In [1]:
```
#Importing required libraries
import pandas as pd
import seaborn as sns
from colorama import Fore, Back, Style
import matplotlib.pyplot as plt
import scipy.stats
import sklearn.metrics as metrics
import warnings

import plotly.graph_objs as go
import plotly.express as px

from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from mlxtend.plotting import plot_confusion_matrix

#Filter out warnings
warnings.filterwarnings("ignore")
from matplotlib.pyplot import figure
```

Data Cleaning and Preparation - Start

In [2]:
```
#Loading CSV as a dataframe
df = pd.read_csv('heart_failure_clinical_records_dataset.csv')

#Executing a shape and head to see the nubmer of columns and lines and a 5 line sample
print(df.shape)
df.head()
```

```
(299, 13)
```

Out[2]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	t
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	

Confirming if any data additional data cleaning steps needed

In [3]:
```
#Checking types for columns and all are numeric so no conversions will be needed.
df.dtypes
```

Out[3]:
```
age                         float64
anaemia                       int64
creatinine_phosphokinase      int64
diabetes                      int64
ejection_fraction             int64
high_blood_pressure           int64
platelets                   float64
serum_creatinine            float64
serum_sodium                  int64
sex                           int64
smoking                       int64
time                          int64
DEATH_EVENT                   int64
dtype: object
```

In [4]:
```
#Confirming if any of the columns have nulls none do so no droping or filling will be needed
df.isnull().sum()
```

Out[4]:
```
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
```

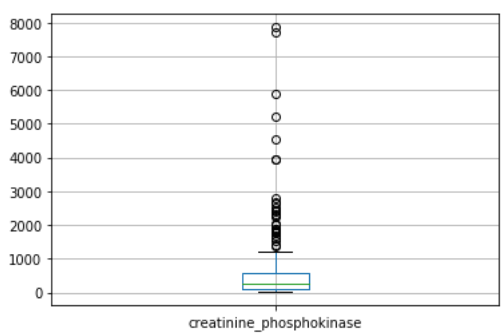
In [5]:
```
#To visualize the numerical columns for potentional issues for cleaning
df.describe()
```

Out[5]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodiu
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.00000	299.0000
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358.029264	1.39388	136.6254
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804.236869	1.03451	4.4124
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	0.50000	113.0000
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.000000	0.90000	134.0000
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.000000	1.10000	137.0000
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.000000	1.40000	140.0000
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	9.40000	148.0000

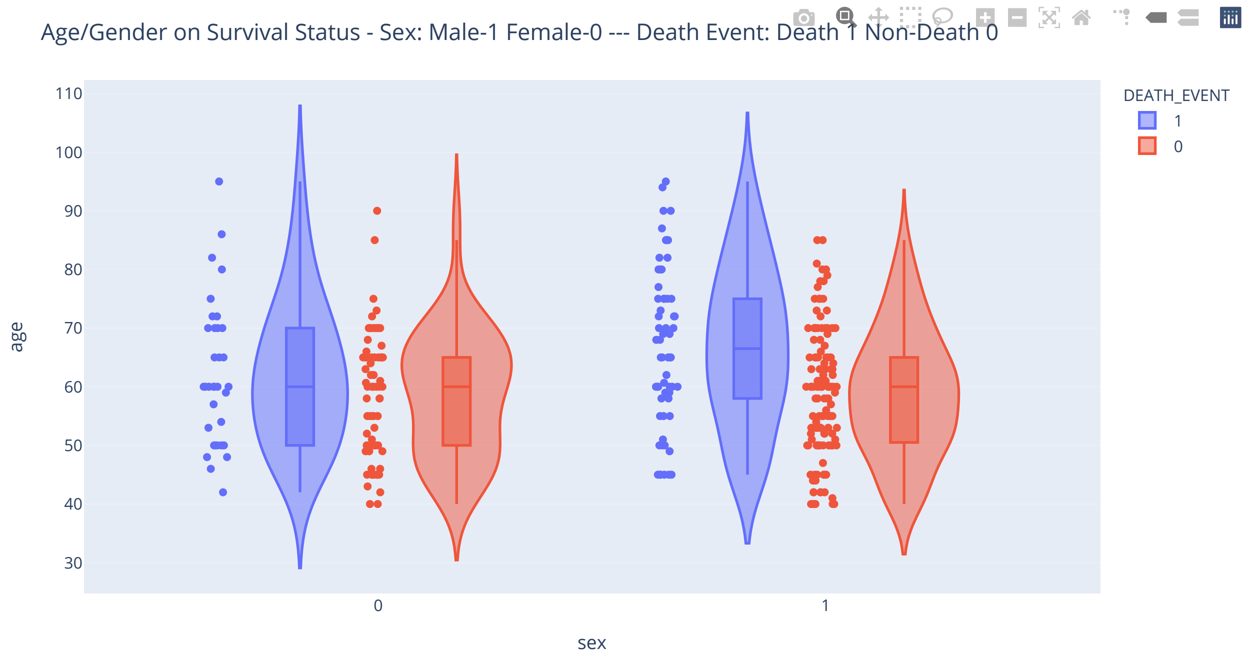
In [6]:
```
#Checking the column creatinine_phosphokinase for outliers. Since no one single outlier exists, will leave the column as is
boxplot = df.boxplot(column=['creatinine_phosphokinase'])
boxplot
```

Out[6]: `<matplotlib.axes._subplots.AxesSubplot at 0x27452348608>`

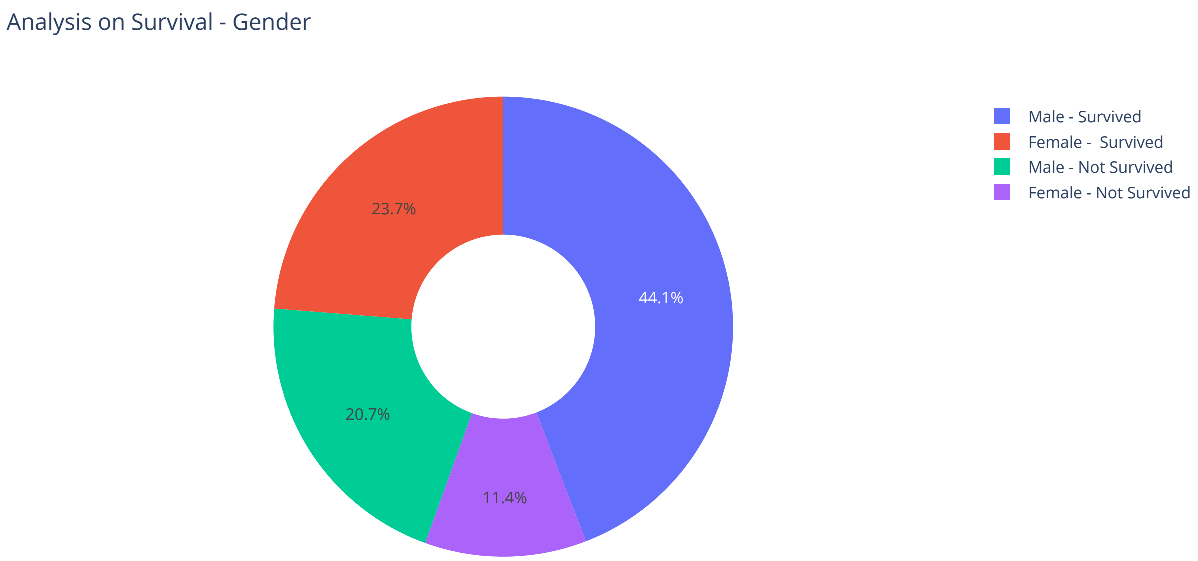


Initial Question of Interest - How does sex and age reflect on death events

In [7]:
```
#Create a violin chart to show the relationship of sex, age and death events
fig = px.violin(df, y="age", x="sex", color="DEATH_EVENT", box=True, points="all", hover_data=df.columns)
fig.update_layout(title_text="Age/Gender on Survival Status - Sex: Male-1 Female-0 --- Death Event: Death 1 Non-Death 0")
fig.show()
```



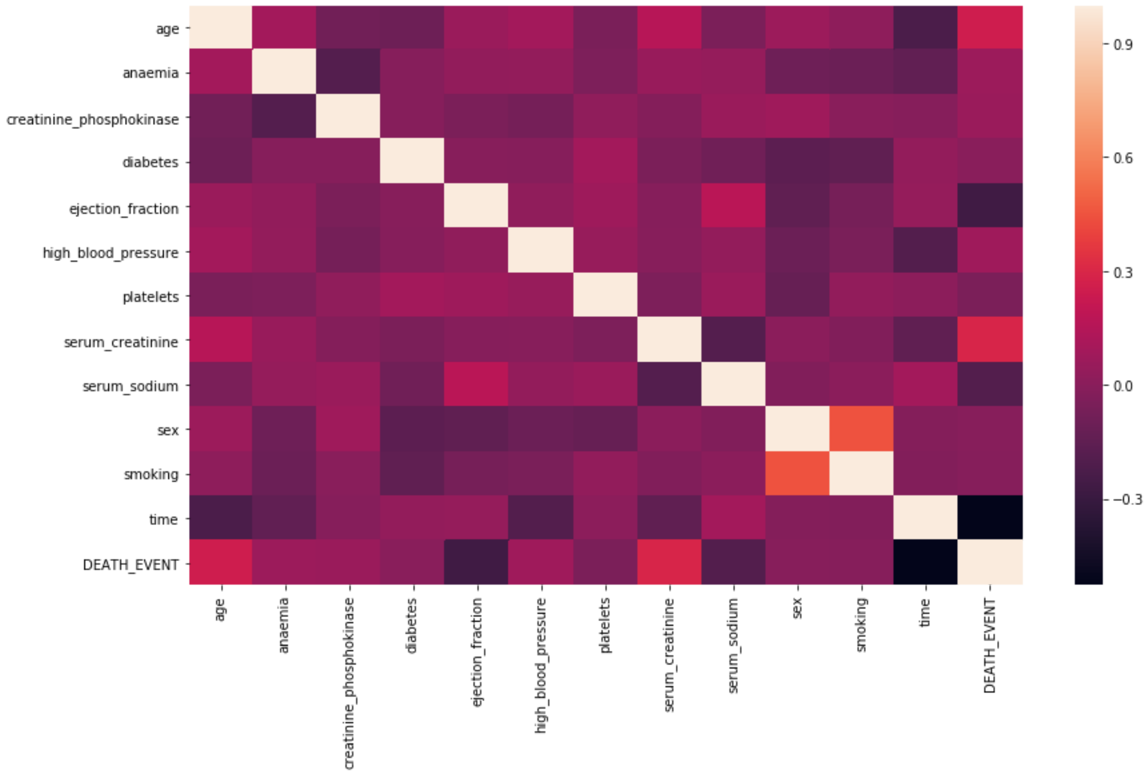
In [8]:
```
#Create a pie chart to show the relationship of sex, age and death events
male = df[df["sex"]==1]
female = df[df["sex"]==0]

male_survi = male[df["DEATH_EVENT"]==0]
male_not = male[df["DEATH_EVENT"]==1]
female_survi = female[df["DEATH_EVENT"]==0]
female_not = female[df["DEATH_EVENT"]==1]

labels = ['Male - Survived','Male - Not Survived', "Female -  Survived", "Female - Not Survived"]
values = [len(male[df["DEATH_EVENT"]==0]),len(male[df["DEATH_EVENT"]==1]),
         len(female[df["DEATH_EVENT"]==0]),len(female[df["DEATH_EVENT"]==1])]
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.4)])
fig.update_layout(
    title_text="Analysis on Survival - Gender")
fig.show()
```



Exploratory Data Analysis - Heat map to see the relationships between variables

In [9]:
```
#Generating correlation heat map
corr = df.corr()

plt.figure(figsize=(14, 8))
sns.heatmap(corr)
```

Out[9]: `<matplotlib.axes._subplots.AxesSubplot at 0x2745460f508>`



Research Methods with correlation and p-values

How does diabetes relate to death events? Correlation and P-Value

In [10]:
```
#Establishing series elements to check for correlation and p-value
stat_x = df['diabetes']
stat_y = df['DEATH_EVENT']

#The slope is the correlation and the pvalue is designated
result = scipy.stats.linregress(stat_x, stat_y)

#Print the results of the linear regression check for diabetes and death events
print("The correlation between diabetes and a cardiac event are: ",result.slope)
print("The P-Value between diabetes and a cardiac event are: ",result.pvalue)
```

```
The correlation between diabetes and a cardiac event are:  -0.0018390804597700802
The P-Value between diabetes and a cardiac event are:  0.9733118267847674
```

How does smoking relate to death events? Correlation and P-Value

In [11]:
```
#Establishing series elements to check for correlation and p-value
stat_x2 = df['smoking']
stat_y2 = df['DEATH_EVENT']

#The slope is the correlation and the pvalue is designated
result2 = scipy.stats.linregress(stat_x2, stat_y2)

#Print the results of the linear regression check for smoking and death events
print("The correlation between diabetes and a cardiac event are: ",result2.slope)
print("The P-Value between diabetes and a cardiac event are: ",result2.pvalue)
```

```
The correlation between diabetes and a cardiac event are:  -0.012623152709359493
The P-Value between diabetes and a cardiac event are:  0.8279207128092422
```

For feature engineering used normalization and used those columns in the following tests

In [12]:
```
#Creating a list of the columns to normalize for Feature Engineering
cols = ['creatinine_phosphokinase','ejection_fraction','platelets','serum_creatinine','serum_sodium']
```

In [13]:
```
#Create a loop to do the normalization for all the columns and add the columns to the dataframe
for c in cols:
    norm_col = df[[c]]
    min_max_scaler = preprocessing.MinMaxScaler()
    norm_col_scaled = min_max_scaler.fit_transform(norm_col)

    df[c + '_norm'] = norm_col_scaled
```

In [14]:
```
#Review the dataframe after the new column additions
print(df.shape)
df.head()
```

```
(299, 18)
```

Out[14]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	t
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	

In [15]:
```
#Creating dataframe for training without the DEATH_EVENT column and normalized columns
#Time was dropped as it is follow up time during interview plus the heatmap shows little correlation with DEATH_EVENT

X = df.drop(['time','creatinine_phosphokinase_norm','ejection_fraction_norm','platelets_norm',
             'serum_creatinine_norm','serum_sodium_norm','DEATH_EVENT'], axis = 1)
print(X.shape)
X
```

```
(299, 11)
```

Out[15]:

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

299 rows × 11 columns

In [16]:
```python
#Creating dataframe for training without the DEATH_EVENT column and non-normalized columns
#Time was dropped as it is follow up time during interview plus the heatmap shows little correlation with DEATH_EVENT

X_norm = df.drop(['time','creatinine_phosphokinase','ejection_fraction','platelets','serum_creatinine',
                  'serum_sodium','DEATH_EVENT'], axis = 1)
print(X_norm.shape)
X_norm
```

(299, 11)

Out[16]:

	age	anaemia	diabetes	high_blood_pressure	sex	smoking	creatinine_phosphokinase_norm	ejection_fraction_norm	platelets_norm	serum_creatinine
0	75.0	0	0	1	1	0	0.071319	0.090909	0.290823	0.
1	55.0	0	0	0	1	0	1.000000	0.363636	0.288833	0.
2	65.0	0	0	0	1	1	0.015693	0.090909	0.165960	0.
3	50.0	1	0	0	1	0	0.011227	0.090909	0.224148	0.
4	65.0	1	1	0	0	0	0.017479	0.090909	0.365984	0.
...	...	...	...	...	...	...	...	...	...	
294	62.0	0	1	1	1	1	0.004848	0.363636	0.157474	0.
295	55.0	0	0	0	0	0	0.229268	0.363636	0.296884	0.
296	45.0	0	1	0	0	0	0.259888	0.696970	0.869075	0.
297	45.0	0	0	0	1	1	0.304925	0.363636	0.139290	0.
298	50.0	0	0	0	1	1	0.022072	0.469697	0.448418	0.

299 rows × 11 columns

In [17]:
```python
#Checking the counts of deaths in the DEATH_EVENTS column to see if there is imbalance that needs to be addressed.
#The value of 1 is a death and there is a noted slant to imbalance will need to be addressed before training
df['DEATH_EVENT'].value_counts()
```

Out[17]:
```
0    203
1     96
Name: DEATH_EVENT, dtype: int64
```

In [18]:
```python
#Creating a series with the DEATH_EVENT that will be used for training.
y = df['DEATH_EVENT']
y
```

Out[18]:
```
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
```

Rechecking Correlation and P-Values using Ordinary Least Squares method for all variables against death events not including normalized variables after the split to answer the question how do all elements fare against death events

In [19]:
```python
#Comparing correlations between death events and various factors
corr_cols = X.columns

#Loop to generate correlations with death events and other factors and then print the results
for c in corr_cols:
    X_corr = df[c]
    X_result = str(round(X_corr.corr(y),3))

    print("Correlation between death events and", c,"is",X_result)
```

```
Correlation between death events and age is 0.254
Correlation between death events and anaemia is 0.066
Correlation between death events and creatinine_phosphokinase is 0.063
Correlation between death events and diabetes is -0.002
Correlation between death events and ejection_fraction is -0.269
Correlation between death events and high_blood_pressure is 0.079
Correlation between death events and platelets is -0.049
Correlation between death events and serum_creatinine is 0.294
Correlation between death events and serum_sodium is -0.195
Correlation between death events and sex is -0.004
Correlation between death events and smoking is -0.013
```

In [20]:
```
#Generating P-Values for death events and various factors
#Importing needed libraries
from sklearn import linear_model
from regressors import stats

#Creating Ordinary Least Squares Models to generate table for P-Vales
ols = linear_model.LinearRegression()
ols.fit(X, y)

#Loading lables into a series variable to help read the table easier
xlabels = X.columns
#Generating the table with P-Values
stats.summary(ols, X, y, xlabels)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-0.3106  0.1674  0.3171  0.4573  1.4415


Coefficients:
                          Estimate  Std. Error  t value   p value
_intercept                1.497829    0.778038   1.9251  0.055164
age                       0.009012    0.001852   4.8653  0.000002
anaemia                   0.054879    0.049092   1.1179  0.264519
creatinine_phosphokinase  0.000049    0.000024   2.0332  0.042913
diabetes                  0.016636    0.048721   0.3415  0.733002
ejection_fraction        -0.010591    0.001995  -5.3077  0.000000
high_blood_pressure       0.067954    0.050077   1.3570  0.175813
platelets                -0.000000    0.000000  -0.8562  0.392602
serum_creatinine          0.106322    0.023131   4.5966  0.000006
serum_sodium             -0.010960    0.000907 -12.0860  0.000000
sex                      -0.062559    0.055514  -1.1269  0.260691
smoking                   0.013404    0.055686   0.2407  0.809946
---
R-squared:  0.23608,    Adjusted R-squared:  0.20680
F-statistic: 8.06 on 11 features
```

Introducing SMOTE to deal with imbalance and comparing changes before and after SMOTE with Logistic Regression

In [21]:
```
#Splitting the data elements from X and y into a test size of 20%
X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size = 0.2)
```

In [22]:
```
#Below steps are used to train and test the model with the Logistic Regression Model before SMOTE
model = LogisticRegression()

model.fit(X_train,y_train)

y_predict = model.predict(X_test)
```

In [23]:
```
#accuracy_score will return a value between 0 and 1, 1 being 100%
score = accuracy_score(y_test, y_predict)
print(print("Accuracy of Logistic Regression before SMOTE is:","{:.2f}%".format(100* score)))

#The rows are the actual and the columns are the predicted
pd.crosstab(y_test, y_predict)
```

```
Accuracy of Logistic Regression before SMOTE is: 70.00%
None
```

Out[23]:

col_0	0	1
DEATH_EVENT		
0	35	4
1	14	7

In [24]:
```
#Applying SMOTE to oversample the minority imbalance
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_train_smote, y_train_smote = sm.fit_sample(X_train, y_train)
```

In [25]:
```
#Show the counts of the minority imbalance before and after SMOTE
from collections import Counter
print("Before SMOTE: ", Counter(y_train))
print("After SMOTE: ", Counter(y_train_smote))
```

```
Before SMOTE:  Counter({0: 164, 1: 75})
After SMOTE:  Counter({1: 164, 0: 164})
```

In [26]:
```
#Create an accuracy list to store the results of the various model tests
accuracy_list = []
```

In [27]:
```
#Traing the model Logistic Regression model before SMOTE
model.fit(X_train,y_train)
y_predict = model.predict(X_test)

#accuracy_score will return a value between 0 and 1, 1 being 100%
score = accuracy_score(y_test, y_predict)

print(print("Accuracy of Logistic Regression after SMOTE is:","{:.2f}%".format(100* score)))

#The rows are the actual and the columns are the predicted
pd.crosstab(y_test, y_predict)
```

```
Accuracy of Logistic Regression after SMOTE is: 70.00%
None
```

Out[27]:

col_0	0	1
DEATH_EVENT		
0	35	4
1	14	7

Beginning of the Linear Regression model with SMOTE data that will also be used in the model comparisons after

In [28]:
```
#Traing the Logistic model after SMOTE
model.fit(X_train_smote,y_train_smote)
y_predict = model.predict(X_test)

#Calculate the accuracy of the Logistic model and append it to the accuracy_list
log_reg_acc = accuracy_score(y_test, y_predict)
accuracy_list.append(100*log_reg_acc)

print("Accuracy of Logistic Regression after SMOTE is: ", "{:.2f}%".format(100* log_reg_acc))

#The rows are the actual and the columns are the predicted
pd.crosstab(y_test, y_predict)
```

```
Accuracy of Logistic Regression after SMOTE is:  65.00%
```

Out[28]:

col_0	0	1
DEATH_EVENT		
0	24	15
1	6	15

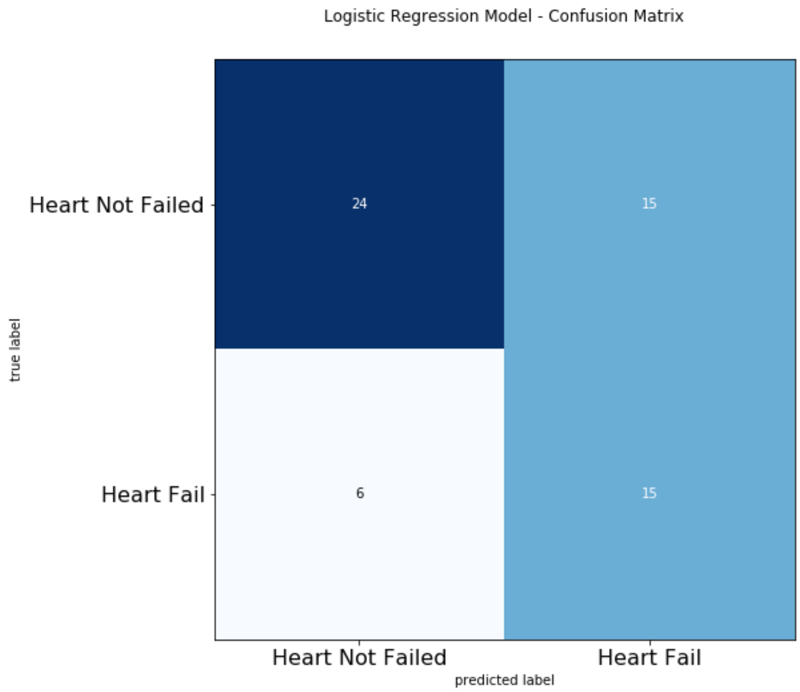
Model Selection - Trying at least 3 models

Tested with 5 models with a comparison bar graph at the end

Imbalance in the data has been handled using SMOTE using a 80% train and 20% test data split

In [29]:
```
#Creat a confusion matrix for the accuracy of the Logistic Regression
cm = confusion_matrix(y_test, y_predict)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("Logistic Regression Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [30]:
```
#Printing report to look at the Logistict accuracy based on y_test
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.80      0.62      0.70        39
           1       0.50      0.71      0.59        21

    accuracy                           0.65        60
   macro avg       0.65      0.66      0.64        60
weighted avg       0.70      0.65      0.66        60
```

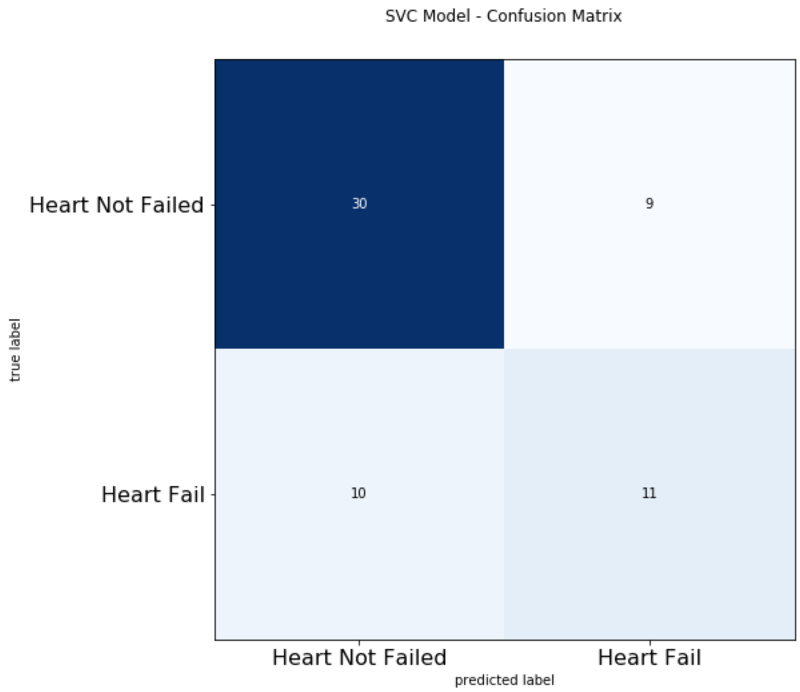
In [31]:
```
#Traing the SVC model after SMOTE
sv_clf = SVC()
sv_clf.fit(X_train_smote, y_train_smote)
sv_clf_pred = sv_clf.predict(X_test)

#Calculate the accuracy of the SVC model and append it to the accuracy_list
sv_clf_acc = accuracy_score(y_test, sv_clf_pred)
accuracy_list.append(100* sv_clf_acc)

print("Accuracy of SVC after SMOTE is : ", "{:.2f}%".format(100* sv_clf_acc))
```

```
Accuracy of SVC after SMOTE is :  68.33%
```

In [32]:

```
#Creat a confusion matrix for the accuracy of the SVC Regression
cm = confusion_matrix(y_test, sv_clf_pred)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("SVC Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [33]:

```
#Printing report to look at the SVC accuracy based on y_test
print(classification_report(y_test, sv_clf_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.77      0.76        39
           1       0.55      0.52      0.54        21

    accuracy                           0.68        60
   macro avg       0.65      0.65      0.65        60
weighted avg       0.68      0.68      0.68        60
```

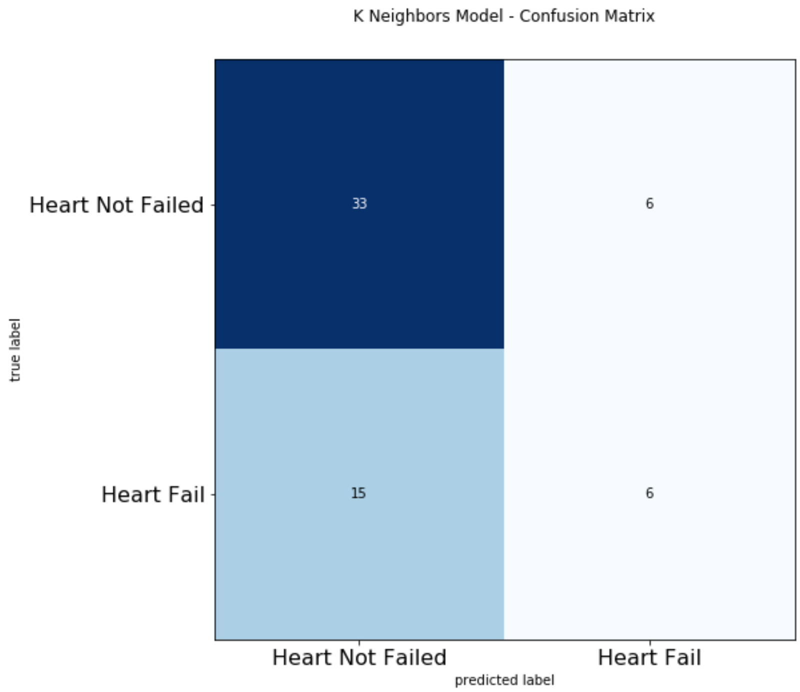
In [34]:

```
#Traing the KNN model after SMOTE
kn_clf = KNeighborsClassifier(n_neighbors=6)
kn_clf.fit(X_train_smote, y_train_smote)
kn_pred = kn_clf.predict(X_test)

#Calculate the accuracy of the KNN model and append it to the accuracy_list
kn_acc = accuracy_score(y_test, kn_pred)
accuracy_list.append(100*kn_acc)

print("Accuracy of KNN Classifier after SMOTE is : ", "{:.2f}%".format(100* kn_acc))
```

```
Accuracy of KNN Classifier after SMOTE is :  65.00%
```

In [35]:
```
#Creat a confusion matrix for the accuracy of the KNN model
cm = confusion_matrix(y_test, kn_pred)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("K Neighbors Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [36]:
```
#Printing report to look at the K Nearest Neighbors accuracy based on y_test
print(classification_report(y_test, kn_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.85      0.76        39
           1       0.50      0.29      0.36        21

    accuracy                           0.65        60
   macro avg       0.59      0.57      0.56        60
weighted avg       0.62      0.65      0.62        60
```

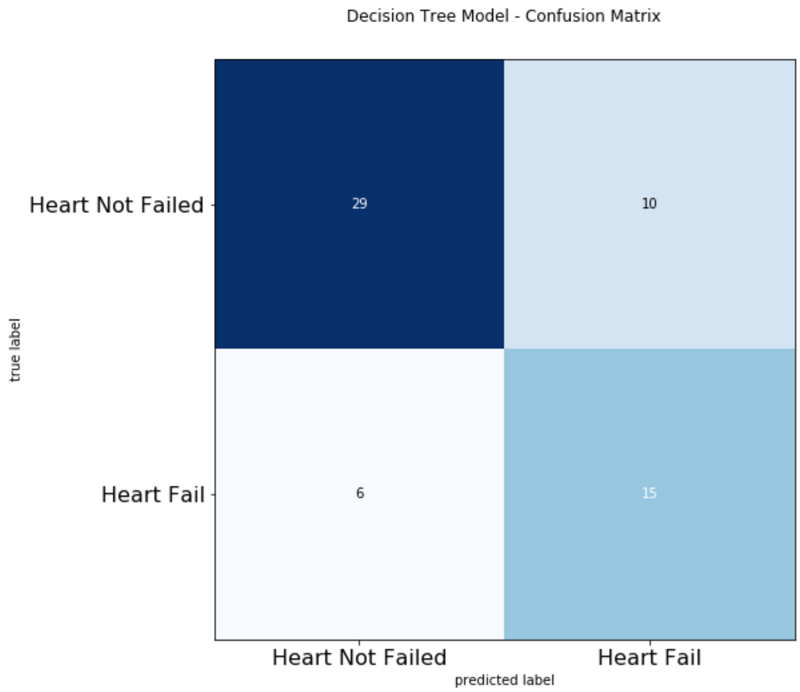
In [37]:
```
#Traing the Decision Tree Classifier model after SMOTE
dt_clf = DecisionTreeClassifier(max_leaf_nodes=3, random_state=0, criterion='entropy')
dt_clf.fit(X_train_smote, y_train_smote)
dt_pred = dt_clf.predict(X_test)

#Calculate the accuracy of the Decision Tree Classifier model and append it to the accuracy_list
dt_acc = accuracy_score(y_test, dt_pred)
accuracy_list.append(100*dt_acc)

print("Accuracy of Decision Tree Classifier after SMOTE is : ", "{:.2f}%".format(100* dt_acc))
```

```
Accuracy of Decision Tree Classifier after SMOTE is :  73.33%
```

In [38]:

```
#Creat a confusion matrix for the accuracy of the Decision Tree Classifier model
cm = confusion_matrix(y_test, dt_pred)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("Decision Tree Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [39]:

```
#Printing report to look at the Decision Tree Classifier accuracy based on y_test
print(classification_report(y_test, dt_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.74      0.78        39
           1       0.60      0.71      0.65        21

    accuracy                           0.73        60
   macro avg       0.71      0.73      0.72        60
weighted avg       0.75      0.73      0.74        60
```

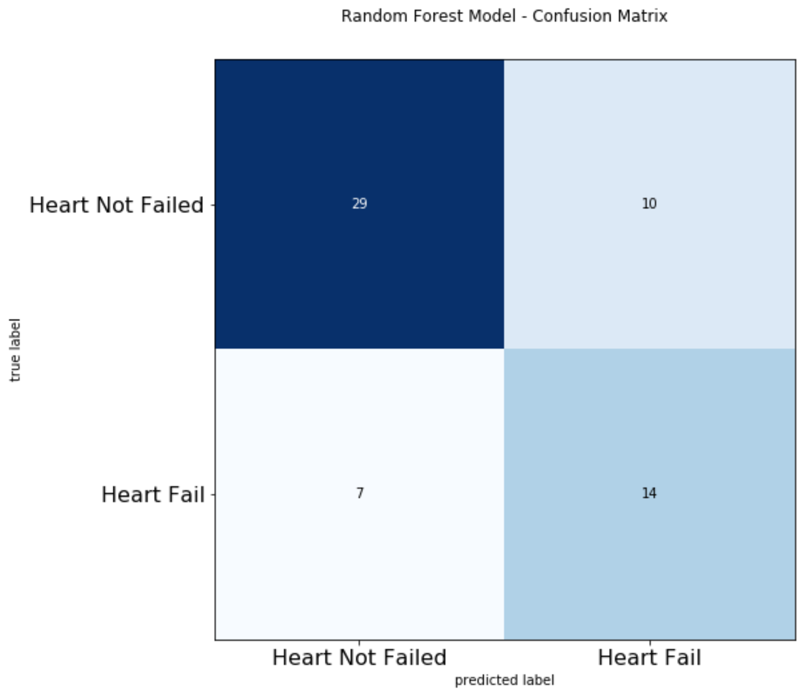
In [40]:

```
#Traing the Random Forest Classifier model after SMOTE
r_clf = RandomForestClassifier(max_features=0.5, max_depth=15, random_state=1)
r_clf.fit(X_train_smote, y_train_smote)
r_pred = r_clf.predict(X_test)

#Calculate the accuracy of the Random Forest Classifier model and append it to the accuracy_list
r_acc = accuracy_score(y_test, r_pred)
accuracy_list.append(100*r_acc)

print("Accuracy of Random Forest Classifier is : ", "{:.2f}%".format(100* r_acc))
```

```
Accuracy of Random Forest Classifier is :  71.67%
```

In [41]:
```
#Creat a confusion matrix for the accuracy of the Random Forest Classifier model
cm = confusion_matrix(y_test, r_pred)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("Random Forest Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

<Figure size 432x288 with 0 Axes>

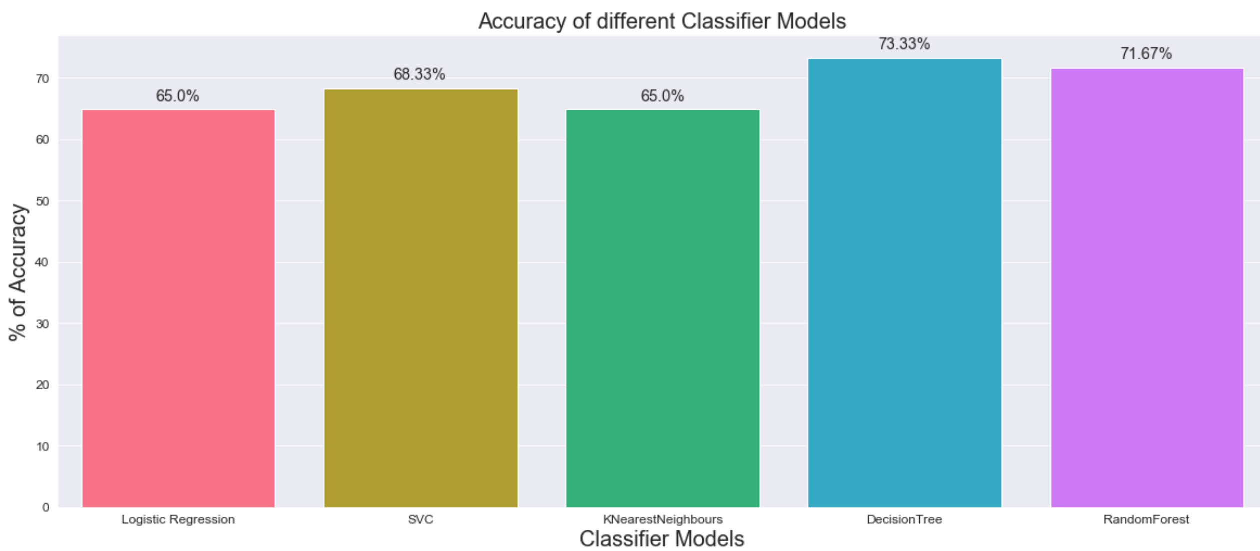


In [42]:
```
#Printing report to look at the Random Forest Classifier accuracy based on y_test
print(classification_report(y_test, r_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.74      0.77        39
           1       0.58      0.67      0.62        21

    accuracy                           0.72        60
   macro avg       0.69      0.71      0.70        60
weighted avg       0.73      0.72      0.72        60
```

In [43]:
```
#Create list of model names that will be part of the Classifier bar chart
model_list = ['Logistic Regression', 'SVC','KNearestNeighbours', 'DecisionTree', 'RandomForest']
```

In [44]:
```
#Creating Bar Chart to show the accuracy of the Classifiers

#Set size of the chart figure and the style
plt.rcParams['figure.figsize']=20,8
sns.set_style('darkgrid')

#Set the axis values and other options for the bar char
ax = sns.barplot(x=model_list, y=accuracy_list, palette = "husl", saturation = 2.0)

#Set the x and y label titles as well as the Bar Chart Title
plt.xlabel('Classifier Models', fontsize = 20 )
plt.ylabel('% of Accuracy', fontsize = 20)
plt.title('Accuracy of different Classifier Models', fontsize = 20)

#Additional setting for the bar chart
plt.xticks(fontsize = 12, horizontalalignment = 'center', rotation = 0)
plt.yticks(fontsize = 12)

#For loop to show the percentage totals over each bar
for i in ax.patches:
    width, height = i.get_width(), i.get_height()
    x, y = i.get_xy()
    ax.annotate(f'{round(height,2)}%', (x + width/2, y + height*1.02), ha='center', fontsize = 'x-large')

#Show the resulting bar chart for the Classifier Models
plt.show()
```



Performance Optimization - using hyperparameter tuning

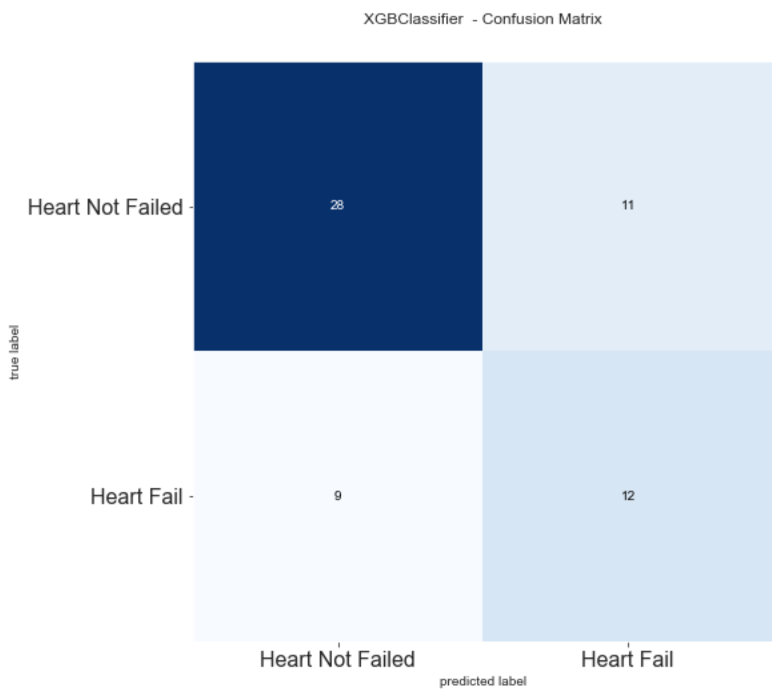
In [45]:
```
from xgboost import XGBClassifier

#Running the second model as per the requirements and to also partake in hyperparameter tuning
model = XGBClassifier()
model.fit(X_train_smote,y_train_smote)
y_predict = model.predict(X_test)

#accuracy_score will return a value between 0 and 1, 1 being 100%
score_b = accuracy_score(y_test, y_predict)
print("Accuracy Before Hyperparametized Tuning is:","{:.2f}%".format(100* score_b))
```

```
Accuracy Before Hyperparametized Tuning is: 66.67%
```

In [46]:

```
#Creat a confusion matrix for the accuracy of the XGBClassifier model before hyper parametized tuning
cm = confusion_matrix(y_test, y_predict)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("XGBClassifier  - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

```
<Figure size 1440x576 with 0 Axes>
```



In [47]:

```
#Printing report to look at the accuracy based on y_test
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.76      0.72      0.74        39
           1       0.52      0.57      0.55        21

    accuracy                           0.67        60
   macro avg       0.64      0.64      0.64        60
weighted avg       0.67      0.67      0.67        60
```

In [48]:
```
#Create lists that can be used in the below for loops to test hyperparameters
maxD = [8, 9, 10, 11, 12]
subS = [0.25, 0.5, 0.75, 1]
nEst = [100, 200, 300, 400, 500]
learnR = [0.05, 0.1, 0.2, 0.3]
randS = [1, 2, 3 ,4]

#Create an empty dataset for holding loop results
resultSet = pd.DataFrame()

#Series of for loops for testing hyperparameters
for m in maxD:
    for s in subS:
        for n in nEst:
            for l in learnR:
                for r in randS:
                    model = XGBClassifier(max_depth = m,
                                          subsample = s,
                                          n_estimators = n,
                                          learning_rate = l,
                                          random_state = r)

                    model.fit(X_train_smote,y_train_smote)
                    y_predict = model.predict(X_test)
                    y_train_predict = model.predict(X_train)

                    #Can only append if ingore_index is True
                    #Loading results in empty dataframe created previous
                    resultSet = resultSet.append({'1 max_depth': m, #replace with m
                                                  '2 subsample': s, #replace with s
                                                  '3 n_estimators': n, #replace with n
                                                  '4 Learning_rate': l, #replace with l
                                                  '5 random_state': r, #replace with r
                                                  'Train Accuracy': accuracy_score(y_train, y_train_predict),
                                                  'Test Accuracy': accuracy_score(y_test, y_predict)},
                                                  ignore_index = True)
```

In [49]:
```
#Loading the max value for test accuracy in a variable from resultSet dataset created in the loop
max_test = resultSet['Test Accuracy'].max()

#Loading the max value for train accuracy in a variable from resultSet dataset created in the loop
max_train = resultSet['Train Accuracy'].max()

#Extracting the records with the max test accuracy and max train accuracy to see the parameter settings to use
rs_check = resultSet.loc[(resultSet['Test Accuracy'] == max_test)]

#Show dataframe with max test accuracy sorted by train accuracy descending
rs_check_sort = rs_check.sort_values('Train Accuracy', ascending=False)
rs_check_sort
```

Out[49]:

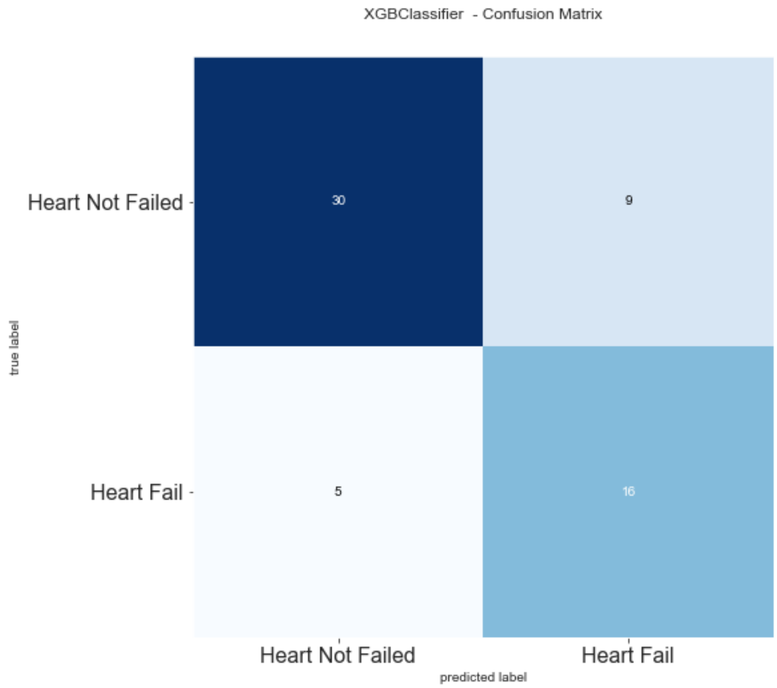
	1 max_depth	2 subsample	3 n_estimators	4 Learning_rate	5 random_state	Test Accuracy	Train Accuracy
576	9.0	1.0	200.0	0.05	1.0	0.766667	1.0
577	9.0	1.0	200.0	0.05	2.0	0.766667	1.0
578	9.0	1.0	200.0	0.05	3.0	0.766667	1.0
579	9.0	1.0	200.0	0.05	4.0	0.766667	1.0
880	10.0	1.0	100.0	0.05	1.0	0.766667	1.0
881	10.0	1.0	100.0	0.05	2.0	0.766667	1.0
882	10.0	1.0	100.0	0.05	3.0	0.766667	1.0
883	10.0	1.0	100.0	0.05	4.0	0.766667	1.0

In [52]:
```
#Running the second model as per the requirements after hyperparameter testing
model = XGBClassifier(max_depth = 9,
                      subsample = 1,
                      n_estimators = 200,
                      learning_rate = 0.05,
                      random_state = 1)
model.fit(X_train_smote,y_train_smote)
y_predict = model.predict(X_test)

#accuracy_score will return a value between 0 and 1, 1 being 100%
score_a = accuracy_score(y_test, y_predict)
print("Accuracy Before Hyperparametized Tuning is:","{:.2f}%".format(100* score_b))
print("Accuracy After Hyperparametized Tuning is:","{:.2f}%".format(100* score_a))
```

```
Accuracy Before Hyperparametized Tuning is: 66.67%
Accuracy After Hyperparametized Tuning is: 76.67%
```

In [53]:
```
#Creat a confusion matrix for the accuracy of the XGBClassifier model after hyper parametized tuning
cm = confusion_matrix(y_test, y_predict)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("XGBClassifier  - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Fail"], fontsize=16)
plt.show()
```

```
<Figure size 1440x576 with 0 Axes>
```



In [54]:
```
#Printing report to look at the accuracy based on y_test
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.86      0.77      0.81        39
           1       0.64      0.76      0.70        21

    accuracy                           0.77        60
   macro avg       0.75      0.77      0.75        60
weighted avg       0.78      0.77      0.77        60
```

In [ ]: