se-prediction-with-neural-networks

April 1, 2024

1 Heart Disease Prediction using Neural Networks

[3]:
```
import sys
import pandas as pd
import numpy as np
import sklearn
import matplotlib
import keras
```

```
Using TensorFlow backend.
```

[4]:
```
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
```

1.0.1 1. Importing the Dataset

The dataset is available through the University of California, Irvine Machine learning repository. Here is the URL:

http:////archive.ics.uci.edu/ml/datasets/Heart+Disease

This dataset contains patient data concerning heart disease diagnosis that was collected at several locations around the world. There are 76 attributes, including age, sex, resting blood pressure, cholestoral levels, echocardiogram data, exercise habits, and many others. To data, all published studies using this data focus on a subset of 14 attributes - so we will do the same. More specifically, we will use the data collected at the Cleveland Clinic Foundation.

To import the necessary data, we will use pandas' built in read_csv() function. Let's get started!

[5]:
```
# import the heart disease dataset
url = "http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/
↪processed.cleveland.data"

# the names will be the names of each column in our pandas DataFrame
names = ['age',
        'sex',
        'cp',
        'trestbps',
        'chol',
```

```
        'fbs',
        'restecg',
        'thalach',
        'exang',
        'oldpeak',
        'slope',
        'ca',
        'thal',
        'class']

# read the csv
cleveland = pd.read_csv(url, names=names)
```

[10]:
```
# print the shape of the DataFrame, so we can see how many examples we have
print ('format(cleveland.shape')
print (cleveland.loc[1])
```

```
format(cleveland.shape
age          67
sex           1
cp            4
trestbps    160
chol        286
fbs           0
restecg       2
thalach     108
exang         1
oldpeak     1.5
slope         2
ca          3.0
thal        3.0
class         2
Name: 1, dtype: object
```

[12]:
```
# print the last twenty or so data points
cleveland.loc[280:]
```

[12]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
280	57.0	1.0	4.0	110.0	335.0	0.0	0.0	143.0	1.0	3.0	
281	47.0	1.0	3.0	130.0	253.0	0.0	0.0	179.0	0.0	0.0	
282	55.0	0.0	4.0	128.0	205.0	0.0	1.0	130.0	1.0	2.0	
283	35.0	1.0	2.0	122.0	192.0	0.0	0.0	174.0	0.0	0.0	
284	61.0	1.0	4.0	148.0	203.0	0.0	0.0	161.0	0.0	0.0	
285	58.0	1.0	4.0	114.0	318.0	0.0	1.0	140.0	0.0	4.4	
286	58.0	0.0	4.0	170.0	225.0	1.0	2.0	146.0	1.0	2.8	
287	58.0	1.0	2.0	125.0	220.0	0.0	0.0	144.0	0.0	0.4	
288	56.0	1.0	2.0	130.0	221.0	0.0	2.0	163.0	0.0	0.0	

```
289  56.0  1.0  2.0     120.0  240.0  0.0       0.0     169.0    0.0      0.0
290  67.0  1.0  3.0     152.0  212.0  0.0       2.0     150.0    0.0      0.8
291  55.0  0.0  2.0     132.0  342.0  0.0       0.0     166.0    0.0      1.2
292  44.0  1.0  4.0     120.0  169.0  0.0       0.0     144.0    1.0      2.8
293  63.0  1.0  4.0     140.0  187.0  0.0       2.0     144.0    1.0      4.0
294  63.0  0.0  4.0     124.0  197.0  0.0       0.0     136.0    1.0      0.0
295  41.0  1.0  2.0     120.0  157.0  0.0       0.0     182.0    0.0      0.0
296  59.0  1.0  4.0     164.0  176.0  1.0       2.0      90.0    0.0      1.0
297  57.0  0.0  4.0     140.0  241.0  0.0       0.0     123.0    1.0      0.2
298  45.0  1.0  1.0     110.0  264.0  0.0       0.0     132.0    0.0      1.2
299  68.0  1.0  4.0     144.0  193.0  1.0       0.0     141.0    0.0      3.4
300  57.0  1.0  4.0     130.0  131.0  0.0       0.0     115.0    1.0      1.2
301  57.0  0.0  2.0     130.0  236.0  0.0       2.0     174.0    0.0      0.0
302  38.0  1.0  3.0     138.0  175.0  0.0       0.0     173.0    0.0      0.0

     slope   ca thal  class
280    2.0  1.0  7.0      2
281    1.0  0.0  3.0      0
282    2.0  1.0  7.0      3
283    1.0  0.0  3.0      0
284    1.0  1.0  7.0      2
285    3.0  3.0  6.0      4
286    2.0  2.0  6.0      2
287    2.0    ?  7.0      0
288    1.0  0.0  7.0      0
289    3.0  0.0  3.0      0
290    2.0  0.0  7.0      1
291    1.0  0.0  3.0      0
292    3.0  0.0  6.0      2
293    1.0  2.0  7.0      2
294    2.0  0.0  3.0      1
295    1.0  0.0  3.0      0
296    2.0  2.0  6.0      3
297    2.0  0.0  7.0      1
298    2.0  0.0  7.0      1
299    2.0  2.0  7.0      2
300    2.0  1.0  7.0      3
301    2.0  1.0  3.0      1
302    1.0    ?  3.0      0
```

[13]:
```
# remove missing data (indicated with a "?")
data = cleveland[~cleveland.isin(['?'])]
data.loc[280:]
```

[13]:
```
      age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
280  57.0  1.0  4.0     110.0  335.0  0.0      0.0    143.0    1.0      3.0
281  47.0  1.0  3.0     130.0  253.0  0.0      0.0    179.0    0.0      0.0
```

282	55.0	0.0	4.0	128.0	205.0	0.0	1.0	130.0	1.0	2.0
283	35.0	1.0	2.0	122.0	192.0	0.0	0.0	174.0	0.0	0.0
284	61.0	1.0	4.0	148.0	203.0	0.0	0.0	161.0	0.0	0.0
285	58.0	1.0	4.0	114.0	318.0	0.0	1.0	140.0	0.0	4.4
286	58.0	0.0	4.0	170.0	225.0	1.0	2.0	146.0	1.0	2.8
287	58.0	1.0	2.0	125.0	220.0	0.0	0.0	144.0	0.0	0.4
288	56.0	1.0	2.0	130.0	221.0	0.0	2.0	163.0	0.0	0.0
289	56.0	1.0	2.0	120.0	240.0	0.0	0.0	169.0	0.0	0.0
290	67.0	1.0	3.0	152.0	212.0	0.0	2.0	150.0	0.0	0.8
291	55.0	0.0	2.0	132.0	342.0	0.0	0.0	166.0	0.0	1.2
292	44.0	1.0	4.0	120.0	169.0	0.0	0.0	144.0	1.0	2.8
293	63.0	1.0	4.0	140.0	187.0	0.0	2.0	144.0	1.0	4.0
294	63.0	0.0	4.0	124.0	197.0	0.0	0.0	136.0	1.0	0.0
295	41.0	1.0	2.0	120.0	157.0	0.0	0.0	182.0	0.0	0.0
296	59.0	1.0	4.0	164.0	176.0	1.0	2.0	90.0	0.0	1.0
297	57.0	0.0	4.0	140.0	241.0	0.0	0.0	123.0	1.0	0.2
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0

	slope	ca	thal	class
280	2.0	1.0	7.0	2
281	1.0	0.0	3.0	0
282	2.0	1.0	7.0	3
283	1.0	0.0	3.0	0
284	1.0	1.0	7.0	2
285	3.0	3.0	6.0	4
286	2.0	2.0	6.0	2
287	2.0	NaN	7.0	0
288	1.0	0.0	7.0	0
289	3.0	0.0	3.0	0
290	2.0	0.0	7.0	1
291	1.0	0.0	3.0	0
292	3.0	0.0	6.0	2
293	1.0	2.0	7.0	2
294	2.0	0.0	3.0	1
295	1.0	0.0	3.0	0
296	2.0	2.0	6.0	3
297	2.0	0.0	7.0	1
298	2.0	0.0	7.0	1
299	2.0	2.0	7.0	2
300	2.0	1.0	7.0	3
301	2.0	1.0	3.0	1
302	1.0	NaN	3.0	0

[14]:
```
# drop rows with NaN values from DataFrame
data = data.dropna(axis=0)
data.loc[280:]
```

[14]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	\
280	57.0	1.0	4.0	110.0	335.0	0.0	0.0	143.0	1.0	3.0	
281	47.0	1.0	3.0	130.0	253.0	0.0	0.0	179.0	0.0	0.0	
282	55.0	0.0	4.0	128.0	205.0	0.0	1.0	130.0	1.0	2.0	
283	35.0	1.0	2.0	122.0	192.0	0.0	0.0	174.0	0.0	0.0	
284	61.0	1.0	4.0	148.0	203.0	0.0	0.0	161.0	0.0	0.0	
285	58.0	1.0	4.0	114.0	318.0	0.0	1.0	140.0	0.0	4.4	
286	58.0	0.0	4.0	170.0	225.0	1.0	2.0	146.0	1.0	2.8	
288	56.0	1.0	2.0	130.0	221.0	0.0	2.0	163.0	0.0	0.0	
289	56.0	1.0	2.0	120.0	240.0	0.0	0.0	169.0	0.0	0.0	
290	67.0	1.0	3.0	152.0	212.0	0.0	2.0	150.0	0.0	0.8	
291	55.0	0.0	2.0	132.0	342.0	0.0	0.0	166.0	0.0	1.2	
292	44.0	1.0	4.0	120.0	169.0	0.0	0.0	144.0	1.0	2.8	
293	63.0	1.0	4.0	140.0	187.0	0.0	2.0	144.0	1.0	4.0	
294	63.0	0.0	4.0	124.0	197.0	0.0	0.0	136.0	1.0	0.0	
295	41.0	1.0	2.0	120.0	157.0	0.0	0.0	182.0	0.0	0.0	
296	59.0	1.0	4.0	164.0	176.0	1.0	2.0	90.0	0.0	1.0	
297	57.0	0.0	4.0	140.0	241.0	0.0	0.0	123.0	1.0	0.2	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	

	slope	ca	thal	class
280	2.0	1.0	7.0	2
281	1.0	0.0	3.0	0
282	2.0	1.0	7.0	3
283	1.0	0.0	3.0	0
284	1.0	1.0	7.0	2
285	3.0	3.0	6.0	4
286	2.0	2.0	6.0	2
288	1.0	0.0	7.0	0
289	3.0	0.0	3.0	0
290	2.0	0.0	7.0	1
291	1.0	0.0	3.0	0
292	3.0	0.0	6.0	2
293	1.0	2.0	7.0	2
294	2.0	0.0	3.0	1
295	1.0	0.0	3.0	0
296	2.0	2.0	6.0	3
297	2.0	0.0	7.0	1
298	2.0	0.0	7.0	1
299	2.0	2.0	7.0	2

```
300    2.0  1.0  7.0      3
301    2.0  1.0  3.0      1
```

[16]:
```python
# print the shape and data type of the dataframe
print (data.shape)
print (data.dtypes)
```

```
(297, 14)
age         float64
sex         float64
cp          float64
trestbps    float64
chol        float64
fbs         float64
restecg     float64
thalach     float64
exang       float64
oldpeak     float64
slope       float64
ca           object
thal         object
class         int64
dtype: object
```

[17]:
```python
# transform data to numeric to enable further analysis
data = data.apply(pd.to_numeric)
data.dtypes
```

[17]:
```
age         float64
sex         float64
cp          float64
trestbps    float64
chol        float64
fbs         float64
restecg     float64
thalach     float64
exang       float64
oldpeak     float64
slope       float64
ca          float64
thal        float64
class         int64
dtype: object
```

[18]:
```python
# print data characteristics, usings pandas built-in describe() function
data.describe()
```

[18]:

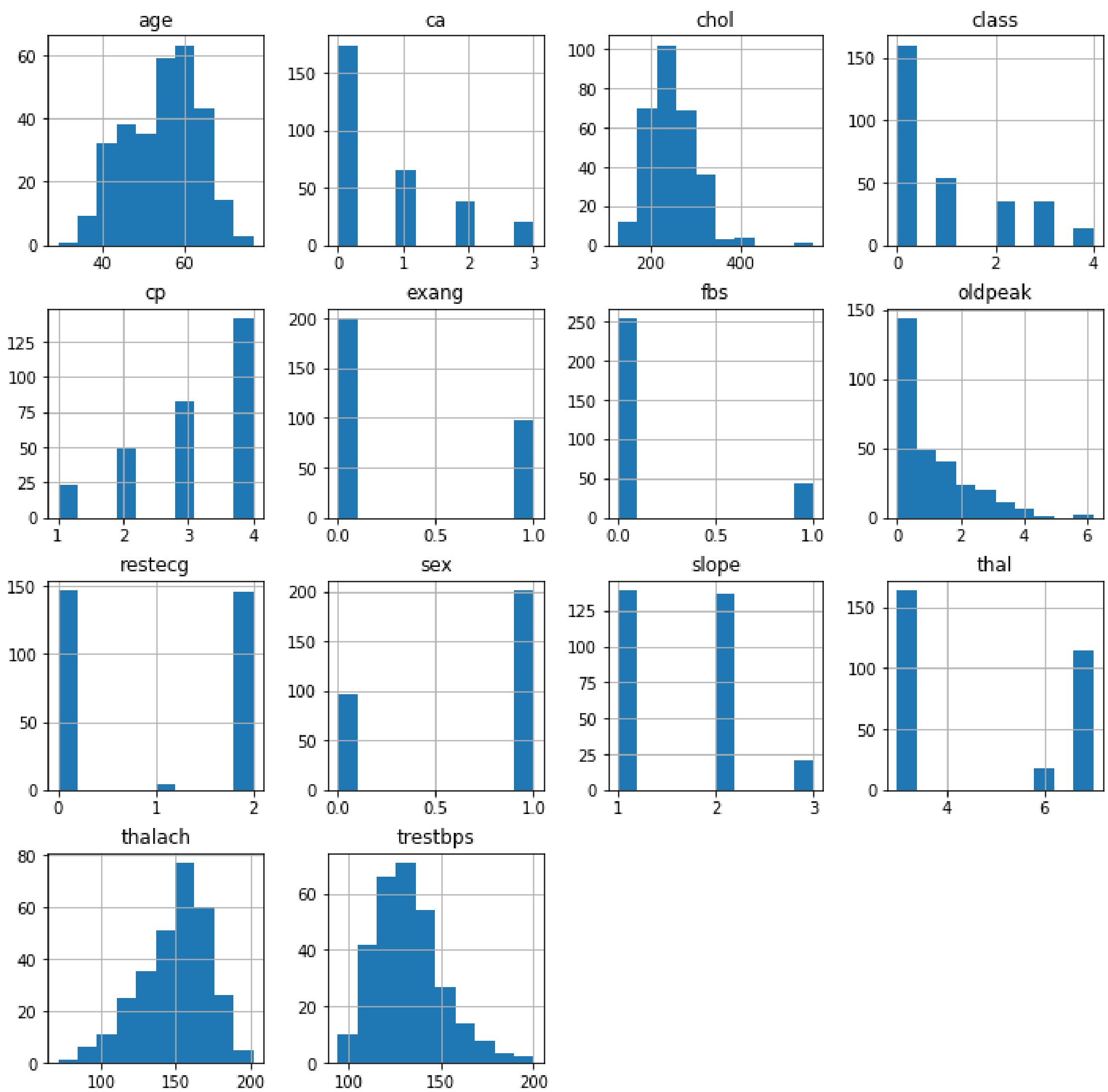
```
             age         sex          cp    trestbps        chol         fbs  \
count 297.000000  297.000000  297.000000  297.000000  297.000000  297.000000
mean   54.542088    0.676768    3.158249  131.693603  247.350168    0.144781
std     9.049736    0.468500    0.964859   17.762806   51.997583    0.352474
min    29.000000    0.000000    1.000000   94.000000  126.000000    0.000000
25%    48.000000    0.000000    3.000000  120.000000  211.000000    0.000000
50%    56.000000    1.000000    3.000000  130.000000  243.000000    0.000000
75%    61.000000    1.000000    4.000000  140.000000  276.000000    0.000000
max    77.000000    1.000000    4.000000  200.000000  564.000000    1.000000

         restecg     thalach       exang     oldpeak       slope          ca  \
count 297.000000  297.000000  297.000000  297.000000  297.000000  297.000000
mean    0.996633  149.599327    0.326599    1.055556    1.602694    0.676768
std     0.994914   22.941562    0.469761    1.166123    0.618187    0.938965
min     0.000000   71.000000    0.000000    0.000000    1.000000    0.000000
25%     0.000000  133.000000    0.000000    0.000000    1.000000    0.000000
50%     1.000000  153.000000    0.000000    0.800000    2.000000    0.000000
75%     2.000000  166.000000    1.000000    1.600000    2.000000    1.000000
max     2.000000  202.000000    1.000000    6.200000    3.000000    3.000000

            thal       class
count 297.000000  297.000000
mean    4.730640    0.946128
std     1.938629    1.234551
min     3.000000    0.000000
25%     3.000000    0.000000
50%     3.000000    0.000000
75%     7.000000    2.000000
max     7.000000    4.000000
```

[19]:

```python
# plot histograms for each variable
data.hist(figsize = (12, 12))
plt.show()
```



1.0.2 2. Create Training and Testing Datasets

Now that we have preprocessed the data appropriately, we can split it into training and testings datasets. We will use Sklearn's train_test_split() function to generate a training dataset (80 percent of the total data) and testing dataset (20 percent of the total data).

Furthermore, the class values in this dataset contain multiple types of heart disease with values ranging from 0 (healthy) to 4 (severe heart disease). Consequently, we will need to convert our class data to categorical labels. For example, the label 2 will become [0, 0, 1, 0, 0].

[20]:
```
# create X and Y datasets for training
from sklearn import model_selection

X = np.array(data.drop(['class'], 1))
```

```
y = np.array(data['class'])

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,␣
↪test_size = 0.2)
```

[22]:
```
# convert the data to categorical labels
from keras.utils.np_utils import to_categorical

Y_train = to_categorical(y_train, num_classes=None)
Y_test = to_categorical(y_test, num_classes=None)
print (Y_train.shape)
print (Y_train[:10])
```

```
(237, 5)
[[1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [1. 0. 0. 0. 0.]]
```

1.0.3 3. Building and Training the Neural Network

Now that we have our data fully processed and split into training and testing datasets, we can begin building a neural network to solve this classification problem. Using keras, we will define a simple neural network with one hidden layer. Since this is a categorical classification problem, we will use a softmax activation function in the final layer of our network and a categorical_crossentropy loss during our training phase.

[23]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam

# define a function to build the keras model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(8, input_dim=13, kernel_initializer='normal',␣
↪activation='relu'))
    model.add(Dense(4, kernel_initializer='normal', activation='relu'))
    model.add(Dense(5, activation='softmax'))

    # compile model
    adam = Adam(lr=0.001)
```

```
    model.compile(loss='categorical_crossentropy', optimizer=adam,␣
 ↪metrics=['accuracy'])
    return model

model = create_model()

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 8)                 112
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 25
=================================================================
Total params: 173
Trainable params: 173
Non-trainable params: 0
_________________________________________________________________
None
```

[24]:

```
# fit the model to the training data
model.fit(X_train, Y_train, epochs=100, batch_size=10, verbose = 1)
```

```
Epoch 1/100
237/237 [==============================] - 1s 3ms/step - loss: 1.4000 -
accuracy: 0.5232
Epoch 2/100
237/237 [==============================] - 0s 405us/step - loss: 1.3282 -
accuracy: 0.5232
Epoch 3/100
237/237 [==============================] - 0s 169us/step - loss: 1.3082 -
accuracy: 0.5232
Epoch 4/100
237/237 [==============================] - 0s 152us/step - loss: 1.2900 -
accuracy: 0.5232
Epoch 5/100
237/237 [==============================] - 0s 152us/step - loss: 1.2700 -
accuracy: 0.5232
Epoch 6/100
237/237 [==============================] - 0s 152us/step - loss: 1.2721 -
accuracy: 0.5232
Epoch 7/100
237/237 [==============================] - 0s 152us/step - loss: 1.2489 -
```

```
accuracy: 0.5232
Epoch 8/100
237/237 [==============================] - 0s 152us/step - loss: 1.2339 -
accuracy: 0.5274
Epoch 9/100
237/237 [==============================] - 0s 169us/step - loss: 1.2489 -
accuracy: 0.5359
Epoch 10/100
237/237 [==============================] - 0s 152us/step - loss: 1.2240 -
accuracy: 0.5190
Epoch 11/100
237/237 [==============================] - 0s 152us/step - loss: 1.1979 -
accuracy: 0.5359
Epoch 12/100
237/237 [==============================] - 0s 169us/step - loss: 1.1957 -
accuracy: 0.5274
Epoch 13/100
237/237 [==============================] - 0s 169us/step - loss: 1.1792 -
accuracy: 0.5359
Epoch 14/100
237/237 [==============================] - 0s 169us/step - loss: 1.1837 -
accuracy: 0.5232
Epoch 15/100
237/237 [==============================] - 0s 152us/step - loss: 1.1838 -
accuracy: 0.5359
Epoch 16/100
237/237 [==============================] - 0s 169us/step - loss: 1.1715 -
accuracy: 0.5359
Epoch 17/100
237/237 [==============================] - 0s 169us/step - loss: 1.1561 -
accuracy: 0.5401
Epoch 18/100
237/237 [==============================] - 0s 186us/step - loss: 1.1466 -
accuracy: 0.5443
Epoch 19/100
237/237 [==============================] - 0s 152us/step - loss: 1.1423 -
accuracy: 0.5401
Epoch 20/100
237/237 [==============================] - 0s 169us/step - loss: 1.1349 -
accuracy: 0.5443
Epoch 21/100
237/237 [==============================] - 0s 152us/step - loss: 1.1336 -
accuracy: 0.5316
Epoch 22/100
237/237 [==============================] - 0s 169us/step - loss: 1.1206 -
accuracy: 0.5401
Epoch 23/100
237/237 [==============================] - 0s 169us/step - loss: 1.1170 -
```

```
accuracy: 0.5316
Epoch 24/100
237/237 [==============================] - 0s 169us/step - loss: 1.1210 -
accuracy: 0.5359
Epoch 25/100
237/237 [==============================] - 0s 169us/step - loss: 1.1030 -
accuracy: 0.5401
Epoch 26/100
237/237 [==============================] - 0s 152us/step - loss: 1.1014 -
accuracy: 0.5401
Epoch 27/100
237/237 [==============================] - 0s 219us/step - loss: 1.0900 -
accuracy: 0.5401
Epoch 28/100
237/237 [==============================] - 0s 321us/step - loss: 1.0963 -
accuracy: 0.5316
Epoch 29/100
237/237 [==============================] - 0s 186us/step - loss: 1.0860 -
accuracy: 0.5401
Epoch 30/100
237/237 [==============================] - 0s 169us/step - loss: 1.0939 -
accuracy: 0.5316
Epoch 31/100
237/237 [==============================] - 0s 169us/step - loss: 1.0751 -
accuracy: 0.5359
Epoch 32/100
237/237 [==============================] - 0s 135us/step - loss: 1.0669 -
accuracy: 0.5316
Epoch 33/100
237/237 [==============================] - 0s 152us/step - loss: 1.0679 -
accuracy: 0.5443
Epoch 34/100
237/237 [==============================] - 0s 186us/step - loss: 1.0601 -
accuracy: 0.5485
Epoch 35/100
237/237 [==============================] - 0s 152us/step - loss: 1.0569 -
accuracy: 0.5485
Epoch 36/100
237/237 [==============================] - 0s 135us/step - loss: 1.0571 -
accuracy: 0.5485
Epoch 37/100
237/237 [==============================] - 0s 169us/step - loss: 1.0550 -
accuracy: 0.5443
Epoch 38/100
237/237 [==============================] - 0s 152us/step - loss: 1.0432 -
accuracy: 0.5485
Epoch 39/100
237/237 [==============================] - 0s 169us/step - loss: 1.0330 -
```

```
accuracy: 0.5485
Epoch 40/100
237/237 [==============================] - 0s 135us/step - loss: 1.0310 -
accuracy: 0.5443
Epoch 41/100
237/237 [==============================] - 0s 169us/step - loss: 1.0538 -
accuracy: 0.5443
Epoch 42/100
237/237 [==============================] - 0s 169us/step - loss: 1.0338 -
accuracy: 0.5485
Epoch 43/100
237/237 [==============================] - 0s 186us/step - loss: 1.0212 -
accuracy: 0.5485
Epoch 44/100
237/237 [==============================] - 0s 135us/step - loss: 1.0265 -
accuracy: 0.5359
Epoch 45/100
237/237 [==============================] - 0s 169us/step - loss: 1.0155 -
accuracy: 0.5485
Epoch 46/100
237/237 [==============================] - 0s 135us/step - loss: 1.0116 -
accuracy: 0.5527
Epoch 47/100
237/237 [==============================] - 0s 135us/step - loss: 1.0118 -
accuracy: 0.5443
Epoch 48/100
237/237 [==============================] - 0s 186us/step - loss: 1.0191 -
accuracy: 0.5527
Epoch 49/100
237/237 [==============================] - 0s 186us/step - loss: 0.9996 -
accuracy: 0.5443
Epoch 50/100
237/237 [==============================] - 0s 152us/step - loss: 0.9967 -
accuracy: 0.5485
Epoch 51/100
237/237 [==============================] - 0s 169us/step - loss: 0.9950 -
accuracy: 0.5401
Epoch 52/100
237/237 [==============================] - 0s 202us/step - loss: 1.0106 -
accuracy: 0.5612
Epoch 53/100
237/237 [==============================] - 0s 219us/step - loss: 0.9987 -
accuracy: 0.5527
Epoch 54/100
237/237 [==============================] - 0s 270us/step - loss: 0.9940 -
accuracy: 0.5570
Epoch 55/100
237/237 [==============================] - 0s 169us/step - loss: 0.9855 -
```

```
accuracy: 0.5527
Epoch 56/100
237/237 [==============================] - 0s 219us/step - loss: 0.9886 -
accuracy: 0.5570
Epoch 57/100
237/237 [==============================] - 0s 186us/step - loss: 0.9833 -
accuracy: 0.5612
Epoch 58/100
237/237 [==============================] - 0s 186us/step - loss: 0.9904 -
accuracy: 0.5570
Epoch 59/100
237/237 [==============================] - 0s 186us/step - loss: 0.9899 -
accuracy: 0.5696
Epoch 60/100
237/237 [==============================] - 0s 202us/step - loss: 0.9757 -
accuracy: 0.5612
Epoch 61/100
237/237 [==============================] - 0s 202us/step - loss: 0.9732 -
accuracy: 0.5612
Epoch 62/100
237/237 [==============================] - 0s 169us/step - loss: 0.9751 -
accuracy: 0.5527
Epoch 63/100
237/237 [==============================] - 0s 186us/step - loss: 1.0040 -
accuracy: 0.5570
Epoch 64/100
237/237 [==============================] - 0s 202us/step - loss: 0.9621 -
accuracy: 0.5654
Epoch 65/100
237/237 [==============================] - 0s 202us/step - loss: 0.9667 -
accuracy: 0.5696
Epoch 66/100
237/237 [==============================] - 0s 219us/step - loss: 0.9681 -
accuracy: 0.5654
Epoch 67/100
237/237 [==============================] - 0s 203us/step - loss: 0.9604 -
accuracy: 0.5527
Epoch 68/100
237/237 [==============================] - 0s 219us/step - loss: 0.9561 -
accuracy: 0.5612
Epoch 69/100
237/237 [==============================] - 0s 202us/step - loss: 0.9502 -
accuracy: 0.5570
Epoch 70/100
237/237 [==============================] - 0s 186us/step - loss: 0.9629 -
accuracy: 0.5654
Epoch 71/100
237/237 [==============================] - 0s 219us/step - loss: 0.9528 -
```

```
accuracy: 0.5570
Epoch 72/100
237/237 [==============================] - 0s 219us/step - loss: 0.9489 -
accuracy: 0.5654
Epoch 73/100
237/237 [==============================] - 0s 186us/step - loss: 0.9746 -
accuracy: 0.6034
Epoch 74/100
237/237 [==============================] - 0s 186us/step - loss: 0.9567 -
accuracy: 0.6076
Epoch 75/100
237/237 [==============================] - 0s 304us/step - loss: 0.9429 -
accuracy: 0.6034
Epoch 76/100
237/237 [==============================] - 0s 287us/step - loss: 0.9469 -
accuracy: 0.6118
Epoch 77/100
237/237 [==============================] - 0s 202us/step - loss: 0.9622 -
accuracy: 0.6034
Epoch 78/100
237/237 [==============================] - 0s 186us/step - loss: 0.9518 -
accuracy: 0.6160
Epoch 79/100
237/237 [==============================] - 0s 253us/step - loss: 0.9448 -
accuracy: 0.6118
Epoch 80/100
237/237 [==============================] - 0s 219us/step - loss: 0.9373 -
accuracy: 0.5992
Epoch 81/100
237/237 [==============================] - 0s 186us/step - loss: 0.9364 -
accuracy: 0.6160
Epoch 82/100
237/237 [==============================] - 0s 169us/step - loss: 0.9321 -
accuracy: 0.6203
Epoch 83/100
237/237 [==============================] - 0s 186us/step - loss: 0.9633 -
accuracy: 0.5992
Epoch 84/100
237/237 [==============================] - 0s 186us/step - loss: 0.9680 -
accuracy: 0.6118
Epoch 85/100
237/237 [==============================] - 0s 169us/step - loss: 0.9340 -
accuracy: 0.6118
Epoch 86/100
237/237 [==============================] - 0s 169us/step - loss: 0.9267 -
accuracy: 0.6245
Epoch 87/100
237/237 [==============================] - 0s 202us/step - loss: 0.9378 -
```

```
accuracy: 0.5992
Epoch 88/100
237/237 [==============================] - 0s 202us/step - loss: 0.9237 -
accuracy: 0.6245
Epoch 89/100
237/237 [==============================] - 0s 186us/step - loss: 0.9309 -
accuracy: 0.6160
Epoch 90/100
237/237 [==============================] - 0s 152us/step - loss: 0.9247 -
accuracy: 0.6203
Epoch 91/100
237/237 [==============================] - 0s 186us/step - loss: 0.9206 -
accuracy: 0.6245
Epoch 92/100
237/237 [==============================] - 0s 169us/step - loss: 0.9295 -
accuracy: 0.6160
Epoch 93/100
237/237 [==============================] - 0s 169us/step - loss: 0.9206 -
accuracy: 0.6160
Epoch 94/100
237/237 [==============================] - 0s 186us/step - loss: 0.9255 -
accuracy: 0.6118
Epoch 95/100
237/237 [==============================] - 0s 186us/step - loss: 0.9193 -
accuracy: 0.6203
Epoch 96/100
237/237 [==============================] - 0s 186us/step - loss: 0.9192 -
accuracy: 0.6160
Epoch 97/100
237/237 [==============================] - 0s 287us/step - loss: 0.9233 -
accuracy: 0.6118
Epoch 98/100
237/237 [==============================] - 0s 219us/step - loss: 0.9203 -
accuracy: 0.6118
Epoch 99/100
237/237 [==============================] - 0s 169us/step - loss: 0.9433 -
accuracy: 0.6076
Epoch 100/100
237/237 [==============================] - 0s 169us/step - loss: 0.9423 -
accuracy: 0.6118
```

[24]:
```
<keras.callbacks.callbacks.History at 0x1c489de8a48>
```

1.0.4 4. Improving Results - A Binary Classification Problem

Although we achieved promising results, we still have a fairly large error. This could be because it is very difficult to distinguish between the different severity levels of heart disease (classes 1 - 4). Let's simplify the problem by converting the data to a binary classification problem - heart disease or no heart disease.

[26]:
```
# convert into binary classification problem - heart disease or no heart disease
Y_train_binary = y_train.copy()
Y_test_binary = y_test.copy()

Y_train_binary[Y_train_binary > 0] = 1
Y_test_binary[Y_test_binary > 0] = 1

print (Y_train_binary[:20])
```

```
[0 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 1 0 1]
```

[27]:
```
# define a new keras model for binary classification
def create_binary_model():
    # create model
    model = Sequential()
    model.add(Dense(8, input_dim=13, kernel_initializer='normal',␣
 ↪activation='relu'))
    model.add(Dense(4, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
    adam = Adam(lr=0.001)
    model.compile(loss='binary_crossentropy', optimizer=adam,␣
 ↪metrics=['accuracy'])
    return model

binary_model = create_binary_model()

print(binary_model.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 8)                 112
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 5
=================================================================
Total params: 153
Trainable params: 153
Non-trainable params: 0
_________________________________________________________________
None
```

[28]:
```
# fit the binary model on the training data
binary_model.fit(X_train, Y_train_binary, epochs=100, batch_size=10, verbose =␣
↪1)
```

```
Epoch 1/100
237/237 [==============================] - 1s 3ms/step - loss: 0.7204 -
accuracy: 0.5612
Epoch 2/100
237/237 [==============================] - 0s 354us/step - loss: 0.6801 -
accuracy: 0.5781
Epoch 3/100
237/237 [==============================] - 0s 388us/step - loss: 0.6782 -
accuracy: 0.6118
Epoch 4/100
237/237 [==============================] - 0s 219us/step - loss: 0.6739 -
accuracy: 0.6371
Epoch 5/100
237/237 [==============================] - 0s 219us/step - loss: 0.6405 -
accuracy: 0.6287
Epoch 6/100
237/237 [==============================] - 0s 186us/step - loss: 0.6310 -
accuracy: 0.6456
Epoch 7/100
237/237 [==============================] - 0s 202us/step - loss: 0.6023 -
accuracy: 0.7595
Epoch 8/100
237/237 [==============================] - 0s 202us/step - loss: 0.5834 -
accuracy: 0.7553
Epoch 9/100
237/237 [==============================] - 0s 219us/step - loss: 0.5740 -
accuracy: 0.7131
Epoch 10/100
237/237 [==============================] - 0s 186us/step - loss: 0.5524 -
accuracy: 0.7257
Epoch 11/100
237/237 [==============================] - 0s 236us/step - loss: 0.5516 -
accuracy: 0.7257
Epoch 12/100
237/237 [==============================] - 0s 219us/step - loss: 0.5221 -
accuracy: 0.7511
Epoch 13/100
237/237 [==============================] - 0s 202us/step - loss: 0.5140 -
accuracy: 0.7553
Epoch 14/100
237/237 [==============================] - 0s 219us/step - loss: 0.5150 -
accuracy: 0.7806
Epoch 15/100
```

```
237/237 [==============================] - 0s 202us/step - loss: 0.5034 -
accuracy: 0.7848
Epoch 16/100
237/237 [==============================] - 0s 337us/step - loss: 0.4994 -
accuracy: 0.7637
Epoch 17/100
237/237 [==============================] - 0s 202us/step - loss: 0.4903 -
accuracy: 0.7595
Epoch 18/100
237/237 [==============================] - 0s 186us/step - loss: 0.4827 -
accuracy: 0.7384
Epoch 19/100
237/237 [==============================] - 0s 219us/step - loss: 0.4748 -
accuracy: 0.7595
Epoch 20/100
237/237 [==============================] - 0s 202us/step - loss: 0.4791 -
accuracy: 0.7764
Epoch 21/100
237/237 [==============================] - 0s 202us/step - loss: 0.4681 -
accuracy: 0.7975
Epoch 22/100
237/237 [==============================] - 0s 186us/step - loss: 0.4652 -
accuracy: 0.7637
Epoch 23/100
237/237 [==============================] - 0s 186us/step - loss: 0.4503 -
accuracy: 0.7890
Epoch 24/100
237/237 [==============================] - 0s 202us/step - loss: 0.4328 -
accuracy: 0.7848
Epoch 25/100
237/237 [==============================] - 0s 186us/step - loss: 0.4461 -
accuracy: 0.7932
Epoch 26/100
237/237 [==============================] - 0s 203us/step - loss: 0.4503 -
accuracy: 0.7806
Epoch 27/100
237/237 [==============================] - 0s 169us/step - loss: 0.4312 -
accuracy: 0.8017
Epoch 28/100
237/237 [==============================] - 0s 186us/step - loss: 0.4383 -
accuracy: 0.7848
Epoch 29/100
237/237 [==============================] - 0s 203us/step - loss: 0.4235 -
accuracy: 0.8186
Epoch 30/100
237/237 [==============================] - 0s 202us/step - loss: 0.4131 -
accuracy: 0.8143
Epoch 31/100
```

```
237/237 [==============================] - 0s 186us/step - loss: 0.4177 -
accuracy: 0.7932
Epoch 32/100
237/237 [==============================] - 0s 169us/step - loss: 0.4180 -
accuracy: 0.8101
Epoch 33/100
237/237 [==============================] - 0s 169us/step - loss: 0.4242 -
accuracy: 0.8186
Epoch 34/100
237/237 [==============================] - 0s 202us/step - loss: 0.4266 -
accuracy: 0.8017
Epoch 35/100
237/237 [==============================] - 0s 203us/step - loss: 0.3976 -
accuracy: 0.8270
Epoch 36/100
237/237 [==============================] - 0s 186us/step - loss: 0.4118 -
accuracy: 0.8312
Epoch 37/100
237/237 [==============================] - 0s 202us/step - loss: 0.3977 -
accuracy: 0.8312
Epoch 38/100
237/237 [==============================] - 0s 321us/step - loss: 0.4095 -
accuracy: 0.8186
Epoch 39/100
237/237 [==============================] - 0s 202us/step - loss: 0.3903 -
accuracy: 0.8312
Epoch 40/100
237/237 [==============================] - 0s 169us/step - loss: 0.4156 -
accuracy: 0.8059
Epoch 41/100
237/237 [==============================] - 0s 202us/step - loss: 0.3944 -
accuracy: 0.8439
Epoch 42/100
237/237 [==============================] - 0s 202us/step - loss: 0.4036 -
accuracy: 0.8312
Epoch 43/100
237/237 [==============================] - 0s 203us/step - loss: 0.3966 -
accuracy: 0.8270
Epoch 44/100
237/237 [==============================] - 0s 186us/step - loss: 0.3934 -
accuracy: 0.8439
Epoch 45/100
237/237 [==============================] - 0s 186us/step - loss: 0.3929 -
accuracy: 0.8143
Epoch 46/100
237/237 [==============================] - 0s 203us/step - loss: 0.4034 -
accuracy: 0.8397
Epoch 47/100
```

```
237/237 [==============================] - 0s 186us/step - loss: 0.4102 -
accuracy: 0.8101
Epoch 48/100
237/237 [==============================] - 0s 152us/step - loss: 0.4004 -
accuracy: 0.8312
Epoch 49/100
237/237 [==============================] - 0s 219us/step - loss: 0.3813 -
accuracy: 0.8270
Epoch 50/100
237/237 [==============================] - 0s 169us/step - loss: 0.3821 -
accuracy: 0.8523
Epoch 51/100
237/237 [==============================] - 0s 236us/step - loss: 0.4201 -
accuracy: 0.8143
Epoch 52/100
237/237 [==============================] - 0s 186us/step - loss: 0.3888 -
accuracy: 0.8397
Epoch 53/100
237/237 [==============================] - 0s 202us/step - loss: 0.3729 -
accuracy: 0.8565
Epoch 54/100
237/237 [==============================] - 0s 202us/step - loss: 0.3802 -
accuracy: 0.8354
Epoch 55/100
237/237 [==============================] - 0s 219us/step - loss: 0.3775 -
accuracy: 0.8650
Epoch 56/100
237/237 [==============================] - 0s 202us/step - loss: 0.3812 -
accuracy: 0.8523
Epoch 57/100
237/237 [==============================] - 0s 202us/step - loss: 0.3903 -
accuracy: 0.8312
Epoch 58/100
237/237 [==============================] - 0s 186us/step - loss: 0.3688 -
accuracy: 0.8608
Epoch 59/100
237/237 [==============================] - 0s 169us/step - loss: 0.3839 -
accuracy: 0.8439
Epoch 60/100
237/237 [==============================] - 0s 354us/step - loss: 0.3804 -
accuracy: 0.8228
Epoch 61/100
237/237 [==============================] - 0s 253us/step - loss: 0.3858 -
accuracy: 0.8354
Epoch 62/100
237/237 [==============================] - 0s 236us/step - loss: 0.3680 -
accuracy: 0.8439
Epoch 63/100
```

```
237/237 [==============================] - 0s 202us/step - loss: 0.3675 -
accuracy: 0.8481
Epoch 64/100
237/237 [==============================] - 0s 169us/step - loss: 0.3742 -
accuracy: 0.8439
Epoch 65/100
237/237 [==============================] - 0s 186us/step - loss: 0.3875 -
accuracy: 0.8354
Epoch 66/100
237/237 [==============================] - 0s 219us/step - loss: 0.3701 -
accuracy: 0.8439
Epoch 67/100
237/237 [==============================] - 0s 203us/step - loss: 0.3670 -
accuracy: 0.8439
Epoch 68/100
237/237 [==============================] - 0s 253us/step - loss: 0.3680 -
accuracy: 0.8523
Epoch 69/100
237/237 [==============================] - ETA: 0s - loss: 0.2902 - accuracy:
0.80 - 0s 236us/step - loss: 0.3826 - accuracy: 0.8354
Epoch 70/100
237/237 [==============================] - 0s 270us/step - loss: 0.3654 -
accuracy: 0.8523
Epoch 71/100
237/237 [==============================] - 0s 270us/step - loss: 0.3763 -
accuracy: 0.8481
Epoch 72/100
237/237 [==============================] - 0s 287us/step - loss: 0.3822 -
accuracy: 0.8439
Epoch 73/100
237/237 [==============================] - 0s 270us/step - loss: 0.3658 -
accuracy: 0.8270
Epoch 74/100
237/237 [==============================] - 0s 270us/step - loss: 0.4259 -
accuracy: 0.8481
Epoch 75/100
237/237 [==============================] - 0s 253us/step - loss: 0.3776 -
accuracy: 0.8354
Epoch 76/100
237/237 [==============================] - 0s 236us/step - loss: 0.3615 -
accuracy: 0.8439
Epoch 77/100
237/237 [==============================] - 0s 219us/step - loss: 0.4168 -
accuracy: 0.8270
Epoch 78/100
237/237 [==============================] - 0s 354us/step - loss: 0.3767 -
accuracy: 0.8565
Epoch 79/100
```

```
237/237 [==============================] - 0s 321us/step - loss: 0.3584 -
accuracy: 0.8692
Epoch 80/100
237/237 [==============================] - 0s 219us/step - loss: 0.3630 -
accuracy: 0.8439
Epoch 81/100
237/237 [==============================] - 0s 202us/step - loss: 0.3670 -
accuracy: 0.8439
Epoch 82/100
237/237 [==============================] - 0s 202us/step - loss: 0.3623 -
accuracy: 0.8439
Epoch 83/100
237/237 [==============================] - 0s 186us/step - loss: 0.3584 -
accuracy: 0.8608
Epoch 84/100
237/237 [==============================] - 0s 202us/step - loss: 0.3776 -
accuracy: 0.8312
Epoch 85/100
237/237 [==============================] - 0s 186us/step - loss: 0.3639 -
accuracy: 0.8608
Epoch 86/100
237/237 [==============================] - 0s 193us/step - loss: 0.3618 -
accuracy: 0.8397
Epoch 87/100
237/237 [==============================] - 0s 194us/step - loss: 0.3549 -
accuracy: 0.8692
Epoch 88/100
237/237 [==============================] - 0s 177us/step - loss: 0.3727 -
accuracy: 0.8354
Epoch 89/100
237/237 [==============================] - 0s 202us/step - loss: 0.3430 -
accuracy: 0.8608
Epoch 90/100
237/237 [==============================] - 0s 186us/step - loss: 0.3659 -
accuracy: 0.8608
Epoch 91/100
237/237 [==============================] - 0s 202us/step - loss: 0.3821 -
accuracy: 0.8439
Epoch 92/100
237/237 [==============================] - 0s 219us/step - loss: 0.3659 -
accuracy: 0.8397
Epoch 93/100
237/237 [==============================] - 0s 219us/step - loss: 0.3703 -
accuracy: 0.8439
Epoch 94/100
237/237 [==============================] - 0s 203us/step - loss: 0.3708 -
accuracy: 0.8523
Epoch 95/100
```

```
237/237 [==============================] - 0s 202us/step - loss: 0.3557 -
accuracy: 0.8650
Epoch 96/100
237/237 [==============================] - 0s 186us/step - loss: 0.3566 -
accuracy: 0.8608
Epoch 97/100
237/237 [==============================] - 0s 186us/step - loss: 0.4049 -
accuracy: 0.8228
Epoch 98/100
237/237 [==============================] - 0s 186us/step - loss: 0.3702 -
accuracy: 0.8650
Epoch 99/100
237/237 [==============================] - 0s 270us/step - loss: 0.3671 -
accuracy: 0.8565
Epoch 100/100
237/237 [==============================] - 0s 270us/step - loss: 0.3684 -
accuracy: 0.8565
```

[28]:
```
<keras.callbacks.callbacks.History at 0x1c48b6c1248>
```

1.0.5 5. Results and Metrics

The accuracy results we have been seeing are for the training data, but what about the testing dataset? If our model's cannot generalize to data that wasn't used to train them, they won't provide any utility.

Let's test the performance of both our categorical model and binary model. To do this, we will make predictions on the training dataset and calculate performance metrics using Sklearn.

[29]:
```python
# generate classification report using predictions for categorical model
from sklearn.metrics import classification_report, accuracy_score

categorical_pred = np.argmax(model.predict(X_test), axis=1)

print('Results for Categorical Model')
print(accuracy_score(y_test, categorical_pred))
print(classification_report(y_test, categorical_pred))
```

```
Results for Categorical Model
0.6666666666666666
              precision    recall  f1-score   support

           0       0.82      0.92      0.87        36
           1       0.33      0.09      0.14        11
           2       0.00      0.00      0.00         6
           3       0.35      1.00      0.52         6
           4       0.00      0.00      0.00         1
```

```
    accuracy                           0.67        60
   macro avg       0.30      0.40      0.31        60
weighted avg       0.59      0.67      0.60        60


E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1437:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

[30]:
```
# generate classification report using predictions for binary model
binary_pred = np.round(binary_model.predict(X_test)).astype(int)

print('Results for Binary Model')
print(accuracy_score(Y_test_binary, binary_pred))
print(classification_report(Y_test_binary, binary_pred))
```

```
Results for Binary Model
0.7833333333333333
              precision    recall  f1-score   support

           0       0.87      0.75      0.81        36
           1       0.69      0.83      0.75        24

    accuracy                           0.78        60
   macro avg       0.78      0.79      0.78        60
weighted avg       0.80      0.78      0.79        60
```

[ ]:
```
```