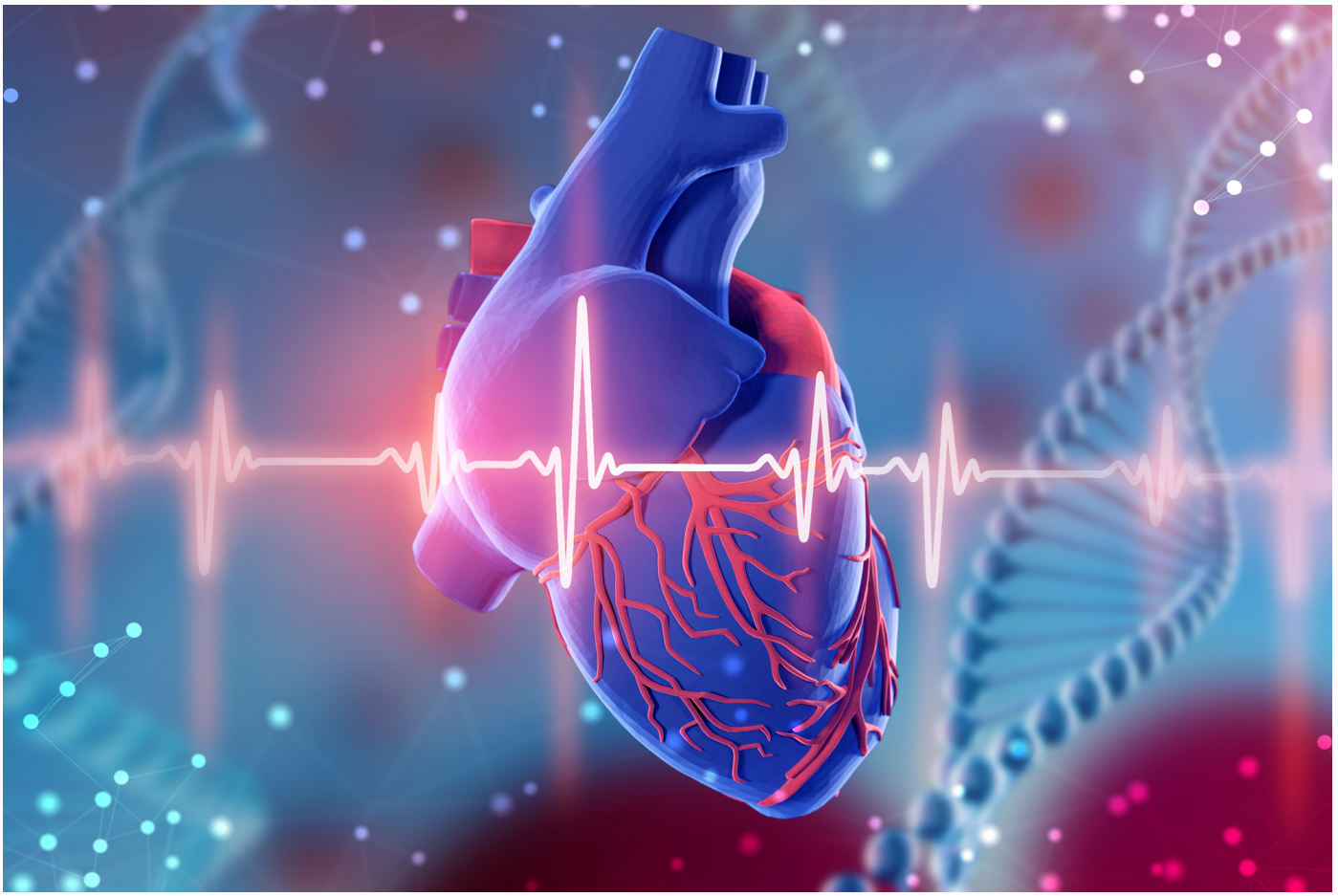
Final Assignment | Sahil Sareen RA1911003010464

18CSE398J Machine Learning - Core Concepts with Applications

Heart Failure Prediction



About the Dataset - Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worlwide. Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict mortality by heart failure. Most cardiovascular diseases can be prevented by addressing behavioral risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

Data Collection

In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
import plotly.figure_factory as ff
import numpy as np
import seaborn as sns
from sklearn.ensemble import ExtraTreesClassifier
import plotly.express as px
import plotly.graph_objs as go
from sklearn.metrics import confusion_matrix, accuracy_score
from mlxtend.plotting import plot_confusion_matrix
from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
```

```
from sklearn import model_selection
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from matplotlib.axes._axes import _log as matplotlib_axes_logger
from colorama import Fore, Back, Style

matplotlib_axes_logger.setLevel('ERROR')
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.filterwarnings("ignore")

file = 'heart_failure-dataset.csv'
df = pd.read_csv(file)

# Some new libraries i studied for this assignment:
# plotly = Plotly's Python graphing library makes interactive, publication-quality graphs.
# mlxtend = Mlxtend is a Python library of useful tools for the day-to-day data science ta
# Coloroma = for producing colored terminal text and cursor positioning.
```

Data Exploration

In [2]:
```
df.head()
```

Out[2]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creati
0	75.0	0	582	0	20	1	265000.00	
1	55.0	0	7861	0	38	0	263358.03	
2	65.0	0	146	0	20	0	162000.00	
3	50.0	1	111	0	20	0	210000.00	
4	65.0	1	160	1	20	0	327000.00	

- Sex - Gender of patient Male = 1, Female =0
- Age - Age of patient
- Diabetes - 0 = No, 1 = Yes
- Anaemia - 0 = No, 1 = Yes
- High_blood_pressure - 0 = No, 1 = Yes
- Smoking - 0 = No, 1 = Yes
- DEATH_EVENT - 0 = No, 1 = Yes

In [3]:
```
df.shape
```

Out[3]:
```
(299, 13)
```

In [4]:
```
df.dtypes
```

Out[4]:
```
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 [5]:
```
df.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 [6]:
```
df.describe()
```

Out[6]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	pla
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.0
mean	60.833893	0.431438	581.839465	0.418060	38.083612	0.351171	263358.0
std	11.894809	0.496107	970.287881	0.494067	11.834841	0.478136	97804.2
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.0
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.0
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.0
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.0
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.0

In [7]:
```
df.memory_usage()
```

Out[7]:
```
Index                       128
age                        2392
anaemia                    2392
creatinine_phosphokinase   2392
diabetes                   2392
ejection_fraction          2392
high_blood_pressure        2392
platelets                  2392
serum_creatinine           2392
serum_sodium               2392
sex                        2392
```

```
smoking                   2392
time                      2392
DEATH_EVENT               2392
dtype: int64
```

In [8]:
```
df.nunique()
```

Out[8]:
```
age                         47
anaemia                      2
creatinine_phosphokinase   208
diabetes                     2
ejection_fraction           17
high_blood_pressure          2
platelets                  176
serum_creatinine            40
serum_sodium                27
sex                          2
smoking                      2
time                       148
DEATH_EVENT                  2
dtype: int64
```

In [9]:
```
df.mean()
```

Out[9]:
```
age                          60.833893
anaemia                       0.431438
creatinine_phosphokinase    581.839465
diabetes                      0.418060
ejection_fraction            38.083612
high_blood_pressure           0.351171
platelets                263358.029264
serum_creatinine              1.393880
serum_sodium                136.625418
sex                           0.648829
smoking                       0.321070
time                        130.260870
DEATH_EVENT                   0.321070
dtype: float64
```

In [10]:
```
df.var()
```

Out[10]:
```
age                        1.414865e+02
anaemia                    2.461224e-01
creatinine_phosphokinase   9.414586e+05
diabetes                   2.441023e-01
ejection_fraction          1.400635e+02
high_blood_pressure        2.286144e-01
platelets                  9.565669e+09
serum_creatinine           1.070211e+00
serum_sodium               1.946996e+01
sex                        2.286144e-01
smoking                    2.187156e-01
time                       6.023965e+03
DEATH_EVENT                2.187156e-01
dtype: float64
```

In [11]:
```
df.skew()
```

Out[11]:
```
age                        0.423062
anaemia                    0.278261
creatinine_phosphokinase   4.463110
diabetes                   0.333929
ejection_fraction          0.555383
```

```
high_blood_pressure          0.626732
platelets                    1.462321
serum_creatinine             4.455996
serum_sodium                -1.048136
sex                         -0.626732
smoking                      0.770349
time                         0.127803
DEATH_EVENT                  0.770349
dtype: float64
```

Deviation of the data from the normal distribution.

In [12]:
```
df.corr()
```

Out[12]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pre
age	1.000000	0.088006	-0.081584	-0.101012	0.060098	0.09
anaemia	0.088006	1.000000	-0.190741	-0.012729	0.031557	0.03
creatinine_phosphokinase	-0.081584	-0.190741	1.000000	-0.009639	-0.044080	-0.07
diabetes	-0.101012	-0.012729	-0.009639	1.000000	-0.004850	-0.01
ejection_fraction	0.060098	0.031557	-0.044080	-0.004850	1.000000	0.02
high_blood_pressure	0.093289	0.038182	-0.070590	-0.012732	0.024445	1.00
platelets	-0.052354	-0.043786	0.024463	0.092193	0.072177	0.04
serum_creatinine	0.159187	0.052174	-0.016408	-0.046975	-0.011302	-0.00
serum_sodium	-0.045966	0.041882	0.059550	-0.089551	0.175902	0.03
sex	0.065430	-0.094769	0.079791	-0.157730	-0.148386	-0.10
smoking	0.018668	-0.107290	0.002421	-0.147173	-0.067315	-0.05
time	-0.224068	-0.141414	-0.009346	0.033726	0.041729	-0.19
DEATH_EVENT	0.253729	0.066270	0.062728	-0.001943	-0.268603	0.07

It gives correlation between each dataset variable.

In [13]:
```
df['sex'].value_counts()
```

Out[13]:
```
1    194
0    105
Name: sex, dtype: int64
```

In [14]:
```
df['diabetes'].value_counts()
```

Out[14]:
```
0    174
1    125
Name: diabetes, dtype: int64
```

In [15]:
```
df['smoking'].value_counts()
```

Out[15]:
```
0    203
1     96
Name: smoking, dtype: int64
```

In [16]:
```
df['high_blood_pressure'].value_counts()
```

Out[16]:
```
0    194
1    105
Name: high_blood_pressure, dtype: int64
```

In [17]:
```
df['DEATH_EVENT'].value_counts()
```

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

In [18]:
```
df.min()
```

Out[18]:
```
age                          40.0
anaemia                       0.0
creatinine_phosphokinase     23.0
diabetes                      0.0
ejection_fraction            14.0
high_blood_pressure           0.0
platelets                 25100.0
serum_creatinine              0.5
serum_sodium                113.0
sex                           0.0
smoking                       0.0
time                          4.0
DEATH_EVENT                   0.0
dtype: float64
```

It gives minimum value of each dataset variable.

In [19]:
```
df.max()
```

Out[19]:
```
age                          95.0
anaemia                       1.0
creatinine_phosphokinase   7861.0
diabetes                      1.0
ejection_fraction            80.0
high_blood_pressure           1.0
platelets                850000.0
serum_creatinine              9.4
serum_sodium                148.0
sex                           1.0
smoking                       1.0
time                        285.0
DEATH_EVENT                   1.0
dtype: float64
```

It gives maximum value of each dataset variable.

In [20]:
```
df.median()
```

Out[20]:
```
age                          60.0
anaemia                       0.0
creatinine_phosphokinase    250.0
diabetes                      0.0
ejection_fraction            38.0
high_blood_pressure           0.0
platelets                262000.0
serum_creatinine              1.1
serum_sodium                137.0
sex                           1.0
smoking                       0.0
time                        115.0
```

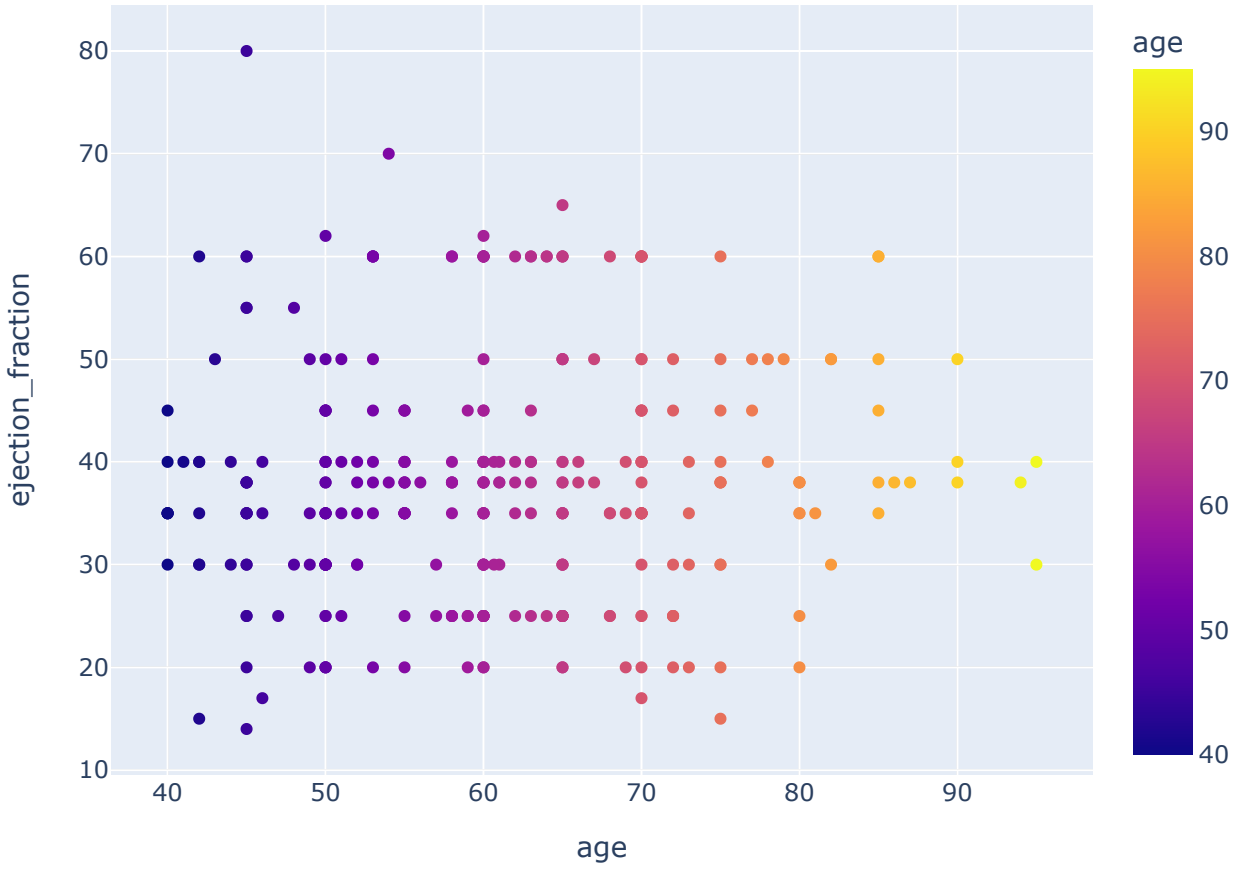
```
DEATH_EVENT                 0.0
dtype: float64
```

It gives middle value of each dataset variable.

Data Visualization

In [21]:
```
sns.set_theme(style="darkgrid")
```

In [22]:
```
fig = px.scatter(df, x="age", y="ejection_fraction", color = 'age')
fig.show()
```

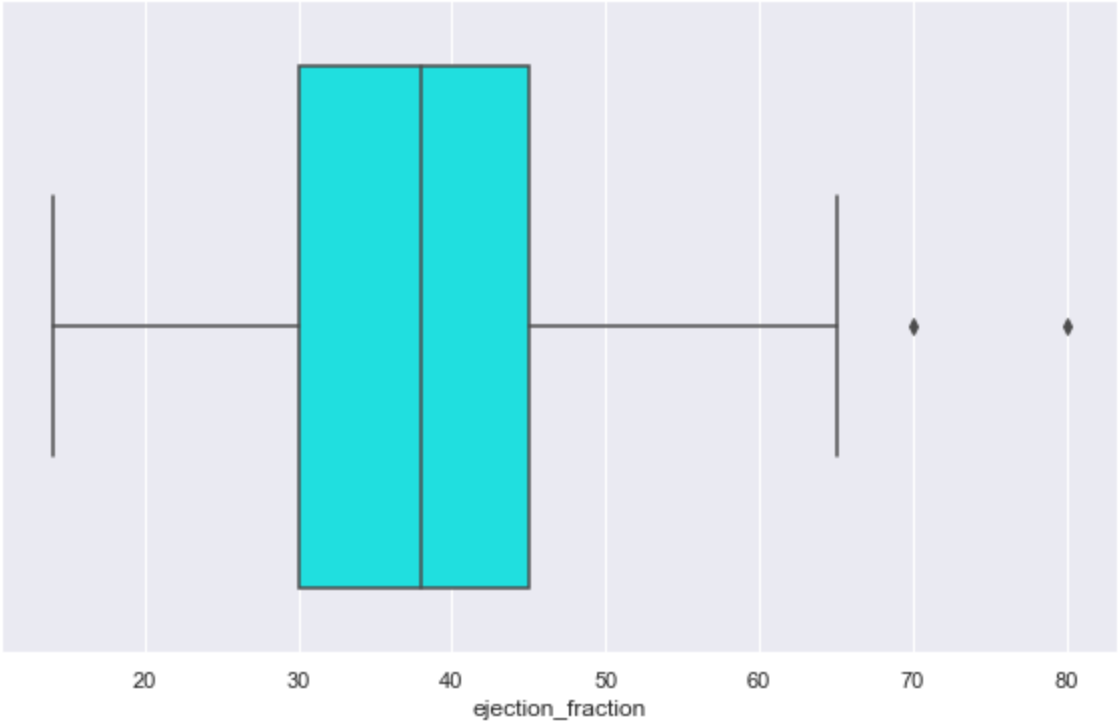


Checking for outliers with BoxPlots

An outlier is a data point that differs significantly from other observations.

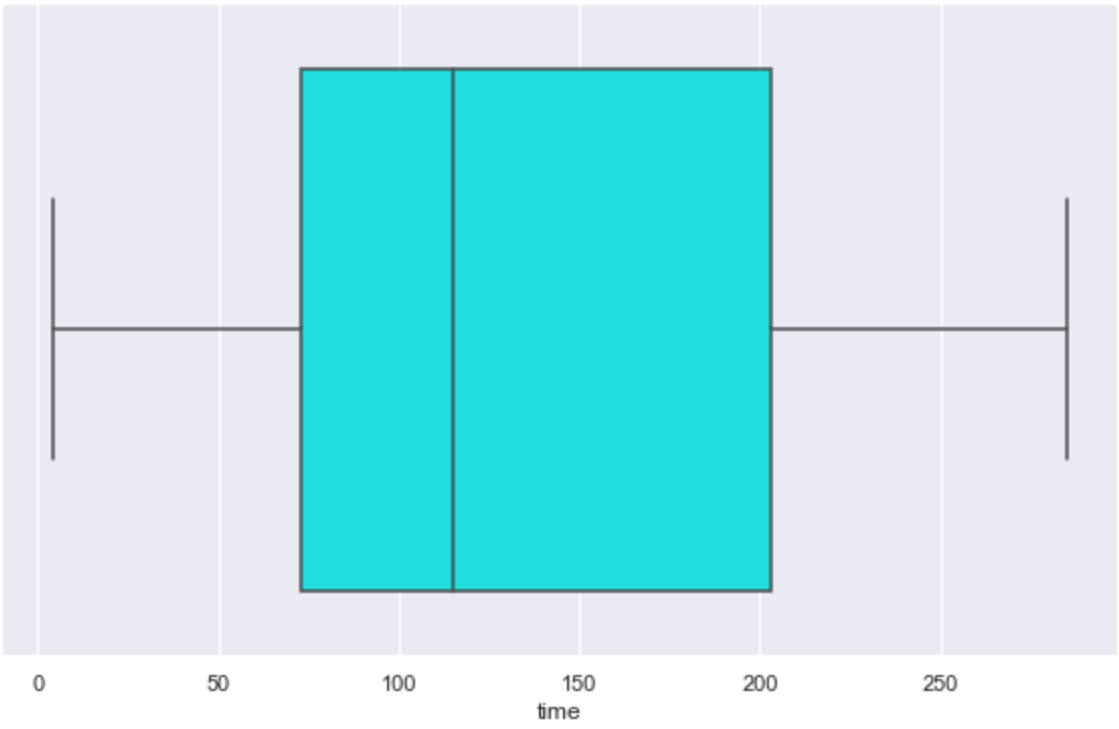
Ejection fraction

In [23]:
```
plt.figure(figsize=(10,6))
sns.boxplot(x = df.ejection_fraction, color = 'cyan')
plt.show()
```



Time

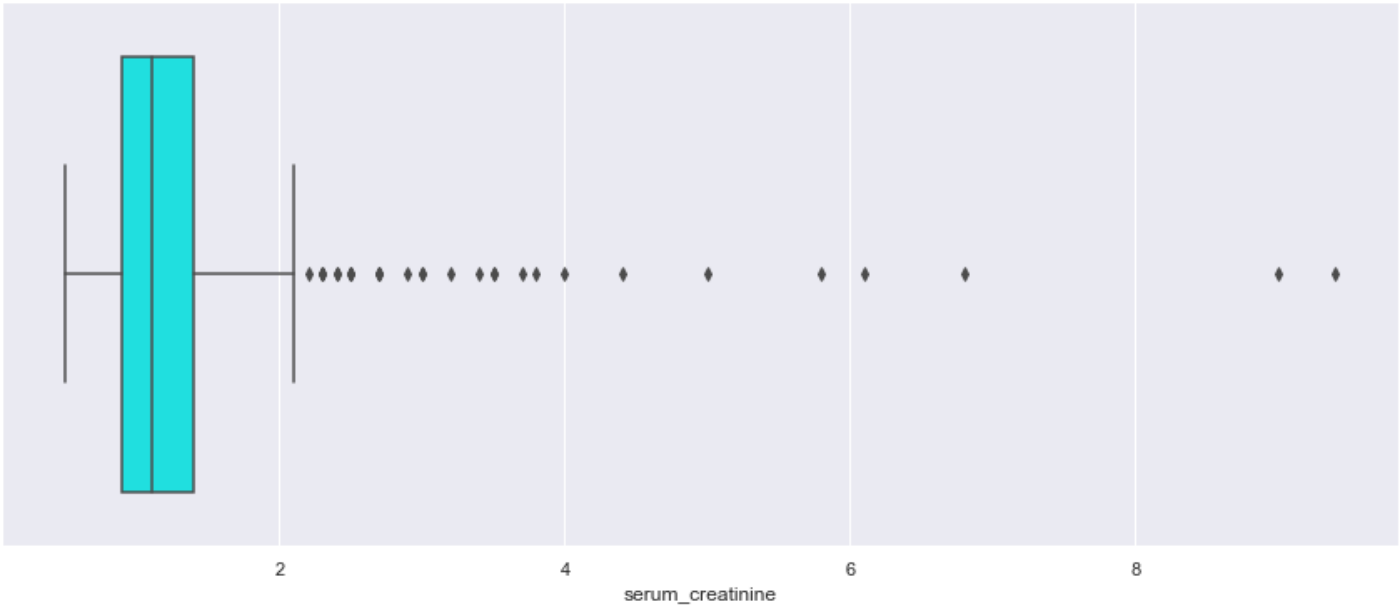
In [24]:
```
plt.figure(figsize=(10,6))
sns.boxplot(x=df.time, color = 'cyan')
plt.show()
```



No outliers in time.

Serum Creatinine

In [25]:
```
plt.figure(figsize=(15,6))
sns.boxplot(x=df.serum_creatinine, color = 'cyan')
plt.show()
```

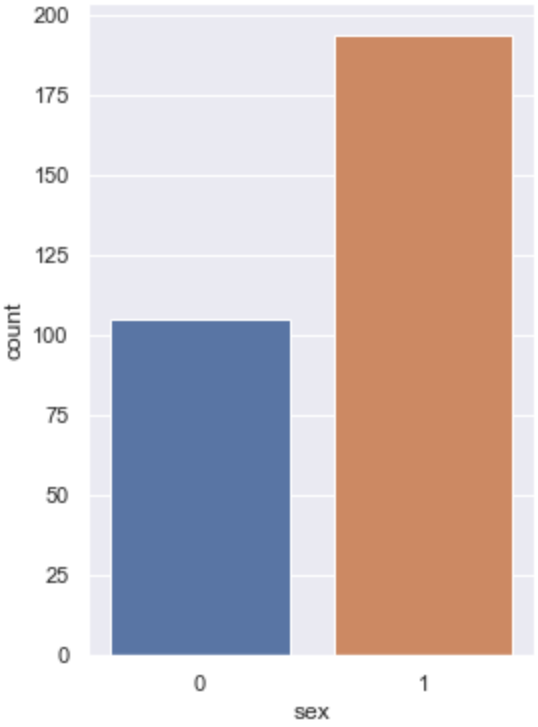


These are not actually outliers instead including them provided better results in predicting death event.

No such outliers observed which impact the model in a negative manner.

In [26]:
```
sns.set(rc={'figure.figsize':(4,6)})
sns.countplot(df['sex'])
```

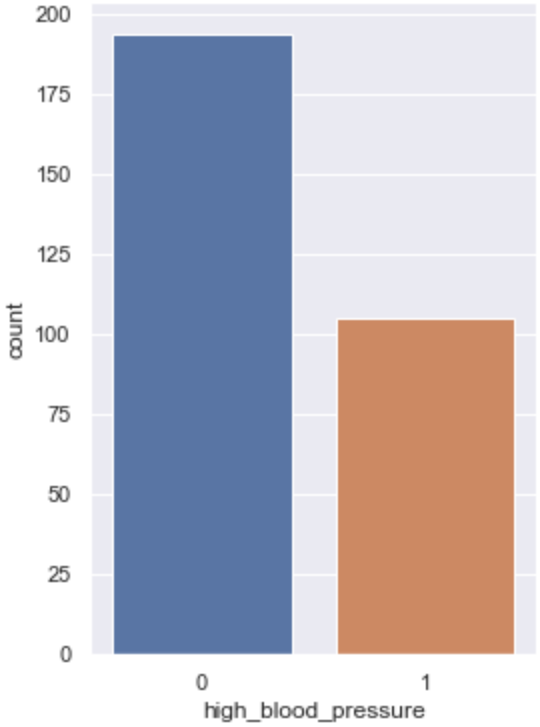
Out[26]:
```
<AxesSubplot:xlabel='sex', ylabel='count'>
```



It can be observed that higher number of the heart patients were male.

In [27]:
```
sns.set(rc={'figure.figsize':(4,6)})
sns.countplot(df['high_blood_pressure'])
```

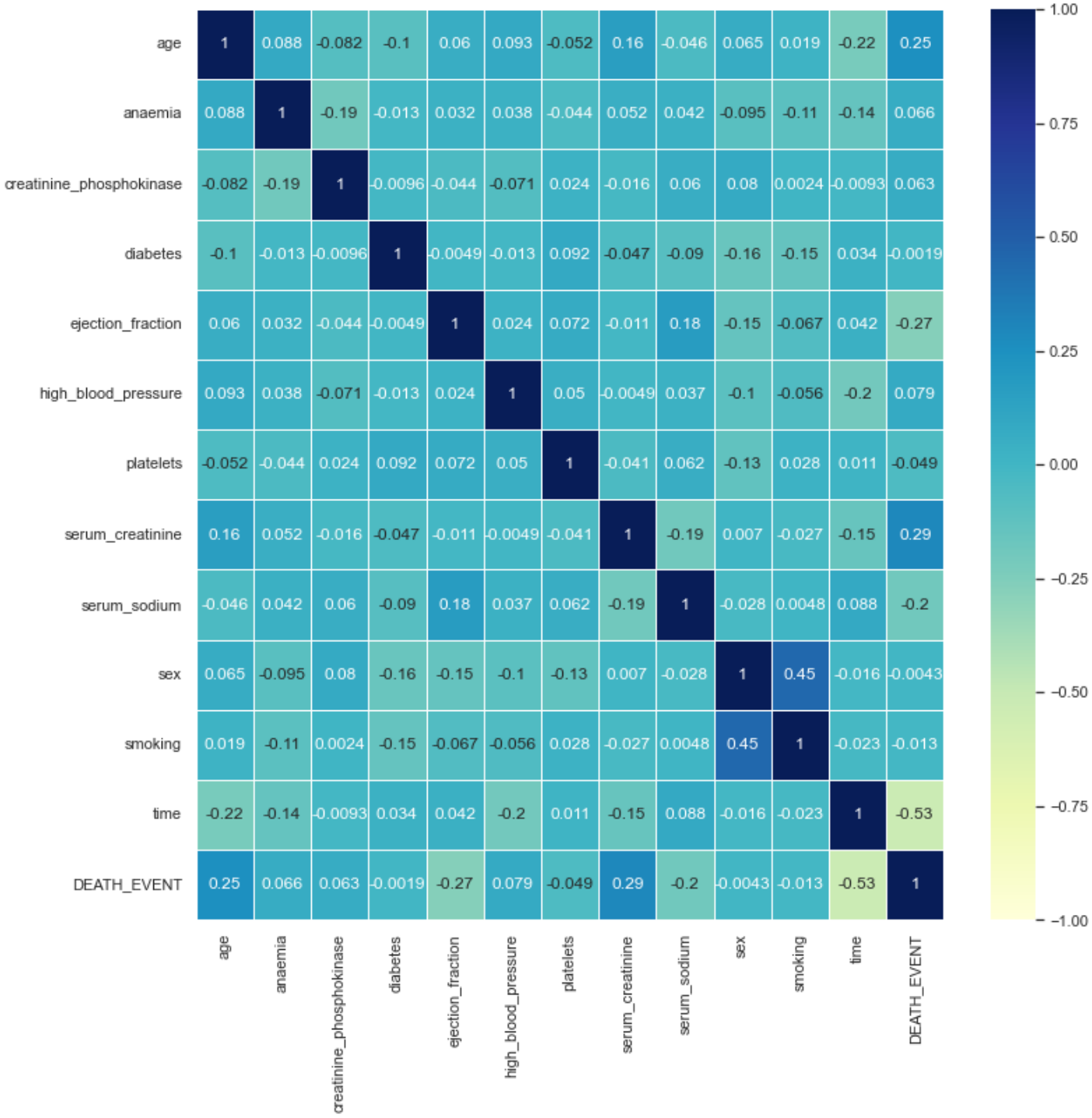
Out[27]:
```
<AxesSubplot:xlabel='high_blood_pressure', ylabel='count'>
```



It can be observed majority of the heart patients did not suffer from high blood pressure.

In [28]:
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(),vmin=-1, cmap="YlGnBu", annot=True, linewidths=.5)
```

Out[28]:
```
<AxesSubplot:>
```



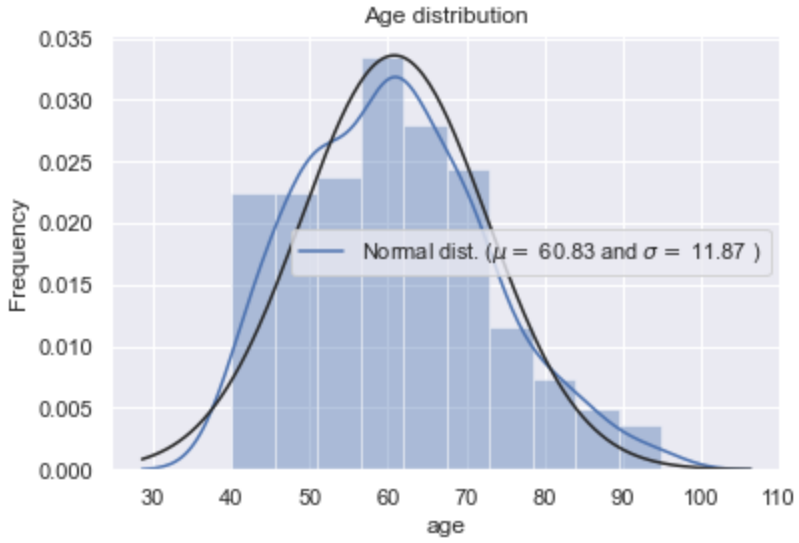
From this heatmap correlation between the variables can be easily observed. It gives a graphical representation of data using colors to visualize the values.

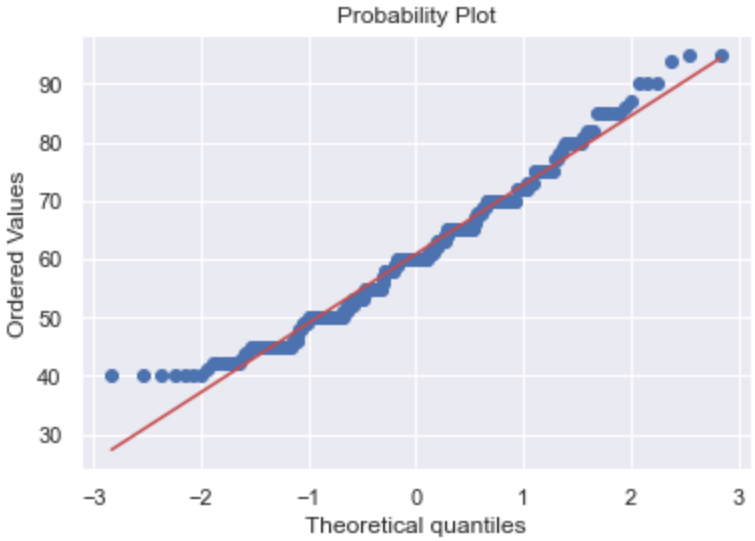
In [29]:

```
from scipy import stats
from scipy.stats import norm, skew

sns.set(rc={'figure.figsize':(6,4)})
sns.distplot(df['age'] , fit=norm);

(mu, sigma) = norm.fit(df['age'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('Age distribution')

fig = plt.figure()
```

```
res = stats.probplot(df['age'], plot=plt)
plt.show()
```

```
mu = 60.83 and sigma = 11.87
```

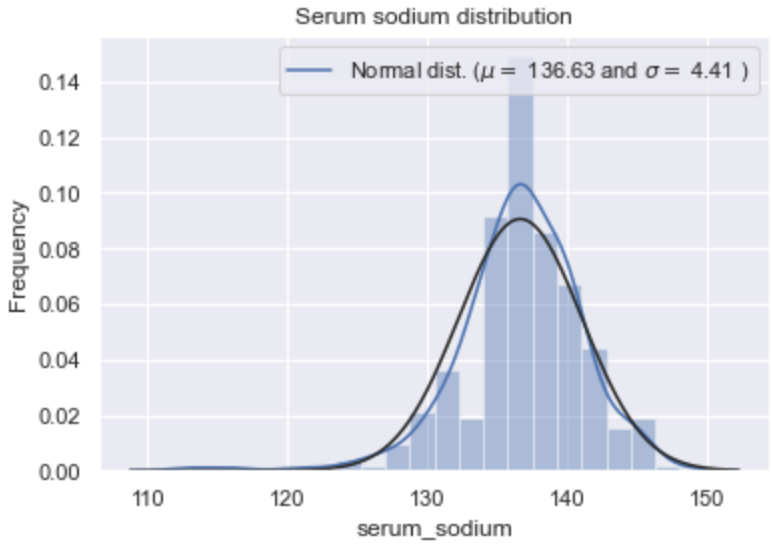


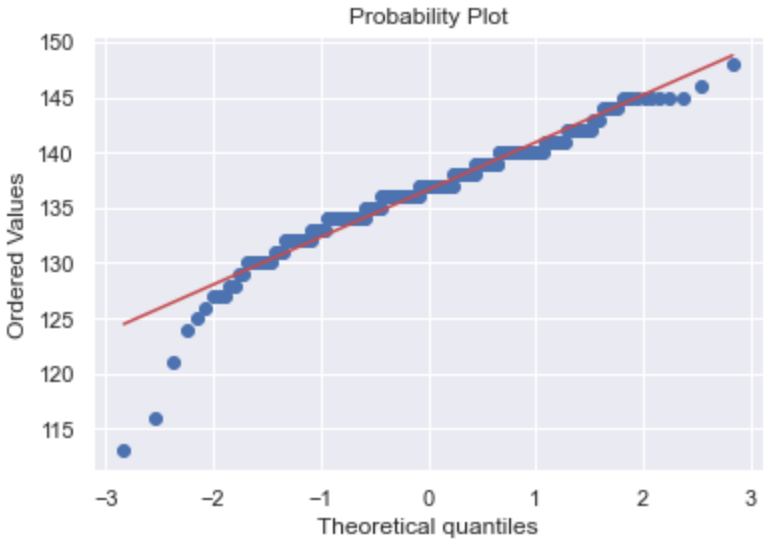


In [30]:

```
from scipy import stats
from scipy.stats import norm, skew

sns.set(rc={'figure.figsize':(6,4)})
sns.distplot(df['serum_sodium'] , fit=norm);

(mu, sigma) = norm.fit(df['serum_sodium'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('Serum sodium distribution')

fig = plt.figure()
res = stats.probplot(df['serum_sodium'], plot=plt)
plt.show()
```

```
mu = 136.63 and sigma = 4.41
```

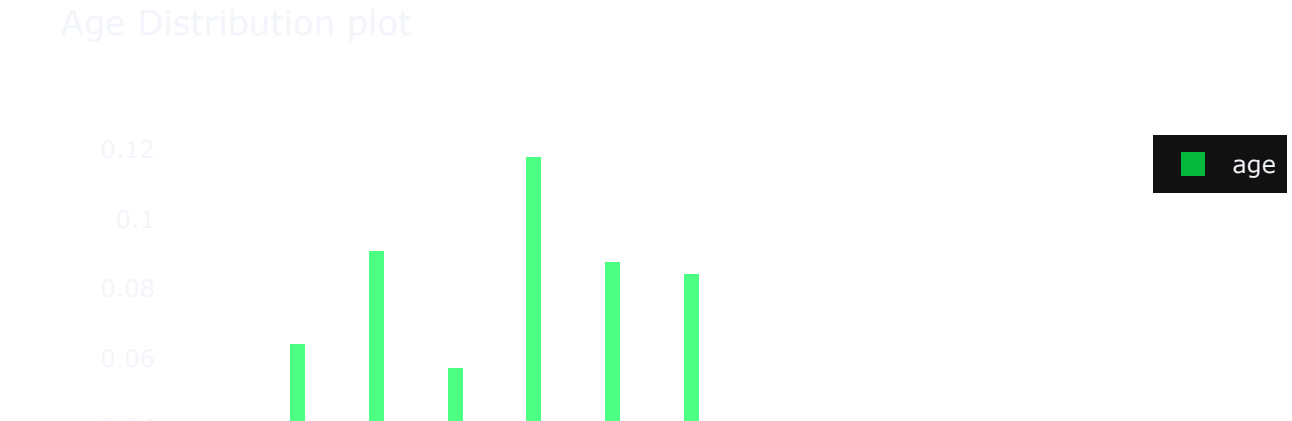


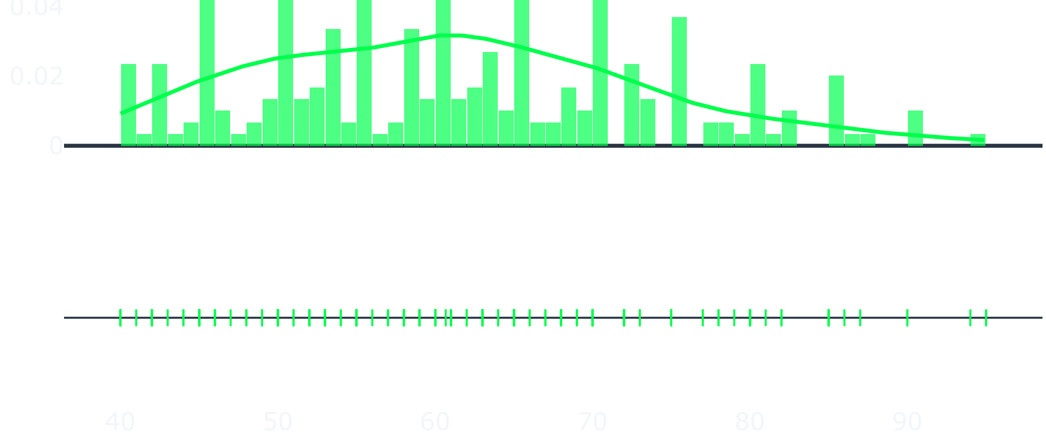


Distribution Plots and Histograms visualizing different features of the dataset

In [31]:

```
hist_age =[df["age"].values]
group_labels = ['age']
red = ['rgb(0, 255, 76)']

fig = ff.create_distplot(hist_age, group_labels, colors = red)
fig.update_layout(title_text='Age Distribution plot', template = 'plotly_dark', bargap=0.(


fig.show()
```

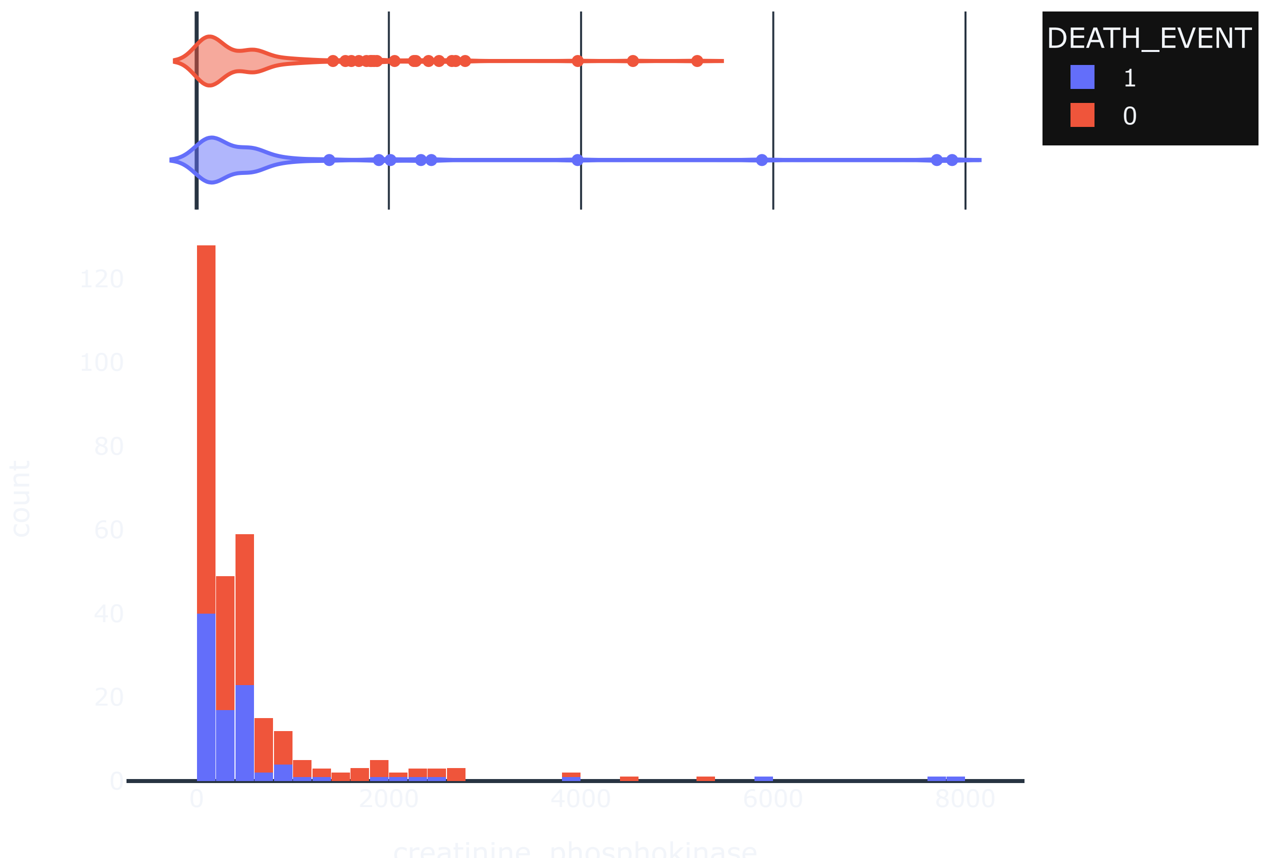




It can be observed that most number of patients are aged between 50 and 80.

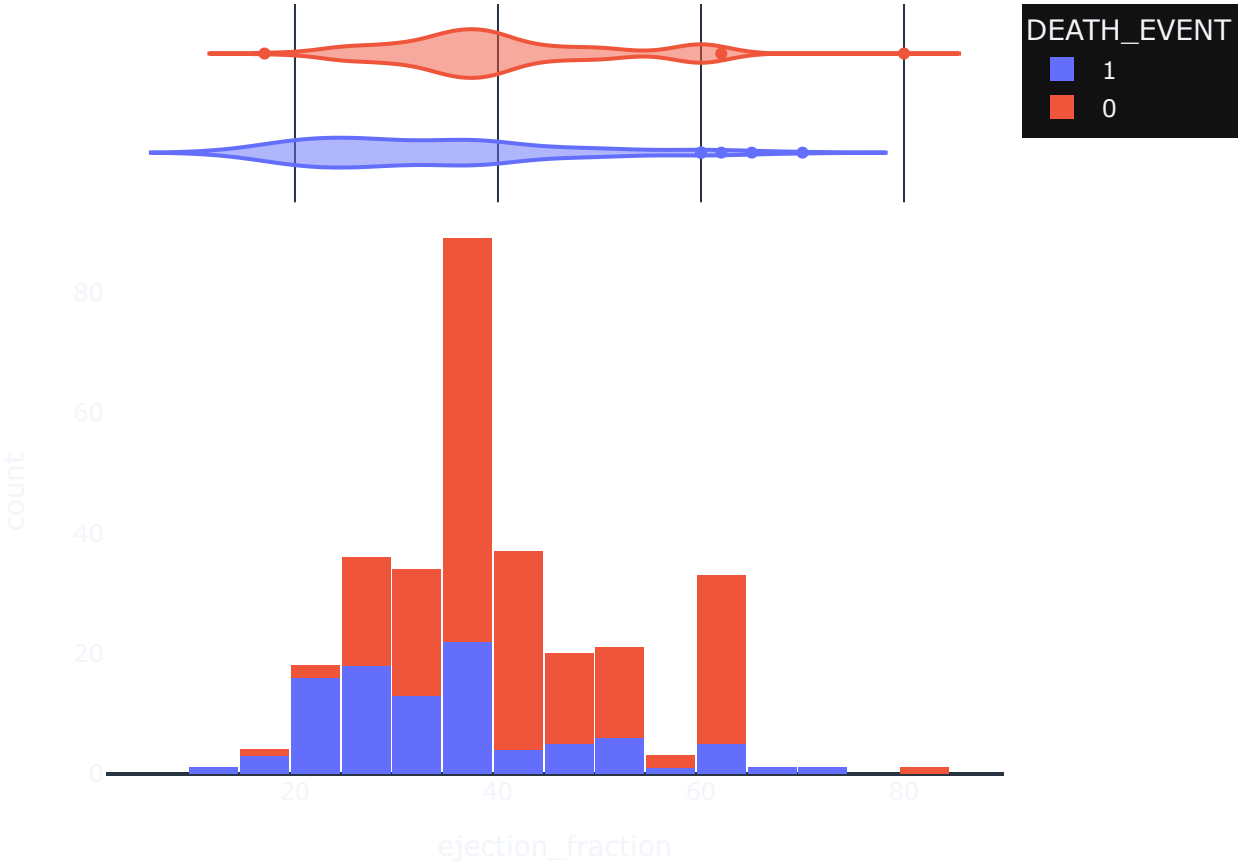
In [32]:

```
fig = px.histogram(df, x="creatinine_phosphokinase", color="DEATH_EVENT", marginal="violin
fig.update_layout(template = 'plotly_dark', bargap=0.05, xaxis =  {'showgrid': False }, ya
fig.show()
```



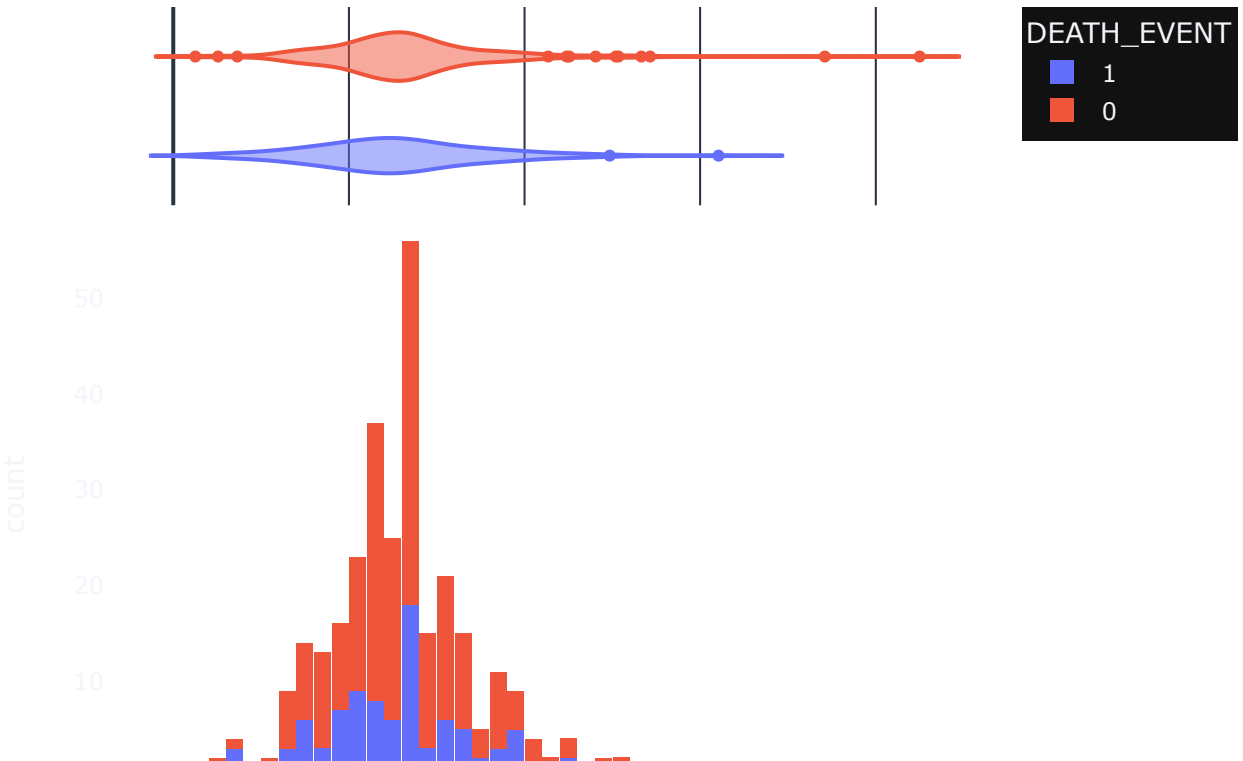
In [33]:

```
fig = px.histogram(df, x="ejection_fraction", color="DEATH_EVENT", marginal="violin", hove
fig.update_layout(template = 'plotly_dark', bargap=0.05, xaxis =  {'showgrid': False }, ya
fig.show()
```



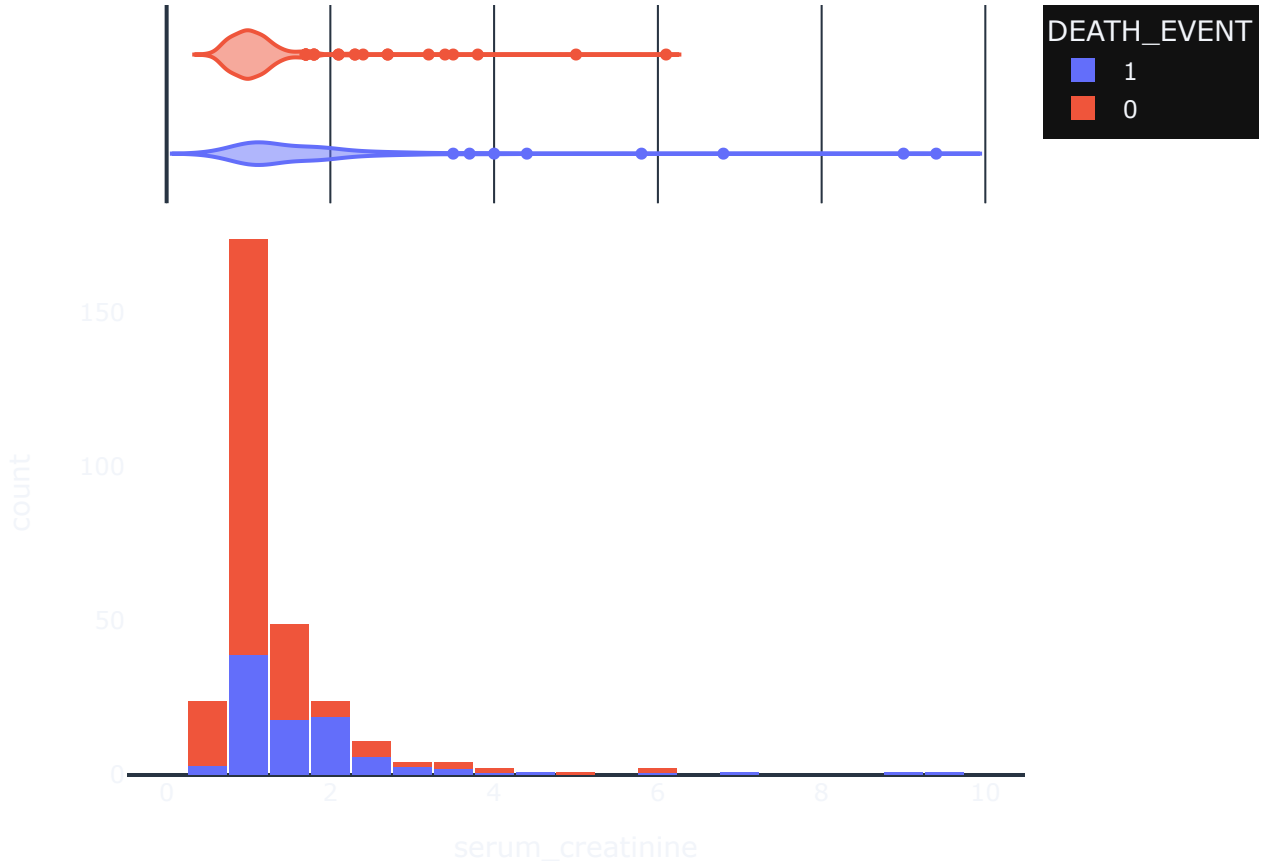
In [34]:

```
fig = px.histogram(df, x="platelets", color="DEATH_EVENT", marginal="violin", hover_data=c
fig.update_layout(template = 'plotly_dark', bargap=0.05, xaxis =  {'showgrid': False }, ya

fig.show()
```



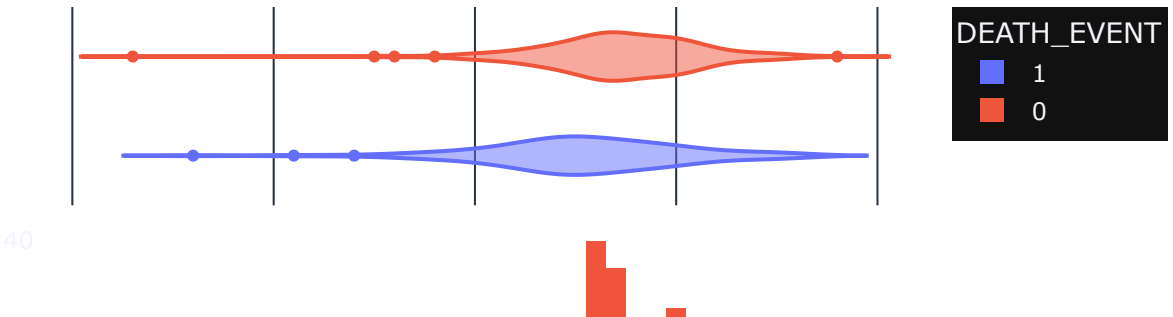
In [35]:

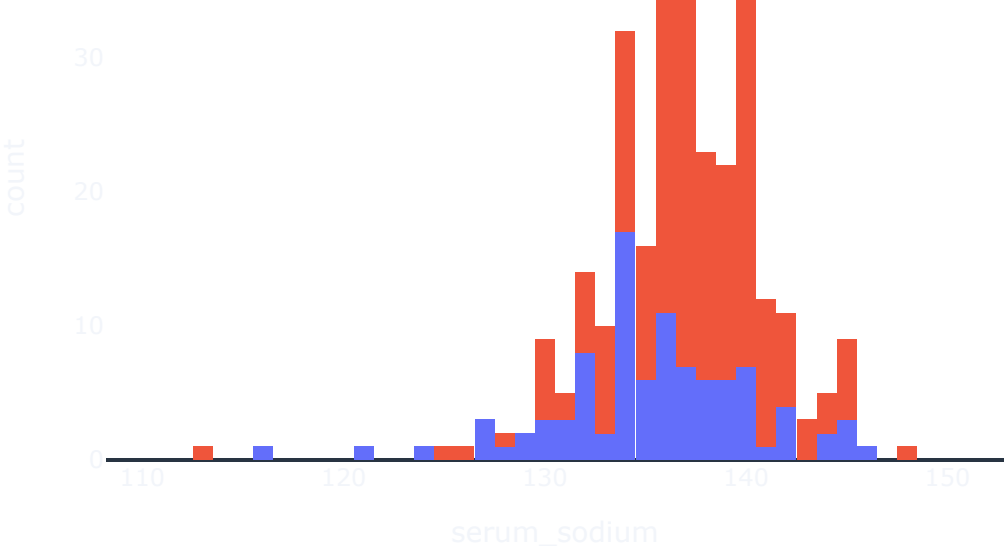
```
fig = px.histogram(df, x="serum_creatinine", color="DEATH_EVENT", marginal="violin", hover
fig.update_layout(template = 'plotly_dark', bargap=0.05, xaxis =  {'showgrid': False }, ya

fig.show()
```



In [36]:

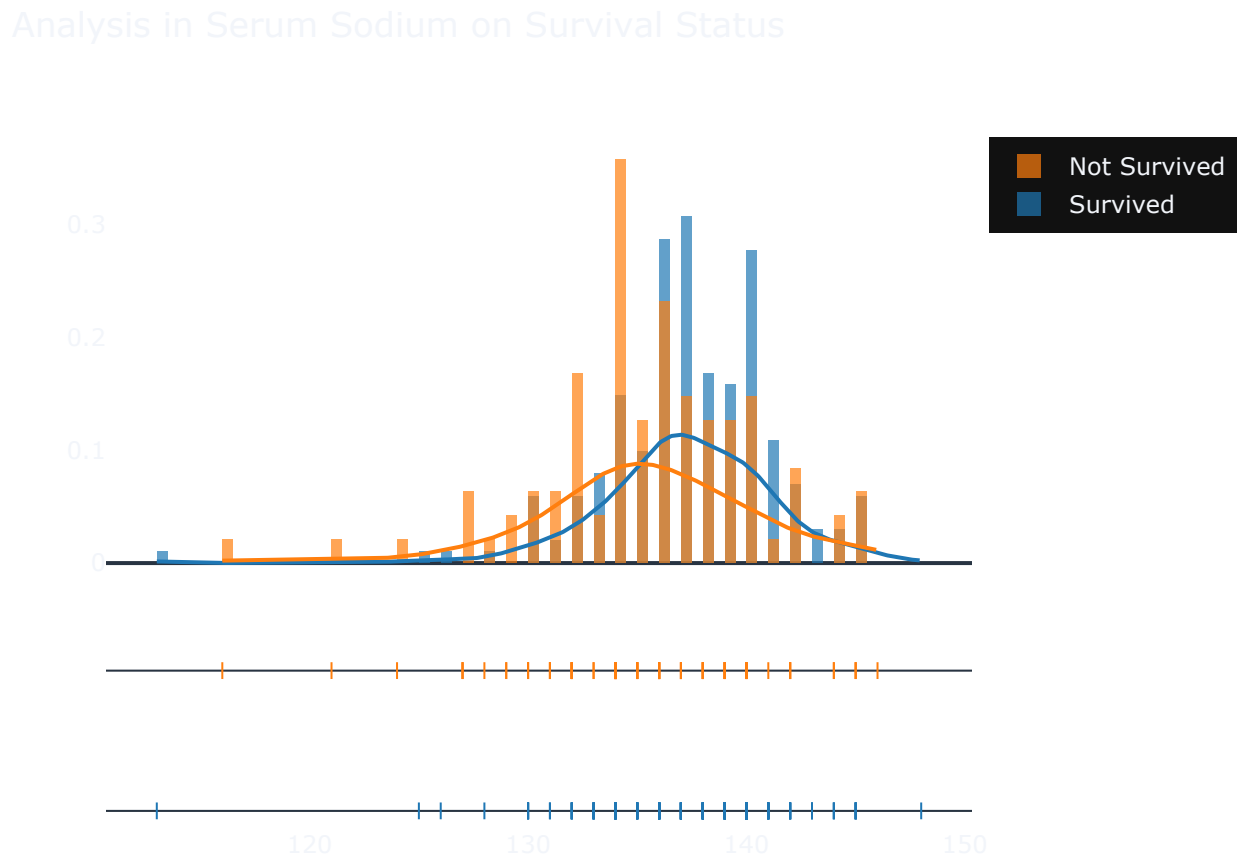
```
fig = px.histogram(df, x="serum_sodium", color="DEATH_EVENT", marginal="violin",hover_data
fig.update_layout(template = 'plotly_dark', bargap=0.05, xaxis =  {'showgrid': False }, ya

fig.show()
```





In [37]:

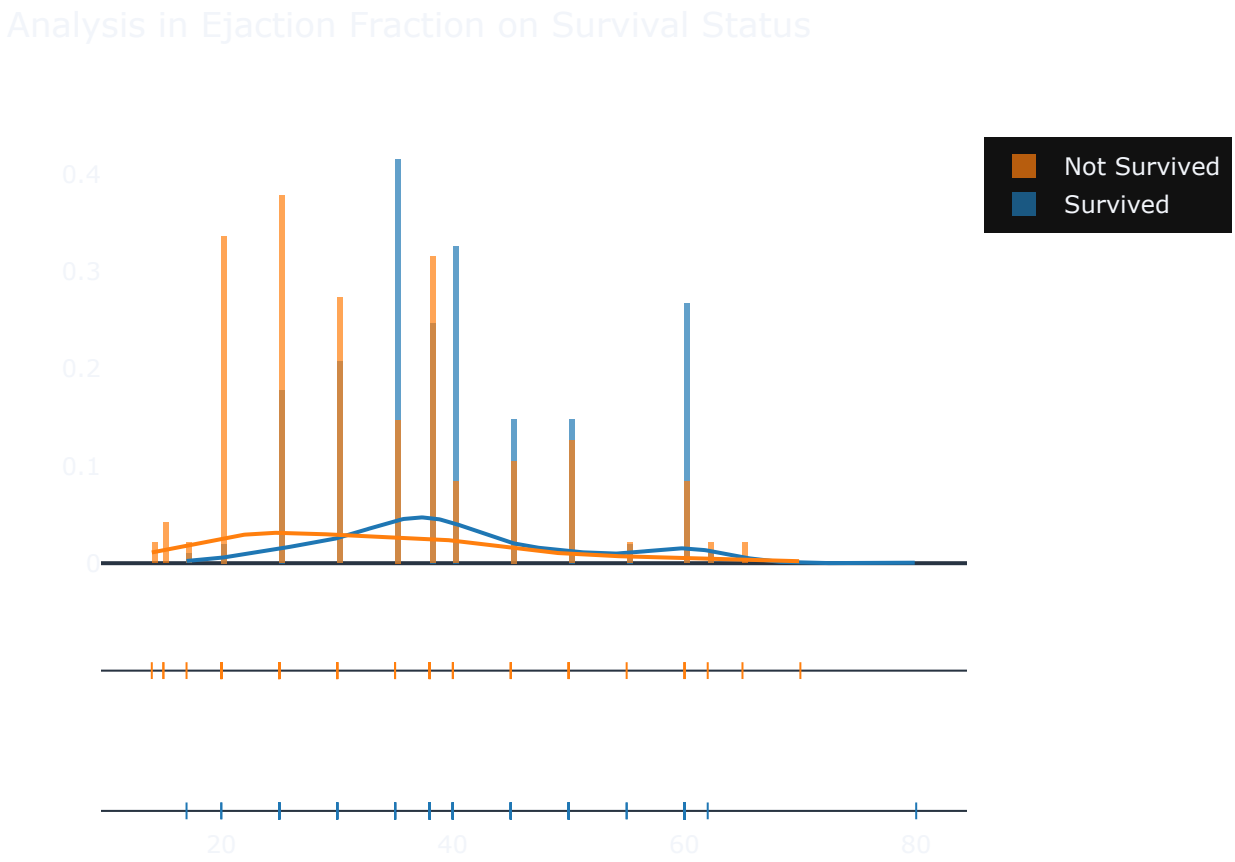
```
surv = df[df['DEATH_EVENT']==0]['serum_sodium']
not_surv = df[df['DEATH_EVENT']==1]['serum_sodium']
hist_data = [surv,not_surv]
group_labels = ['Survived', 'Not Survived']
fig = ff.create_distplot(hist_data, group_labels, bin_size=0.5)
fig.update_layout(title_text="Analysis in Serum Sodium on Survival Status", template = 'pl
fig.show()
```



In [38]:

```
surv = df[df['DEATH_EVENT']==0]['ejection_fraction']
```

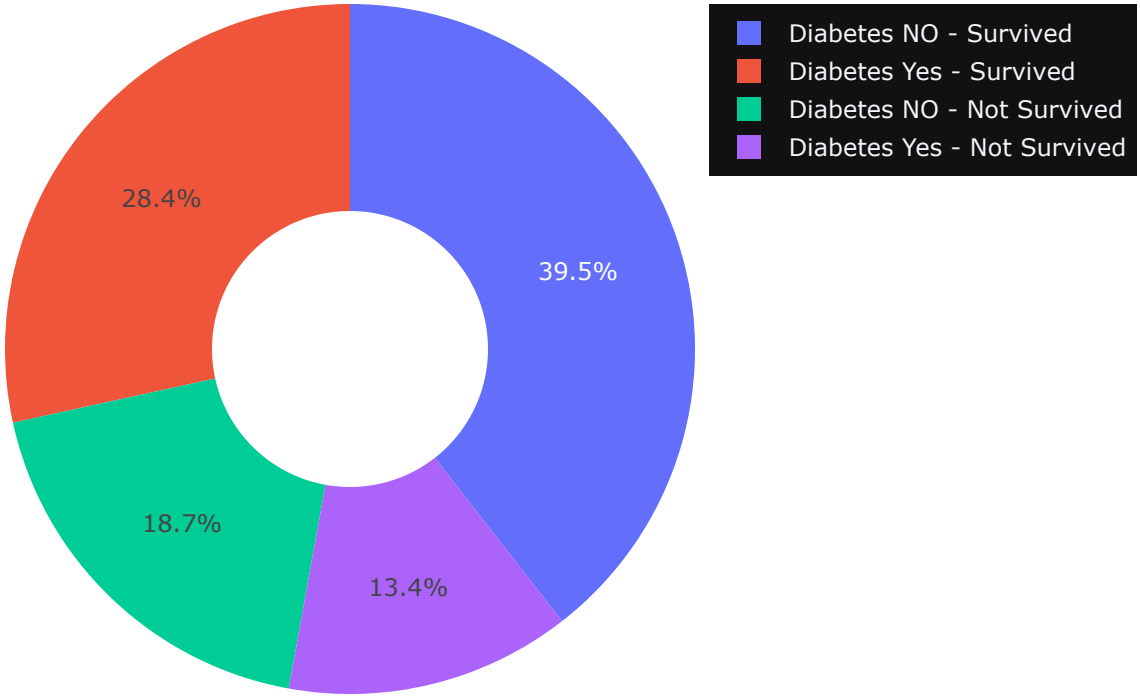
```
not_surv = df[df['DEATH_EVENT']==1]['ejection_fraction']
hist_data = [surv,not_surv]
group_labels = ['Survived', 'Not Survived']
fig = ff.create_distplot(hist_data, group_labels, bin_size=0.5)
fig.update_layout(title_text="Analysis in Ejaction Fraction on Survival Status", template

fig.show()
```



From the above, graphs Data is deeply visualized.

In [39]:

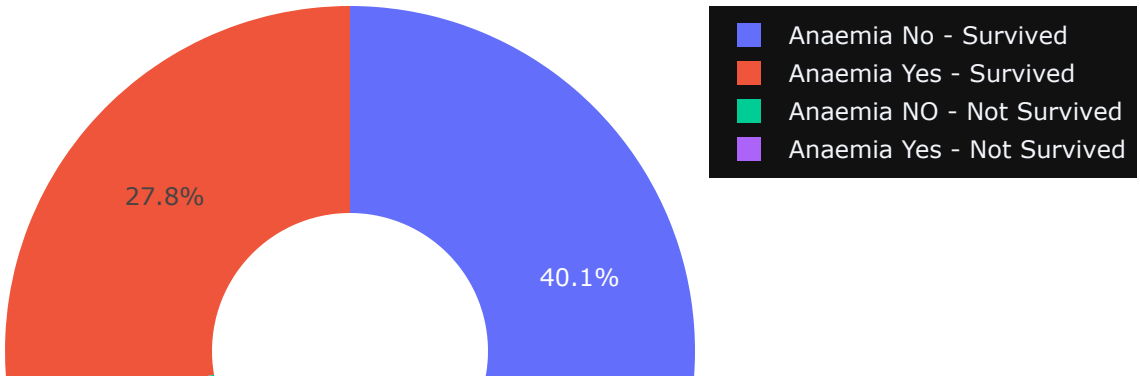
```
diabetes_yes = df[df['diabetes']==1]
diabetes_no = df[df['diabetes']==0]

diabetes_yes_survi = diabetes_yes[df["DEATH_EVENT"]==0]
diabetes_yes_not_survi = diabetes_yes[df["DEATH_EVENT"]==1]
diabetes_no_survi = diabetes_no[df["DEATH_EVENT"]==0]
diabetes__no_not_survi = diabetes_no[df["DEATH_EVENT"]==1]

labels = ['Diabetes Yes - Survived','Diabetes Yes - Not Survived', 'Diabetes NO - Survived
values = [len(diabetes_yes[df["DEATH_EVENT"]==0]),len(diabetes_yes[df["DEATH_EVENT"]==1]),
          len(diabetes_no[df["DEATH_EVENT"]==0]),len(diabetes_no[df["DEATH_EVENT"]==1])]
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.4)])
fig.update_layout(title_text="Analysis on Survival - Diabetes", template = 'plotly_dark',
fig.show()
```

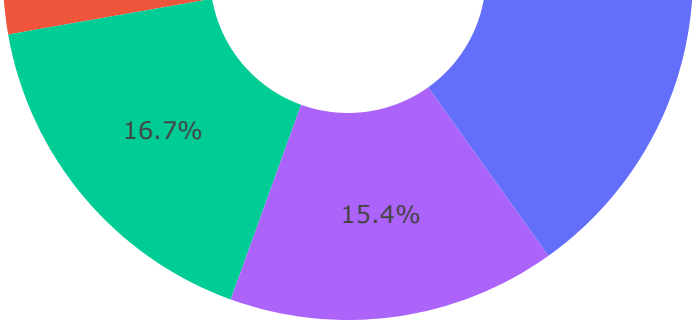


From the above pie chart it can be observed that in our dataset if a diabetic patient survived or not and vice-versa.

In [40]:

```
anaemia_yes = df[df['anaemia']==1]
anaemia_no = df[df['anaemia']==0]

anaemia_yes_survi = anaemia_yes[df["DEATH_EVENT"]==0]
anaemia_yes_not_survi = anaemia_yes[df["DEATH_EVENT"]==1]
anaemia_no_survi = anaemia_no[df["DEATH_EVENT"]==0]
anaemia_no_not_survi = anaemia_no[df["DEATH_EVENT"]==1]

labels = ['Anaemia Yes - Survived','Anaemia Yes - Not Survived', 'Anaemia No - Survived',
values = [len(anaemia_yes[df["DEATH_EVENT"]==0]),len(anaemia_yes[df["DEATH_EVENT"]==1]),
          len(anaemia_no[df["DEATH_EVENT"]==0]),len(anaemia_no[df["DEATH_EVENT"]==1])]
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.4)])
fig.update_layout(title_text="Analysis on Survival - Anaemia", template = 'plotly_dark', k

fig.show()
```

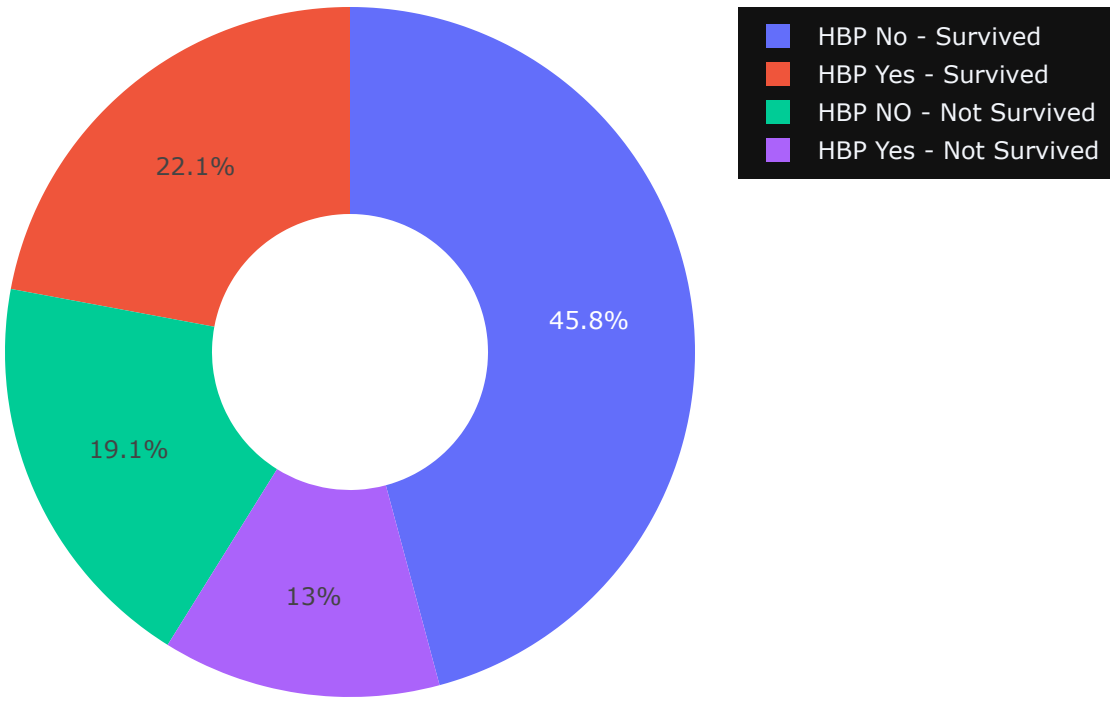




From the above pie chart it can be observed that in our dataset if a Anaemic patient survived or not and vice-versa.

In [41]:

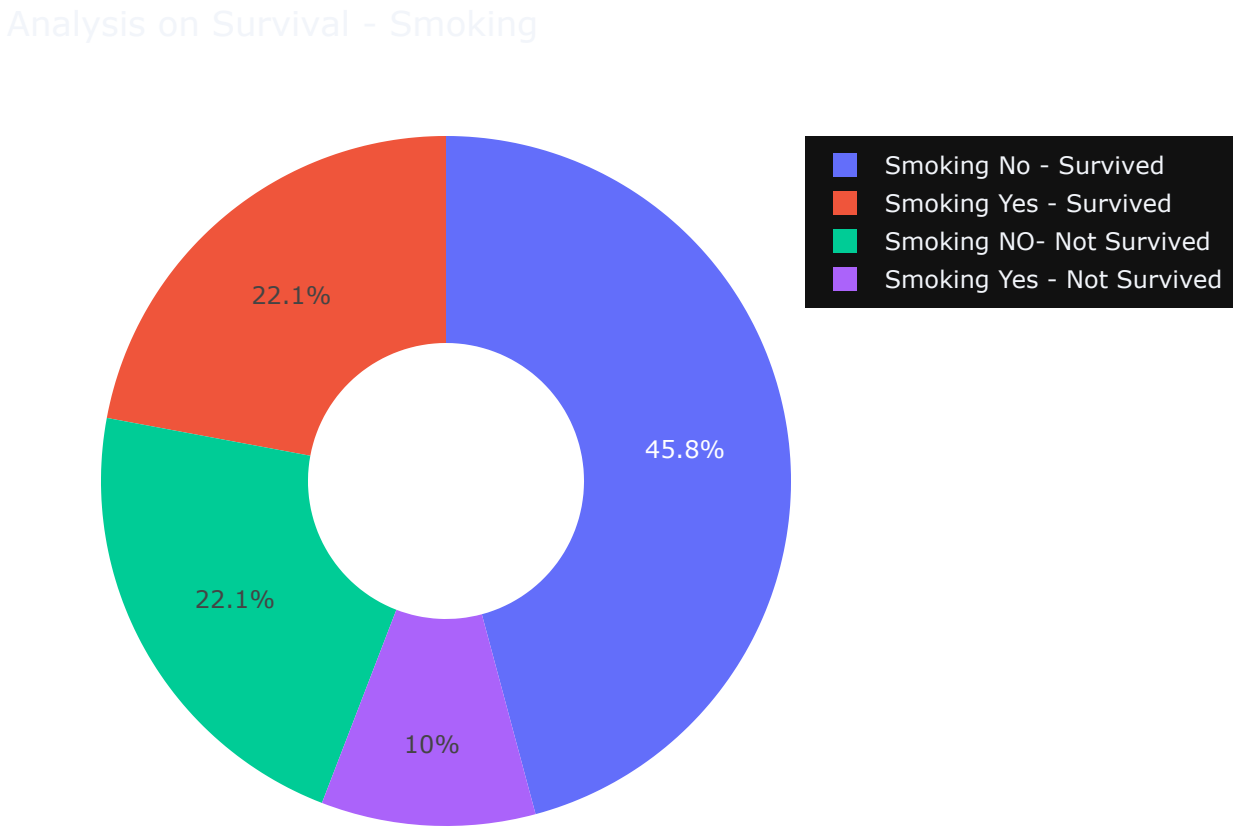
```
hbp_yes = df[df['high_blood_pressure']==1]
hbp_no = df[df['high_blood_pressure']==0]

hbp_yes_survi = hbp_yes[df["DEATH_EVENT"]==0]
hbp_yes_not_survi = hbp_yes[df["DEATH_EVENT"]==1]
hbp_no_survi = hbp_no[df["DEATH_EVENT"]==0]
hbp_no_not_survi = hbp_no[df["DEATH_EVENT"]==1]

labels = ['HBP Yes - Survived','HBP Yes - Not Survived', 'HBP No - Survived', 'HBP NO - No
values = [len(hbp_yes[df["DEATH_EVENT"]==0]),len(hbp_yes[df["DEATH_EVENT"]==1]),
          len(hbp_no[df["DEATH_EVENT"]==0]),len(hbp_no[df["DEATH_EVENT"]==1])]
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.4)])
fig.update_layout(title_text="Analysis on Survival - HBP(high blood pressure)", template =

fig.show()
```



From the above pie chart it can be observed that in our dataset if a high blood pressure patient survived or not and vice-versa.

In [42]:

```
smoking_yes = df[df['smoking']==1]
smoking_no = df[df['smoking']==0]

smoking_yes_survi = df[df["DEATH_EVENT"]==0]
smoking_yes_not_survi = df[df["DEATH_EVENT"]==1]
smoking_no_survi = df[df["DEATH_EVENT"]==0]
smoking_no_not_survi = df[df["DEATH_EVENT"]==1]

labels = ['Smoking Yes - Survived','Smoking Yes - Not Survived', 'Smoking No - Survived',
values = [len(smoking_yes[df["DEATH_EVENT"]==0]),len(smoking_yes[df["DEATH_EVENT"]==1]),
          len(smoking_no[df["DEATH_EVENT"]==0]),len(smoking_no[df["DEATH_EVENT"]==1])]
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.4)])
fig.update_layout(title_text="Analysis on Survival - Smoking", template = 'plotly_dark', k

fig.show()
```

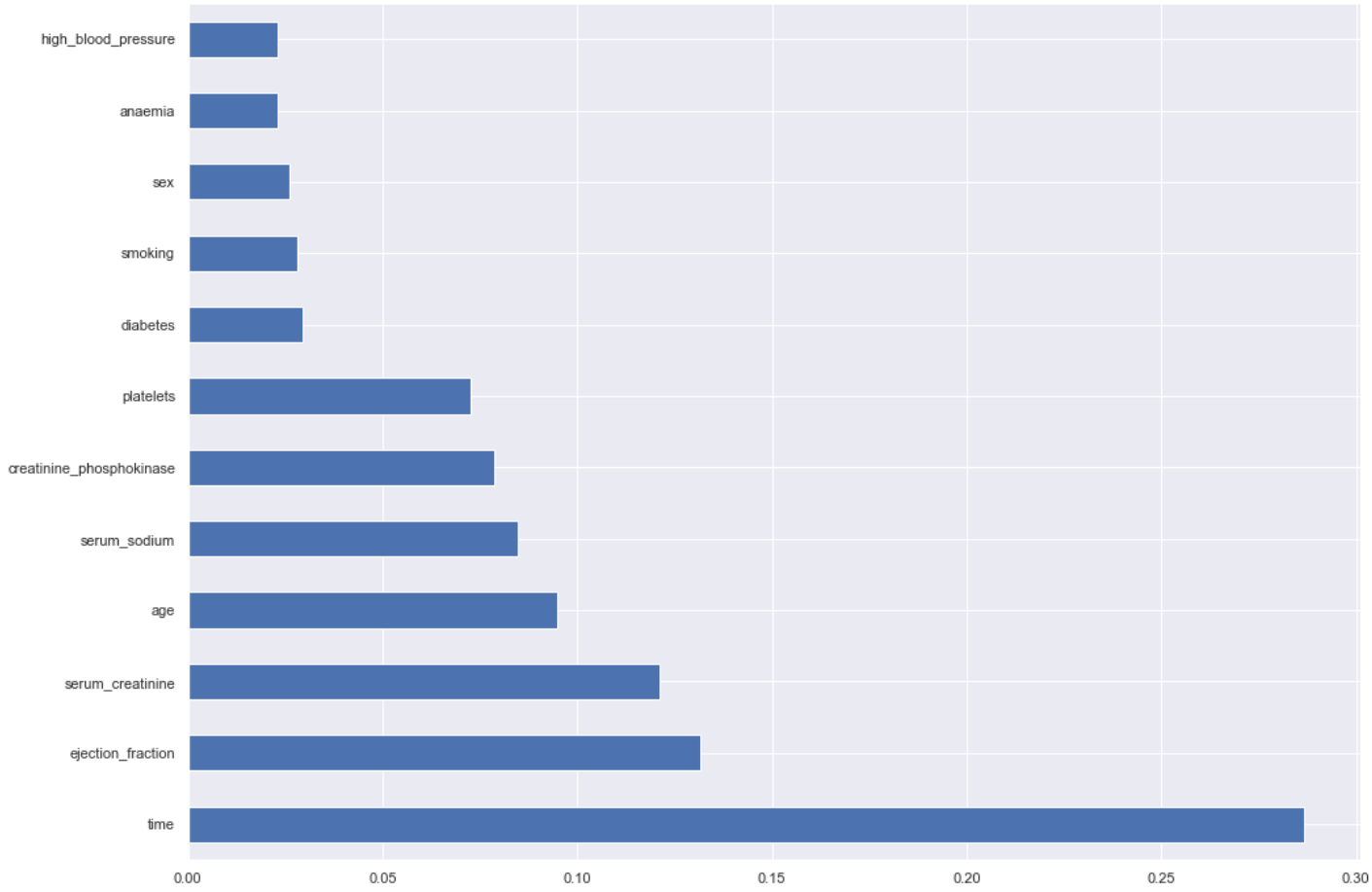


From the above pie chart it can be observed that in our dataset if a patient who smokes survived or not and vice-versa.

These are some more graphs which help in understanding the data more deeply and vastly. The above analysis of this dataset together, helps to understand, visualise, explore and breakdown the data.

In [43]:
```
plt.rcParams['figure.figsize']=16,12

x = df.iloc[:, :-1]
y = df.iloc[:,-1]

model = ExtraTreesClassifier()
model.fit(x,y)
print(model.feature_importances_)
feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(12).plot(kind='barh')
plt.show()
```

```
[0.09485565 0.02319326 0.07864496 0.02936773 0.1314365  0.02318414
 0.07273385 0.12113146 0.08464541 0.02593444 0.02820532 0.28666728]
```



After observing the graph, For the best results while model building we will select only 4 features : time, ejection_fraction, serum_creatinine and age.

Data Modelling

We will be predicting if a DEATH_EVENT occurs or not hence it is our target variable.

In [44]:
```
from sklearn.model_selection import train_test_split

Features = ['time','ejection_fraction','serum_creatinine','age']
x = df[Features]
y = df["DEATH_EVENT"]
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.2, random_state=2)
```

I chose these features for model building because they gave the best results.

In [45]:
```
accu_store = []
```

For modelling I'll be testing out three different models out of which two have been taught in class and one which i've learnt on my own.

In [46]:
```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

1. SVM

In [47]:
```
sv = SVC(kernel='rbf', gamma = 'scale')
sv.fit(x_train, y_train)
sv_pred = sv.predict(x_test)
sv_acc = accuracy_score(y_test, sv_pred)
accu_store.append(100* sv_acc)
```

In [48]:
```
sv_pred
```

Out[48]:
```
array([0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

In [49]:
```
sv.intercept_
```

Out[49]:
```
array([0.03901731])
```

In [50]:
```
print(Fore.BLACK + Back.YELLOW + Style.DIM + "Accuracy of SVM is : ", "{:.2f}%".format(100
```

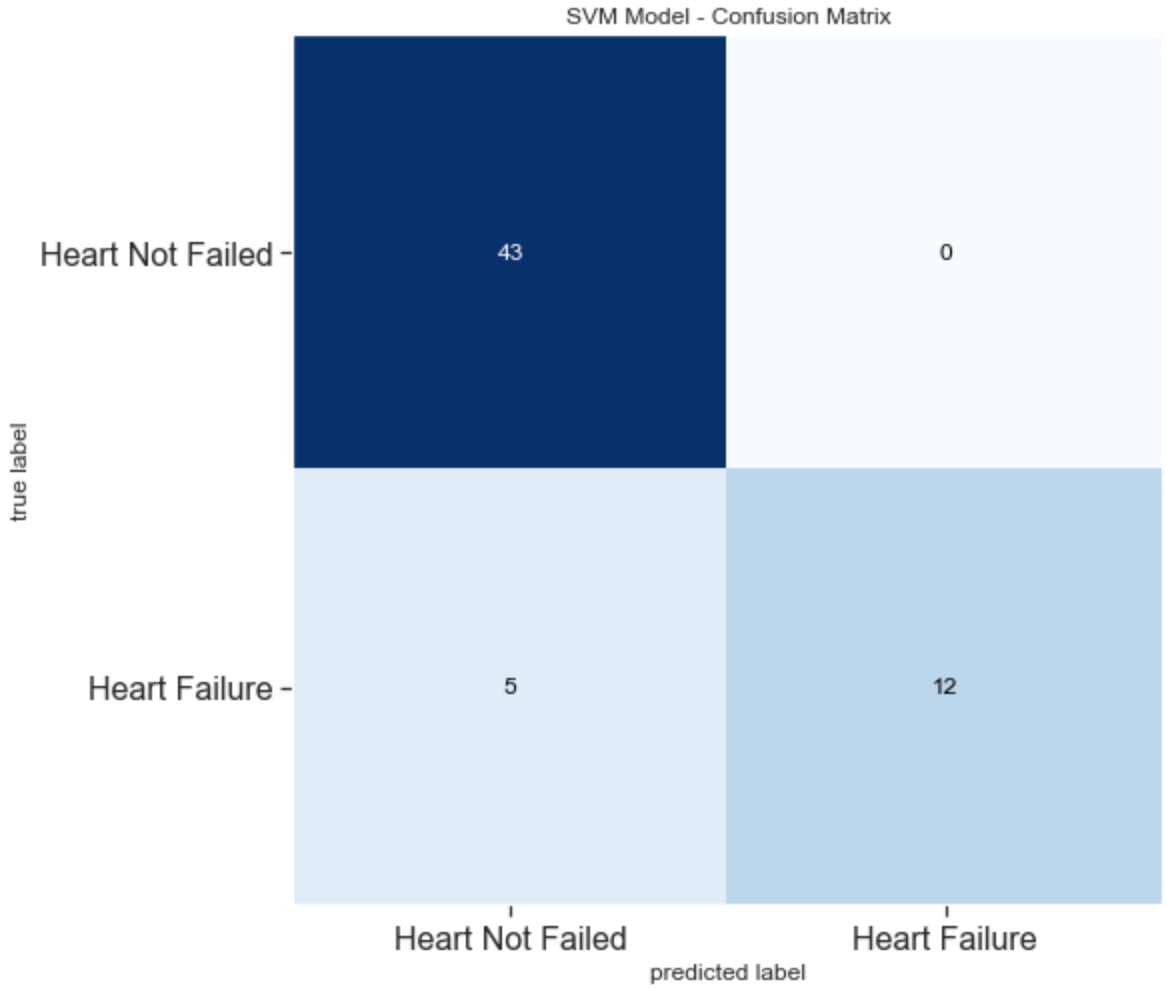
```
Accuracy of SVM is :  91.67%
```

In [51]:
```
cm = confusion_matrix(y_test, sv_pred)
cm
```

Out[51]:
```
array([[43,  0],
       [ 5, 12]], dtype=int64)
```

In [52]:
```
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("SVM Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Failure"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Failure"], fontsize=16)
plt.show()
```

```
<Figure size 1152x864 with 0 Axes>
```

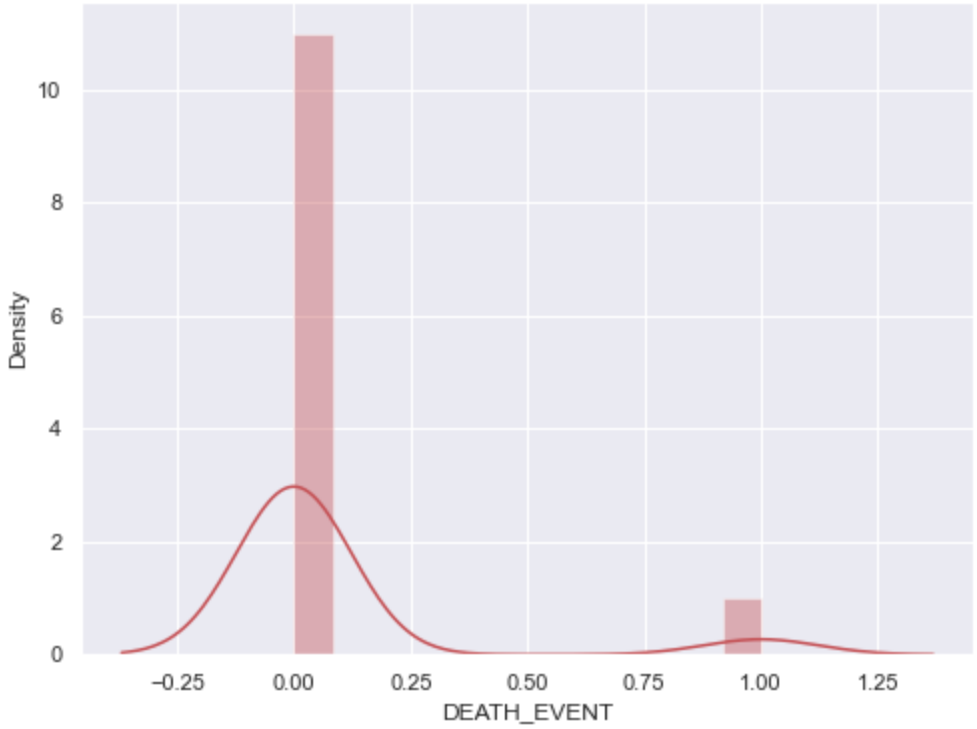


Model Evaluation

In [53]:
```
print('MAE:', metrics.mean_absolute_error(y_test, sv_pred))
print('MSE:', metrics.mean_squared_error(y_test, sv_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, sv_pred)))
```

```
MAE: 0.08333333333333333
MSE: 0.08333333333333333
RMSE: 0.28867513459481287
```

In [54]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.distplot((y_test-sv_pred),bins=12, color='r');
```



K-fold model validation

In [55]:
```
score_sv = cross_val_score(sv, x_train, y_train, scoring='r2', cv=5)
score_sv
```

Out[55]:
```
array([0.34375   , 0.4375    , 0.0625    , 0.0625    , 0.21666667])
```

In [56]:
```
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

hyper_params = [{'n_features_to_select': list(range(1, 12))}]

sv = SVC()
sv.fit(x_train, y_train)
rfe = RFE(model)

sv_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

sv_cv.fit(x_train, y_train)
```

Out[56]:
```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=ExtraTreesClassifier()),
             param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                  10, 11]}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [57]:
```
sv_cv_results = pd.DataFrame(sv_cv.cv_results_)
sv_cv_results
```

Out[57]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	sp

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	sp
0	0.295213	0.012962	0.010253	0.005081	1	{'n_features_to_select': 1}	
1	0.215369	0.005343	0.010327	0.002650	2	{'n_features_to_select': 2}	
2	0.142061	0.002992	0.006601	0.003323	3	{'n_features_to_select': 3}	
3	0.062482	0.005952	0.013051	0.006612	4	{'n_features_to_select': 4}	
4	0.067175	0.001302	0.008602	0.000490	5	{'n_features_to_select': 5}	
5	0.065614	0.002967	0.003201	0.003920	6	{'n_features_to_select': 6}	
6	0.067612	0.001024	0.008802	0.000400	7	{'n_features_to_select': 7}	
7	0.066204	0.001155	0.008402	0.000490	8	{'n_features_to_select': 8}	
8	0.065943	0.001374	0.008666	0.000922	9	{'n_features_to_select': 9}	
9	0.085450	0.034178	0.010001	0.003034	10	{'n_features_to_select': 10}	
10	0.067235	0.001454	0.008802	0.000400	11	{'n_features_to_select': 11}	

11 rows × 21 columns

In [58]:
```
kfold = model_selection.KFold(n_splits= 5)
svCV = SVC()
scoring = 'accuracy'
sv_results = model_selection.cross_val_score(svCV, x_train, y_train, cv=kfold, scoring=sc
print("K-fold cross validation average accuracy of Support Vector Machine Model: %.3f" %
```

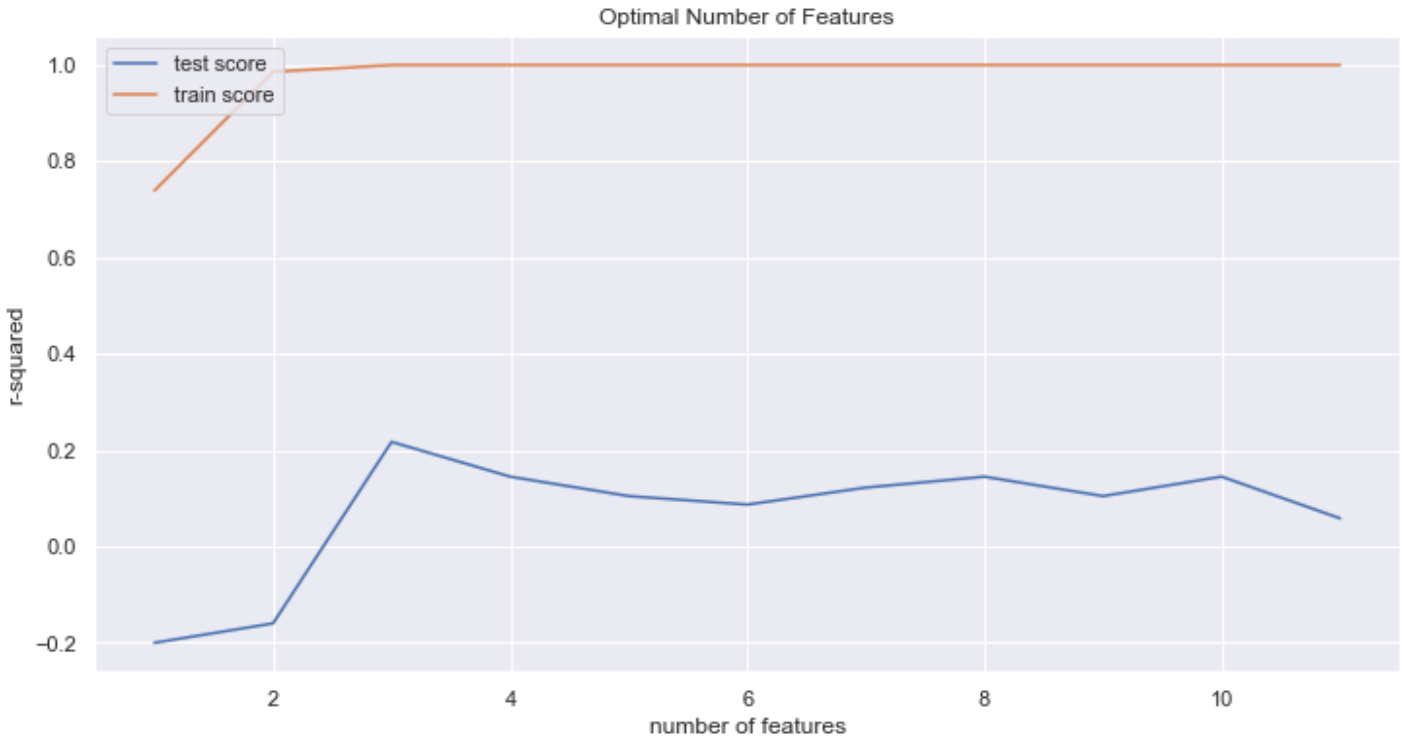
```
K-fold cross validation average accuracy of Support Vector Machine Model: 0.832
```

In [59]:
```
print(classification_report(y_test, sv_pred))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        43
           1       1.00      0.71      0.83        17

    accuracy                           0.92        60
   macro avg       0.95      0.85      0.89        60
weighted avg       0.93      0.92      0.91        60
```

In [60]:
```
plt.figure(figsize=(12,6))

plt.plot(sv_cv_results["param_n_features_to_select"], sv_cv_results["mean_test_score"])
plt.plot(sv_cv_results["param_n_features_to_select"], sv_cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
```

Out[60]:
```
<matplotlib.legend.Legend at 0x1d1e720a0d0>
```



2. Random Forest Classifier

In [61]:
```
rf = RandomForestClassifier(criterion='entropy', n_estimators=100, max_features=4, max_dep
rf.fit(x_train, y_train)
rf_pred = rf.predict(x_test)
rf_acc = accuracy_score(y_test, rf_pred)
accu_store.append(100*rf_acc)
```

In [62]:
```
rf_pred
```

Out[62]:
```
array([0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

In [63]:
```
print(Fore.BLACK + Back.YELLOW + Style.DIM + "Accuracy of Random Forest Classifier is : ",
```

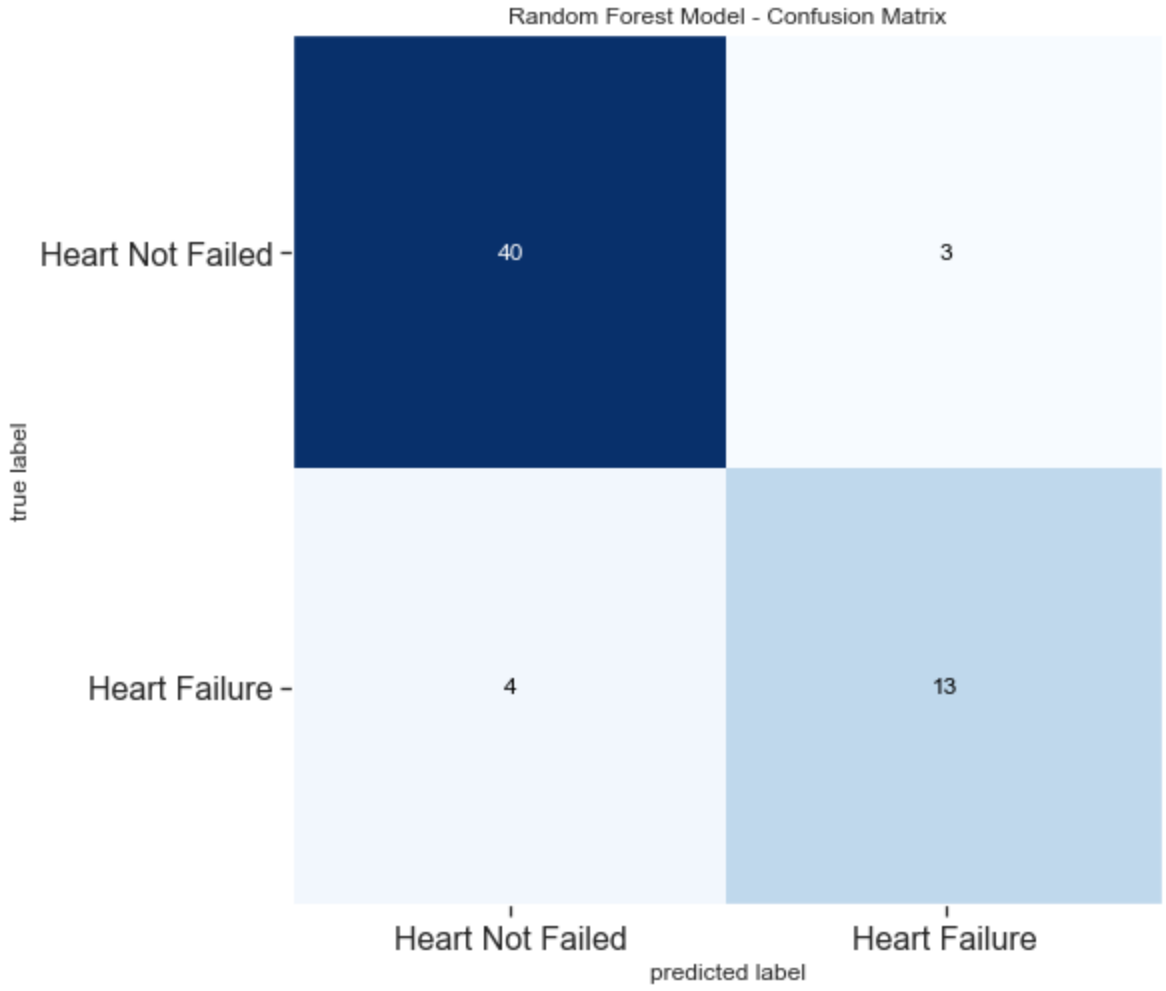
```
Accuracy of Random Forest Classifier is :  88.33%
```

In [64]:
```
cm = confusion_matrix(y_test, rf_pred)
cm
```

Out[64]:
```
array([[40,  3],
       [ 4, 13]], dtype=int64)
```

In [65]:
```
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 Failure"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Failure"], fontsize=16)
plt.show()
```

```
<Figure size 576x432 with 0 Axes>
```



Model Evaluation

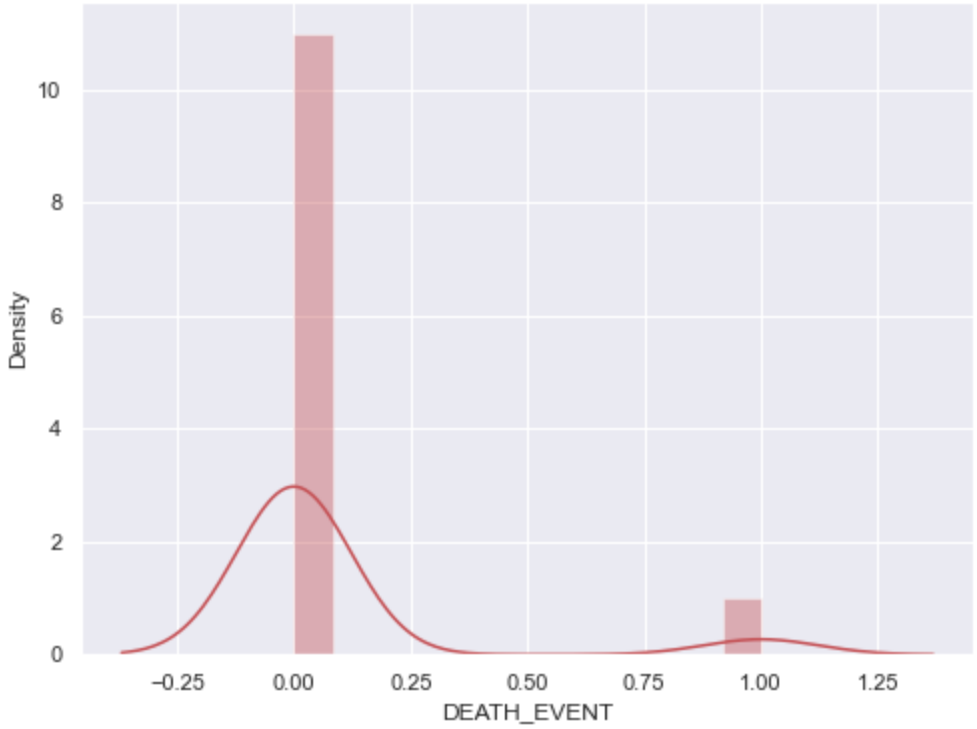
In [66]:
```
print('MAE:', metrics.mean_absolute_error(y_test, rf_pred))
print('MSE:', metrics.mean_squared_error(y_test, rf_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, rf_pred)))
```

```
MAE: 0.11666666666666667
MSE: 0.11666666666666667
RMSE: 0.3415650255319866
```

In [67]:
```
rf.feature_importances_
```

Out[67]:
```
array([0.48366623, 0.14696551, 0.22429206, 0.14507621])
```

In [68]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.distplot((y_test-sv_pred),bins=12, color='r');
```



K-fold model validation

In [69]:
```
score_rf = cross_val_score(rf, x_train, y_train, scoring='r2', cv=5)
score_rf
```

Out[69]:
```
array([ 0.34375,  0.4375 ,  0.0625 , -0.03125, -0.175  ])
```

In [70]:
```
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

hyper_params = [{'n_features_to_select': list(range(1, 12))}]

rf = RandomForestClassifier(max_features=0.5, max_depth=15, random_state=1)
rf.fit(x_train, y_train)
rfe = RFE(model)

rf_cv = GridSearchCV(estimator = rfe,
                     param_grid = hyper_params,
                     scoring= 'r2',
                     cv = folds,
                     verbose = 1,
                     return_train_score=True)

# fit the model
rf_cv.fit(x_train, y_train)
```

Out[70]:
```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=ExtraTreesClassifier()),
             param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                  10, 11]}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [71]:
```
rf_cv_results = pd.DataFrame(rf_cv.cv_results_)
rf_cv_results
```

Out[71]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	sp

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	sp
0	0.289580	0.005304	0.009928	0.002888	1	{'n_features_to_select': 1}	
1	0.214970	0.004444	0.007001	0.003522	2	{'n_features_to_select': 2}	
2	0.140937	0.011651	0.011452	0.003428	3	{'n_features_to_select': 3}	
3	0.065549	0.003114	0.005201	0.004262	4	{'n_features_to_select': 4}	
4	0.067617	0.004247	0.006402	0.003201	5	{'n_features_to_select': 5}	
5	0.063039	0.006171	0.010128	0.002780	6	{'n_features_to_select': 6}	
6	0.065327	0.008100	0.011051	0.003735	7	{'n_features_to_select': 7}	
7	0.063458	0.004663	0.011051	0.003735	8	{'n_features_to_select': 8}	
8	0.067063	0.007996	0.008257	0.004991	9	{'n_features_to_select': 9}	
9	0.065628	0.006251	0.000000	0.000000	10	{'n_features_to_select': 10}	
10	0.065833	0.001704	0.006801	0.003430	11	{'n_features_to_select': 11}	

11 rows × 21 columns

In [72]:
```
kfold = model_selection.KFold(n_splits= 5)
rfCV = RandomForestClassifier(max_features=0.5, max_depth=15, random_state=1)
scoring = 'accuracy'
rf_results = model_selection.cross_val_score(rfCV, x_train, y_train, cv=kfold, scoring=sc
print("K-fold cross validation average accuracy of Random Forest Classifier: %.3f" % (rf_
```

```
K-fold cross validation average accuracy of Random Forest Classifier: 0.803
```

In [73]:
```
print(classification_report(y_test, rf_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.93      0.92        43
           1       0.81      0.76      0.79        17

    accuracy                           0.88        60
   macro avg       0.86      0.85      0.85        60
weighted avg       0.88      0.88      0.88        60
```

In [74]:
```
plt.figure(figsize=(12,6))

plt.plot(rf_cv_results["param_n_features_to_select"], rf_cv_results["mean_test_score"])
plt.plot(rf_cv_results["param_n_features_to_select"], rf_cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
```

Out[74]: <matplotlib.legend.Legend at 0x1d1e7277fd0>

Gradient Boosting Classifier

In [75]:
```
gradientboost = GradientBoostingClassifier(max_depth=2, random_state=1)
gradientboost.fit(x_train,y_train)
gradientboost_pred = gradientboost.predict(x_test)
gradientboost_acc = accuracy_score(y_test, gradientboost_pred)
accu_store.append(100*gradientboost_acc)
```

In [76]:
```
gradientboost_pred
```

Out[76]:
```
array([0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

In [77]:
```
print(Fore.BLACK + Back.YELLOW + Style.DIM + "Accuracy of Gradient Boosting is : ", "{:.2f
```

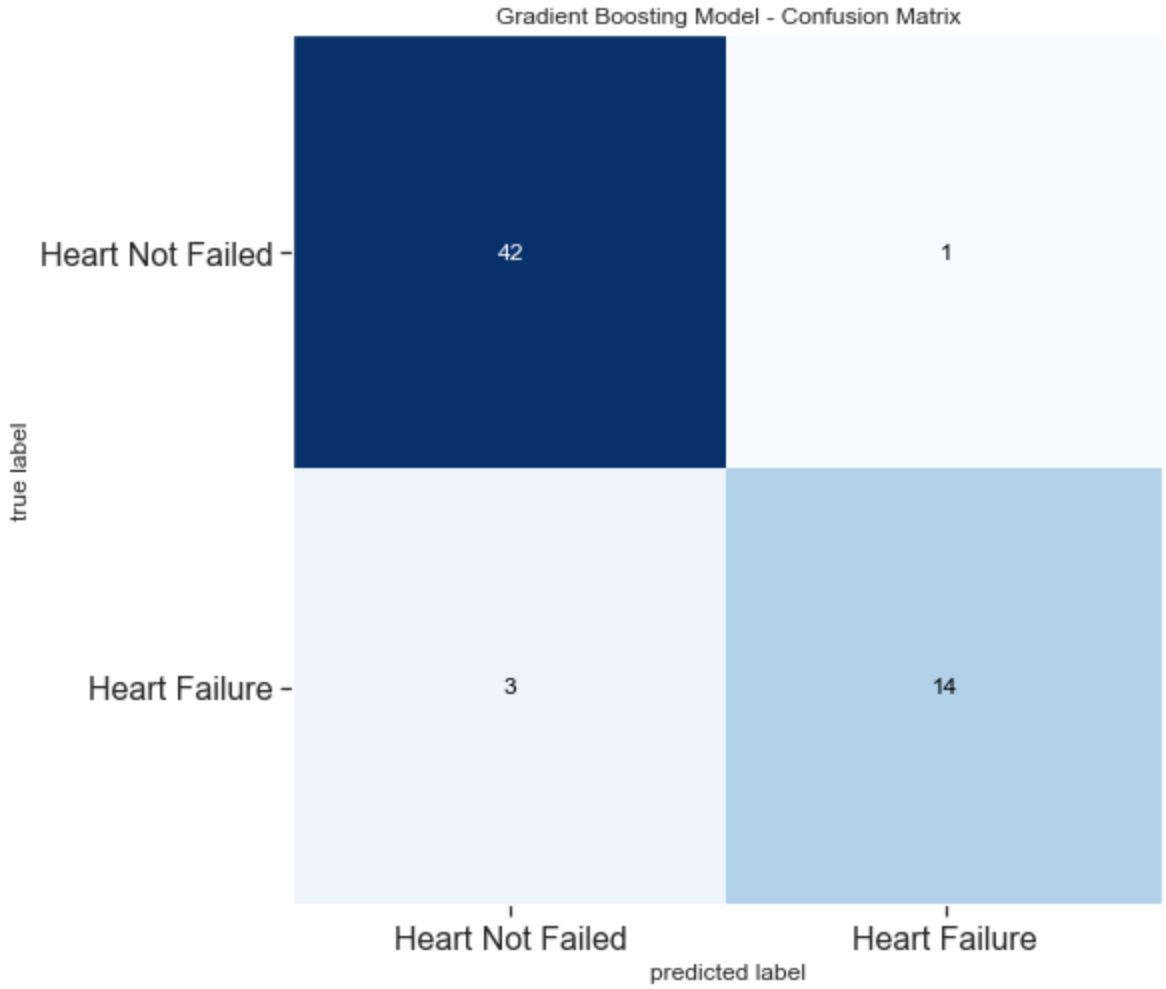
```
Accuracy of Gradient Boosting is :  93.33%
```

In [78]:
```
cm = confusion_matrix(y_test, gradientboost_pred)
cm
```

Out[78]:
```
array([[42,  1],
       [ 3, 14]], dtype=int64)
```

In [79]:
```
cm = confusion_matrix(y_test, gradientboost_pred)
plt.figure()
plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("Gradient Boosting Model - Confusion Matrix")
plt.xticks(range(2), ["Heart Not Failed","Heart Failure"], fontsize=16)
plt.yticks(range(2), ["Heart Not Failed","Heart Failure"], fontsize=16)
plt.show()
```

```
<Figure size 576x432 with 0 Axes>
```

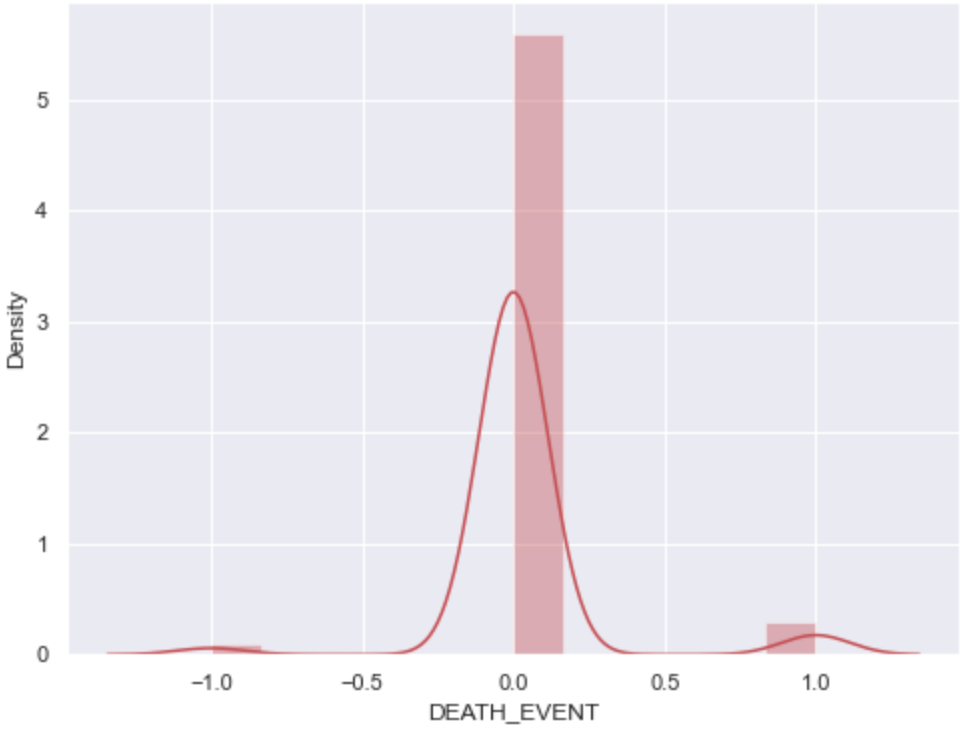


Model Evaluation

In [80]:
```
print('MAE:', metrics.mean_absolute_error(y_test, gradientboost_pred))
print('MSE:', metrics.mean_squared_error(y_test, gradientboost_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, gradientboost_pred)))
```

```
MAE: 0.06666666666666667
MSE: 0.06666666666666667
RMSE: 0.2581988897471611
```

In [81]:
```
sns.set(rc={'figure.figsize':(8,6)})
sns.distplot((y_test-gradientboost_pred),bins=12, color='r');
```



K-fold model validation

In [82]:
```
score_gradientboost = cross_val_score(gradientboost, x_train, y_train, scoring='r2', cv=5)
score_gradientboost
```

Out[82]:
```
array([ 0.34375,  0.4375 ,  0.0625 , -0.03125, -0.175  ])
```

In [83]:
```
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

hyper_params = [{'n_features_to_select': list(range(1, 12))}]

gradientboost = GradientBoostingClassifier(max_depth=2, random_state=1)
gradientboost.fit(x_train, y_train)
rfe = RFE(model)

gradientboost_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
gradientboost_cv.fit(x_train, y_train)
```

Out[83]:
```
Fitting 5 folds for each of 11 candidates, totalling 55 fits
GridSearchCV(cv=KFold(n_splits=5, random_state=100, shuffle=True),
             estimator=RFE(estimator=ExtraTreesClassifier()),
             param_grid=[{'n_features_to_select': [1, 2, 3, 4, 5, 6, 7, 8, 9,
                                                  10, 11]}],
             return_train_score=True, scoring='r2', verbose=1)
```

In [84]:
```
gb_cv_results = pd.DataFrame(gradientboost_cv.cv_results_)
gb_cv_results
```

Out[84]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	sp

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_features_to_select	params	sp
0	0.290993	0.002861	0.008602	0.000490	1	{'n_features_to_select': 1}	
1	0.215105	0.005523	0.008400	0.000488	2	{'n_features_to_select': 2}	
2	0.142896	0.001324	0.008602	0.000490	3	{'n_features_to_select': 3}	
3	0.066802	0.003785	0.010127	0.002776	4	{'n_features_to_select': 4}	
4	0.065475	0.001136	0.008201	0.000401	5	{'n_features_to_select': 5}	
5	0.066062	0.000640	0.008414	0.000480	6	{'n_features_to_select': 6}	
6	0.066703	0.000316	0.008602	0.000490	7	{'n_features_to_select': 7}	
7	0.066555	0.000571	0.008802	0.000400	8	{'n_features_to_select': 8}	
8	0.066415	0.001238	0.006802	0.003430	9	{'n_features_to_select': 9}	
9	0.067782	0.000869	0.008602	0.000489	10	{'n_features_to_select': 10}	
10	0.063856	0.007106	0.009929	0.002888	11	{'n_features_to_select': 11}	

11 rows × 21 columns

In [85]:
```
kfold = model_selection.KFold(n_splits= 5)
gradientboostCV = GradientBoostingClassifier(max_depth=2, random_state=1)
scoring = 'accuracy'
gb_results = model_selection.cross_val_score(gradientboostCV, x_train, y_train, cv=kfold,
print("K-fold cross validation average accuracy of Gradient boost Classifier: %.3f" % (gb_
```

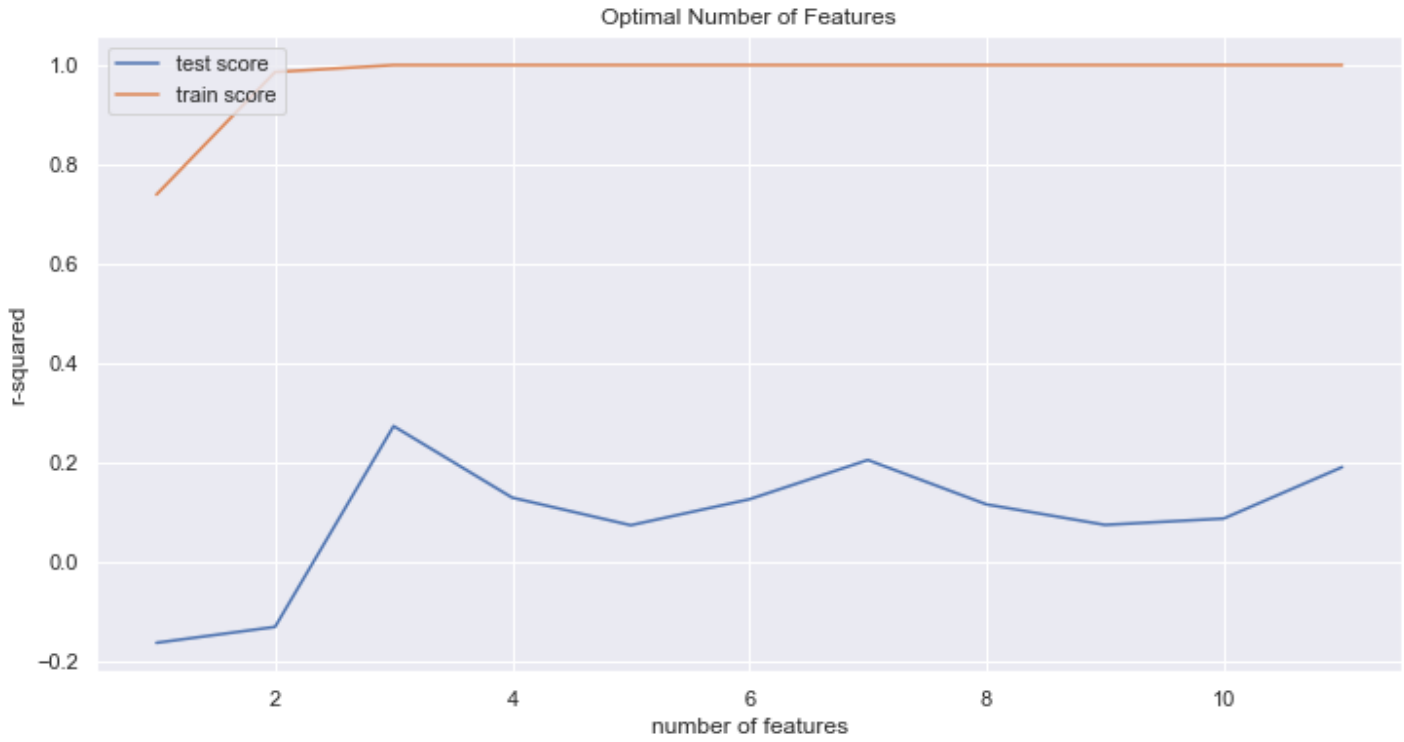
```
K-fold cross validation average accuracy of Gradient boost Classifier: 0.807
```

In [86]:
```
print(classification_report(y_test, gradientboost_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        43
           1       0.93      0.82      0.87        17

    accuracy                           0.93        60
   macro avg       0.93      0.90      0.91        60
weighted avg       0.93      0.93      0.93        60
```

In [87]:
```
plt.figure(figsize=(12,6))

plt.plot(gb_cv_results["param_n_features_to_select"], gb_cv_results["mean_test_score"])
plt.plot(gb_cv_results["param_n_features_to_select"], gb_cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
```

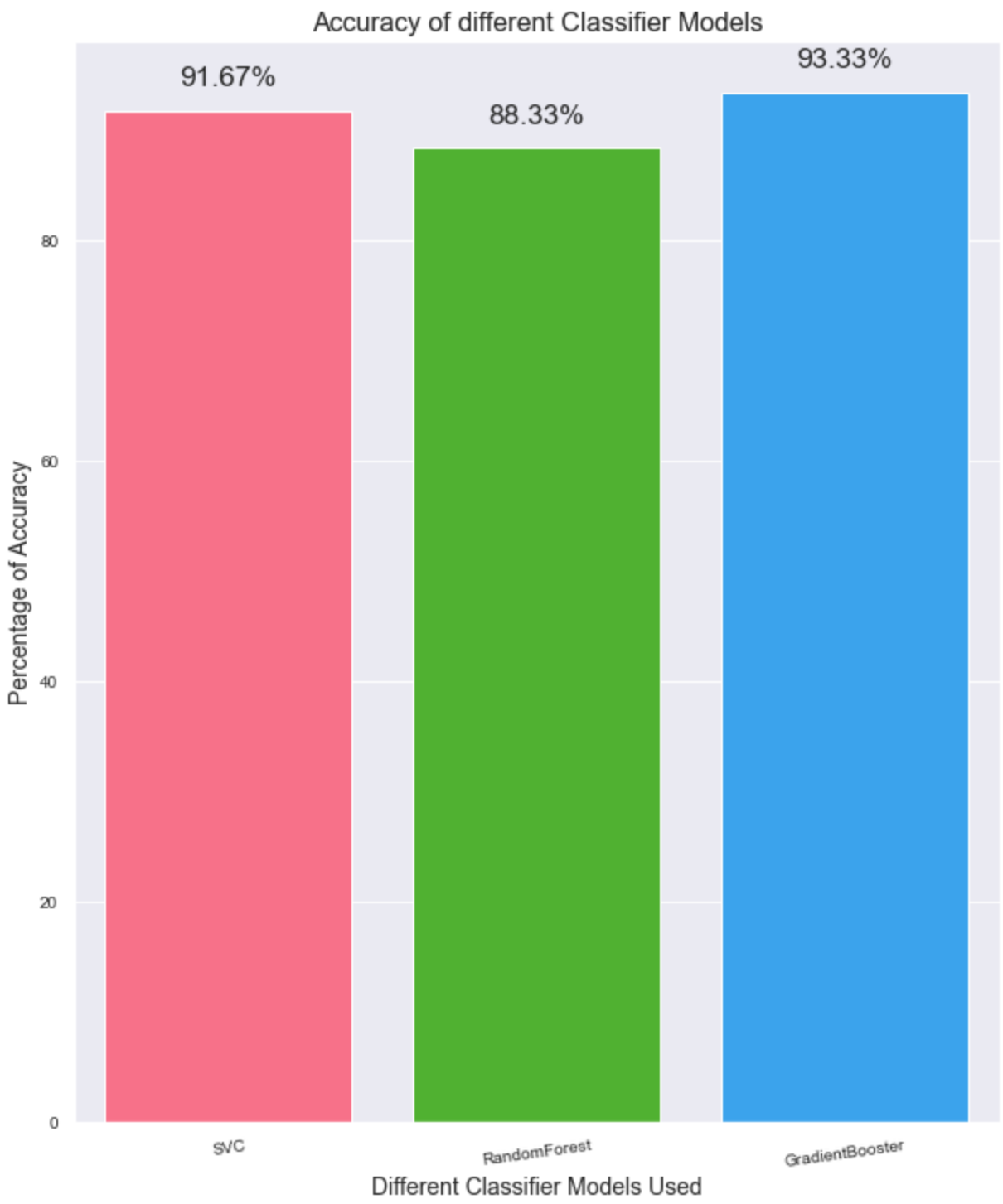
Out[87]: <matplotlib.legend.Legend at 0x1d1e47abc10>



Conclusion

In [88]:
```
model_list = ['SVC', 'RandomForest', 'GradientBooster']
```

In [89]:
```
plt.rcParams['figure.figsize']=10,12
sns.set_style('darkgrid')
ax = sns.barplot(x=model_list, y=accu_store, palette = "husl", saturation =2.0)
plt.xlabel('Different Classifier Models Used', fontsize = 14 )
plt.ylabel('Percentage of Accuracy', fontsize = 14)
plt.title('Accuracy of different Classifier Models', fontsize = 16)
plt.xticks(fontsize = 10, horizontalalignment = 'center', rotation = 8)
plt.yticks(fontsize = 10)
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*1.025), ha='center', fonts
plt.show()
```



Hence, It can be observed that for this dataset Gradient Booster Classifier Model performed the best.