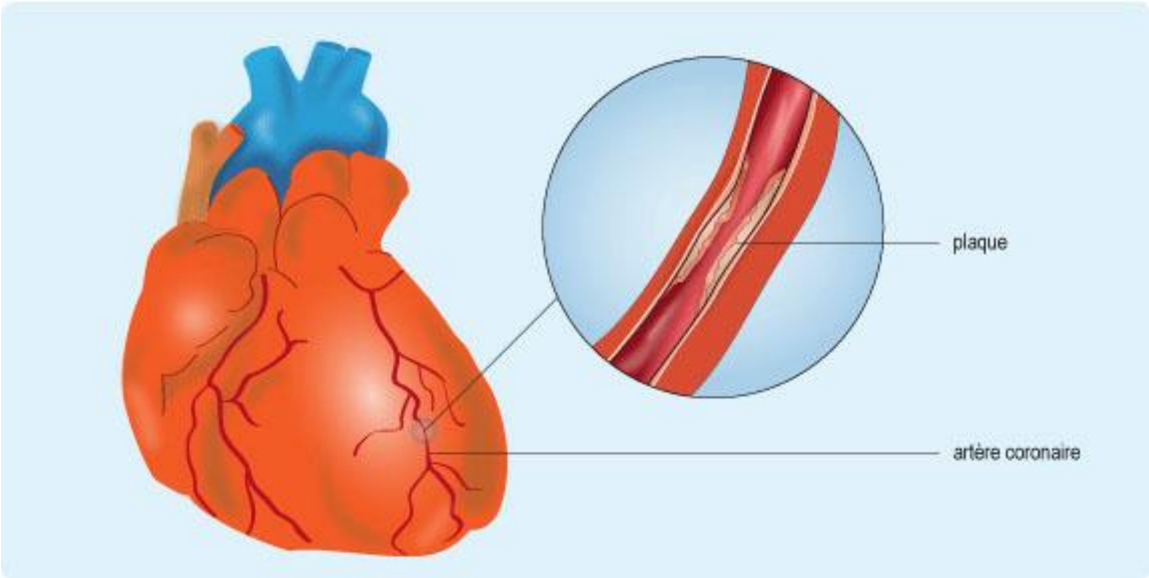
1. Définition de angine de poitrine :



L'angine de poitrine survient lorsque la circulation du sang au cœur est insuffisante en raison d'une maladie du cœur. Lorsqu'il ne reçoit pas assez de sang, le cœur manque d'oxygène, ce qui cause une douleur thoracique.

2. Plan du rapport :

Contenu

3. Data visualisation :

Notre but ici est de visualiser la base de données "probleme card.csv" .

In [2]:
```
cd C:/Users/ASUS/Desktop
```
```
C:\Users\ASUS\Desktop
```

In [3]:
```
# Read in data and display first 5 rows
data = pd.read_csv("probleme card.csv")
data.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [7]:
```
data.describe()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0C
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.31
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.61
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.0C
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.0C
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.0C
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.0C
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.0C

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

```
: age         0
  sex         0
  cp          0
  trestbps    0
  chol        0
  fbs         0
  restecg     0
  thalach     0
  exang       0
  oldpeak     0
  slope       0
  ca          0
  thal        0
  target      0
  dtype: int64
```

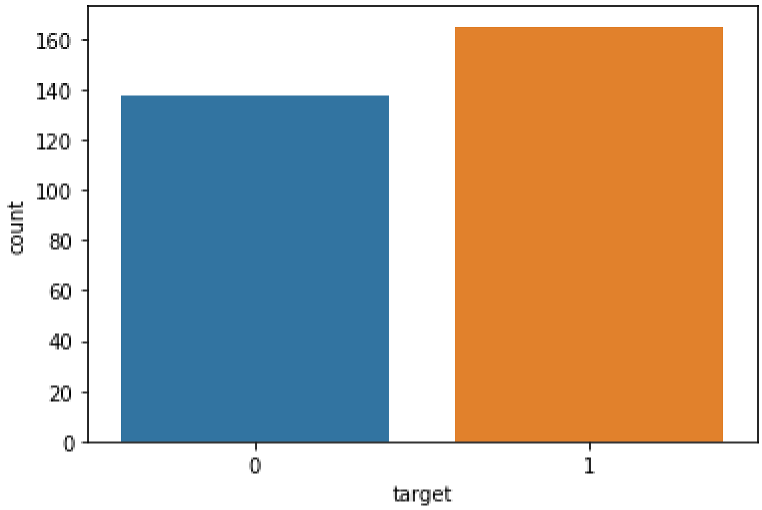
Étudier l'équilibrage de la targette « target » :

```
In [4]: data['target'].value_counts()
```

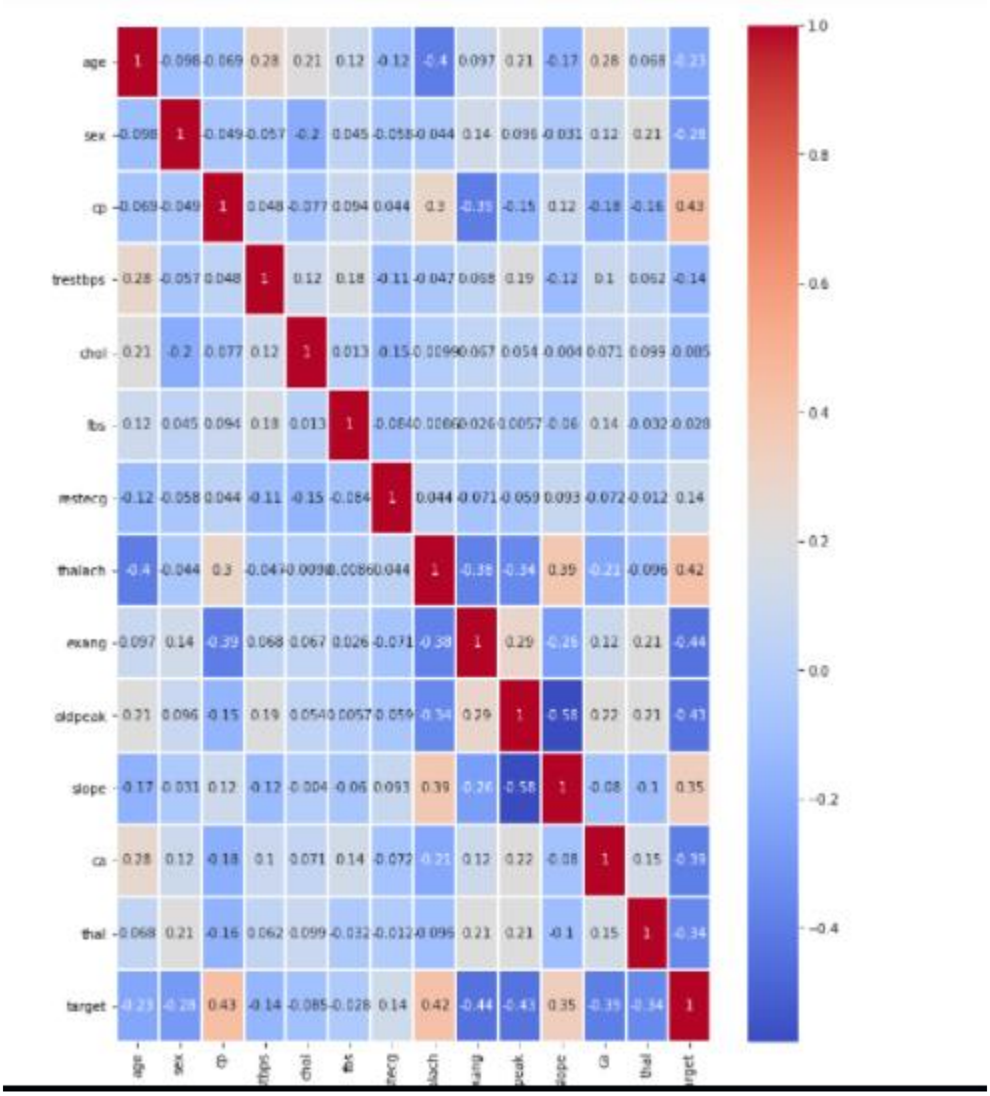
```
Out[4]: 1    165
        0    138
        Name: target, dtype: int64
```

```
In [5]: sb.countplot(x='target',data=data)
```

```
Out[5]: <AxesSubplot:xlabel='target', ylabel='count'>
```



Étudier la corrélation :



4. Utilisation des algorithmes de Machine Learning avant l'ingénierie des données :

a. Model préparation :

```
In [10]: data.columns
```

```
Out[10]: Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
               'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
              dtype='object')
```

```
In [11]: X_data = data[['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
               'exang', 'oldpeak', 'slope', 'ca', 'thal']]
         y_data = data['target']
```

```
In [12]: X_train,X_test,y_train,y_test = train_test_split(X_data,y_data,test_size=0.3, stratify=y_data)
```

```
In [13]: 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:

```
In [14]: clf = RandomForestClassifier(max_depth=2, random_state=0)
```

```
In [15]: clf.fit(X_train,y_train)
```

```
Out[15]: RandomForestClassifier(max_depth=2, random_state=0)
```

Evaluation :

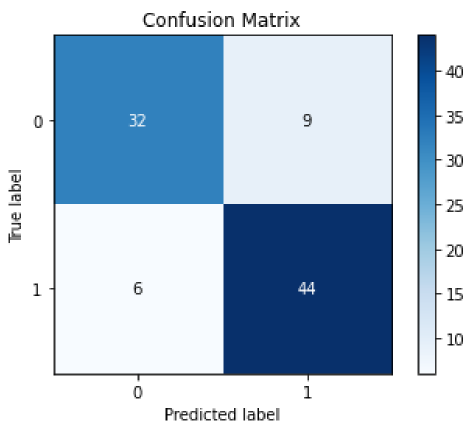
```
#Predict the response for test dataset
predictions = clf.predict(X_test)
predictions
```

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

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

```
[[32  9]
 [ 6 44]]
```

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



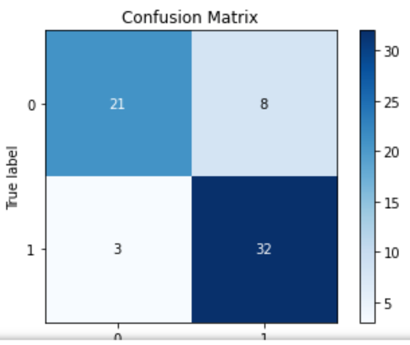
```
: acc = metrics.accuracy_score(y_test, predictions)
  print("Accuracy = %.2f" %(acc))
  f1 = metrics.f1_score(y_test, predictions)
  print("F1 = %.2f" %(f1))
  p = metrics.precision_score(y_test, predictions)
  print("Precision = %.2f" %(p))
  r = metrics.recall_score(y_test, predictions)
  print("Recall = %.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 = 0.84
  F1 = 0.85
  Precision = 0.83
  Recall = 0.88
  log-loss = 5.69
  ROC-AUC = 0.83
```

c. Adaboost classifier :

```
In [23]: ad=AdaBoostClassifier(base_estimator=clf)

In [24]: ad.fit(X_test,y_test)

Out[24]: AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=2,
                                                                  random_state=0))
```

Evaluation :

```
In [25]: y_pred = ad.predict(X_val)

In [27]: print(metrics.confusion_matrix(y_val,y_pred))
         # using scikiptlot
         skplt.metrics.plot_confusion_matrix(y_val,y_pred)

         [[21  8]
          [ 3 32]]

Out[27]: <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 = 0.83
F1 = 0.85
Precision = 0.80
Recall = 0.91
log-loss = 5.94
ROC-AUC = 0.82
```

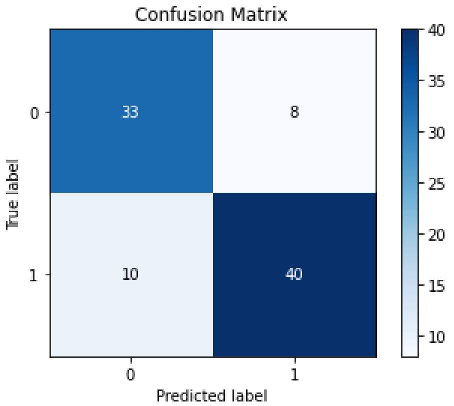
d. SVM:

]:
```
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)
```

```
[[33  8]
 [10 40]]
```

:
```
<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))
```

```
Accuracy: 0.8021978021978022
Precision: 0.8333333333333334
Recall: 0.8
```

e. Gradient Boosting Classifier:

```
In [32]: gb_clf = GradientBoostingClassifier(n_estimators=60, learning_rate=0.09, max_depth=3, random_state=0).fit(X_train, y_train)

In [33]: gb_clf.score(X_test, y_test)

Out[33]: 0.8351648351648352
```

5. Data engineering:

Le but est de transformer une base de données en une base de données Barnier de 0 ou 1.

Le problème est « comment divise chaque variable numérique dans notre base de donnée? »

C'est pour cela en utilise :

_ La logique du flow (Exemples : Age)

```
data['AgeBand'] = pd.cut(data['age'], 5)
data[['AgeBand', 'target']].groupby(['AgeBand'], as_index=False).mean().sort_values(by='AgeBand', ascending=True)
```

	AgeBand	target
0	(28.952, 38.6]	0.750000
1	(38.6, 48.2]	0.704225
2	(48.2, 57.8]	0.577320
3	(57.8, 67.4]	0.377358
4	(67.4, 77.0]	0.588235

```
data.loc[ data['age'] < 38.6, 'age_group'] = 1
data.loc[(data['age'] >= 38.6) & (data['age'] < 48.2), 'age_group'] = 2
data.loc[(data['age'] >= 48.2) & (data['age'] < 57.8), 'age_group'] = 3
data.loc[(data['age'] >= 57.8) & (data['age'] < 67.4), 'age_group'] = 4
data.loc[ data['age'] >= 67.4, 'age_group'] = 5
data['age_group'].astype('int')
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	AgeBand	age_group
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1	(57.8, 67.4]	4.0
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1	(28.952, 38.6]	1.0
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1	(38.6, 48.2]	2.0
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1	(48.2, 57.8]	3.0
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1	(48.2, 57.8]	3.0

```
#drop age and AgeBand
data.drop(['age','AgeBand'] , axis= 1,inplace=True)
```

```
#change age_group to be dummy column
data = pd.get_dummies(data, columns = ['age_group'], prefix="AgeGrp")
data.head()
```

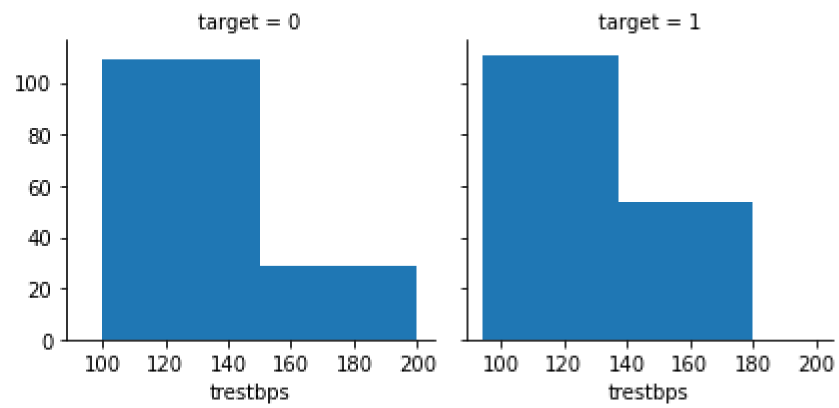
	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	AgeGrp_1.0	AgeGrp_2.0	AgeGrp_3.0	AgeGrp_4.0	AgeGrp_5.0
0	1	3	145	233	1	0	150	0	2.3	0	0	1	1	0	0	0	1	0
1	1	2	130	250	0	1	187	0	3.5	0	0	2	1	1	0	0	0	0
2	0	1	130	204	0	0	172	0	1.4	2	0	2	1	0	1	0	0	0
3	1	1	120	236	0	1	178	0	0.8	2	0	2	1	0	0	1	0	0
4	0	0	120	354	0	1	163	1	0.6	2	0	2	1	0	0	1	0	0

```
data = pd.get_dummies(data,columns = ['sex'], prefix="Sex")
```

_ Histogramme (Exemples : trestbps)

```
#trestbps
g= sb.FacetGrid(data,col='target')
g.map(plt.hist,'trestbps',bins=2)
```

```
<seaborn.axisgrid.FacetGrid at 0x1f63ec5e730>
```



On prend deux ensembles, parce que 2 est mieux que 3 et que 1 ou niveaux de perturbation.

```python
data['TrestbpsBand'] = pd.cut(data['trestbps'], 2)
data[['TrestbpsBand', 'target']].groupby(['TrestbpsBand'], as_index=False).mean().sort_values(by='TrestbpsBand', ascending=True)
```

	TrestbpsBand	target
0	(93.894, 147.0]	0.568000
1	(147.0, 200.0]	0.433962

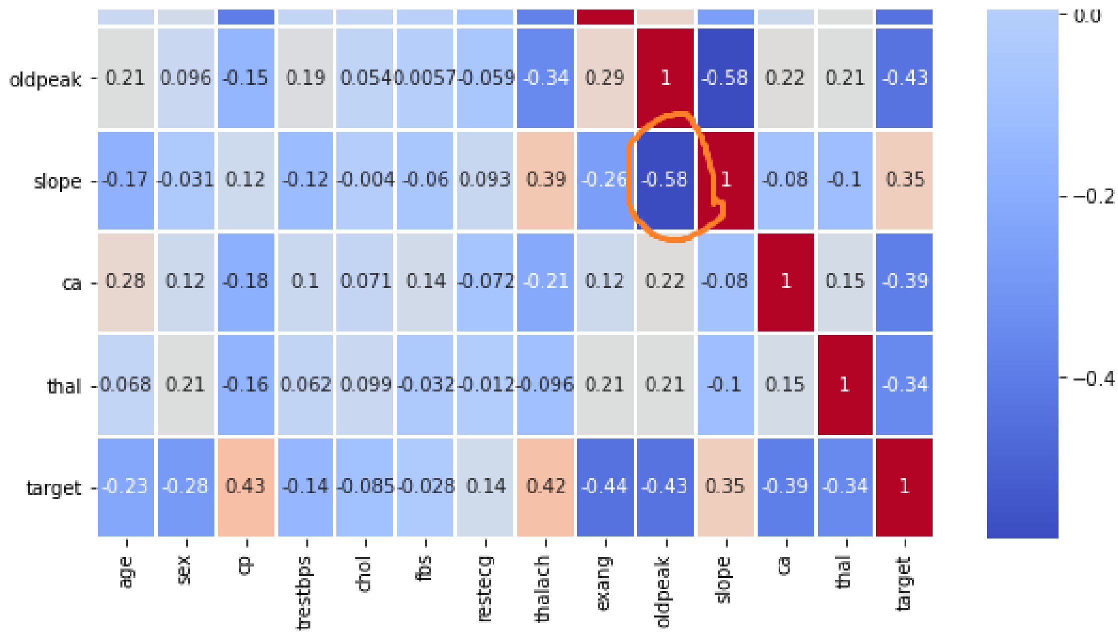
```python
data.loc[ data['trestbps'] <= 147.0, 'trestbps_group'] = 1
data.loc[ data['trestbps'] > 147.0, 'trestbps_group'] = 2
data['trestbps_group'].astype('int')
data.head()
```

```python
#drop age and AgeBand
data.drop(['trestbps','TrestbpsBand'] , axis= 1,inplace=True)
```

```python
#change trestbps_group to be dummy column
data = pd.get_dummies(data, columns = ['trestbps_group'], prefix="TrestbpsGrp")
data.head()
```

	chol	fbs	thalach	exang	oldpeak	target	AgeGrp_1.0	AgeGrp_2.0	AgeGrp_3.0	AgeGrp_4.0	...	Ca_1	Ca_2	Ca_3	Ca_4	Thal_0	Thal_1	Thal_2	Thal_3
0	233	1	150	0	2.3	1	0	0	0	1	...	0	0	0	0	0	1	0	0
1	250	0	187	0	3.5	1	1	0	0	0	...	0	0	0	0	0	0	1	0
2	204	0	172	0	1.4	1	0	1	0	0	...	0	0	0	0	0	0	1	0
3	236	0	178	0	0.8	1	0	0	1	0	...	0	0	0	0	0	0	1	0
4	354	0	163	1	0.6	1	0	0	1	0	...	0	0	0	0	0	0	1	0

_ La corrélation si la corrélation plus que 0.4.



Le choix du même nombre

6. Utilisation des algorithmes de Machine Learning après l'ingénierie des données :

a. Data Préparation:

```
: data.columns
```

```
: Index(['fbs', 'exang', 'target', 'AgeGrp_1.0', 'AgeGrp_2.0', 'AgeGrp_3.0',
         'AgeGrp_4.0', 'AgeGrp_5.0', 'Sex_0', 'Sex_1', 'Restecg_0', 'Restecg_1',
         'Restecg_2', 'CP_0', 'CP_1', 'CP_2', 'CP_3', 'Slope_0', 'Slope_1',
         'Slope_2', 'Ca_0', 'Ca_1', 'Ca_2', 'Ca_3', 'Ca_4', 'Thal_0', 'Thal_1',
         'Thal_2', 'Thal_3', 'TrestbpsGrp_1.0', 'TrestbpsGrp_2.0', 'CholGrp_1.0',
         'CholGrp_2.0', 'CholGrp_3.0', 'CholGrp_4.0', 'ThalachGrp_1.0',
         'ThalachGrp_2.0', 'ThalachGrp_3.0', 'ThalachGrp_4.0', 'ThalachGrp_5.0',
         'OldpeakGrp_1.0', 'OldpeakGrp_2.0', 'OldpeakGrp_3.0'],
        dtype='object')
```

```
dtype='object')
```

```
]: X_data = data[['fbs', 'exang', 'AgeGrp_1.0', 'AgeGrp_2.0', 'AgeGrp_3.0',
        'AgeGrp_4.0', 'AgeGrp_5.0', 'Sex_0', 'Sex_1', 'Restecg_0', 'Restecg_1',
        'Restecg_2', 'CP_0', 'CP_1', 'CP_2', 'CP_3', 'Slope_0', 'Slope_1',
        'Slope_2', 'Ca_0', 'Ca_1', 'Ca_2', 'Ca_3', 'Ca_4', 'Thal_0', 'Thal_1',
        'Thal_2', 'Thal_3', 'TrestbpsGrp_1.0', 'TrestbpsGrp_2.0', 'CholGrp_1.0',
        'CholGrp_2.0', 'CholGrp_3.0', 'CholGrp_4.0', 'ThalachGrp_1.0',
        'ThalachGrp_2.0', 'ThalachGrp_3.0', 'ThalachGrp_4.0', 'ThalachGrp_5.0',
        'OldpeakGrp_1.0', 'OldpeakGrp_2.0', 'OldpeakGrp_3.0']]
   y_data = data['target']
```

```
]: X_data
```

]:

	fbs	exang	AgeGrp_1.0	AgeGrp_2.0	AgeGrp_3.0	AgeGrp_4.0	AgeGrp_5.0	Sex_0	Sex_1	Restecg_0	...	CholGrp_3.0	CholGrp_4.0	ThalachGrp_1.0	Tha
0	1	0	0	0	0	1	0	0	1	1	...	0	0	0	
1	0	0	1	0	0	0	0	0	1	0	...	0	0	0	
2	0	0	0	1	0	0	0	1	0	1	...	0	0	0	
3	0	0	0	0	1	0	0	0	1	0	...	0	0	0	

```
: y_data
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

```
: 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 classifier :

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

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

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

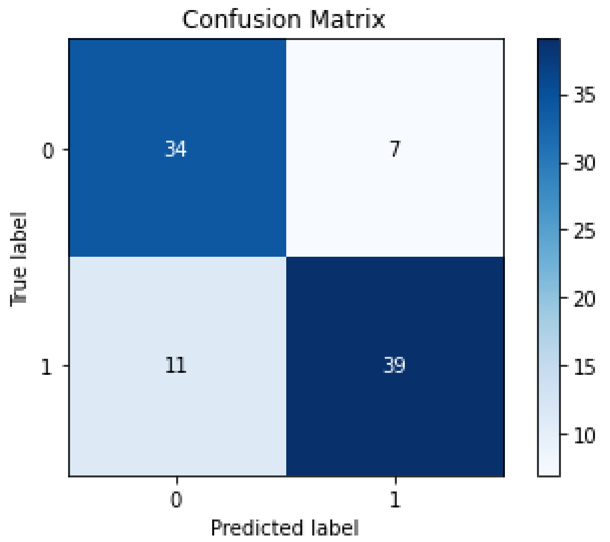
```
#Predict the response for test dataset
predictions = clf.predict(X_test)
predictions
```

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

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

```
[[34  7]
 [11 39]]
```

```
<AxesSubplot:title={'center':'Confusion Matrix'}, x]
```



```
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 = 0.80
F1 = 0.81
Precision = 0.85
Recall = 0.78
log-loss = 6.83
ROC-AUC = 0.80
```

c. Adaboost classifier:

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

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

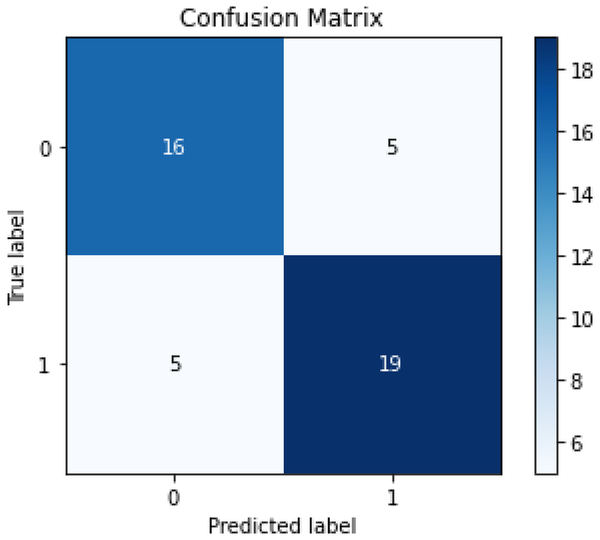
```
AdaBoostClassifier(base_estimator=RandomForestClassifier(max_depth=2,
                                                         random_state=0))
```

```
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)
```

```
[[16  5]
 [ 5 19]]
```

<AxesSubplot:title={'center':'Confusion Matrix'}>




```
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 = 0.78
F1 = 0.79
Precision = 0.79
Recall = 0.79
log-loss = 7.68
ROC-AUC = 0.78
```

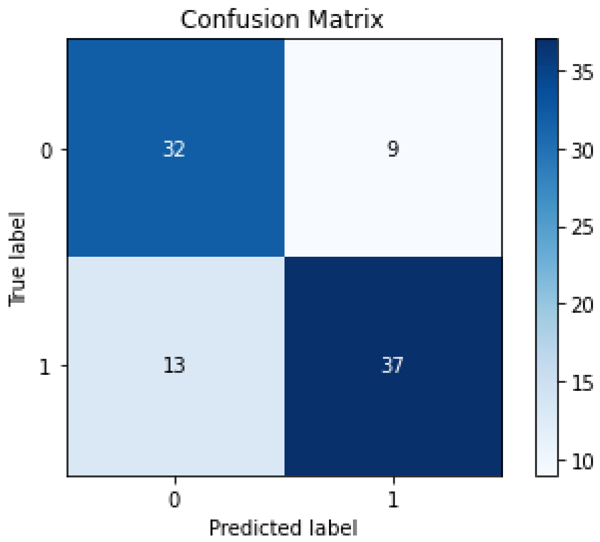
d. SVM:

```
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)
```

```
[[32  9]
 [13 37]]
```

<AxesSubplot:title={'center':'Confusion Mat




```
# 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))
```

```
Accuracy: 0.7582417582417582
Precision: 0.8043478260869565
Recall: 0.74
```

e. gradient boosting classifier:

```
gb_clf = GradientBoostingClassifier(n_estimators=60, learning_rate=0.09, max_depth=3, random_state=0).fit(X_train, y_train)
```

```
gb_clf.score(X_test, y_test)
```

```
0.7582417582417582
```