Predicting_Disease

November 20, 2020

[194]:
```python
#Importing the necessary libraries

import pandas as pd
import numpy as np
import xgboost as xgb
import seaborn as sns #visualisation
import matplotlib.pyplot as plt
```

[195]:
```python
df = pd.read_csv('/Users/harshith/Downloads/Cardio.csv')
```

[196]:
```python
df.head()
```

[196]:
```
   id  age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  \
0   0   50       2     168    62.0    110     80            1     1      0
1   1   55       1     156    85.0    140     90            3     1      0
2   2   51       1     165    64.0    130     70            3     1      0
3   3   48       2     169    82.0    150    100            1     1      0
4   4   47       1     156    56.0    100     60            1     1      0

   alco  active  cardio
0     0       1       0
1     0       1       1
2     0       0       1
3     0       1       1
4     0       0       0
```

[197]:
```python
df.shape
```

[197]:
```
(70000, 13)
```

[198]:
```python
# Checking for null Values

df.isnull().sum()
```

[198]:
```
id             0
age            0
gender         0
```

```
height         0
weight         0
ap_hi          0
ap_lo          0
cholesterol    0
gluc           0
smoke          0
alco           0
active         0
cardio         0
dtype: int64
```

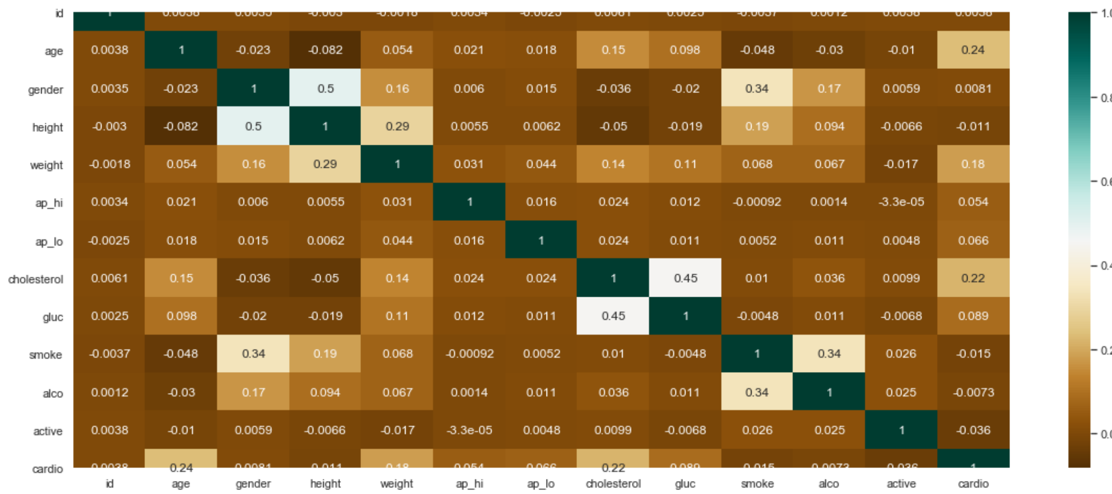
[199]:
```python
df.dtypes
```

[199]:
```
id               int64
age              int64
gender           int64
height           int64
weight         float64
ap_hi            int64
ap_lo            int64
cholesterol      int64
gluc             int64
smoke            int64
alco             int64
active           int64
cardio           int64
dtype: object
```

[200]:
```python
# Finding the correlation between the features

import seaborn as sns #visualisation
import matplotlib.pyplot as plt #visualisation
%matplotlib inline
sns.set(color_codes=True)

plt.figure(figsize=(20,8))
c= df.corr()
sns.heatmap(c,cmap='BrBG',annot=True)
c
```

[200]:
```
                   id       age    gender    height    weight     ap_hi  \
id           1.000000  0.003814  0.003502 -0.003038 -0.001830  0.003356
age          0.003814  1.000000 -0.022913 -0.081506  0.053561  0.020854
gender       0.003502 -0.022913  1.000000  0.499033  0.155406  0.006005
height      -0.003038 -0.081506  0.499033  1.000000  0.290968  0.005488
weight      -0.001830  0.053561  0.155406  0.290968  1.000000  0.030702
```

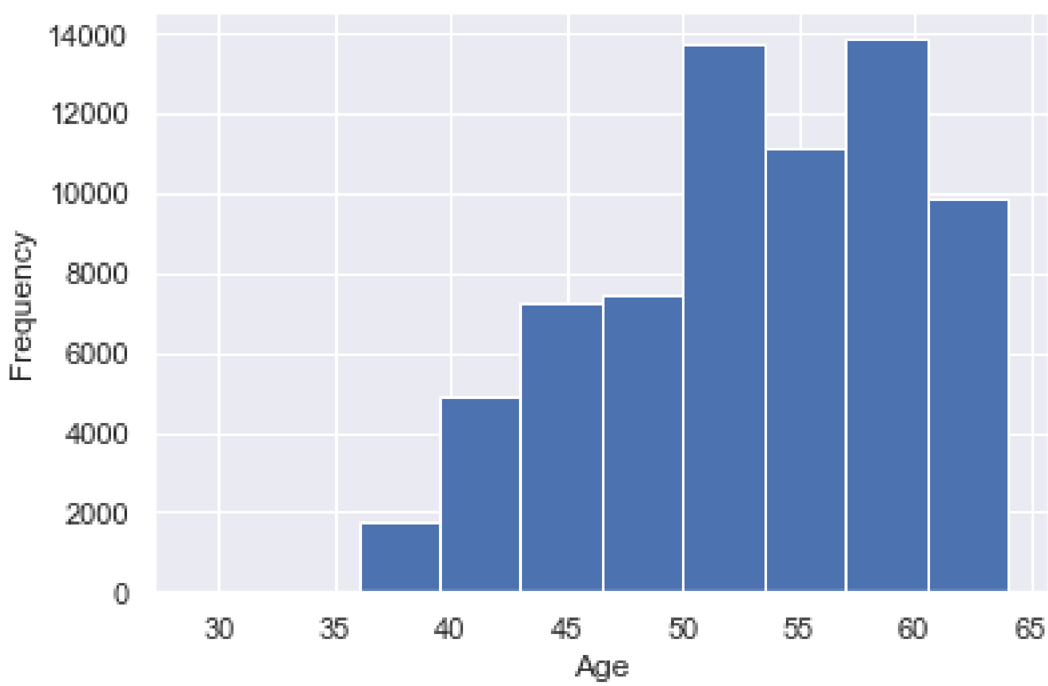
```
ap_hi         0.003356  0.020854  0.006005  0.005488  0.030702  1.000000
ap_lo        -0.002529  0.017620  0.015254  0.006150  0.043710  0.016086
cholesterol   0.006106  0.154012 -0.035821 -0.050226  0.141768  0.023778
gluc          0.002467  0.098388 -0.020491 -0.018595  0.106857  0.011841
smoke        -0.003699 -0.047649  0.338135  0.187989  0.067780 -0.000922
alco          0.001210 -0.029756  0.170966  0.094419  0.067113  0.001408
active        0.003755 -0.009998  0.005866 -0.006570 -0.016867 -0.000033
cardio        0.003799  0.237985  0.008109 -0.010821  0.181660  0.054475

                ap_lo  cholesterol      gluc     smoke      alco    active  \
id          -0.002529     0.006106  0.002467 -0.003699  0.001210  0.003755
age          0.017620     0.154012  0.098388 -0.047649 -0.029756 -0.009998
gender       0.015254    -0.035821 -0.020491  0.338135  0.170966  0.005866
height       0.006150    -0.050226 -0.018595  0.187989  0.094419 -0.006570
weight       0.043710     0.141768  0.106857  0.067780  0.067113 -0.016867
ap_hi        0.016086     0.023778  0.011841 -0.000922  0.001408 -0.000033
ap_lo        1.000000     0.024019  0.010806  0.005186  0.010601  0.004780
cholesterol  0.024019     1.000000  0.451578  0.010354  0.035760  0.009911
gluc         0.010806     0.451578  1.000000 -0.004756  0.011246 -0.006770
smoke        0.005186     0.010354 -0.004756  1.000000  0.340094  0.025858
alco         0.010601     0.035760  0.011246  0.340094  1.000000  0.025476
active       0.004780     0.009911 -0.006770  0.025858  0.025476  1.000000
cardio       0.065719     0.221147  0.089307 -0.015486 -0.007330 -0.035653

               cardio
id           0.003799
age          0.237985
gender       0.008109
height      -0.010821
weight       0.181660
ap_hi        0.054475
ap_lo        0.065719
cholesterol  0.221147
gluc         0.089307
smoke       -0.015486
alco        -0.007330
active      -0.035653
cardio       1.000000
```



[265]:

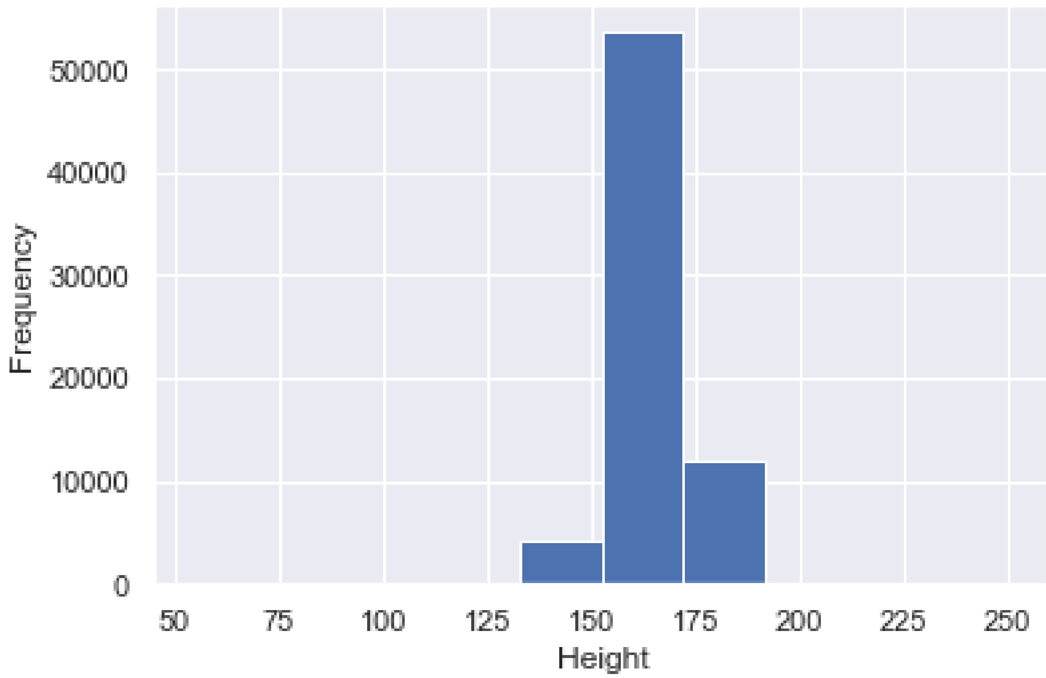
```
df['age'].hist()
plt.xlabel('Age')
plt.ylabel('Frequency')
```

[265]: Text(0, 0.5, 'Frequency')



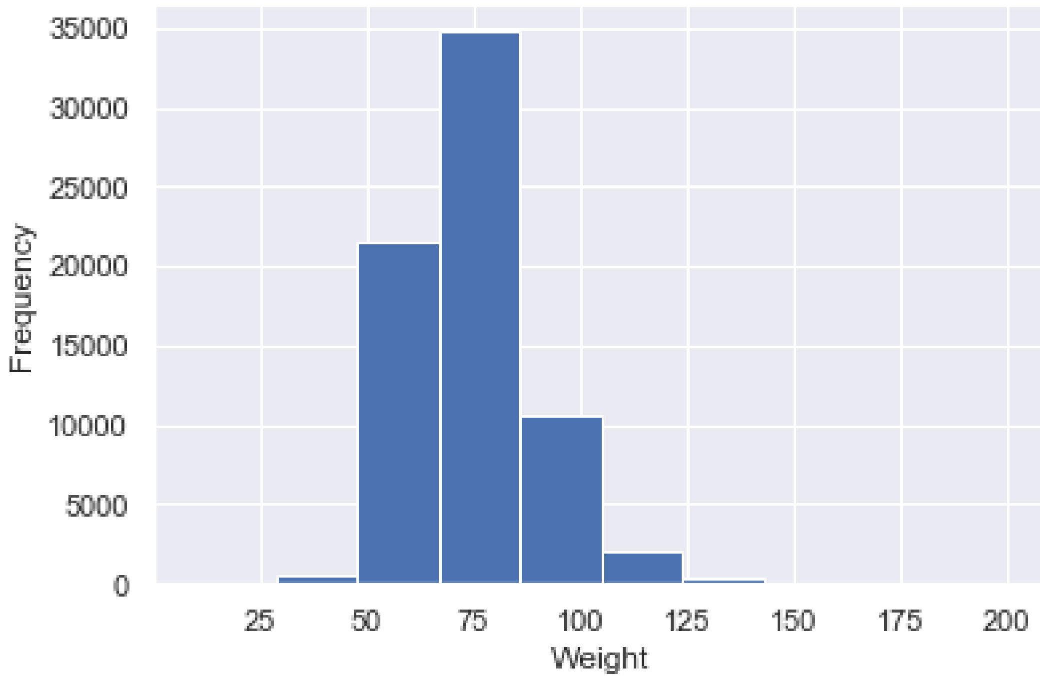
[267]:
```
df['height'].hist()
plt.xlabel('Height')
plt.ylabel('Frequency')
```

[267]: Text(0, 0.5, 'Frequency')



[268]:
```
df['weight'].hist()
plt.xlabel('Weight')
plt.ylabel('Frequency')
```

[268]: Text(0, 0.5, 'Frequency')

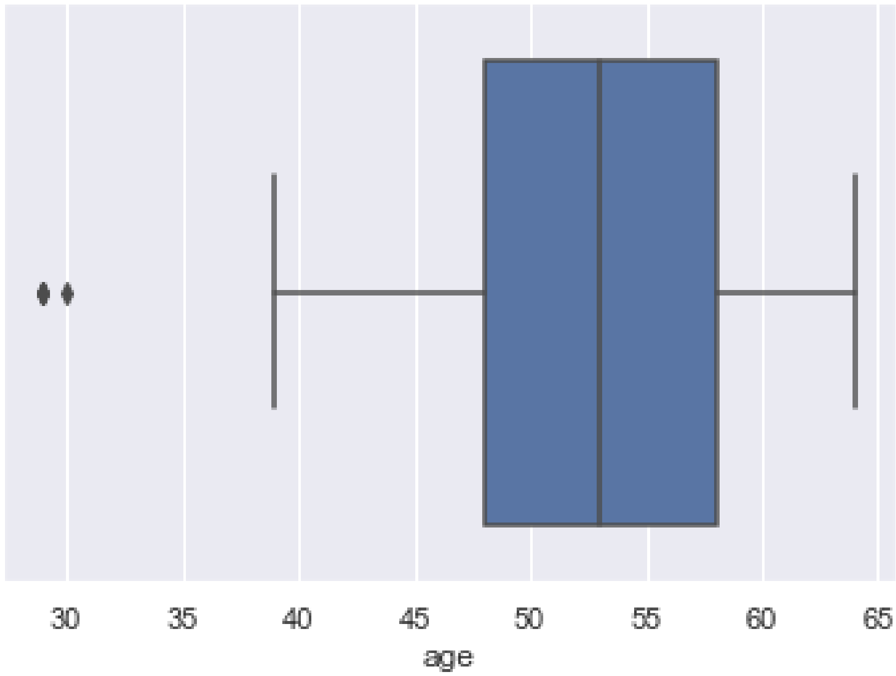


[204]:
```
# Getting the Individual Value counts in the feature column to see if the class␣
↪is balanced

df['cardio'].value_counts()
```

[204]:
```
0    63831
1     6169
Name: smoke, dtype: int64
```

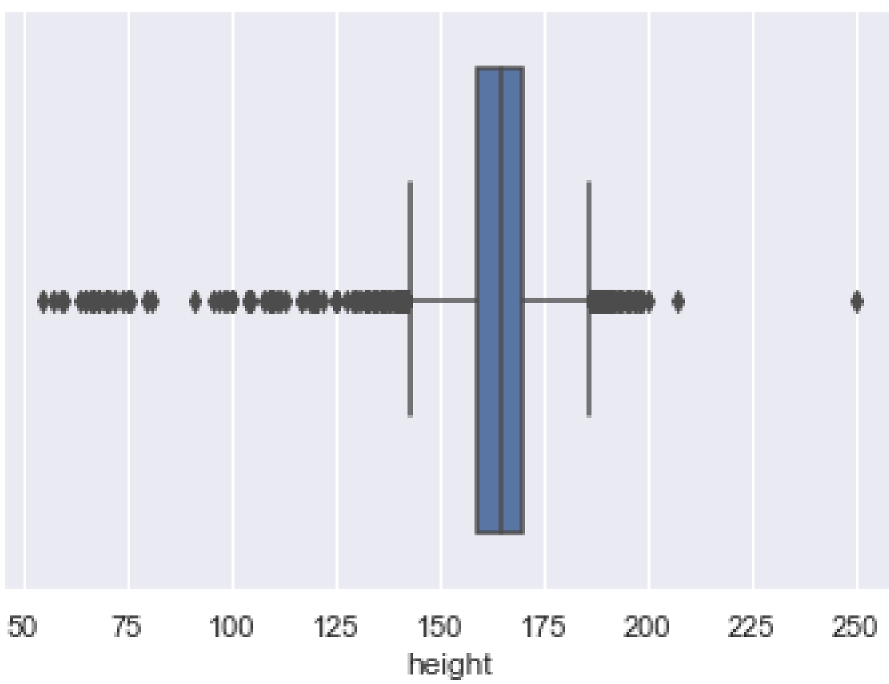
[206]:
```
# Checking for Outliers, but not dropping the values beacuse they might be␣
↪genuine readings. We can further
# investigate by going in depth analysis on the readings of the sensors.

sns.boxplot(x=df['age'])
```

[206]: <matplotlib.axes._subplots.AxesSubplot at 0x1a257fbb10>



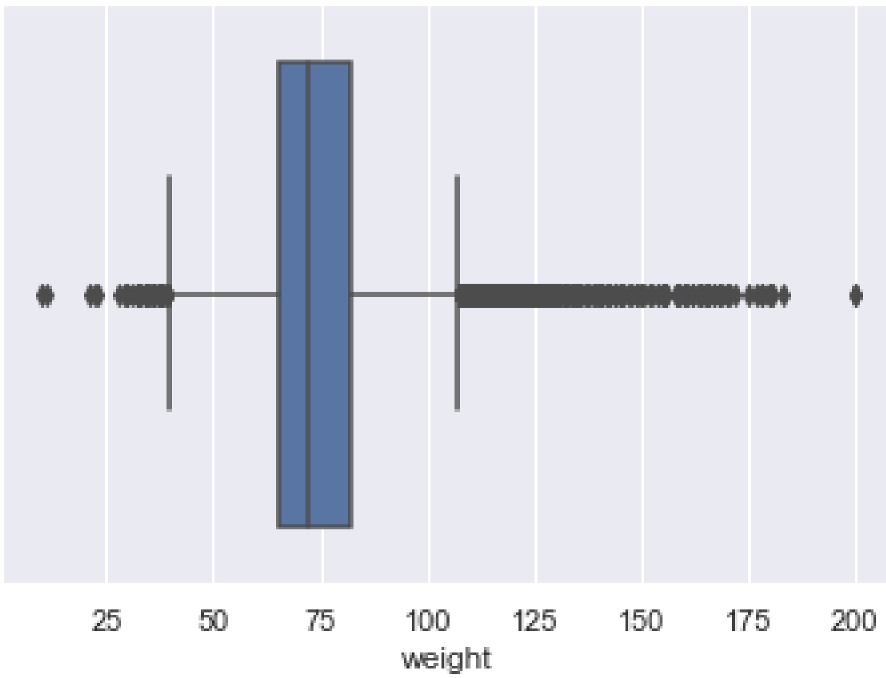
[207]:
```
sns.boxplot(x=df['height'])
```

[207]: <matplotlib.axes._subplots.AxesSubplot at 0x1a26c60b10>



[208]:
```
sns.boxplot(x=df['weight'])
```

[208]: <matplotlib.axes._subplots.AxesSubplot at 0x1a26ce6d50>



[210]:
```
df.columns
```

[210]:
```
Index(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo',
       'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio'],
      dtype='object')
```

[211]:
```
df['cholesterol'].value_counts()
```

[211]:
```
1    52385
2     9549
3     8066
Name: cholesterol, dtype: int64
```

[212]:
```
df['gluc'].value_counts()
```

[212]:
```
1    59479
3     5331
2     5190
Name: gluc, dtype: int64
```

[213]:
```
# One-hot Encoding is performed on the categorical variables.
df_encoded = pd.get_dummies(df, columns=['cholesterol', 'gluc', 'gender'],␣
 ↪drop_first=False)
df_encoded.head()
```

[213]:
```
   id  age  height  weight  ap_hi  ap_lo  smoke  alco  active  cardio  \
0   0   50     168    62.0    110     80      0     0       1       0
1   1   55     156    85.0    140     90      0     0       1       1
2   2   51     165    64.0    130     70      0     0       0       1
3   3   48     169    82.0    150    100      0     0       1       1
4   4   47     156    56.0    100     60      0     0       0       0

   cholesterol_1  cholesterol_2  cholesterol_3  gluc_1  gluc_2  gluc_3  \
0              1              0              0       1       0       0
1              0              0              1       1       0       0
2              0              0              1       1       0       0
3              1              0              0       1       0       0
4              1              0              0       1       0       0

   gender_1  gender_2
0         0         1
1         1         0
2         1         0
3         0         1
4         1         0
```

[ ]:
```
# Tried binning as a part of feature Engineering but the the accuracy and the␣
 ↪other metrics were decreased when
# featured engineered.

# I think the model will be generalize better if the values are not binned␣
 ↪since it is Cardio detection.
# Hence, binning is not peformed but the code is commented out.
```

[176]:
```
#df_encoded['Height_binned'] = pd.qcut(df_encoded.height , q = 3, labels =␣
 ↪False)
```

[177]:
```
#df_encoded['Weight_binned'] = pd.qcut(df_encoded.weight , q = 3, labels =␣
 ↪False)
```

[179]:
```
#df_encoded = df_encoded.drop('height', axis = 1)
#df_encoded = df_encoded.drop('weight', axis = 1)
```

[215]:
```
df_encoded.head()
```

[215]:
```
   id  age  height  weight  ap_hi  ap_lo  smoke  alco  active  cardio  \
0   0   50     168    62.0    110     80      0     0       1       0
```

1	1	55	156	85.0	140	90	0	0	1	1
2	2	51	165	64.0	130	70	0	0	0	1
3	3	48	169	82.0	150	100	0	0	1	1
4	4	47	156	56.0	100	60	0	0	0	0

	cholesterol_1	cholesterol_2	cholesterol_3	gluc_1	gluc_2	gluc_3 \
0	1	0	0	1	0	0
1	0	0	1	1	0	0
2	0	0	1	1	0	0
3	1	0	0	1	0	0
4	1	0	0	1	0	0

	gender_1	gender_2
0	0	1
1	1	0
2	1	0
3	0	1
4	1	0

[216]:
```
df_encoded.columns
```

[216]:
```
Index(['id', 'age', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco',
       'active', 'cardio', 'cholesterol_1', 'cholesterol_2', 'cholesterol_3',
       'gluc_1', 'gluc_2', 'gluc_3', 'gender_1', 'gender_2'],
      dtype='object')
```

[217]:
```
# Separating the input features and target

X = df_encoded.drop('cardio', axis=1)
X = X.drop('id', axis = 1)
y = df_encoded.cardio

# setting up testing and training sets

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=123, stratify = y)
```

[218]:
```
X.head()
```

[218]:

	age	height	weight	ap_hi	ap_lo	smoke	alco	active	cholesterol_1 \
0	50	168	62.0	110	80	0	0	1	1
1	55	156	85.0	140	90	0	0	1	0
2	51	165	64.0	130	70	0	0	0	0
3	48	169	82.0	150	100	0	0	1	1
4	47	156	56.0	100	60	0	0	0	1

	cholesterol_2	cholesterol_3	gluc_1	gluc_2	gluc_3	gender_1	gender_2
0	0	0	1	0	0	0	1
1	0	1	1	0	0	1	0
2	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1
4	0	0	1	0	0	1	0

[258]:

```
# Performed Randomized Search for tuning of few parameters

from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb

# Create the parameter grid for hyper para,eter tuning: gbm_param_grid
gbm_param_grid = {
    'colsample_bytree': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,1],
    'n_estimators': [50, 75, 100, 125, 150, 200, 250, 300, 350, 400, 450, 500],
    'max_depth': [2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32],
    'eta' : [0.1, 0.01, 0.001, 0.2, 0.3]
}


gbm = xgb.XGBClassifier(objective = 'reg:logistic' , random_state = 123)

random_search = RandomizedSearchCV(estimator = gbm, param_distributions =␣
 ↪gbm_param_grid, scoring='roc_auc', n_jobs=-1, cv=4, verbose=1)
random_search.fit(X_train,y_train)


best_n_estim   = random_search.best_params_['n_estimators']
best_max_depth = random_search.best_params_['max_depth']
best_colsample = random_search.best_params_['colsample_bytree']
best_eta = random_search.best_params_['eta']

#The best parameters are selected based on the best AUC value

print("The best parameters are:", random_search.best_params_)


# The final model is built with best params for simplicity
best_gbm = xgb.XGBClassifier(objective = 'reg:logistic' , n_estimators =␣
 ↪best_n_estim, max_depth = best_max_depth, eta = best_eta, random_state = 123)

# Fit grid_mse to the data
best_gbm.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```

```
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed:  7.4min finished
```

```
The best parameters are: {'n_estimators': 450, 'max_depth': 5, 'eta': 0.001,
'colsample_bytree': 0.2}
```

[258]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=450, n_jobs=1,
              nthread=None, objective='reg:logistic', random_state=123,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

[261]:
```python
y_pred = best_gbm.predict(X_test)
```

[262]:
```python
# Probabilities for each class
rf_probs = best_gbm.predict_proba(X_test)[:, 1]

from sklearn.metrics import roc_auc_score

# Calculate roc auc
auc_value = roc_auc_score(y_test, rf_probs)
print("The Auc Value:" , auc_value)
```

```
The Auc Value: 0.7995944815002387
```

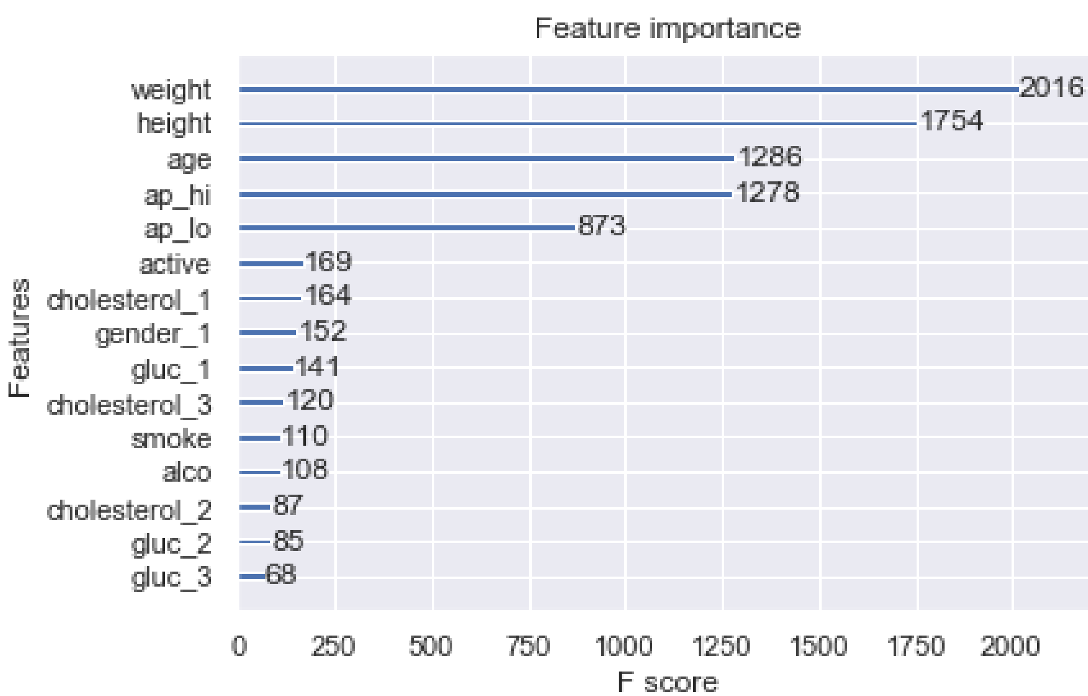
[263]:
```python
#The metric evaluation for my final model.

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print("The confusion Matrix:\n",cm)

from sklearn.metrics import f1_score
f1_score = f1_score(y_test, y_pred)
print("F1 score:", f1_score)

from sklearn.metrics import accuracy_score
Training_accuracy = accuracy_score(y_train, best_gbm.predict(X_train))
Testing_accuracy = accuracy_score(y_test, y_pred)
print("Training accuracy:", Training_accuracy)
print("Testing accuracy:", Testing_accuracy)

from sklearn.metrics import precision_score, recall_score
precision_score = precision_score(y_test, y_pred)

recall_score = recall_score(y_test, y_pred)

print("Precision Score:", precision_score)
```

```python
print("Recall Score:", recall_score)
```

```
The confusion Matrix:
 [[6829 1926]
 [2715 6030]]
F1 score: 0.7221124483563859
Training accuracy: 0.7554666666666666
Testing accuracy: 0.7348
Precision Score: 0.7579185520361991
Recall Score: 0.6895368782161235
```

[264]:

```python
#Plotting the feature Importance. This makes sense.

xgb.plot_importance(best_gbm)
plt.show()
```



[ ]: