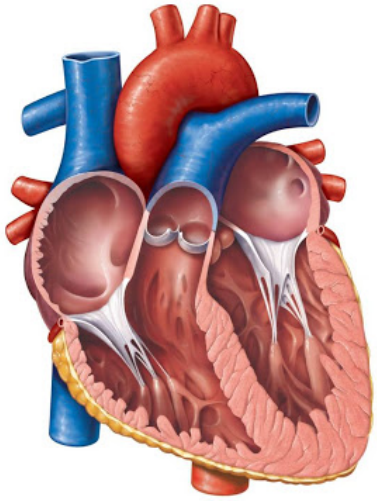
NEURAL NETWORK AND FUZZY LOGIC

IA-1 REPORT

TOPIC: HEART DISEASE PREDICTION



GROUP MEMBERS

18130011 – SHIKTA DAS

1813035 – NAVNEET PARAB

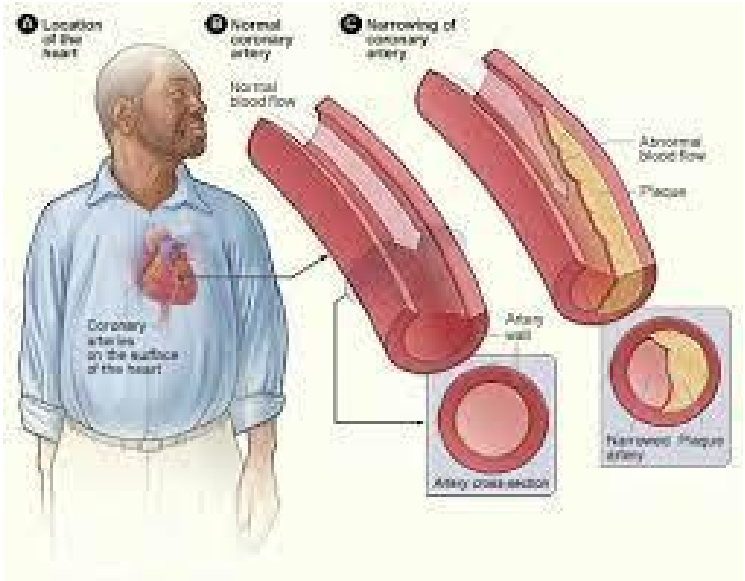
1813057 – ADITYA SHENOY

INTRODUCTION

Heart diseases are a growing cause of concern in today's hyper-tensed world. To move a step forward in solving this problem, we have designed a model which will help us in predicting the occurrence of a heart disease in people.

The data used consists of 13 parameters or 'features' which have been used to train our Neural Network model. These features consists of Age, Sex, Blood Pressure, Heart Rate, etc. Once we get a good fit, we will use our model to predict the occurrence of heart diseases.

A model like this will be really helpful for diagnostics and early treatment of at-risk heart patients.



DATASET

The dataset used is of historical data regarding heart diseases in Cleveland, Ohio. We picked this dataset from the University of California Irvine Machine Learning repository. The link for the same can be found here

```
import sys
import pandas as pd
import numpy as np
import sklearn
import matplotlib
import keras
```

```
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import seaborn as sns
```

```
# read the csv
cleveland = pd.read_csv('heart.csv')
```

The overview of the dataset is as follows:

```
cleveland.head()
```

	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

FEATURE SPECIFICATION

1. **AGE:** age in years
2. **SEX:** (1 = male; 0 = female)
3. **CP:** chest pain type
4. **TRESTBPS:** resting blood pressure (in mm Hg on admission to the hospital)
5. **CHOL:** serum cholesterol in mg/dl
6. **FBS:** (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. **RESTECG:** resting electrocardiographic results
8. **THALACH:** maximum heart rate achieved
9. **EXANG**: exercise induced angina (1 = yes; 0 = no)
10. **OLDPEAK**: ST depression induced by exercise relative to rest
11. **SLOPE**: the slope of the peak exercise ST segment
12. **CA**: number of major vessels (0-3) colored by fluoroscopy
13. **THAL**: 3 = normal; 6 = fixed defect; 7 = reversible defect
14. **TARGET**: 1 or 0

PREPROCESSING

In preprocessing we clean the data i.e. we fill or remove the missing values, change the data types and correct the errors in spelling. Also duplicate items should be removed since it would be redundant.

Fortunately in our dataset, there were no null values or missing values. All values were already typecast to float or integer types.

```
data=cleveland

# print the shape and data type of the dataframe
print(data.shape)
print(data.dtypes)
```

```
(303, 14)
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
```

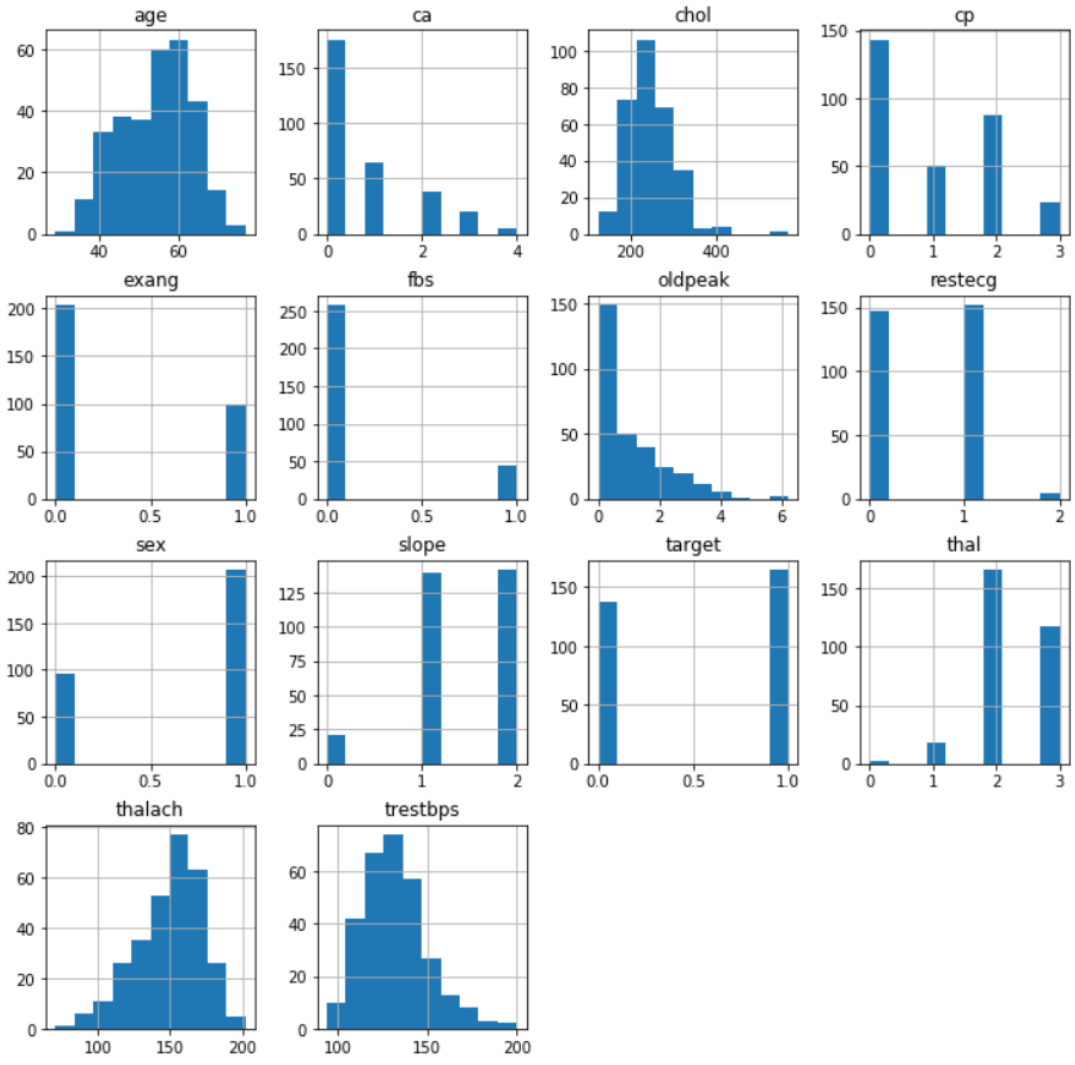
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   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
```

EXPLORATORY DATA ANALYSIS

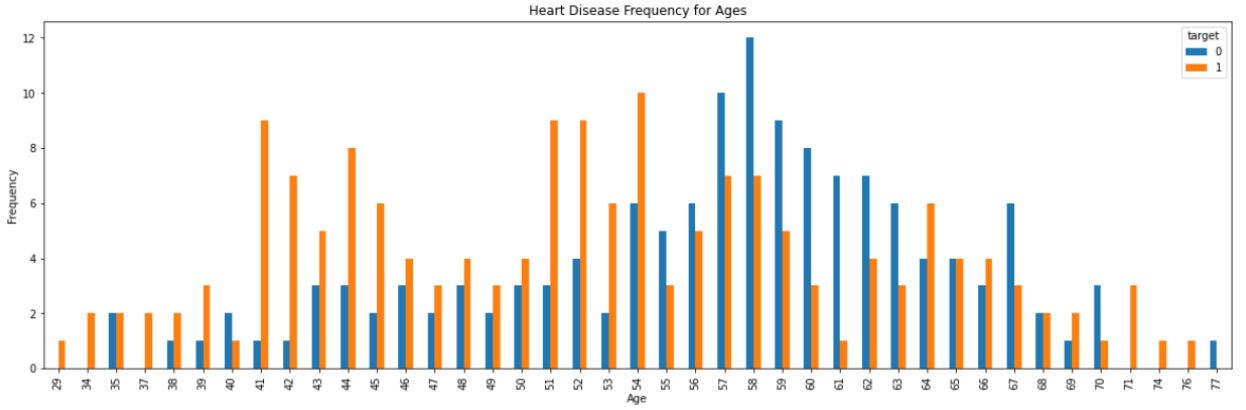
Firstly, we plot a histogram for each variable using matplotlib.pyplot

```
# plot histograms for each variable
data.hist(figsize = (10, 10))
plt.show()
```



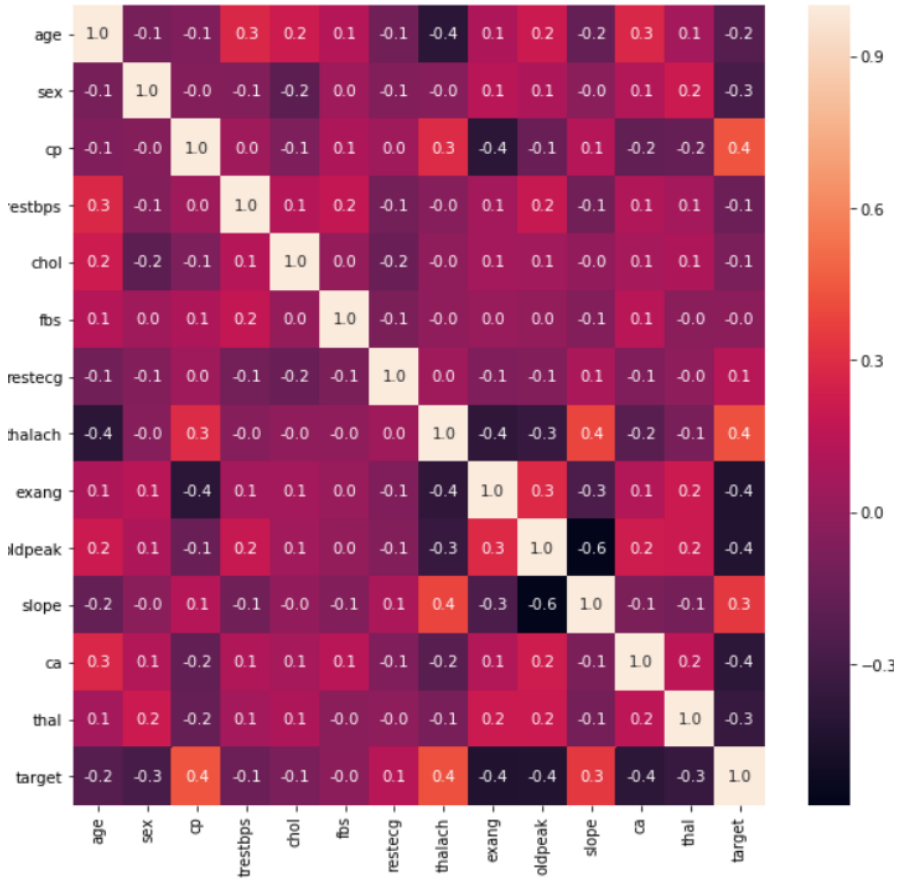
We also plotted the frequency of target variable versus each age:

```
pd.crosstab(data.age,data.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



We also plotted a heatmap to see the correlation between each variable:

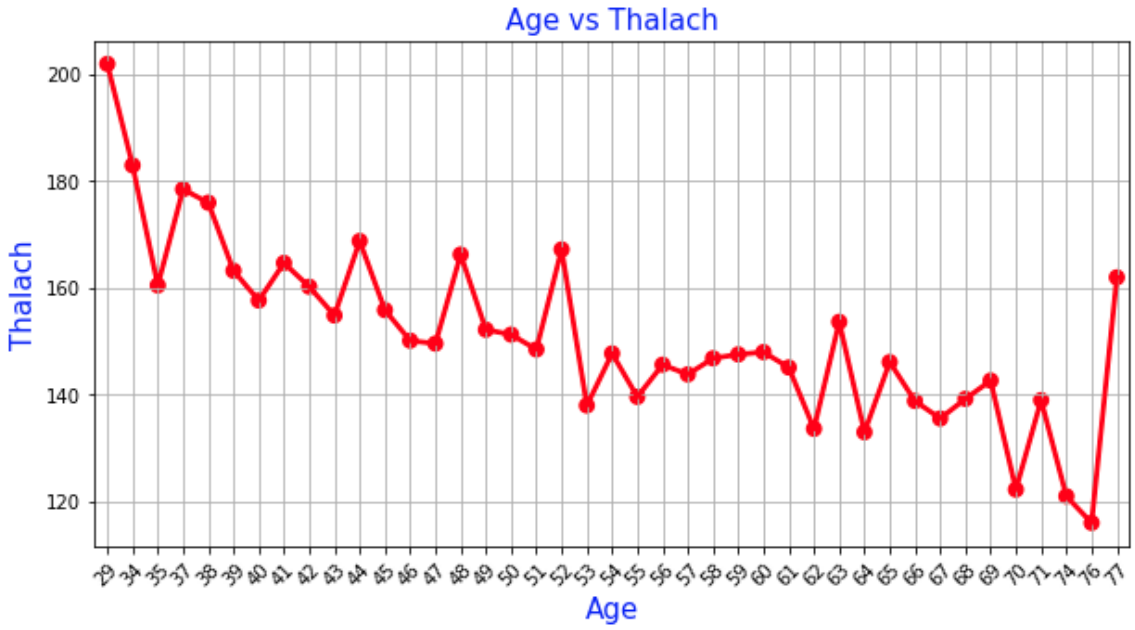
```
plt.figure(figsize=(20,15))
sns.heatmap(data.corr(),annot=True,fmt='.1f')
plt.show()
```



Here we have arranged unique instances of age and made it into a pandas series. We have then grouped the ‘thalach’ feature with this series and plotted it out.

```
#Sequential plotting of age vs thalach

plt.figure(figsize=(10,5))
sns.pointplot(x=age_unique,y=mean_thalach,color='red',alpha=0.8)
plt.xlabel('Age',fontsize = 15,color='blue')
plt.xticks(rotation=45)
plt.ylabel('Thalach',fontsize = 15,color='blue')
plt.title('Age vs Thalach',fontsize = 15,color='blue')
plt.grid()
plt.show()
```



DATA SCALING & TRAIN-TEST SPLIT

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

X[0]
```

```
array([ 63. ,   1. ,   3. , 145. , 233. ,   1. ,   0. , 150. ,   0. ,
         2.3,   0. ,   0. ,   1. ])
```

```
#Standardizing and Normalization of input data

mean = X.mean(axis=0)
X -= mean
std = X.std(axis=0)
X /= std
X[0]
```

```
array([ 0.9521966 ,  0.68100522,  1.97312292,  0.76395577, -0.25633371,
        2.394438  , -1.00583187,  0.01544279, -0.69663055,  1.08733806,
       -2.27457861, -0.71442887, -2.14887271])
```

```
#Train Test Split

from sklearn import model_selection

X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, stratify=y, random_state=101, test_size = 0.2)
```

We have selected 80% of our data for training and rest 20% for testing.

The `stratify = y` argument ensures that the ratio of values for y remains the same for the training set as well as the testing set.

To check the robustness of the model, we will be deploying it with Categorical input as well as Binary input. We will use the Keras library for the same.

Categorical Input:

```
# 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])
```

```
(242, 2)
[[0. 1.]
 [1. 0.]
 [1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]]
```

BUILDING CATEGORICAL MODEL

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.layers import Dropout
from keras import regularizers

# define a function to build the keras model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.00
    model.add(Dropout(0.25))
    model.add(Dense(8, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation=
    model.add(Dropout(0.25))
    model.add(Dense(2, activation='softmax'))

    # compile model
    adam = Adam(lr=0.001)
    model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    return model

model = create_model()

print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 16)                224
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout_3 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 18
=================================================================
Total params: 378
Trainable params: 378
Non-trainable params: 0
_________________________________________________________________
None
```

In our Neural Network, we have included 3 Dense Layers and 2 Dropout Layers. Furthermore, we have used the Adam compiler.

The summary of our model can be seen above.

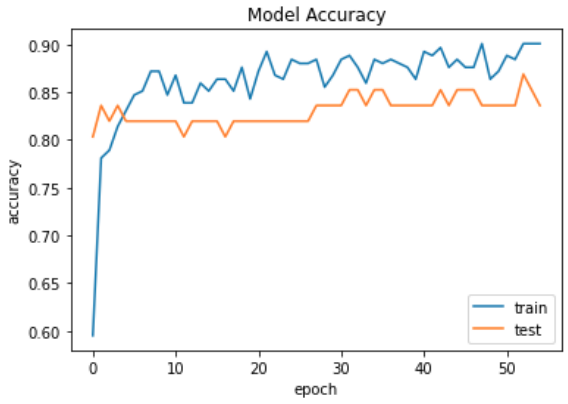
We fit the model to our data and ran it for 55 epochs

```
# fit the model to the training data
history=model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=55, batch_size=10)
```

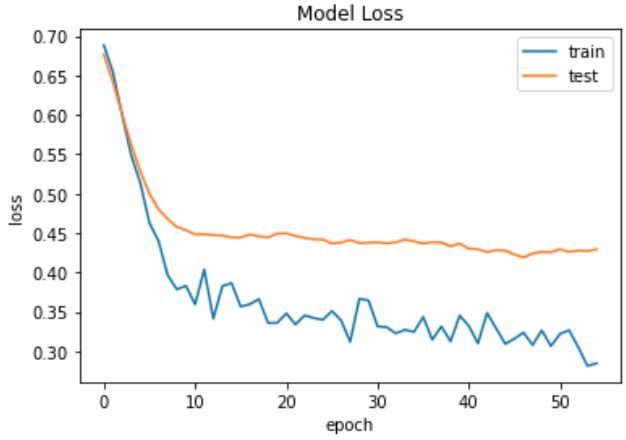
```
361
Epoch 50/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3067 - accuracy: 0.8719 - val_loss: 0.4257 - val_accuracy: 0.8
361
Epoch 51/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3223 - accuracy: 0.8884 - val_loss: 0.4293 - val_accuracy: 0.8
361
Epoch 52/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3267 - accuracy: 0.8843 - val_loss: 0.4262 - val_accuracy: 0.8
361
Epoch 53/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3052 - accuracy: 0.9008 - val_loss: 0.4278 - val_accuracy: 0.8
689
Epoch 54/55
25/25 [==============================] - 0s 2ms/step - loss: 0.2816 - accuracy: 0.9008 - val_loss: 0.4272 - val_accuracy: 0.8
525
Epoch 55/55
25/25 [==============================] - 0s 2ms/step - loss: 0.2848 - accuracy: 0.9008 - val_loss: 0.4294 - val_accuracy: 0.8
361
```

MODEL ACCURACY & LOSSES

```python
import matplotlib.pyplot as plt
%matplotlib inline
# Model accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```




```python
# Model Losss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



BUILDING BINARY MODEL

Binary Input:

```
# 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])
```

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

We have designed and implemented the same Neural Network as we did for the categorical data. We have fitted the model and run it for the same number of epochs (i.e. 55) too.

```
# define a new keras model for binary classification
def create_binary_model():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, kernel_initializer='normal',  kernel_regularizer=regularizers.l2(0.001),activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(8, kernel_initializer='normal',  kernel_regularizer=regularizers.l2(0.001),activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(1, activation='sigmoid'))

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

binary_model = create_binary_model()

print(binary_model.summary())
```

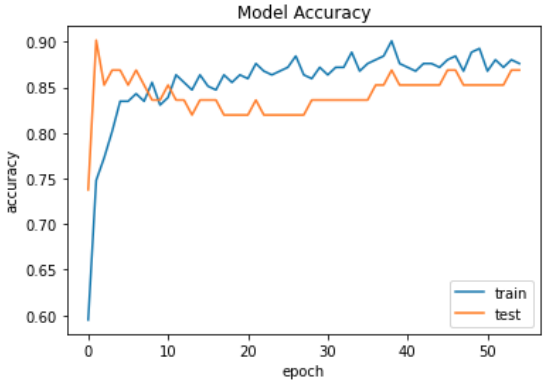
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 16)                224
_________________________________________________________________
dropout_6 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 136
_________________________________________________________________
dropout_7 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 9
=================================================================
Total params: 369
Trainable params: 369
Non-trainable params: 0
_________________________________________________________________
None
```

```
# fit the binary model on the training data
history=binary_model.fit(X_train, Y_train_binary, validation_data=(X_test, Y_test_binary), epochs=55, batch_size=10)
```

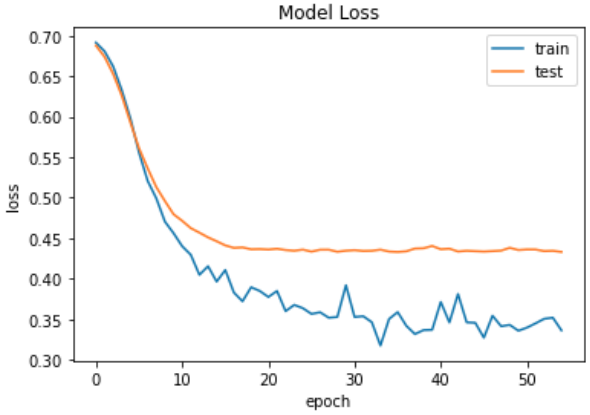
```
525
Epoch 50/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3357 - accuracy: 0.8926 - val_loss: 0.4354 - val_accuracy: 0.8
525
Epoch 51/55
25/25 [==============================] - 0s 3ms/step - loss: 0.3396 - accuracy: 0.8678 - val_loss: 0.4361 - val_accuracy: 0.8
525
Epoch 52/55
25/25 [==============================] - 0s 3ms/step - loss: 0.3449 - accuracy: 0.8802 - val_loss: 0.4360 - val_accuracy: 0.8
525
Epoch 53/55
25/25 [==============================] - 0s 3ms/step - loss: 0.3505 - accuracy: 0.8719 - val_loss: 0.4340 - val_accuracy: 0.8
525
Epoch 54/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3518 - accuracy: 0.8802 - val_loss: 0.4344 - val_accuracy: 0.8
689
Epoch 55/55
25/25 [==============================] - 0s 2ms/step - loss: 0.3362 - accuracy: 0.8760 - val_loss: 0.4330 - val_accuracy: 0.8
689
```

MODEL ACCURACY & LOSSES

```python
import matplotlib.pyplot as plt
%matplotlib inline
# Model accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```




```python
# Model Losss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```



COMPARISON OF METRICS

Using the `metrics` module of scikit-learn library, we find the various prediction results for the categorical and the binary models.

```
# 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
0.8360655737704918
              precision    recall  f1-score   support

           0       0.85      0.79      0.81        28
           1       0.83      0.88      0.85        33

    accuracy                           0.84        61
   macro avg       0.84      0.83      0.83        61
weighted avg       0.84      0.84      0.84        61
```

```
# generate classification report using predictions for binary model
from sklearn.metrics import classification_report, accuracy_score
# 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
0.8688524590163934
              precision    recall  f1-score   support

           0       0.88      0.82      0.85        28
           1       0.86      0.91      0.88        33

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

INFERENCE

After our entire project, we learnt the following

1. The size of our dataset was too small to get appreciable results of more than 90%
2. As a result of a smaller dataset, our Neural Network had to be quite simple in complexity.
3. Among Binary and Categorical inputs, we found that better results were obtained in the former model.