In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df=pd.read_csv("heart_failure_clinical_records_dataset.csv")
```

In [3]:

```
df
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	se
0	75.0	0	582	0	20	1	265000.00	1.9	130	
1	55.0	0	7861	0	38	0	263358.03	1.1	136	
2	65.0	0	146	0	20	0	162000.00	1.3	129	
3	50.0	1	111	0	20	0	210000.00	1.9	137	
4	65.0	1	160	1	20	0	327000.00	2.7	116	
...	...	...	...	...	...	...	...	...	...	
294	62.0	0	61	1	38	1	155000.00	1.1	143	
295	55.0	0	1820	0	38	0	270000.00	1.2	139	
296	45.0	0	2060	1	60	0	742000.00	0.8	138	
297	45.0	0	2413	0	38	0	140000.00	1.4	140	
298	50.0	0	196	0	45	0	395000.00	1.6	136	

299 rows × 13 columns

In [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    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 [5]:

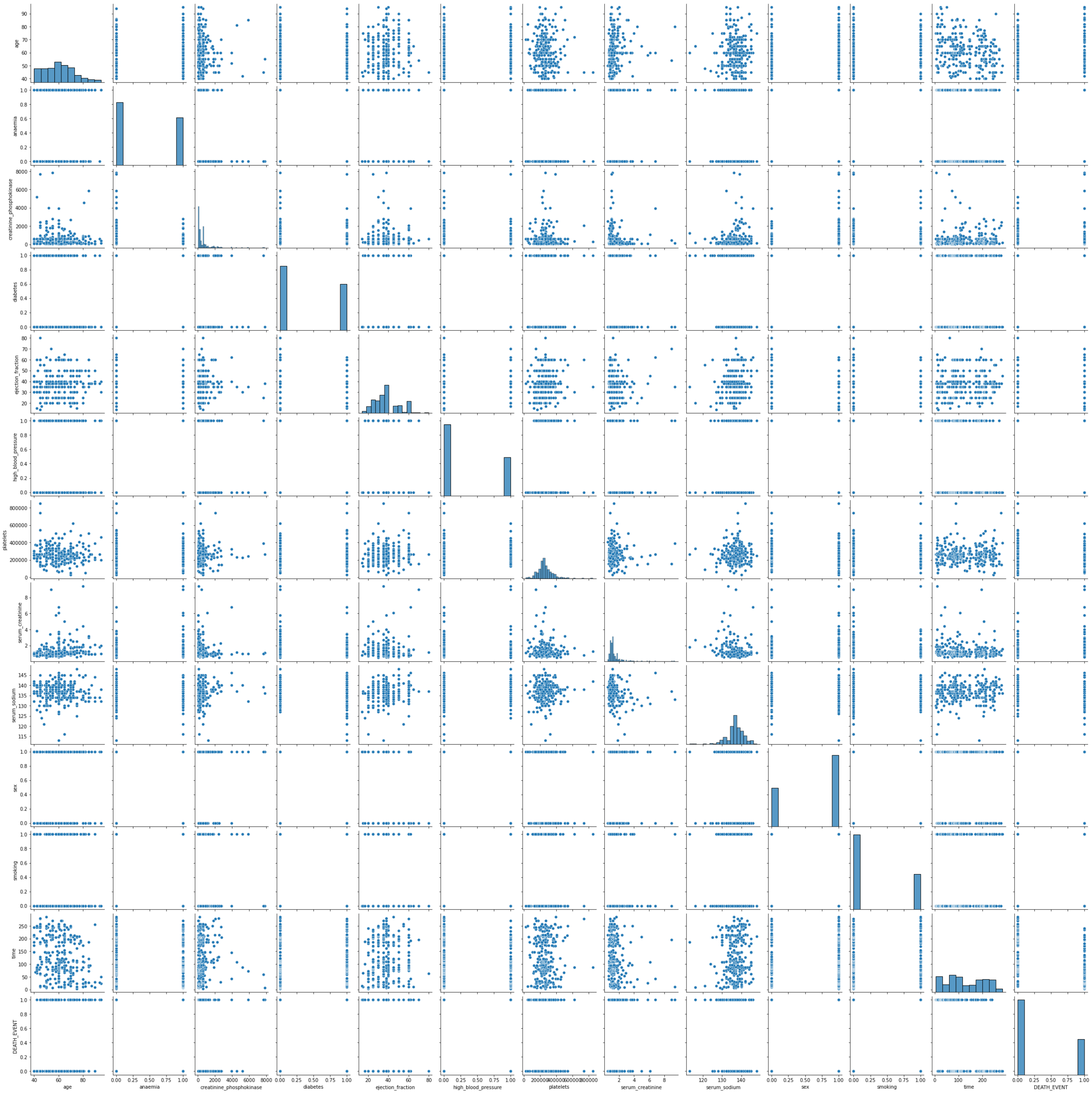
```
df.describe()
```

Out[5]:

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

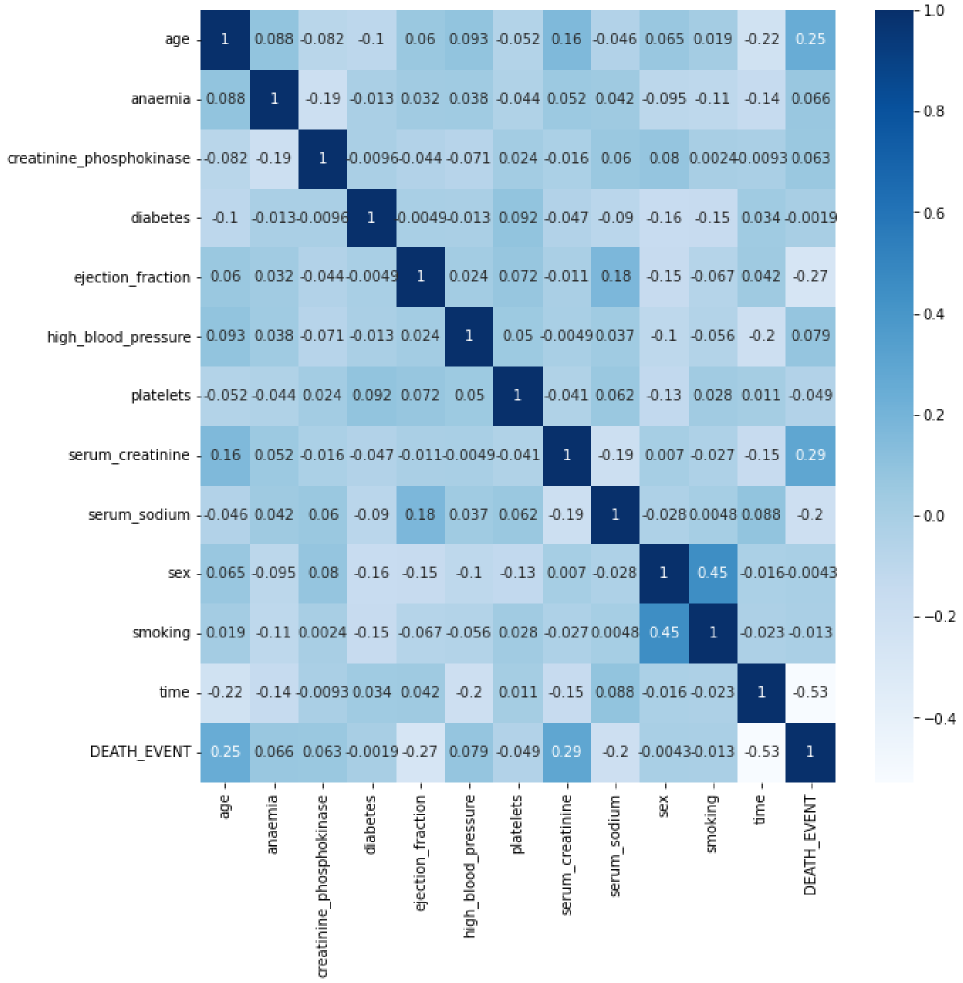
In [6]:

```
sns.pairplot(data=df)
plt.show()
```



In [7]:

```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),annot=True,cmap="Blues")
plt.show()
```



In [8]:

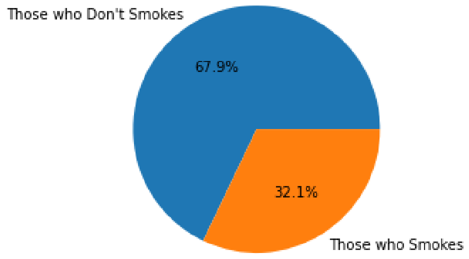
```
df["smoking"].value_counts()
```

Out[8]:

```
0    203
1     96
Name: smoking, dtype: int64
```

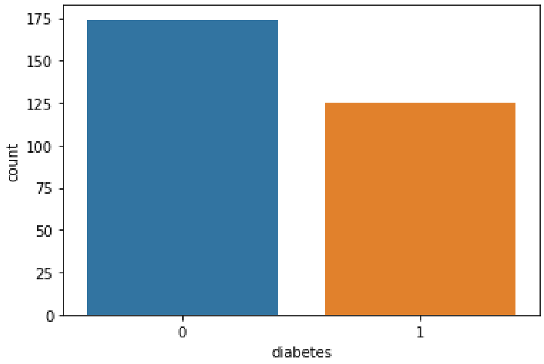
In [41]:

```
plt.pie(df["smoking"].value_counts(),labels=["Those who Don't Smokes","Those who Smokes"],autopct='%1.1f%%')
plt.show()
```



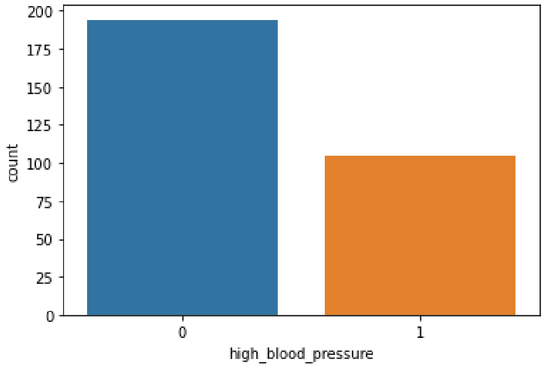
In [10]:

```
sns.countplot(data=df,x="diabetes")
plt.show()
```



In [42]:

```
sns.countplot(data=df,x="high_blood_pressure")
plt.show()
```



In [12]:

```
df.drop("time",axis=1,inplace=True)
```

In [13]:

```
x=df.iloc[:,[0,3,5,10]]
y=df.iloc[:,-1]
```

In [14]:

```
x
```

Out[14]:

	age	diabetes	high_blood_pressure	smoking
0	75.0	0	1	0
1	55.0	0	0	0
2	65.0	0	0	1
3	50.0	0	0	0
4	65.0	1	0	0
...	...	...	...	...
294	62.0	1	1	1
295	55.0	0	0	0
296	45.0	1	0	0
297	45.0	0	0	1
298	50.0	0	0	1

299 rows × 4 columns

In [15]:

```
y
```

Out[15]:

```
0      1
1      1
2      1
3      1
4      1
      ..
294    0
295    0
296    0
297    0
298    0
Name: DEATH_EVENT, Length: 299, dtype: int64
```

In [16]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x=sc.fit_transform(x)
```

In [17]:

```
x
```

Out[17]:

```
array([[ 1.19294523, -0.84757938,  1.35927151, -0.68768191],
       [-0.49127928, -0.84757938, -0.73568819, -0.68768191],
       [ 0.35083298, -0.84757938, -0.73568819,  1.4541607 ],
       ...,
       [-1.33339153,  1.1798305 , -0.73568819, -0.68768191],
       [-1.33339153, -0.84757938, -0.73568819,  1.4541607 ],
       [-0.9123354 , -0.84757938, -0.73568819,  1.4541607 ]])
```

In [18]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

In [19]:

```
xtrain
```

Out[19]:

```
array([[-0.99654663, -0.84757938,  1.35927151, -0.68768191],
       [ 0.7718891 ,  1.1798305 ,  1.35927151,  1.4541607 ],
       [ 1.19294523, -0.84757938,  1.35927151, -0.68768191],
       [-0.07022315,  1.1798305 , -0.73568819, -0.68768191],
       [-0.9123354 ,  1.1798305 , -0.73568819, -0.68768191],
       [-0.74391295,  1.1798305 , -0.73568819,  1.4541607 ],
       [ 2.03505748, -0.84757938, -0.73568819, -0.68768191],
       [-1.33339153,  1.1798305 , -0.73568819, -0.68768191],
       [-0.9123354 ,  1.1798305 ,  1.35927151,  1.4541607 ],
       [-0.49127928, -0.84757938,  1.35927151, -0.68768191],
       [-1.58602521, -0.84757938, -0.73568819,  1.4541607 ],
       [-1.33339153, -0.84757938, -0.73568819,  1.4541607 ],
       [-0.65970173, -0.84757938, -0.73568819,  1.4541607 ],
       [-0.49127928, -0.84757938, -0.73568819, -0.68768191],
       [-0.15443437, -0.84757938,  1.35927151,  1.4541607 ],
       [ 0.18241053,  1.1798305 , -0.73568819, -0.68768191],
       [ 0.35083298, -0.84757938, -0.73568819, -0.68768191],
       [ 1.19294523,  1.1798305 , -0.73568819, -0.68768191],
```

In [20]:

```
ytrain
```

Out[20]:

```
14     0
210    0
236    0
44     1
163    1
      ..
203    0
255    0
72     1
235    0
37     1
Name: DEATH_EVENT, Length: 209, dtype: int64
```

In [21]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [22]:

```
from sklearn.metrics import classification_report
```

In [23]:

```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    train=model.score(xtrain,ytrain)
    test=model.score(xtest,ytest)

    print(f"Training Acc:- {train}\n Testing Acc:- {test}")
    print(classification_report(ytest,ypred))
    return model
```

In [24]:

```
knn=mymodel(KNeighborsClassifier())
knn
```

```
Training Acc:- 0.7607655502392344
 Testing Acc:- 0.5777777777777777
              precision    recall  f1-score   support

           0       0.70      0.72      0.71        64
           1       0.25      0.23      0.24        26

    accuracy                           0.58        90
   macro avg       0.47      0.47      0.47        90
weighted avg       0.57      0.58      0.57        90
```

Out[24]:

```
KNeighborsClassifier()
```

In [25]:

```
mymodel(DecisionTreeClassifier())
```

```
Training Acc:- 0.8421052631578947
 Testing Acc:- 0.6444444444444445
              precision    recall  f1-score   support

           0       0.73      0.80      0.76        64
           1       0.35      0.27      0.30        26

    accuracy                           0.64        90
   macro avg       0.54      0.53      0.53        90
weighted avg       0.62      0.64      0.63        90
```

Out[25]:

```
DecisionTreeClassifier()
```

In [26]:

```
mymodel(RandomForestClassifier())
```

```
Training Acc:- 0.8421052631578947
 Testing Acc:- 0.6111111111111112
              precision    recall  f1-score   support

           0       0.72      0.75      0.73        64
           1       0.30      0.27      0.29        26

    accuracy                           0.61        90
   macro avg       0.51      0.51      0.51        90
weighted avg       0.60      0.61      0.60        90
```

Out[26]:

```
RandomForestClassifier()
```

In [27]:

```
from sklearn.model_selection import GridSearchCV
```

In [28]:

```
param={
   "n_neighbors":range(1,100),
    "leaf_size":range(1,100,10)
}
```

In [29]:

```
grid=GridSearchCV(KNeighborsClassifier(),param,verbose=2)
grid.fit(xtrain,ytrain)
```

```
Fitting 5 folds for each of 990 candidates, totalling 4950 fits
[CV] END ........................leaf_size=1, n_neighbors=1; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=1; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=1; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=1; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=1; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=2; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=2; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=2; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=2; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=2; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=3; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=3; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=3; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=3; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=3; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=4; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=4; total time=   0.0s
[CV] END ........................leaf_size=1, n_neighbors=4; total time=   0.0s
```

In [30]:

```
grid.best_params_
```

Out[30]:

```
{'leaf_size': 51, 'n_neighbors': 5}
```

In [31]:

```
grid.best_score_
```

Out[31]:

```
0.6844367015098722
```

In [32]:

```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

In [33]:

```
Ada=mymodel(AdaBoostClassifier())
Ada
```

```
Training Acc:- 0.7081339712918661
 Testing Acc:- 0.7111111111111111
              precision    recall  f1-score   support

           0       0.73      0.94      0.82        64
           1       0.50      0.15      0.24        26

    accuracy                           0.71        90
   macro avg       0.62      0.55      0.53        90
weighted avg       0.66      0.71      0.65        90
```

Out[33]:

```
AdaBoostClassifier()
```

In [34]:

```
mymodel(GradientBoostingClassifier())
```

```
Training Acc:- 0.7894736842105263
 Testing Acc:- 0.6555555555555556
              precision    recall  f1-score   support

           0       0.73      0.83      0.77        64
           1       0.35      0.23      0.28        26

    accuracy                           0.66        90
   macro avg       0.54      0.53      0.53        90
weighted avg       0.62      0.66      0.63        90
```

Out[34]:

```
GradientBoostingClassifier()
```

In [35]:

```
df.head(5)
```

Out[35]:

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

In [36]:

```
def predictDeathEvent():
    age=float(input("Enter the Age of the person:- "))
    diabetes=int(input("Enter If you have Diabetes or not:- "))
    high_blood_pressure=int(input("Enter your If you have High Blood Pressure:- "))
    smoking=int(input("Enter If you Smoke or not:- "))

    newob=[[age,diabetes,high_blood_pressure,smoking]]
    ypred=Ada.predict(newob)
    return ypred[0]
```

In [37]:

```
predictDeathEvent()
```

```
Enter the Age of the person:- 20
Enter If you have Diabetes or not:- 1
Enter your If you have High Blood Pressure:- 1
Enter If you Smoke or not:- 1
```

Out[37]:

```
1
```

In [ ]: