In [22]:

```
!pip install imblearn
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.10.1-py3-none-any.whl (226 kB)
     ------------------------------------ 226.0/226.0 kB 655.8 kB/s eta 0:00:00
Requirement already satisfied: joblib>=1.1.1 in c:\users\jkuma\appdata\roaming\python\py
thon39\site-packages (from imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\lib\site
-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\programdata\anaconda3\lib\site-package
s (from imbalanced-learn->imblearn) (1.9.1)
Requirement already satisfied: numpy>=1.17.3 in c:\programdata\anaconda3\lib\site-packag
es (from imbalanced-learn->imblearn) (1.21.5)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\jkuma\appdata\roaming\pyt
hon\python39\site-packages (from imbalanced-learn->imblearn) (1.2.2)
Installing collected packages: imbalanced-learn, imblearn
Successfully installed imbalanced-learn-0.10.1 imblearn-0.0
```

In [1]:

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

from sklearn.linear_model import LogisticRegressionCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from imblearn.over_sampling import SMOTE
import plotly.express as px
from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import NearMiss
```

In [2]:

```
df=pd.read_csv(r"C:\Users\Jkuma\Downloads\heart_2020_cleaned.csv\keyfactorofheartdisease
```

In [3]:

```
df
```

Loading [MathJax]/extensions/Safe.js

Out[3]:

| | HeartDisease | BMI | Smoking | AlcoholDrinking | Stroke | PhysicalHealth | MentalHealth | DiffWalking | S |
|---|---|---|---|---|---|---|---|---|---|
| **0** | No | 16.60 | Yes | No | No | 3.0 | 30.0 | No | Fema |
| **1** | No | 20.34 | No | No | Yes | 0.0 | 0.0 | No | Fema |
| **2** | No | 26.58 | Yes | No | No | 20.0 | 30.0 | No | Ma |
| **3** | No | 24.21 | No | No | No | 0.0 | 0.0 | No | Fema |
| **4** | No | 23.71 | No | No | No | 28.0 | 0.0 | Yes | Fema |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **319790** | Yes | 27.41 | Yes | No | No | 7.0 | 0.0 | Yes | Ma |
| **319791** | No | 29.84 | Yes | No | No | 0.0 | 0.0 | No | Ma |
| **319792** | No | 24.24 | No | No | No | 0.0 | 0.0 | No | Fema |
| **319793** | No | 32.81 | No | No | No | 0.0 | 0.0 | No | Fema |
| **319794** | No | 46.56 | No | No | No | 0.0 | 0.0 | No | Fema |

319795 rows × 18 columns

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 18 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   HeartDisease      319795 non-null  object 
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object 
 3   AlcoholDrinking   319795 non-null  object 
 4   Stroke            319795 non-null  object 
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object 
 8   Sex               319795 non-null  object 
 9   AgeCategory       319795 non-null  object 
 10  Race              319795 non-null  object 
 11  Diabetic          319795 non-null  object 
 12  PhysicalActivity  319795 non-null  object 
 13  GenHealth         319795 non-null  object 
 14  SleepTime         319795 non-null  float64
 15  Asthma            319795 non-null  object 
 16  KidneyDisease     319795 non-null  object 
 17  SkinCancer        319795 non-null  object 
dtypes: float64(4), object(14)
memory usage: 43.9+ MB
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke',
       'PhysicalHealth', 'MentalHealth', 'DiffWalking', 'Sex', 'AgeCategory',
       'Race', 'Diabetic', 'PhysicalActivity', 'GenHealth', 'SleepTime',
       'Asthma', 'KidneyDisease', 'SkinCancer'],
      dtype='object')
```

In [6]:

```
df.shape
```

Out[6]:

```
(319795, 18)
```

In [ ]:

Loading [MathJax]/extensions/Safe.js

```
In [ ]:
```

Before training process for model,select only those columns that would significantly impact the likehood of heart disease

```
In [7]: new_df=df[['HeartDisease', 'BMI', 'Smoking', 'AlcoholDrinking', 'Stroke','PhysicalHealth
```

```
In [8]: new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 319795 entries, 0 to 319794
Data columns (total 16 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   HeartDisease      319795 non-null  object 
 1   BMI               319795 non-null  float64
 2   Smoking           319795 non-null  object 
 3   AlcoholDrinking   319795 non-null  object 
 4   Stroke            319795 non-null  object 
 5   PhysicalHealth    319795 non-null  float64
 6   MentalHealth      319795 non-null  float64
 7   DiffWalking       319795 non-null  object 
 8   Sex               319795 non-null  object 
 9   AgeCategory       319795 non-null  object 
 10  Diabetic          319795 non-null  object 
 11  PhysicalActivity  319795 non-null  object 
 12  GenHealth         319795 non-null  object 
 13  SleepTime         319795 non-null  float64
 14  Asthma            319795 non-null  object 
 15  KidneyDisease     319795 non-null  object 
dtypes: float64(4), object(12)
memory usage: 39.0+ MB
```

```
In [9]: new_df.describe().T
```

Out[9]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **BMI** | 319795.0 | 28.325399 | 6.356100 | 12.02 | 24.03 | 27.34 | 31.42 | 94.85 |
| **PhysicalHealth** | 319795.0 | 3.371710 | 7.950850 | 0.00 | 0.00 | 0.00 | 2.00 | 30.00 |
| **MentalHealth** | 319795.0 | 3.898366 | 7.955235 | 0.00 | 0.00 | 0.00 | 3.00 | 30.00 |
| **SleepTime** | 319795.0 | 7.097075 | 1.436007 | 1.00 | 6.00 | 7.00 | 8.00 | 24.00 |

```
In [10]: new_df.isnull().sum()
```

Out[10]:
```
HeartDisease        0
BMI                 0
Smoking             0
AlcoholDrinking     0
Stroke              0
PhysicalHealth      0
MentalHealth        0
DiffWalking         0
Sex                 0
AgeCategory         0
Diabetic            0
PhysicalActivity    0
GenHealth           0
SleepTime           0
Asthma              0
KidneyDisease       0
dtype: int64
```

In [11]:
```
new_df.nunique()
```

Out[11]:
```
HeartDisease           2
BMI                 3604
Smoking                2
AlcoholDrinking        2
Stroke                 2
PhysicalHealth        31
MentalHealth          31
DiffWalking            2
Sex                    2
AgeCategory           13
Diabetic               4
PhysicalActivity       2
GenHealth              5
SleepTime             24
Asthma                 2
KidneyDisease          2
dtype: int64
```

In [12]:
```
new_df.dtypes
```

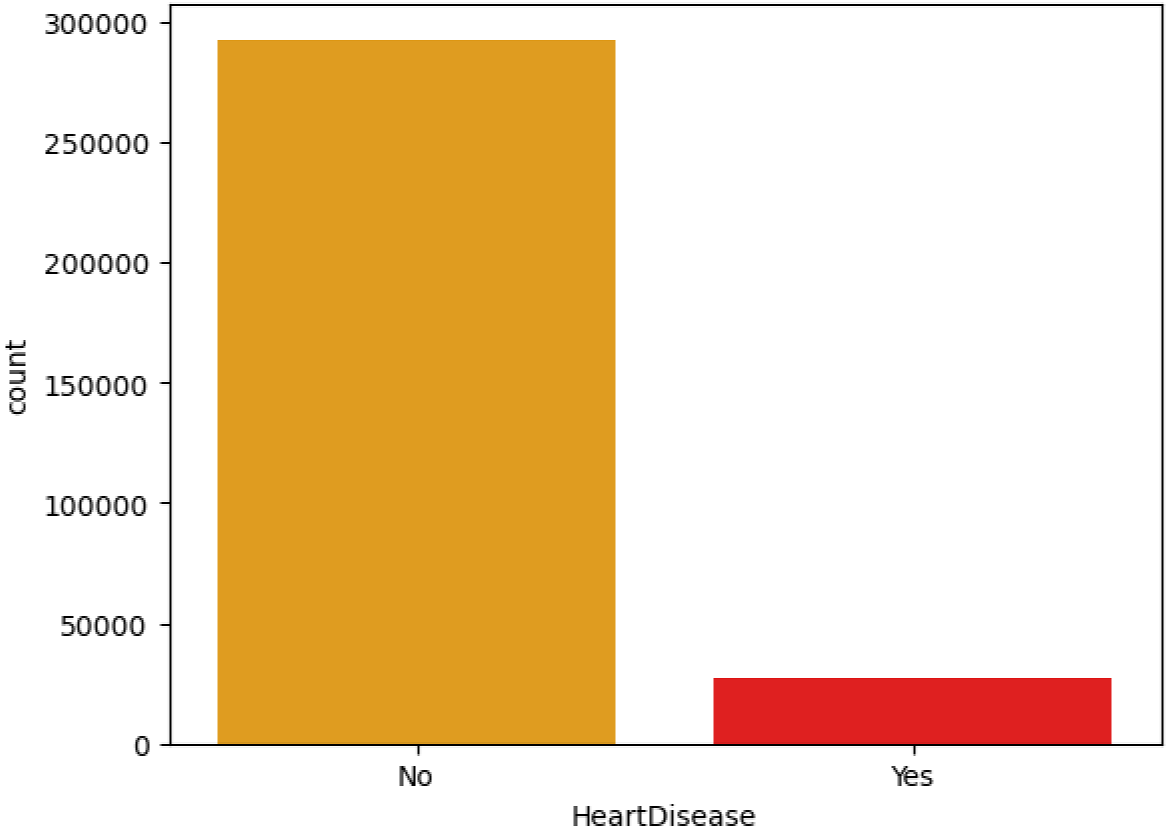
Out[12]:
```
HeartDisease         object
BMI                 float64
Smoking              object
AlcoholDrinking      object
Stroke               object
PhysicalHealth      float64
MentalHealth        float64
DiffWalking          object
Sex                  object
AgeCategory          object
Diabetic             object
PhysicalActivity     object
GenHealth            object
SleepTime           float64
Asthma               object
KidneyDisease        object
dtype: object
```

## Data Visualization

In [13]:
```
sns.countplot(x=new_df['HeartDisease'],palette=["Orange", "Red"])
```

Out[13]:
```
<AxesSubplot:xlabel='HeartDisease', ylabel='count'>
```

Loading [MathJax]/extensions/Safe.js

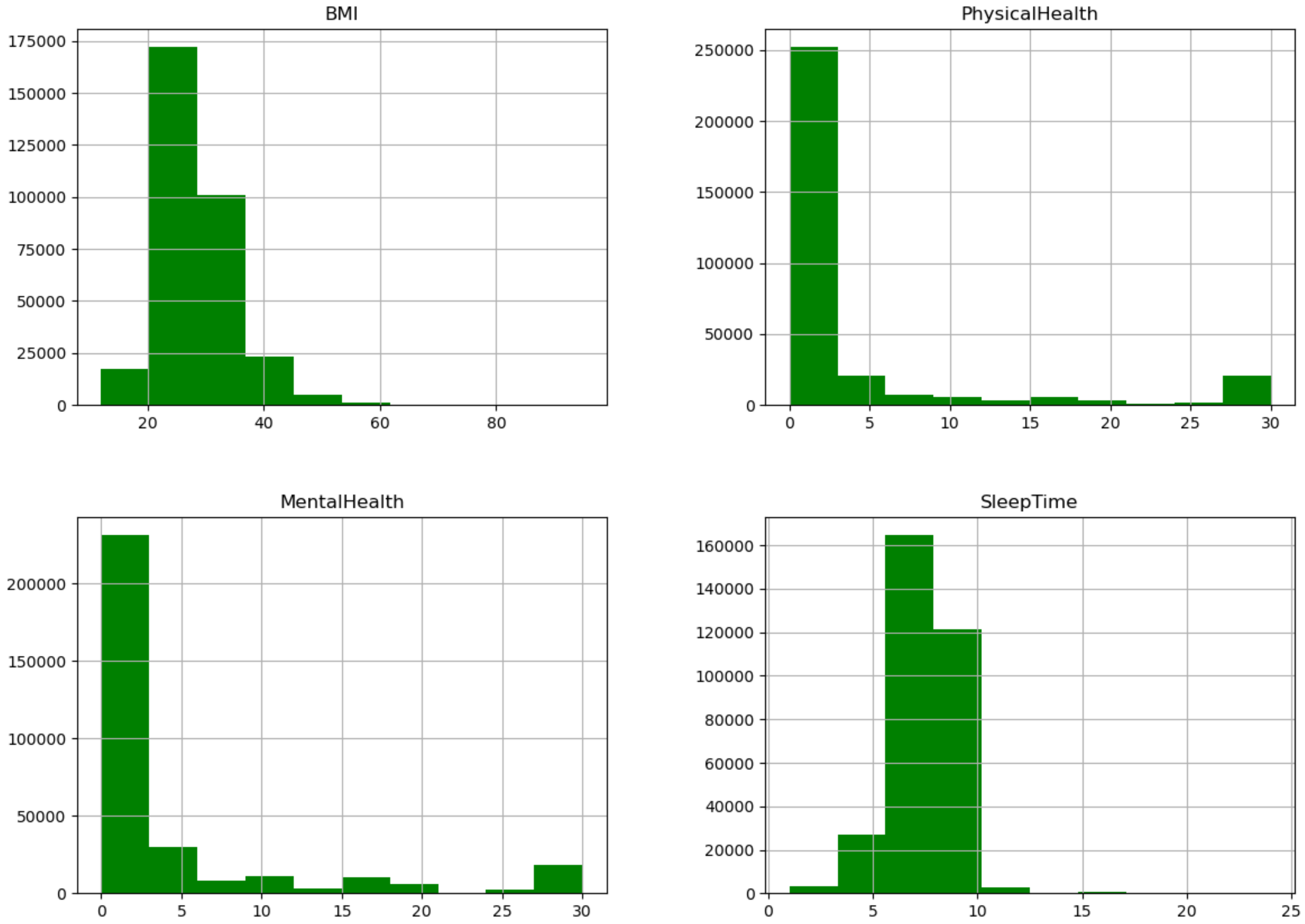


The graph shows that the amount of records for heart disease are unbalanced, showing an uneven distribution of information between people who have the condition and those who do not. The forecasts for heart disease made by the trained model could be significantly biassed as a result. We will employ SMOTE (Synthetic Minority Oversampling Technique) to balance the class distribution. However, in order for SMOTE to work properly, all categorical data must first be converted into binary using dummy variables.

The nearest minority class data elements are taken into account by the SMOTE algorithm, which then generates new combinations based on those entries.

In [14]:

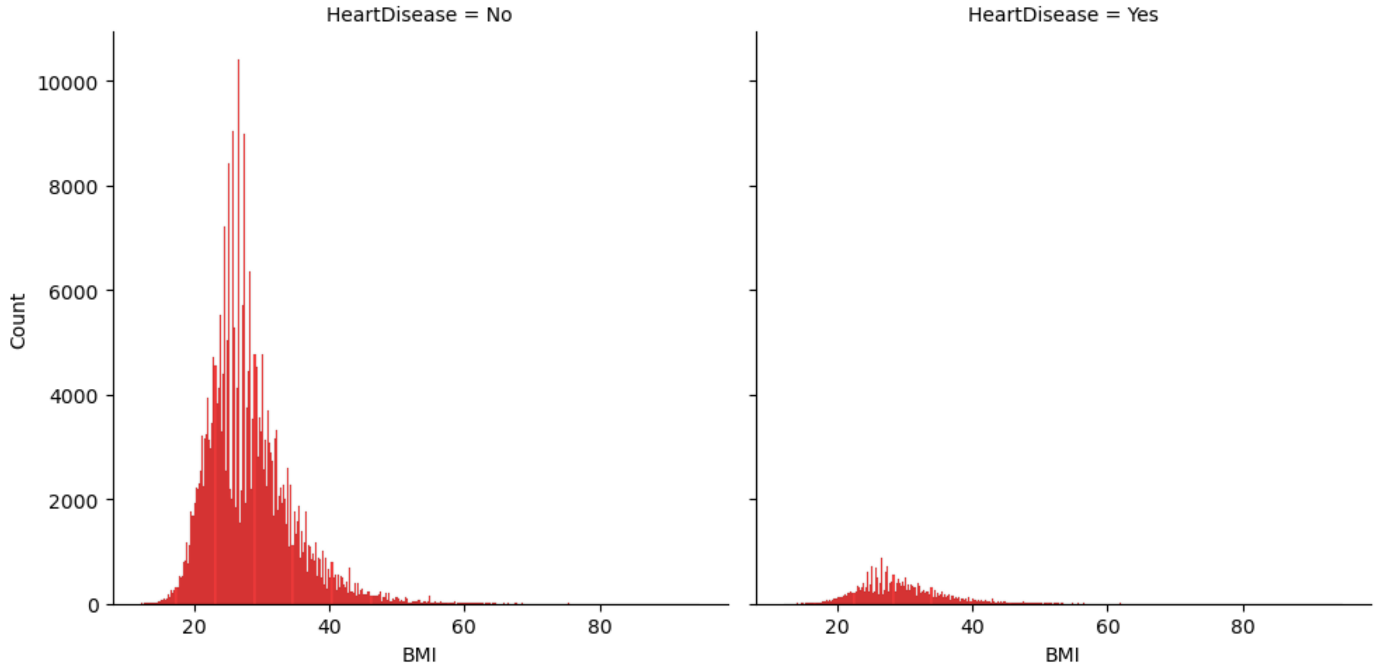
```
new_df.hist(bins=10,figsize=(14,10),color="green")

plt.savefig('histogram1.png')
```



```
In [ ]:
```

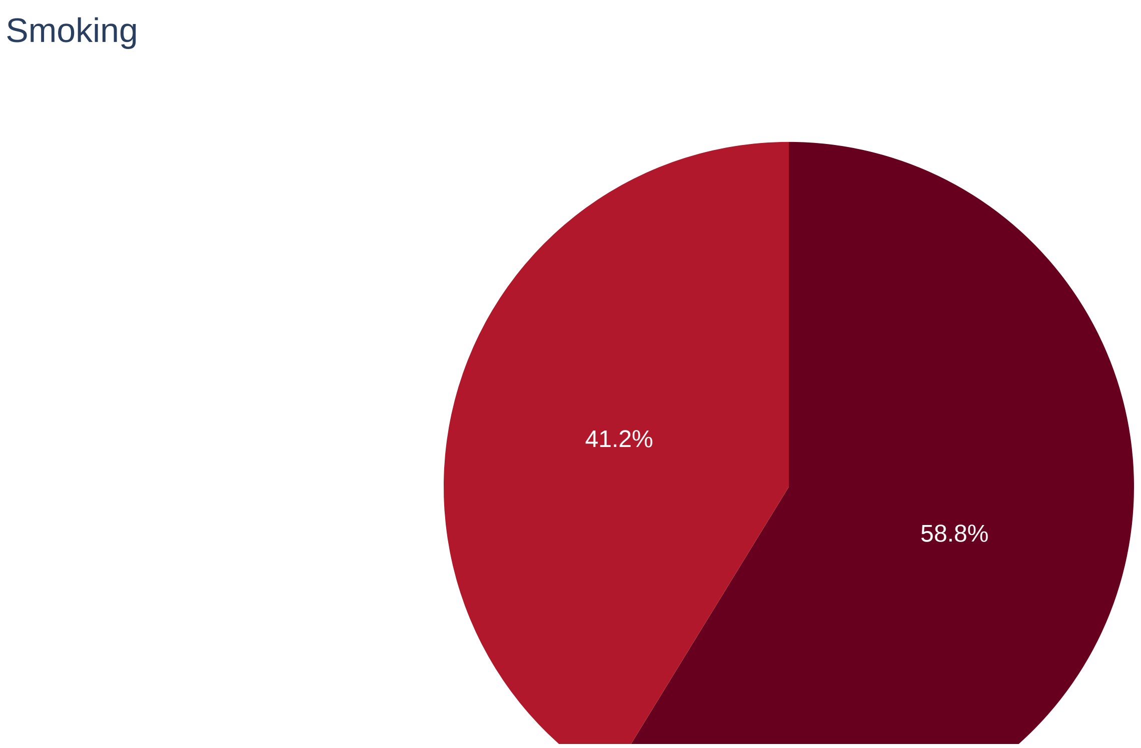
```
In [15]: sns.displot(df,x="BMI",col='HeartDisease',color="red")
```

```
Out[15]: <seaborn.axisgrid.FacetGrid at 0x27404fe7490>
```



```
In [16]: fig = px.pie(new_df,names='Smoking',title='Smoking',color_discrete_sequence=px.colors.se
         fig.show()
         #
```

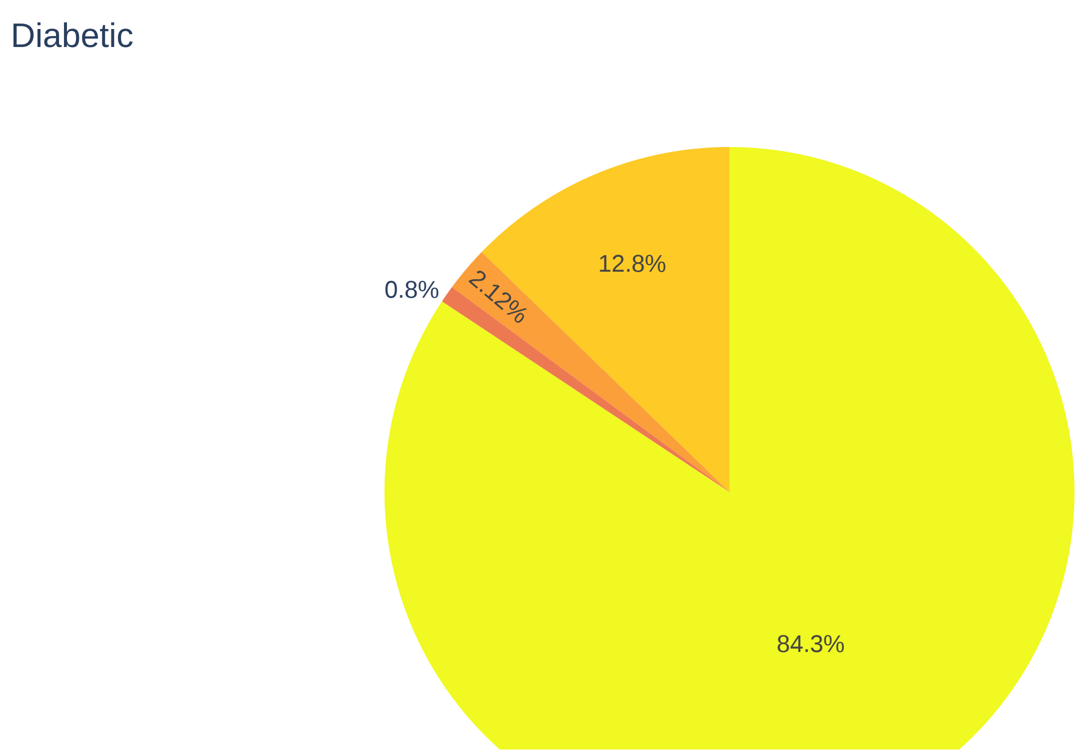
Loading [MathJax]/extensions/Safe.js



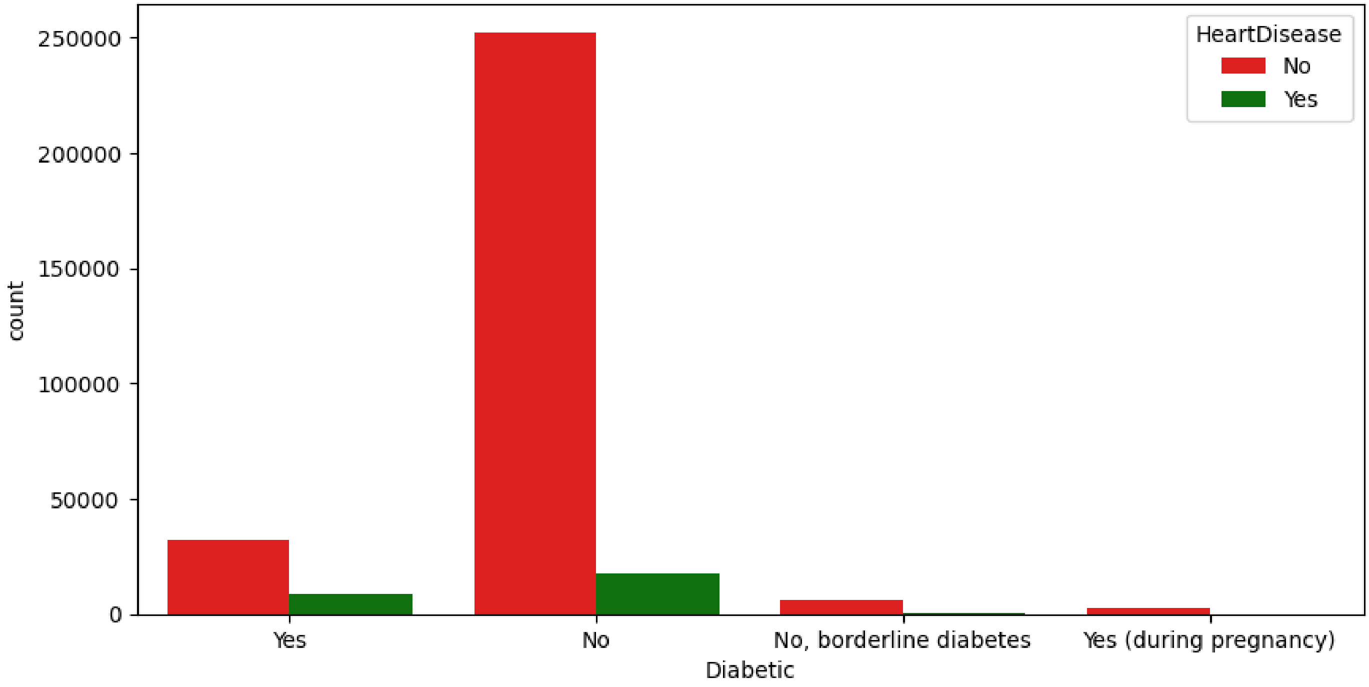
In [17]:

```
fig = px.pie(new_df,names='Diabetic',title='Diabetic',color_discrete_sequence=px.colors.
fig.show()
```

Loading [MathJax]/extensions/Safe.js



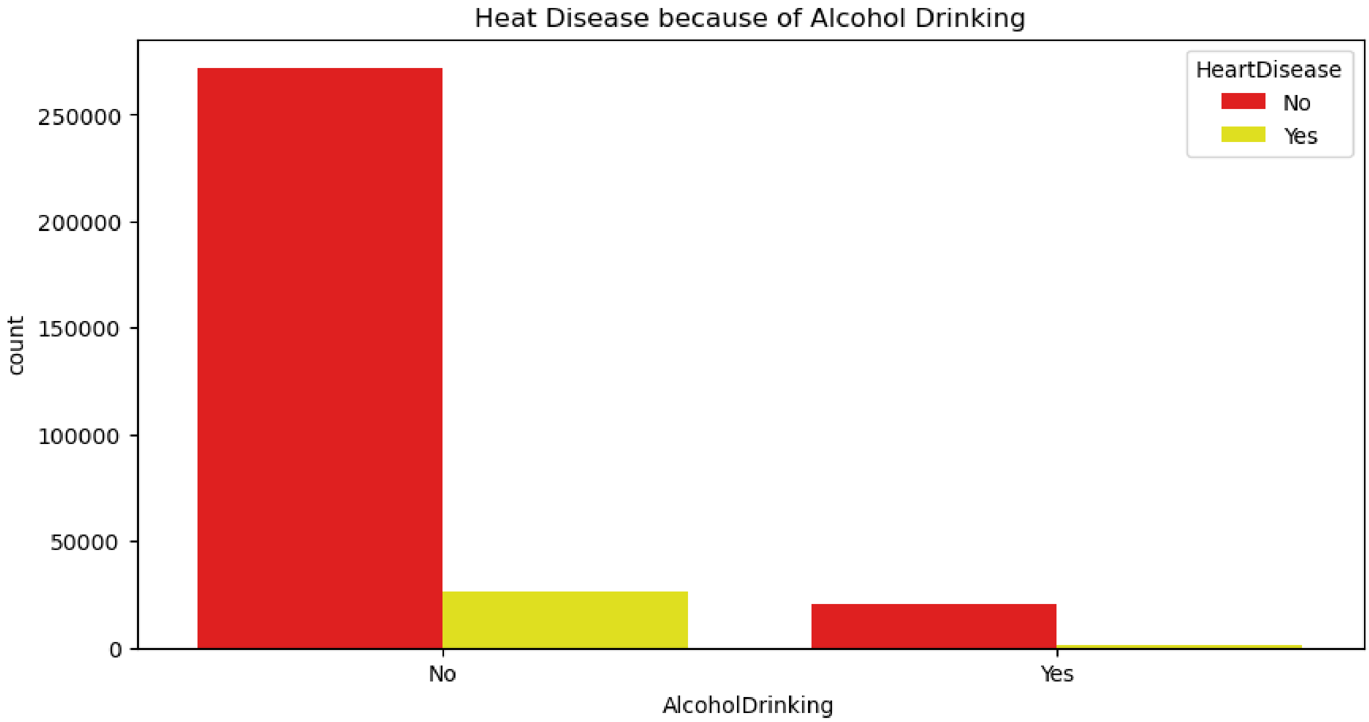
```
In [18]: plt.figure(figsize=(10,5))
         sns.countplot(x=new_df['Diabetic'],hue="HeartDisease",data=new_df,palette=[ "Red","Green
         plt.show()
```



```
In [19]: plt.figure(figsize=(10,5))
         sns.countplot(x=new_df['AlcoholDrinking'],data=new_df,hue="HeartDisease",palette=['Red',
         plt.title("Heat Disease because of Alcohol Drinking")
```

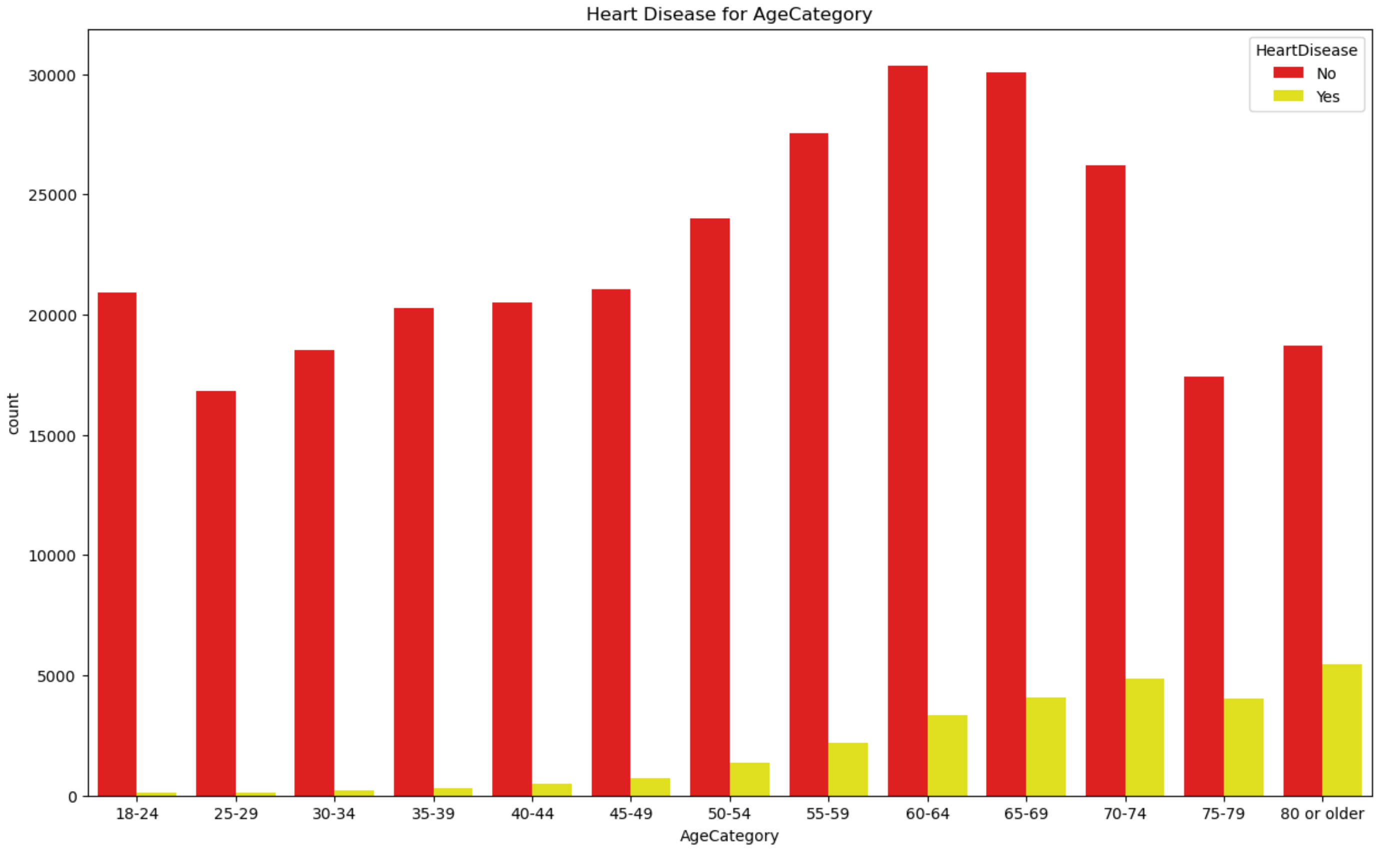
Loading [MathJax]/extensions/Safe.js

```
Out[19]: Text(0.5, 1.0, 'Heat Disease because of Alcohol Drinking')
```



```
In [20]: plt.figure(figsize=(15,9))
         sns.countplot(x=new_df['AgeCategory'].sort_values(ascending=True),data=new_df,hue='Heart
         plt.title("Heart Disease for AgeCategory")
```

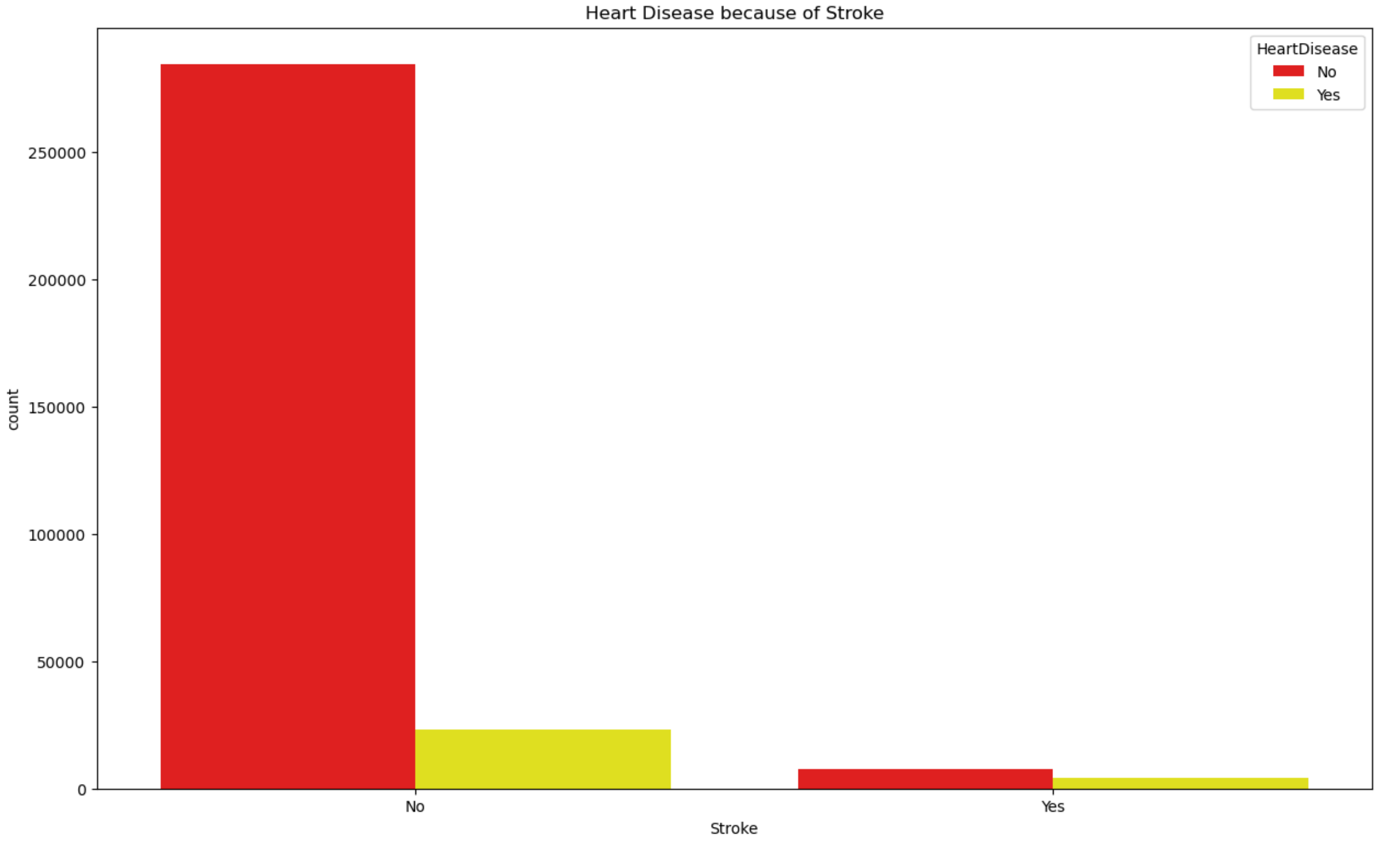
```
Out[20]: Text(0.5, 1.0, 'Heart Disease for AgeCategory')
```



```
In [21]: plt.figure(figsize=(15,9))
         sns.countplot(x=new_df['Stroke'].sort_values(ascending=True),data=new_df,hue='HeartDisea
         plt.title("Heart Disease because of Stroke")
```

```
Out[21]: Text(0.5, 1.0, 'Heart Disease because of Stroke')
```

Loading [MathJax]/extensions/Safe.js



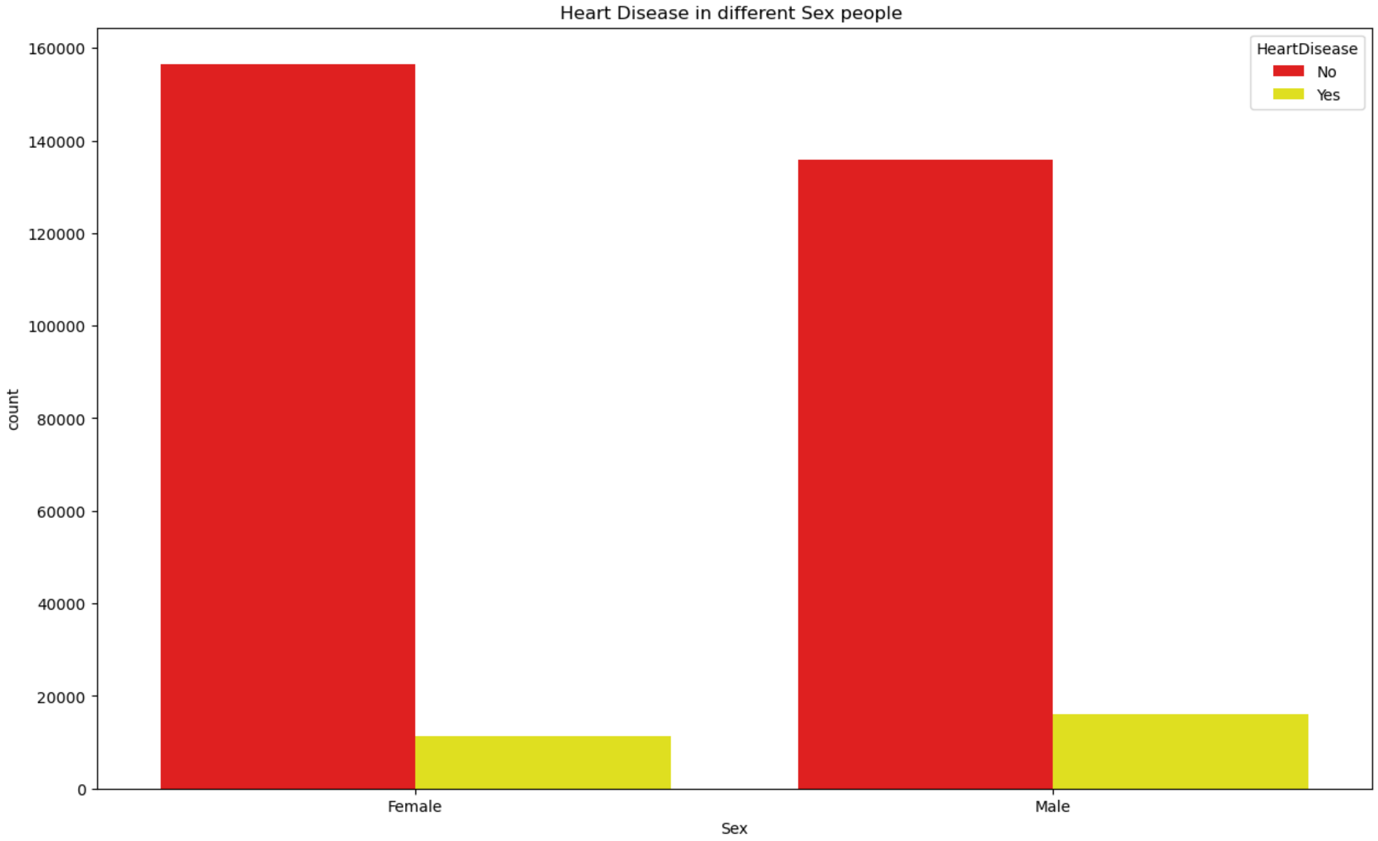
The frequency of heart disease cases is directly proportional with the age of the individuals, which is expected. Heart disease appears to be more prevalent among individuals who identify as White, indicating the need for more diversity in the racial makeup of the sample group. Insufficient data is available to confirm Alcohol Drinking as a significant indicator of heart disease

In [22]:

```
plt.figure(figsize=(15,9))
sns.countplot(x=new_df['Sex'].sort_values(ascending=True),data=new_df,hue='HeartDisease'
plt.title("Heart Disease in different Sex people ")
```

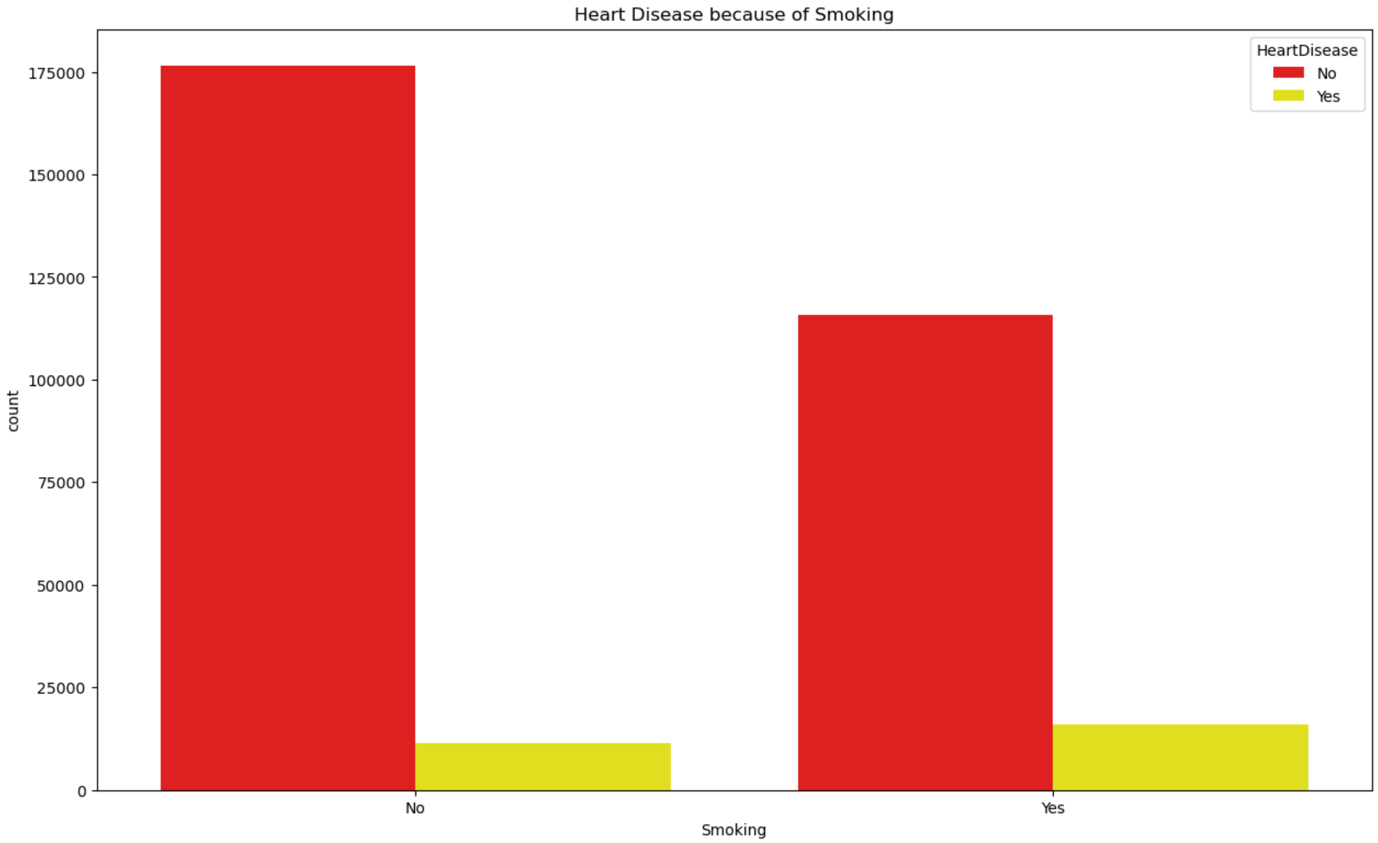
Out[22]:

```
Text(0.5, 1.0, 'Heart Disease in different Sex people ')
```



```
In [23]: plt.figure(figsize=(15,9))
         sns.countplot(x=new_df['Smoking'].sort_values(ascending=True),data=new_df,hue='HeartDise
         plt.title("Heart Disease because of Smoking")
```

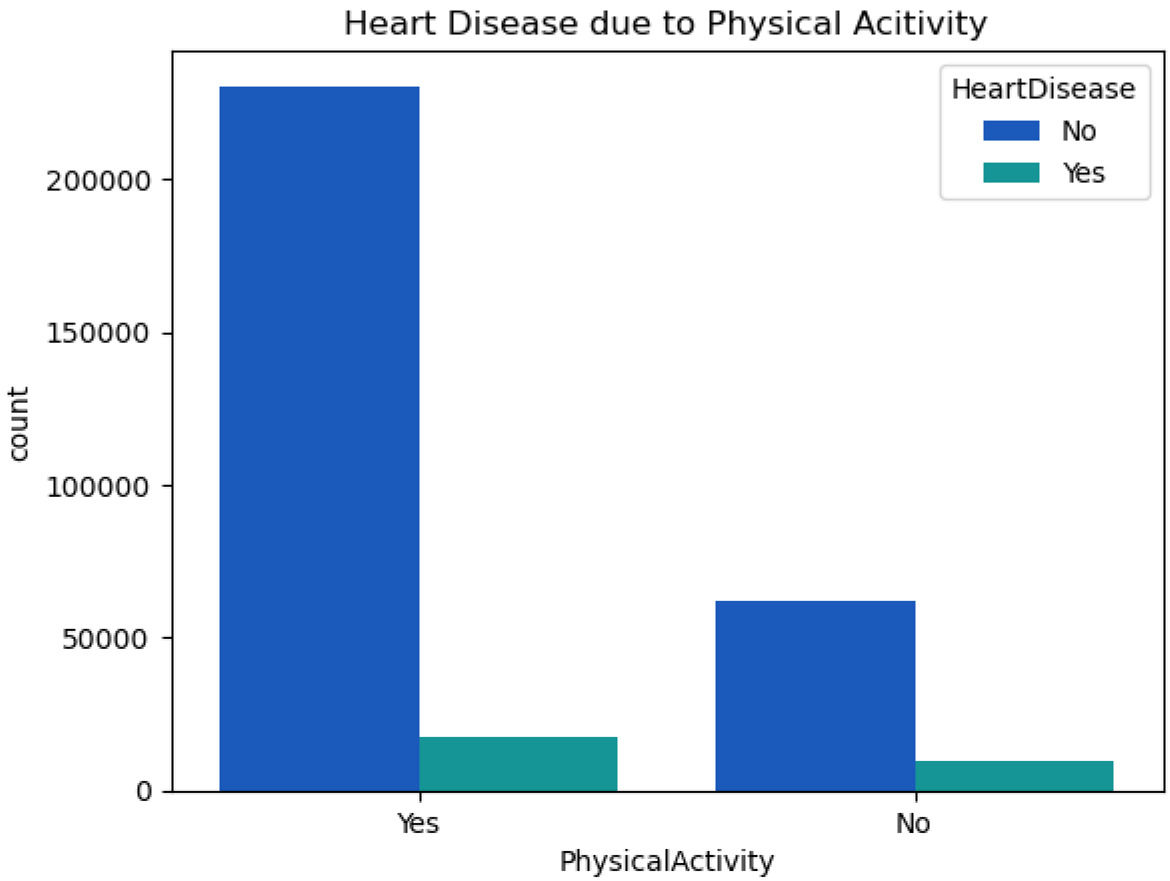
```
Out[23]: Text(0.5, 1.0, 'Heart Disease because of Smoking')
```



```
In [24]: sns.countplot(x=new_df['PhysicalActivity'],hue="HeartDisease",data=new_df,palette="winte
         plt.title("Heart Disease due to Physical Acitivity")
```

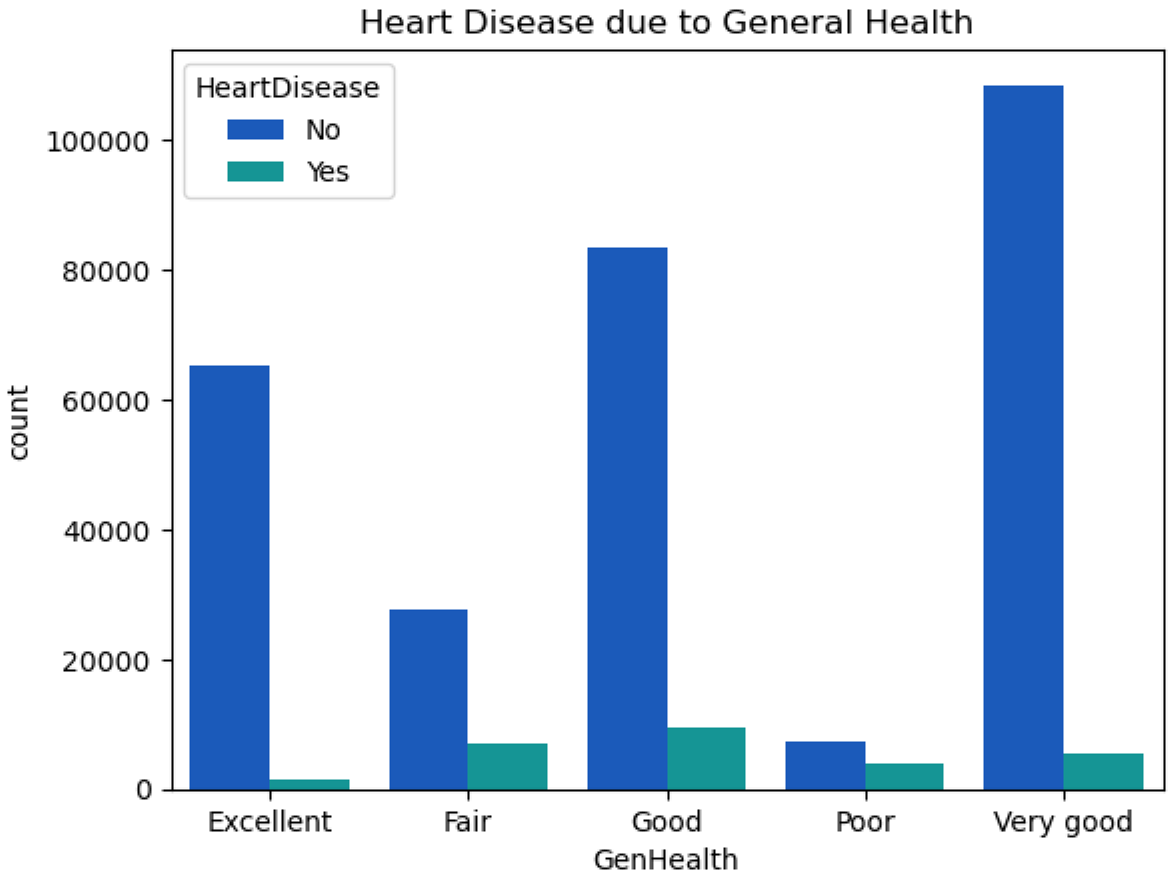
```
Out[24]: Text(0.5, 1.0, 'Heart Disease due to Physical Acitivity')
```

Loading [MathJax]/extensions/Safe.js



```
In [25]: sns.countplot(x=new_df['GenHealth'].sort_values(ascending=True),hue="HeartDisease",data=
         plt.title("Heart Disease due to General Health")
```

```
Out[25]: Text(0.5, 1.0, 'Heart Disease due to General Health')
```



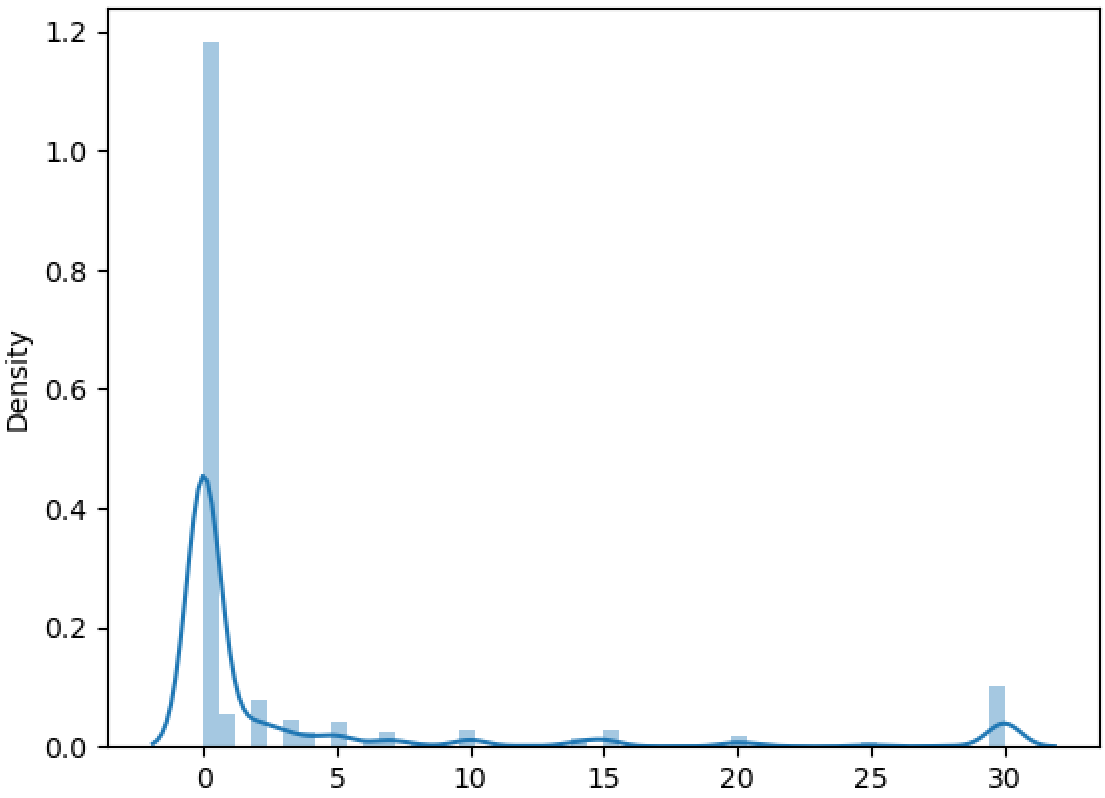
```
In [26]: sns.distplot(x=new_df['PhysicalHealth'])
```

Loading [MathJax]/extensions/Safe.js

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adap
t your code to use either `displot` (a figure-level function with similar flexibility) o
r `histplot` (an axes-level function for histograms).
```

Out[26]:

```
<AxesSubplot:ylabel='Density'>
```



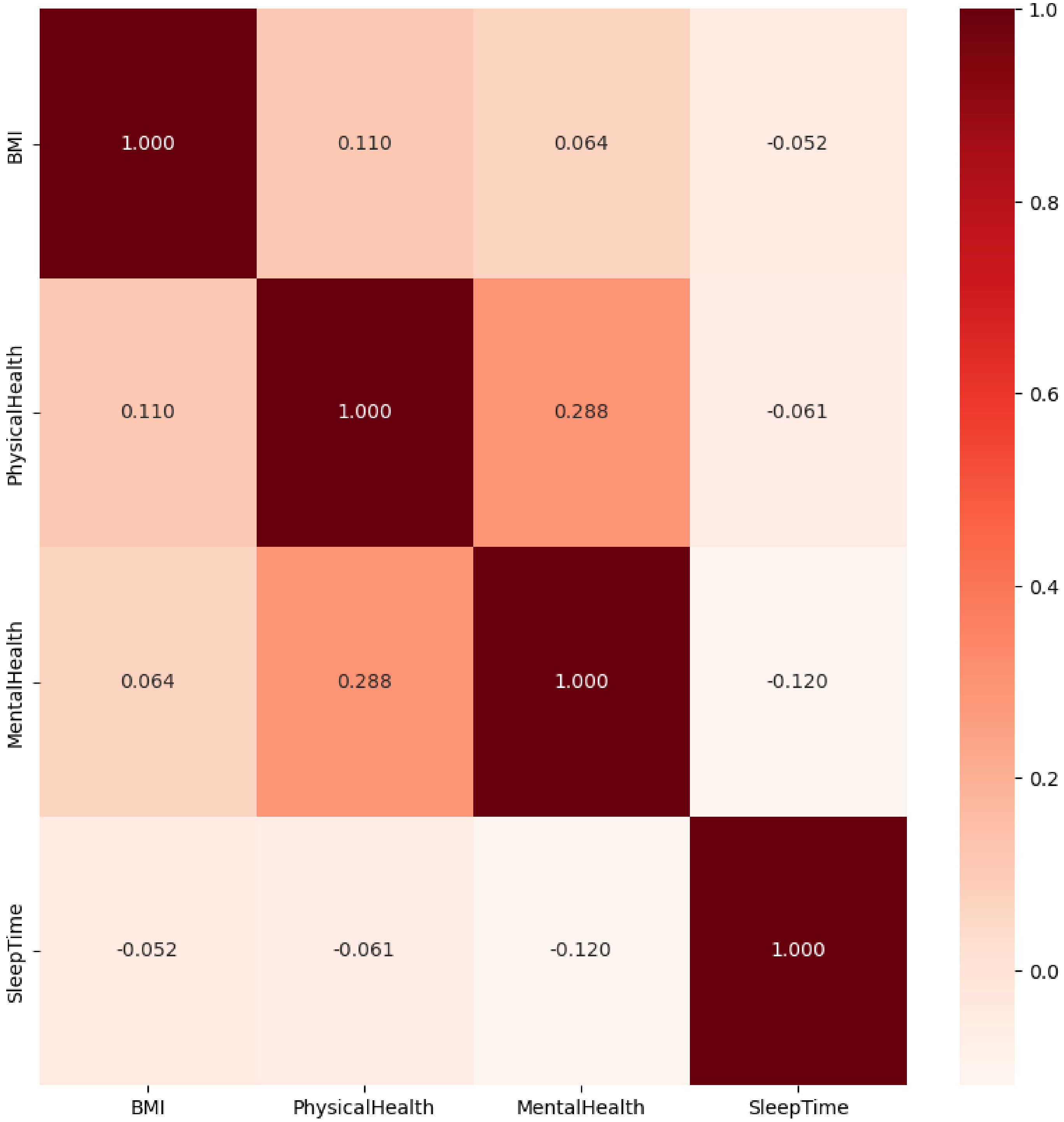
In [27]:

```
plt.figure(figsize=(10,10))
cor=new_df.corr()
sns.heatmap(cor,annot=True,cmap=plt.cm.Reds,fmt='.3f')
```

Out[27]:

```
<AxesSubplot:>
```

Loading [MathJax]/extensions/Safe.js



In [ ]:

as we know bmi<18.5 is underweight

bmi between 18.5-24.9 is normal

bmi between 25-29.9 is overweight

bmi between 30-34.9 is obese

bmi >35 is extremely obese

# Segment Bmi

In [28]:

```
value,index=new_df['BMI'].value_counts().values,new_df['BMI'].value_counts().index
print("BMI Column's Count Values : ")
pd.DataFrame(value,index,columns=['Count'])
```

```
BMI Column's Count Values :
```

Out[28]:

|  | Count |
|---|---|
| **26.63** | 3762 |
| **27.46** | 2767 |
| **27.44** | 2723 |
| **24.41** | 2696 |
| **27.12** | 2525 |
| **...** | ... |
| **59.85** | 1 |
| **50.59** | 1 |
| **92.53** | 1 |
| **62.95** | 1 |
| **46.56** | 1 |

3604 rows × 1 columns

In [29]:

```python
bins = [0, 18.5, 25, 30, 35, np.inf]
names = ['Underweight', 'Normal weight', 'Overweight', 'Obese', 'Extremly Obese']
new_df['SegmentBMI'] = pd.cut(new_df['BMI'],bins, labels=names)
new_df.drop('BMI',axis=1,inplace=True)
new_df.head()
```

```
C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2431362878.py:3: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2431362878.py:4: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
```

Out[29]:

|  | HeartDisease | Smoking | AlcoholDrinking | Stroke | PhysicalHealth | MentalHealth | DiffWalking | Sex | AgeCateg |
|---|---|---|---|---|---|---|---|---|---|
| **0** | No | Yes | No | No | 3.0 | 30.0 | No | Female | 5 |
| **1** | No | No | No | Yes | 0.0 | 0.0 | No | Female | 80 or ( |
| **2** | No | Yes | No | No | 20.0 | 30.0 | No | Male | 6 |
| **3** | No | No | No | No | 0.0 | 0.0 | No | Female | 7 |
| **4** | No | No | No | No | 28.0 | 0.0 | Yes | Female | 4 |

Loading [MathJax]/extensions/Safe.js

In [30]:

```
value,index=new_df['SegmentBMI'].value_counts().values,new_df['SegmentBMI'].value_counts
print("SegmentBMI Column's Count Values:")
pd.DataFrame(value,index,columns=['Count'])
```

```
SegmentBMI Column's Count Values:
```

Out[30]:

| | Count |
|---|---|
| **Overweight** | 114355 |
| **Normal weight** | 97778 |
| **Obese** | 61169 |
| **Extremly Obese** | 41379 |
| **Underweight** | 5114 |

In [31]:

```
#object Columns
obj=new_df.select_dtypes(include=object).columns
pd.DataFrame(obj,columns=['Object Columns'])
```

Out[31]:

| | Object Columns |
|---|---|
| **0** | HeartDisease |
| **1** | Smoking |
| **2** | AlcoholDrinking |
| **3** | Stroke |
| **4** | DiffWalking |
| **5** | Sex |
| **6** | AgeCategory |
| **7** | Diabetic |
| **8** | PhysicalActivity |
| **9** | GenHealth |
| **10** | Asthma |
| **11** | KidneyDisease |

In [32]:

```
new_df['SegmentBMI'].dtypes
```

Out[32]:

```
CategoricalDtype(categories=['Underweight', 'Normal weight', 'Overweight', 'Obese',
                  'Extremly Obese'],
, ordered=True)
```

In [33]:

```
obj=list(obj)
obj.append('SegmentBMI')
pd.DataFrame(obj,columns=['Object Columns'])
```

Out[33]:

|  | Object Columns |
|---|---|
| **0** | HeartDisease |
| **1** | Smoking |
| **2** | AlcoholDrinking |
| **3** | Stroke |
| **4** | DiffWalking |
| **5** | Sex |
| **6** | AgeCategory |
| **7** | Diabetic |
| **8** | PhysicalActivity |
| **9** | GenHealth |
| **10** | Asthma |
| **11** | KidneyDisease |
| **12** | SegmentBMI |

In [34]:

```
#Transform Object Columns
#label_encoder = preprocessing.LabelEncoder()
label=LabelEncoder()
for col in obj:
    new_df[col]=label.fit_transform(new_df[col])
new_df
```

```
C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g: 
```

Loading [MathJax]/extensions/Safe.js

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\Jkuma\AppData\Local\Temp\ipykernel_1768\2400054535.py:5: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```

Loading [MathJax]/extensions/Safe.js

```
guide/indexing.html#returning-a-view-versus-a-copy
```

Out[34]:

| | HeartDisease | Smoking | AlcoholDrinking | Stroke | PhysicalHealth | MentalHealth | DiffWalking | Sex | AgeCa |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0 | 0 | 3.0 | 30.0 | 0 | 0 | |
| **1** | 0 | 0 | 0 | 1 | 0.0 | 0.0 | 0 | 0 | |
| **2** | 0 | 1 | 0 | 0 | 20.0 | 30.0 | 0 | 1 | |
| **3** | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | |
| **4** | 0 | 0 | 0 | 0 | 28.0 | 0.0 | 1 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **319790** | 1 | 1 | 0 | 0 | 7.0 | 0.0 | 1 | 1 | |
| **319791** | 0 | 1 | 0 | 0 | 0.0 | 0.0 | 0 | 1 | |
| **319792** | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | |
| **319793** | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | |
| **319794** | 0 | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | |

319795 rows × 16 columns

In [35]:

```
'''sns.pairplot(data = new_df , hue= 'HeartDisease')
plt.legend('HeartDisease')
plt.show()'''
```

Out[35]:

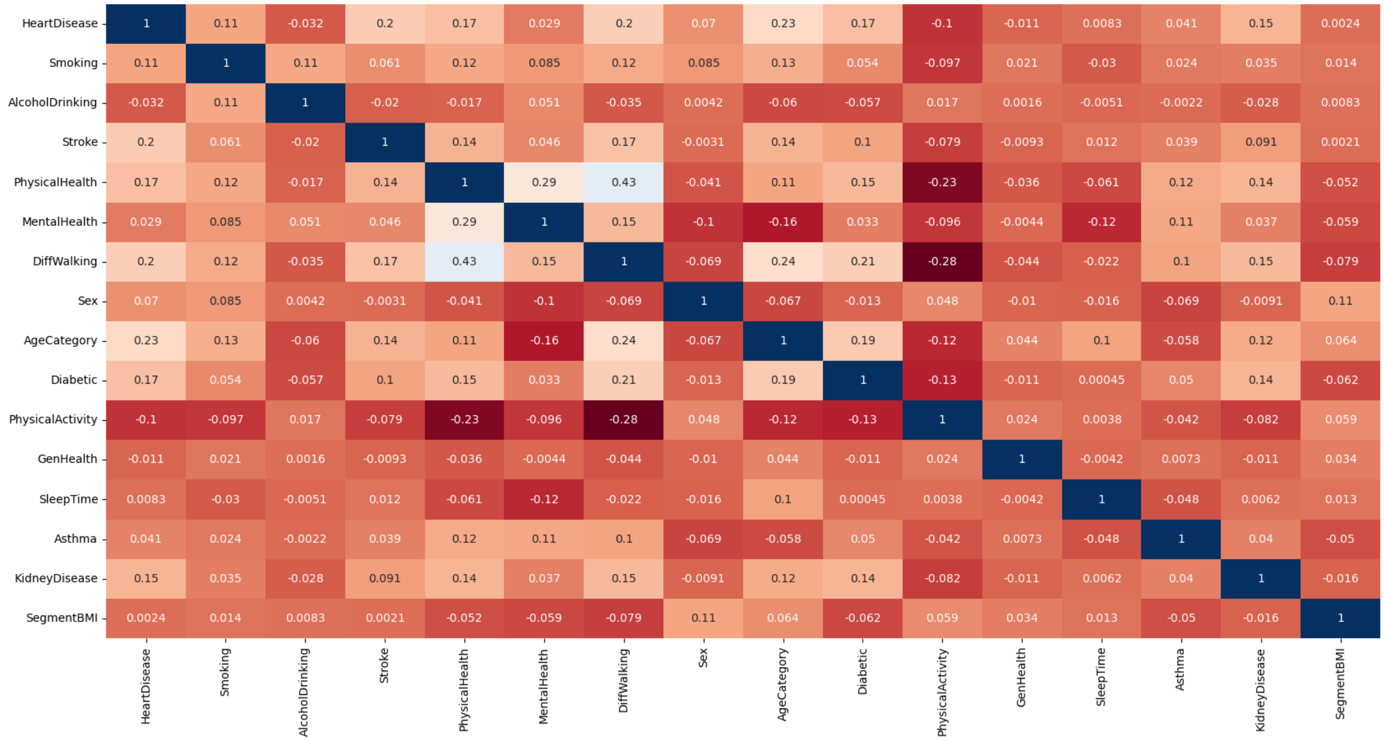
```
"sns.pairplot(data = new_df , hue= 'HeartDisease')\nplt.legend('HeartDisease')\nplt.show
()"
```

In [36]:

```
plt.figure(figsize=(20,10))
sns.heatmap(new_df.corr(),annot=True,cbar=False,cmap='RdBu')
plt.savefig("heatmap.png")
new_df.corr()
```

Out[36]:

| | HeartDisease | Smoking | AlcoholDrinking | Stroke | PhysicalHealth | MentalHealth | DiffWalking |
|---|---|---|---|---|---|---|---|
| **HeartDisease** | 1.000000 | 0.107764 | -0.032080 | 0.196835 | 0.170721 | 0.028591 | 0.201258 |
| **Smoking** | 0.107764 | 1.000000 | 0.111768 | 0.061226 | 0.115352 | 0.085157 | 0.120074 |
| **AlcoholDrinking** | -0.032080 | 0.111768 | 1.000000 | -0.019858 | -0.017254 | 0.051282 | -0.035328 |
| **Stroke** | 0.196835 | 0.061226 | -0.019858 | 1.000000 | 0.137014 | 0.046467 | 0.174143 |
| **PhysicalHealth** | 0.170721 | 0.115352 | -0.017254 | 0.137014 | 1.000000 | 0.287987 | 0.428373 |
| **MentalHealth** | 0.028591 | 0.085157 | 0.051282 | 0.046467 | 0.287987 | 1.000000 | 0.152235 |
| **DiffWalking** | 0.201258 | 0.120074 | -0.035328 | 0.174143 | 0.428373 | 0.152235 | 1.000000 |
| **Sex** | 0.070040 | 0.085052 | 0.004200 | -0.003091 | -0.040904 | -0.100058 | -0.068860 |
| **AgeCategory** | 0.233432 | 0.128331 | -0.059528 | 0.137822 | 0.110763 | -0.155506 | 0.243263 |
| **Diabetic** | 0.168553 | 0.053847 | -0.057372 | 0.101518 | 0.151361 | 0.032945 | 0.205502 |
| **PhysicalActivity** | -0.100030 | -0.097174 | 0.017487 | -0.079455 | -0.232283 | -0.095808 | -0.278524 |
| **GenHealth** | -0.011062 | 0.020625 | 0.001629 | -0.009335 | -0.035703 | -0.004412 | -0.043552 |
| **SleepTime** | 0.008327 | -0.030336 | -0.005065 | 0.011900 | -0.061387 | -0.119717 | -0.022216 |
| **Asthma** | 0.041444 | 0.024149 | -0.002202 | 0.038866 | 0.117907 | 0.114008 | 0.103222 |
| **KidneyDisease** | 0.145197 | 0.034920 | -0.028280 | 0.091167 | 0.142197 | 0.037281 | 0.153064 |
| **SegmentBMI** | 0.002387 | 0.013652 | 0.008293 | 0.002065 | -0.051750 | -0.059308 | -0.078957 |



In [37]:

```
X=new_df.drop(['HeartDisease'],axis=1)
y=new_df['HeartDisease']
```

In [38]:

```
X.head()
```

Out[38]:

|   | Smoking | AlcoholDrinking | Stroke | PhysicalHealth | MentalHealth | DiffWalking | Sex | AgeCategory | Diabetic | Ph |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 3.0 | 30.0 | 0 | 0 | 7 | 2 |  |
| **1** | 0 | 0 | 1 | 0.0 | 0.0 | 0 | 0 | 12 | 0 |  |
| **2** | 1 | 0 | 0 | 20.0 | 30.0 | 0 | 1 | 9 | 2 |  |
| **3** | 0 | 0 | 0 | 0.0 | 0.0 | 0 | 0 | 11 | 0 |  |
| **4** | 0 | 0 | 0 | 28.0 | 0.0 | 1 | 0 | 4 | 0 |  |

In [39]:

```
y.head()
```

Out[39]:

```
0    0
1    0
2    0
3    0
4    0
Name: HeartDisease, dtype: int32
```

In [40]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, f1_score,
```

# NearMiss(ubdersampling )

In [42]:

```
nm=NearMiss()
X_res,y_res=nm.fit_resample(X,y)
X_res.shape,y_res.shape
```

Out[42]:

```
((54746, 15), (54746,))
```

Loading [MathJax]/extensions/Safe.js

In [43]:

```
Xtrain,Xtest,ytrain,ytest = train_test_split(X_res, y_res, test_size = 0.2, random_state
```

In [44]:

```
bag_clf = BaggingClassifier(LogisticRegression(), n_estimators = 100, bootstrap = True,n
bag_clf.fit(Xtrain, ytrain)
bag_clf.oob_score_
```

Out[44]:

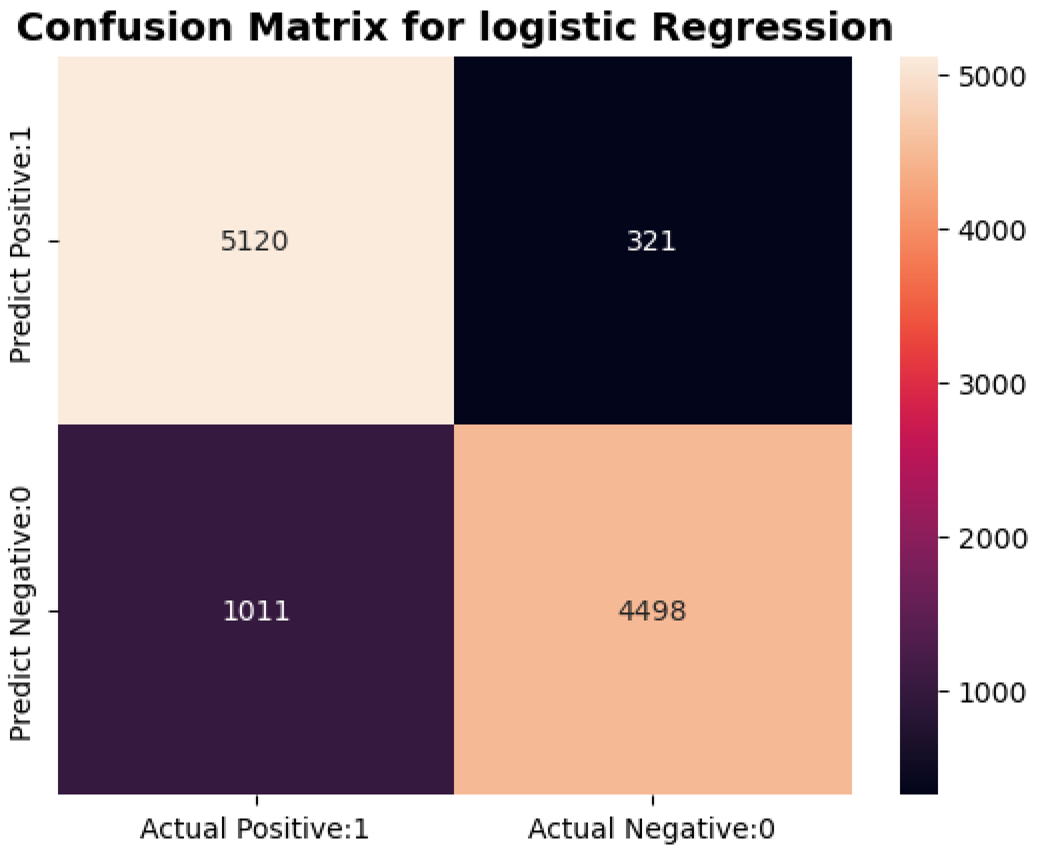
```
0.8776372271440315
```

In [45]:

```
y_pred_lr = bag_clf.predict(Xtest)

# Create the confusion matrix
cm = confusion_matrix(ytest, y_pred_lr)
#print(confusion_matrix_rf)

TN, FP, FN, TP = cm.ravel()
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], in
sns.heatmap(cm_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix for logistic Regression', fontsize=14, fontweight='bold')
```

Out[45]:

```
Text(0.5, 1.0, 'Confusion Matrix for logistic Regression')
```



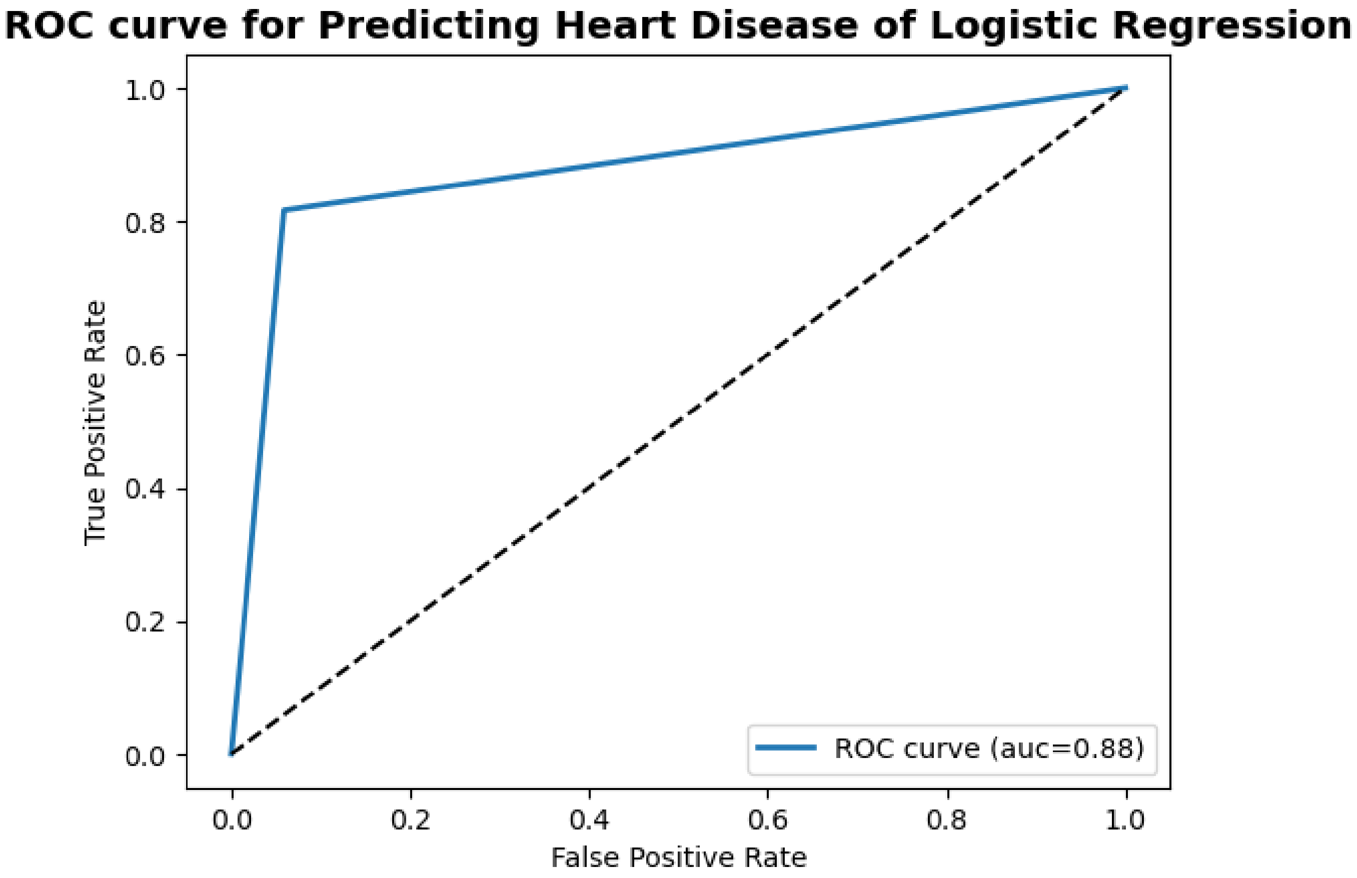
In [46]:

```
print("Classification Report of Test Dataset Logistics Regression\n")
print(classification_report(ytest, y_pred_lr))
```

```
Classification Report of Test Dataset Logistics Regression

              precision    recall  f1-score   support

           0       0.84      0.94      0.88      5441
           1       0.93      0.82      0.87      5509

    accuracy                           0.88     10950
   macro avg       0.88      0.88      0.88     10950
weighted avg       0.88      0.88      0.88     10950
```

In [47]:

```
y_prob_test = bag_clf.predict_proba(Xtest)[:,1]
fpr, tpr, thresholds = roc_curve(ytest, y_pred_lr)

auc = round(roc_auc_score(ytest, y_pred_lr), 2)

plt.plot(fpr, tpr, linewidth=2, label="ROC curve (auc=" + str(auc) + ")")
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Predicting Heart Disease of Logistic Regression', fontsize=14,
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



In [48]:

```
# Evaluate model on training dataset
train_accuracy = accuracy_score(ytrain, bag_clf.predict(Xtrain))
print("Training Accuracy for logistic regression:", round(train_accuracy, 2))

# Evaluate model on testing dataset
test_accuracy = accuracy_score(ytest,  bag_clf.predict(Xtest))
print("Testing Accuracy for logistic regression:", round(test_accuracy, 2))
```

```
Training Accuracy for logistic regression: 0.88
Testing Accuracy for logistic regression: 0.88
```

In [49]:

```
ypred = bag_clf.predict(Xtest)
print("Accuracy Score : ", accuracy_score(ytest, ypred))
print("Precision Score : ", precision_score(ytest, ypred))
print("Recall Score : ", recall_score(ytest, ypred))
print("F1 Score : ", f1_score(ytest, ypred))
```

```
Accuracy Score :  0.8783561643835617
Precision Score :  0.9333886698485163
Recall Score :  0.8164821201669995
F1 Score :  0.8710302091402015
```

Loading [MathJax]/extensions/Safe.js

In [50]:
```
bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators = 50,
                             bootstrap = True,
                            n_jobs = -1, oob_score = True)
bag_clf.fit(Xtrain, ytrain)
bag_clf.oob_score_
```

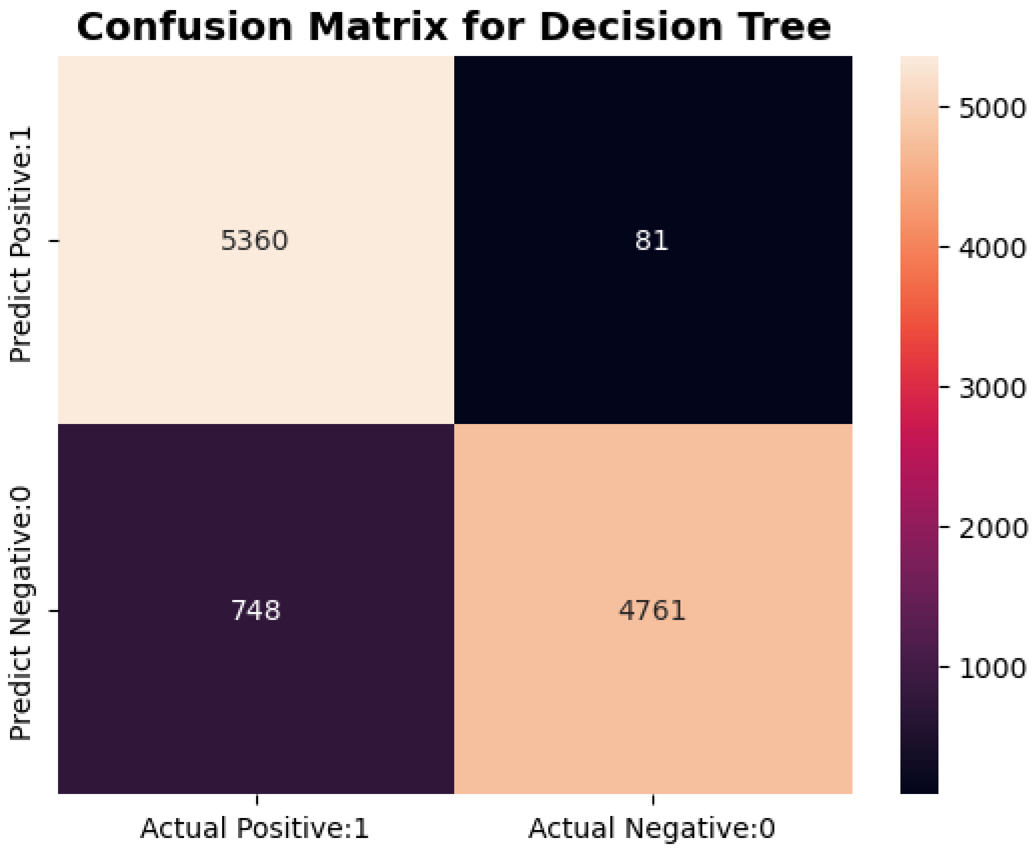
Out[50]:
```
0.9212256827107499
```

In [51]:
```
y_pred_df = bag_clf.predict(Xtest)

# Create the confusion matrix
confusion_matrix_rf = confusion_matrix(ytest, y_pred_df)
print(confusion_matrix_rf)
```

```
[[5360   81]
 [ 748 4761]]
```

In [52]:
```
y_pred_df = bag_clf.predict(Xtest)

# Create the confusion matrix
cm = confusion_matrix(ytest, y_pred_df)
#print(confusion_matrix_rf)

TN, FP, FN, TP = cm.ravel()
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], in
sns.heatmap(cm_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix for Decision Tree', fontsize=14, fontweight='bold')
```

Out[52]:
```
Text(0.5, 1.0, 'Confusion Matrix for Decision Tree')
```



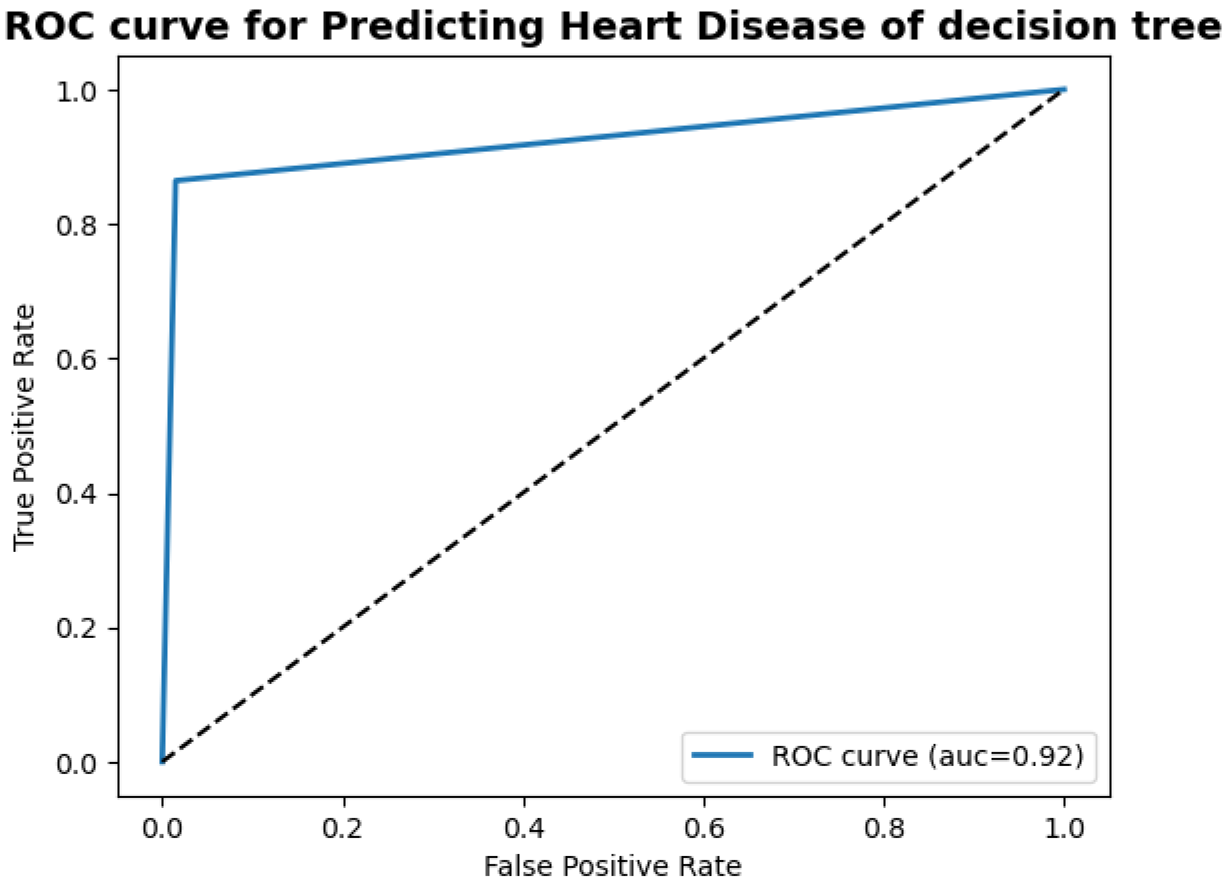
In [53]:
```
print("Classification Report of Test Dataset for Decision Tree\n")
print(classification_report(ytest, y_pred_df))
```

Loading [MathJax]/extensions/Safe.js

```
Classification Report of Test Dataset for Decision Tree

              precision    recall  f1-score   support

           0       0.88      0.99      0.93      5441
           1       0.98      0.86      0.92      5509

    accuracy                           0.92     10950
   macro avg       0.93      0.92      0.92     10950
weighted avg       0.93      0.92      0.92     10950
```

In [54]:

```
y_prob_test = bag_clf.predict_proba(Xtest)[:,1]
fpr, tpr, thresholds = roc_curve(ytest, y_pred_df)

auc = round(roc_auc_score(ytest, y_pred_df), 2)

plt.plot(fpr, tpr, linewidth=2, label="ROC curve (auc=" + str(auc) + ")")
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Predicting Heart Disease of decision tree', fontsize=14, fontwe
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



In [55]:

```
# Evaluate model on training dataset
train_accuracy = accuracy_score(ytrain, bag_clf.predict(Xtrain))
print("Training Accuracyfor Decision Tree:", round(train_accuracy, 2))

# Evaluate model on testing dataset
test_accuracy = accuracy_score(ytest,  bag_clf.predict(Xtest))
print("Testing Accuracyfor Decision Tree:", round(test_accuracy, 2))
```

```
Training Accuracyfor Decision Tree: 0.93
Testing Accuracyfor Decision Tree: 0.92
```

Loading [MathJax]/extensions/Safe.js

In [56]:

```
ypred = bag_clf.predict(Xtest)
print("Accuracy Score : ", accuracy_score(ytest, ypred))
print("Precision Score : ", precision_score(ytest, ypred))
print("Recall Score : ", recall_score(ytest, ypred))
print("F1 Score : ", f1_score(ytest, ypred))
```

```
Accuracy Score :  0.9242922374429223
Precision Score :  0.983271375464684
Recall Score :  0.8642221818841895
F1 Score :  0.9199111196985799
```

In [57]:

```
rnd_clf = RandomForestClassifier(n_estimators = 50, n_jobs = -1, oob_score= True)
rnd_clf.fit(Xtrain, ytrain)
rnd_clf.oob_score_
```

Out[57]:

```
0.9217965110969039
```

In [58]:

```
y_pred_rf = bag_clf.predict(Xtest)

# Create the confusion matrix
confusion_matrix_rf = confusion_matrix(ytest, y_pred_rf)
print(confusion_matrix_rf)
```

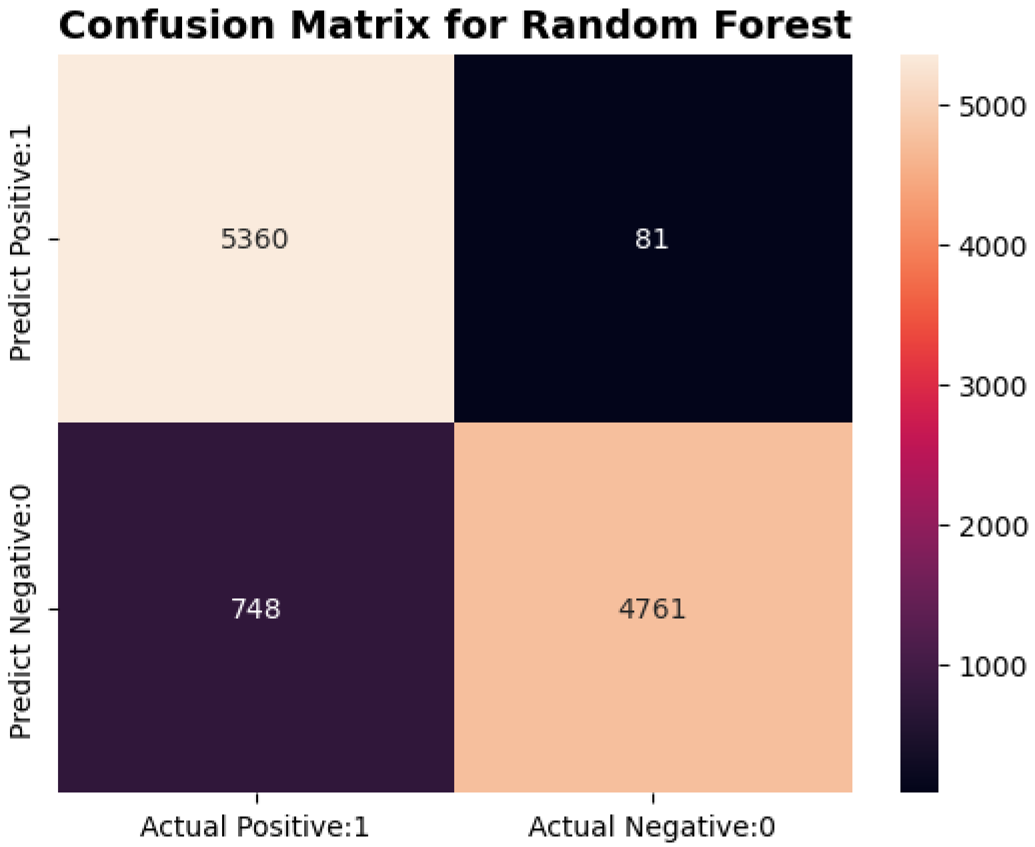
```
[[5360   81]
 [ 748 4761]]
```

In [59]:

```
y_pred_rf = bag_clf.predict(Xtest)

# Create the confusion matrix
cm = confusion_matrix(ytest, y_pred_rf)
#print(confusion_matrix_rf)

TN, FP, FN, TP = cm.ravel()
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], in
sns.heatmap(cm_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix for Random Forest', fontsize=14, fontweight='bold')
```

Out[59]:

```
Text(0.5, 1.0, 'Confusion Matrix for Random Forest')
```



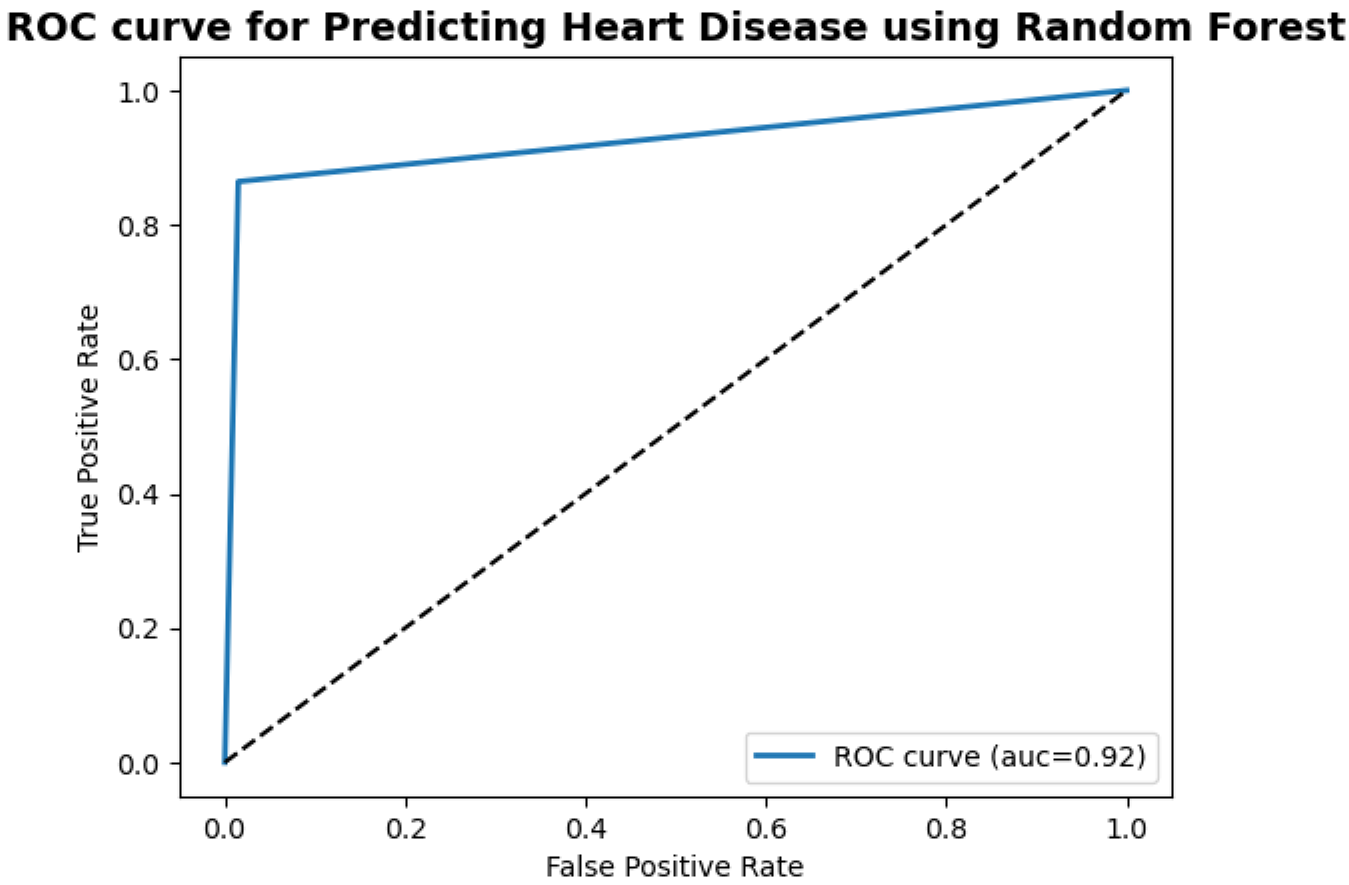
In [60]:

```
print("Classification Report of Test Dataset for Random Forest\n")
print(classification_report(ytest, y_pred_rf))
```

```
Classification Report of Test Dataset for Random Forest

              precision    recall  f1-score   support

           0       0.88      0.99      0.93      5441
           1       0.98      0.86      0.92      5509

    accuracy                           0.92     10950
   macro avg       0.93      0.92      0.92     10950
weighted avg       0.93      0.92      0.92     10950
```

In [61]:

```
y_prob_test = bag_clf.predict_proba(Xtest)[:,1]
fpr, tpr, thresholds = roc_curve(ytest, y_pred_rf)

auc = round(roc_auc_score(ytest, y_pred_rf), 2)

plt.plot(fpr, tpr, linewidth=2, label="ROC curve (auc=" + str(auc) + ")")
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Predicting Heart Disease using Random Forest', fontsize=14, fon
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



In [62]:

```
ypred = rnd_clf.predict(Xtest)
print("Accuracy Score : ", accuracy_score(ytest, ypred))
print("Precision Score : ", precision_score(ytest, ypred))
print("Recall Score : ", recall_score(ytest, ypred))
print("F1 Score : ", f1_score(ytest, ypred))
```

```
Accuracy Score :  0.9252054794520548
Precision Score :  0.9841040462427746
Recall Score :  0.8653113087674714
F1 Score :  0.920892494929006
```

In [63]:

```
# Evaluate model on training dataset
train_accuracy = accuracy_score(ytrain, rnd_clf.predict(Xtrain))
print("Training Accuracy for Random Forest:", round(train_accuracy, 2))

# Evaluate model on testing dataset
test_accuracy = accuracy_score(ytest,  rnd_clf.predict(Xtest))
print("Testing Accuracy for Random Forest:", round(test_accuracy, 2))
```

```
Training Accuracy for Random Forest: 0.93
Testing Accuracy for Random Forest: 0.93
```

## SMOTE (oversampling)

In [64]:

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state = 0)
sm.fit(X,y)
x_resem, y_resem = sm.fit_resample(X, y)
```

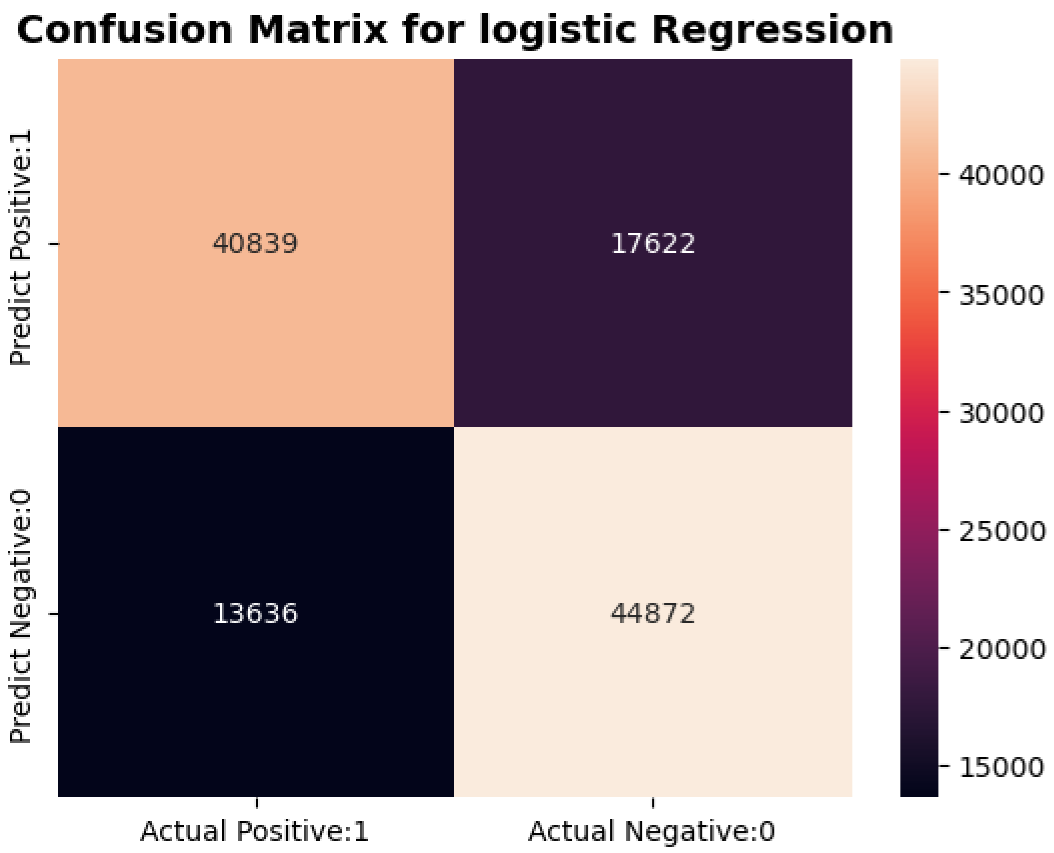
In [65]:

```
x_resem.shape,y_resem.shape
```

Out[65]:

```
((584844, 15), (584844,))
```

Loading [MathJax]/extensions/Safe.js

In [66]:
```
y_resem.value_counts()
```
Out[66]:
```
0    292422
1    292422
Name: HeartDisease, dtype: int64
```
In [67]:
```
Xtrain,Xtest,ytrain,ytest = train_test_split(x_resem, y_resem, test_size = 0.2, random_s
```
In [68]:
```
bag_clf = BaggingClassifier(LogisticRegression(), n_estimators = 100, bootstrap = True,n
bag_clf.fit(Xtrain, ytrain)
bag_clf.oob_score_
```
Out[68]:
```
0.7344461661768634
```
In [69]:
```
y_pred_lr = bag_clf.predict(Xtest)

# Create the confusion matrix
cm = confusion_matrix(ytest, y_pred_lr)
#print(confusion_matrix_rf)

TN, FP, FN, TP = cm.ravel()
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], in
sns.heatmap(cm_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix for logistic Regression', fontsize=14, fontweight='bold')
```
Out[69]:
```
Text(0.5, 1.0, 'Confusion Matrix for logistic Regression')
```


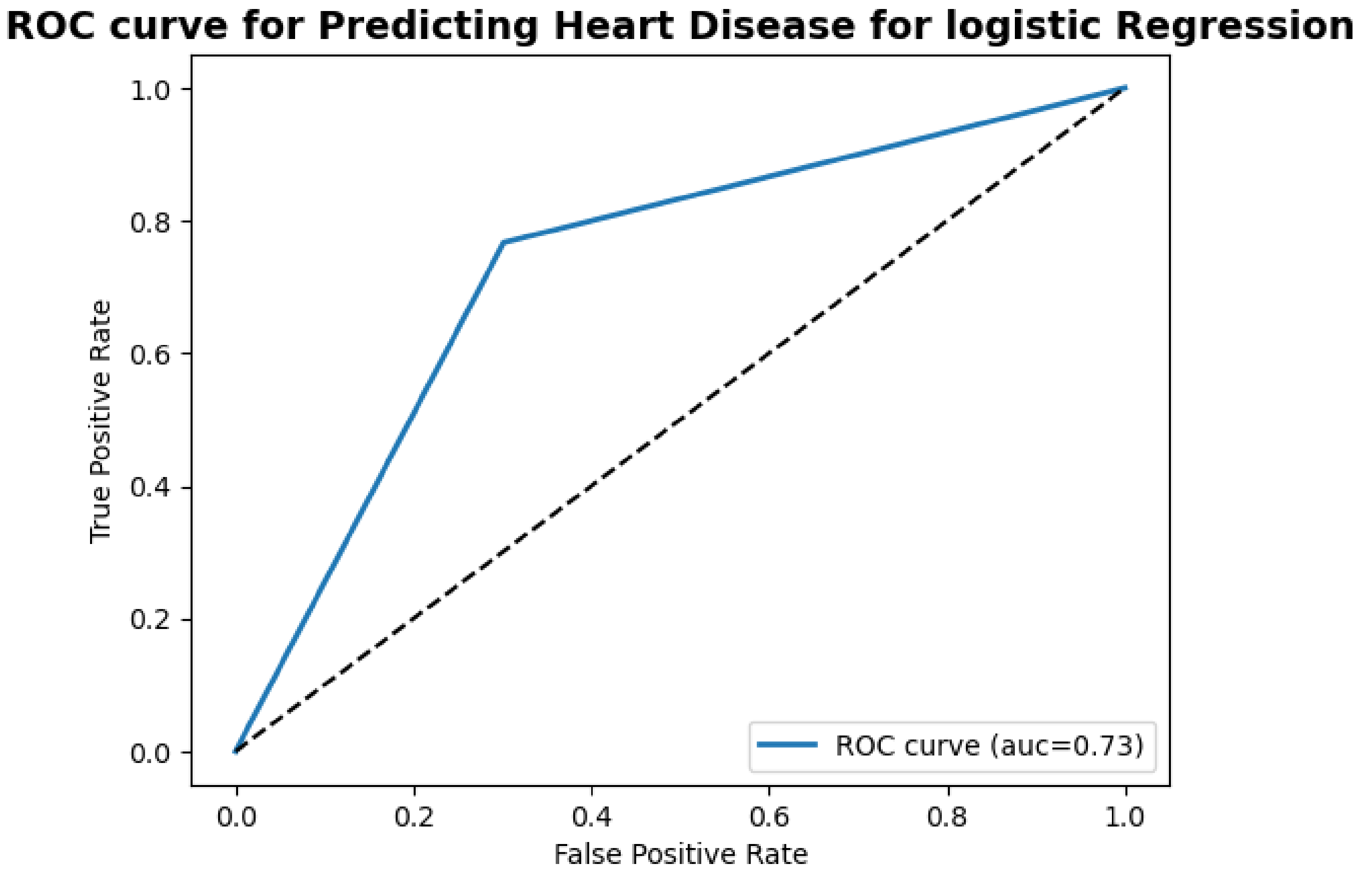
Confusion Matrix for logistic Regression

In [70]:
```
print("Classification Report of Test Dataset for logistic Regression\n")
print(classification_report(ytest, y_pred_lr))
```

```
Classification Report of Test Dataset for logistic Regression

              precision    recall  f1-score   support

           0       0.75      0.70      0.72     58461
           1       0.72      0.77      0.74     58508

    accuracy                           0.73    116969
   macro avg       0.73      0.73      0.73    116969
weighted avg       0.73      0.73      0.73    116969
```

In [71]:

```
y_prob_test = bag_clf.predict_proba(Xtest)[:,1]
fpr, tpr, thresholds = roc_curve(ytest, y_pred_lr)

auc = round(roc_auc_score(ytest, y_pred_lr), 2)

plt.plot(fpr, tpr, linewidth=2, label="ROC curve (auc=" + str(auc) + ")")
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Predicting Heart Disease for logistic Regression', fontsize=14,
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



In [72]:

```
ypred = bag_clf.predict(Xtest)
print("Accuracy Score : ", accuracy_score(ytest, ypred))
print("Precision Score : ", precision_score(ytest, ypred))
print("Recall Score : ", recall_score(ytest, ypred))
print("F1 Score : ", f1_score(ytest, ypred))
```

```
Accuracy Score :  0.732766801460216
Precision Score :  0.7180209300092809
Recall Score :  0.7669378546523552
F1 Score :  0.7416736913439448
```

In [73]:

```
# Evaluate model on training dataset
train_accuracy = accuracy_score(ytrain, bag_clf.predict(Xtrain))
print("Training Accuracy for logistic Regression:", round(train_accuracy, 2))

# Evaluate model on testing dataset
test_accuracy = accuracy_score(ytest,  bag_clf.predict(Xtest))
print("Testing Accuracy for logistic Regression:", round(test_accuracy, 2))
```

```
Training Accuracy for logistic Regression: 0.73
Testing Accuracy for logistic Regression: 0.73
```

In [74]:

```
bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators = 50,
                             bootstrap = True,
                            n_jobs = -1, oob_score = True)
bag_clf.fit(Xtrain, ytrain)
bag_clf.oob_score_
```

Out[74]:

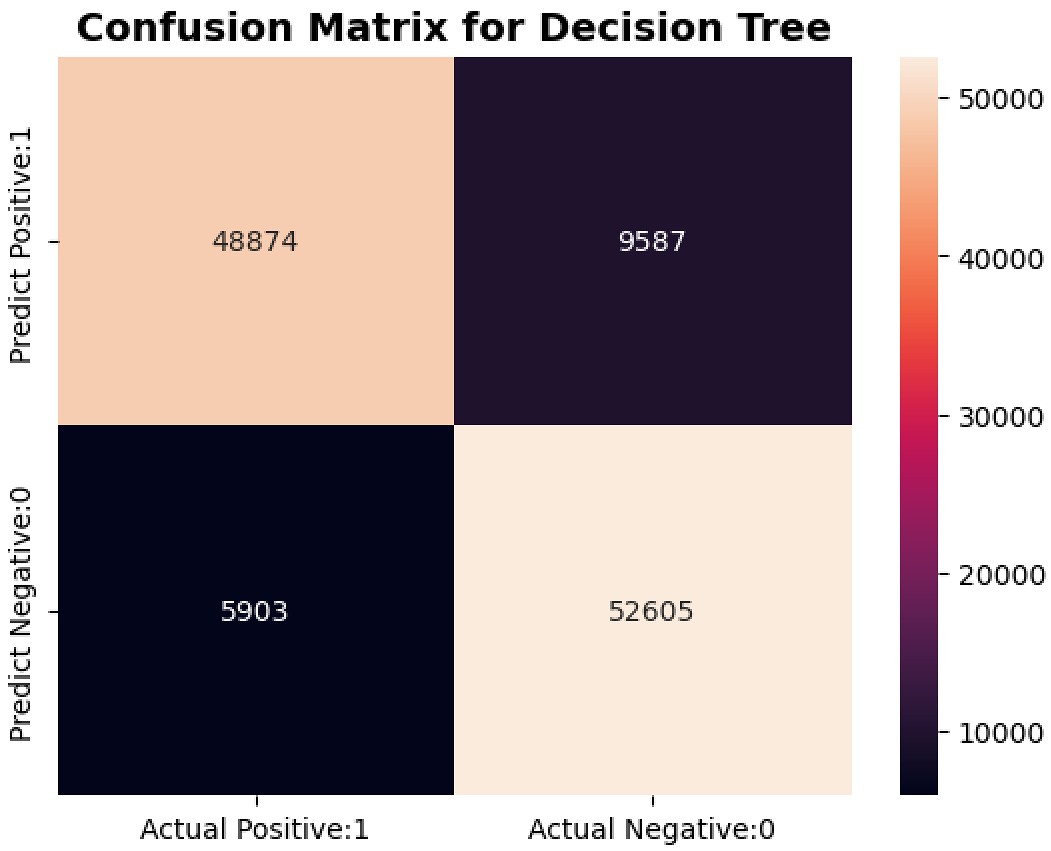
```
0.8678706919583222
```

In [75]:

```
y_pred_df = bag_clf.predict(Xtest)

# Create the confusion matrix
cm = confusion_matrix(ytest, y_pred_df)
#print(confusion_matrix_rf)

TN, FP, FN, TP = cm.ravel()
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], in
sns.heatmap(cm_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix for Decision Tree', fontsize=14, fontweight='bold')
```

Out[75]:

```
Text(0.5, 1.0, 'Confusion Matrix for Decision Tree')
```



In [76]:

```
print("Classification Report of Test Dataset for Decision Tree\n")
print(classification_report(ytest, y_pred_df))
```

Loading [MathJax]/extensions/Safe.js

```
Classification Report of Test Dataset for Decision Tree

              precision    recall  f1-score   support

           0       0.89      0.84      0.86     58461
           1       0.85      0.90      0.87     58508

    accuracy                           0.87    116969
   macro avg       0.87      0.87      0.87    116969
weighted avg       0.87      0.87      0.87    116969
```

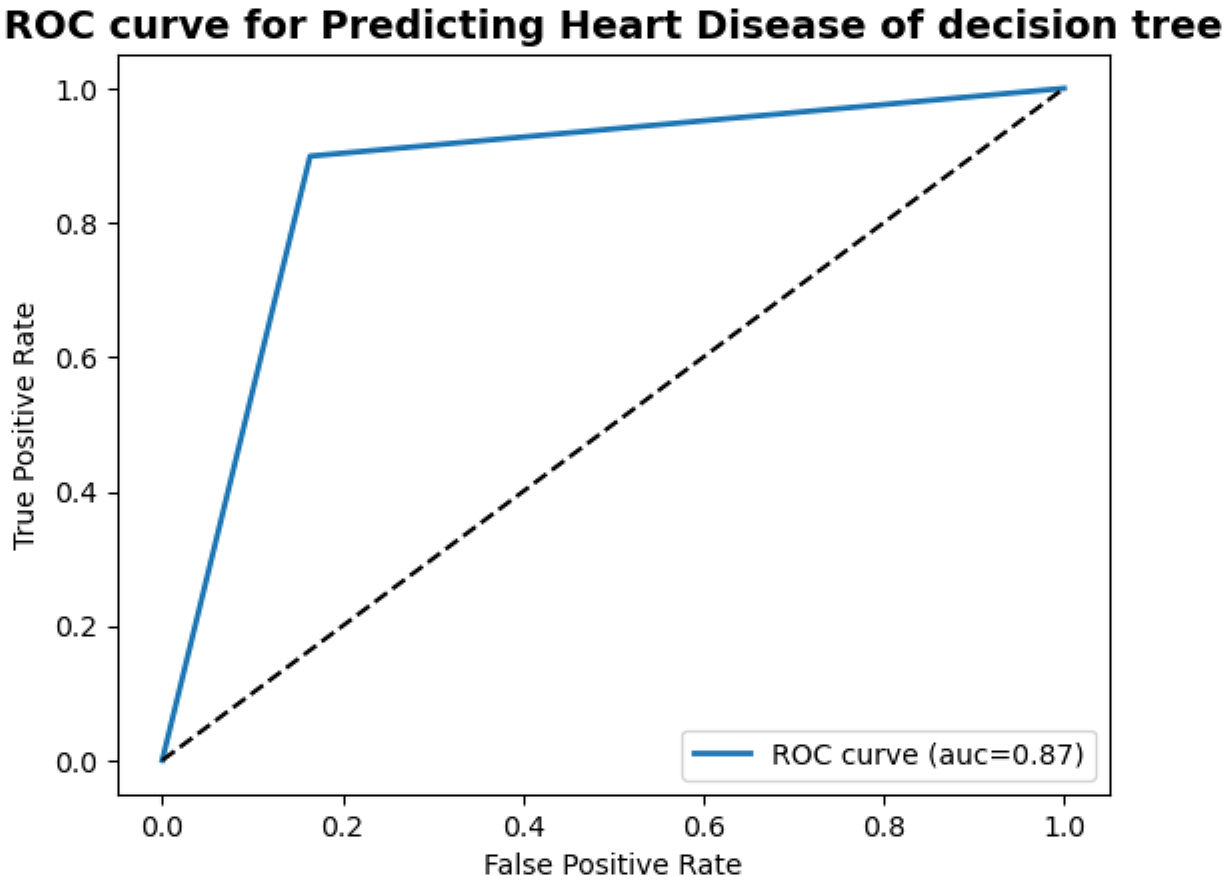
In [77]:

```python
train_accuracy = accuracy_score(ytrain, bag_clf.predict(Xtrain))
print("Training Accuracy for Random Forest:", round(train_accuracy, 2))

# Evaluate model on testing dataset
test_accuracy = accuracy_score(ytest,  bag_clf.predict(Xtest))
print("Testing Accuracy for Random Forest:", round(test_accuracy, 2))
```

```
Training Accuracy for Random Forest: 0.92
Testing Accuracy for Random Forest: 0.87
```

In [78]:

```python
y_prob_test = bag_clf.predict_proba(Xtest)[:,1]
fpr, tpr, thresholds = roc_curve(ytest, y_pred_df)

auc = round(roc_auc_score(ytest, y_pred_df), 2)

plt.plot(fpr, tpr, linewidth=2, label="ROC curve (auc=" + str(auc) + ")")
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Predicting Heart Disease of decision tree', fontsize=14, fontwe
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



In [79]:

```
ypred = bag_clf.predict(Xtest)
print("Accuracy Score : ", accuracy_score(ytest, ypred))
print("Precision Score : ", precision_score(ytest, ypred))
print("Recall Score : ", recall_score(ytest, ypred))
print("F1 Score : ", f1_score(ytest, ypred))
```

```
Accuracy Score :  0.8675717497798562
Precision Score :  0.8458483406225881
Recall Score :  0.899107814315991
F1 Score :  0.8716652858326429
```

In [80]:

```
rnd_clf = RandomForestClassifier(n_estimators = 50, n_jobs = -1, oob_score= True)
rnd_clf.fit(Xtrain, ytrain)
rnd_clf.oob_score_
```

Out[80]:

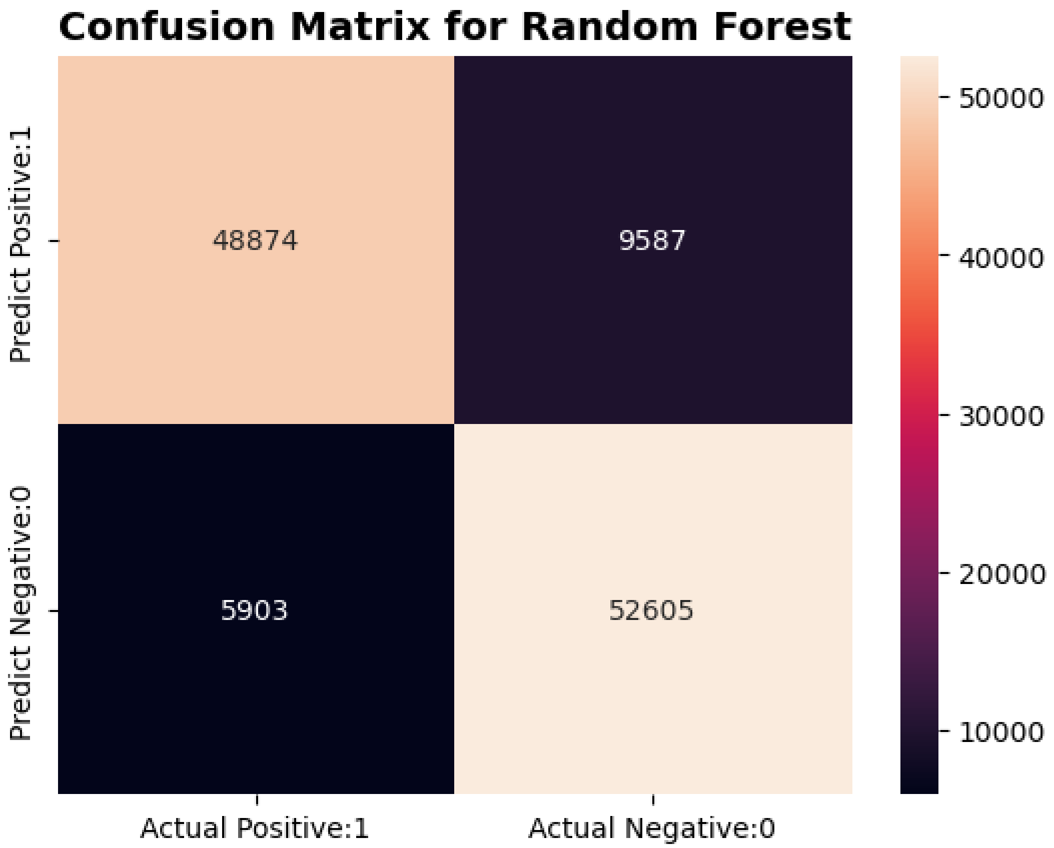
```
0.8680138925995191
```

In [81]:

```
y_pred_rf = bag_clf.predict(Xtest)

# Create the confusion matrix
cm = confusion_matrix(ytest, y_pred_rf)
#print(confusion_matrix_rf)

TN, FP, FN, TP = cm.ravel()
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'], in
sns.heatmap(cm_matrix, annot=True, fmt='d')
plt.title('Confusion Matrix for Random Forest', fontsize=14, fontweight='bold')
```

Out[81]:

```
Text(0.5, 1.0, 'Confusion Matrix for Random Forest')
```



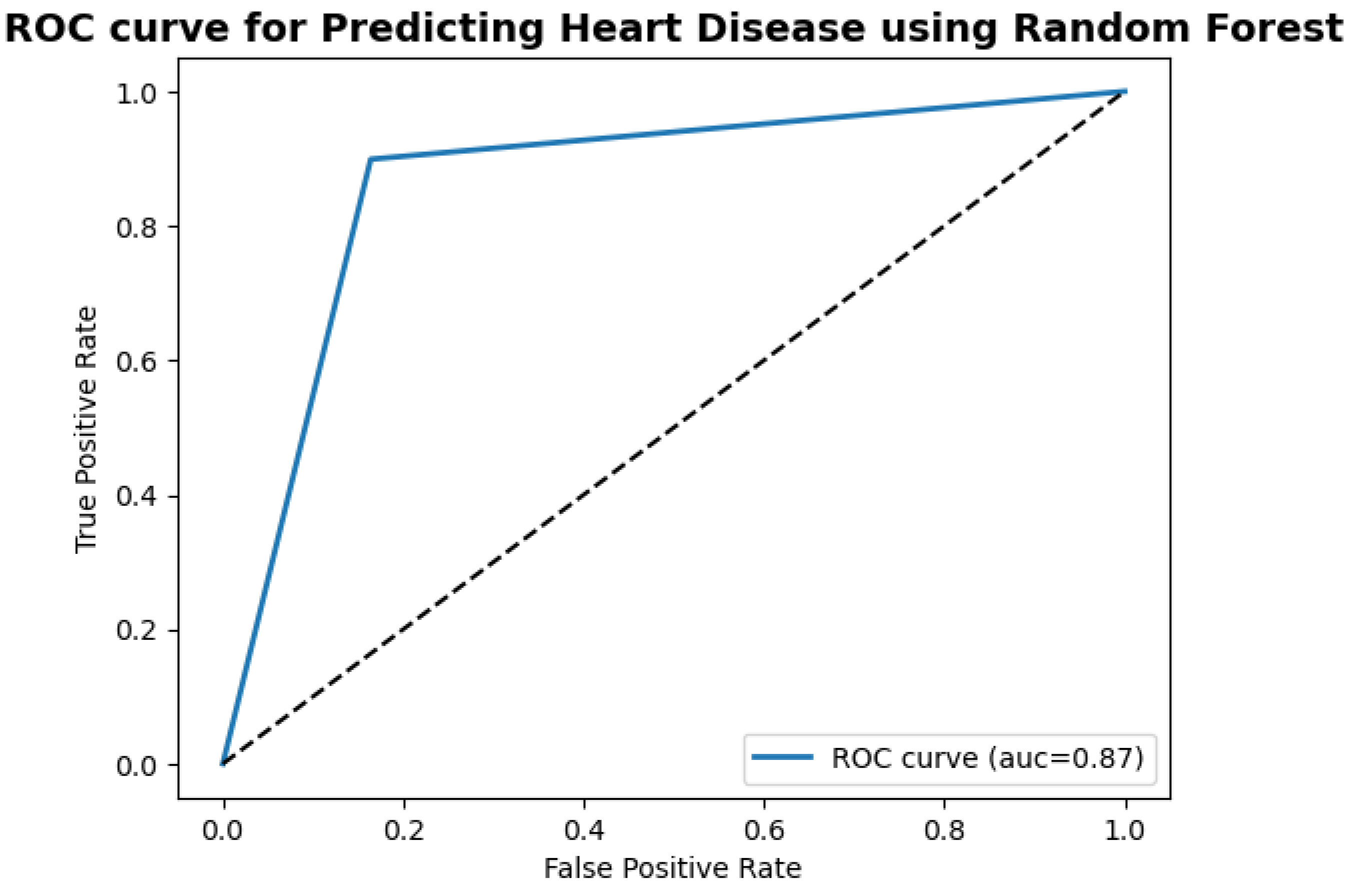
In [82]:

```
print("Classification Report of Test Dataset for Random Forest\n")
print(classification_report(ytest, y_pred_rf))
```

```
Classification Report of Test Dataset for Random Forest

              precision    recall  f1-score   support

           0       0.89      0.84      0.86     58461
           1       0.85      0.90      0.87     58508

    accuracy                           0.87    116969
   macro avg       0.87      0.87      0.87    116969
weighted avg       0.87      0.87      0.87    116969
```

In [83]:

```
y_prob_test = bag_clf.predict_proba(Xtest)[:,1]
fpr, tpr, thresholds = roc_curve(ytest, y_pred_rf)

auc = round(roc_auc_score(ytest, y_pred_rf), 2)

plt.plot(fpr, tpr, linewidth=2, label="ROC curve (auc=" + str(auc) + ")")
plt.plot([0,1], [0,1], 'k--' )
plt.title('ROC curve for Predicting Heart Disease using Random Forest', fontsize=14, fon
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```



In [84]:

```
ypred = rnd_clf.predict(Xtest)
print("Accuracy Score : ", accuracy_score(ytest, ypred))
print("Precision Score : ", precision_score(ytest, ypred))
print("Recall Score : ", recall_score(ytest, ypred))
print("F1 Score : ", f1_score(ytest, ypred))
```

```
Accuracy Score :  0.8690165770417803
Precision Score :  0.8462994146419693
Recall Score :  0.9019450331578588
F1 Score :  0.8732366398318758
```

In [85]:

```
# Evaluate model on training dataset
train_accuracy = accuracy_score(ytrain, bag_clf.predict(Xtrain))
print("Training Accuracy for Random Forest:", round(train_accuracy, 2))

# Evaluate model on testing dataset
test_accuracy = accuracy_score(ytest,  bag_clf.predict(Xtest))
print("Testing Accuracy for Random Forest:", round(test_accuracy, 2))
```

```
Training Accuracy for Random Forest: 0.92
Testing Accuracy for Random Forest: 0.87
```

In [ ]:

In [ ]: