---------- HR----------

In [1]:
```
import pydot
import pandas as pd
import numpy as np
import seaborn as sns
sns.set()
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
import sklearn
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz, DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.exceptions import NotFittedError
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras import optimizers
from keras.wrappers.scikit_learn import KerasClassifier
from IPython.display import display
```

In [2]:
```
# Some useful functions we'll use in this notebook
def display_confusion_matrix(target, prediction, score=None):
    cm = metrics.confusion_matrix(target, prediction)
    plt.figure(figsize=(6,6))
    sns.heatmap(cm, annot=True, fmt=".4f", linewidths=.5, square=True, cmap='Blue
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    if score:
        score_title = 'Accuracy Score: {0}'.format(round(score, 5))
        plt.title(score_title, size = 25)

def visualize_tree(tree, feature_names):
    with open("dt.dot", 'w') as f:
        export_graphviz(tree, out_file=f, feature_names=feature_names)
    try:
        subprocess.check_call(["dot", "-Tpng", "dt.dot", "-o", "dt.png"])
    except:
        exit("Could not run dot, ie graphviz, to produce visualization")

def draw_missing_data_table(df):
    total = df.isnull().sum().sort_values(ascending=False)
    percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False
    missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    return missing_data
```

In [3]:
```
# Create dataframe for training dataset and print five first rows as preview
train_df_raw = pd.read_csv('hr.csv')
train_df_raw.head()
```

Out[3]:

	Condition	HR	RMSSD
0	0.0	61.0	0.061420
1	0.0	61.0	0.061420
2	0.0	64.0	0.049663
3	0.0	60.0	0.052487
4	0.0	61.0	0.051189

In [4]:
```
train_df_raw=train_df_raw.dropna()
```

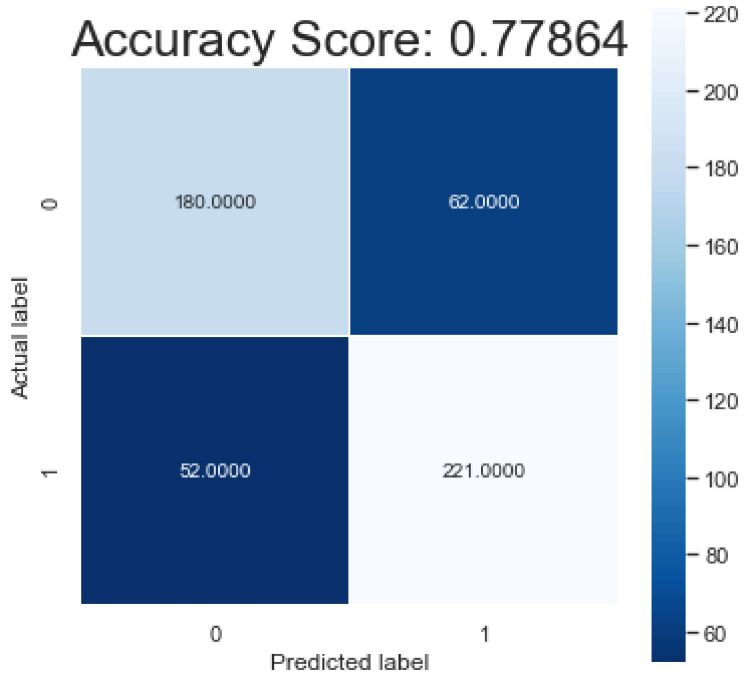
In [5]:
```
# Let's divide the train dataset in two datasets to evaluate perfomance of the mo
train_df = train_df_raw.copy()
X = train_df.drop(['Condition'], 1)
Y = train_df['Condition']
# We scale our data, it is essential for a smooth working of the models
# Scaling means that each columns as a 0 mean and a 1 variance
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X.values), index=X.index, columns=X.columns)
# Split dataset for model testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_s
X_train.head()
```

Out[5]:

	HR	RMSSD
1899	1.221560	0.004979
1808	-0.855015	-0.128974
1619	0.430484	-0.816860
372	-0.261708	-0.062841
1645	0.628253	-0.732941

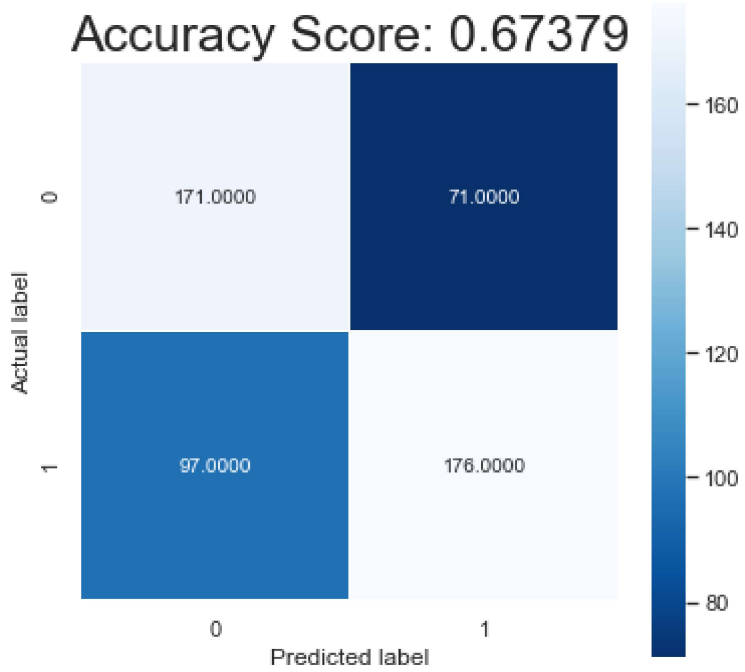
RANDOM FOREST

In [6]:
```
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, Y_train)
rf_prediction = rf.predict(X_test)
score = metrics.accuracy_score(Y_test, rf_prediction)
display_confusion_matrix(Y_test, rf_prediction, score=score)
rf_score=rf_prediction
```



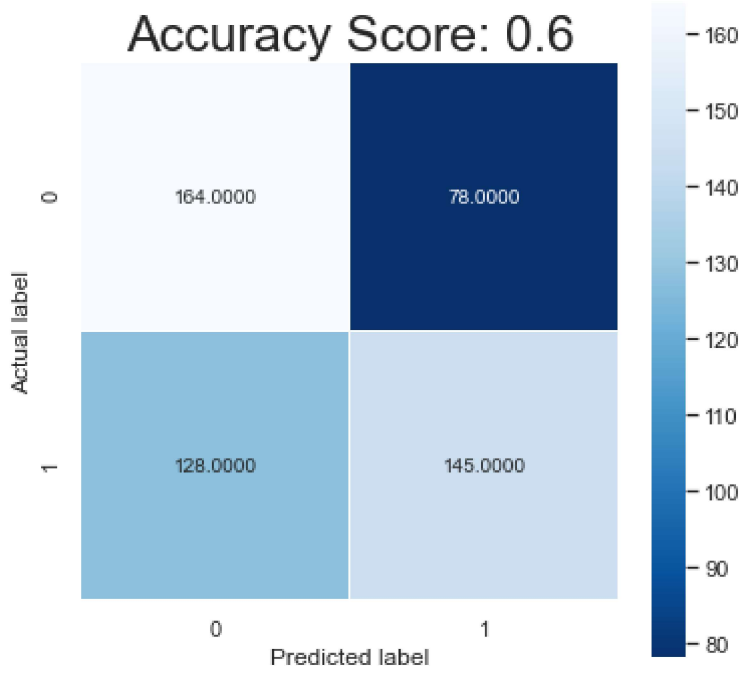
DECISION TREE

In [7]:
```
dt = DecisionTreeClassifier(min_samples_split=15, min_samples_leaf=20, random_sta
dt.fit(X_train, Y_train)
dt_prediction = dt.predict(X_test)

score = metrics.accuracy_score(Y_test, dt_prediction)
display_confusion_matrix(Y_test, dt_prediction, score=score)
```



SVM

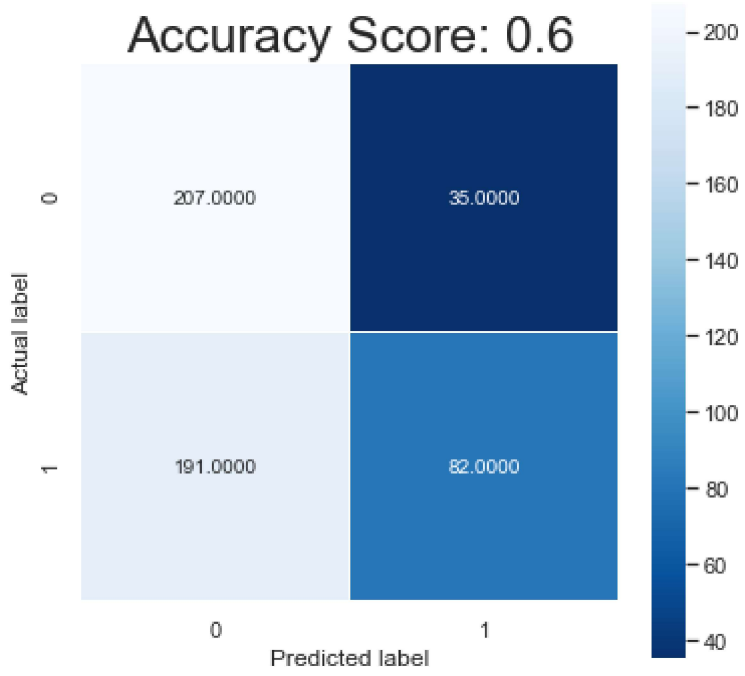
In [8]:
```
svm = SVC(gamma='auto', random_state=42)
svm.fit(X_train, Y_train)
svm_prediction = svm.predict(X_test)

score = metrics.accuracy_score(Y_test, svm_prediction)
display_confusion_matrix(Y_test, svm_prediction, score=score)
```



NAIVE BAYES

In [9]:
```
#train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, Y_train)
NB_y_pred = gnb.predict(X_test)
from sklearn.metrics import accuracy_score
y_pred_train = gnb.predict(X_train)
y_pred_train
# print the scores on training and test set
print('Training set score: {:.4f}'.format(gnb.score(X_train, Y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, Y_test)))
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y_test, NB_y_pred)
display_confusion_matrix(Y_test, NB_y_pred, score=score)
```

```
Training set score: 0.5539
Test set score: 0.5612
```



K NEAREST NEIGHBORS

In [10]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, Y_train)
KNN_y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, KNN_y_pred))
print(classification_report(Y_test, KNN_y_pred))
```

```
[[173  69]
 [ 63 210]]
              precision    recall  f1-score   support

         0.0       0.73      0.71      0.72       242
         1.0       0.75      0.77      0.76       273

    accuracy                           0.74       515
   macro avg       0.74      0.74      0.74       515
weighted avg       0.74      0.74      0.74       515
```

In [11]:
```
print('********************* HR  *********************')
print('SVM ---->',metrics.accuracy_score(Y_test, svm_prediction))
print('RF ---->',metrics.accuracy_score(Y_test, rf_prediction))
print('DT ---->',metrics.accuracy_score(Y_test, dt_prediction))
print('NB ---->',metrics.accuracy_score(Y_test, NB_y_pred))
print('KNN ---->',metrics.accuracy_score(Y_test, KNN_y_pred))
```

```
********************* HR  *********************
SVM ----> 0.6
RF ----> 0.7786407766990291
DT ----> 0.6737864077669903
NB ----> 0.5611650485436893
KNN ----> 0.7436893203883496
```

Confusion Matrix

As we can see from the table above:

True Positive(TP) : Values that the model predicted as yes, and is actually yes.

True Negative(TN) : Values that the model predicted as not, and is actually no.

False Positive(FP): Values that the model predicted as yes, but actually no.

False Negative(FN): Values that the model predicted as no, but actually yes.

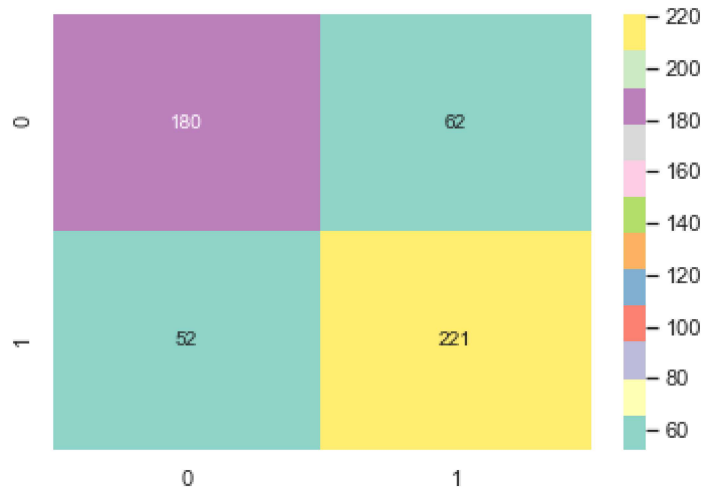
In [12]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, prec
cm = np.array(confusion_matrix(Y_test, rf_prediction, labels=[0,1]))
confusion_mat= pd.DataFrame(cm, index = ['NORMAL', 'STRESSED'],
                            columns =['NORMAL','STRESSED'])
confusion_mat
```

Out[12]:

	NORMAL	STRESSED
NORMAL	180	62
STRESSED	52	221

In [13]:
```
sns.heatmap(cm,annot=True,fmt='g',cmap='Set3')
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2adeb653220>
```



Accuracy_Score

Accuracy_Score is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations

In [14]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test, rf_prediction))
```

```
0.7786407766990291
```

Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

In [15]:
```
print(precision_score(Y_test, rf_prediction))
```

```
0.7809187279151943
```

Recall

Recall also called Sensitivity, is the ratio of positive instances that are correctly detected by the classifier to the all observations in actual class

In [16]:
```
print(recall_score(Y_test, rf_prediction))
```

```
0.8095238095238095
```

Classification Report

In [17]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, rf_prediction))
```

```
              precision    recall  f1-score   support

         0.0       0.78      0.74      0.76       242
         1.0       0.78      0.81      0.79       273

    accuracy                           0.78       515
   macro avg       0.78      0.78      0.78       515
weighted avg       0.78      0.78      0.78       515
```

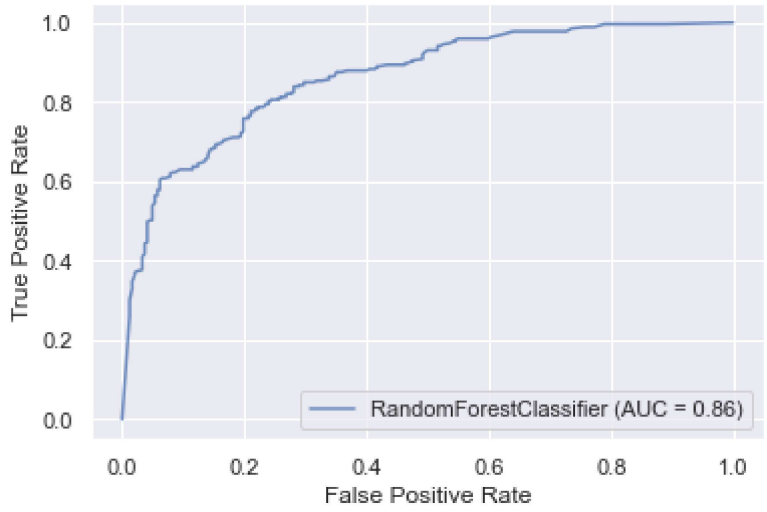
The ROC Curve

Area Under Curve

Area Under Curve is a common way to compare classifiers. A perfect classifier will have ROC AUC equal to 1

Sckit-Learn provides a function to compute the ROC AUC.

In [18]:
```
from sklearn.metrics import roc_auc_score,auc,f1_score
from sklearn.metrics import plot_roc_curve
ax = plt.gca()
rfc_disp = plot_roc_curve(rf, X_test, Y_test, ax=ax, alpha=0.8)
plt.show()
```



K FOLD VALIDATION

In [19]:
```
cv_score_rf1 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=5, n_jobs=
cv_score_rf2 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=10, n_jobs=
cv_score_rf3 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=20, n_jobs=
cv_score_rf4 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=50, n_jobs=
```

In [20]:
```
cv_result = { 'rf5': cv_score_rf1, 'rf10': cv_score_rf2, 'rf20': cv_score_rf3, 'r
cv_data = {model: [score.mean(), score.std()] for model, score in cv_result.items
cv_df = pd.DataFrame(cv_data, index=['Mean_accuracy', 'Variance'])
cv_df
```

Out[20]:

	rf5	rf10	rf20	rf50
Mean_accuracy	0.774757	0.783010	0.786408	0.793647
Variance	0.020330	0.018555	0.039794	0.065397

---------- GSR----------

In [21]:
```
# Create dataframe for training dataset and print five first rows as preview
train_df_raw = pd.read_csv('gsr.csv')
train_df_raw.head()
```

Out[21]:

	Condition	SCL
0	0.0	80.239727
1	0.0	77.365127
2	0.0	77.359559
3	0.0	76.728772
4	0.0	76.512877

In [22]:
```
train_df_raw=train_df_raw.dropna()
```

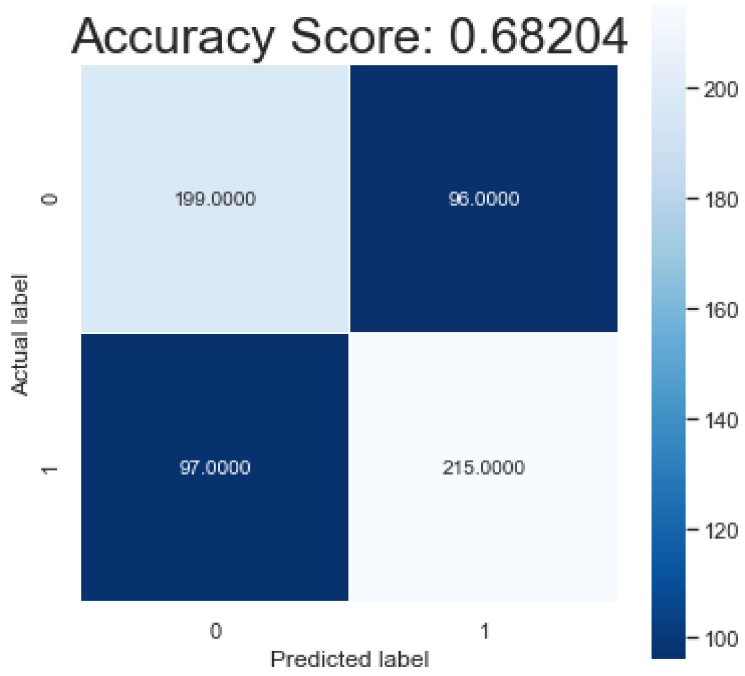
In [23]:
```
# Let's divide the train dataset in two datasets to evaluate perfomance of the mo
train_df = train_df_raw.copy()
X = train_df.drop(['Condition'], 1)
Y = train_df['Condition']
# We scale our data, it is essential for a smooth working of the models
# Scaling means that each columns as a 0 mean and a 1 variance
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X.values), index=X.index, columns=X.columns)
# Split dataset for model testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_s
X_train.head()
```

Out[23]:

	SCL
1378	-0.279137
1223	-0.500019
3119	-0.220554
1239	-0.570723
7	-0.763328

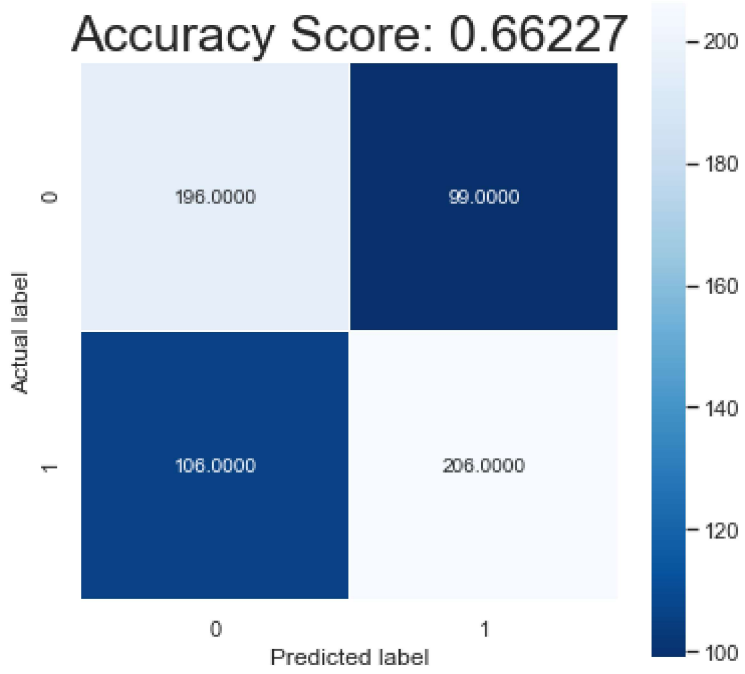
RANDOM FOREST

In [24]:
```
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, Y_train)
rf_prediction = rf.predict(X_test)
score = metrics.accuracy_score(Y_test, rf_prediction)
display_confusion_matrix(Y_test, rf_prediction, score=score)
rf_score=rf_prediction
```



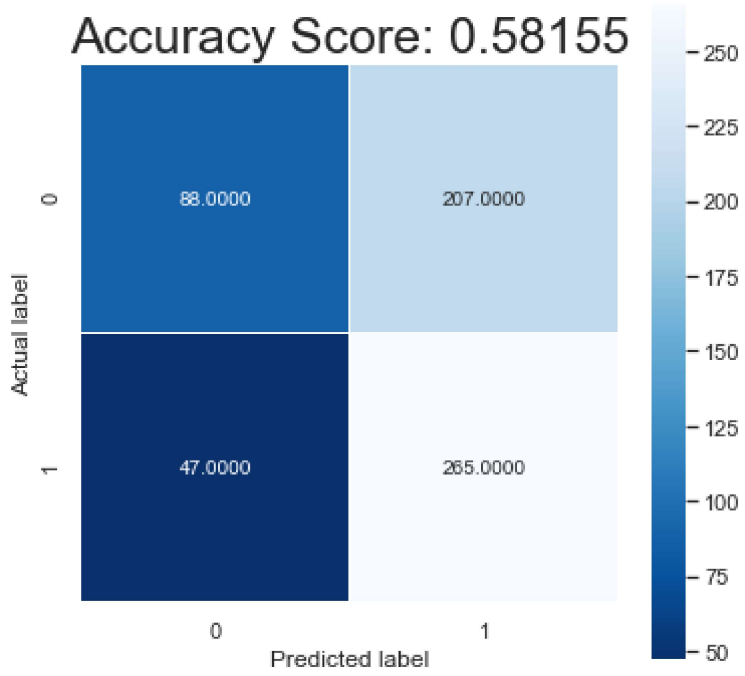
DECISION TREE

In [25]:
```
dt = DecisionTreeClassifier(min_samples_split=15, min_samples_leaf=20, random_sta
dt.fit(X_train, Y_train)
dt_prediction = dt.predict(X_test)

score = metrics.accuracy_score(Y_test, dt_prediction)
display_confusion_matrix(Y_test, dt_prediction, score=score)
```



SVM

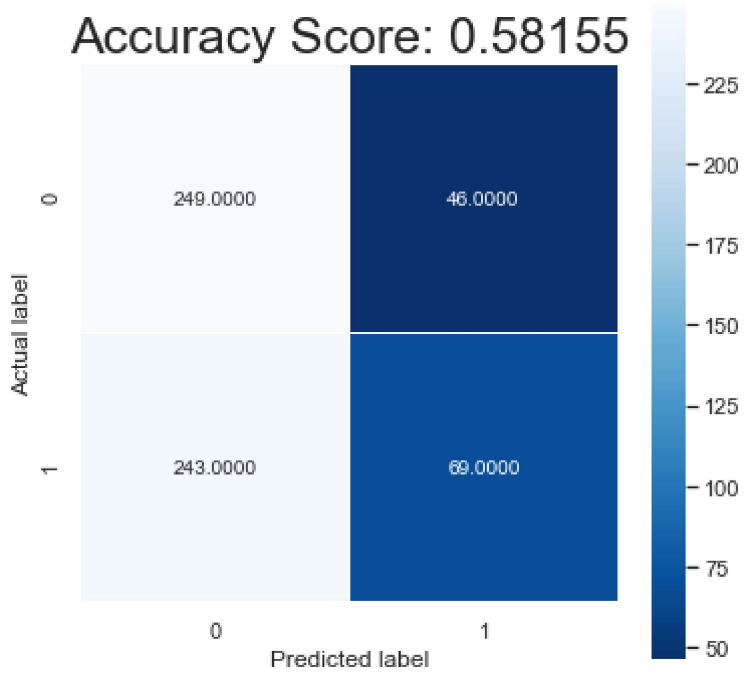
In [26]:
```
svm = SVC(gamma='auto', random_state=42)
svm.fit(X_train, Y_train)
svm_prediction = svm.predict(X_test)

score = metrics.accuracy_score(Y_test, svm_prediction)
display_confusion_matrix(Y_test, svm_prediction, score=score)
```



NAIVE BAYES

In [27]:
```
#train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, Y_train)
NB_y_pred = gnb.predict(X_test)
from sklearn.metrics import accuracy_score
y_pred_train = gnb.predict(X_train)
y_pred_train
# print the scores on training and test set
print('Training set score: {:.4f}'.format(gnb.score(X_train, Y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, Y_test)))
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y_test, NB_y_pred)
display_confusion_matrix(Y_test, NB_y_pred, score=score)
```

```
Training set score: 0.5307
Test set score: 0.5239
```



K NEAREST NEIGHBORS

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, Y_train)
KNN_y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, KNN_y_pred))
print(classification_report(Y_test, KNN_y_pred))
```

```
[[184 111]
 [ 88 224]]
              precision    recall  f1-score   support

         0.0       0.68      0.62      0.65       295
         1.0       0.67      0.72      0.69       312

    accuracy                           0.67       607
   macro avg       0.67      0.67      0.67       607
weighted avg       0.67      0.67      0.67       607
```

In [29]:
```
print('********************** GSR  **********************')
print('SVM ---->',metrics.accuracy_score(Y_test, svm_prediction))
print('RF ---->',metrics.accuracy_score(Y_test, rf_prediction))
print('DT ---->',metrics.accuracy_score(Y_test, dt_prediction))
print('NB ---->',metrics.accuracy_score(Y_test, NB_y_pred))
print('KNN ---->',metrics.accuracy_score(Y_test, KNN_y_pred))
```

```
********************** GSR  **********************
SVM ----> 0.5815485996705108
RF ----> 0.6820428336079077
DT ----> 0.6622734761120264
NB ----> 0.5238879736408567
KNN ----> 0.6721581548599671
```

Confusion Matrix

In [30]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, prec
cm = np.array(confusion_matrix(Y_test, rf_prediction, labels=[0,1]))
confusion_mat= pd.DataFrame(cm, index = ['NORMAL', 'STRESSED'],
                            columns =['NORMAL','STRESSED'])
confusion_mat
```

Out[30]:

	NORMAL	STRESSED
NORMAL	199	96
STRESSED	97	215

In [31]:
```
sns.heatmap(cm,annot=True,fmt='g',cmap='Set3')
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x2adec6d2e50>

Accuracy_Score

In [32]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test, rf_prediction))
```

```
0.6820428336079077
```

Precision

In [33]:
```
print(precision_score(Y_test, rf_prediction))
```

```
0.6913183279742765
```

Recall

In [34]:
```
print(recall_score(Y_test, rf_prediction))
```

```
0.6891025641025641
```

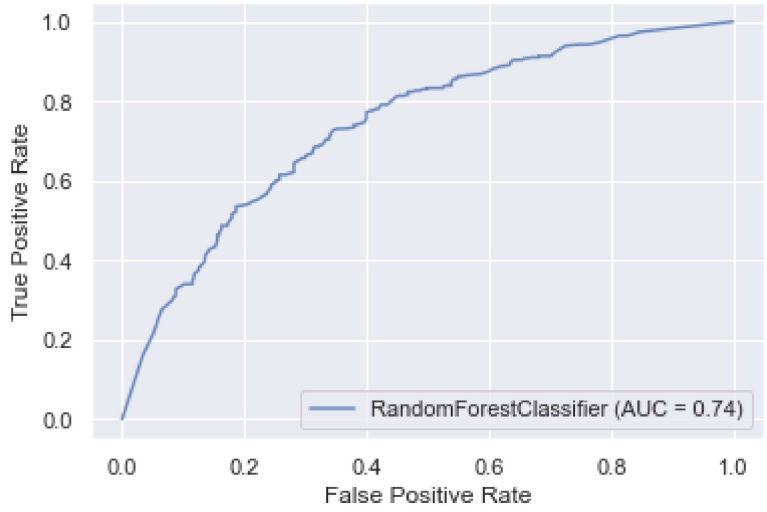
Classification Report

In [35]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, rf_prediction))
```

```
              precision    recall  f1-score   support

         0.0       0.67      0.67      0.67       295
         1.0       0.69      0.69      0.69       312

    accuracy                           0.68       607
   macro avg       0.68      0.68      0.68       607
weighted avg       0.68      0.68      0.68       607
```

Area Under Curve

In [36]:
```
from sklearn.metrics import roc_auc_score,auc,f1_score
from sklearn.metrics import plot_roc_curve
ax = plt.gca()
rfc_disp = plot_roc_curve(rf, X_test, Y_test, ax=ax, alpha=0.8)
plt.show()
```



K FOLD VALIDATION

In [37]:
```
cv_score_rf1 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=5, n_jobs=
cv_score_rf2 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=10, n_jobs=
cv_score_rf3 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=20, n_jobs=
cv_score_rf4 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=50, n_jobs=
```

In [38]:
```
cv_result = { 'rf5': cv_score_rf1, 'rf10': cv_score_rf2, 'rf20': cv_score_rf3, 'r
cv_data = {model: [score.mean(), score.std()] for model, score in cv_result.items
cv_df = pd.DataFrame(cv_data, index=['Mean_accuracy', 'Variance'])
cv_df
```

Out[38]:

	rf5	rf10	rf20	rf50
Mean_accuracy	0.643711	0.651564	0.663111	0.661454
Variance	0.010383	0.034683	0.038813	0.067435

----------FACIAL IMAGE DATA ----------

In [39]:
```
# Create dataframe for training dataset and print five first rows as preview
train_df_raw = pd.read_csv('face.csv')
train_df_raw.head()
```

Out[39]:

	Condition	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unna
0	0.0	0.968862	0.023946	-6.954147	-5.818151	-0.584952	0.0	0.000000	0.0
1	0.0	0.884570	0.076952	-10.564172	-6.567912	-3.906502	0.0	0.020921	0.0
2	0.0	0.931965	0.031468	-10.721106	-7.055848	-2.452367	0.0	0.000000	0.0
3	0.0	0.806947	0.105516	-10.782755	-5.616126	-4.669924	0.0	0.118644	0.0
4	0.0	0.951412	0.028358	-3.880091	-4.621940	-5.893645	0.0	0.033333	0.1

5 rows × 23 columns

In [40]:
```
train_df_raw=train_df_raw.dropna()
```

In [41]:
```
# Let's divide the train dataset in two datasets to evaluate perfomance of the mo
train_df = train_df_raw.copy()
X = train_df.drop(['Condition'], 1)
Y = train_df['Condition']
# We scale our data, it is essential for a smooth working of the models
# Scaling means that each columns as a 0 mean and a 1 variance
sc = StandardScaler()
X = pd.DataFrame(sc.fit_transform(X.values), index=X.index, columns=X.columns)
# Split dataset for model testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_s
X_train.head()
```

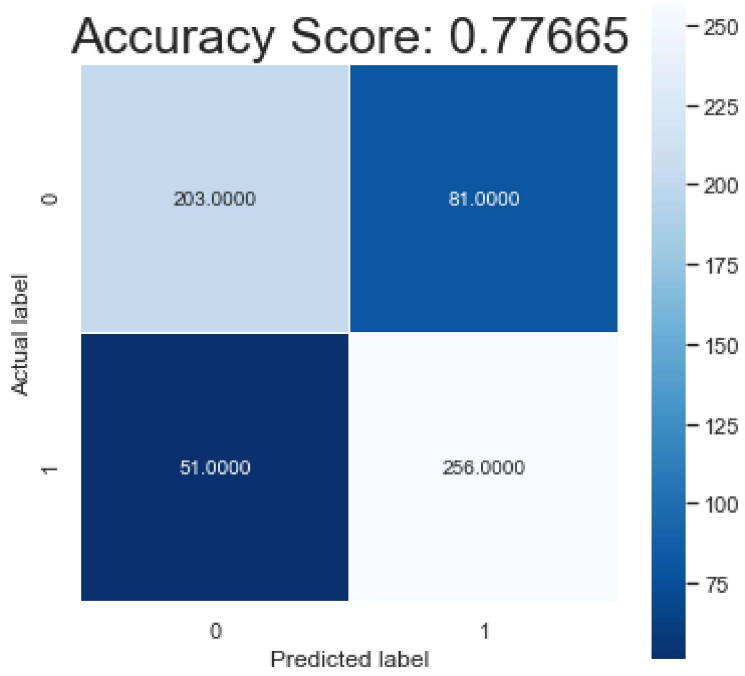
Out[41]:

	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	U
1456	-0.693505	0.393409	0.504792	-0.358339	-0.555449	1.337681	0.545038	1.255386	·
1850	0.522378	-0.470295	0.545422	-0.172419	0.519753	-0.636463	-0.701866	-0.682973	·
956	0.143040	-0.032850	0.527802	1.040186	-0.148998	-0.427209	-0.491951	1.451099	·
2443	1.081654	-0.439785	-1.254598	0.025693	-0.814183	-0.636463	-0.353466	-0.701949	·
3031	-0.837463	-0.514203	0.456152	0.754571	0.645487	3.852384	-0.593098	0.808577	

5 rows × 22 columns

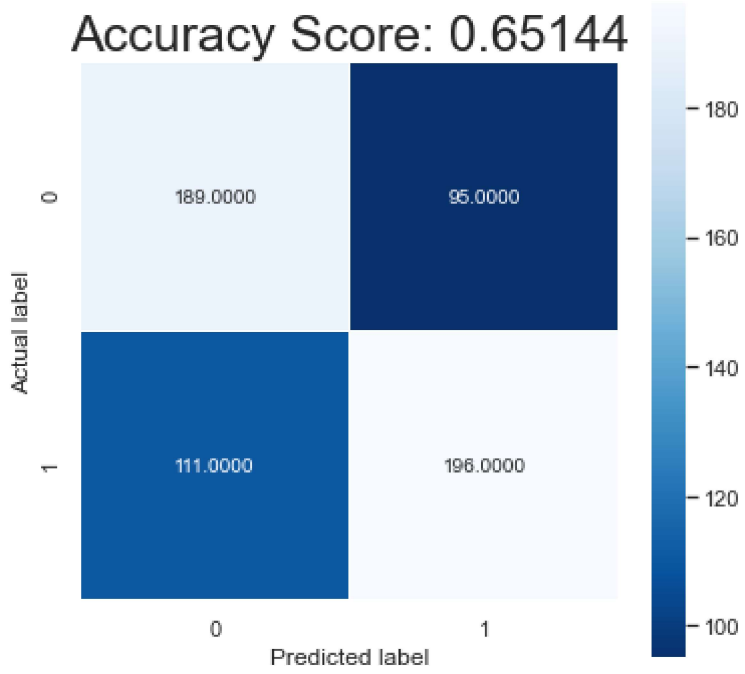
RANDOM FOREST

In [42]:
```
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, Y_train)
rf_prediction = rf.predict(X_test)
score = metrics.accuracy_score(Y_test, rf_prediction)
display_confusion_matrix(Y_test, rf_prediction, score=score)
rf_score=rf_prediction
```



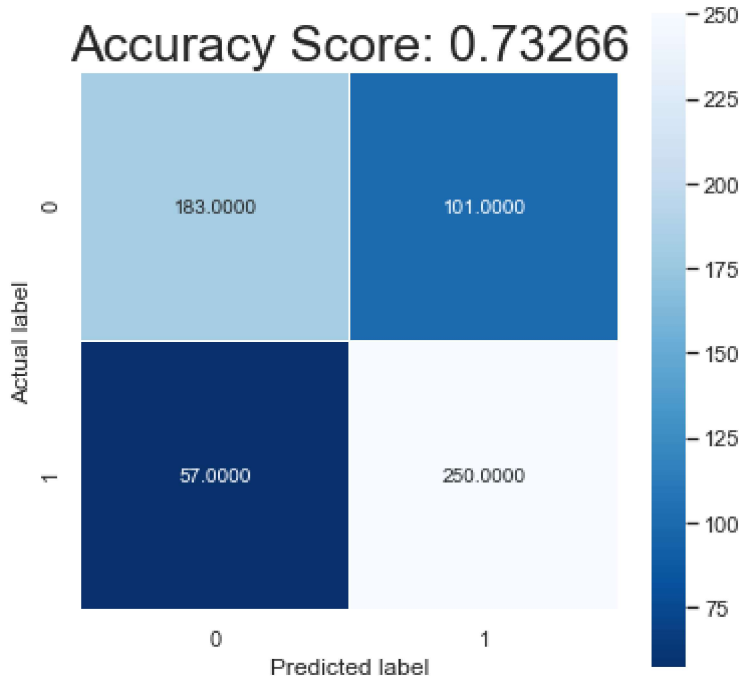
DECISION TREE

In [43]:
```
dt = DecisionTreeClassifier(min_samples_split=15, min_samples_leaf=20, random_sta
dt.fit(X_train, Y_train)
dt_prediction = dt.predict(X_test)

score = metrics.accuracy_score(Y_test, dt_prediction)
display_confusion_matrix(Y_test, dt_prediction, score=score)
```



SVM

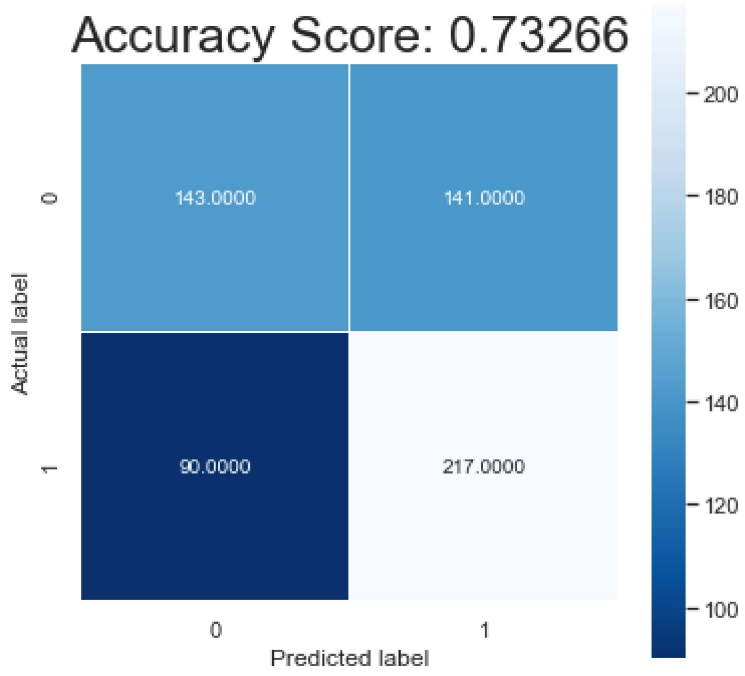
In [44]:
```
svm = SVC(gamma='auto', random_state=42)
svm.fit(X_train, Y_train)
svm_prediction = svm.predict(X_test)

score = metrics.accuracy_score(Y_test, svm_prediction)
display_confusion_matrix(Y_test, svm_prediction, score=score)
```



NAIVE BAYES

In [45]:
```
#train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, Y_train)
NB_y_pred = gnb.predict(X_test)
from sklearn.metrics import accuracy_score
y_pred_train = gnb.predict(X_train)
y_pred_train
# print the scores on training and test set
print('Training set score: {:.4f}'.format(gnb.score(X_train, Y_train)))
print('Test set score: {:.4f}'.format(gnb.score(X_test, Y_test)))
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y_test, NB_y_pred)
display_confusion_matrix(Y_test, NB_y_pred, score=score)
```

```
Training set score: 0.6227
Test set score: 0.6091
```



K NEAREST NEIGHBORS

In [46]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, Y_train)
KNN_y_pred = classifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, KNN_y_pred))
print(classification_report(Y_test, KNN_y_pred))
```

```
[[193  91]
 [ 67 240]]
              precision    recall  f1-score   support

         0.0       0.74      0.68      0.71       284
         1.0       0.73      0.78      0.75       307

    accuracy                           0.73       591
   macro avg       0.73      0.73      0.73       591
weighted avg       0.73      0.73      0.73       591
```

In [47]:
```
print('********************* FACIAL IMAGE DATA *********************')
print('SVM ---->',metrics.accuracy_score(Y_test, svm_prediction))
print('RF ---->',metrics.accuracy_score(Y_test, rf_prediction))
print('DT ---->',metrics.accuracy_score(Y_test, dt_prediction))
print('NB ---->',metrics.accuracy_score(Y_test, NB_y_pred))
print('KNN ---->',metrics.accuracy_score(Y_test, KNN_y_pred))
```

```
********************* FACIAL IMAGE DATA *********************
SVM ----> 0.7326565143824028
RF ----> 0.7766497461928934
DT ----> 0.6514382402707276
NB ----> 0.6091370558375635
KNN ----> 0.7326565143824028
```

Confusion Matrix

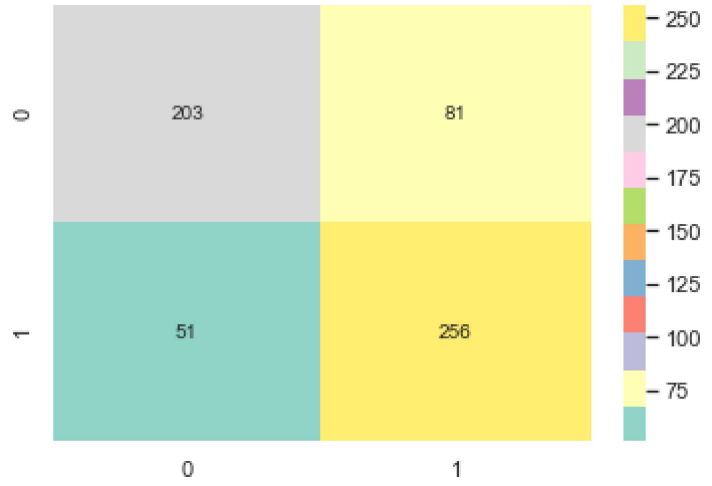
In [48]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, prec
cm = np.array(confusion_matrix(Y_test, rf_prediction, labels=[0,1]))
confusion_mat= pd.DataFrame(cm, index = ['NORMAL', 'STRESSED'],
                            columns =['NORMAL','STRESSED'])
confusion_mat
```

Out[48]:

	NORMAL	STRESSED
NORMAL	203	81
STRESSED	51	256

In [49]:
```
sns.heatmap(cm,annot=True,fmt='g',cmap='Set3')
```

Out[49]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2adeca44d60>
```



Accuracy_Score

In [50]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test, rf_prediction))
```

```
0.7766497461928934
```

Precision

In [51]:
```
print(precision_score(Y_test, rf_prediction))
```

```
0.7596439169139466
```

Recall

In [52]:
```
print(recall_score(Y_test, rf_prediction))
```

```
0.8338762214983714
```

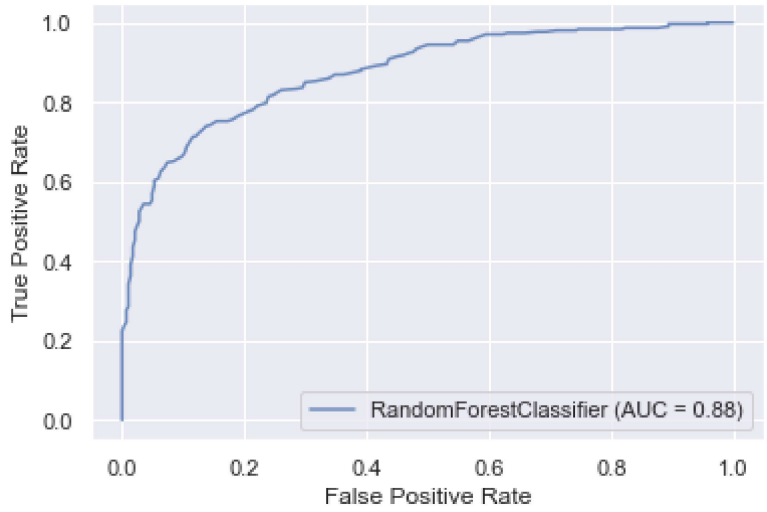
Classification Report

In [53]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, rf_prediction))
```

```
              precision    recall  f1-score   support

         0.0       0.80      0.71      0.75       284
         1.0       0.76      0.83      0.80       307

    accuracy                           0.78       591
   macro avg       0.78      0.77      0.77       591
weighted avg       0.78      0.78      0.78       591
```

Area Under Curve

In [54]:
```
from sklearn.metrics import roc_auc_score,auc,f1_score
from sklearn.metrics import plot_roc_curve
ax = plt.gca()
rfc_disp = plot_roc_curve(rf, X_test, Y_test, ax=ax, alpha=0.8)
plt.show()
```



K FOLD VALIDATION

In [*]:
```
cv_score_rf1 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=5, n_jobs=-
cv_score_rf2 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=10, n_jobs=
cv_score_rf3 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=20, n_jobs=
cv_score_rf4 = cross_val_score(estimator=rf, X=X_train, y=Y_train, cv=50, n_jobs=
```

In [*]:
```
cv_result = { 'rf5': cv_score_rf1, 'rf10': cv_score_rf2, 'rf20': cv_score_rf3, 'r
cv_data = {model: [score.mean(), score.std()] for model, score in cv_result.items
cv_df = pd.DataFrame(cv_data, index=['Mean_accuracy', 'Variance'])
cv_df
```