Importing Libraries

In [ ]:

```python
#importing basic modules
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#importing modules required model building
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import confusion_matrix, precision_score, recall_score,
from scipy.stats import zscore

#importing models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier

#importing system modules to avoid warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the Data

In [ ]:

```python
heart_data = pd.read_csv('SuddenCardiacArrest.csv')
```

In [ ]:

```python
heart_data.head(5)
```

Out[ ]:

	PatientName	Age	Sex	ECG-Resting	ST-Slope	BloodPressure-Resting	HeartRate-Max	ChestPainTyp
0	Patient 1	40	M	Normal	Up	140	172	AT
1	Patient 2	49	F	Normal	Flat	160	156	NA
2	Patient 3	37	M	ST	Up	130	98	AT
3	Patient 4	48	F	Normal	Flat	138	108	AS
4	Patient 5	54	M	Normal	Up	150	122	NA

EDA

Removing identifiable features:

- Patient Name was the only identifiable feature

```
In [ ]: heart_data = heart_data.drop('PatientName', axis = 1)
        heart_data.head(5)
```

Out[ ]:

	Age	Sex	ECG-Resting	ST-Slope	BloodPressure-Resting	HeartRate-Max	ChestPainType	Cholesterol
0	40	M	Normal	Up	140	172	ATA	289
1	49	F	Normal	Flat	160	156	NAP	180
2	37	M	ST	Up	130	98	ATA	283
3	48	F	Normal	Flat	138	108	ASY	214
4	54	M	Normal	Up	150	122	NAP	195

Data Dimensions

```
In [ ]: heart_data.shape
```

```
Out[ ]: (1221, 12)
```

Data Types

```
In [ ]: heart_data.dtypes
```

```
Out[ ]: Age                        int64
        Sex                       object
        ECG-Resting               object
        ST-Slope                  object
        BloodPressure-Resting      int64
        HeartRate-Max              int64
        ChestPainType             object
        Cholesterol                int64
        BloodSugar-Fasting        object
        ExerciseAngina            object
        OldPeak                  float64
        SCA                        int64
        dtype: object
```

Summary Statistics

```
In [ ]: heart_data.describe()
```

Out[ ]:

	Age	BloodPressure-Resting	HeartRate-Max	Cholesterol	OldPeak	S
count	1221.000000	1221.000000	1221.000000	1221.000000	1221.000000	1221.0000
mean	53.741196	132.221130	139.985258	210.684685	0.925143	0.5298
std	9.341351	18.286927	25.443021	100.425185	1.092282	0.499
min	28.000000	0.000000	60.000000	0.000000	-2.600000	0.0000
25%	47.000000	120.000000	122.000000	188.000000	0.000000	0.0000
50%	54.000000	130.000000	141.000000	228.000000	0.600000	1.0000
75%	60.000000	140.000000	160.000000	269.000000	1.600000	1.0000
max	77.000000	200.000000	202.000000	603.000000	6.200000	1.0000

Understanding the data

In [ ]:

```
target_column = 'SCA'
heart_data[target_column].value_counts()
```

Out[ ]:

```
SCA
1    647
0    574
Name: count, dtype: int64
```

In [ ]:

```
features = heart_data.drop(columns='SCA')
features.head()
```

Out[ ]:

	Age	Sex	ECG-Resting	ST-Slope	BloodPressure-Resting	HeartRate-Max	ChestPainType	Cholesterol
0	40	M	Normal	Up	140	172	ATA	289
1	49	F	Normal	Flat	160	156	NAP	180
2	37	M	ST	Up	130	98	ATA	283
3	48	F	Normal	Flat	138	108	ASY	214
4	54	M	Normal	Up	150	122	NAP	195

In [ ]:

```
numeric_columns = features.select_dtypes(include=np.number).columns.values
categorical_columns = features.drop(columns=numeric_columns).columns.values

print(f'''
There are {features.shape[0]} observations and {features.shape[1]} features.

Numeric features: {', '.join(numeric_columns)}.

Categorical features: {', '.join(categorical_columns)}.
''')
```

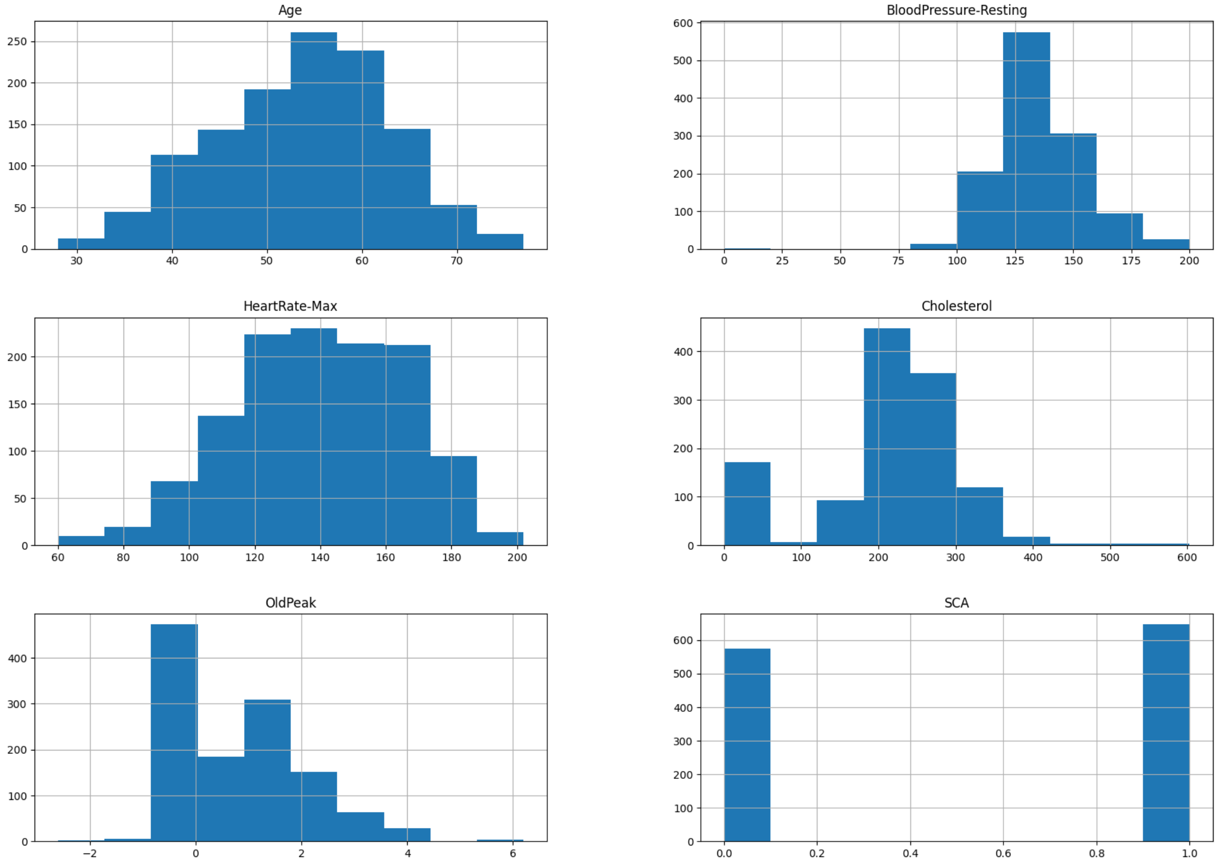
```
There are 1221 observations and 11 features.

Numeric features: Age, BloodPressure-Resting, HeartRate-Max, Cholesterol, Ol
dPeak.

Categorical features: Sex, ECG-Resting, ST-Slope, ChestPainType, BloodSugar-
Fasting, ExerciseAngina.
```

In [ ]:

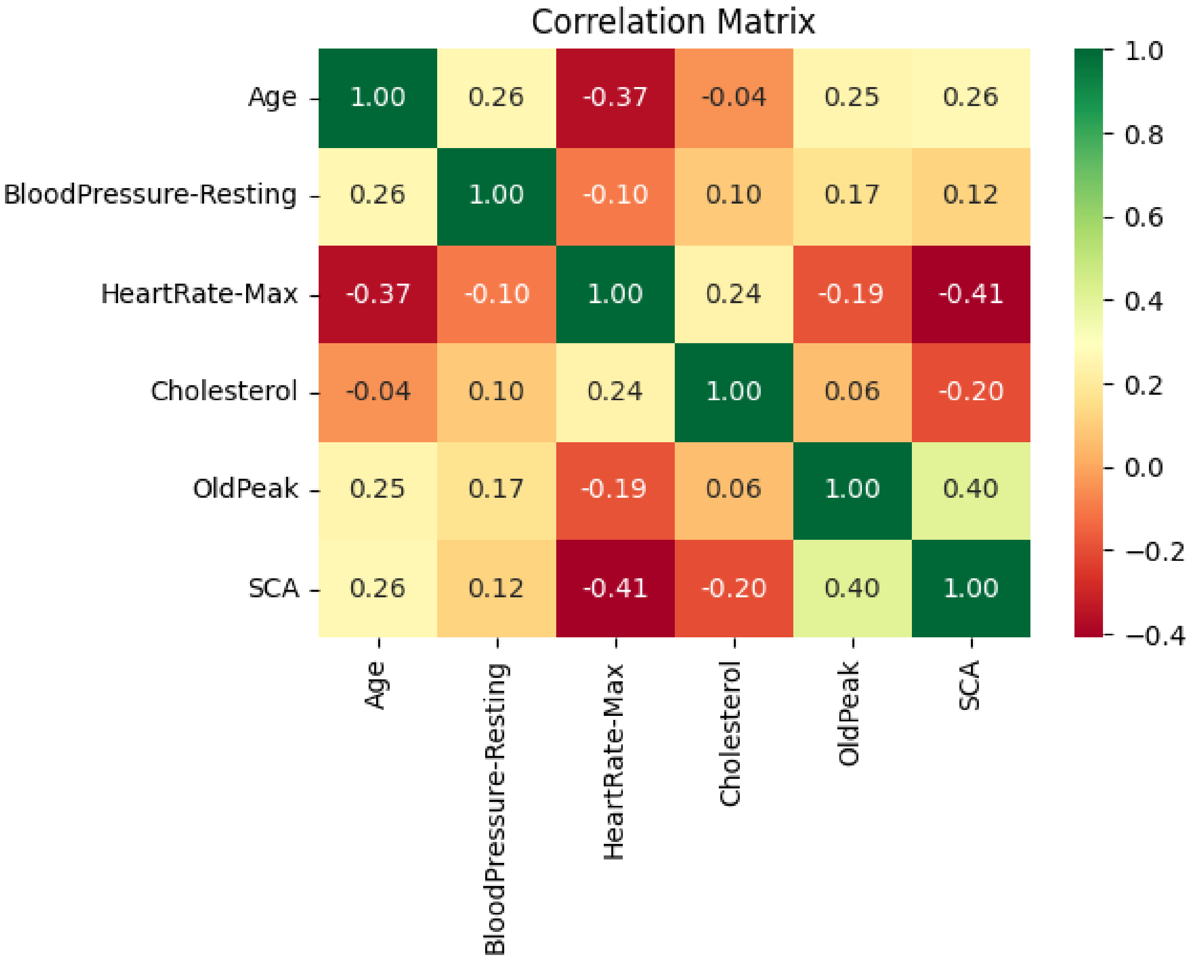
```python
heart_data.hist(figsize=(20, 14));
```



Correlation Plots for numerical data

In [ ]:

```python
plt.figure(figsize=(6,4))
sns.heatmap(heart_data[['Age','BloodPressure-Resting', 'HeartRate-Max', 'Cho
plt.title("Correlation Matrix")
plt.show()
```



For categorical variables, we check the distribution by taking the count of each category/group

In [ ]:
```
heart_data['Sex'].value_counts()
```

Out[ ]:
```
Sex
M    931
F    290
Name: count, dtype: int64
```

In [ ]:
```
heart_data['ECG-Resting'].value_counts()
```

Out[ ]:
```
ECG-Resting
Normal    703
LVH       336
ST        182
Name: count, dtype: int64
```

In [ ]:
```
heart_data['ST-Slope'].value_counts()
```

Out[ ]:
```
ST-Slope
Flat    600
Up      537
Down     84
Name: count, dtype: int64
```

In [ ]:

```
heart_data['ChestPainType'].value_counts()
```

Out[ ]:

```
ChestPainType
ASY    640
NAP    289
ATA    223
TA      69
Name: count, dtype: int64
```

In [ ]:

```
heart_data['BloodSugar-Fasting'].value_counts()
```

Out[ ]:

```
BloodSugar-Fasting
Normal    962
High      259
Name: count, dtype: int64
```

In [ ]:

```
heart_data['ExerciseAngina'].value_counts()
```

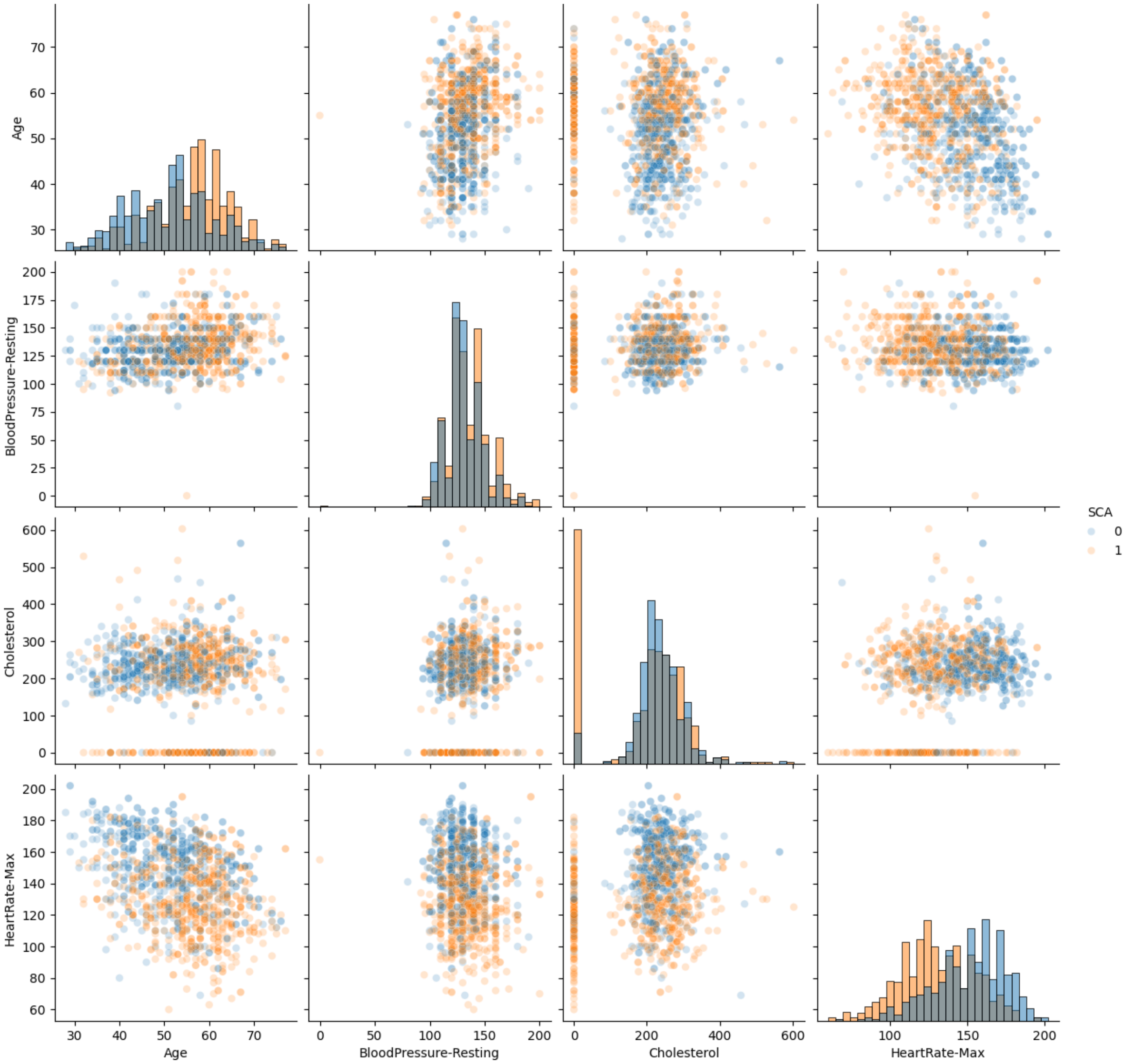
Out[ ]:

```
ExerciseAngina
N    751
Y    470
Name: count, dtype: int64
```

Plotting the data against target variable for inspection

In [ ]:

```
columns = ['Age', 'BloodPressure-Resting', 'Cholesterol','HeartRate-Max']
sns.pairplot(data=heart_data, vars=columns,
             hue=target_column, plot_kws={'alpha': 0.2},
             height=3, diag_kind='hist', diag_kws={'bins': 30});
```



Data Preprocessing and Wrangling

Missing values

```
In [ ]: heart_data.isnull().sum()
```

Out[ ]:

```
Age                     0
Sex                     0
ECG-Resting             0
ST-Slope                0
BloodPressure-Resting   0
HeartRate-Max           0
ChestPainType           0
Cholesterol             0
BloodSugar-Fasting      0
ExerciseAngina          0
OldPeak                 0
SCA                     0
dtype: int64
```

There are no null values to be handled. **But, there are some rows that have either Resting BP or Cholesterol set to 0. We are removing those rows for cleaner data**

In [ ]:

```
heart_data = heart_data.loc[(heart_data['BloodPressure-Resting'] != 0) & (he

heart_data
```

Out[ ]:

	Age	Sex	ECG-Resting	ST-Slope	BloodPressure-Resting	HeartRate-Max	ChestPainType	Cholest
0	40	M	Normal	Up	140	172	ATA	
1	49	F	Normal	Flat	160	156	NAP	
2	37	M	ST	Up	130	98	ATA	
3	48	F	Normal	Flat	138	108	ASY	
4	54	M	Normal	Up	150	122	NAP	
...	...	...	...	...	...	...	...	
1216	45	M	Normal	Flat	110	132	TA	
1217	68	M	Normal	Flat	144	141	ASY	
1218	57	M	Normal	Flat	130	115	ASY	
1219	57	F	LVH	Flat	130	174	ATA	
1220	38	M	Normal	Up	138	173	NAP	

1049 rows × 12 columns

Duplicate Data

In [ ]:

```
duplicate_rows = heart_data.duplicated().sum()
print("There are", duplicate_rows, "duplicate rows")
```

```
There are 303 duplicate rows
```

In [ ]:

```
# Removing duplicate rows
heart_data = heart_data.drop_duplicates()
```

```
#Checking once again
duplicate_rows = heart_data.duplicated().sum()
print("After removing, there are", duplicate_rows, "duplicate rows")
```

```
After removing, there are 0 duplicate rows
```

Feature Engineering

Adding a new column 'HeartRisk' which is calculated using Age, Resting BP, Max Heart Rate and Cholesterol using the formula below:

- Risk = Age/(BloodPressure-Resting + Cholesterol + HeartRate-Max)

In [ ]:

```
heart_data['HeartRisk'] = heart_data['Age']/(heart_data['BloodPressure-Resti
heart_data.head()
```

Out[ ]:

	Age	Sex	ECG-Resting	ST-Slope	BloodPressure-Resting	HeartRate-Max	ChestPainType	Cholesterol
0	40	M	Normal	Up	140	172	ATA	289
1	49	F	Normal	Flat	160	156	NAP	180
2	37	M	ST	Up	130	98	ATA	283
3	48	F	Normal	Flat	138	108	ASY	214
4	54	M	Normal	Up	150	122	NAP	195

Outliers

In [ ]:

```
# Calculate Z-scores for numerical features
z_scores = zscore(heart_data.select_dtypes(include=np.number))
z_scores
```

Out[ ]:

	Age	BloodPressure-Resting	HeartRate-Max	Cholesterol	OldPeak	SCA	Heart
0	-1.356073	0.403980	1.296470	0.750494	-0.840942	-0.955416	-1.58
1	-0.408656	1.561980	0.643613	-1.093405	0.091771	1.046664	-0.21
2	-1.671879	-0.175019	-1.722993	0.648995	-0.840942	-0.955416	-1.33
3	-0.513925	0.288180	-1.314958	-0.518244	0.558127	1.046664	0.02
4	0.117686	0.982980	-0.743708	-0.839657	-0.840942	-0.955416	0.50
...	...	...	...	...	...	...	
913	-0.829731	-1.333019	-0.335672	0.327582	0.278313	1.046664	-0.63
914	1.591446	0.635580	0.031560	-0.873490	2.330281	1.046664	1.63
915	0.433492	-0.175019	-1.029333	-1.922314	0.278313	1.046664	2.03
916	0.433492	-0.175019	1.378077	-0.146081	-0.840942	1.046664	0.07
917	-1.566610	0.288180	1.337274	-1.177987	-0.840942	-0.955416	-1.09

746 rows × 7 columns

In [ ]:

```
threshold = 4
outliers = np.abs(z_scores) > threshold
outliers
```

Out[ ]:

	Age	BloodPressure-Resting	HeartRate-Max	Cholesterol	OldPeak	SCA	HeartRisk
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
913	False	False	False	False	False	False	False
914	False	False	False	False	False	False	False
915	False	False	False	False	False	False	False
916	False	False	False	False	False	False	False
917	False	False	False	False	False	False	False

746 rows × 7 columns

In [ ]:

```
# Remove outliers or handle them as needed
df_no_outliers = heart_data[~outliers.any(axis=1)]
```

```
df_no_outliers
```

Out[ ]:

	Age	Sex	ECG-Resting	ST-Slope	BloodPressure-Resting	HeartRate-Max	ChestPainType	Choleste
0	40	M	Normal	Up	140	172	ATA	2
1	49	F	Normal	Flat	160	156	NAP	1
2	37	M	ST	Up	130	98	ATA	2
3	48	F	Normal	Flat	138	108	ASY	2
4	54	M	Normal	Up	150	122	NAP	1
...	...	...	...	...	...	...	...	
913	45	M	Normal	Flat	110	132	TA	2
914	68	M	Normal	Flat	144	141	ASY	1
915	57	M	Normal	Flat	130	115	ASY	1
916	57	F	LVH	Flat	130	174	ATA	2
917	38	M	Normal	Up	138	173	NAP	1

738 rows × 13 columns

Categorical Data Encoding

In [ ]:

```
df_encoded = pd.get_dummies(df_no_outliers, columns=['Sex','ECG-Resting','ST
df_encoded.head()
```

Out[ ]:

	Age	BloodPressure-Resting	HeartRate-Max	Cholesterol	OldPeak	SCA	HeartRisk	Sex_F	S
0	40	140	172	289	0.0	0	0.066556	False	
1	49	160	156	180	1.0	1	0.098790	True	
2	37	130	98	283	0.0	0	0.072407	False	
3	48	138	108	214	1.5	1	0.104348	True	
4	54	150	122	195	0.0	0	0.115632	False	

5 rows × 23 columns

Feature Scaling

In [ ]:

```
scaler = StandardScaler()
df_encoded[['Age', 'BloodPressure-Resting', 'Cholesterol', 'OldPeak', 'Heart
    df_encoded[['Age', 'BloodPressure-Resting', 'Cholesterol', 'OldPeak', 'H

df_encoded
```

Out[ ]:

	Age	BloodPressure-Resting	HeartRate-Max	Cholesterol	OldPeak	SCA	HeartRisk
0	-1.359533	0.402577	172	0.856427	-0.853378	0	-1.631185
1	-0.407702	1.560419	156	-1.170842	0.108579	1	-0.222797
2	-1.676810	-0.176343	98	0.744834	-0.853378	0	-1.375531
3	-0.513461	0.286793	108	-0.538483	0.589557	1	0.020021
4	0.121093	0.981498	122	-0.891860	-0.853378	0	0.513034
...	...	...	...	...	...	...	...
913	-0.830738	-1.334185	132	0.391457	0.300970	1	-0.653490
914	1.601718	0.634146	141	-0.929058	2.417273	1	1.676448
915	0.438369	-0.176343	115	-2.082184	0.300970	1	2.084370
916	0.438369	-0.176343	174	-0.129310	-0.853378	1	0.072789
917	-1.571051	0.286793	173	-1.263836	-0.853378	0	-1.122893

738 rows × 23 columns

Model Building and Evalauations

Evaluation module

In [ ]:

```python
def evaluate(model, x_test, y_test, average='weighted'):
    y_pred = model.predict(x_test)
    acc = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average=average)
    recall = recall_score(y_test, y_pred, average=average)
    f1 = f1_score(y_test, y_pred, average=average)
    scores = cross_val_score(model, x, y, cv=5, scoring='accuracy')
    print(f'Accuracy: {acc:.2f}')
    print(f'Precision: {precision:.2f}')
    print(f'Recall: {recall:.2f}')
    print(f'F1-score: {f1:.2f}')
    print(f"{type(model).__name__} Cross-Validation Accuracy: {np.mean(score
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title('Confusion Matrix')
    plt.show()
```

Preparing the data for models

In [ ]:

```python
x = df_encoded.drop("SCA", axis=1)
y = df_encoded["SCA"]
```

In [ ]:

```python
# Split data into training and testing sets
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, rar
```

In [ ]:

```
x_train = np.array(x_train)
x_test = np.array(x_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

Decision Tree Model

In [ ]:

```
decision_tree = DecisionTreeClassifier(random_state=142)
decision_tree.fit(x_train, y_train)
decision_tree
```

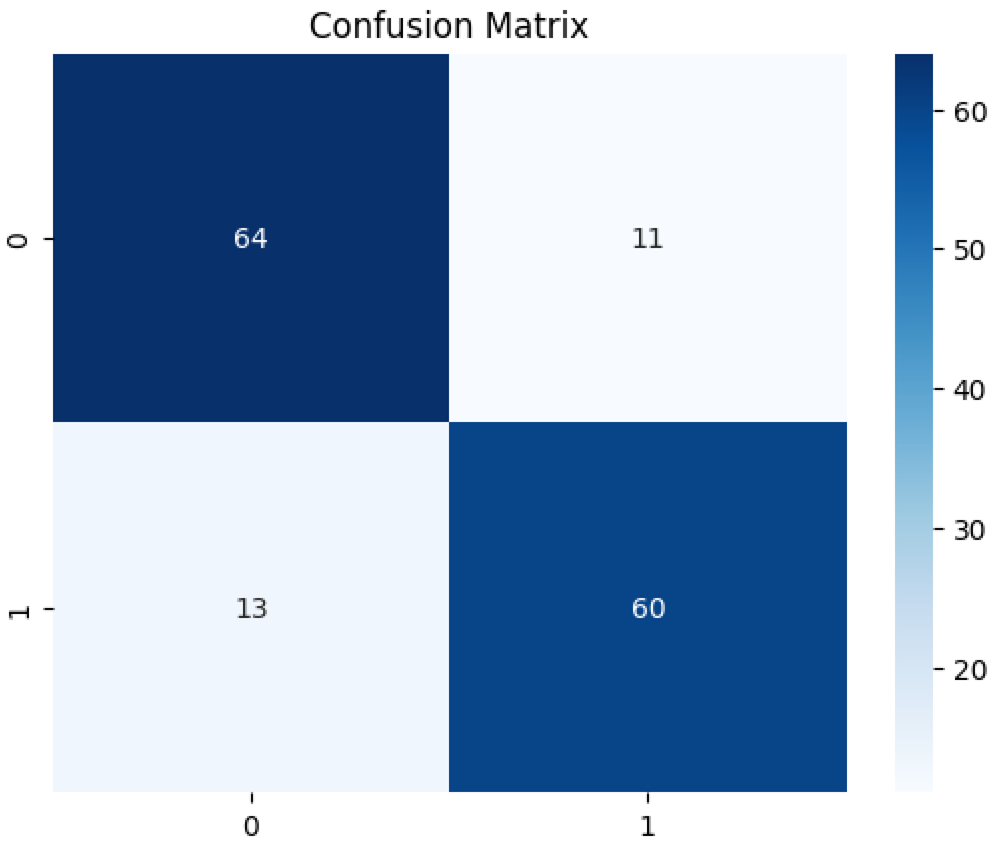
Out[ ]:

```
▾        DecisionTreeClassifier
DecisionTreeClassifier(random_state=142)
```

In [ ]:

```
evaluate(decision_tree, x_test, y_test, average='weighted')
```

```
Accuracy: 0.84
Precision: 0.84
Recall: 0.84
F1-score: 0.84
DecisionTreeClassifier Cross-Validation Accuracy: 0.78
```



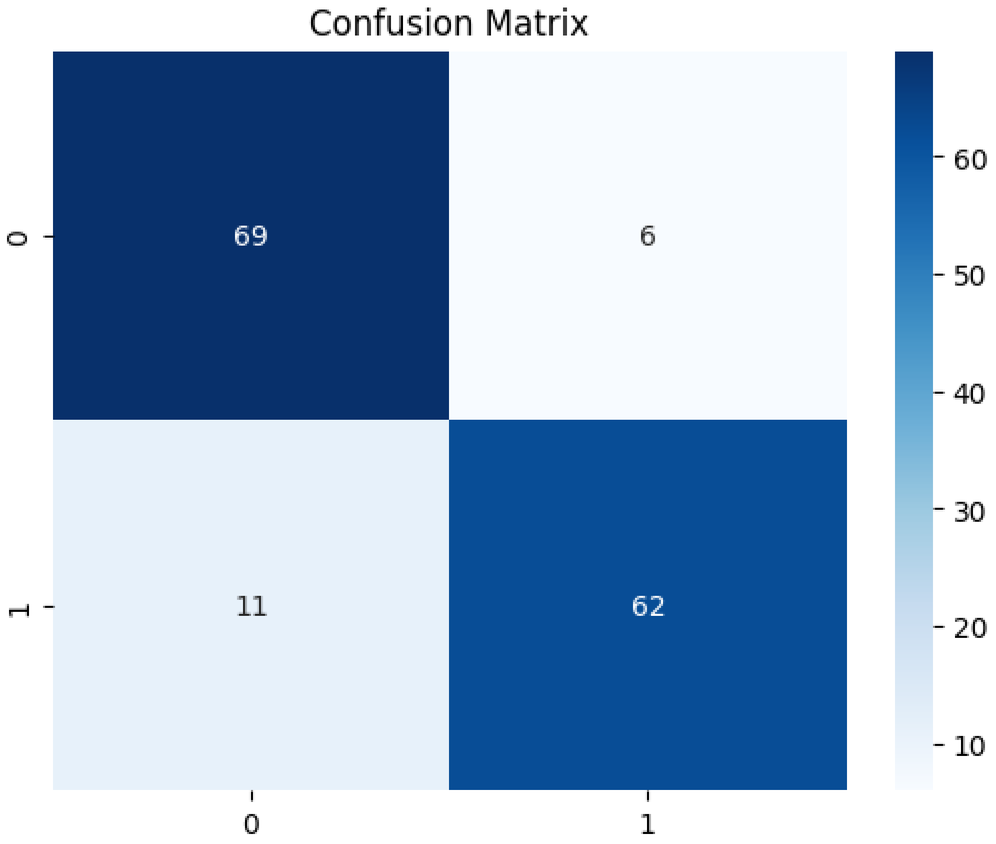
Random Forest

In [ ]:
```
random_forest = RandomForestClassifier(random_state=42)
random_forest.fit(x_train, y_train)
```

Out[ ]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [ ]:
```
evaluate(random_forest, x_test, y_test)
```

```
Accuracy: 0.89
Precision: 0.89
Recall: 0.89
F1-score: 0.88
RandomForestClassifier Cross-Validation Accuracy: 0.86
```


Confusion Matrix

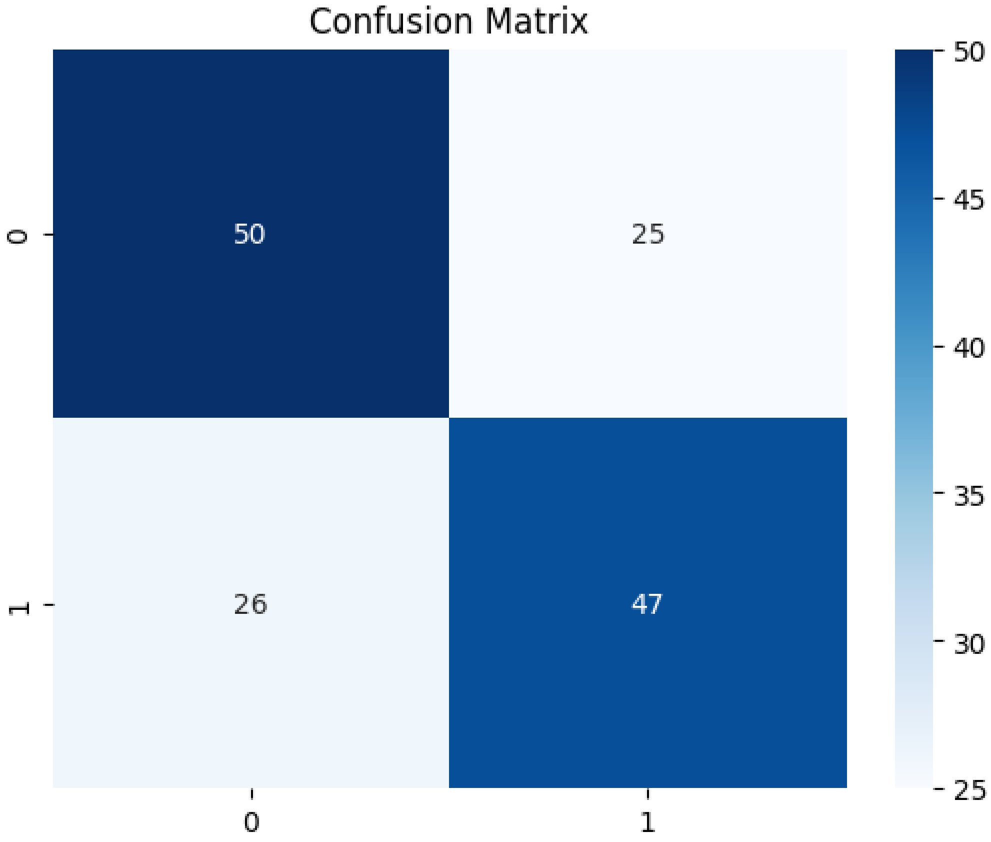
Support Vector Machine

In [ ]:
```
svm = SVC(random_state=42)
svm.fit(x_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC(random_state=42)
```

In [ ]:
```
evaluate(svm, x_test, y_test)
```

```
Accuracy: 0.66
Precision: 0.66
Recall: 0.66
F1-score: 0.66
SVC Cross-Validation Accuracy: 0.67
```



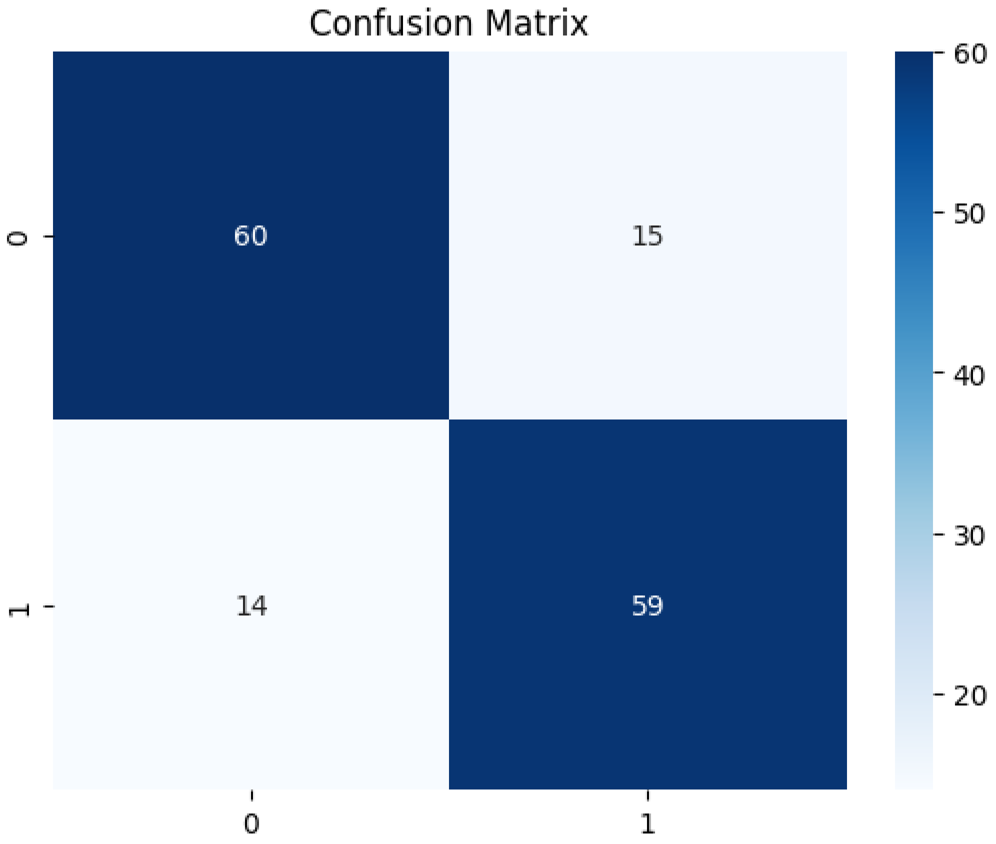
K-Nearest Neighbors

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
```

Out[ ]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [ ]:
```
evaluate(knn, x_test, y_test)
```

```
Accuracy: 0.80
Precision: 0.80
Recall: 0.80
F1-score: 0.80
KNeighborsClassifier Cross-Validation Accuracy: 0.79
```



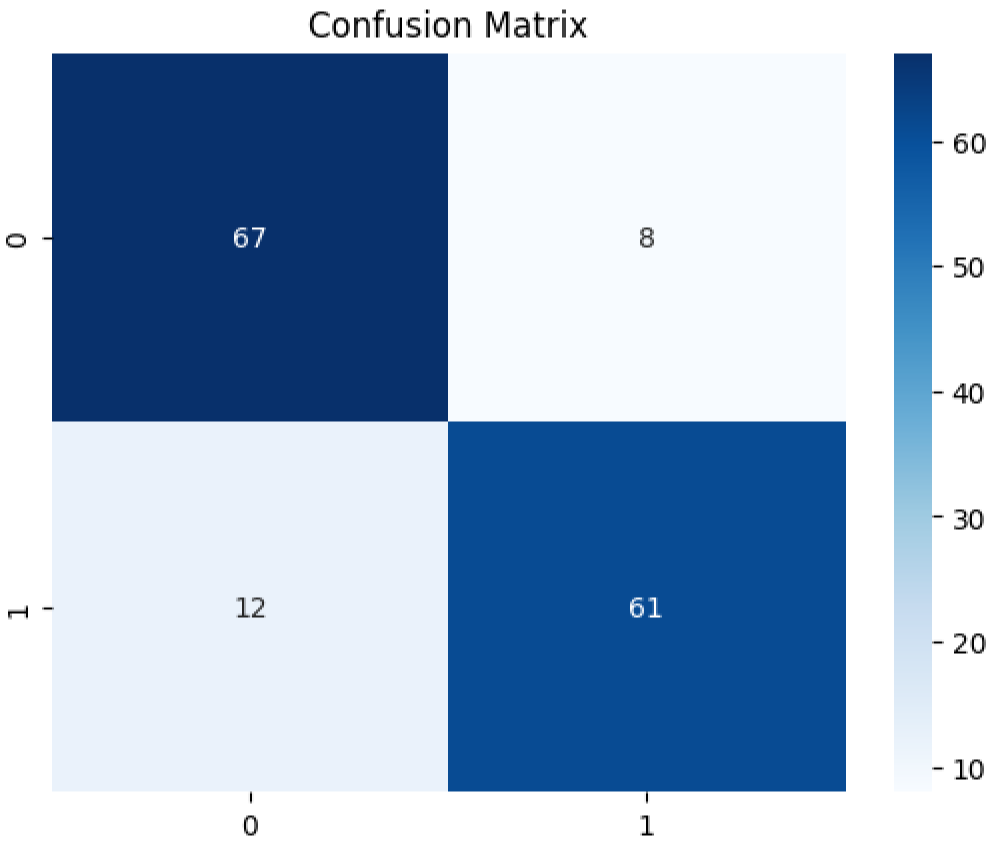
Gradient Boost

In [ ]:
```
gradient_boost = GradientBoostingClassifier(random_state=42)
gradient_boost.fit(x_train, y_train)
```

Out[ ]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier(random_state=42)
```

In [ ]:
```
evaluate(gradient_boost, x_test, y_test)
```

```
Accuracy: 0.86
Precision: 0.87
Recall: 0.86
F1-score: 0.86
GradientBoostingClassifier Cross-Validation Accuracy: 0.85
```



```
In [ ]:
```