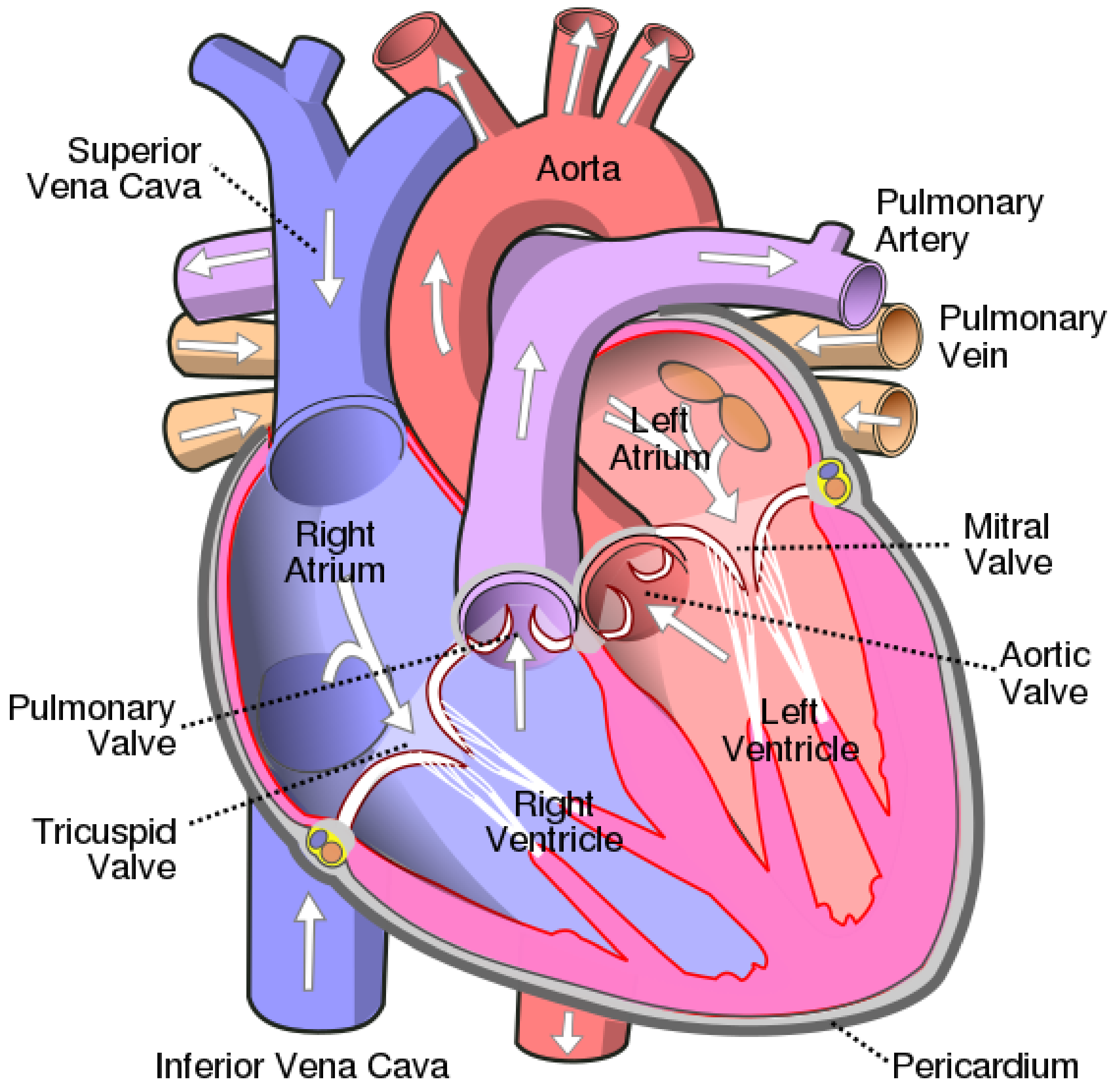
Heart Failure Prediction



What is Heart Failure?

Heart Failure is a condition when the heart muscle does not pump blood as well as it should to meet the body's demands. Blood is the most important fluid that circulates throughout the body by supplying oxygen to all the parts of the body

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 worldwide. Four out of 5CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease

> *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, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help*

This dataset contains person's information like age, sex , blood pressure, smoke, diabetes,ejection fraction

Library

In [379]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
colors = ['#97C1A9','#FFFFFF']

import imblearn
from collections import Counter
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import MinMaxScaler,StandardScaler
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
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 RocCurveDisplay
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

import warnings
warnings.filterwarnings("ignore")
```

Dataset

In [380]:

```
data=pd.read_csv('/content/drive/MyDrive/projek/heart_failure_clinical_records_dataset.cs
```

In [381]:

```
data.head()
```

Out[381]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
0	75.0	0	582	0	20	1
1	55.0	0	7861	0	38	0
2	65.0	0	146	0	20	0
3	50.0	1	111	0	20	0
4	65.0	1	160	1	20	0

Dataset Attributes

- Age : age [years]
- anaemia : Decrease of red blood cells or hemoglobin (boolean)
- creatinine_phosphokinase : Level of the CPK enzyme in the blood (mcg/L)
- diabetes : If the patient has diabetes (boolean)
- ejection_fraction : Percentage of blood leaving the heart at each contraction (percentage)

- high_blood_pressure : If the patient has hypertension (boolean)
- 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)
- sex : Woman or man (binary)
- smoking : If the patient smokes or not (boolean)
- time : Follow-up period (days)
- DEATH_EVENT : If the patient deceased during the follow-up period (boolean)

Data Info

In [382]:

```
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 [383]:

```
data.shape
```

Out[383]:

```
(299, 13)
```

In [384]:

```
data.columns
```

Out[384]:

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

In [385]:

```
data.describe().T
```

Out[385]:

	count	mean	std	min	25%	50%
age	299.0	60.833893	11.894809	40.0	51.0	60.0
anaemia	299.0	0.431438	0.496107	0.0	0.0	0.0
creatinine_phosphokinase	299.0	581.839465	970.287881	23.0	116.5	250.0
diabetes	299.0	0.418060	0.494067	0.0	0.0	0.0
ejection_fraction	299.0	38.083612	11.834841	14.0	30.0	38.0
high_blood_pressure	299.0	0.351171	0.478136	0.0	0.0	0.0
platelets	299.0	263358.029264	97804.236869	25100.0	212500.0	262000.0
serum_creatinine	299.0	1.393880	1.034510	0.5	0.9	1.1
serum_sodium	299.0	136.625418	4.412477	113.0	134.0	137.0
sex	299.0	0.648829	0.478136	0.0	0.0	1.0
smoking	299.0	0.321070	0.467670	0.0	0.0	0.0
time	299.0	130.260870	77.614208	4.0	73.0	115.0
DEATH_EVENT	299.0	0.321070	0.467670	0.0	0.0	0.0

In [386]:

```
data.isnull().mean()*100
```

Out[386]:

```
age                         0.0
anaemia                     0.0
creatinine_phosphokinase    0.0
diabetes                    0.0
ejection_fraction           0.0
high_blood_pressure         0.0
platelets                   0.0
serum_creatinine            0.0
serum_sodium                0.0
sex                         0.0
smoking                     0.0
time                        0.0
DEATH_EVENT                 0.0
dtype: float64
```

EDA

In [387]:

```
data['age'] = data['age'].astype(int)
data['platelets'] = data['platelets'].astype(int)
df = data.copy(deep = True)
```

In [388]:

```
df.loc[df['DEATH_EVENT']==0,'Status']='Survived'
df.loc[df['DEATH_EVENT']==1,'Status']='Not Survived'
```

In [389]:

```
col = list(data.columns)
categorical_features = []
numerical_features = []
for i in col:
    if len(data[i].unique()) > 6:
        numerical_features.append(i)
    else:
        categorical_features.append(i)

print('Categorical Features :',*categorical_features)
print('Numerical Features :',*numerical_features)
```

```
Categorical Features : anaemia diabetes high_blood_pressure sex smoking DE
ATH_EVENT
Numerical Features : age creatinine_phosphokinase ejection_fraction platel
ets serum_creatinine serum_sodium time
```

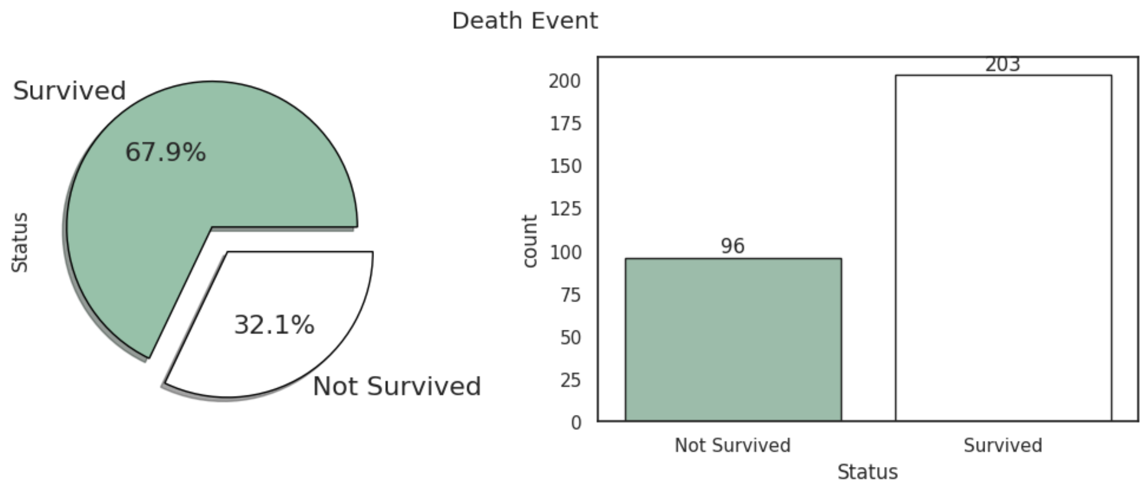
Target - Death Event

In [390]:

```
    sns.set(style='white')

    fig = plt.subplots(1,2,figsize = (13,4))
    plt.subplot(1,2,1)
    df['Status'].value_counts().plot.pie(explode=[0.1,0.1], autopct='%1.1f%%', shadow=Tru

    plt.subplot(1,2,2)
    ax=sns.countplot(data=df, x='Status',palette = colors,edgecolor = 'k')
    ax.bar_label(ax.containers[0])

    plt.suptitle('Death Event')
```

Out[390]:

```
Text(0.5, 0.98, 'Death Event')
```



- The dataset has very low data points (299)
- The dataset is unbalanced with 2:1 ratio for No Death Event cases : Death Event cases
- Visualizations and Predictions will be biased towards No Death Event cases.

Categorical Features

In [391]:

```
# Categorical Plot
def catplot(df,x):
    sns.set(style='white')
    fig = plt.subplots(1,3,figsize = (15,4))
    plt.subplot(1,3,1)
    df[x].value_counts().plot.pie(explode=[0.1,0.1], autopct='%1.1f%%', shadow=True, text

    plt.subplot(1,3,2)
    ax=sns.histplot(data=df,x=x,kde = True,color=colors[0],edgecolor = 'k')
    ax.bar_label(ax.containers[0])
    # ax.set_xlim(-1,2)
    # ax.set_xticks(range(-1,2))

    plt.subplot(1,3,3)
    ax=sns.countplot(data=df, x=x, hue='Status',palette = colors,edgecolor = 'k')
    for container in ax.containers:
        ax.bar_label(container)
    tit = x + ' vs Death Event'
    plt.suptitle(tit)
```

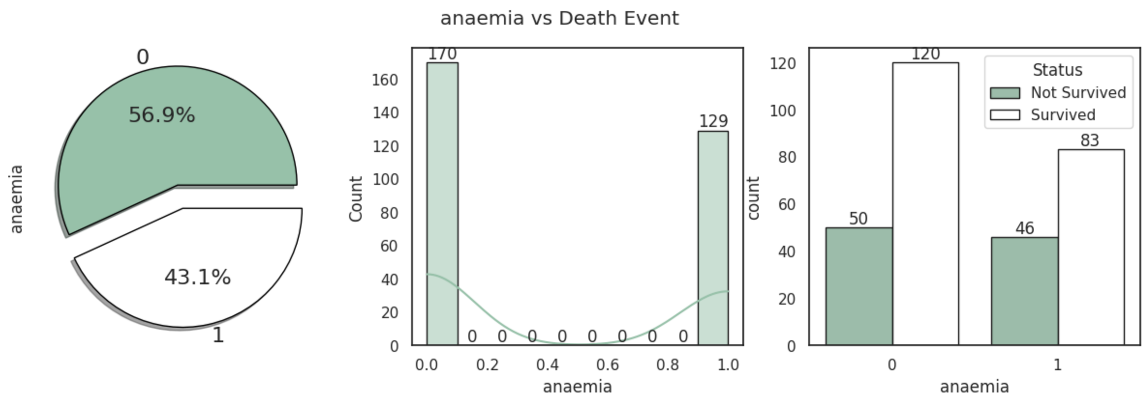
In [392]:

```
# for i in range(len(categorical_features)):
#     catplot(df,categorical_features[i])
```

Anemia

In [393]:

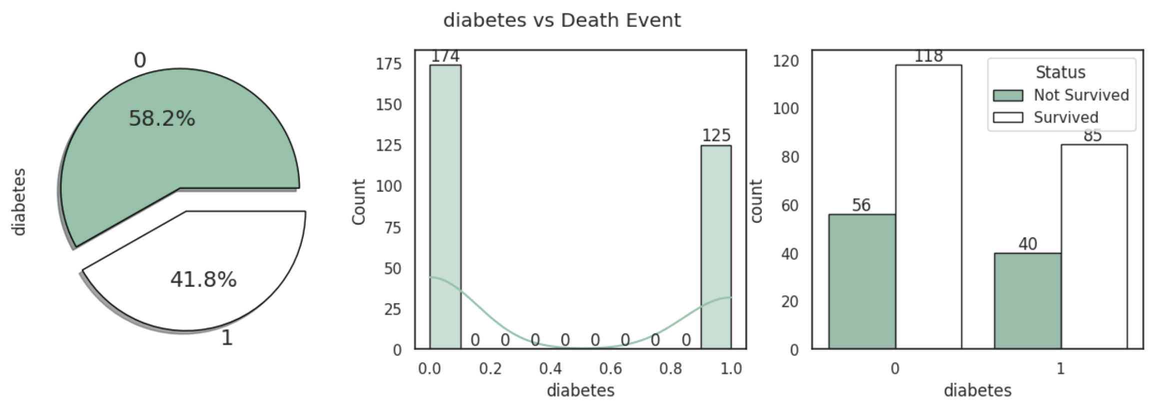
```
catplot(df,'anaemia')
```



Diabetes

In [394]:

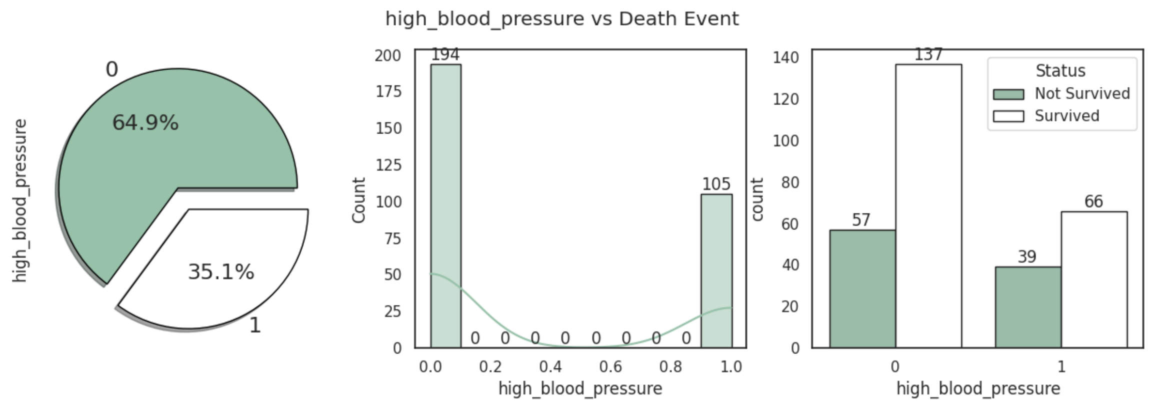
```
catplot(df,'diabetes')
```



high_blood_pressure

In [395]:

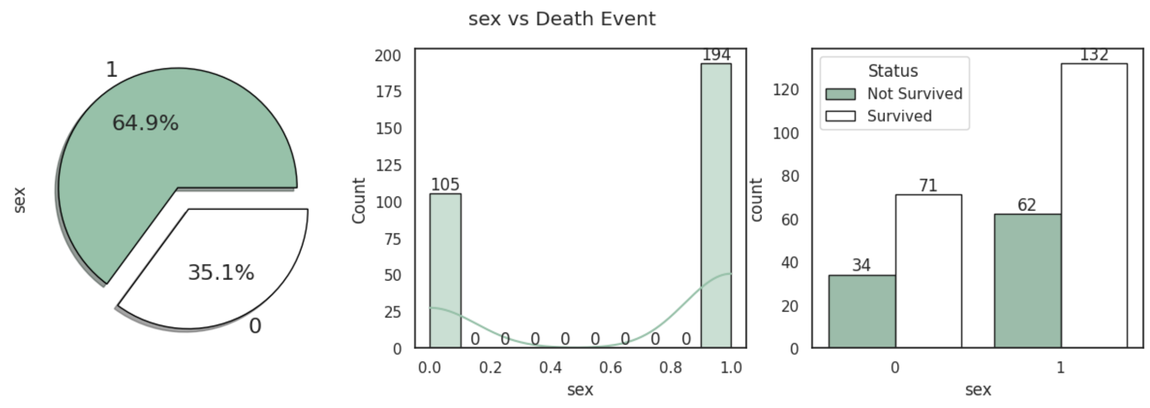
```
catplot(df,'high_blood_pressure')
```



sex

In [396]:

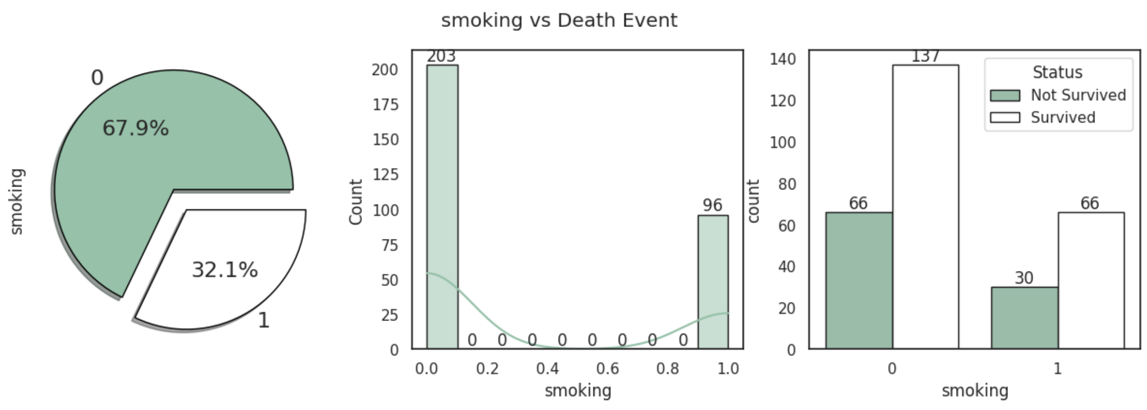
```
catplot(df,'sex')
```



smoking

In [397]:

```
catplot(df,'smoking')
```



Sumary

Categorical Features Insight :

- All the graphs have the same pattern
- There are more cases of male population

Categorical Features Sumary :

- anaemia : Anaemia = No Anaemia
- diabetes : Diabetes = No Diabetes
- high_blood_pressure : High Blood Pressure > No High Blood Pressure (Needs more data)
- sex : Male > Female
- smoking : No Smoking > Smoking

Genaral Information

- anaemia : High chances of heart failures due to anaemia.
- diabetes : High chances of heart failures due to diabetes.
- high blood pressure : High chances with heart failures due to high blood pressure.
- sex : male > female but by small margin are prone to more heart failures.
- smoking : Smoking increases the chances of suffering from heart failures.

Numerical Features

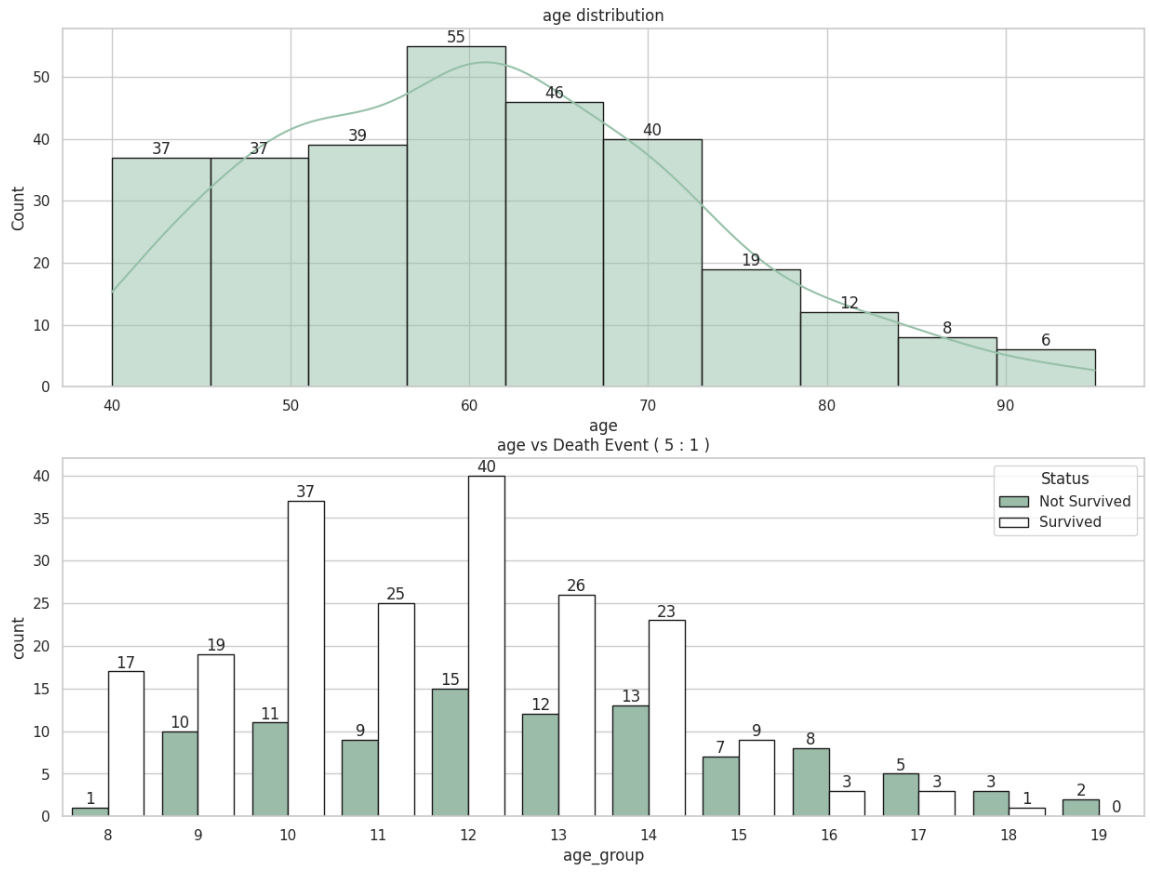
In [398]:

```
# Numerical Plot
def numplot(df,x,scale):
    sns.set(style='whitegrid')
    fig = plt.subplots(2,1,figsize = (15,11))

    plt.subplot(2,1,1)
    ax=sns.histplot(data=df, x=x, kde=True,color=colors[0],edgecolor = 'k')
    ax.bar_label(ax.containers[0])
    tit=x + ' distribution'
    plt.title(tit)

    plt.subplot(2,1,2)
    tar=x + '_group'
    Tstr= str(scale)
    tit2=x + ' vs Death Event ( ' + Tstr + ' : 1 )'
    df[tar] = [ int(i / scale) for i in df[x]]
    ax=sns.countplot(data=df, x=tar, hue='Status',palette = colors,edgecolor = 'k')
    for container in ax.containers:
        ax.bar_label(container)
    plt.title(tit2)
```

age

In [399]:

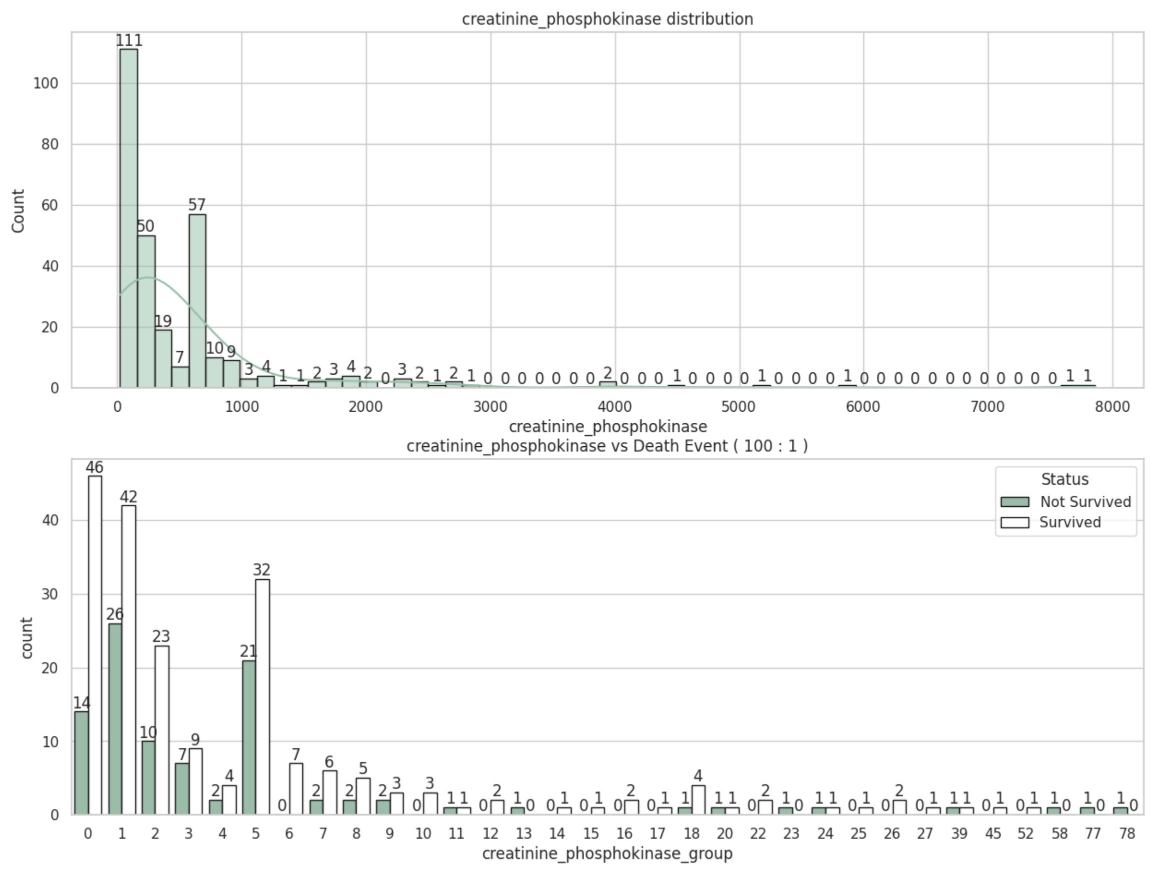
```
numplot(df,'age',5)
```



Creatinine Phosphokinase

In [400]:

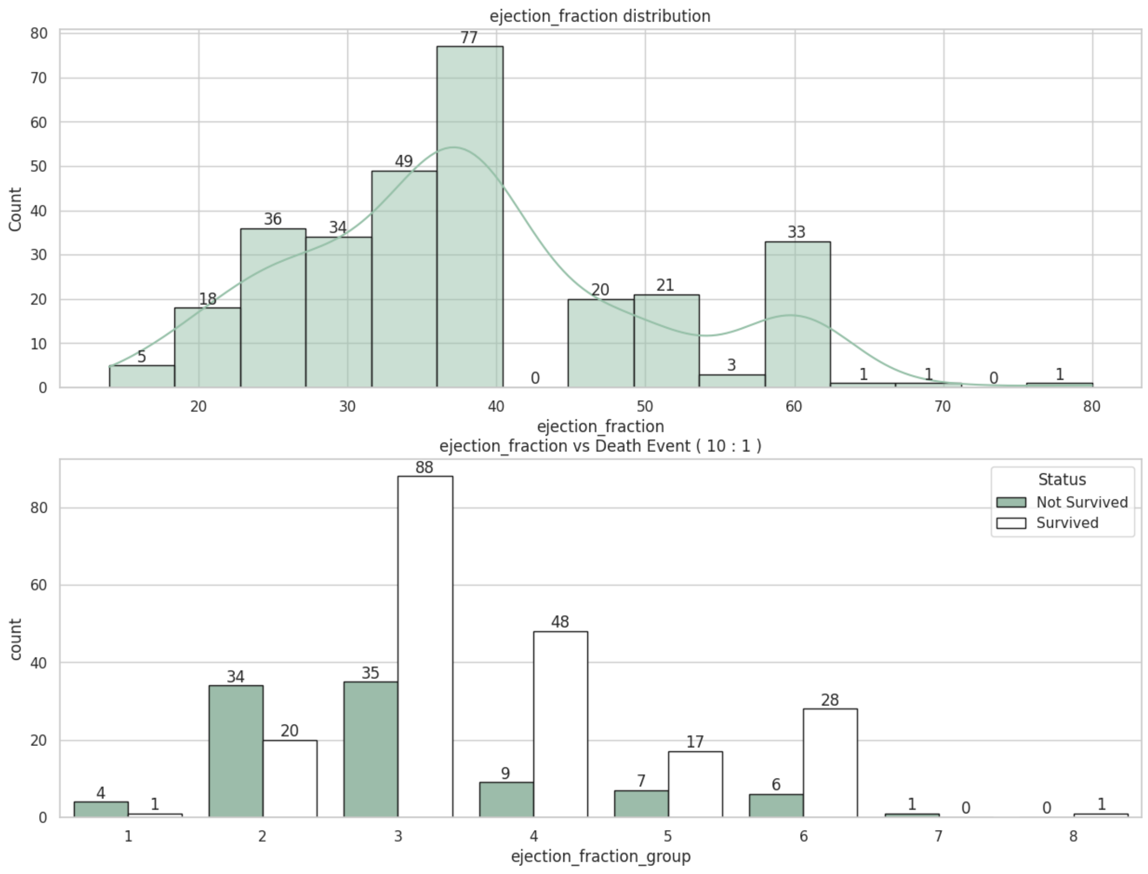
```
numplot(df,'creatinine_phosphokinase',100)
```



ejection_fraction

In [401]:

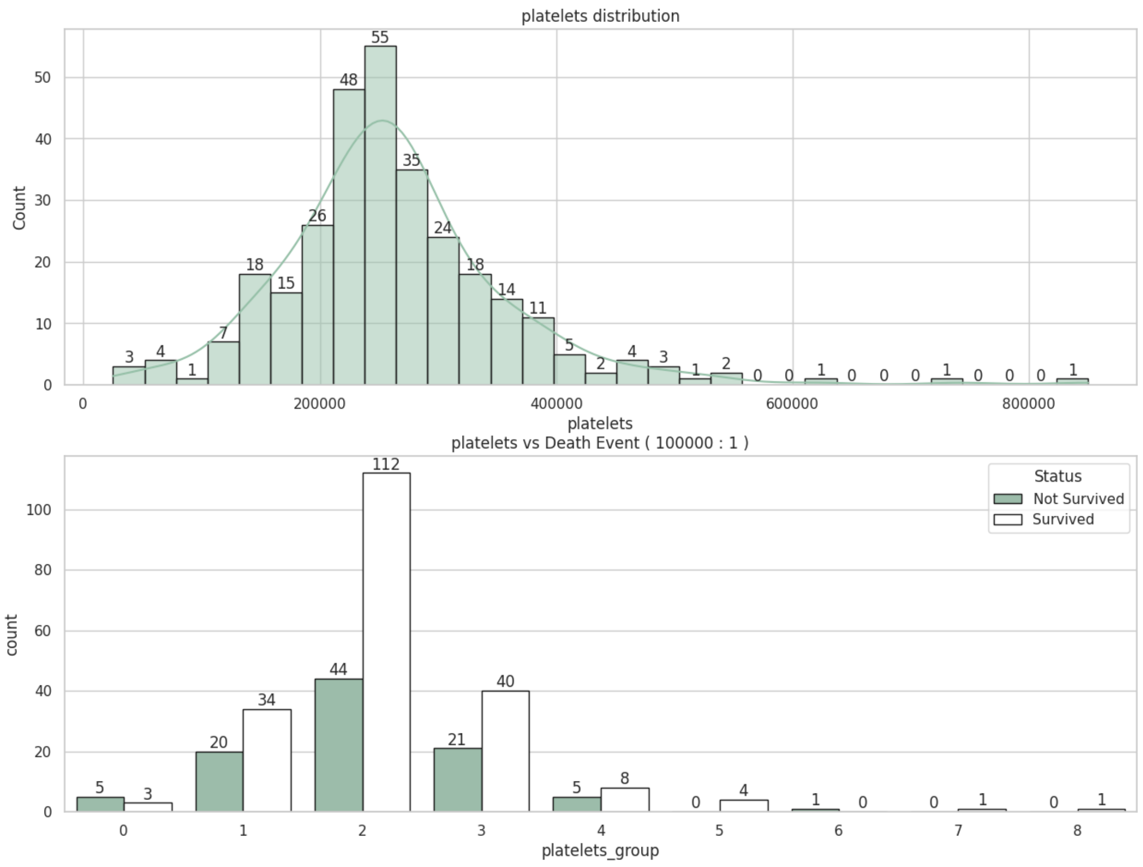
```
numplot(df,'ejection_fraction',10)
```



platelets

In [402]:

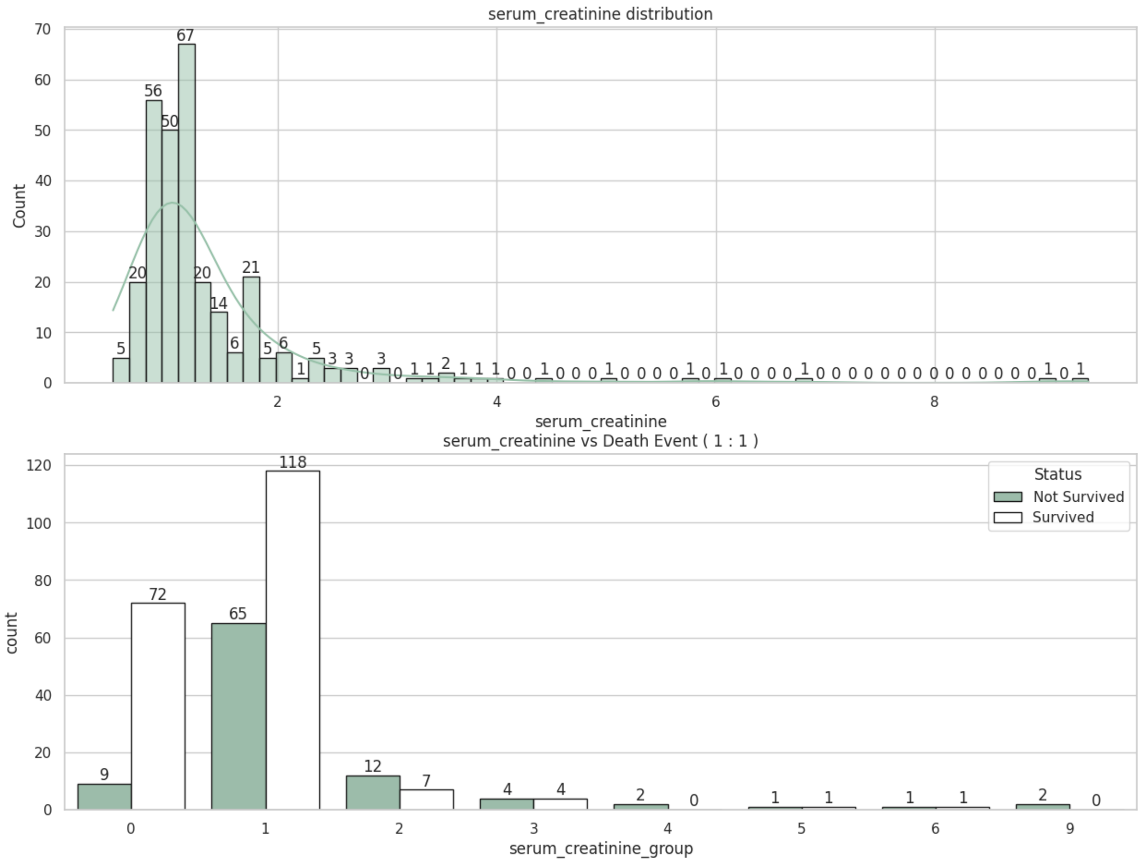
```
numplot(df,'platelets',10**5)
```



serum_creatinine

In [403]:

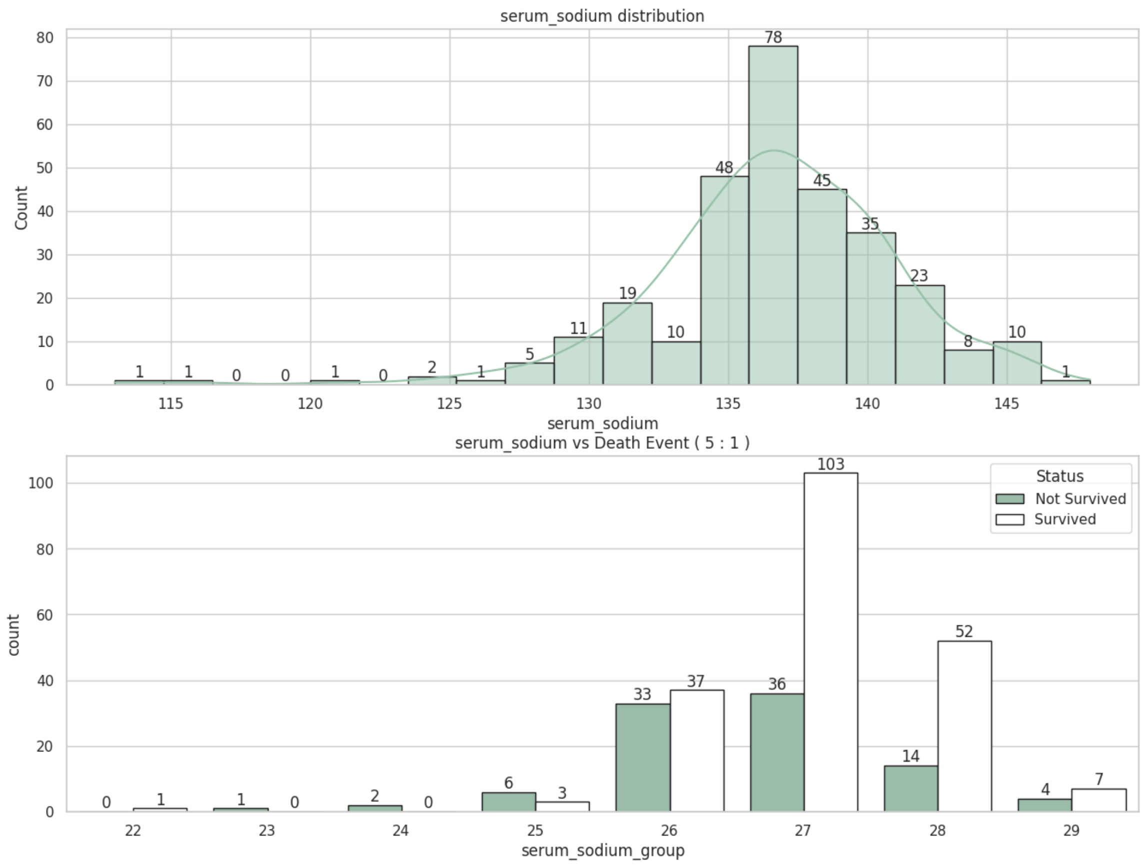
```
numplot(df,'serum_creatinine',1)
```



serum_sodium

In [404]:

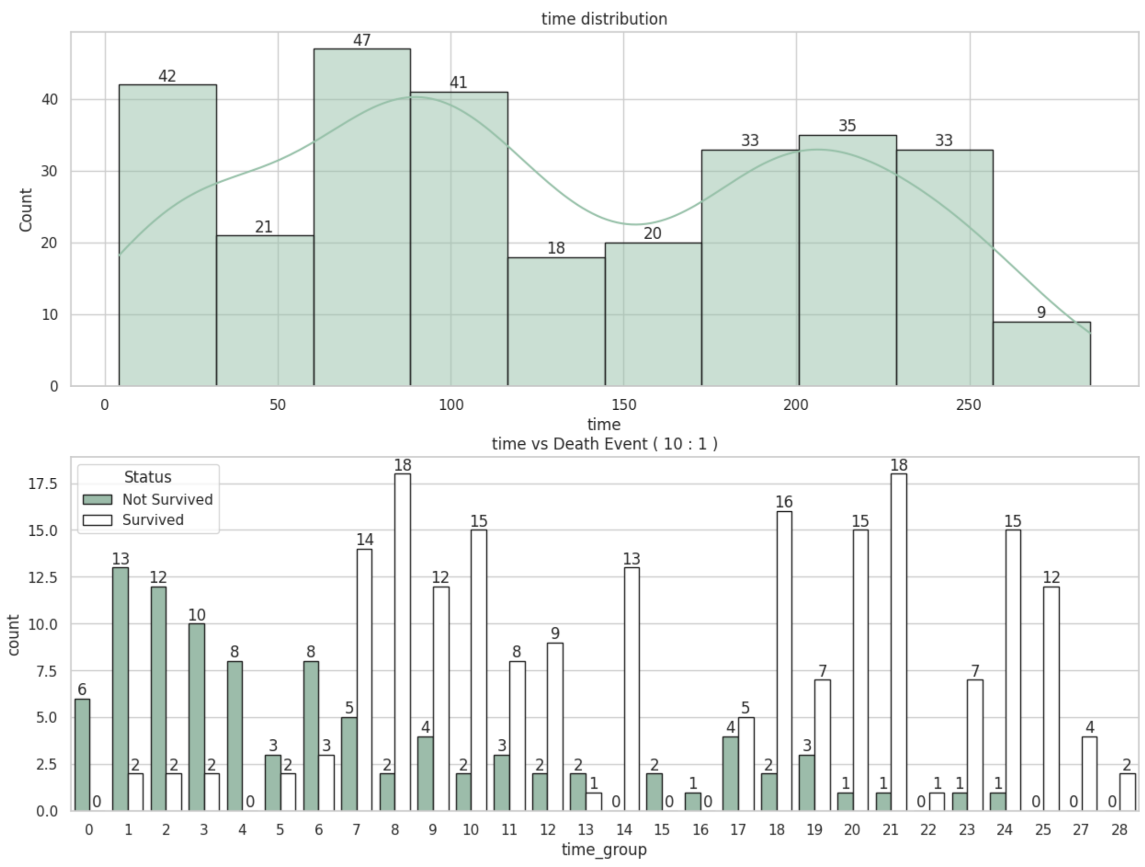
```
numplot(df,'serum_sodium',5)
```



time

In [405]:

```
numplot(df,'time',10)
```



Sumary

Numerical Features Insight :

- Cases of DEATH_EVENT initiate from the age of 45. Some specific peaks of high cases of DEATH_EVENT can be observed at 45, 50, 60, 65, and 70
- High cases of DEATH_EVENT can be observed for ejaction_fraction values from 20 - 60.
- serum_creatinine values from 0.6 to 3.0 have higher probability to lead to DEATH_EVENT.
- serum_sodium values 127 - 145 indicate towards a DEATH_EVENT due to heart failure.
- DEATH_EVENT cases are on a high for the values between 0(0x100) - 500(5x100) for creatinine_phosphokinase.
- platelets values between 0(0x10^5) - 400,000(4x10^5) are prone to heart failures leading to DEATH_EVENT.
- For the time feature, values from 0(0x10) - 60(6*10) have higher probability to lead to a DEATH_EVENT.

Categorical Features Sumary :

- age : 50 - 70
- creatinine_phosphokinase : 0 - 500
- ejaction_fraction : 20 - 40
- platelets : 200,000 - 300,000
- serum_creatinine : 1 - 2
- serum_sodium : 130 - 140

- time : 0 - 50

General Information

- age : General aging leads to heart failures.
- creatinine_phosphokinase : > 120 mcg/L
- ejection_fraction : Normal Range 55% - 70%. Below 55% is prone to heart failures.
- platelets : Low and very high values of platelets led to heart failure.
- serum_creatinine : 0.8 - 1.7 (mg/dL) is the range of values that leads to most heart failures.
- serum_sodium : Above 130 (mEq/L), chances of heart failure increases by alot.
- time : Ideal follow-up period is of 14 days. Anything above 14 days can lead to worse situations.

Features Engineering

Scaling

In [406]:

```
mms = MinMaxScaler() # Normalization
ss = StandardScaler() # Standardization

# Normalization
df['age'] = mms.fit_transform(df[['age']])
df['creatinine_phosphokinase'] = mms.fit_transform(df[['creatinine_phosphokinase']])
df['ejection_fraction'] = mms.fit_transform(df[['ejection_fraction']])
df['serum_creatinine'] = mms.fit_transform(df[['serum_creatinine']])
df['time'] = mms.fit_transform(df[['time']])

# Standardization
df['platelets'] = ss.fit_transform(df[['platelets']])
df['serum_sodium'] = ss.fit_transform(df[['serum_sodium']])
df.head()
```

Out[406]:

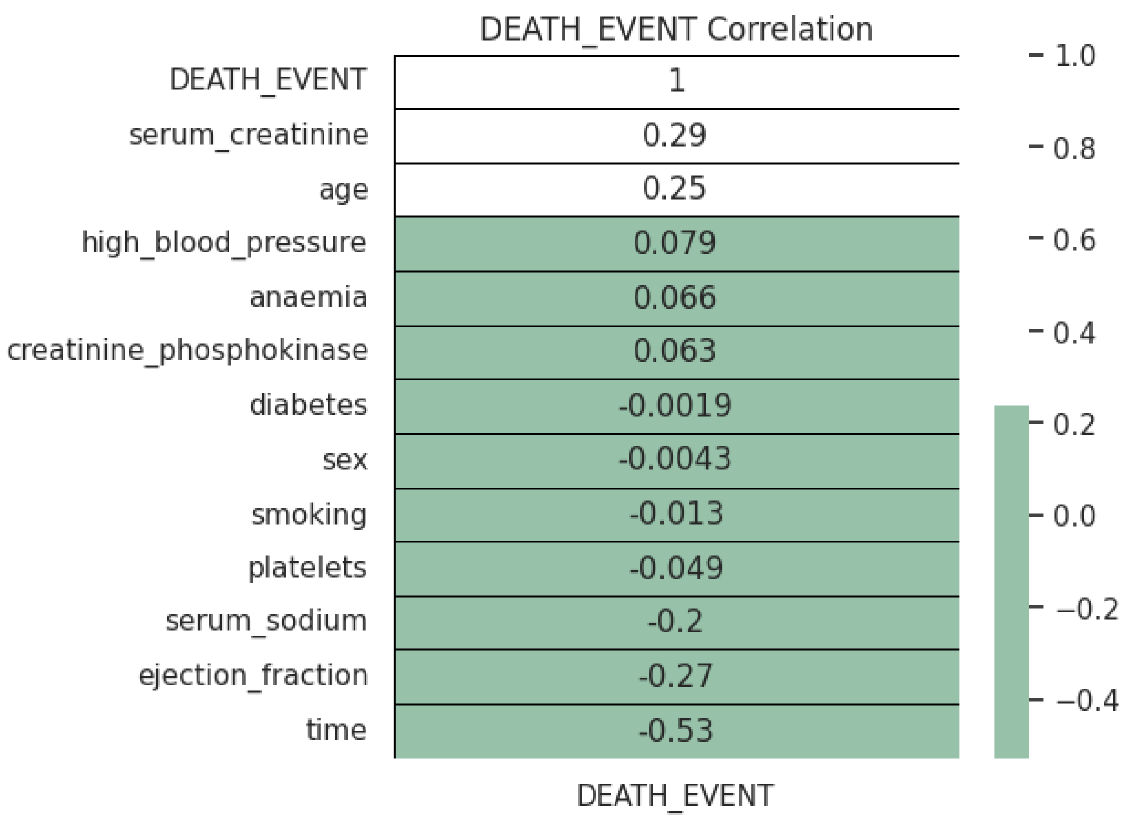
	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_press
0	0.636364	0	0.071319	0	0.090909	
1	0.272727	0	1.000000	0	0.363636	
2	0.454545	0	0.015693	0	0.090909	
3	0.181818	1	0.011227	0	0.090909	
4	0.454545	1	0.017479	1	0.090909	

5 rows × 21 columns

Correlation

In [407]:

```
corr = data.corrwith(data['DEATH_EVENT']).sort_values(ascending = False).to_frame()
corr.columns = ['DEATH_EVENT']

plt.subplots(figsize = (5,5))
sns.heatmap(corr,annot = True,cmap = colors,linewidths = 0.4,linecolor = 'black');

plt.title('DEATH_EVENT Correlation');
```



Insight :

> *Features like high_blood_pressure, anaemia, creatinine_phosphokinase, diabetes, sex, smoking, and platelets do not display any kind of correlation with DEATH_EVENT.*

We will create 2 models :

- Based on the statistical test, we will drop the following features : high_blood_pressure, anaemia, creatinine_phosphokinase, diabetes, sex, smoking, and platelets
- Based on the General information., we will drop the following features : sex, platelets.

In [408]:

```
df1=data.copy()
df2=data.copy()

# Dataset for model based on Statistical Test :
df1 = df1.drop(columns = ['anaemia', 'diabetes', 'high_blood_pressure', 'sex', 'smoking',

# Dataset for model based on General Information :
df2 = df2.drop(columns = ['sex','platelets'])
```

Data Balancing

In [409]:

```
over = SMOTE()

f1 = df1.iloc[:,:5].values
t1 = df1.iloc[:,5].values
f1, t1 = over.fit_resample(f1, t1)
Counter(t1)
```

Out[409]:

```
Counter({1: 203, 0: 203})
```

In [410]:

```
over = SMOTE()

f2 = df2.iloc[:,:10].values
t2 = df2.iloc[:,10].values
f2, t2 = over.fit_resample(f2, t2)
Counter(t2)
```

Out[410]:

```
Counter({1: 203, 0: 203})
```

Model

In [411]:

```
x_train1, x_test1, y_train1, y_test1 = train_test_split(f1, t1, test_size = 0.15, random_
x_train2, x_test2, y_train2, y_test2 = train_test_split(f2, t2, test_size = 0.15, random_
```

In [412]:

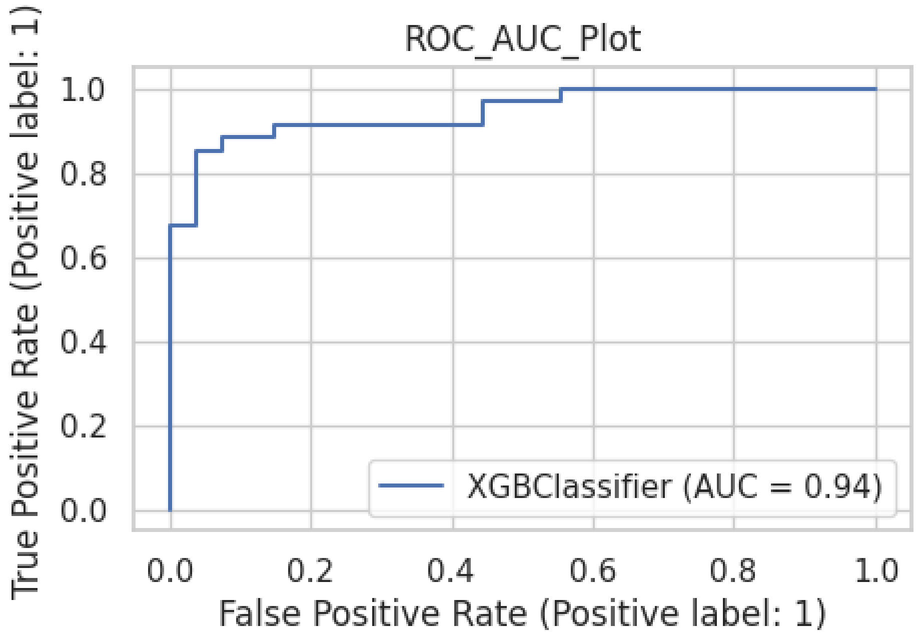
```
def model(classifier,x_train,y_train,x_test,y_test):
    sns.set(rc={'figure.figsize':(5,3)})
    sns.set(style='whitegrid')
    classifier.fit(x_train,y_train)
    prediction = classifier.predict(x_test)
    cv = RepeatedStratifiedKFold(n_splits = 10,n_repeats = 3,random_state = 1)
    print("Cross Validation Score : ",'{0:.2%}'.format(cross_val_score(classifier,x_trair
    print("ROC_AUC Score : ",'{0:.2%}'.format(roc_auc_score(y_test,prediction)))
    # plot_roc_curve(classifier, x_test,y_test)
    RocCurveDisplay.from_estimator(classifier, x_test,y_test)
    plt.title('ROC_AUC_Plot')
    plt.show()

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)))
```

XGB Classifier

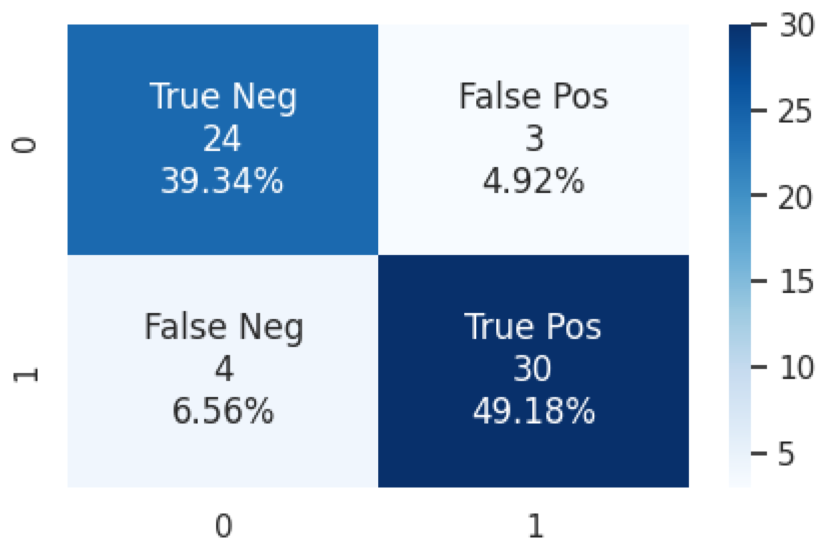
In [413]:

```
classifier_xgb = XGBClassifier(random_state=1)

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

```
Cross Validation Score :  94.59%
ROC_AUC Score :  88.56%
```



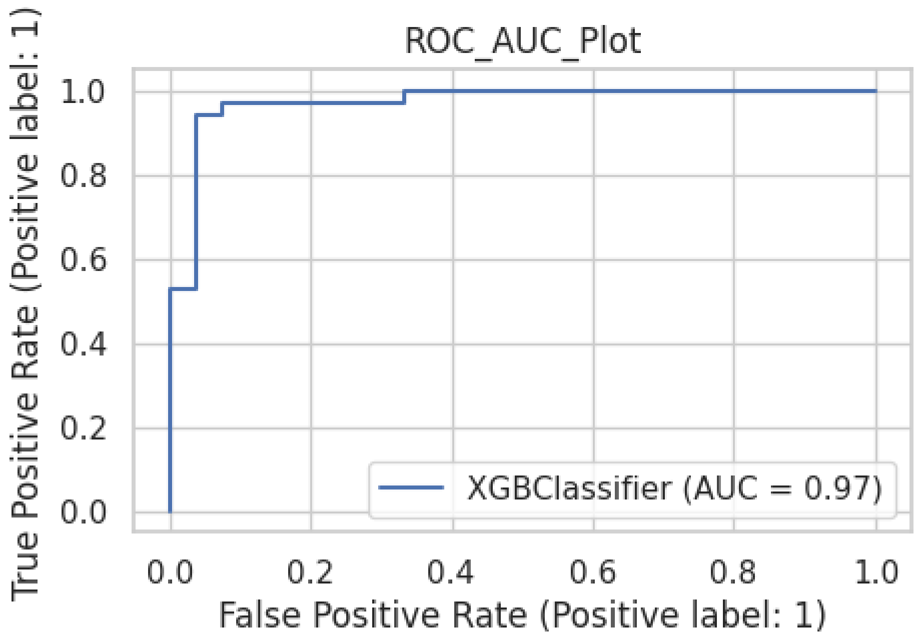
```
              precision    recall  f1-score   support

           0       0.86      0.89      0.87        27
           1       0.91      0.88      0.90        34

    accuracy                           0.89        61
   macro avg       0.88      0.89      0.88        61
weighted avg       0.89      0.89      0.89        61
```



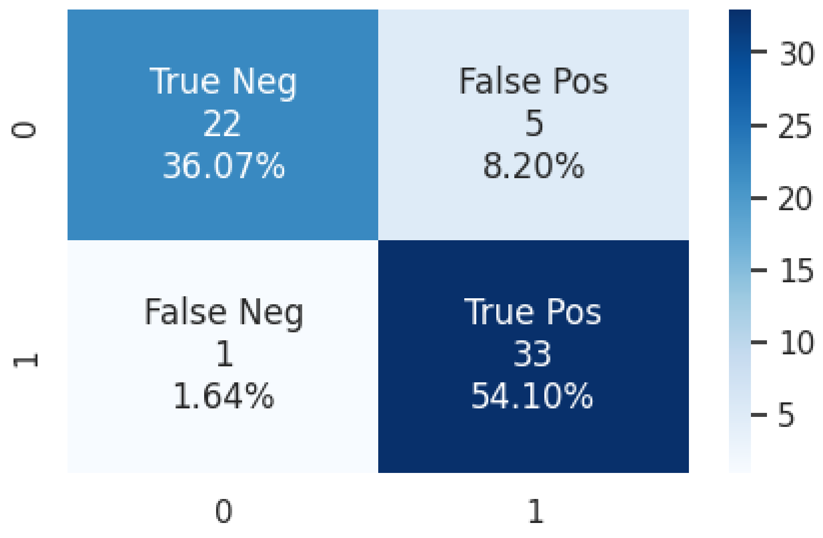
In [414]:

```
model(classifier_xgb,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_xgb,x_test2,y_test2)
```

```
Cross Validation Score :  94.24%
ROC_AUC Score :  89.27%
```



```
              precision    recall  f1-score   support

           0       0.96      0.81      0.88        27
           1       0.87      0.97      0.92        34

    accuracy                           0.90        61
   macro avg       0.91      0.89      0.90        61
weighted avg       0.91      0.90      0.90        61
```

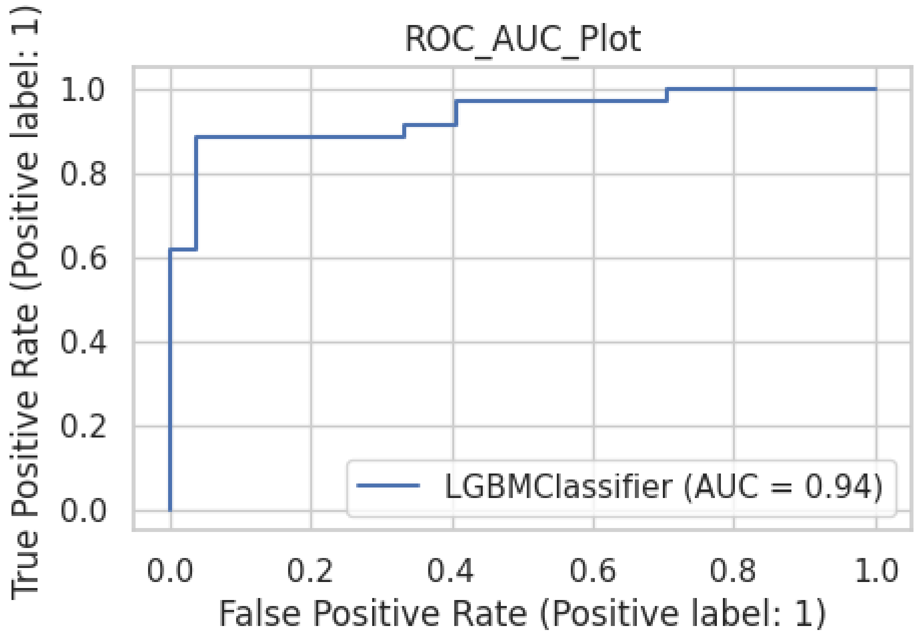


LGBMClassifier

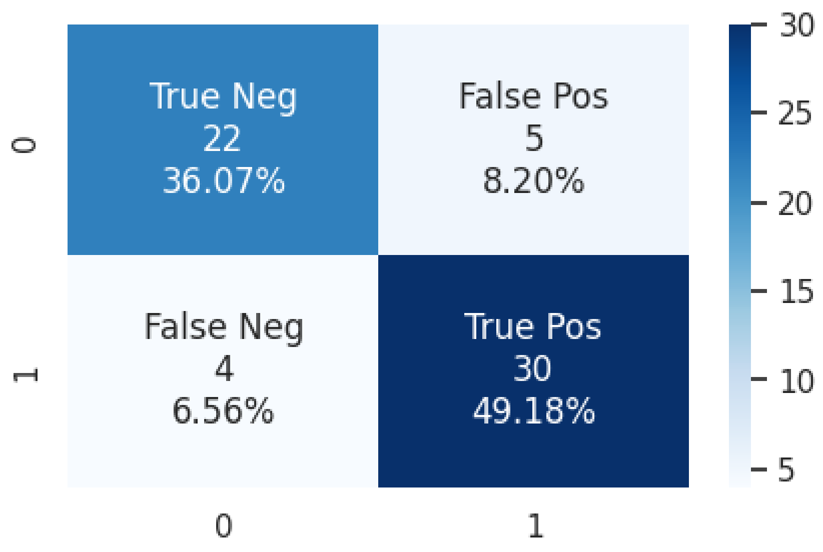
In [415]:

```
classifier_lgbm = LGBMClassifier(random_state=1)

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

```
Cross Validation Score :  94.60%
ROC_AUC Score :  84.86%
```



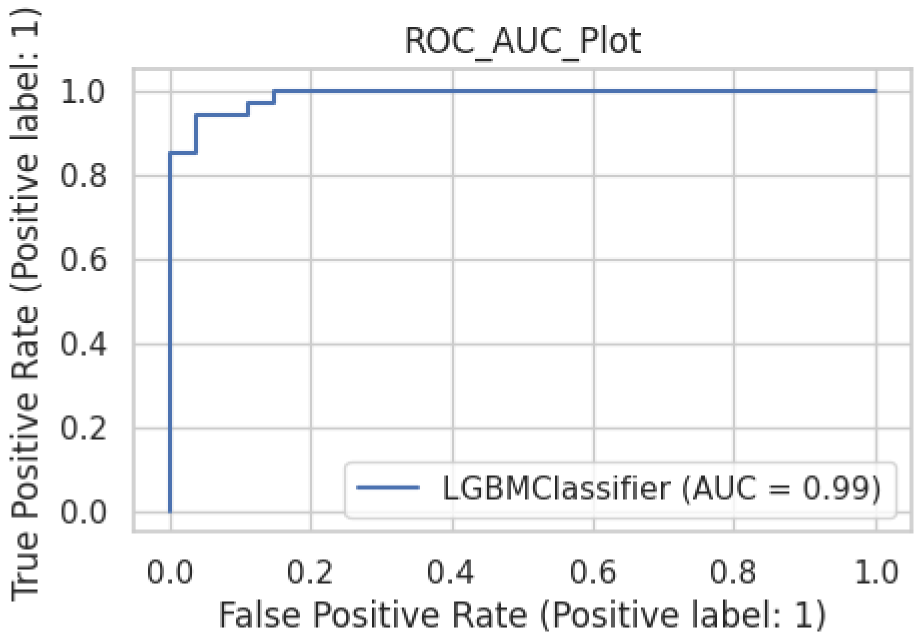
```
              precision    recall  f1-score   support

           0       0.85      0.81      0.83        27
           1       0.86      0.88      0.87        34

    accuracy                           0.85        61
   macro avg       0.85      0.85      0.85        61
weighted avg       0.85      0.85      0.85        61
```



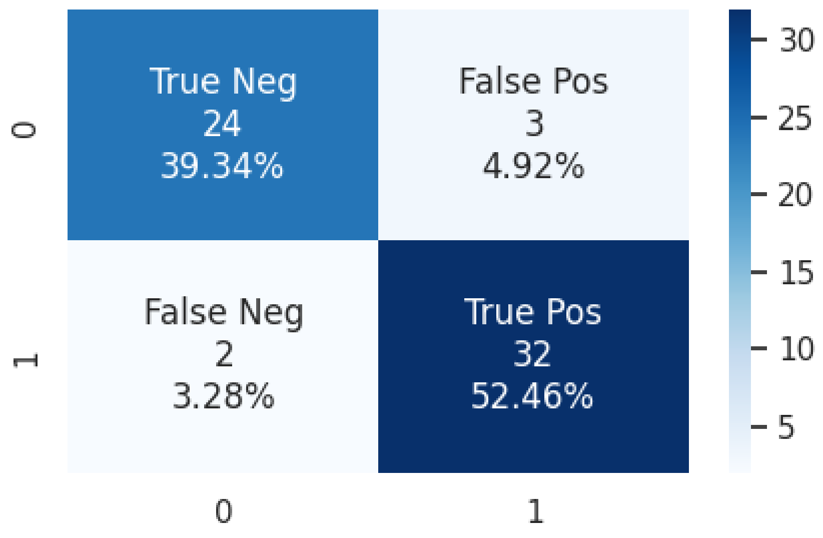
In [416]:

```
model(classifier_lgbm,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_lgbm,x_test2,y_test2)
```

```
Cross Validation Score :  93.84%
ROC_AUC Score :  91.50%
```



```
              precision    recall  f1-score   support

           0       0.92      0.89      0.91        27
           1       0.91      0.94      0.93        34

    accuracy                           0.92        61
   macro avg       0.92      0.92      0.92        61
weighted avg       0.92      0.92      0.92        61
```

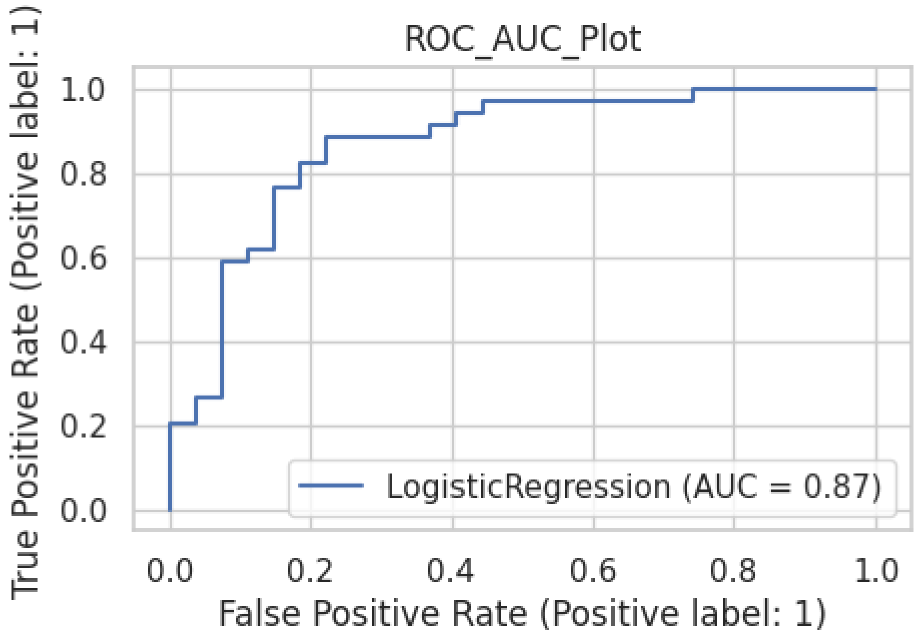


Logistic Regression

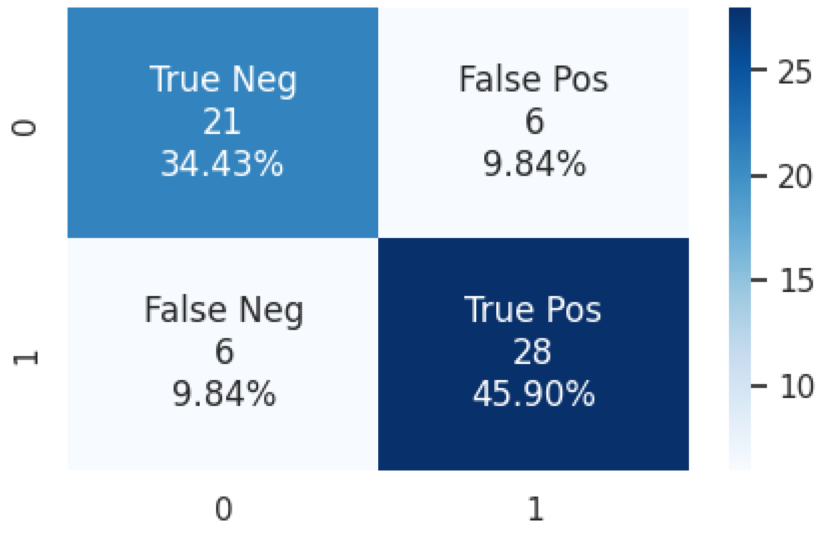
In [417]:

```
classifier_lr = LogisticRegression(random_state = 1)

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

```
Cross Validation Score :  90.15%
ROC_AUC Score :  80.07%
```



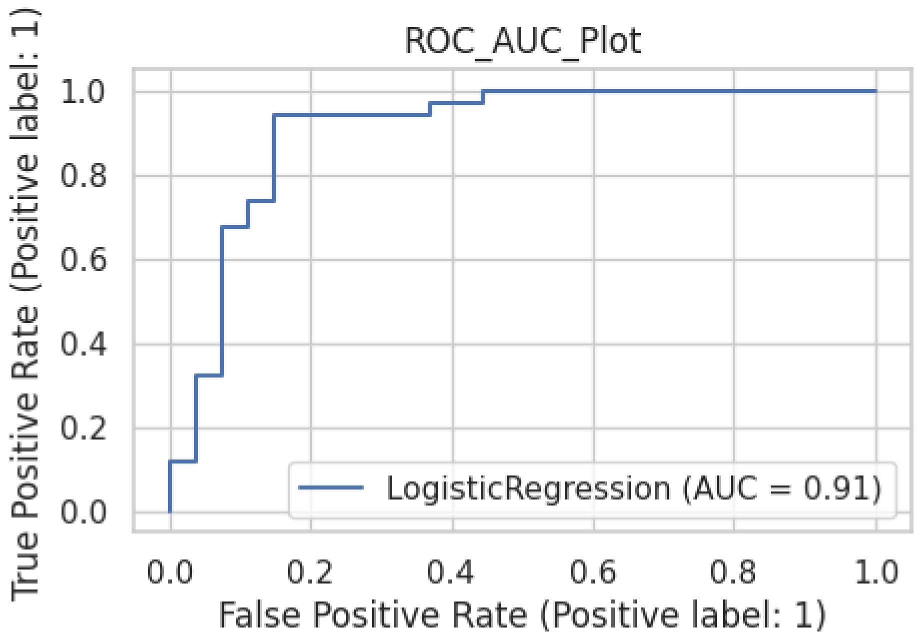
```
              precision    recall  f1-score   support

           0       0.78      0.78      0.78        27
           1       0.82      0.82      0.82        34

    accuracy                           0.80        61
   macro avg       0.80      0.80      0.80        61
weighted avg       0.80      0.80      0.80        61
```



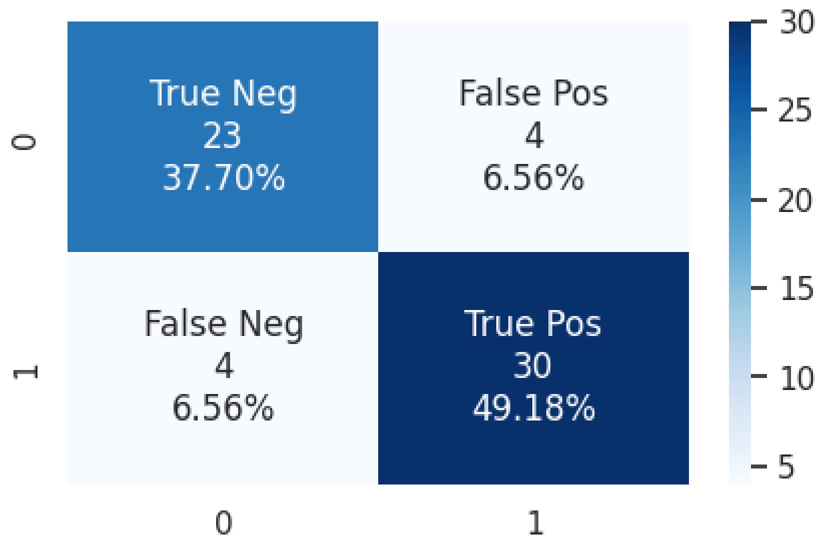
In [418]:

```
model(classifier_lr,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_lr,x_test2,y_test2)
```

```
Cross Validation Score :  90.57%
ROC_AUC Score :  86.71%
```



```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        27
           1       0.88      0.88      0.88        34

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

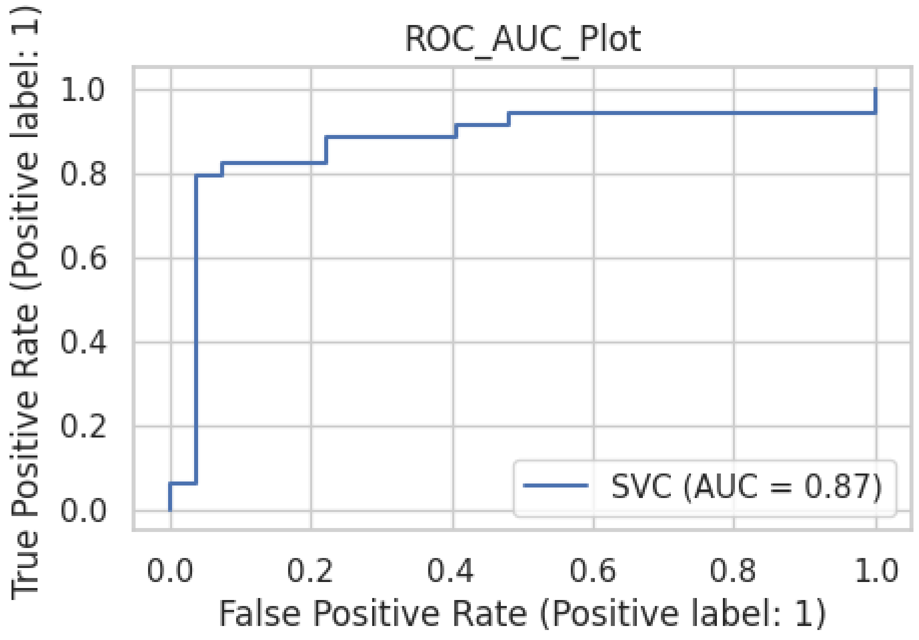


Support Vector Classifier

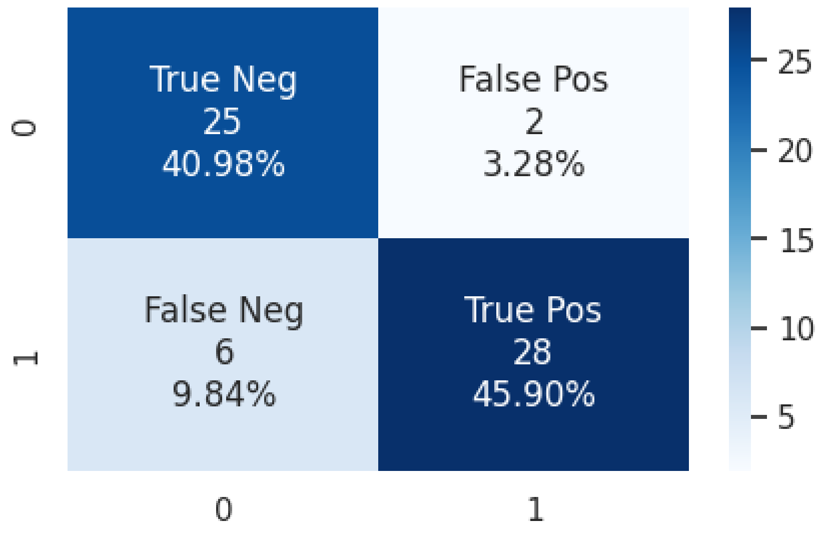
In [419]:

```
classifier_svc = SVC()

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

```
Cross Validation Score :  88.89%
ROC_AUC Score :  87.47%
```



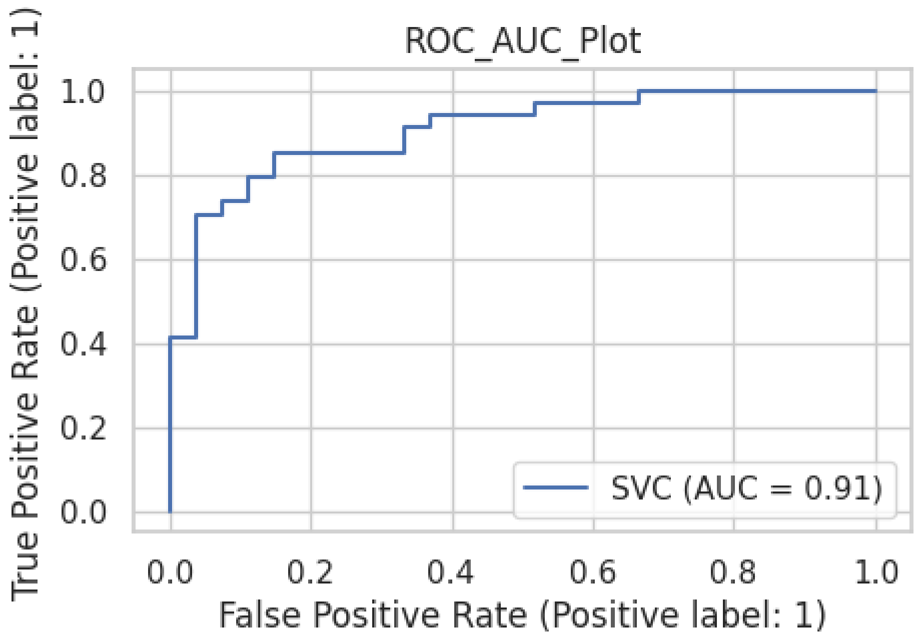
```
              precision    recall  f1-score   support

           0       0.81      0.93      0.86        27
           1       0.93      0.82      0.87        34

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.88      0.87      0.87        61
```



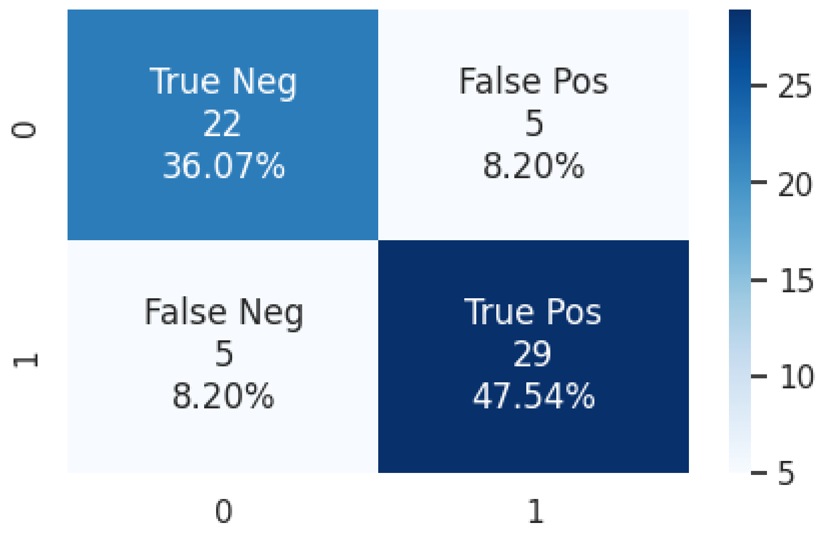
In [420]:

```
model(classifier_svc,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_svc,x_test2,y_test2)
```

```
Cross Validation Score :  82.44%
ROC_AUC Score :  83.39%
```



```
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        27
           1       0.85      0.85      0.85        34

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

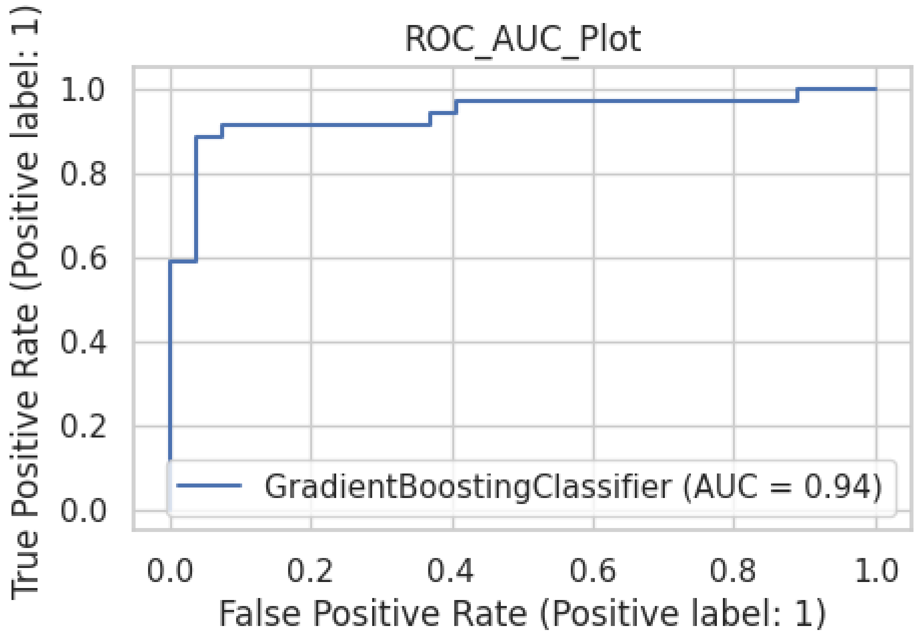


Gradient Booster Classifier

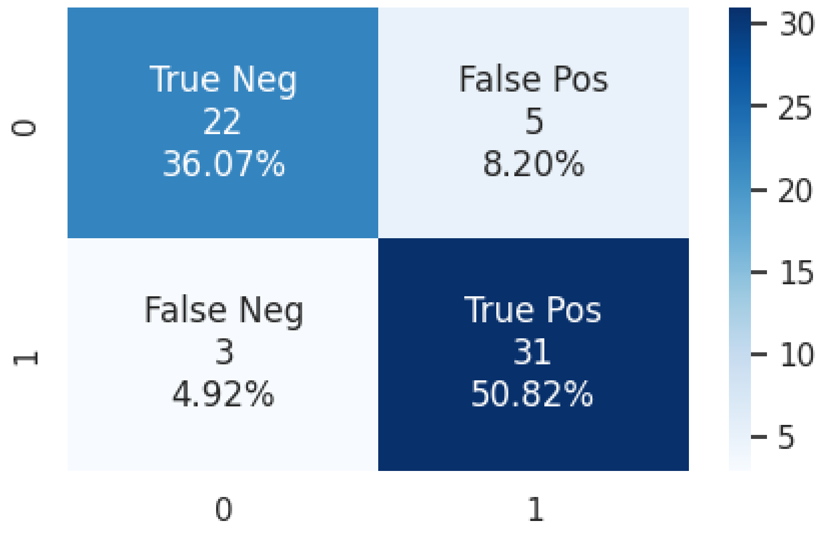
In [421]:

```
classifier_grad = GradientBoostingClassifier(random_state=1)

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

```
Cross Validation Score :  94.18%
ROC_AUC Score :  86.33%
```



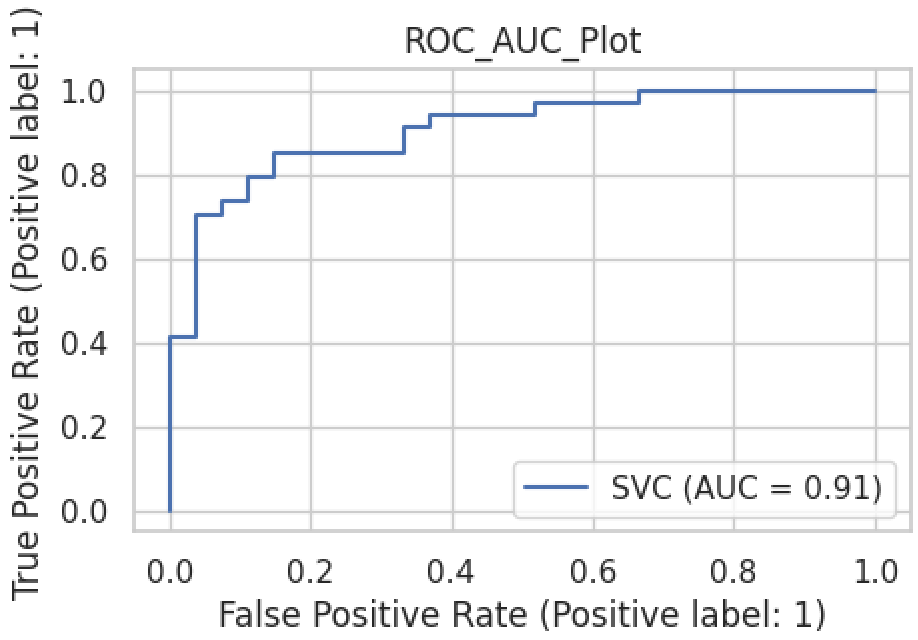
```
              precision    recall  f1-score   support

           0       0.88      0.81      0.85        27
           1       0.86      0.91      0.89        34

    accuracy                           0.87        61
   macro avg       0.87      0.86      0.87        61
weighted avg       0.87      0.87      0.87        61
```



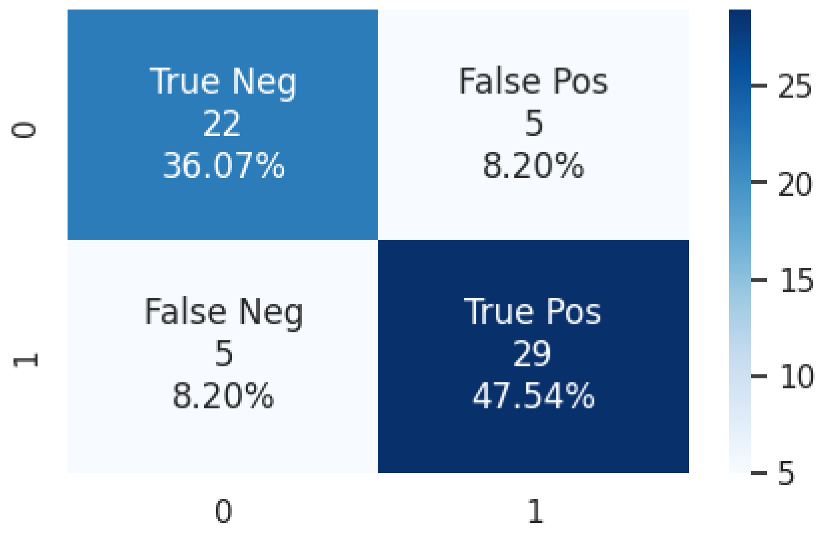
In [422]:

```
model(classifier_svc,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_svc,x_test2,y_test2)
```

```
Cross Validation Score :  82.44%
ROC_AUC Score :  83.39%
```



```
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        27
           1       0.85      0.85      0.85        34

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

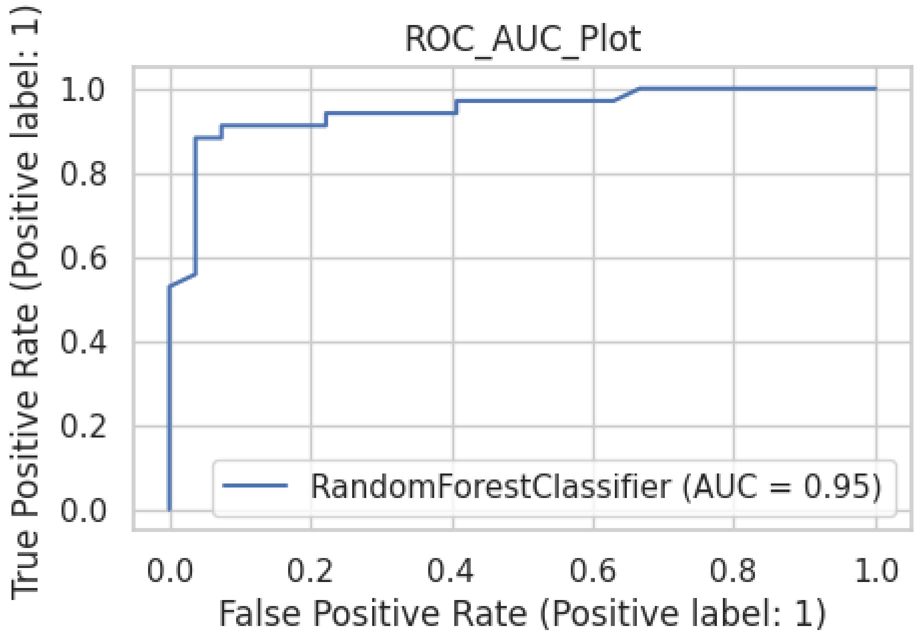


Random Forest Classifier

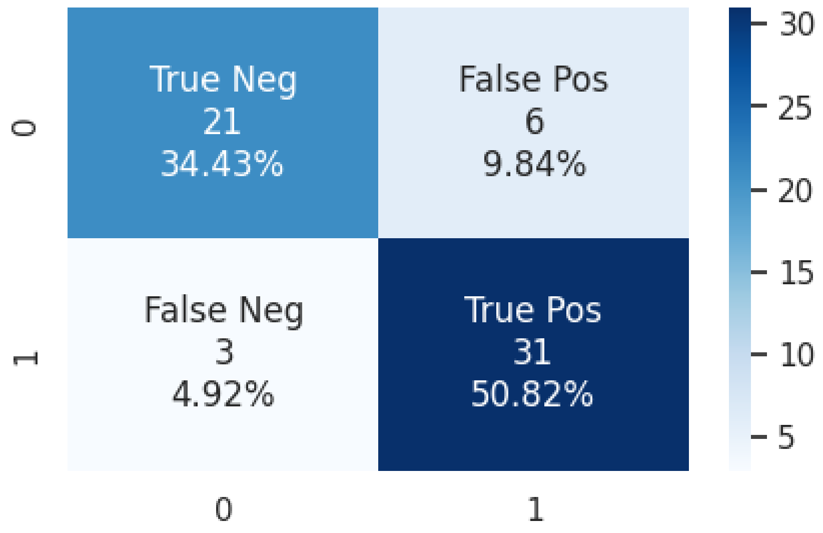
In [423]:

```
classifier_rdf = RandomForestClassifier(random_state=1)

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

```
Cross Validation Score :  95.13%
ROC_AUC Score :  84.48%
```



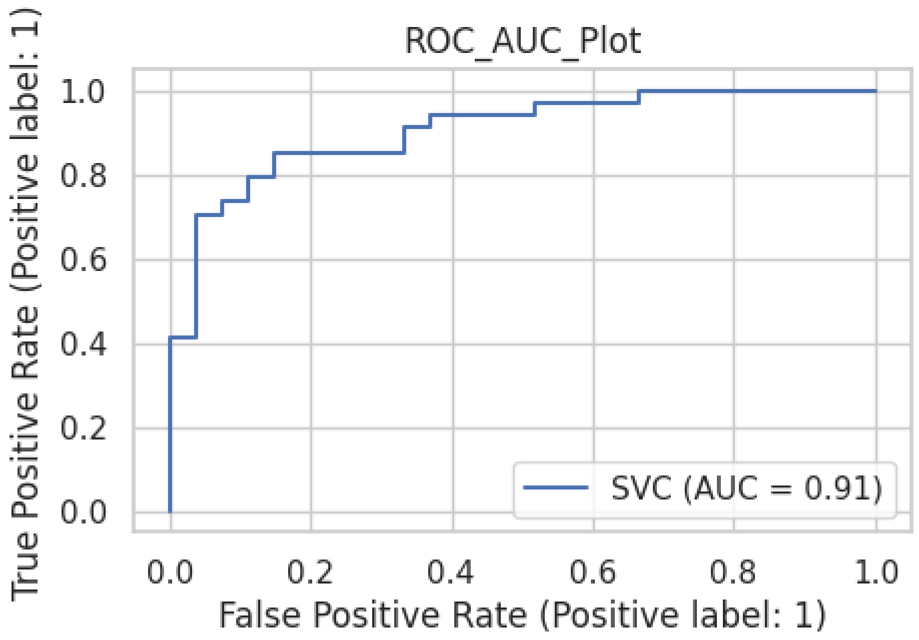
```
              precision    recall  f1-score   support

           0       0.88      0.78      0.82        27
           1       0.84      0.91      0.87        34

    accuracy                           0.85        61
   macro avg       0.86      0.84      0.85        61
weighted avg       0.85      0.85      0.85        61
```



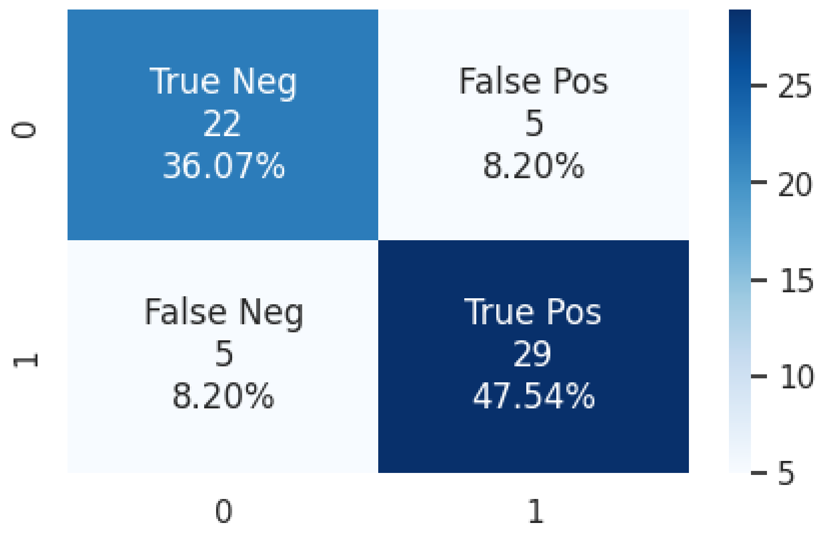
In [424]:

```
model(classifier_svc,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_svc,x_test2,y_test2)
```

```
Cross Validation Score :  82.44%
ROC_AUC Score :  83.39%
```



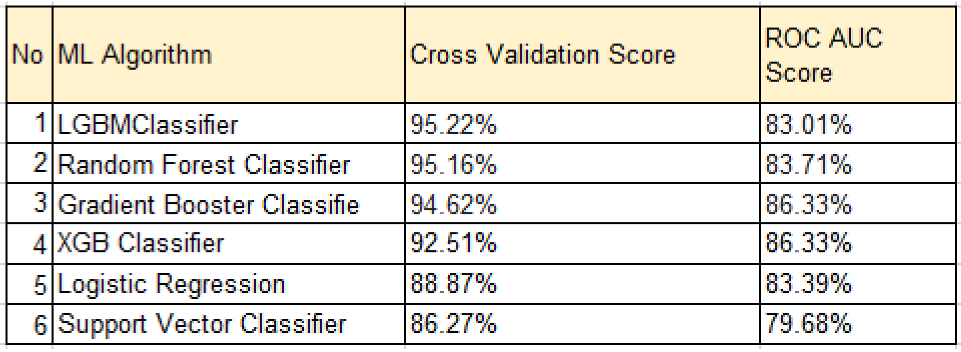
```
              precision    recall  f1-score   support

           0       0.81      0.81      0.81        27
           1       0.85      0.85      0.85        34

    accuracy                           0.84        61
   macro avg       0.83      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```



Result

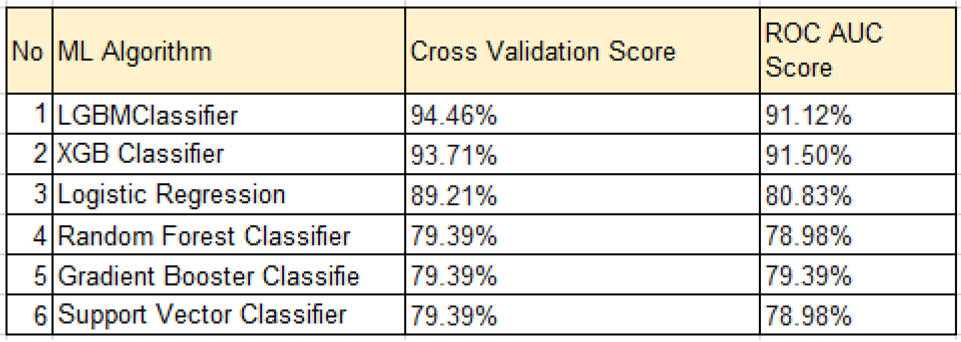
Type *Markdown* and LaTeX: α^2

Dataset 1



No	ML Algorithm	Cross Validation Score	ROC AUC Score
1	LGBMClassifier	95.22%	83.01%
2	Random Forest Classifier	95.16%	83.71%
3	Gradient Booster Classifie	94.62%	86.33%
4	XGB Classifier	92.51%	86.33%
5	Logistic Regression	88.87%	83.39%
6	Support Vector Classifier	86.27%	79.68%

Dataset 2



No	ML Algorithm	Cross Validation Score	ROC AUC Score
1	LGBMClassifier	94.46%	91.12%
2	XGB Classifier	93.71%	91.50%
3	Logistic Regression	89.21%	80.83%
4	Random Forest Classifier	79.39%	78.98%
5	Gradient Booster Classifie	79.39%	79.39%
6	Support Vector Classifier	79.39%	78.98%

*From these results it is found that **Dataset 2** shows better results and **LGBM** is the best model*

Hyperparameter Tuning

In [425]:

```
# pip install flaml
```

In [426]:

```
# from flaml import AutoML

# automl = AutoML()
# settings = {
#     "time_budget": 1200,  # total running time in seconds
#     "metric": 'roc_auc',  # primary metrics for regression can be chosen from: ['mae','
#     "estimator_list": ['lgbm'],  # list of ML learners; we tune lightgbm in this exampl
#     "task": 'classification',  # task type
#     "log_file_name": '/content/drive/MyDrive/heart_lg2.log',  # flaml log file
#     "seed": 1,    # random seed
# }
# automl.fit(X_train=x_train2, y_train=y_train2, **settings)
```

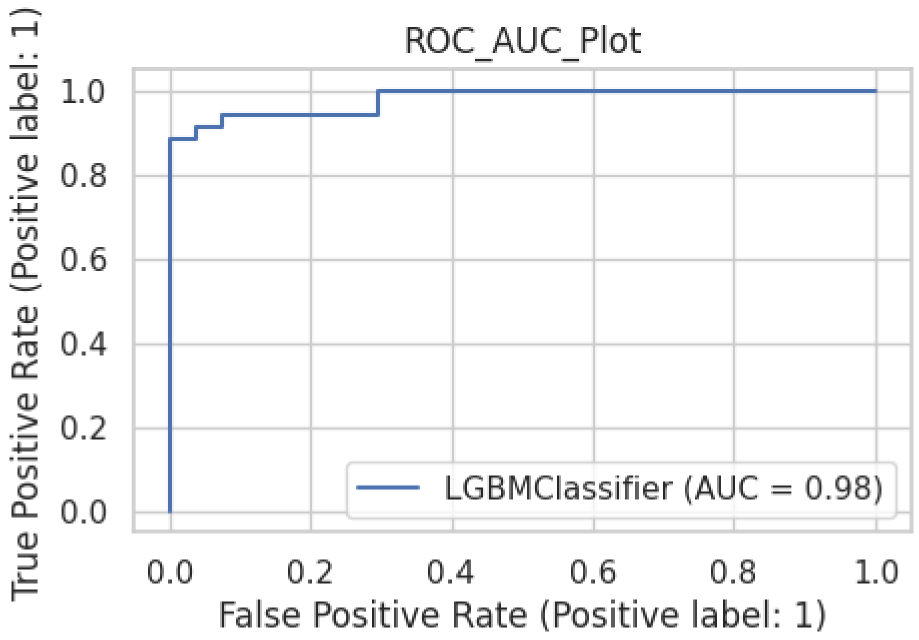
In [427]:

```
# I used flaml hyperparameter tuning for 20 minutes and got this results

classifier_lgbm = LGBMClassifier(colsample_bytree=0.26649620250942635,
               learning_rate=0.02058909150877934, max_bin=127,
               min_child_samples=7, n_estimators=184, num_leaves=48,
               reg_alpha=0.004090180440029941, reg_lambda=0.0009765625,
               verbose=-1)
```

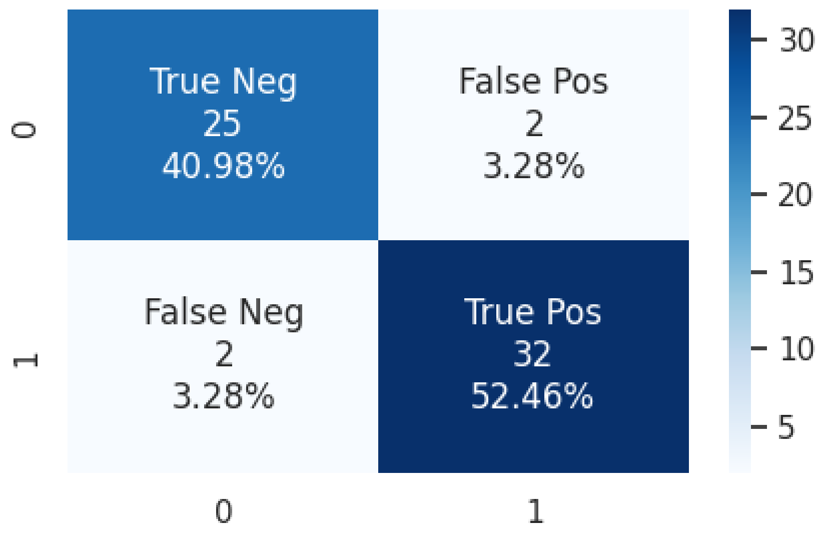
In [428]:

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

```
Cross Validation Score :  94.36%
ROC_AUC Score :  93.36%
```



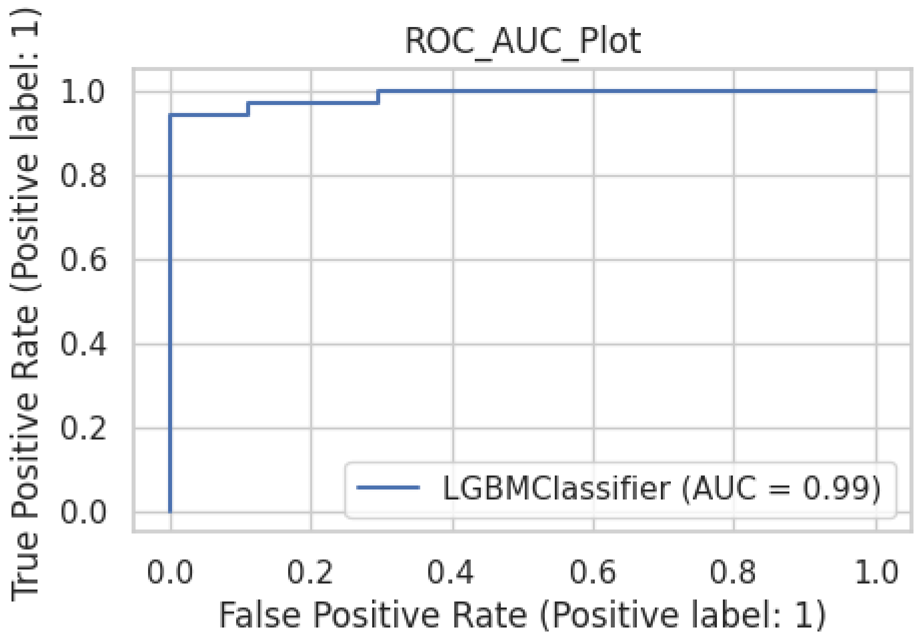
```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        27
           1       0.94      0.94      0.94        34

    accuracy                           0.93        61
   macro avg       0.93      0.93      0.93        61
weighted avg       0.93      0.93      0.93        61
```



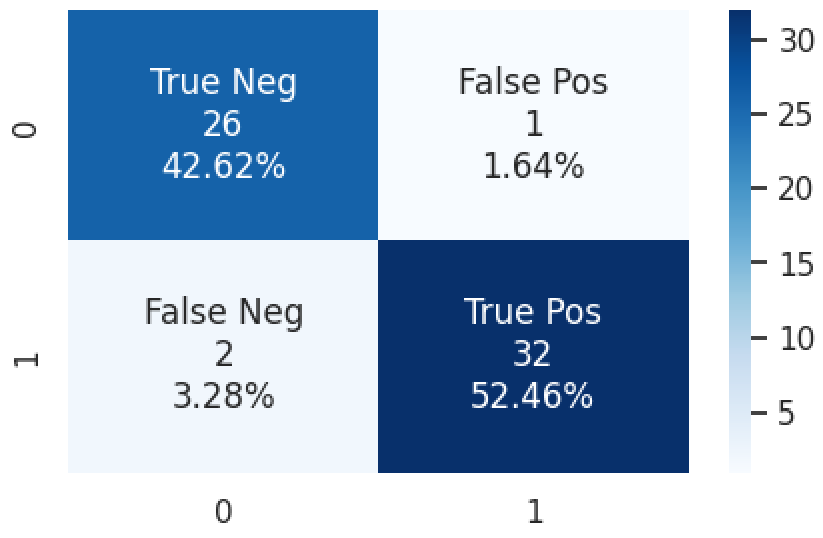
In [429]:

```
model(classifier_lgbm,x_train2,y_train2,x_test2,y_test2)
model_evaluation(classifier_lgbm,x_test2,y_test2)
```

```
Cross Validation Score :  96.03%
ROC_AUC Score :  95.21%
```



```
              precision    recall  f1-score   support

           0       0.93      0.96      0.95        27
           1       0.97      0.94      0.96        34

    accuracy                           0.95        61
   macro avg       0.95      0.95      0.95        61
weighted avg       0.95      0.95      0.95        61
```



The Final Results

After using flaml, the results of both datasets improved

> Dataset 1 :

- Cross Validation Score : 94%
- ROC_AUC Score : 93%

> Dataset 2 :

- Cross Validation Score : 96 %
- ROC_AUC Score : 95%