Capstone Project

In [3]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn import preprocessing
```

Introduction/Business Problem

Cardiovascular diseases are the top cause of death globally claiming approximately 17.9 million lives a year. This is approximately 31% of all deaths worlwide. Those with the disease or at high risk of the disease need early detection in order to implement behavorial changes and/or undergo medical procedures/pharmacetuical regimins to manage/mitigate the risk of heart failure.

Doctors are a key group of stakeholders interested in addressing this problem. Machine learning models can utilize available data across 12 features that are known to be correlated with death by heart failure to predict an impending death event. With this prediction, Doctors can initiate new cardivascular management protocols and/or elevate exisiting ones to prevent an impending death event.

Data

*From Kaggle (https://www.kaggle.com/andrewmvd/heart-failure-clinical-data)

The data set contains 12 features commonly used to predict heart failure. There is a mixture of continous and discrete features which will be used to predict an dependent variable: **death_event**

Continuous

- age
- creatinine_phosphokinase
- ejection_fraction
- platelets
- serum_creatinine
- serum_sodium
- time

Discrete

- anaemia
- diabetes
- high_blood_pressure
- sex
- smoking
- death_event

Data Cleaning

After examining the feature set, I decided to drop the time feature. The time feature is the time (hour of the day) of the death_event. This feature is not useful in predicting whether or not a death_event **will** occur since it already has happened.

In [4]:
```
df = pd.read_csv('heart_failure_clinical_records_dataset.csv')
df.drop(columns = 'time', inplace = True)
df.head()
```

Out[4]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressu
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	

Methodology

Exploration

First, I want to get a sense of the data. For each feature, I would like to explore the range (max,min), standard deviation, mean, etc.

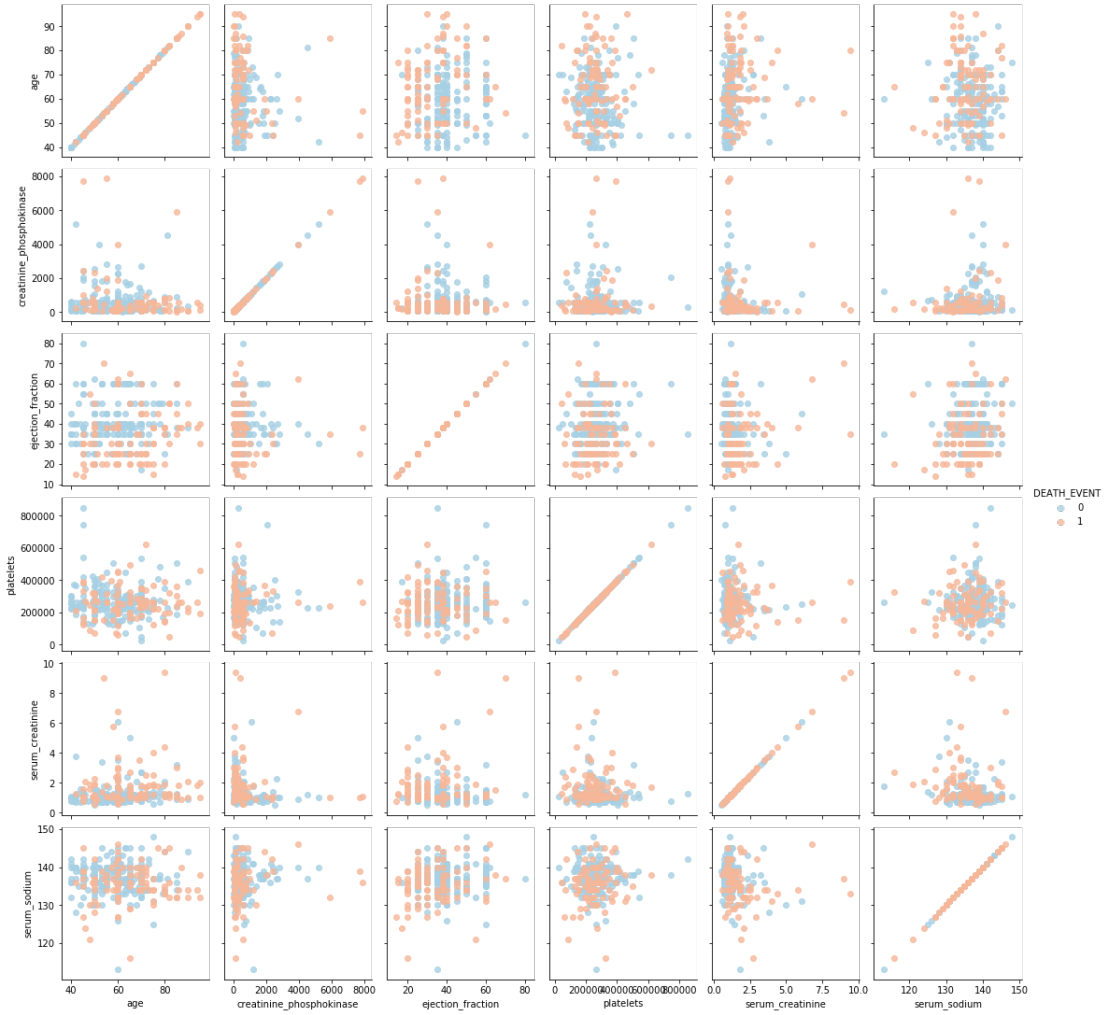
In [6]:
```
df.describe()
```

Out[6]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	hig
count	299.000000	299.000000	299.000000	299.000000	299.000000	
mean	60.833893	0.431438	581.839465	0.418060	38.083612	
std	11.894809	0.496107	970.287881	0.494067	11.834841	
min	40.000000	0.000000	23.000000	0.000000	14.000000	
25%	51.000000	0.000000	116.500000	0.000000	30.000000	
50%	60.000000	0.000000	250.000000	0.000000	38.000000	
75%	70.000000	1.000000	582.000000	1.000000	45.000000	
max	95.000000	1.000000	7861.000000	1.000000	80.000000	

Next, I would like to visualize the data to identify any patterns in the continuous features.

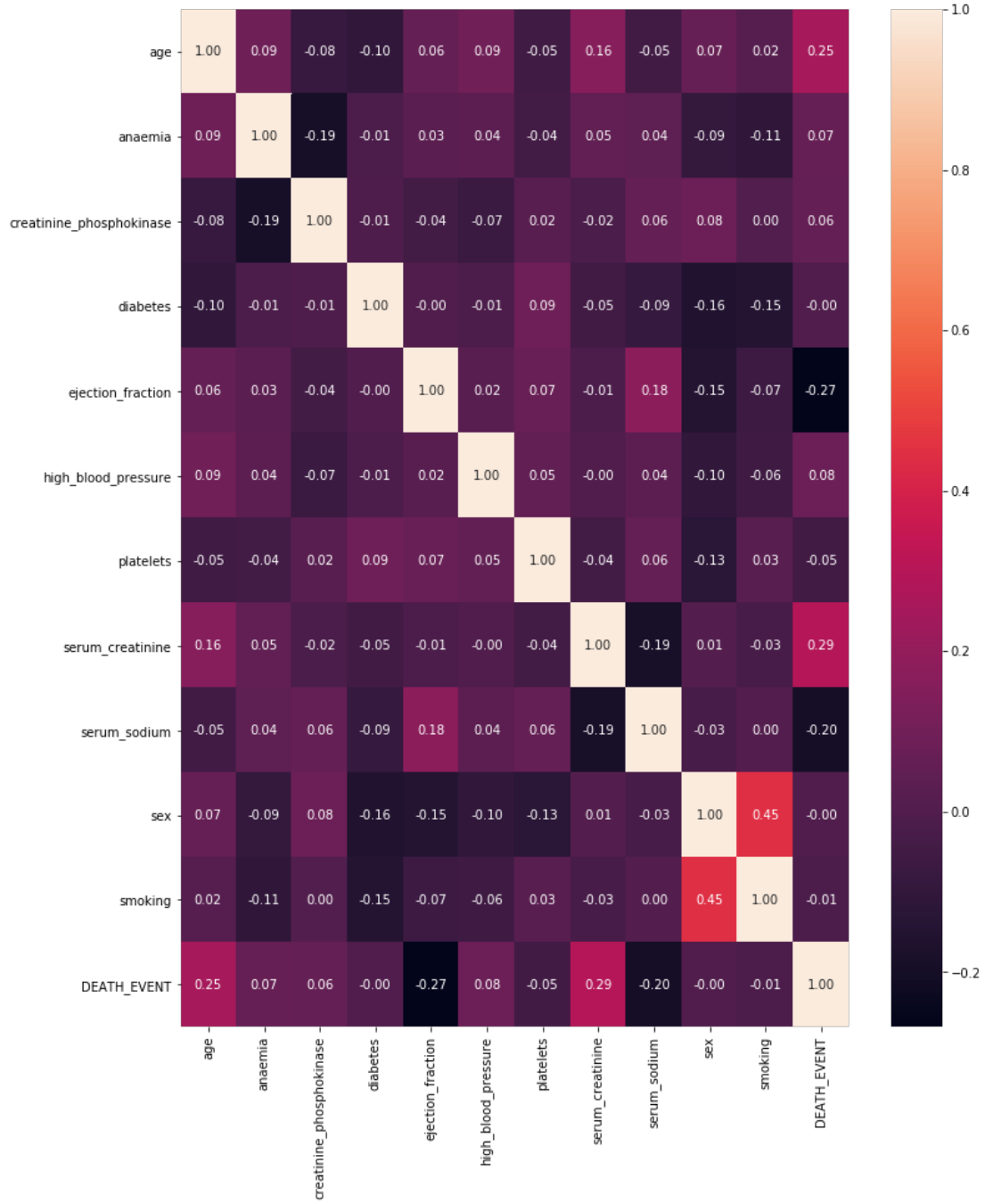
In [7]:
```
g = sns.PairGrid(df, vars=['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine', 'serum_sodium'],
                 hue='DEATH_EVENT', palette='RdBu_r')
g.map(plt.scatter, alpha=0.8)
g.add_legend();
```



Comparing each feature against itself, no clear pattern emerges.

Perhaps some features (both continous and discrete) are more correlated with the dependent variable (DEATH_EVENT) than others?

In [36]:
```
correlation = df.corr()
plt.figure(figsize=(12,15))
sns.heatmap(correlation,annot=True,fmt=".2f")

plt.show()
```



Sex and smoking are the most highly correlated features at .45. Serum_creatinine and age are most correlated to DEATH_EVENT directly at .29 and .25 respectively. Ejection_fraction appears to be negatively correlated to DEATH_EVENT at -.27.

Choice of Model

There does not appear to be a signficant linear relationship between the features or between the features and DEATH_EVENT. Predicitng DEATH_EVENT appears to be a classification problem where DEATH_EVENT is either 1 (someone died) or 0 (someone is still alive). This problem doesn't lend itself to linear regression since the dependent variable is not continuous.

The models I will compare are the various classification models KNN, SVM, Logistic, and Decision Tree in order to determine which is the most accurate.

I do not know which model will be most accurate, but perhaps the Logistic Regression model will be most useful as it also offers insight into the probability of the prediction. Doctors and patients would want to know if the probability is high or low for a given predicted death event.

Feature Selection and Labels

In [11]:
```
Feature = df[['age', 'creatinine_phosphokinase', 'ejection_fraction
', 'platelets', 'serum_creatinine', 'serum_sodium', 'anaemia', 'sex
', 'smoking', 'diabetes', 'high_blood_pressure']]

X = Feature
X[0:5]
```

Out[11]:

	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_s
0	75.0	582	20	265000.00	1.9	
1	55.0	7861	38	263358.03	1.1	
2	65.0	146	20	162000.00	1.3	
3	50.0	111	20	210000.00	1.9	
4	65.0	160	20	327000.00	2.7	

In [12]:
```
y = df['DEATH_EVENT'].values
y[0:5]
```

Out[12]:
```
array([1, 1, 1, 1, 1])
```

Normalization

In [46]:
```
X= preprocessing.StandardScaler().fit(X).transform(X)
```

Train, Test, Split

In [48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_siz
e=0.2, random_state=4)
```

KNN

In [49]:
```
from sklearn.neighbors import KNeighborsClassifier
k = 2
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
yhat = neigh.predict(X_test)
yhat[0:5]
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neig
h.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.7866108786610879
Test set Accuracy:  0.7333333333333333
```

Find the best K

In [51]:
```
Ks = 10
mean_acc = np.zeros((Ks-1))
std_acc = np.zeros((Ks-1))
ConfustionMx = [];
for n in range(1,Ks):
    neigh = KNeighborsClassifier(n_neighbors = n).fit(X_train,y_tra
in)
    yhat=neigh.predict(X_test)
    mean_acc[n-1] = metrics.accuracy_score(y_test, yhat)
    std_acc[n-1]=np.std(yhat==y_test)/np.sqrt(yhat.shape[0])

mean_acc

print( "The best accuracy was with", mean_acc.max(), "with k=", mea
n_acc.argmax()+1)
```

```
The best accuracy was with 0.7333333333333333 with k= 2
```

In [52]:
```
neigh_best_k = KNeighborsClassifier(n_neighbors = mean_acc.argmax()
+ 1).fit(X,y)
```

How accurate was KNN?

In [20]:
```
from sklearn.metrics import jaccard_similarity_score
from sklearn.metrics import f1_score

predKNN = neigh_best_k.predict(X_test)
jaccard_KNN = jaccard_similarity_score(y_test, predKNN)
f1_KNN = f1_score(y_test, predKNN, average='weighted')

print ('jaccard_KNN = {0}'.format(jaccard_KNN))
print ('f1_KNN = {0}'.format(f1_KNN))
```

```
jaccard_KNN = 0.75
f1_KNN = 0.6796536796536796

/Users/thaddeuscsiwinski/opt/miniconda3/lib/python3.7/site-packag
es/sklearn/metrics/_classification.py:664: FutureWarning: jaccard
_similarity_score has been deprecated and replaced with jaccard_s
core. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification task
s.
  FutureWarning)
```

The accuracy of the KNN model was 75% and 68% for Jaccard and F1 scores respectively.

SVM

In [53]:
```
from sklearn import svm

clf = svm.SVC(kernel='rbf')
clf.fit(X, y)
```

Out[53]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, c
oef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kerne
l='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=
True,
    tol=0.001, verbose=False)
```

How accurate was SVM?

In [35]:
```
predSVM = clf.predict(X_test)

jaccard_SVM = jaccard_similarity_score(y_test, predSVM)
f1_SVM = f1_score(y_test, predSVM, average='weighted')

print ('jaccard_SVM = {0}'.format(jaccard_SVM))
print ('f1_SVM = {0}'.format(f1_SVM))
```

```
jaccard_SVM = 0.8333333333333334
f1_SVM = 0.8101472995090016

/Users/thaddeuscsiwinski/opt/miniconda3/lib/python3.7/site-packag
es/sklearn/metrics/_classification.py:664: FutureWarning: jaccard
_similarity_score has been deprecated and replaced with jaccard_s
core. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification task
s.
  FutureWarning)
```

The accuracy of the SVM model was 83% and 81% for Jaccard and F1 scores respectively. SVM performed signficantly better than KNN.

Logistic Regression

In [54]:
```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(C=0.01, solver='liblinear').fit(X,y)
```

How accurate was Logistic Regression?

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

predLR = LR.predict(X_test)
LR_prob = LR.predict_proba(X_test)

jaccard_LR = jaccard_similarity_score(y_test, predLR)
f1_LR = f1_score(y_test, predLR, average='weighted')
log_loss_LR = log_loss(y_test, LR_prob)

print ('jaccard_LR = {0}'.format(jaccard_LR))
print ('f1_LR = {0}'.format(f1_LR))
print ('log_loss_LR = {0}'.format(log_loss_LR))
```

```
jaccard_LR = 0.75
f1_LR = 0.7094736842105263
log_loss_LR = 0.5795495323852766

/Users/thaddeuscsiwinski/opt/miniconda3/lib/python3.7/site-packag
es/sklearn/metrics/_classification.py:664: FutureWarning: jaccard
_similarity_score has been deprecated and replaced with jaccard_s
core. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification task
s.
  FutureWarning)
```

The accuracy of the Logistic Regression model was 75% and 71% for Jaccard and F1 scores respectively. The log loss was .579. A log loss close to 0 indicates a higher probabilty that the prediction was correct.

.579 indicates that the model predicts correctly a little less than half the time which isn't too comforting for patients and doctors when the stakes are death. The model performed similarily to KNN, but didn't perform as well as SVM.

Decision Tree

In [56]:
```
from sklearn.tree import DecisionTreeClassifier

heartTree = DecisionTreeClassifier(criterion="entropy", max_depth =
4)
heartTree.fit(X,y)
```

Out[56]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterio
n='entropy',
                       max_depth=4, max_features=None, max_leaf_n
odes=None,
                       min_impurity_decrease=0.0, min_impurity_sp
lit=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='dep
recated',
                       random_state=None, splitter='best')
```

How accurate was the Decision Tree?

In [45]:
```
predTree = heartTree.predict(X_test)

jaccard_Tree = jaccard_similarity_score(y_test, predTree)
f1_Tree = f1_score(y_test, predTree, average='weighted')

print ('jaccard_Tree = {0}'.format(jaccard_Tree))
print ('f1_Tree = {0}'.format(f1_Tree))
```

```
jaccard_Tree = 0.8333333333333334
f1_Tree = 0.8356867779204106

/Users/thaddeuscsiwinski/opt/miniconda3/lib/python3.7/site-packag
es/sklearn/metrics/_classification.py:664: FutureWarning: jaccard
_similarity_score has been deprecated and replaced with jaccard_s
core. It will be removed in version 0.23. This implementation has
surprising behavior for binary and multiclass classification task
s.
  FutureWarning)
```

The accuracy of the Decision Tree model was 83% and 84% for Jaccard and F1 scores respectively. The Decision Tree beats out SVM on the F1 score by 2%.

The 2 best models

Decision Tree and SVM

Let's see how the Decision Tree and SVM models compare

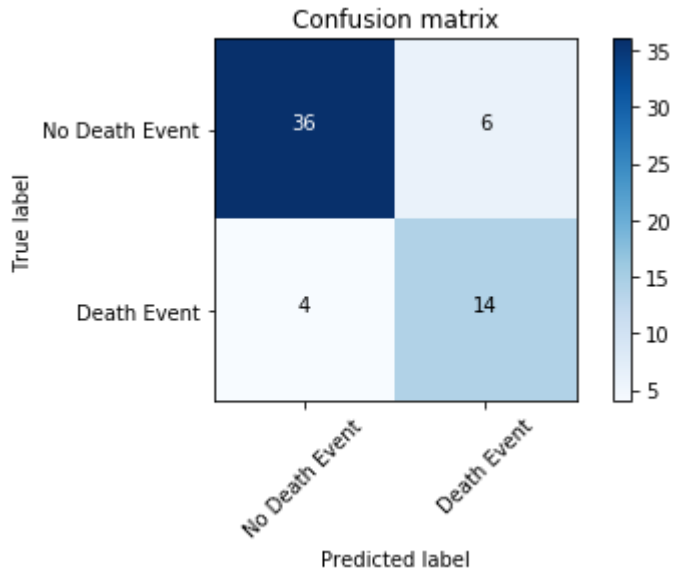
Doctors want a model to predict accurately, but false positives (predicting death event and no death occurs) is preferrable to false negatives (predicting no death event and the patient dies). With an incorrect death prediction at least preventative actions can be take **just in case**.

In [71]:
```
from sklearn.metrics import classification_report, confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shap
e[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Decision Tree Confusion Matrix

In [72]:
```
cnf_matrix = confusion_matrix(y_test, predTree, labels=[0,1])
np.set_printoptions(precision=2)

print (classification_report(y_test, predTree))


plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['No Death Event','Death
Event'],normalize= False,  title='Confusion matrix')
```

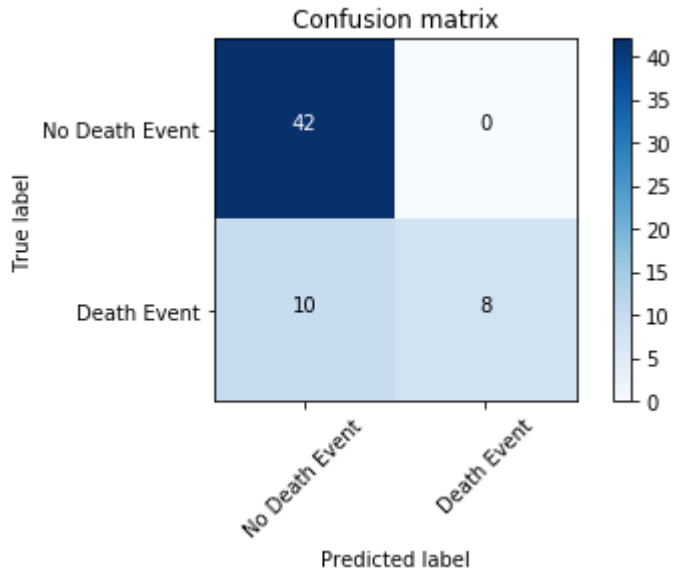
```
              precision    recall  f1-score   support

           0       0.90      0.86      0.88        42
           1       0.70      0.78      0.74        18

    accuracy                           0.83        60
   macro avg       0.80      0.82      0.81        60
weighted avg       0.84      0.83      0.84        60

Confusion matrix, without normalization
[[36  6]
 [ 4 14]]
```



SVM Confusion Matrix

In [73]:
```
cnf_matrix_2 = confusion_matrix(y_test, predSVM, labels=[0,1])
np.set_printoptions(precision=2)

print (classification_report(y_test, predSVM))


plt.figure()
plot_confusion_matrix(cnf_matrix_2, classes=['No Death Event','Deat
h Event'],normalize= False,  title='Confusion matrix')
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89        42
           1       1.00      0.44      0.62        18

    accuracy                           0.83        60
   macro avg       0.90      0.72      0.75        60
weighted avg       0.87      0.83      0.81        60

Confusion matrix, without normalization
[[42  0]
 [10  8]]
```



Results and Discussion

The goal was to identify the ideal model for predicting a death event based on a feature set of discrete and continous variables known to be indicators of heart failure. The dependent variable (DEATH_EVENT) is a discrete variable that a classification model woud be suited to predict.

I compared KNN, SVM, Logistic Regression, and Decision Tree models. SVM and the Decision tree were the most accurate.

SVM
jaccard_SVM = 0.8333333333333334
f1_SVM = 0.8101472995090016

Decision Tree
jaccard_Tree = 0.8333333333333334
f1_Tree = 0.8356867779204106

These scores were incredibly close with the Decision Tree having a slightly higher F1 score. Since they were so close, I wanted to look at the confusion matrix for each model to see if there were any signficant difference in false negatives.

A false positive on a death prediction is not ideal, but doctors can formulate mitigation plans for their already high risk patients. Even though the death prediction turns out to be incorrect, any lifestyle changes on the part of the patient to improve their health will be beneficial. The false negative is of greater concern. When the model predicts no death event and the patient ends up dying, no action can be taken after the fact.

When comparing the confusion matrices of SVM and Decision Tree, we see that the SVM model predicted 10 patients will **not** have a death event when in fact they **did** have a death event compared to the Decision Tree which made that mistake for only 4 patients.

This breaks the tie and the **Decision Tree** has proven to be the best model at predicting a death event.

Conclusion

The purpose of this project was aid doctors in predicting a death event due to heart failure in their patients and selecting a model that was most accurate in those predictions. Using classification models, I determined that a Decision Tree was most useful and could predict death events with 83% accuracy. This model also minimized the false negative predictions which could be the most damaging to a patient.

A follow up project would be to compare various feature selections and see how they change the performance of each model in hopes of improving the winner (Decision Tree) and perhaps improving Logistic Regression to a useful point where probablities associated with each prediction may become useful.

In [ ]: