Project 3 - Supervised Machine Learning - Classification

Objective

The main objective of my analysis is to focus on **prediction** of heart failure based on the dataset described

Dataset Features

1. Age = age of the patients
2. Anaemia - Decrease of red blood cells or hemoglobin
3. Creatinine_phosphokinase - Level of the CPK enzyme in the blood (mcg/L)
4. Diabetes - If the patient has diabetes
5. Ejection_fraction - Percentage of blood leaving the heart at each contraction
6. High_blood_pressure - If the patient has hypertension
7. Platelets - Platelets in the blood (kiloplatelets/mL)
8. Serum_creatinine - Level of serum creatinine in the blood (mg/dL)
9. Serum_sodium - Level of serum sodium in the blood (mEq/L)
10. Sex - Woman or man
11. Smoking - If the patient smokes or not
12. Time - Follow-up period (days)

Dependent Variable

DEATH_EVENT

- If the patient deceased during the follow-up period

Import Packages

In [1]:
```
%config Completer.use_jedi = False
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from scipy import stats
import random
```

In [2]:
```
data = pd.read_csv('heart_failure_clinical_records_dataset.csv')
data.head()
```

Out[2]:

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

In [3]:
```
data.describe()
```

Out[3]:

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

EDA

EDA consists of:

- Finding Missing Values.
- Check for discrete and continuous variables for easy visualization.
- Correlation Matrix/Heatmap for finding relationship between independent variables and dependent variable.
- Also the heatmap depicts the correlation between the feature sets so that one of the correlated features can be dropped.
- Using Countplot, find the distribution of each feature individually and also wrt dependent variable.
- Find the balance in the dependent variable, so that necessary steps can be taken.
- Outliers Correction and Distribution Graph for better understanding of data with dependent variable.

In [4]:
```
data.isnull().sum()
```

Out[4]:
```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

Observations

- No missing values present.
- So let's see the analysis ahead.

In [5]:
```
for feature in data.columns:
    print(feature, ':', len(data[feature].unique()))
```

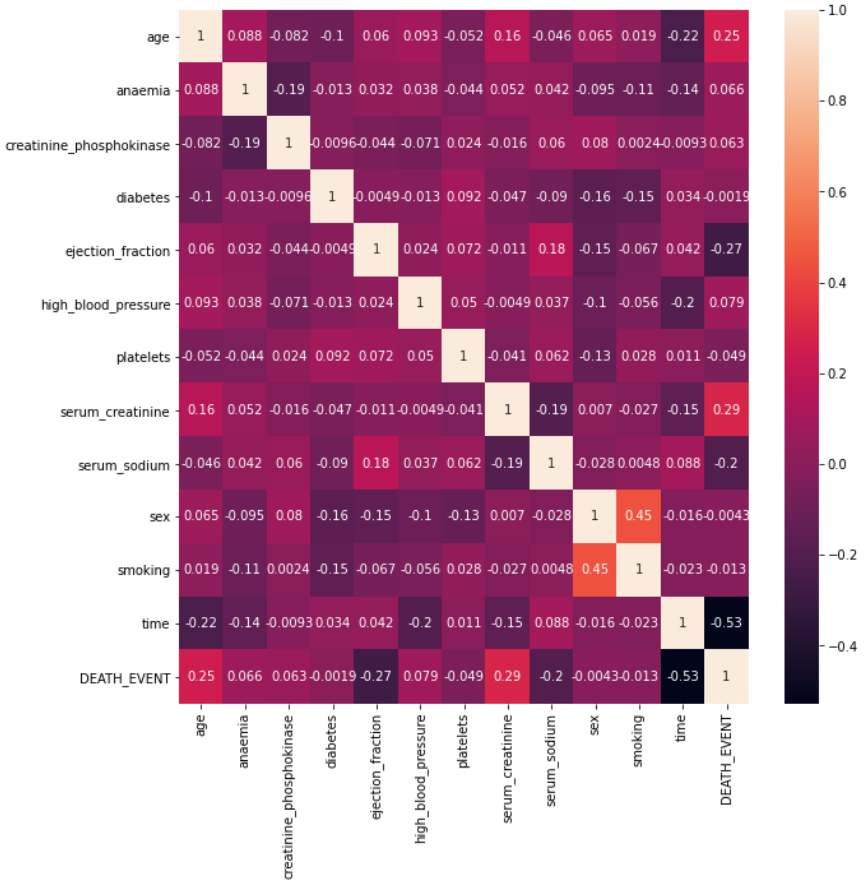
```
age : 47
anaemia : 2
creatinine_phosphokinase : 208
diabetes : 2
ejection_fraction : 17
high_blood_pressure : 2
platelets : 176
serum_creatinine : 40
serum_sodium : 27
sex : 2
smoking : 2
time : 148
DEATH_EVENT : 2
```

In [6]:
```
discrete_features, continuous_features = [], []
for feature in data.columns:
    if feature == 'DEATH_EVENT':
        label = ['DEATH_EVENT']
    elif len(data[feature].unique()) >= 10:
        continuous_features.append(feature)
    else:
        discrete_features.append(feature)

print('Discrete: ', discrete_features, '\n', 'Continuous', continuous_features)
```

```
Discrete:  ['anaemia', 'diabetes', 'high_blood_pressure', 'sex', 'smoking']
 Continuous ['age', 'creatinine_phosphokinase', 'ejection_fraction', 'platelets', 'serum_creatinine', 'serum_sodium', 'tim
e']
```

In [7]:

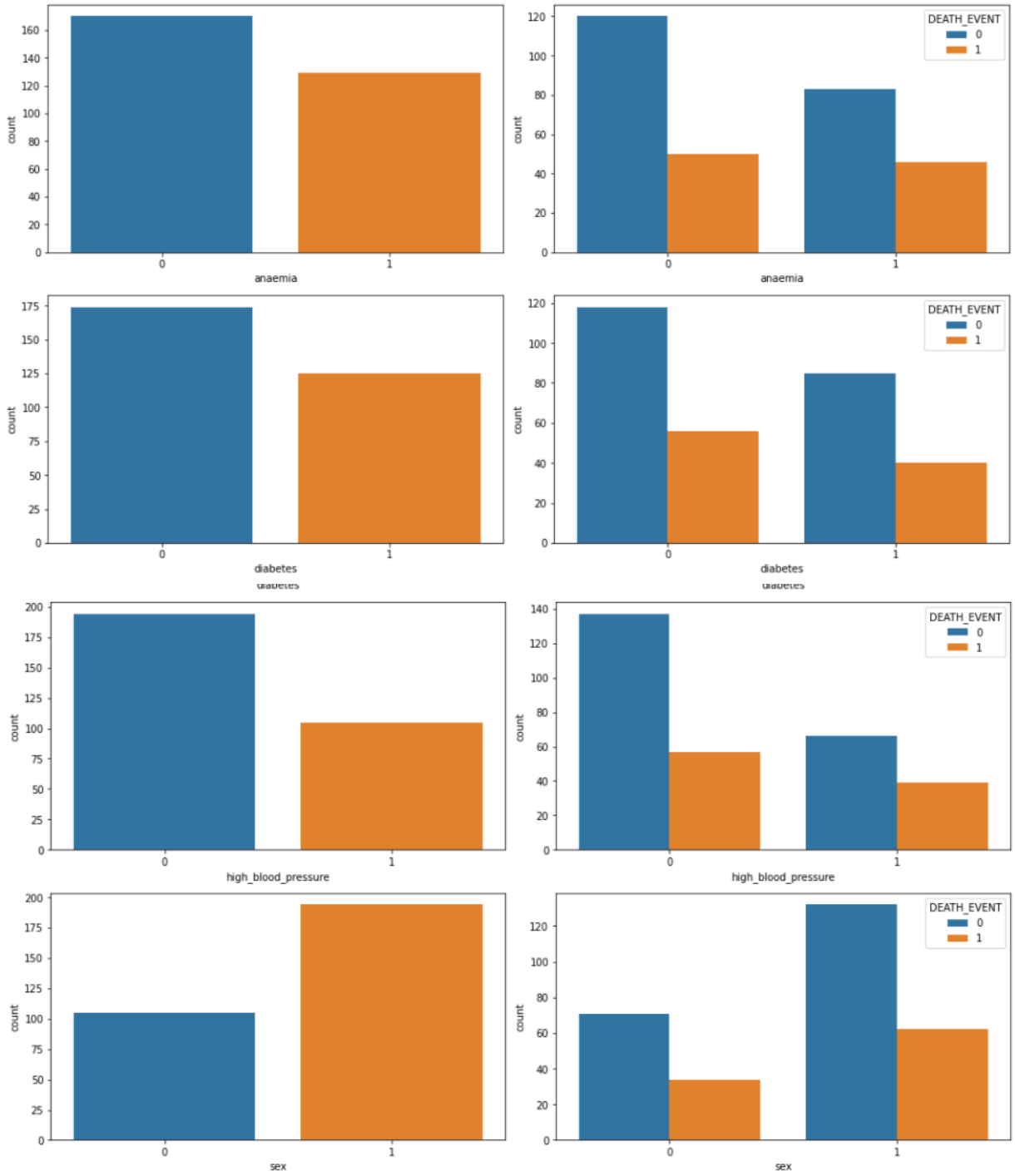
```
correlation = data.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(correlation, annot=True)
plt.show()
```

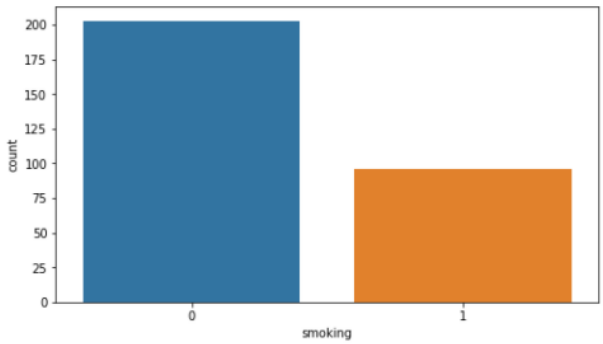


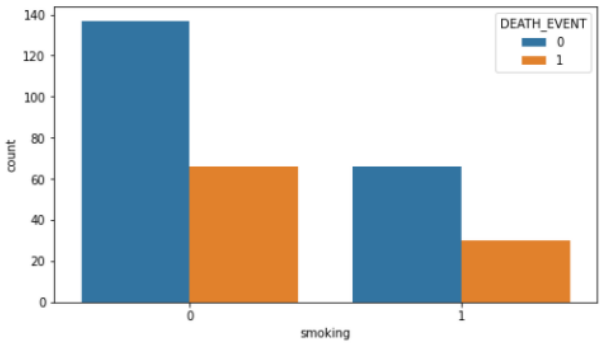
Observations

- There is nothing to conclude from discrete features correlation matrix.
- From the correlation matrix for continuous features, time is inversely correlated to death. Thus patients with less follow up time are prone to heart failure.
- Based on EDA, features such as **anaemia, diabetes, age, sex, smoking** are less contributing.

In [8]:
```
fig, ax = plt.subplots(len(discrete_features), 2, figsize=(14,20))

for i in range(len(discrete_features)):
    sns.countplot(ax=ax[i, 0], x=discrete_features[i], data=data)
    sns.countplot(ax=ax[i, 1], x=discrete_features[i], hue='DEATH_EVENT', data=data)
fig.tight_layout(pad=1)
plt.show()
```

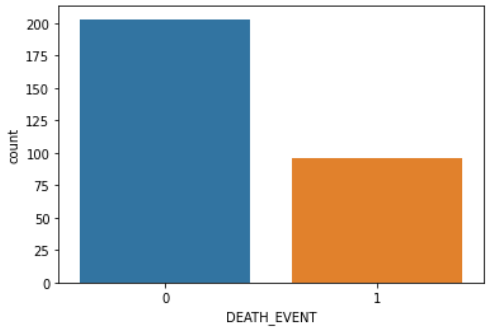






In [9]:
```
sns.countplot(x='DEATH_EVENT', data=data)
```

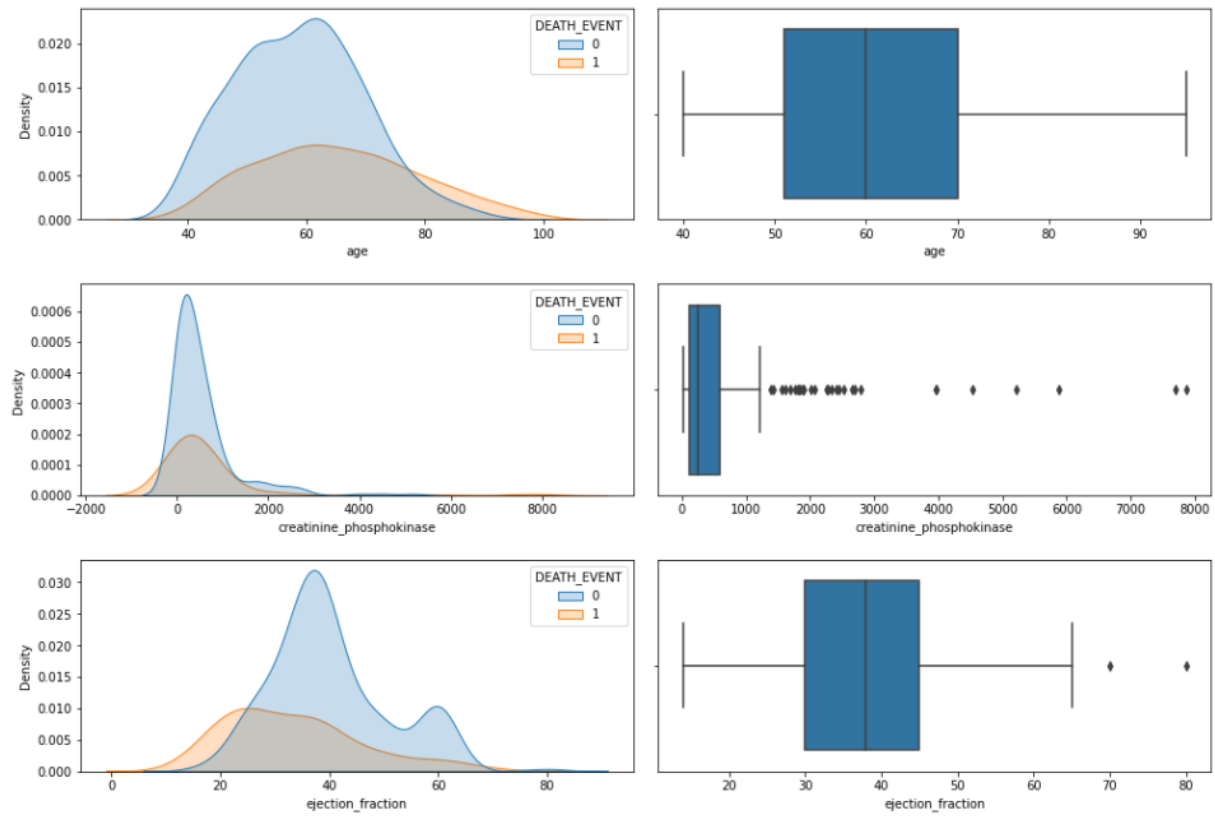
Out[9]: <AxesSubplot:xlabel='DEATH_EVENT', ylabel='count'>

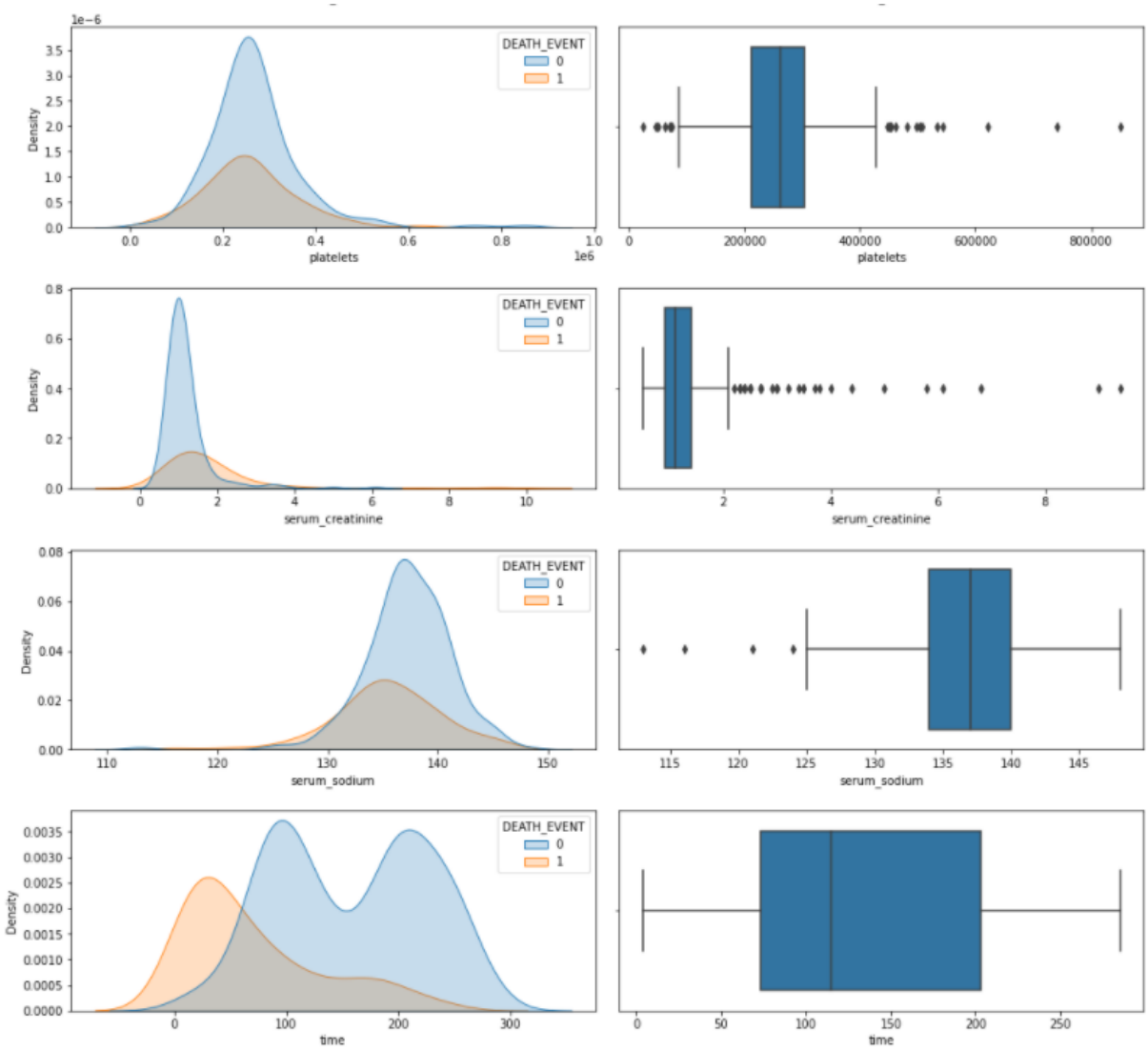


Observations

- There is an imbalance with the target variable, so we can apply cross validation technique with over sampling method compared to under sampling as the data size is small.

In [10]:
```
fig, ax = plt.subplots(len(continuous_features), 2, figsize=(14,22))

for i in range(len(continuous_features)):
    sns.kdeplot(ax=ax[i, 0], x=continuous_features[i], hue='DEATH_EVENT', data=data, fill = True)
    sns.boxplot(ax=ax[i, 1], x=continuous_features[i], data=data)
fig.tight_layout(pad=1)
plt.show()
```





Observations and Insights:

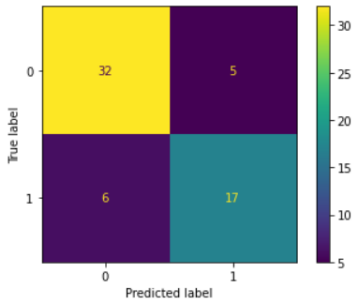
1. No missing values present in the data.
2. From the correlation matrix for continuous features, time is inversely correlated to death. Thus patients with less follow up time are prone to heart failure.
3. Smoking and Sex features are slightly correlated.
4. Based on EDA, features such as anaemia, diabetes, age, sex, smoking are less contributing.
5. There is an imbalance with the target variable, so we can apply cross validation technique with over sampling method compared to under sampling as the data size is small.
6. creatinine_phosphokinase, serum_creatinine and serum_sodium are highly skewed.
7. From KDE Plots and boxplots, we can find that there are outliers in the data.
8. creatinine_phosphokinase, serum_creatinine contains many outliers and can be treated using IQR Formula.

Model Building with SMOTE

In [39]:
```
X = data[['ejection_fraction', 'serum_creatinine', 'time']]
y = data['DEATH_EVENT']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

lof = LocalOutlierFactor()
outlier_rows = lof.fit_predict(X_train)

mask = outlier_rows != -1
X_train, y_train = X_train[mask], y_train[mask]

oversample = SMOTE(sampling_strategy='minority')
X_train, y_train = oversample.fit_resample(X_train, y_train)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

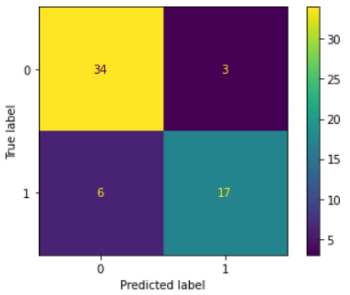
In [40]:
```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
conf = plot_confusion_matrix(model, X_test, y_test)
print ("The accuracy of Logistic Regression is : ", accuracy_score(y_test, y_pred)*100, "%")
print(classification_report(y_test, y_pred))
```

```
The accuracy of Logistic Regression is :  81.66666666666667 %
              precision    recall  f1-score   support

           0       0.84      0.86      0.85        37
           1       0.77      0.74      0.76        23

    accuracy                           0.82        60
   macro avg       0.81      0.80      0.80        60
weighted avg       0.82      0.82      0.82        60
```



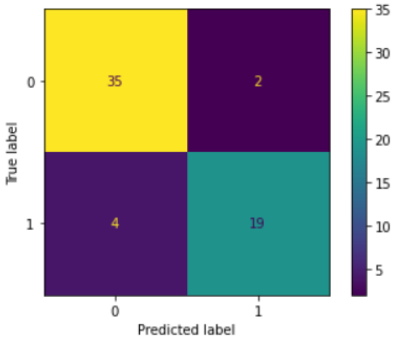
In [41]:
```
model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
conf = plot_confusion_matrix(model, X_test, y_test)
print ("The accuracy of Random Forest is : ", accuracy_score(y_test, y_pred)*100, "%")
print(classification_report(y_test, y_pred))
```

```
The accuracy of Random Forest is :  85.0 %
              precision    recall  f1-score   support

           0       0.85      0.92      0.88        37
           1       0.85      0.74      0.79        23

    accuracy                           0.85        60
   macro avg       0.85      0.83      0.84        60
weighted avg       0.85      0.85      0.85        60
```



In [42]:
```
model = GradientBoostingClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
conf = plot_confusion_matrix(model, X_test, y_test)
print ("The accuracy of Gradient Boost is : ", accuracy_score(y_test, y_pred)*100, "%")
print(classification_report(y_test, y_pred))
```

```
The accuracy of Gradient Boost is :  90.0 %
              precision    recall  f1-score   support

           0       0.90      0.95      0.92        37
           1       0.90      0.83      0.86        23

    accuracy                           0.90        60
   macro avg       0.90      0.89      0.89        60
weighted avg       0.90      0.90      0.90        60
```



Best Model to Choose

Models	Accuracy	Recall
Logistic Regression	82%	81%
Random Forest	85%	83%
Gradient Boosting	90%	89%

Based on both accuracy and recall score, **Gradient Boosting** outperformed other algorithms. The reason for considering recall score here is, the data and prediction to be made about heart failure and in such case less False Negatives is to be predicted and to calculate it with proportion I have used **Recall Score**.

Future Scope

- Ahead while revisiting the model again, I have plans to do more in-depth analysis of the data.
- To add additional derived features to the data based on analysis.
- Can try different classifiers to train the model and also few ensemble methods as well to see any improvement.

Notebook File available at: GitHub

Thank You for reviewing,
Aditya Mahimkar