Project Title : Predict survival of patients with heart failure

(Group E)

- Santhoshi Indrakanti - 11610865
- Sahithi Chindam - 11617204
- Bakht Singh Basaram - 11546730
- Shanmukha Nadh Uppugandla

Data Set Information:

The HCV dataset was obtained from the University of California at Irvine (UCI) Machine Learning Repository. This dataset comprised clinical laboratory and demographic data (includes age and sex) of 299 patients who had heart failure, collected during their follow-up period. A total of 13 features were defined for each patient record. These features are:

- anaemia: decrease of red blood cells or hemoglobin (boolean)
- high blood pressure: if the patient has hypertension (boolean)
- creatinine phosphokinase (CPK): level of the CPK enzyme in the blood (mcg/L)
- diabetes: if the patient has diabetes (boolean)
- ejection fraction: percentage of blood leaving the heart at each contraction (percentage)
- platelets: platelets in the blood (kiloplatelets/mL)
- serum creatinine: level of serum creatinine in the blood (mg/dL)
- serum sodium: level of serum sodium in the blood (mEq/L)
- smoking: if the patient smokes or not (boolean)
- time: follow-up period (days)
- death event(target): if the patient deceased during the follow-up period (boolean)

Note:

1. Dataset is collected from https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records (https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records)
2. Most of the comments mentioned and procedure followed in this notebook are from a research article https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1023-5 (https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-020-1023-5)
3. Some part of the code is reused from below sources: https://www.kaggle.com/code/calebreigada/liver-disease-analysis-eda-smote-optuna-shap/notebook (https://www.kaggle.com/code/calebreigada/liver-disease-analysis-eda-smote-optuna-shap/notebook) https://www.kaggle.com/code/tanmay111999/diabetes-classification-xgb-lgbm-stack-smote (https://www.kaggle.com/code/tanmay111999/diabetes-classification-xgb-lgbm-stack-smote)

In [180]:
```
# import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from matplotlib.gridspec import GridSpec
import warnings #warnings
warnings.filterwarnings('ignore')
```

In [181]:
```
# load and read csv file
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')
```

Initial Understanding the Dataset

In [182]:
```
# know number of samples, features, datatypes of each feature
data.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
```

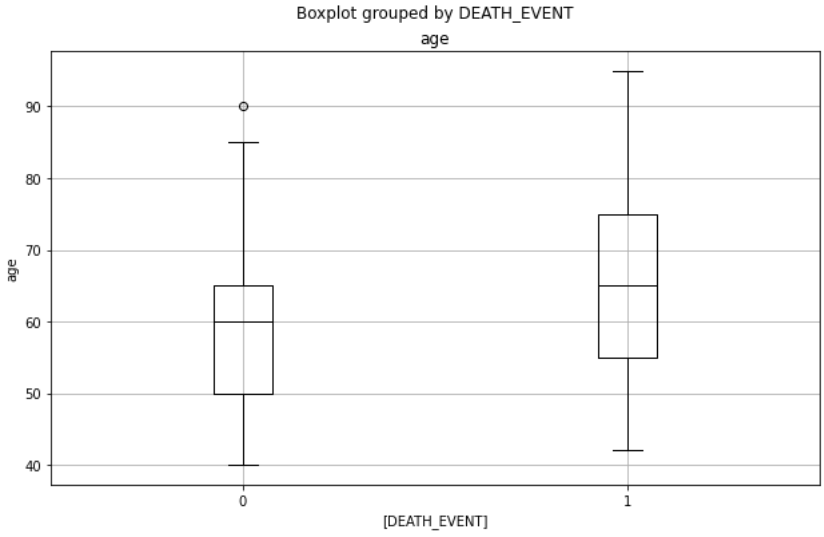
In [183]:
```
#see first five samples of the dataset
data.head()
```

Out[183]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	D
0	75.0	0	582	0	20	1	265000.00	1.9	130	1	0	4	
1	55.0	0	7861	0	38	0	263358.03	1.1	136	1	0	6	
2	65.0	0	146	0	20	0	162000.00	1.3	129	1	1	7	
3	50.0	1	111	0	20	0	210000.00	1.9	137	1	0	7	
4	65.0	1	160	1	20	0	327000.00	2.7	116	0	0	8	

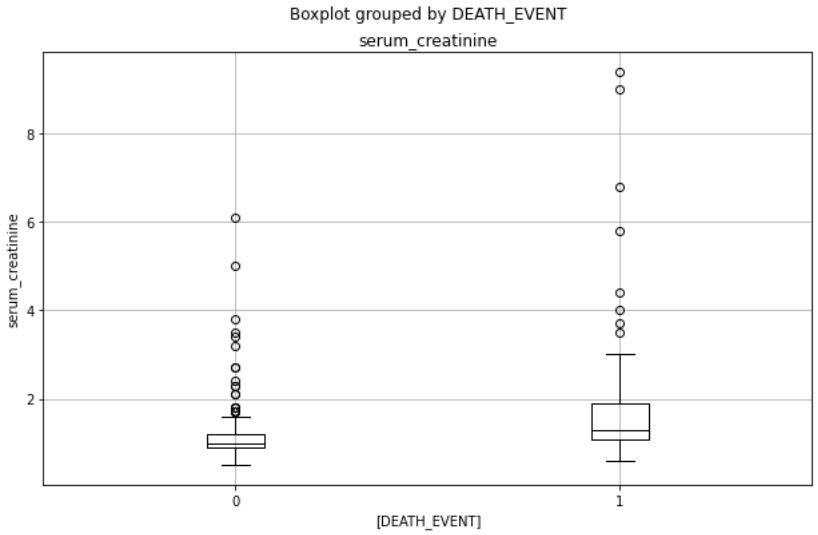
In [184]:
```
ax = data[['age', 'DEATH_EVENT']].boxplot(by='DEATH_EVENT', figsize=(10,6))
ax.set_ylabel('age')
```

Out[184]:
```
Text(0, 0.5, 'age')
```



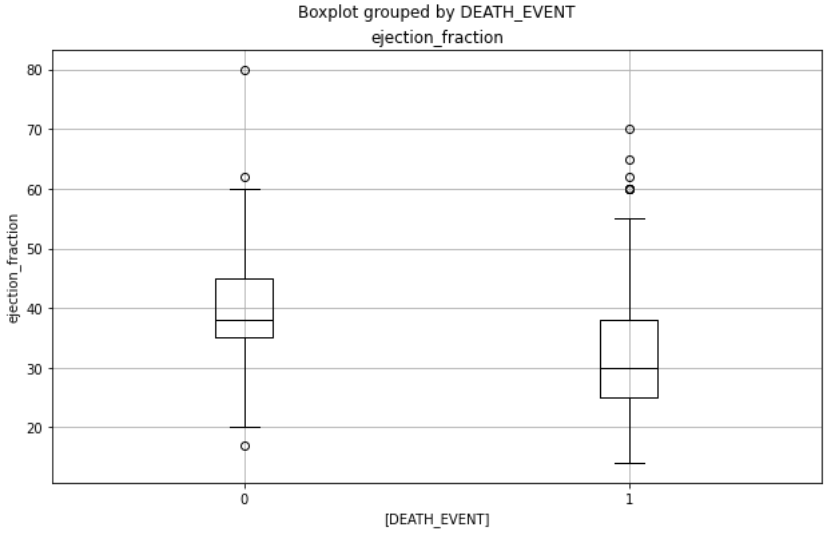
In [185]:
```
ax = data[['serum_creatinine', 'DEATH_EVENT']].boxplot(by='DEATH_EVENT', figsize=(10,6))
ax.set_ylabel('serum_creatinine')
```

Out[185]:
```
Text(0, 0.5, 'serum_creatinine')
```



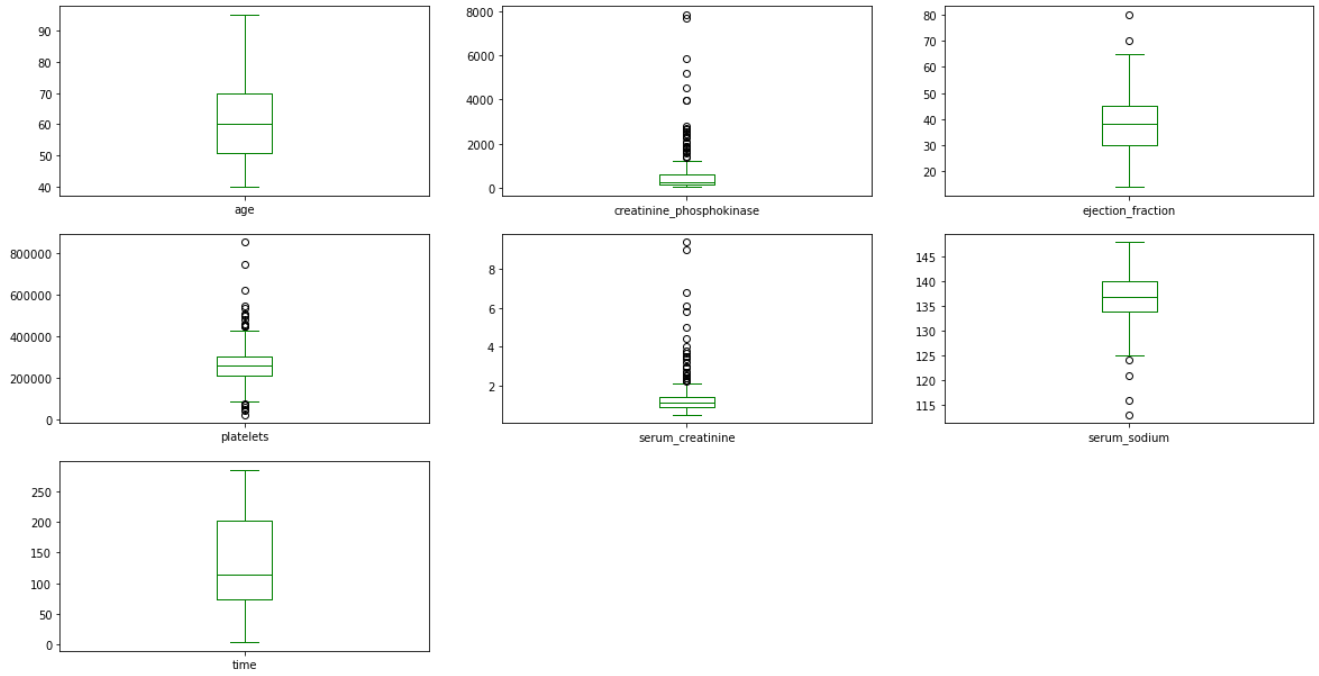
In [186]:
```
ax = data[['ejection_fraction', 'DEATH_EVENT']].boxplot(by='DEATH_EVENT', figsize=(10,6))
ax.set_ylabel('ejection_fraction')
```

Out[186]:
```
Text(0, 0.5, 'ejection_fraction')
```

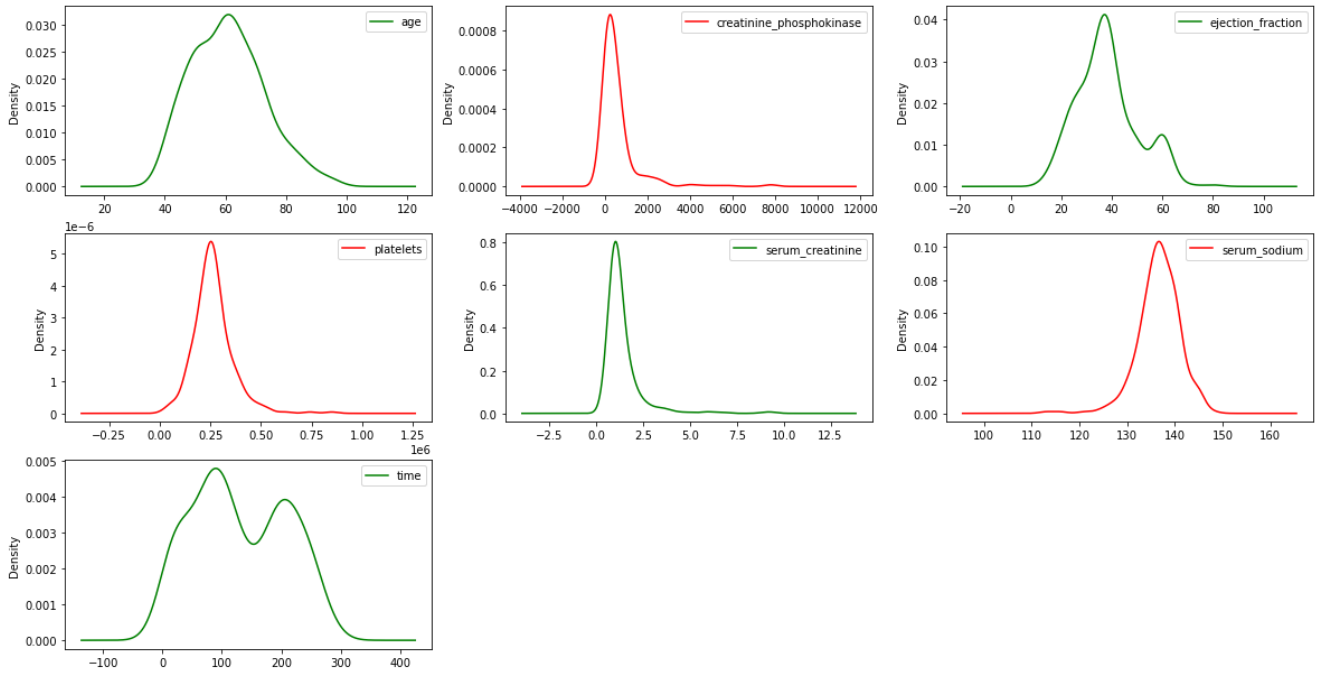


In [187]:
```
numerical_data = data[["age","creatinine_phosphokinase","ejection_fraction","platelets","serum_creatinine","serum_sodiu
```

In [188]:
```
numerical_data.plot(kind="box", subplots=True, layout=(5,3), sharex=False, figsize=(20,18))
plt.show()
```



In [189]:
```
numerical_data.plot(kind="kde", subplots=True, layout=(5,3), sharex=False, figsize=(20,18))
plt.show()
```



In [190]:
```
# converting age column from float dtype to int dtype
data['age'] = data['age'].apply(np.int64)
```

In [191]:
```
data['age'].dtype
```

Out[191]:
```
dtype('int64')
```

In [192]:
```
# checking if there are any null values in the dataset
data.isna().sum()
```

Out[192]:
```
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 [193]:
```
# get quick statistics of the data like measures of central tendency and measures of dispersion
data.describe()
```

Out[193]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	
count	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.000000	299.00000	299.000000	29
mean	60.829431	0.431438	581.839465	0.418060	38.083612	0.351171	263358.029264	1.39388	136.625418	
std	11.894997	0.496107	970.287881	0.494067	11.834841	0.478136	97804.236869	1.03451	4.412477	
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	0.50000	113.000000	
25%	51.000000	0.000000	116.500000	0.000000	30.000000	0.000000	212500.000000	0.90000	134.000000	
50%	60.000000	0.000000	250.000000	0.000000	38.000000	0.000000	262000.000000	1.10000	137.000000	
75%	70.000000	1.000000	582.000000	1.000000	45.000000	1.000000	303500.000000	1.40000	140.000000	
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	9.40000	148.000000	

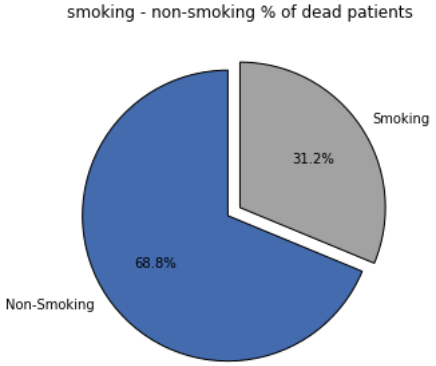
In [194]:
```
#Getting all the columns
print("Features of the dataset:")
data.columns
```

```
Features of the dataset:
```

Out[194]:
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

In [195]:
```
# make the list of categorical and numerical columns
categorical_columns = ["anaemia","diabetes","high_blood_pressure","sex","smoking","DEATH_EVENT"]
numerical_attributes = ["age","creatinine_phosphokinase","ejection_fraction","platelets","serum_creatinine","serum_sodi
```

In [196]:
```
colors = ['#446BAD','#A2A2A2']
dead_smoking_percent = data.value_counts(["DEATH_EVENT", "smoking"])
circle_dead_smoking_percent_values = [dead_smoking_percent[1][0] / sum(dead_smoking_percent[1]) * 100,dead_smoking_perc

fig1 = plt.subplots(nrows = 1,ncols = 1,figsize = (20,5))
plt.subplot(1,1,1)
plt.pie(circle_dead_smoking_percent_values,labels = ['Non-Smoking','Smoking'],autopct = '%1.1f%%',startangle = 90,explo
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})
plt.title('smoking - non-smoking % of dead patients');
plt.show()
```

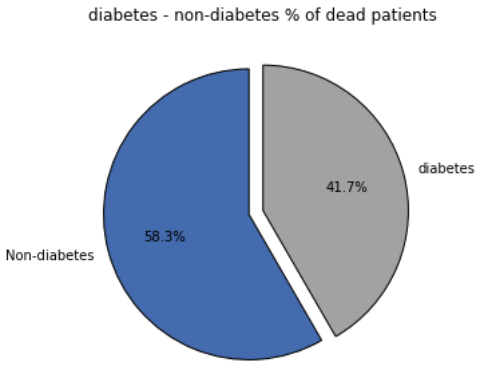


In [197]:
```
dead_hbp_percent = data.value_counts(["DEATH_EVENT", "high_blood_pressure"])
```

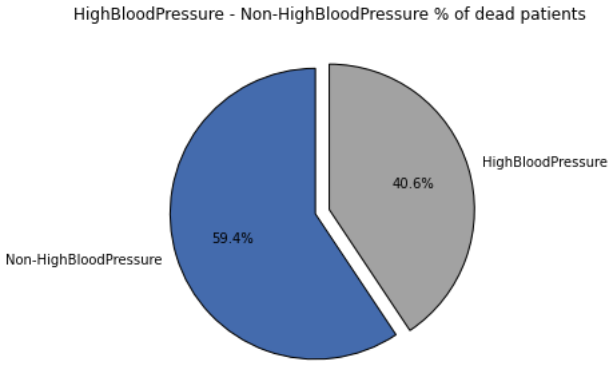
In [198]:
```
dead_hbp_percent
```

Out[198]:
```
DEATH_EVENT  high_blood_pressure
0            0                     137
             1                      66
1            0                      57
             1                      39
dtype: int64
```

In [199]:
```
colors = ['#446BAD','#A2A2A2']
dead_diabetes_percent = data.value_counts(["DEATH_EVENT", "diabetes"])
circle_dead_diabetes_percent_values = [dead_diabetes_percent[1][0] / sum(dead_diabetes_percent[1]) * 100,dead_diabetes_

fig1 = plt.subplots(nrows = 1,ncols = 1,figsize = (20,5))
plt.subplot(1,1,1)
plt.pie(circle_dead_diabetes_percent_values,labels = ['Non-diabetes','diabetes'],autopct = '%1.1f%%',startangle = 90,ex
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})
plt.title('diabetes - non-diabetes % of dead patients');
plt.show()
```



In [200]:
```
colors = ['#446BAD','#A2A2A2']
dead_hbp_percent = data.value_counts(["DEATH_EVENT", "high_blood_pressure"])
circle_dead_hbp_percent_values = [dead_hbp_percent[1][0] / sum(dead_hbp_percent[1]) * 100,dead_hbp_percent[1][1] / sum(

fig1 = plt.subplots(nrows = 1,ncols = 1,figsize = (20,5))
plt.subplot(1,1,1)
plt.pie(circle_dead_hbp_percent_values,labels = ['Non-HighBloodPressure','HighBloodPressure'],autopct = '%1.1f%%',start
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})
plt.title('HighBloodPressure - Non-HighBloodPressure % of dead patients');
plt.show()
```



In [201]:
```
# knowing if the dataset is balanced or not
data['DEATH_EVENT'].value_counts()
```

Out[201]:
```
0    203
1     96
Name: DEATH_EVENT, dtype: int64
```

In [202]:
```
data['anaemia'].value_counts()
```

Out[202]:
```
0    170
1    129
Name: anaemia, dtype: int64
```

In [203]:
```
data['diabetes'].value_counts()
```

Out[203]:
```
0    174
1    125
Name: diabetes, dtype: int64
```

In [204]:
```
data['high_blood_pressure'].value_counts()
```

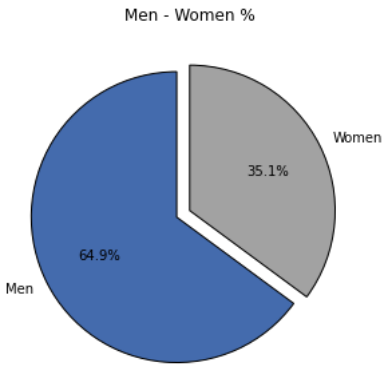
Out[204]:
```
0    194
1    105
Name: high_blood_pressure, dtype: int64
```

In [205]:
```
data['smoking'].value_counts()
```

Out[205]:
```
0    203
1     96
Name: smoking, dtype: int64
```

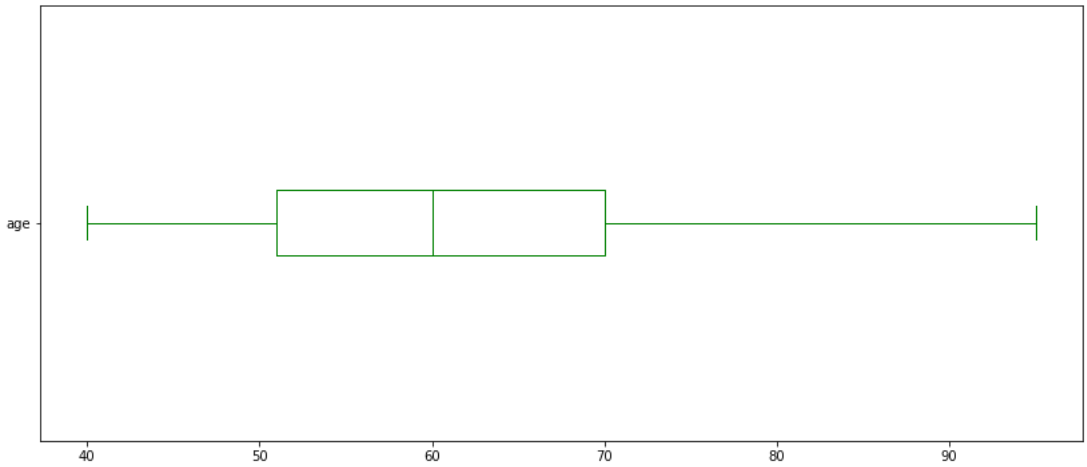
Visualizations

In [206]:
```
colors = ['#446BAD','#A2A2A2']
unique_values_sex = list(data['sex'].value_counts())
circle_sex_percentage_values = [unique_values_sex[0] / sum(unique_values_sex) * 100,unique_values_sex[1] / sum(unique_v

fig1 = plt.subplots(nrows = 1,ncols = 1,figsize = (20,5))
plt.subplot(1,1,1)
plt.pie(circle_sex_percentage_values,labels = ['Men','Women'],autopct = '%1.1f%%',startangle = 90,explode = (0.1,0),col
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})
plt.title('Men - Women %');
plt.show()
```

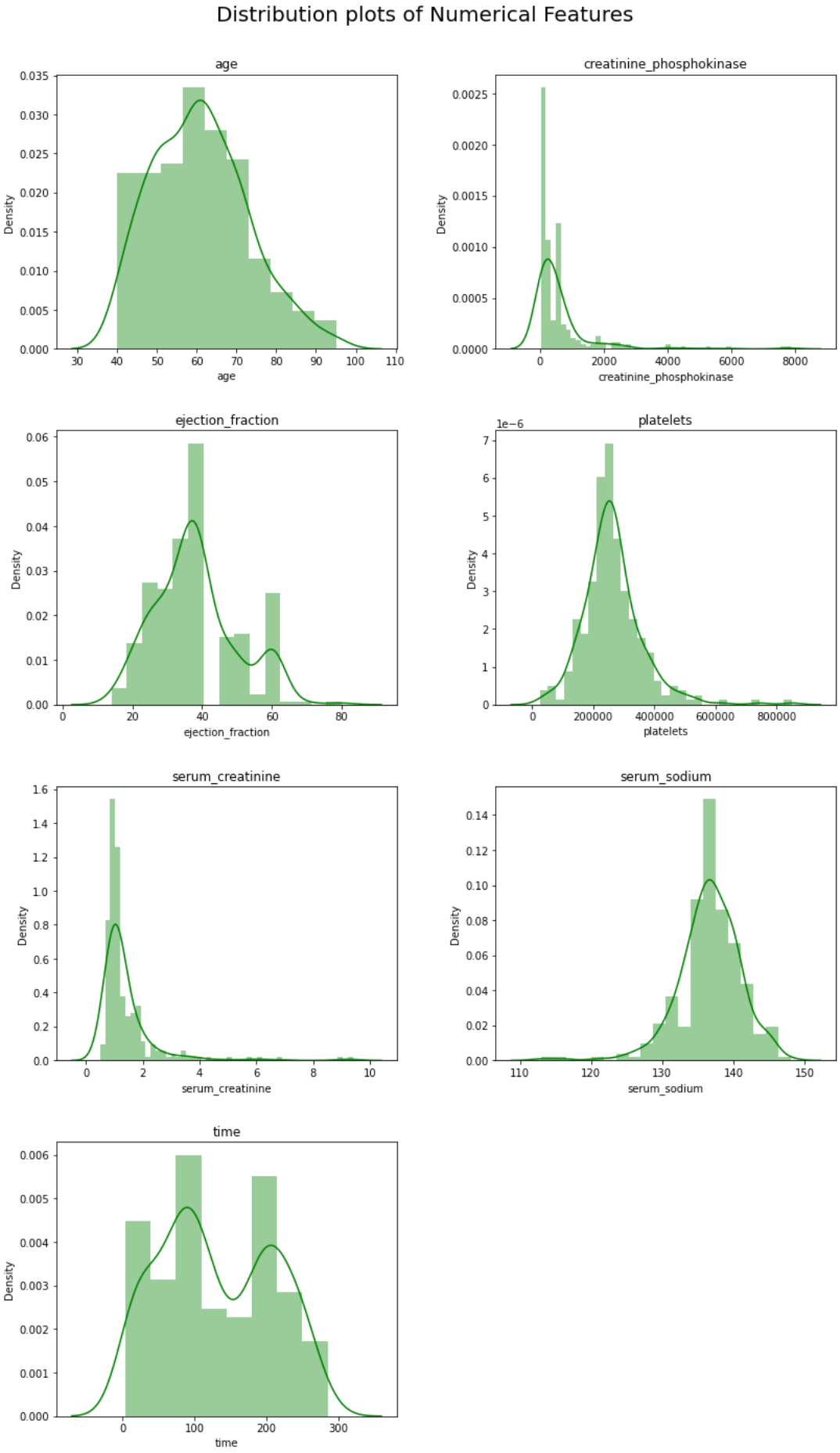


In [207]:
```
data['age'].plot(kind='box',vert=False, figsize=(14,6))
```

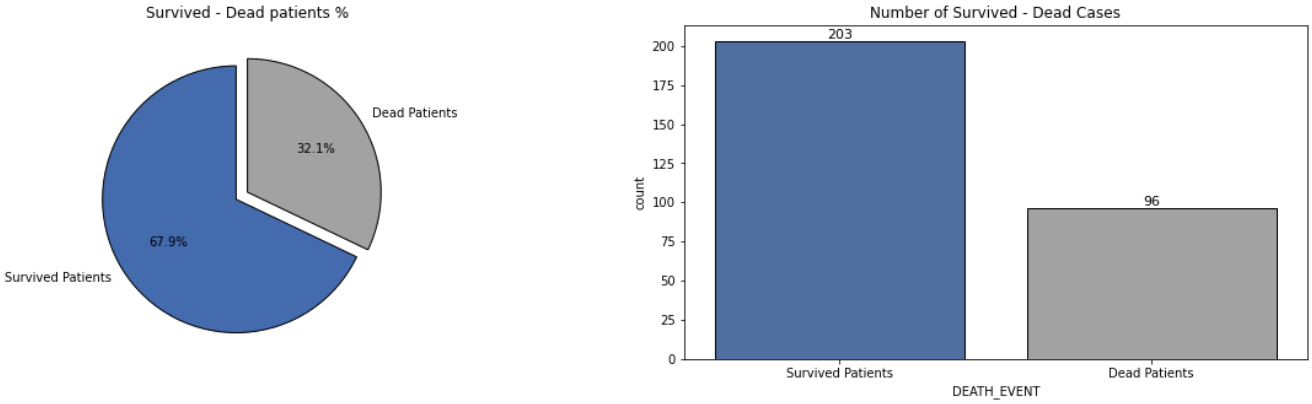
Out[207]:
```
<AxesSubplot:>
```



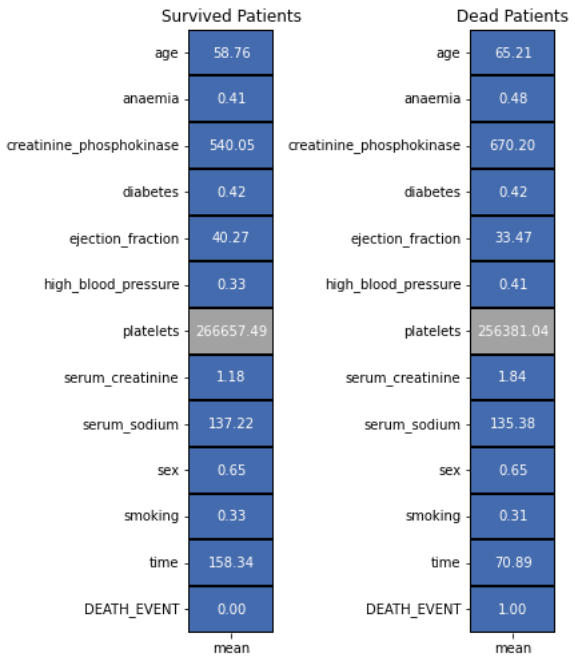
In [208]:
```
fig, ax = plt.subplots(nrows = 4,ncols = 2,figsize = (12,20))#
for i in range(len(numerical_attributes)):
    plt.subplot(4,2,i+1)
    sns.distplot(data[numerical_attributes[i]])
    title = numerical_attributes[i]
    plt.title(title)
plt.suptitle('Distribution plots of Numerical Features',size=20)
fig.tight_layout(pad=3)
ax[3,1].set_axis_off()
plt.show()
```


Distribution plots of Numerical Features
age
Density
age
creatinine_phosphokinase
Density
creatinine_phosphokinase
ejection_fraction
Density
ejection_fraction
platelets
1e−6
Density
platelets
serum_creatinine
Density
serum_creatinine
serum_sodium
Density
serum_sodium
time
Density
time

In [209]:
```
colors = ['#446BAD','#A2A2A2']
unique_values_death_event = list(data['DEATH_EVENT'].value_counts())
circle_percentage_values = [unique_values_death_event[0] / sum(unique_values_death_event) * 100,unique_values_death_eve

fig = plt.subplots(nrows = 1,ncols = 2,figsize = (20,5))
plt.subplot(1,2,1)
plt.pie(circle_percentage_values,labels = ['Survived Patients','Dead Patients'],autopct = '%1.1f%%',startangle = 90,exp
       wedgeprops = {'edgecolor' : 'black','linewidth': 1,'antialiased' : True})
plt.title('Survived - Dead patients %');

plt.subplot(1,2,2)
ax = sns.countplot('DEATH_EVENT',data = data,palette = colors,edgecolor = 'black')
for rect in ax.patches:
    ax.text(rect.get_x() + rect.get_width() / 2, rect.get_height() + 2, rect.get_height(), horizontalalignment='center'
ax.set_xticklabels(['Survived Patients','Dead Patients'])

plt.title('Number of Survived - Dead Cases');
plt.show()
```

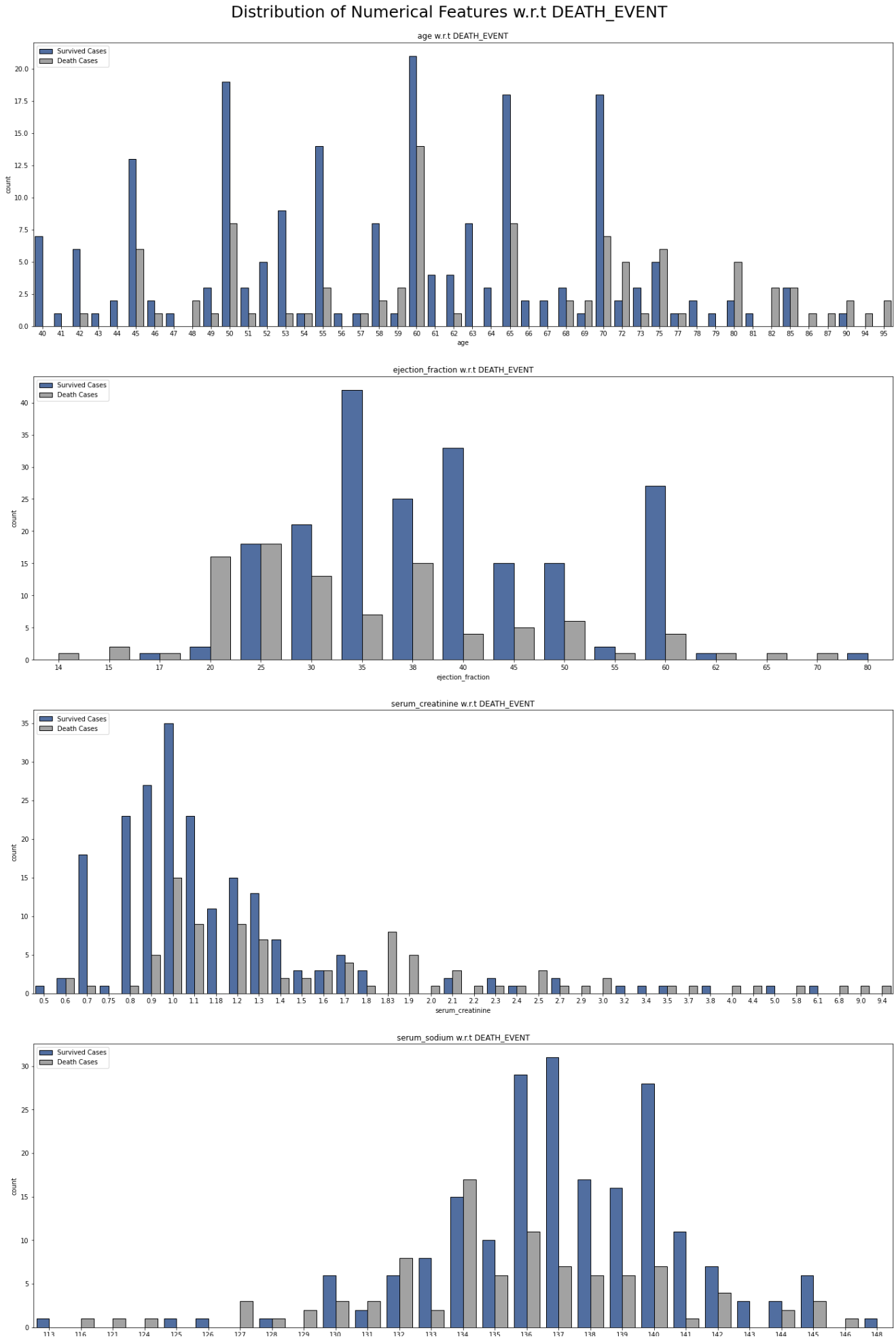


In [210]:
```
survived = data[data['DEATH_EVENT'] == 0].describe().T
dead = data[data['DEATH_EVENT'] == 1].describe().T

fig,ax = plt.subplots(nrows = 1,ncols = 3,figsize = (6,7))
plt.subplot(1,2,1)
sns.heatmap(survived[['mean']],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False,fmt = '.2f'
plt.title('Survived Patients');

plt.subplot(1,2,2)
sns.heatmap(dead[['mean']],annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black',cbar = False,fmt = '.2f',)
plt.title('Dead Patients');

fig.tight_layout(pad = 1)
```



In [211]:
```
numerical_attr1 = ["age","ejection_fraction","serum_creatinine","serum_sodium"]
numerical_attr2 = [i for i in numerical_attributes if i not in numerical_attr1]
```

In [212]:
```
fig, ax = plt.subplots(nrows = 7,ncols = 1,figsize = (20,30))
for i in range(len(numerical_attr1)):
    plt.subplot(4,1,i+1)
    sns.countplot(numerical_attr1[i],data = data,hue = "DEATH_EVENT",palette = colors,edgecolor = 'black')
    plt.legend(['Survived Cases','Death Cases'],loc = 'upper left')
    title = numerical_attr1[i] + ' w.r.t DEATH_EVENT'
    plt.title(title);
    plt.suptitle('Distribution of Numerical Features w.r.t DEATH_EVENT',size=25)
    fig.tight_layout(pad=3)
```


Distribution of Numerical Features w.r.t DEATH_EVENT
age w.r.t DEATH_EVENT
Survived Cases
Death Cases
count
age
ejection_fraction w.r.t DEATH_EVENT
Survived Cases
Death Cases
count
ejection_fraction
serum_creatinine w.r.t DEATH_EVENT
Survived Cases
Death Cases
count
serum_creatinine
serum_sodium w.r.t DEATH_EVENT
Survived Cases
Death Cases
count

serum_sodium

In [213]:
```
print(numerical_attr2)
```

```
['creatinine_phosphokinase', 'platelets', 'time']
```

In [214]:
```
print(numerical_attr1)
```

```
['age', 'ejection_fraction', 'serum_creatinine', 'serum_sodium']
```

Data Scaling using Standardization and Normalization Techniques

In [215]:
```
from sklearn.preprocessing import MinMaxScaler,StandardScaler
min_max_s = MinMaxScaler() # Normalization
standard_s = StandardScaler() # Standardization

data_cp = data.copy(deep = True)
data_cp['age'] = standard_s.fit_transform(data_cp[['age']])
data_cp['creatinine_phosphokinase'] = min_max_s.fit_transform(data_cp[['creatinine_phosphokinase']])
data_cp['ejection_fraction'] = min_max_s.fit_transform(data_cp[['ejection_fraction']])
data_cp['serum_creatinine'] = min_max_s.fit_transform(data_cp[['serum_creatinine']])
data_cp['serum_sodium'] = min_max_s.fit_transform(data_cp[['serum_sodium']])
data_cp['platelets'] = standard_s.fit_transform(data_cp[['platelets']])
data_cp['time'] = min_max_s.fit_transform(data_cp[['time']])
data_cp.head()
```

Out[215]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking
0	1.193302	0	0.071319	0	0.090909	1	1.681648e-02	0.157303	0.485714	1	0
1	-0.490896	0	1.000000	0	0.363636	0	7.535660e-09	0.067416	0.657143	1	0
2	0.351203	0	0.015693	0	0.090909	0	-1.038073e+00	0.089888	0.457143	1	1
3	-0.911945	1	0.011227	0	0.090909	0	-5.464741e-01	0.157303	0.685714	1	0
4	0.351203	1	0.017479	1	0.090909	0	6.517986e-01	0.247191	0.085714	0	0

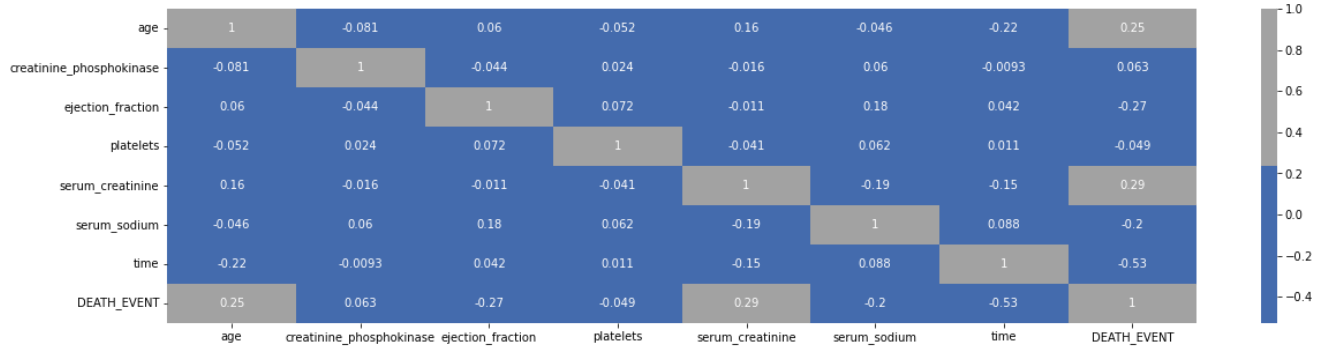
In [216]:
```
data_cp2 = data_cp[numerical_attributes+["DEATH_EVENT"]]
```

In [217]:
```
data_cp2.head()
```

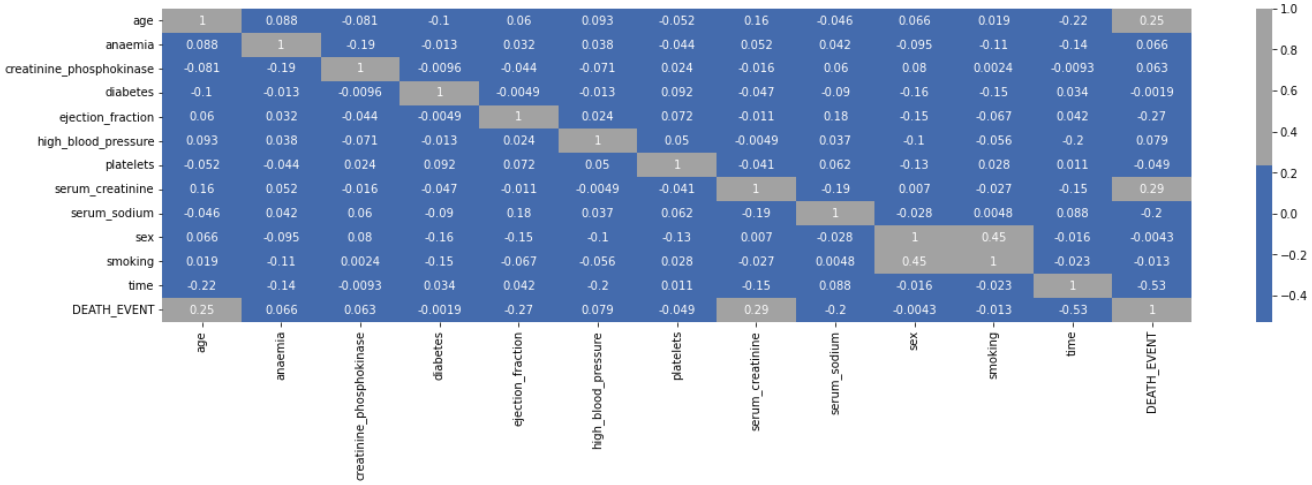
Out[217]:

	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	serum_sodium	time	DEATH_EVENT
0	1.193302	0.071319	0.090909	1.681648e-02	0.157303	0.485714	0.000000	1
1	-0.490896	1.000000	0.363636	7.535660e-09	0.067416	0.657143	0.007117	1
2	0.351203	0.015693	0.090909	-1.038073e+00	0.089888	0.457143	0.010676	1
3	-0.911945	0.011227	0.090909	-5.464741e-01	0.157303	0.685714	0.010676	1
4	0.351203	0.017479	0.090909	6.517986e-01	0.247191	0.085714	0.014235	1

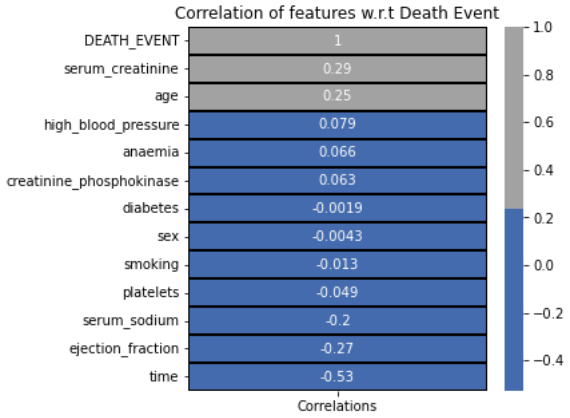
In [218]:
```
plt.figure(figsize = (20,5))
sns.heatmap(data_cp2.corr(),cmap = colors,annot = True);
```



In [219]:
```
plt.figure(figsize = (20,5))
sns.heatmap(data_cp.corr(),cmap = colors,annot = True);
```



In [220]:
```
corr = data_cp.corrwith(data_cp['DEATH_EVENT']).sort_values(ascending = False).to_frame()
corr.columns = ['Correlations']
plt.subplots(figsize = (5,5))
sns.heatmap(corr,annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black');
plt.title('Correlation of features w.r.t Death Event');
```



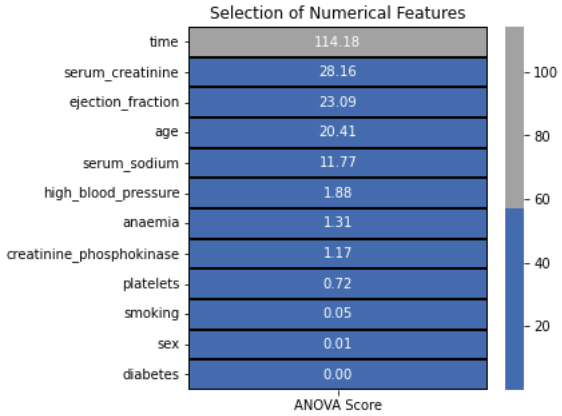
Feature Selection

In [221]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [222]:
```
all_features = ['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT']
```

In [223]:
```
predictors = [i for i in all_features if i!="DEATH_EVENT"]
target = ["DEATH_EVENT"]
```

In [224]:
```
features = data_cp.loc[:,predictors]
target = data_cp.loc[:,target]

best_features = SelectKBest(score_func = f_classif,k = 'all')
fit = best_features.fit(features,target)

featureScores = pd.DataFrame(data = fit.scores_,index = list(features.columns),columns = ['ANOVA Score'])

plt.subplots(figsize = (5,5))
sns.heatmap(featureScores.sort_values(ascending = False,by = 'ANOVA Score'),annot = True,cmap = colors,linewidths = 0.4
plt.title('Selection of Numerical Features');
```



In [225]:
```
data_cp.drop(columns = ['diabetes','sex','smoking','platelets'],inplace = True)
data_cp.head()
```

Out[225]:

	age	anaemia	creatinine_phosphokinase	ejection_fraction	high_blood_pressure	serum_creatinine	serum_sodium	time	DEATH_EVENT
0	1.193302	0	0.071319	0.090909	1	0.157303	0.485714	0.000000	1
1	-0.490896	0	1.000000	0.363636	0	0.067416	0.657143	0.007117	1
2	0.351203	0	0.015693	0.090909	0	0.089888	0.457143	0.010676	1
3	-0.911945	1	0.011227	0.090909	0	0.157303	0.685714	0.010676	1
4	0.351203	1	0.017479	0.090909	0	0.247191	0.085714	0.014235	1

In [226]:
```
data_cp.columns
```

Out[226]:
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'ejection_fraction',
       'high_blood_pressure', 'serum_creatinine', 'serum_sodium', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

In [227]:
```
data_cp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64  
 2   creatinine_phosphokinase  299 non-null    float64
 3   ejection_fraction         299 non-null    float64
 4   high_blood_pressure       299 non-null    int64  
 5   serum_creatinine          299 non-null    float64
 6   serum_sodium              299 non-null    float64
 7   time                      299 non-null    float64
 8   DEATH_EVENT               299 non-null    int64  
dtypes: float64(6), int64(3)
memory usage: 21.1 KB
```

Balancing the Dataset using SMOTE

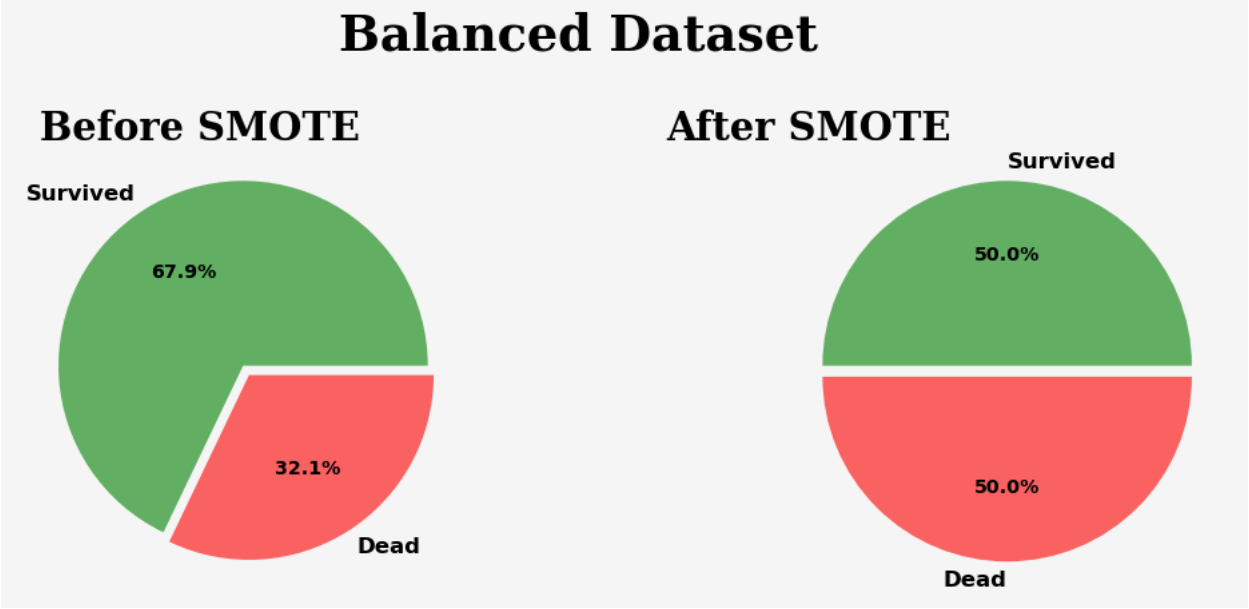
In [228]:
```
X = data_cp.iloc[:, 0:8].values
y = data_cp.DEATH_EVENT.values
```

In [229]:
```
import imblearn
from collections import Counter
from imblearn.over_sampling import SMOTE
```

In [230]:
```
sm = SMOTE()
X_bal, y_bal = sm.fit_resample(X, y)
Counter(y_bal)
```

Out[230]:
```
Counter({1: 203, 0: 203})
```

In [231]:
```
# plots target data

fig = plt.figure(figsize = (24,8), dpi = 85)
gs = GridSpec(ncols=13, nrows=5, left=0.05, right=0.5, wspace=0.2, hspace=0.1)
fig.patch.set_facecolor('#f5f5f5')
sns.set_palette(sns.color_palette(['green','red']))

ax1 = fig.add_subplot(gs[:, 0:5])
ax2 = fig.add_subplot(gs[:, 8:])


# axes list
axes = [ax1,ax2]

# setting of axes; visibility of axes and spines turn off
for ax in axes:
    ax.axes.get_yaxis().set_visible(False)
    ax.set_facecolor('#f5f5f5')

    for loc in ['left', 'right', 'top', 'bottom']:
        ax.spines[loc].set_visible(False)

#-------------------------------------------------------------
#ax1
pre_smote_count = [len(y[y==0]), len(y[y==1])]
ax1_plot = ax1.pie(pre_smote_count,
                   labels=['Survived', 'Dead'],
                    autopct='%1.1f%%', explode=[0,0.06],
                   colors=['green','red'])

for piece in ax1_plot[0]:
    piece.set_alpha(0.6)

for i, text in enumerate(ax1_plot[1]):
    text.set_weight('bold')
    text.set_size(14)

for i, text in enumerate(ax1_plot[2]):
    text.set_weight('bold')
    text.set_size(12)


fig.text(0.06, 0.75, 'Before SMOTE',  {'font':'Serif', 'weight':'bold','color': 'black', 'size':24.5})

#-------------------------------------------------------------
#ax2
post_smote_count = [len(y_bal[y_bal==0]), len(y_bal[y_bal==1])]
ax2_plot = ax2.pie(post_smote_count,
                   labels=['Survived', 'Dead'],
                    autopct='%1.1f%%', explode=[0,0.06],
                   colors=['green','red'])

for piece in ax2_plot[0]:
    piece.set_alpha(0.6)

for i, text in enumerate(ax2_plot[1]):
    text.set_weight('bold')
    text.set_size(14)

for i, text in enumerate(ax2_plot[2]):
    text.set_weight('bold')
    text.set_size(12)

fig.text(0.29, 0.75, 'After SMOTE',  {'font':'Serif', 'weight':'bold','color': 'black', 'size':24.5})
#---------------------------------
fig.text(0.17, 0.85, 'Balanced Dataset',
         {'font':'Serif', 'weight':'bold','color': 'black', 'size':31})
plt.show()
```



Applying Machine Learning Algorithms for Training and Testing

In [261]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import precision_recall_curve
```

In [262]:
```
x_train1, x_test1, y_train1, y_test1 = train_test_split(X_bal, y_bal, test_size = 0.25, random_state = 2)
```

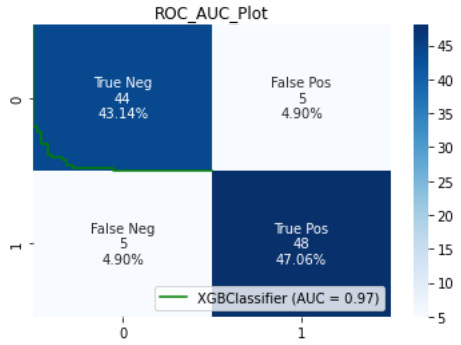
In [263]:
```
def model(classifier,x_train,y_train,x_test,y_test):
    classifier.fit(x_train,y_train)
    prediction = classifier.predict(x_test)
    cv = RepeatedStratifiedKFold(n_splits = 10,n_repeats = 3,random_state = 1)
    classifier_name = str(classifier).split('(')[0]
    print(f"{classifier_name} Results:")
    print("Cross Validation Score : ",'{0:.2%}'.format(cross_val_score(classifier,x_train,y_train,cv = cv,scoring = 'ro
    print("ROC_AUC Score : ",'{0:.2%}'.format(roc_auc_score(y_test,prediction)))
    plot_roc_curve(classifier, x_test,y_test)
    plt.title('ROC_AUC_Plot')
    #plt.show()
    return classifier

def model_evaluation(classifier,x_test,y_test):
    # Confusion Matrix
    cm = confusion_matrix(y_test,classifier.predict(x_test))
    names = ['True Neg','False Pos','False Neg','True Pos']
    counts = [value for value in cm.flatten()]
    percentages = ['{0:.2%}'.format(value) for value in cm.flatten()/np.sum(cm)]
    labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(names,counts,percentages)]
    labels = np.asarray(labels).reshape(2,2)
    sns.heatmap(cm,annot = labels,cmap = 'Blues',fmt ='')

    # Classification Report
    print(classification_report(y_test,classifier.predict(x_test)))
    return classification_report(y_test,classifier.predict(x_test))
```

Xgboost Classifier

In [264]:
```
from xgboost import XGBClassifier

classifier_xgb = XGBClassifier(learning_rate= 0.01,max_depth = 3,n_estimators = 1000)
```

In [265]:
```
model(classifier_xgb,x_train1,y_train1,x_test1,y_test1)
Xg_result = model_evaluation(classifier_xgb,x_test1,y_test1)
```

```
XGBClassifier Results:
Cross Validation Score :  93.00%
ROC_AUC Score :  90.18%
              precision    recall  f1-score   support

           0       0.90      0.90      0.90        49
           1       0.91      0.91      0.91        53

    accuracy                           0.90       102
   macro avg       0.90      0.90      0.90       102
weighted avg       0.90      0.90      0.90       102
```

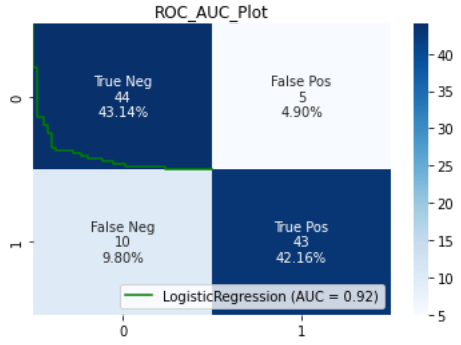


Logistic Regression

In [266]:
```
from sklearn.linear_model import LogisticRegression
Logistic_reg = LogisticRegression()

model(Logistic_reg,x_train1,y_train1,x_test1,y_test1)
model_evaluation(Logistic_reg,x_test1,y_test1)
```

```
LogisticRegression Results:
Cross Validation Score :  89.10%
ROC_AUC Score :  85.46%
              precision    recall  f1-score   support

           0       0.81      0.90      0.85        49
           1       0.90      0.81      0.85        53

    accuracy                           0.85       102
   macro avg       0.86      0.85      0.85       102
weighted avg       0.86      0.85      0.85       102
```

Out[266]:
```
'              precision    recall  f1-score   support\n\n           0       0.81      0.90      0.85        49\n           1       0.90      0.81      0.85        53\n\n    accuracy                           0.85       102\n   macro avg       0.86      0.85      0.85       102\nweighted avg       0.86      0.85      0.85       102\n'
```

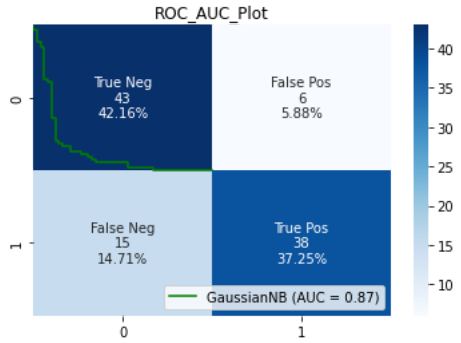


Gaussian Naive Bayes Classifier

In [267]:
```
from sklearn.naive_bayes import GaussianNB
gaussian_cls = GaussianNB()

model(gaussian_cls,x_train1,y_train1,x_test1,y_test1)
model_evaluation(gaussian_cls,x_test1,y_test1)
```

```
GaussianNB Results:
Cross Validation Score :  87.08%
ROC_AUC Score :  79.73%
              precision    recall  f1-score   support

           0       0.74      0.88      0.80        49
           1       0.86      0.72      0.78        53

    accuracy                           0.79       102
   macro avg       0.80      0.80      0.79       102
weighted avg       0.80      0.79      0.79       102
```

Out[267]:
```
'              precision    recall  f1-score   support\n\n           0       0.74      0.88      0.80        49\n
1       0.86      0.72      0.78        53\n\n    accuracy                           0.79       102\n   macro avg
0.80      0.80      0.79       102\nweighted avg       0.80      0.79      0.79       102\n'
```

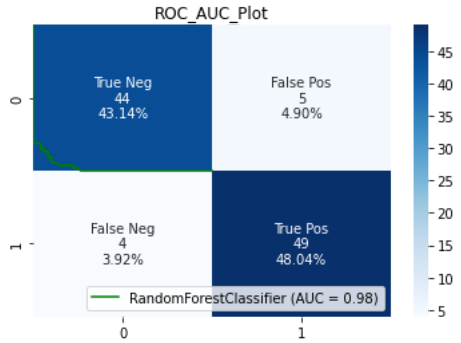


Random Forest Classifier

In [268]:
```
from sklearn.ensemble import RandomForestClassifier
rf_cls = RandomForestClassifier()

model(rf_cls,x_train1,y_train1,x_test1,y_test1)
model_evaluation(rf_cls,x_test1,y_test1)
```

```
RandomForestClassifier Results:
Cross Validation Score :  93.72%
ROC_AUC Score :  91.12%
              precision    recall  f1-score   support

           0       0.92      0.90      0.91        49
           1       0.91      0.92      0.92        53

    accuracy                           0.91       102
   macro avg       0.91      0.91      0.91       102
weighted avg       0.91      0.91      0.91       102
```

Out[268]:
```
'              precision    recall  f1-score   support\n\n           0       0.92      0.90      0.91        49\n
1       0.91      0.92      0.92        53\n\n    accuracy                           0.91       102\n   macro avg
0.91      0.91      0.91       102\nweighted avg       0.91      0.91      0.91       102\n'
```

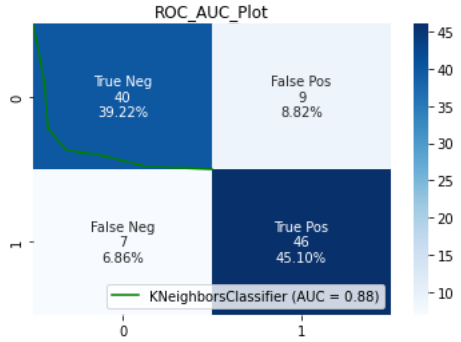


KNN Classifier

In [269]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_cls = KNeighborsClassifier()

model(knn_cls,x_train1,y_train1,x_test1,y_test1)
model_evaluation(knn_cls,x_test1,y_test1)
```

```
KNeighborsClassifier Results:
Cross Validation Score :  86.18%
ROC_AUC Score :  84.21%
              precision    recall  f1-score   support

           0       0.85      0.82      0.83        49
           1       0.84      0.87      0.85        53

    accuracy                           0.84       102
   macro avg       0.84      0.84      0.84       102
weighted avg       0.84      0.84      0.84       102
```

Out[269]:
```
'              precision    recall  f1-score   support\n\n           0       0.85      0.82      0.83        49\n
1       0.84      0.87      0.85        53\n\n    accuracy                           0.84       102\n   macro avg
0.84      0.84      0.84       102\nweighted avg       0.84      0.84      0.84       102\n'
```

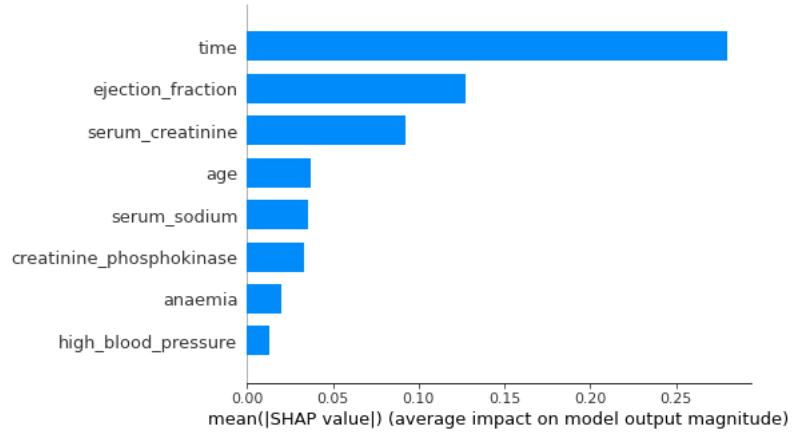


In [270]:
```
#shap
import shap
```

In [271]:
```
feature_names = ['age', 'anaemia', 'creatinine_phosphokinase', 'ejection_fraction',
       'high_blood_pressure', 'serum_creatinine', 'serum_sodium', 'time']
explainer = shap.Explainer(model.predict, x_train1, feature_names=feature_names)
shap_values = explainer(x_test1)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [271], in <cell line: 3>()
      1 feature_names = ['age', 'anaemia', 'creatinine_phosphokinase', 'ejection_fraction',
      2        'high_blood_pressure', 'serum_creatinine', 'serum_sodium', 'time']
----> 3 explainer = shap.Explainer(model.predict, x_train1, feature_names=feature_names)
      4 shap_values = explainer(x_test1)

AttributeError: 'function' object has no attribute 'predict'
```

In [179]:
```
#plots importance of each feature
shap.summary_plot(shap_values, x_test1, plot_type="bar")
```



In [ ]: