```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from ydata_profiling import ProfileReport
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import pandas as pd
import numpy as np
```

```
!pip install ydata_profiling
```

```
data = pd.read_csv("/content/heart_failure_clinical_records - heart_failure_clinical_records.csv")
data.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	s
0	55.0	0	748	0	45	0	263358.03	1.3	137	1	
1	65.0	0	56	0	25	0	305000.00	5.0	130	1	
2	45.0	0	582	1	38	0	319000.00	0.9	140	0	
3	60.0	1	754	1	40	1	328000.00	1.2	126	1	
4	95.0	1	582	0	30	0	461000.00	2.0	132	1	

```
data.shape
```

```
(5000, 13)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   age                       5000 non-null   float64
 1   anaemia                   5000 non-null   int64  
 2   creatinine_phosphokinase  5000 non-null   int64  
 3   diabetes                  5000 non-null   int64  
 4   ejection_fraction         5000 non-null   int64  
 5   high_blood_pressure       5000 non-null   int64  
 6   platelets                 5000 non-null   float64
 7   serum_creatinine          5000 non-null   float64
 8   serum_sodium              5000 non-null   int64  
 9   sex                       5000 non-null   int64  
 10  smoking                   5000 non-null   int64  
 11  time                      5000 non-null   int64  
 12  DEATH_EVENT               5000 non-null   int64  
dtypes: float64(3), int64(10)
memory usage: 507.9 KB
```

```
data.columns
```

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

```
data.isna().sum()
```

```
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
```

```
print(data['age'].nunique())
data['age'].unique()
```

```
48
array([55.   , 65.   , 45.   , 60.   , 95.   , 70.   , 63.   , 50.   ,
       53.   , 60.667, 72.   , 64.   , 75.   , 66.   , 58.   , 42.   ,
       69.   , 68.   , 49.   , 51.   , 44.   , 59.   , 90.   , 61.   ,
       46.   , 80.   , 56.   , 41.   , 85.   , 82.   , 67.   , 52.   ,
       43.   , 81.   , 48.   , 57.   , 40.   , 86.   , 77.   , 73.   ,
       62.   , 87.   , 79.   , 47.   , 94.   , 71.   , 78.   , 54.   ])
```

```
data.describe()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	60.288736	0.474400	586.760600	0.439400	37.734600	0.364800	265075.404370	1.369106
std	11.697243	0.499394	976.733979	0.496364	11.514855	0.481422	97999.758622	1.009750
min	40.000000	0.000000	23.000000	0.000000	14.000000	0.000000	25100.000000	0.500000
25%	50.000000	0.000000	121.000000	0.000000	30.000000	0.000000	215000.000000	0.900000
50%	60.000000	0.000000	248.000000	0.000000	38.000000	0.000000	263358.030000	1.100000
75%	68.000000	1.000000	582.000000	1.000000	45.000000	1.000000	310000.000000	1.400000
max	95.000000	1.000000	7861.000000	1.000000	80.000000	1.000000	850000.000000	9.400000

```
# Generate the profile report
report = ProfileReport(data)

# Save the report to a file
report.to_file("data_profile_report.html")

# Alternatively, you can also display the report directly
report.to_notebook_iframe()
```

Summarize dataset: 100% 71/71 [00:15<00:00, 2.99it/s, Completed]

Generate report structure: 100% 1/1 [00:07<00:00, 7.87s/it]

Render HTML: 100% 1/1 [00:02<00:00, 2.04s/it]

Export report to file: 100% 1/1 [00:00<00:00, 18.81it/s]

Overview

Dataset statistics

Number of variables	13
Number of observations	5000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	451
Duplicate rows (%)	9.0%
Total size in memory	507.9 KiB
Average record size in memory	104.0 B

Variable types

Numeric	7
Categorical	6

Alerts

Dataset has 451 (9.0%) duplicate rows	Duplicates
`DEATH_EVENT` is highly overall correlated with `time`	High correlation
`time` is highly overall correlated with `DEATH_EVENT`	High correlation

Reproduction

Analysis started	2024-05-06 14:27:58.739594
Analysis	2024-05-06 14:28:12.949112

```
data.duplicated().sum()
```

```
3680
```

```
data=data.drop_duplicates()
```

```
data.shape
```

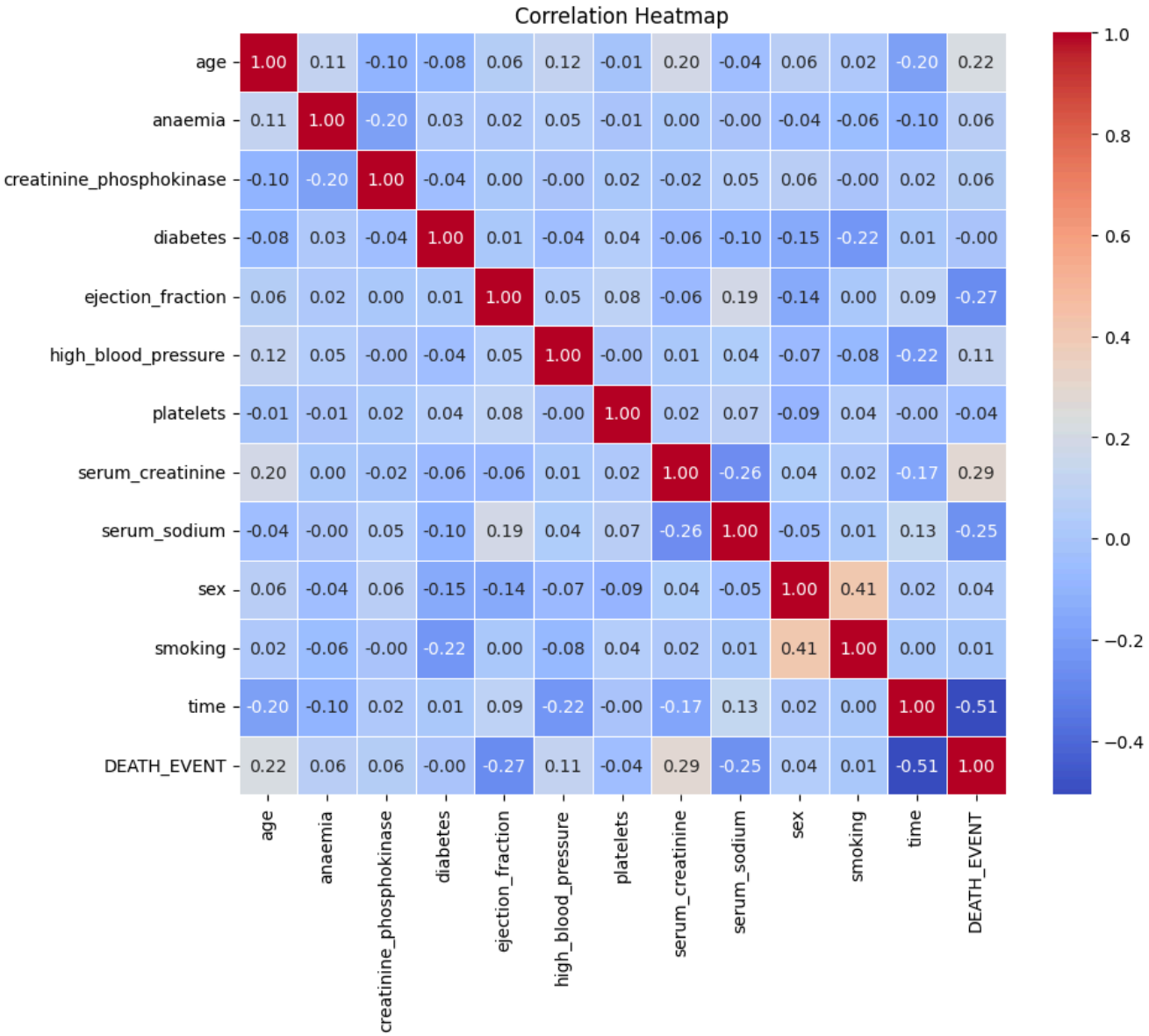
```
(1320, 13)
```

```
# Calculate the correlation matrix
correlation_matrix = data.corr()

# Print the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                              age   anaemia  creatinine_phosphokinase  \
age                      1.000000  0.108039                 -0.098890
anaemia                  0.108039  1.000000                 -0.200294
creatinine_phosphokinase -0.098890 -0.200294                  1.000000
diabetes                 -0.077437  0.031989                 -0.042517
```

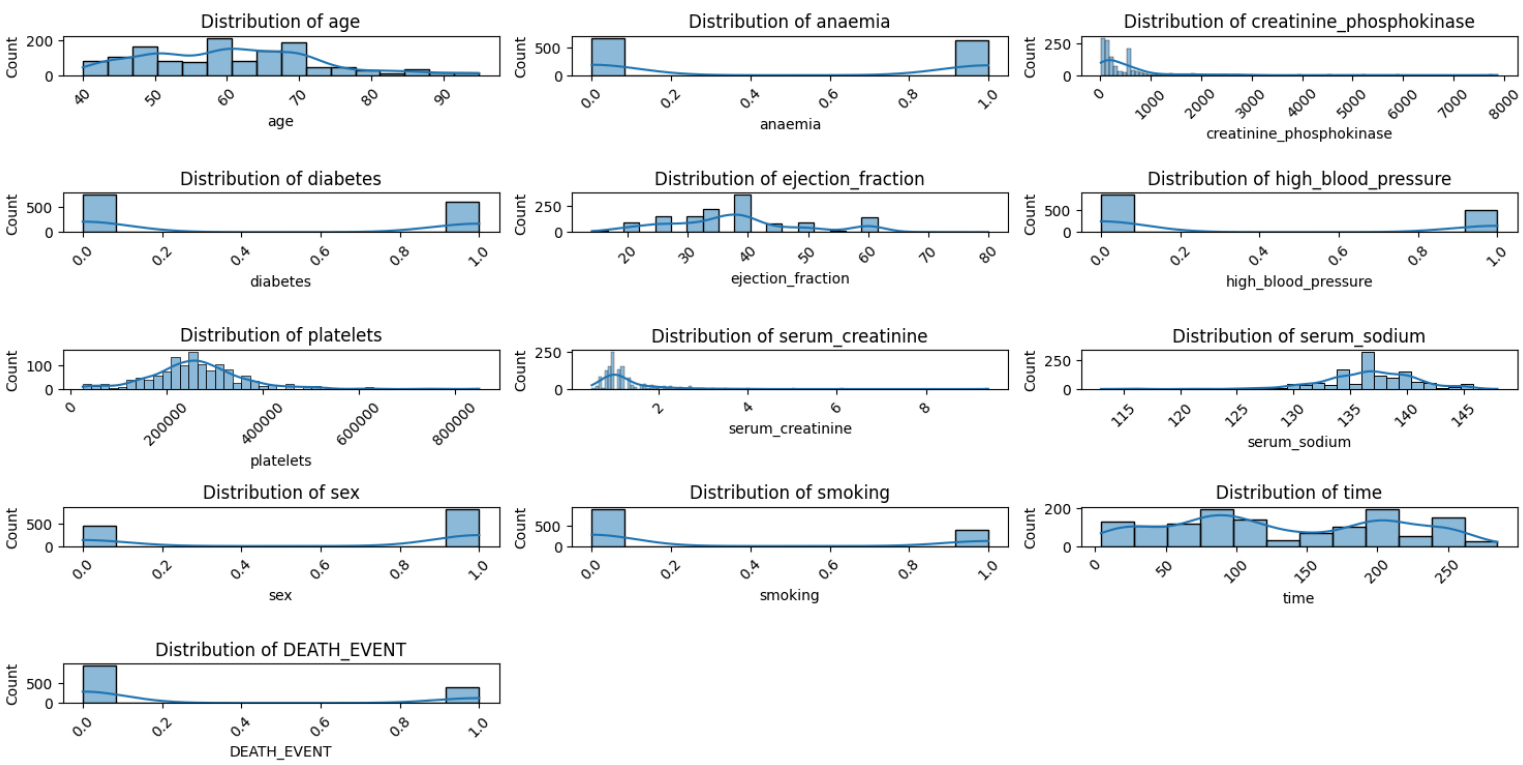
```
ejection_fraction         0.057771  0.024339                 0.002157
high_blood_pressure       0.122868  0.047177                -0.004945
platelets                -0.009855 -0.006089                 0.015418
serum_creatinine          0.197325  0.003655                -0.018248
serum_sodium             -0.044933 -0.003755                 0.047212
sex                       0.059648 -0.037188                 0.061105
smoking                   0.022495 -0.056350                -0.002144
time                     -0.198010 -0.097733                 0.019553
DEATH_EVENT               0.224602  0.063510                 0.055221

                          diabetes  ejection_fraction  high_blood_pressure  \
age                      -0.077437           0.057771             0.122868
anaemia                   0.031989           0.024339             0.047177
creatinine_phosphokinase -0.042517           0.002157            -0.004945
diabetes                  1.000000           0.012477            -0.038261
ejection_fraction         0.012477           1.000000             0.049202
high_blood_pressure      -0.038261           0.049202             1.000000
platelets                 0.044104           0.083884            -0.004260
serum_creatinine         -0.063715          -0.060202             0.013098
serum_sodium             -0.095644           0.194937             0.037283
sex                      -0.149128          -0.143921            -0.065553
smoking                  -0.222771           0.002126            -0.078545
time                      0.008653           0.086484            -0.219173
DEATH_EVENT              -0.001485          -0.271767             0.113721

                          platelets  serum_creatinine  serum_sodium       sex  \
age                       -0.009855          0.197325     -0.044933  0.059648
anaemia                   -0.006089          0.003655     -0.003755 -0.037188
creatinine_phosphokinase   0.015418         -0.018248      0.047212  0.061105
diabetes                   0.044104         -0.063715     -0.095644 -0.149128
ejection_fraction          0.083884         -0.060202      0.194937 -0.143921
high_blood_pressure       -0.004260          0.013098      0.037283 -0.065553
platelets                  1.000000          0.023062      0.065051 -0.090300
serum_creatinine           0.023062          1.000000     -0.263781  0.037234
serum_sodium               0.065051         -0.263781      1.000000 -0.047862
sex                       -0.090300          0.037234     -0.047862  1.000000
smoking                    0.043759          0.020209      0.011111  0.411603
time                      -0.001018         -0.165679      0.130820  0.017673
DEATH_EVENT               -0.044523          0.290229     -0.250990  0.044045

                           smoking      time  DEATH_EVENT
age                       0.022495 -0.198010     0.224602
anaemia                  -0.056350 -0.097733     0.063510
creatinine_phosphokinase -0.002144  0.019553     0.055221
diabetes                 -0.222771  0.008653    -0.001485
ejection_fraction         0.002126  0.086484    -0.271767
high_blood_pressure      -0.078545 -0.219173     0.113721
platelets                 0.043759 -0.001018    -0.044523
serum_creatinine          0.020209 -0.165679     0.290229
serum_sodium              0.011111  0.130820    -0.250990
sex                       0.411603  0.017673     0.044045
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'correlation_matrix' is your calculated correlation matrix
# Replace 'correlation_matrix' with the name of your correlation matrix if different

# Create a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

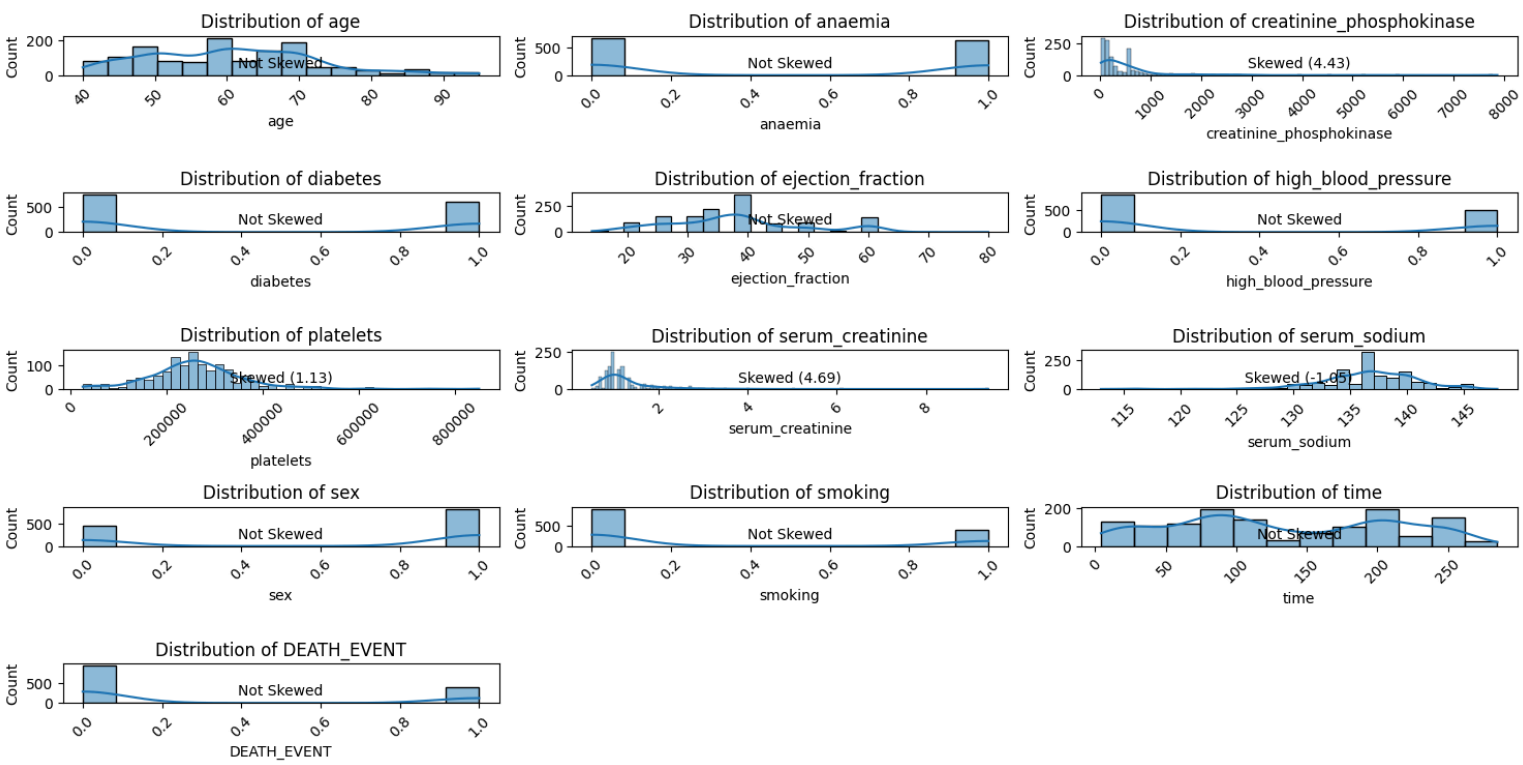


Statistical Analysis

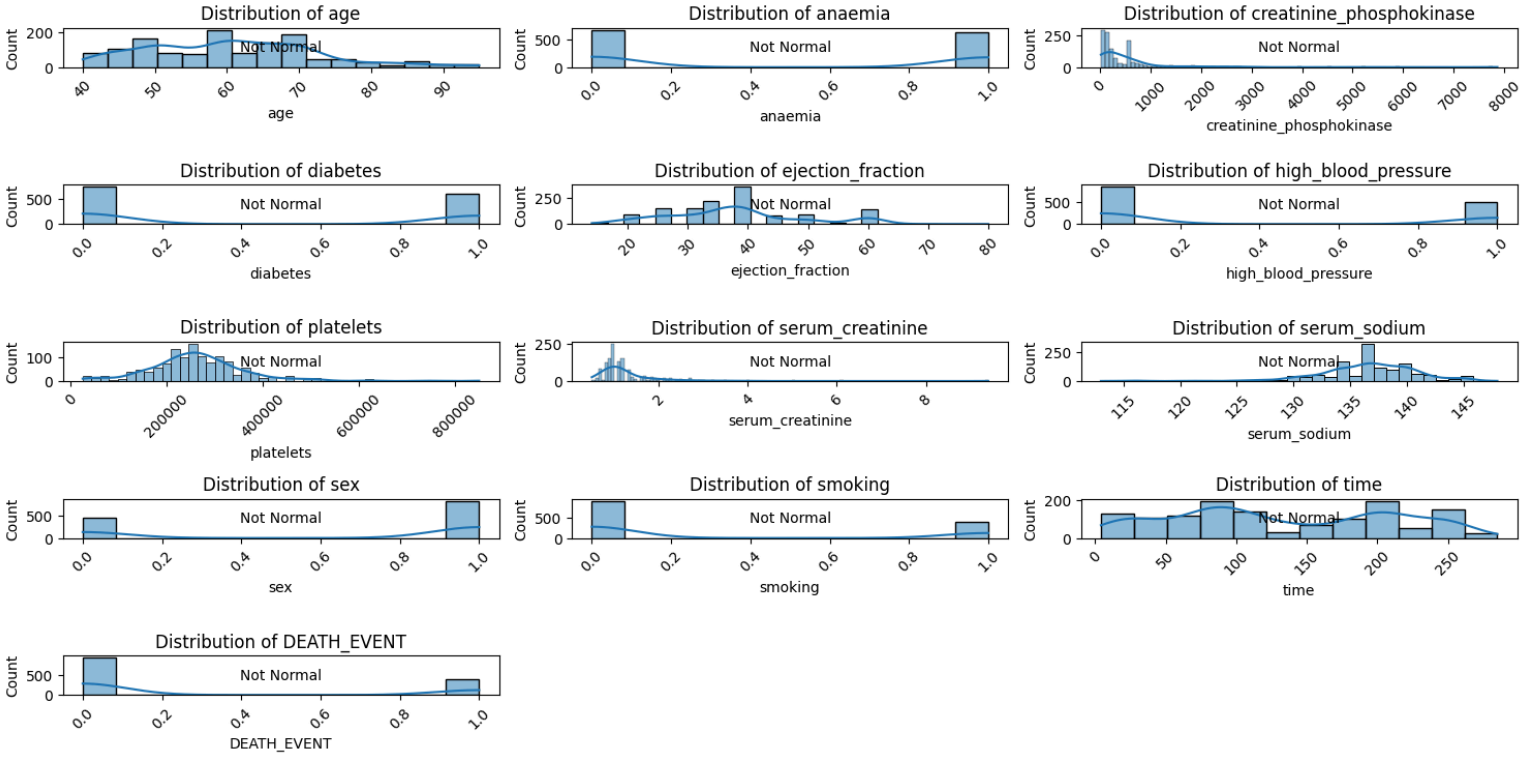
```
from scipy import stats
from scipy.stats import zscore


plt.figure(figsize=(15, 10))

# Iterate through each channel and plot on a separate subplot
for i, column in enumerate(data.columns):
    plt.subplot(7, 3, i+1)
    sns.histplot(data[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.xticks(rotation=45)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```




```
plt.figure(figsize=(15, 10))

# Iterate through each column and plot on a separate subplot
for i, column in enumerate(data.columns):
    plt.subplot(7, 3, i+1)
    sns.histplot(data[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.xticks(rotation=45)

    # Check for skewness
    skewness = stats.skew(data[column])
    if skewness < -1 or skewness > 1:
        plt.text(0.5, 0.3, f"Skewed ({skewness:.2f})", horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)
    else:
        plt.text(0.5, 0.3, f"Not Skewed", horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



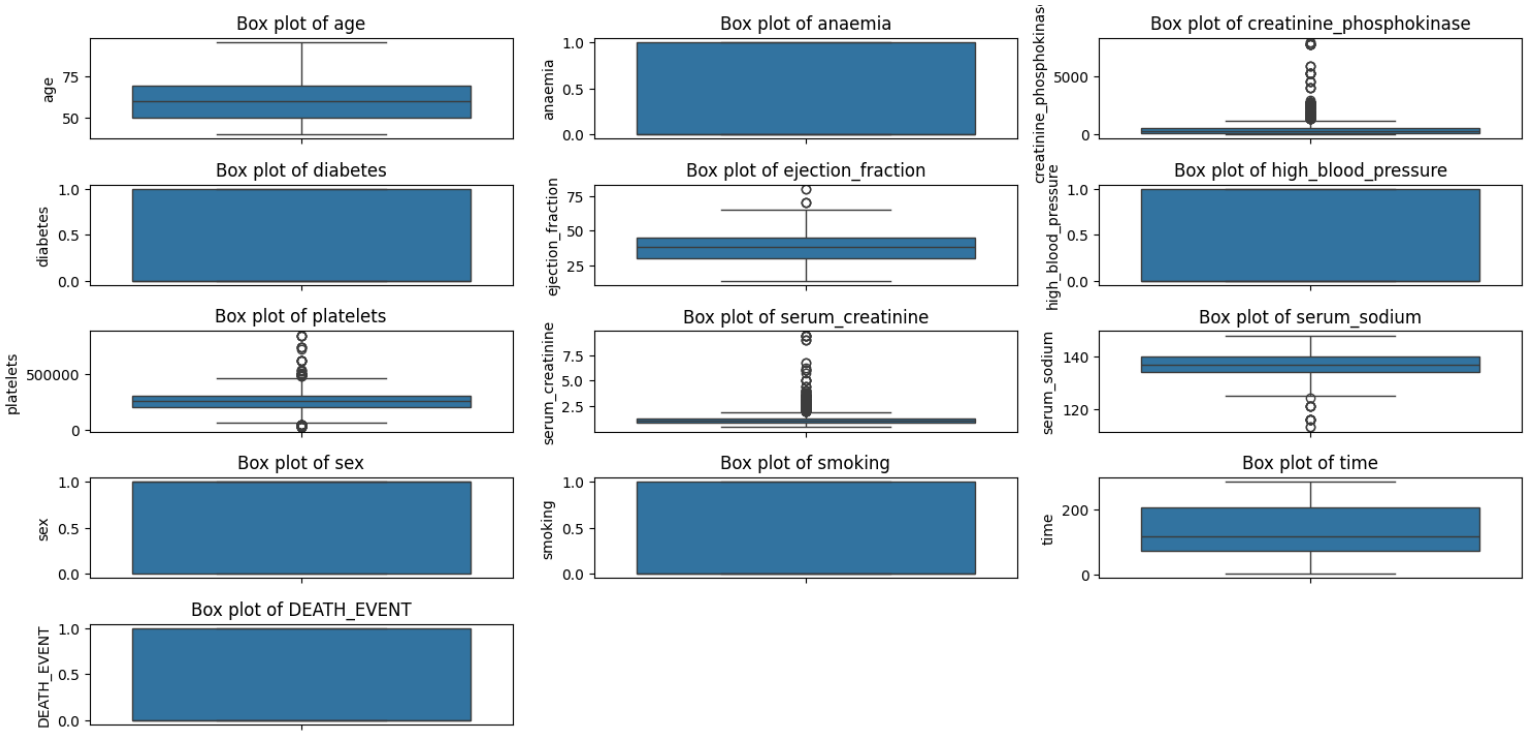

```
plt.figure(figsize=(15, 10))

# Iterate through each column and plot on a separate subplot
for i, column in enumerate(data.columns):
    plt.subplot(7, 3, i+1)
    sns.histplot(data[column], kde=True)
    plt.title(f'Distribution of {column}')
    plt.xticks(rotation=45)

    # Add additional analysis to detect distribution type
    # Check for normal distribution
    k2, p = stats.normaltest(data[column])
    if p < 0.05:
        plt.text(0.5, 0.5, "Not Normal", horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)
    else:
        plt.text(0.5, 0.5, "Normal", horizontalalignment='center', verticalalignment='center', transform=plt.gca().transAxes)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```




```
# Function to detect outliers using Z-score for a specific column
def detect_outliers_z_score(data, threshold=3):
    z_scores = (data - data.mean()) / data.std()
    outliers = (np.abs(z_scores) > threshold).any(axis=1)
    return outliers


# Detect outliers for each numeric column
outliers = detect_outliers_z_score(data[data.columns])

# Print indices of rows containing outliers
outlier_indices = data.index[outliers].tolist()
print("Index of rows with outliers:", outlier_indices)
print("Count of outliers:", len(outlier_indices))
```

```
Index of rows with outliers: [1, 41, 60, 79, 85, 89, 118, 137, 147, 150, 173, 176, 188, 217, 220, 241, 260, 273, 275, 296, 370, 372, 451, 453,
Count of outliers: 86
```

```
plt.figure(figsize=(15, 10))

# Iterate through each column and plot on a separate subplot
for i, column in enumerate(data.columns):
    plt.subplot(7, 3, i+1)
    sns.boxplot(data[column])
    plt.title(f'Box plot of {column}')

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```




```
# Remove outliers based on Z-score for a specific column
def remove_outliers_z_score_column(data_column, threshold=3):
    z_scores = zscore(data_column)
    filtered_data = data_column[np.abs(z_scores) <= threshold]
    return filtered_data


updated_data = remove_outliers_z_score_column(data)


# Detect outliers for each numeric column
outliers = detect_outliers_z_score(updated_data[updated_data.columns])

# Print indices of rows containing outliers
outlier_indices = updated_data.index[outliers].tolist()
print("Index of rows with outliers:", outlier_indices)
print("Count of outliers:", len(outlier_indices))
```

```
Index of rows with outliers: [12, 29, 38, 70, 120, 123, 154, 172, 192, 197, 199, 233, 243, 246, 265, 279, 292, 298, 358, 389, 419, 421, 433, 4
Count of outliers: 83
```

```
# Make a deep copy of the original DataFrame
data_analysis = updated_data.copy(deep=True)
```

Machine Learning Algorithms

Define X and Y

```
X = data.drop( 'DEATH_EVENT', axis=1)
y = data[ 'DEATH_EVENT']
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
# Normalize the features
min_max_scaler = MinMaxScaler()
min_max_scaler.fit(X)
scaled_features = min_max_scaler.transform(X)


scaled_features = pd.DataFrame(scaled_features,columns=data.columns[:-1])
scaled_features.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	se
0	0.272727	0.0	0.092498	0.0	0.469697	0.0	0.288833	0.089888	0.685714	1.
1	0.454545	0.0	0.004210	0.0	0.166667	0.0	0.339314	0.505618	0.485714	1.
2	0.090909	0.0	0.071319	1.0	0.363636	0.0	0.356286	0.044944	0.771429	0.
3	0.363636	1.0	0.093264	1.0	0.393939	1.0	0.367196	0.078652	0.371429	1.
4	1.000000	1.0	0.071319	0.0	0.242424	0.0	0.528428	0.168539	0.542857	1.

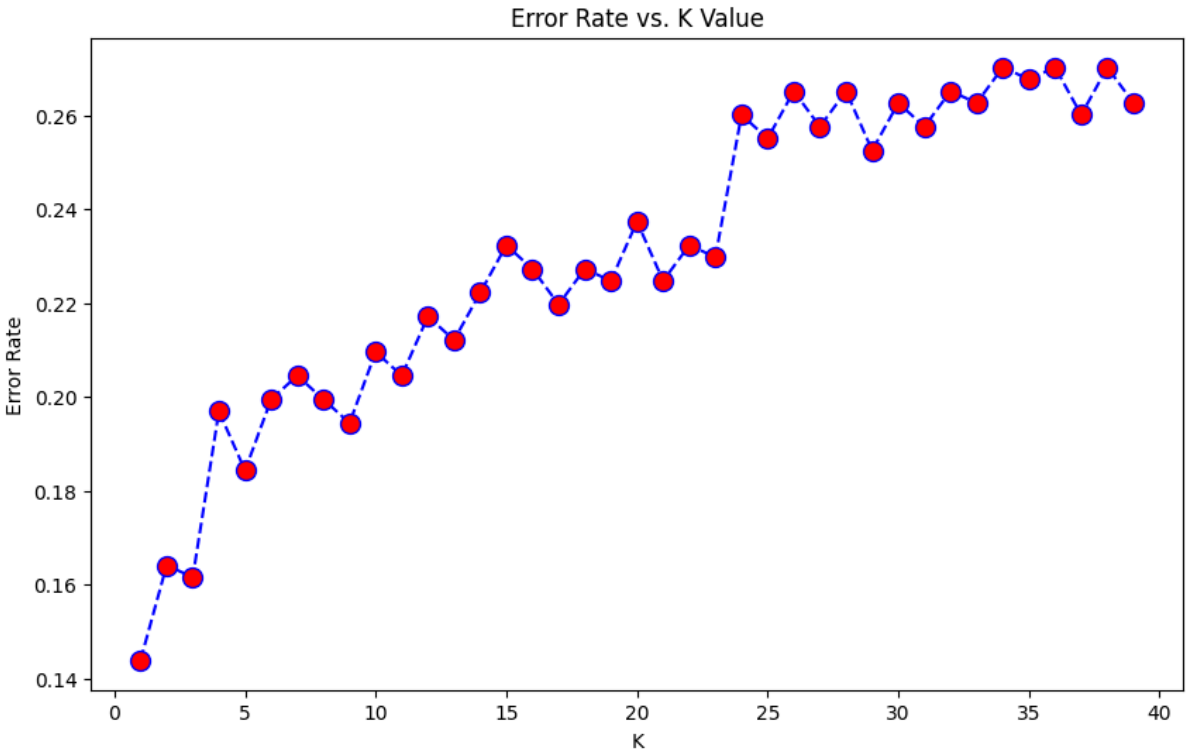
Train Test Split

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(scaled_features,  y,  test_size=0.30)
```

KNN

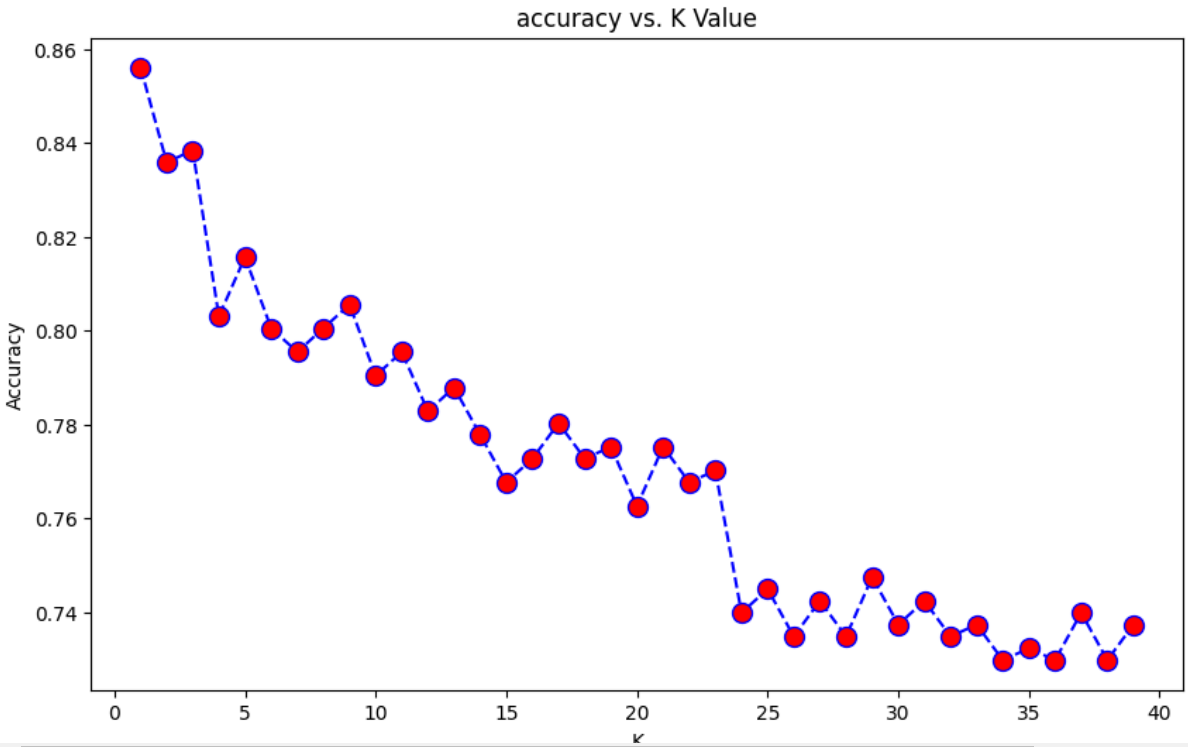
```
from sklearn.neighbors import KNeighborsClassifier


from sklearn.metrics import classification_report,confusion_matrix


error_rate = []
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:",min(error_rate),"at K =",error_rate.index(min(error_rate)))
```

```
Minimum error: 0.14393939393939395 at K = 0
```




```
acc = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors = i).fit(X_train,y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:",max(acc),"at K =",acc.index(max(acc)))
```

```
Maximum accuracy: 0.8560606060606061 at K = 0
```




```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)


#Training Accuracy
print(knn.score(X_train, y_train))
#Testing Accuracy
print(knn.score(X_test, y_test))
```

```
0.9134199134199135
0.8383838383838383
```

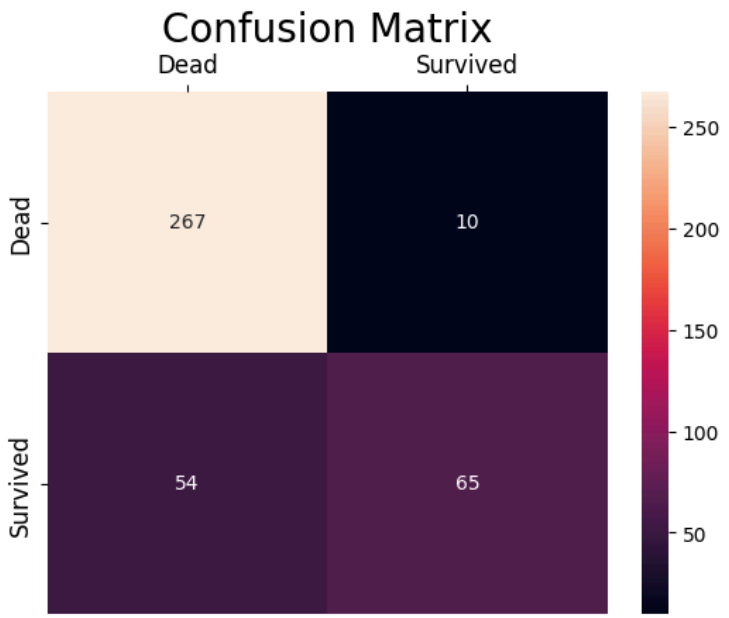
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89       277
           1       0.87      0.55      0.67       119

    accuracy                           0.84       396
   macro avg       0.85      0.76      0.78       396
weighted avg       0.84      0.84      0.83       396
```

```
print(confusion_matrix(y_test,pred))
```

```
[[267  10]
 [ 54  65]]
```

```
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test,pred), annot=True, ax = ax, fmt = 'g');
ax.set_title('Confusion Matrix', fontsize=20)
# assuming 0 means death
ax.xaxis.set_ticklabels(['Dead', 'Survived'], fontsize = 12)
ax.xaxis.tick_top()

ax.yaxis.set_ticklabels(['Dead', 'Survived'], fontsize = 12)
plt.show()
```



Decision Tree

Using Entropy

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split


# Initializing and training the Decision Tree Classifier with Gini impurity
dt_gini = DecisionTreeClassifier(criterion='gini', random_state=42)
dt_gini.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

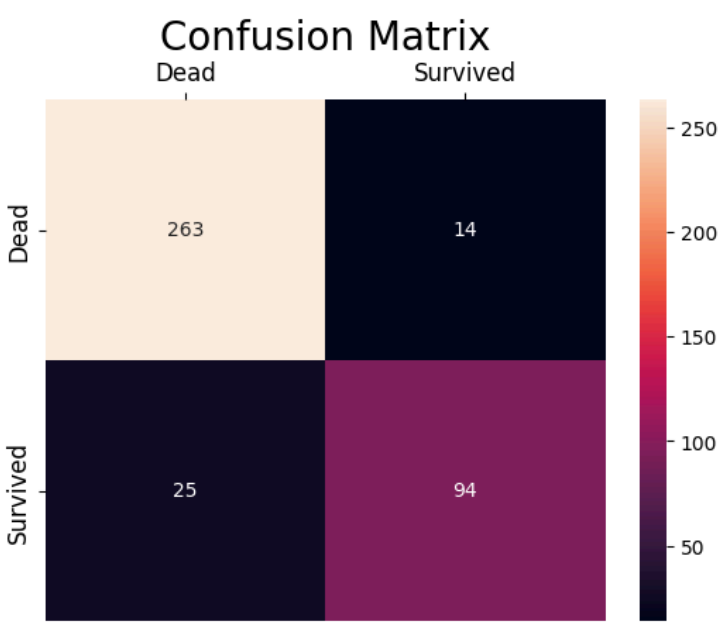
```
# Making predictions and evaluating the models
y_pred_gini = dt_gini.predict(X_test)

accuracy_gini = accuracy_score(y_test, y_pred_gini)

accuracy_gini
```

```
0.9015151515151515
```

```
print('Classification Report Decsion Tree Entropy:')
print(classification_report(y_test, y_pred_gini))
```

```
Classification Report Decsion Tree Entropy:
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       277
           1       0.87      0.79      0.83       119

    accuracy                           0.90       396
   macro avg       0.89      0.87      0.88       396
weighted avg       0.90      0.90      0.90       396
```

```
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, y_pred_gini), annot=True, ax = ax, fmt = 'g');
ax.set_title('Confusion Matrix', fontsize=20)

ax.xaxis.set_ticklabels(['Dead', 'Survived'], fontsize = 12)
ax.xaxis.tick_top()

ax.yaxis.set_ticklabels(['Dead', 'Survived'], fontsize = 12)
plt.show()
```




```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, roc_auc_score, roc_curve

# Evaluation metrics for Gini model
confusion_gini = confusion_matrix(y_test, y_pred_gini)
precision_gini = precision_score(y_test, y_pred_gini)
recall_gini = recall_score(y_test, y_pred_gini)
f1_score_gini = f1_score(y_test, y_pred_gini)
roc_auc_gini = roc_auc_score(y_test, y_pred_gini)
roc_auc_entropy = roc_auc_score(y_test, y_pred_gini)

# Printing the evaluation metrics
print("Gini Model Evaluation Metrics:")
print("Confusion Matrix:\n", confusion_gini)
print("Precision: {:.2f}".format(precision_gini))
print("Recall: {:.2f}".format(recall_gini))
print("F1 Score: {:.2f}".format(f1_score_gini))
print("ROC AUC: {:.2f}".format(roc_auc_gini))
```

```
Gini Model Evaluation Metrics:
Confusion Matrix:
 [[263  14]
 [ 25  94]]
Precision: 0.87
Recall: 0.79
F1 Score: 0.83
ROC AUC: 0.87
```

```
# Initializing and training the Decision Tree Classifier with Information Gain (Entropy)
dt_entropy = DecisionTreeClassifier(criterion='entropy', random_state=42)
dt_entropy.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=42)
```

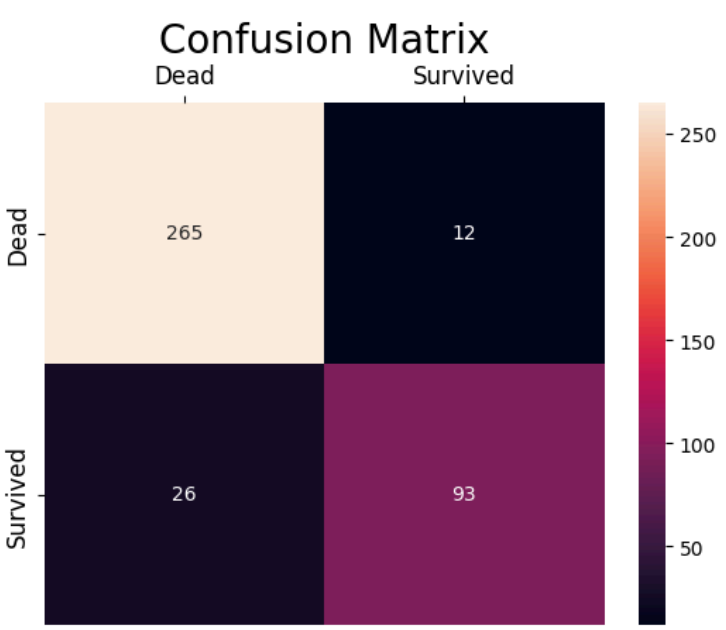
```
# Making predictions and evaluating the models
y_pred_entropy = dt_entropy.predict(X_test)

accuracy_entropy = accuracy_score(y_test, y_pred_entropy)

accuracy_entropy
```

```
0.9040404040404041
```

```
print('Classification Report Decsion Tree Entropy:')
print(classification_report(y_test, y_pred_entropy))
```

```
Classification Report DT Entropy:
                precision    recall  f1-score   support

             0       0.91      0.96      0.93       277
             1       0.89      0.78      0.83       119

      accuracy                           0.90       396
     macro avg       0.90      0.87      0.88       396
  weighted avg       0.90      0.90      0.90       396
```

```
ax= plt.subplot()
sns.heatmap(confusion_matrix(y_test, y_pred_entropy), annot=True, ax = ax, fmt = 'g');
ax.set_title('Confusion Matrix', fontsize=20)

ax.xaxis.set_ticklabels(['Dead', 'Survived'], fontsize = 12)
ax.xaxis.tick_top()

ax.yaxis.set_ticklabels(['Dead', 'Survived'], fontsize = 12)
plt.show()
```




```
# Evaluation metrics for Entropy model
confusion_entropy = confusion_matrix(y_test, y_pred_entropy)
precision_entropy = precision_score(y_test, y_pred_entropy)
recall_entropy = recall_score(y_test, y_pred_entropy)
f1_score_entropy = f1_score(y_test, y_pred_entropy)
roc_auc_entropy = roc_auc_score(y_test, y_pred_entropy)

print("\nEntropy Model Evaluation Metrics:")
print("Confusion Matrix:\n", confusion_entropy)
print("Precision: {:.2f}".format(precision_entropy))
print("Recall: {:.2f}".format(recall_entropy))
print("F1 Score: {:.2f}".format(f1_score_entropy))
print("ROC AUC: {:.2f}".format(roc_auc_entropy))
```

```
Entropy Model Evaluation Metrics:
Confusion Matrix:
 [[265  12]
 [ 26  93]]
Precision: 0.89
Recall: 0.78
F1 Score: 0.83
```

```
# ROC curve calculations
fpr_gini, tpr_gini, _ = roc_curve(y_test, y_pred_gini)
fpr_entropy, tpr_entropy, _ = roc_curve(y_test, y_pred_entropy)

# Plotting ROC curves
plt.figure(figsize=(10, 6))
plt.plot(fpr_gini, tpr_gini, label='Gini - AUC: {:.3f}'.format(roc_auc_gini))
plt.plot(fpr_entropy, tpr_entropy, label='Entropy - AUC: {:.3f}'.format(roc_auc_entropy))
plt.plot([0, 1], [0, 1], 'k--') # Dashed diagonal line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
```