```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

datos = pd.read_csv('/content/heart.csv')
df = pd.DataFrame(datos)
df.head()
```

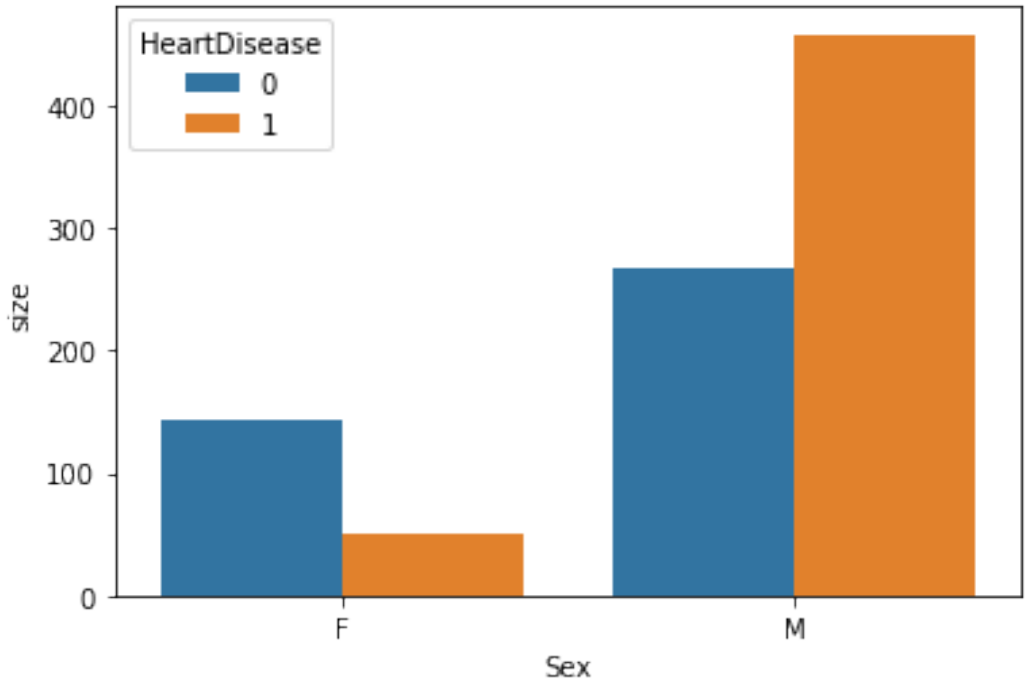
```
   Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG
MaxHR  \
0   40   M           ATA        140          289          0     Normal
172
1   49   F           NAP        160          180          0     Normal
156
2   37   M           ATA        130          283          0         ST
98
3   48   F           ASY        138          214          0     Normal
108
4   54   M           NAP        150          195          0     Normal
122

  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
```

```
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

```
count_series = df.groupby(['Sex', 'HeartDisease']).size()
dfSex = count_series.to_frame(name = 'size').reset_index()
count_series = df.groupby(['ChestPainType', 'HeartDisease']).size()
dfCPT = count_series.to_frame(name = 'size').reset_index()
count_series = df.groupby(['RestingECG', 'HeartDisease']).size()
dfRes = count_series.to_frame(name = 'size').reset_index()
count_series = df.groupby(['ExerciseAngina', 'HeartDisease']).size()
dfea = count_series.to_frame(name = 'size').reset_index()
count_series = df.groupby(['ST_Slope', 'HeartDisease']).size()
dfSlope = count_series.to_frame(name = 'size').reset_index()
```

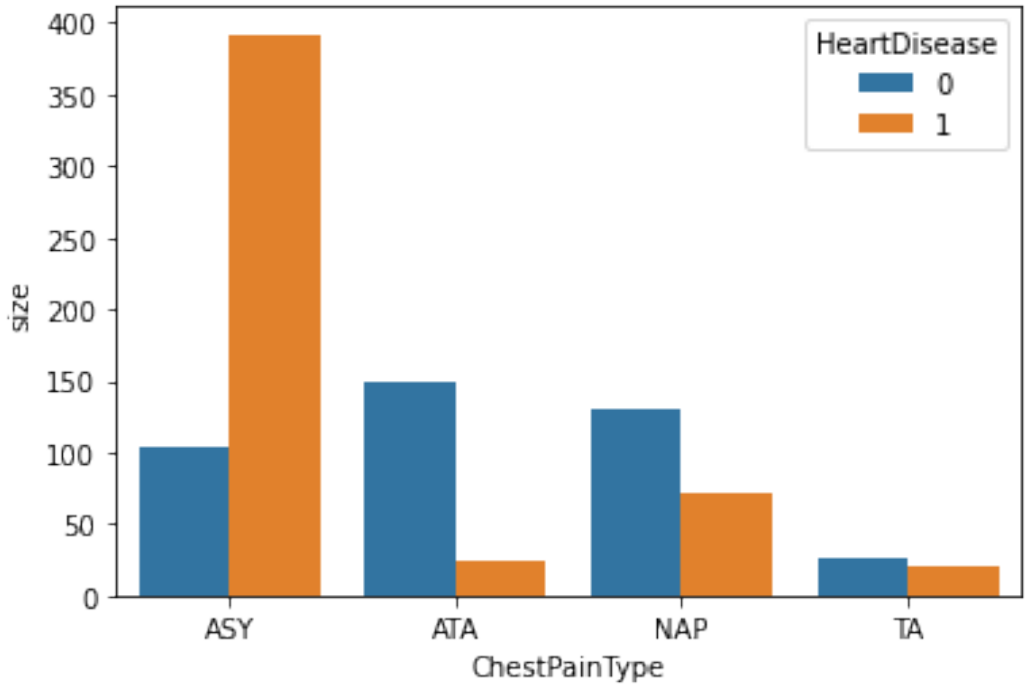
```
import seaborn as sns
sns.barplot(data=dfSex, x="Sex", y="size", hue="HeartDisease")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe94af08390>
```



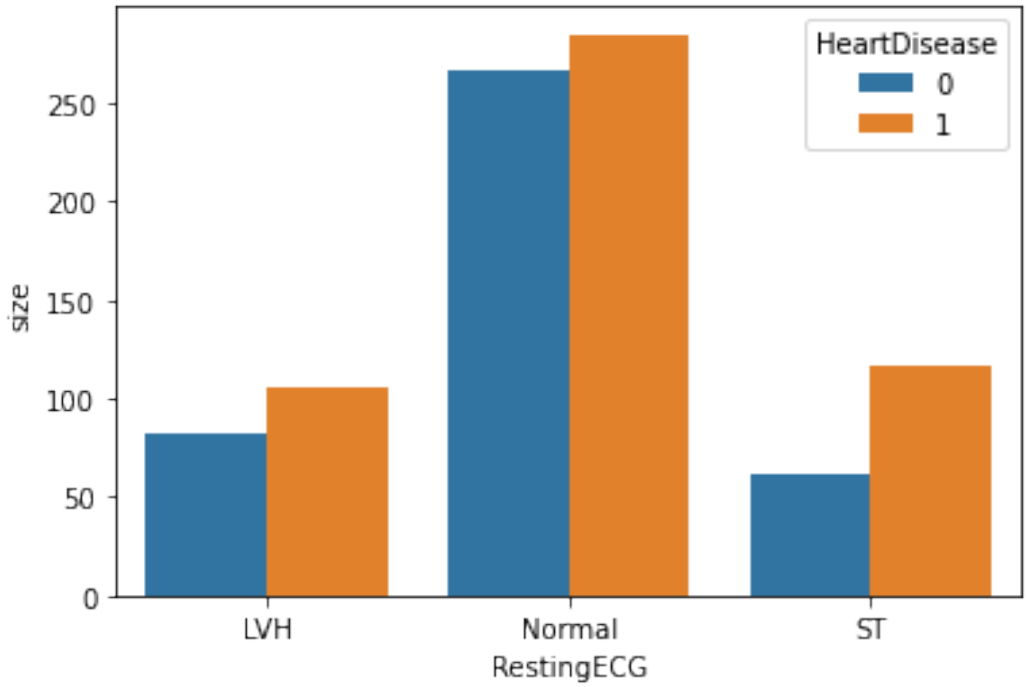

```
sns.barplot(data=dfCPT, x="ChestPainType", y="size",
hue="HeartDisease")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe94ae72750>
```



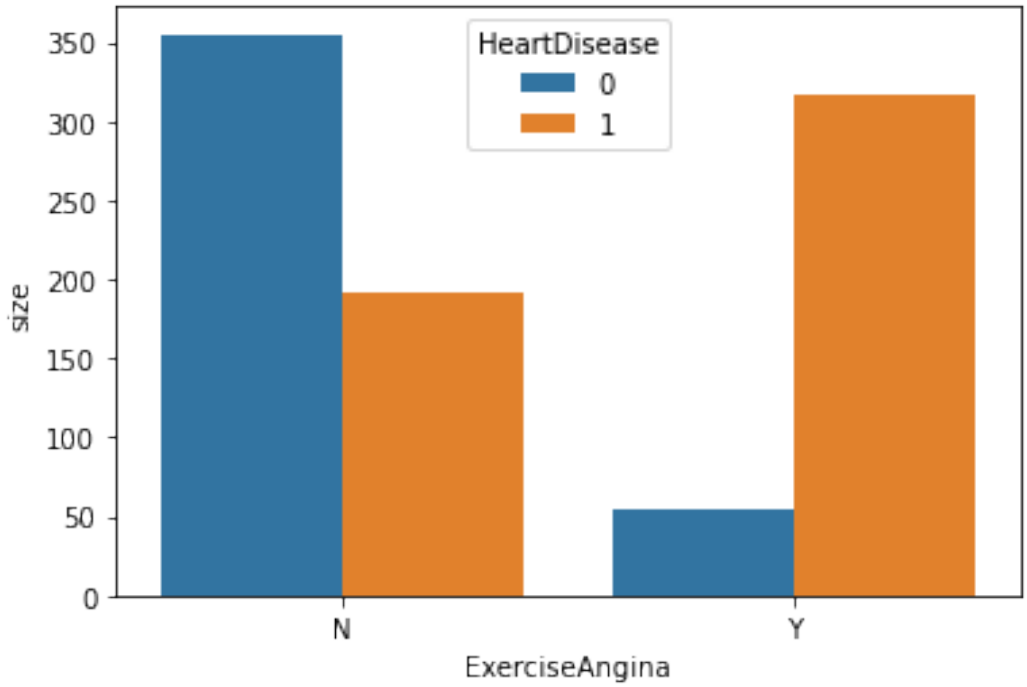

```
sns.barplot(data=dfRes, x="RestingECG", y="size", hue="HeartDisease")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe94a961110>
```



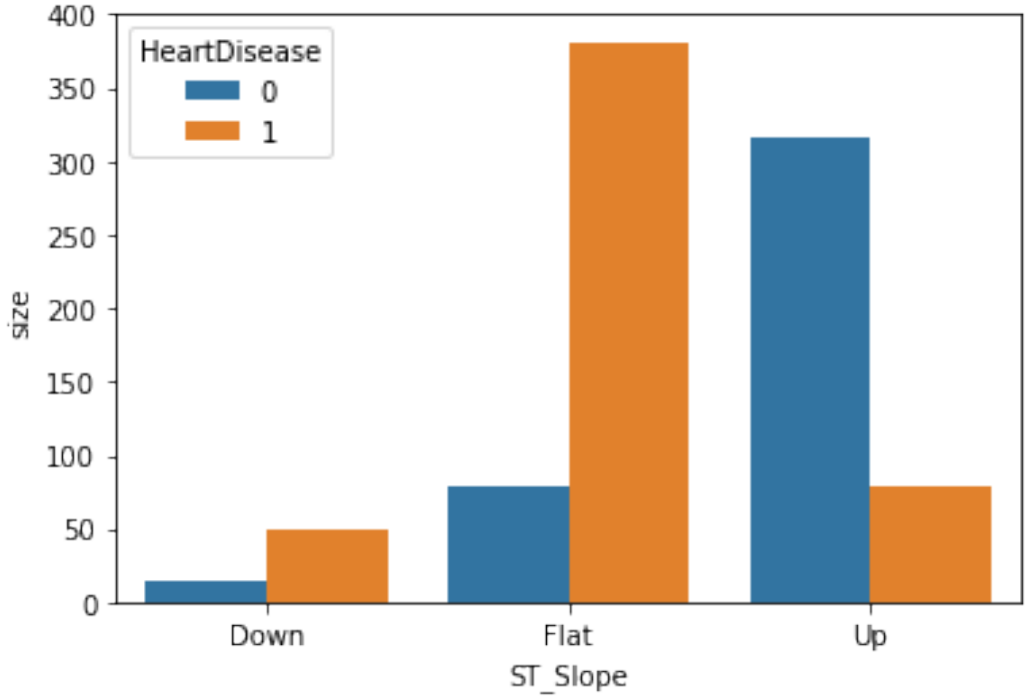

```
sns.barplot(data=dfea, x="ExerciseAngina", y="size",
hue="HeartDisease")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe94a8e2390>
```




```
sns.barplot(data=dfSlope, x="ST_Slope", y="size", hue="HeartDisease")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe94a872190>
```



- change a Sex column values for number. M = 1, F = 0.

```
df['Sex'].replace({'F':0,'M':1},inplace=True)
df.head(5)
```

```
   Age  Sex ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
0   40    1           ATA        140          289          0
Normal
1   49    0           NAP        160          180          0
Normal
2   37    1           ATA        130          283          0
ST
3   48    0           ASY        138          214          0
Normal
4   54    1           NAP        150          195          0
Normal

   MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0    172              N      0.0       Up             0
1    156              N      1.0     Flat             1
2     98              N      0.0       Up             0
3    108              Y      1.5     Flat             1
4    122              N      0.0       Up             0
```

- Change a ChestPainType column values for number. ASY= 1, ATA= 2, NAP=3, TA=4.

```
df['ChestPainType'].replace({'ASY':1,'ATA':2,'NAP':3,'TA':4},inplace=T
rue)
df.head(5)
```

```
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
0   40    1              2        140          289          0
Normal
1   49    0              3        160          180          0
Normal
2   37    1              2        130          283          0
ST
3   48    0              1        138          214          0
Normal
4   54    1              3        150          195          0
Normal

   MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0    172              N      0.0       Up             0
1    156              N      1.0     Flat             1
2     98              N      0.0       Up             0
3    108              Y      1.5     Flat             1
4    122              N      0.0       Up             0
```

- Change a RestingECG column values for number. LVH= 1, Normal= 2, ST=3.

```
df['RestingECG'].replace({'LVH':1,'Normal':2,'ST':3},inplace=True)
df.head(5)
```

```
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
0   40    1              2        140          289          0
2
1   49    0              3        160          180          0
2
2   37    1              2        130          283          0
3
3   48    0              1        138          214          0
2
4   54    1              3        150          195          0
2

   MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0    172              N      0.0       Up             0
1    156              N      1.0     Flat             1
2     98              N      0.0       Up             0
3    108              Y      1.5     Flat             1
4    122              N      0.0       Up             0
```

- Change a ExerciseAngina column values for number. N= 0, Y= 1.

```
df['ExerciseAngina'].replace({'N':0,'Y':1},inplace=True)
df.head(5)
```

```
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
0   40    1              2        140          289          0
2
1   49    0              3        160          180          0
2
2   37    1              2        130          283          0
3
3   48    0              1        138          214          0
2
4   54    1              3        150          195          0
2

   MaxHR  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0    172               0      0.0       Up             0
1    156               0      1.0     Flat             1
2     98               0      0.0       Up             0
3    108               1      1.5     Flat             1
4    122               0      0.0       Up             0
```

- Change a ST_Slope column values for number. Down= 1, Flat= 2, Up= 3.

```
df['ST_Slope'].replace({'Down':1,'Flat':2, 'Up':3},inplace=True)
df.head(5)
```

```
   Age  Sex  ChestPainType  RestingBP  Cholesterol  FastingBS
RestingECG  \
```

```
0   40    1              2        140          289          0
2
1   49    0              3        160          180          0
2
2   37    1              2        130          283          0
3
3   48    0              1        138          214          0
2
4   54    1              3        150          195          0
2
```

```
   MaxHR  ExerciseAngina  Oldpeak  ST_Slope  HeartDisease
0    172               0      0.0         3             0
1    156               0      1.0         2             1
2     98               0      0.0         3             0
3    108               1      1.5         2             1
4    122               0      0.0         3             0
```

```
df.notnull().sum()
```

```
Age               918
Sex               918
ChestPainType     918
RestingBP         918
Cholesterol       918
FastingBS         918
RestingECG        918
MaxHR             918
ExerciseAngina    918
Oldpeak           918
ST_Slope          918
HeartDisease      918
dtype: int64
```

Detect Outliers

```
from sklearn.neighbors import LocalOutlierFactor
clf = LocalOutlierFactor()
y_pred = clf.fit_predict(df)
```

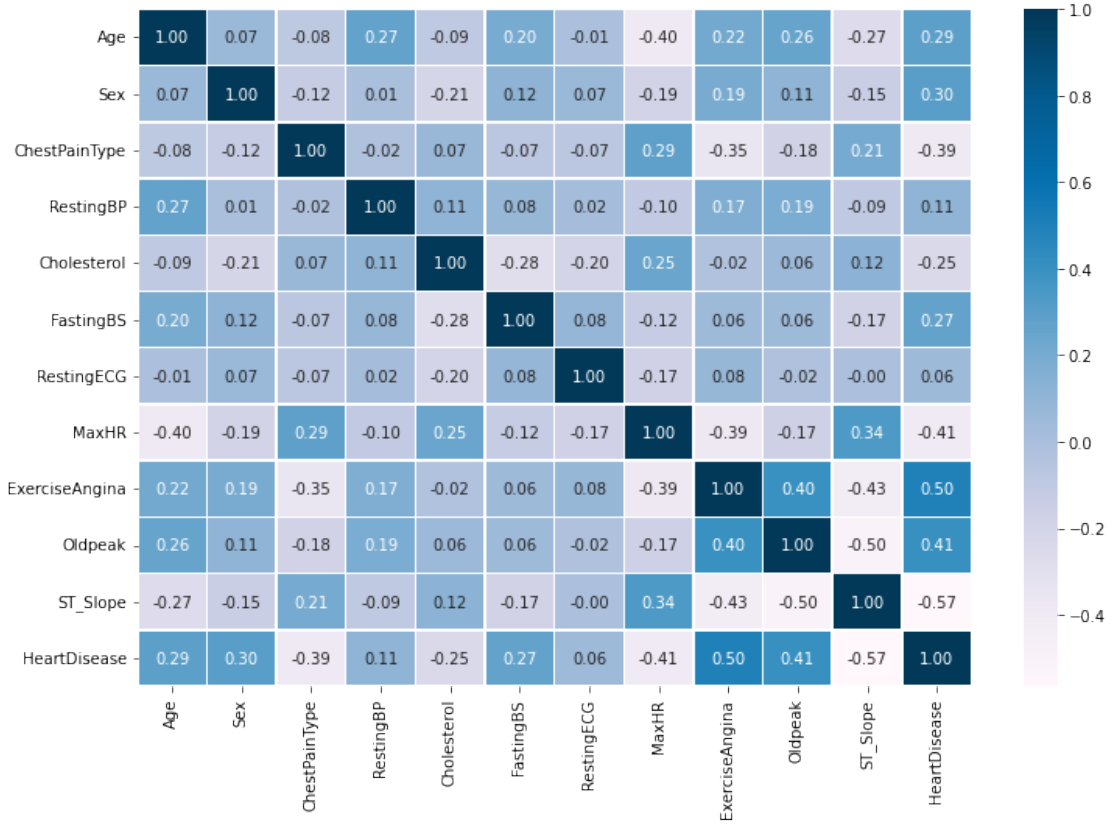
```
x_score = clf.negative_outlier_factor_
outlier_score = pd.DataFrame()
outlier_score["score"] = x_score

#threshold
threshold2 = -1.5
filtre2 = outlier_score["score"] < threshold2
outlier_index = outlier_score[filtre2].index.tolist()
```

```
len(outlier_index)
```

```
21
```

```
df.drop(outlier_index, inplace=True)
```

```
import seaborn as sns
f,ax = plt.subplots(figsize=(12,8))
sns.heatmap(df.corr(), cmap="PuBu", annot=True, linewidths=0.5, fmt=
'.2f',ax=ax)
plt.show()
```



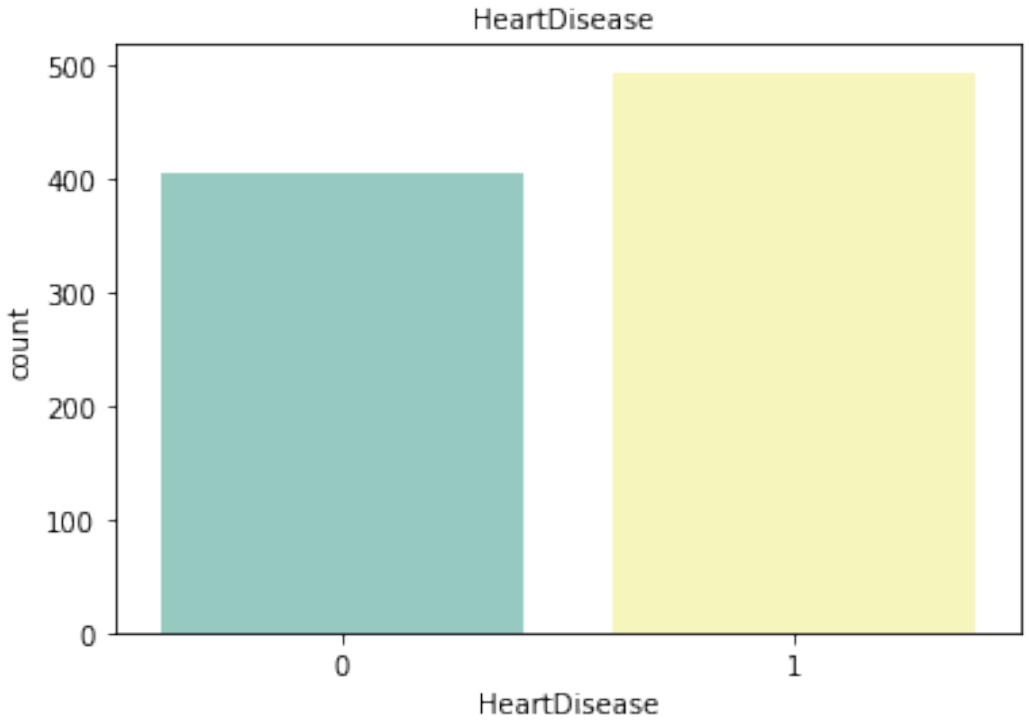
Dealing with Imbalanced Data

```
from imblearn import under_sampling
from imblearn import over_sampling
from imblearn.over_sampling import SMOTE
from collections import Counter
```

```
x = df.drop(['HeartDisease'], axis = 1)
y = df.loc[:,'HeartDisease'].values
```

```
sns.countplot(df["HeartDisease"], palette="Set3")
plt.title("HeartDisease ",fontsize=10)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From
version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  FutureWarning
```

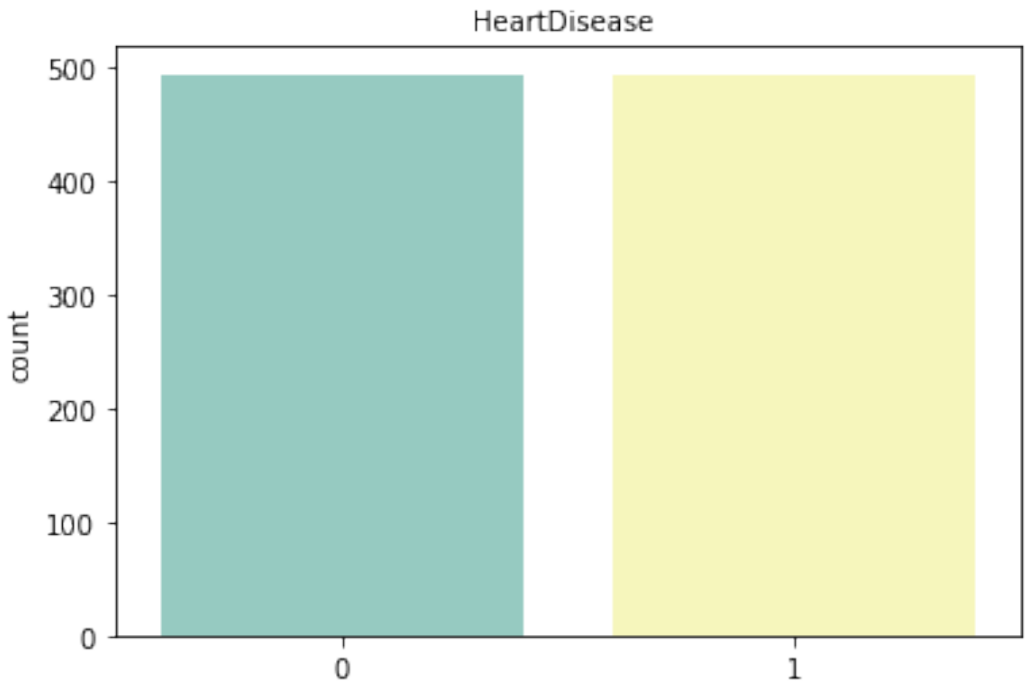



```
sm = SMOTE(random_state=42)
print('Original dataset shape %s' % Counter(y))
x, y = sm.fit_resample(x, y)
print('Resampled dataset shape %s' % Counter(y))
```

```
Original dataset shape Counter({1: 493, 0: 404})
Resampled dataset shape Counter({0: 493, 1: 493})
```

```
sns.countplot(y, palette='Set3')
plt.title("HeartDisease ",fontsize=10)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43:
FutureWarning: Pass the following variable as a keyword arg: x. From
version 0.12, the only valid positional argument will be `data`, and
passing other arguments without an explicit keyword will result in an
error or misinterpretation.
  FutureWarning
```



Data Scaling

```
from sklearn import preprocessing
x= preprocessing.StandardScaler().fit(x).transform(x)
x[0:5]
```

```
array([[-1.42891278,  0.56563809,  0.22453695,  0.46448924,
0.87084565,
        -0.51863597,  0.05815202,  1.37240322, -0.76665616, -
0.80759242,
         1.00899201],
       [-0.46617859, -1.76791488,  1.28383566,  1.64228632, -
0.20000326,
        -0.51863597,  0.05815202,  0.73206745, -0.76665616,
0.15058123,
        -0.64086889],
       [-1.74982417,  0.56563809,  0.22453695, -0.1244093 ,
0.81189984,
        -0.51863597,  1.65087129, -1.58914974, -0.76665616, -
0.80759242,
         1.00899201],
       [-0.57314905, -1.76791488, -0.83476177,  0.34670954,
0.13402301,
        -0.51863597,  0.05815202, -1.18893988,  1.30436569,
0.62966805,
        -0.64086889],
```

```
       [ 0.06867374,  0.56563809,  1.28383566,  1.05338778, -
0.05263873,
        -0.51863597,  0.05815202, -0.62864608, -0.76665616, -
0.80759242,
         1.00899201]])
```

Train / Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Validation set:', X_test.shape,  y_test.shape)
```

```
Train set: (788, 11) (788,)
Validation set: (198, 11) (198,)
```

Classifiers

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ClassPredictionError
from yellowbrick.classifier import ROCAUC
from yellowbrick.style import set_palette
```

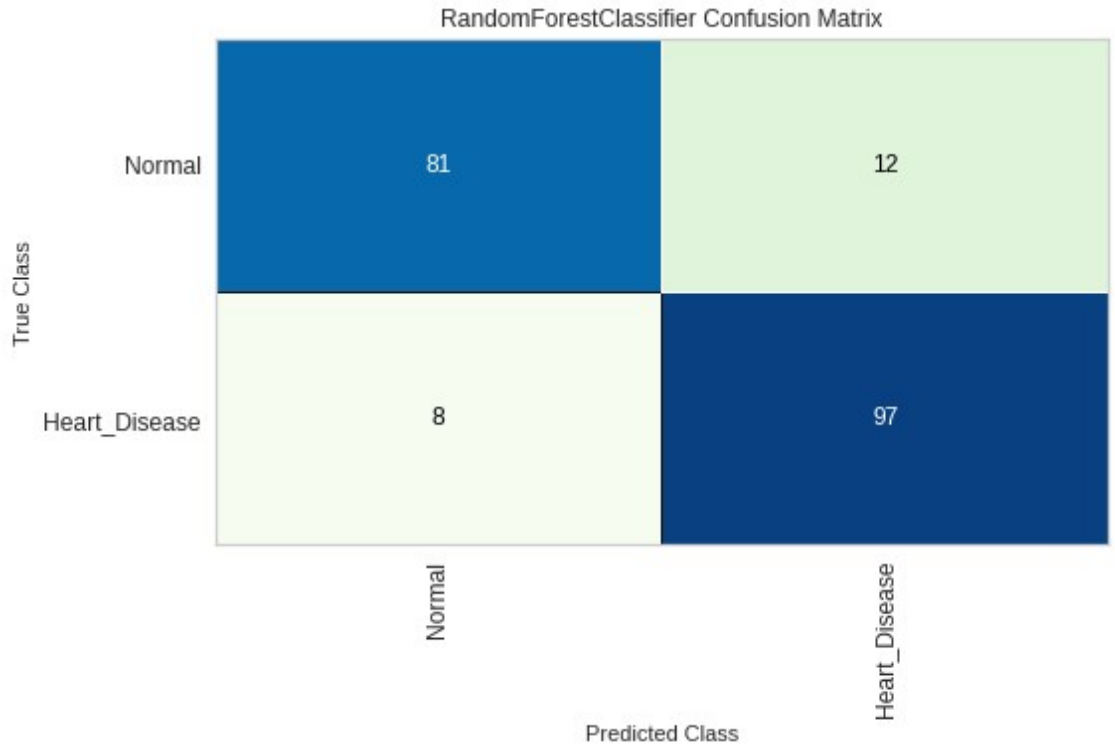
```
r_forest = RandomForestClassifier()
r_forest.fit(X_train,y_train)
predicted = r_forest.predict(X_test)
score = r_forest.score(X_test, y_test)
rf_score_ = np.mean(score)

print('Accuracy : %.3f' % (rf_score_))
```

```
Accuracy : 0.899
```

Confusion Matrix

```
classes = ['Normal', 'Heart_Disease']
# 0=Normal, 1=Heart_Disease

r_forest_cm = ConfusionMatrix(r_forest, classes=classes, cmap='GnBu')

r_forest_cm.fit(X_train, y_train)
```

```
r_forest_cm.score(X_test, y_test)
r_forest_cm.show()
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe94a8d79d0>
```

Classification Report

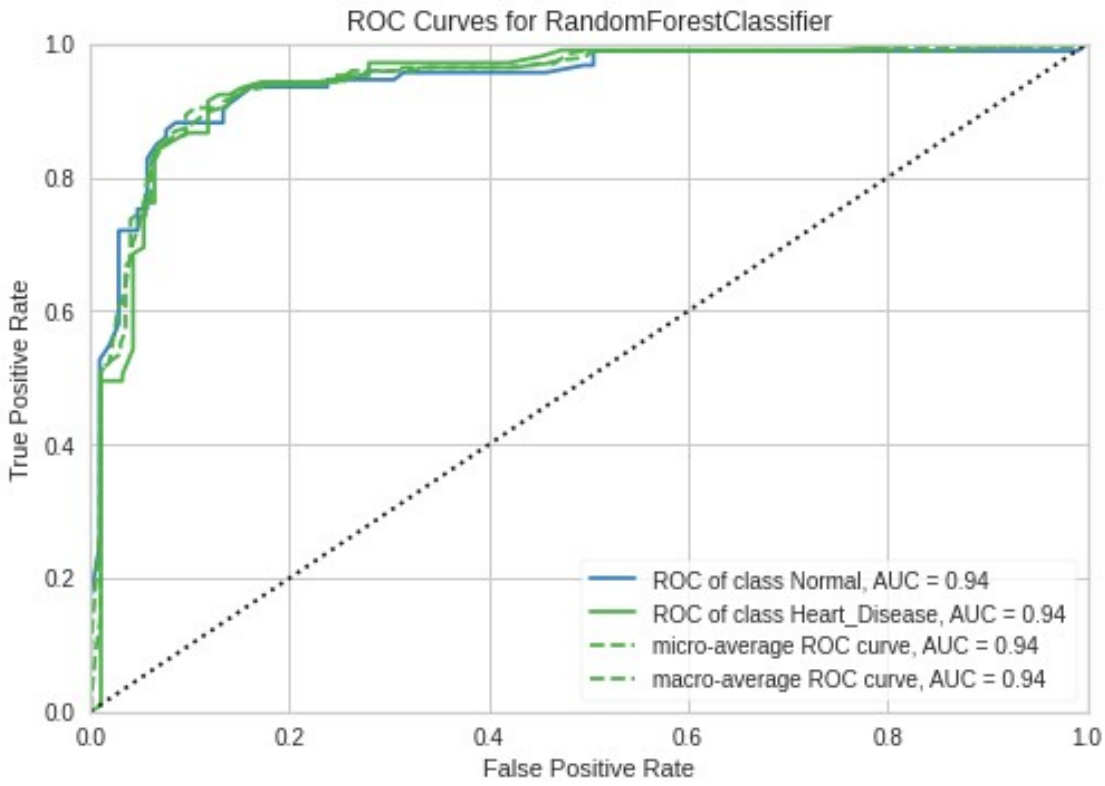
```
print(classification_report(y_test, predicted))
```

```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89        93
           1       0.89      0.92      0.91       105

    accuracy                           0.90       198
   macro avg       0.90      0.90      0.90       198
weighted avg       0.90      0.90      0.90       198
```

ROC Curve

```
visualizer = ROCAUC(r_forest, classes=classes)

set_palette('bold')
```

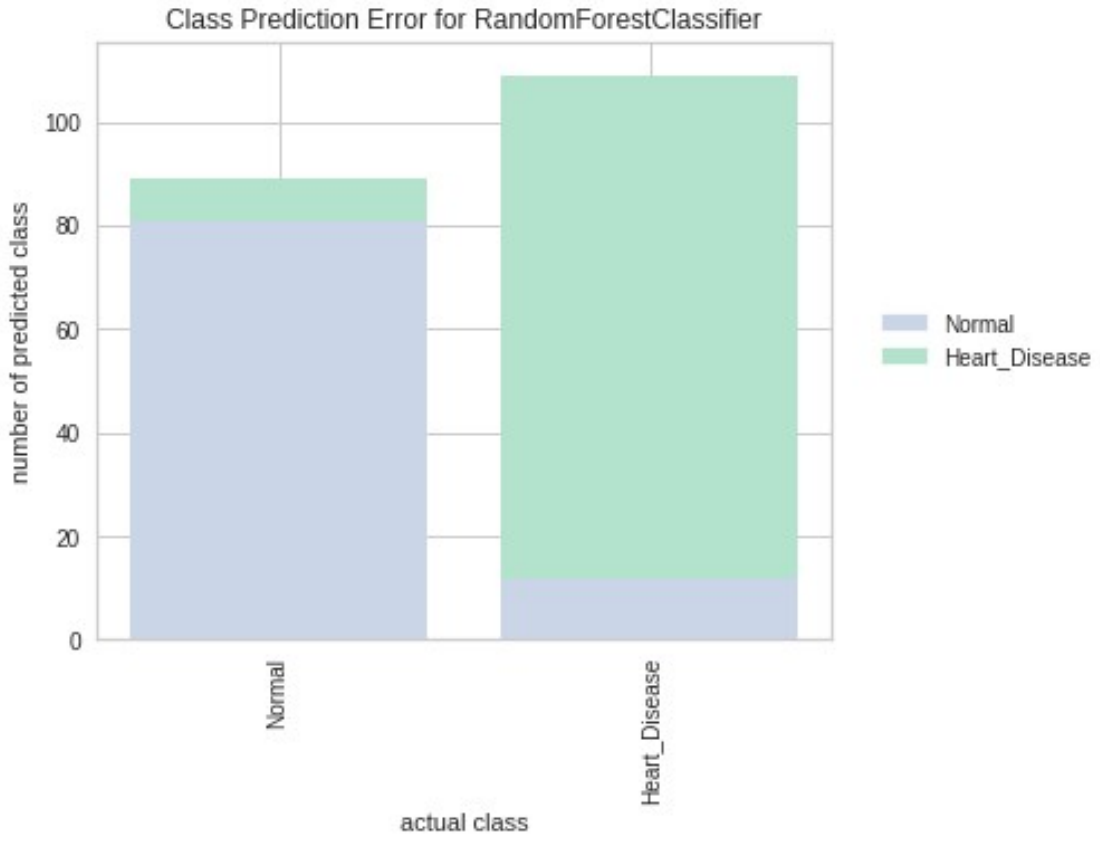
```
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9438bc490>
```

Class Prediction Error

```
visualizer = ClassPredictionError(r_forest, classes=classes)

set_palette('pastel')

visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```




```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe9437c3910>
```

Features Reduction

```
x1 = df.drop(["Age", "Sex", "RestingBP", "MaxHR", "ExerciseAngina",
"ST_Slope","HeartDisease"], axis=1)
y1 = df["HeartDisease"]
x1
```

```
     ChestPainType  Cholesterol  FastingBS  RestingECG  Oldpeak
0                2          289          0           2      0.0
1                3          180          0           2      1.0
2                2          283          0           3      0.0
3                1          214          0           2      1.5
4                3          195          0           2      0.0
..             ...          ...        ...         ...      ...
913              4          264          0           2      1.2
914              1          193          1           2      3.4
915              1          131          0           2      1.2
916              2          236          0           1      0.0
917              3          175          0           2      0.0

[897 rows x 5 columns]
```

Data Scaling with features reduction

```
scala = preprocessing.StandardScaler()
x1= scala.fit(x1).transform(x1)
x1[0:5]
```

```
array([[ 0.22211945,  0.87238586, -0.55116156,  0.01577419, -
0.82934459],
       [ 1.26526683, -0.1653641 , -0.55116156,  0.01577419,
0.10558327],
       [ 0.22211945,  0.81526201, -0.55116156,  1.58793488, -
0.82934459],
       [-0.82102793,  0.15833772, -0.55116156,  0.01577419,  0.5730472
],
       [ 1.26526683, -0.02255447, -0.55116156,  0.01577419, -
0.82934459]])
```

Train / Test Split with features reduction

```
X1_train, X1_test, y1_train, y1_test = train_test_split(x1, y1,
test_size=0.2, random_state=42)
print ('Train set:', X1_train.shape,  y1_train.shape)
print ('Validation set:', X1_test.shape,  y1_test.shape)
```

```
Train set: (717, 5) (717,)
Validation set: (180, 5) (180,)
```

Test which is the best model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
AdaBoostClassifier, GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import PolynomialFeatures
```

```
classifiers = [LogisticRegression(solver='liblinear'),
    KNeighborsClassifier(),
    DecisionTreeClassifier(),
```

```
    RandomForestClassifier(),
    MLPClassifier(),
    AdaBoostClassifier(),
    GaussianNB(),
    SVC(),GaussianProcessClassifier()]

for i in classifiers:
    model = i
    model.fit(X1_train, y1_train)
    y_pred = model.predict(X1_test)
    print('_____________________________________________')
    print(i)
    print('.............................................')
    print('Train',model.score(X1_train, y1_train)*100)
    print('Test',model.score(X1_test, y1_test)*100)
```

```
_____________________________________________
LogisticRegression(solver='liblinear')
.............................................
Train 79.07949790794979
Test 78.33333333333333
_____________________________________________
KNeighborsClassifier()
.............................................
Train 85.21617852161785
Test 80.0
_____________________________________________
DecisionTreeClassifier()
.............................................
Train 98.74476987447699
Test 75.0
_____________________________________________
RandomForestClassifier()
.............................................
Train 98.74476987447699
Test 82.22222222222221

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:696: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization
hasn't converged yet.
  ConvergenceWarning,


_____________________________________________
MLPClassifier()
.............................................
Train 81.72942817294282
Test 77.77777777777779
_____________________________________________
AdaBoostClassifier()
.............................................
```

```
Train 81.72942817294282
Test 81.11111111111111
_________________________________________________
GaussianNB()
.................................................
Train 77.54532775453278
Test 76.66666666666667
_________________________________________________
SVC()
.................................................
Train 82.56624825662483
Test 81.11111111111111
_________________________________________________
GaussianProcessClassifier()
.................................................
Train 83.96094839609484
Test 80.55555555555556
```

Choose Random Forest because it has the best balance between test/training result.

```
import pickle

model_forrest = RandomForestClassifier()
model_forrest.fit(X1_train, y1_train)
with open('modelforrest_pickle', 'wb') as f:
  pickle.dump(model_forrest, f)
with open('model_scaler', 'wb') as f:
  pickle.dump(scala, f)
```