Predicting Heart failure in patients with commorbidities using deep learning model

1. Introduction

Heart failure prediction in patients with comorbidities is essential for optimizing clinical care and improving patient outcomes. Comorbid conditions such as diabetes, hypertension, and chronic kidney disease significantly impact the management and prognosis of heart failure. Deep learning models offer a promising approach to analyze the intricate interactions between various risk factors and predict heart failure risk accurately. By leveraging large datasets containing diverse patient information, including medical history, demographic data, and laboratory results, these models can uncover complex patterns and provide valuable insights into individual patient risk profiles.

In this study, we develop and assess the performance of a deep learning model specifically tailored to predict heart failure in patients with comorbidities. Through comprehensive data preprocessing and model training, we aim to enhance the accuracy and reliability of heart failure risk predictions. By integrating deep learning techniques into clinical practice, we strive to empower healthcare professionals with advanced tools for early detection and proactive management of heart failure in patients with complex health conditions.

2. Research Design

Data Collection

- Import modules
- Accessing

Import modules

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, con
from tensorflow import keras
import warnings
warnings.filterwarnings('ignore')
```

Accessing

In [2]:
```
heart = pd.read_csv('heart.csv')
heart.head()
```

Out[2]:

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

In [3]:
```
heart.shape
```

Out[3]:
```
(918, 12)
```

In [4]:
```
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
```

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

```
False
False
```

In [6]:
```
categorical = heart.select_dtypes(exclude = 'int')
numerical = heart.select_dtypes(include = 'int')
```

In [7]:
```
categorical.head()
```

Out[7]:

	Sex	ChestPainType	RestingECG	ExerciseAngina	Oldpeak	ST_Slope
0	M	ATA	Normal	N	0.0	Up
1	F	NAP	Normal	N	1.0	Flat
2	M	ATA	ST	N	0.0	Up
3	F	ASY	Normal	Y	1.5	Flat
4	M	NAP	Normal	N	0.0	Up

In [8]:
```
pd.DataFrame(categorical.nunique(), columns = ["Number of unique values"])
```

Out[8]:

	Number of unique values
Sex	2
ChestPainType	4
RestingECG	3
ExerciseAngina	2
Oldpeak	53
ST_Slope	3

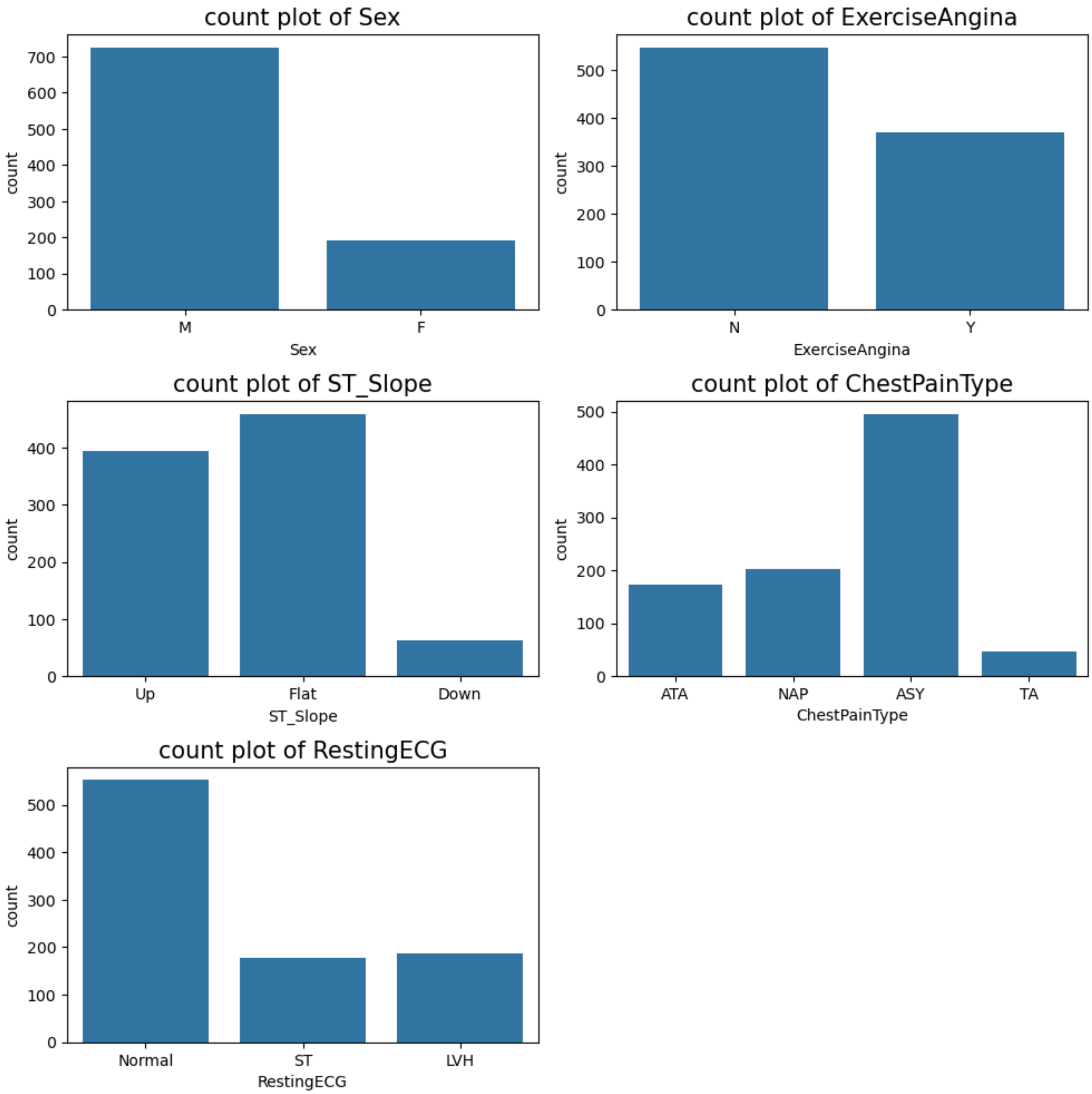
In [9]:
```
categorical.Oldpeak.head()
```

Out[9]:
```
0    0.0
1    1.0
2    0.0
3    1.5
4    0.0
Name: Oldpeak, dtype: float64
```

Oldpeak is a numerical feature

In [10]:
```
categorical = categorical.drop('Oldpeak', axis = 1)
numerical['Oldpeak'] = heart['Oldpeak']
```

In [11]:
```
fig, ax = plt.subplots(nrows = 3, ncols = 2, figsize = (10, 10))
for idx, col in enumerate(categorical.columns):
    axes = ax[(idx) % 3][(idx) % 2]
    sns.countplot(data = categorical, x = col, ax = axes)
    ax[2][1].axis('off')
    axes.set_title(f'count plot of {col}', fontsize = 15)
    fig.tight_layout(pad = 1)
plt.savefig('plots/cat_countplot.png')
```



Data Preprocessing

Numerical Preprocessing

- Removal of outliers
- Standard Scaling
- Label Encoding

In [12]:
```
numerical.head()
```

Out[12]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	HeartDisease	Oldpeak
0	40	140	289	0	172	0	0.0
1	49	160	180	0	156	1	1.0
2	37	130	283	0	98	0	0.0
3	48	138	214	0	108	1	1.5

4	54	150	195	0	122	0	0.0

In [13]:
```
pd.DataFrame(numerical.nunique(), columns = ["Number of unique values"])
```

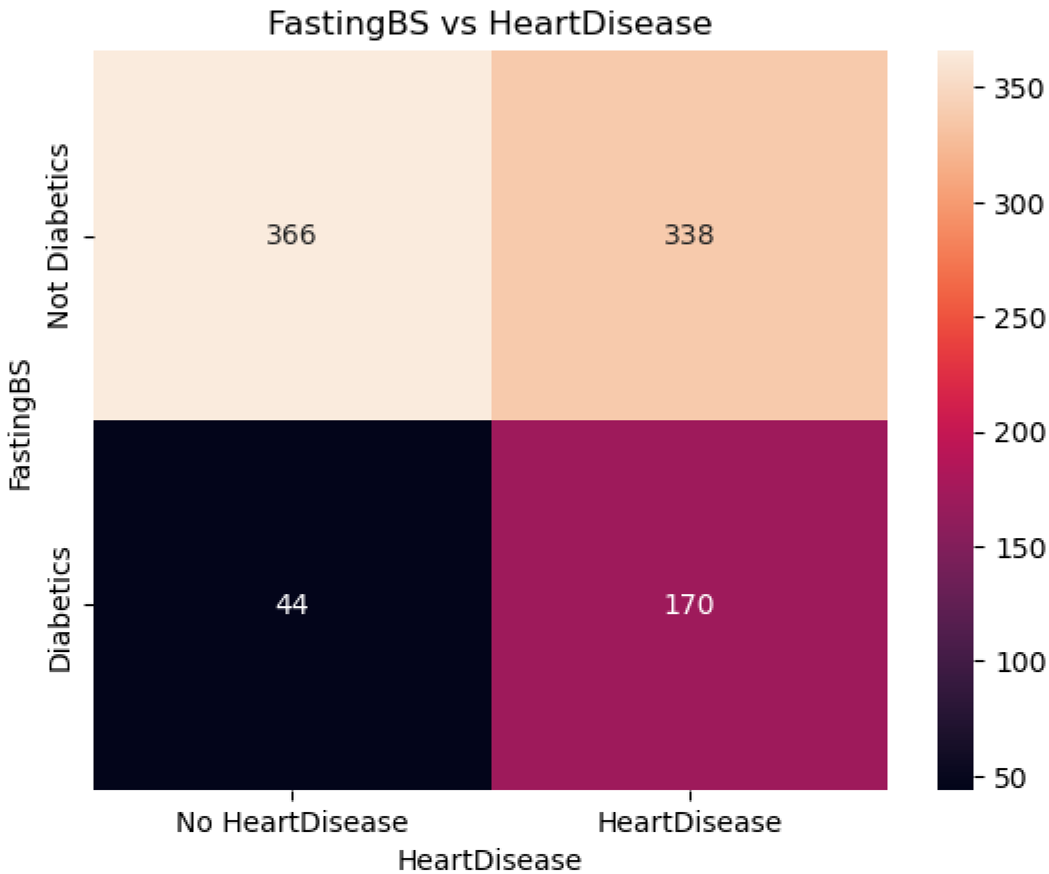
Out[13]:

	Number of unique values
Age	50
RestingBP	67
Cholesterol	222
FastingBS	2
MaxHR	119
HeartDisease	2
Oldpeak	53

FastingBS and HeartDisease are categorical with just two unique values

In [14]:
```
categorical[['FastingBS', 'HeartDisease']] = numerical[['FastingBS', 'HeartDisease']]
numerical.drop(['FastingBS', 'HeartDisease'], axis = 1, inplace = True)
```

In [15]:
```
conf_mat = confusion_matrix(categorical['FastingBS'], categorical['HeartDisease'])
sns.heatmap(conf_mat, annot = True, fmt = '.0f')
plt.title('FastingBS vs HeartDisease')
plt.xlabel('HeartDisease')
plt.ylabel('FastingBS')

plt.xticks([0.5,1.5], ['No HeartDisease', 'HeartDisease'])
plt.yticks([0.5,1.5], ['Not Diabetics', 'Diabetics'])


plt.show()
plt.savefig("plots/fastingbs_heartdisease.png")
```



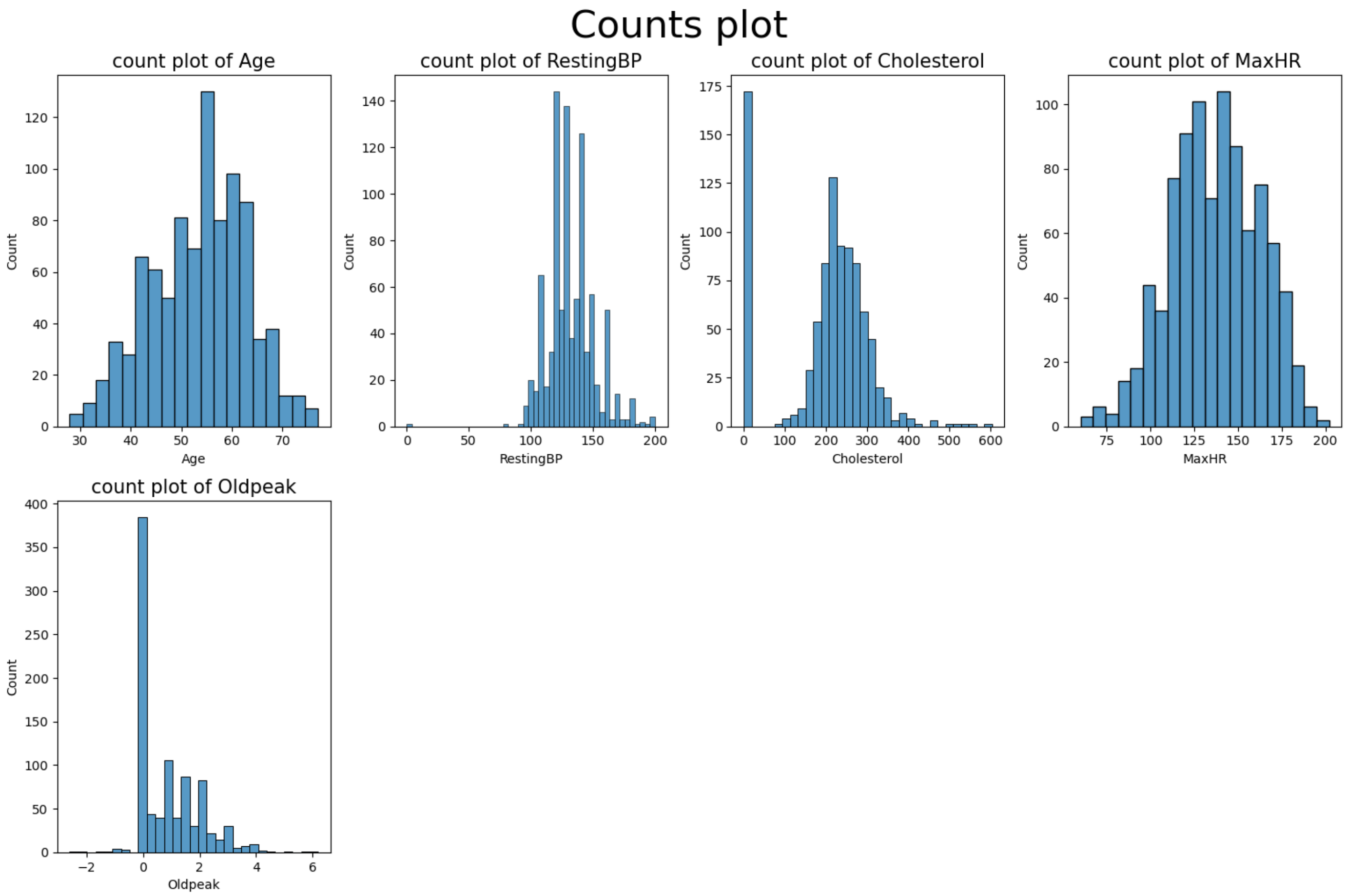
<Figure size 640x480 with 0 Axes>

In [16]:
```
numerical.head()
```

Out[16]:

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

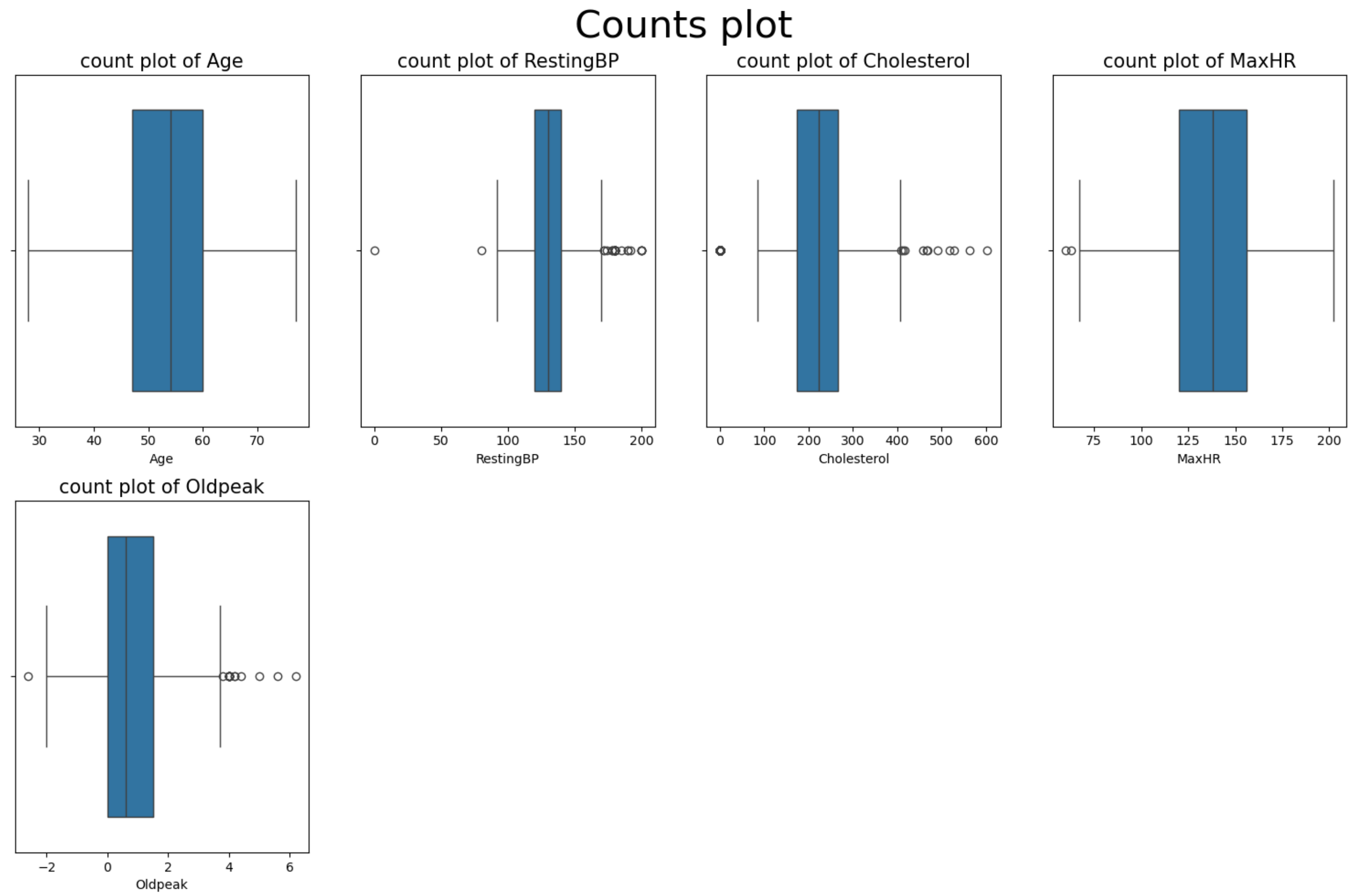
In [17]:
```
fig, ax = plt.subplots(nrows = 2, ncols = 4, figsize = (15, 10))
for idx, col in enumerate(numerical.columns):
    axes = ax[idx // 4][idx % 4]
    sns.histplot(data = numerical, x = col, ax = axes)
    axes.set_title(f'count plot of {col}', fontsize = 15)
    fig.tight_layout(pad = 1)
    plt.suptitle('Counts plot', fontsize = 30)
for idx in range(1,4):
    axes = ax[1][idx].axis('off')

plt.savefig('plots/countplot.png')
```



FastingBS and HeartDisease are categorical

Checking for outliers

In [18]:
```
fig, ax = plt.subplots(nrows = 2, ncols = 4, figsize = (15, 10))
for idx, col in enumerate(numerical.columns):
    axes = ax[idx // 4][idx % 4]
    sns.boxplot(data = numerical, x = col, ax = axes)
    axes.set_title(f'count plot of {col}', fontsize = 15)
    fig.tight_layout(pad = 1)
    plt.suptitle('Counts plot', fontsize = 30)
for idx in range(1,4):
    axes = ax[1][idx].axis('off')

plt.savefig('plots/boxplots.png')
```

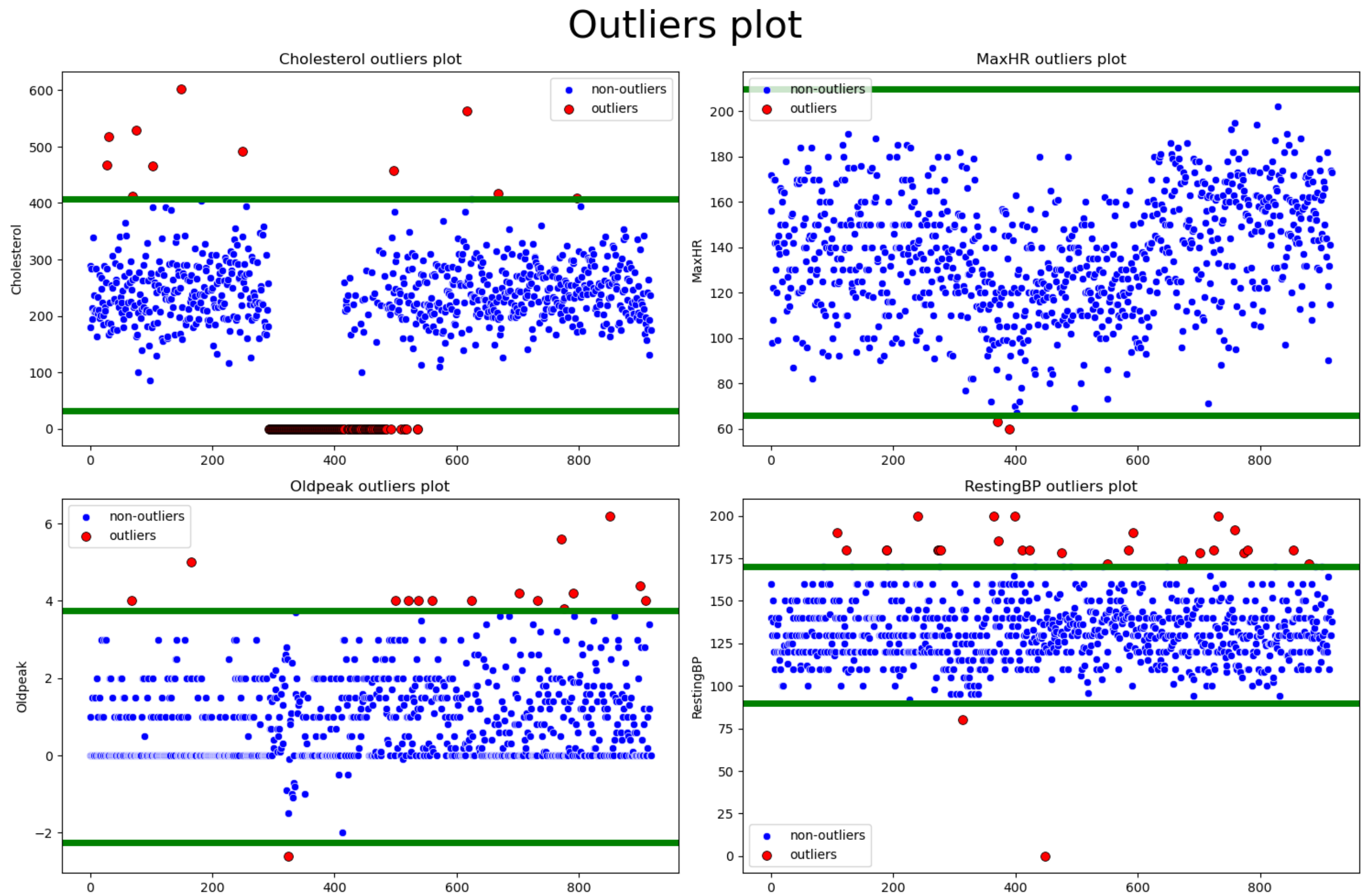


there are some outliers in Cholesterol, MaxHR, Oldpeak and RestingBP

In [19]:

```
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize = (15, 10))
for idx, col in enumerate(['Cholesterol', 'MaxHR', 'Oldpeak', 'RestingBP']):
    axes = ax[idx // 2][idx % 2]

    q1 = numerical[col].quantile(0.75)
    q3 = numerical[col].quantile(0.25)
    iqr = q3 - q1
    upperlimit = q1 - 1.5*iqr
    lowerlimit = q3 + 1.5 * iqr
    outliers = numerical[col][(numerical[col] < lowerlimit) | (numerical[col] > upperlim

    sns.scatterplot(numerical[col], label = 'non-outliers', color = 'blue', ax = axes)
    sns.scatterplot(outliers, label = 'outliers', color = 'red', edgecolors='black', s=5
    axes.axhline(upperlimit, color = 'green', linewidth = 5)
    axes.axhline(lowerlimit, color = 'green', linewidth = 5)
    axes.set_title(f'{col} outliers plot')
    fig.tight_layout(pad = 1)
    plt.suptitle('Outliers plot', fontsize = 30)
    plt.legend()



plt.show()
plt.savefig('plots/outliers.png')
```

```
No artists with labels found to put in legend.  Note that artists whose label start with
an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with
an underscore are ignored when legend() is called with no argument.
No artists with labels found to put in legend.  Note that artists whose label start with
an underscore are ignored when legend() is called with no argument.
```




```
<Figure size 640x480 with 0 Axes>
```

removing the outliers

In [20]:
```
for col in numerical.columns:
    numerical[col] = numerical[col][(numerical[col] > lowerlimit) | (numerical[col] < up
```

scaling the features

In [21]:
```
ss = StandardScaler()
numerical = pd.DataFrame(ss.fit_transform(numerical))
```

Categorical Preprocessing

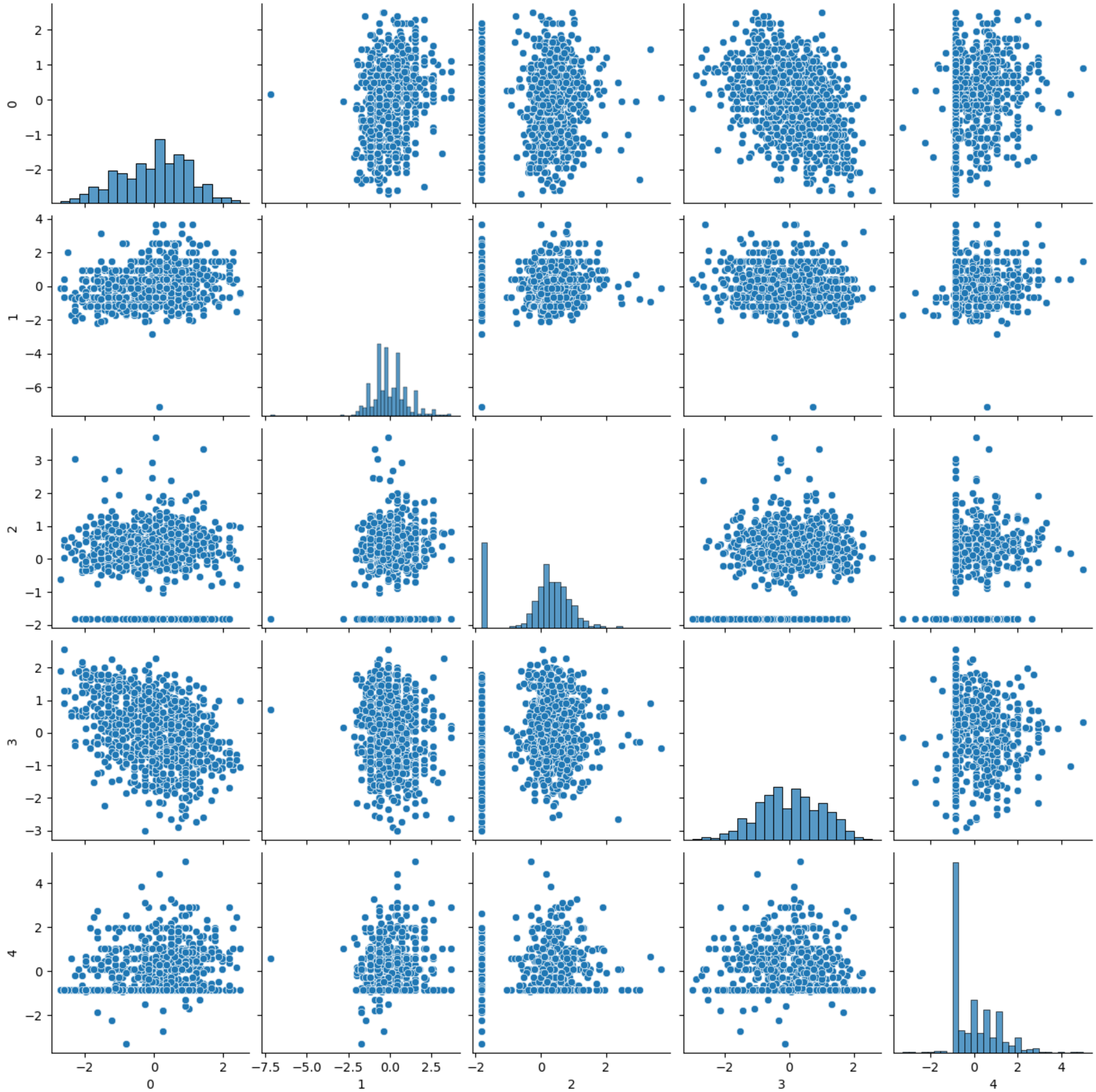
- Label Encoding

label encoding the features

In [22]:
```
le = LabelEncoder()
for i in categorical.columns:
    categorical[i] = le.fit_transform(X = categorical[i])
```

In [23]:
```
categorical[['FastingBS', 'HeartDisease']] = heart[['FastingBS', 'HeartDisease']]
```

In [24]:
```
sns.pairplot(numerical)
plt.savefig('plots/numerical_scatter.png')
```



In [25]:
```python
data = pd.concat([numerical, categorical], axis = 1)
```

In [26]:
```python
data.head()
```

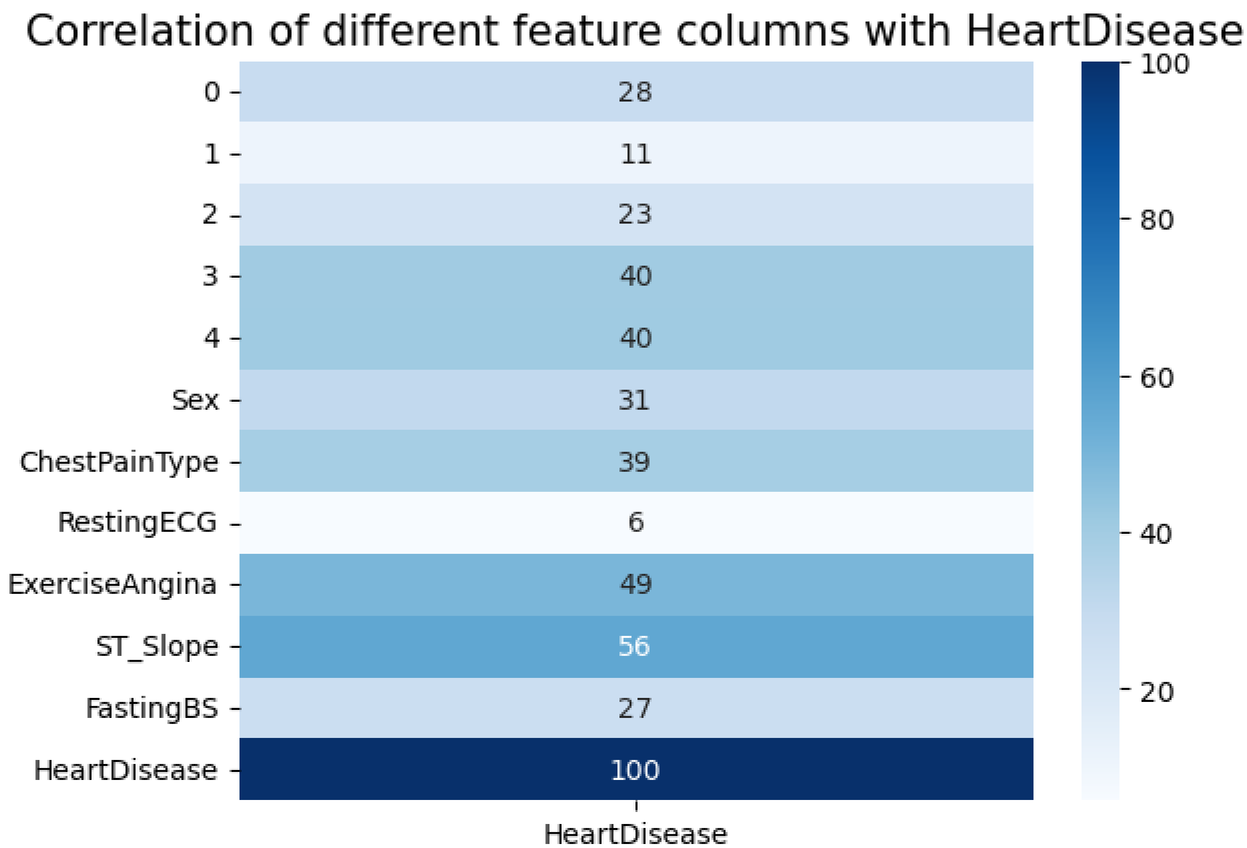
Out[26]:

	0	1	2	3	4	Sex	ChestPainType	RestingECG	ExerciseAngina	ST_
0	-1.433140	0.410909	0.825070	1.382928	-0.832432	1	1	1	0	
1	-0.478484	1.491752	-0.171961	0.754157	0.105664	0	2	1	0	
2	-1.751359	-0.129513	0.770188	-1.525138	-0.832432	1	1	2	0	
3	-0.584556	0.302825	0.139040	-1.132156	0.574711	0	0	1	1	
4	0.051881	0.951331	-0.034755	-0.581981	-0.832432	1	2	1	0	

In [27]:
```python
assert (numerical.shape[0], categorical.shape[1] + numerical.shape[1]) == (data.shape[0]
print('correctly merged')
```

```
correctly merged
```

In [28]:
```
sns.heatmap((data.corr().abs()*100)[['HeartDisease']], annot = True, cmap = 'Blues', fmt
plt.title('Correlation of different feature columns with HeartDisease', fontsize = 15)
plt.savefig('plots/correlation_plot.png')
```



Model training

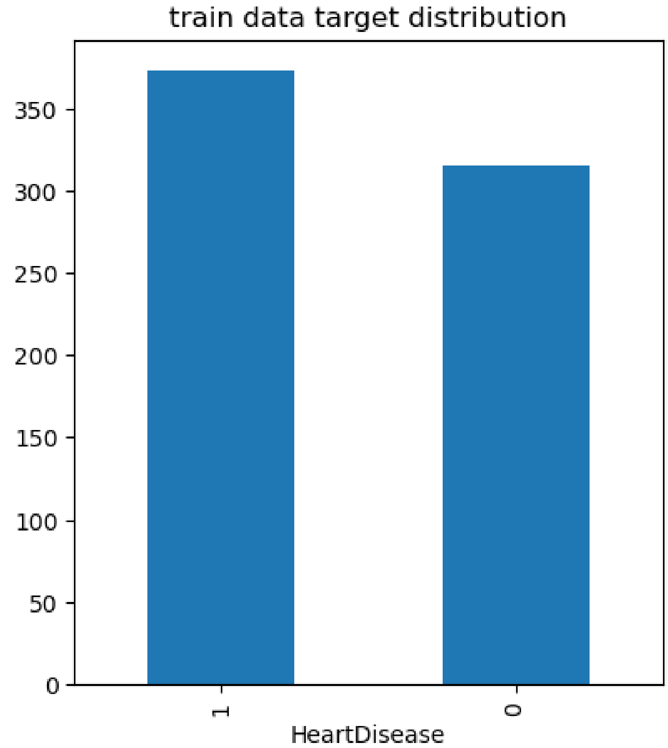
In [29]:
```
X = data.drop('HeartDisease', axis = 1)
y = data['HeartDisease']
```

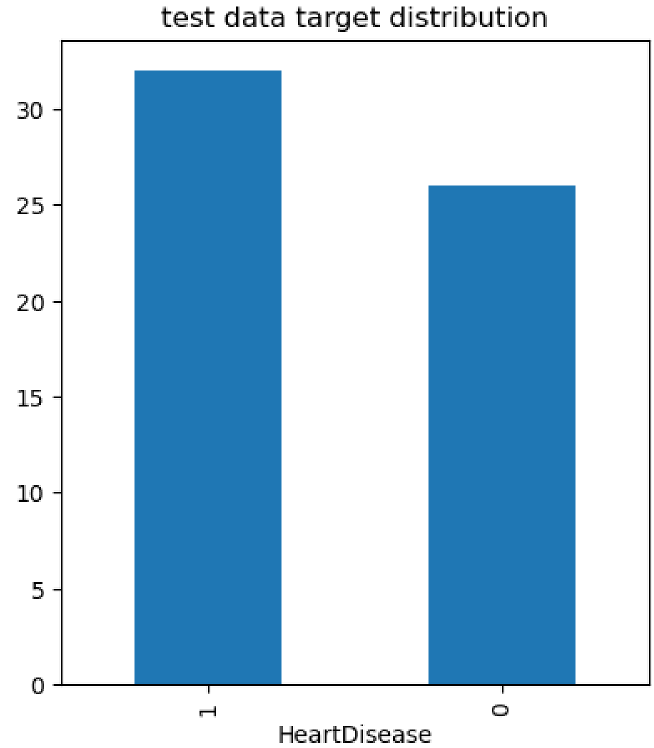
In [30]:
```
train_test_split
```

Out[30]:
```
<function sklearn.model_selection._split.train_test_split(*arrays, test_size=None, train
_size=None, random_state=None, shuffle=True, stratify=None)>
```

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test)
```

In [32]:
```
fig, ax = plt.subplots(1, 2, figsize = (10, 5))
pd.Series(y_train).value_counts().plot(kind = 'bar', ax = ax[0])
pd.Series(y_test).value_counts().plot(kind = 'bar', ax = ax[1])
ax[0].set_title('train data target distribution')
ax[1].set_title('test data target distribution')
plt.savefig('plots/train_test_distribution.png')
```





In [33]:
```
model = keras.Sequential([keras.layers.Dense(units = 50, activation = 'relu'),
                          keras.layers.Dense(units = 32, activation = 'relu'),
                          keras.layers.Dense(units = 32, activation = 'relu'),
                          keras.layers.Dense(units = 32, activation = 'relu'),
                          keras.layers.Dense(units = 32, activation = 'relu'),
                          keras.layers.Dense(units =1, activation = 'sigmoid')])
model.compile(optimizer = 'Adam', loss = 'binary_crossentropy', metrics = 'accuracy')
```

In [34]:
```
history = model.fit(X_train, y_train, validation_data = (X_val, y_val), epochs = 100, ve
```

```
Epoch 1/100
22/22 [==============================] - 3s 40ms/step - loss: 0.6440 - accuracy: 0.6613
- val_loss: 0.5545 - val_accuracy: 0.8140
Epoch 2/100
22/22 [==============================] - 0s 10ms/step - loss: 0.4799 - accuracy: 0.8140
- val_loss: 0.4013 - val_accuracy: 0.8372
Epoch 3/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3765 - accuracy: 0.8416 -
val_loss: 0.3586 - val_accuracy: 0.8779
Epoch 4/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3443 - accuracy: 0.8576 -
val_loss: 0.3576 - val_accuracy: 0.8721
Epoch 5/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3294 - accuracy: 0.8605
- val_loss: 0.3614 - val_accuracy: 0.8721
Epoch 6/100
22/22 [==============================] - 0s 10ms/step - loss: 0.3140 - accuracy: 0.8663
- val_loss: 0.3731 - val_accuracy: 0.8779
Epoch 7/100
22/22 [==============================] - 0s 9ms/step - loss: 0.3044 - accuracy: 0.8663 -
val_loss: 0.3570 - val_accuracy: 0.8895
Epoch 8/100
22/22 [==============================] - 0s 10ms/step - loss: 0.2959 - accuracy: 0.8634
- val_loss: 0.3635 - val_accuracy: 0.8837
Epoch 9/100
22/22 [==============================] - 0s 10ms/step - loss: 0.2807 - accuracy: 0.8823
- val_loss: 0.3864 - val_accuracy: 0.8663
Epoch 10/100
22/22 [==============================] - 0s 10ms/step - loss: 0.2754 - accuracy: 0.8779
- val_loss: 0.3658 - val_accuracy: 0.8953
```

```
Epoch 11/100
22/22 [==============================] - 0s 10ms/step - loss: 0.2714 - accuracy: 0.8852
- val_loss: 0.3737 - val_accuracy: 0.8779
Epoch 12/100
22/22 [==============================] - 0s 9ms/step - loss: 0.2556 - accuracy: 0.8866 -
val_loss: 0.3799 - val_accuracy: 0.9012
Epoch 13/100
22/22 [==============================] - 0s 9ms/step - loss: 0.2579 - accuracy: 0.8924 -
val_loss: 0.3752 - val_accuracy: 0.9012
Epoch 14/100
22/22 [==============================] - 0s 10ms/step - loss: 0.2429 - accuracy: 0.9041
- val_loss: 0.4040 - val_accuracy: 0.8895
Epoch 15/100
22/22 [==============================] - 0s 10ms/step - loss: 0.2250 - accuracy: 0.9172
- val_loss: 0.4244 - val_accuracy: 0.8779
Epoch 16/100
22/22 [==============================] - 0s 9ms/step - loss: 0.2231 - accuracy: 0.9099 -
val_loss: 0.4936 - val_accuracy: 0.8430
Epoch 17/100
22/22 [==============================] - 0s 13ms/step - loss: 0.2185 - accuracy: 0.9201
- val_loss: 0.4643 - val_accuracy: 0.8605
Epoch 18/100
22/22 [==============================] - 0s 9ms/step - loss: 0.2082 - accuracy: 0.9244 -
val_loss: 0.4412 - val_accuracy: 0.8779
Epoch 19/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1939 - accuracy: 0.9244 -
val_loss: 0.4844 - val_accuracy: 0.8663
Epoch 20/100
22/22 [==============================] - 0s 10ms/step - loss: 0.1965 - accuracy: 0.9273
- val_loss: 0.4634 - val_accuracy: 0.8837
Epoch 21/100
22/22 [==============================] - 0s 9ms/step - loss: 0.2103 - accuracy: 0.9215 -
val_loss: 0.4819 - val_accuracy: 0.8779
Epoch 22/100
22/22 [==============================] - 0s 9ms/step - loss: 0.2003 - accuracy: 0.9273 -
val_loss: 0.5250 - val_accuracy: 0.8547
Epoch 23/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1657 - accuracy: 0.9375 -
val_loss: 0.5122 - val_accuracy: 0.8779
Epoch 24/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1669 - accuracy: 0.9360 -
val_loss: 0.5667 - val_accuracy: 0.8605
Epoch 25/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1517 - accuracy: 0.9360 -
val_loss: 0.6317 - val_accuracy: 0.8430
Epoch 26/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1442 - accuracy: 0.9462 -
val_loss: 0.6149 - val_accuracy: 0.8663
Epoch 27/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1326 - accuracy: 0.9549 -
val_loss: 0.5961 - val_accuracy: 0.8837
Epoch 28/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1344 - accuracy: 0.9477 -
val_loss: 0.6680 - val_accuracy: 0.8605
Epoch 29/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1244 - accuracy: 0.9535 -
val_loss: 0.6441 - val_accuracy: 0.8895
Epoch 30/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1113 - accuracy: 0.9593 -
val_loss: 0.7219 - val_accuracy: 0.8547
Epoch 31/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1026 - accuracy: 0.9651 -
val_loss: 0.7850 - val_accuracy: 0.8430
Epoch 32/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1066 - accuracy: 0.9564 -
val_loss: 0.7874 - val_accuracy: 0.8488
```

```
Epoch 33/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0982 - accuracy: 0.9593 -
val_loss: 0.7110 - val_accuracy: 0.8779
Epoch 34/100
22/22 [==============================] - 0s 9ms/step - loss: 0.1256 - accuracy: 0.9477 -
val_loss: 0.8575 - val_accuracy: 0.8372
Epoch 35/100
22/22 [==============================] - 0s 12ms/step - loss: 0.1124 - accuracy: 0.9520
- val_loss: 0.9410 - val_accuracy: 0.8140
Epoch 36/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0981 - accuracy: 0.9637
- val_loss: 0.7739 - val_accuracy: 0.8663
Epoch 37/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0887 - accuracy: 0.9695 -
val_loss: 0.8231 - val_accuracy: 0.8663
Epoch 38/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0758 - accuracy: 0.9753 -
val_loss: 0.8540 - val_accuracy: 0.8488
Epoch 39/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0705 - accuracy: 0.9753 -
val_loss: 0.8734 - val_accuracy: 0.8547
Epoch 40/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0603 - accuracy: 0.9826 -
val_loss: 1.0080 - val_accuracy: 0.8372
Epoch 41/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0662 - accuracy: 0.9753 -
val_loss: 0.9231 - val_accuracy: 0.8663
Epoch 42/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0618 - accuracy: 0.9797 -
val_loss: 0.9735 - val_accuracy: 0.8547
Epoch 43/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0523 - accuracy: 0.9855 -
val_loss: 1.0920 - val_accuracy: 0.8372
Epoch 44/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0523 - accuracy: 0.9840 -
val_loss: 1.0856 - val_accuracy: 0.8488
Epoch 45/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0449 - accuracy: 0.9898
- val_loss: 0.9830 - val_accuracy: 0.8663
Epoch 46/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0495 - accuracy: 0.9840 -
val_loss: 1.1420 - val_accuracy: 0.8547
Epoch 47/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0362 - accuracy: 0.9927 -
val_loss: 1.2391 - val_accuracy: 0.8256
Epoch 48/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0505 - accuracy: 0.9884 -
val_loss: 1.1699 - val_accuracy: 0.8547
Epoch 49/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0380 - accuracy: 0.9898 -
val_loss: 1.1406 - val_accuracy: 0.8547
Epoch 50/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0354 - accuracy: 0.9927
- val_loss: 1.3217 - val_accuracy: 0.8372
Epoch 51/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0519 - accuracy: 0.9840 -
val_loss: 1.2615 - val_accuracy: 0.8430
Epoch 52/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0363 - accuracy: 0.9884 -
val_loss: 1.0898 - val_accuracy: 0.8663
Epoch 53/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0366 - accuracy: 0.9898 -
val_loss: 1.1762 - val_accuracy: 0.8663
Epoch 54/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0256 - accuracy: 0.9971 -
val_loss: 1.2317 - val_accuracy: 0.8663
```

```
Epoch 55/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0220 - accuracy: 0.9956 -
val_loss: 1.3156 - val_accuracy: 0.8547
Epoch 56/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0210 - accuracy: 0.9971 -
val_loss: 1.3116 - val_accuracy: 0.8547
Epoch 57/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0272 - accuracy: 0.9927 -
val_loss: 1.4931 - val_accuracy: 0.8256
Epoch 58/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0168 - accuracy: 0.9971 -
val_loss: 1.4536 - val_accuracy: 0.8256
Epoch 59/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0163 - accuracy: 0.9971 -
val_loss: 1.5411 - val_accuracy: 0.8256
Epoch 60/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0159 - accuracy: 0.9971 -
val_loss: 1.3595 - val_accuracy: 0.8721
Epoch 61/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0283 - accuracy: 0.9942 -
val_loss: 1.6943 - val_accuracy: 0.8256
Epoch 62/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0242 - accuracy: 0.9942 -
val_loss: 1.3586 - val_accuracy: 0.8837
Epoch 63/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0404 - accuracy: 0.9826 -
val_loss: 1.7207 - val_accuracy: 0.8140
Epoch 64/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0601 - accuracy: 0.9826 -
val_loss: 1.4496 - val_accuracy: 0.8314
Epoch 65/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0786 - accuracy: 0.9767 -
val_loss: 1.3803 - val_accuracy: 0.8488
Epoch 66/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0701 - accuracy: 0.9782 -
val_loss: 1.2212 - val_accuracy: 0.8605
Epoch 67/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0312 - accuracy: 0.9913 -
val_loss: 1.2279 - val_accuracy: 0.8605
Epoch 68/100
22/22 [==============================] - 0s 11ms/step - loss: 0.0262 - accuracy: 0.9927
- val_loss: 1.3129 - val_accuracy: 0.8663
Epoch 69/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0149 - accuracy: 0.9971 -
val_loss: 1.3992 - val_accuracy: 0.8430
Epoch 70/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0163 - accuracy: 0.9971 -
val_loss: 1.5793 - val_accuracy: 0.8430
Epoch 71/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0098 - accuracy: 0.9985 -
val_loss: 1.5288 - val_accuracy: 0.8488
Epoch 72/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0091 - accuracy: 0.9971 -
val_loss: 1.4953 - val_accuracy: 0.8488
Epoch 73/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0071 - accuracy: 0.9985 -
val_loss: 1.6332 - val_accuracy: 0.8430
Epoch 74/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0068 - accuracy: 0.9985 -
val_loss: 1.6138 - val_accuracy: 0.8488
Epoch 75/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0059 - accuracy: 0.9985 -
val_loss: 1.6298 - val_accuracy: 0.8488
Epoch 76/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0052 - accuracy: 0.9985 -
val_loss: 1.6421 - val_accuracy: 0.8488
```

```
Epoch 77/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0048 - accuracy: 0.9985 -
val_loss: 1.6312 - val_accuracy: 0.8488
Epoch 78/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0050 - accuracy: 1.0000 -
val_loss: 1.7480 - val_accuracy: 0.8488
Epoch 79/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0052 - accuracy: 0.9985 -
val_loss: 1.7667 - val_accuracy: 0.8488
Epoch 80/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0051 - accuracy: 1.0000 -
val_loss: 1.6669 - val_accuracy: 0.8488
Epoch 81/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0031 - accuracy: 1.0000 -
val_loss: 1.7943 - val_accuracy: 0.8488
Epoch 82/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0033 - accuracy: 1.0000 -
val_loss: 1.7702 - val_accuracy: 0.8488
Epoch 83/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0029 - accuracy: 1.0000 -
val_loss: 1.7833 - val_accuracy: 0.8488
Epoch 84/100
22/22 [==============================] - 0s 13ms/step - loss: 0.0025 - accuracy: 1.0000
- val_loss: 1.8413 - val_accuracy: 0.8488
Epoch 85/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0023 - accuracy: 1.0000 -
val_loss: 1.8531 - val_accuracy: 0.8488
Epoch 86/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0022 - accuracy: 1.0000 -
val_loss: 1.8312 - val_accuracy: 0.8488
Epoch 87/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0021 - accuracy: 1.0000 -
val_loss: 1.9058 - val_accuracy: 0.8488
Epoch 88/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0019 - accuracy: 1.0000 -
val_loss: 1.9091 - val_accuracy: 0.8488
Epoch 89/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0017 - accuracy: 1.0000
- val_loss: 1.9471 - val_accuracy: 0.8430
Epoch 90/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0017 - accuracy: 1.0000
- val_loss: 1.8918 - val_accuracy: 0.8488
Epoch 91/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0016 - accuracy: 1.0000
- val_loss: 1.9731 - val_accuracy: 0.8488
Epoch 92/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0014 - accuracy: 1.0000 -
val_loss: 1.9588 - val_accuracy: 0.8488
Epoch 93/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0014 - accuracy: 1.0000
- val_loss: 1.9909 - val_accuracy: 0.8488
Epoch 94/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0014 - accuracy: 1.0000 -
val_loss: 2.0315 - val_accuracy: 0.8430
Epoch 95/100
22/22 [==============================] - 0s 11ms/step - loss: 0.0013 - accuracy: 1.0000
- val_loss: 2.0031 - val_accuracy: 0.8488
Epoch 96/100
22/22 [==============================] - 0s 10ms/step - loss: 0.0012 - accuracy: 1.0000
- val_loss: 2.0306 - val_accuracy: 0.8488
Epoch 97/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0011 - accuracy: 1.0000 -
val_loss: 2.0338 - val_accuracy: 0.8488
Epoch 98/100
22/22 [==============================] - 0s 9ms/step - loss: 0.0011 - accuracy: 1.0000 -
val_loss: 2.0889 - val_accuracy: 0.8372
```

```
Epoch 99/100
22/22 [==============================] - 0s 8ms/step - loss: 0.0010 - accuracy: 1.0000 -
val_loss: 2.0779 - val_accuracy: 0.8488
Epoch 100/100
22/22 [==============================] - 0s 8ms/step - loss: 9.9752e-04 - accuracy: 1.00
00 - val_loss: 2.0878 - val_accuracy: 0.8430
```

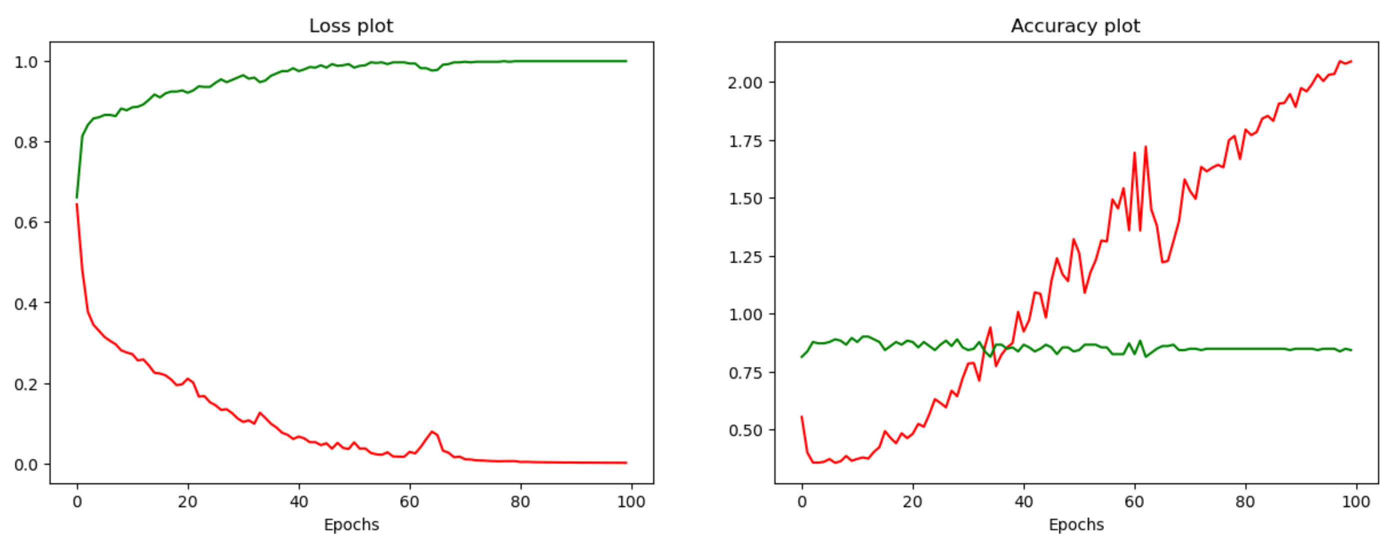
In [35]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                600

 dense_1 (Dense)             (None, 32)                1632

 dense_2 (Dense)             (None, 32)                1056

 dense_3 (Dense)             (None, 32)                1056

 dense_4 (Dense)             (None, 32)                1056

 dense_5 (Dense)             (None, 1)                 33

=================================================================
Total params: 5,433
Trainable params: 5,433
Non-trainable params: 0
_________________________________________________________________
```

In [36]:
```
history.history.keys()
```

Out[36]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [37]:
```
figure, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (15,5))
sns.lineplot(x = np.arange(0, 100), y = history.history['loss'], ax = ax[0], color = 'r'
sns.lineplot(x = np.arange(0, 100), y = history.history['accuracy'], ax = ax[0], color =

sns.lineplot(x = np.arange(0, 100), y = history.history['val_loss'], ax = ax[1], color =
sns.lineplot(x = np.arange(0, 100), y = history.history['val_accuracy'], ax = ax[1], col

ax[0].set_title('Loss plot')
ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs')
ax[1].set_title('Accuracy plot')
plt.savefig('plots/loss_acc_plot.png')
```



Evaluation

- Accuracy Score
- Recall Score
- Precision Score
- F1 Score
- Confusion Matrix

In [38]:
```
prediction = model.predict(X_test)
prediction = np.round(prediction)
```

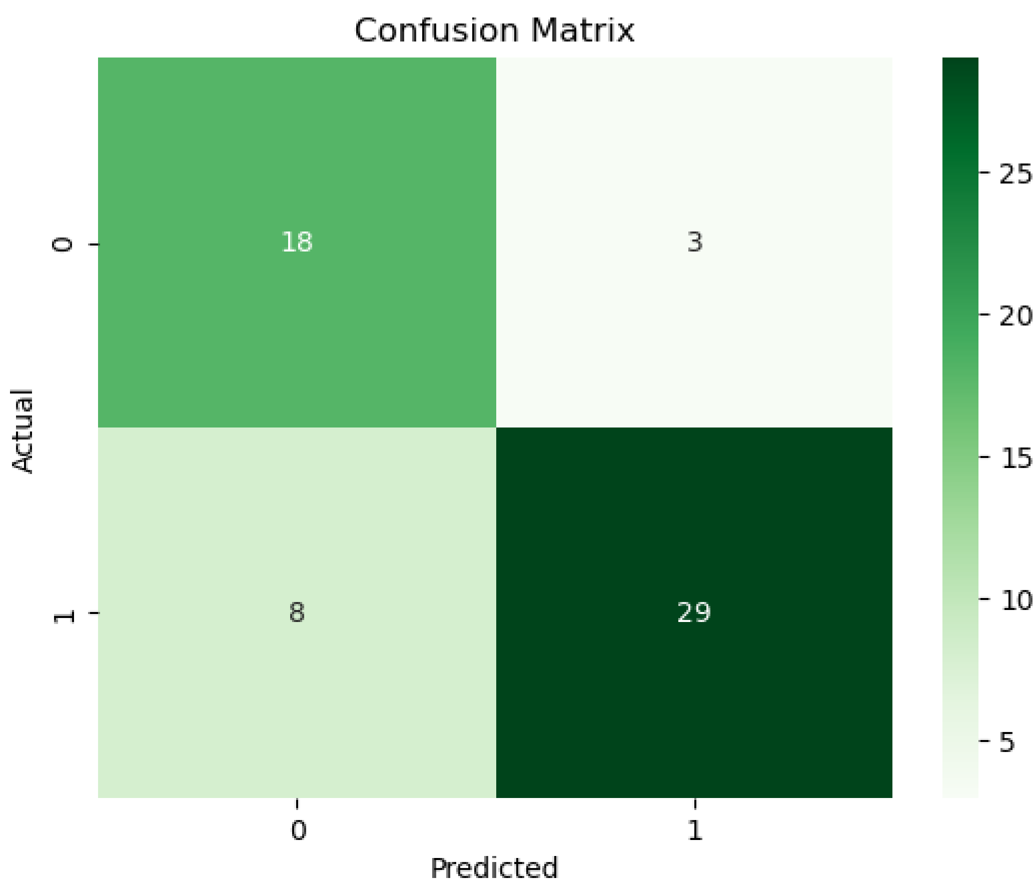
```
2/2 [==============================] - 0s 8ms/step
```

In [39]:
```
print(f'accuracy score: {accuracy_score(prediction, y_test)}')
print(f'recall score: {recall_score(prediction, y_test)}')
print(f'precision score: {precision_score(prediction, y_test)}')
print(f'f1 score: {f1_score(prediction, y_test)}')
```

```
accuracy score: 0.8103448275862069
recall score: 0.7837837837837838
precision score: 0.90625
f1 score: 0.8405797101449275
```

In [40]:
```
conf_mat = confusion_matrix(prediction, y_test)
```

In [41]:
```
sns.heatmap(conf_mat, annot = True, fmt = '.0f', cmap = 'Greens')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
plt.savefig('plots/conf_matrix.png')
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:

```
In [ ]:
```