Our task is to Heartfailure prediction using SVM classifier

Here is the description about the dataset

age: The age of the patient.

anaemia: Presence of anemia in the patient (1 = Yes, 0 = No).

creatinine_phosphokinase: Level of the enzyme creatinine phosphokinase in the blood (measured in mcg/L).

diabetes: Whether the patient has diabetes (1 = Yes, 0 = No).

ejection_fraction: Percentage of blood leaving the heart with each contraction (measured in percentage).

high_blood_pressure: Whether the patient has high blood pressure (1 = Yes, 0 = No).

platelets: Number of platelets in the blood (measured in kiloplatelets/mL).

serum_creatinine: Level of serum creatinine in the blood (measured in mg/dL).

serum_sodium: Level of serum sodium in the blood (measured in mEq/L).

sex: The biological sex of the patient (1 = Male, 0 = Female).

smoking: Whether the patient smokes (1 = Yes, 0 = No).

time: Follow-up period for the patient (measured in days).

DEATH_EVENT: The target variable indicating if the patient died during the follow-up period (1 = Yes, 0 = No).

In [2]:
```
#Step1:importing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_r
```

```
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\_distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.EL2C6PLE4ZYW3ECEVIV3OXXGRN2NRFM2.gfortran-win_amd64.dll
c:\users\vamsi2001\appdata\local\programs\python\python39\lib\site-packages\num
py\.libs\libopenblas.XWYDX2IKJW2NMTWSFYNGFUWKQU3LYTCZ.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

In [3]:
```
df=pd.read_csv('HeartFailure.csv')
df.head()
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	plate
0	75.0	0	582	0	20	1	26500
1	55.0	0	7861	0	38	0	26335
2	65.0	0	146	0	20	0	16200
3	50.0	1	111	0	20	0	21000
4	65.0	1	160	1	20	0	32700

In [4]:
```
df.info()
df.describe()
```

```
<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
```

Out[4]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_
count	299.000000	299.000000	299.000000	299.000000	299.000000	29
mean	60.833893	0.431438	581.839465	0.418060	38.083612	
std	11.894809	0.496107	970.287881	0.494067	11.834841	
min	40.000000	0.000000	23.000000	0.000000	14.000000	
25%	51.000000	0.000000	116.500000	0.000000	30.000000	
50%	60.000000	0.000000	250.000000	0.000000	38.000000	
75%	70.000000	1.000000	582.000000	1.000000	45.000000	
max	95.000000	1.000000	7861.000000	1.000000	80.000000	

In [ ]:
```
1
```

In [34]:
```
1 df.columns
```

Out[34]:
```
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
       'DEATH_EVENT'],
      dtype='object')
```

In [5]:
```
1 cat_cols=['anaemia','diabetes','high_blood_pressure','sex','smoking','DEATH_
2 for i in cat_cols:
3     df[i]=df[i].astype('category')
4 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    category
 2   creatinine_phosphokinase  299 non-null    int64   
 3   diabetes                  299 non-null    category
 4   ejection_fraction         299 non-null    int64   
 5   high_blood_pressure       299 non-null    category
 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    category
 10  smoking                   299 non-null    category
 11  time                      299 non-null    int64   
 12  DEATH_EVENT               299 non-null    category
dtypes: category(6), float64(3), int64(4)
memory usage: 19.0 KB
```

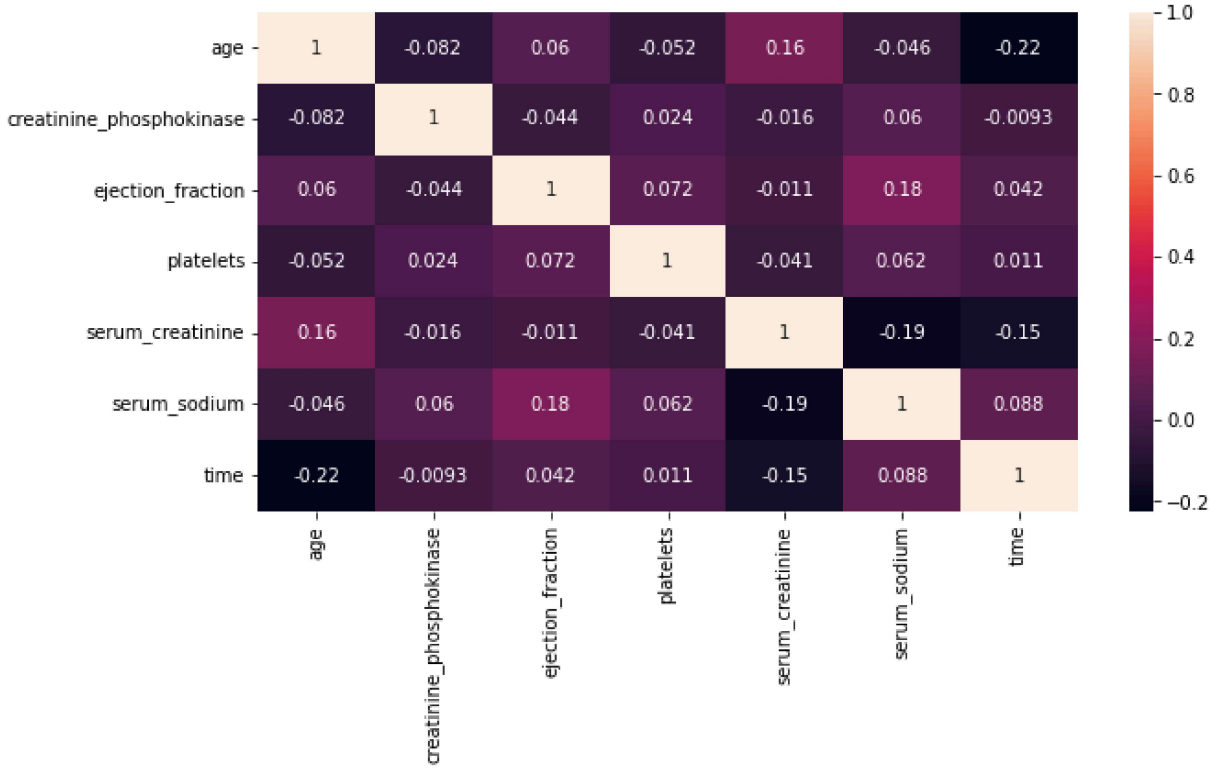
In [6]:
```
1 df.describe()
```

Out[6]:

	age	creatinine_phosphokinase	ejection_fraction	platelets	serum_creatinine	se
count	299.000000	299.000000	299.000000	299.000000	299.00000	
mean	60.833893	581.839465	38.083612	263358.029264	1.39388	
std	11.894809	970.287881	11.834841	97804.236869	1.03451	
min	40.000000	23.000000	14.000000	25100.000000	0.50000	
25%	51.000000	116.500000	30.000000	212500.000000	0.90000	
50%	60.000000	250.000000	38.000000	262000.000000	1.10000	
75%	70.000000	582.000000	45.000000	303500.000000	1.40000	
max	95.000000	7861.000000	80.000000	850000.000000	9.40000	

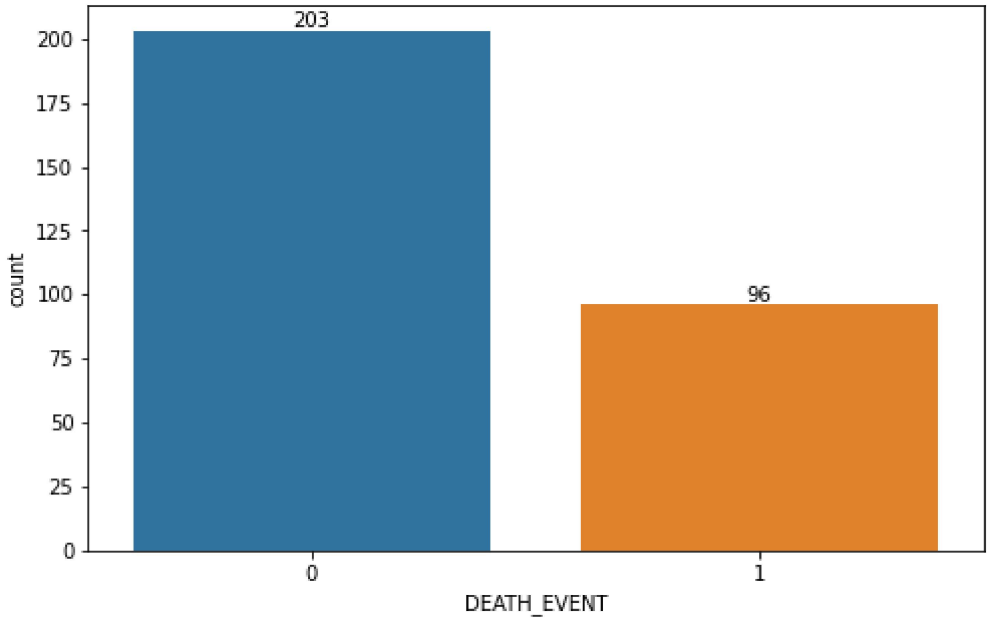
In [7]:
```
plt.figure(figsize=(10,5))
corrmat=df.corr()
sns.heatmap(corrmat, annot=True)
```

Out[7]: <AxesSubplot:>



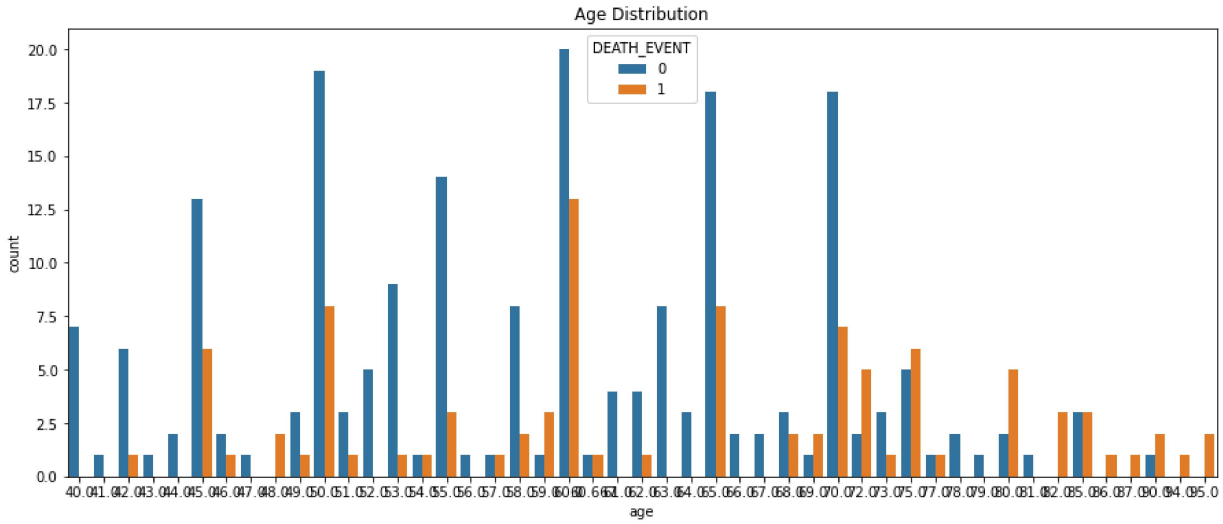
In [8]:
```
event_counts=df['DEATH_EVENT'].value_counts()

plt.figure(figsize=(8,5))
ax=sns.countplot(x=df['DEATH_EVENT'])
ax.bar_label(ax.containers[0])
```

Out[8]:
```
[Text(0, 0, '203'), Text(0, 0, '96')]
```



In [9]:
```
plt.figure(figsize=(15,6))
ax=sns.countplot(x=df['age'], data=df, hue=df['DEATH_EVENT'])
ax.set_title('Age Distribution')
```

Out[9]:
```
Text(0.5, 1.0, 'Age Distribution')
```



In [10]:
```
X=df.drop(columns='DEATH_EVENT', axis=1)
Y=df['DEATH_EVENT']
```

In [11]:
```
cols=list(X.columns)
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
X_scaled_df=pd.DataFrame(X_scaled, columns=cols)
X_scaled_df.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%
age	299.0	5.703353e-16	1.001676	-1.754448	-0.828124	-0.070223	0.771889
anaemia	299.0	1.009969e-16	1.001676	-0.871105	-0.871105	-0.871105	1.147968
creatinine_phosphokinase	299.0	0.000000e+00	1.001676	-0.576918	-0.480393	-0.342574	0.000166
diabetes	299.0	9.060014e-17	1.001676	-0.847579	-0.847579	-0.847579	1.179830
ejection_fraction	299.0	-3.267546e-17	1.001676	-2.038387	-0.684180	-0.007077	0.585389
high_blood_pressure	299.0	0.000000e+00	1.001676	-0.735688	-0.735688	-0.735688	1.359272
platelets	299.0	7.723291e-17	1.001676	-2.440155	-0.520870	-0.013908	0.411120
serum_creatinine	299.0	1.425838e-16	1.001676	-0.865509	-0.478205	-0.284552	0.005926
serum_sodium	299.0	-8.673849e-16	1.001676	-5.363206	-0.595996	0.085034	0.766064
sex	299.0	-8.911489e-18	1.001676	-1.359272	-1.359272	0.735688	0.735688
smoking	299.0	-1.188199e-17	1.001676	-0.687682	-0.687682	-0.687682	1.454161
time	299.0	-1.901118e-16	1.001676	-1.629502	-0.739000	-0.196954	0.938759

In [42]:
```
X_scaled_df.head()
```

Out[42]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
0	1.192945	-0.871105	0.000166	-0.847579	-1.530560	1.359272
1	-0.491279	-0.871105	7.514640	-0.847579	-0.007077	-0.735688
2	0.350833	-0.871105	-0.449939	-0.847579	-1.530560	-0.735688
3	-0.912335	1.147968	-0.486071	-0.847579	-1.530560	-0.735688
4	0.350833	1.147968	-0.435486	1.179830	-1.530560	-0.735688

In [12]:
```

x_train, x_test, y_train, y_test = train_test_split(X_scaled_df, Y, test_siz
```

In [14]:
```
from sklearn.svm import SVC
modelsvm=SVC(kernel='sigmoid').fit(x_train, y_train)
```

In [15]:
```
y_pred=modelsvm.predict(x_test)
y_pred
```

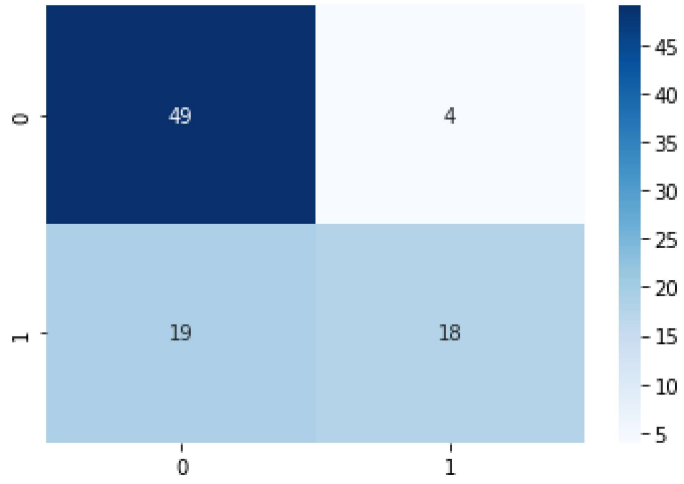
Out[15]:
```
array([0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0], dtype=int64)
```

In [16]:
```
print(accuracy_score(y_test, y_pred))

```

```
0.7444444444444445
```

In [17]:
```
sns.heatmap(confusion_matrix(y_test, y_pred),annot=True,cmap='Blues')

```

Out[17]:
```
<AxesSubplot:>
```



In [18]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.92      0.81        53
           1       0.82      0.49      0.61        37

    accuracy                           0.74        90
   macro avg       0.77      0.71      0.71        90
weighted avg       0.76      0.74      0.73        90
```

GridSearchCV is a hyperparameter tuning technique provided by scikit learn.

It exhaustively searches through all possible combinations of hyperparameter values you specify.

For each combination, it evaluates model performance using crossvalidation.

Finally, it picks the combination that gives the best performance.

In [49]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.01, 0.1, 1],
    'kernel': ['rbf', 'linear', 'poly']
}

grid = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid.fit(x_train, y_train)

print("Best Parameters:", grid.best_params_)

best_model = grid.best_estimator_
y_pred = best_model.predict(x_test)

print("Accuracy after tuning:", accuracy_score(y_test, y_pred))

```

```
Best Parameters: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Accuracy after tuning: 0.7777777777777778
```

C: Regularization parameter which controls the trade-off between correct classification and margin size.

gamma: Kernel coefficient (for rbf, poly, sigmoid). It controls how far the influence of a single training example reaches.

kernel: The type of kernel function to use (linear, rbf, or poly).

This creates 4 × 5 × 3 = 60 total combinations to try

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```