Supervised_Final_Haert_Failure_Project_5509

April 24, 2023

1 Supervised-learning-final-project

1.1 Heart Failure Clinical Records

1.2 Kavitha Sundaram

Heart failure is a serious condition and there is no cure for this disease. It is a situation in which the patient's heart is not pumping the blood well as the normal heart pumps. Heart Failure prediction is a complex task in the medical field. The rates of heart failure have been increasing day by day as the rate of population is also increasing day by day.

This paper aims at analyzing the machine learning algorithms based on the percentage of various performance metrics (such as, Accuracy, Precision and Recall). The machine learning methodology is proposed. The most suitable algorithm for each metrics is predicted. It is analyzed using the specific variables in the dataset by using the python programming as well as different supervised machine learning algorithms which include, Decision Tree, Logistic Regression, KNN and Random Forest. Anaconda jupyter notebook is used for implementing python scripting.

DataSource: https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records

Provide the names, email addresses, institutions, and other contact information of the donors and creators of the data set.The original dataset version was collected by Tanvir Ahmad, Assia Munir, Sajjad Haider Bhatti, Muhammad Aftab, and Muhammad Ali Raza (Government College University, Faisalabad, Pakistan) and made available by them on FigShare under the Attribution 4.0 International (CC BY 4.0: freedom to share and adapt the material) copyright in July 2017.

1.3 Contents:

- Anaysis & Results
- Conclusion
- Reference

1.4 Imports:

Below listed are the main libraries used in this project: 1. Pandas 2. NumPy 3. Seaborn 4. Plotly 5. scikit-learn 6. Matplotlib

[1]:
```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
import os
import matplotlib.animation as animation
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import warnings
warnings.filterwarnings('ignore')
# Prints the current working directory
os.getcwd()
#changing my working directory as per project folder BBC files.
%cd "/Users/kavithasundaram/Documents/SKavitha/spring march-may 2023/DTSA-5509/
 ↪final exam/dataset_heart"
```

```
/Users/kavithasundaram/Documents/SKavitha/spring march-may 2023/DTSA-5509/final
exam/dataset_heart
```

[2]:
```
#list of datafiles from UCI ML Data repository dataset
os.listdir("./")
```

[2]:
```
['.DS_Store', 'model.png', 'heart_failure_clinical_records_dataset.csv']
```

1.5 Description:

This dataset contains the medical records of 299 patients who had heart failure, collected during their follow-up period, where each patient profile has 13 clinical features.

Provide the names, email addresses, institutions, and other contact information of the donors and creators of the data set.The original dataset version was collected by Tanvir Ahmad, Assia Munir, Sajjad Haider Bhatti, Muhammad Aftab, and Muhammad Ali Raza (Government College University, Faisalabad, Pakistan) and made available by them on FigShare under the Attribution 4.0 International (CC BY 4.0: freedom to share and adapt the material) copyright in July 2017.

HF: Heart Failure is medical term.

[3]:
```
# Load in heart data
hf_rec = pd.read_csv("./heart_failure_clinical_records_dataset.csv")
display(hf_rec.info(),hf_rec.head(),hf_rec.describe())
```

```
<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
```

```
None
```

```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  75.0        0                       582         0                 20   
1  55.0        0                      7861         0                 38   
2  65.0        0                       146         0                 20   
3  50.0        1                       111         0                 20   
4  65.0        1                       160         1                 20   

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    1  265000.00               1.9           130    1   
1                    0  263358.03               1.1           136    1   
2                    0  162000.00               1.3           129    1   
3                    0  210000.00               1.9           137    1   
4                    0  327000.00               2.7           116    0   

   smoking  time  DEATH_EVENT  
0        0     4            1  
1        0     6            1  
2        1     7            1  
3        0     7            1  
4        0     8            1  
              age     anaemia  creatinine_phosphokinase    diabetes  \
count  299.000000  299.000000                299.000000  299.000000   
mean    60.833893    0.431438                581.839465    0.418060   
std     11.894809    0.496107                970.287881    0.494067   
min     40.000000    0.000000                 23.000000    0.000000   
```

```
25%     51.000000    0.000000                 116.500000    0.000000
50%     60.000000    0.000000                 250.000000    0.000000
75%     70.000000    1.000000                 582.000000    1.000000
max     95.000000    1.000000                7861.000000    1.000000

       ejection_fraction  high_blood_pressure      platelets
count         299.000000           299.000000     299.000000  \
mean           38.083612             0.351171  263358.029264
std            11.834841             0.478136   97804.236869
min            14.000000             0.000000   25100.000000
25%            30.000000             0.000000  212500.000000
50%            38.000000             0.000000  262000.000000
75%            45.000000             1.000000  303500.000000
max            80.000000             1.000000  850000.000000

       serum_creatinine  serum_sodium         sex    smoking        time
count         299.00000    299.000000  299.000000  299.00000  299.000000  \
mean            1.39388    136.625418    0.648829    0.32107  130.260870
std             1.03451      4.412477    0.478136    0.46767   77.614208
min             0.50000    113.000000    0.000000    0.00000    4.000000
25%             0.90000    134.000000    0.000000    0.00000   73.000000
50%             1.10000    137.000000    1.000000    0.00000  115.000000
75%             1.40000    140.000000    1.000000    1.00000  203.000000
max             9.40000    148.000000    1.000000    1.00000  285.000000

       DEATH_EVENT
count    299.00000
mean       0.32107
std        0.46767
min        0.00000
25%        0.00000
50%        0.00000
75%        1.00000
max        1.00000
```

1.6 Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data:

For future analysis , am going to rename some variables in short form to predict the analysis way better.

[4]:
```
#renaming creatinine_phosphokinase as CPK:
hf_rec["CPK"] = hf_rec["creatinine_phosphokinase"]
hf_rec = hf_rec.drop("creatinine_phosphokinase", axis=1)
#renaming ejection_fraction as EF:
hf_rec["EF"] = hf_rec["ejection_fraction"]
hf_rec = hf_rec.drop("ejection_fraction", axis=1)
#renaming high_blood_pressure as HBP:
hf_rec["high_BP"] = hf_rec["high_blood_pressure"]
```

```
hf_rec = hf_rec.drop("high_blood_pressure", axis=1)
```

Lets Check null values and data types of all variables for model analysis.

```
[5]: hf_rec.isna().sum()
```

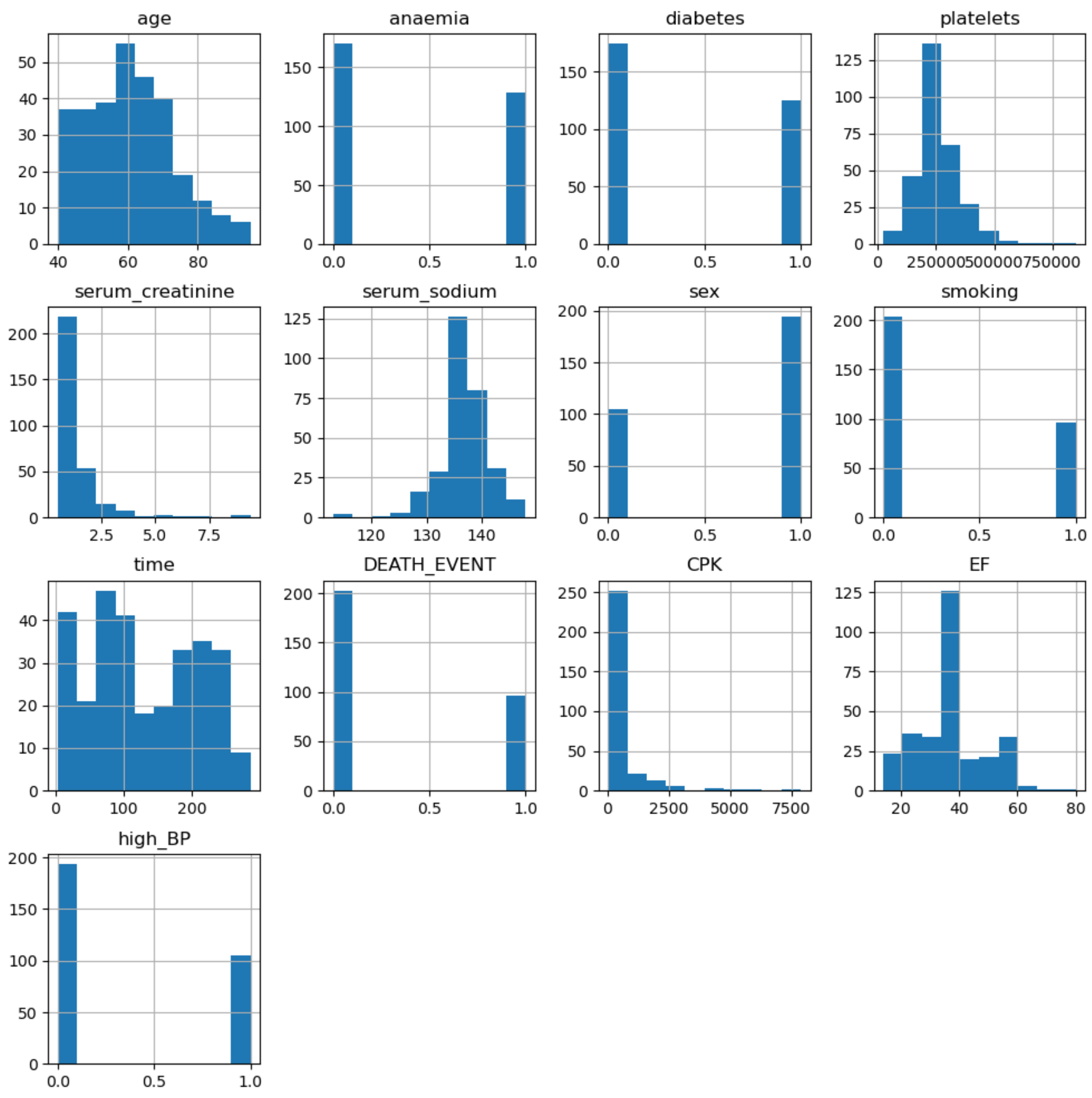
```
[5]: age                  0
     anaemia              0
     diabetes             0
     platelets            0
     serum_creatinine     0
     serum_sodium         0
     sex                  0
     smoking              0
     time                 0
     DEATH_EVENT          0
     CPK                  0
     EF                   0
     high_BP              0
     dtype: int64
```

```
[6]: hf_rec.dtypes
```

```
[6]: age                  float64
     anaemia                int64
     diabetes               int64
     platelets            float64
     serum_creatinine     float64
     serum_sodium           int64
     sex                    int64
     smoking                int64
     time                   int64
     DEATH_EVENT            int64
     CPK                    int64
     EF                     int64
     high_BP                int64
     dtype: object
```

```
[7]: hf_rec.hist(figsize=(12,12))
```

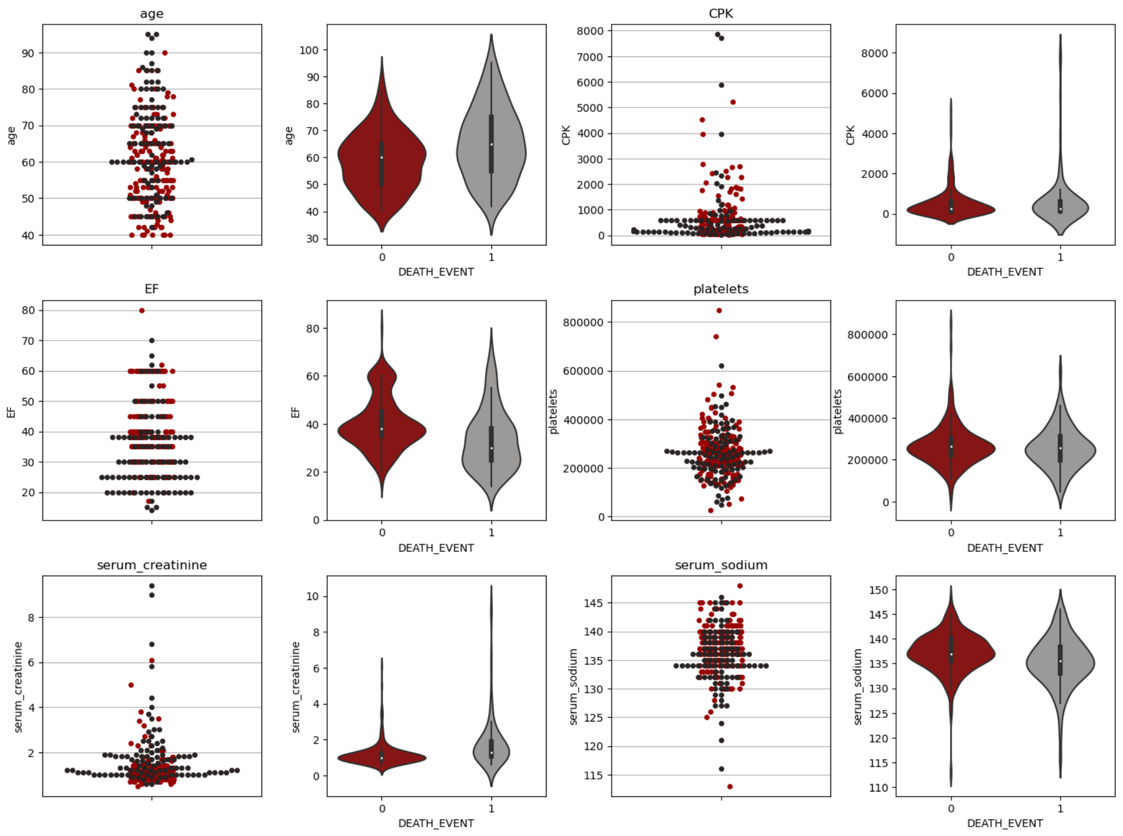
```
[7]: array([[<Axes: title={'center': 'age'}>,
             <Axes: title={'center': 'anaemia'}>,
             <Axes: title={'center': 'diabetes'}>,
             <Axes: title={'center': 'platelets'}>],
            [<Axes: title={'center': 'serum_creatinine'}>,
             <Axes: title={'center': 'serum_sodium'}>,
             <Axes: title={'center': 'sex'}>,
             <Axes: title={'center': 'smoking'}>],
            [<Axes: title={'center': 'time'}>,
             <Axes: title={'center': 'DEATH_EVENT'}>,
             <Axes: title={'center': 'CPK'}>, <Axes: title={'center': 'EF'}>],
            [<Axes: title={'center': 'high_BP'}>, <Axes: >, <Axes: >,
             <Axes: >]], dtype=object)
```



After analysing above histograms, we can easily divide our variables into 1. categorical(anaemia,diabetes,sex,smoking,high_BP) 2. numerical(age,platelets,serum_creatinine,serum_sodium,time,CPK,EF)

1.7 Data PreProcessing:

[8]:
```
numerical = ["age", "CPK", "EF", "platelets", "serum_creatinine",␣
↪"serum_sodium"]
categorical = ["anaemia", "diabetes", "high_BP", "sex", "smoking"]
plt.figure(figsize=(18, 27))

for i, col in enumerate(numerical):
    plt.subplot(6, 4, i*2+1)
    plt.subplots_adjust(hspace =.25, wspace=.3)

    plt.grid(True)
    plt.title(col)
    sns.stripplot(hf_rec.loc[hf_rec["DEATH_EVENT"]==0, col], label="alive",␣
↪color = "#990303")
    sns.swarmplot(hf_rec.loc[hf_rec["DEATH_EVENT"]==1, col], label="dead", ␣
↪color = "#292323")
    plt.subplot(6, 4, i*2+2)
    sns.violinplot(y = col, data = hf_rec, x="DEATH_EVENT", palette =␣
↪["#990303", "#9C9999"])
```

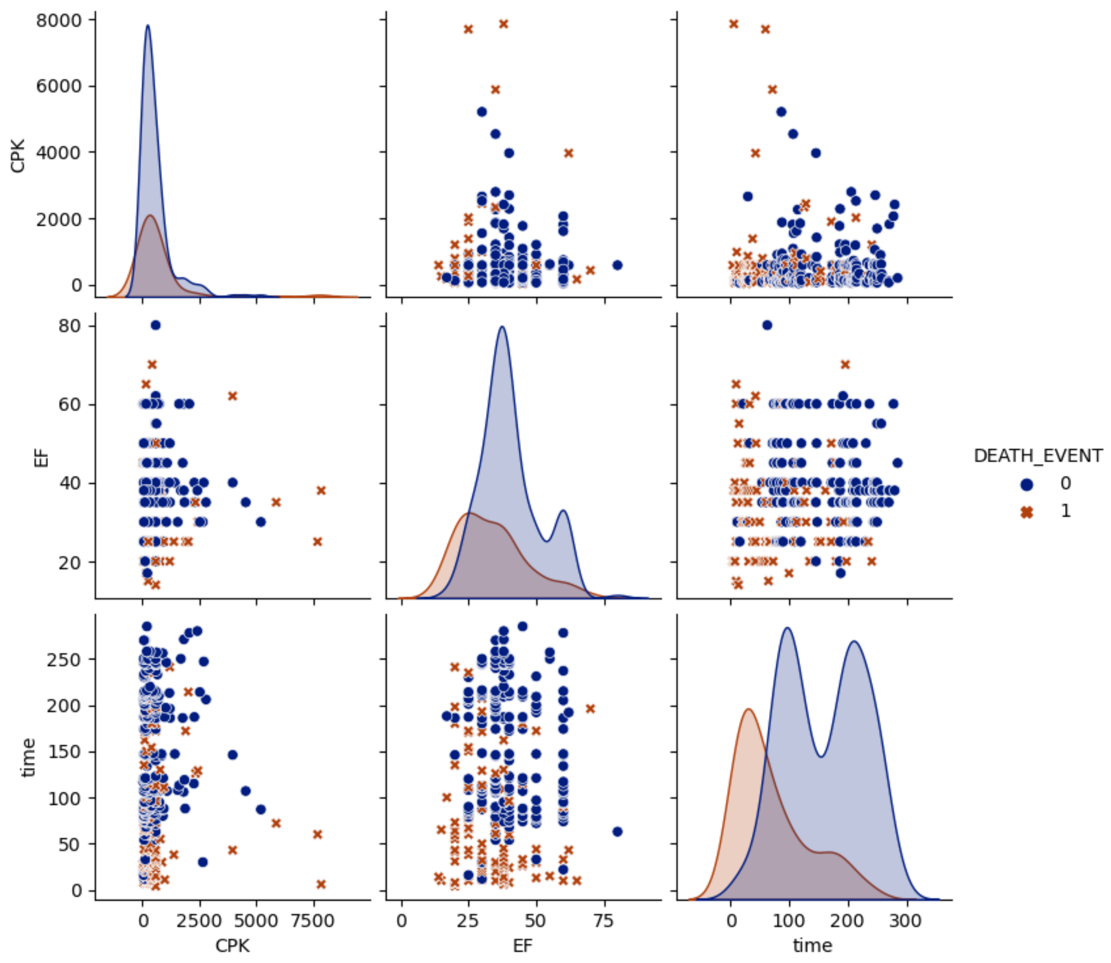


1. Look at the structure of EF and serum creatinine, both are having differents in voilin plots. Lets analyse more of categorical datatypes.
2. After looking up with **serum__creatinine** values its over normal for patients with high level serum are vulnerable to heart failure.
3. EF **ejection__fraction** values its under normal for patients with high level ejection fraction are also vulnerable to heart failure.

Lets look deep into death events and calculate percentage of CPK,EF patients .

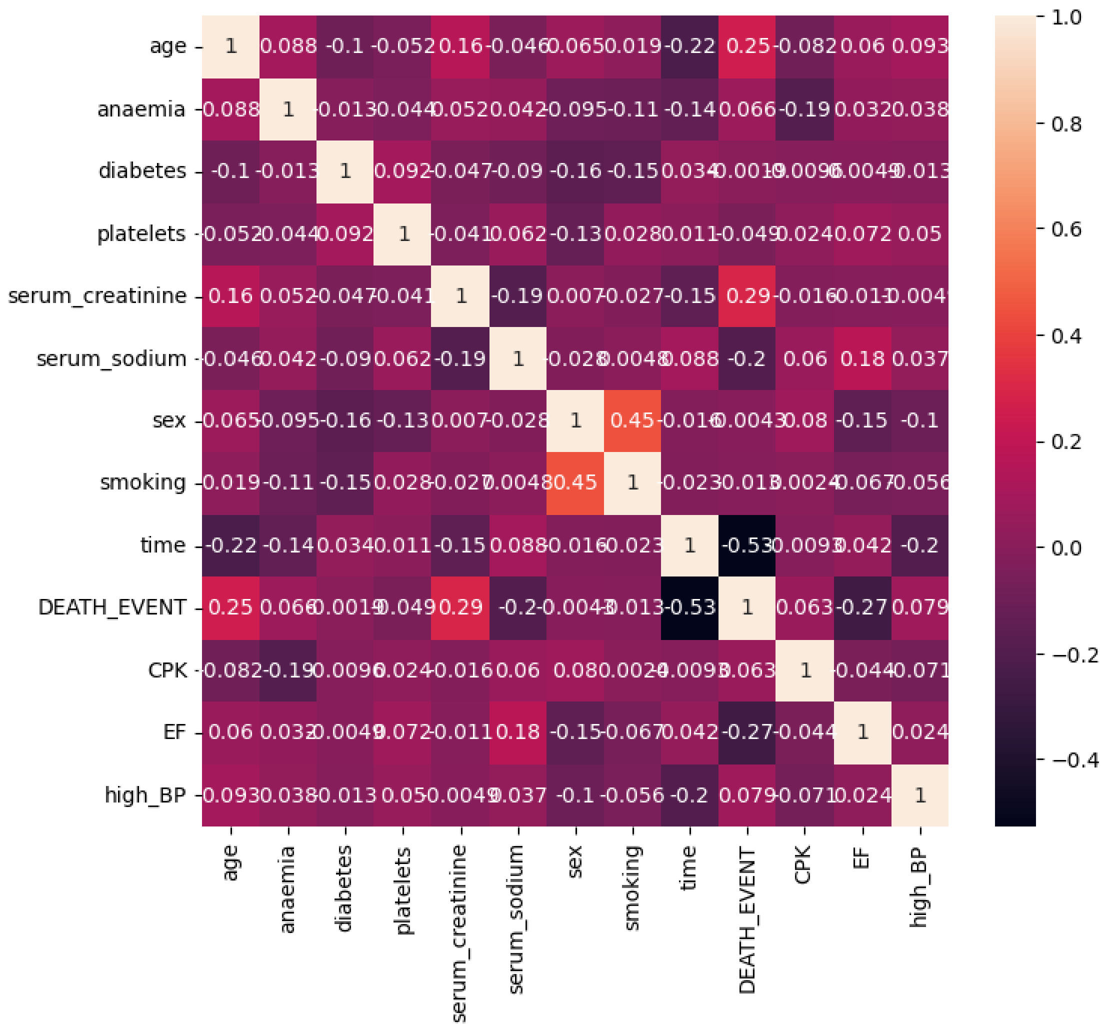
```
[9]: #sns.pairplot(hf_rec, hue="DEATH_EVENT")
     sns.pairplot(hf_rec,vars= ['CPK','EF','time'],
                  hue= 'DEATH_EVENT',markers=["o", "X" ],palette='dark')
     Data1 = hf_rec[['CPK', 'EF','DEATH_EVENT']][(hf_rec['CPK'] > 210) &␣
      ↪(hf_rec['EF'] < 50)]
     death_count =hf_rec[hf_rec['DEATH_EVENT'] == 1 ].count()[0]
     death_c = Data1['DEATH_EVENT'].count()
     total = hf_rec.shape[0]
     print(r'%s patients of 299 has not normal value for each feature. Representing␣
      ↪%s percent of patients and %s percent of total death.  '%(Data1.
      ↪shape[0],round(((Data1.shape[0]*100)/total),2), (death_c*100)/death_count ))
```

```
142 patients of 299 has not normal value for each feature. Representing 47.49
percent of patients and 147.91666666666666 percent of total death.
```



1. Normal medical range for **Creatine phosphokinase**: 2 - 210 mcg/L .
2. Normal medical range for **Ejection fraction** : 50 %.
3. Total of 299 patients, approximately 92% of patients had heart failure and passed away due to high level of CPK(which is more than 210mcg/L and EF).
4. Dataset has some unbalanced data with values.

[10]:
```
plt.figure(figsize=(8, 7))
sns.heatmap(hf_rec.corr(method='pearson'), annot=True);
```



1. Most of the varuables are uncorrelated. as you can see **sex** and **smoking** are positively correlated.

```
[11]: hf_rec.corrwith(hf_rec["DEATH_EVENT"])
```

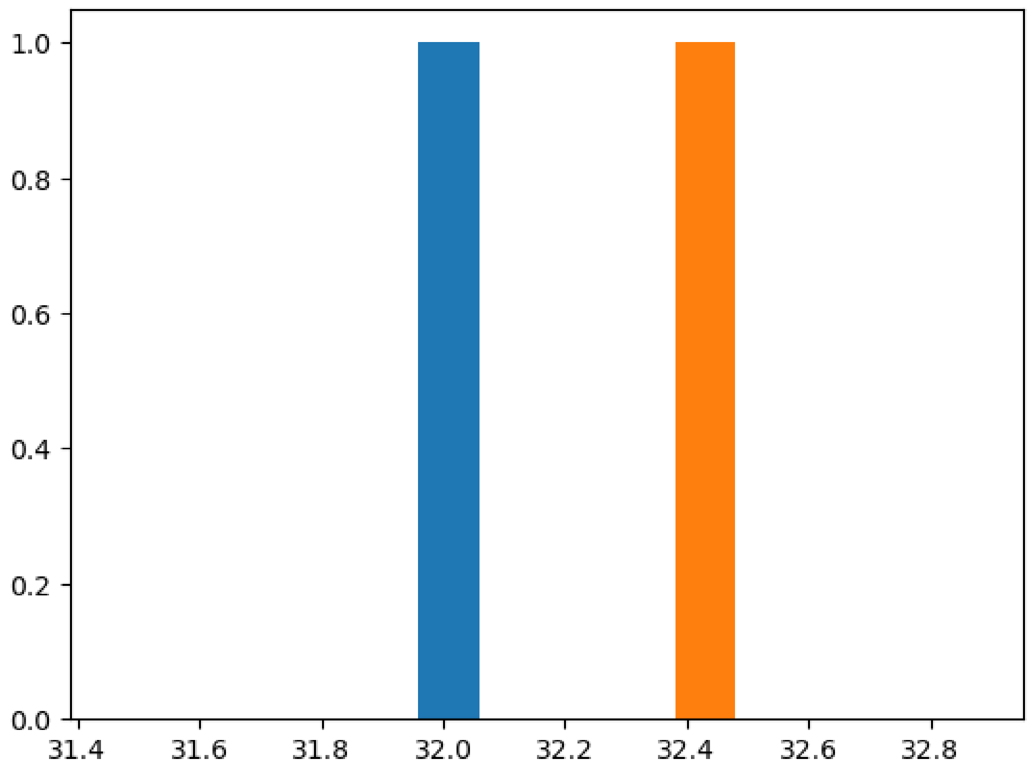
```
[11]: age                    0.253729
      anaemia                0.066270
      diabetes              -0.001943
      platelets             -0.049139
      serum_creatinine       0.294278
      serum_sodium          -0.195204
      sex                   -0.004316
      smoking               -0.012623
      time                  -0.526964
      DEATH_EVENT            1.000000
```

```
CPK                    0.062728
EF                    -0.268603
high_BP                0.079351
dtype: float64
```

1. The color coding indicates the strength of correlation between variables, with darker shades indicating higher positive correlation and lighter shades indicating lower correlation or negative correlation.
2. Age is positively correlated with serum creatinine, serum sodium, and ejection fraction, indicating that older individuals tend to have higher levels of these variables.
3. diabetes with age>60 is more vulnerable to heart failure than non-diabetes aged < 50.

```
[12]: data2 = hf_rec[['sex','DEATH_EVENT']]
      men   = data2[(data2['sex']==1) & (data2['DEATH_EVENT']==1)]
      mdp = men['sex'].count()*100/data2[(data2['sex']==1)].count()[0]
      plt.hist(mdp)
      women  = data2[(data2['sex']==0) & (data2['DEATH_EVENT']==1)]
      wdp = women['sex'].count()*100/data2[(data2['sex']==0)].count()[0]
      plt.hist(wdp)
```

```
[12]: (array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
       array([31.88095238, 31.98095238, 32.08095238, 32.18095238, 32.28095238,
              32.38095238, 32.48095238, 32.58095238, 32.68095238, 32.78095238,
              32.88095238]),
       <BarContainer object of 10 artists>)
```



1. 32% mens and 32.4% womens are vulnerable to heart failure and death
2. 68% mens and 67.6% womens are not vulnerable to heart failure.

1.8 Validation and Splitting Data:

1. Splitting data into train and test samples with .80:.20 ratio.
2. scaling data for future model classification .
3. The normalization has been done to make all the attribute values between zero and one (0–1) to reach better accuracy.

[13]:
```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import normalize
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.svm import SVC, SVR

from sklearn import metrics
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import␣
 ↪f1_score,accuracy_score,roc_curve,roc_auc_score,recall_score
from mlxtend.plotting import plot_confusion_matrix

from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score

from sklearn.metrics import roc_curve
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score
```

[14]:
```
X = hf_rec.drop('DEATH_EVENT',axis=1)
y = hf_rec['DEATH_EVENT']
x_train,x_test,y_train,y_test=train_test_split(X,y,random_state=2,test_size=.20)
X.head()
```

[14]:
```
    age  anaemia  diabetes  platelets  serum_creatinine  serum_sodium  sex
0  75.0        0         0  265000.00               1.9           130    1  \
1  55.0        0         0  263358.03               1.1           136    1
2  65.0        0         0  162000.00               1.3           129    1
3  50.0        1         0  210000.00               1.9           137    1
4  65.0        1         1  327000.00               2.7           116    0

   smoking  time   CPK  EF  high_BP
0        0     4   582  20        1
1        0     6  7861  38        0
2        1     7   146  20        0
3        0     7   111  20        0
4        0     8   160  20        0
```

[15]:
```python
x_train = normalize(x_train)
x_test = normalize(x_test)
```

[16]:
```python
scale=StandardScaler()
x_train_scale=scale.fit_transform(x_train)
x_test_scale=scale.transform(x_test)
```

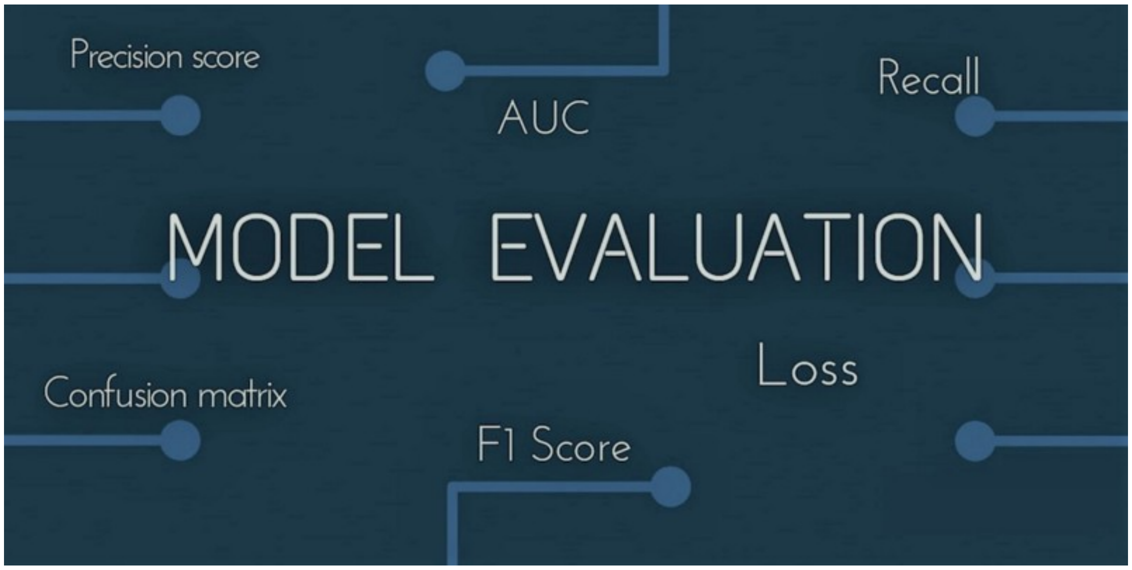
1.9 Classification Models:

Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data. In classification, the model is fully trained using the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data.

1. KNN (K Nearest Neigbors) Model
2. Naive Bayes classifier
3. Decision tree
4. Support Vector Machine
5. Random Forest classifier

[17]:
```python
from IPython import display
display.Image("./model.png")
```

[17]:



1.9.1 1. KNN Model:

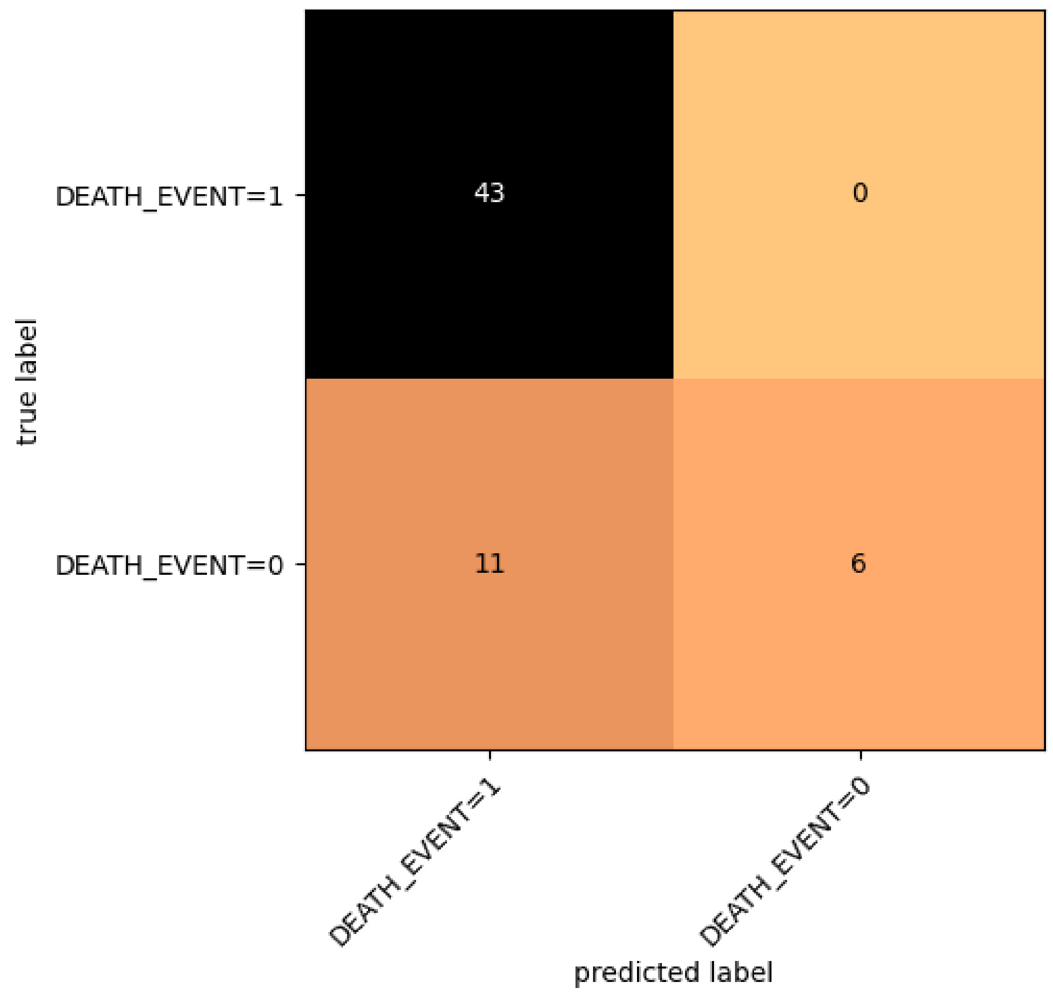
K Nearest Neighbor algorithm falls under the Supervised Learning category and is used for classification (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint.

[18]:
```
knn_m = KNeighborsClassifier(n_neighbors=31,leaf_size=30)
knn_m.fit(x_train,y_train)
pred_knn = knn_m.predict(x_test)
score_knn = round(accuracy_score(pred_knn,y_test)*100,2)
score_knn
```

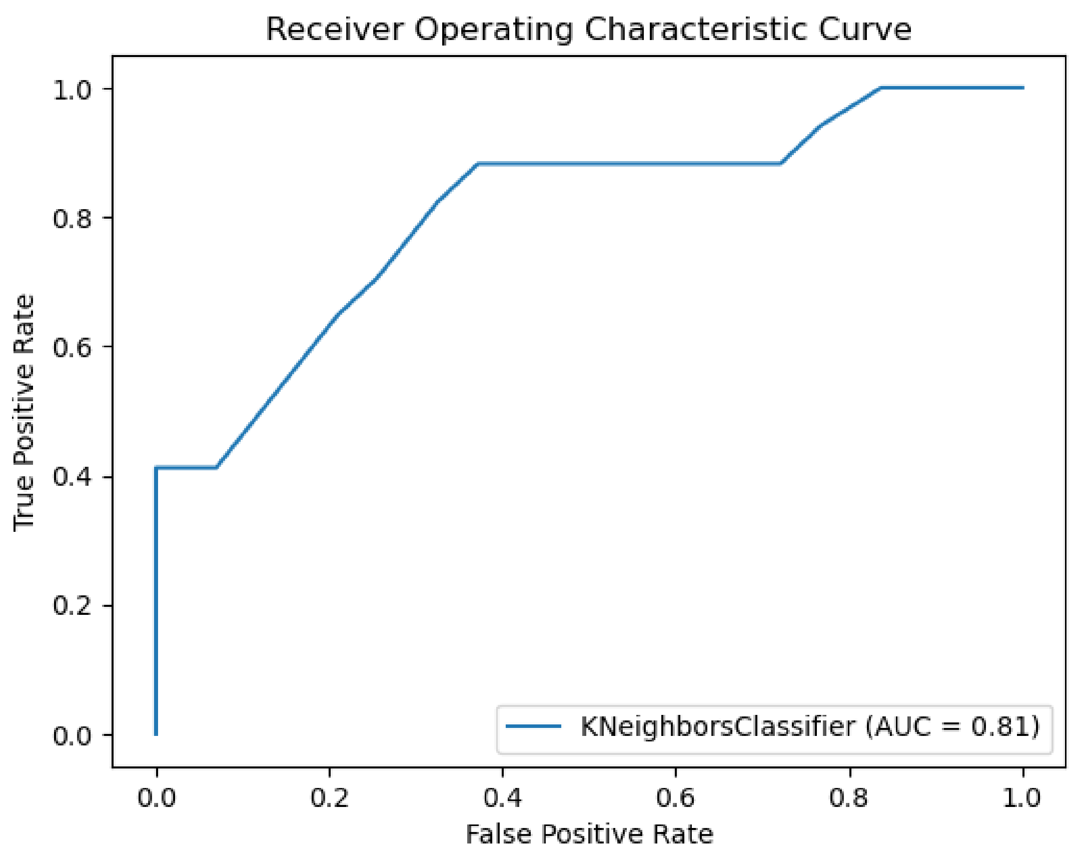
[18]:
```
81.67
```

[19]:
```
conf_mat=confusion_matrix(y_test,pred_knn)
plot_confusion_matrix(conf_mat,class_names=['DEATH_EVENT=1','DEATH_EVENT=0'],figsize=(12,5),cm
  ↪
pred_knn = np.around(pred_knn)
print(metrics.classification_report(y_test,pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.80      1.00      0.89        43
           1       1.00      0.35      0.52        17

    accuracy                           0.82        60
   macro avg       0.90      0.68      0.70        60
weighted avg       0.85      0.82      0.78        60
```



```
[20]: RocCurveDisplay.from_estimator(knn_m,x_test,y_test)
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('Receiver Operating Characteristic Curve');
```



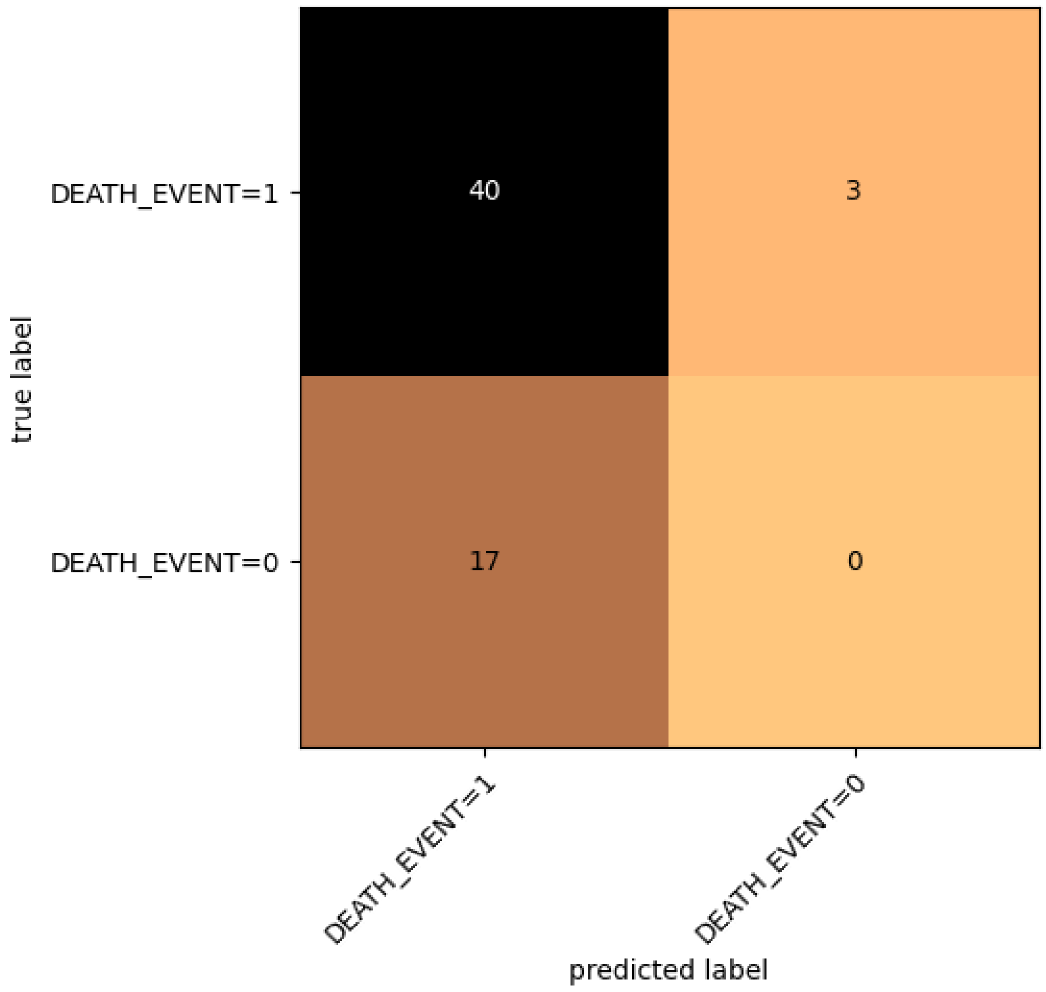
1. We can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.78 in our case
2. accuracy predicted is this case = 81%
3. **43 out of 43 death count was predicted exactly**
4. **6 out of 17 values were predicted correctly and 11 was incorrectly forecasted**

1.9.2 2. Naive Bayes:

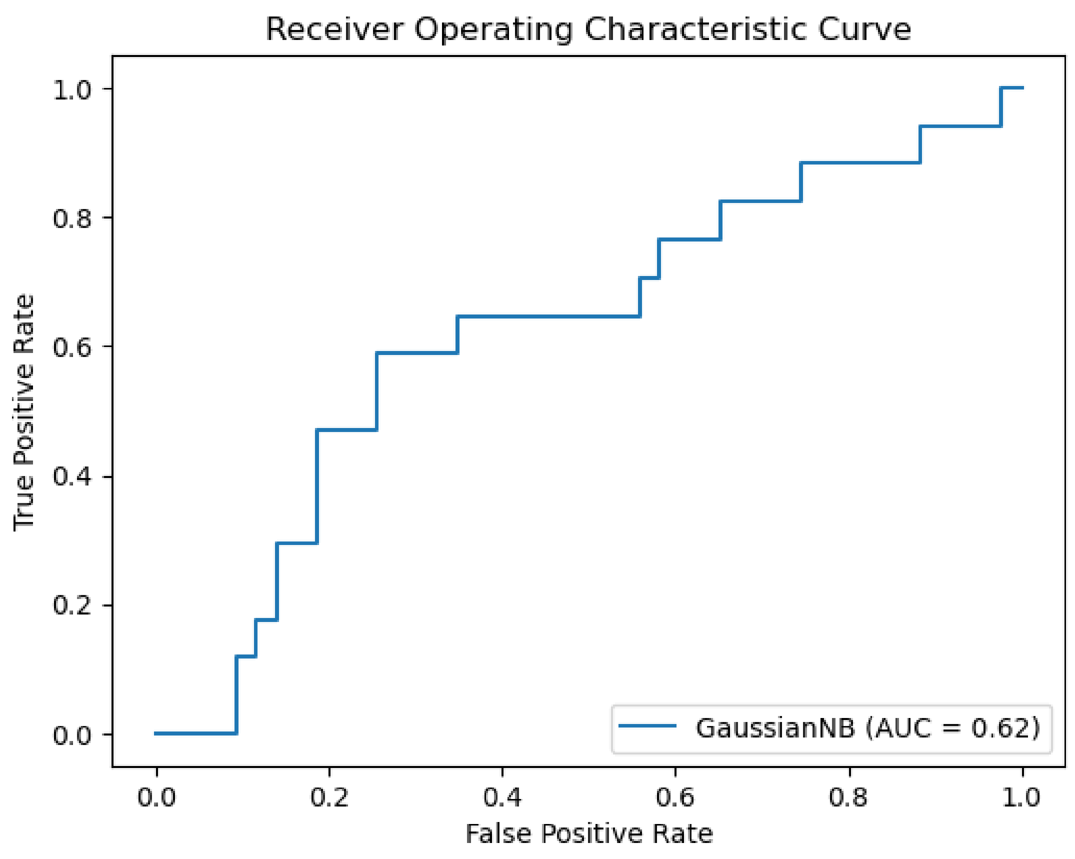
[21]:
```
nb_m = GaussianNB( var_smoothing=1e-50)
nb_m.fit(x_train,y_train)
nb_m.predict(x_test)



pred_nb = nb_m.predict(x_test)
score_nb = round(accuracy_score(pred_nb,y_test)*100,2)
score_nb
```

[21]:
```
66.67
```

[22]:
```
conf_mat=confusion_matrix(y_test,pred_nb)
plot_confusion_matrix(conf_mat,class_names=['DEATH_EVENT=1','DEATH_EVENT=0'],figsize=(12,5),cm
  ↪ #fn
```



[23]:
```
RocCurveDisplay.from_estimator(nb_m,x_test,y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
```



[24]:
```
pred_nb = np.around(pred_nb)
print(metrics.classification_report(y_test,pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.70      0.93      0.80        43
           1       0.00      0.00      0.00        17

    accuracy                           0.67        60
   macro avg       0.35      0.47      0.40        60
weighted avg       0.50      0.67      0.57        60
```

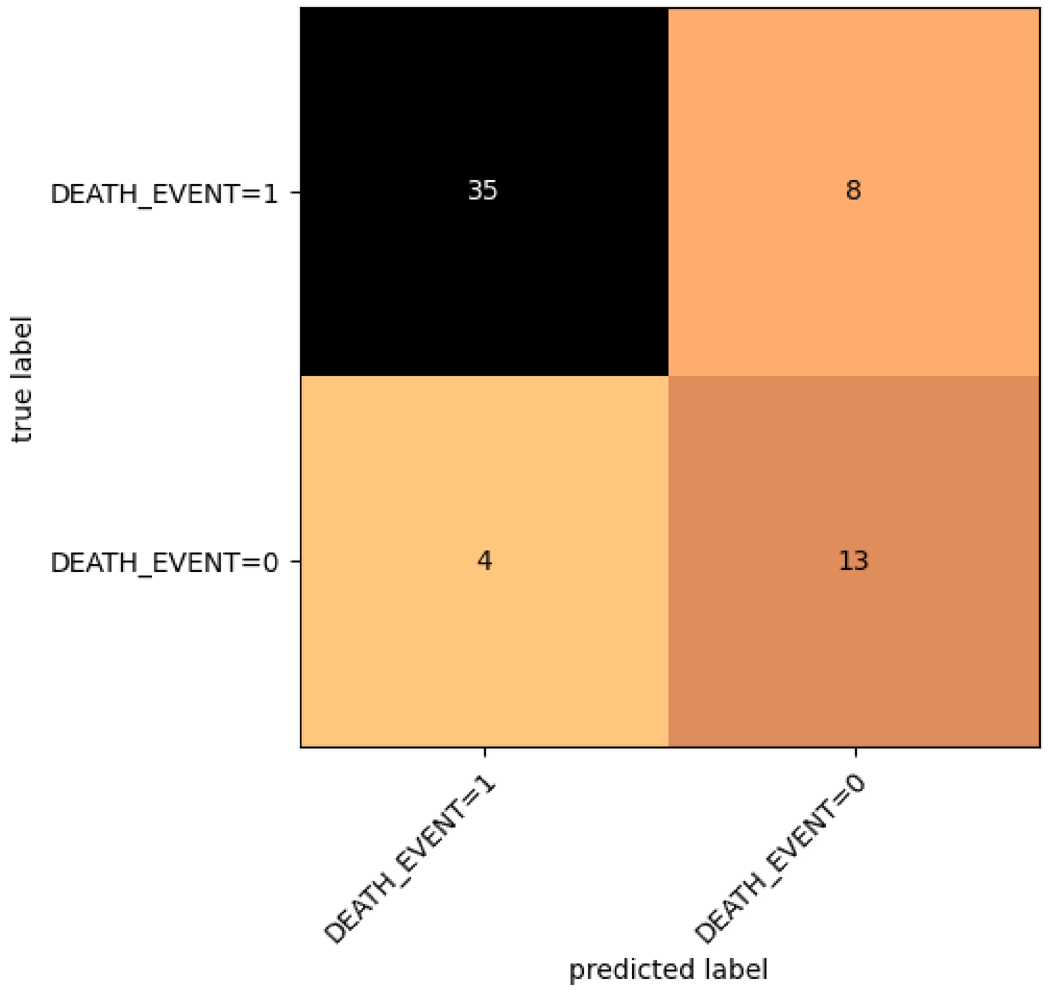
1. We can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.57 in our case
2. accuracy predicted is this case = 67%
3. **40 out of 43 death count was predicted exactly and 3 was incorrectly forecsated**
4. **17 out of 17 values were incorrectly forecasted**

1.9.3 3. Decision Tree Classifier:

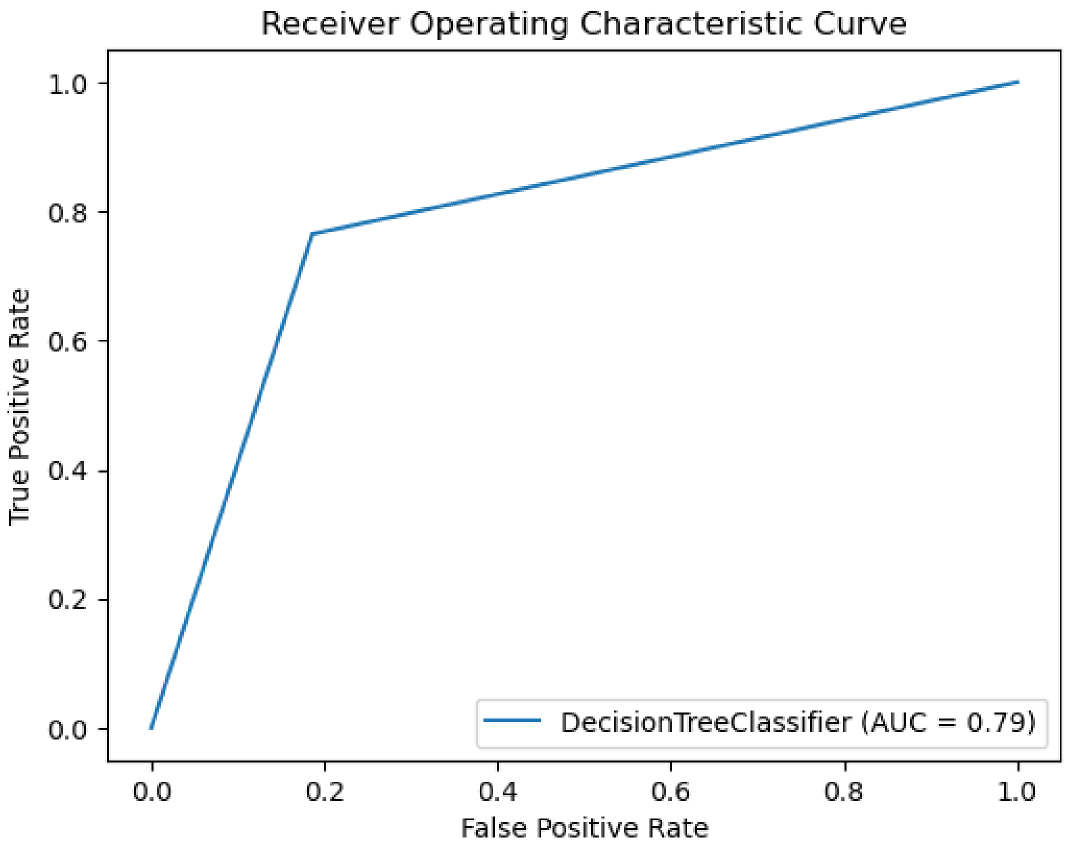
[25]:
```
dt_m=DecisionTreeClassifier()
dt_m.fit(x_train,y_train)
dt_m.predict(x_test)
pred_dt = dt_m.predict(x_test)
score_dt = round(accuracy_score(pred_dt,y_test)*100,2)
score_dt
```

[25]:
```
80.0
```

[26]:
```
conf_mat=confusion_matrix(y_test,pred_dt)
plot_confusion_matrix(conf_mat,class_names=['DEATH_EVENT=1','DEATH_EVENT=0'],figsize=(12,5),cm
  ↪ #fn
```



[27]:
```
RocCurveDisplay.from_estimator(dt_m,x_test,y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
```



[28]:
```
pred_dt = np.around(pred_dt)
print(metrics.classification_report(y_test,pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.90      0.81      0.85        43
           1       0.62      0.76      0.68        17

    accuracy                           0.80        60
   macro avg       0.76      0.79      0.77        60
weighted avg       0.82      0.80      0.81        60
```

1. We can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.82 in our case
2. accuracy predicted is this case = 81.67%

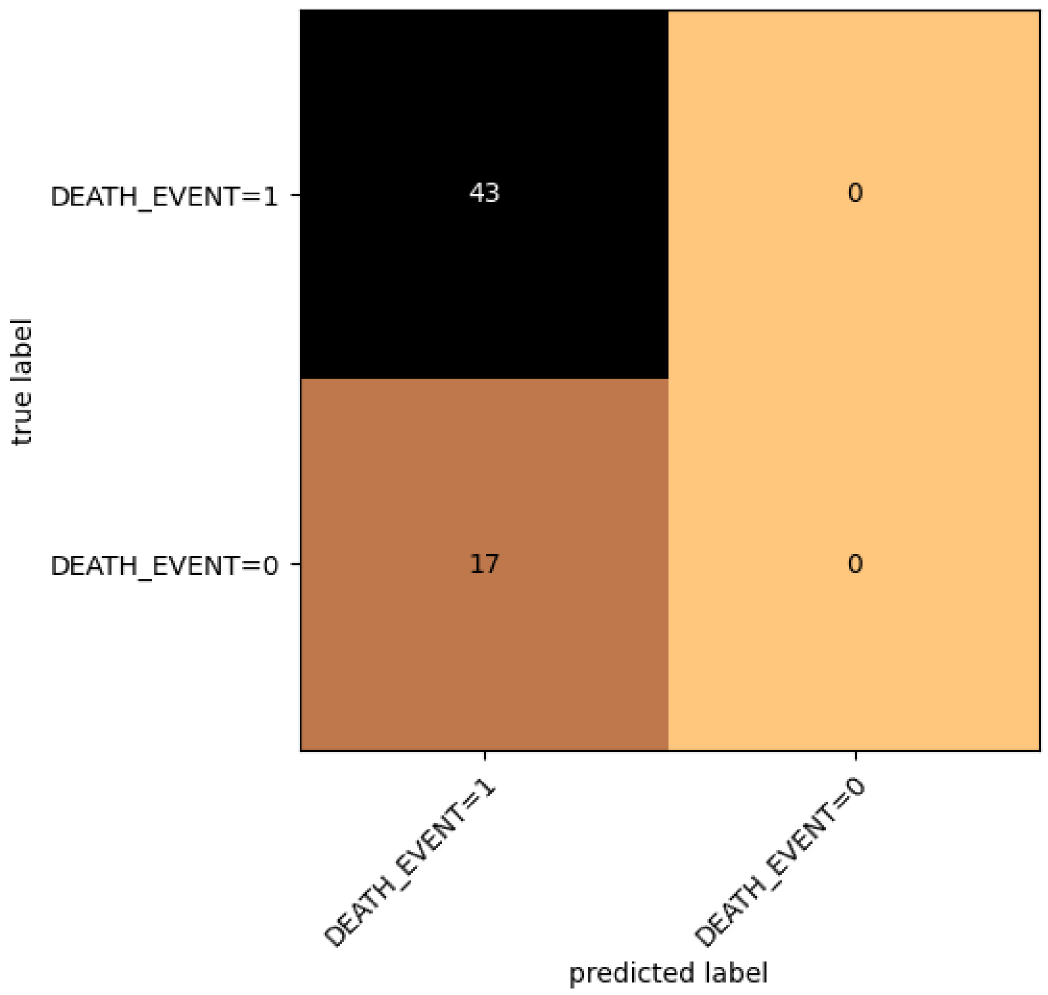
3. **36 out of 43 death count was predicted exactly and 7 was incorrectly forecsated**
4. **13 out of 17 values were predicted exactly and 4 was incorrectly forecasted**

1.9.4 4. Support Vector Machine:

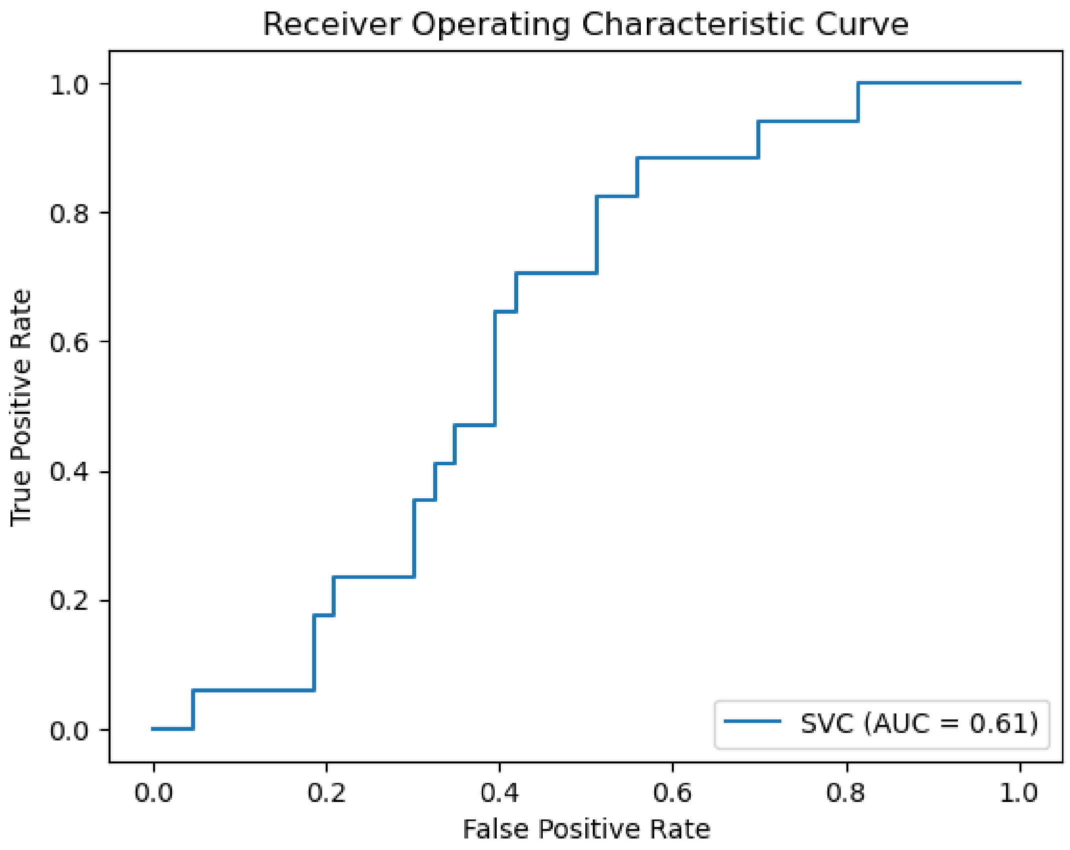
[29]:
```
svm_m = SVC(C=8.0,
    kernel='rbf',
    degree=3,
    gamma='scale',
    coef0=0.01,
    shrinking=True,
    probability=True,
    tol=0.1,
    cache_size=300,
    class_weight=None,
    verbose=False,
    max_iter=-1,
    decision_function_shape='ovo')
svm_m.fit(x_train,y_train)
pred_svm = svm_m.predict(x_test)
score_svm = round(accuracy_score(pred_svm,y_test)*100,2)
score_svm
```

[29]:
```
71.67
```

[30]:
```
conf_mat=confusion_matrix(y_test,pred_svm)
plot_confusion_matrix(conf_mat,class_names=['DEATH_EVENT=1','DEATH_EVENT=0'],figsize=(12,5),cm
  ↪
```



[31]:
```
RocCurveDisplay.from_estimator(svm_m,x_test,y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
```



[32]:
```
pred_svm = np.around(pred_svm)
print(metrics.classification_report(y_test,pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.72      1.00      0.83        43
           1       0.00      0.00      0.00        17

    accuracy                           0.72        60
   macro avg       0.36      0.50      0.42        60
weighted avg       0.51      0.72      0.60        60
```

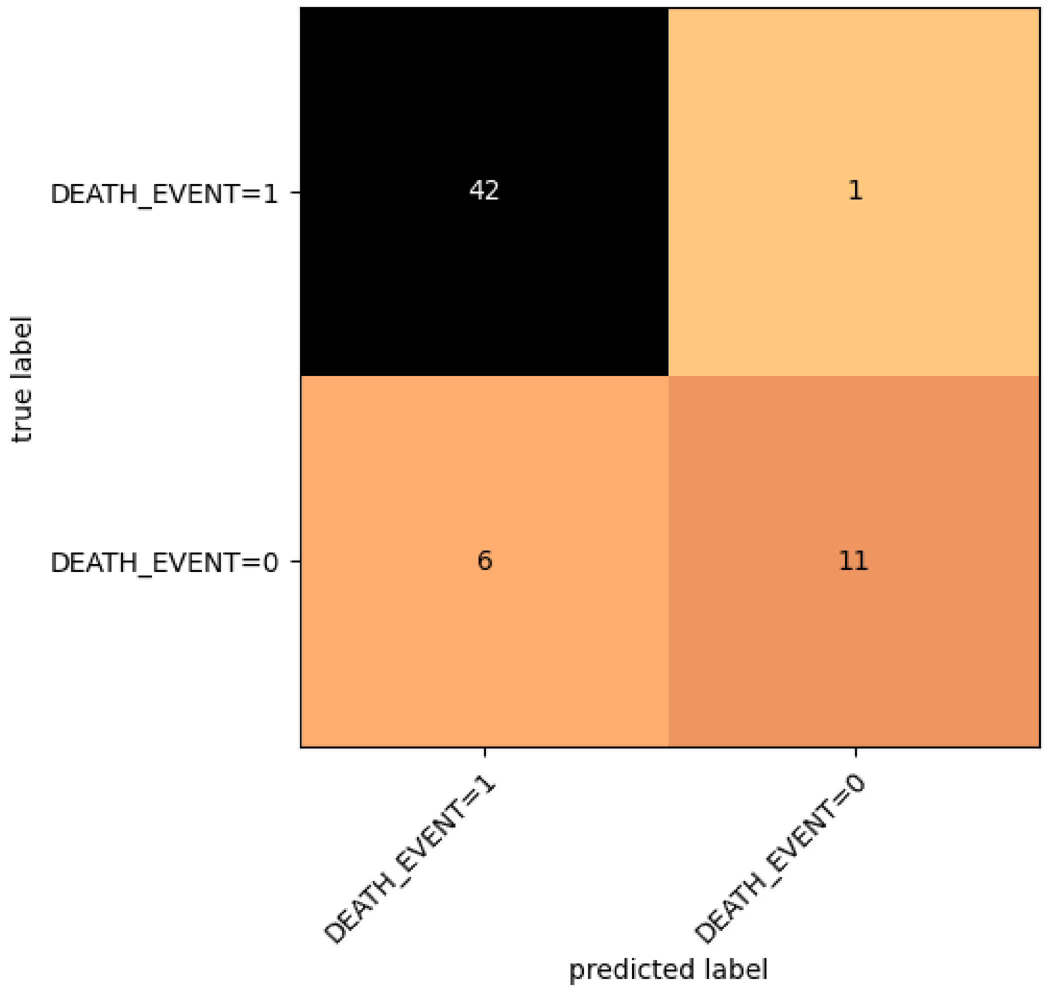
1. We can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.60 in our case
2. accuracy predicted is this case = 71.67%
3. **43 out of 43 death count was predicted exactly**
4. **17 out of 17 values were incorrectly forecasted**

1.9.5 Random Forest model:

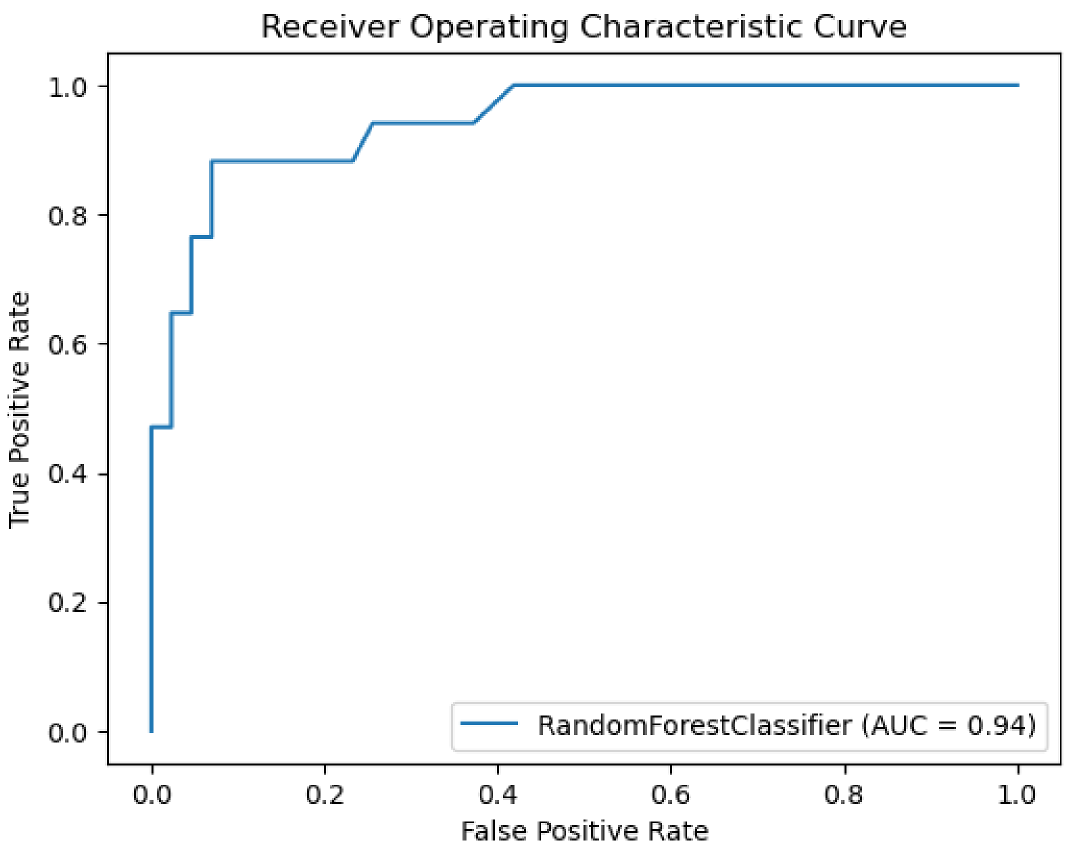
[33]:
```
rf_m=RandomForestClassifier()
rf_m.fit(x_train,y_train)
rf_m.predict(x_test)
pred_rf = rf_m.predict(x_test)
score_rf = round(accuracy_score(pred_rf,y_test)*100,2)
score_rf
```

[33]:
```
88.33
```

[34]:
```
conf_mat=confusion_matrix(y_test,pred_rf)
plot_confusion_matrix(conf_mat,class_names=['DEATH_EVENT=1','DEATH_EVENT=0'],figsize=(12,5),cm
  ↪
```



[35]:
```python
RocCurveDisplay.from_estimator(rf_m,x_test,y_test)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic Curve');
```



[36]:
```python
pred_rf = np.around(pred_rf)
print(metrics.classification_report(y_test,pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.88      0.98      0.92        43
           1       0.92      0.65      0.76        17

    accuracy                           0.88        60
   macro avg       0.90      0.81      0.84        60
weighted avg       0.89      0.88      0.88        60
```

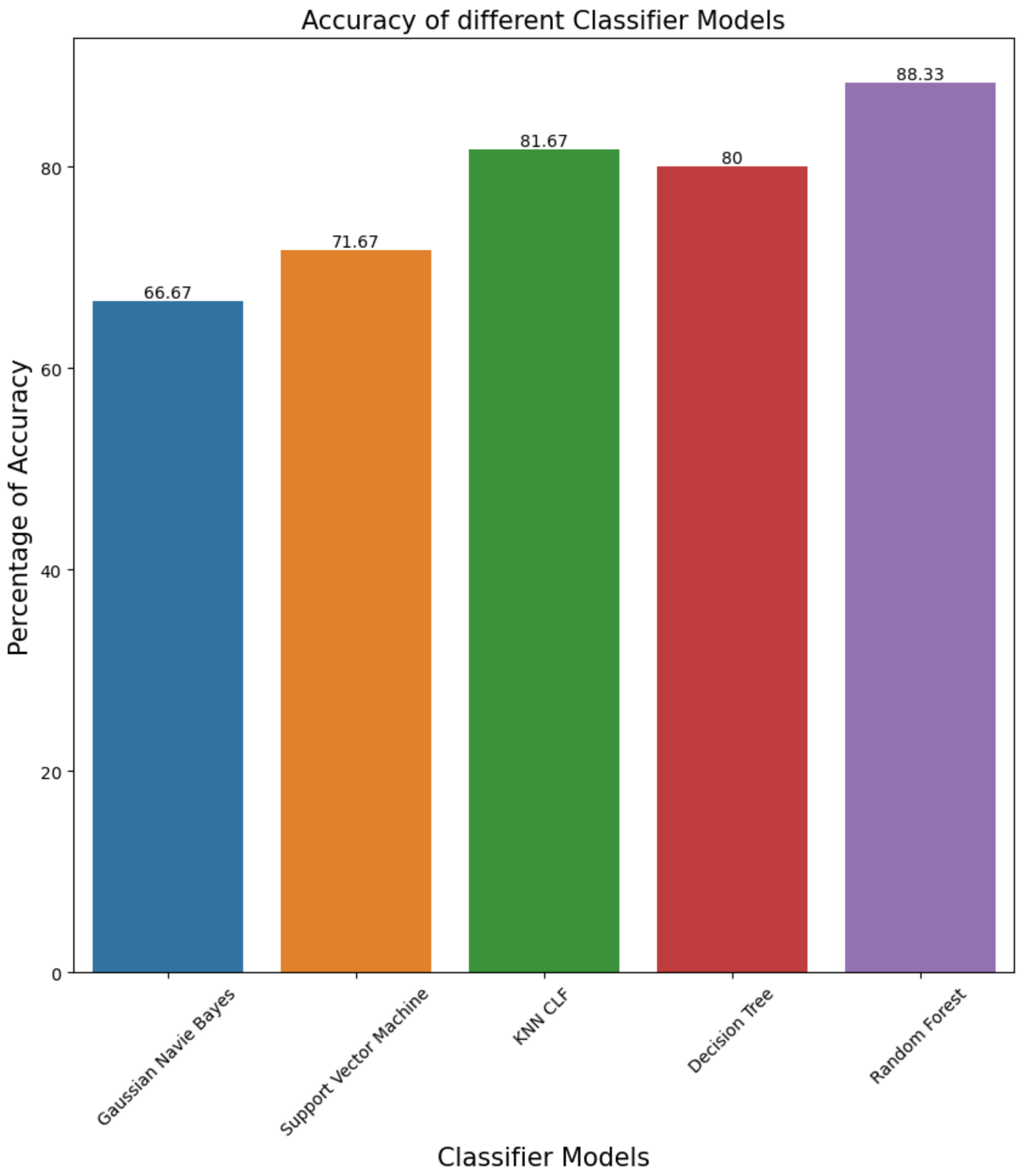
1. We can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.90 in our case
2. accuracy predicted is this case = 90%

3. **41 out of 43 death count was predicted exactly and 2 were incorrectly forecasted**
4. **13 out of 17 values was predicted exactly and 4 were incorrectly forecasted**

1.9.6 Results and Analysis :

[37]:
```
scores = [score_nb,score_svm,score_knn,score_dt,score_rf]
#Models = ["KNN CLF","Gaussian Navie Bayes","Decision Tree","Support Vector␣
↪Machine","Random Forest"]
```

[38]:
```
model_names = ['Gaussian Navie Bayes','Support Vector Machine','KNN␣
↪CLF','Decision Tree','Random Forest']
scores = [round(score, 3) for score in scores]

plt.figure(figsize=(10, 10))
ax = sns.barplot(x=model_names, y=scores)
ax.bar_label(ax.containers[0])
plt.xlabel('Classifier Models', fontsize = 15)
plt.ylabel('Percentage of Accuracy', fontsize = 15)
plt.title('Accuracy of different Classifier Models', fontsize = 15)
plt.xticks(fontsize = 10, horizontalalignment = 'center', rotation = 45)
plt.yticks(fontsize = 10)
plt.show()
```



1.10 CONCLUSION:

Best overall model seems to be the random forest trained on the oversampled dataset, that delivers the best results in terms of accuracy and f1 score.
For the models that allow it, it's possible to evaluate the ROC curve to select a threshold according to the main goal (minimize false positives or maximize true positives) but the results in the table are obtained by fixing the threshold at 0.5.
I have used almost all clasiification algorithm models to predict the accuracy of heart failure ac-

cording to the feature provided with dataset.
Random-forest makes the best model out of all.as 90% accuracy.
I also want to look into feature selection for logistic regression algorithms. I focused mainly on tuning my random forest algorithm here, but maybe I could get more consistent results from my logistic regression by applying feature selection beyond collinearity corrections.

1.10.1 GITHUB REPOSITORY URL

https://github.com/kavishant87/Supervised_Final_Project_5509

1.10.2 REFERENCES:

kagle references: https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data
medium: https://medium.com/@ammar.j.alashhab/using-machine-learning-algorithm-in-heart-failure
gridDB: https://griddb.net/en/blog/heart-failure-prediction-using-machine-learning-python-and-griddb
stackoverflow: https://stackoverflow.com/questions/54084452/how-to-plot-seaborn-pairplot-as-subplot
plotly: https://plotly.com/python/violin/