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
```

----------HR GSR AND FACIAL IMAGE

DATA----------

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

Out[3]:

	Condition	HR	RMSSD	SCL	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	0.0	61.0	0.061420	80.239727	0.968862	0.023946	-6.954147	-5.818151	-0.584952
1	0.0	61.0	0.061420	77.365127	0.884570	0.076952	-10.564172	-6.567912	-3.906502
2	0.0	64.0	0.049663	77.359559	0.931965	0.031468	-10.721106	-7.055848	-2.452367
3	0.0	60.0	0.052487	76.728772	0.806947	0.105516	-10.782755	-5.616126	-4.669924
4	0.0	61.0	0.051189	76.512877	0.951412	0.028358	-3.880091	-4.621940	-5.893645

5 rows × 26 columns

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	SCL	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unn
882	-0.466880	-0.514140	-0.227917	0.495755	-0.119921	1.529137	-0.578054	-1.878399	0.
2845	-0.564016	-0.373841	-0.342219	0.951577	-0.268785	-1.596594	0.285085	-0.087922	-0.
444	1.670115	0.129043	-0.411088	1.027672	0.093078	0.234768	0.893334	-1.643508	-0.
1638	0.213073	-0.805802	-0.127388	0.898400	-0.528324	-0.823113	-1.053265	-1.471973	-0.
1457	0.310209	1.367904	1.465853	-0.015720	-0.374289	0.383245	-0.948984	0.039969	0.

5 rows × 25 columns

In [6]:
```
# Create dataframe for training dataset and print five first rows as preview
train_df_raw = pd.read_csv('face_phy.csv')
train_df_raw.head()
```

Out[6]:

	Condition	HR	RMSSD	SCL	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	0.0	61.0	0.061420	80.239727	0.968862	0.023946	-6.954147	-5.818151	-0.584952
1	0.0	61.0	0.061420	77.365127	0.884570	0.076952	-10.564172	-6.567912	-3.906502
2	0.0	64.0	0.049663	77.359559	0.931965	0.031468	-10.721106	-7.055848	-2.452367
3	0.0	60.0	0.052487	76.728772	0.806947	0.105516	-10.782755	-5.616126	-4.669924
4	0.0	61.0	0.051189	76.512877	0.951412	0.028358	-3.880091	-4.621940	-5.893645

5 rows × 26 columns

In [7]:
```
train_df_raw=train_df_raw.dropna()
```

In [8]:
```
# PRE MODELING TASK
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier, NeighborhoodComponentsAnalysi
from sklearn.decomposition import PCA

X = train_df_raw.drop(['Condition'], 1)
Y = train_df_raw['Condition']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, rando
print("X_train",len(X_train))
print("X_test",len(X_test))
print("Y_train",len(Y_train))
print("Y_test",len(Y_test))
```

```
X_train 1909
X_test 478
Y_train 1909
Y_test 478
```

In [9]:
```
# to standardize the range
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```

```

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)
rf_score=rf_prediction
# print the scores on training and test set
print('Training set score: {:.4f}'.format(rf.score(X_train, Y_train)))
print('Test set score: {:.4f}'.format(rf.score(X_test, Y_test)))
```

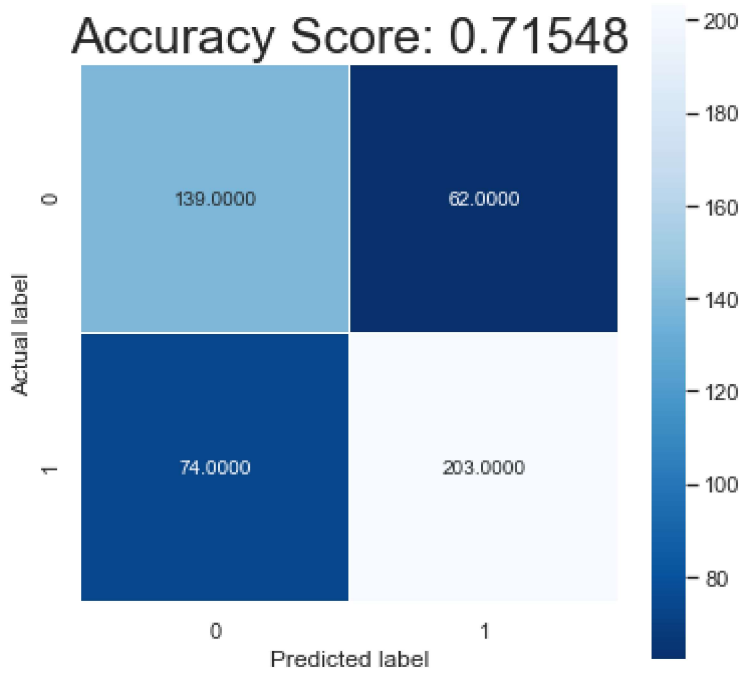
```
Training set score: 1.0000
Test set score: 0.8849
```

DECISION TREE

In [11]:
```
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)
```



In [12]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, prec

print(precision_score(Y_test, dt_prediction))
print(recall_score(Y_test, dt_prediction))
```

```
0.7660377358490567
0.7328519855595668
```

SVM

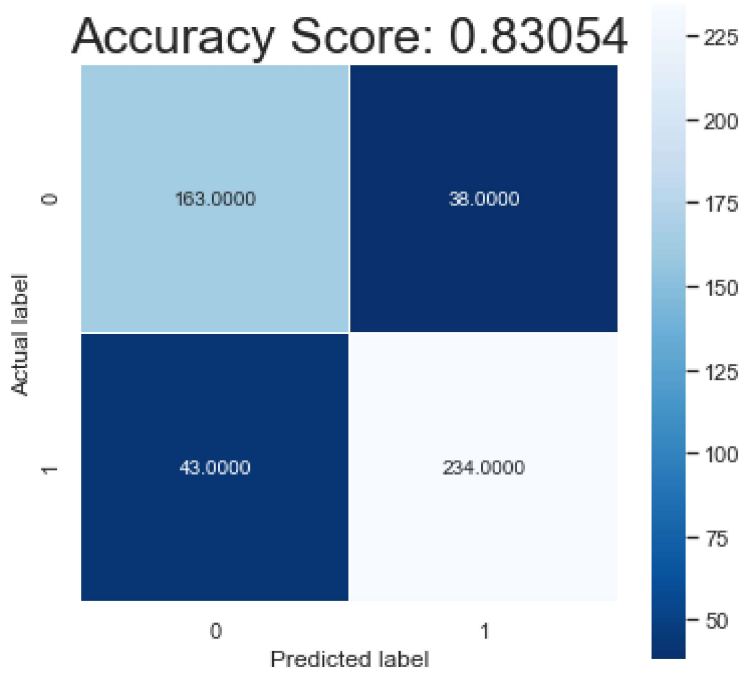
In [13]:
```
svm = SVC(gamma='auto', random_state=42)
svm.fit(X_train, Y_train)
svm_prediction = svm.predict(X_test)

score = metrics.accuracy_score(Y_test, svm_prediction)
display_confusion_matrix(Y_test, svm_prediction, score=score)
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(Y_test, svm_prediction))
print(classification_report(Y_test, svm_prediction))
```

```
[[163  38]
 [ 43 234]]
              precision    recall  f1-score   support

         0.0       0.79      0.81      0.80       201
         1.0       0.86      0.84      0.85       277

    accuracy                           0.83       478
   macro avg       0.83      0.83      0.83       478
weighted avg       0.83      0.83      0.83       478
```



In [14]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, prec

print(precision_score(Y_test, svm_prediction))
print(recall_score(Y_test, svm_prediction))
```

```
0.8602941176470589
0.8447653429602888
```

NAIVE BAYES

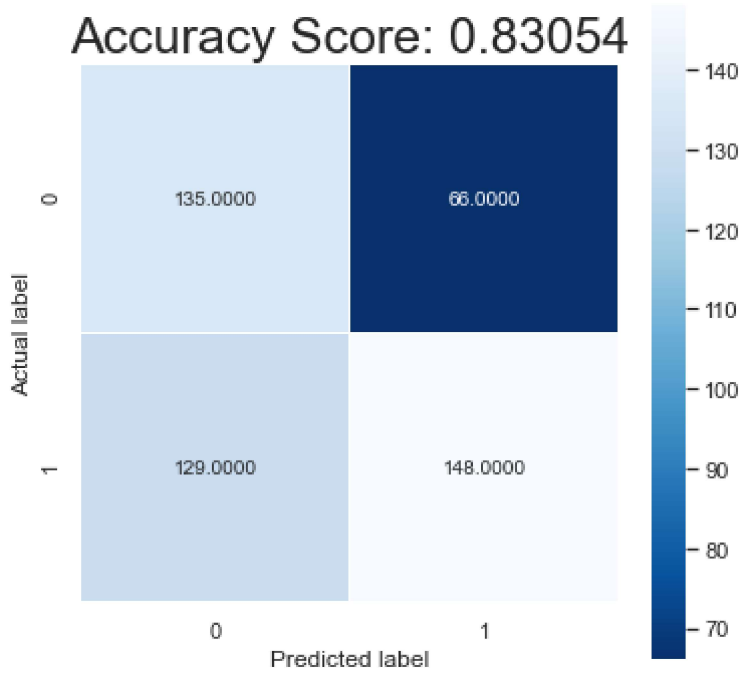
In [15]:
```
#train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, Y_train)
NB_y_pred = gnb.predict(X_test)
```

In [16]:
```
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)))
```

```
Training set score: 0.6286
Test set score: 0.5921
```

In [17]:
```
from sklearn.metrics import confusion_matrix

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



In [18]:
```
print(precision_score(Y_test, NB_y_pred))
print(recall_score(Y_test, NB_y_pred))
```

```
0.6915887850467289
0.5342960288808665
```

K NEAREST NEIGHBORS

In [19]:
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, Y_train)
KNN_y_pred = classifier.predict(X_test)
```

In [20]:
```
print(precision_score(Y_test, KNN_y_pred))
print(recall_score(Y_test, KNN_y_pred))
```

```
0.8415492957746479
0.8628158844765343
```

In [21]:
```
print('********************* HR + GSR + 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))
```

```
********************* HR + GSR + FACIAL IMAGE DATA *********************
SVM ----> 0.8305439330543933
RF ----> 0.8849372384937239
DT ----> 0.7154811715481172
NB ----> 0.5920502092050209
KNN ----> 0.8263598326359832
```

Confusion Matrix

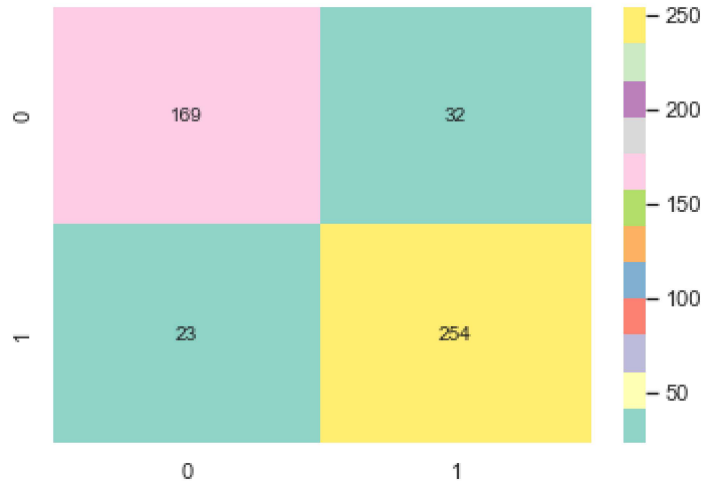
In [22]:
```
from sklearn.metrics import precision_score, recall_score, confusion_matrix, prec
cm = np.array(confusion_matrix(Y_test, rf_prediction, labels=[0,1]))
confusion_mat= pd.DataFrame(cm, index = ['NORMAL', 'STRESSED'],
                            columns =['NORMAL','STRESSED'])
confusion_mat
```

Out[22]:

	NORMAL	STRESSED
NORMAL	169	32
STRESSED	23	254

In [23]:
```
sns.heatmap(cm,annot=True,fmt='g',cmap='Set3')
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x22e4ccfa160>
```



Accuracy_Score

In [24]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(Y_test, rf_prediction))
```

```
0.8849372384937239
```

Precision

In [25]:
```
rf_pre=precision_score(Y_test, rf_prediction)
rf_pre
```

Out[25]:
```
0.8881118881118881
```

Recall

In [26]:
```
rf_recall=recall_score(Y_test, rf_prediction)
rf_recall
```

Out[26]: 0.9169675090252708

Classification Report

In [27]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test, rf_prediction))
```

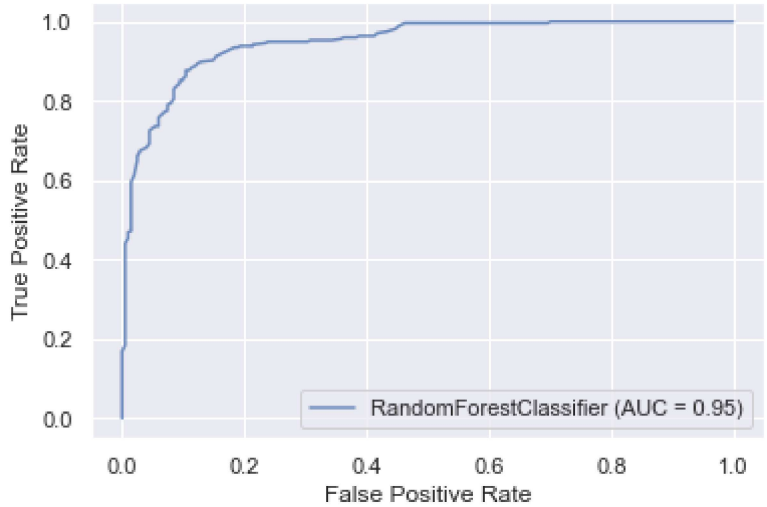
```
              precision    recall  f1-score   support

         0.0       0.88      0.84      0.86       201
         1.0       0.89      0.92      0.90       277

    accuracy                           0.88       478
   macro avg       0.88      0.88      0.88       478
weighted avg       0.88      0.88      0.88       478
```

Area Under Curve

In [28]:
```
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 [29]:
```
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 [30]:
```
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[30]:

	RF5	RF10	RF20	RF50
Mean_accuracy	0.871665	0.873748	0.880016	0.878516
Variance	0.012461	0.022806	0.037503	0.055084

In [31]:
```
cv_score_dt1 = cross_val_score(estimator=dt, X=X_train, y=Y_train, cv=5, n_jobs=
cv_score_dt2 = cross_val_score(estimator=dt, X=X_train, y=Y_train, cv=10, n_jobs=
cv_score_dt3 = cross_val_score(estimator=dt, X=X_train, y=Y_train, cv=20, n_jobs=
cv_score_dt4 = cross_val_score(estimator=dt, X=X_train, y=Y_train, cv=50, n_jobs=
```

In [32]:
```
cv_result = { 'rf5': cv_score_dt1, 'rf10': cv_score_dt2, 'rf20': cv_score_dt3, '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[32]:

	rf5	rf10	rf20	rf50
Mean_accuracy	0.733373	0.722373	0.733432	0.724507
Variance	0.023499	0.020612	0.035718	0.071758

In [33]:
```
cv_score_svm1 = cross_val_score(estimator=svm, X=X_train, y=Y_train, cv=5, n_jobs
cv_score_svm2 = cross_val_score(estimator=svm, X=X_train, y=Y_train, cv=10, n_job
cv_score_svm3 = cross_val_score(estimator=svm, X=X_train, y=Y_train, cv=20, n_job
cv_score_svm4 = cross_val_score(estimator=svm, X=X_train, y=Y_train, cv=50, n_job
```

In [34]:
```
cv_result = { 'rf5': cv_score_svm1, 'rf10': cv_score_svm2, 'rf20': cv_score_svm3,
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[34]:

	rf5	rf10	rf20	rf50
Mean_accuracy	0.803548	0.807746	0.811930	0.810864
Variance	0.018381	0.023710	0.036918	0.064591

In [35]:
```
cv_score_knn1 = cross_val_score(estimator=classifier, X=X_train, y=Y_train, cv=5,
cv_score_knn2 = cross_val_score(estimator=classifier, X=X_train, y=Y_train, cv=10
cv_score_knn3 = cross_val_score(estimator=classifier, X=X_train, y=Y_train, cv=20
cv_score_knn4 = cross_val_score(estimator=classifier, X=X_train, y=Y_train, cv=50
```

In [36]:
```
cv_result = { 'knn5': cv_score_knn1, 'knn10': cv_score_knn2, 'knn20': cv_score_kn
cv_data = {model: [score.mean()] for model, score in cv_result.items()}
cv_df = pd.DataFrame(cv_data, index=['Mean_accuracy'])
cv_df
```

Out[36]:

	knn5	knn10	knn20	knn50
Mean_accuracy	0.830781	0.837071	0.840702	0.840742

In [37]:
```
dictionary={"model":["KNN","SVM","DT","RF"],"score":[metrics.accuracy_score(Y_tes
                                                     metrics.accuracy_score(
                                                     metrics.accuracy_score(
                                                     metrics.accuracy_score(

df1=pd.DataFrame(dictionary)
```

In [38]:
```
pip install plotly==4.14.3
```

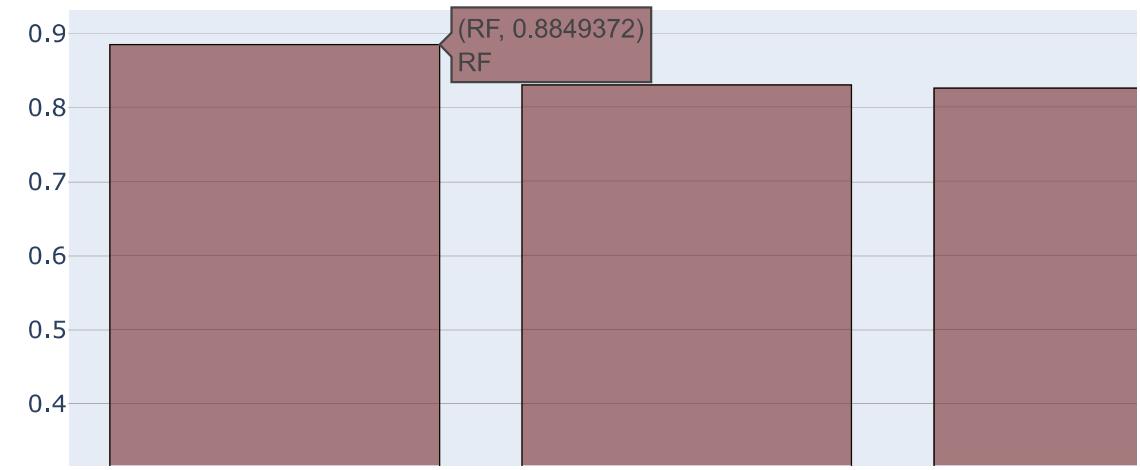
```
Requirement already satisfied: plotly==4.14.3 in c:\users\drashti\anaconda3\lib
\site-packages (4.14.3)
Requirement already satisfied: six in c:\users\drashti\anaconda3\lib\site-packa
ges (from plotly==4.14.3) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\drashti\anaconda3\li
b\site-packages (from plotly==4.14.3) (1.3.3)
Note: you may need to restart the kernel to use updated packages.
```

In [39]:
```
import matplotlib.pyplot as plt
import plotly.graph_objs as go

#sort the values of data
import plotly
plotly.offline.init_notebook_mode()

new_index5=df1.score.sort_values(ascending=False).index.values
sorted_data5=df1.reindex(new_index5)

# create trace1
trace1 = go.Bar(
                x = sorted_data5.model,
                y = sorted_data5.score,
                name = "score",
                marker = dict(color = 'rgba(100, 10, 10, 0.5)',
                              line=dict(color='rgb(10,10,0)')),
                text = sorted_data5.model)
dat = [trace1]
layout = go.Layout(barmode = "group",title= 'Scores of Classifications')
fig = go.Figure(data = dat, layout = layout)
plotly.offline.iplot(fig)
```



In [40]:
```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(Y_test, rf_prediction))
print('Mean Squared Error:', metrics.mean_squared_error(Y_test, rf_prediction))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(Y_test, rf_p
```

```
Mean Absolute Error: 0.11506276150627615
Mean Squared Error: 0.11506276150627615
Root Mean Squared Error: 0.3392090233267331
```