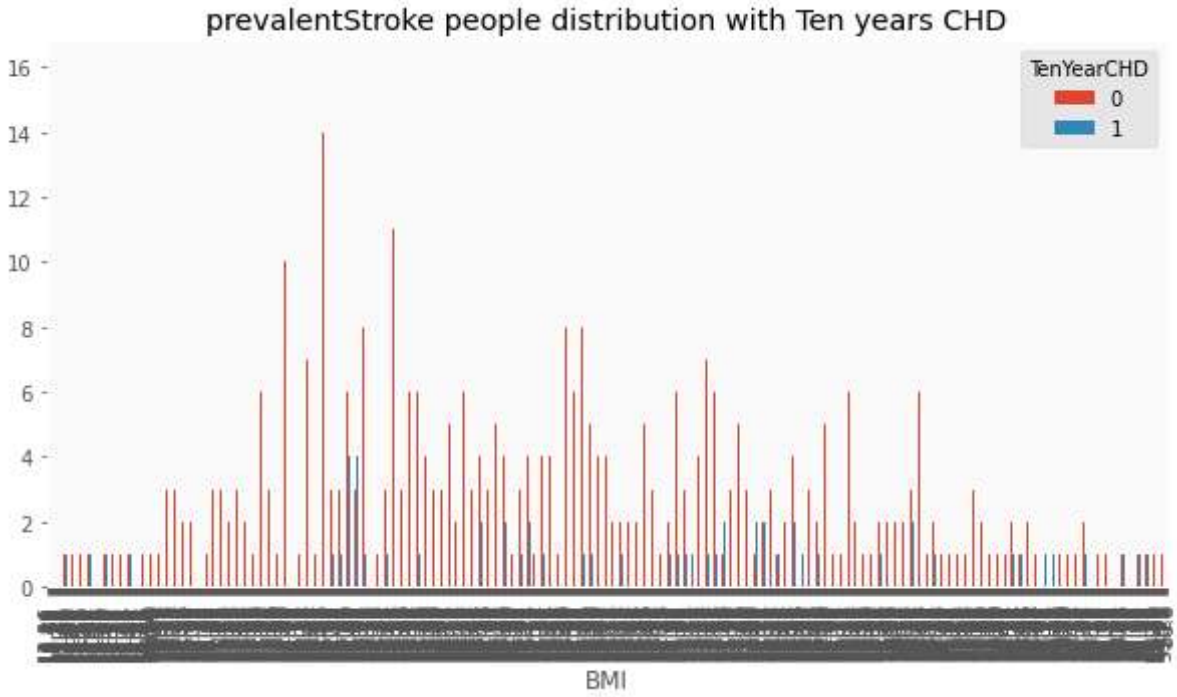
In [68]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['BMI'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('BMI')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['BMI'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

Out[68]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```



In [69]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['BMI','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[69]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



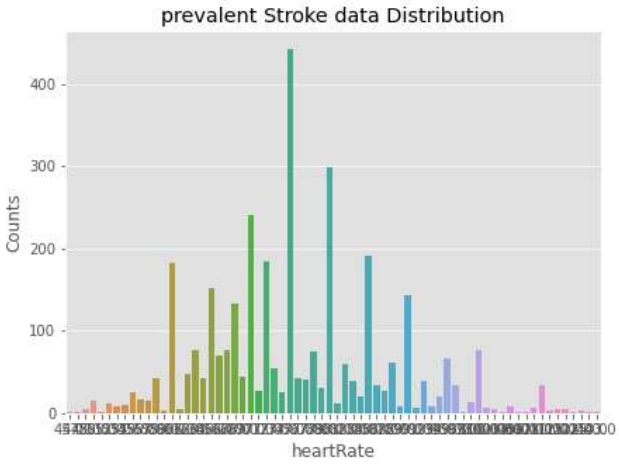
Heart Rate

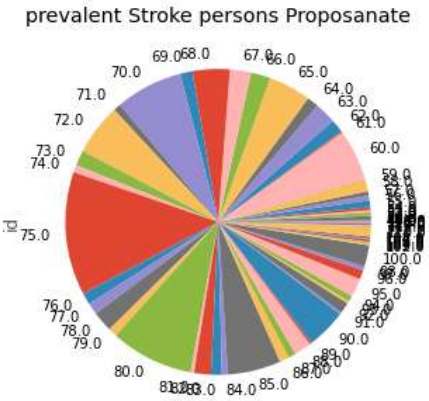
In [70]:
```
df.groupby(['heartRate'])['id'].count()
```

Out[70]:
```
heartRate
45.0      1
47.0      1
48.0      4
50.0     15
51.0      1
         ..
120.0     5
122.0     2
125.0     3
140.0     1
143.0     1
Name: id, Length: 68, dtype: int64
```

In [71]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['heartRate'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('heartRate')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['heartRate'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

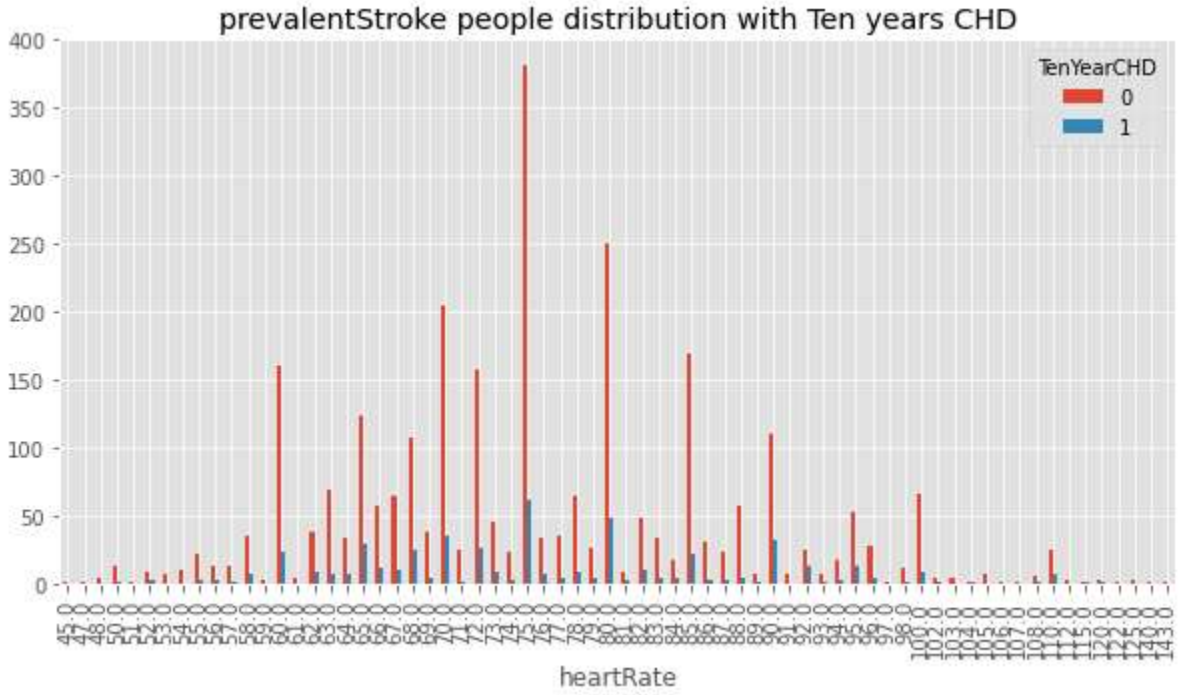
Out[71]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [72]:
```
plt.rcParams['figure.figsize'] = (10, 5)
df_copy.groupby(['heartRate','TenYearCHD'])['id'].count().unstack().plot(kind='bar'
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[72]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



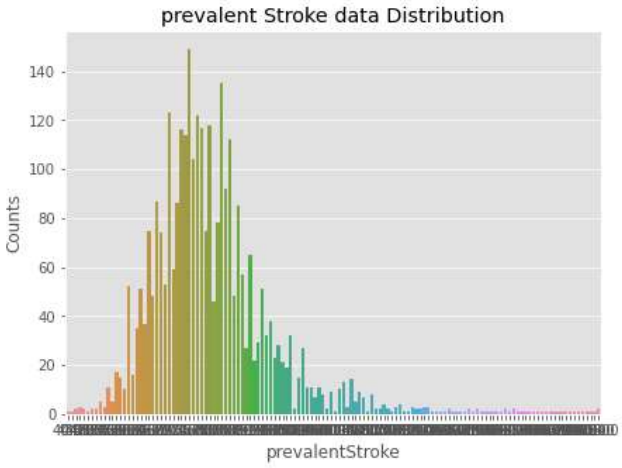
Glucose

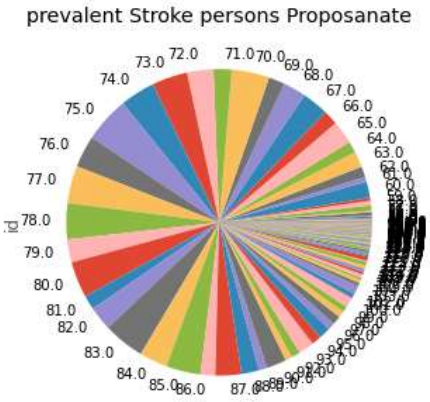
In [73]:
```
df.groupby(['glucose'])['id'].count()
```

Out[73]:
```
glucose
40.0     1
43.0     1
44.0     2
45.0     3
47.0     2
        ..
332.0    1
348.0    1
368.0    1
386.0    1
394.0    2
Name: id, Length: 132, dtype: int64
```

In [74]:
```
fig, ax = plt.subplots(1,2,figsize=(15,5))
ax1=plt.subplot(1,2,1)
Education_status=pd.DataFrame(df.groupby(['glucose'])['id'].count())
sns.barplot(x=Education_status.index,y=Education_status['id'])
plt.xlabel('prevalentStroke')
plt.ylabel('Counts')
plt.title('prevalent Stroke data Distribution')
ax2=plt.subplot(1,2,2)
df.groupby(['glucose'])['id'].count().plot(kind='pie')
plt.title('prevalent Stroke persons Proposanate')
```

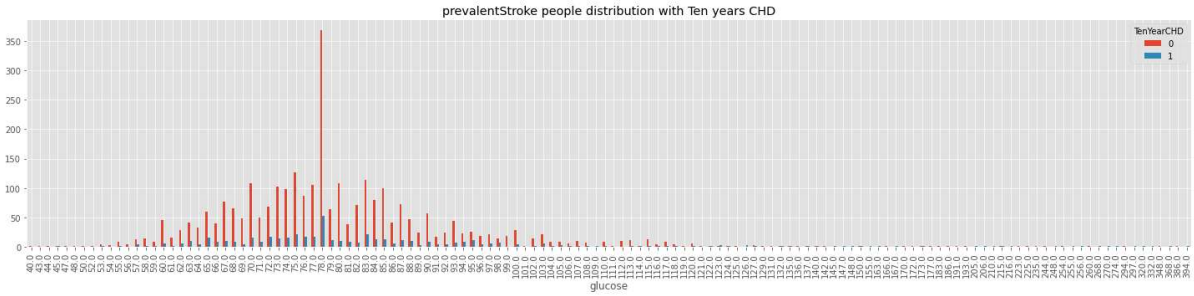
Out[74]:
```
Text(0.5, 1.0, 'prevalent Stroke persons Proposanate')
```





In [75]:
```
plt.rcParams['figure.figsize'] = (25, 5)
df_copy.groupby(['glucose','TenYearCHD'])['id'].count().unstack().plot(kind='bar')
plt.title('prevalentStroke people distribution with Ten years CHD')
```

Out[75]:
```
Text(0.5, 1.0, 'prevalentStroke people distribution with Ten years CHD')
```



In [76]:
```
le=LabelEncoder()
df_copy['sex']=le.fit_transform(df_copy['sex'])
df_copy['is_smoking']=le.fit_transform(df_copy['is_smoking'])
```

In [77]:
```
df_copy.head()
```

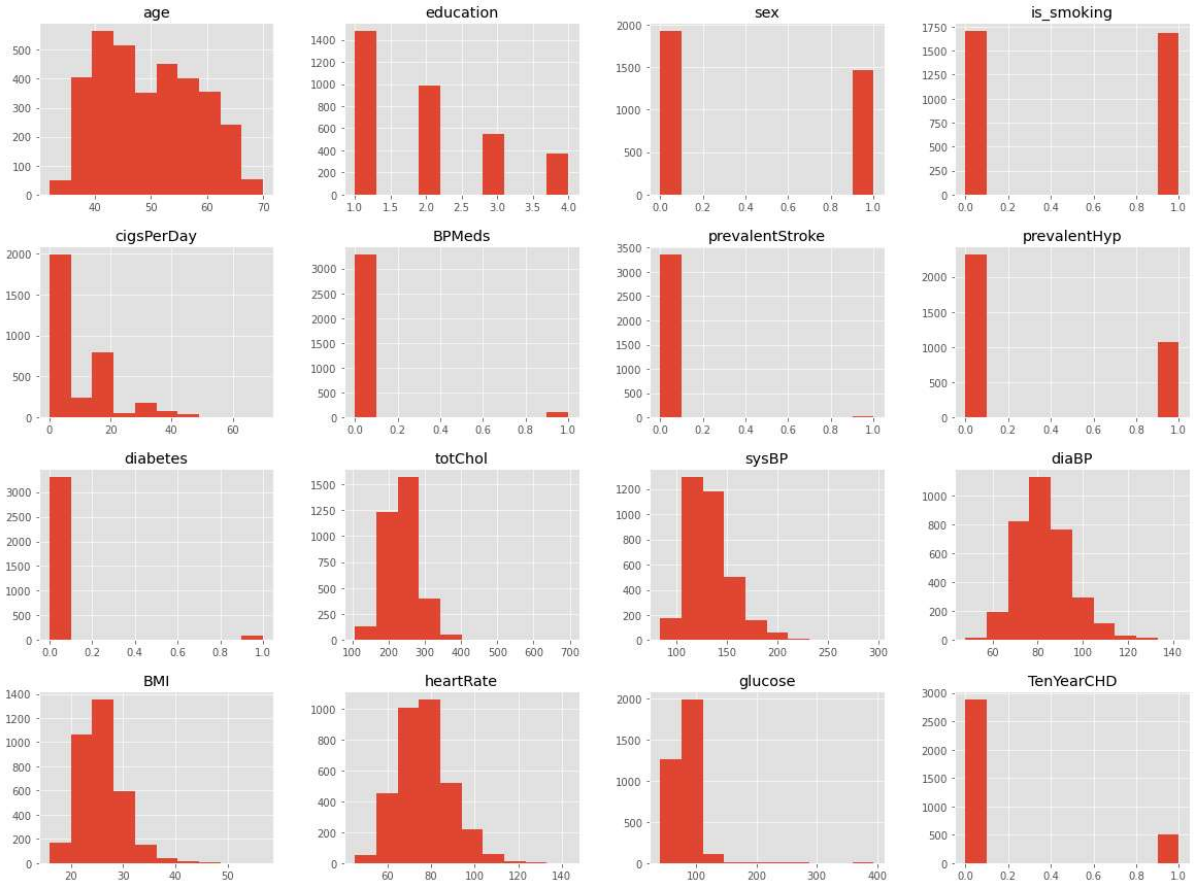
Out[77]:

	id	age	education	sex	is_smoking	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diab
0	0	64	2.0	0	1	3.0	0.0	0	0	
1	1	36	4.0	1	0	0.0	0.0	0	1	
2	2	46	1.0	0	1	10.0	0.0	0	0	
3	3	50	1.0	1	1	20.0	0.0	0	1	
4	4	64	1.0	0	1	30.0	0.0	0	0	

In [78]:
```
df_copy.drop(['id'],axis=1,inplace=True) #Id is not useful for Model training.
```

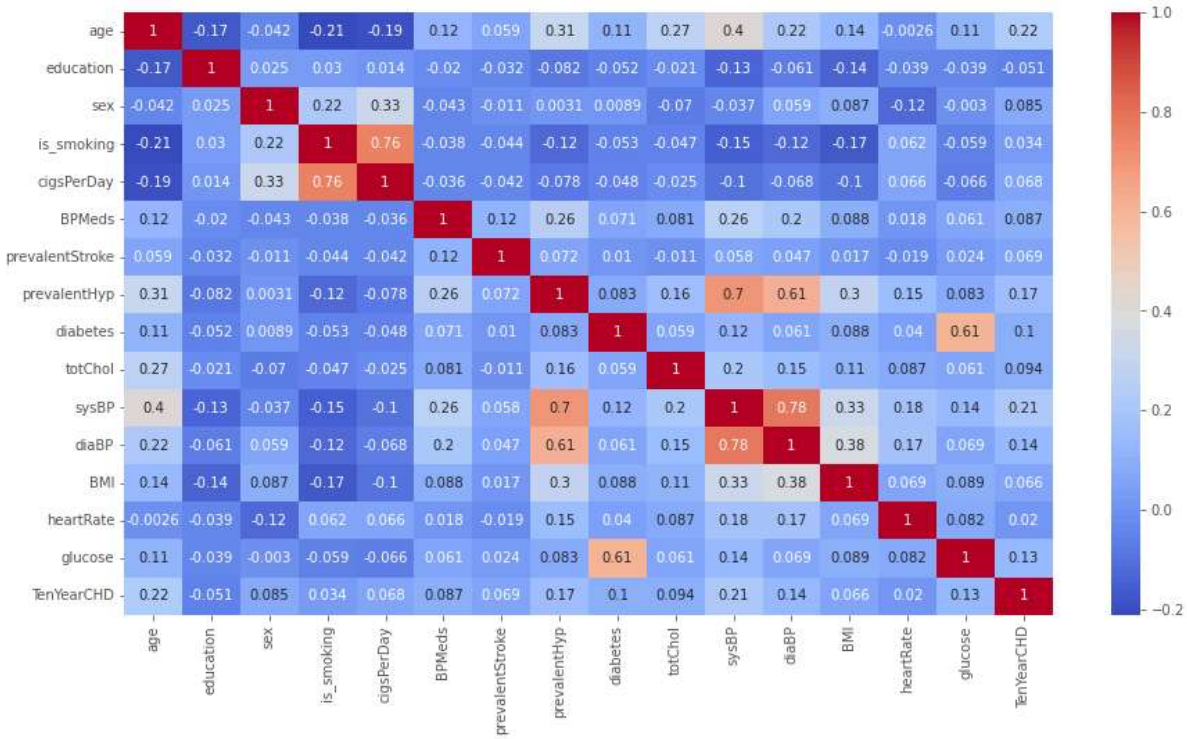
In [79]:
```
fig = plt.figure(figsize = (20,15))
ax = fig.gca()
df_copy.hist(ax = ax)
```

Out[79]:
```
array([[<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'education'}>,
        <AxesSubplot:title={'center':'sex'}>,
        <AxesSubplot:title={'center':'is_smoking'}>],
       [<AxesSubplot:title={'center':'cigsPerDay'}>,
        <AxesSubplot:title={'center':'BPMeds'}>,
        <AxesSubplot:title={'center':'prevalentStroke'}>,
        <AxesSubplot:title={'center':'prevalentHyp'}>],
       [<AxesSubplot:title={'center':'diabetes'}>,
        <AxesSubplot:title={'center':'totChol'}>,
        <AxesSubplot:title={'center':'sysBP'}>,
        <AxesSubplot:title={'center':'diaBP'}>],
       [<AxesSubplot:title={'center':'BMI'}>,
        <AxesSubplot:title={'center':'heartRate'}>,
        <AxesSubplot:title={'center':'glucose'}>,
        <AxesSubplot:title={'center':'TenYearCHD'}>]], dtype=object)
```

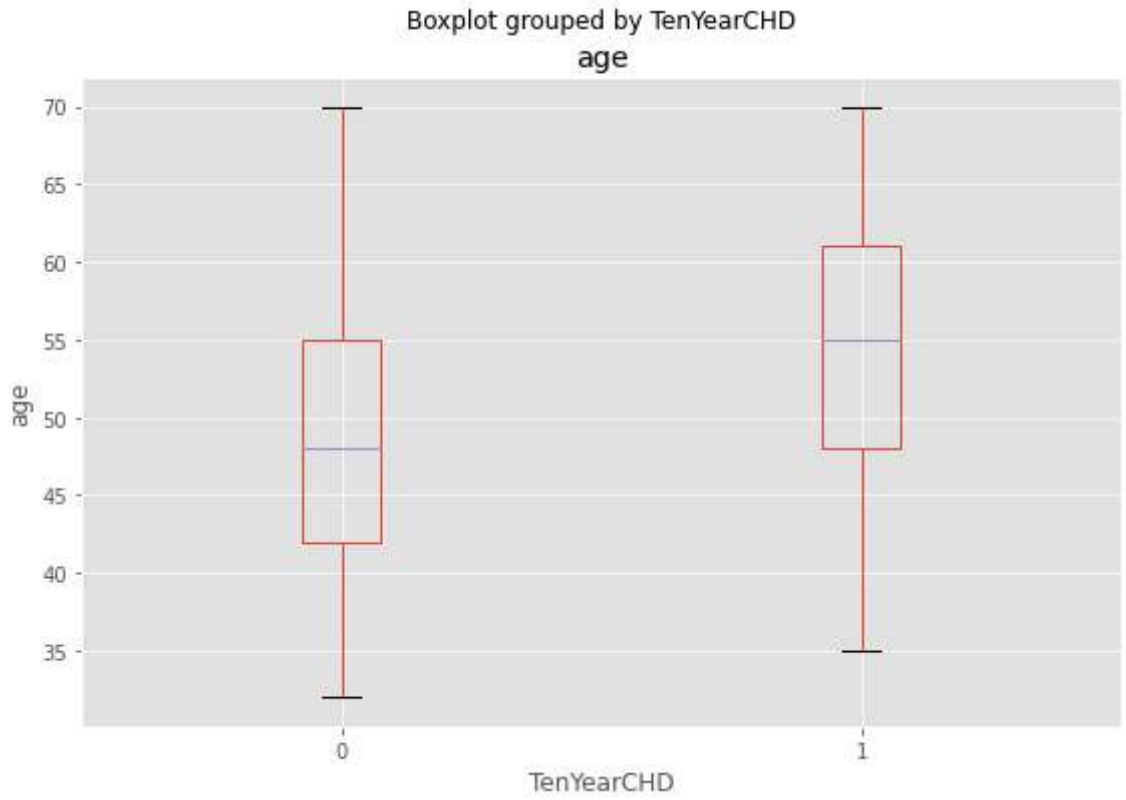


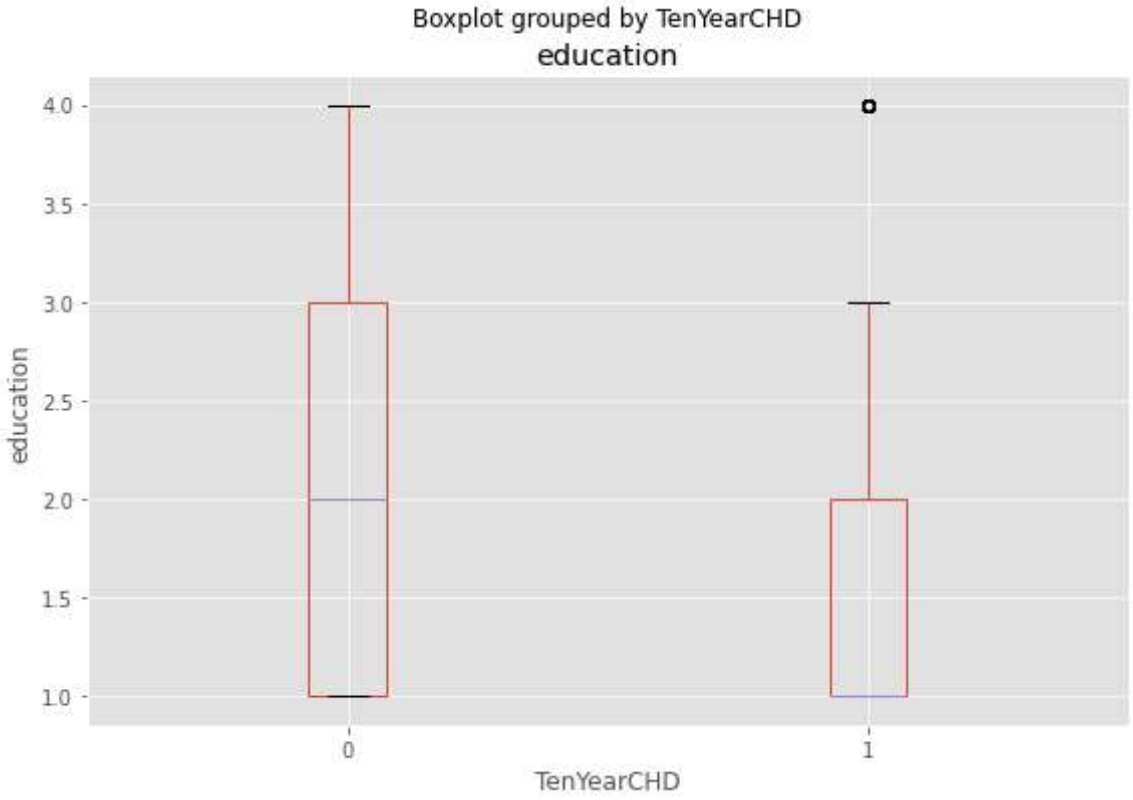
In [80]:
```
plt.figure(figsize=(15,8))
correlation = df_copy.corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm')
```

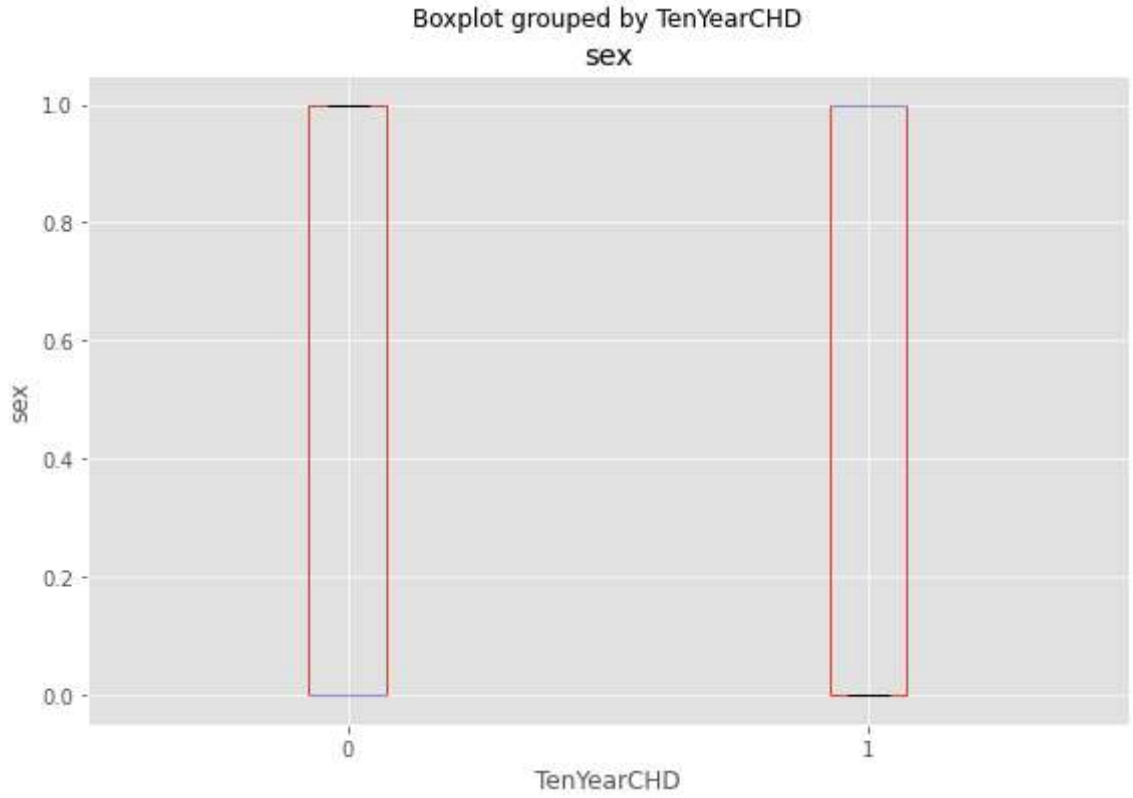
Out[80]:
```
<AxesSubplot:>
```

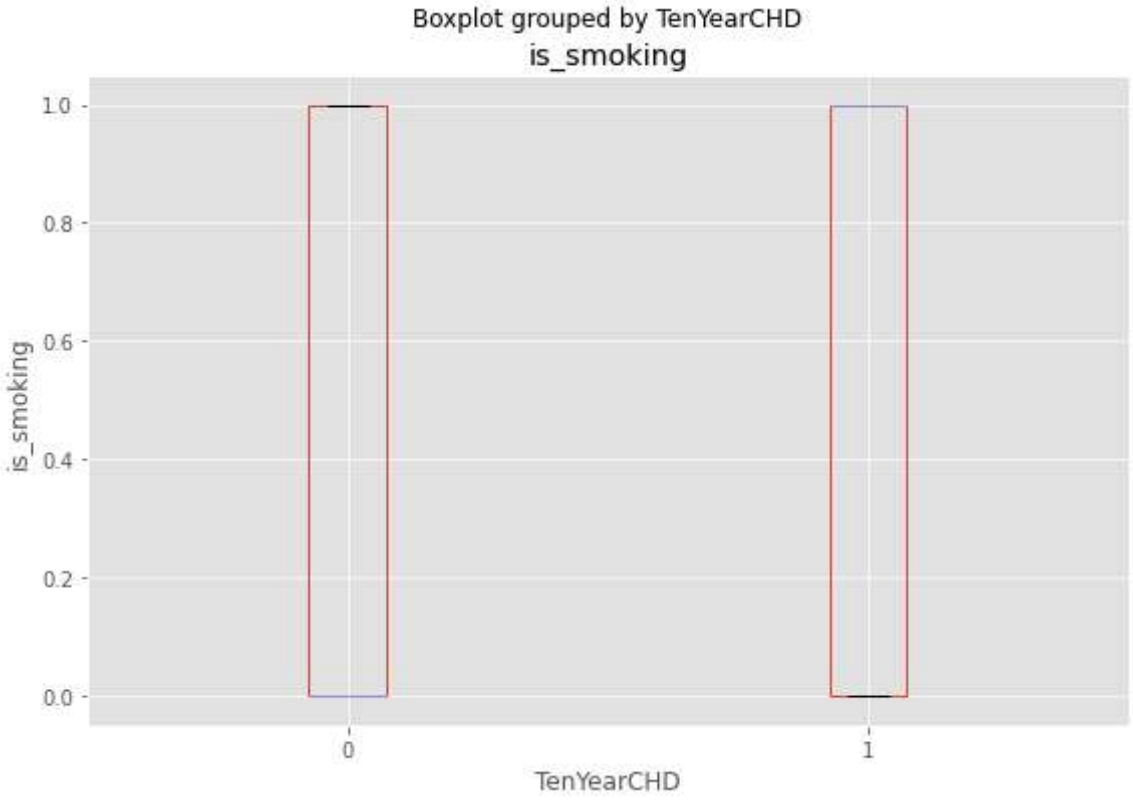


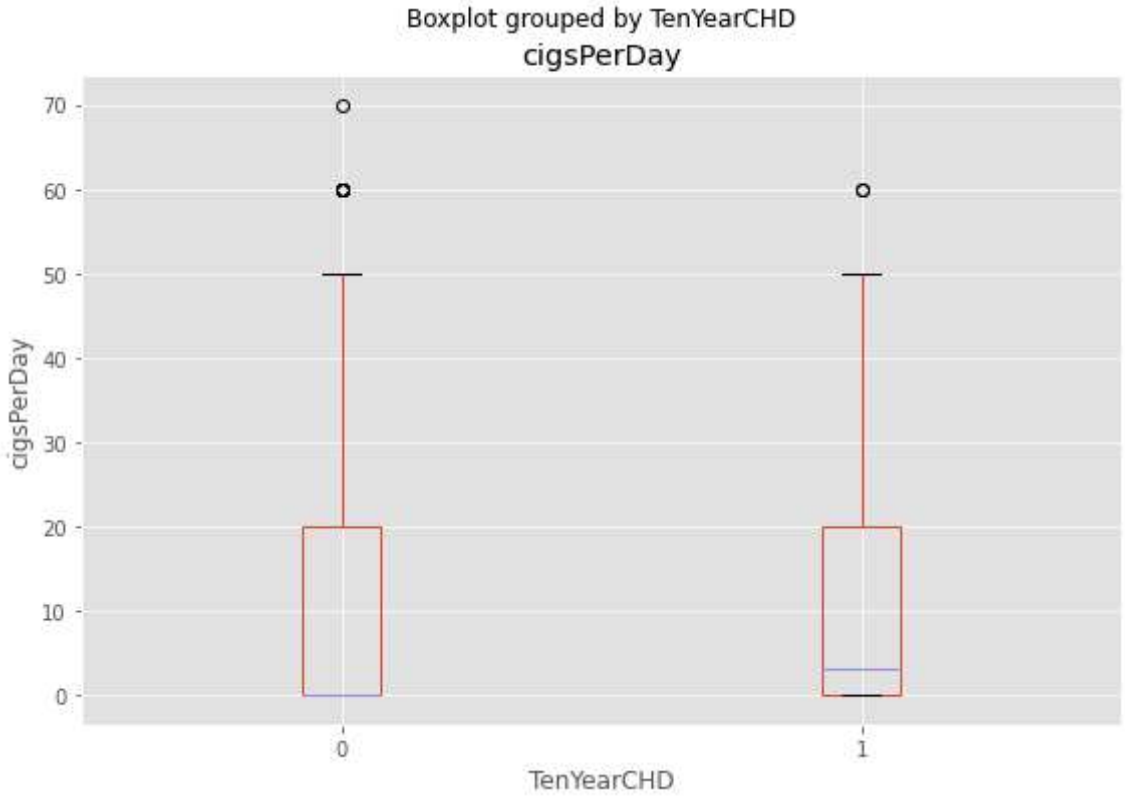
In [81]:
```
for i in df_copy.columns[:-1]:
  fig = plt.figure(figsize=(9, 6))
  ax = fig.gca()
  df_copy.boxplot(column = i, by = 'TenYearCHD', ax = ax)
  ax.set_ylabel(i)
plt.show()
```

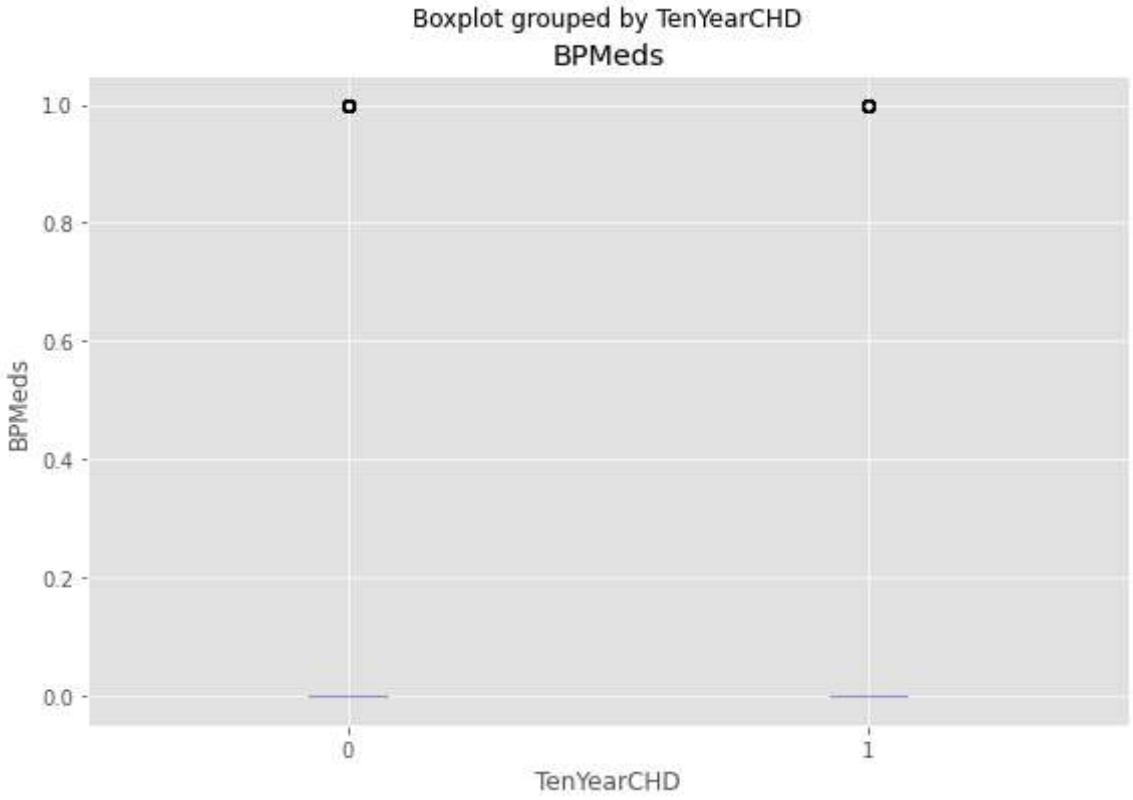


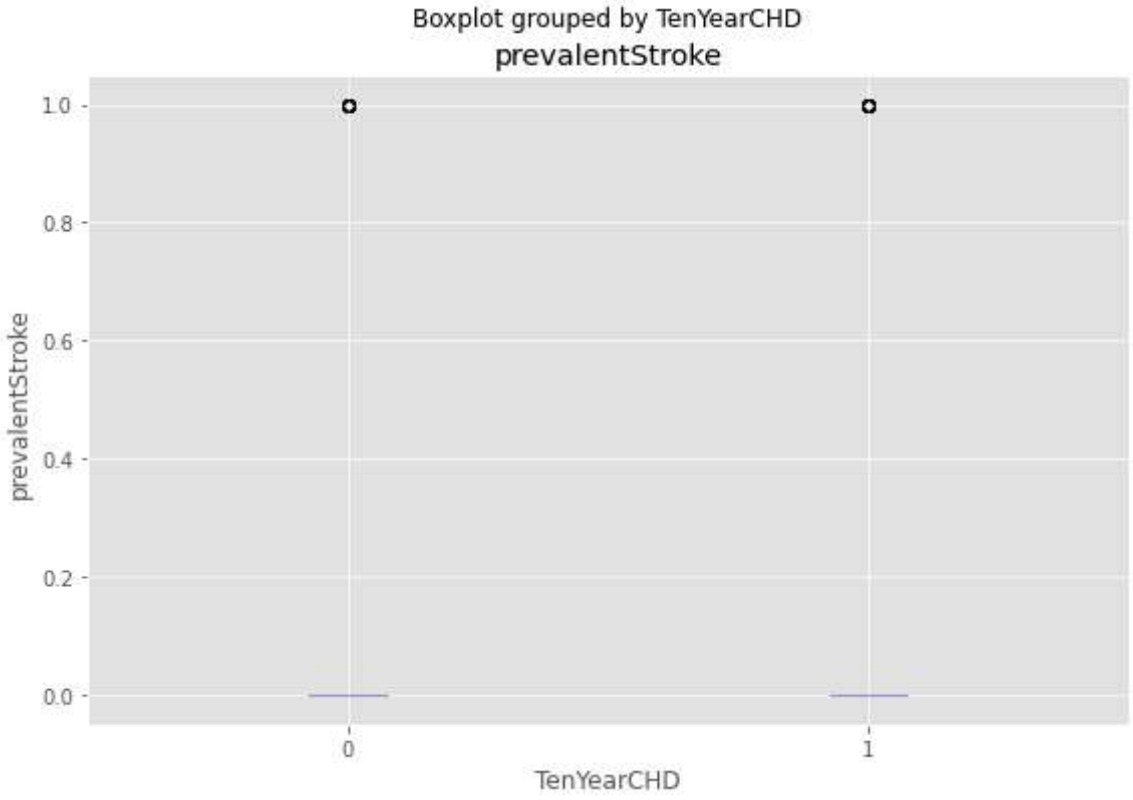

Boxplot grouped by TenYearCHD
education
4.0
3.5
3.0
2.5
2.0
1.5
1.0
education
0
1
TenYearCHD

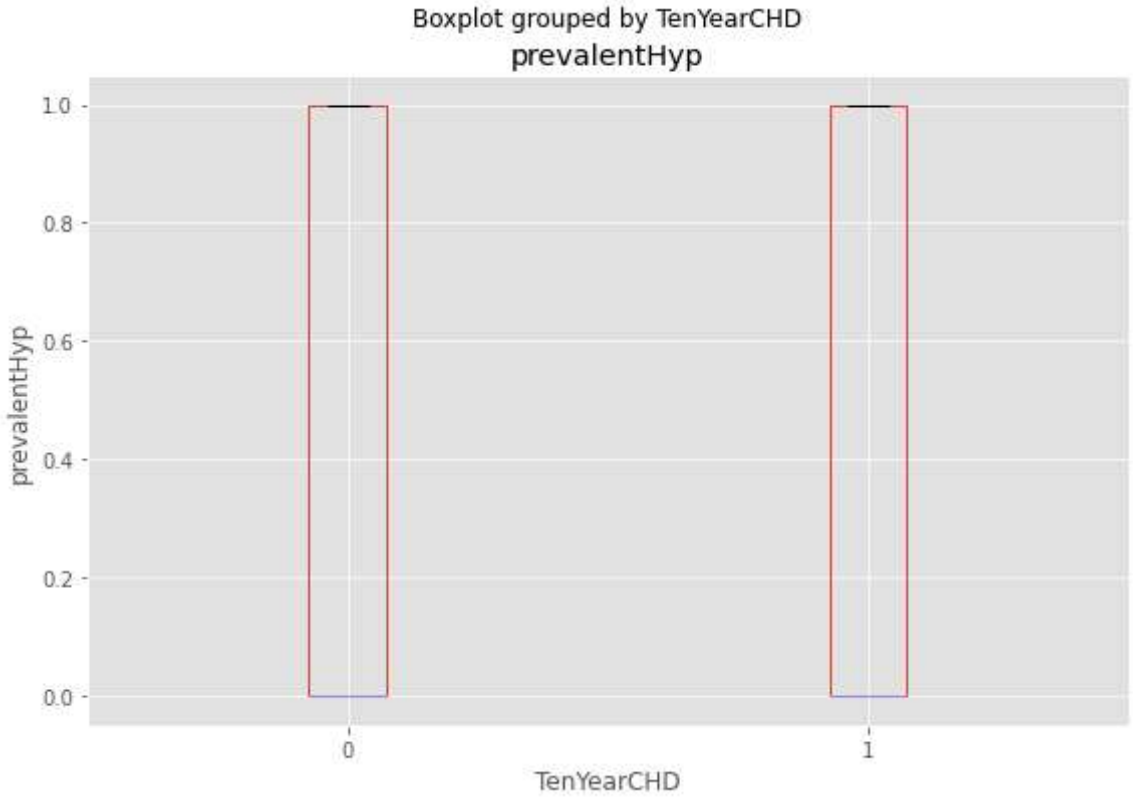

Boxplot grouped by TenYearCHD
sex
1.0
0.8
0.6
0.4
0.2
0.0
sex
0
1
TenYearCHD

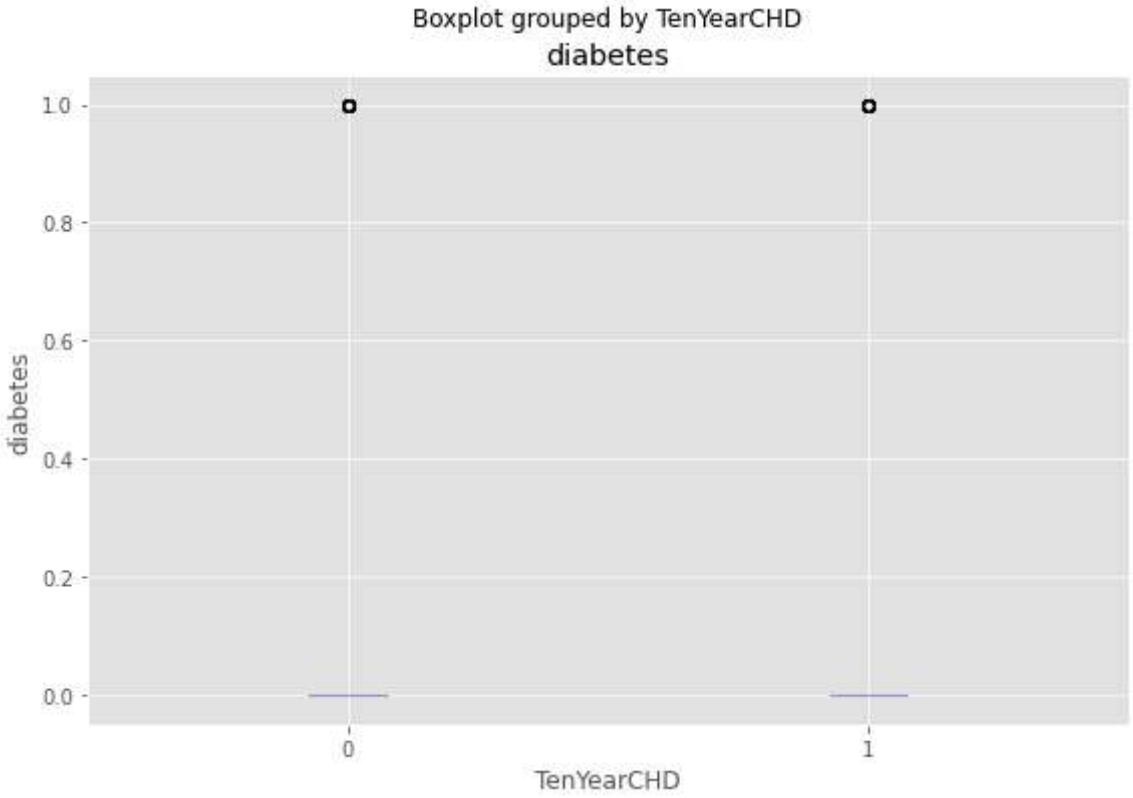

Boxplot grouped by TenYearCHD
is_smoking
1.0
0.8
0.6
0.4
0.2
0.0
is_smoking
0
1
TenYearCHD

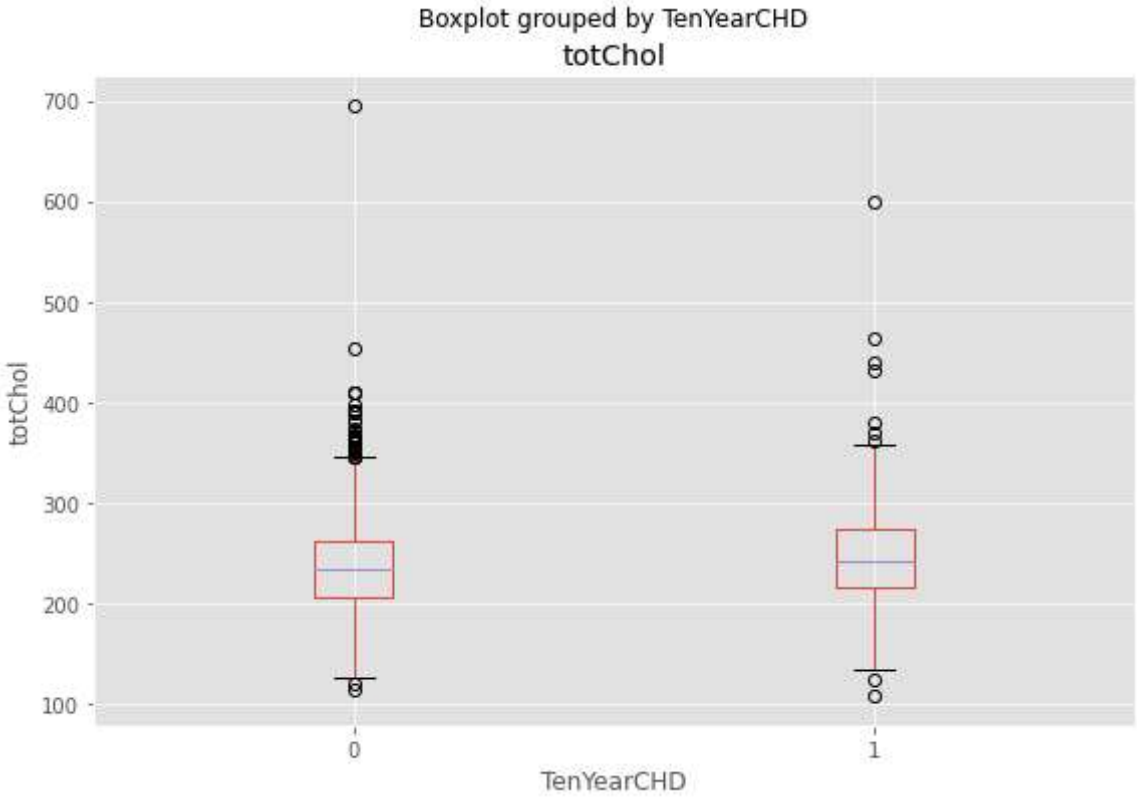

Boxplot grouped by TenYearCHD
cigsPerDay
70
60
50
40
30
20
10
0
cigsPerDay
0
1
TenYearCHD

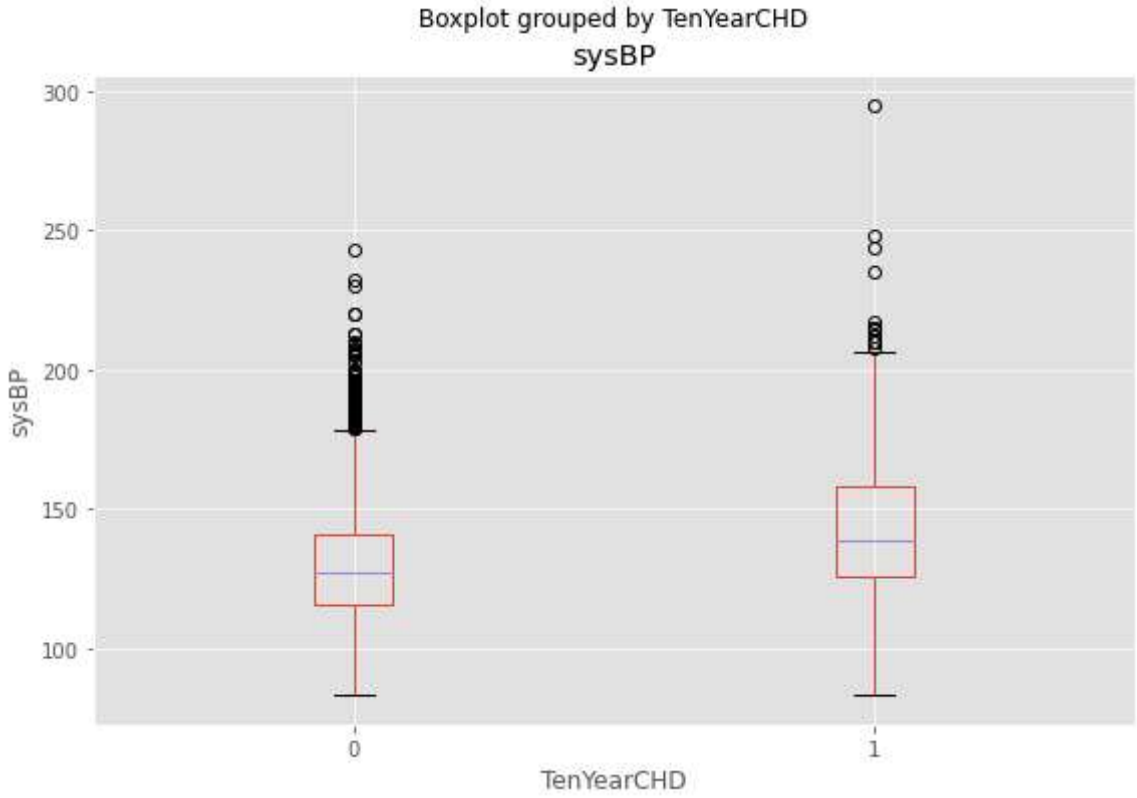

Boxplot grouped by TenYearCHD
BPMeds
BPMeds
1.0
0.8
0.6
0.4
0.2
0.0
0
1
TenYearCHD

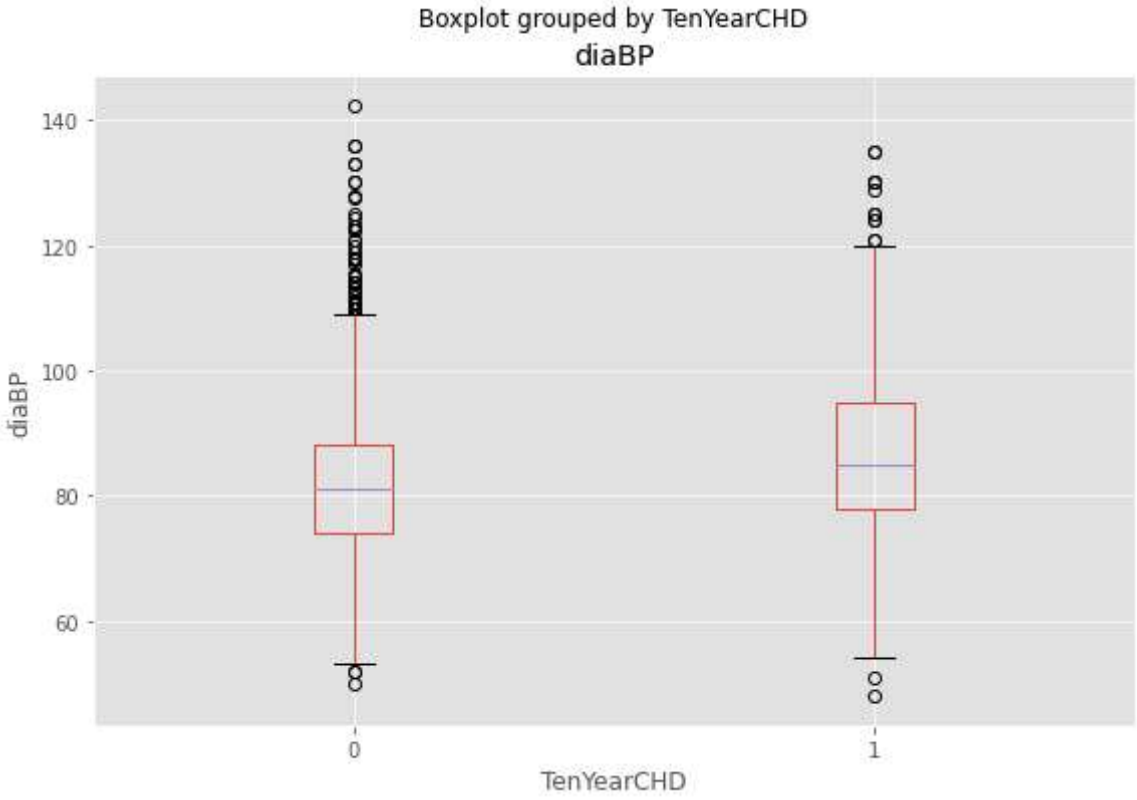

Boxplot grouped by TenYearCHD
prevalentStroke
prevalentStroke
1.0
0.8
0.6
0.4
0.2
0.0
0
1
TenYearCHD

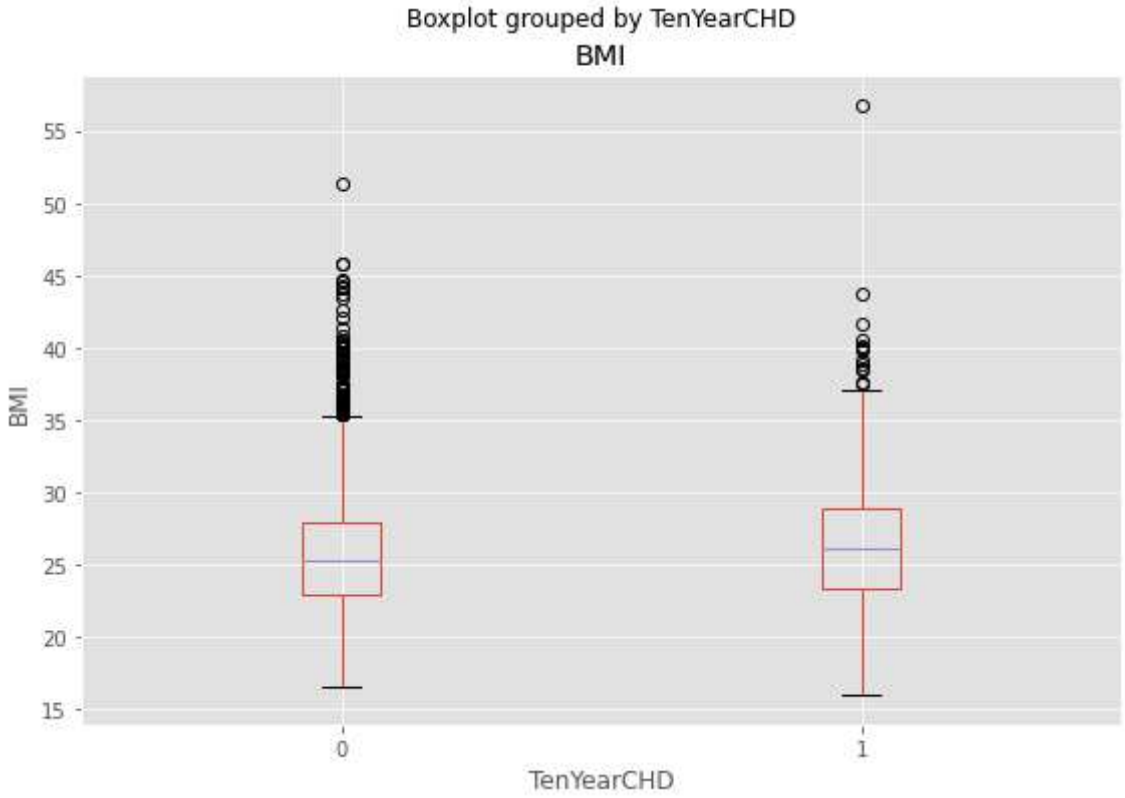

Boxplot grouped by TenYearCHD
prevalentHyp
prevalentHyp
1.0
0.8
0.6
0.4
0.2
0.0
0
1
TenYearCHD

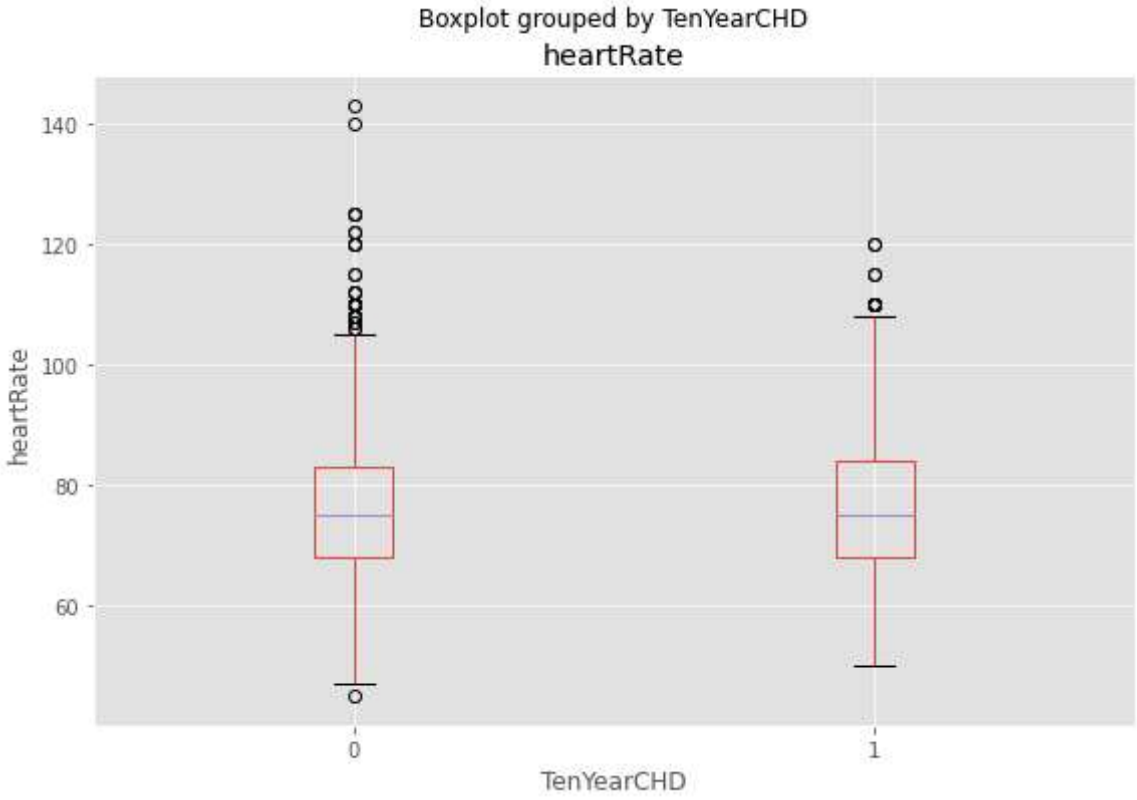

Boxplot grouped by TenYearCHD
diabetes
diabetes
1.0
0.8
0.6
0.4
0.2
0.0
0
1
TenYearCHD

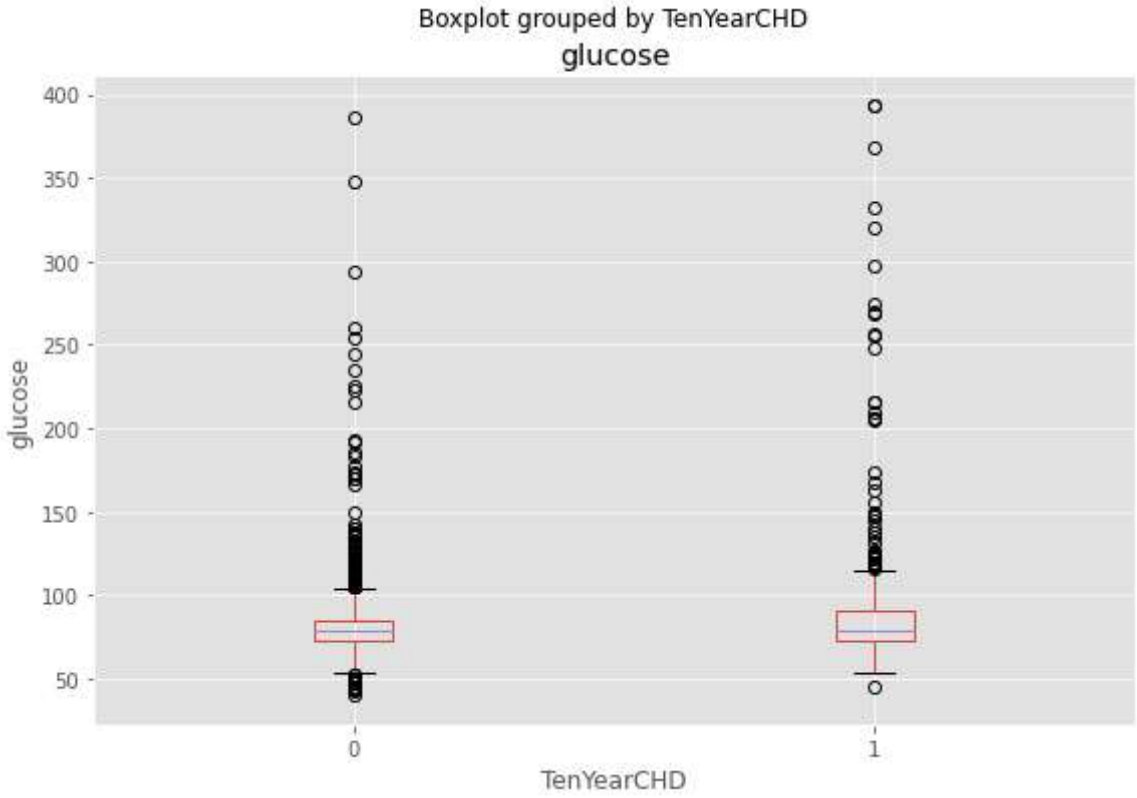

Boxplot grouped by TenYearCHD
totChol
700
600
500
400
300
200
100
totChol
0
1
TenYearCHD


Boxplot grouped by TenYearCHD
sysBP
300
250
200
150
100
sysBP
0
1
TenYearCHD


Boxplot grouped by TenYearCHD
diaBP
140
120
100
80
60
diaBP
0
1
TenYearCHD


Boxplot grouped by TenYearCHD
BMI
55
50
45
40
35
30
25
20
15
BMI
0
1
TenYearCHD





Classfication- Mechine Learning

Data Splitting

In [82]:
```
X=df_copy[['age', 'education', 'sex', 'is_smoking', 'cigsPerDay', 'BPMeds',
       'prevalentStroke', 'prevalentHyp', 'diabetes', 'totChol', 'sysBP',
        'diaBP', 'BMI', 'heartRate', 'glucose']].copy()
y=df_copy['TenYearCHD'].copy()
```

In [83]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X,y , test_size = 0.2, random_
print(X_train.shape)
print(X_test.shape)
```

```
(2712, 15)
(678, 15)
```

In [84]:
```
y_train.value_counts()
```

Out[84]:
```
0    2305
1     407
Name: TenYearCHD, dtype: int64
```

In [85]:
```
y_test.value_counts()
```

Out[85]:
```
0    574
1    104
Name: TenYearCHD, dtype: int64
```

Logistic Regression

In [86]:
```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(fit_intercept=True, max_iter=10000)
clf.fit(X_train, y_train)
```

Out[86]:
```
LogisticRegression(max_iter=10000)
```

In [87]:
```
# Get the model coefficients
clf.coef_
```

Out[87]:
```
array([[ 0.06212838, -0.02507037,  0.49230777,  0.0174176 ,  0.02358251,
        -0.13485735,  1.10667872,  0.02272118,  0.02131566,  0.00331402,
         0.0196609 , -0.00199628, -0.00907629, -0.00607949,  0.00966585]])
```

In [88]:
```
clf.intercept_
```

Out[88]:
```
array([-8.80449634])
```

In [89]:
```
# Get the predicted classes
train_class_preds = clf.predict(X_train)
test_class_preds = clf.predict(X_test)
```

In [90]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
```

In [91]:
```
# Get the accuracy scores
train_accuracy = accuracy_score(train_class_preds,y_train)
test_accuracy = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy)
print("The accuracy on test data is ", test_accuracy)
```

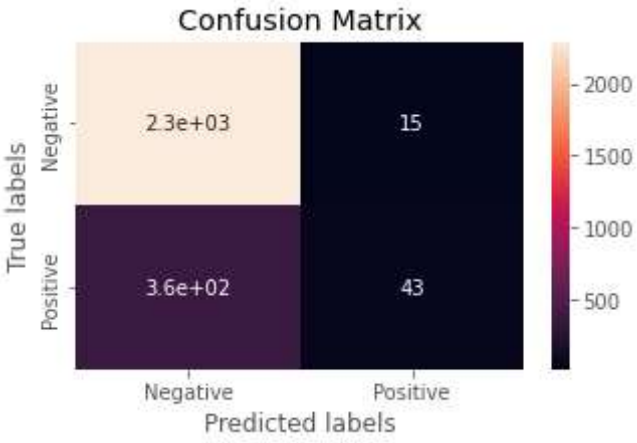
```
The accuracy on train data is  0.8602507374631269
The accuracy on test data is  0.8480825958702065
```

In [92]:
```
# Get the confusion matrix for both train and test
plt.figure(figsize=(5,3))
labels = ['Negative', 'Positive']
cm = confusion_matrix(y_train, train_class_preds)
print(cm)
```

```
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax) #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

Out[92]:

```
[[2290   15]
 [ 364   43]]
[Text(0, 0.5, 'Negative'), Text(0, 1.5, 'Positive')]
```

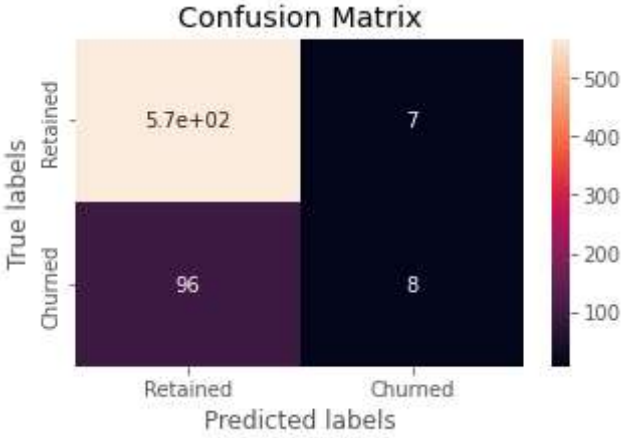


In [93]:

```
# Get the confusion matrix for both train and test
plt.figure(figsize=(5,3))
labels = ['Retained', 'Churned']
cm = confusion_matrix(y_test, test_class_preds)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

Out[93]:

```
[[567   7]
 [ 96   8]]
[Text(0, 0.5, 'Retained'), Text(0, 1.5, 'Churned')]
```



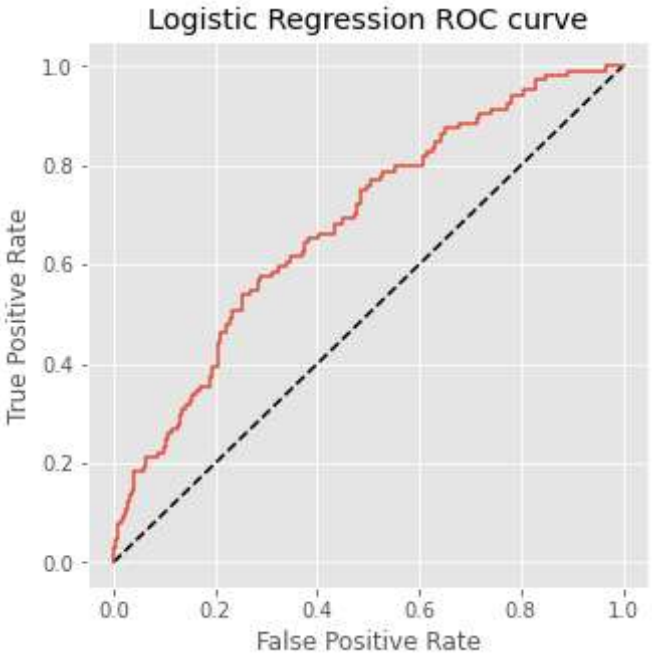
In [94]:
```
y_lr_predict_pro=clf.predict_proba(X_test)[:,1]
```

In [95]:
```
fpr, tpr, thresholds = roc_curve(y_test, y_lr_predict_pro)
```

In [96]:
```
roc_auc_score(y_test,y_lr_predict_pro)
```

Out[96]:
```
0.6813354328598231
```

In [97]:
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC curve')
plt.show()
```



Handling Data Imbalalance

In [98]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='minority')
X_sm, y_sm = smote.fit_resample(X, y)
# X_sm, y_sm = smote.fit(X,y)
```

In [99]:
```
y_sm=pd.DataFrame(y_sm)
```

In [100…
```
y_sm.value_counts()
```

Out[100]:
```
TenYearCHD
0             2879
1             2879
dtype: int64
```

Logistic Regression

In [101…
```
X_train, X_test, y_train, y_test = train_test_split( X_sm,y_sm , test_size = 0.2, r
print(X_train.shape)
print(X_test.shape)
```

```
(4606, 15)
(1152, 15)
```

In [102…
```
y_train.value_counts()
```

Out[102]:
```
TenYearCHD
0             2326
1             2280
dtype: int64
```

In [103…
```
y_test.value_counts()
```

Out[103]:
```
TenYearCHD
1             599
0             553
dtype: int64
```

In [104…
```
clf = LogisticRegression(fit_intercept=True, max_iter=10000)
clf.fit(X_train, y_train)
```

Out[104]:
```
LogisticRegression(max_iter=10000)
```

In [105…
```
# Get the predicted classes
train_class_preds = clf.predict(X_train)
test_class_preds = clf.predict(X_test)
```

In [106…
```
# Get the accuracy scores
train_accuracy = accuracy_score(train_class_preds,y_train)
test_accuracy = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy)
print("The accuracy on test data is ", test_accuracy)
```

```
The accuracy on train data is  0.6726009552757273
The accuracy on test data is  0.6701388888888888
```

In [107…
```
# Get the confusion matrix for both train and test

cm = confusion_matrix(y_train, train_class_preds)
print('Confusion Matrix for training Data')
print(cm)
cm = confusion_matrix(y_test, test_class_preds)
print('Confusion Matrix for Test Data')
print(cm)
```

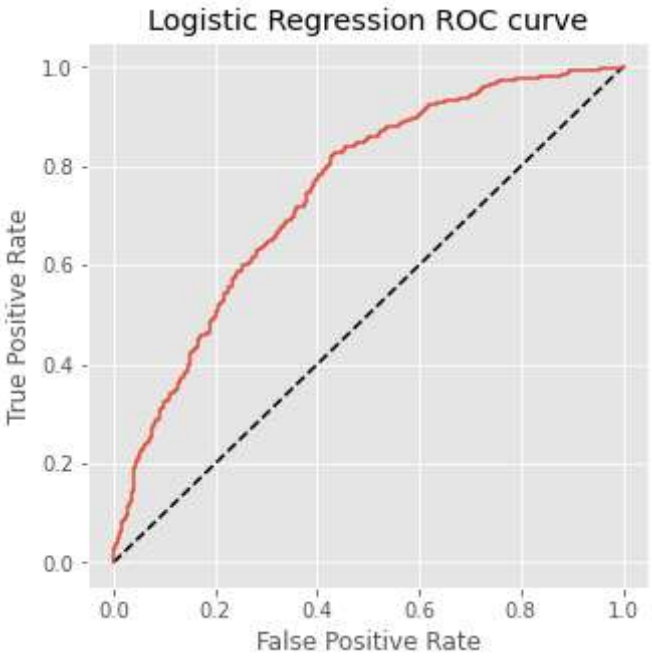
```
Confusion Matrix for training Data
[[1585  741]
 [ 767 1513]]
Confusion Matrix for Test Data
[[378 175]
 [205 394]]
```

In [108…
```
y_lr_predict_pro=clf.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_lr_predict_pro)
```

In [109…
```
roc_auc_score(y_test,y_lr_predict_pro)
```

Out[109]:
```
0.7447493864095373
```

In [110…
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC curve')
plt.show()
```

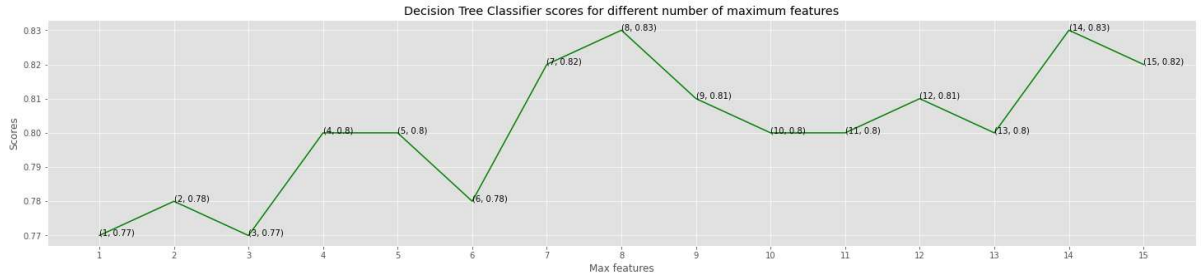


Decision Tree Classifier

In [111…
```
dt_scores = []
for i in range(1, len(X.columns) + 1):
    dt_classifier = DecisionTreeClassifier(max_features = i, random_state = 0)
    dt_classifier.fit(X_train, y_train)
    dt_scores.append(round(dt_classifier.score(X_test, y_test),2))
```

In [112…
```
plt.plot([i for i in range(1, len(X.columns) + 1)], dt_scores, color = 'green')
for i in range(1, len(X.columns) + 1):
    plt.text(i, dt_scores[i-1], (i, dt_scores[i-1]))
plt.xticks([i for i in range(1, len(X.columns) + 1)])
plt.xlabel('Max features')
plt.ylabel('Scores')
plt.title('Decision Tree Classifier scores for different number of maximum features
```

Out[112]:
```
Text(0.5, 1.0, 'Decision Tree Classifier scores for different number of maximum fe
atures')
```

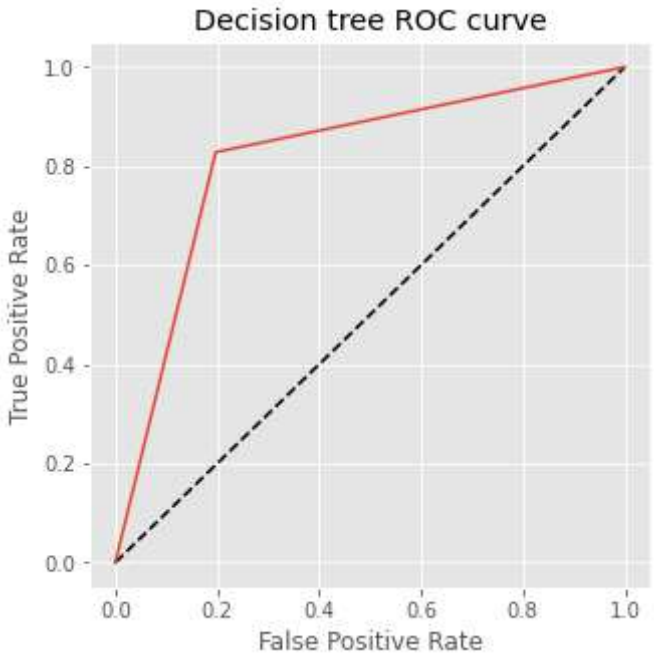


In [113…
```
y_dt_predict_pro=dt_classifier.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_dt_predict_pro)
```

In [114…
```
roc_auc_score(y_test,y_dt_predict_pro)
```

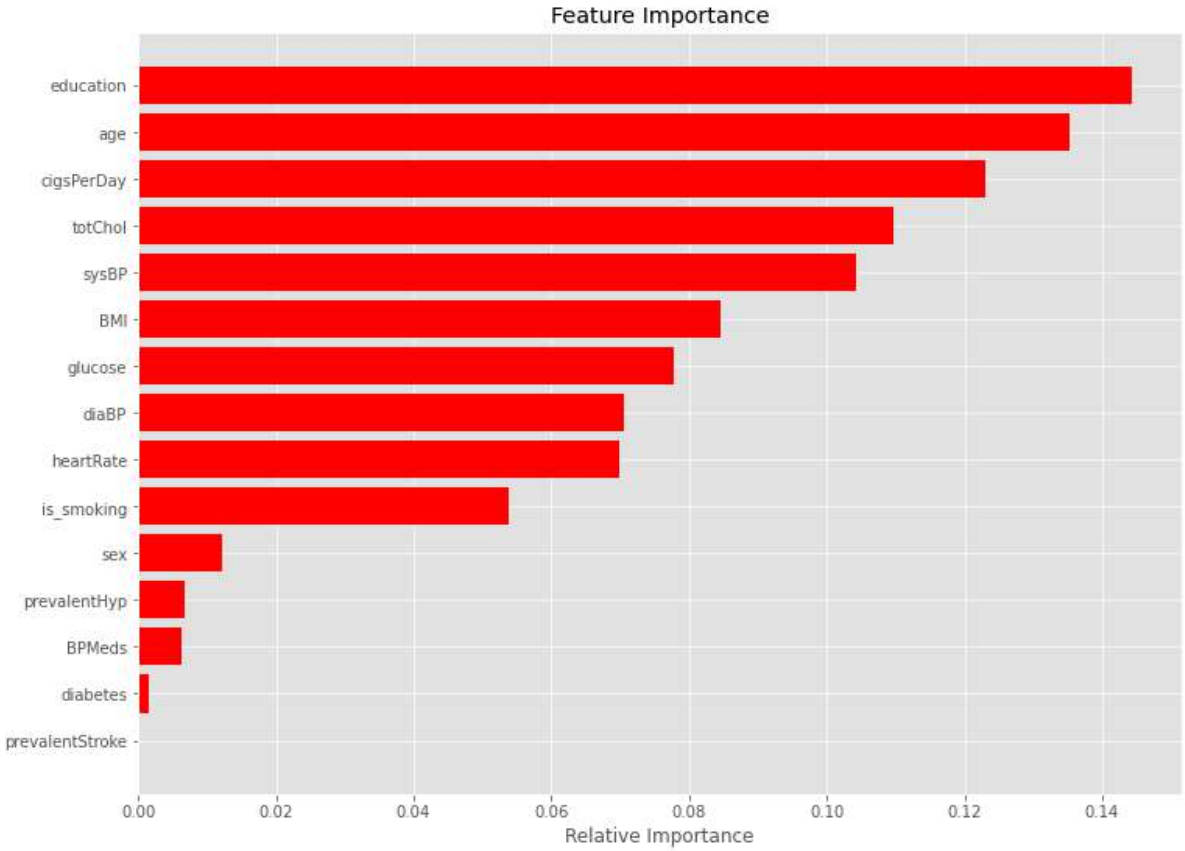
Out[114]:
```
0.8154700268983568
```

In [115…
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Decision Tree')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Decision tree ROC curve')
plt.show()
```



In [116…
```
features = X.columns
importances = dt_classifier.feature_importances_
indices = np.argsort(importances)
```

In [117…
```
plt.figure(figsize=(12,9))
plt.title('Feature Importance')
plt.barh(range(len(indices)), importances[indices], color='red', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Random Forest

In [118…
```
classifier = RandomForestClassifier() # For GBM, use GradientBoostingClassifier()
grid_values = {'n_estimators':[50, 65, 80, 95,120], 'max_depth':[3, 5, 7,9,12]}
GSclassifier = GridSearchCV(classifier, param_grid = grid_values, scoring = 'roc_au

# Fit the object to train dataset
GSclassifier.fit(X_train, y_train)
```

Out[118]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [3, 5, 7, 9, 12],
                         'n_estimators': [50, 65, 80, 95, 120]},
             scoring='roc_auc')
```

In [119…
```
bestvalues=GSclassifier.best_params_
GSclassifier.best_params_
```

Out[119]:
```
{'max_depth': 12, 'n_estimators': 120}
```

In [120…
```
classifier = RandomForestClassifier(max_depth=bestvalues['max_depth'],n_estimators=

classifier.fit(X_train, y_train)
```

Out[120]:
```
RandomForestClassifier(max_depth=12, n_estimators=120)
```

In [121…
```
y_train_preds_rf =  classifier.predict(X_train)
y_test_preds_rf= classifier.predict(X_test)
```

In [122…
```
# Obtain accuracy on train set
accuracy_score(y_train,y_train_preds_rf)
```

Out[122]:
```
0.9897959183673469
```

In [123…
```
# Obtain accuracy on test set
accuracy_score(y_test,y_test_preds_rf)
```

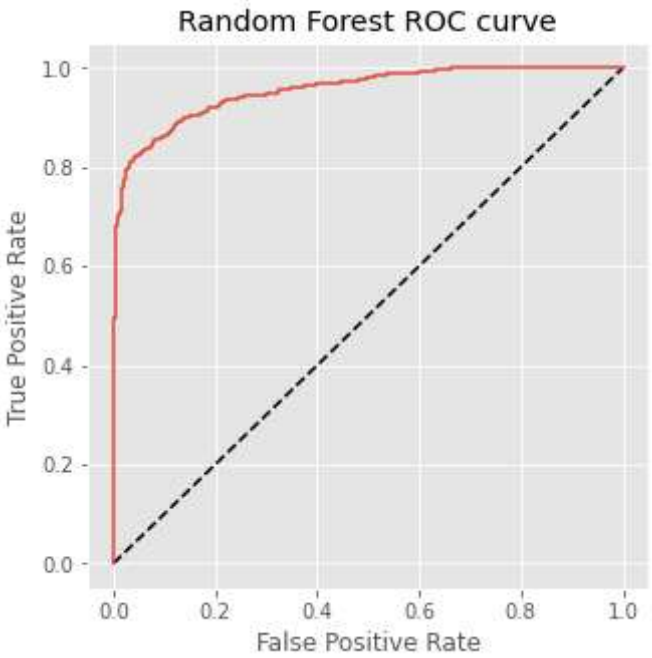
Out[123]:
```
0.8802083333333334
```

In [124…
```
y_rf_predict_pro=classifier.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_rf_predict_pro)
```

In [125…
```
roc_auc_score(y_test,y_rf_predict_pro)
```

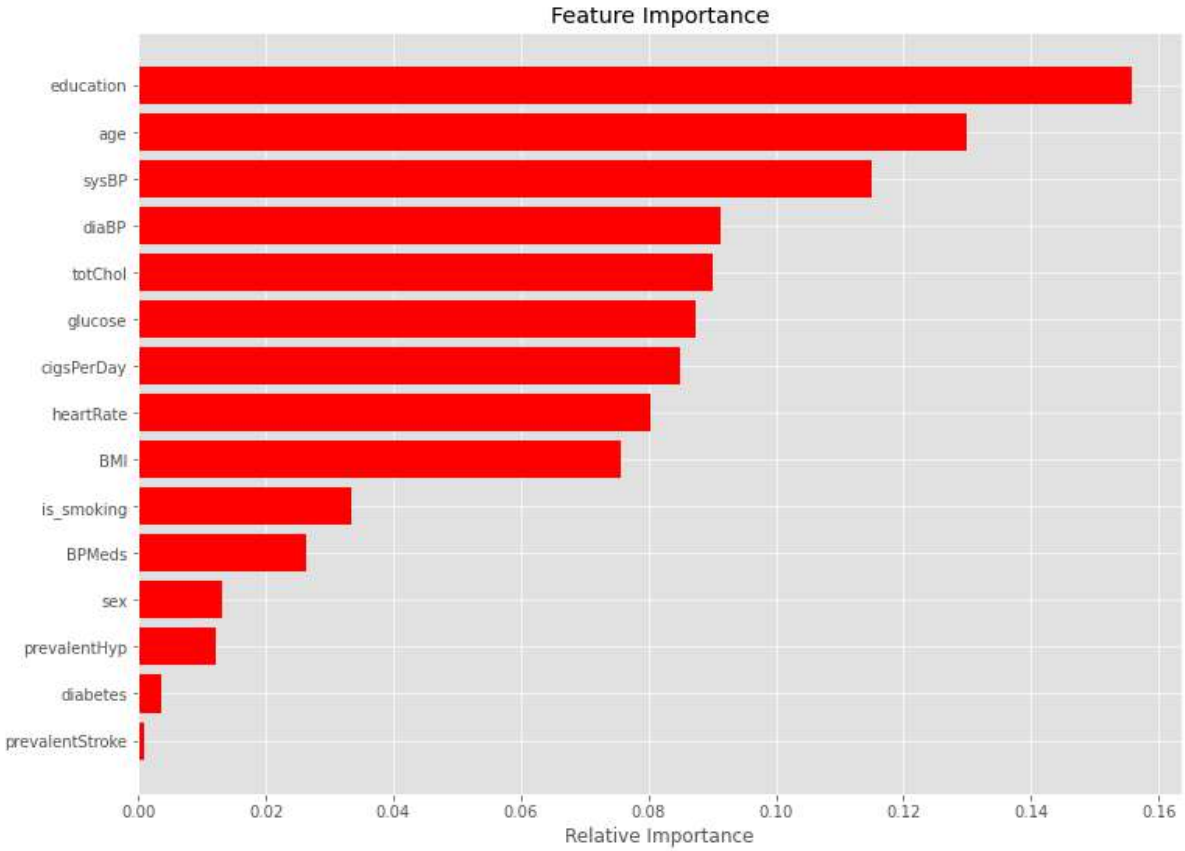
Out[125]:
```
0.9545173239304809
```

In [126…
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Random Forest')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC curve')
plt.show()
```



In [127…
```
features = X.columns
importances = classifier.feature_importances_
indices = np.argsort(importances)
```

In [128…
```
plt.figure(figsize=(12,9))
plt.title('Feature Importance')
plt.barh(range(len(indices)), importances[indices], color='red', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



KNN

In [129…
```
param_grid = {'n_neighbors':np.arange(1,50)}
```

In [130…
```
knn = KNeighborsClassifier()
knn_cv= GridSearchCV(knn,param_grid,cv=5)
knn_cv.fit(X_sm,y_sm)
```

Out[130]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9,
10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])})
```

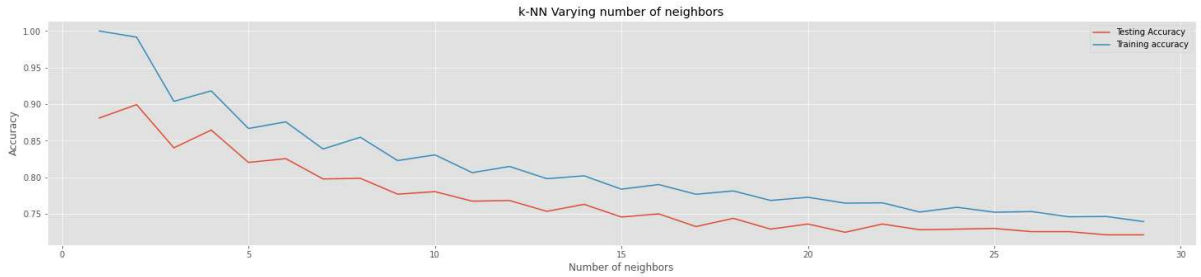
In [131…
```
bestPermet=knn_cv.best_params_
knn_cv.best_params_
```

Out[131]:
```
{'n_neighbors': 2}
```

In [132…
```
#Setup arrays to store training and test accuracies
neighbors = np.arange(1,30)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    # Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    # Fit the model
    knn.fit(X_train, y_train)

    # Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)
```

```
    # Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

In [133…
```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



In [134…
```
# Setup a knn classifier with k neighbors
knn = KNeighborsClassifier(n_neighbors=bestPermet['n_neighbors'])
```

In [135…
```
# Fit the model
knn.fit(X_train,y_train)
```

Out[135]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [136…
```
knn.score(X_test,y_test)
```

Out[136]:
```
0.8993055555555556
```

In [137…
```
y_test_pred_knn = knn.predict(X_test)
```

In [138…
```
confusion_matrix(y_test,y_test_pred_knn)
```

Out[138]:
```
array([[459,  94],
       [ 22, 577]], dtype=int64)
```

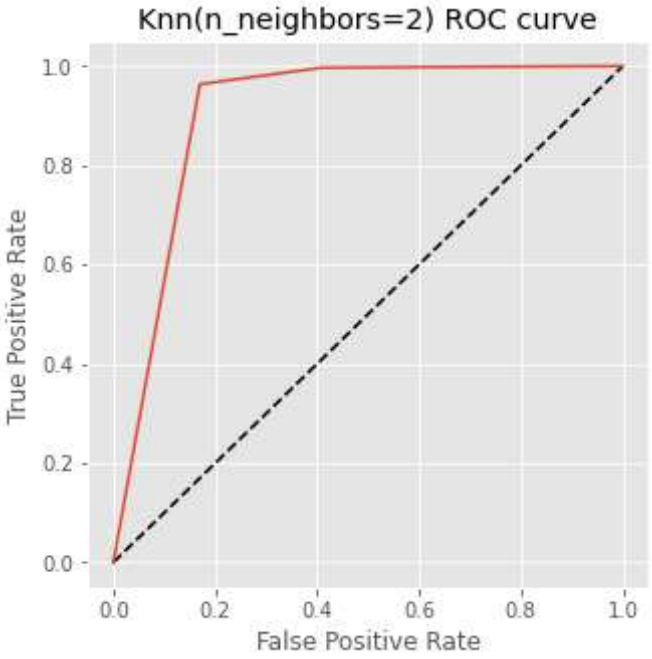
In [139…
```
y_pred_proba = knn.predict_proba(X_test)[:,1]
```

In [140…
```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

In [141…
```
roc_auc_score(y_test,y_pred_proba)
```

Out[141]:
```
0.9061183950345211
```

In [142…
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Knn(n_neighbors=2) ROC curve')
plt.show()
```

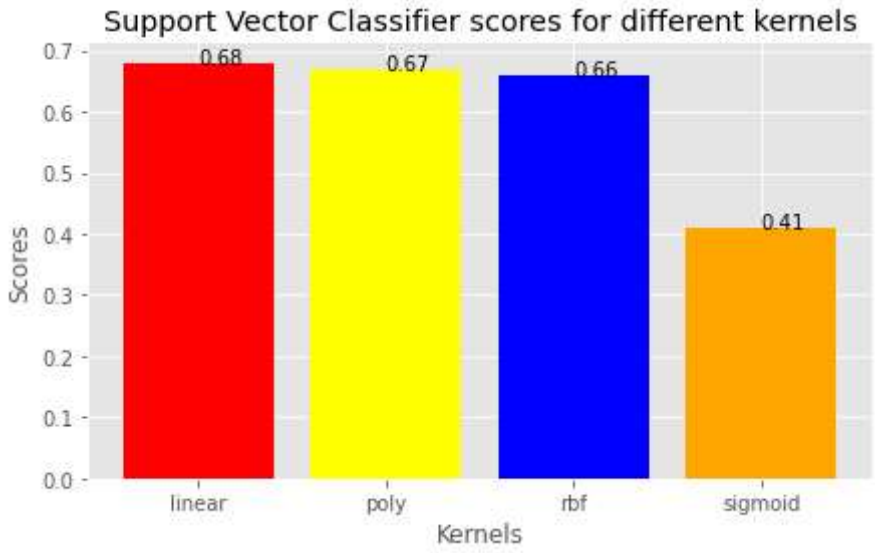


SVM

In [143…
```
svc_scores = []
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
for i in range(len(kernels)):
    svc_classifier = SVC(kernel = kernels[i])
    svc_classifier.fit(X_train, y_train)
    svc_scores.append(round(svc_classifier.score(X_test, y_test),2))
```

In [144…
```
# colors = rainbow(np.linspace(0, 1, len(kernels)))
plt.figure(figsize=(7,4))
plt.bar(kernels, svc_scores,color=['red', 'yellow', 'blue', 'orange'])
for i in range(len(kernels)):
    plt.text(i, svc_scores[i], svc_scores[i])
plt.xlabel('Kernels')
plt.ylabel('Scores')
plt.title('Support Vector Classifier scores for different kernels')
```

Out[144]:
```
Text(0.5, 1.0, 'Support Vector Classifier scores for different kernels')
```



In [145…
```
svm=SVC(probability=True)
svm.fit(X_train,y_train)
```

Out[145]:
```
SVC(probability=True)
```

In [146…
```
svm.score(X_test,y_test)
```

Out[146]:
```
0.6631944444444444
```

In [147…
```
y_svm_predi=svm.predict(X_test)
```

In [148…
```
confusion_matrix(y_test,y_svm_predi)
```

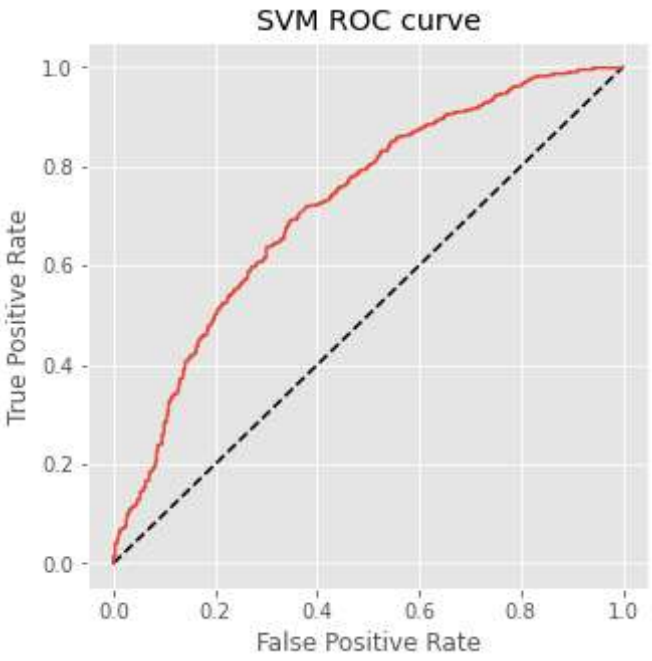
Out[148]:
```
array([[330, 223],
       [165, 434]], dtype=int64)
```

In [149…
```
y_svm_predict_pro=svm.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_svm_predict_pro)
```

In [150…
```
roc_auc_score(y_test,y_svm_predict_pro)
```

Out[150]:
```
0.7202661458066035
```

In [151…
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='SVM')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVM ROC curve')
plt.show()
```



Cat Boost

In [152…
```
!pip install catboost
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: catboost in c:\users\malak\appdata\roaming\python\p
ython39\site-packages (1.2.3)
Requirement already satisfied: pandas>=0.24 in c:\programdata\anaconda4\lib\site-p
ackages (from catboost) (1.4.2)
Requirement already satisfied: matplotlib in c:\programdata\anaconda4\lib\site-pac
kages (from catboost) (3.5.1)
Requirement already satisfied: six in c:\programdata\anaconda4\lib\site-packages
(from catboost) (1.16.0)
Requirement already satisfied: scipy in c:\programdata\anaconda4\lib\site-packages
(from catboost) (1.7.3)
Requirement already satisfied: plotly in c:\programdata\anaconda4\lib\site-package
s (from catboost) (5.6.0)
Requirement already satisfied: graphviz in c:\users\malak\appdata\roaming\python\p
ython39\site-packages (from catboost) (0.20.1)
Requirement already satisfied: numpy>=1.16.0 in c:\programdata\anaconda4\lib\site-
packages (from catboost) (1.21.5)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda4\lib\site-p
ackages (from pandas>=0.24->catboost) (2021.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda4
\lib\site-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda4\lib\site-p
ackages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda4\lib\sit
e-packages (from matplotlib->catboost) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\programdata\anaconda4\lib\si
te-packages (from matplotlib->catboost) (3.0.4)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda4\lib\site-
packages (from matplotlib->catboost) (9.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda4\lib\s
ite-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: fonttools>=4.22.0 in c:\programdata\anaconda4\lib\s
ite-packages (from matplotlib->catboost) (4.25.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\programdata\anaconda4\lib\sit
e-packages (from plotly->catboost) (8.0.1)
```

In [153…]
```
from catboost import CatBoostClassifier
```

In [154…]
```
catboost=CatBoostClassifier(iterations=100,learning_rate=0.03)
```

In [155…]
```
catboost.fit(X_train,y_train,verbose=10)
```

```
0:      learn: 0.6859826        total: 58.2ms   remaining: 5.76s
10:     learn: 0.6364291        total: 95.6ms   remaining: 774ms
20:     learn: 0.5919202        total: 120ms    remaining: 453ms
30:     learn: 0.5594548        total: 146ms    remaining: 326ms
40:     learn: 0.5314641        total: 173ms    remaining: 249ms
50:     learn: 0.5116290        total: 199ms    remaining: 191ms
60:     learn: 0.4925263        total: 225ms    remaining: 144ms
70:     learn: 0.4757321        total: 251ms    remaining: 103ms
80:     learn: 0.4619529        total: 291ms    remaining: 68.3ms
90:     learn: 0.4498779        total: 323ms    remaining: 32ms
99:     learn: 0.4428400        total: 349ms    remaining: 0us
```

Out[155]:
```
<catboost.core.CatBoostClassifier at 0x203703be970>
```

In [156…]
```
y_catboost_pred=catboost.predict(X_test)
y_catboost_pre_prob=catboost.predict_proba(X_test)[:,1]
```

In [157…]
```
catboost.score(X_test,y_test)
```

Out[157]:
```
0.8342013888888888
```

In [158…
```
confusion_matrix(y_test,y_catboost_pred)
```

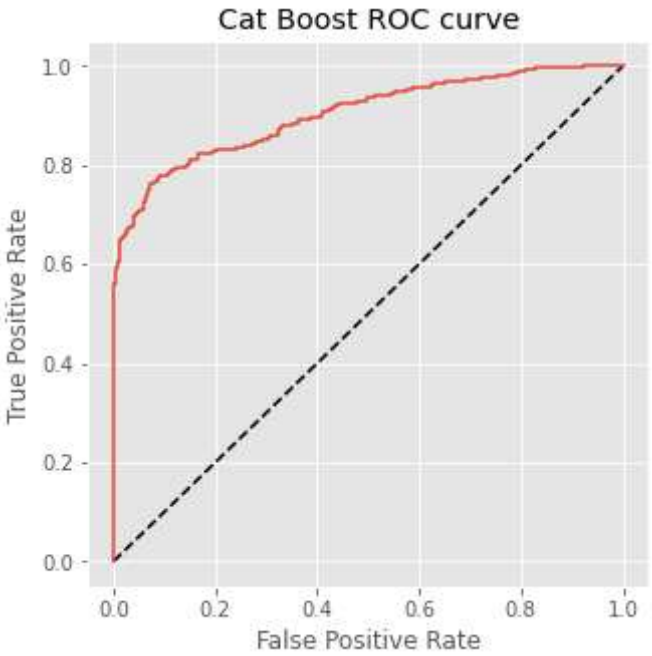
Out[158]:
```
array([[495,  58],
       [133, 466]], dtype=int64)
```

In [159…
```
roc_auc_score(y_test,y_catboost_pre_prob)
```

Out[159]:
```
0.9032262933702041
```

In [160…
```
fpr, tpr, thresholds = roc_curve(y_test, y_catboost_pre_prob)
```

In [161…
```
plt.figure(figsize=(5,5))
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Cat Boost')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Cat Boost ROC curve')
plt.show()
```



Neural Networks

In [162…
```
X_train=pd.DataFrame(X_train)
y_train = pd.DataFrame(y_train)
```

In [163…
```
# #since we shuffled, the index numbers were messed up, this resets them
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

#convert to numpy arrays with float values
X_train = np.array(X_train, dtype=float)
y_train = np.array(y_train, dtype=float)

#reshape y_train to make matrix multiplication possible
y_train = np.array(y_train).reshape(-1, 1)
```

In [164…
```
class Perceptron:
    def __init__(self, x, y):
```

```
        self.input = np.array(x, dtype=float)
        self.label = np.array(y, dtype=float)
        self.weights = np.random.rand(x.shape[1], y.shape[1]) #randomly initialize
        self.z = self.input@self.weights #dot product of the vectors
        self.yhat = self.sigmoid(self.z) #apply activation function


    def sigmoid(self, x):
        return 1.0/(1.0+np.exp(-x))

    def sigmoid_deriv(self, x):
        s = sigmoid(x)
        return s(1-s)

    def forward_prop(self):
        self.yhat = self.sigmoid(self.input @ self.weights) #@ symbol represents ma
        return self.yhat

    def back_prop(self):
        gradient = self.input.T @ (-2.0*(self.label - self.yhat)*self.sigmoid(self.

        self.weights -= gradient #process of finding the minimum loss
```

In [165…

```
simple_nn = Perceptron(X_train, y_train)
training_iterations = 1000

history = [] #we will store how the mean squared error changes after each iteration

def mse(yhat, y):
    sum = 0.0
    for pred, label in zip(yhat, y):
        sum += (pred-label)**2
    return sum/len(yhat)

for i in range(training_iterations):
    simple_nn.forward_prop()
    simple_nn.back_prop()
    yhat = simple_nn.forward_prop()
    history.append(mse(yhat, simple_nn.label))



yhat = simple_nn.forward_prop()
print(f'Final Mean Squared Error: {mse(yhat, simple_nn.label)}')
```

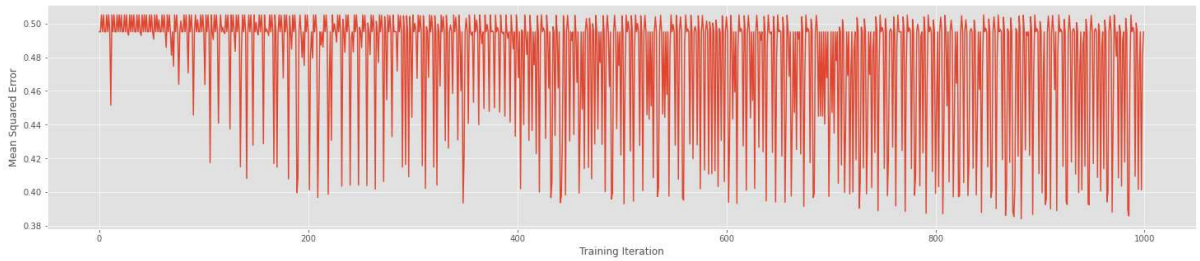
```
Final Mean Squared Error: [0.49500651]
```

In [166…

```
plt.plot(history)
plt.ylabel('Mean Squared Error')
plt.xlabel('Training Iteration')
```

Out[166]:

```
Text(0.5, 0, 'Training Iteration')
```



In [ ]:

In [ ]: