code-team-neon

August 3, 2023

0.1 Importing libraries

[54]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew
from scipy.stats import kurtosis
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn import linear_model
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.cluster import KMeans
```

0.2 Reading data

[55]:
```
df=pd.read_csv('heart (1).csv')
df.head()
```

[55]:
```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
```

```
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

[56]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

[57]:
```
df.describe()
```

[57]:
```
               age          sex           cp     trestbps        chol  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.00000
mean     54.434146     0.695610     0.942439   131.611707   246.00000
std       9.072290     0.460373     1.029641    17.516718    51.59251
min      29.000000     0.000000     0.000000    94.000000   126.00000
25%      48.000000     0.000000     0.000000   120.000000   211.00000
50%      56.000000     1.000000     1.000000   130.000000   240.00000
75%      61.000000     1.000000     2.000000   140.000000   275.00000
max      77.000000     1.000000     3.000000   200.000000   564.00000

               fbs      restecg      thalach        exang      oldpeak  \
count  1025.000000  1025.000000  1025.000000  1025.000000  1025.000000
mean      0.149268     0.529756   149.114146     0.336585     1.071512
std       0.356527     0.527878    23.005724     0.472772     1.175053
min       0.000000     0.000000    71.000000     0.000000     0.000000
```

```
25%         0.000000     0.000000   132.000000     0.000000     0.000000
50%         0.000000     1.000000   152.000000     0.000000     0.800000
75%         0.000000     1.000000   166.000000     1.000000     1.800000
max         1.000000     2.000000   202.000000     1.000000     6.200000

              slope           ca         thal       target
count   1025.000000  1025.000000  1025.000000  1025.000000
mean       1.385366     0.754146     2.323902     0.513171
std        0.617755     1.030798     0.620660     0.500070
min        0.000000     0.000000     0.000000     0.000000
25%        1.000000     0.000000     2.000000     0.000000
50%        1.000000     0.000000     2.000000     1.000000
75%        2.000000     1.000000     3.000000     1.000000
max        2.000000     4.000000     3.000000     1.000000
```

[58]:
```
df.shape
```

[58]: (1025, 14)

[59]:
```
df.isnull().sum()
```

[59]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

[60]:
```
df.dtypes
```

[60]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
```

```
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

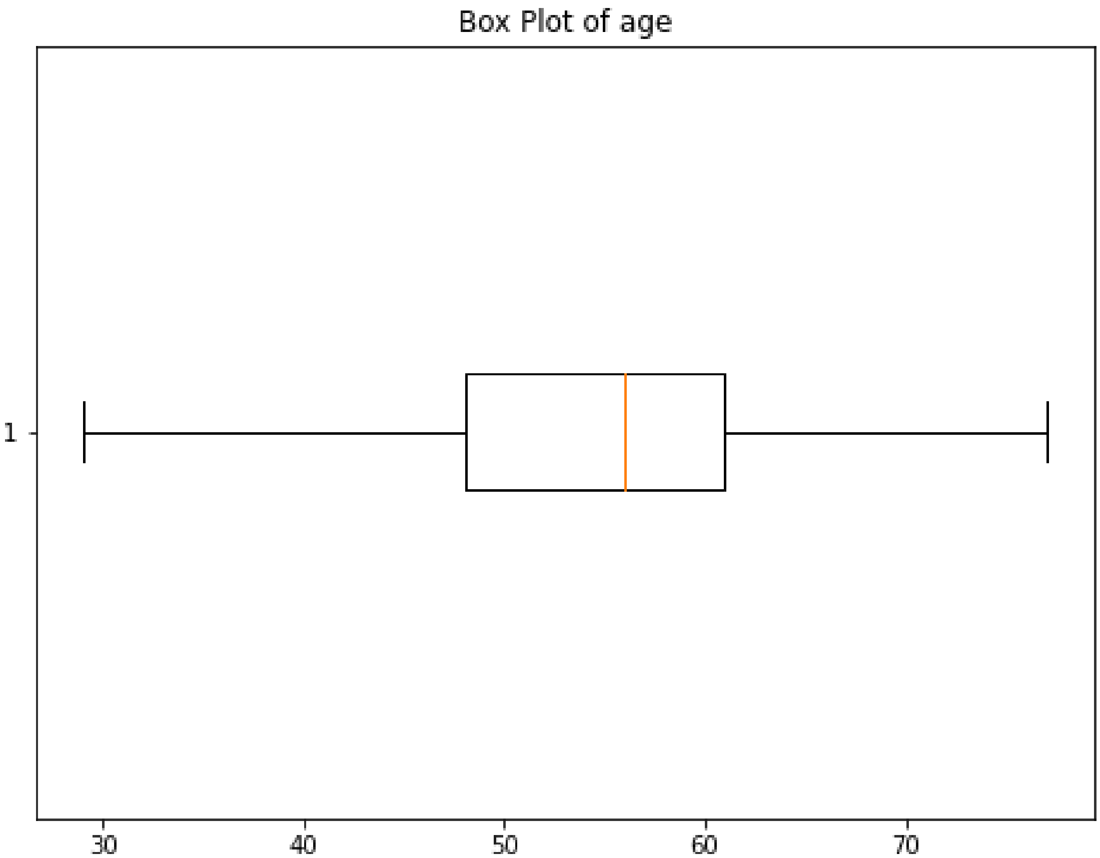
[61]:
```
df.shape
```

[61]:
```
(1025, 14)
```

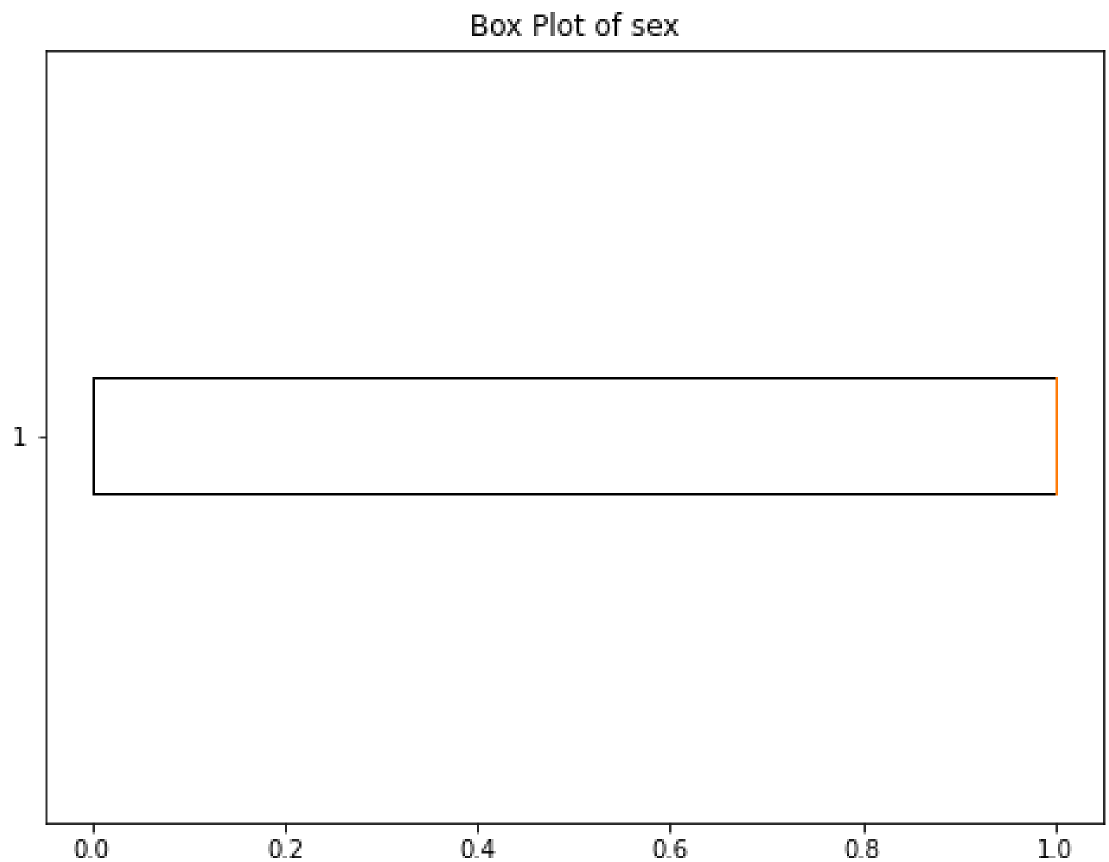
0.3 Check Outliers

[62]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming your dataset is loaded into the DataFrame called 'df'
# Replace 'df' with your actual DataFrame name if different.

# List of columns with numeric data
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns

# Function to detect and visualize outliers using box plots
def detect_and_visualize_outliers(data_frame, column_name):
    # Create a box plot for the given column
    plt.figure(figsize=(8, 6))
    plt.boxplot(data_frame[column_name], vert=False)
    plt.title(f'Box Plot of {column_name}')
    plt.show()

    # Calculate the Interquartile Range (IQR) for the column
    Q1 = data_frame[column_name].quantile(0.25)
    Q3 = data_frame[column_name].quantile(0.75)
    IQR = Q3 - Q1

    # Calculate the lower and upper bounds for outlier detection
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Find and display the outliers
    outliers = data_frame[(data_frame[column_name] < lower_bound) | 
 ↪(data_frame[column_name] > upper_bound)]
    print(f"Outliers in '{column_name}':")
#     print(outliers)

# Loop through each numeric column and visualize outliers
```

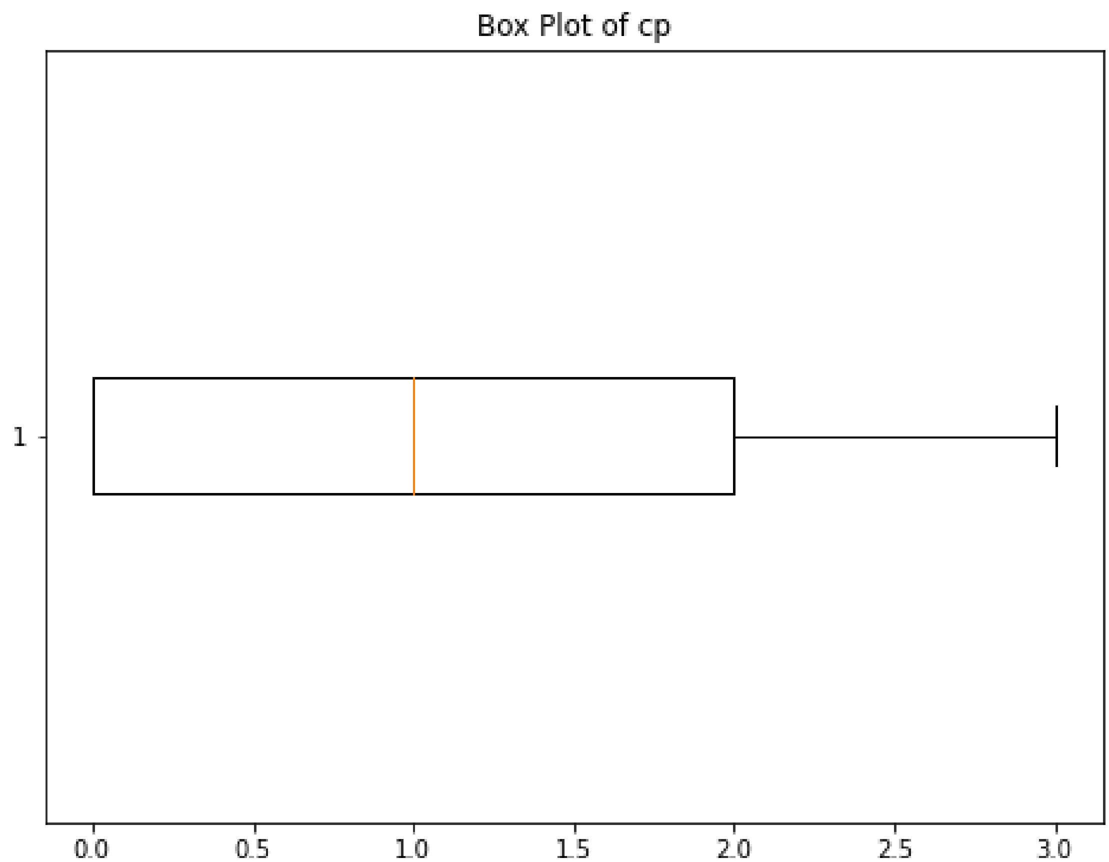
```
for col in numeric_columns:
    detect_and_visualize_outliers(df, col)
```



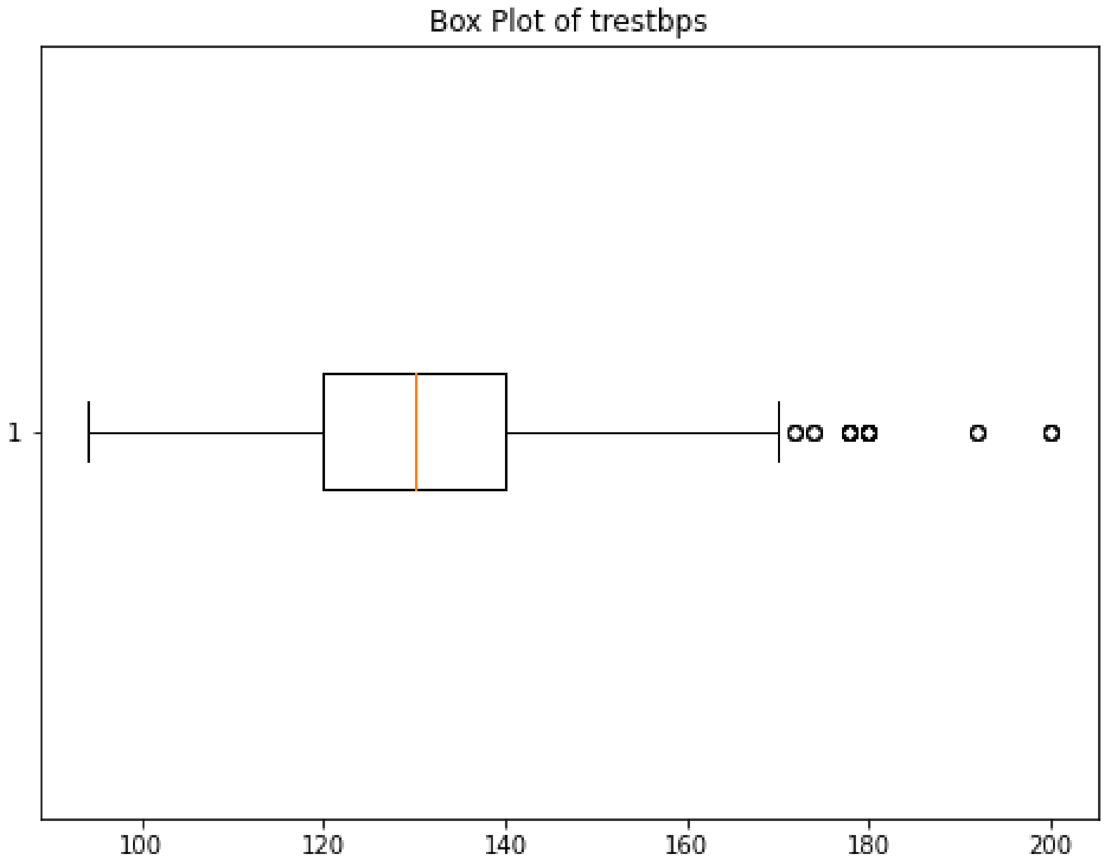

```
Outliers in 'age':
```



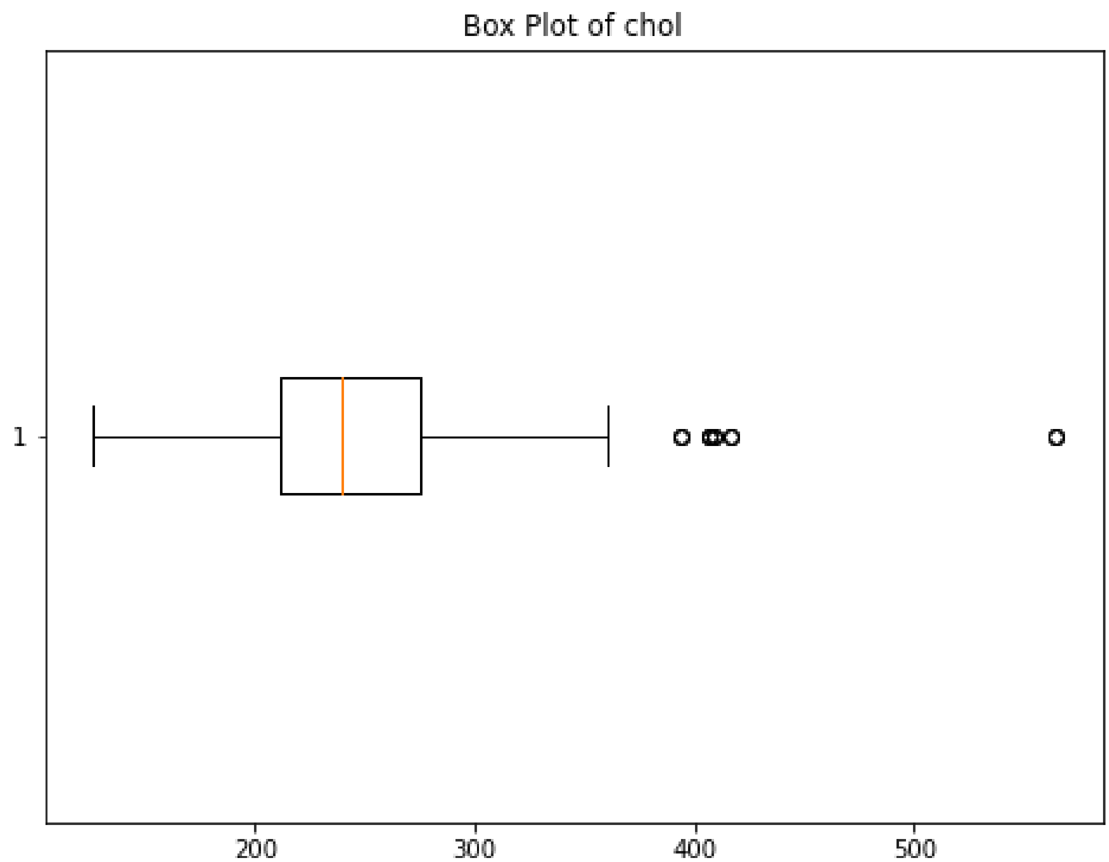

```
Outliers in 'sex':
```



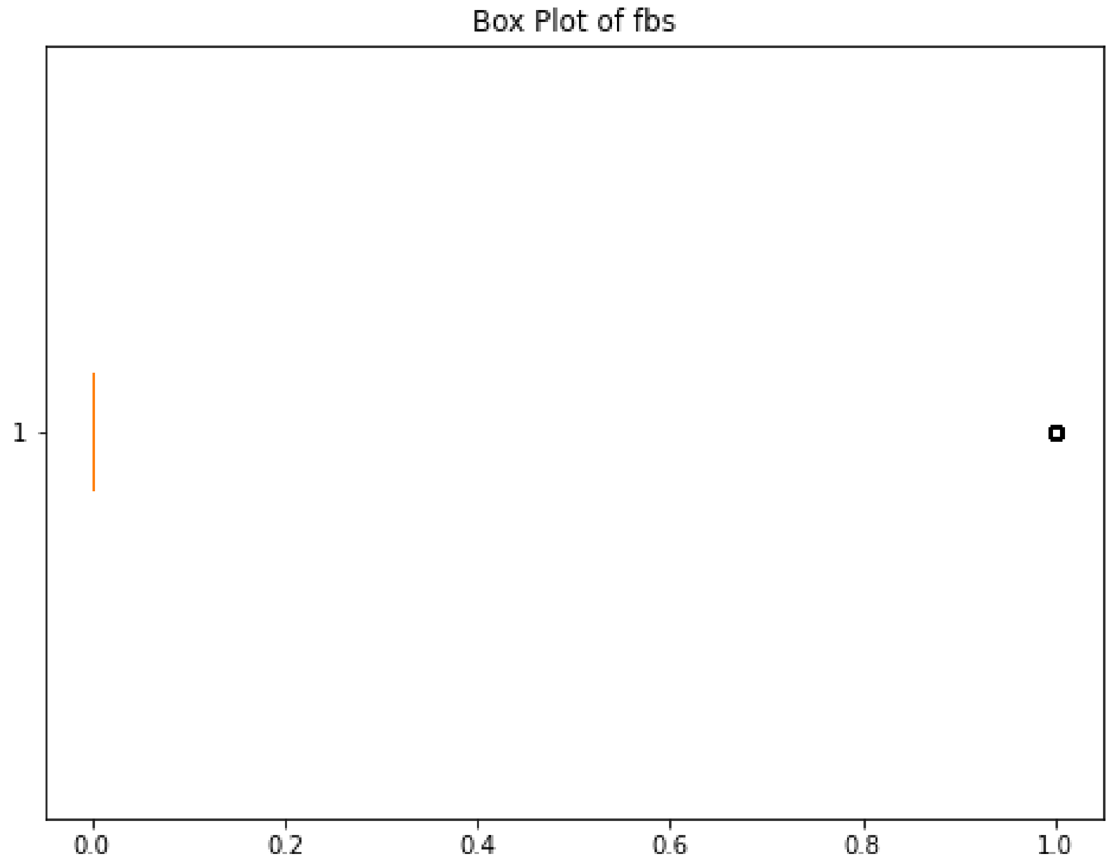

```
Outliers in 'cp':
```



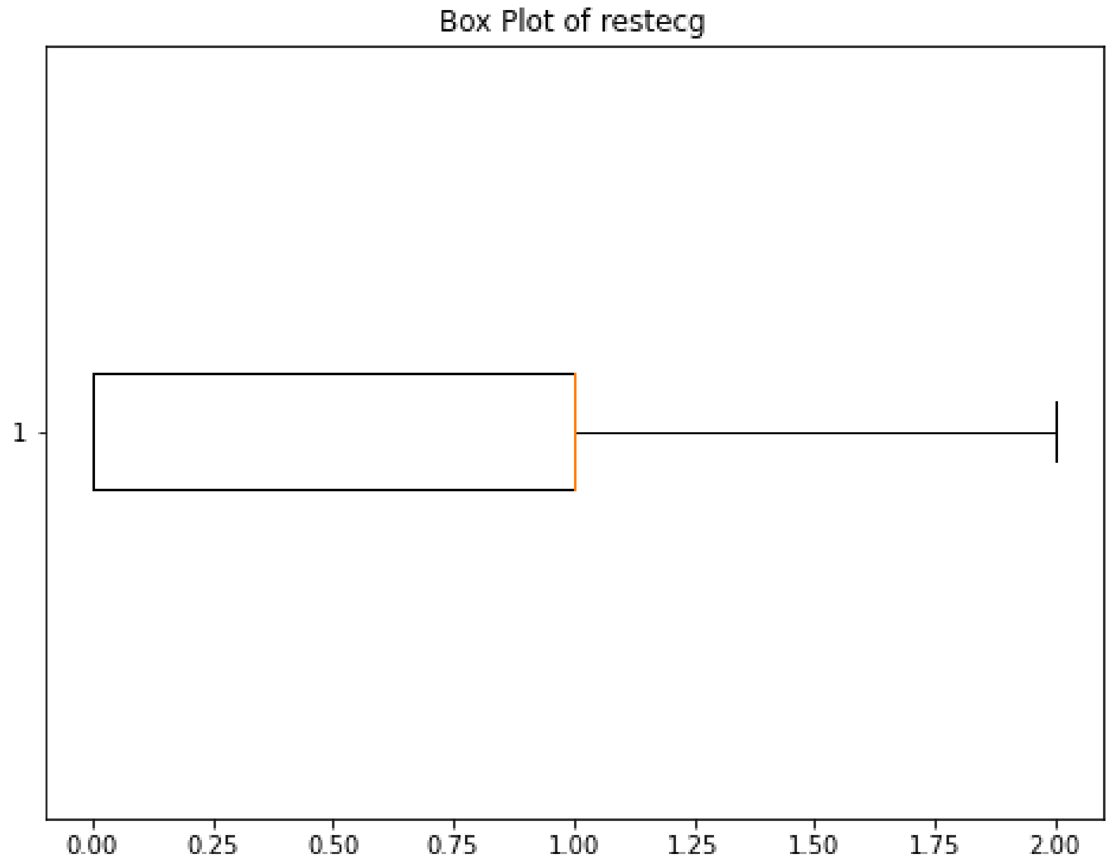

```
Outliers in 'trestbps':
```



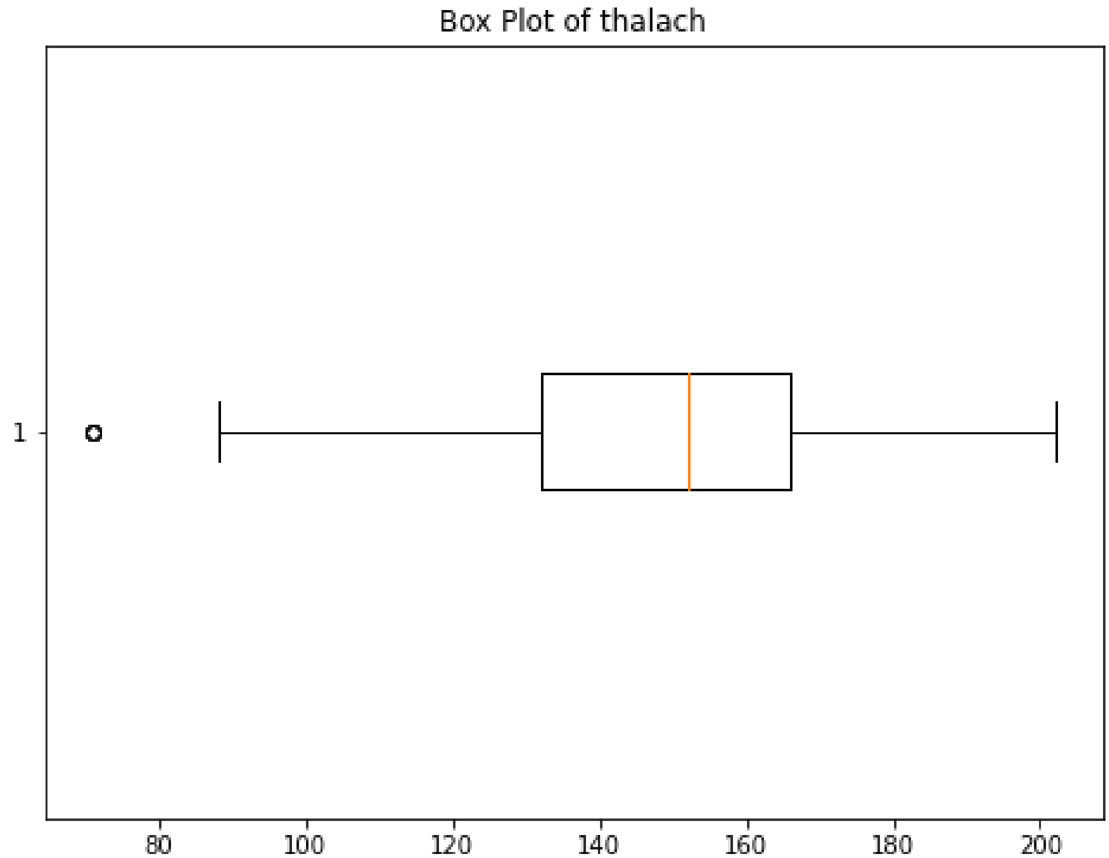

```
Outliers in 'chol':
```



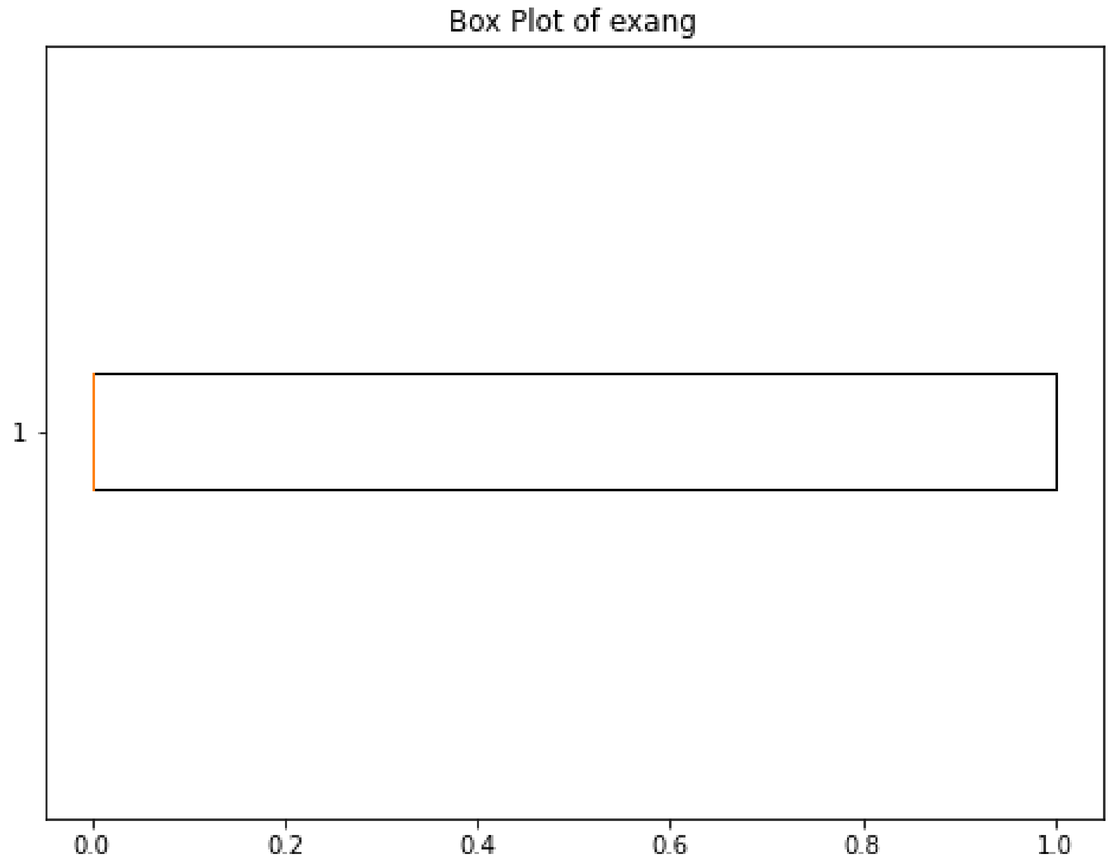

```
Outliers in 'fbs':
```



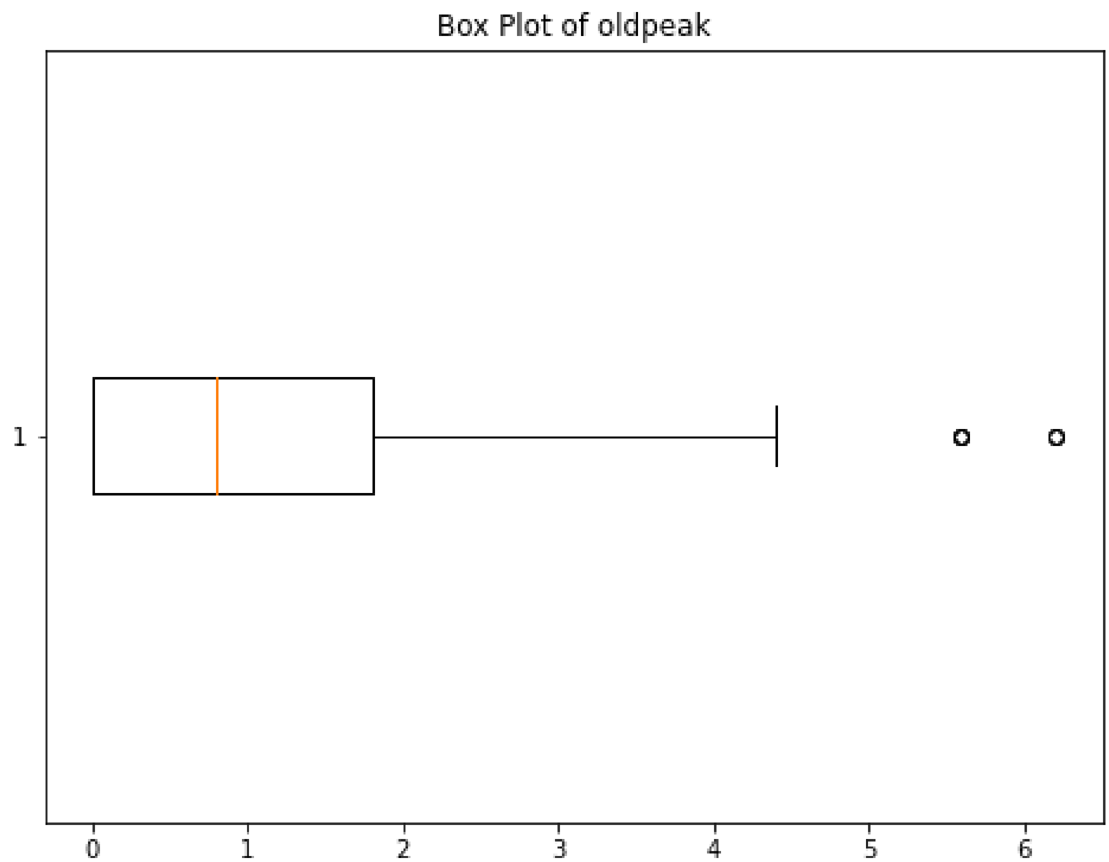

```
Outliers in 'restecg':
```



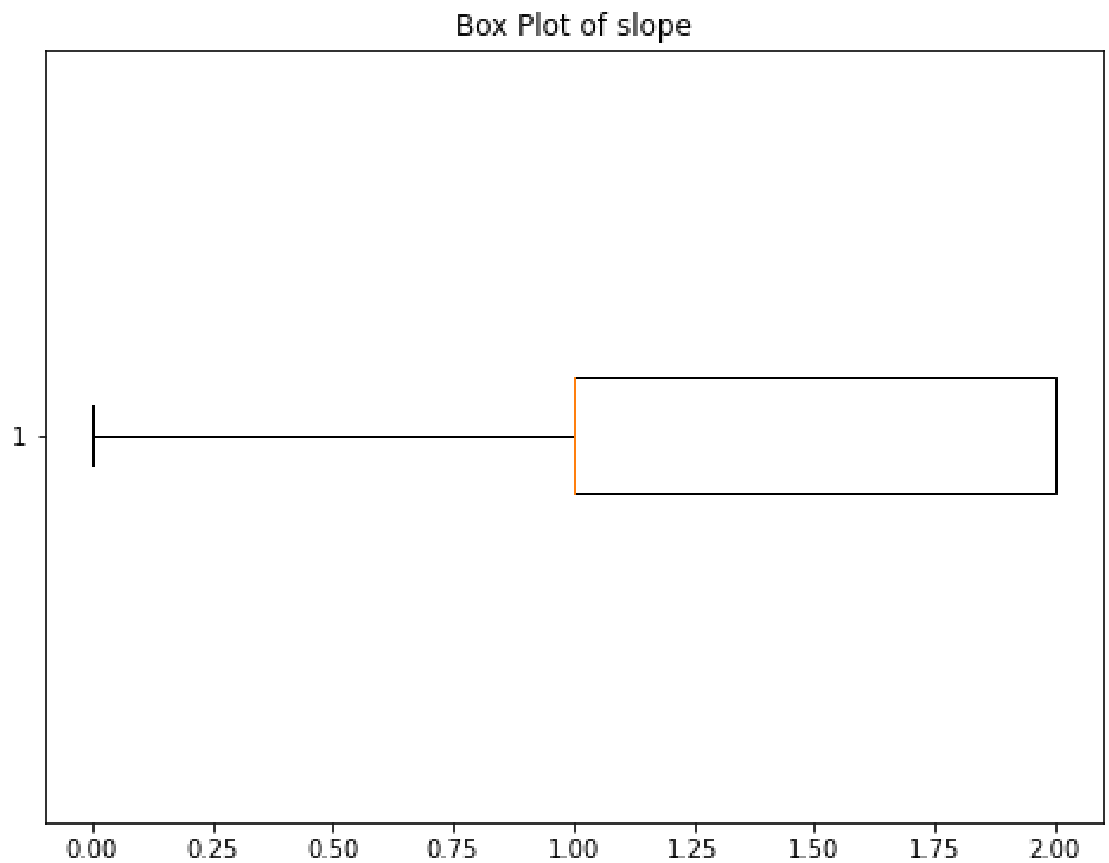

```
Outliers in 'thalach':
```



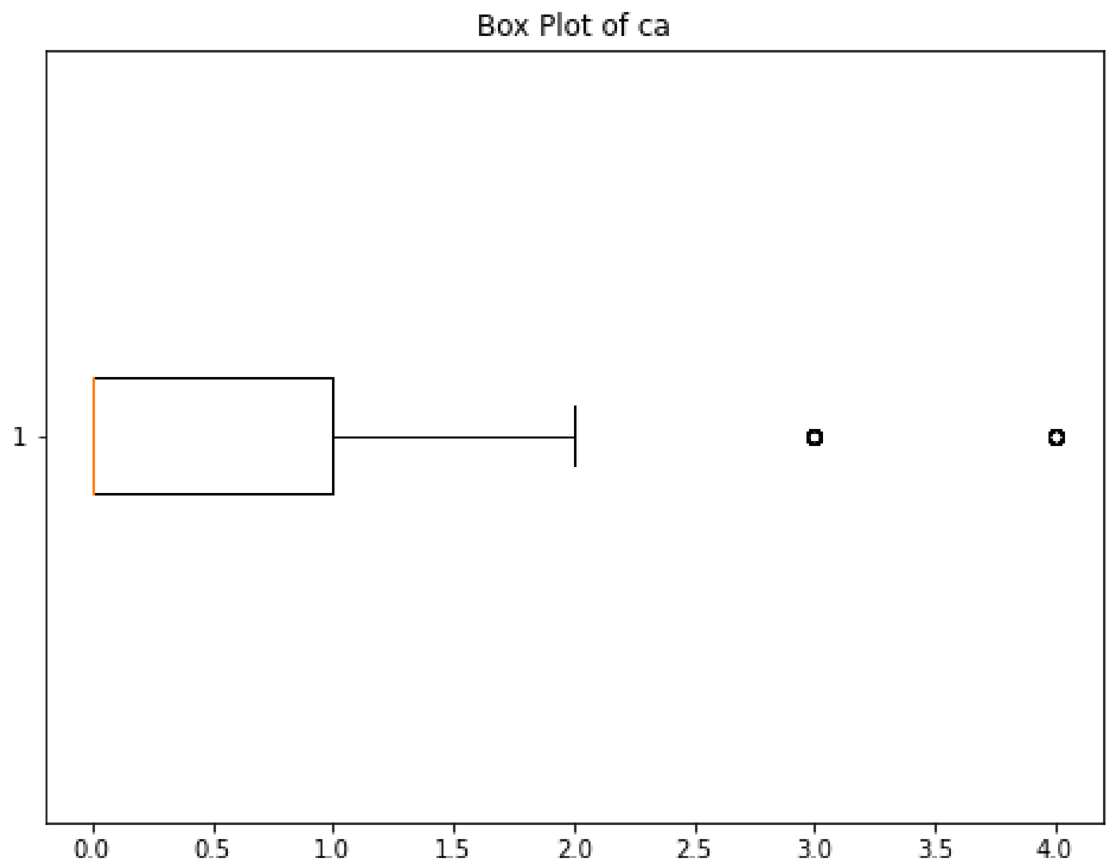

```
Outliers in 'exang':
```



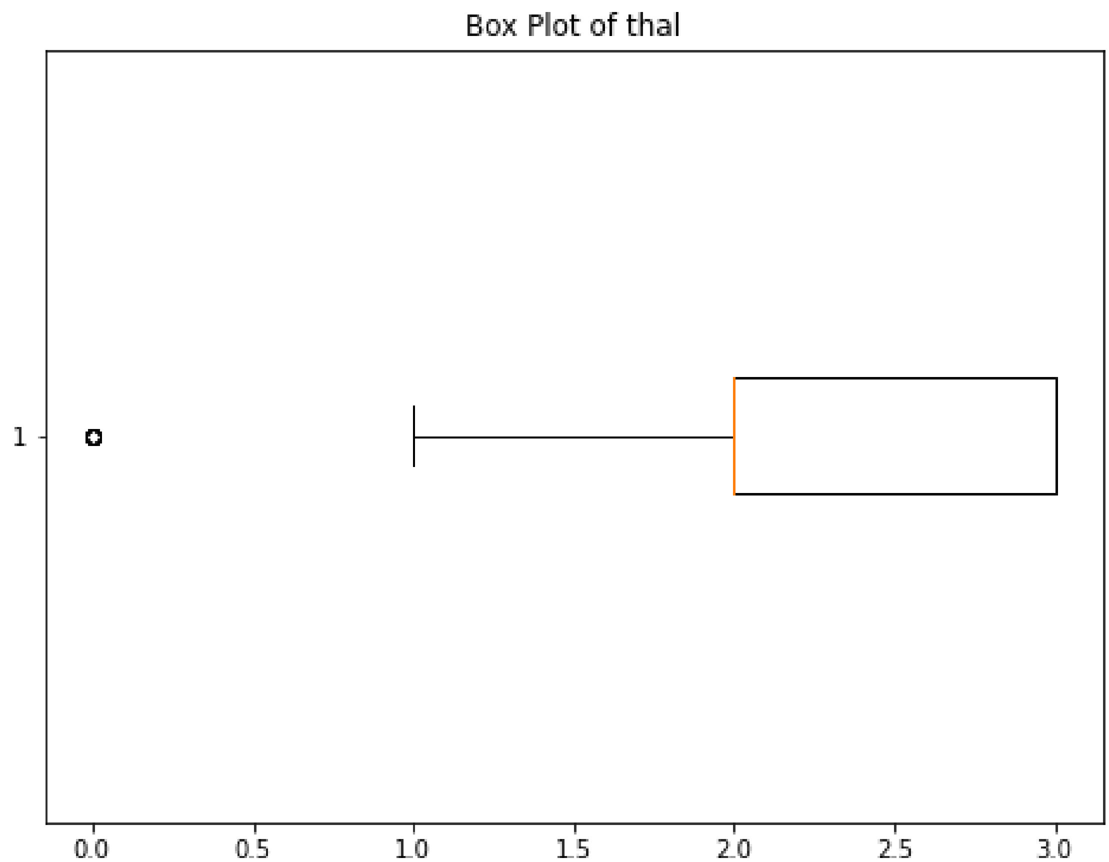

```
Outliers in 'oldpeak':
```



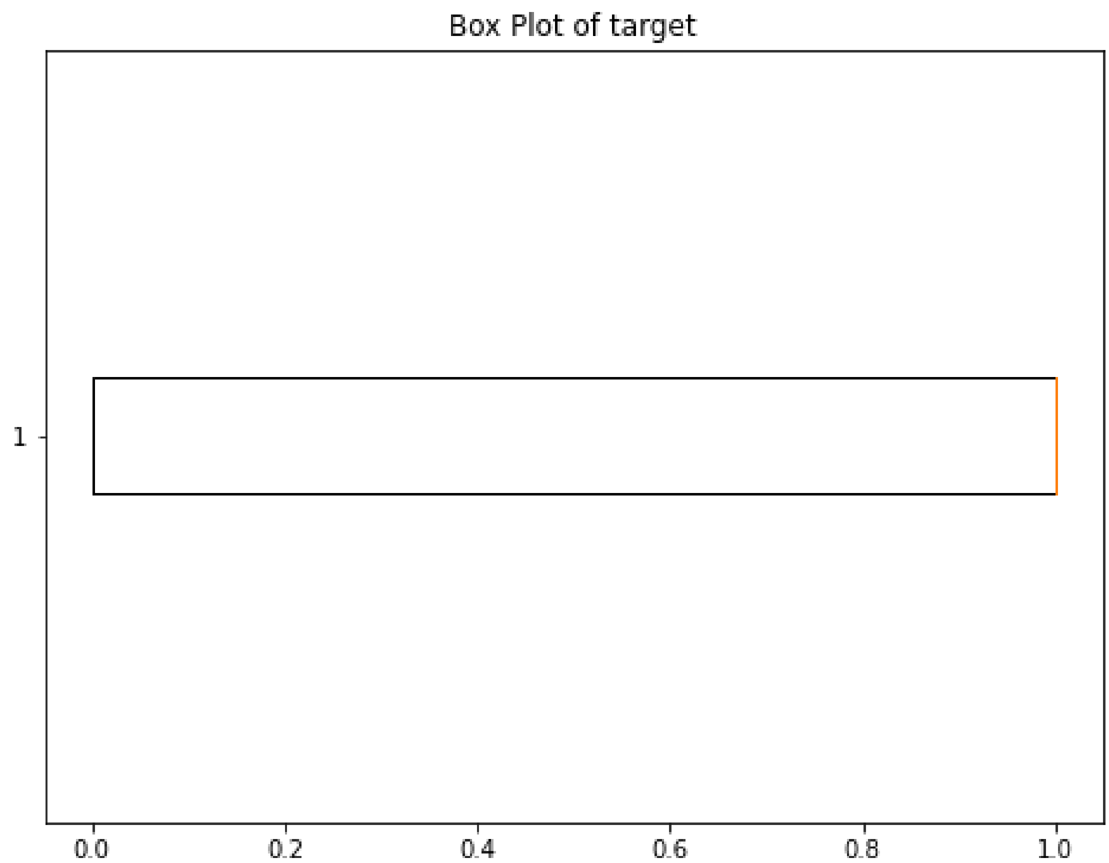

```
Outliers in 'slope':
```




```
Outliers in 'ca':
```




```
Outliers in 'thal':
```




```
Outliers in 'target':
```

0.4 Data Split

[63]:
```
x=df.drop('target',axis=1)
x.head()
```

[63]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	52	1	0	125	212	0	1	168	0	1.0	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	
4	62	0	0	138	294	1	1	106	0	1.9	1	

	ca	thal
0	2	3
1	0	3
2	0	3
3	1	3
4	3	2

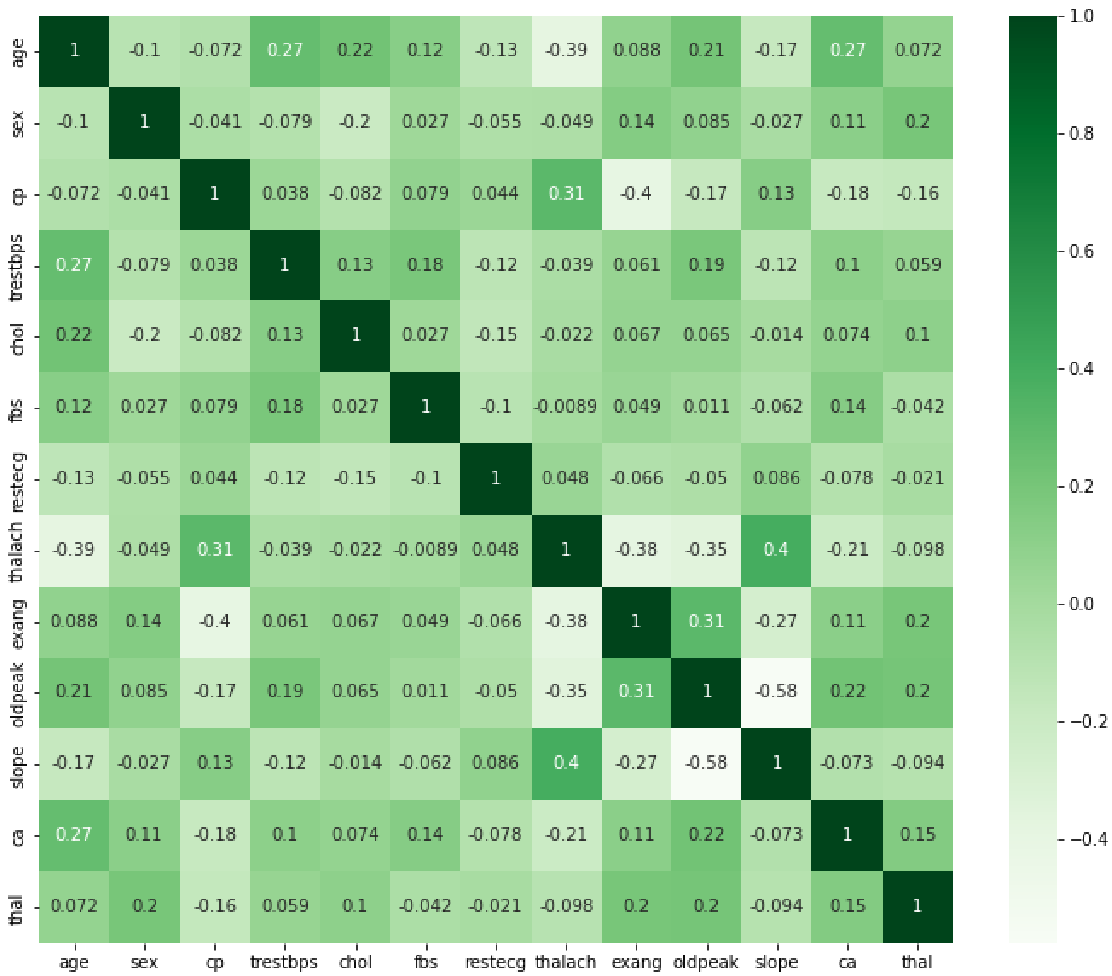
[64]:
```
y=df.target
y.head()
```

[64]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int64
```

[65]:
```
X_train, X_test, y_train, y_test =train_test_split(x,y,test_size=0.25)
```

1 Correlation Matrix

[66]:
```
plt.figure(figsize=(12,10))
cor=x.corr()
sns.heatmap(cor,annot=True,cmap=plt.cm.Greens)
plt.show()
```



2 Neural Network

[67]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Step 1: Load the dataset
# Replace this with your actual dataset loading process
# Assuming 'df' contains the dataset with the features and target 'cardio'
# For example:
# df = pd.read_csv('your_dataset.csv')

# Step 2: Data Preprocessing
# Perform data preprocessing steps here, such as encoding categorical variables␣
↪and scaling
X = df.drop(columns=['target'])
y = df['target']

# Perform feature scaling using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
↪25, random_state=42)

# Step 4: Build the Neural Network Model
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Step 5: Compile the model
optimizer = Adam(learning_rate=0.001)
model.compile(loss='binary_crossentropy', optimizer=optimizer,␣
↪metrics=['accuracy'])

# Step 6: Train the model
early_stopping = EarlyStopping(patience=5, restore_best_weights=True)
```

```python
history = model.fit(X_train, y_train, validation_split=0.2, epochs=100,
 ↪batch_size=64, callbacks=[early_stopping])

# Step 7: Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/100
10/10 [==============================] - 1s 22ms/step - loss: 0.7826 - accuracy:
0.3420 - val_loss: 0.6932 - val_accuracy: 0.5455
Epoch 2/100
10/10 [==============================] - 0s 5ms/step - loss: 0.6276 - accuracy:
0.6954 - val_loss: 0.6058 - val_accuracy: 0.7143
Epoch 3/100
10/10 [==============================] - 0s 4ms/step - loss: 0.5222 - accuracy:
0.8241 - val_loss: 0.5423 - val_accuracy: 0.7857
Epoch 4/100
10/10 [==============================] - 0s 4ms/step - loss: 0.4462 - accuracy:
0.8550 - val_loss: 0.4964 - val_accuracy: 0.8052
Epoch 5/100
10/10 [==============================] - 0s 5ms/step - loss: 0.3914 - accuracy:
0.8664 - val_loss: 0.4696 - val_accuracy: 0.8052
Epoch 6/100
10/10 [==============================] - 0s 5ms/step - loss: 0.3531 - accuracy:
0.8632 - val_loss: 0.4528 - val_accuracy: 0.8052
Epoch 7/100
10/10 [==============================] - 0s 5ms/step - loss: 0.3292 - accuracy:
0.8697 - val_loss: 0.4408 - val_accuracy: 0.8247
Epoch 8/100
10/10 [==============================] - 0s 5ms/step - loss: 0.3113 - accuracy:
0.8827 - val_loss: 0.4375 - val_accuracy: 0.8117
Epoch 9/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2973 - accuracy:
0.8876 - val_loss: 0.4351 - val_accuracy: 0.8182
Epoch 10/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2857 - accuracy:
0.8941 - val_loss: 0.4278 - val_accuracy: 0.8117
Epoch 11/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2760 - accuracy:
0.8958 - val_loss: 0.4224 - val_accuracy: 0.8117
Epoch 12/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2670 - accuracy:
0.8990 - val_loss: 0.4180 - val_accuracy: 0.8117
Epoch 13/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2591 - accuracy:
0.9007 - val_loss: 0.4166 - val_accuracy: 0.8247
```

```
Epoch 14/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2528 - accuracy:
0.9055 - val_loss: 0.4116 - val_accuracy: 0.8247
Epoch 15/100
10/10 [==============================] - 0s 4ms/step - loss: 0.2458 - accuracy:
0.9055 - val_loss: 0.4101 - val_accuracy: 0.8312
Epoch 16/100
10/10 [==============================] - 0s 6ms/step - loss: 0.2395 - accuracy:
0.9072 - val_loss: 0.4038 - val_accuracy: 0.8312
Epoch 17/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2348 - accuracy:
0.9153 - val_loss: 0.4007 - val_accuracy: 0.8377
Epoch 18/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2285 - accuracy:
0.9169 - val_loss: 0.4007 - val_accuracy: 0.8377
Epoch 19/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2226 - accuracy:
0.9202 - val_loss: 0.3966 - val_accuracy: 0.8377
Epoch 20/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2179 - accuracy:
0.9218 - val_loss: 0.3944 - val_accuracy: 0.8312
Epoch 21/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2113 - accuracy:
0.9202 - val_loss: 0.3912 - val_accuracy: 0.8377
Epoch 22/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2059 - accuracy:
0.9218 - val_loss: 0.3902 - val_accuracy: 0.8377
Epoch 23/100
10/10 [==============================] - 0s 5ms/step - loss: 0.2007 - accuracy:
0.9202 - val_loss: 0.3869 - val_accuracy: 0.8377
Epoch 24/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1961 - accuracy:
0.9202 - val_loss: 0.3827 - val_accuracy: 0.8442
Epoch 25/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1910 - accuracy:
0.9251 - val_loss: 0.3808 - val_accuracy: 0.8442
Epoch 26/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1857 - accuracy:
0.9235 - val_loss: 0.3779 - val_accuracy: 0.8442
Epoch 27/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1809 - accuracy:
0.9251 - val_loss: 0.3762 - val_accuracy: 0.8442
Epoch 28/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1765 - accuracy:
0.9251 - val_loss: 0.3761 - val_accuracy: 0.8442
Epoch 29/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1726 - accuracy:
0.9365 - val_loss: 0.3743 - val_accuracy: 0.8442
```

```
Epoch 30/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1674 - accuracy:
0.9365 - val_loss: 0.3720 - val_accuracy: 0.8442
Epoch 31/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1629 - accuracy:
0.9365 - val_loss: 0.3697 - val_accuracy: 0.8571
Epoch 32/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1583 - accuracy:
0.9446 - val_loss: 0.3684 - val_accuracy: 0.8571
Epoch 33/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1539 - accuracy:
0.9463 - val_loss: 0.3652 - val_accuracy: 0.8571
Epoch 34/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1491 - accuracy:
0.9430 - val_loss: 0.3629 - val_accuracy: 0.8571
Epoch 35/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1448 - accuracy:
0.9463 - val_loss: 0.3537 - val_accuracy: 0.8571
Epoch 36/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1401 - accuracy:
0.9479 - val_loss: 0.3525 - val_accuracy: 0.8571
Epoch 37/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1353 - accuracy:
0.9528 - val_loss: 0.3519 - val_accuracy: 0.8571
Epoch 38/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1324 - accuracy:
0.9593 - val_loss: 0.3472 - val_accuracy: 0.8701
Epoch 39/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1274 - accuracy:
0.9658 - val_loss: 0.3473 - val_accuracy: 0.8701
Epoch 40/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1246 - accuracy:
0.9577 - val_loss: 0.3480 - val_accuracy: 0.8636
Epoch 41/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1203 - accuracy:
0.9642 - val_loss: 0.3377 - val_accuracy: 0.8701
Epoch 42/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1158 - accuracy:
0.9707 - val_loss: 0.3365 - val_accuracy: 0.8896
Epoch 43/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1129 - accuracy:
0.9739 - val_loss: 0.3374 - val_accuracy: 0.8896
Epoch 44/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1084 - accuracy:
0.9739 - val_loss: 0.3297 - val_accuracy: 0.8896
Epoch 45/100
10/10 [==============================] - 0s 4ms/step - loss: 0.1050 - accuracy:
0.9739 - val_loss: 0.3335 - val_accuracy: 0.8896
```

```
Epoch 46/100
10/10 [==============================] - 0s 5ms/step - loss: 0.1010 - accuracy:
0.9739 - val_loss: 0.3312 - val_accuracy: 0.8896
Epoch 47/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0974 - accuracy:
0.9788 - val_loss: 0.3270 - val_accuracy: 0.8896
Epoch 48/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0942 - accuracy:
0.9805 - val_loss: 0.3249 - val_accuracy: 0.8896
Epoch 49/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0915 - accuracy:
0.9821 - val_loss: 0.3199 - val_accuracy: 0.8961
Epoch 50/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0884 - accuracy:
0.9805 - val_loss: 0.3170 - val_accuracy: 0.8961
Epoch 51/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0865 - accuracy:
0.9870 - val_loss: 0.3258 - val_accuracy: 0.9026
Epoch 52/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0822 - accuracy:
0.9870 - val_loss: 0.3154 - val_accuracy: 0.8961
Epoch 53/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0803 - accuracy:
0.9870 - val_loss: 0.3135 - val_accuracy: 0.9026
Epoch 54/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0766 - accuracy:
0.9886 - val_loss: 0.3152 - val_accuracy: 0.9091
Epoch 55/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0746 - accuracy:
0.9886 - val_loss: 0.3098 - val_accuracy: 0.9091
Epoch 56/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0715 - accuracy:
0.9886 - val_loss: 0.3092 - val_accuracy: 0.9091
Epoch 57/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0689 - accuracy:
0.9886 - val_loss: 0.3089 - val_accuracy: 0.9091
Epoch 58/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0670 - accuracy:
0.9886 - val_loss: 0.3071 - val_accuracy: 0.9091
Epoch 59/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0647 - accuracy:
0.9886 - val_loss: 0.3065 - val_accuracy: 0.9091
Epoch 60/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0625 - accuracy:
0.9886 - val_loss: 0.2994 - val_accuracy: 0.9091
Epoch 61/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0606 - accuracy:
0.9919 - val_loss: 0.2962 - val_accuracy: 0.9091
```

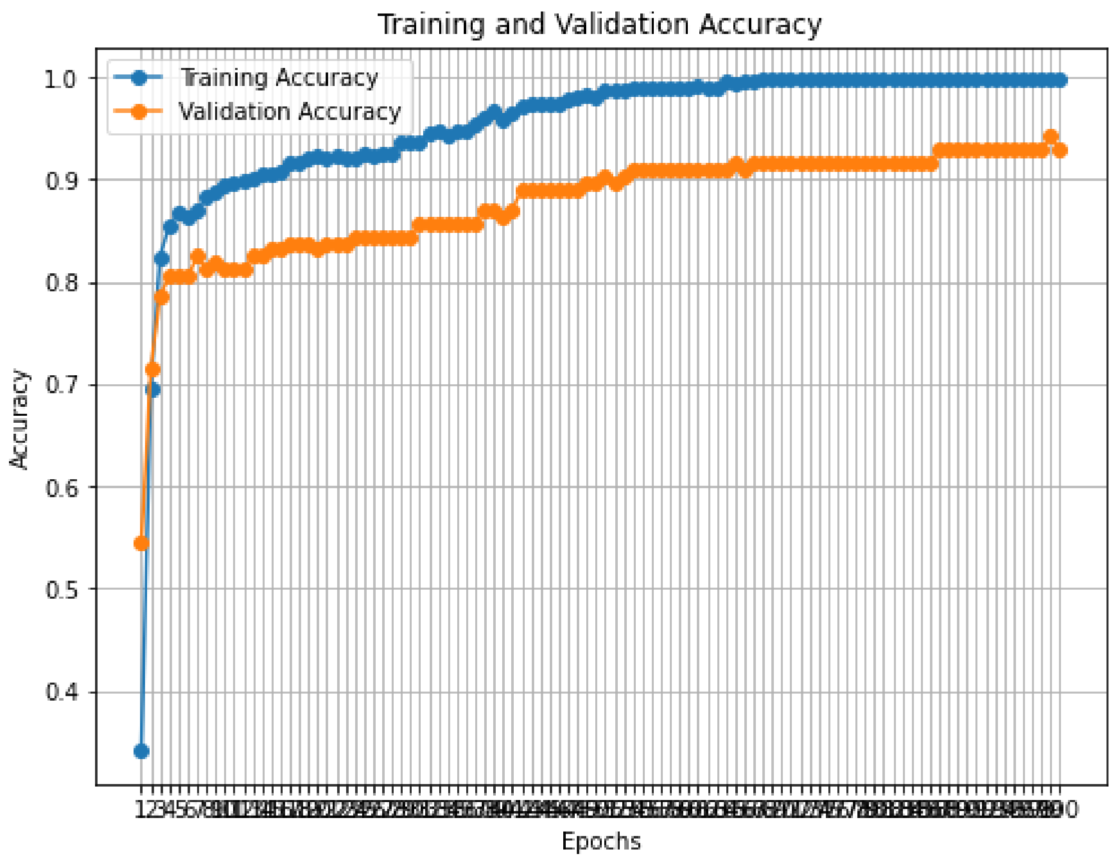
```
Epoch 62/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0583 - accuracy:
0.9886 - val_loss: 0.2963 - val_accuracy: 0.9091
Epoch 63/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0563 - accuracy:
0.9886 - val_loss: 0.2953 - val_accuracy: 0.9091
Epoch 64/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0548 - accuracy:
0.9951 - val_loss: 0.2939 - val_accuracy: 0.9091
Epoch 65/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0534 - accuracy:
0.9935 - val_loss: 0.2940 - val_accuracy: 0.9156
Epoch 66/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0515 - accuracy:
0.9951 - val_loss: 0.2921 - val_accuracy: 0.9091
Epoch 67/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0501 - accuracy:
0.9951 - val_loss: 0.2909 - val_accuracy: 0.9156
Epoch 68/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0480 - accuracy:
0.9967 - val_loss: 0.2866 - val_accuracy: 0.9156
Epoch 69/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0468 - accuracy:
0.9967 - val_loss: 0.2897 - val_accuracy: 0.9156
Epoch 70/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0453 - accuracy:
0.9967 - val_loss: 0.2849 - val_accuracy: 0.9156
Epoch 71/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0448 - accuracy:
0.9967 - val_loss: 0.2881 - val_accuracy: 0.9156
Epoch 72/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0430 - accuracy:
0.9967 - val_loss: 0.2883 - val_accuracy: 0.9156
Epoch 73/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0416 - accuracy:
0.9967 - val_loss: 0.2846 - val_accuracy: 0.9156
Epoch 74/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0401 - accuracy:
0.9967 - val_loss: 0.2781 - val_accuracy: 0.9156
Epoch 75/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0388 - accuracy:
0.9967 - val_loss: 0.2807 - val_accuracy: 0.9156
Epoch 76/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0376 - accuracy:
0.9967 - val_loss: 0.2771 - val_accuracy: 0.9156
Epoch 77/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0367 - accuracy:
0.9967 - val_loss: 0.2819 - val_accuracy: 0.9156
```

```
Epoch 78/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0354 - accuracy:
0.9967 - val_loss: 0.2790 - val_accuracy: 0.9156
Epoch 79/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0345 - accuracy:
0.9967 - val_loss: 0.2821 - val_accuracy: 0.9156
Epoch 80/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0337 - accuracy:
0.9967 - val_loss: 0.2763 - val_accuracy: 0.9156
Epoch 81/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0326 - accuracy:
0.9967 - val_loss: 0.2788 - val_accuracy: 0.9156
Epoch 82/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0316 - accuracy:
0.9967 - val_loss: 0.2778 - val_accuracy: 0.9156
Epoch 83/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0308 - accuracy:
0.9967 - val_loss: 0.2804 - val_accuracy: 0.9156
Epoch 84/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0300 - accuracy:
0.9967 - val_loss: 0.2755 - val_accuracy: 0.9156
Epoch 85/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0295 - accuracy:
0.9967 - val_loss: 0.2776 - val_accuracy: 0.9156
Epoch 86/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0283 - accuracy:
0.9967 - val_loss: 0.2745 - val_accuracy: 0.9156
Epoch 87/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0276 - accuracy:
0.9967 - val_loss: 0.2726 - val_accuracy: 0.9286
Epoch 88/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0269 - accuracy:
0.9967 - val_loss: 0.2706 - val_accuracy: 0.9286
Epoch 89/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0263 - accuracy:
0.9967 - val_loss: 0.2728 - val_accuracy: 0.9286
Epoch 90/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0255 - accuracy:
0.9967 - val_loss: 0.2736 - val_accuracy: 0.9286
Epoch 91/100
10/10 [==============================] - 0s 4ms/step - loss: 0.0248 - accuracy:
0.9967 - val_loss: 0.2707 - val_accuracy: 0.9286
Epoch 92/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0241 - accuracy:
0.9967 - val_loss: 0.2731 - val_accuracy: 0.9286
Epoch 93/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0236 - accuracy:
0.9967 - val_loss: 0.2702 - val_accuracy: 0.9286
```

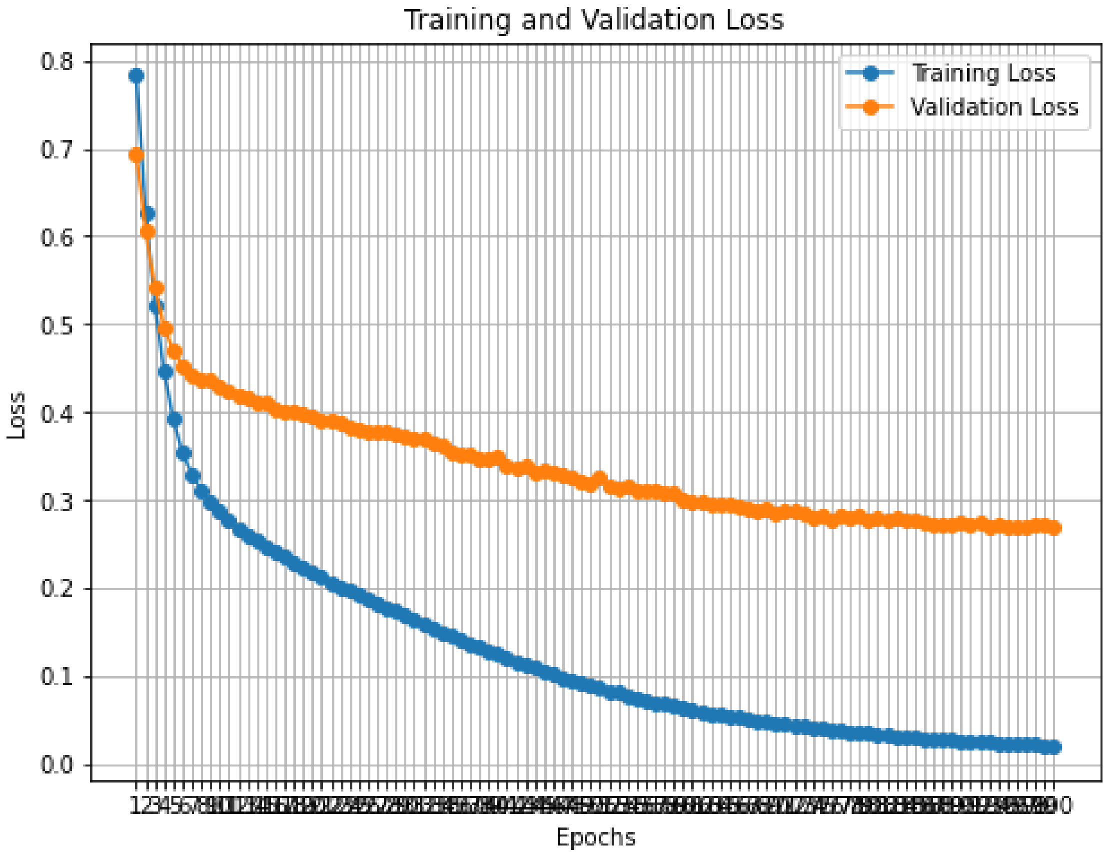
```
Epoch 94/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0227 - accuracy:
0.9967 - val_loss: 0.2712 - val_accuracy: 0.9286
Epoch 95/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0222 - accuracy:
0.9967 - val_loss: 0.2701 - val_accuracy: 0.9286
Epoch 96/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0217 - accuracy:
0.9967 - val_loss: 0.2693 - val_accuracy: 0.9286
Epoch 97/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0212 - accuracy:
0.9967 - val_loss: 0.2699 - val_accuracy: 0.9286
Epoch 98/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0207 - accuracy:
0.9967 - val_loss: 0.2719 - val_accuracy: 0.9286
Epoch 99/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0202 - accuracy:
0.9967 - val_loss: 0.2719 - val_accuracy: 0.9416
Epoch 100/100
10/10 [==============================] - 0s 5ms/step - loss: 0.0194 - accuracy:
0.9967 - val_loss: 0.2678 - val_accuracy: 0.9286
9/9 [==============================] - 0s 2ms/step - loss: 0.2137 - accuracy:
0.9572
Test Loss: 0.21372650563716888
Test Accuracy: 0.957198441028595
```

[68]:

```python
import matplotlib.pyplot as plt

# Access the training accuracy and validation accuracy from the history object
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

# Create a list with the number of epochs
epochs = range(1, len(train_accuracy) + 1)

# Plot the training accuracy and validation accuracy over epochs
plt.figure(figsize=(8, 6))
plt.plot(epochs, train_accuracy, label='Training Accuracy', marker='o')
plt.plot(epochs, val_accuracy, label='Validation Accuracy', marker='o')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.xticks(epochs)
plt.show()
```



[69]:
```python
import matplotlib.pyplot as plt

# Access the training loss and validation loss from the history object
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Create a list with the number of epochs
epochs = range(1, len(train_loss) + 1)

# Plot the training loss and validation loss over epochs
plt.figure(figsize=(8, 6))
plt.plot(epochs, train_loss, label='Training Loss', marker='o')
plt.plot(epochs, val_loss, label='Validation Loss', marker='o')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.xticks(epochs)
plt.show()
```



3 Logistic Regression model

[70]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Separate features (independent variables) and target variable (dependent␣
 ↪variable)
X = df.drop(columns=['target'])
y = df['target']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the Logistic Regression model
```

```python
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Predict the target values on the test data
y_pred = model.predict(X_test)

# Calculate accuracy on the test data
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Generate the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Generate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion)
```

```
Accuracy: 0.7804878048780488
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.69      0.76       102
           1       0.74      0.87      0.80       103

    accuracy                           0.78       205
   macro avg       0.79      0.78      0.78       205
weighted avg       0.79      0.78      0.78       205

Confusion Matrix:
[[70 32]
 [13 90]]
```

```
C:\Users\gayathriboddu\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

4 GradientBoostingClassifier

[71]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix


X = df.drop(columns=['target'])
y = df['target']
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)
model = GradientBoostingClassifier(random_state=42)
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)


# Get the best hyperparameters
best_model = grid_search.best_estimator_
best_model.fit(X_train, y_train)
train_accuracy = best_model.score(X_train, y_train)
test_accuracy = best_model.score(X_test, y_test)

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
y_pred = best_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Train Accuracy: 1.0
Test Accuracy: 0.9766536964980544
Confusion Matrix:
[[132   0]
 [  6 119]]
```

5 SVM

[73]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix


# Replace 'target' with the actual column name for the target variable in your␣
 ↪dataset
target_column_name = 'target'

# Separate features (independent variables) and target variable (dependent␣
 ↪variable)
X = df.drop(columns=[target_column_name])
y = df[target_column_name]

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the SVM classifier with a linear kernel
svm_classifier = SVC(kernel='linear')

# Train the classifier on the training data
svm_classifier.fit(X_train, y_train)

# Predict the target values on the test data
y_pred = svm_classifier.predict(X_test)

# Calculate accuracy on the test data
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Generate the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Generate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion)
```

```
Accuracy: 0.8048780487804879
Classification Report:
```

```
              precision    recall  f1-score   support

           0       0.88      0.71      0.78       102
           1       0.76      0.90      0.82       103

    accuracy                           0.80       205
   macro avg       0.82      0.80      0.80       205
weighted avg       0.82      0.80      0.80       205

Confusion Matrix:
[[72 30]
 [10 93]]
```

6 k-NN classifier

[74]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix


# Separate features (independent variables) and target variable (dependent␣
 ↪variable)
X = df.drop(columns=['target'])
y = df['target']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize the k-NN classifier with k=3 (you can choose any value for k)
knn = KNeighborsClassifier(n_neighbors=3)

# Train the classifier on the training data
knn.fit(X_train, y_train)

# Predict the target values on the test data
y_pred = knn.predict(X_test)

# Calculate accuracy on the test data
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Generate the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
```

```
print(report)

# Generate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion)
```

```
Accuracy: 0.9024390243902439
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       102
           1       0.90      0.91      0.90       103

    accuracy                           0.90       205
   macro avg       0.90      0.90      0.90       205
weighted avg       0.90      0.90      0.90       205

Confusion Matrix:
[[91 11]
 [ 9 94]]
```

7 Naive Bayes classifier

[75]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix


# Separate features (independent variables) and target variable (dependent␣
↪variable)
X = df.drop(columns=['target'])
y = df['target']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize the Naive Bayes classifier (Gaussian Naive Bayes)
naive_bayes_classifier = GaussianNB()

# Train the classifier on the training data
naive_bayes_classifier.fit(X_train, y_train)
```

```python
# Predict the target values on the test data
y_pred = naive_bayes_classifier.predict(X_test)

# Calculate accuracy on the test data
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Generate the classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)

# Generate the confusion matrix
confusion = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(confusion)
```

```
Accuracy: 0.8
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.71      0.78       102
           1       0.75      0.89      0.82       103

    accuracy                           0.80       205
   macro avg       0.81      0.80      0.80       205
weighted avg       0.81      0.80      0.80       205

Confusion Matrix:
[[72 30]
 [11 92]]
```

8 Train the Gradient Boosting Classifier with Regularization

[39]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Step 1: Load the dataset
# Replace this with your actual dataset loading process
# Assuming 'df' contains the dataset with the features and target 'cardio'
# For example:
# df = pd.read_csv('your_dataset.csv')
```

```
# Step 2: Data Preprocessing (if needed)
# If needed, perform data preprocessing steps here, such as encoding␣
↪categorical variables, scaling, etc.

# Step 3: Train-Test Split
X = df.drop(columns=['target'])
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)

# Step 4: Train the Gradient Boosting Classifier with Regularization
# Apply L1 (Lasso) regularization
model_l1 = XGBClassifier(booster='gbtree', reg_alpha=0.1, random_state=42)
model_l1.fit(X_train, y_train)

# Apply L2 (Ridge) regularization
model_l2 = XGBClassifier(booster='gbtree', reg_lambda=0.1, random_state=42)
model_l2.fit(X_train, y_train)

# Step 5: Evaluate the models
y_pred_l1 = model_l1.predict(X_test)
y_pred_l2 = model_l2.predict(X_test)

accuracy_l1 = accuracy_score(y_test, y_pred_l1)
accuracy_l2 = accuracy_score(y_test, y_pred_l2)

print("Accuracy with L1 Regularization:", accuracy_l1)
print("Accuracy with L2 Regularization:", accuracy_l2)

# Step 6: Calculate and compare the confusion matrices
conf_matrix_l1 = confusion_matrix(y_test, y_pred_l1)
conf_matrix_l2 = confusion_matrix(y_test, y_pred_l2)

print("Confusion Matrix with L1 Regularization:")
print(conf_matrix_l1)

print("Confusion Matrix with L2 Regularization:")
print(conf_matrix_l2)
```

```
Accuracy with L1 Regularization: 0.9883268482490273
Accuracy with L2 Regularization: 0.9883268482490273
Confusion Matrix with L1 Regularization:
[[132   0]
 [  3 122]]
Confusion Matrix with L2 Regularization:
[[132   0]
```

```
[  3 122]]
```

9 Random Forest classifier

[40]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report

# Load the dataset and perform train-test split (replace with your data loading␣
↪process)
# Assuming 'df' contains the dataset with the features and target 'target'
# For example:
# df = pd.read_csv('your_dataset.csv')

X = df.drop(columns=['target'])
y = df['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Perform GridSearchCV to find the best hyperparameters
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(rf_classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_rf_model = grid_search.best_estimator_

# Fit the best model to the training data
best_rf_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = best_rf_model.predict(X_test)

# Calculate accuracy
```

```
test_accuracy = accuracy_score(y_test, y_pred)

# Print the results
print("Best Hyperparameters:", grid_search.best_params_)
print("Test Accuracy:", test_accuracy)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 50}
Test Accuracy: 0.9883268482490273
Confusion Matrix:
[[132   0]
 [  3 122]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       132
           1       1.00      0.98      0.99       125

    accuracy                           0.99       257
   macro avg       0.99      0.99      0.99       257
weighted avg       0.99      0.99      0.99       257
```

10 Decision Tree Classifier

[41]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report

# Load the dataset and perform train-test split (replace with your data loading␣
↪process)
# Assuming 'df' contains the dataset with the features and target 'target'
# For example:
# df = pd.read_csv('your_dataset.csv')
```

```python
X = df.drop(columns=['target'])
y = df['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Create a Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Perform GridSearchCV to find the best hyperparameters
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(dt_classifier, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_dt_model = grid_search.best_estimator_

# Fit the best model to the training data
best_dt_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = best_dt_model.predict(X_test)

# Calculate accuracy
test_accuracy = accuracy_score(y_test, y_pred)

# Print the results
print("Best Hyperparameters:", grid_search.best_params_)
print("Test Accuracy:", test_accuracy)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': None,
```

```
'min_samples_leaf': 1, 'min_samples_split': 2}
Test Accuracy: 0.9766536964980544
Confusion Matrix:
[[132   0]
 [  6 119]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       132
           1       1.00      0.95      0.98       125

    accuracy                           0.98       257
   macro avg       0.98      0.98      0.98       257
weighted avg       0.98      0.98      0.98       257
```

11 Decision Tree classifier with Gini impurity and Entropy

[42]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Load the dataset and perform train-test split (replace with your data loading␣
 ↪process)
# Assuming 'df' contains the dataset with the features and target 'target'
# For example:
# df = pd.read_csv('your_dataset.csv')

X = df.drop(columns=['target'])
y = df['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)

# Create a Decision Tree classifier with Gini impurity
dt_gini_classifier = DecisionTreeClassifier(criterion='gini', random_state=42)

# Perform GridSearchCV to find the best hyperparameters for Gini impurity
param_grid = {
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
grid_search_gini = GridSearchCV(dt_gini_classifier, param_grid, cv=5)
grid_search_gini.fit(X_train, y_train)

# Get the best hyperparameters for Gini impurity
best_dt_gini_model = grid_search_gini.best_estimator_

# Fit the best model to the training data for Gini impurity
best_dt_gini_model.fit(X_train, y_train)

# Make predictions on the test data for Gini impurity
y_pred_gini = best_dt_gini_model.predict(X_test)

# Calculate accuracy for Gini impurity
test_accuracy_gini = accuracy_score(y_test, y_pred_gini)

# Confusion Matrix for Gini impurity
conf_matrix_gini = confusion_matrix(y_test, y_pred_gini)

# Create a Decision Tree classifier with entropy
dt_entropy_classifier = DecisionTreeClassifier(criterion='entropy',␣
↪random_state=42)

# Perform GridSearchCV to find the best hyperparameters for entropy
grid_search_entropy = GridSearchCV(dt_entropy_classifier, param_grid, cv=5)
grid_search_entropy.fit(X_train, y_train)

# Get the best hyperparameters for entropy
best_dt_entropy_model = grid_search_entropy.best_estimator_

# Fit the best model to the training data for entropy
best_dt_entropy_model.fit(X_train, y_train)

# Make predictions on the test data for entropy
y_pred_entropy = best_dt_entropy_model.predict(X_test)

# Calculate accuracy for entropy
test_accuracy_entropy = accuracy_score(y_test, y_pred_entropy)

# Confusion Matrix for entropy
conf_matrix_entropy = confusion_matrix(y_test, y_pred_entropy)

# Print the results for Gini impurity
print("Gini Impurity - Best Hyperparameters:", grid_search_gini.best_params_)
print("Gini Impurity - Test Accuracy:", test_accuracy_gini)
print("Gini Impurity - Confusion Matrix:")
print(conf_matrix_gini)
```

```
# Print the results for entropy
print("Entropy - Best Hyperparameters:", grid_search_entropy.best_params_)
print("Entropy - Test Accuracy:", test_accuracy_entropy)
print("Entropy - Confusion Matrix:")
print(conf_matrix_entropy)
```

```
Gini Impurity - Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2}
Gini Impurity - Test Accuracy: 0.9766536964980544
Gini Impurity - Confusion Matrix:
[[132   0]
 [  6 119]]
Entropy - Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1,
'min_samples_split': 2}
Entropy - Test Accuracy: 0.9883268482490273
Entropy - Confusion Matrix:
[[132   0]
 [  3 122]]
```

[43]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
c:\users\gayathriboddu\anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: scipy>=1.5.0 in
c:\users\gayathriboddu\anaconda3\lib\site-packages (from imbalanced-learn)
(1.7.3)
Requirement already satisfied: numpy>=1.17.3 in
c:\users\gayathriboddu\anaconda3\lib\site-packages (from imbalanced-learn)
(1.21.5)
Requirement already satisfied: scikit-learn>=1.0.2 in
c:\users\gayathriboddu\anaconda3\lib\site-packages (from imbalanced-learn)
(1.0.2)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\gayathriboddu\anaconda3\lib\site-packages (from imbalanced-learn)
(1.3.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\gayathriboddu\anaconda3\lib\site-packages (from imbalanced-learn)
(2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

12 Data Augmentation

[44]:
```
from sklearn.model_selection import cross_val_score
from imblearn.over_sampling import SMOTE

# Data Augmentation with SMOTE
```

```
smote = SMOTE(random_state=42)
X_train_augmented, y_train_augmented = smote.fit_resample(X_train, y_train)

# GridSearchCV with augmented data
grid_search.fit(X_train_augmented, y_train_augmented)

# Get the best hyperparameters
best_model = grid_search.best_estimator_
best_model.fit(X_train_augmented, y_train_augmented)

# Calculate train accuracy and test accuracy
train_accuracy = best_model.score(X_train, y_train)
test_accuracy = best_model.score(X_test, y_test)

# Cross-validation for validation accuracy
val_accuracy = cross_val_score(best_model, X_train_augmented,
    y_train_augmented, cv=5).mean()

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
print("Validation Accuracy:", val_accuracy)

y_pred = best_model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Train Accuracy: 1.0
Test Accuracy: 0.9766536964980544
Validation Accuracy: 0.9750543478260869
Confusion Matrix:
[[132   0]
 [  6 119]]
```

[45]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report


# Load and explore the dataset:
# Replace 'path_to_csv' with the actual path to your downloaded CSV file.
data = df
# Print some information about the dataset.
```

```python
print(data.head()) # Show the first few rows
print(data.info()) # Summary of the dataset

# Data Preprocessing:

# Drop rows with missing values
data.dropna(inplace=True)

# Split the data into features (X) and target (y)
X = data.drop('target', axis=1) # Assuming 'target' is the target column
y = data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Scale the features to have zero mean and unit variance
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a Gradient Boosting Classifier model:
# Create the Gradient Boosting Classifier model
model = GradientBoostingClassifier(n_estimators=100, random_state=42)

# Train the model
model.fit(X_train_scaled, y_train)

# Make Predictions:
# Predict on the test set
y_pred = model.predict(X_test_scaled)

# Evaluate the Model:

# Calculate accuracy and other metrics
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", classification_rep)
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
```

```
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None
Accuracy: 0.9317073170731708
Confusion Matrix:
 [[93  9]
 [ 5 98]]
Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.91      0.93       102
           1       0.92      0.95      0.93       103

    accuracy                           0.93       205
   macro avg       0.93      0.93      0.93       205
weighted avg       0.93      0.93      0.93       205
```

[ ]:

```
[ ]:
```