

# Heart\_Failure\_Project

March 13, 2023

```
[1]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np
import plotly.express as px
import scipy.stats as stats
from scipy.stats import chi2_contingency, Rand
```

## 1 Heart Failure PCA and Classification

```
[2]: hfd = pd.read_csv(r'C:\Users\samue\Documents\College Notes\PyRe\Data_\
sets\heart_failure_dataset.csv')
hfd = hfd.dropna()
hfd.head
```

```
[2]: <bound method NDFrame.head of
diabetes  ejection_fraction  \
0      75.0          0          582          0          20
1      55.0          0       7861          0          38
2      65.0          0        146          0          20
3      50.0          1        111          0          20
4      65.0          1        160          1          20
..      ...          ...          ...          ...          ...
294    62.0          0          61          1          38
295    55.0          0       1820          0          38
296    45.0          0       2060          1          60
297    45.0          0       2413          0          38
298    50.0          0        196          0          45

      high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                1  265000.00          1.9          130      1
1                0  263358.03          1.1          136      1
2                0  162000.00          1.3          129      1
3                0  210000.00          1.9          137      1
```

4	0	327000.00	2.7	116	0
..	...	...	...	...	...
294	1	155000.00	1.1	143	1
295	0	270000.00	1.2	139	0
296	0	742000.00	0.8	138	0
297	0	140000.00	1.4	140	1
298	0	395000.00	1.6	136	1

	smoking	time	DEATH_EVENT
0	0	4	1
1	0	6	1
2	1	7	1
3	0	7	1
4	0	8	1
..	...	...	...
294	1	270	0
295	0	271	0
296	0	278	0
297	1	280	0
298	1	285	0

[299 rows x 13 columns]>

```
[3]: hfd.describe()
```

```
[3]:
```

	age	anaemia	creatinine_phosphokinase	diabetes \
count	299.000000	299.000000	299.000000	299.000000
mean	60.833893	0.431438	581.839465	0.418060
std	11.894809	0.496107	970.287881	0.494067
min	40.000000	0.000000	23.000000	0.000000
25%	51.000000	0.000000	116.500000	0.000000
50%	60.000000	0.000000	250.000000	0.000000
75%	70.000000	1.000000	582.000000	1.000000
max	95.000000	1.000000	7861.000000	1.000000

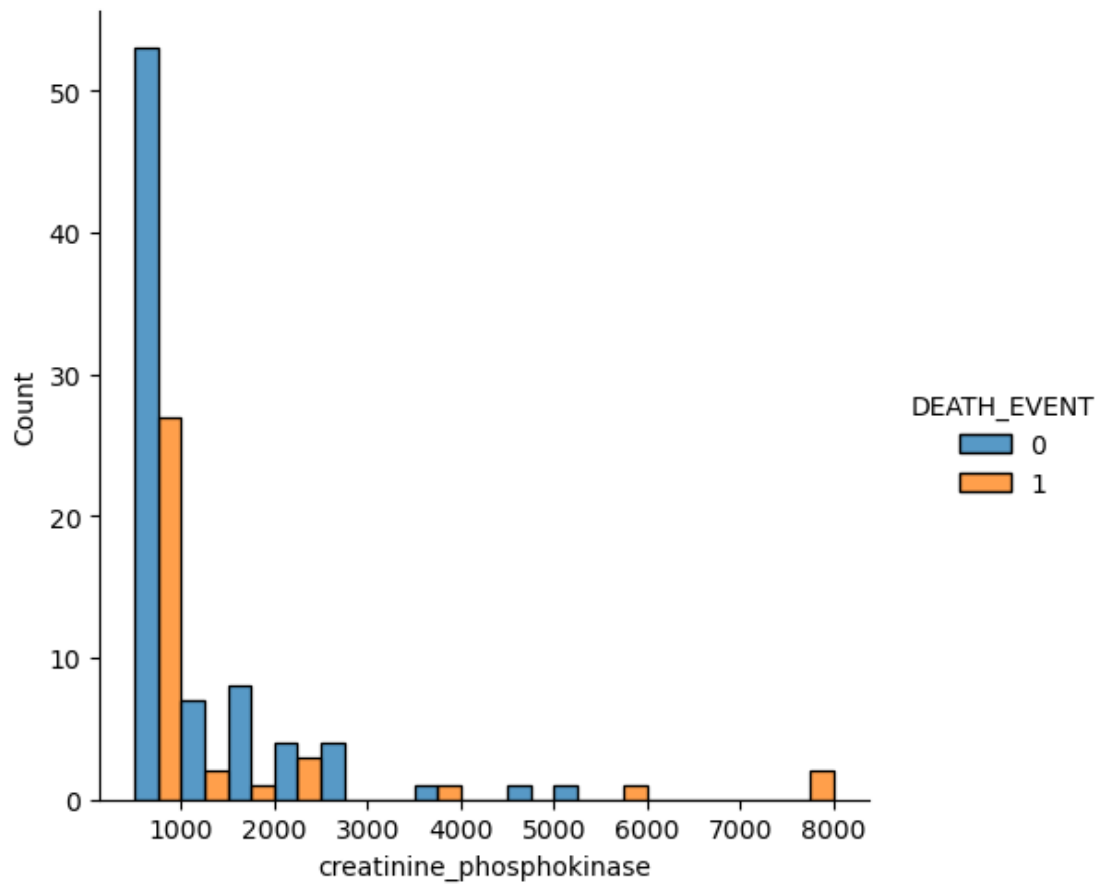
	ejection_fraction	high_blood_pressure	platelets \
count	299.000000	299.000000	299.000000
mean	38.083612	0.351171	263358.029264
std	11.834841	0.478136	97804.236869
min	14.000000	0.000000	25100.000000
25%	30.000000	0.000000	212500.000000
50%	38.000000	0.000000	262000.000000
75%	45.000000	1.000000	303500.000000
max	80.000000	1.000000	850000.000000

	serum_creatinine	serum_sodium	sex	smoking	time \
count	299.000000	299.000000	299.000000	299.000000	299.000000

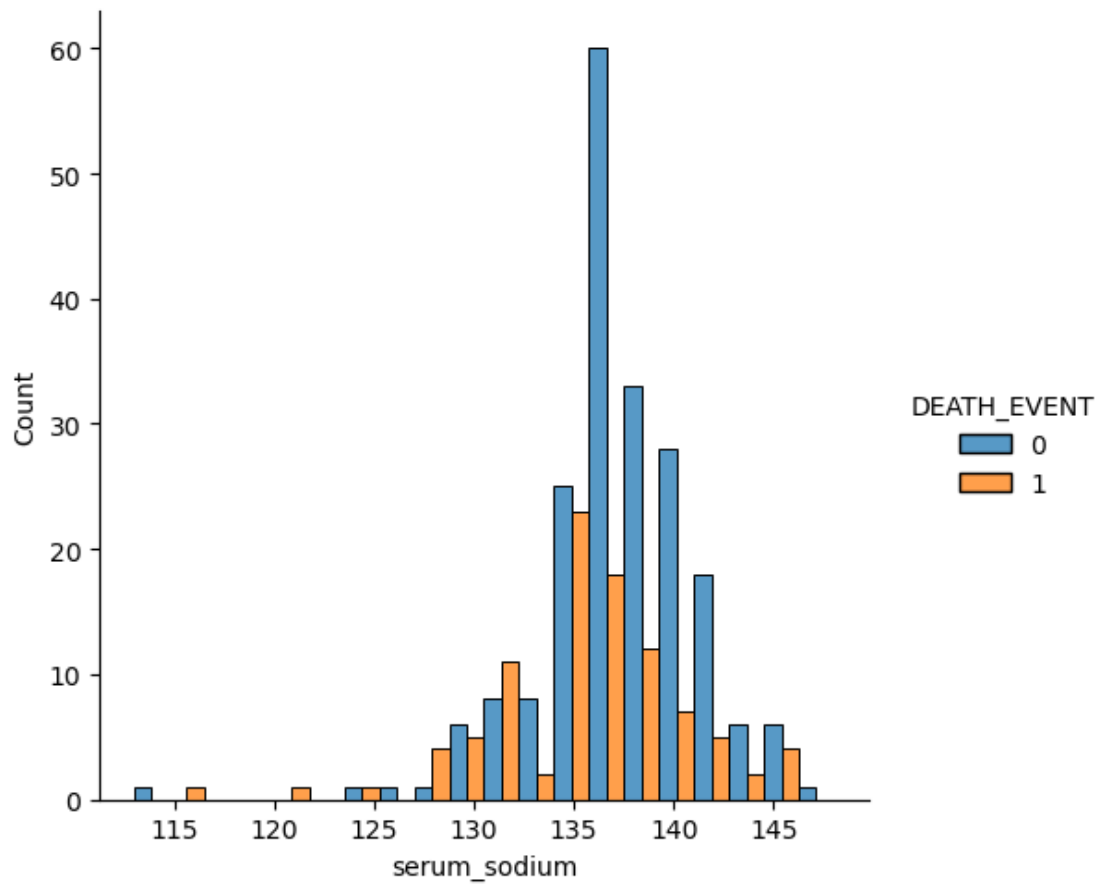
mean	1.39388	136.625418	0.648829	0.32107	130.260870
std	1.03451	4.412477	0.478136	0.46767	77.614208
min	0.50000	113.000000	0.000000	0.00000	4.000000
25%	0.90000	134.000000	0.000000	0.00000	73.000000
50%	1.10000	137.000000	1.000000	0.00000	115.000000
75%	1.40000	140.000000	1.000000	1.00000	203.000000
max	9.40000	148.000000	1.000000	1.00000	285.000000

	DEATH_EVENT
count	299.00000
mean	0.32107
std	0.46767
min	0.00000
25%	0.00000
50%	0.00000
75%	1.00000
max	1.00000

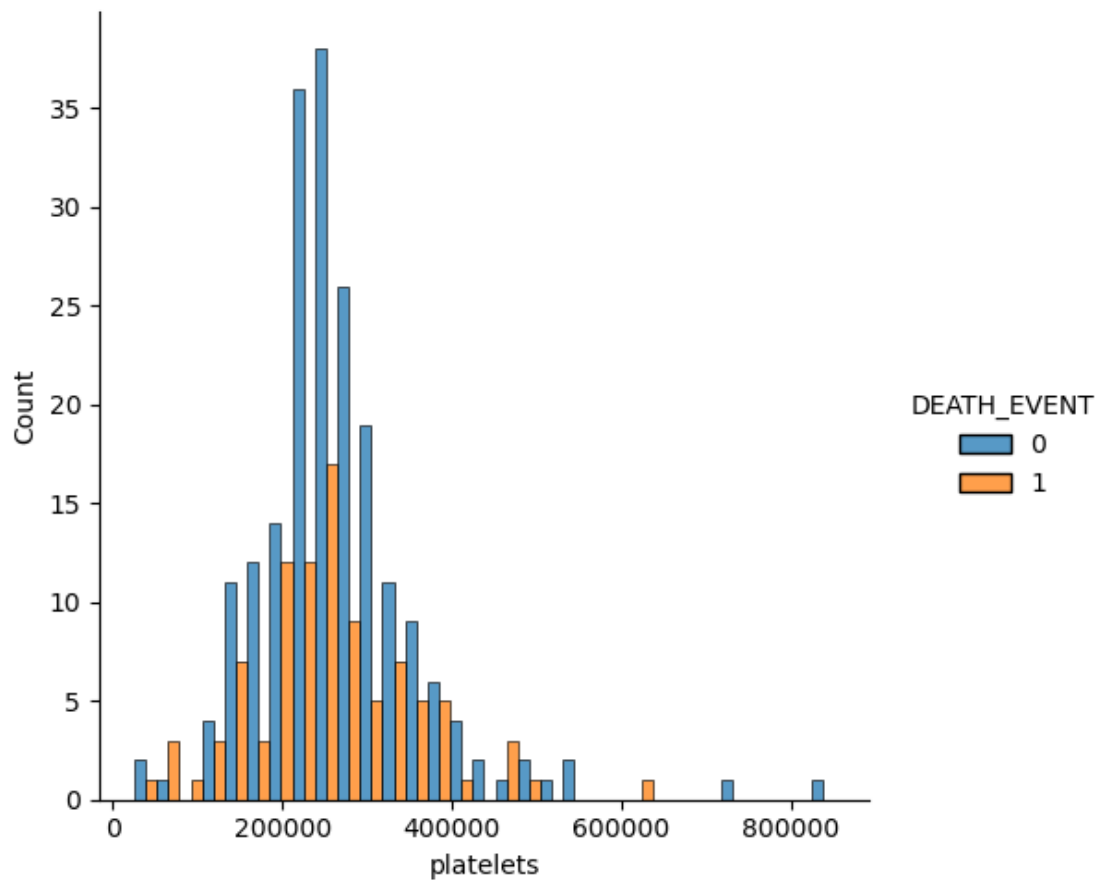
```
[4]: sns.displot(hfd, x = "creatinine_phosphokinase", hue = "DEATH_EVENT", multiple_
      ↪= "dodge",
      bins = [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000,
      ↪5500, 6000, 6500, 7000, 7500, 8000])
plt.show()
```



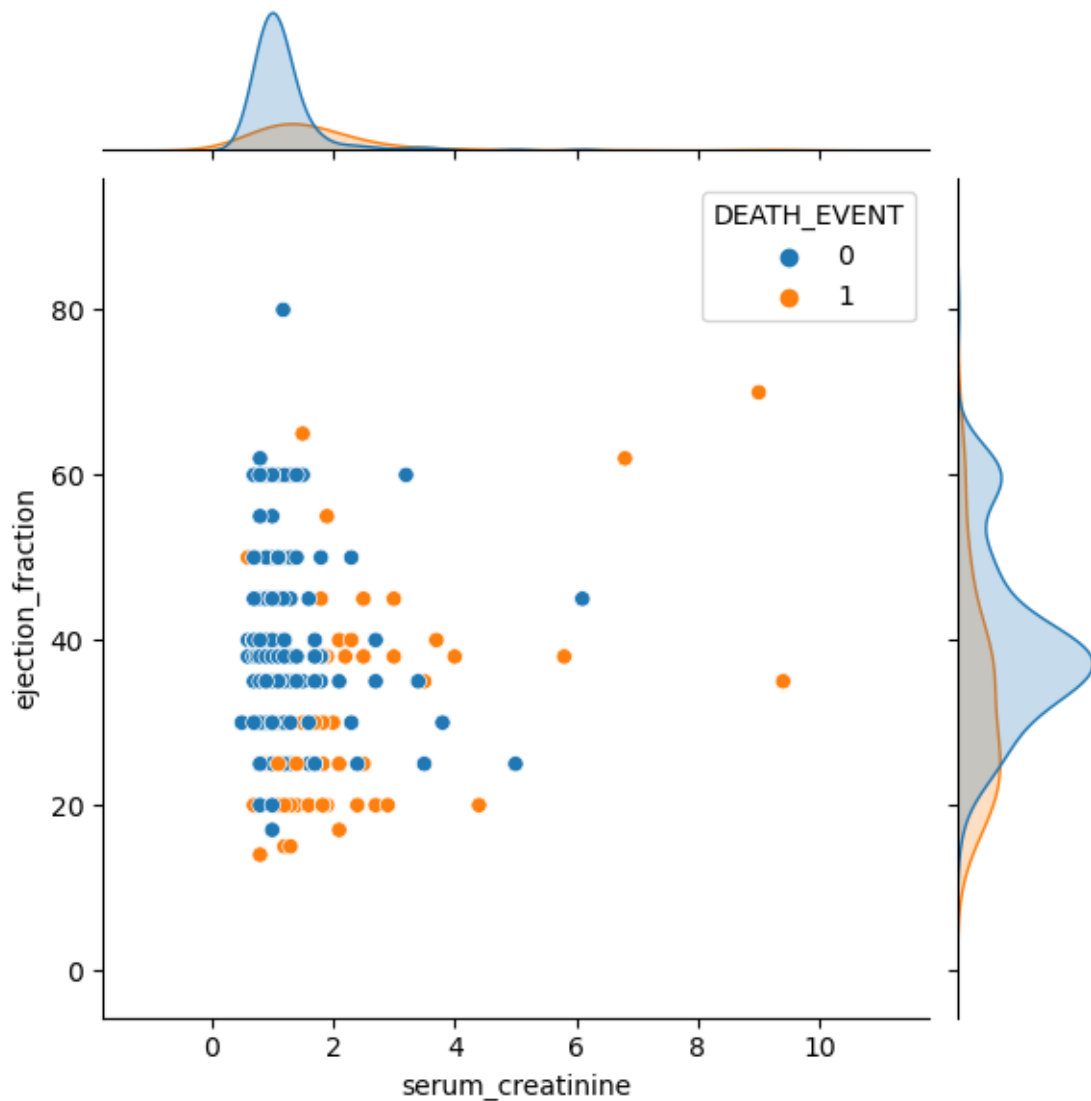
```
[5]: sns.displot(hfd, x = "serum_sodium", hue = "DEATH_EVENT", multiple = "dodge")  
plt.show()
```



```
[6]: sns.displot(hfd, x = "platelets", hue = "DEATH_EVENT", multiple = "dodge")  
plt.show()
```



```
[7]: sns.jointplot(data = hfd, x = "serum_creatinine", y = "ejection_fraction", hue="DEATH_EVENT")  
plt.show()
```



The plots above display the distribution of our continuous variables. As we see the distributions for these variables are relatively normal, save for creatine phosphokinase (CPK) which is skewed right.

```
[8]: stats.mannwhitneyu(x = hfd['DEATH_EVENT'], y = hfd['creatinine_phosphokinase'],
    ↪ alternative = 'two-sided')
```

```
[8]: MannwhitneyResult(statistic=0.0, pvalue=7.80940515793462e-104)
```

```
[9]: stats.mannwhitneyu(x = hfd['DEATH_EVENT'], y = hfd['ejection_fraction'],
    ↪ alternative = 'two-sided')
```

```
[9]: MannwhitneyResult(statistic=0.0, pvalue=5.748810376178891e-104)
```

```
[10]: stats.mannwhitneyu(x = hfd['DEATH_EVENT'], y = hfd['platelets'], alternative = 'two-sided')
```

```
[10]: MannwhitneyuResult(statistic=0.0, pvalue=8.63331766603266e-104)
```

```
[11]: stats.mannwhitneyu(x = hfd['DEATH_EVENT'], y = hfd['serum_creatinine'], alternative = 'two-sided')
```

```
[11]: MannwhitneyuResult(statistic=10176.0, pvalue=2.2614587328253158e-63)
```

```
[12]: stats.mannwhitneyu(x = hfd['DEATH_EVENT'], y = hfd['serum_sodium'], alternative = 'two-sided')
```

```
[12]: MannwhitneyuResult(statistic=0.0, pvalue=6.761757176362062e-104)
```

```
[13]: stats.mannwhitneyu(x = hfd['DEATH_EVENT'], y = hfd['age'], alternative = 'two-sided')
```

```
[13]: MannwhitneyuResult(statistic=0.0, pvalue=7.787877997331131e-104)
```

```
[14]: hfd_target = hfd.loc[:, 'DEATH_EVENT']
      hfd_target.head

      hfd_features = hfd.drop(['anaemia', 'sex', 'diabetes', 'time', 'smoking', 'DEATH_EVENT'], axis = 1)
      hfd_features.head
```

```
[14]: <bound method NDFrame.head of
      ejection_fraction  high_blood_pressure  \
0      75.0      582      20      1
1      55.0      7861      38      0
2      65.0      146      20      0
3      50.0      111      20      0
4      65.0      160      20      0
..      ...      ...      ...      ...
294    62.0      61      38      1
295    55.0      1820      38      0
296    45.0      2060      60      0
297    45.0      2413      38      0
298    50.0      196      45      0

      platelets  serum_creatinine  serum_sodium
0      265000.00      1.9      130
1      263358.03      1.1      136
2      162000.00      1.3      129
3      210000.00      1.9      137
4      327000.00      2.7      116
```



```

..      ...      ...      ...
294  155000.00      1.1      143
295  270000.00      1.2      139
296  742000.00      0.8      138
297  140000.00      1.4      140
298  395000.00      1.6      136

```

[299 rows x 7 columns]>

Separating our target variable from our feature variables. The binary variable ‘high blood pressure’ is retained.

```

[15]: x = StandardScaler().fit_transform(hfd_features)

hfd_pcamodel = PCA(n_components = 4)
hfd_pca = hfd_pcamodel.fit_transform(x)
hfd_pca.shape

```

[15]: (299, 4)

Creating our PCA model with four principal components to start.

```

[16]: hfd_pca2 = pd.DataFrame(data = hfd_pca, columns = ['PC1', 'PC2', 'PC3', 'PC4'])
hfd_pca2.head

```

```

[16]: <bound method NDFrame.head of          PC1      PC2      PC3      PC4
0      2.044649 -0.164567 -1.244152  0.939179
1     -1.784539  3.546248  3.683013  2.623146
2      1.931770  1.397081 -0.695478 -0.648093
3      0.571228  1.433565 -0.519324 -0.430294
4      3.891392  1.654035 -0.789666  1.785356
..      ...      ...      ...      ...
294   -0.509334 -1.029473 -0.749240 -1.192754
295   -0.887347  0.977491  0.892423  0.212809
296   -3.324198 -0.178264  1.087546  3.933024
297   -0.954457  1.778759  1.208409 -0.577548
298   -0.650700  0.140269  0.214062  0.828817

```

[299 rows x 4 columns]>

```

[17]: hfd_final = pd.concat([hfd_pca2, hfd_target], axis = 1, join = 'inner')
hfd_final.head

```

```

[17]: <bound method NDFrame.head of          PC1      PC2      PC3      PC4
DEATH_EVENT
0      2.044649 -0.164567 -1.244152  0.939179      1
1     -1.784539  3.546248  3.683013  2.623146      1
2      1.931770  1.397081 -0.695478 -0.648093      1

```

```

3      0.571228  1.433565 -0.519324 -0.430294      1
4      3.891392  1.654035 -0.789666  1.785356      1
..      ...      ...      ...      ...      ...
294 -0.509334 -1.029473 -0.749240 -1.192754      0
295 -0.887347  0.977491  0.892423  0.212809      0
296 -3.324198 -0.178264  1.087546  3.933024      0
297 -0.954457  1.778759  1.208409 -0.577548      0
298 -0.650700  0.140269  0.214062  0.828817      0

[299 rows x 5 columns]>

```

```
[18]: hfd_pcamodel.explained_variance_
```

```
[18]: array([1.34980664, 1.21784302, 1.00188564, 0.98827669])
```

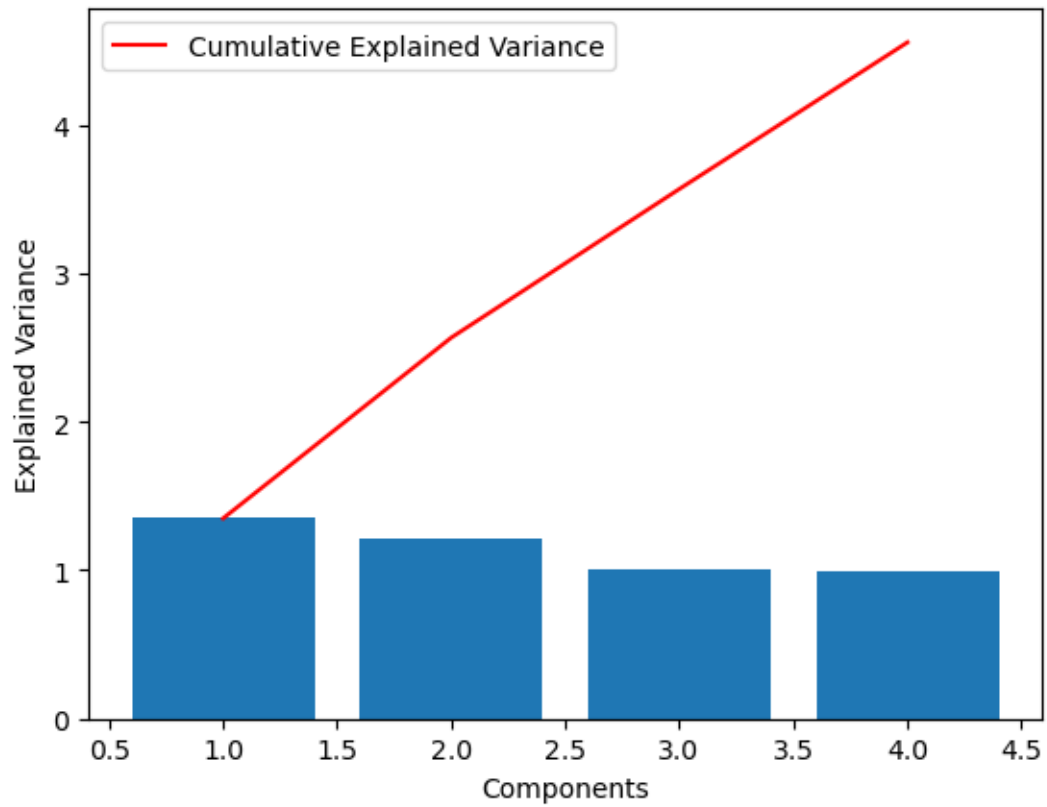
```
[19]: hfd_pcamodel.explained_variance_ratio_
```

```
[19]: array([0.1921846 , 0.17339571, 0.14264784, 0.1407102 ])
```

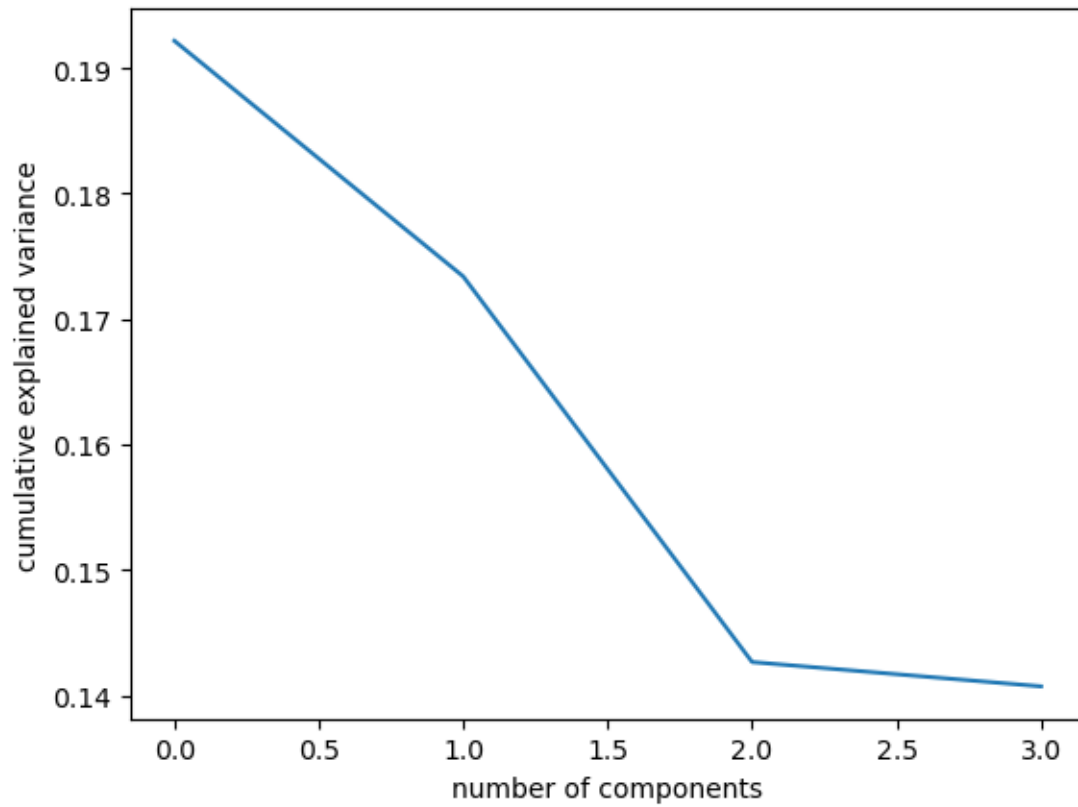
The explained variance (eigenvalues) tells us the first four principal components account for 65% of the variance in the data. Not a great PCA problem likely due to weak relationships between the variables. A correlation matrix should be created to get specific correlation coefficients to determine the true strength of PCA for this data.

```
[20]: plt.bar(range(1,len(hfd_pcamodel.explained_variance_)+1),hfd_pcamodel.
        ↪explained_variance_)
plt.ylabel('Explained Variance')
plt.xlabel('Components')
plt.plot(range(1,len(hfd_pcamodel.explained_variance_)+1), np.
        ↪cumsum(hfd_pcamodel.explained_variance_), c = 'red',
        label = "Cumulative Explained Variance")
plt.legend(loc = 'upper left')
plt.show()

```

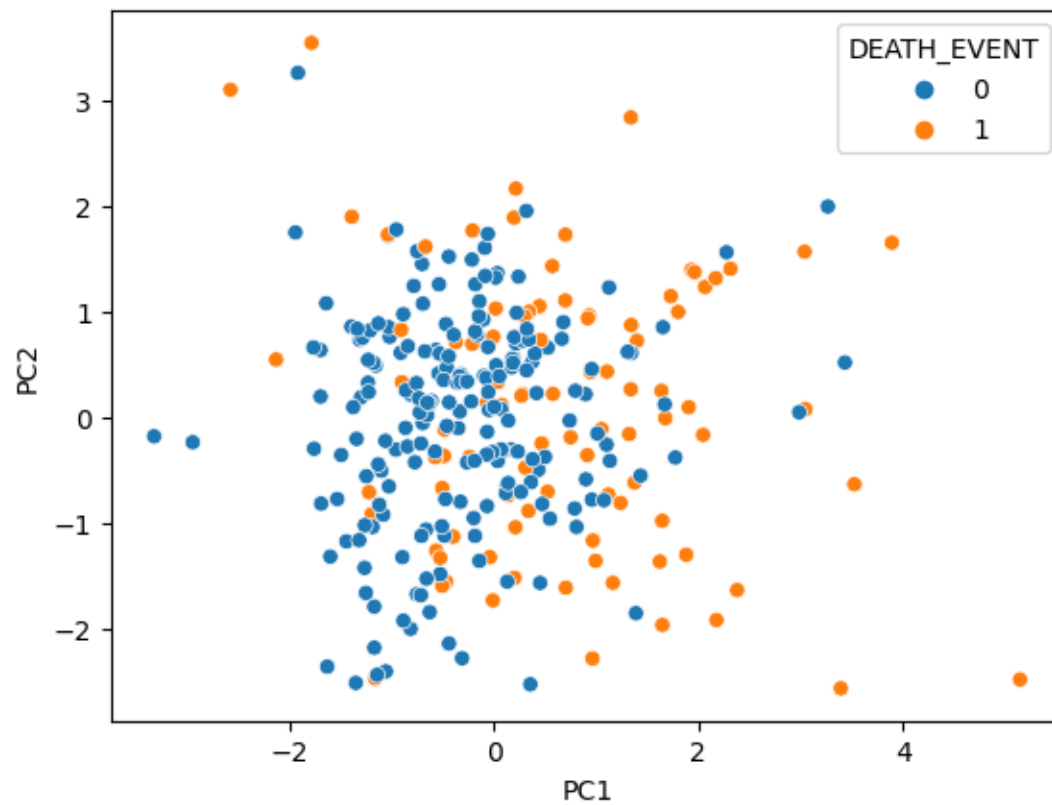


```
[21]: plt.plot(hfd_pcamodel.explained_variance_ratio_)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```

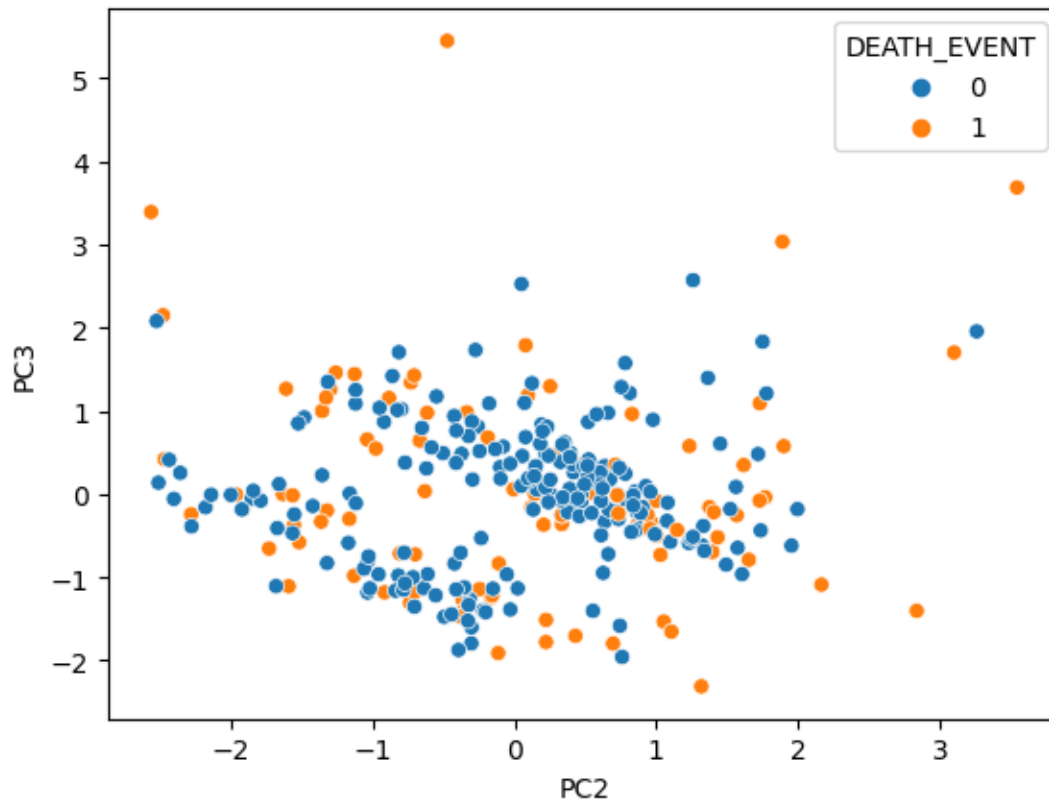


Scree plot shows that only the first two principal components should be retained.

```
[22]: hfd_pcafig = sns.scatterplot(data = hfd_final, x = 'PC1', y = 'PC2', hue = "DEATH_EVENT")  
plt.show()
```



```
[23]: hfd_pcafig2 = sns.scatterplot(data = hfd_final, x = 'PC2', y = 'PC3', hue = "DEATH_EVENT")
plt.show()
```

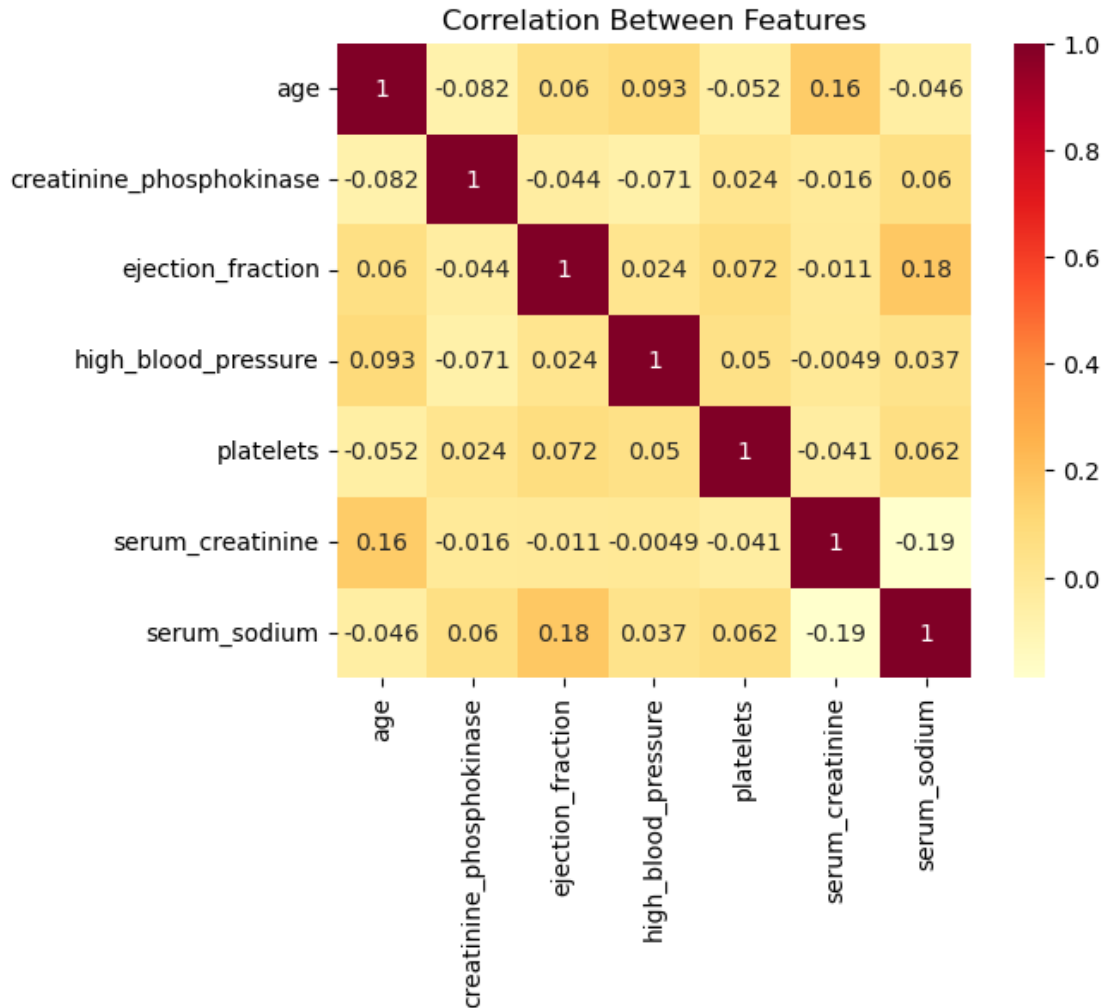


Interesting separation between the second and third principal components, likely due to the binary variable 'high blood pressure'.

```
[24]: hfd_pcafig3 = px.scatter_3d(
      hfd_final, x = 'PC1', y = 'PC2', z = 'PC3', color = 'DEATH_EVENT',
      title = '3 Dimensions',
      labels = {'0': 'PC1', '1': 'PC2', '2': 'PC3'})
      hfd_pcafig3.show()
```

Created a 3D plot because I wanted to see the separation between the second and third principal components.

```
[25]: correlation = hfd_features.corr()
      sns.heatmap(correlation, square = True, annot = True, cmap = 'YlOrRd')
      plt.title("Correlation Between Features")
      plt.show()
```



As we see from the correlation matrix all the variables have weak correlations, less than 0.2, meaning that PCA will not help in reducing the data in a meaningful way.

```
[46]: from sklearn.model_selection import train_test_split, RandomizedSearchCV
      from sklearn.linear_model import LogisticRegression
      from sklearn import metrics
      from sklearn.metrics import classification_report
      from sklearn.metrics import mean_squared_error
      from sklearn.model_selection import cross_val_score
      from sklearn.ensemble import RandomForestClassifier
      from sklearn import tree
      from sklearn.metrics import confusion_matrix, accuracy_score, precision_score
```

```
[27]: hfd_features_train, hfd_features_test, hfd_target_train, hfd_target_test = \
      ↪ train_test_split(hfd_features, hfd_target, test_size = 0.25, random_state = \
      ↪ 10)
```

Partitioning our data between testing and training sets. In this line of code, I'm assigning 75% of the data as our training set and the remaining 25% as our testing set.

```
[28]: hflogreg = LogisticRegression(random_state = 10)
      hflogreg.fit(hfd_features_train, hfd_target_train)
      y_pred = hflogreg.predict(hfd_features_test)
```

Creating a logistic regression model.

```
[29]: hf_scores = cross_val_score(hflogreg, hfd_features, hfd_target, scoring =_
      ↪ "neg_mean_squared_error", cv = 10, n_jobs = 1)

      hf_rmse = np.sqrt(-hf_scores)
      print("RMSE Values: ", np.round(hf_rmse, 2))
      print("RMSE Average: ", np.mean(hf_rmse))
```

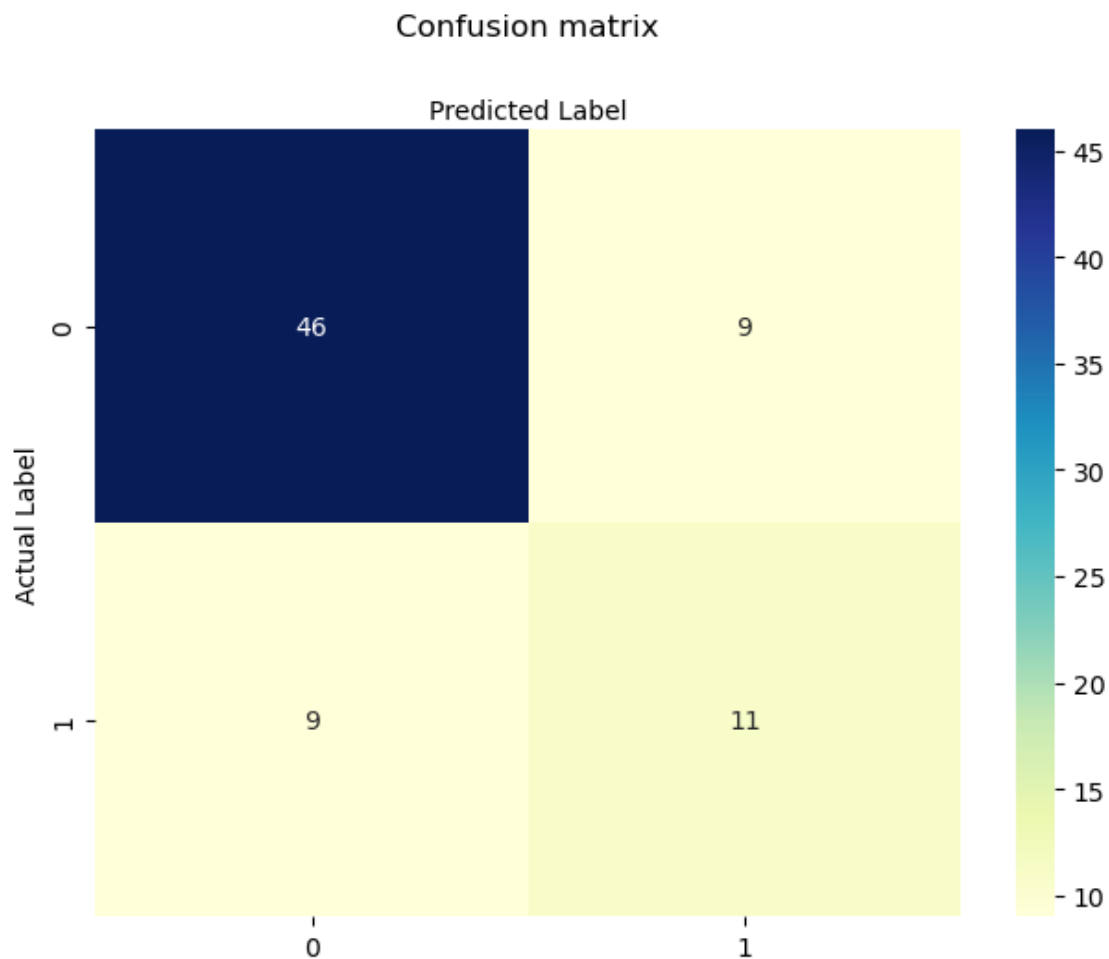
```
RMSE Values:  [0.41 0.58 0.45 0.61 0.52 0.58 0.55 0.37 0.45 0.49]
RMSE Average: 0.4983478483772646
```

```
[30]: con_matrix = metrics.confusion_matrix(hfd_target_test, y_pred)
      con_matrix
```

```
[30]: array([[46,  9],
            [ 9, 11]], dtype=int64)
```

```
[31]: class_names = [0,1]
      fig, ax = plt.subplots()
      tick_marks = np.arange(len(class_names))
      plt.xticks(tick_marks, class_names)
      plt.yticks(tick_marks, class_names)
      # Heatmap
      sns.heatmap(pd.DataFrame(con_matrix), annot = True, cmap = "YlGnBu", fmt = 'g')
      ax.xaxis.set_label_position("top")
      plt.tight_layout()
      plt.title('Confusion matrix', y = 1.1)
      plt.ylabel('Actual Label')
      plt.xlabel('Predicted Label')
      plt.show()
```





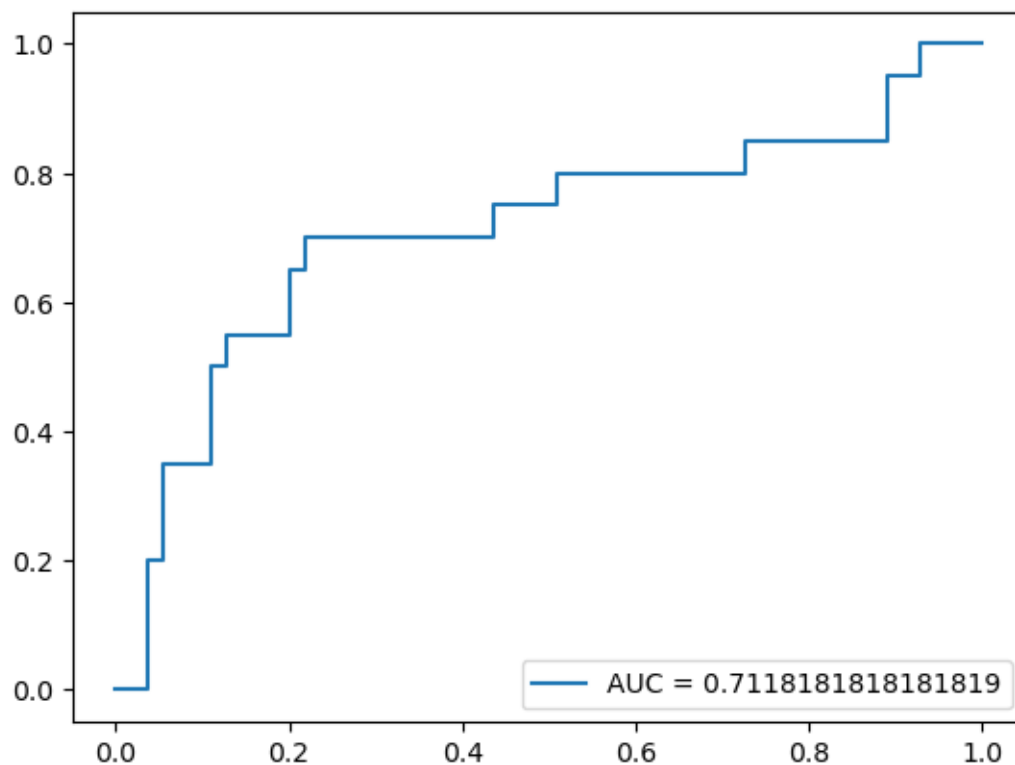
```
[32]: target_names = ['Alive', 'Death Event']
print(classification_report(hfd_target_test, y_pred, target_names =
↪target_names))
```

	precision	recall	f1-score	support
Alive	0.84	0.84	0.84	55
Death Event	0.55	0.55	0.55	20
accuracy			0.76	75
macro avg	0.69	0.69	0.69	75
weighted avg	0.76	0.76	0.76	75

The results from our logistic regression model give us an AUC of 0.71, which means that our model has a 71% chance of predicting survival from a patient afflicted with heart failure. The sensitivity value calculated is 0.55 and our specificity value is 0.84.

Given our model performance ( $AUC = 0.71$ ) we can say that the model is satisfactory in predicting survival in patients with heart failure, however due to the domain that our question arises (healthcare) a model accuracy of 71% would be unacceptable in a healthcare setting. Because of this, more time should be spent tuning our model further in an attempt to reach a model accuracy of at least .90 or 90%.

```
[33]: y_pred_proba = hflogreg.predict_proba(hfd_features_test)[:,:1]
fpr, tpr, _ = metrics.roc_curve(hfd_target_test, y_pred_proba)
auc = metrics.roc_auc_score(hfd_target_test, y_pred_proba)
plt.plot(fpr, tpr, label = "AUC = " +str(auc))
plt.legend(loc=4)
plt.show()
```



```
[ ]:
```