HEART FAILURE PREDICTIONS

In This dataset we are going to create a model on predicting the heart failure.

Importing libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

In [2]:
```
heart=pd.read_csv("heart_failure.csv")
heart.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 [3]:
```
heart.describe()
```

Out[3]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_
count	299.000000	299.000000	299.000000	299.000000	299.000000	2
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 [4]:
```
heart.head(5)
```

Out[4]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	plat
0	75.0	0	582	0	20	1	2650
1	55.0	0	7861	0	38	0	2633
2	65.0	0	146	0	20	0	1620
3	50.0	1	111	0	20	0	2100
4	65.0	1	160	1	20	0	3270

Sex - Gender of patient Male = 1, Female =0 Age - Age of patient Diabetes - 0 = No, 1 = Yes Anaemia - 0 = No, 1 = Yes High_blood_pressure - 0 = No, 1 = Yes Smoking - 0 = No, 1 = Yes DEATH_EVENT - 0 = No, 1 = Yes

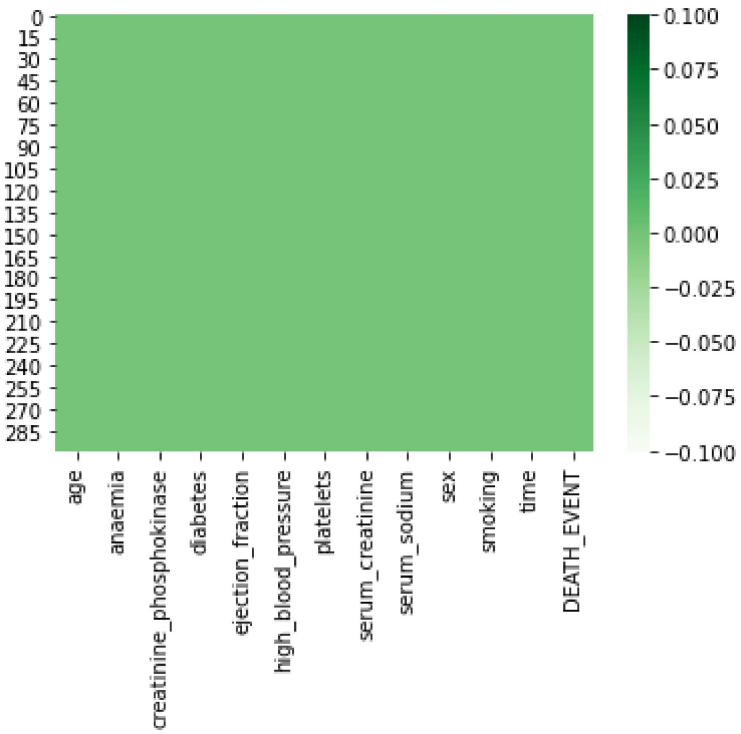
Now we are going to finding the null values in the data set

In [5]:
```
heart.isnull().sum()
```

Out[5]:
```
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 [6]:
```
sns.heatmap(heart.isnull(),cmap="Greens")
```

Out[6]: <AxesSubplot:>



In [7]:
```
heart["platelets"]=heart["platelets"].astype(int)
heart["age"]=heart["age"].astype(int)
heart["anaemia"]=heart["anaemia"].astype(int)
heart["platelets"]=heart["platelets"].astype(int)
heart
```

Out[7]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	pla
0	75	0	582	0	20	1	2(
1	55	0	7861	0	38	0	2(
2	65	0	146	0	20	0	1(
3	50	1	111	0	20	0	2
4	65	1	160	1	20	0	3
...	...	...	...	...	...	...	
294	62	0	61	1	38	1	1
295	55	0	1820	0	38	0	2
296	45	0	2060	1	60	0	7
297	45	0	2413	0	38	0	1
298	50	0	196	0	45	0	3

299 rows × 13 columns

we notice that there is no null in the dataset and no categorial values

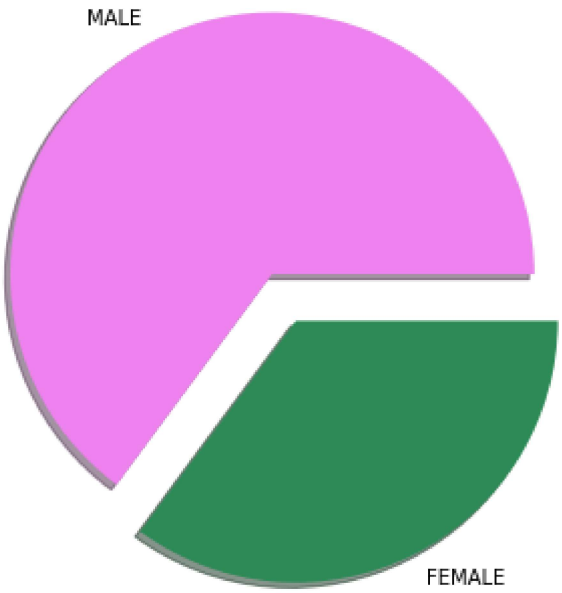
Now we use the visualization techniques to visualize our data and further fitting into models

In [8]:
```
print("HEART FAILED:",heart.DEATH_EVENT.value_counts()[1])
print("HEART NOT FAILED:",heart.DEATH_EVENT.value_counts()[0])
```

```
HEART FAILED: 96
HEART NOT FAILED: 203
```

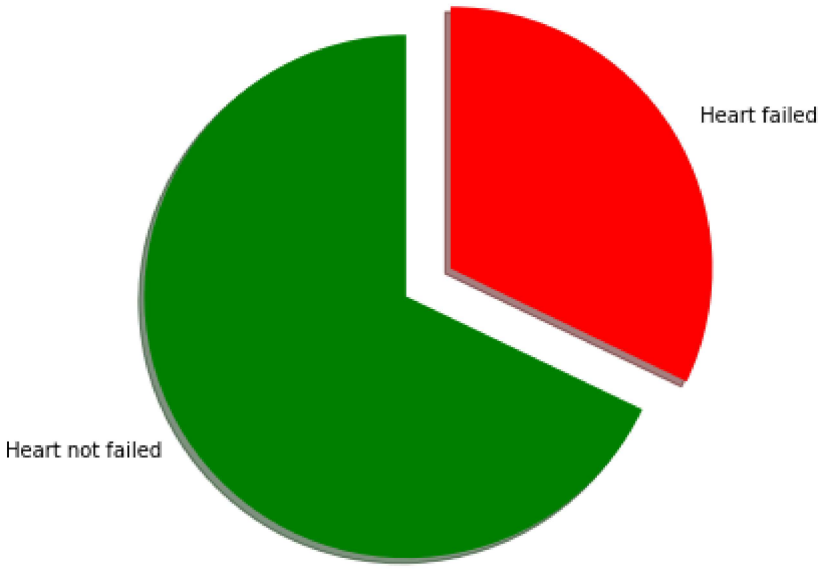
In [9]:
```
fig, ax = plt.subplots(figsize=(6,6))

plt.pie(x=heart["sex"].value_counts(),colors=["violet","seagreen"],explode = ( 0.
```

Out[9]:
```
([<matplotlib.patches.Wedge at 0x1e8d58bf550>,
  <matplotlib.patches.Wedge at 0x1e8d58bfc70>],
 [Text(-0.5408530634991104, 1.071203978569734, 'MALE'),
  Text(0.5408531637924622, -1.0712039279314114, 'FEMALE')])
```

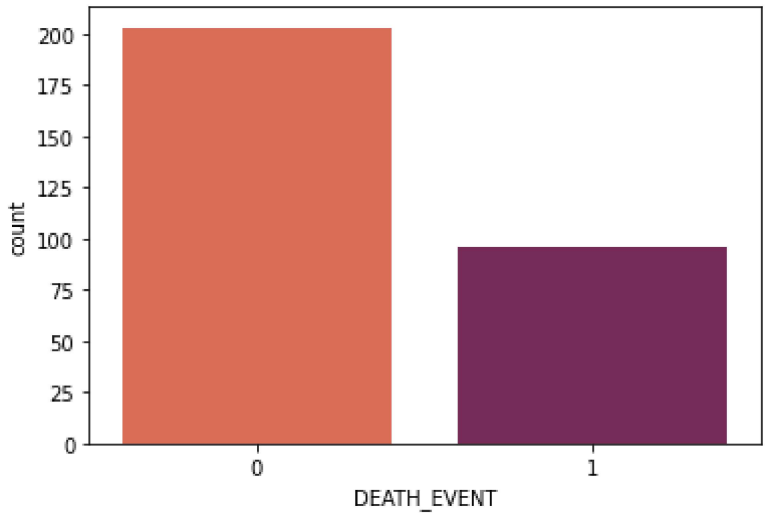


In [10]:
```
fig, ax = plt.subplots(figsize=(6,6))

plt.pie(x=heart["DEATH_EVENT"].value_counts(),colors=["green","red"],explode = (
```

Out[10]:
```
([<matplotlib.patches.Wedge at 0x1e8d590c940>,
  <matplotlib.patches.Wedge at 0x1e8d59190a0>],
 [Text(-1.0153497129885487, -0.639581863668813, 'Heart not failed'),
  Text(1.0153497728705203, 0.6395817686049088, ' Heart failed')])
```

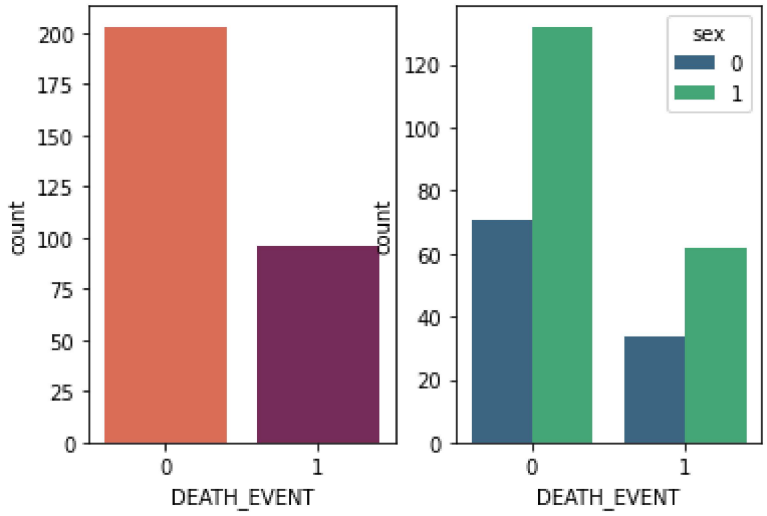


In [11]:
```
sns.countplot(x='DEATH_EVENT',data=heart,palette='rocket_r')
plt.show()
```



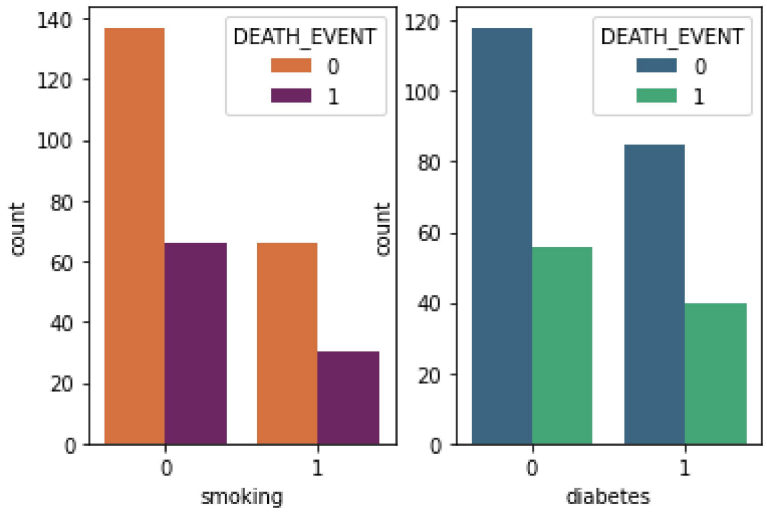
In [12]:
```
plt.subplot(1,2,1)
sns.countplot(x='DEATH_EVENT',data=heart,palette='rocket_r')

plt.subplot(1,2,2)
sns.countplot(x='DEATH_EVENT',hue="sex",data=heart,palette='viridis')
```

Out[12]: <AxesSubplot:xlabel='DEATH_EVENT', ylabel='count'>



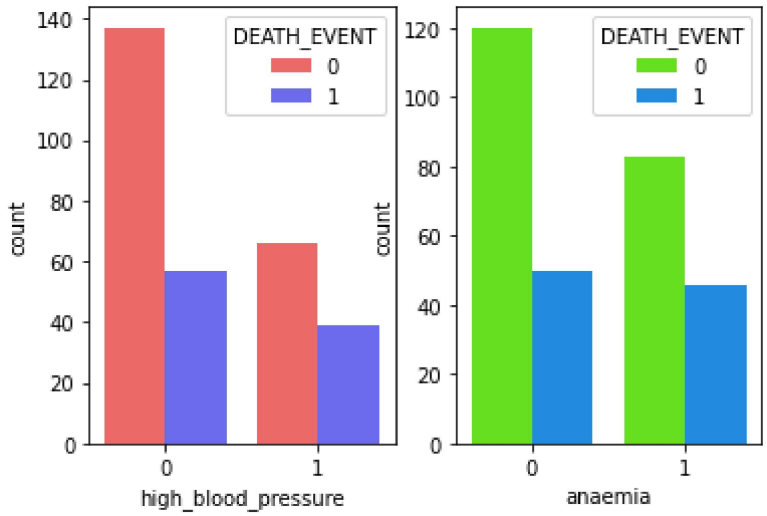
In [13]:
```
plt.subplot(1,2,1)
sns.countplot(x='smoking',hue="DEATH_EVENT",data=heart,palette="inferno_r")

plt.subplot(1,2,2)
sns.countplot(x='diabetes',hue="DEATH_EVENT",data=heart,palette='viridis')
```

Out[13]: <AxesSubplot:xlabel='diabetes', ylabel='count'>



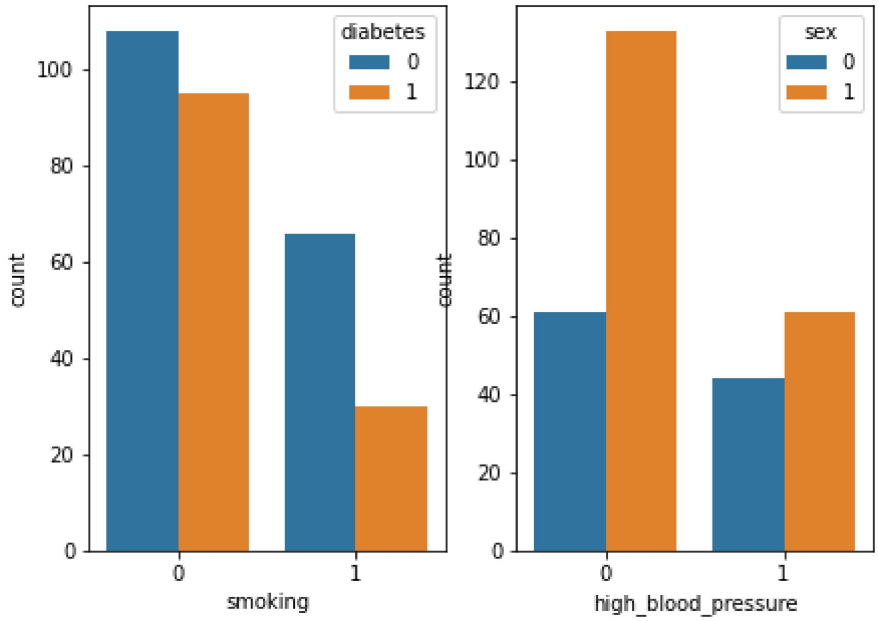
In [14]:
```
plt.subplot(1,2,1)
sns.countplot(x='high_blood_pressure',hue="DEATH_EVENT",data=heart,palette='seism

plt.subplot(1,2,2)
sns.countplot(x='anaemia',hue="DEATH_EVENT",data=heart,palette='gist_rainbow')
```

Out[14]: <AxesSubplot:xlabel='anaemia', ylabel='count'>



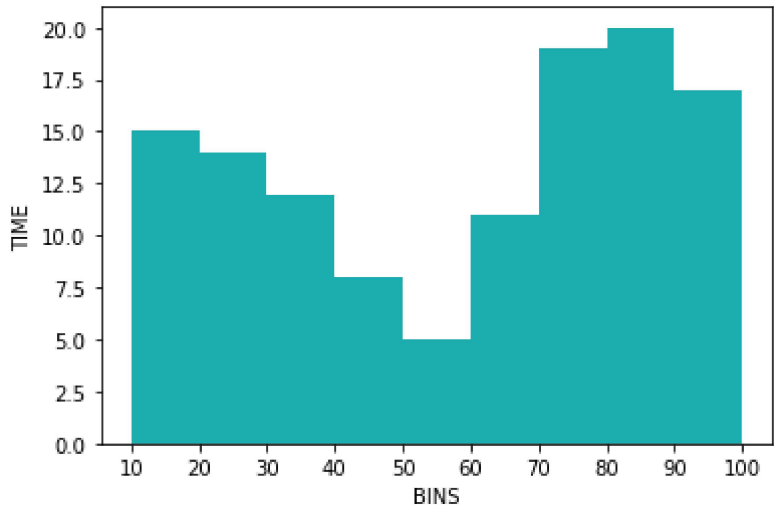
In [15]:
```
plt.figure(figsize=(7,5))
plt.subplot(1,2,1)
sns.countplot('smoking',hue='diabetes',data=heart)

plt.subplot(1,2,2)
sns.countplot('high_blood_pressure',hue='sex',data=heart)
```

Out[15]: <AxesSubplot:xlabel='high_blood_pressure', ylabel='count'>

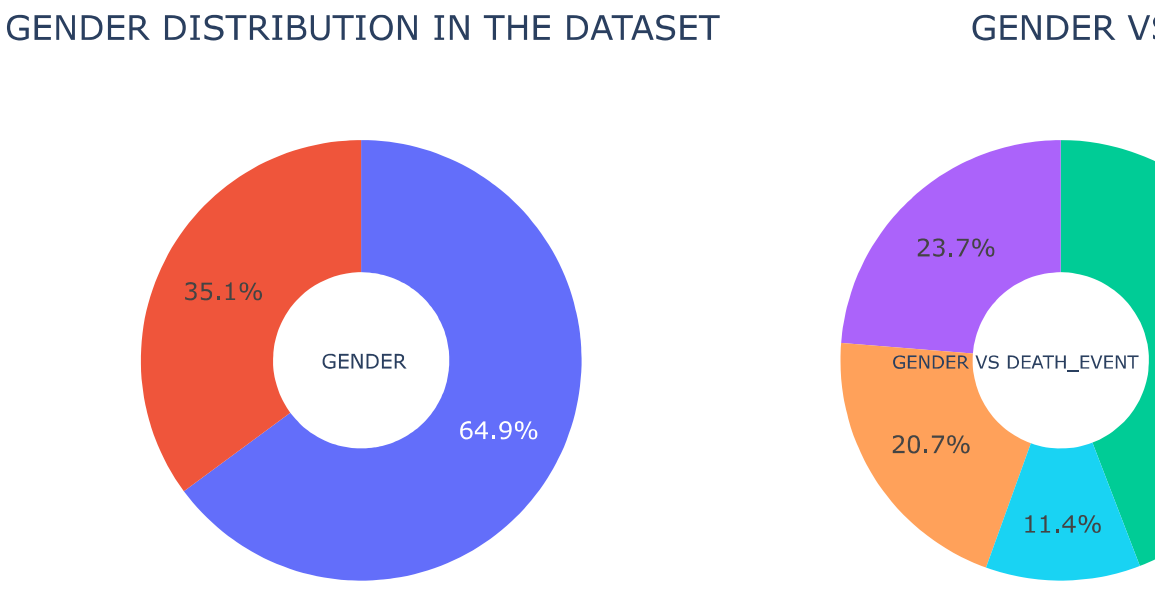


In [16]:
```
bins=[10,20,30,40,50,60,70,80,90,100]
plt.hist(heart.time,bins=bins,color='#1aadad')
plt.xticks(bins)
plt.xlabel("BINS")
plt.ylabel("TIME")
plt.show
```

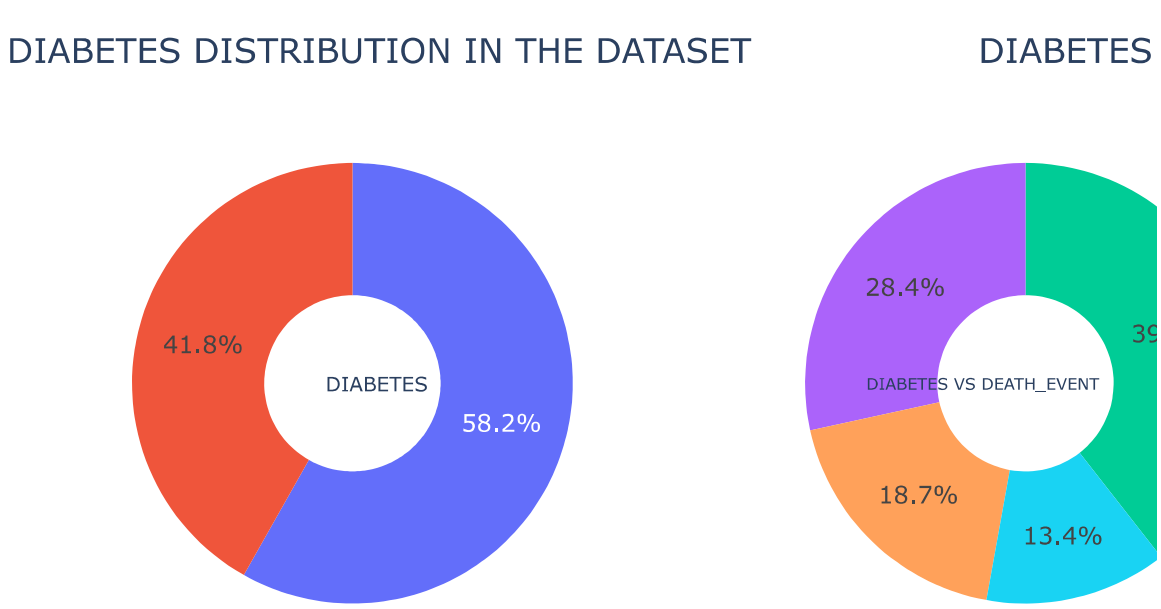
Out[16]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



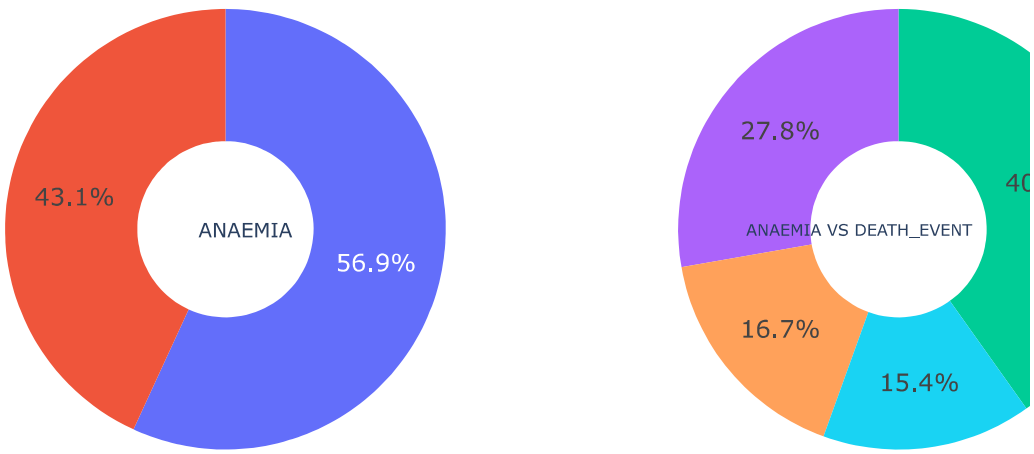
In [17]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

d1 = heart[(heart["DEATH_EVENT"]==0) & (heart["sex"]==1)]
d2 = heart[(heart["DEATH_EVENT"]==1) & (heart["sex"]==1)]
d3 = heart[(heart["DEATH_EVENT"]==0) & (heart["sex"]==0)]
d4 = heart[(heart["DEATH_EVENT"]==1) & (heart["sex"]==0)]

label1 = ["Male","Female"]
label2 = ['Male - Survived','Male - Died', "Female -  Survived", "Female - Died"]
values1 = [(len(d1)+len(d2)), (len(d3)+len(d4))]
values2 = [len(d1),len(d2),len(d3),len(d4)]

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]
fig.add_trace(go.Pie(labels=label1, values=values1, name="GENDER"),
              1, 1)
fig.add_trace(go.Pie(labels=label2, values=values2, name="GENDER VS DEATH_EVENT")
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent")

fig.update_layout(
    title_text="GENDER DISTRIBUTION IN THE DATASET  \
                  GENDER VS DEATH_EVENT",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='GENDER', x=0.19, y=0.5, font_size=10, showarrow=False
                 dict(text='GENDER VS DEATH_EVENT', x=0.84, y=0.5, font_size=9, s
    autosize=False,width=900, height=400, paper_bgcolor="white")

fig.show()
```



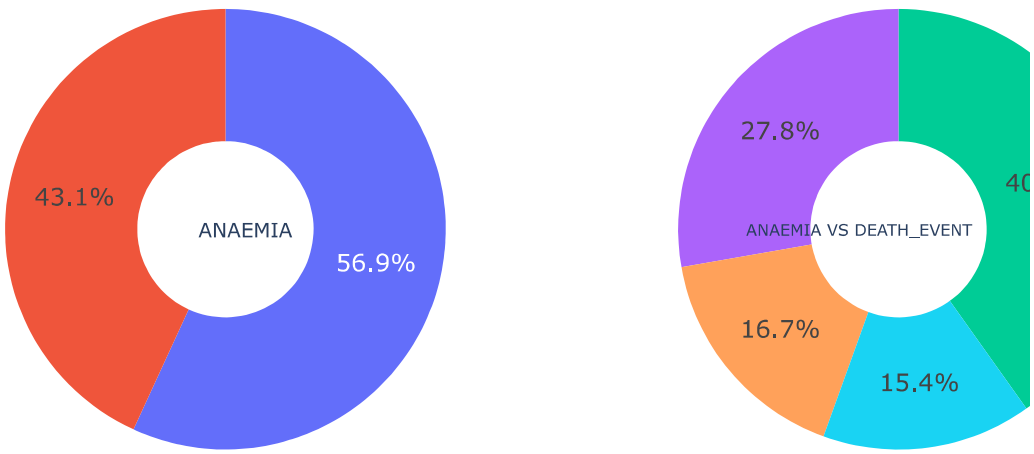
In [18]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

d1 =heart[(heart["DEATH_EVENT"]==0) & (heart["diabetes"]==0)]
d2 = heart[(heart["DEATH_EVENT"]==0) & (heart["diabetes"]==1)]
d3 = heart[(heart["DEATH_EVENT"]==1) & (heart["diabetes"]==0)]
d4 = heart[(heart["DEATH_EVENT"]==1) & (heart["diabetes"]==1)]

label1 = ["No Diabetes","Diabetes"]
label2 = ['No Diabetes - Survived','Diabetes - Survived', "No Diabetes -  Died",
values1 = [(len(d1)+len(d3)), (len(d2)+len(d4))]
values2 = [len(d1),len(d2),len(d3),len(d4)]

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]
fig.add_trace(go.Pie(labels=label1, values=values1, name="DIABETES"),
              1, 1)
fig.add_trace(go.Pie(labels=label2, values=values2, name="DIABETES VS DEATH_EVENT
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent")

fig.update_layout(
    title_text="DIABETES DISTRIBUTION IN THE DATASET \
                  DIABETES VS DEATH_EVENT",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='DIABETES', x=0.20, y=0.5, font_size=10, showarrow=Fal
                 dict(text='DIABETES VS DEATH_EVENT', x=0.84, y=0.5, font_size=8,
    autosize=False,width=900, height=400, paper_bgcolor="white")
fig.show()
```



In [19]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

d1 = heart[(heart["DEATH_EVENT"]==0) & (heart["anaemia"]==0)]
d2 = heart[(heart["DEATH_EVENT"]==1) & (heart["anaemia"]==0)]
d3 = heart[(heart["DEATH_EVENT"]==0) & (heart["anaemia"]==1)]
d4 = heart[(heart["DEATH_EVENT"]==1) & (heart["anaemia"]==1)]

label1 = ["No Anaemia","Anaemia"]
label2 = ['No Anaemia - Survived','No Anaemia - Died', "Anaemia -  Survived", "Ar
values1 = [(len(d1)+len(d2)), (len(d3)+len(d4))]
values2 = [len(d1),len(d2),len(d3),len(d4)]

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]
fig.add_trace(go.Pie(labels=label1, values=values1, name="ANAEMIA"),
              1, 1)
fig.add_trace(go.Pie(labels=label2, values=values2, name="ANAEMIA VS DEATH_EVENT"
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent")

fig.update_layout(
    title_text="ANAEMIA DISTRIBUTION IN THE DATASET \
                  ANAEMIA VS DEATH_EVENT",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='ANAEMIA', x=0.20, y=0.5, font_size=10, showarrow=Fals
                 dict(text='ANAEMIA VS DEATH_EVENT', x=0.84, y=0.5, font_size=8,
    autosize=False,width=900, height=400, paper_bgcolor="white")
fig.show()
```

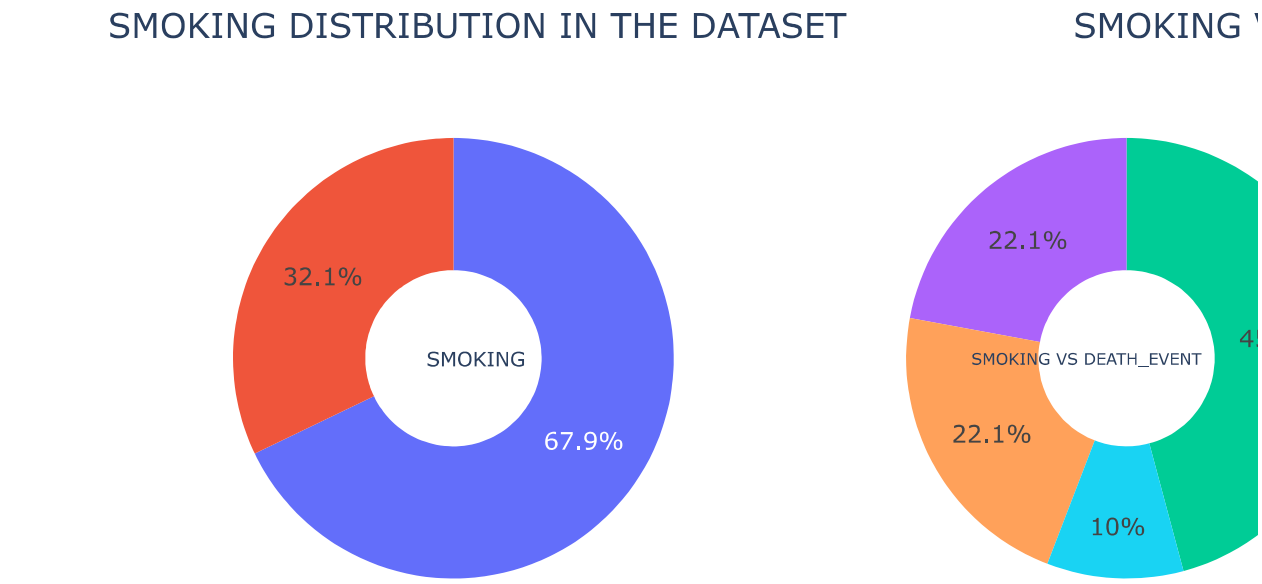


In [20]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

d1 = heart[(heart["DEATH_EVENT"]==0) & (heart["anaemia"]==0)]
d2 = heart[(heart["DEATH_EVENT"]==1) & (heart["anaemia"]==0)]
d3 = heart[(heart["DEATH_EVENT"]==0) & (heart["anaemia"]==1)]
d4 = heart[(heart["DEATH_EVENT"]==1) & (heart["anaemia"]==1)]

label1 = ["No Anaemia","Anaemia"]
label2 = ['No Anaemia - Survived','No Anaemia - Died', "Anaemia -  Survived", "Ar
values1 = [(len(d1)+len(d2)), (len(d3)+len(d4))]
values2 = [len(d1),len(d2),len(d3),len(d4)]

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]
fig.add_trace(go.Pie(labels=label1, values=values1, name="ANAEMIA"),
              1, 1)
fig.add_trace(go.Pie(labels=label2, values=values2, name="ANAEMIA VS DEATH_EVENT"
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent")

fig.update_layout(
    title_text="ANAEMIA DISTRIBUTION IN THE DATASET \
                  ANAEMIA VS DEATH_EVENT",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='ANAEMIA', x=0.20, y=0.5, font_size=10, showarrow=Fals
                 dict(text='ANAEMIA VS DEATH_EVENT', x=0.84, y=0.5, font_size=8,
    autosize=False,width=900, height=400, paper_bgcolor="white")
fig.show()
```

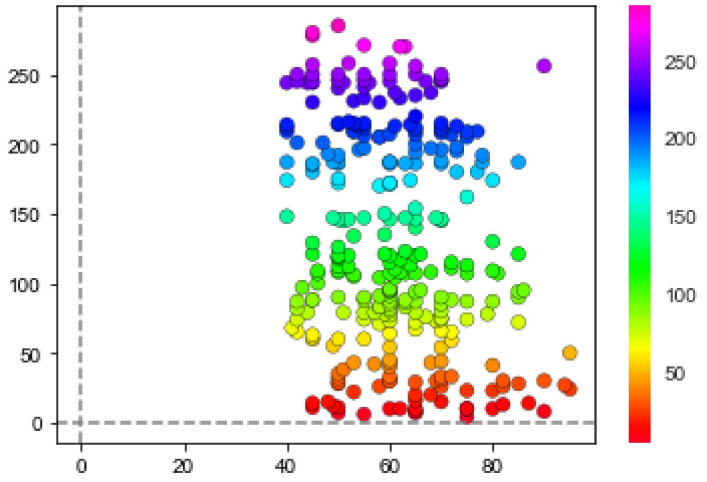


In [ ]:

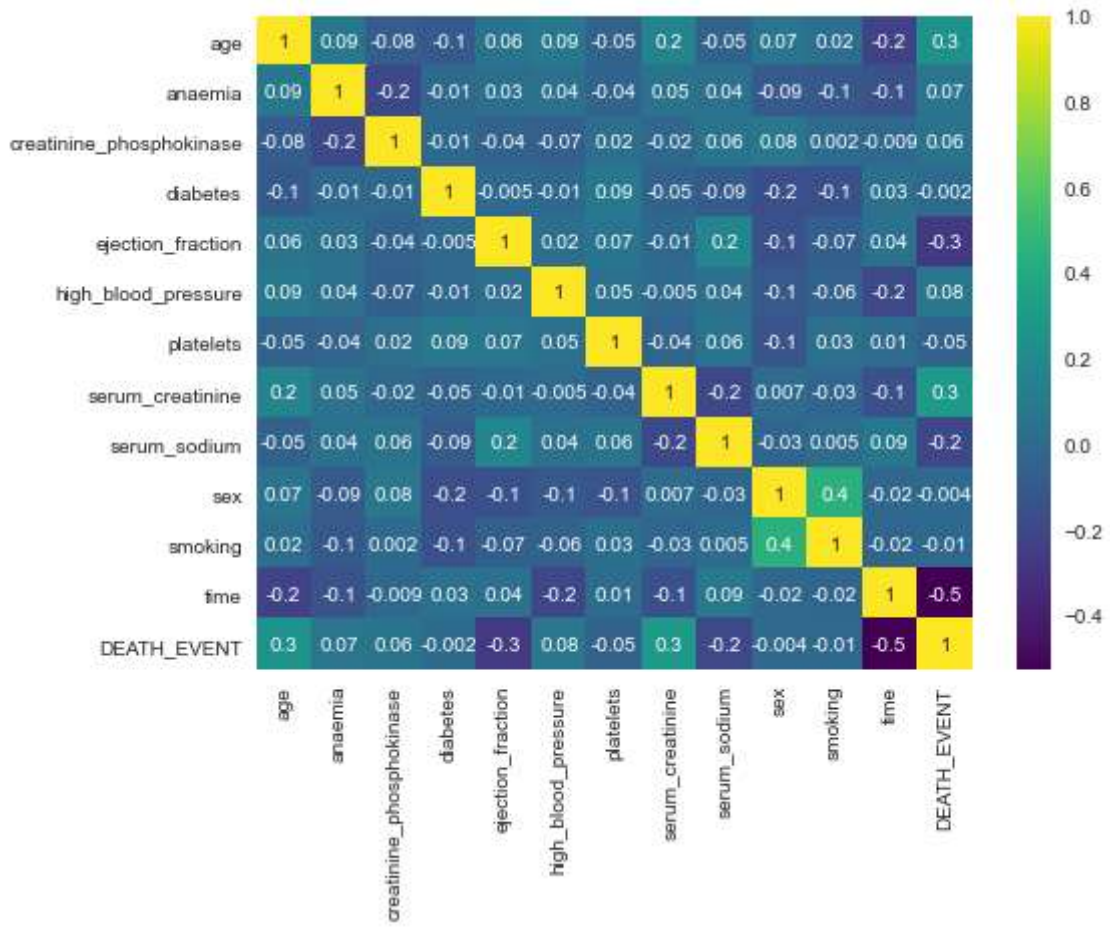
In [21]:
```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

d1 = heart[(heart["DEATH_EVENT"]==0) & (heart["smoking"]==0)]
d2 = heart[(heart["DEATH_EVENT"]==1) & (heart["smoking"]==0)]
d3 = heart[(heart["DEATH_EVENT"]==0) & (heart["smoking"]==1)]
d4 = heart[(heart["DEATH_EVENT"]==1) & (heart["smoking"]==1)]

label1 = ["No Smoking","Smoking"]
label2 = ['No Smoking - Survived','No Smoking - Died', "Smoking - Survived", "Smo
values1 = [(len(d1)+len(d2)), (len(d3)+len(d4))]
values2 = [len(d1),len(d2),len(d3),len(d4)]

# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]
fig.add_trace(go.Pie(labels=label1, values=values1, name="SMOKING"),
              1, 1)
fig.add_trace(go.Pie(labels=label2, values=values2, name="SMOKING VS DEATH_EVENT"
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent")

fig.update_layout(
    title_text="SMOKING DISTRIBUTION IN THE DATASET \
                  SMOKING VS DEATH_EVENT",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='SMOKING', x=0.20, y=0.5, font_size=10, showarrow=Fals
                 dict(text='SMOKING VS DEATH_EVENT', x=0.84, y=0.5, font_size=8,
    autosize=False,width=900, height=400, paper_bgcolor="white")
fig.show()
```



In [22]:
```
plt.axvline(0,c=(.5,.5,.5), ls='--')
plt.axhline(0,c=(.5,.5,.5), ls='--')
plt.style.use('seaborn')
plt.scatter(heart.age,heart.time,c=heart.time , cmap="gist_rainbow",edgecolor='k'
plt.colorbar();
```

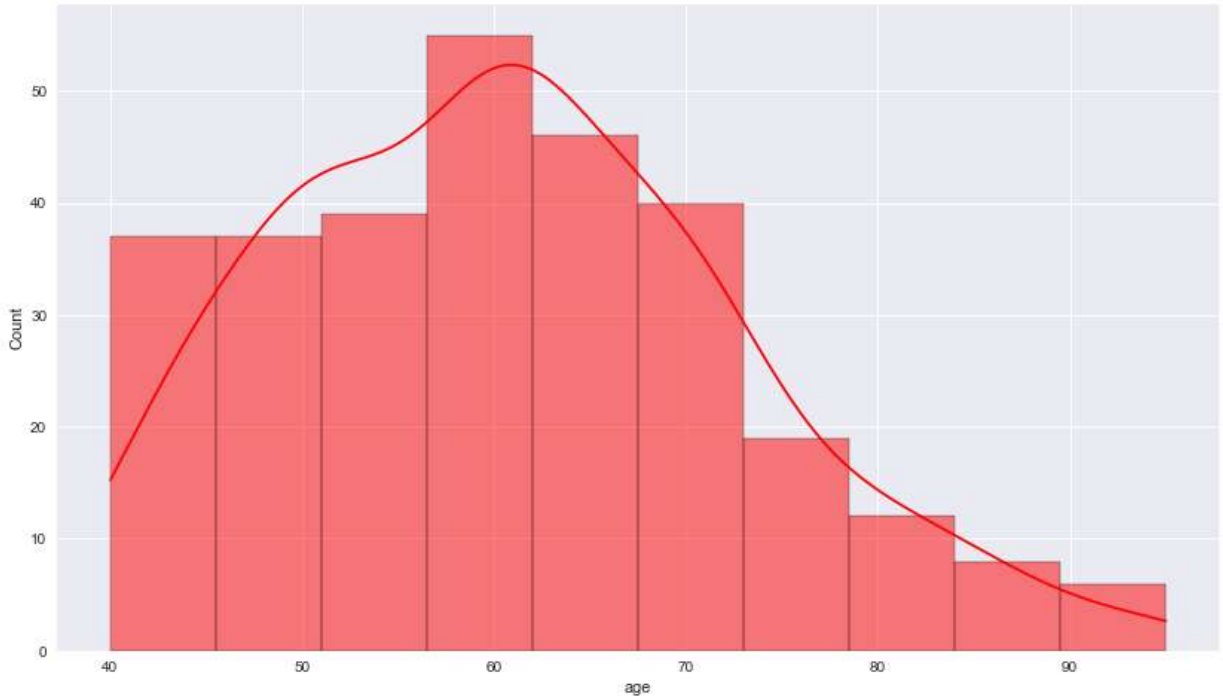


In [23]:
```
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(heart.corr(), annot=True, fmt='.1g', cmap='viridis');
```



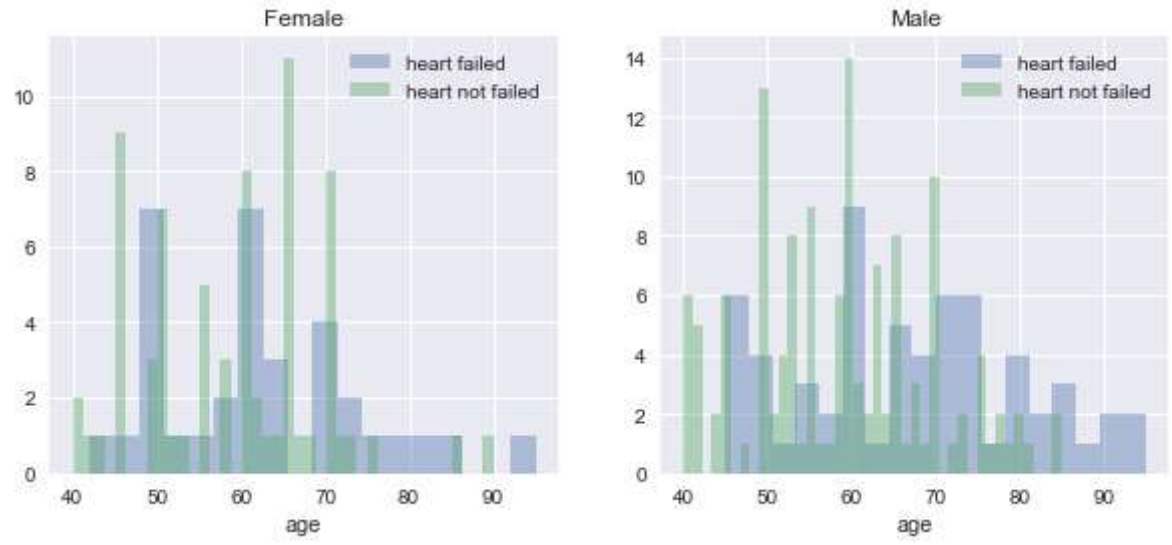
In [24]:
```
fig, ax = plt.subplots(figsize=(14,8))
sns.histplot(x=heart["age"], kde=True, color='red')
```

Out[24]: <AxesSubplot:xlabel='age', ylabel='Count'>



In [25]:
```
healthy = 'heart not failed'
unhealthy = 'heart failed'
fig, axes = plt.subplots(nrows=1, ncols=2,figsize=(10, 4))
women = heart[heart['sex']==0]
men = heart[heart['sex']==1]
ax = sns.distplot(women[women['DEATH_EVENT']==1].age.dropna(), bins=18, label = u
ax = sns.distplot(women[women['DEATH_EVENT']==0].age.dropna(), bins=40, label =he
ax.legend()
ax.set_title('Female')
ax = sns.distplot(men[men['DEATH_EVENT']==1].age.dropna(), bins=18, label = unhea
ax = sns.distplot(men[men['DEATH_EVENT']==0].age.dropna(), bins=40, label =health
ax.legend()
ax.set_title('Male')
```

Out[25]:
```
Text(0.5, 1.0, 'Male')
```



NOW BUILD THE MACHINE LEARNING MODLES

In [26]:
```
X = heart.drop("DEATH_EVENT", axis=1)
X.head()
```

Out[26]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platel
0	75	0	582	0	20	1	265(
1	55	0	7861	0	38	0	263
2	65	0	146	0	20	0	162(
3	50	1	111	0	20	0	210(
4	65	1	160	1	20	0	327(

In [27]:
```
Y = heart["DEATH_EVENT"]
Y.head()
```

Out[27]:
```
0    1
1    1
2    1
3    1
4    1
Name: DEATH_EVENT, dtype: int64
```

Spliting the data

In [28]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y,train_size=0.8,test_size
```

In [29]:
```
print("x_train=",X_train)
print("y_train=",y_train)
print("x_test=",X_test)
print("y_test=",y_test)
```

```
x_train=      age  anaemia  creatinine_phosphokinase  diabetes  ejection_frac
tion  \
161   45        1                       130         0               35
260   55        0                        66         0               40
218   68        1                      1021         1               35
123   60        1                       582         0               30
39    60        0                       235         1               38
..   ...      ...                       ...       ...              ...
203   60        0                        59         0               25
255   52        1                       191         1               30
72    85        0                      5882         0               35
235   77        1                       109         0               50
37    82        1                       855         1               50

     high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
161                    0     174000               0.8           139    1
260                    0     203000               1.0           138    1
218                    0     271000               1.1           134    1
123                    1     127000               0.9           145    0
```

In [30]:
```
len(X_train), len(X_test),len(y_train), len(y_test)
```

Out[30]:
```
(239, 60, 239, 60)
```

Logistic regression

In [31]:
```
logr = LogisticRegression()
logr.fit(X_train, y_train)
```

Out[31]:
```
LogisticRegression()
```

In [32]:
```
Y_pred = logr.predict(X_test)
```

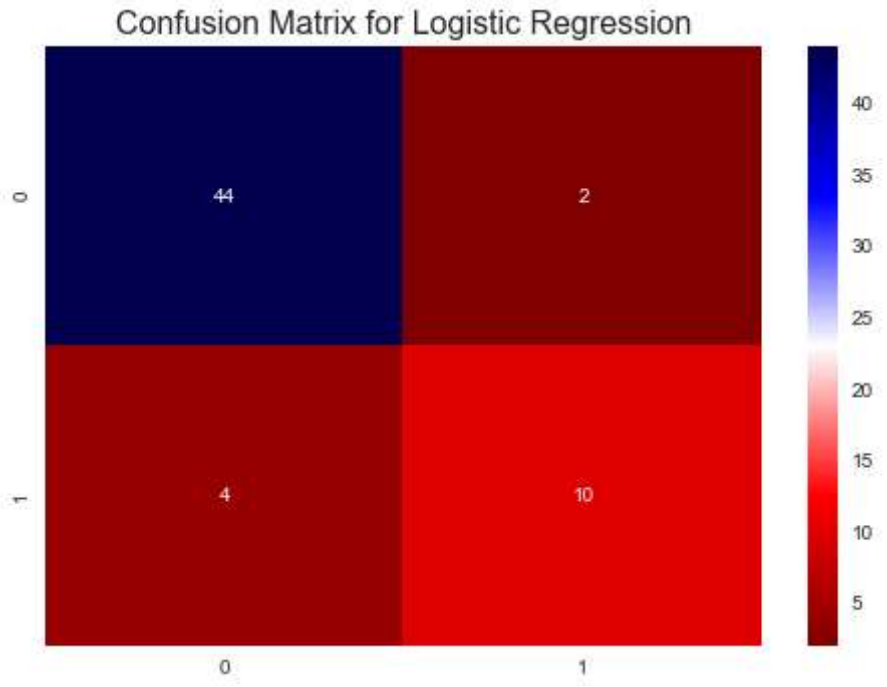
In [33]:
```
LogisticRegressionScore = logr.score(X_test, y_test)
```

In [34]:
```


cf = confusion_matrix(y_test, Y_pred)
cf
```

Out[34]:
```
array([[44,  2],
       [ 4, 10]], dtype=int64)
```

In [35]:
```
sns.heatmap(cf, annot=True, cmap='seismic_r')
plt.title("Confusion Matrix for Logistic Regression", fontsize=16, y=1)
```

Out[35]:
```
Text(0.5, 1, 'Confusion Matrix for Logistic Regression')
```



In [36]:
```
print(classification_report(y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94        46
           1       0.83      0.71      0.77        14

    accuracy                           0.90        60
   macro avg       0.88      0.84      0.85        60
weighted avg       0.90      0.90      0.90        60
```

In [37]:
```
acc_log = round(logr.score(X_train, y_train) * 100,4)
print(acc_log)
```

```
79.0795
```

RANDOM FOREST

In [38]:
```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
```

Out[38]:
```
RandomForestClassifier()
```

In [39]:
```
Y_prediction = random_forest.predict(X_test)
```

In [40]:
```
random_forest.score(X_train, y_train)
```

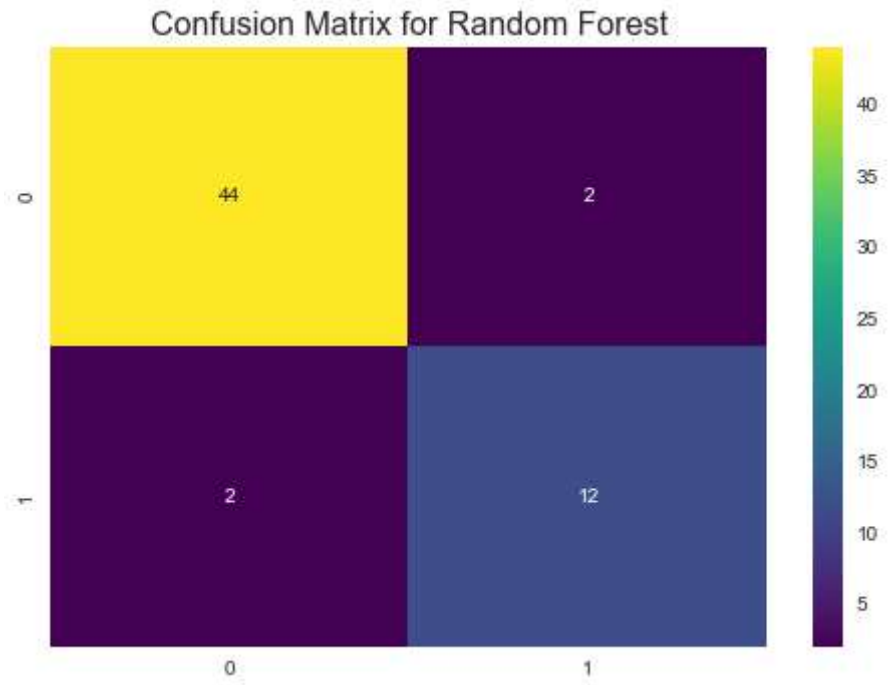
Out[40]:
```
1.0
```

In [41]:
```
cf = confusion_matrix(y_test, Y_prediction)
cf
```

Out[41]:
```
array([[44,  2],
       [ 2, 12]], dtype=int64)
```

In [42]:
```
sns.heatmap(cf, annot=True, cmap='viridis')
plt.title("Confusion Matrix for Random Forest", fontsize=16, y=1)
```

Out[42]: Text(0.5, 1, 'Confusion Matrix for Random Forest')



In [43]:
```
print(classification_report(y_test,Y_prediction))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        46
           1       0.86      0.86      0.86        14

    accuracy                           0.93        60
   macro avg       0.91      0.91      0.91        60
weighted avg       0.93      0.93      0.93        60
```

In [44]:
```
acc_forest = round(random_forest.score(X_train, y_train) * 100,4)
print(acc_forest)
```

```
100.0
```

DECISION TREE

In [45]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train,y_train)
```

Out[45]:
```
DecisionTreeClassifier()
```

In [46]:
```
Y_predictions =decision_tree.predict(X_test)
```

In [47]:
```
decision_tree.score(X_train, y_train)
```

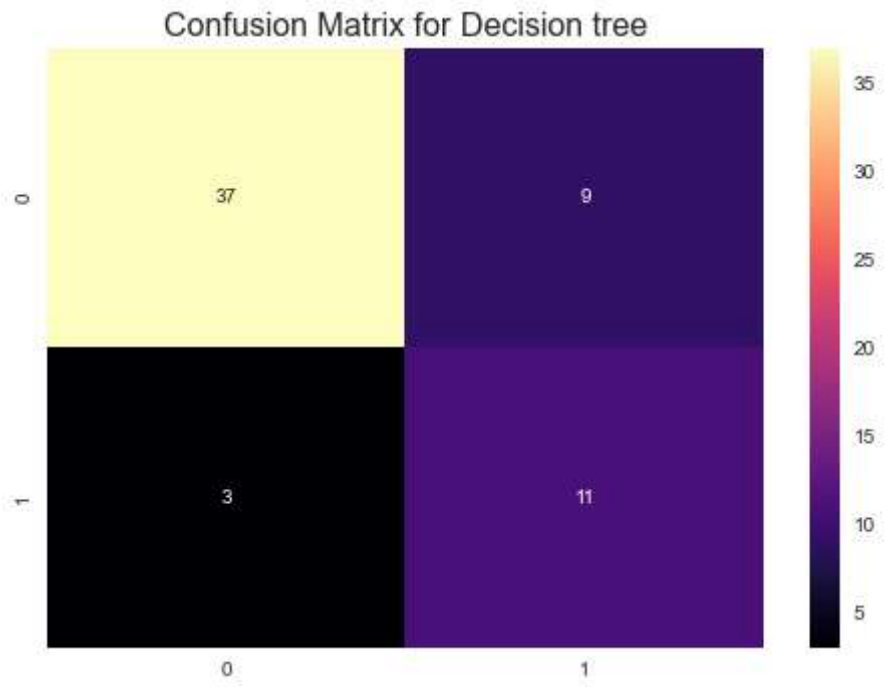
Out[47]:
```
1.0
```

In [48]:
```
cf = confusion_matrix(y_test, Y_predictions)
cf
```

Out[48]:
```
array([[37,  9],
       [ 3, 11]], dtype=int64)
```

In [49]:
```
sns.heatmap(cf, annot=True,cmap="magma")
plt.title("Confusion Matrix for Decision tree", fontsize=16, y=1)
```

Out[49]:
```
Text(0.5, 1, 'Confusion Matrix for Decision tree')
```



In [50]:
```
print(classification_report(y_test,Y_predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        46
           1       0.55      0.79      0.65        14

    accuracy                           0.80        60
   macro avg       0.74      0.80      0.75        60
weighted avg       0.84      0.80      0.81        60
```

In [51]:
```
acc_decision = round(random_forest.score(X_train, y_train) * 100,4)
print(acc_decision)
```

```
100.0
```

In these three models let us see which one is the best model for this dataset

In [52]:
```
results = pd.DataFrame({
    'Model': ['Logistic Regression',
              'Random Forest',
              'Decision Tree'],
    'Score': [  acc_log,
              acc_forest,
              acc_decision]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

Out[52]:

Score	Model
100.0000	Random Forest
100.0000	Decision Tree
79.0795	Logistic Regression

In the above table we observed that Random Forest and Decision Tree are more accurate than LogisticRegression

K-FOLD CROSS VALIDATION

In the above observations Random Forest and Decision Tree got 100% accuracy

Let validate our Random Forest

In [*]:
```
rf = RandomForestClassifier(n_estimators=100)
scores = cross_val_score(rf, X_train, y_train, cv=10, scoring = "accuracy")
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

In our model Random Forest has average accuracy 83% with a standard deviation 5%

In [*]:
```
df = DecisionTreeClassifier()
scores = cross_val_score(df, X_train,y_train, cv=10, scoring = "accuracy")
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

In our model Decision Tree has average accuracy 75% with a standard deviation 6%

CONCLUSION

We select the dataset about heart failure .Firstly we import all the required libraries and then we perform th data cleaning process by checking the null values after that we tried to visualize our data with some visualization techniques for better undestanding of the dataset after that we create models for predicting the heart failure we create model by using logisticregression ,random forest and decision tree.we got 100% accuracy in random forest and decision tree and logistic regression has 80% accuracy and then we apply cross validation on random forest and decision tree.