In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
# Data loading and Halting
df = pd.read_csv('C:/Users/Dhaval_Patel/Desktop/project file/heart.csv')
```

In [3]:
```
df.describe()
```

Out[3]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
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	303.000000	303.000000
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	2.313531	0.544554
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	0.612277	0.498835
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	0.000000	0.000000
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	2.000000	0.000000
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	2.000000	1.000000
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	3.000000	1.000000
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	3.000000	1.000000

In [4]:
```
# missing data check
df.isnull().sum()
```

Out[4]:
```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

In [5]:
```
df.dtypes
```

Out[5]:
```
age           int64
sex           int64
cp            int64
trtbps        int64
chol          int64
fbs           int64
restecg       int64
thalachh      int64
exng          int64
oldpeak     float64
slp           int64
caa           int64
thall         int64
output        int64
dtype: object
```

In [6]:
```
#removing duplicate rows

print('Duplicate Rows count :',df.duplicated().sum())
```

```
Duplicate Rows count : 1
```

In [7]:
```
df = df.drop_duplicates(keep="first")
```

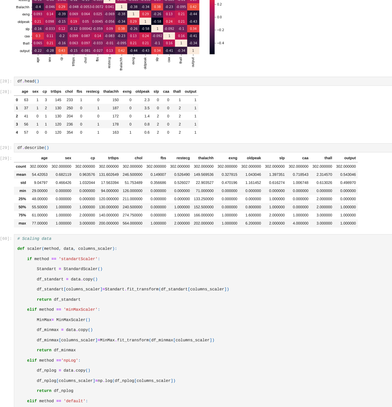
In [8]:
```
print('Duplicate Rows count :',df.duplicated().sum())
```

```
Duplicate Rows count : 0
```

In [9]:
```
#correlation Heatmap

plt.figure(figsize=(12,6))
sns.heatmap(df.corr(),annot=True)
```

Out[9]:
```
<AxesSubplot:>
```



In [10]:
```
df.head()
```

Out[10]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
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 [11]:
```
df.describe()
```

Out[11]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
count	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	0.526490	149.569536	0.327815	1.043046	1.397351	0.718543	2.314570	0.543046
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356588	0.526027	22.903527	0.470236	1.161452	0.616274	1.006726	0.613029	0.498970
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.250000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	152.500000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

In [12]:
```
# Scaling data

def scaler(method, data, columns_scaler):

    if method == 'standardScaler':

        Standard = StandardScaler()

        df_standard = data.copy()

        df_standard[columns_scaler]=Standard.fit_transform(df_standard[columns_scaler])

        return df_standard

    elif method == 'minMaxScaler':

        MinMax= MinMaxScaler()

        df_minmax = data.copy()

        df_minmax[columns_scaler]=MinMax.fit_transform(df_minmax[columns_scaler])

        return df_minmax

    elif method =='nplog':

        df_nplog = data.copy()

        df_nplog[columns_scaler]=np.log(df_nplog[columns_scaler])

        return df_nplog

    elif method == 'default':

        return data
```

In [13]:
```
method = 'minMaxScaler'
data = df
columns_scaler = ['age','trtbps', 'chol', 'thalachh']

df_scaler = scaler(method, data, columns_scaler)

df_scaler.head()
```

Out[13]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	0.708333	1	3	0.481132	0.244292	1	0	0.603053	0	2.3	0	0	1	1
1	0.166667	1	2	0.339623	0.283105	0	1	0.885496	0	3.5	0	0	2	1
2	0.250000	0	1	0.339623	0.178082	0	0	0.770992	0	1.4	2	0	2	1
3	0.562500	1	1	0.245283	0.251142	0	1	0.816794	0	0.8	2	0	2	1
4	0.583333	0	0	0.245283	0.520548	0	1	0.702290	1	0.6	2	0	2	1

In [14]:
```
def encoder(method, dataframe, columns_label, columns_onehot):

    if method == 'labelEncoder':

        df_lbl = dataframe.copy()

        for col in columns_label:
            lbl = LabelEncoder()
            label.fit(list(dataframe[col].values))
            df_lbl[col] = label.transform(df_lbl[col].values)

        return df_lbl

    elif method == 'oneHotEncoder':

        df_oh = dataframe.copy()

        df_oh= pd.get_dummies(data = df_oh, prefix = 'OHE', prefix_sep='_',
                              columns=columns_onehot,
                              drop_first=True,
                              dtype='int8')

        return df_oh

    elif method == 'default':

        return dataframe
```

Training and prediction WITHOUT Hyperparameter Tuning

In [15]:
```
X = df_scaler.drop('output',axis=1)
y = df_scaler['output']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=0)

print('Train : ', X_train.shape, y_train.shape )
print('Test : ', X_test.shape, y_test.shape )
```

```
Train :  (211, 13) (211,)
Test :  (91, 13) (91,)
```

In [16]:
```
RandomForestClassifier=RandomForestClassifier(random_state = 42)

RandomForestClassifier.fit(X_train, y_train)

train_pred = RandomForestClassifier.predict(X_train)
test_pred = RandomForestClassifier.predict(X_test)

print('Train Accuracy Score :', accuracy_score(y_train,train_pred))

print('Test Accuracy Score :', accuracy_score(y_test, test_pred))
```

```
Train Accuracy Score :  1.0
Test Accuracy Score :  0.8131868131868132
```

In [17]:
```
Hyperparameter Tuning
```

In [18]:
```
def classifier_gridsearch(param_grid,data, model_params, func_input):
    lasts={}
    model_params=model_params
    for params in param_grid.data:

        result = {}
        result['encoder'] = params['encoder']
        result['scaler'] = params['scaler']
        result['random_state'] = params['random_state']
        result['test_size'] = params['test_size']

        data = encoder(method=params['encoder'], dataframe = func_input['data'], columns_label = func_input['columns_label'], columns_onehot = func_input['columns_onehot'])
        data = scaler(params['scaler'], data,func_input['columns_scaler'])

        X = data.drop(func_input['output'],axis=1)
        y = data[func_input['output']].values.reshape(-1,)
        X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=params['test_size'],random_state=params['random_state'])

        for model_name, mp in model_params.items():
            reset()
            reservoir()
            gs = GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
            clf.fit(X_train, y_train)
            pred = clf.predict(X_test)
            result['model_name'] = model_name
            result['best_score']=clf.best_score_
            result['best_estimator']=clf.best_estimator_
            result['best_params']=clf.best_params_
            lasts.append(result)
    result = pd.DataFrame(lasts, columns=['encoder','scaler','random_state','test_size','model','best_score','test_score','best_params'])

    return result
```

In [19]:
```
from sklearn.ensemble import RandomForestClassifier

params_grid_data = {
    'encoder': ['default'],
    'scaler': ['standardScaler', 'minMaxScaler', 'nplog', 'default'],
    'random_state' : [16],
    'test_size' : [0.3]
}

params_grid_data = {dict(zip(params_grid_data.keys(), v)) for v in itertools.product(*params_grid_data.values())}

func_input = {
    'columns_label' : [],
    'columns_onehot' : [],
    'columns_scaler' : ['age','trtbps', 'chol', 'thalachh'],
    'output':['output'],
    'data' : df,
}

RFC_params = {
    'RFC' : {
        'model': RandomForestClassifier(),
        'params' : {'criterion' : ['gini', 'entropy'], # 'gini', 'entropy'
                    'max_depth' : [5, 10, 20, 50], # None, 2, 3, 5, 10, 20, 50
                    'max_features' : ['auto', 'sqrt', 'log2'], # 'auto', 'sqrt', 'log2'
                    'n_estimators' : [200, 300, 400], # 200, 250, 300, 400
                    'random_state' : [42]}}
}
```

In [20]:
```
RFC_result=classifier_gridsearch(param_grid_data, RFC_params, func_input)
```

In [21]:
```
RFC_result.head()
```

Out[21]:

	encoder	scaler	random_state	test_size	model	best_score	test_score	best_params
0	default	standardScaler	16	0.3	RFC	0.833807	0.824176	{'criterion': 'entropy', 'max_depth': 20, 'max...
1	default	minMaxScaler	16	0.3	RFC	0.838863	0.824176	{'criterion': 'entropy', 'max_depth': 20, 'max...
2	default	nplog	16	0.3	RFC	0.833807	0.824176	{'criterion': 'entropy', 'max_depth': 20, 'max...
3	default	default	16	0.3	RFC	0.833807	0.824176	{'criterion': 'entropy', 'max_depth': 20, 'max...

Best Hyperparameter and Data Selection for the Model

In [22]:
```
def best_params(model_name, result):

    best_index =np.argmax(result['test_score'])

    best_params = result['best_params'][best_index]
    best_encoder = result['encoder'][best_index]
    best_scaler = result['scaler'][best_index]
    best_random_state = result['random_state'][best_index]
    best_test_size = result['test_size'][best_index]

    print('\nUnbest Model Name :', model_name, '\nbest Params :', best_params, '\nbest Encoder :', best_encoder, '\nbest Scaler :', best_scaler, '\nbest Random State :', best_random_state, '\nbest Test Size :', best_test_size)

    best_params = {
        'params' : best_params,
        'encoder' : best_encoder,
        'scaler' : best_scaler,
        'random_state' : best_random_state,
        'test_size' : best_test_size
    }

    return best_params
```

In [23]:
```
best_params = best_params('RFC Classifier', RFC_result)
```

```
Model Name:  RFC Classifier
best Params :  {'criterion': 'entropy', 'max_depth': 20, 'max_features': 'auto', 'n_estimators': 200, 'random_state': 42}
Best Encoder:  default
Best Scaler:  standardScaler
Best Random State :  16
Best Test Size :  0.3
```

In [24]:
```
def best_data(best_params,rfc, func_input):

    data = encoder(best_params['encoder'], func_input['data'], func_input['columns_label'], func_input['columns_onehot'])
    data = scaler(best_params['scaler'], data, func_input['columns_scaler'])

    X = data.drop(func_input['output'],axis=1)
    y = data[func_input['output']].values.reshape(-1,)
    X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=best_params['test_size'],random_state=best_params['random_state'])

    return X, y, X_train, X_test, y_train, y_test
```

Training and Prediction with The Best Parameter

In [25]:
```
RFC=RandomForestClassifier(criterion = best_params['params']['criterion'], max_depth = best_params['params']['max_depth'], max_features = best_params['params']['max_features'], n_estimators = best_params['params']['n_estimators'], random_state = best_params['params']['random_state'])

X, y, X_train, X_test, y_train, y_test = best_data(best_params, func_input)

RFC.fit(X_train, y_train)

train_pred = RFC.predict(X_train)
test_pred = RFC.predict(X_test)

print(accuracy_score(y_train,train_pred))

print(accuracy_score(y_test, test_pred))
```

```
1.0
0.8241758241758241
```

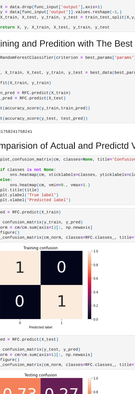
Comparision of Actual and Predictd Values

In [26]:
```
def plot_confusion_matrix(cm, classes=None, title='Confusion matrix'):

    if classes is not None:
        sns.heatmap(cm, xticklabels=classes, yticklabels=classes, vmin=0., vmax=1., annot=True, annot_kws={'size':50})
    else:
        sns.heatmap(cm, vmin=0., vmax=1.)
    plt.title(title)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [27]:
```
y_pred = RFC.predict(X_train)

cm = confusion_matrix(y_train, y_pred)
cm_norm = cm/cm.sum(axis=1)[:, np.newaxis]
plt.figure()
plot_confusion_matrix(cm_norm, classes=RFC.classes_, title='Training confusion')
```



In [28]:
```
y_pred = RFC.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
cm_norm = cm/cm.sum(axis=1)[:, np.newaxis]
plt.figure()
plot_confusion_matrix(cm_norm, classes=RFC.classes_, title='Testing confusion')
```

In [ ]:
```
```