Final Project - Classification (Heart Failure Prediction)

March 29, 2025

1 The Objectives

- Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation and the benefits that your analysis provides to the business or stakeholders of this data.
- Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.
- Brief summary of data exploration and actions taken for data cleaning and feature engineering.
- Summary of training at least three different classifier models, preferably of different nature in explainability and predictability. For example, you can start with a simple logistic regression as a baseline, adding other models or ensemble models. Preferably, all your models use the same training and test splits, or the same cross-validation method.
- A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.
- Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classifier model.
- Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model after adding specific data features that may help you achieve a better explanation or a better prediction.

[167]:
```
# pandas, dfs libs
import pandas as pd
import numpy as np

# plotting libs
import matplotlib.pyplot as plt
import seaborn as sns

# preprocessing, split etc libs
from sklearn.preprocessing import RobustScaler, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV

# models libs
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

# metrics libs
from sklearn.metrics import accuracy_score, precision_score, recall_score␣
 ↪,f1_score, confusion_matrix, classification_report
```

2 The Dataset

(Brief description of the data set you chose, a summary of its attributes, and an outline of what you are trying to accomplish with this analysis.)

2.1 Context (brief description of the data)

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Four out of 5CVD deaths are due to heart attacks and strokes, and one-third of these deaths occur prematurely in people under 70 years of age. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease.

People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

2.2 Source

This dataset was created by combining different datasets already available independently but not combined before. In this dataset, 5 heart datasets are combined over 11 common features which makes it the largest heart disease dataset available so far for research purposes. The five datasets used for its curation are:

Cleveland: 303 observations Hungarian: 294 observations Switzerland: 123 observations Long Beach VA: 200 observations Stalog (Heart) Data Set: 270 observations Total: 1190 observations Duplicated: 272 observations

Final dataset: 918 observations

Every dataset used can be found under the Index of heart disease datasets from UCI Machine Learning Repository on the following link: https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

2.2.1 Citation

fedesoriano. (September 2021). Heart Failure Prediction Dataset. Retrieved [Date Retrieved] from https://www.kaggle.com/fedesoriano/heart-failure-prediction.

2.2.2 Acknowledgements

Creators:

Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

2.2.3 Donor:

David W. Aha (aha '@' ics.uci.edu) (714) 856-8779

2.3 Attribute Information (summary of the attributes)

The dataset contains the following features.

- Age: age of the patient [years]
- Sex: sex of the patient [M: Male, F: Female]
- ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- RestingBP: resting blood pressure [mm Hg]
- Cholesterol: serum cholesterol [mm/dl]
- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
- MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
- ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
- Oldpeak: oldpeak = ST [Numeric value measured in depression]
- ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
- HeartDisease: output class [1: heart disease, 0: Normal]

3 The Dataset Exploration and Visualisation

(Brief summary of data exploration and actions taken for data cleaning and feature engineering.)

In this section we will explore the dataset, take steps to clean it, feature engineer, as well as look for patterns in the dataset to analyse if the given data is a good for machine learning model creation.

[120]:
```
df_heart = pd.read_csv('./input/heart_failure_prediction_dataset.csv')

# reading first 5 rows
df_heart.head()
```

[120]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	\
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

```
  ExerciseAngina  Oldpeak ST_Slope  HeartDisease
0              N      0.0       Up             0
1              N      1.0     Flat             1
2              N      0.0       Up             0
3              Y      1.5     Flat             1
4              N      0.0       Up             0
```

[121]:
```
# showing columns and their data types
df_heart.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

[122]:
```
# showing column names
df_heart.columns
```

[122]:
```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

[123]:
```
# key statistical metrics
df_heart.describe()
```

[123]:
```
              Age   RestingBP  Cholesterol   FastingBS       MaxHR  \
count  918.000000  918.000000   918.000000  918.000000  918.000000   
mean    53.510893  132.396514   198.799564    0.233115  136.809368   
std      9.432617   18.514154   109.384145    0.423046   25.460334   
min     28.000000    0.000000     0.000000    0.000000   60.000000   
25%     47.000000  120.000000   173.250000    0.000000  120.000000   
50%     54.000000  130.000000   223.000000    0.000000  138.000000   
```

```
75%     60.000000  140.000000   267.000000    0.000000  156.000000
max     77.000000  200.000000   603.000000    1.000000  202.000000

          Oldpeak  HeartDisease
count  918.000000    918.000000
mean     0.887364      0.553377
std      1.066570      0.497414
min     -2.600000      0.000000
25%      0.000000      0.000000
50%      0.600000      1.000000
75%      1.500000      1.000000
max      6.200000      1.000000
```

[124]:
```
df_heart.describe(include='all').T
```

[124]:
```
                count unique     top freq        mean         std   min  \
Age             918.0    NaN     NaN  NaN   53.510893    9.432617  28.0
Sex               918      2       M  725         NaN         NaN   NaN
ChestPainType     918      4     ASY  496         NaN         NaN   NaN
RestingBP       918.0    NaN     NaN  NaN  132.396514   18.514154   0.0
Cholesterol     918.0    NaN     NaN  NaN  198.799564  109.384145   0.0
FastingBS       918.0    NaN     NaN  NaN    0.233115    0.423046   0.0
RestingECG        918      3  Normal  552         NaN         NaN   NaN
MaxHR           918.0    NaN     NaN  NaN  136.809368   25.460334  60.0
ExerciseAngina    918      2       N  547         NaN         NaN   NaN
Oldpeak         918.0    NaN     NaN  NaN    0.887364     1.06657  -2.6
ST_Slope          918      3    Flat  460         NaN         NaN   NaN
HeartDisease    918.0    NaN     NaN  NaN    0.553377    0.497414   0.0

                  25%    50%    75%    max
Age              47.0   54.0   60.0   77.0
Sex               NaN    NaN    NaN    NaN
ChestPainType     NaN    NaN    NaN    NaN
RestingBP       120.0  130.0  140.0  200.0
Cholesterol     173.25  223.0  267.0  603.0
FastingBS          0.0    0.0    0.0    1.0
RestingECG         NaN    NaN    NaN    NaN
MaxHR            120.0  138.0  156.0  202.0
ExerciseAngina     NaN    NaN    NaN    NaN
Oldpeak            0.0    0.6    1.5    6.2
ST_Slope           NaN    NaN    NaN    NaN
HeartDisease       0.0    1.0    1.0    1.0
```

[125]:
```
df_heart.describe(include='all')
```

[125]:
```
                Age  Sex ChestPainType   RestingBP  Cholesterol   FastingBS  \
count   918.000000  918           918  918.000000   918.000000  918.000000
```

```
unique         NaN    2             4         NaN          NaN         NaN
top            NaN    M           ASY         NaN          NaN         NaN
freq           NaN  725           496         NaN          NaN         NaN
mean     53.510893  NaN           NaN  132.396514   198.799564    0.233115
std       9.432617  NaN           NaN   18.514154   109.384145    0.423046
min      28.000000  NaN           NaN    0.000000     0.000000    0.000000
25%      47.000000  NaN           NaN  120.000000   173.250000    0.000000
50%      54.000000  NaN           NaN  130.000000   223.000000    0.000000
75%      60.000000  NaN           NaN  140.000000   267.000000    0.000000
max      77.000000  NaN           NaN  200.000000   603.000000    1.000000

       RestingECG       MaxHR ExerciseAngina     Oldpeak ST_Slope  \
count         918  918.000000            918  918.000000      918
unique          3         NaN              2         NaN        3
top        Normal         NaN              N         NaN     Flat
freq          552         NaN            547         NaN      460
mean          NaN  136.809368            NaN    0.887364      NaN
std           NaN   25.460334            NaN    1.066570      NaN
min           NaN   60.000000            NaN   -2.600000      NaN
25%           NaN  120.000000            NaN    0.000000      NaN
50%           NaN  138.000000            NaN    0.600000      NaN
75%           NaN  156.000000            NaN    1.500000      NaN
max           NaN  202.000000            NaN    6.200000      NaN

        HeartDisease
count     918.000000
unique           NaN
top              NaN
freq             NaN
mean        0.553377
std         0.497414
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
```

[126]:
```
# checking for any missing/null columns
print("missing values ?", "Yes" if df_heart.isnull().sum().any() else "No")
```

```
missing values ? No
```

[127]:
```
# checking for any duplicates
print("duplicate values ?", "Yes" if df_heart.duplicated().sum() > 0 else "No")
```

```
duplicate values ? No
```

[128]:
```
# checking gender counts
df_heart['Sex'].value_counts()
```

[128]:
```
Sex
M    725
F    193
Name: count, dtype: int64
```

[129]:
```
df_heart['ChestPainType'].value_counts()
```

[129]:
```
ChestPainType
ASY    496
NAP    203
ATA    173
TA      46
Name: count, dtype: int64
```

[130]:
```
df_heart['RestingBP'].value_counts()
```

[130]:
```
RestingBP
120    132
130    118
140    107
110     58
150     55
      ...
185      1
98       1
92       1
113      1
164      1
Name: count, Length: 67, dtype: int64
```

[131]:
```
df_heart['Cholesterol'].value_counts()
```

[131]:
```
Cholesterol
0      172
254     11
223     10
220     10
230      9
      ...
392      1
316      1
153      1
466      1
131      1
Name: count, Length: 222, dtype: int64
```

[132]:
```
df_heart[df_heart['Cholesterol'] == 0]
```

[132]:
```
     Age Sex ChestPainType  RestingBP  Cholesterol  FastingBS RestingECG  \
293   65   M           ASY        115            0          0     Normal
294   32   M            TA         95            0          1     Normal
295   61   M           ASY        105            0          1     Normal
296   50   M           ASY        145            0          1     Normal
297   57   M           ASY        110            0          1         ST
..   ...  ..           ...        ...          ...        ...        ...
514   43   M           ASY        122            0          0     Normal
515   63   M           NAP        130            0          1         ST
518   48   M           NAP        102            0          1         ST
535   56   M           ASY        130            0          0        LVH
536   62   M           NAP        133            0          1         ST

     MaxHR ExerciseAngina  Oldpeak ST_Slope  HeartDisease
293     93              Y      0.0     Flat             1
294    127              N      0.7       Up             1
295    110              Y      1.5       Up             1
296    139              Y      0.7     Flat             1
297    131              Y      1.4       Up             1
..     ...            ...      ...      ...           ...
514    120              N      0.5       Up             1
515    160              N      3.0     Flat             0
518    110              Y      1.0     Down             1
535    122              Y      1.0     Flat             1
536    119              Y      1.2     Flat             1

[172 rows x 12 columns]
```

[133]:
```
df_heart['FastingBS'].value_counts()
```

[133]:
```
FastingBS
0    704
1    214
Name: count, dtype: int64
```

[134]:
```
df_heart['RestingECG'].value_counts()
```

[134]:
```
RestingECG
Normal    552
LVH       188
ST        178
Name: count, dtype: int64
```

[135]:
```
# checking gender counts
df_heart['HeartDisease'].value_counts()
```

[135]:
```
HeartDisease
1    508
0    410
Name: count, dtype: int64
```

[136]:
```
df_heart['MaxHR'].value_counts()
```

[136]:
```
MaxHR
150    43
140    41
120    36
130    33
160    25
       ..
63      1
83      1
60      1
78      1
202     1
Name: count, Length: 119, dtype: int64
```

[137]:
```
df_heart['ExerciseAngina'].value_counts()
```

[137]:
```
ExerciseAngina
N    547
Y    371
Name: count, dtype: int64
```

[138]:
```
df_heart['Oldpeak'].value_counts()
```

[138]:
```
Oldpeak
 0.0    368
 1.0     86
 2.0     76
 1.5     53
 3.0     28
 1.2     26
 0.2     22
 0.5     19
 1.4     18
 1.8     17
 2.5     16
 0.8     16
 1.6     16
 0.1     14
 0.6     14
 0.4     11
 0.3     11
```

```
 4.0      8
 0.7      7
 2.8      7
 1.9      7
 1.3      7
 2.6      7
 1.1      7
 1.7      6
 2.2      5
 0.9      4
 2.4      4
 3.6      4
 3.4      3
 4.2      2
 3.5      2
-0.5      2
 2.3      2
 3.2      2
 2.1      2
-1.0      2
-0.1      2
 5.6      1
 2.9      1
 6.2      1
 3.8      1
-1.5      1
 3.1      1
-2.0      1
 3.7      1
-0.8      1
-0.7      1
-1.1      1
-2.6      1
-0.9      1
 5.0      1
 4.4      1
Name: count, dtype: int64
```

[139]:
```
label_encoder = LabelEncoder()

features = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']

for i in features:
    if df_heart[i].dtype == 'object':
        df_heart[i] = label_encoder.fit_transform(df_heart[i])
```

[140]:
```
df_heart.head()
```

[140]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	1	1	140	289	0	1	172	0	0.0	2	0
1	49	0	2	160	180	0	1	156	0	1.0	1	1
2	37	1	1	130	283	0	2	98	0	0.0	2	0
3	48	0	0	138	214	0	1	108	1	1.5	1	1
4	54	1	2	150	195	0	1	122	0	0.0	2	0

[141]:
```
# Unique value counts
pd.DataFrame([[i, len(df_heart[i].unique())] for i in df_heart.columns],␣
 ↪columns=['Feature', 'Unique Counts'])
```

[141]:

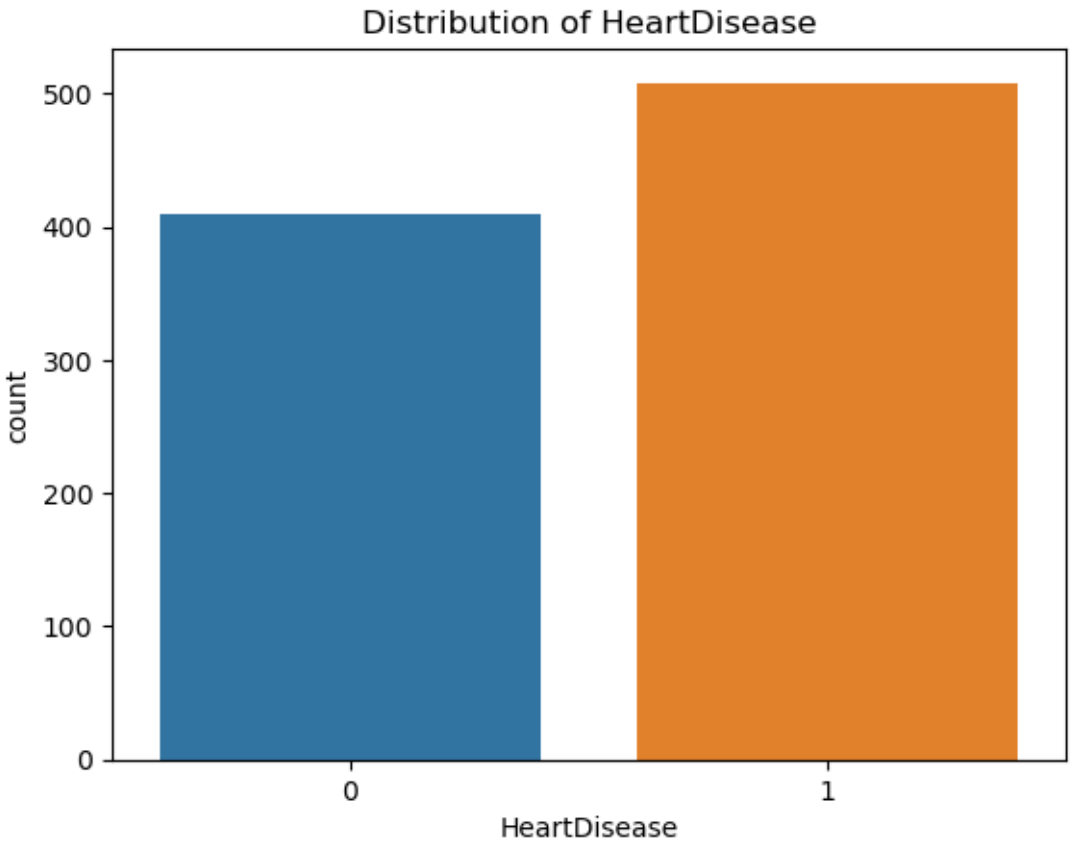
	Feature	Unique Counts
0	Age	50
1	Sex	2
2	ChestPainType	4
3	RestingBP	67
4	Cholesterol	222
5	FastingBS	2
6	RestingECG	3
7	MaxHR	119
8	ExerciseAngina	2
9	Oldpeak	53
10	ST_Slope	3
11	HeartDisease	2

[142]:
```
# checking age group counts
df_heart['Age'].value_counts()
```

[142]:
```
Age
54    51
58    42
55    41
56    38
57    38
52    36
51    35
59    35
62    35
53    33
60    32
```

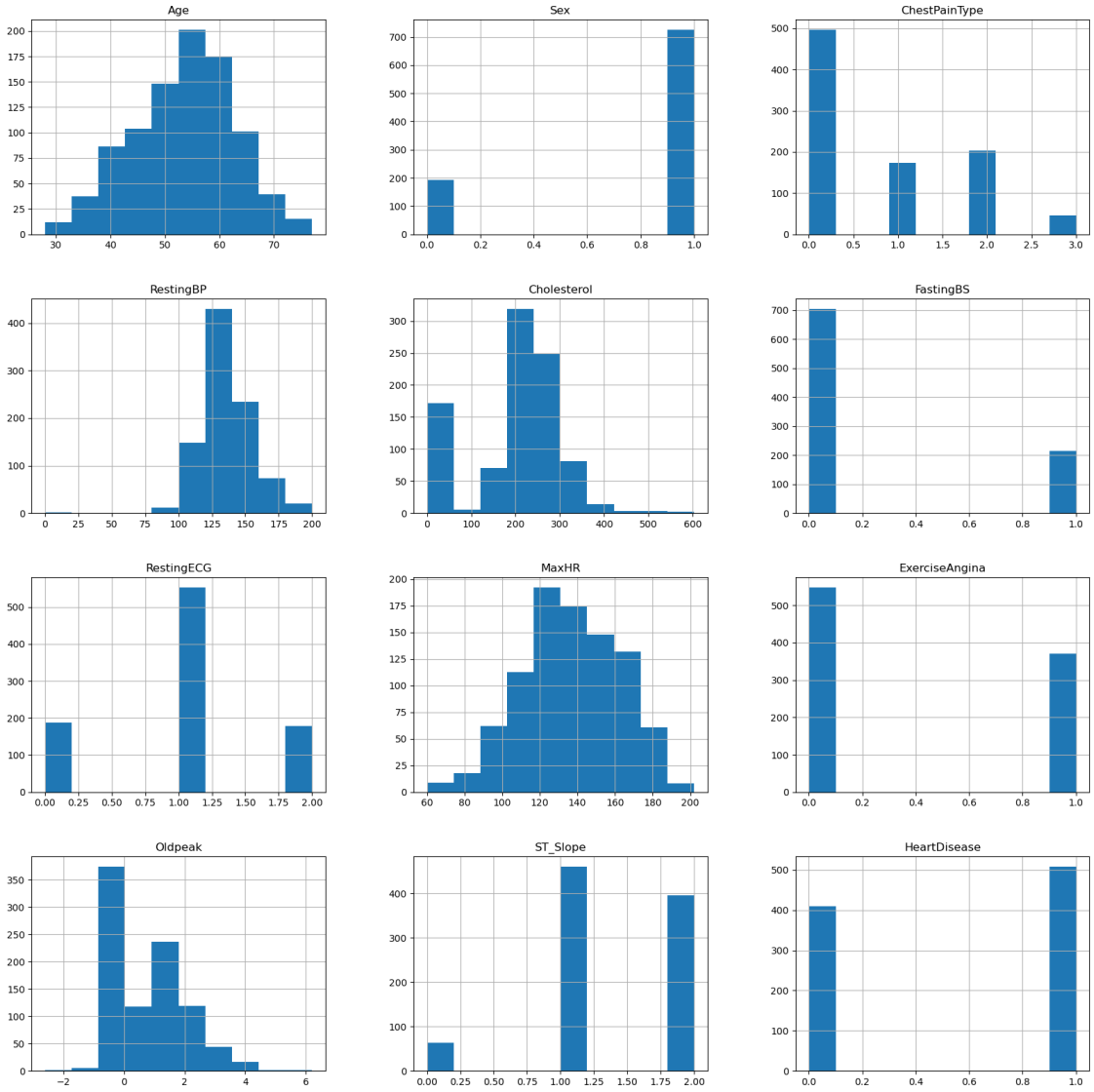
```
48    31
61    31
63    30
50    25
46    24
41    24
43    24
64    22
65    21
49    21
47    19
44    19
42    18
45    18
38    16
67    15
39    15
66    13
69    13
40    13
35    11
37    11
68    10
34     7
74     7
70     7
36     6
71     5
32     5
72     4
29     3
75     3
33     2
77     2
76     2
31     2
30     1
28     1
73     1
Name: count, dtype: int64
```

[143]:
```
sns.countplot(x='HeartDisease', data=df_heart)
plt.title('Distribution of HeartDisease')
plt.show()
```



[144]:
```
# data visualisation using histograms
df_heart.hist(figsize=(20,20))
```

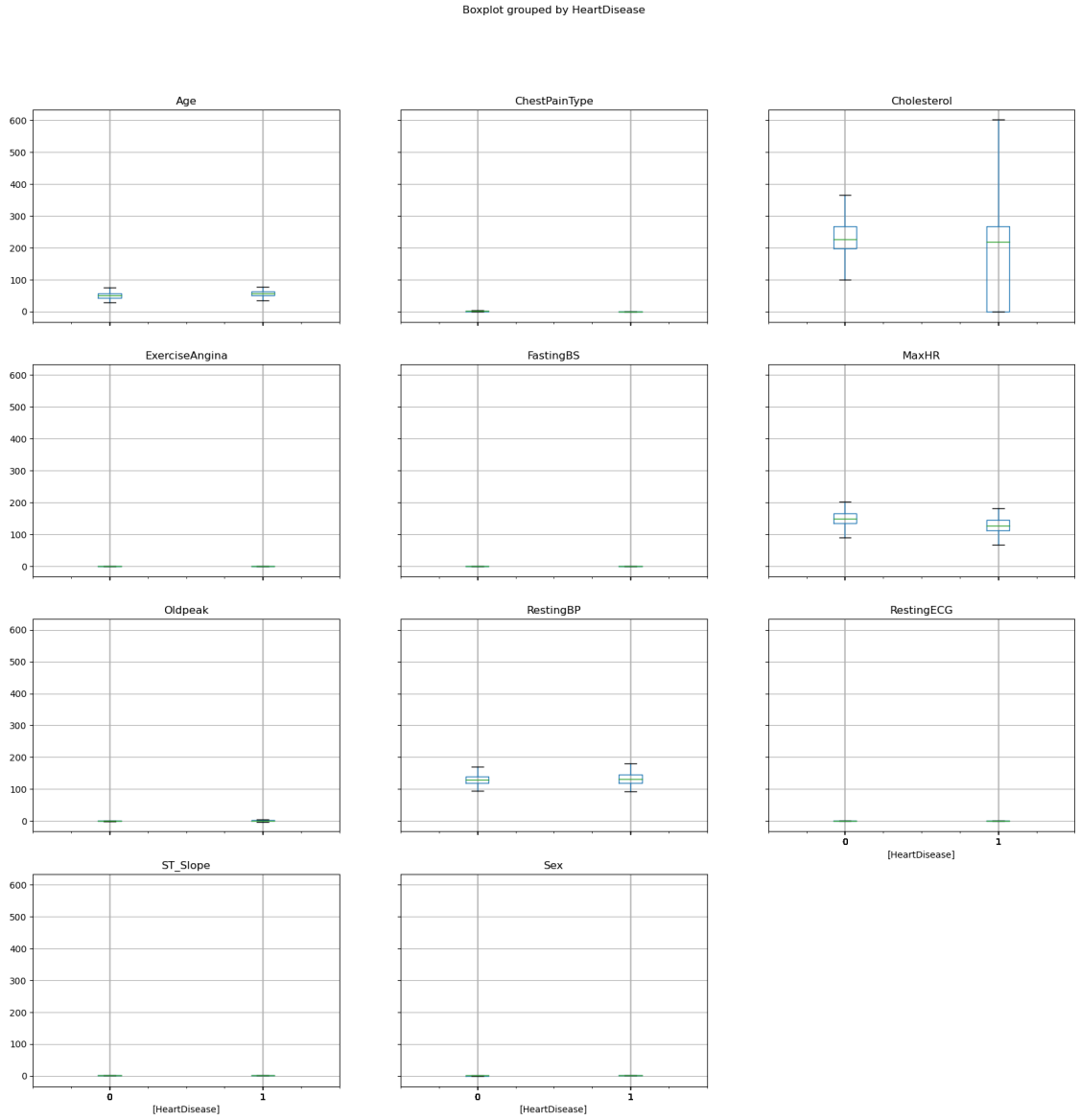
[144]:
```
array([[<Axes: title={'center': 'Age'}>, <Axes: title={'center': 'Sex'}>,
        <Axes: title={'center': 'ChestPainType'}>],
       [<Axes: title={'center': 'RestingBP'}>,
        <Axes: title={'center': 'Cholesterol'}>,
        <Axes: title={'center': 'FastingBS'}>],
       [<Axes: title={'center': 'RestingECG'}>,
        <Axes: title={'center': 'MaxHR'}>,
        <Axes: title={'center': 'ExerciseAngina'}>],
       [<Axes: title={'center': 'Oldpeak'}>,
        <Axes: title={'center': 'ST_Slope'}>,
        <Axes: title={'center': 'HeartDisease'}>]], dtype=object)
```



[145]:
```
# checking outliers - HeartDisease is the target variable
df_heart.boxplot(by="HeartDisease", showfliers=False, figsize=(20,20))
```

[145]:
```
array([[<Axes: title={'center': 'Age'}, xlabel='[HeartDisease]'>,
        <Axes: title={'center': 'ChestPainType'}, xlabel='[HeartDisease]'>,
        <Axes: title={'center': 'Cholesterol'}, xlabel='[HeartDisease]'>],
       [<Axes: title={'center': 'ExerciseAngina'}, xlabel='[HeartDisease]'>,
        <Axes: title={'center': 'FastingBS'}, xlabel='[HeartDisease]'>,
        <Axes: title={'center': 'MaxHR'}, xlabel='[HeartDisease]'>],
       [<Axes: title={'center': 'Oldpeak'}, xlabel='[HeartDisease]'>,
        <Axes: title={'center': 'RestingBP'}, xlabel='[HeartDisease]'>,
        <Axes: title={'center': 'RestingECG'}, xlabel='[HeartDisease]'>],
       [<Axes: title={'center': 'ST_Slope'}, xlabel='[HeartDisease]'>,
```

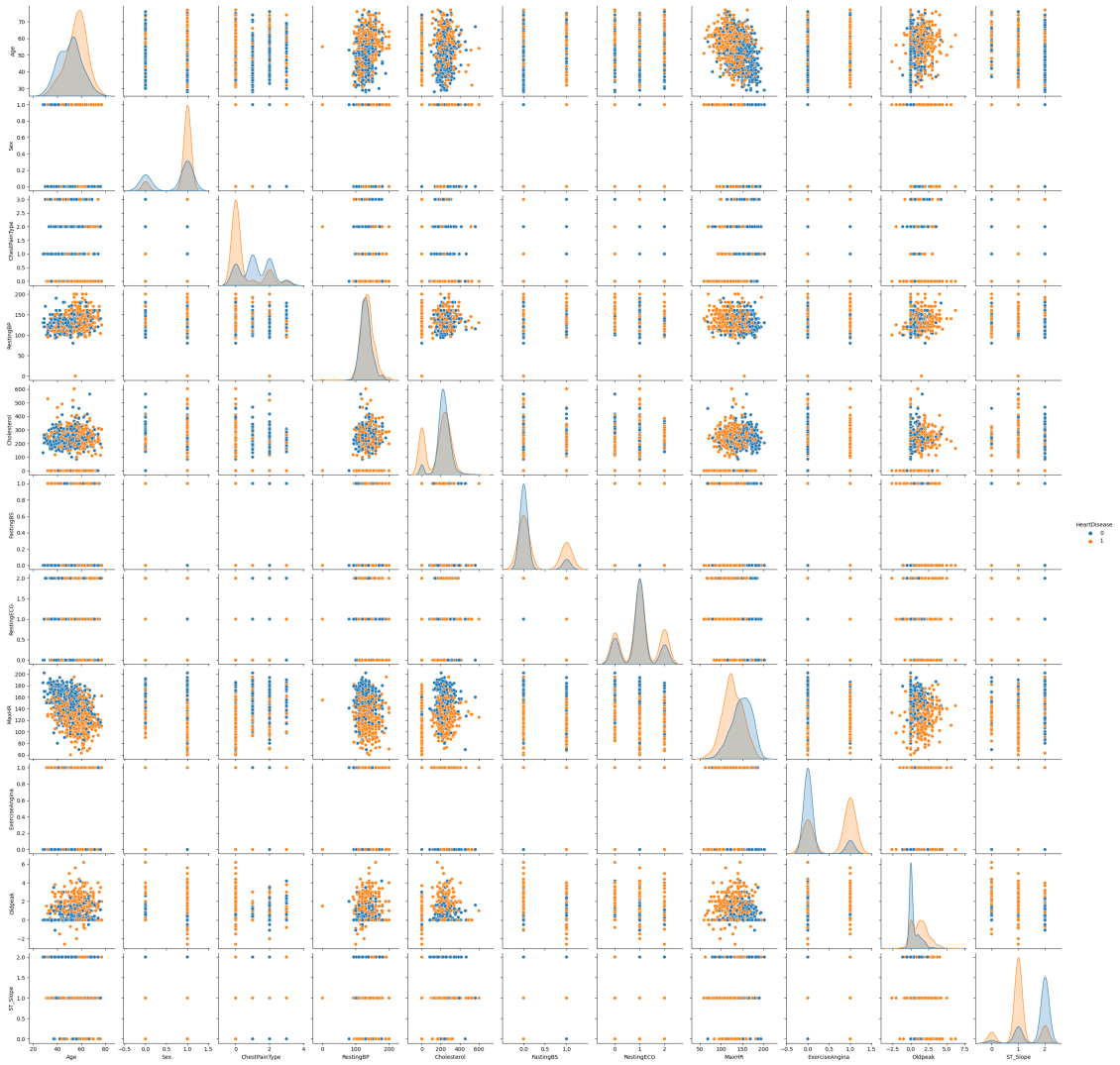
```
<Axes: title={'center': 'Sex'}, xlabel='[HeartDisease]'>,
<Axes: >]], dtype=object)
```



[146]:
```
# identifying corrleation between variables - target HeartDisease, has vs not
sns.pairplot(df_heart, hue='HeartDisease')
```

```
C:\Users\micha\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

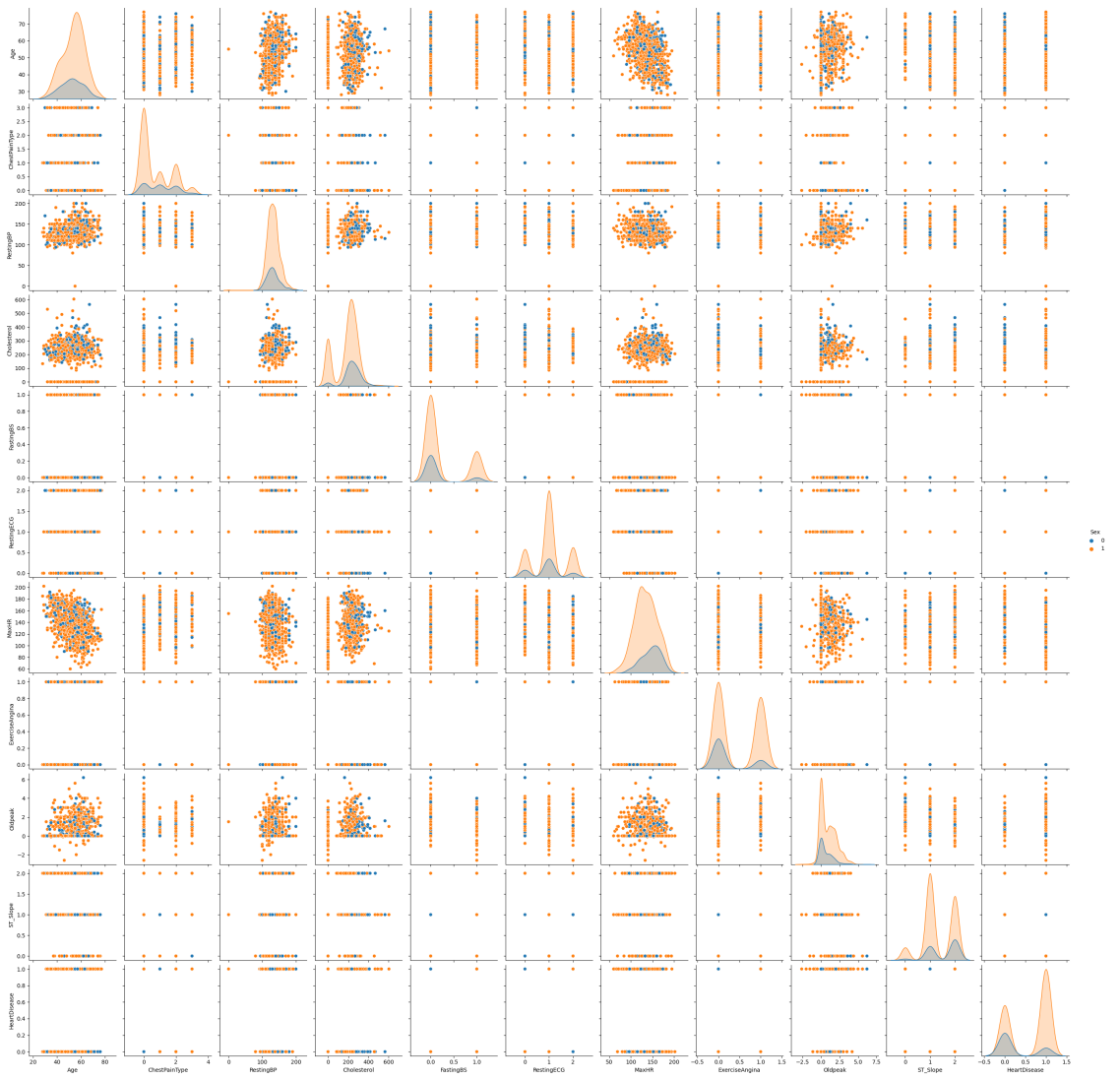
[146]:
```
<seaborn.axisgrid.PairGrid at 0x2860a745710>
```



[147]:
```
# identifying correleation between variables - target Sex, M vs F
sns.pairplot(df_heart, hue='Sex')
```

```
C:\Users\micha\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

[147]: <seaborn.axisgrid.PairGrid at 0x28615a85690>



4 Training and Testing Models / Classifiers

Summary of training at least three different classifier models, preferably of different nature in explainability and predictability. For example, you can start with a simple logistic regression as a baseline, adding other models or ensemble models. Preferably, all your models use the same training and test splits, or the same cross-validation method.

[148]:
```
model_accuracy_comparison = []
```

4.1 Dataset splitting

[149]:
```
X, y = df_heart.drop('HeartDisease', axis=1), df_heart['HeartDisease']

# splitting the dataset into training and test samples, 70% / 30% split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

4.2 Dataset validation post splitting

[150]:
```
print('Positive samples in training set: {}'.format(y_train.
 ↪value_counts(normalize=True)[1]))
print('Negative samples in training set: {}'.format(y_train.
 ↪value_counts(normalize=True)[0]))
print('Positive samples in test set: {}'.format(y_test.
 ↪value_counts(normalize=True)[1]))
print('Negative samples in test set: {}'.format(y_test.
 ↪value_counts(normalize=True)[0]))
```

```
Positive samples in training set: 0.5358255451713395
Negative samples in training set: 0.46417445482866043
Positive samples in test set: 0.5942028985507246
Negative samples in test set: 0.4057971014492754
```

4.3 Preprocessing using robust scaler

[151]:
```
robust_scaler = RobustScaler()

scale_columns = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR']

X_train[scale_columns] = robust_scaler.fit_transform(X_train[scale_columns])
X_test[scale_columns] = robust_scaler.transform(X_test[scale_columns])
```

4.4 Show Model Confusion Matrix

[152]:
```
def show_confusion_matrix(model_name, y_test, y_pred):
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',␣
 ↪cmap='Blues')
    plt.title(f"Confusion Matrix ({model_name})")
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
```

4.5 Model Evaluation

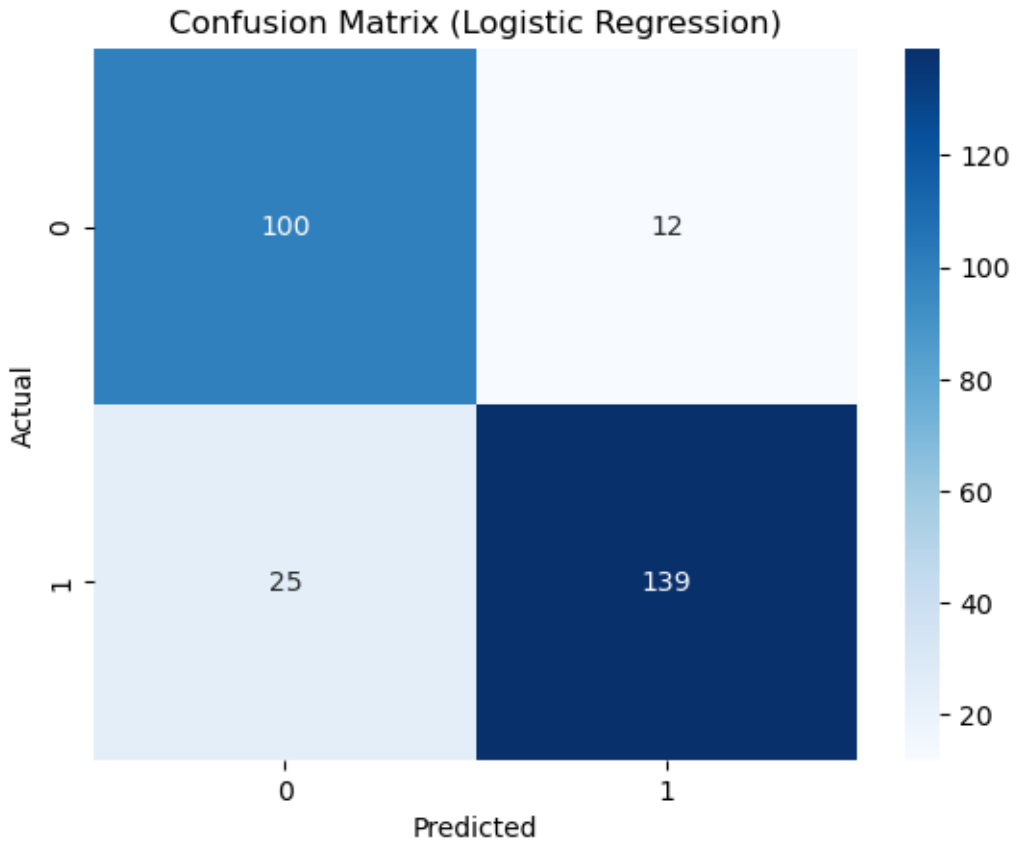
[153]:
```
def model_evaluation(model_lr, model_name):
    model_lr.fit(X_train, y_train)

    y_pred = model_lr.predict(X_test)
```

```
    y_accuracy = accuracy_score(y_test, y_pred)
    y_precision = precision_score(y_test, y_pred)
    y_recall = recall_score(y_test, y_pred)

    print(f"Model Name: {model_name}")
    print(f"Model Accuracy: {y_accuracy:.4f}")
    print(f"Model Precision: {y_precision:.4f}")
    print(f"Model Recall: {y_recall:.4f}")
    print("Model Classification Report:\n", classification_report(y_test,␣
 ↪y_pred))
    print("Model Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

    show_confusion_matrix(model_name, y_test, y_pred)

    model_accuracy_comparison.append([model_name, y_accuracy])

    #print("Model's Accuracy:", y_accuracy)

    return y_accuracy
```

4.6 Logistic Regression

[154]:
```
model_lr = LogisticRegression(max_iter=1000)

lr_y_accuracy = model_evaluation(model_lr, "Logistic Regression")
```

```
Model Name: Logistic Regression
Model Accuracy: 0.8659
Model Precision: 0.9205
Model Recall: 0.8476
Model Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       112
           1       0.92      0.85      0.88       164

    accuracy                           0.87       276
   macro avg       0.86      0.87      0.86       276
weighted avg       0.87      0.87      0.87       276

Model Confusion Matrix:
 [[100  12]
 [ 25 139]]
```

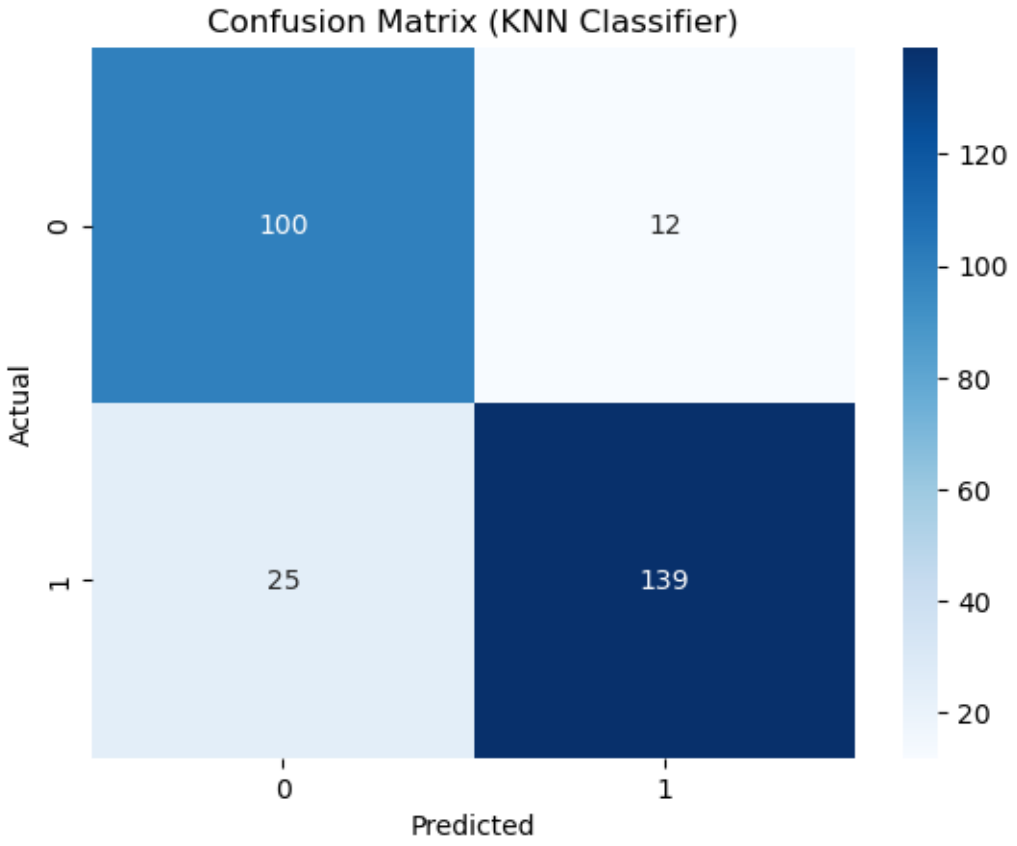


4.7 KNN Classifier

[155]:
```
model_knn = KNeighborsClassifier(n_neighbors=5, weights='distance')

knn_y_accuracy = model_evaluation(model_lr, "KNN Classifier")
```

```
Model Name: KNN Classifier
Model Accuracy: 0.8659
Model Precision: 0.9205
Model Recall: 0.8476
Model Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       112
           1       0.92      0.85      0.88       164

    accuracy                           0.87       276
   macro avg       0.86      0.87      0.86       276
weighted avg       0.87      0.87      0.87       276
```

```
Model Confusion Matrix:
 [[100  12]
 [ 25 139]]
```

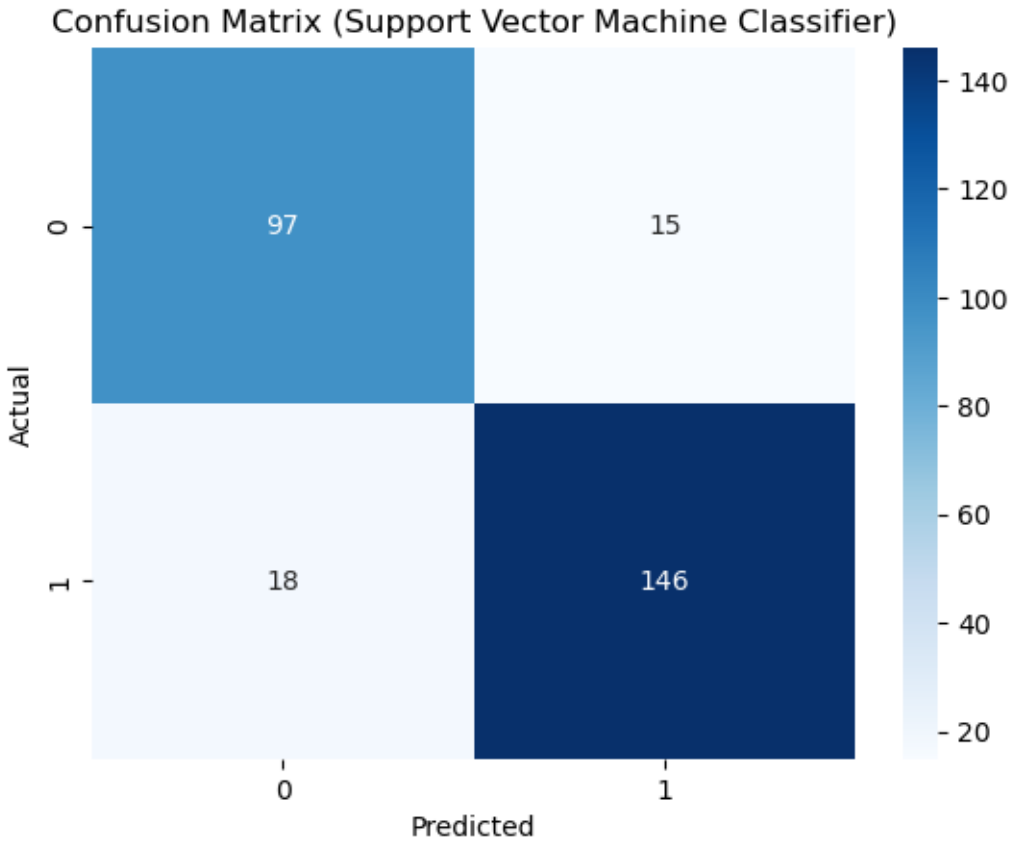


4.8 Support Vector Machine Classifier

[156]:
```
model_svc = SVC(random_state=42)

svc_y_accuracy = model_evaluation(model_svc, "Support Vector Machine␣
 ↪Classifier")
```

```
Model Name: Support Vector Machine Classifier
Model Accuracy: 0.8804
Model Precision: 0.9068
Model Recall: 0.8902
Model Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.87      0.85       112
           1       0.91      0.89      0.90       164
```

```
    accuracy                           0.88       276
   macro avg       0.88      0.88      0.88       276
weighted avg       0.88      0.88      0.88       276

Model Confusion Matrix:
 [[ 97  15]
 [ 18 146]]
```

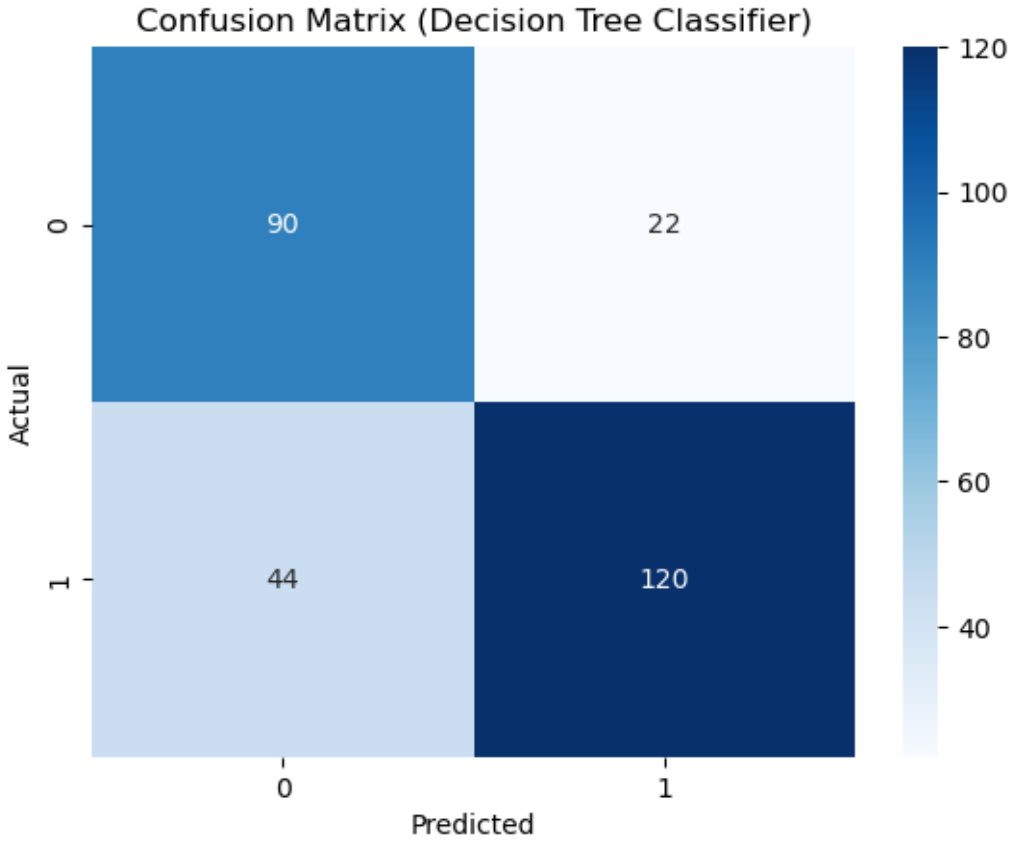


4.9 Decision Tree Classifier

[157]:
```
model_dtc = DecisionTreeClassifier(random_state=42)

dtc_y_accuracy = model_evaluation(model_dtc, "Decision Tree Classifier")
```

```
Model Name: Decision Tree Classifier
Model Accuracy: 0.7609
Model Precision: 0.8451
Model Recall: 0.7317
Model Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.67      0.80      0.73       112
              1       0.85      0.73      0.78       164

    accuracy                            0.76       276
   macro avg       0.76      0.77      0.76       276
weighted avg       0.77      0.76      0.76       276

Model Confusion Matrix:
 [[ 90  22]
 [ 44 120]]
```

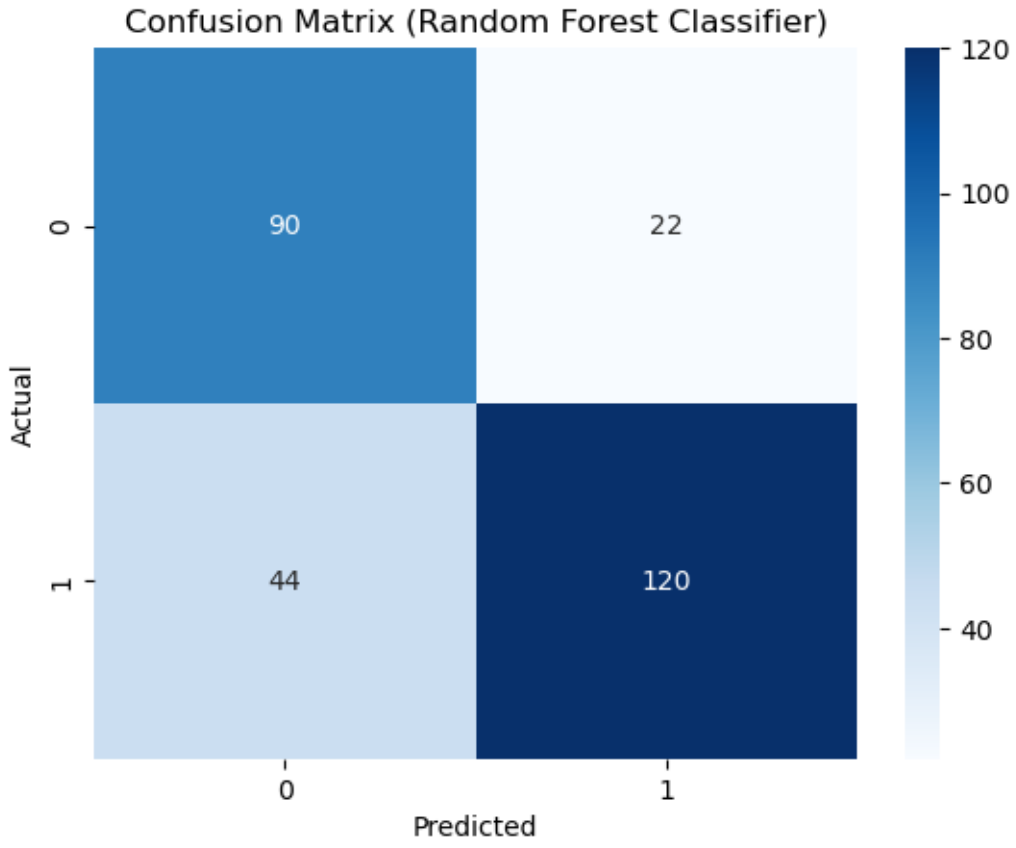


4.10 Random Forest Classifier

[158]:
```
model_rfc = RandomForestClassifier(random_state=42)

rfc_y_accuracy = model_evaluation(model_dtc, "Random Forest Classifier")
```

```
Model Name: Random Forest Classifier
Model Accuracy: 0.7609
Model Precision: 0.8451
```

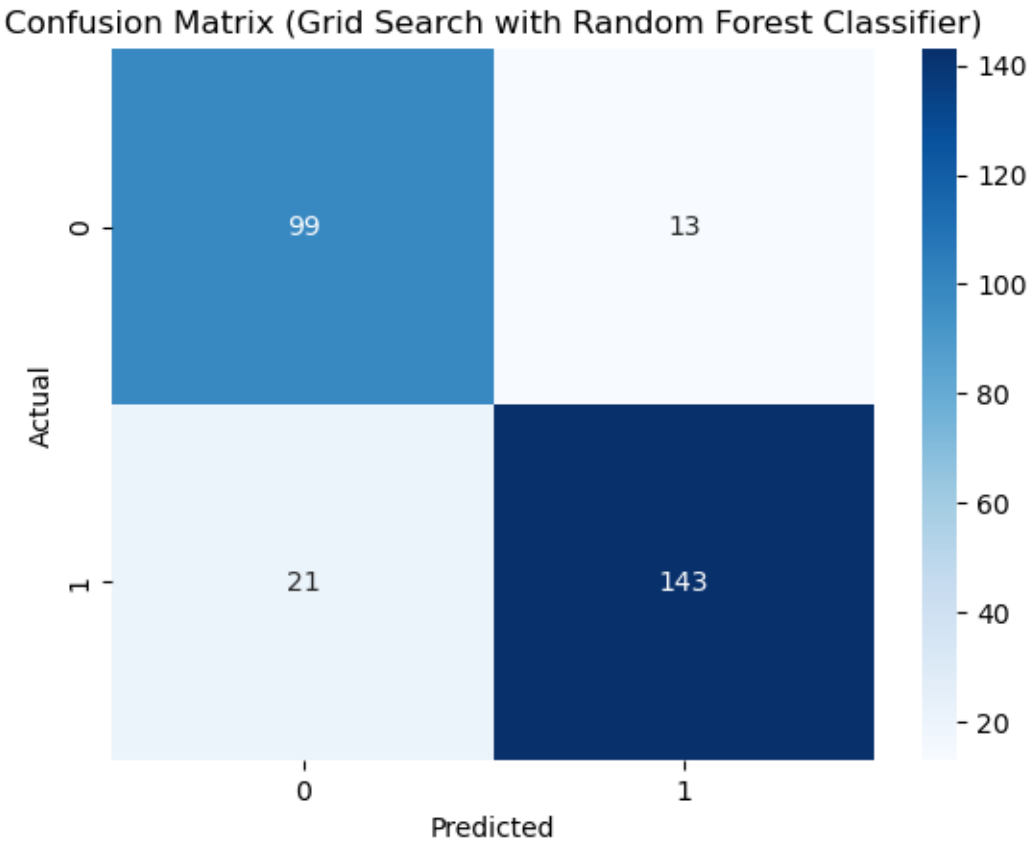
```
Model Recall: 0.7317
Model Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.80      0.73       112
           1       0.85      0.73      0.78       164

    accuracy                           0.76       276
   macro avg       0.76      0.77      0.76       276
weighted avg       0.77      0.76      0.76       276

Model Confusion Matrix:
 [[ 90  22]
 [ 44 120]]
```



4.11 Grid Search with Random Forest Classifier

[159]:
```
grid_params = {'n_estimators': [400, 800, 900, 1000]}
class_weight_label = { 1: 0.7, 0: 0.3 }

model_gscv_rfc = GridSearchCV(RandomForestClassifier(random_state=42,␣
 ↪class_weight = class_weight_label, warm_start=True),
                              param_grid=grid_params, scoring='f1', n_jobs=-1)

gscv_rfc_y_accuracy = model_evaluation(model_gscv_rfc, "Grid Search with Random␣
 ↪Forest Classifier")
```

```
Model Name: Grid Search with Random Forest Classifier
Model Accuracy: 0.8768
Model Precision: 0.9167
Model Recall: 0.8720
Model Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       112
           1       0.92      0.87      0.89       164

    accuracy                           0.88       276
   macro avg       0.87      0.88      0.87       276
weighted avg       0.88      0.88      0.88       276

Model Confusion Matrix:
 [[ 99  13]
 [ 21 143]]
```

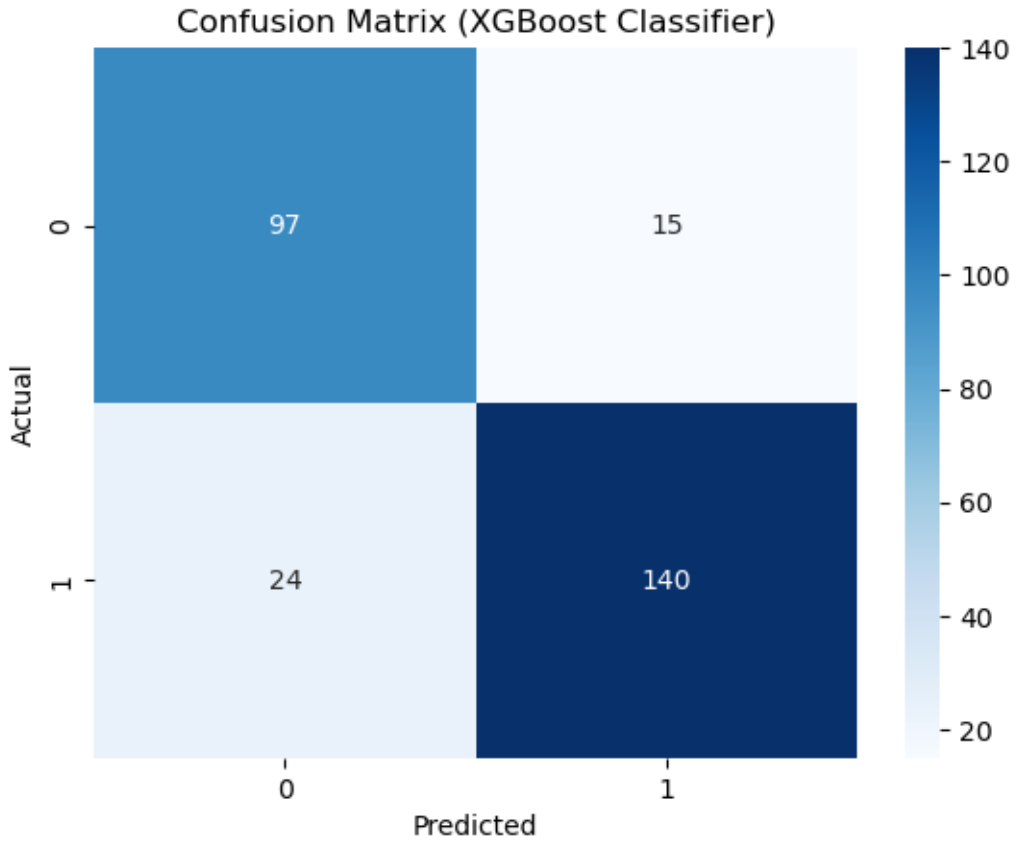


4.12 XGBoost Classifier

[160]:
```
model_xgbc = XGBClassifier(random_state=42)

xgbc_y_accuracy = model_evaluation(model_xgbc, "XGBoost Classifier")
```

```
Model Name: XGBoost Classifier
Model Accuracy: 0.8587
Model Precision: 0.9032
Model Recall: 0.8537
Model Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.87      0.83       112
           1       0.90      0.85      0.88       164

    accuracy                           0.86       276
   macro avg       0.85      0.86      0.86       276
weighted avg       0.86      0.86      0.86       276
```

```
Model Confusion Matrix:
 [[ 97  15]
 [ 24 140]]
```

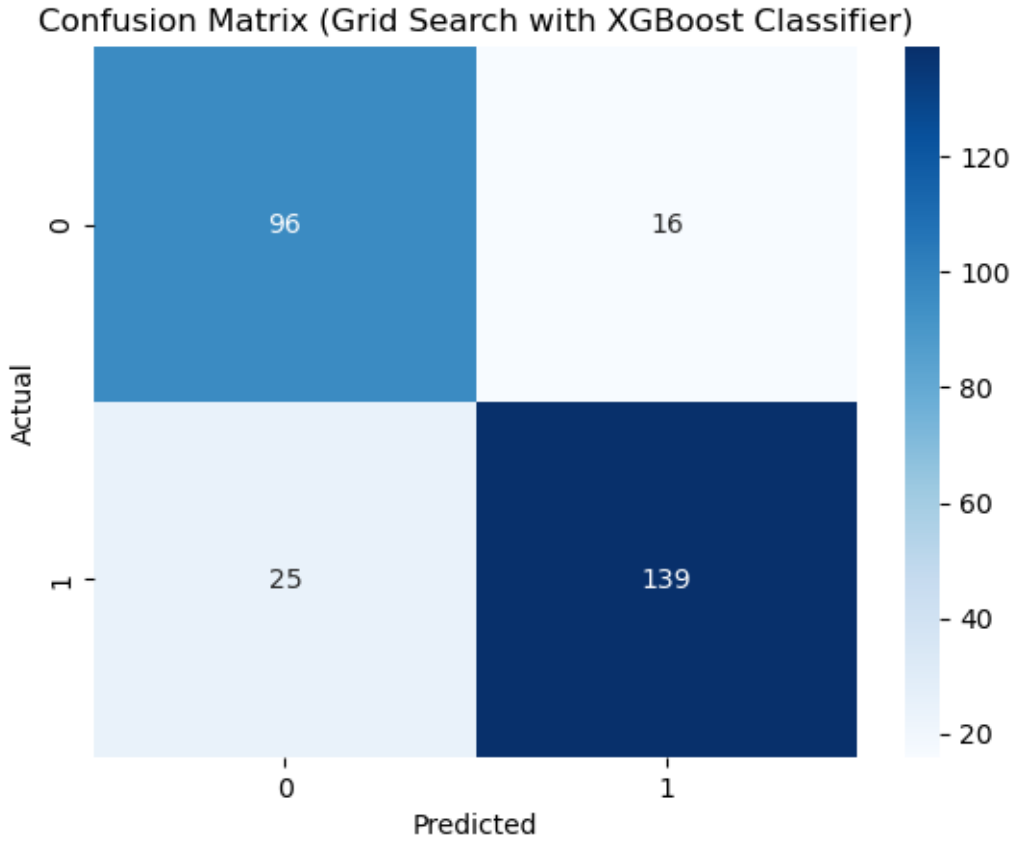


4.13 Grid Search with XGBoost Classifier

[161]:
```
param_grid = {'n_estimators': [400, 800, 900, 1000], 'learning_rate': [0.1, 0.
↪01]}

model_gscv_xgbc = GridSearchCV(XGBClassifier(random_state=42),␣
↪param_grid=param_grid,scoring='f1', n_jobs=-1)

gscv_xgbc_y_accuracy = model_evaluation(model_gscv_xgbc, "Grid Search with␣
↪XGBoost Classifier")
```

```
Model Name: Grid Search with XGBoost Classifier
Model Accuracy: 0.8514
Model Precision: 0.8968
Model Recall: 0.8476
Model Classification Report:
              precision    recall  f1-score   support
```

```
              0       0.79      0.86      0.82       112
              1       0.90      0.85      0.87       164

    accuracy                            0.85       276
   macro avg       0.85      0.85      0.85       276
weighted avg       0.85      0.85      0.85       276

Model Confusion Matrix:
 [[ 96  16]
 [ 25 139]]
```

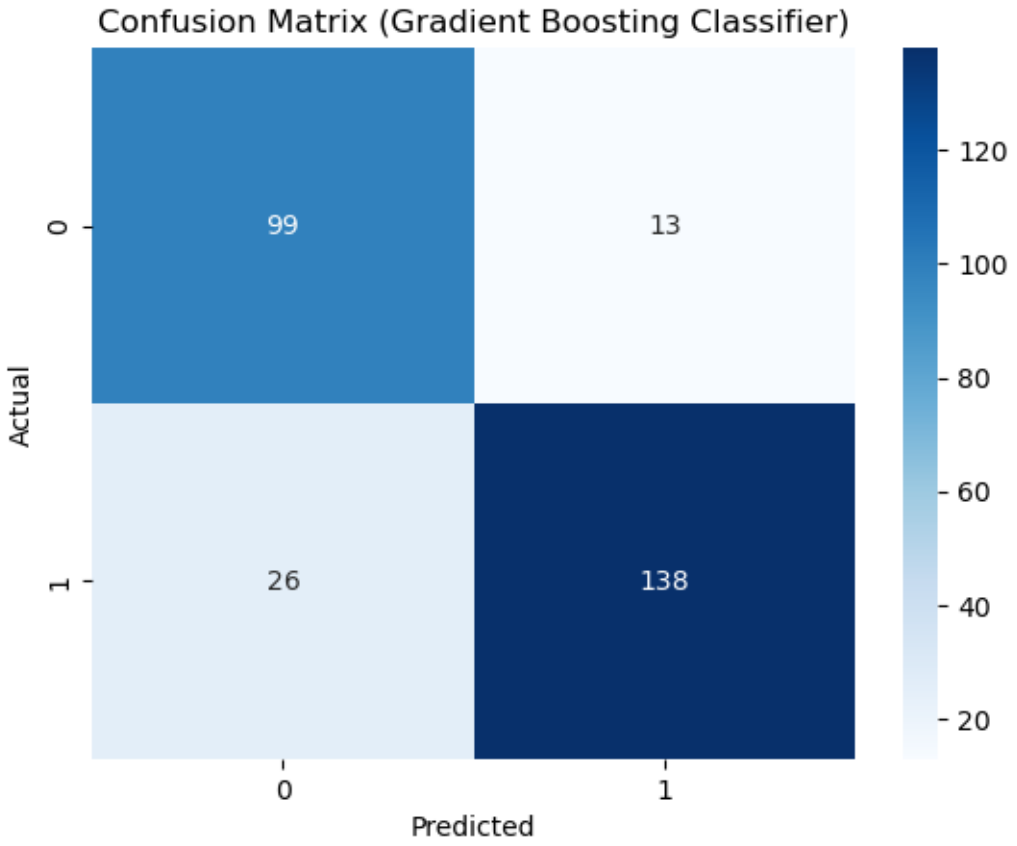


4.14 Gradient Boosting Classifier

[162]:
```
model_gbc = GradientBoostingClassifier(random_state=42)

gbc_y_accuracy = model_evaluation(model_gbc, "Gradient Boosting Classifier")
```

```
Model Name: Gradient Boosting Classifier
Model Accuracy: 0.8587
```

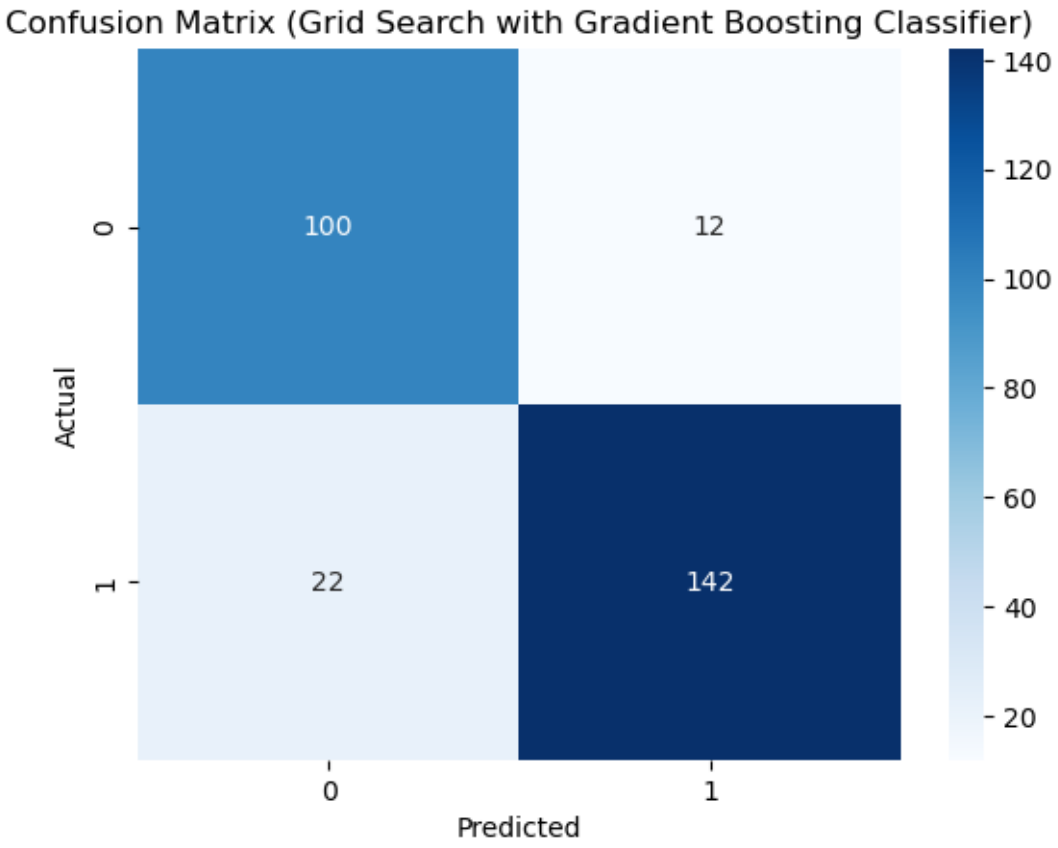
```
Model Precision: 0.9139
Model Recall: 0.8415
Model Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.88      0.84       112
           1       0.91      0.84      0.88       164

    accuracy                           0.86       276
   macro avg       0.85      0.86      0.86       276
weighted avg       0.86      0.86      0.86       276

Model Confusion Matrix:
 [[ 99  13]
 [ 26 138]]
```



4.15 Grid Search with Gradient Boosting Classifier

[163]:
```
param_grid = {'n_estimators': [100, 400, 800, 900, 1000], 'learning_rate': [0.
↪1, 0.01, 0.001], 'subsample': [1.0, 0.5],
              'max_features': [2, 3, 4]}

model_gscv_gbc = GridSearchCV(GradientBoostingClassifier(random_state=42),␣
↪param_grid=param_grid, scoring='f1', n_jobs=-1)

gscv_gbc_y_accuracy = model_evaluation(model_gscv_gbc, "Grid Search with␣
↪Gradient Boosting Classifier")
```

```
Model Name: Grid Search with Gradient Boosting Classifier
Model Accuracy: 0.8768
Model Precision: 0.9221
Model Recall: 0.8659
Model Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       112
           1       0.92      0.87      0.89       164

    accuracy                           0.88       276
   macro avg       0.87      0.88      0.87       276
weighted avg       0.88      0.88      0.88       276

Model Confusion Matrix:
 [[100  12]
 [ 22 142]]
```



4.16 Models Accuracy Comparison

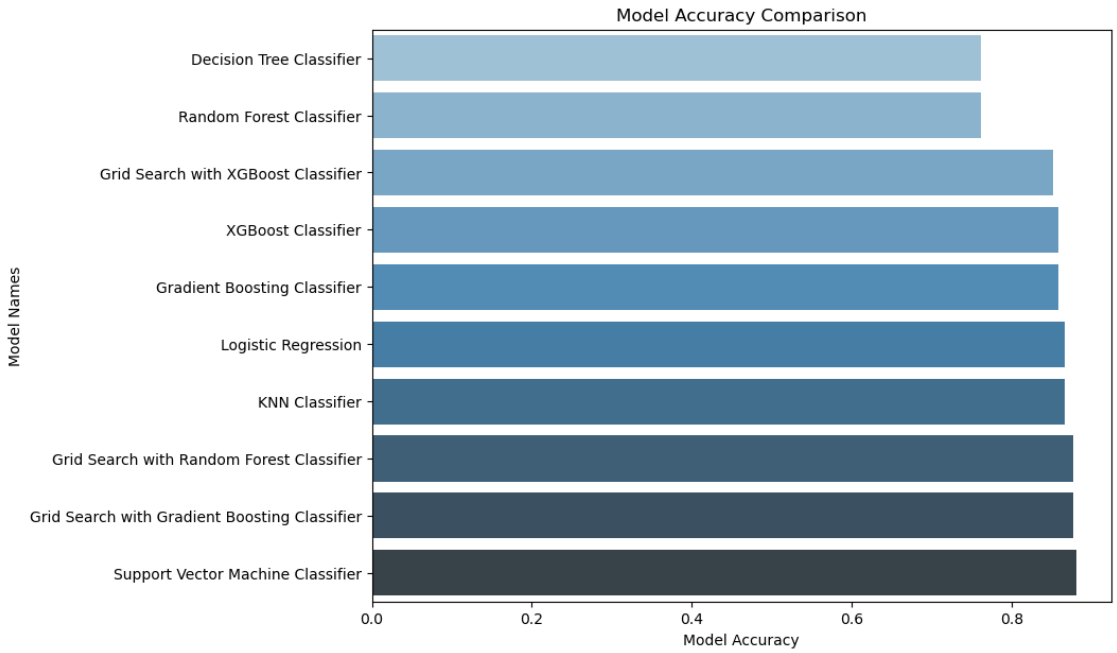
[164]:
```
comparison_accuracy_df = pd.DataFrame(model_accuracy_comparison).
↪rename(columns={0: 'Model Names', 1: 'Model Accuracy'})
comparison_accuracy_df = comparison_accuracy_df.sort_values(by='Model Accuracy')
comparison_accuracy_df.head(20)
```

[164]:

	Model Names	Model Accuracy
3	Decision Tree Classifier	0.760870
4	Random Forest Classifier	0.760870
7	Grid Search with XGBoost Classifier	0.851449
6	XGBoost Classifier	0.858696
8	Gradient Boosting Classifier	0.858696
0	Logistic Regression	0.865942
1	KNN Classifier	0.865942
5	Grid Search with Random Forest Classifier	0.876812
9	Grid Search with Gradient Boosting Classifier	0.876812
2	Support Vector Machine Classifier	0.880435

[165]:
```
plt.figure(figsize=(9, 7))
sns.barplot(x='Model Accuracy', y='Model Names', data=comparison_accuracy_df,␣
↪palette="Blues_d")
plt.title('Model Accuracy Comparison')
```

[165]: Text(0.5, 1.0, 'Model Accuracy Comparison')



5 Model Recommendation, Accuracy and Explainability

(A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.)

In conclusion, I would recommend using the Support Vector Machine (SVM) model. The SVM model was found to have the highest level of accuracy of 0.88. In addition, SVM also saw the lowest rate of false negatives, which in the domain of heart disease detection, is the most crucial to reduce. Both of these factors considered, coupled with its easy explainability make it the best classifier model in this case.

6 Key Findings and Insights

(Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classifier model.)

1. SVM and Grid Search with Gradient Boosting / Random Forest Classifier were the top performing models.
2. Decision Tree and Standard Random Forest classifiers performed less well.

3. Boosting algorithms proved to support underfitting well.
4. Data appears to be normally distributed.

7 Next Steps

(Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model after adding specific data features that may help you achieve a better explanation or a better prediction.)

1. Straitfy the data for Males and Females to ensure the best train and test split of the data to ensure proportionality.
2. Have a larger data set - 919 records is unlikely to show a complete range of patients.
3. Use Cross-Validation to find the best weights/metrics for each model to improve their accuracy.
4. Normalise the data to consider both magnitude and variance.
5. Experiment with even more models, such as AdaBoost, Voting and CatBoost Classifiers.