Logistic regression using a dataset from keggle to predict 10 year risk of coronary heart disease in the form of yes or no.

This analysis will be completed using python which will allow for supervised machine learning with sklearn.

Attributes

Demographic

Sex: male or female Age: Age of the patient

Education: no information provided

Behavioral

Current Smoker: whether or not the patient is a current smoker Cigs Per Day: the number of cigarettes that the person smoked on average in one day

Information on medical history

BP Meds: whether or not the patient was on blood pressure medication Prevalent Stroke: whether or not the patient had previously had a stroke Prevalent Hyp: whether or not the patient was hypertensive Diabetes: whether or not the patient had diabetes

Information on current medical condition

Tot Chol: total cholesterol level Sys BP: systolic blood pressure Dia BP: diastolic blood pressure BMI: Body Mass Index Heart Rate: heart rate - In medical research, variables such as heart rate though in fact discrete, yet are considered continuous because of large number of possible values. Glucose: glucose level

Target variable to predict

TenYearCHD: 10 year risk of coronary heart disease (binary: “1:Yes”, “0:No”)

In [1]:
```
#import libraries and packages for log
import pandas as pd
from pandas import Series
from pandas import DataFrame

import numpy as np
from numpy import ndarray
from numpy.random import randn
from numpy import loadtxt, where

import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib.patches import Patch
from matplotlib.lines import Line2D
```

```
plt.rc("font", size=14)
get_ipython().run_line_magic('matplotlib', 'inline')

import seaborn as sns

import random

import scipy

from tkinter import *

from scipy import stats

from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
import statsmodels.formula.api as smf

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn import preprocessing
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVR
from sklearn.metrics import log_loss, roc_auc_score, recall_score, precision_score, av
from sklearn.datasets import load_digits
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from mlxtend.classifier import StackingCVClassifier

import os

import pylab
from pylab import scatter, show, legend, xlabel, ylabel

from collections import Counter
import pandas_profiling as pp

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

In [2]:
```
#read csv into Pandas dataframe
df = pd.read_csv('logistic_regression_heart.csv')
```

In [3]:
```
#view data layout
df.head()
```

Out[3]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabe
0	1	39	4.0	0	0.0	0.0	0	0	
1	0	46	2.0	0	0.0	0.0	0	0	
2	1	48	1.0	1	20.0	0.0	0	0	
3	0	61	3.0	1	30.0	0.0	0	1	
4	0	46	3.0	1	23.0	0.0	0	0	

Data Processing

In [4]:
```
#obtain index and datatypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4238 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   male             4238 non-null   int64  
 1   age              4238 non-null   int64  
 2   education        4133 non-null   float64
 3   currentSmoker    4238 non-null   int64  
 4   cigsPerDay       4209 non-null   float64
 5   BPMeds           4185 non-null   float64
 6   prevalentStroke  4238 non-null   int64  
 7   prevalentHyp     4238 non-null   int64  
 8   diabetes         4238 non-null   int64  
 9   totChol          4188 non-null   float64
 10  sysBP            4238 non-null   float64
 11  diaBP            4238 non-null   float64
 12  BMI              4219 non-null   float64
 13  heartRate        4237 non-null   float64
 14  glucose          3850 non-null   float64
 15  TenYearCHD       4238 non-null   int64  
dtypes: float64(9), int64(7)
memory usage: 529.9 KB
```

In [5]:
```
#number of columns and rows
df.shape
```

Out[5]:
```
(4238, 16)
```

In [6]:
```
#check data for null values
df.isnull().sum()
```

Out[6]:
```
male                0
age                 0
education         105
currentSmoker       0
cigsPerDay         29
BPMeds             53
prevalentStroke     0
prevalentHyp        0
diabetes            0
totChol            50
sysBP               0
diaBP               0
BMI                19
heartRate           1
glucose           388
TenYearCHD          0
dtype: int64
```

In [7]:
```
#impute null values with mean
df.fillna(df.mean(), inplace=True)
```

In [8]:
```
#confirm imputation worked- expect to see 0 null values
df.isnull().sum()
```

Out[8]:
```
male              0
age               0
education         0
currentSmoker     0
cigsPerDay        0
BPMeds            0
prevalentStroke   0
prevalentHyp      0
diabetes          0
totChol           0
sysBP             0
diaBP             0
BMI               0
heartRate         0
glucose           0
TenYearCHD        0
dtype: int64
```

In [9]:
```
#check data for duplicates
df.duplicated().sum()
```

Out[9]:
```
0
```

Data Visualization

In [10]:
```
#obtain summary statistics
df.describe()
```

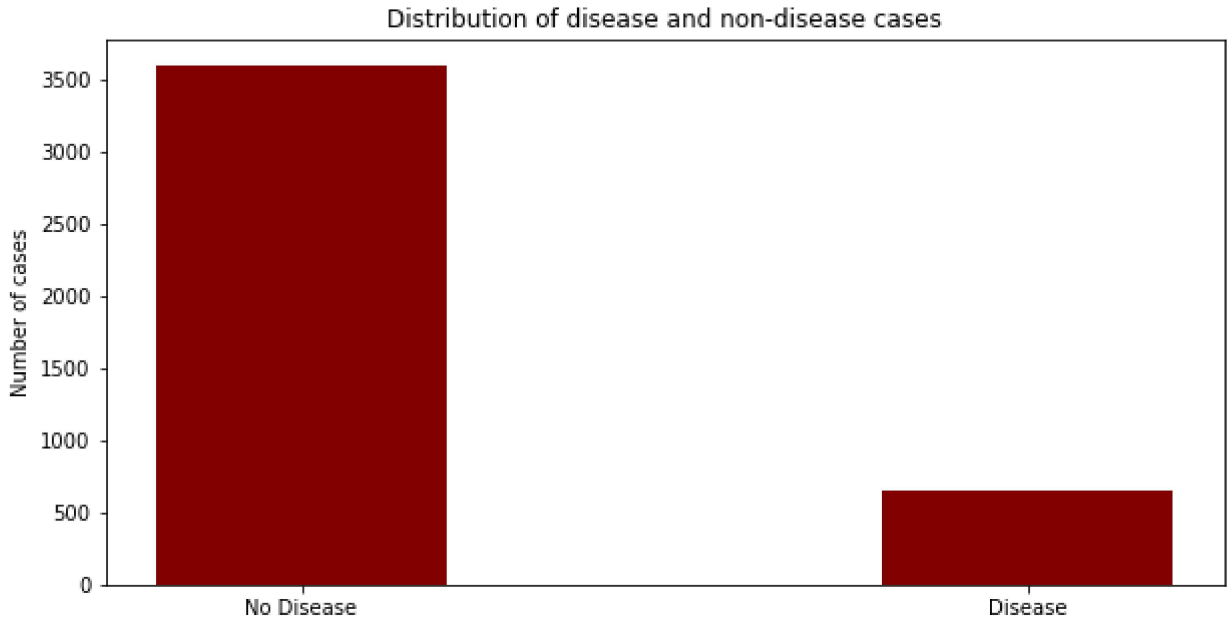
Out[10]:

	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStro
count	4238.000000	4238.000000	4238.000000	4238.000000	4238.000000	4238.00000	4238.0000
mean	0.429212	49.584946	1.978950	0.494101	9.003089	0.02963	0.0058
std	0.495022	8.572160	1.007075	0.500024	11.879230	0.16852	0.0765
min	0.000000	32.000000	1.000000	0.000000	0.000000	0.00000	0.0000
25%	0.000000	42.000000	1.000000	0.000000	0.000000	0.00000	0.0000
50%	0.000000	49.000000	2.000000	0.000000	0.000000	0.00000	0.0000
75%	1.000000	56.000000	3.000000	1.000000	20.000000	0.00000	0.0000
max	1.000000	70.000000	4.000000	1.000000	70.000000	1.00000	1.0000

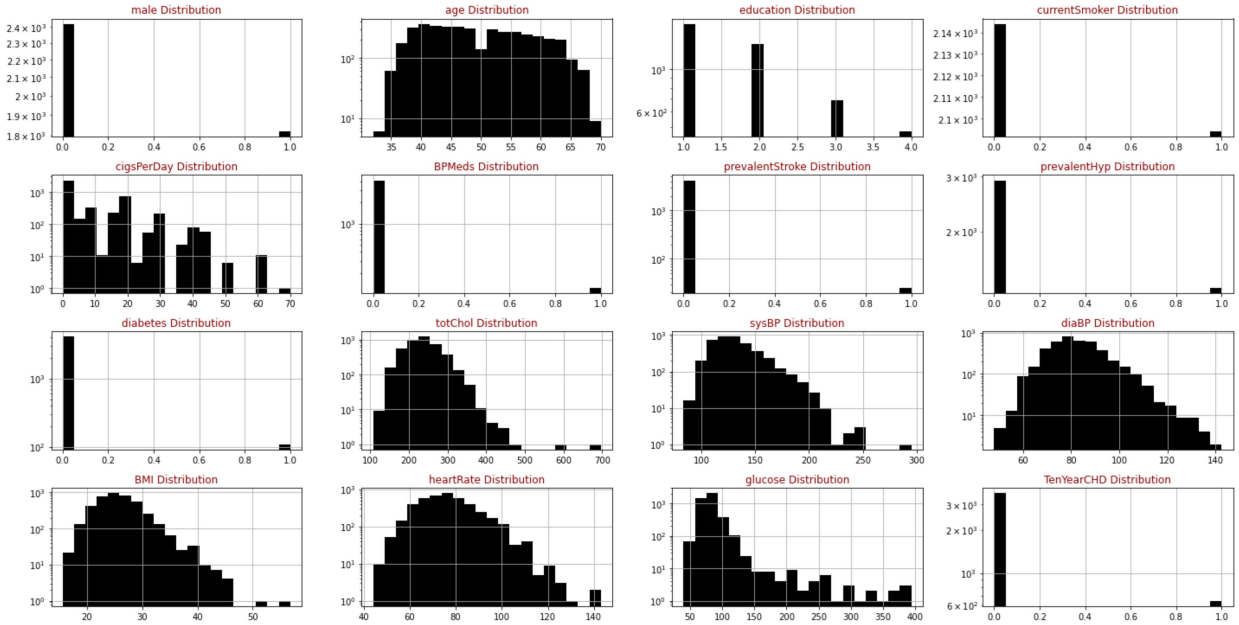
In [11]:
```
# Bar chart of class ratio
target_pd = pd.DataFrame(index = ["No Disease","Disease"], columns= ["Quantity", "Perc
# No disease
target_pd.loc["No Disease"]["Quantity"] = len(df[df.columns[-1]][df[df.columns[-1]]==0
target_pd.loc["No Disease"]["Percentage"] = target_pd.iloc[0,0]/len(df[df.columns[-1]]
# Disease
target_pd.loc["Disease"]["Quantity"] = len(df[df.columns[-1]][df[df.columns[-1]]==1].c
target_pd.loc["Disease"]["Percentage"] = target_pd.iloc[1,0]/len(df[df.columns[-1]])*1
# Plot barchart
fig = plt.figure(figsize = (10, 5))
plt.bar(list(target_pd.index), target_pd.iloc[:,0], color ='maroon',width = 0.4)
plt.ylabel("Number of cases")
plt.title("Distribution of disease and non-disease cases");
# Print the dataframe
target_pd
```

Out[11]:

	Quantity	Percentage
No Disease	3594	84.804153
Disease	644	15.195847



In [12]:
```
# Histogram of features (check for skew)
fig=plt.figure(figsize=(20,20))
for i, feature in enumerate(df.columns):
    ax=fig.add_subplot(8,4,i+1)
    df[feature].hist(bins=20,ax=ax,facecolor='black')
    ax.set_title(feature+" Distribution",color='DarkRed')
    ax.set_yscale('log')
fig.tight_layout()
```



Correlation

In [13]:
```
pp.ProfileReport(df)
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Value	Count	Frequency (%)
32	1	< 0.1%
33	5	0.1%
34	18	0.4%
35	42	1.0%
36	84	2.0%
37	92	2.2%
38	144	3.4%
39	169	4.0%
40	191	4.5%
41	174	4.1%

Value	Count	Frequency (%)
70	2	< 0.1%
69	7	0.2%
68	18	0.4%
67	45	1.1%
66	38	0.9%
65	57	1.3%
64	93	2.2%

Out[13]:

In [14]:
```
#Scaling and splitting the data into the train and test data
y = df["TenYearCHD"]
X = df.drop('TenYearCHD',axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [15]:
```
#logistic regression
m1 = 'Logistic Regression'
lr = LogisticRegression()
model = lr.fit(X_train, y_train)
lr_predict = lr.predict(X_test)
lr_conf_matrix = confusion_matrix(y_test, lr_predict)
lr_acc_score = accuracy_score(y_test, lr_predict)
print("confussion matrix")
print(lr_conf_matrix)
print("\n")
print("Accuracy of Logistic Regression:",lr_acc_score*100,'\n')
print(classification_report(y_test,lr_predict))
```

```
confussion matrix
[[708   2]
 [129   9]]


Accuracy of Logistic Regression: 84.55188679245283

              precision    recall  f1-score   support

           0       0.85      1.00      0.92       710
           1       0.82      0.07      0.12       138

    accuracy                           0.85       848
   macro avg       0.83      0.53      0.52       848
weighted avg       0.84      0.85      0.79       848
```

Additional Models

In [16]:
```
m3 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=20, random_state=2,max_depth=5)
rf.fit(X_train,y_train)
rf_predicted = rf.predict(X_test)
rf_conf_matrix = confusion_matrix(y_test, rf_predicted)
rf_acc_score = accuracy_score(y_test, rf_predicted)
print("confussion matrix")
print(rf_conf_matrix)
print("\n")
print("Accuracy of Random Forest:",rf_acc_score*100,'\n')
print(classification_report(y_test,rf_predicted))
```

```
confussion matrix
[[709   1]
 [136   2]]


Accuracy of Random Forest: 83.84433962264151

              precision    recall  f1-score   support

           0       0.84      1.00      0.91       710
           1       0.67      0.01      0.03       138

    accuracy                           0.84       848
   macro avg       0.75      0.51      0.47       848
weighted avg       0.81      0.84      0.77       848
```

In [17]:
```
#extreme gradient boost
m4 = 'Extreme Gradient Boost'
xgb = XGBClassifier(learning_rate=0.01, n_estimators=25, max_depth=15,gamma=0.6, subsa
                    reg_lambda=2, booster='dart', colsample_bylevel=0.6, colsample_byr
xgb.fit(X_train, y_train)
xgb_predicted = xgb.predict(X_test)
xgb_conf_matrix = confusion_matrix(y_test, xgb_predicted)
xgb_acc_score = accuracy_score(y_test, xgb_predicted)
print("confussion matrix")
print(xgb_conf_matrix)
print("\n")
print("Accuracy of Extreme Gradient Boost:",xgb_acc_score*100,'\n')
print(classification_report(y_test,xgb_predicted))
```

```
confussion matrix
[[710   0]
 [138   0]]


Accuracy of Extreme Gradient Boost: 83.72641509433963

              precision    recall  f1-score   support

           0       0.84      1.00      0.91       710
           1       0.00      0.00      0.00       138

    accuracy                           0.84       848
   macro avg       0.42      0.50      0.46       848
weighted avg       0.70      0.84      0.76       848
```

In [18]:
```
#k-neighbors classifier
m5 = 'K-NeighborsClassifier'
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)
knn_predicted = knn.predict(X_test)
knn_conf_matrix = confusion_matrix(y_test, knn_predicted)
knn_acc_score = accuracy_score(y_test, knn_predicted)
print("confussion matrix")
print(knn_conf_matrix)
print("\n")
print("Accuracy of K-NeighborsClassifier:",knn_acc_score*100,'\n')
print(classification_report(y_test,knn_predicted))
```

```
confussion matrix
[[709   1]
 [135   3]]


Accuracy of K-NeighborsClassifier: 83.9622641509434

              precision    recall  f1-score   support

           0       0.84      1.00      0.91       710
           1       0.75      0.02      0.04       138

    accuracy                           0.84       848
   macro avg       0.80      0.51      0.48       848
weighted avg       0.83      0.84      0.77       848
```

In [19]:
```
#decision tree classifier
m6 = 'DecisionTreeClassifier'
dt = DecisionTreeClassifier(criterion = 'entropy',random_state=0,max_depth = 6)
dt.fit(X_train, y_train)
dt_predicted = dt.predict(X_test)
dt_conf_matrix = confusion_matrix(y_test, dt_predicted)
dt_acc_score = accuracy_score(y_test, dt_predicted)
print("confussion matrix")
print(dt_conf_matrix)
print("\n")
print("Accuracy of DecisionTreeClassifier:",dt_acc_score*100,'\n')
print(classification_report(y_test,dt_predicted))
```

```
confussion matrix
[[706   4]
 [135   3]]


Accuracy of DecisionTreeClassifier: 83.60849056603774

              precision    recall  f1-score   support

           0       0.84      0.99      0.91       710
           1       0.43      0.02      0.04       138

    accuracy                           0.84       848
   macro avg       0.63      0.51      0.48       848
weighted avg       0.77      0.84      0.77       848
```

In [20]:
```
#support vector classifier
m7 = 'Support Vector Classifier'
svc =  SVC(kernel='rbf', C=2)
svc.fit(X_train, y_train)
svc_predicted = svc.predict(X_test)
svc_conf_matrix = confusion_matrix(y_test, svc_predicted)
svc_acc_score = accuracy_score(y_test, svc_predicted)
print("confussion matrix")
print(svc_conf_matrix)
print("\n")
print("Accuracy of Support Vector Classifier:",svc_acc_score*100,'\n')
print(classification_report(y_test,svc_predicted))
```

```
confussion matrix
[[707   3]
 [129   9]]


Accuracy of Support Vector Classifier: 84.43396226415094

              precision    recall  f1-score   support

           0       0.85      1.00      0.91       710
           1       0.75      0.07      0.12       138

    accuracy                           0.84       848
   macro avg       0.80      0.53      0.52       848
weighted avg       0.83      0.84      0.79       848
```

In [ ]: