Heart Disease: Categorical ML Modeling

1. Project Overview:

The purpose of this project is to use categorical multiple machine learning models to solve a business problem.

My chosen problem is to see if I can predict heart disease from a variety of general factors. The business problem in this case is for a web application where, with just a few questions, an individual or their doctor could screen for the possibility that they have heart disease.

In [201]:

```
# Load Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.ticker as mtick
import sqlite3
import seaborn as sns
from imblearn.over_sampling import SMOTENC
from sklearn.linear_model import LinearRegression
from sklearn import tree, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classificatio
n_report, plot_confusion_matrix, recall_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, Ext
raTreesClassifier
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardSc
aler
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc, f1_score, make_scorer, recall_s
core
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
from matplotlib.pyplot import figure
from bs4 import BeautifulSoup
import time
import requests     # to get images
import shutil       # to save files locally
import datetime
from scipy.stats import norm
import warnings
warnings.filterwarnings('ignore')
import xgboost
from xgboost import XGBClassifier
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE, ADASYN
import random
from random import randint
from sklearn.datasets import *
from IPython.display import Image, display_svg, SVG
import os
from dtreeviz.trees import *
from sklearn.tree import plot_tree
os.environ["PATH"] += os.pathsep + "C:\\Users\\tmcro\\anaconda3\\pkgs\\grap
hviz-2.38-hfd603c8_2\\Library\\bin\\graphviz\\"
```

B) The Data

This project utilizes a dataset from kaggle: https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease (https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease)

This dataset come from the CDC and is a major part of the Behavioral Risk Factor Surveillance System (BRFSS), which conducts annual telephone surveys to gather data on the health status of U.S. residents.

According to the CDC: 'Established in 1984 with 15 states, BRFSS now collects data in all 50 states as well as the District of Columbia and three U.S. territories. BRFSS completes more than 400,000 adult interviews each year, making it the largest continuously conducted health survey system in the world.'

In [202]:
```
# Load Data
df = pd.read_csv('heart_2020_cleaned.csv')
df.head()
```

Out[202]:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffW
0	No	16.60	Yes	No	No	3.0	30.0	
1	No	20.34	No	No	Yes	0.0	0.0	
2	No	26.58	Yes	No	No	20.0	30.0	
3	No	24.21	No	No	No	0.0	0.0	
4	No	23.71	No	No	No	28.0	0.0	

Define Target Variable

In this case, the target variable is whether or not an individual had heart disease.

In [203]:
```
target = ['HeartDisease']
```

Define Scoring Metric

For the purposes of this analysis, I think a custom scoring metric is necessary.

My reasoning is this:

- False negatives could cause patients with heart disease to not recieve further testing. This would be the worst possibility, out of the options.
- False positives would cost more due to testing people who did not actually have heart disease, or could cause people without heart disease to needlessly worry about their health. This is also costly, but not as costly as missing an individual with heart disease.

Thus, I want to minimize false negatives while keeping false positives to an appropriate level. The F1-score is a geometric average of recall and precision. I will make a recall-weighted F-score by adding a 2x weight to the recall (or false negatives) in this equation.

In [204]:
```
def my_custom_score(y_true, y_pred):
    cf = confusion_matrix(y_true, y_pred)
    precision = cf[1,1] / sum(cf[:,1])
    recall    = cf[1,1] / sum(cf[1,:])
    f1_score  = 2*precision*recall / (precision + recall)
    rwf_score = 2*precision* (recall*2) /(precision + (recall*2))
    return rwf_score

my_scorer = make_scorer(my_custom_score, greater_is_better= True)

# Change class metric
class_metric = my_scorer
```

Describe Data

The data contains 18 variables and approximately 320,000 observations.

The variables in the dataset include:

- 1. HeartDisease - Respondents that have ever reported having coronary heart disease (CHD) or myocardial infarction (MI)
- 1. Smoking (Question: Have you smoked at least 100 cigarettes in your entire life? [Note: 5 packs = 100 cigarettes])
- 1. AlcoholDrinking (Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week)
- 1. Stroke - Has the individual had a stroke?
- 1. PhysicalHealth -ORDINAL Categorical Variable - (Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good? (0-30 days)
- 1. MentalHealth - ORDINAL Categorical Variable - (Thinking about your mental health, for how many days during the past 30 days was your mental health not good? (0-30 days))
- 1. DiffWalking (Do you have serious difficulty walking or climbing stairs?)
- 1. Sex - Male or Female
- 1. AgeCategory - ORDINAL Categorical Variable (Fourteen-level age category)
- 1. Race
- 1. Diabetic - Yes/No/Borderline
- 1. PhysicalActivity - Adults who reported doing physical activity or exercise during the past 30 days other than their regular job
- 1. GenHealth - Is the individuals general health good / fair/ poor / very good / great?
- 1. Asthma - Yes/No
- 1. KidneyDisease - Yes/No
- 1. SkinCancer - Yes/No
- 1. SleepTime - How many hours per night do you sleep (Continuous Variable)
- 1. BMI - What is your body mass index

In [205]:
```python
# Describe Data
df.describe().round(2)
```

Out[205]:

	BMI	PhysicalHealth	MentalHealth	SleepTime
count	319795.00	319795.00	319795.00	319795.00
mean	28.33	3.37	3.90	7.10
std	6.36	7.95	7.96	1.44
min	12.02	0.00	0.00	1.00
25%	24.03	0.00	0.00	6.00
50%	27.34	0.00	0.00	7.00
75%	31.42	2.00	3.00	8.00
max	94.85	30.00	30.00	24.00

Check for missing values

In [206]:
```python
nothere = df.isna().sum()
nothere = pd.DataFrame(nothere)
nothere = nothere.loc[nothere[0] > 0]
nothere
```

Out[206]:

0

Check Dtypes

In [207]:
```
df.dtypes
```

Out[207]:
```
HeartDisease          object
BMI                  float64
Smoking               object
AlcoholDrinking       object
Stroke                object
PhysicalHealth       float64
MentalHealth         float64
DiffWalking           object
Sex                   object
AgeCategory           object
Race                  object
Diabetic              object
PhysicalActivity      object
GenHealth             object
SleepTime            float64
Asthma                object
KidneyDisease         object
SkinCancer            object
dtype: object
```

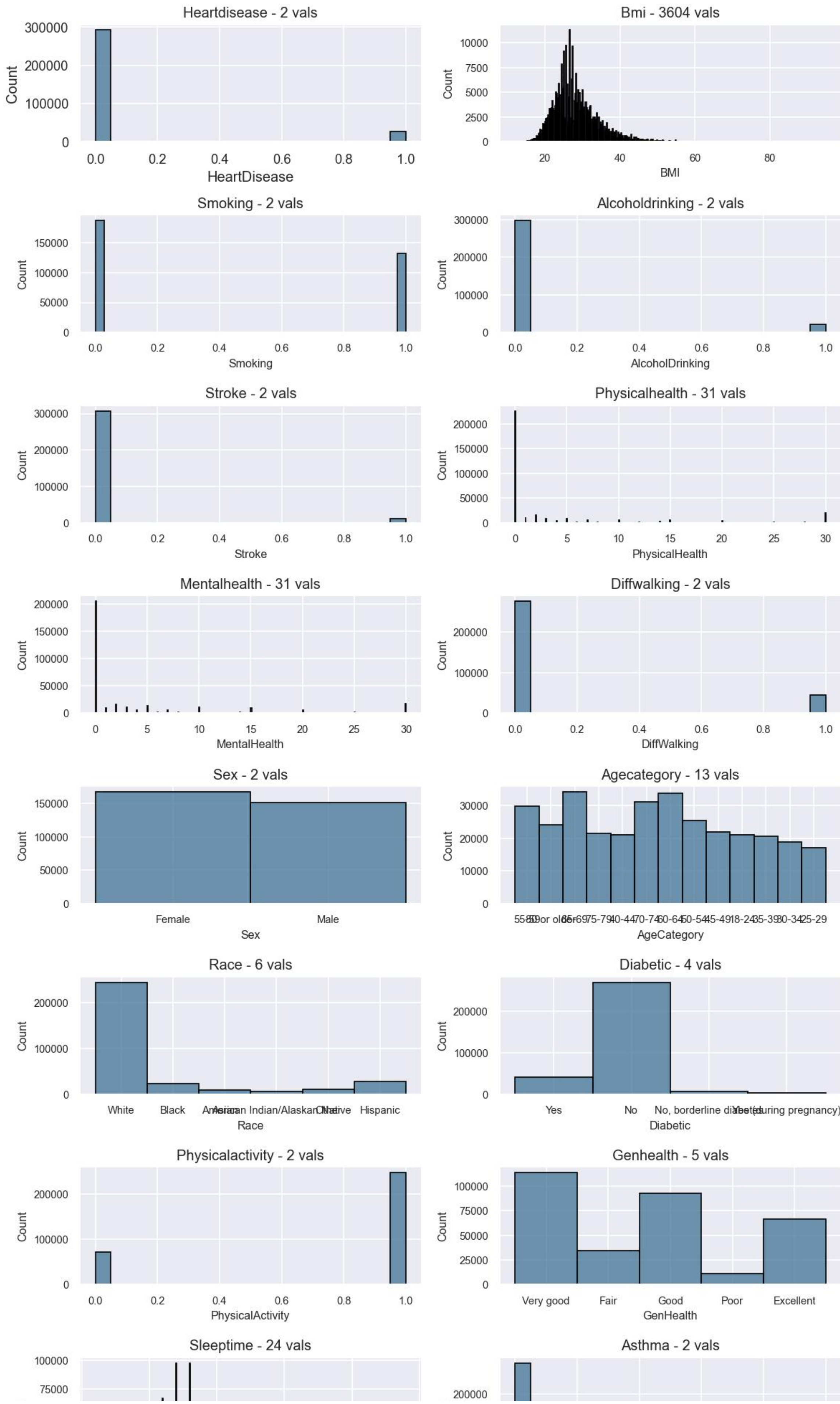
Set visuals

In [208]:
```
# Set visual parameters for plots
plt.rcParams.update({'font.family':'Open Sans'})
plt.rcParams['figure.figsize'] = (7,5)
sns.set_style('darkgrid')
sns.set(font_scale = 1.25)

# Primary Colors
bluez = '#417396'
redz = '#802b37'
```

Check Distributions

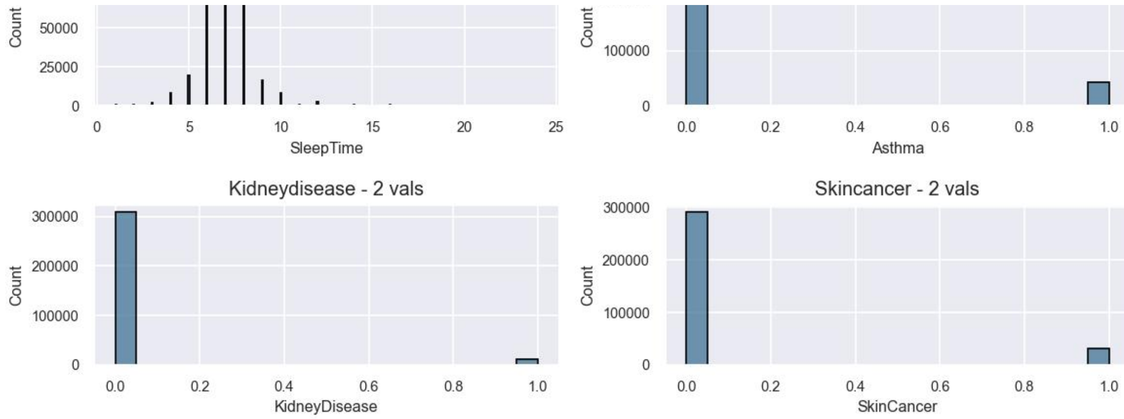
In [305]:

```
# Check the Variable's Distributions
num = len(df.columns)/2
plt.figure(figsize = (10,20), dpi = 120)

for n, column in enumerate(df.columns, 1):
    plt.subplot(int(num),2,n)
    sns.set(font_scale = .8)
    sns.histplot(df[column], color='#417396', edgecolor="black", linewidth=
1)
    plt.tight_layout()
    col = str.capitalize(column)
    lu = df[column].nunique(dropna= True)
    plt.title(f'{col} - {lu} vals', fontsize = 12)
    #plt.suptitle(f'{lu} Unique Values', fontsize = 10)
    plt.plot()
```


Heartdisease - 2 vals
Count
HeartDisease
Bmi - 3604 vals
Count
BMI
Smoking - 2 vals
Count
Smoking
Alcoholdrinking - 2 vals
Count
AlcoholDrinking
Stroke - 2 vals
Count
Stroke
Physicalhealth - 31 vals
Count
PhysicalHealth
Mentalhealth - 31 vals
Count
MentalHealth
Diffwalking - 2 vals
Count
DiffWalking
Sex - 2 vals
Count
Female
Male
Sex
Agecategory - 13 vals
Count
AgeCategory
Race - 6 vals
Count
White
Black
Hispanic
Race
Diabetic - 4 vals
Count
Yes
No
Diabetic
Physicalactivity - 2 vals
Count
PhysicalActivity
Genhealth - 5 vals
Count
Very good
Fair
Good
Poor
Excellent
GenHealth
Sleeptime - 24 vals
Asthma - 2 vals

Identify Variab

In [210]:



Change Yes/No to 1/0

In [211]:

```
yes_no_cols = ['Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking',
                'PhysicalActivity', 'Asthma', 'KidneyDisease', 'SkinCancer
']
```

In [212]:

```
for colz in yes_no_cols:
    df[colz] = np.where(df[colz] == "Yes", 1, 0)
```

In [213]:

```
df[target] = np.where(df[target] == "Yes", 1, 0)
```

Check for Class Imbalance

In [214]:

```
df.HeartDisease.value_counts()
```

Out[214]:

```
0    292422
1     27373
Name: HeartDisease, dtype: int64
```

In [215]:

```
no_hd = len(df.loc[df['HeartDisease'] == 0])
hd = len(df.loc[df['HeartDisease'] == 1])
```

In [216]:

```
no_hd2 = no_hd / (no_hd + hd)
no_hd2
```

Out[216]:

```
0.9144045404086993
```

In [217]:

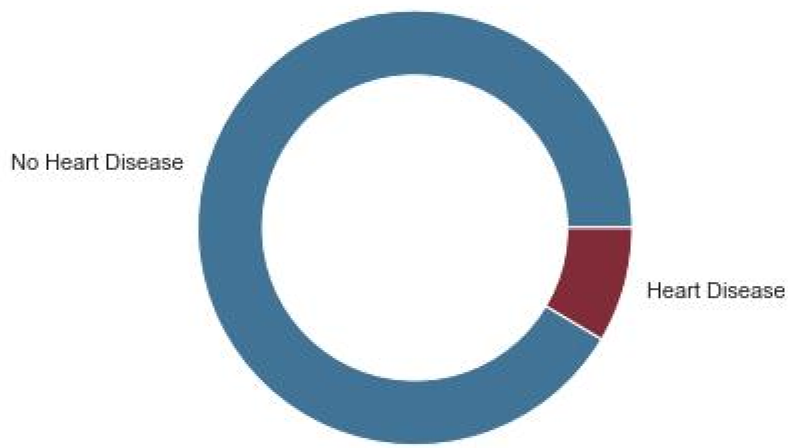
```
hd2 = hd / (hd+no_hd)
hd2
```

Out[217]:

```
0.08559545959130067
```

In [218]:
```
df.BMI
```
Out[218]:
```
0         16.60
1         20.34
2         26.58
3         24.21
4         23.71
          ...
319790    27.41
319791    29.84
319792    24.24
319793    32.81
319794    46.56
Name: BMI, Length: 319795, dtype: float64
```

In [219]:
```
names = ['No Heart Disease', 'Heart Disease']
size = [no_hd, hd]

my_circle = plt.Circle( (0,0), 0.7, color='white')
plt.pie(size, labels=names, colors=[bluez, redz])
p = plt.gcf()
p.gca().add_artist(my_circle)
```
Out[219]:
```
<matplotlib.patches.Circle at 0x2a8047cefd0>
```



Data Splitting

In [220]:
```
# Split the outcome and predictor variables
y = df['HeartDisease']
X = df.drop('HeartDisease', axis=1)
```

In [221]:
```
#Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.25, rando
m_state=120)
```

Variable Processing

A) Seperate by Data Type

In [222]:
```
X_train_cat = X_train[categorical]
X_test_cat = X_test[categorical]
X_train_cont = X_train[continuous]
X_test_cont = X_test[continuous]

cat_shape = X_train_cat.shape
cont_shape = X_train_cont.shape

print(f' categorical shape is {cat_shape}')
print(f' continuous shape is {cont_shape}')
```

```
 categorical shape is (239846, 13)
 continuous shape is (239846, 4)
```

In [223]:
```
cat_cols = list(X_train_cat.columns)
cont_cols = list(X_train_cont.columns)
```

B) Standardize Continuous Data

In [224]:
```
ss = StandardScaler()
X_train_cont_scaled = ss.fit_transform(X_train_cont)
X_test_cont_scaled = ss.transform(X_test_cont)
```

In [225]:
```
X_train_cont_df = pd.DataFrame(X_train_cont_scaled, columns = cont_cols)
X_test_cont_df = pd.DataFrame(X_test_cont_scaled, columns = cont_cols)
X_train_cont_df
```

Out[225]:

	BMI	SleepTime	PhysicalHealth	MentalHealth
0	-0.294135	0.630476	-0.171854	0.138829
1	-0.460959	-0.764522	1.839772	3.282268
2	0.300766	0.630476	-0.423308	0.390305
3	-0.388564	0.630476	-0.423308	-0.489858
4	-0.737950	-0.067023	-0.423308	1.396205
...	...	...	...	...
239841	-0.070654	4.117970	-0.423308	-0.489858
239842	-0.944119	0.630476	-0.423308	-0.489858
239843	-0.265807	-0.067023	-0.423308	-0.489858
239844	0.017480	1.327974	-0.423308	-0.489858
239845	-0.508174	0.630476	0.582505	-0.489858

239846 rows × 4 columns

C) Encode Ordinal Data (Not in this version)

In [226]:
```
#ord_encode = OrdinalEncoder(categories=[['18-24', '25-29', '30-34', '35-39
', '40-44', '45-49', '50-54', '55-59', '60-64', '65-69', '70-74', '75-79',
'80 or older'],
                                        #['0', '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' ],
                                        #['0', '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' ],
                                        #['Poor', 'Fair', 'Good', 'Very goo
d', 'Excellent']])

#X_train_ord_encoded = ord_encode.fit_transform(X_train_ord[['AgeCategory',
'PhysicalHealth', 'MentalHealth', 'GenHealth']])
#X_test_ord_encoded = ord_encode.transform(X_test_ord[['AgeCategory', 'Phys
icalHealth', 'MentalHealth', 'GenHealth']])
```

Out[226]:
```
"\nord_encode = OrdinalEncoder(categories=[['18-24', '25-29', '30-34', '35-
39', '40-44', '45-49', '50-54', '55-59', '60-64', '65-69', '70-74', '75-79
', '80 or older'],\n                                        ['0', '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' ],\n                                        ['0', '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' ],\n                                        ['Poor', 'Fair', 'Good', 'V
ery good', 'Excellent']])\n\nX_train_ord_encoded = ord_encode.fit_transform
(X_train_ord[['AgeCategory', 'PhysicalHealth', 'MentalHealth', 'GenHealth
']])\nX_test_ord_encoded = ord_encode.transform(X_test_ord[['AgeCategory',
'PhysicalHealth', 'MentalHealth', 'GenHealth']])\n"
```

In [227]:
```
#X_train_ord_df= pd.DataFrame(X_train_ord_encoded, columns = ord_cols)
#X_test_ord_df= pd.DataFrame(X_test_ord_encoded, columns = ord_cols)
#X_train_ord_df
```

Out[227]:
```
'\n#X_train_ord_df= pd.DataFrame(X_train_ord_encoded, columns = ord_cols)\n
X_test_ord_df= pd.DataFrame(X_test_ord_encoded, columns = ord_cols)\nX_trai
n_ord_df\n'
```

D) Encode Categorical Data

In [228]:
```
ohe = OneHotEncoder()
X_train_cat_encoded = ohe.fit_transform(X_train_cat)
X_test_cat_encoded = ohe.transform(X_test_cat)
```

In [229]:
```
X_train_cat_encoded
```

Out[229]:
```
<239846x46 sparse matrix of type '<class 'numpy.float64'>'
        with 3117998 stored elements in Compressed Sparse Row format>
```

Combine variable types into DataFrame

In [230]:
```
columns = ohe.get_feature_names(input_features = X_train_cat.columns)
X_train_cat_df = pd.DataFrame(X_train_cat_encoded.todense(), columns=column
s)
X_Test_cat_df = pd.DataFrame(X_test_cat_encoded.todense(), columns=columns)
X_train_cat_df
```

Out[230]:

	Smoking_0	Smoking_1	AlcoholDrinking_0	AlcoholDrinking_1	Stroke_0	Stroke_1	Di
0	1.0	0.0	1.0	0.0	1.0	0.0	
1	1.0	0.0	1.0	0.0	1.0	0.0	
2	1.0	0.0	1.0	0.0	1.0	0.0	
3	1.0	0.0	1.0	0.0	1.0	0.0	
4	1.0	0.0	1.0	0.0	1.0	0.0	
...	...	...	...	...	...	...	
239841	0.0	1.0	1.0	0.0	1.0	0.0	
239842	1.0	0.0	1.0	0.0	1.0	0.0	
239843	1.0	0.0	1.0	0.0	1.0	0.0	
239844	1.0	0.0	1.0	0.0	1.0	0.0	
239845	0.0	1.0	1.0	0.0	1.0	0.0	

239846 rows × 46 columns

In [231]:
```
X_all_train = pd.concat([X_train_cat_df, X_train_cont_df], axis = 1)
X_all_test = pd.concat([X_Test_cat_df, X_test_cont_df], axis = 1)
```

In [232]:
```
X_all_train
```

Out[232]:

	Smoking_0	Smoking_1	AlcoholDrinking_0	AlcoholDrinking_1	Stroke_0	Stroke_1	Di
0	1.0	0.0	1.0	0.0	1.0	0.0	
1	1.0	0.0	1.0	0.0	1.0	0.0	
2	1.0	0.0	1.0	0.0	1.0	0.0	
3	1.0	0.0	1.0	0.0	1.0	0.0	
4	1.0	0.0	1.0	0.0	1.0	0.0	
...	...	...	...	...	...	...	
239841	0.0	1.0	1.0	0.0	1.0	0.0	
239842	1.0	0.0	1.0	0.0	1.0	0.0	
239843	1.0	0.0	1.0	0.0	1.0	0.0	
239844	1.0	0.0	1.0	0.0	1.0	0.0	
239845	0.0	1.0	1.0	0.0	1.0	0.0	

239846 rows × 50 columns

SMOTENC

In [233]:
```
Smote_NC = SMOTENC(categorical_features=[ 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],
                                            random_state= 0)
```

Out[233]:
```
'\nSmote_NC = SMOTENC(categorical_features=[ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, \n                                        11, 12, 13, 14, 15, 1
6, 17,\n                                        18, 19, 20, 21, 22, 23,
24, 25, \n                                        26, 27, 28, 29, 30, 3
1, 32, 33, 34,\n                                        35, 36, 37, 38,
39, 40, 41, 42, 43, 44, 45, 46], \n
random_state= 0)\n'
```

In [234]:
```
print(y_train.value_counts())
# Fit SMOTE to training data
#_train_resampled, y_train_resampled = Smote_NC.fit_resample(X_all_train, y
_train)
# Preview synthetic sample class distribution
print('\n')
print(pd.Series(y_train_resampled).value_counts())
```

In [306]:
```
# One of my computers could not run this, the other could. Thus this code.

#X_train_resampled.to_csv('X_train_resample_SmoteNC_NonOrdinal.csv')
#y_train_resampled.to_csv('y_train_resample_SmoteNC_NonOrdinal.csv')
#X_train_resampled = pd.read_csv('X_train_resample_SmoteNC_NonOrdinal.csv')
#y_train_resampled = pd.read_csv('y_train_resample_SmoteNC_NonOrdinal.csv')
```

Fitting and Testing ML Models

A) Code Additions

In [236]:
```
# SOURCE: The origin of this confusion matrix code was found on medium, '
# from https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30f
ea
def make_confusion_matrix(cf,
                          group_names=None,
                          categories='auto',
                          count=True,
                          percent=True,
                          cbar=True,
                          xyticks=True,
                          xyplotlabels=True,
                          sum_stats=True,
                          figsize=None,
                          cmap='Blues',
                          title=None):

    # CODE TO GENERATE SUMMARY STATISTICS & TEXT FOR SUMMARY STATS
    if sum_stats:
        #Accuracy is sum of diagonal divided by total observations
        accuracy  = np.trace(cf) / float(np.sum(cf))

        #if it is a binary confusion matrix, show some more stats
        if len(cf)==2:
            #Metrics for Binary Confusion Matrices
            a = cf[0,0]
            b = cf[0,1]
            c = cf[1,0]
            d = cf[1,1]
            tn = ((a / (a+b))*100).round(2).astype(str) + '%'
            fp = ((b / (a+b))*100).round(2).astype(str) + '%'
            fn = ((c / (c+d))*100).round(2).astype(str) + '%'
            tp = ((d / (c+d))*100).round(2).astype(str) + '%'
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            rwf_score = 2*precision* (recall*2) /(precision + (recall*2))
            stats_text = "\n\nAccuracy={:0.3f}\nPrecision={:0.3f}\nRecall=
{:0.3f}\nF1 Score={:0.3f}\n\nRecall-Weighted F Score={:0.3f}".format(
                accuracy,precision,recall,f1_score, rwf_score)
        else:
            stats_text = "\n\nAccuracy={:0.3f}".format(accuracy)
    else:
        stats_text = ""

    # CODE TO GENERATE TEXT INSIDE EACH SQUARE
    blanks = ['' for i in range(cf.size)]

    if group_names and len(group_names)==cf.size:
        group_labels = ["{}\n".format(value) for value in group_names]
    else:
        group_labels = blanks

    if count:
        group_counts = ["{0:0.0f}\n".format(value) for value in cf.flatten
()]
```

```
    else:
        group_counts = blanks

    if percent:
        group_percentages =  [tn,fp,fn,tp]
        # old = group_percentages = ["{0:.2%}".format(value) for value in c
f.flatten()/np.sum(cf)]
    else:
        group_percentages = blanks

    box_labels = [f"{v1}{v2}{v3}".strip() for v1, v2, v3 in zip(group_label
s,group_counts,group_percentages)]
    box_labels = np.asarray(box_labels).reshape(cf.shape[0],cf.shape[1])

    # SET FIGURE PARAMETERS ACCORDING TO OTHER ARGUMENTS
    if figsize==None:
        #Get default figure size if not set
        figsize = plt.rcParams.get('figure.figsize')

    if xyticks==False:
        #Do not show categories if xyticks is False
        categories=False


    # MAKE THE HEATMAP VISUALIZATION
    plt.figure(figsize=figsize)
    sns.heatmap(cf,annot=box_labels,fmt="",cmap=cmap,cbar=cbar,xticklabels=
categories,yticklabels=categories)

    if xyplotlabels:
        plt.ylabel('True label', weight = 'bold')
        plt.xlabel('Predicted label' + stats_text, weight = 'bold')
    else:
        plt.xlabel(stats_text)

    if title:
        plt.title(title,size = 20, weight = 'bold')
```

In [237]:

```
dfcols = ['Model', 'RWF Score', 'F1',  'Recall',  'Precision',  'Accuracy']
model_summary = pd.DataFrame(columns=dfcols)
model_summary
```

Out[237]:

Model	RWF Score	F1	Recall	Precision	Accuracy

In [238]:
```
# Define Result Saving Initial Function
def save_result(cf, model_name):
            global model_summary
            accuracy  = np.trace(cf) / float(np.sum(cf))
            precision = cf[1,1] / sum(cf[:,1])
            recall    = cf[1,1] / sum(cf[1,:])
            f1_score  = 2*precision*recall / (precision + recall)
            rwf_score = 2*precision* (recall*2) /(precision + (recall*2))
            row = [(model_name, rwf_score, f1_score, recall, precision, acc
uracy)]
            res = pd.DataFrame(columns = dfcols, data = row)
            yeep = [model_summary, res]
            model_summary = pd.concat(yeep)
            model_summary = model_summary.sort_values('RWF Score', ascendin
g = False)
            model_summary = model_summary.drop_duplicates()
            return model_summary.round(3)
```

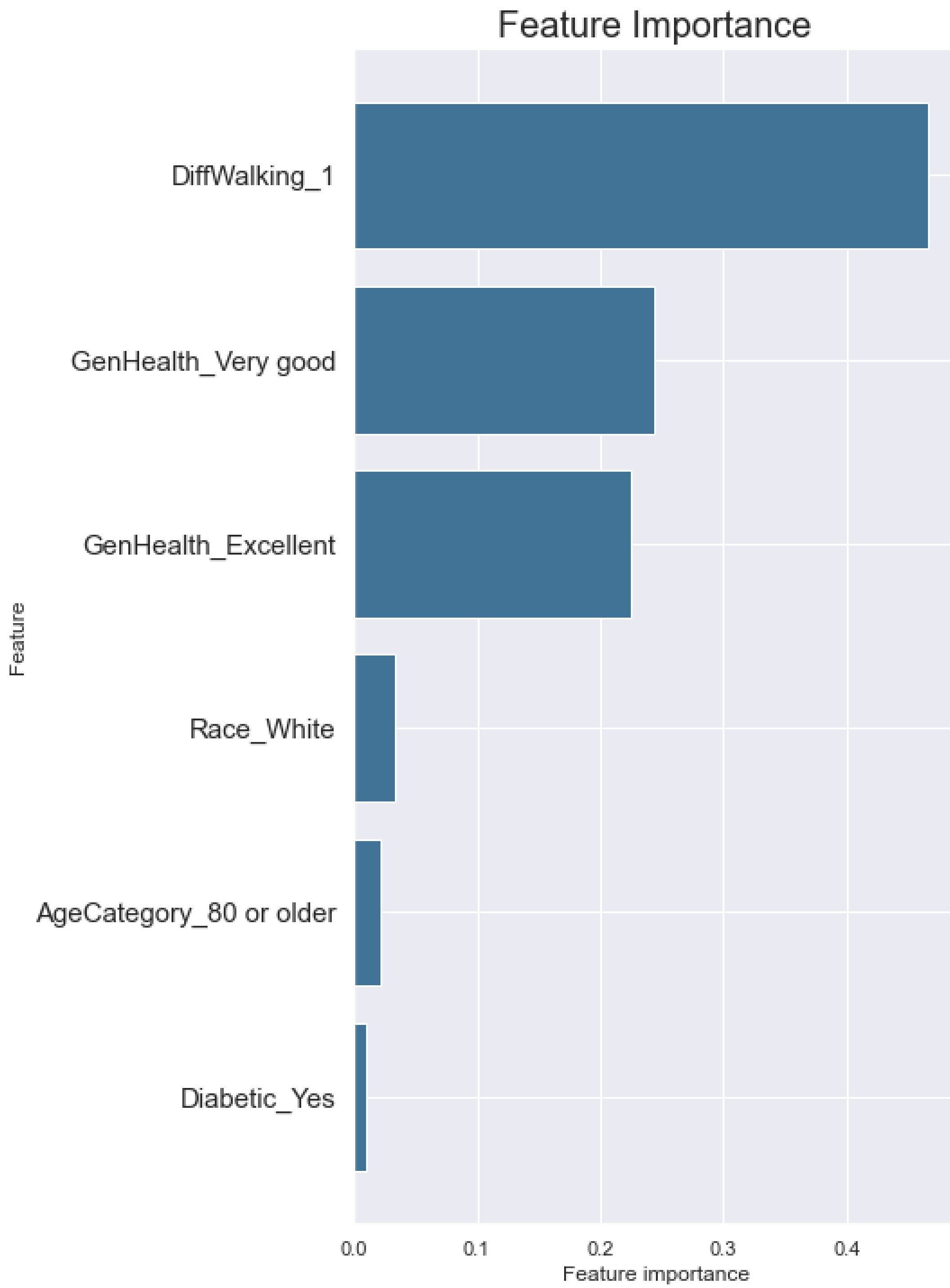
In [239]:
```
y_train_resampled = y_train_resampled['HeartDisease']
X_train_resampled = X_train_resampled.drop(columns = ['Unnamed: 0'])
```

Decision Tree -- The Initial Model

In [240]:
```
# Instantiate and fit a DecisionTreeClassifier
tree_clf = DecisionTreeClassifier(criterion='gini', max_depth=3)
tree_clf.fit(X_train_resampled, y_train_resampled)
```

Out[240]:
```
DecisionTreeClassifier(max_depth=3)
```

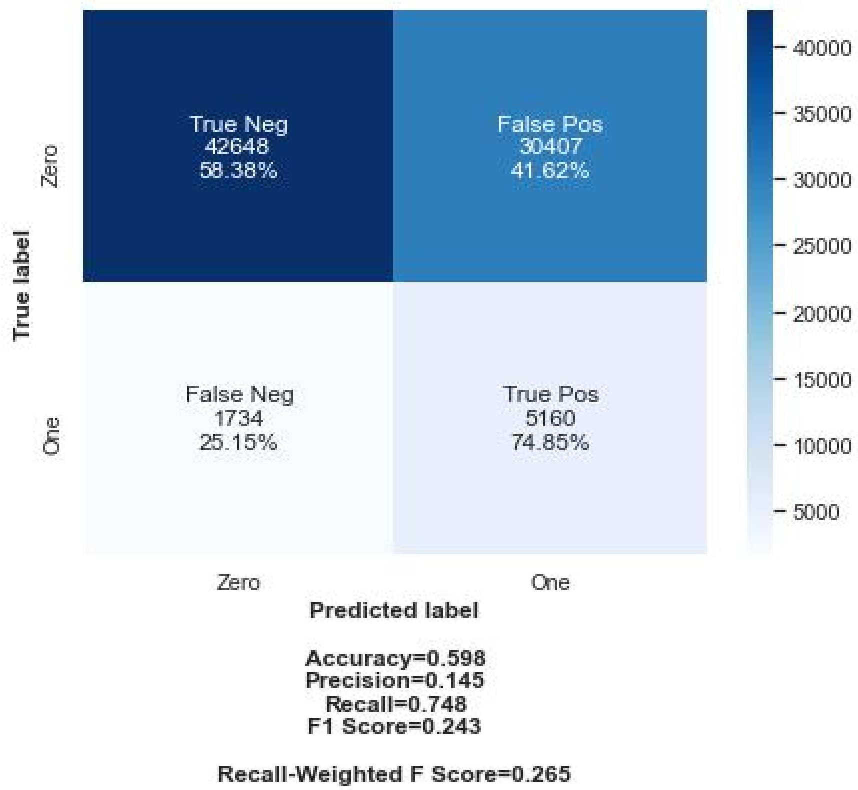
In [241]:
```
def plot_feature_importances(model):
    n_features = X_train_resampled.shape[1]
    imp_df = pd.DataFrame(model.feature_importances_)
    nm_df = pd.DataFrame(X_train_resampled.columns.values)
    imp_feats = pd.merge(nm_df, imp_df, left_index=True, right_index=True)
    imp_feats= imp_feats.round(3)
    imp_feats= imp_feats.rename(columns = {'0_x' : 'Feature', '0_y' : 'Impo
rtance' })
    imp_feats = imp_feats.loc[imp_feats['Importance'] > .005]
    imp_feats = imp_feats.sort_values('Importance', ascending = True)
    n_features = imp_feats.shape[0]
    plt.figure(figsize=(6, 12))

    plt.barh(range(n_features), imp_feats['Importance'], align='center', co
lor = bluez)
    plt.yticks(np.arange(n_features), imp_feats['Feature'].values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
    plt.yticks(size = 15 )
    plt.title('Feature Importance', fontsize = 20)

plot_feature_importances(tree_clf)
```



In [242]:
```
# Test set predictions
dec_tree_pred = tree_clf.predict(X_all_test)
cf_matrix = confusion_matrix(y_test, dec_tree_pred)
```

In [243]:

```
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



In [244]:

```
# Check for overfitting
# Predict on training and test sets
training_preds = tree_clf.predict(X_all_train)
test_preds = tree_clf.predict(X_all_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 60.0%
Validation accuracy: 59.8%
```

Fit Check

The similarity in the Training and Validation scores indicate overfitting was not an issue.

In [245]:
```
save_result(cf_matrix, 'Decision Tree - Initial Model')
```

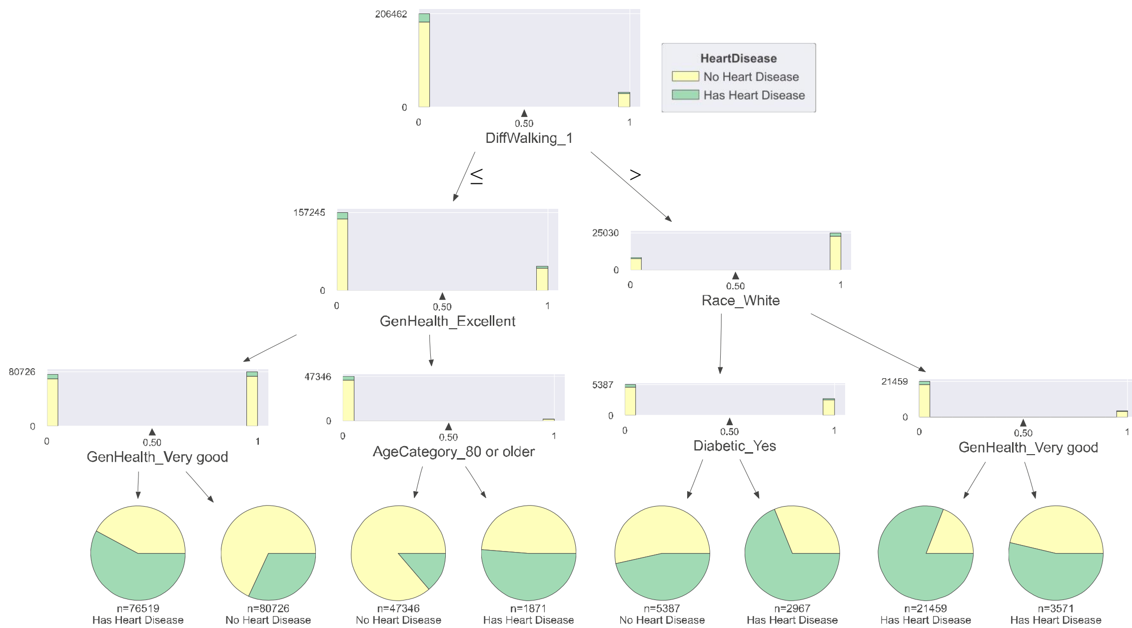
Out[245]:

	Model	RWF Score	F1	Recall	Precision	Accuracy
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

In [246]:
```
viz = dtreeviz(tree_clf, X_all_train, y, target_name='HeartDisease',
                feature_names = X_all_train.columns, class_names =['No Hear
t Disease', 'Has Heart Disease'])
```

In [247]:
```
viz
```

Out[247]:



Vanilla Model: Logistic Regression

In [248]:
```
# Initial Model
logreg_s = LogisticRegression(fit_intercept=False, solver='liblinear')
# Probability scores for test set
y_score_s = logreg_s.fit(X_train_resampled, y_train_resampled)
```

In [249]:
```
X_all_test.head(3)
```

Out[249]:

	Smoking_0	Smoking_1	AlcoholDrinking_0	AlcoholDrinking_1	Stroke_0	Stroke_1	DiffWalk
0	0.0	1.0	1.0	0.0	1.0	0.0	
1	1.0	0.0	1.0	0.0	1.0	0.0	
2	0.0	1.0	1.0	0.0	1.0	0.0	

3 rows × 50 columns

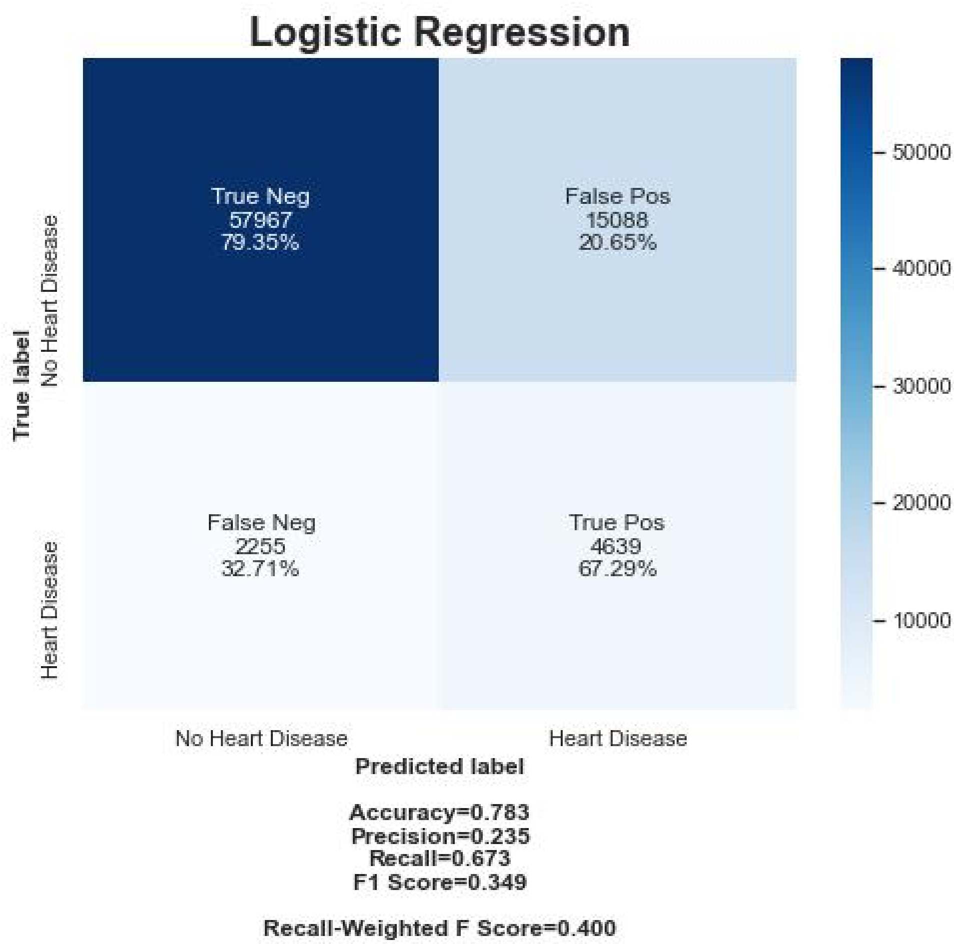
In [250]:
```
log_reg_pred = logreg_s.predict(X_all_test)
cm = classification_report(y_test,log_reg_pred)
cf_matrix = confusion_matrix(y_test, log_reg_pred)
```

In [251]:
```
save_result(cf_matrix, 'Logistic Regression')
```

Out[251]:

	Model	RWF Score	F1	Recall	Precision	Accuracy
0	Logistic Regression	0.400	0.349	0.673	0.235	0.783
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

In [252]:
```
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['No Heart Disease', 'Heart Disease']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues', title= "Logistic Regression",
                      figsize = (8,6))
```



```
In [253]: # Check for overfitting

          # Predict on training and test sets
          training_preds = logreg_s.predict(X_all_train)
          test_preds = logreg_s.predict(X_all_test)

          # Accuracy of training and test sets
          training_accuracy = accuracy_score(y_train, training_preds)
          test_accuracy = accuracy_score(y_test, test_preds)

          print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
          print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 78.51%
Validation accuracy: 78.31%
```

Fit Check

The similarity in the Training and Validation scores indicate overfitting was not an issue.

```
In [254]: y_score_s.get_params()
```

```
Out[254]: {'C': 1.0,
           'class_weight': None,
           'dual': False,
           'fit_intercept': False,
           'intercept_scaling': 1,
           'l1_ratio': None,
           'max_iter': 100,
           'multi_class': 'auto',
           'n_jobs': None,
           'penalty': 'l2',
           'random_state': None,
           'solver': 'liblinear',
           'tol': 0.0001,
           'verbose': 0,
           'warm_start': False}
```

```
In [255]: y_score_s.get_params().keys()
```

```
Out[255]: dict_keys(['C', 'class_weight', 'dual', 'fit_intercept', 'intercept_scaling
          ', 'l1_ratio', 'max_iter', 'multi_class', 'n_jobs', 'penalty', 'random_stat
          e', 'solver', 'tol', 'verbose', 'warm_start'])
```

Vanilla Model: Random Forest Model

In [256]:
```
# fit a RandomForest
forest = RandomForestClassifier(n_estimators=100, max_depth= 5)
forest.fit(X_train_resampled, y_train_resampled)
```

Out[256]:
```
RandomForestClassifier(max_depth=5)
```

In [257]:
```
random_forest_pred = forest.predict(X_all_test)
cf_matrix = confusion_matrix(y_test, random_forest_pred)

#plot Confusion Matrix
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```

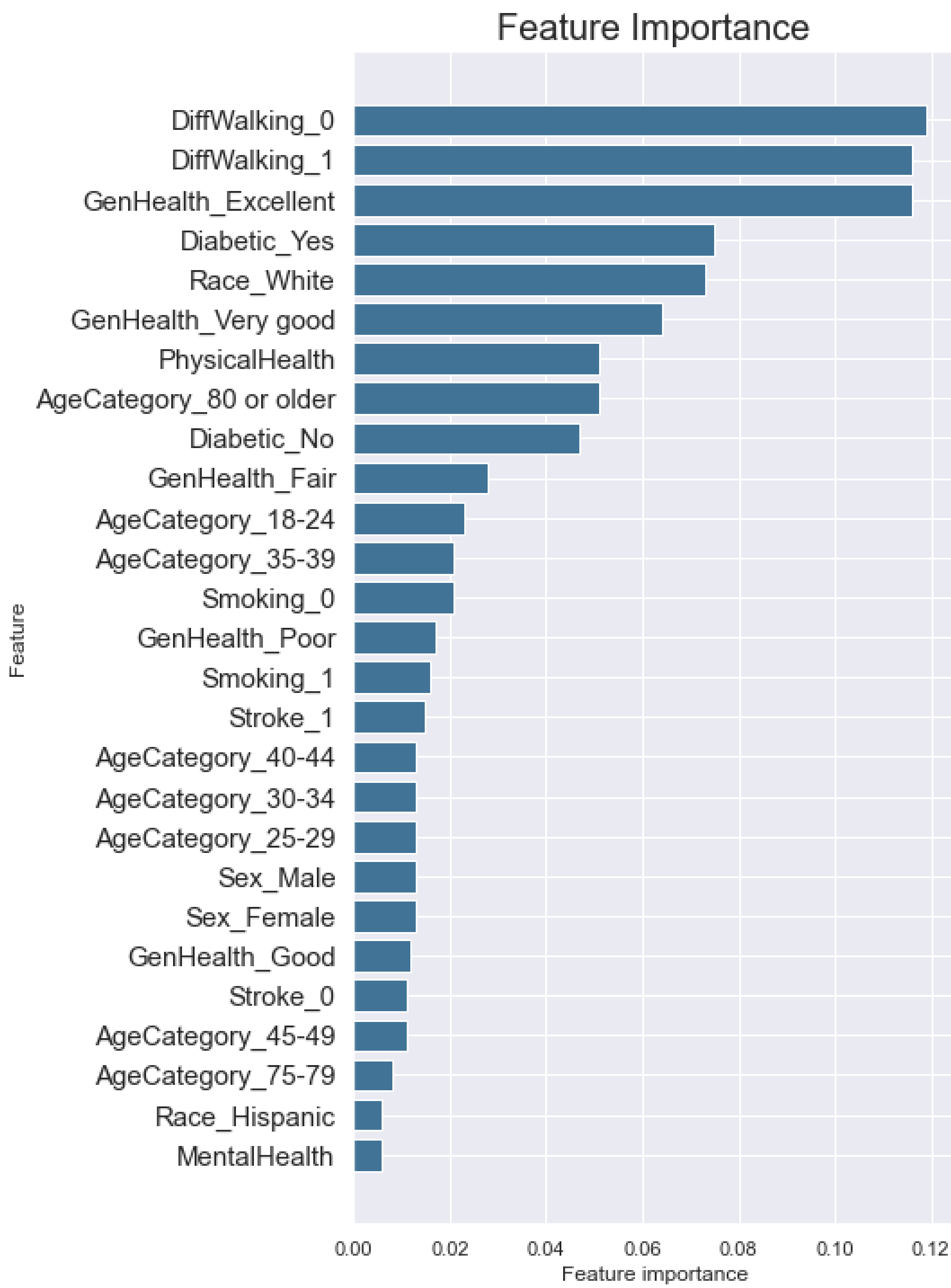
In [258]:
```
# Check for overfitting

# Predict on training and test sets
training_preds = forest.predict(X_all_train)
test_preds = forest.predict(X_all_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 76.55%
Validation accuracy: 76.43%
```

Fit Check

The similarity in the Training and Validation scores indicate overfitting was not an issue.

In [259]:
```
plot_feature_importances(forest)
```



In [260]:
```
save_result(cf_matrix, 'Random Forest')
```

Out[260]:

	Model	RWF Score	F1	Recall	Precision	Accuracy
0	Logistic Regression	0.400	0.349	0.673	0.235	0.783
0	Random Forest	0.383	0.336	0.692	0.222	0.764
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

Vanilla Model: XGBOOST Model

In [261]:
```
# Instantiate XGBClassifier
XGB = XGBClassifier()

# Fit XGBClassifier
XGB.fit(X_train_resampled, y_train_resampled)

# Predict on training and test sets
training_preds = XGB.predict(X_train_resampled)
xgboost_preds = XGB.predict(X_all_test)
```

In [262]:
```
cf_matrix = confusion_matrix(y_test, xgboost_preds)
```

In [263]:
```
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```

In [264]:
```
# Check for overfitting

# Predict on training and test sets
training_preds = XGB.predict(X_all_train)
test_preds = XGB.predict(X_all_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 84.8%
Validation accuracy: 84.11%
```

Fit Check

The similarity in the Training and Validation scores indicate overfitting was not an issue.

In [265]:
```
save_result(cf_matrix, 'XGBoost')
```

Out[265]:

	Model	RWF Score	F1	Recall	Precision	Accuracy
0	XGBoost	0.422	0.347	0.490	0.269	0.841
0	Logistic Regression	0.400	0.349	0.673	0.235	0.783
0	Random Forest	0.383	0.336	0.692	0.222	0.764
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

In [266]:
```
XGB.get_xgb_params()
```

Out[266]:
```
{'objective': 'binary:logistic',
 'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'gamma': 0,
 'gpu_id': -1,
 'interaction_constraints': '',
 'learning_rate': 0.300000012,
 'max_delta_step': 0,
 'max_depth': 6,
 'min_child_weight': 1,
 'monotone_constraints': '()',
 'n_jobs': 0,
 'num_parallel_tree': 1,
 'random_state': 0,
 'reg_alpha': 0,
 'reg_lambda': 1,
 'scale_pos_weight': 1,
 'subsample': 1,
 'tree_method': 'exact',
 'validate_parameters': 1,
 'verbosity': None}
```

Vanilla Model: Bagged Trees

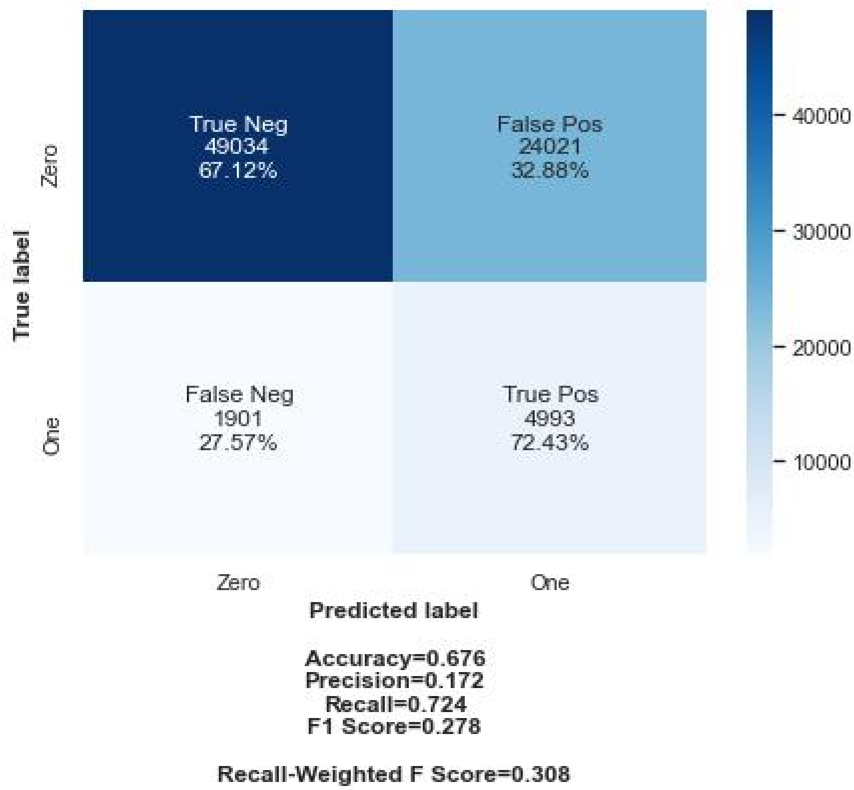
In [267]:
```
bagged_tree =  BaggingClassifier(DecisionTreeClassifier(criterion='gini', m
ax_depth=5),
                                 n_estimators=20)
```

In [268]:
```
bagged_tree.fit(X_train_resampled, y_train_resampled)
```

Out[268]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(max_depth=5),
                  n_estimators=20)
```

In [269]:
```
bagged_pred = bagged_tree.predict(X_all_test)
cf_matrix = confusion_matrix(y_test, bagged_pred)
```

In [270]:
```
#  Plot the Confusion Matrix
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



In [271]:
```
# Check for overfitting

# Predict on training and test sets
training_preds = bagged_tree.predict(X_all_train)
test_preds = bagged_tree.predict(X_all_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 67.98%
Validation accuracy: 67.58%
```

Fit Check

The similarity in the Training and Validation scores indicate overfitting was not an issue.

In [272]:
```
save_result(cf_matrix, 'Bagged Trees')
```

Out[272]:

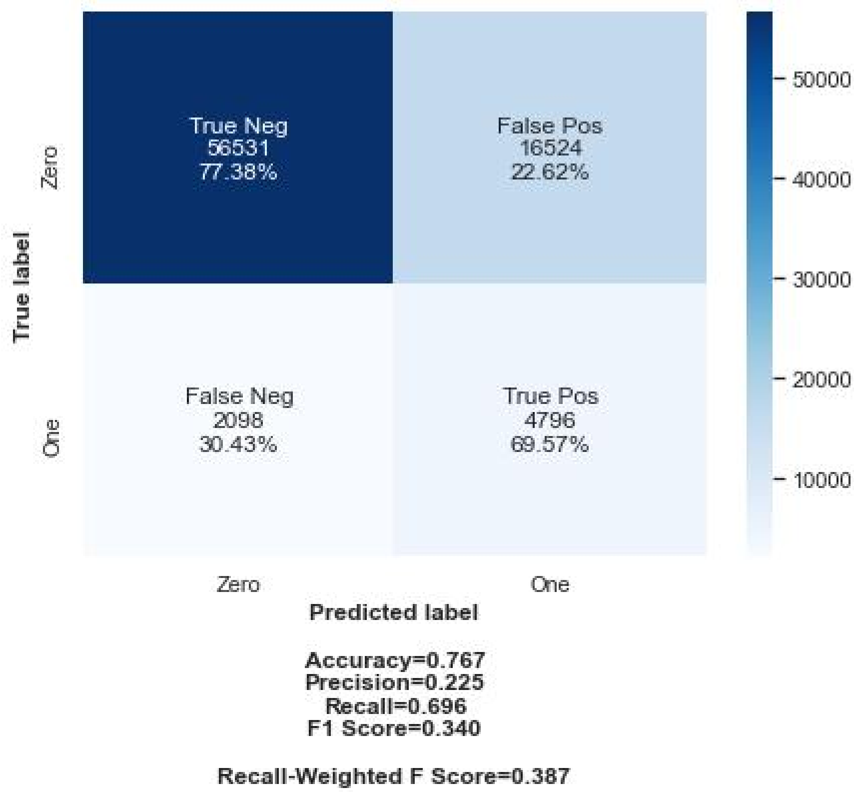
	Model	RWF Score	F1	Recall	Precision	Accuracy
0	XGBoost	0.422	0.347	0.490	0.269	0.841
0	Logistic Regression	0.400	0.349	0.673	0.235	0.783
0	Random Forest	0.383	0.336	0.692	0.222	0.764
0	Bagged Trees	0.308	0.278	0.724	0.172	0.676
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

Vanilla Model: Extra Trees

In [273]:
```
ExTrees = ExtraTreesClassifier(n_estimators=100, max_depth= 5)
ExTrees.fit(X_train_resampled, y_train_resampled)
```

Out[273]:
```
ExtraTreesClassifier(max_depth=5)
```

In [274]:
```
extrees_pred = ExTrees.predict(X_all_test)
cf_matrix = confusion_matrix(y_test, extrees_pred)

#plot Confusion Matrix
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



In [275]:
```
# Check for overfitting
# Predict on training and test sets
training_preds = ExTrees.predict(X_all_train)
test_preds = ExTrees.predict(X_all_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 76.71%
Validation accuracy: 76.71%
```

Fit Check

The similarity in the Training and Validation scores indicate overfitting was not an issue.

In [276]:
```
save_result(cf_matrix, 'Extra Trees')
```

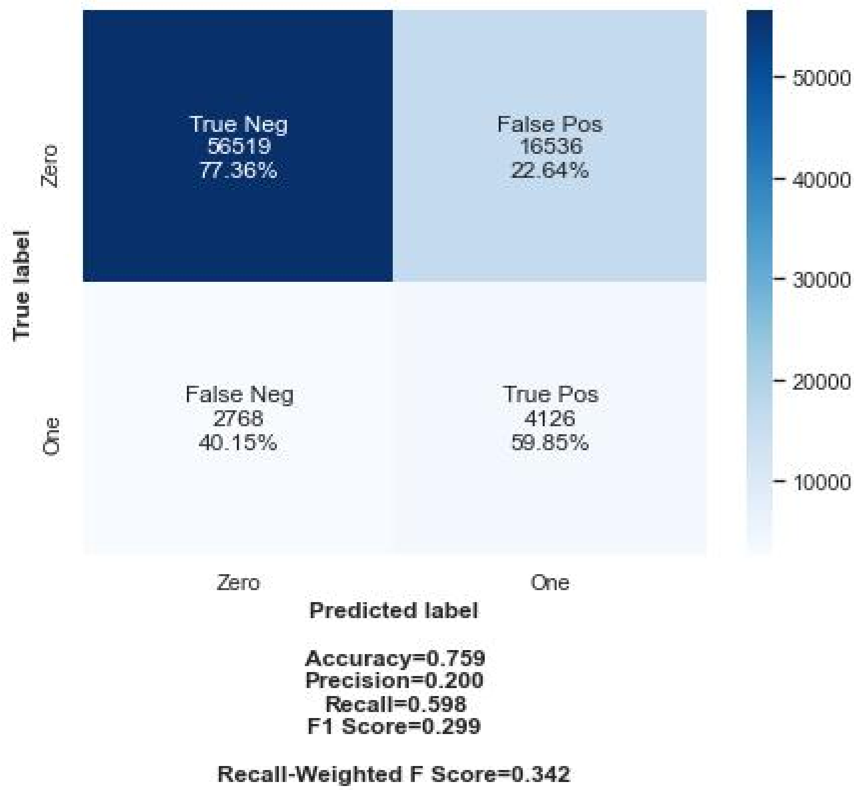
Out[276]:

	Model	RWF Score	F1	Recall	Precision	Accuracy
0	XGBoost	0.422	0.347	0.490	0.269	0.841
0	Logistic Regression	0.400	0.349	0.673	0.235	0.783
0	Extra Trees	0.387	0.340	0.696	0.225	0.767
0	Random Forest	0.383	0.336	0.692	0.222	0.764
0	Bagged Trees	0.308	0.278	0.724	0.172	0.676
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

Vanilla Model: KNN

In [277]:
```
# Running takes 9 minutes
knclf = KNeighborsClassifier()
# Fit
knclf.fit(X_train_resampled, y_train_resampled)
# Predict
knn_preds = knclf.predict(X_all_test)
training_accuracy = accuracy_score(y_train, training_preds)
```

In [278]:
```python
cf_matrix = confusion_matrix(y_test, knn_preds)
labels = ['True Neg','False Pos','False Neg','True Pos']
categories = ['Zero', 'One']
make_confusion_matrix(cf_matrix,
                      group_names=labels,
                      categories=categories,
                      cmap='Blues')
```



In [279]:
```python
# Check for overfitting
# Predict on training and test sets
training_preds = knclf.predict(X_all_train)
test_preds = knclf.predict(X_all_test)

# Accuracy of training and test sets
training_accuracy = accuracy_score(y_train, training_preds)
test_accuracy = accuracy_score(y_test, test_preds)

print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 83.7%
Validation accuracy: 75.85%
```

Vanilla Model Results

In [280]:
```
model_summary.round(3)
```

Out[280]:

	Model	RWF Score	F1	Recall	Precision	Accuracy
0	XGBoost	0.422	0.347	0.490	0.269	0.841
0	Logistic Regression	0.400	0.349	0.673	0.235	0.783
0	Extra Trees	0.387	0.340	0.696	0.225	0.767
0	Random Forest	0.383	0.336	0.692	0.222	0.764
0	Bagged Trees	0.308	0.278	0.724	0.172	0.676
0	Decision Tree - Initial Model	0.265	0.243	0.748	0.145	0.598

XGBoost Tuning

Here, I wrote my own code to take over for GridCV, because GridCV was working very slowly and sometimes not working correctly at all.

In [281]:
```
# My Tuning Code
lr = [.2,.3,.4]
md = [4,6,8]
cw = [1,2,3]
```

In [282]:
```python
search_results = pd.DataFrame()
i = 1
tot = len(lr) * len(md) * len(cw)
for l in lr:
    for m in md:
        for c in cw:
            XGB = XGBClassifier(objective= 'binary:logistic',
                        base_score= 0.5,
                        booster= 'gbtree',
                        colsample_bylevel= 1,
                        colsample_bynode= 1,
                        colsample_bytree= 1,
                        gpu_id= -1,
                        interaction_constraints= '',
                        learning_rate= l,
                        max_delta_step= 0,
                        max_depth= m,
                        min_child_weight= c,
                        gamma= 0,
                        monotone_constraints= '()',
                        n_jobs= 0,
                        num_parallel_tree= 1,
                        random_state= 0,
                        reg_alpha= 0,
                        reg_lambda= 1,
                        scale_pos_weight= 1,
                        subsample= 1,
                        tree_method= 'exact',
                        validate_parameters= 1,
                        verbosity= None)

            XGB.fit(X_train_resampled, y_train_resampled)
            training_preds = XGB.predict(X_train_resampled)
            xgboost_preds = XGB.predict(X_all_test)
            cf_matrix = confusion_matrix(y_test, xgboost_preds)
            score = my_custom_score(y_test, xgboost_preds).round(3)
                        #Create results colums
            row = [(score, l,m,c)]
            search_results = search_results.append(row)
            #print(f' lrate = {l},maxD= {m}, gam= {c}... score: {score}')
            print(f'{i} / {tot} completed')
            i += 1

search_results = search_results.sort_values(0, ascending = False)
search_results = search_results.rename(columns = {0:'rwF Score' , 1: 'Learn
ingRate' , 2:'MaxDepth' , 3: 'MinChildWeight'})
final_t = search_results['rwF Score'][0]
fin_lr = search_results['LearningRate'][0]
fin_md = search_results['MaxDepth'][0]
fin_mcw = search_results['MinChildWeight'][0]
print(f' The Best Result had a rwT-Score of {final_t}, Learning Rate of {fi
n_lr}, MaxDepth of {fin_md}, and MinChildWeight of {fin_mcw}')
search_results
```

```
1 / 27 completed
2 / 27 completed
3 / 27 completed
4 / 27 completed
5 / 27 completed
6 / 27 completed
7 / 27 completed
8 / 27 completed
9 / 27 completed
10 / 27 completed
11 / 27 completed
12 / 27 completed
13 / 27 completed
14 / 27 completed
15 / 27 completed
16 / 27 completed
17 / 27 completed
18 / 27 completed
19 / 27 completed
20 / 27 completed
21 / 27 completed
22 / 27 completed
23 / 27 completed
24 / 27 completed
25 / 27 completed
26 / 27 completed
27 / 27 completed
 The Best Result had a rwT-Score of 0    0.426
0    0.424
0    0.423
0    0.423
0    0.422
0    0.422
0    0.421
0    0.420
0    0.420
0    0.420
0    0.419
0    0.418
0    0.417
0    0.416
0    0.416
0    0.416
0    0.415
0    0.415
0    0.415
0    0.414
0    0.414
0    0.413
0    0.412
0    0.412
0    0.409
0    0.407
0    0.405
Name: rwF Score, dtype: float64, Learning Rate of 0    0.4
0    0.4
```

```
0    0.4
0    0.2
0    0.3
0    0.3
0    0.2
0    0.3
0    0.3
0    0.3
0    0.2
0    0.2
0    0.3
0    0.4
0    0.2
0    0.3
0    0.4
0    0.4
0    0.3
0    0.2
0    0.2
0    0.2
0    0.3
0    0.2
0    0.4
0    0.4
0    0.4
Name: LearningRate, dtype: float64, MaxDepth of 0    4
0    4
0    4
0    6
0    6
0    6
0    6
0    4
0    4
0    4
0    6
0    4
0    6
0    6
0    4
0    8
0    6
0    6
0    8
0    4
0    8
0    8
0    8
0    8
0    8
0    8
0    8
Name: MaxDepth, dtype: int64, and MinChildWeight of 0    2
0    1
0    3
0    1
```

```
0    3
0    1
0    3
0    3
0    2
0    1
0    2
0    3
0    2
0    2
0    2
0    2
0    1
0    3
0    1
0    1
0    3
0    2
0    3
0    1
0    2
0    1
0    3
Name: MinChildWeight, dtype: int64
```

Out[282]:

	rwF Score	LearningRate	MaxDepth	MinChildWeight
0	0.426	0.4	4	2
0	0.424	0.4	4	1
0	0.423	0.4	4	3
0	0.423	0.2	6	1
0	0.422	0.3	6	3
0	0.422	0.3	6	1
0	0.421	0.2	6	3
0	0.420	0.3	4	3
0	0.420	0.3	4	2
0	0.420	0.3	4	1
0	0.419	0.2	6	2
0	0.418	0.2	4	3
0	0.417	0.3	6	2
0	0.416	0.4	6	2
0	0.416	0.2	4	2
0	0.416	0.3	8	2
0	0.415	0.4	6	1
0	0.415	0.4	6	3
0	0.415	0.3	8	1

	rwF Score	LearningRate	MaxDepth	MinChildWeight
0	0.414	0.2	4	1
0	0.414	0.2	8	3
0	0.413	0.2	8	2
0	0.412	0.3	8	3
0	0.412	0.2	8	1
0	0.409	0.4	8	2
0	0.407	0.4	8	1

Final Model Interpretation

In [283]:

```
#Final model
final_model = XGBClassifier(objective= 'binary:logistic',
                    base_score= 0.5,
                    booster= 'gbtree',
                    colsample_bylevel= 1,
                    colsample_bynode= 1,
                    colsample_bytree= 1,
                    gpu_id= -1,
                    interaction_constraints= '',
                    learning_rate= .4,
                    max_delta_step= 0,
                    max_depth= 4,
                    min_child_weight= 2,
                    gamma= 0,
                    monotone_constraints= '()',
                    n_jobs= 0,
                    num_parallel_tree= 1,
                    random_state= 0,
                    reg_alpha= 0,
                    reg_lambda= 1,
                    scale_pos_weight= 1,
                    subsample= 1,
                    tree_method= 'exact',
                    validate_parameters= 1,
                    verbosity= None)
```

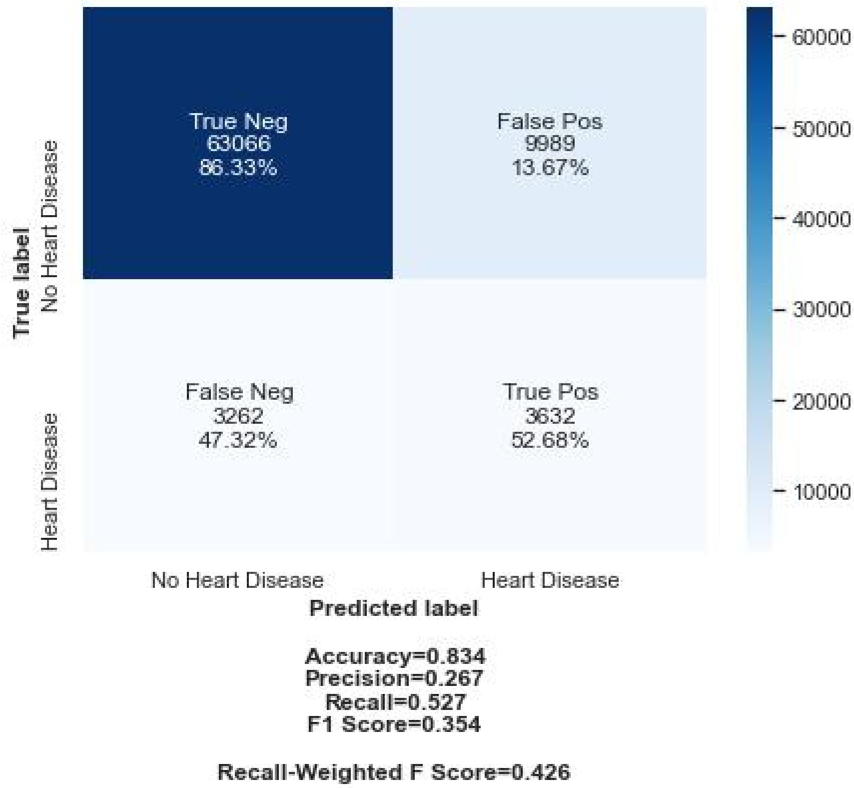
In [284]:

```
# Fit XGBClassifier
final_model.fit(X_train_resampled, y_train_resampled)

# Predict on training and test sets
training_preds = final_model.predict(X_train_resampled)
xgboost_preds = final_model.predict(X_all_test)
```

In [285]:

```
cf_matrix = confusion_matrix(y_test, xgboost_preds)
```

```
In [286]: labels = ['True Neg','False Pos','False Neg','True Pos']
          categories = ['No Heart Disease', 'Heart Disease']
          make_confusion_matrix(cf_matrix,
                                group_names=labels,
                                categories=categories,
                                cmap='Blues')
```



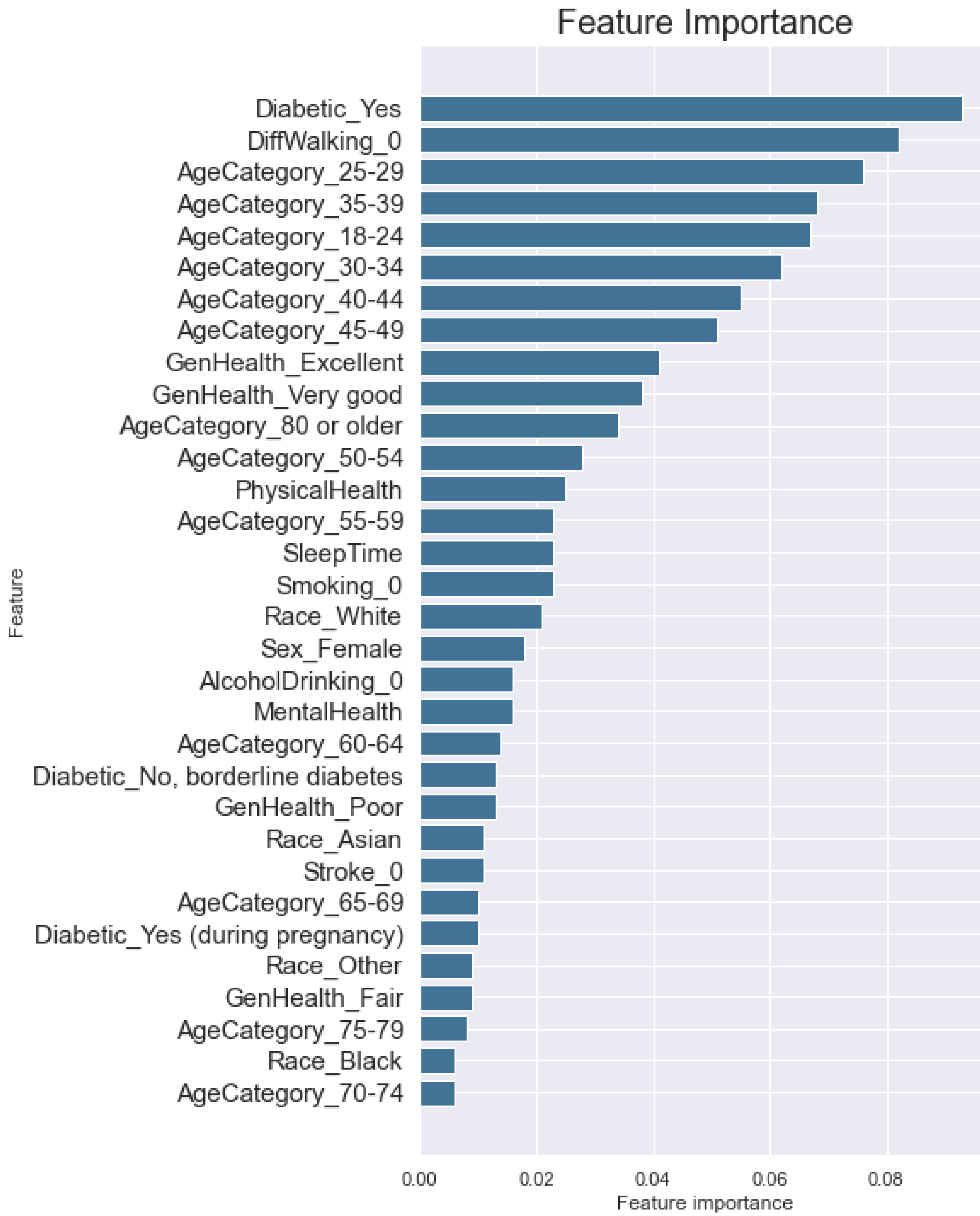

```
In [287]: # Check for overfitting

          # Predict on training and test sets
          training_preds = XGB.predict(X_all_train)
          test_preds = XGB.predict(X_all_test)

          # Accuracy of training and test sets
          training_accuracy = accuracy_score(y_train, training_preds)
          test_accuracy = accuracy_score(y_test, test_preds)

          print('Training Accuracy: {:.4}%'.format(training_accuracy * 100))
          print('Validation accuracy: {:.4}%'.format(test_accuracy * 100))
```

```
Training Accuracy: 87.38%
Validation accuracy: 85.43%
```

Fit Check

While this model's training and validation accuracy difference is larger than other models, it is still well within the range of acceptable fit.

In [288]:
```
# add importance chart
plot_feature_importances(final_model)
```



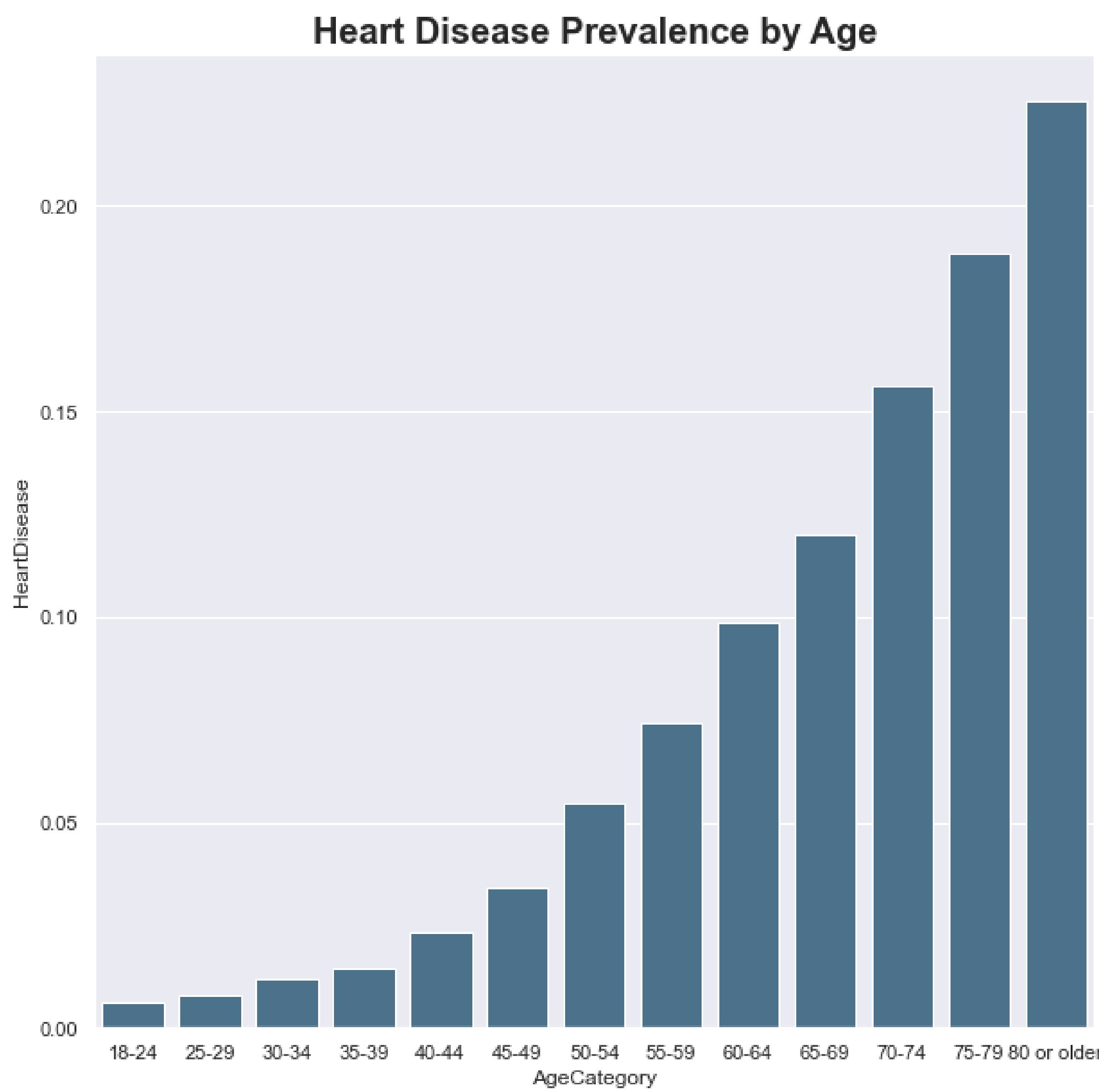
In [289]:
```
# Percent by Age Category
#plt.figure(figsize = (14,14))
# groupby age category
grp = df.groupby('AgeCategory')['HeartDisease'].mean()
grp
```

Out[289]:
```
AgeCategory
18-24          0.006172
25-29          0.007844
30-34          0.012051
35-39          0.014404
40-44          0.023136
45-49          0.034143
50-54          0.054487
55-59          0.073999
60-64          0.098765
65-69          0.120084
70-74          0.156028
75-79          0.188483
80 or older    0.225603
Name: HeartDisease, dtype: float64
```

In [290]:
```
grp = pd.DataFrame(grp)
grp = grp.reset_index()
grp = grp.sort_values('AgeCategory')
```

In [291]:
```
plt.figure(figsize = (10,10))
ax = sns.barplot(x = 'AgeCategory', y = 'HeartDisease', data = grp, color = 
bluez)
plt.title('Heart Disease Prevalence by Age', fontsize = 20, weight = 'bold
')
```

Out[291]: Text(0.5, 1.0, 'Heart Disease Prevalence by Age')

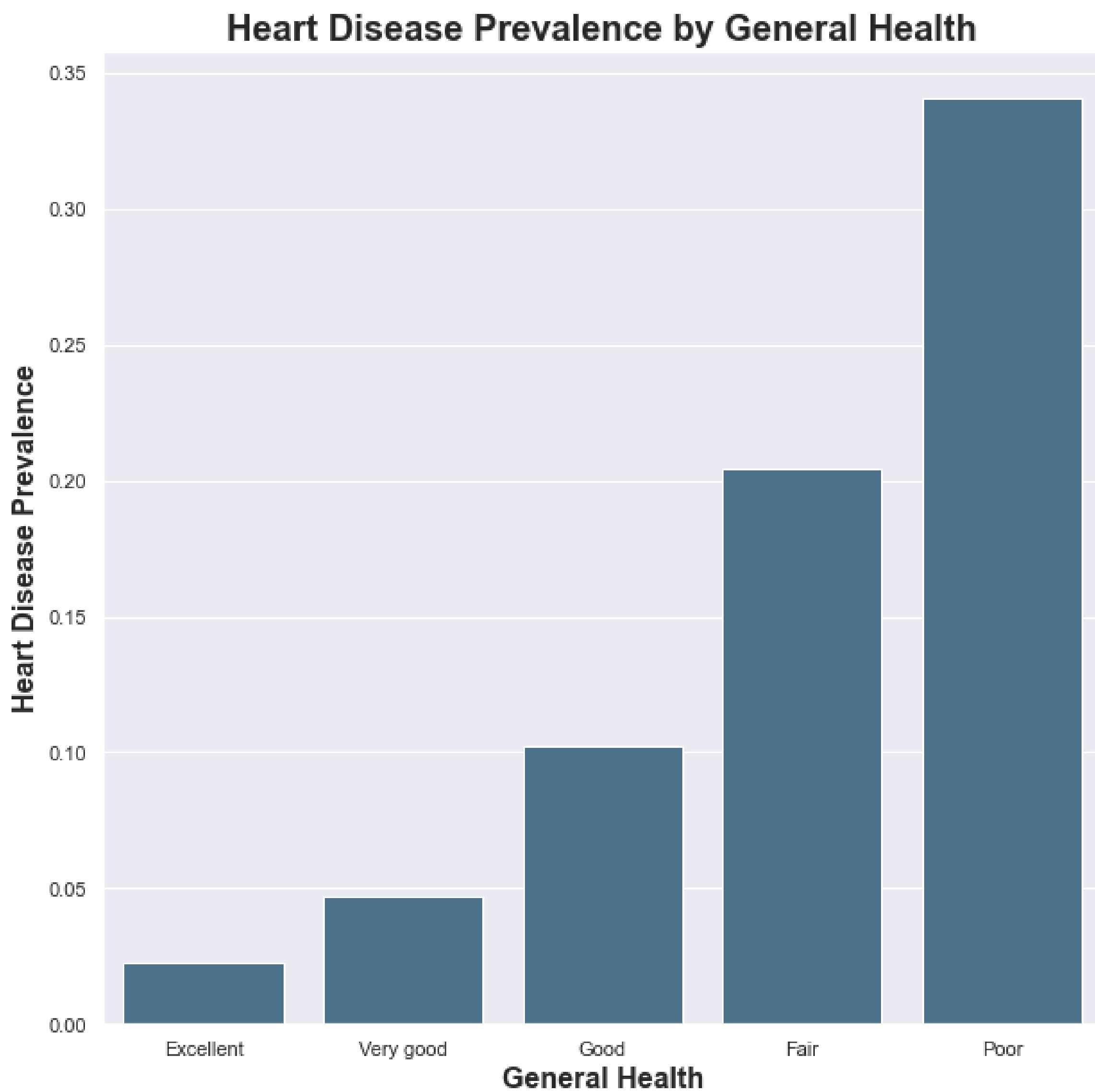


In [292]:
```
grp2 = df.groupby('GenHealth')['HeartDisease'].mean()
grp2 = grp2.reset_index()
grp2= grp2.sort_values('HeartDisease')
```

In [293]:
```
grp2['n'] = 1
```

In [294]:
```
plt.figure(figsize = (10,10))
ax = sns.barplot(x = 'GenHealth', y = 'HeartDisease', data = grp2, color = 
bluez)
plt.title('Heart Disease Prevalence by General Health', fontsize = 20, weig
ht = 'bold')
plt.xlabel('General Health', fontsize = 16, weight = 'bold')
plt.ylabel('Heart Disease Prevalence', fontsize = 16, weight = 'bold')
```

Out[294]:
```
Text(0, 0.5, 'Heart Disease Prevalence')
```

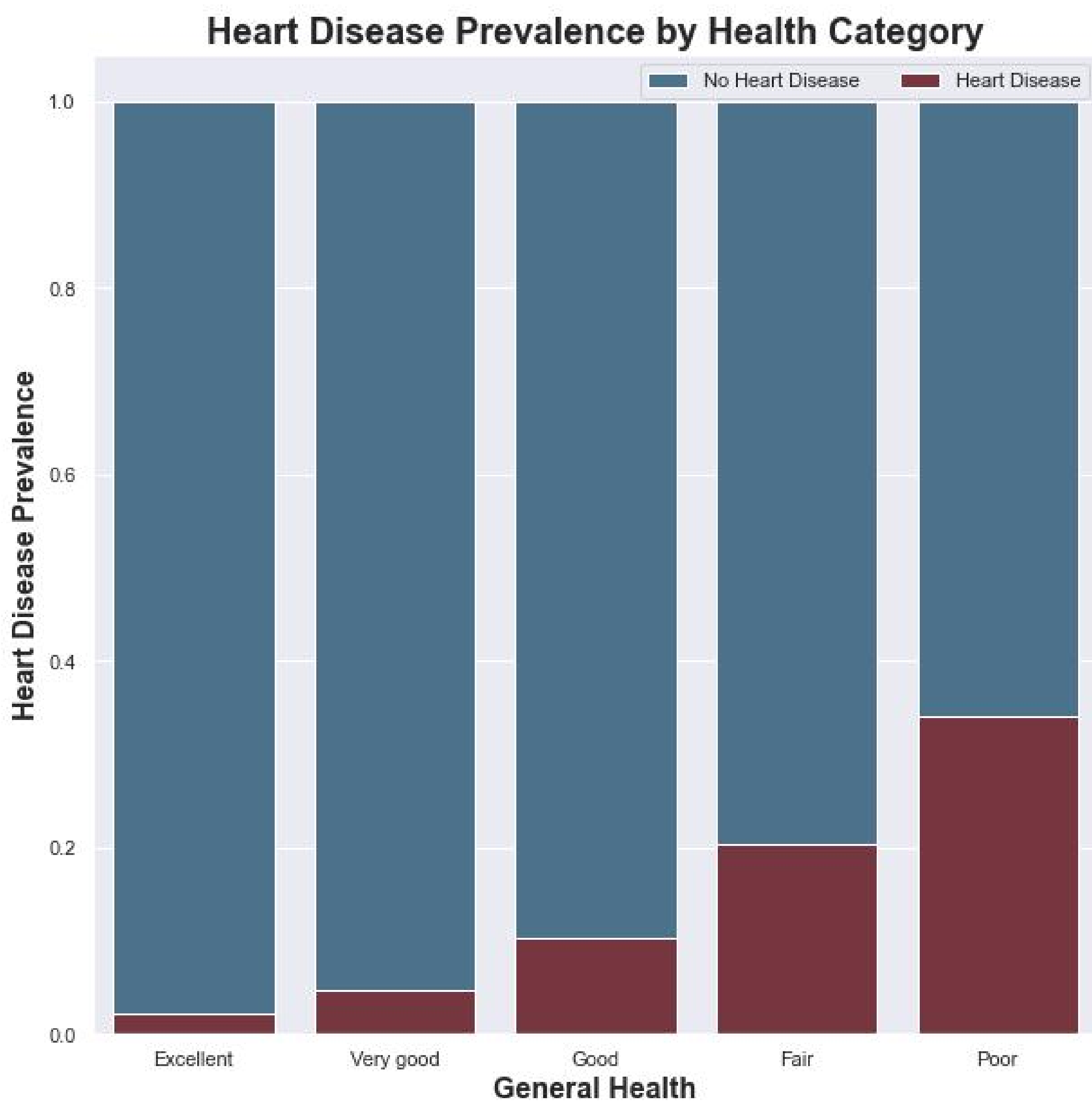


In [295]:

```
f, ax = plt.subplots(figsize=(10, 10))

# bar 1- top bars (no heart disease)
sns.barplot(x='GenHealth',  y='n', data=grp2, color=bluez, label = 'No Hear
t Disease')
sns.barplot(x = 'GenHealth', y = 'HeartDisease', data = grp2, color = redz,
label = 'Heart Disease')
#ax = sns.barplot(x = 'Diabetic', y = 'HeartDisease', data = grp3, color =
bluez)
ax.legend(ncol=2, loc="upper right", frameon=True)
plt.title('Heart Disease Prevalence by Health Category', fontsize = 20, wei
ght = 'bold')
plt.xlabel('General Health', fontsize = 16, weight = 'bold')
plt.ylabel('Heart Disease Prevalence', fontsize = 16, weight = 'bold')
```

Out[295]:

```
Text(0, 0.5, 'Heart Disease Prevalence')
```



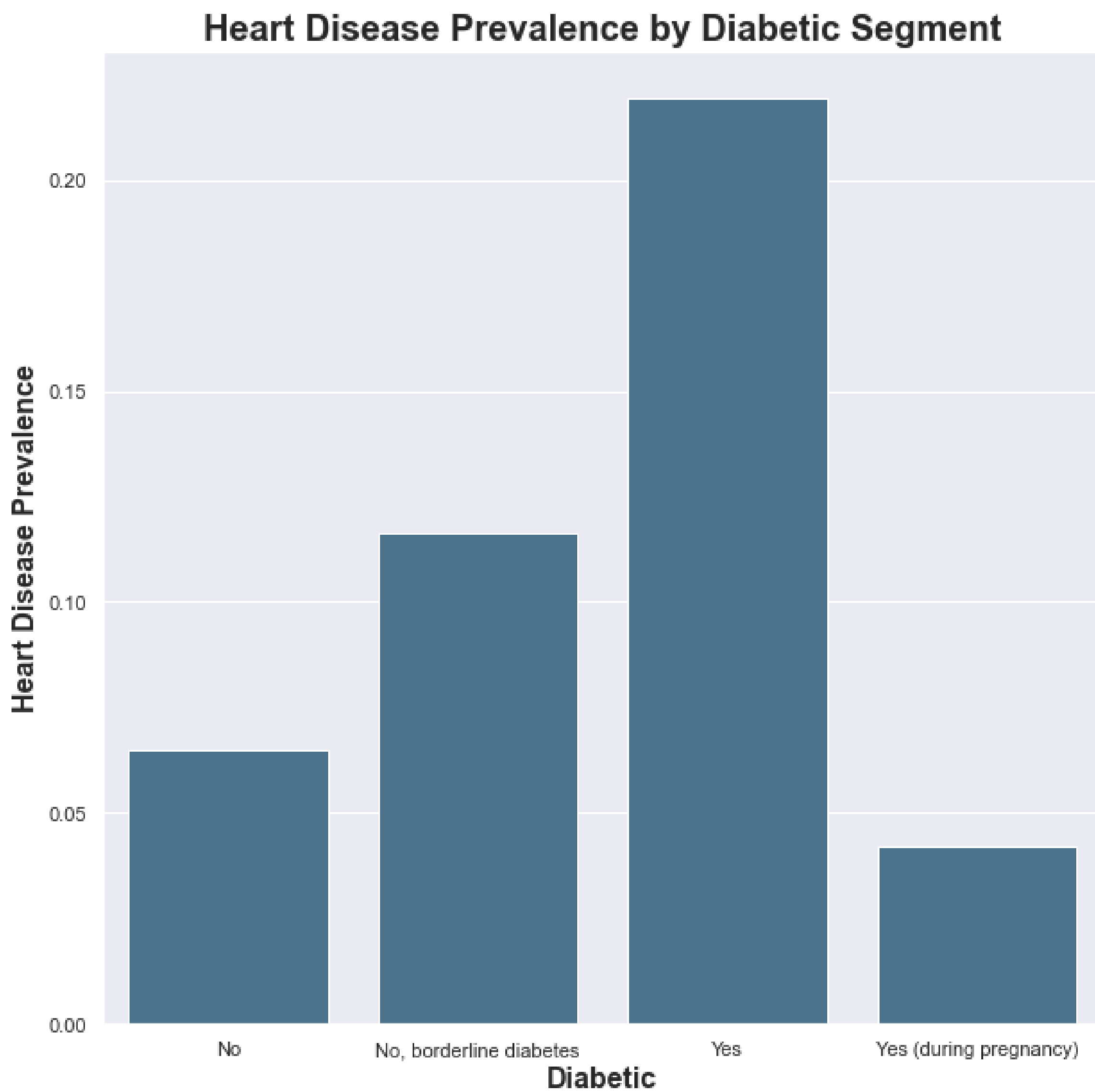
In [296]:
```
grp3 = df.groupby('Diabetic')['HeartDisease'].mean()
grp3 = grp3.reset_index()
grp3
```

Out[296]:

	Diabetic	HeartDisease
0	No	0.064969
1	No, borderline diabetes	0.116355
2	Yes	0.219524
3	Yes (during pregnancy)	0.042204

In [297]:
```
plt.figure(figsize = (10,10))
ax = sns.barplot(x = 'Diabetic', y = 'HeartDisease', data = grp3, color = b
luez)
plt.title('Heart Disease Prevalence by Diabetic Segment', fontsize = 20, we
ight = 'bold')
plt.xlabel('Diabetic', fontsize = 16, weight = 'bold')
plt.ylabel('Heart Disease Prevalence', fontsize = 16, weight = 'bold')
```

Out[297]:
```
Text(0, 0.5, 'Heart Disease Prevalence')
```



In [298]:
```
total = grp3.groupby('Diabetic')['HeartDisease'].sum().reset_index()
total['no_heart_disease'] = 1- total['HeartDisease']
total
```

Out[298]:

	Diabetic	HeartDisease	no_heart_disease
0	No	0.064969	0.935031
1	No, borderline diabetes	0.116355	0.883645
2	Yes	0.219524	0.780476
3	Yes (during pregnancy)	0.042204	0.957796

In [299]:
```
total['HeartDisease'] = total['HeartDisease'] * 100
total['no_heart_disease'] = total['no_heart_disease'] * 100
```

In [300]:
```
total['n'] = 100
total
```

Out[300]:

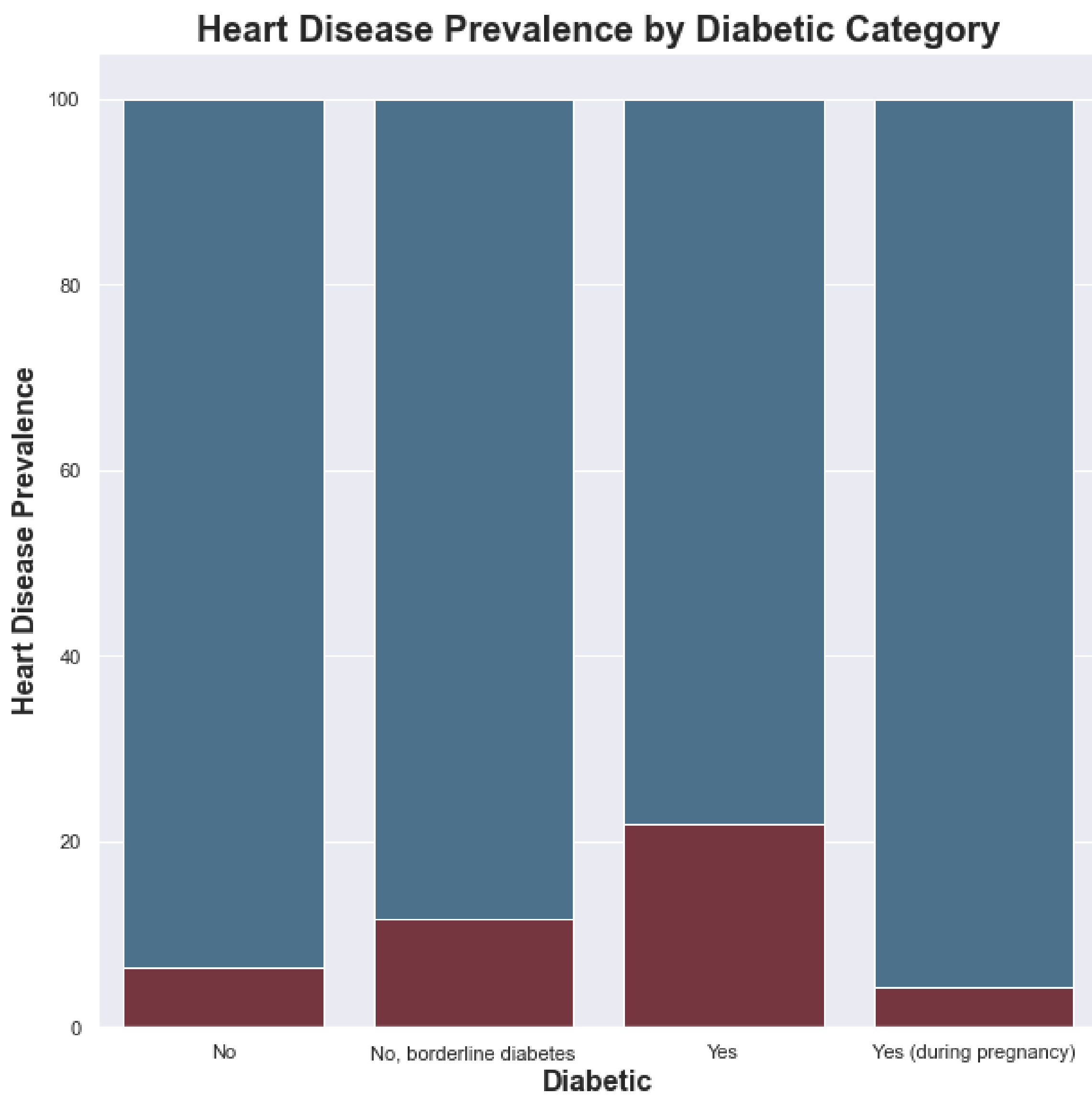
	Diabetic	HeartDisease	no_heart_disease	n
0	No	6.496868	93.503132	100
1	No, borderline diabetes	11.635452	88.364548	100
2	Yes	21.952355	78.047645	100
3	Yes (during pregnancy)	4.220399	95.779601	100

In [301]:

```
plt.figure(figsize = (10,10))

# bar 1- top bars (no heart disease)
bar1 = sns.barplot(x='Diabetic',  y='n', data=total, color=bluez)
bar2 = sns.barplot(x='Diabetic',  y='HeartDisease', data=total, color=redz)
#ax = sns.barplot(x = 'Diabetic', y = 'HeartDisease', data = grp3, color =
bluez)
plt.title('Heart Disease Prevalence by Diabetic Category', fontsize = 20, w
eight = 'bold')
plt.xlabel('Diabetic', fontsize = 16, weight = 'bold')
plt.ylabel('Heart Disease Prevalence', fontsize = 16, weight = 'bold')
```

Out[301]:

```
Text(0, 0.5, 'Heart Disease Prevalence')
```



In [302]:
```
grp4 = df.groupby('DiffWalking')['HeartDisease'].mean()
grp4 = grp4.reset_index()
grp4
```

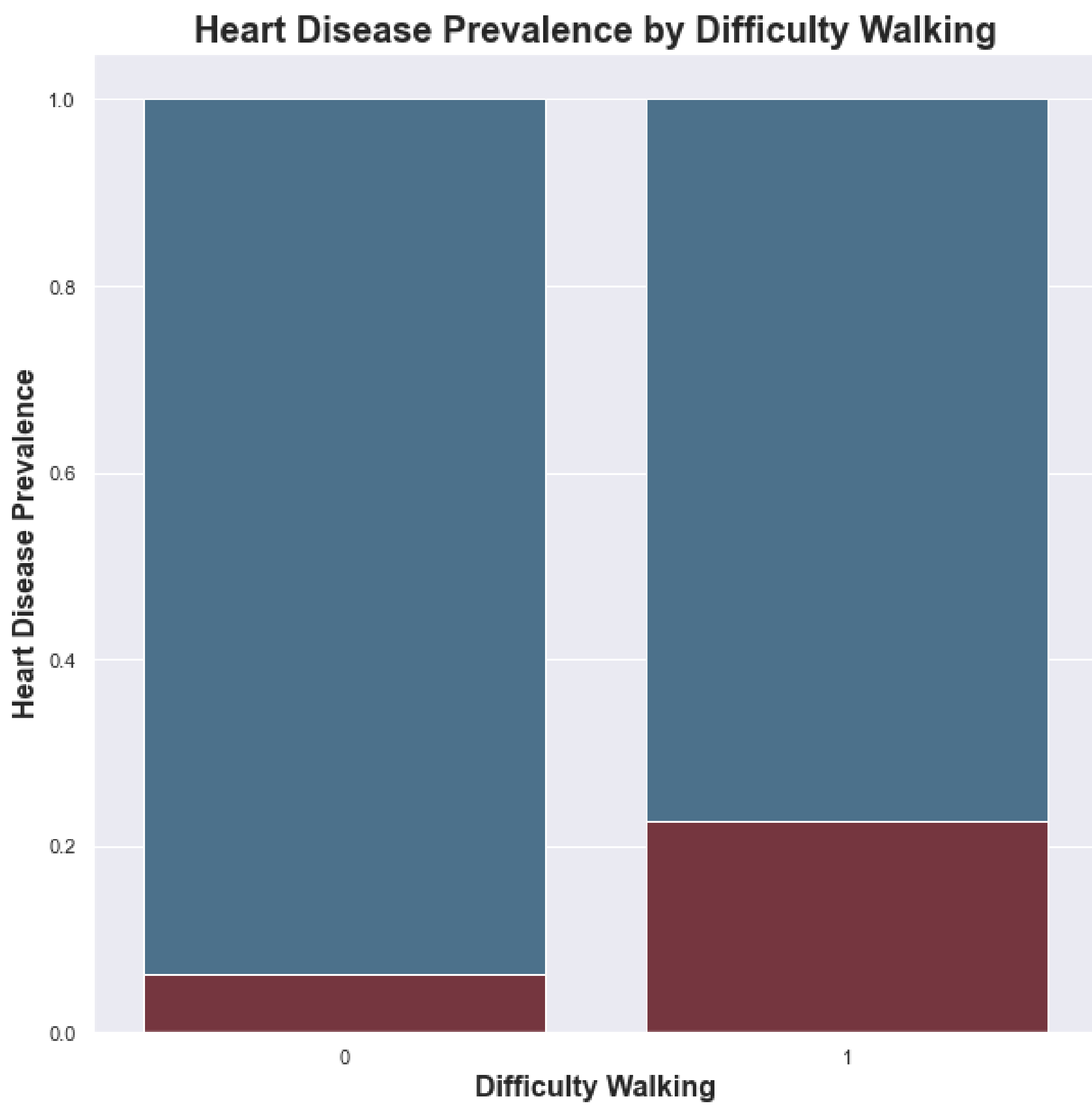
Out[302]:

	DiffWalking	HeartDisease
0	0	0.062985
1	1	0.225805

In [303]:
```
grp4['n'] = 1
```

In [304]:
```
plt.figure(figsize = (10,10))

# bar 1- top bars (no heart disease)
bar1 = sns.barplot(x='DiffWalking',  y='n', data=grp4, color=bluez)
bar2 = sns.barplot(x='DiffWalking',  y='HeartDisease', data=grp4, color=red
z)
#ax = sns.barplot(x = 'Diabetic', y = 'HeartDisease', data = grp3, color =
bluez)
plt.title('Heart Disease Prevalence by Difficulty Walking', fontsize = 20,
weight = 'bold')
plt.xlabel('Difficulty Walking', fontsize = 16, weight = 'bold')
plt.ylabel('Heart Disease Prevalence', fontsize = 16, weight = 'bold')
```

Out[304]:
```
Text(0, 0.5, 'Heart Disease Prevalence')
```



Conclusion

The Final Model is an optimized XGBoost model.

The model's most important features, by far, were:

1. the presense of diabetes,
2. difficulty walking,
3. age category, and
4. general health

Looking into these features, we find that the presence of diabetes increases the prevalance of diabetes by almost twenty percent.

Dificulty walking increases the presence of diatbetes by a very similar level, coming in a bit under twenty percent.

The difference between someone

A response of "poor" to general health caategory made an individual 30 percent more likely to have heart disease than one who responded "Excellent" or "Very good". It is interesting to note almost 10 percent of individuals who indicated they had "Good" general health had heart disease.

In the future, I would like a larger set of variables, with more specific questions. While I know that the point of this dataset is to find what general questions can lead to specific results, it would be helpful to have more than 17 variables. It would also be interesting to see this same project done with more specific data, possibly medical data, to see what variables we need to achieve higher scores all around.

The final product can predict if an individual has heart disease based on the answers to a few simple questions with an 86 percent accuracy.

The algorithm is optimized to penalize false negatives more than false positives, because the goal is to maximize finding sick individuals.

As mentioned, this product can be in the form of a web application, phone application, or both. Further, it can be used to inform individuals and/or their doctors about their heart disease risks.

The initial model achieved a recall-weighted F score (rwF Score) of .265. The final model achieved a rwF Score of .426, an improvement of over 60%.

Other metric improvements are as follows:

|Initial Model | Final Model |