1. Data visualisation :

Our goal here is to visualize the database "heart_failure_clinical_records_dataset.csv" .

```
# reading the data file
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')
# print the first 5 rows of the data
data.head()
```

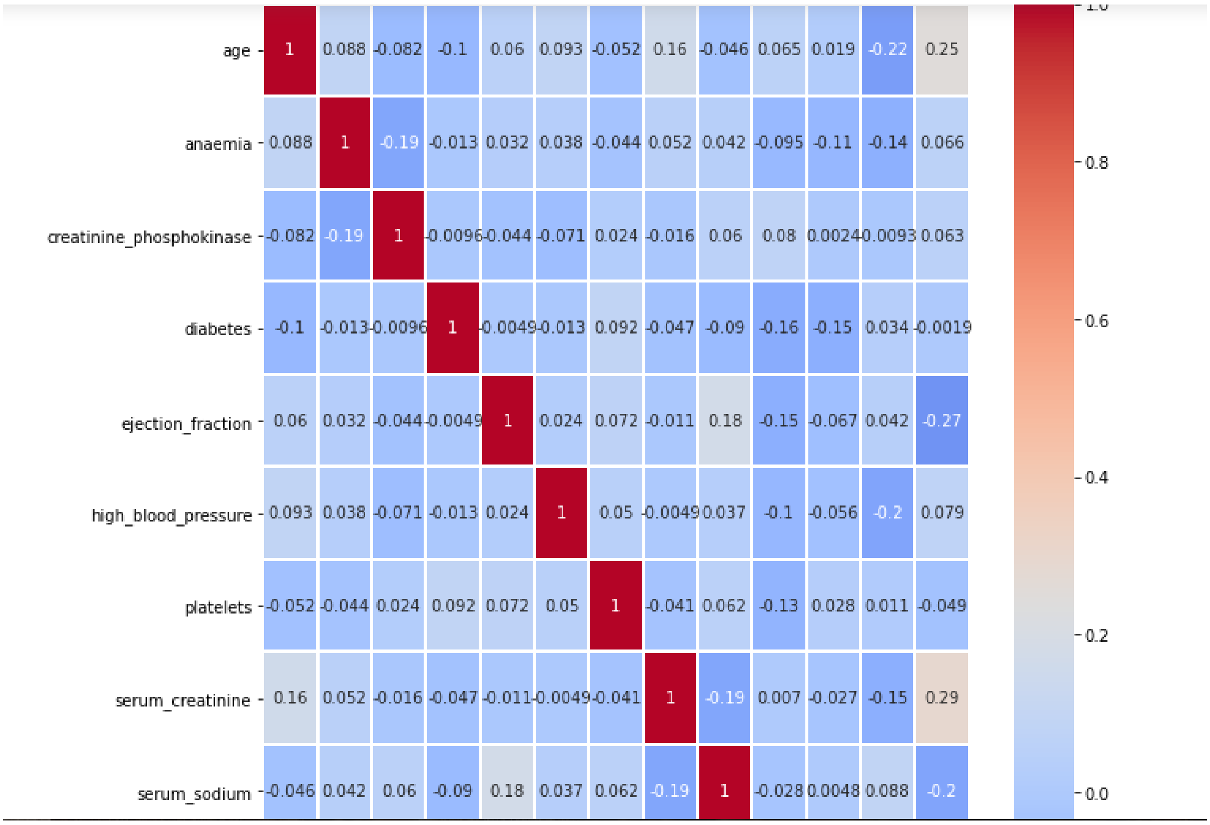
	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8

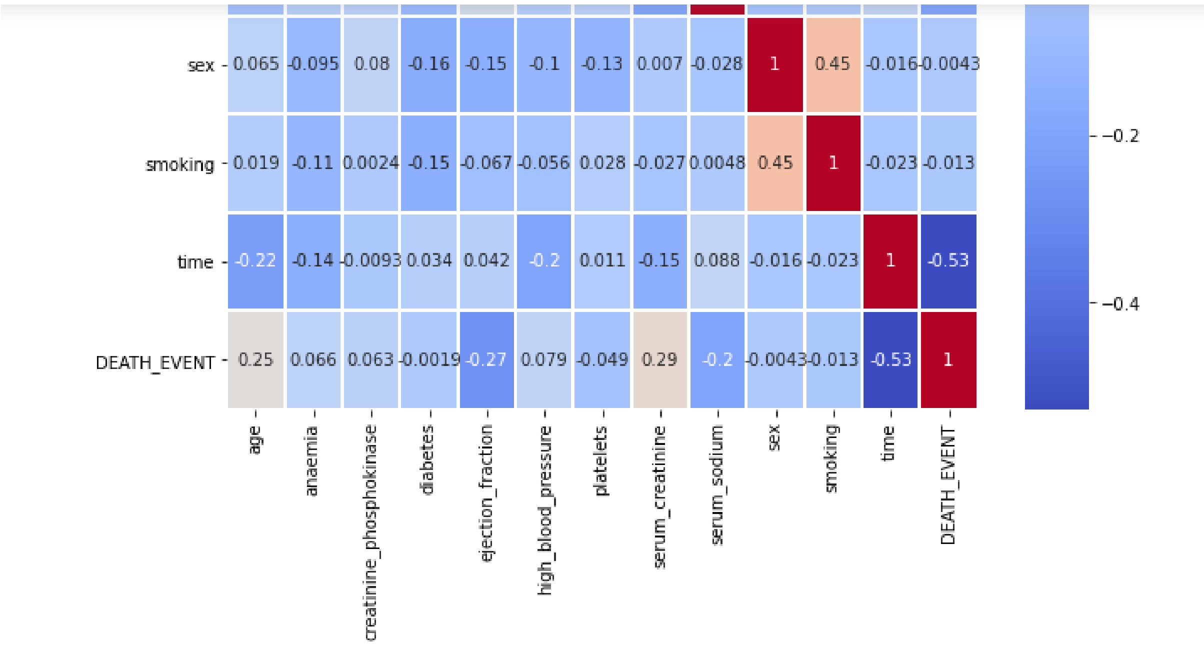
]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

```
data.isnull().sum()
```

```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```





2. Machine Learning applications before data Normalization:

a. Model preparation:

```
data.columns
```

```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

```
X_data = data[['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time']]
y_data = data['DEATH_EVENT']
```

```
X_train,X_test,y_train,y_test = train_test_split(X_data,y_data,test_size=0.3, stratify=y_data)
```

```
X_train,X_val,y_train,y_val = train_test_split(X_train,y_train,test_size=0.3, stratify=y_train)
```

b. Random forest :

```
clf = RandomForestClassifier(max_depth=2, random_state=0)
```

```
clf.fit(X_train,y_train)
```

```
RandomForestClassifier(max_depth=2, random_state=0)
```

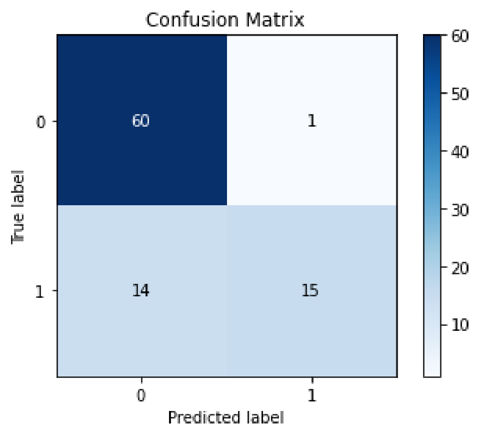
```
#Predict the response for test dataset
predictions = clf.predict(X_test)
predictions
```

```
array([0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1], dtype=int64)
```

```
print(metrics.confusion_matrix(y_test,predictions))
# using sciktplot
skplt.metrics.plot_confusion_matrix(y_test,predictions)
```

```
[[60  1]
 [14 15]]
```

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



```
acc = metrics.accuracy_score(y_test, predictions)
print("Accuracy = %.2f" %(acc))
f1 = metrics.f1_score(y_test, predictions)
print("F1 = %.2f" %(f1))
p = metrics.precision_score(y_test, predictions)
print("Precision = %.2f" %(p))
r = metrics.recall_score(y_test, predictions)
print("Recall = %.2f" %(r))
loss = metrics.log_loss(y_test, predictions)
print("log-loss = %.2f" %(loss))
auc = metrics.roc_auc_score(y_test, predictions)
print("ROC-AUC = %.2f" %(auc))
```

```
Accuracy = 0.83
F1 = 0.67
Precision = 0.94
Recall = 0.52
log-loss = 5.76
ROC-AUC = 0.75
```

c. Adaboost classifier:

```
ad=AdaBoostClassifier(base_estimator=clf)
```

```
ad.fit(X_test,y_test)
```

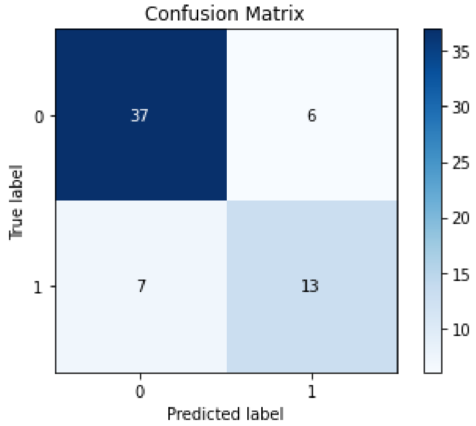
```
AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=2,
                                                         random_state=0))
```

```
y_pred = ad.predict(X_val)
```

```
print(metrics.confusion_matrix(y_val,y_pred))
# using scikiptlot
skplt.metrics.plot_confusion_matrix(y_val,y_pred)
```

```
[[37  6]
 [ 7 13]]
```

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



```
acc = metrics.accuracy_score(y_val,y_pred)
print("Accuracy = %.2f" %(acc))
f1 = metrics.f1_score(y_val,y_pred)
print("F1 = %.2f" %(f1))
p = metrics.precision_score(y_val,y_pred)
print("Precision = %.2f" %(p))
r = metrics.recall_score(y_val,y_pred)
print("Recall = %.2f" %(r))
loss = metrics.log_loss(y_val,y_pred)
print("log-loss = %.2f" %(loss))
auc = metrics.roc_auc_score(y_val,y_pred)
print("ROC-AUC = %.2f" %(auc))
```

```
Accuracy = 0.79
F1 = 0.67
Precision = 0.68
Recall = 0.65
log-loss = 7.13
ROC-AUC = 0.76
```

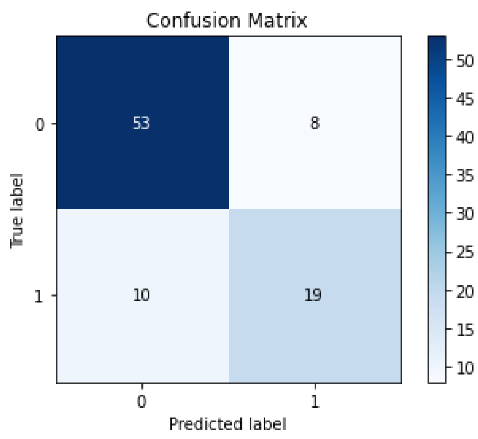
d. SVM :

```
SVM = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
SVM.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = SVM.predict(X_test)
```

```
print(metrics.confusion_matrix(y_test, y_pred))
# using scikiptlot
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

```
[[53  8]
 [10 19]]
```

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.8
Precision: 0.7037037037037037
Recall: 0.6551724137931034
F1 = 0.6785714285714286
```

e. Gradient Boosting Classifier:

```
gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.5, max_depth=3, random_state=0).fit(X_train, y_train)
```

```
#Train the model using the training sets
gb_clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = gb_clf.predict(X_test)
```

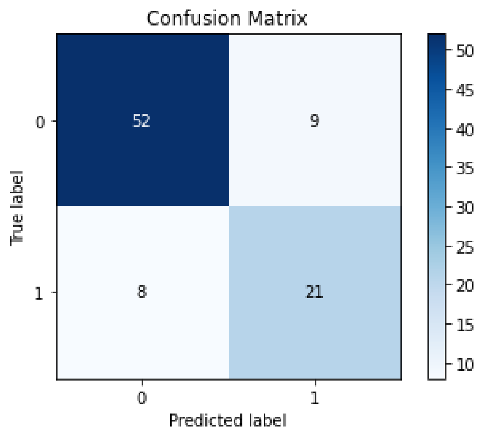
```
print(metrics.confusion_matrix(y_test, y_pred))
# using scikitplot
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

```
[[52  9]
 [ 8 21]]
```

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```

```python
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.8111111111111111
Precision: 0.7
Recall: 0.7241379310344828
F1 = 0.711864406779661
```

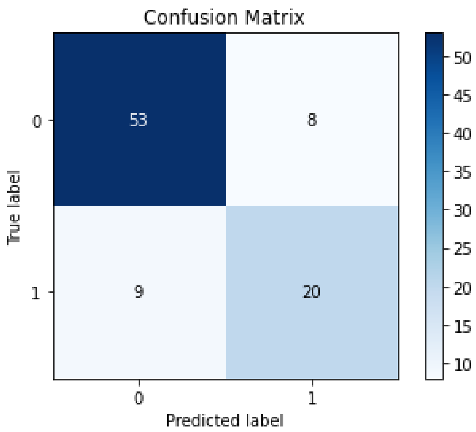
f. Logistic Regression :

```python
# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)
```

```python
print(metrics.confusion_matrix(y_test, y_pred))
# using scikiptlot
skplt.metrics.plot_confusion_matrix(y_test, y_pred)
```

```
[[53  8]
 [ 9 20]]
```

```
<AxesSubplot:title={'center':'Confusion Matrix'}, xlabel='Predicted label', ylabel='True label'>
```



```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.8111111111111111
Precision: 0.7142857142857143
Recall: 0.6896551724137931
F1 = 0.7017543859649122
```

3. Normalisation :

```
from sklearn.preprocessing import StandardScaler
import numpy as np

# 4 samples/observations and 2 variables/features
scaler = StandardScaler()
scaler.fit(X_data)
scaled_data_x=scaler.transform(X_data)
scaled_data_x=pd.DataFrame(scaled_data_x, columns=X_data.columns)
```

```
scaled_data_x
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium
0	1.192945	-0.871105	0.000166	-0.847579	-1.530560	1.359272	1.681648e-02	0.490057	-1.504036
1	-0.491279	-0.871105	7.514640	-0.847579	-0.007077	-0.735688	7.535660e-09	-0.284552	-0.141976
2	0.350833	-0.871105	-0.449939	-0.847579	-1.530560	-0.735688	-1.038073e+00	-0.090900	-1.731046
3	-0.912335	1.147968	-0.486071	-0.847579	-1.530560	-0.735688	-5.464741e-01	0.490057	0.085034
4	0.350833	1.147968	-0.435486	1.179830	-1.530560	-0.735688	6.517986e-01	1.264666	-4.682176
...	...	...	...	...	...	...	...	...	...
294	0.098199	-0.871105	-0.537688	1.179830	-0.007077	1.359272	-1.109765e+00	-0.284552	1.447094
295	-0.491279	-0.871105	1.278215	-0.847579	-0.007077	-0.735688	6.802472e-02	-0.187726	0.539054
296	-1.333392	-0.871105	1.525979	1.179830	1.854958	-0.735688	4.902082e+00	-0.575031	0.312044
297	-1.333392	-0.871105	1.890398	-0.847579	-0.007077	-0.735688	-1.263389e+00	0.005926	0.766064
298	-0.912335	-0.871105	-0.398321	-0.847579	0.585389	-0.735688	1.348231e+00	0.199578	-0.141976

299 rows × 12 columns

```
: X_data=scaled_data_x
```

```
: y_data
```

```
: 0      1
  1      1
  2      1
  3      1
  4      1
        ..
  294    0
  295    0
  296    0
  297    0
  298    0
  Name: DEATH_EVENT, Length: 299, dtype: int64
```

4. Machine learning applications after Normalization:

a. Model preparation :

```
X_train,X_test,y_train,y_test = train_test_split(X_data,y_data,test_size=0.3, stratify=y_data)
```

```
X_train,X_val,y_train,y_val = train_test_split(X_train,y_train,test_size=0.3, stratify=y_train)
```

b. Random forest :

```
acc = metrics.accuracy_score(y_test, predictions)
print("Accuracy = %.2f" %(acc))
f1 = metrics.f1_score(y_test, predictions)
print("F1 = %.2f" %(f1))
p = metrics.precision_score(y_test, predictions)
print("Precision = %.2f" %(p))
r = metrics.recall_score(y_test, predictions)
print("Recall = %.2f" %(r))
loss = metrics.log_loss(y_test, predictions)
print("log-loss = %.2f" %(loss))
auc = metrics.roc_auc_score(y_test, predictions)
print("ROC-AUC = %.2f" %(auc))
```

```
Accuracy = 0.83
F1 = 0.68
Precision = 0.89
Recall = 0.55
log-loss = 5.76
ROC-AUC = 0.76
```

Before :

```
Accuracy = 0.83
F1 = 0.67
Precision = 0.94
Recall = 0.52
log-loss = 5.76
ROC-AUC = 0.75
```

Normalization with Random forest has a negative effect on the evaluation of the module

c. Adaboost classifier:

```
acc = metrics.accuracy_score(y_val,y_pred)
print("Accuracy = %.2f" %(acc))
f1 = metrics.f1_score(y_val,y_pred)
print("F1 = %.2f" %(f1))
p = metrics.precision_score(y_val,y_pred)
print("Precision = %.2f" %(p))
r = metrics.recall_score(y_val,y_pred)
print("Recall = %.2f" %(r))
loss = metrics.log_loss(y_val,y_pred)
print("log-loss = %.2f" %(loss))
auc = metrics.roc_auc_score(y_val,y_pred)
print("ROC-AUC = %.2f" %(auc))
```

```
Accuracy = 0.89
F1 = 0.80
Precision = 0.93
Recall = 0.70
log-loss = 3.84
ROC-AUC = 0.84
```

Before :

```
Accuracy = 0.79
F1 = 0.67
Precision = 0.68
Recall = 0.65
log-loss = 7.13
ROC-AUC = 0.76
```

Normalization with Adaboost classifier has a positive effect on the evaluation of the module and accuracy

d. SVM:

```python
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.8111111111111111
Precision: 0.7
Recall: 0.7241379310344828
F1 = 0.711864406779661
```

Before :

```
Accuracy: 0.8
Precision: 0.7037037037037037
Recall: 0.6551724137931034
F1 = 0.6785714285714286
```

Normalization with SVM has a positive effect on the evaluation of the module and accuracy

e. Gradient Boosting Classifier:

```python
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.8444444444444444
Precision: 0.8
Recall: 0.6896551724137931
F1 = 0.7407407407407408
```

Before :

```
Accuracy: 0.8111111111111111
Precision: 0.7
Recall: 0.7241379310344828
F1 = 0.711864406779661
```

Normalization with Gradient Boosting Classifier has a positive effect on the evaluation of the module and accuracy

f. Logistic Regression :

```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.8222222222222222
Precision: 0.7241379310344828
Recall: 0.7241379310344828
F1 = 0.7241379310344829
```

Before :

```
Accuracy: 0.8111111111111111
Precision: 0.7142857142857143
Recall: 0.6896551724137931
F1 = 0.7017543859649122
```

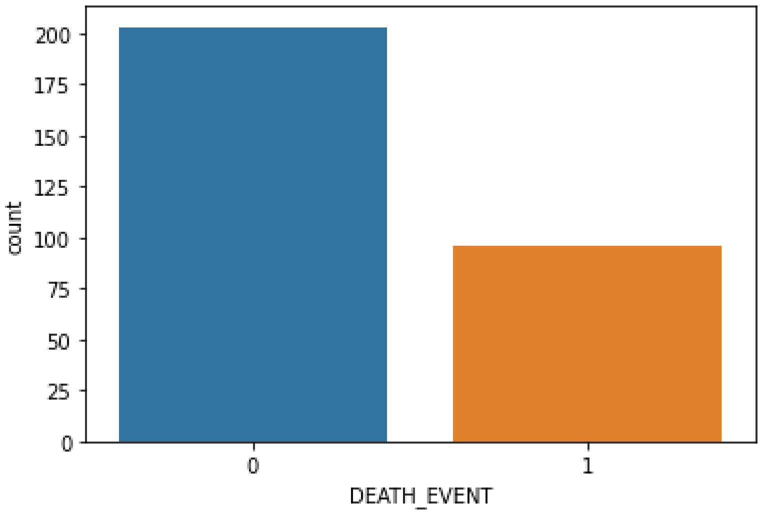
Normalization with Logistic Regression has a positive effect on the evaluation of the module and accuracy

5. Data balancing:

```python
#preparation des donnees
y = y_data
#X (les autres) sont les variables qui précèdent la dernière
X= X_data
```

```python
sb.countplot(x=y,data=X)
```

```
<AxesSubplot:xlabel='DEATH_EVENT', ylabel='count'>
```

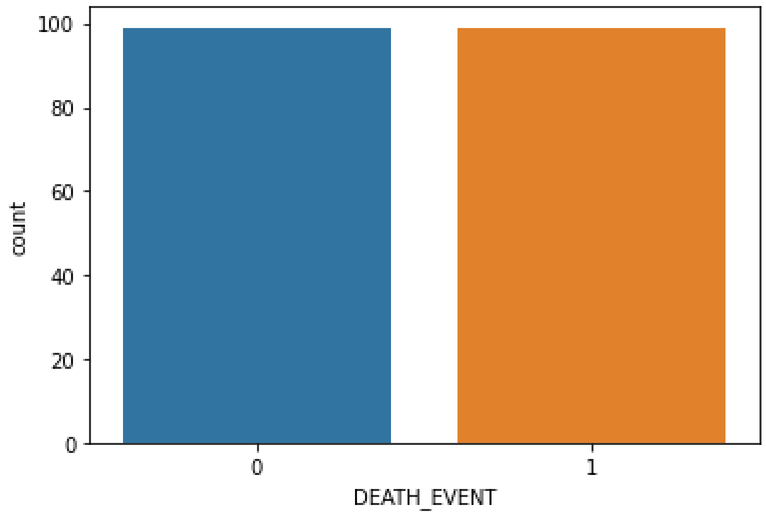



```python
# Division de la bd
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3, stratify=y)
X_train,X_val,y_train,y_val = train_test_split(X_train,y_train,test_size=0.3, stratify=y_train)
# Sur-échantillonnage
rOs = RandomOverSampler()
X_ro, y_ro = rOs.fit_resample(X_train, y_train)
X_rtest, y_rtest = rOs.fit_resample(X_test, y_test)
X_rval, y_rval = rOs.fit_resample(X_val, y_val)
print(X_ro.shape)
print(X_rtest.shape)
print(X_rval.shape)
```

```
(198, 12)
(122, 12)
(86, 12)
```

```python
sb.countplot(x=y_ro,data=X_ro)
```

```
<AxesSubplot:xlabel='DEATH_EVENT', ylabel='count'>
```



6. Machine learning applications after data balancing :

a. model preparation :

```python
X_train, y_train=X_ro, y_ro
X_test, y_test=X_rtest, y_rtest
X_val, y_val=X_rval, y_rval
```

b. Random forest :

```python
acc = metrics.accuracy_score(y_test, predictions)
print("Accuracy = %.2f" %(acc))
f1 = metrics.f1_score(y_test, predictions)
print("F1 = %.2f" %(f1))
p = metrics.precision_score(y_test, predictions)
print("Precision = %.2f" %(p))
r = metrics.recall_score(y_test, predictions)
print("Recall = %.2f" %(r))
loss = metrics.log_loss(y_test, predictions)
print("log-loss = %.2f" %(loss))
auc = metrics.roc_auc_score(y_test, predictions)
print("ROC-AUC = %.2f" %(auc))
```

```
Accuracy = 0.81
F1 = 0.80
Precision = 0.85
Recall = 0.75
log-loss = 6.51
ROC-AUC = 0.81
```

Before :

```
Accuracy = 0.83
F1 = 0.67
Precision = 0.94
Recall = 0.52
log-loss = 5.76
ROC-AUC = 0.75
```

Data balancing with Adaboost classifier has a positive effect on the evaluation of the module and accuracy

c. Adaboost classifier:

```
acc = metrics.accuracy_score(y_val,y_pred)
print("Accuracy = %.2f" %(acc))
f1 = metrics.f1_score(y_val,y_pred)
print("F1 = %.2f" %(f1))
p = metrics.precision_score(y_val,y_pred)
print("Precision = %.2f" %(p))
r = metrics.recall_score(y_val,y_pred)
print("Recall = %.2f" %(r))
loss = metrics.log_loss(y_val,y_pred)
print("log-loss = %.2f" %(loss))
auc = metrics.roc_auc_score(y_val,y_pred)
print("ROC-AUC = %.2f" %(auc))
```

```
Accuracy = 0.83
F1 = 0.82
Precision = 0.85
Recall = 0.79
log-loss = 6.02
ROC-AUC = 0.83
```

Before :

```
Accuracy = 0.89
F1 = 0.80
Precision = 0.93
Recall = 0.70
log-loss = 3.84
ROC-AUC = 0.84
```

Data balancing with Adaboost classifier has a negative effect to accuracy

Data balancing with Adaboost classifier has a positive effect on the evaluation of the module

d. SVM:

```python
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.7950819672131147
Precision: 0.7903225806451613
Recall: 0.8032786885245902
F1 = 0.7967479674796747
```

Before :

```
Accuracy: 0.8111111111111111
Precision: 0.7
Recall: 0.7241379310344828
F1 = 0.711864406779661
```

Data balancing with SVM has a negative effect to accuracy

Data balancing with SVM has a positive effect on the evaluation of the module

e. Gradient Boosting Classifier:

```python
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.7704918032786885
Precision: 0.8367346938775511
Recall: 0.6721311475409836
F1 = 0.7454545454545455
```

Before :

```
Accuracy: 0.8444444444444444
Precision: 0.8
Recall: 0.6896551724137931
F1 = 0.7407407407407408
```

Data balancing with Gradient Boosting Classifier has a negative effect to accuracy

f. Logistic Regression :

```
# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))


print("F1 =" ,metrics.f1_score(y_test,y_pred))
```

```
Accuracy: 0.7622950819672131
Precision: 0.7962962962962963
Recall: 0.7049180327868853
F1 = 0.7478260869565216
```

Before :

```
Accuracy: 0.8222222222222222
Precision: 0.7241379310344828
Recall: 0.7241379310344828
F1 = 0.7241379310344829
```

Data balancing with Logistic Regression has a positive effect on the evaluation of the module

Data balancing with Gradient Boosting Classifier has a negative effect to accuracy