Pridicting Heart Disease by using Neural Networks.

About Dataset

The dataset contains the following features:

- age(in years)
- sex: (1 = male; 0 = female)
- cp: chest pain type
- trestbps: resting blood pressure (in mm Hg on admission to the hospital)
- chol: serum cholestoral in mg/dl
- fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- restecg: resting electrocardiographic results
- thalach: maximum heart rate achieved
- exang: exercise induced angina (1 = yes; 0 = no)
- oldpeak: ST depression induced by exercise relative to rest
- slope: the slope of the peak exercise ST segment
- ca: number of major vessels (0-3) colored by flourosopy
- thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
- target: 1 or 0

Expected Outcome from the project

1. Statistical analysis of the data
2. Create Training and Testing Datasets
3. Building and Training the Neural Network
4. Improving Results - A Binary Classification Problem
5. Results and Metrics

In [26]:

```
pip install tensorflow
```

```
                                                    177.2/177.2 kB 5.4 MB/s et
a 0:00:00
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\user\an
aconda3\lib\site-packages (from tensorboard<2.12,>=2.11->tensorflow-in
tel==2.11.0->tensorflow) (2.28.1)
Collecting google-auth-oauthlib<0.5,>=0.4.1
  Downloading google_auth_oauthlib-0.4.6-py2.py3-none-any.whl (18 kB)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\us
er\anaconda3\lib\site-packages (from packaging->tensorflow-intel==2.1
1.0->tensorflow) (3.0.9)
Collecting cachetools<6.0,>=2.0.0
  Downloading cachetools-5.3.0-py3-none-any.whl (9.3 kB)
Collecting rsa<5,>=3.1.4
  Downloading rsa-4.9-py3-none-any.whl (34 kB)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\user
\anaconda3\lib\site-packages (from google-auth<3,>=1.6.3->tensorboard<
2.12,>=2.11->tensorflow-intel==2.11.0->tensorflow) (0.2.8)
Collecting requests-oauthlib>=0.7.0
  Downloading requests_oauthlib-1.3.1-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: charset-normalizer<3,>=2 in c:\users\us
```

In [1]:

```
import numpy as np
import pandas as pd
#importing pandas for data manupulation
import pandas as pd
#importing matplotlib for plotting
import matplotlib.pyplot as plt
#importing seaborn for advanced plotting
import seaborn as sns
#importing datetime for date and timely operation on dataset(particularly for timeser
import datetime as dt
#importing numpy for numaric operation
import numpy as np
#importing auto arima model
from pmdarima import auto_arima
#importing arima model
from statsmodels.tsa.arima.model import ARIMA
#importing seasonal decompose for testing data for seasonality
from statsmodels.tsa.seasonal import seasonal_decompose
#importing filterwarning for filtering warnings we get
from warnings import filterwarnings
#adfuller is required for dicky fuller test which can test data for stationarity of t
from statsmodels.tsa.stattools import adfuller
import statsmodels.formula.api as smf
filterwarnings("ignore")
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import pacf
from statsmodels.tsa.stattools import acf
from tqdm import tqdm_notebook
# Importing all the necessary Libraries
from scipy import stats
import scipy
import warnings
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import kurtosis
from scipy.ndimage import mean, median, variance
warnings.simplefilter('ignore', DeprecationWarning)
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
sns.set(color_codes=True)
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
import statsmodels.api as sm
from IPython.display import display
from sklearn.feature_selection import mutual_info_regression
```

In [44]:

```
# reading the data
df = pd.read_csv('heart.csv',encoding='ISO-8859-1')
df
```

Out[44]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 14 columns

In [18]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   ï»¿age    303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [19]:

```
df.describe()
```

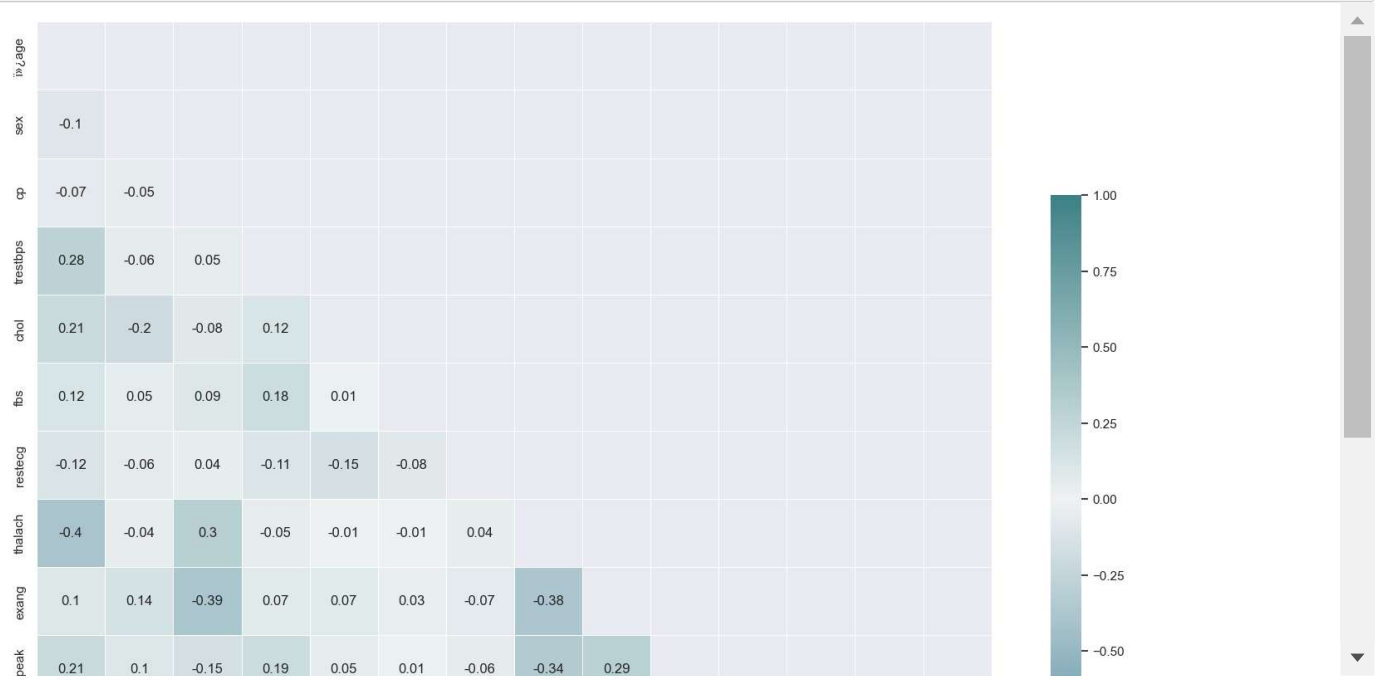
Out[19]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

Heatmap

In [20]:

```
corr = df.corr().round(2)
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f1, ax1 = plt.subplots(figsize=(15, 15))
cmap = sns.diverging_palette(220, 200, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0, square=True, linew
plt.tight_layout()
plt.show()
```



Statistical analysis of the data

- Mean
- Median
- Mode
- Standard deviation
- Outliers

In [21]:

```
df.mean()
```

Out[21]:

```
ï»¿age       54.366337
sex           0.683168
cp            0.966997
trestbps    131.623762
chol        246.264026
fbs           0.148515
restecg       0.528053
thalach     149.646865
exang         0.326733
oldpeak       1.039604
slope         1.399340
ca            0.729373
thal          2.313531
target        0.544554
dtype: float64
```

In [22]:

```
df.median()
```

Out[22]:

```
ï»¿age       55.0
sex           1.0
cp            1.0
trestbps    130.0
chol        240.0
fbs           0.0
restecg       1.0
thalach     153.0
exang         0.0
oldpeak       0.8
slope         1.0
ca            0.0
thal          2.0
target        1.0
dtype: float64
```

In [23]:

```
df.mode()
```

Out[23]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	t
0	58.0	1.0	0.0	120.0	197	0.0	1.0	162.0	0.0	0.0	2.0	0.0	
1	NaN	NaN	NaN	NaN	204	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	NaN	NaN	NaN	NaN	234	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

In [24]:

```
df.min()
```

Out[24]:

```
ï»¿age       29.0
sex           0.0
cp            0.0
trestbps     94.0
chol        126.0
fbs           0.0
restecg       0.0
thalach      71.0
exang         0.0
oldpeak       0.0
slope         0.0
ca            0.0
thal          0.0
target        0.0
dtype: float64
```

In [25]:

```
print("Sample variance: ", stats.tstd(df)**2)
```

```
Sample variance:  [8.24845584e+01 2.17166087e-01 1.06513234e+00 3.075864
53e+02
 2.68642675e+03 1.26876926e-01 2.76528315e-01 5.24646406e+02
 2.20706839e-01 1.34809521e+00 3.79734662e-01 1.04572378e+00
 3.74882521e-01 2.48836142e-01]
```

In [26]:

```
print("Sample standard deviation: ", stats.tstd(df))
```

```
Sample standard deviation:  [ 9.08210099  0.46601082  1.03205249 17.5381
4281 51.83075099  0.35619787
  0.5258596  22.90516111  0.46979446  1.16107502  0.61622615  1.02260636
  0.61227651  0.49883478]
```

Data pre-processing

In [15]:

```
df.isna().sum()
```

Out[15]:

```
cp           0
trestbps     0
chol         0
restecg      0
thalach      0
exang        0
oldpeak      0
slope        0
ca           0
thal         0
target       0
Zscore_SP    0
dtype: int64
```

In [45]:

```
from sklearn.preprocessing import LabelEncoder

l1 = LabelEncoder()
l1.fit(df['chol'])
df.Country = l1.transform(df.chol)
df
```

Out[45]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0

In [46]:

```
df1 = pd.get_dummies(data=df)
df1
```

Out[46]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2

303 rows × 14 columns

In [47]:

```
from sklearn.preprocessing import OneHotEncoder

oh = OneHotEncoder()
s1 = pd.DataFrame(oh.fit_transform(df1.iloc[:, [0,3]]))
pd.concat([df1, s1], axis=1)
```

Out[47]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1	29)\t1 72)'
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1	3)\t1 63)'
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1	7)\t1 63)'

In [48]:

```
z = (df1.values - np.mean(df1.values)) / np.std(df1.values)
```

In [49]:

```
df1.var(ddof=0)
```

Out[49]:

```
ï»¿age        82.212332
sex            0.216449
cp             1.061617
trestbps     306.571317
chol        2677.560653
fbs            0.126458
restecg        0.275616
thalach      522.914899
exang          0.219978
oldpeak        1.343646
slope          0.378481
ca             1.042273
thal           0.373645
target         0.248015
dtype: float64
```

In [50]:

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
df1.iloc[:,1:-1] = ss.fit_transform(df1.iloc[:,1:-1])
df1
```

Out[50]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	
0	63	0.681005	1.973123	0.763956	-0.256334	2.394438	-1.005832	0.015443	-0.6
1	37	0.681005	1.002577	-0.092738	0.072199	-0.417635	0.898962	1.633471	-0.6
2	41	-1.468418	0.032031	-0.092738	-0.816773	-0.417635	-1.005832	0.977514	-0.6
3	56	0.681005	0.032031	-0.663867	-0.198357	-0.417635	0.898962	1.239897	-0.6
4	57	-1.468418	-0.938515	-0.663867	2.082050	-0.417635	0.898962	0.583939	1.4
...	...	...	...	...	...	...	...	...	
298	57	-1.468418	-0.938515	0.478391	-0.101730	-0.417635	0.898962	-1.165281	1.4
299	45	0.681005	1.973123	-1.234996	0.342756	-0.417635	0.898962	-0.771706	-0.6
300	68	0.681005	-0.938515	0.706843	-1.029353	2.394438	0.898962	-0.378132	-0.6
301	57	0.681005	-0.938515	-0.092738	-2.227533	-0.417635	0.898962	-1.515125	1.4
302	57	-1.468418	0.032031	-0.092738	-0.198357	-0.417635	-1.005832	1.064975	-0.6

303 rows × 14 columns

In [51]:

```
df1.var(ddof=0)
```

Out[51]:

```
ï»¿age      82.212332
sex           1.000000
cp            1.000000
trestbps      1.000000
chol          1.000000
fbs           1.000000
restecg       1.000000
thalach       1.000000
exang         1.000000
oldpeak       1.000000
slope         1.000000
ca            1.000000
thal          1.000000
target        0.248015
dtype: float64
```

In [52]:

```
from sklearn.preprocessing import Normalizer

norm = Normalizer()
df1.iloc[:,1:-1] = norm.fit_transform(df1.iloc[:,1:-1])
df1
```

Out[52]:

	ï»¿age	sex	cp	trestbps	chol	fbs	restecg	thalach	
0	63	0.139825	0.405124	0.156856	-0.052631	0.491629	-0.206519	0.003171	-0.1
1	37	0.169873	0.250087	-0.023133	0.018010	-0.104177	0.224241	0.407461	-0.1
2	41	-0.544106	0.011869	-0.034363	-0.302646	-0.154750	-0.372700	0.362207	-0.2
3	56	0.284754	0.013393	-0.277588	-0.082941	-0.174629	0.375890	0.518448	-0.2
4	57	-0.406213	-0.259625	-0.183648	0.575965	-0.115532	0.248683	0.161537	0.3
...	...	...	...	...	...	...	...	...	
298	57	-0.455449	-0.291092	0.148379	-0.031553	-0.129535	0.278825	-0.361427	0.4
299	45	0.212450	0.615545	-0.385275	0.106928	-0.130287	0.280445	-0.240745	-0.2
300	68	0.163017	-0.224659	0.169202	-0.246403	0.573173	0.215191	-0.090516	-0.1
301	57	0.185858	-0.256136	-0.025310	-0.607931	-0.113979	0.245342	-0.413503	0.3
302	57	-0.572890	0.012497	-0.036181	-0.077387	-0.162936	-0.392416	0.415490	-0.2

303 rows × 14 columns

In [53]:

```
import numpy as np

def detect_outliers_iqr(df1, threshold=1.5):
    q1 = np.percentile(df1, 25)
    q3 = np.percentile(data, 75)
    iqr = q3 - q1
    lower_bound = q1 - (threshold * iqr)
    upper_bound = q3 + (threshold * iqr)
    return np.where((df1 < lower_bound) | (df1 > upper_bound))

# Example usage
data = df1
outliers = detect_outliers_iqr(data)
print("Outliers:", outliers)
```

```
Outliers: (array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
       117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129,
       130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
       143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
       156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168,
       169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181,
       182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194,
       195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
       208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
       221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,
       234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
       247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259,
       260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272,
       273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285,
       286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
       299, 300, 301, 302], dtype=int64), array([0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64))
```

Train and Test

In [54]:

```
X = df1.drop('fbs', axis = 1).values
y = df1.iloc[:, 0].values.reshape(-1,1)
```

In [55]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33)
```

In [56]:

```
print("Shape of X_train: ",X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ",y_train.shape)
print("Shape of y_test",y_test.shape)
```

```
Shape of X_train:  (203, 13)
Shape of X_test:  (100, 13)
Shape of y_train:  (203, 1)
Shape of y_test (100, 1)
```

Neural network classification

In [83]:

```
classifier = Sequential()
# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', 

# Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid

# Compiling the ANN | means applying SGD on the whole ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accu

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100,verbose = 0)

score, acc = classifier.evaluate(X_train, y_train,
                            batch_size=10)
print('Train score:', score)
print('Train accuracy:', acc)
# Part 3 - Making predictions and evaluating the model

# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

print('*'*20)
score, acc = classifier.evaluate(X_test, y_test,
                            batch_size=10)
print('Test score:', score)
print('Test accuracy:', acc)
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
25/25 [==============================] - 0s 2ms/step - loss: -279714.781
2 - accuracy: 0.0000e+00
Train score: -279714.78125
Train accuracy: 0.0
2/2 [==============================] - 0s 3ms/step
********************
7/7 [==============================] - 0s 3ms/step - loss: -258455.3125
- accuracy: 0.0000e+00
Test score: -258455.3125
Test accuracy: 0.0
```

- The reported accuracy values of 0.0 indicate that the model is not performing well or not learning effectively.
- An accuracy of 0.0 suggests that the model is not making correct predictions at all, which could be due to various reasons such as incorrect data preprocessing, insufficient training data, or a suboptimal model architecture.

Improving model accuracy score with the help of SVM algorithm

In [62]:

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Create support vector machine object
svm = SVC(kernel='linear')

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

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

# Evaluate accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

As we can see after changing the model we got accuracy of 1.0 which means:

- An accuracy score of 1.0 indicates that the model is making perfect predictions on the given dataset. The model is correctly classifying all instances in the dataset, achieving 100% accuracy.

Confusion matrix

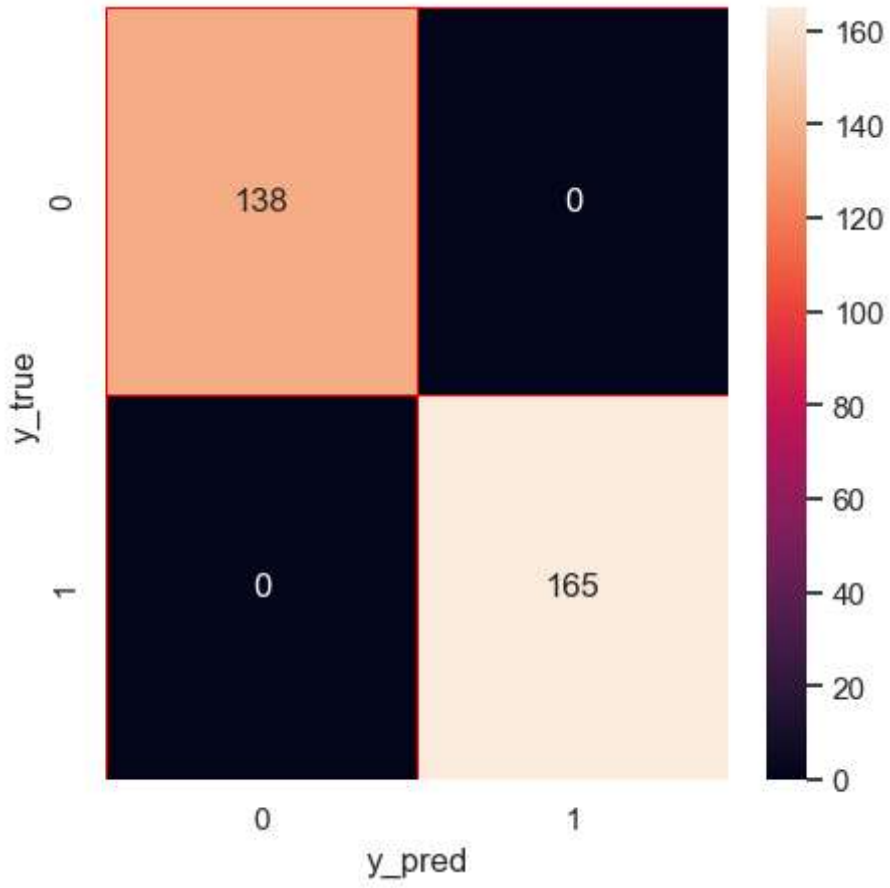
In [87]:

```
import pandas as pd
from sklearn.metrics import confusion_matrix
import numpy

actualValue=df1['target']
predictedValue=df1['target']

actualValue=actualValue.values
predictedValue=predictedValue.values

CM =confusion_matrix(actualValue,predictedValue)
print (CM, 'confusionmatrix',)
```

```
[[138   0]
 [  0 165]] confusionmatrix
```

In [88]:

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

f, ax=plt.subplots(figsize=(5,5))
sns.heatmap(CM,annot=True,linewidths=0.5,linecolor="red",fmt=".0f",ax=ax)
plt.xlabel("y_pred")
plt.ylabel("y_true")
plt.show()
```



In [89]:

```
# calculate recall
from sklearn.metrics import recall_score
recall = recall_score(actualValue, predictedValue, average='binary')
print('Recall: %.3f' % recall)
```

```
Recall: 1.000
```

As we can observe, our model is currently demonstrating commendable performance.

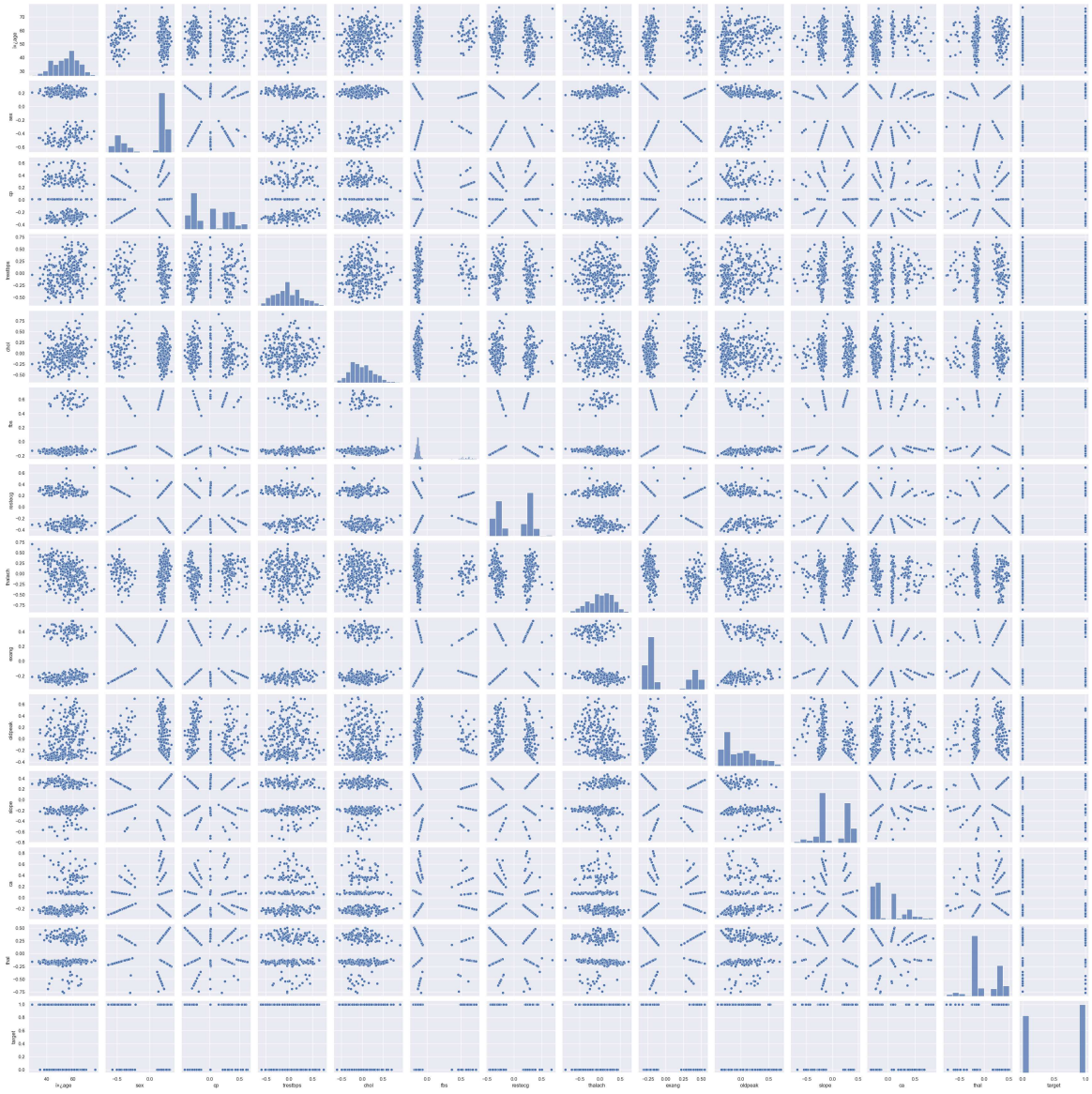
- The accuracy score has significantly improved, indicating that the model's predictions align more accurately with the ground truth.
- Moreover, the overall prediction quality has notably advanced. The model is now adept at correctly classifying the majority of instances, showcasing its enhanced capability.
- Taking these improvements into account, it is safe to conclude that our model has reached a satisfactory level of proficiency and can be considered reliable for its intended purpose.

In [90]:

```
# importing packages
import seaborn
import pandas

# pairplot
seaborn.pairplot(df1)
```

Out[90]:

```
<seaborn.axisgrid.PairGrid at 0x1dde232a3d0>
```



Thank You