In [1]:
```
#import libraries

import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
%matplotlib inline
```

## PROJECT DESCRIPTION:

Heart failure occurs when the heart muscle doesn’t pump blood well as it should. Cardiovascular diseases (CVDs) are the number one cause of death globally, taking an estimated 17.9 million lives each year, which account to 31% of all deaths worldwide.

This project is focused on the 12 features that can be used to predict mortality by heart failure. Data was collected on these 12 features for the analysis.

## Description of attributes in failure dataset

- age: Age of the patient
- anaemia: Haemoglobin level of patient (Boolean)
- creatinine_phosphokinase: Level of the CPK enzyme in the blood (mcg/L)
- diabetes: If the patient has diabetes (Boolean)
- ejection_fraction: Percentage of blood leaving the heart at each contraction
- high_blood_pressure: If the patient has hypertension (Boolean)
- platelets: Platelet count of blood (kiloplatelets/mL)
- serum_creatinine: Level of serum creatinine in the blood (mg/dL)
- serum_sodium: Level of serum sodium in the blood (mEq/L)
- sex: Sex of the patient
- smoking: If the patient smokes or not (Boolean)
- time: Follow-up period (days)
- DEATH_EVENT: If the patient deceased during the follow-up period (Boolean)
- For attributes with boolean values; 0 means negative while 1 means positive

In [2]:
```python
#Let's read our data set

heart_failure = pd.read_csv('heart_failure_clinical_records_dataset.csv')
heart_failure
```

Out[2]:

| | age | anaemia | creatinine_phosphokinase | diabetes | ejection_fraction | high_blood_pressure | platelets | serum_creatinine | serum_sodium | sex |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 75.0 | 0 | 582 | 0 | 20 | 1 | 265000.00 | 1.9 | 130 | 1 |
| **1** | 55.0 | 0 | 7861 | 0 | 38 | 0 | 263358.03 | 1.1 | 136 | 1 |
| **2** | 65.0 | 0 | 146 | 0 | 20 | 0 | 162000.00 | 1.3 | 129 | 1 |
| **3** | 50.0 | 1 | 111 | 0 | 20 | 0 | 210000.00 | 1.9 | 137 | 1 |
| **4** | 65.0 | 1 | 160 | 1 | 20 | 0 | 327000.00 | 2.7 | 116 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **294** | 62.0 | 0 | 61 | 1 | 38 | 1 | 155000.00 | 1.1 | 143 | 1 |
| **295** | 55.0 | 0 | 1820 | 0 | 38 | 0 | 270000.00 | 1.2 | 139 | 0 |
| **296** | 45.0 | 0 | 2060 | 1 | 60 | 0 | 742000.00 | 0.8 | 138 | 0 |
| **297** | 45.0 | 0 | 2413 | 0 | 38 | 0 | 140000.00 | 1.4 | 140 | 1 |
| **298** | 50.0 | 0 | 196 | 0 | 45 | 0 | 395000.00 | 1.6 | 136 | 1 |

299 rows × 13 columns

In [3]:
```
#check information about our data set

heart_failure.info()
```

```
<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
```

In [49]:
```python
#checking the summary of our data, let's transpose for quick graps of our summary

heart_failure.describe().T
```

Out[49]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **age** | 299.0 | 60.833893 | 11.894809 | 40.0 | 51.0 | 60.0 | 70.0 | 95.0 |
| **anaemia** | 299.0 | 0.431438 | 0.496107 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **creatinine_phosphokinase** | 299.0 | 581.839465 | 970.287881 | 23.0 | 116.5 | 250.0 | 582.0 | 7861.0 |
| **diabetes** | 299.0 | 0.418060 | 0.494067 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **ejection_fraction** | 299.0 | 38.083612 | 11.834841 | 14.0 | 30.0 | 38.0 | 45.0 | 80.0 |
| **high_blood_pressure** | 299.0 | 0.351171 | 0.478136 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **platelets** | 299.0 | 263358.029264 | 97804.236869 | 25100.0 | 212500.0 | 262000.0 | 303500.0 | 850000.0 |
| **serum_creatinine** | 299.0 | 1.393880 | 1.034510 | 0.5 | 0.9 | 1.1 | 1.4 | 9.4 |
| **serum_sodium** | 299.0 | 136.625418 | 4.412477 | 113.0 | 134.0 | 137.0 | 140.0 | 148.0 |
| **sex** | 299.0 | 0.648829 | 0.478136 | 0.0 | 0.0 | 1.0 | 1.0 | 1.0 |
| **smoking** | 299.0 | 0.321070 | 0.467670 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **time** | 299.0 | 130.260870 | 77.614208 | 4.0 | 73.0 | 115.0 | 203.0 | 285.0 |
| **DEATH_EVENT** | 299.0 | 0.321070 | 0.467670 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |

In [5]:
```python
#check for the sum of null values

heart_failure.isnull().sum()
```

Out[5]:
```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

In [6]:
```
#check the size of the data

heart_failure.shape
```

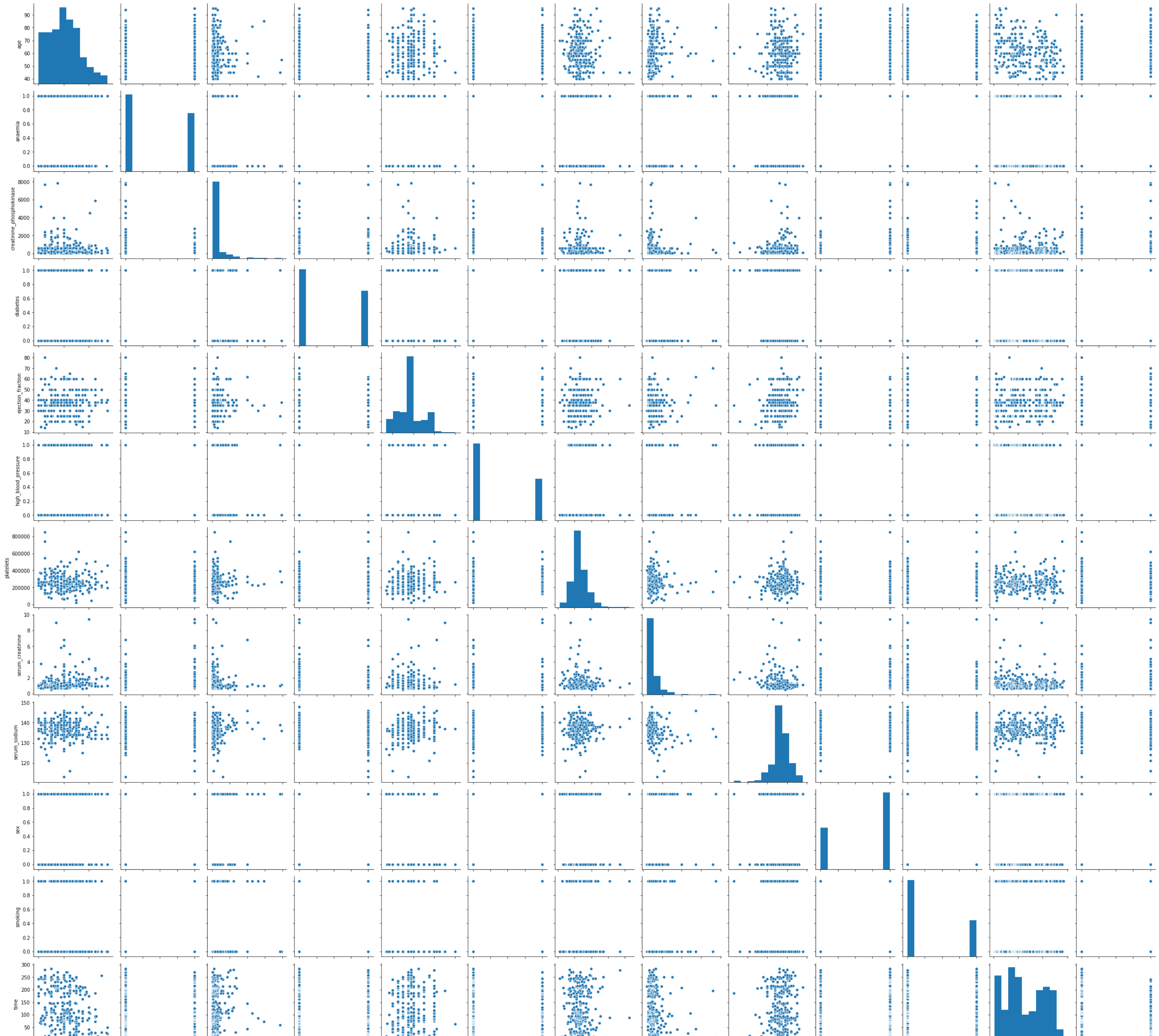
Out[6]:
```
(299, 13)
```

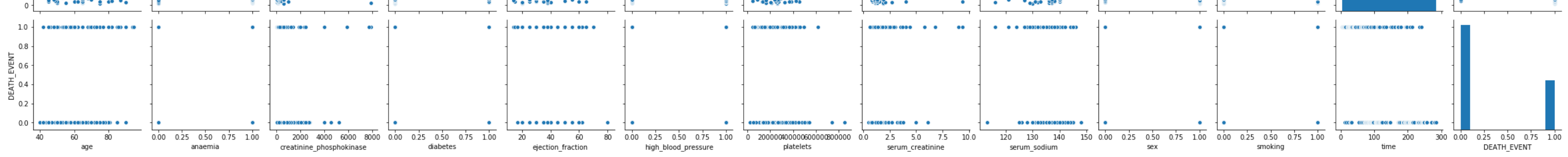
In [7]:
```
#To check for the unique columns in the data set

heart_failure.columns.unique
```

Out[7]:
```
<bound method Index.unique of Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')>
```

In [8]:
```
#comparing the correlation among features

sns.pairplot(heart_failure)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x23313601e48>
```

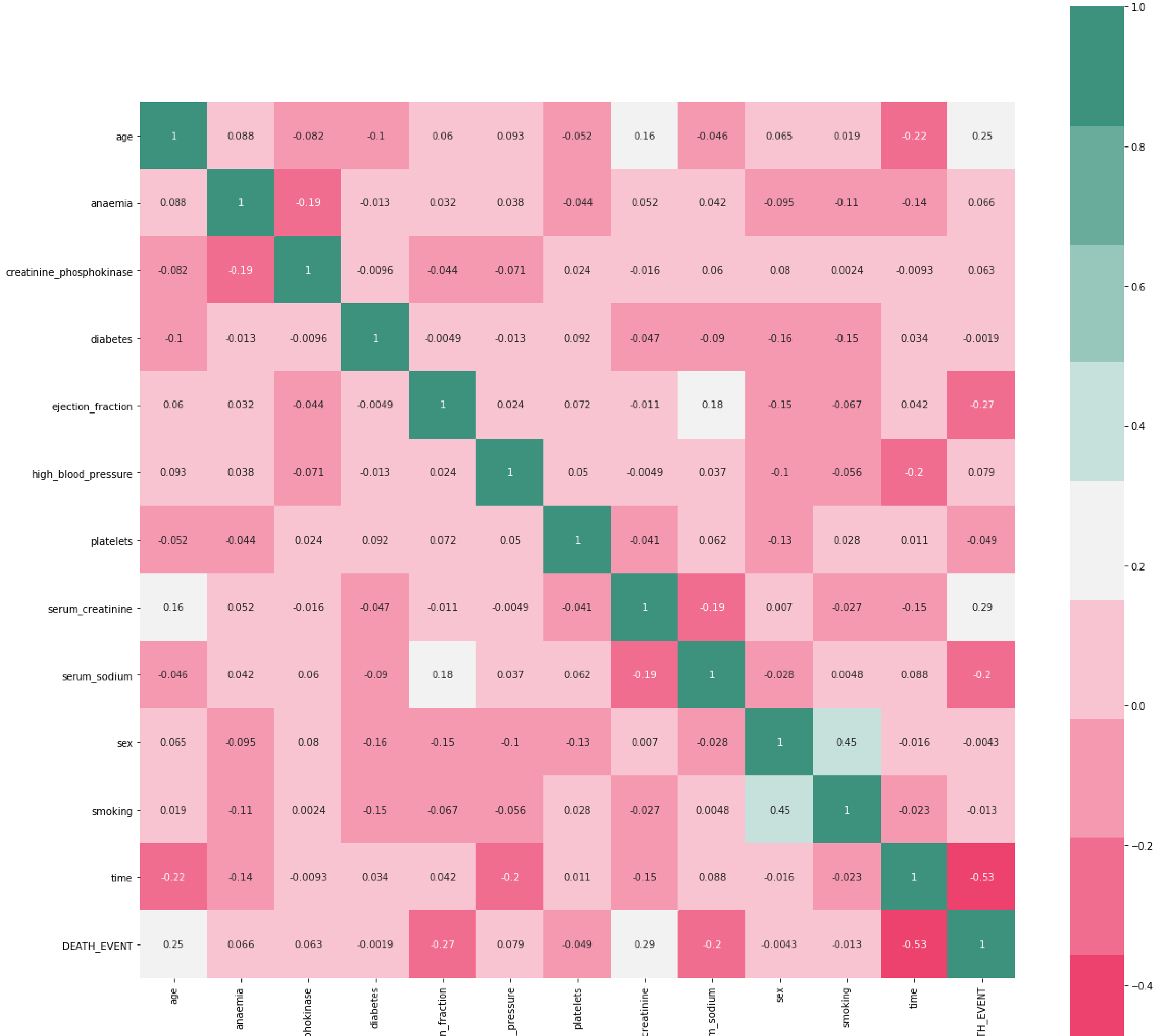

age
anaemia
creatinine_phosphokinase
diabetes
ejection_fraction
high_blood_pressure
platelets
serum_creatinine
serum_sodium
sex
smoking
time



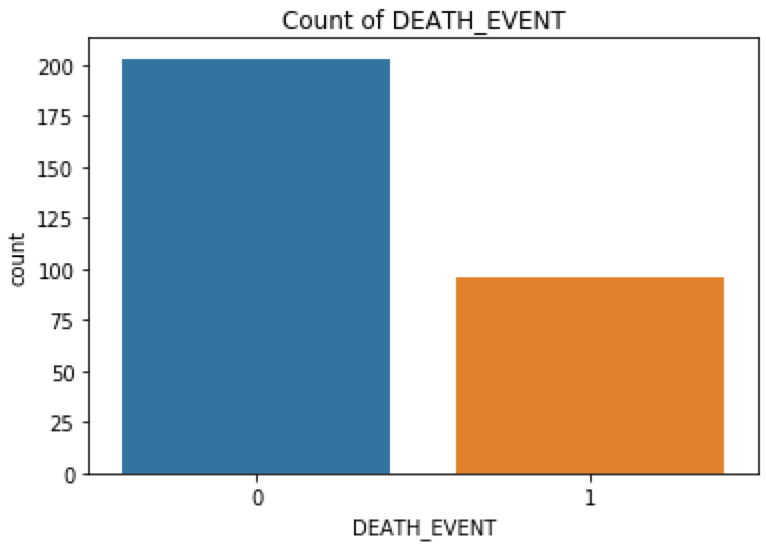
DEATH_EVENT
age
anaemia
creatinine_phosphokinase
diabetes
ejection_fraction
high_blood_pressure
platelets
serum_creatinine
serum_sodium
sex
smoking
time
DEATH_EVENT

```
In [9]: #checking for correlation among our features using heatmap to visualize

        cmap = sns.diverging_palette(2, 165, s=80, l=55, n=9)
        corrmat = heart_failure.corr()
        plt.subplots(figsize=(20,20))
        sns.heatmap(corrmat,cmap= cmap,annot=True, square=True)
```

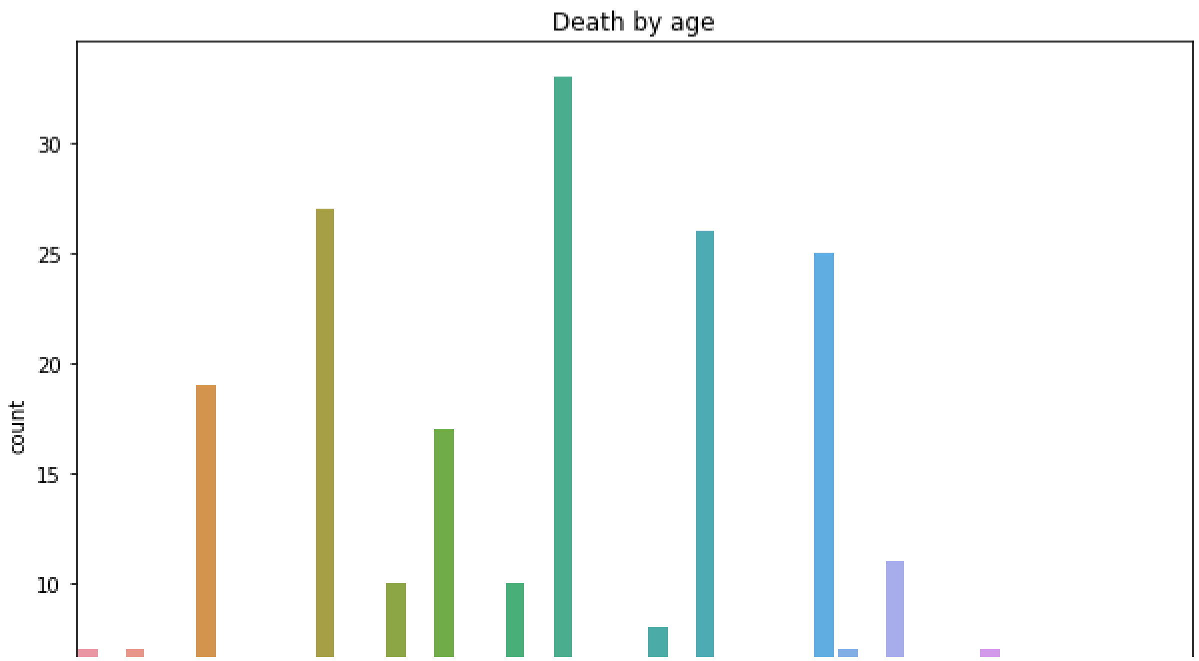
```
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x2331a4a9888>
```


age
anaemia
creatinine_phosphokinase
diabetes
ejection_fraction
high_blood_pressure
platelets
serum_creatinine
serum_sodium
sex
smoking
time
DEATH_EVENT
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
In [10]: #getting the total counts of our label (Death event)

         sns.countplot(x='DEATH_EVENT', data = heart_failure).set(title='Count of DEATH_EVENT')

Out[10]: [Text(0.5, 1.0, 'Count of DEATH_EVENT')]
```

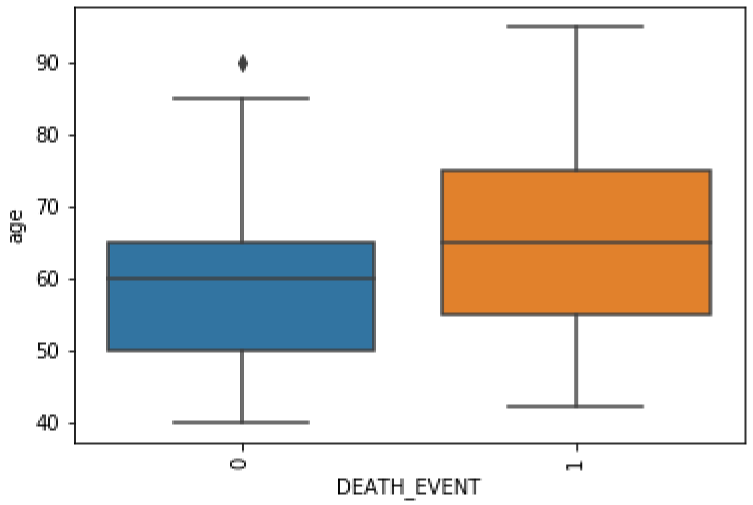


In [11]:
```python
#to obtain the death event by age

plt.figure(figsize =(10,7))
sns.countplot(x='age', data = heart_failure).set(title='Death by age')
plt.xticks(rotation=90)
plt.show()
```



- Most of the death occurred between the ages of 50 and 60

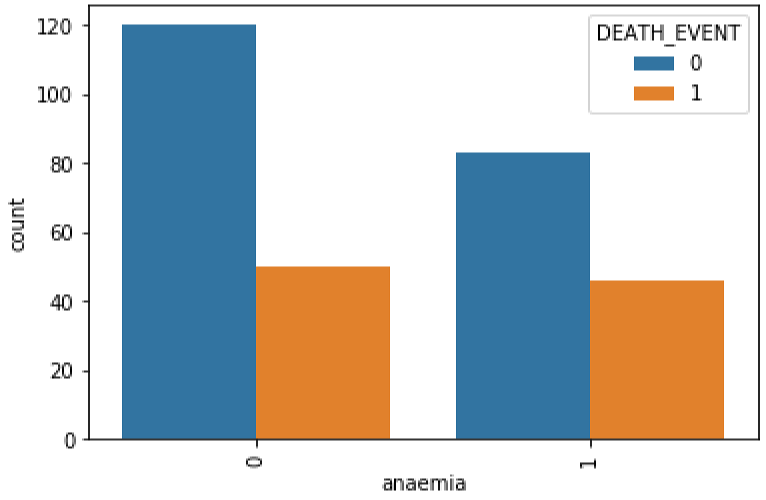
```
In [12]: #box plot to determine possible outliers

def plt_compare(continoues_features):
    sns.boxplot(y=heart_failure[continoues_features],x=heart_failure.DEATH_EVENT)
    plt.xticks(rotation=90)
    plt.show()
attributes=['age','creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine', 'serum_sodium', 'ti
for attribute in attributes:

    plt_compare(attribute)
```

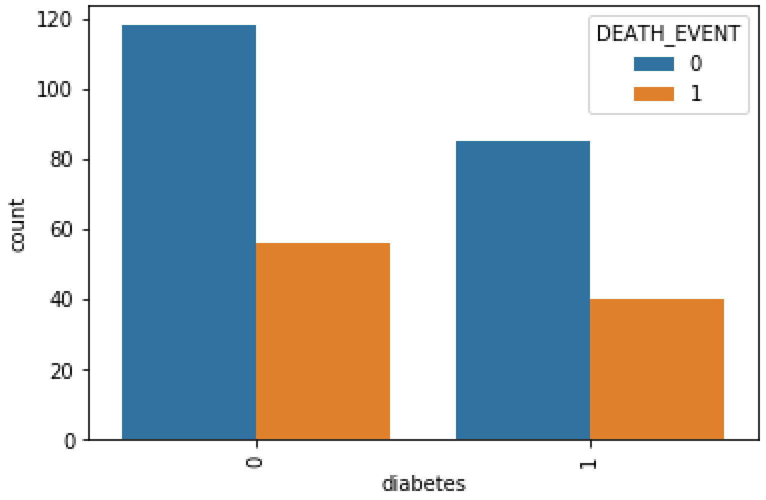


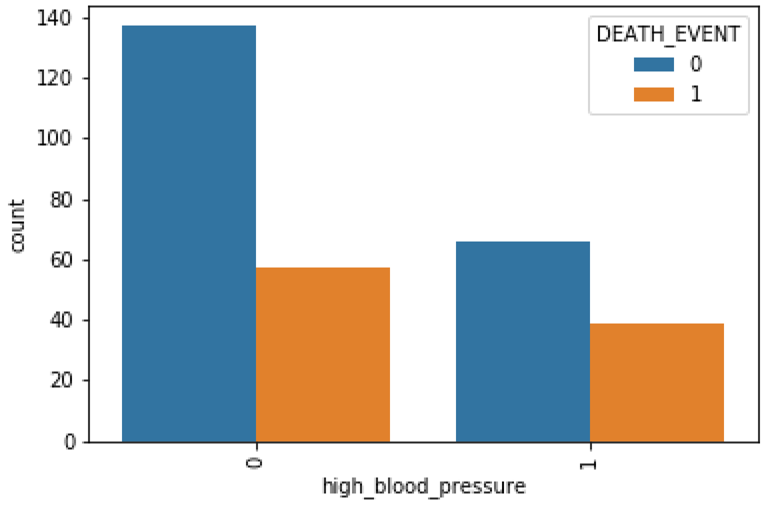
For our continuous features,

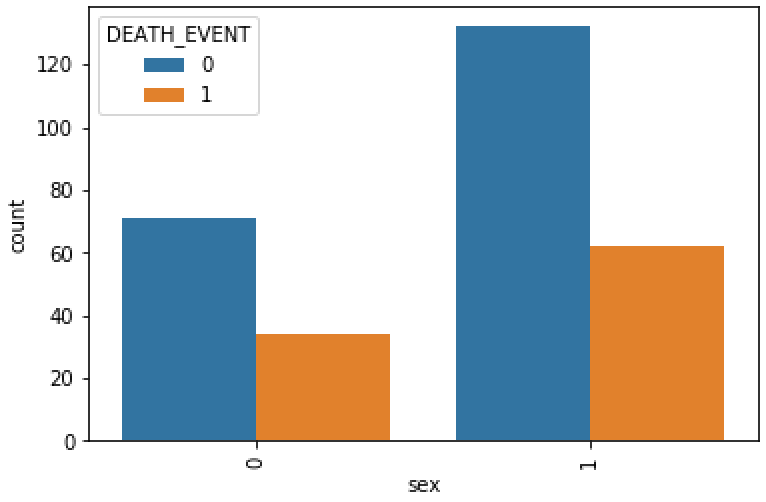
- we noticed all the plots had outliers however they will still be used in preprocessing

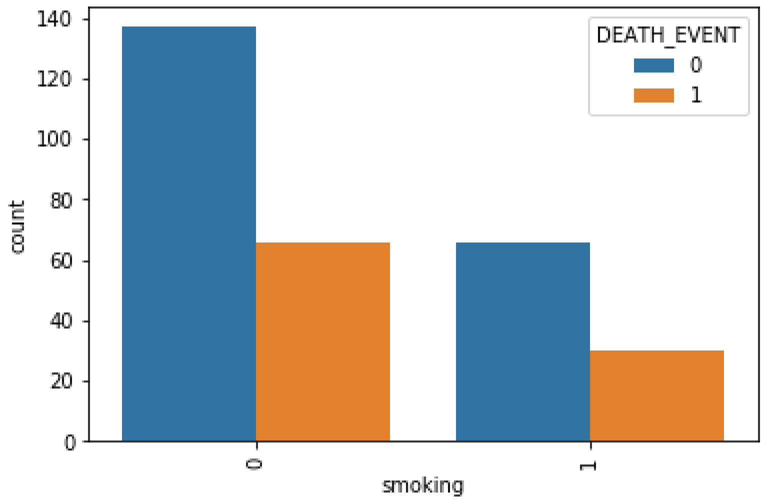
```
In [13]: #To observe the effect of the categorical features on the death event caused by heart failure

         def plt_check(categorical_features):
             sns.countplot(heart_failure[categorical_features],hue=heart_failure.DEATH_EVENT)
             plt.xticks(rotation=90)
             plt.show()
         attributes=['anaemia', 'diabetes','high_blood_pressure', 'sex', 'smoking']

         for attribute in attributes:

             plt_check(attribute)
```






DEATH_EVENT
0
1
count
high_blood_pressure


DEATH_EVENT
0
1
count
sex


DEATH_EVENT
0
1
count
smoking

As seen, although, all categorical features contribute to the death event but appear not to have a strong significancance in contributing to the target variable, we can decide to drop them as features and still be fine but we will work with them regardless.

In [14]:
```
# Split features into features and label  (x,y)
X= heart_failure.drop([ 'DEATH_EVENT'], axis = 'columns')
X
```

Out[14]:

| | age | anaemia | creatinine_phosphokinase | diabetes | ejection_fraction | high_blood_pressure | platelets | serum_creatinine | serum_sodium | sex | s |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 75.0 | 0 | 582 | 0 | 20 | 1 | 265000.00 | 1.9 | 130 | 1 | |
| **1** | 55.0 | 0 | 7861 | 0 | 38 | 0 | 263358.03 | 1.1 | 136 | 1 | |
| **2** | 65.0 | 0 | 146 | 0 | 20 | 0 | 162000.00 | 1.3 | 129 | 1 | |
| **3** | 50.0 | 1 | 111 | 0 | 20 | 0 | 210000.00 | 1.9 | 137 | 1 | |
| **4** | 65.0 | 1 | 160 | 1 | 20 | 0 | 327000.00 | 2.7 | 116 | 0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **294** | 62.0 | 0 | 61 | 1 | 38 | 1 | 155000.00 | 1.1 | 143 | 1 | |
| **295** | 55.0 | 0 | 1820 | 0 | 38 | 0 | 270000.00 | 1.2 | 139 | 0 | |
| **296** | 45.0 | 0 | 2060 | 1 | 60 | 0 | 742000.00 | 0.8 | 138 | 0 | |
| **297** | 45.0 | 0 | 2413 | 0 | 38 | 0 | 140000.00 | 1.4 | 140 | 1 | |
| **298** | 50.0 | 0 | 196 | 0 | 45 | 0 | 395000.00 | 1.6 | 136 | 1 | |

299 rows × 12 columns

```
In [15]: y = heart_failure.DEATH_EVENT
         y
```

```
Out[15]: 0      1
         1      1
         2      1
         3      1
         4      1
               ..
         294    0
         295    0
         296    0
         297    0
         298    0
         Name: DEATH_EVENT, Length: 299, dtype: int64
```

```
In [16]: from sklearn import preprocessing
```

```
In [17]: from sklearn.preprocessing import StandardScaler
```

In [18]:
```
col_names = list(X.columns)
s_scaler = preprocessing.StandardScaler()
X_scaled= s_scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=col_names)
X_scaled.describe().T
```

Out[18]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **age** | 299.0 | 5.265205e-16 | 1.001676 | -1.754448 | -0.828124 | -0.070223 | 0.771889 | 2.877170 |
| **anaemia** | 299.0 | 3.594301e-16 | 1.001676 | -0.871105 | -0.871105 | -0.871105 | 1.147968 | 1.147968 |
| **creatinine_phosphokinase** | 299.0 | 3.713120e-18 | 1.001676 | -0.576918 | -0.480393 | -0.342574 | 0.000166 | 7.514640 |
| **diabetes** | 299.0 | 1.113936e-16 | 1.001676 | -0.847579 | -0.847579 | -0.847579 | 1.179830 | 1.179830 |
| **ejection_fraction** | 299.0 | 3.341808e-18 | 1.001676 | -2.038387 | -0.684180 | -0.007077 | 0.585389 | 3.547716 |
| **high_blood_pressure** | 299.0 | -4.841909e-16 | 1.001676 | -0.735688 | -0.735688 | -0.735688 | 1.359272 | 1.359272 |
| **platelets** | 299.0 | 1.009969e-16 | 1.001676 | -2.440155 | -0.520870 | -0.013908 | 0.411120 | 6.008180 |
| **serum_creatinine** | 299.0 | -2.227872e-18 | 1.001676 | -0.865509 | -0.478205 | -0.284552 | 0.005926 | 7.752020 |
| **serum_sodium** | 299.0 | -8.627435e-16 | 1.001676 | -5.363206 | -0.595996 | 0.085034 | 0.766064 | 2.582144 |
| **sex** | 299.0 | -5.940993e-18 | 1.001676 | -1.359272 | -1.359272 | 0.735688 | 0.735688 | 0.735688 |
| **smoking** | 299.0 | -3.861645e-17 | 1.001676 | -0.687682 | -0.687682 | -0.687682 | 1.454161 | 1.454161 |
| **time** | 299.0 | -1.069379e-16 | 1.001676 | -1.629502 | -0.739000 | -0.196954 | 0.938759 | 1.997038 |

In [19]:
```
# import train, test, split library so as to split data into training and testing data
from sklearn.model_selection import train_test_split
```

In [20]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=101)
```

In [22]:
```
from sklearn.svm import SVC
```

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [24]:
```
from sklearn.linear_model import LogisticRegression
```

```
In [25]: # creating an array for our model of choice
         models=[]
         models.append(('LR',LogisticRegression(solver='liblinear',multi_class='ovr')))
         models.append(('RFC',RandomForestClassifier()))
         models.append(('SVM',SVC(gamma='auto')))
```

```
In [26]: from sklearn.model_selection import StratifiedKFold
```

```
In [27]: from sklearn.model_selection import cross_val_score
```

```
In [28]: results=[]
         names=[]
         for name,model in models:
             kfold=StratifiedKFold(n_splits=20)#random_state=1
             cv_results=cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
             results.append(cv_results)
             names.append(name)
             print('%s: %f (%f)' %(name,cv_results.mean(),cv_results.std()))

         LR: 0.812879 (0.084091)
         RFC: 0.844318 (0.096794)
         SVM: 0.660606 (0.035339)
```

- CV (cross validation): helps to evaluate and compare across different models
- Stratified Kfold: helps us to divide each of our classifier category in a uniform way
- kfold: we used 20 splits and took the average of each classifier alongside their standard deviation

```
In [29]: from sklearn.linear_model import LogisticRegression
```

```
In [30]: lr = LogisticRegression()
```

```
In [31]: lr.fit(X_train,y_train)

Out[31]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)
```

```
In [32]: #lets now predict the death event from heart failure
         y_pred = lr.predict(X_test)
```

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

```
Out[33]: 0.84
```

```
In [34]: from sklearn.metrics import precision_score, recall_score, confusion_matrix, classification_report, accuracy_score, f1_
```

```
In [35]: print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89        55
           1       0.70      0.70      0.70        20

    accuracy                           0.84        75
   macro avg       0.80      0.80      0.80        75
weighted avg       0.84      0.84      0.84        75
```

- accuracy is a score used to evaluate the models performance, the higher it is the better
- Recall measures the model ability to correctly predict the true positive values
- Precison is the ratio of true positive to the sum of both true and false positive
- f score combines precision and recall into one metric, its value should be closest to 1
- support is a number of actual occurrences of each class in the dataset

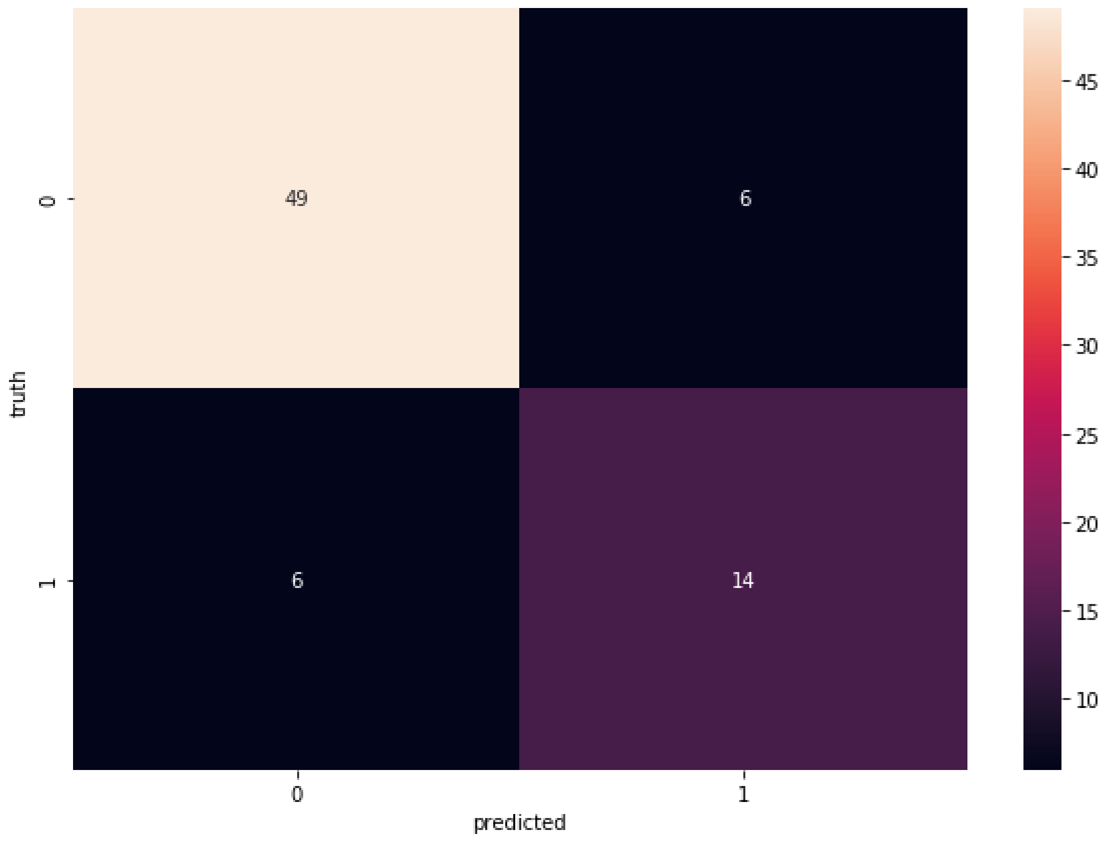
```
In [36]: from sklearn.metrics import accuracy_score, confusion_matrix
```

```
In [37]: # using cofusion metrics to check the behaviour of our linear regression model
         cm = confusion_matrix(y_test, y_pred)
         cm
```

```
Out[37]: array([[49,  6],
                [ 6, 14]], dtype=int64)
```

```
In [38]: plt.figure(figsize=(10,7))
         sns.heatmap(cm, annot = True)
         plt.xlabel('predicted')
         plt.ylabel('truth')
```

```
Out[38]: Text(69.0, 0.5, 'truth')
```



```
In [39]: svc_model= SVC(kernel = 'linear')
         svc_model.fit(X_train, y_train)
```

```
Out[39]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)
```

```
In [40]: prediction= svc_model.predict(X_test)
```

In [41]:
```
print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))
```

```
0.8125
0.8533333333333334
```

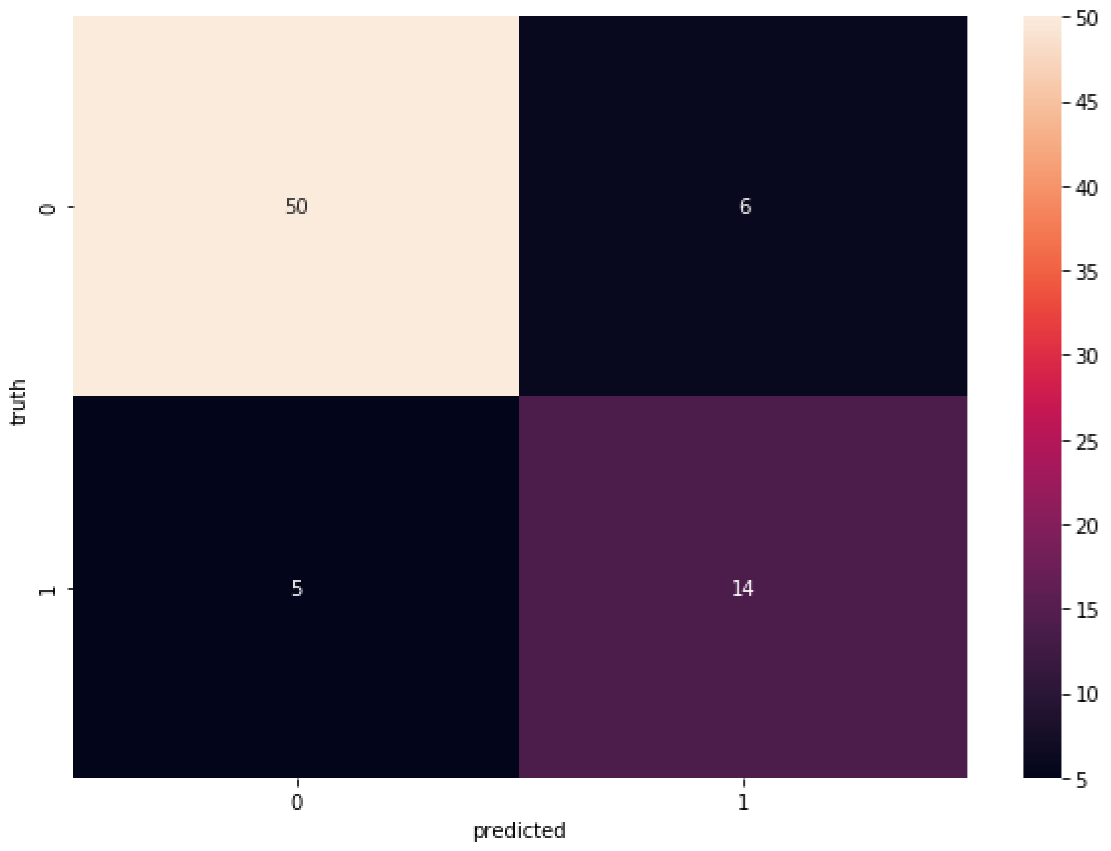
In [50]:
```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.89      0.91      0.90        55
           1       0.74      0.70      0.72        20

    accuracy                           0.85        75
   macro avg       0.81      0.80      0.81        75
weighted avg       0.85      0.85      0.85        75
```

In [42]:
```
# using cofusion metrics to check the behaviour of svm model
cm2 = confusion_matrix(prediction, y_test)
cm2
```

Out[42]:
```
array([[50,  6],
       [ 5, 14]], dtype=int64)
```

```
In [43]: plt.figure(figsize=(10,7))
         sns.heatmap(cm2, annot = True)
         plt.xlabel('predicted')
         plt.ylabel('truth')

Out[43]: Text(69.0, 0.5, 'truth')
```



```
In [44]: rf = RandomForestClassifier()
         rf.fit(X_train,y_train)
         pred_y=rf.predict(X_test)

In [45]: rf.score(X_test,y_test)

Out[45]: 0.8666666666666667
```

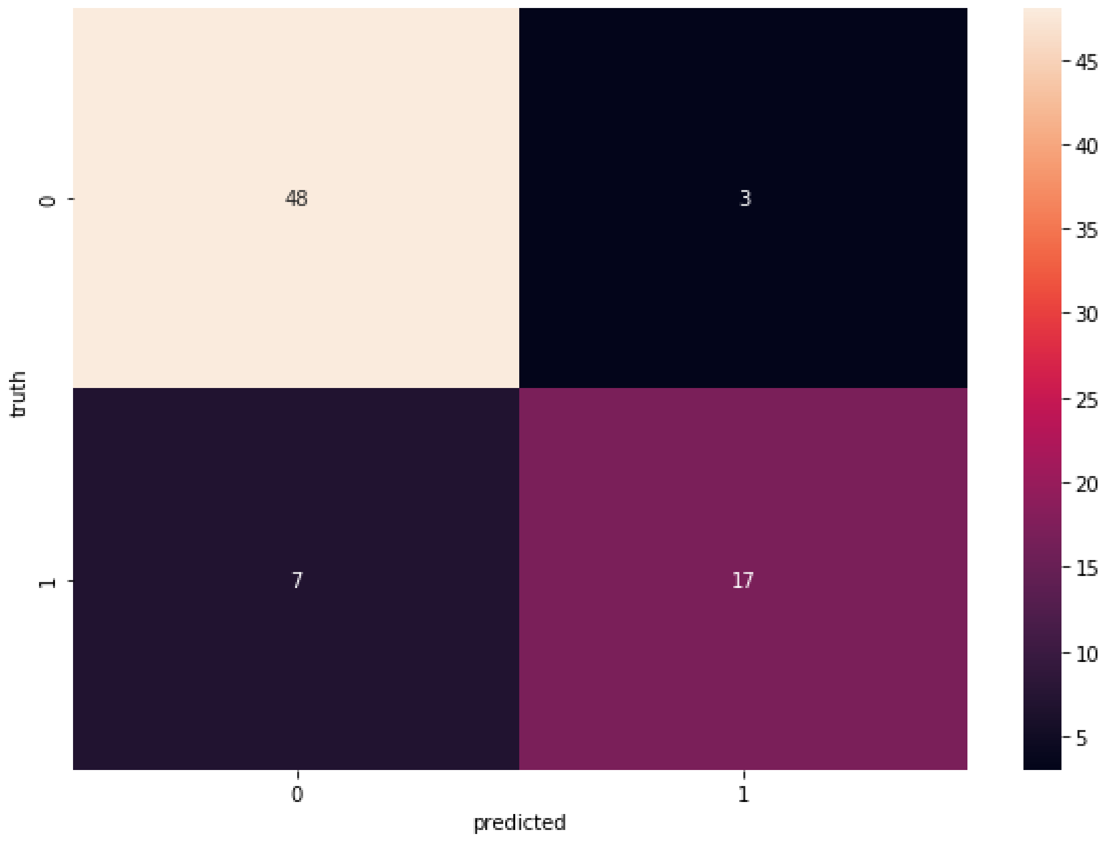
```
In [51]: print(classification_report(y_test, pred_y))
```

```
              precision    recall  f1-score   support

           0       0.94      0.87      0.91        55
           1       0.71      0.85      0.77        20

    accuracy                           0.87        75
   macro avg       0.82      0.86      0.84        75
weighted avg       0.88      0.87      0.87        75
```

```
In [46]: # using cofusion metrics to check the behaviour of our Random forest classifier model
         cm3 = confusion_matrix(pred_y, y_test)
         cm3
```

```
Out[46]: array([[48,  3],
                [ 7, 17]], dtype=int64)
```

```
In [47]: plt.figure(figsize=(10,7))
         sns.heatmap(cm3, annot = True)
         plt.xlabel('predicted')
         plt.ylabel('truth')

Out[47]: Text(69.0, 0.5, 'truth')
```



3 models were used to predict the death event by heart failure. Confusion metrics was made for all the 3 models and the random forest classifier appear to give the best prediction and behaviour.