Assignment 10 : Neural networks

```
from google.colab import files
uploaded = files.upload()
```

Choose Files heart.csv

- **heart.csv**(text/csv) - 11328 bytes, last modified: 30/11/2020 - 100% done

```
Saving heart.csv to heart (1).csv
```

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import keras
```

```
data = pd.read_csv('heart.csv',header=0)
```

Exploratory Data Analysis

```
data.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

```
data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	s
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.0
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.3
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.6
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.0
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.0
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.0
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.0
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.0

```
data = data[~data.isin(['?'])]
data = data.dropna(axis=0)
```

```
data.info()
print(data.dtypes)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 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: 35.5 KB
     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
```

```
#Convert all values to integer
data = data.apply(pd.to_numeric)
data.dtypes
```

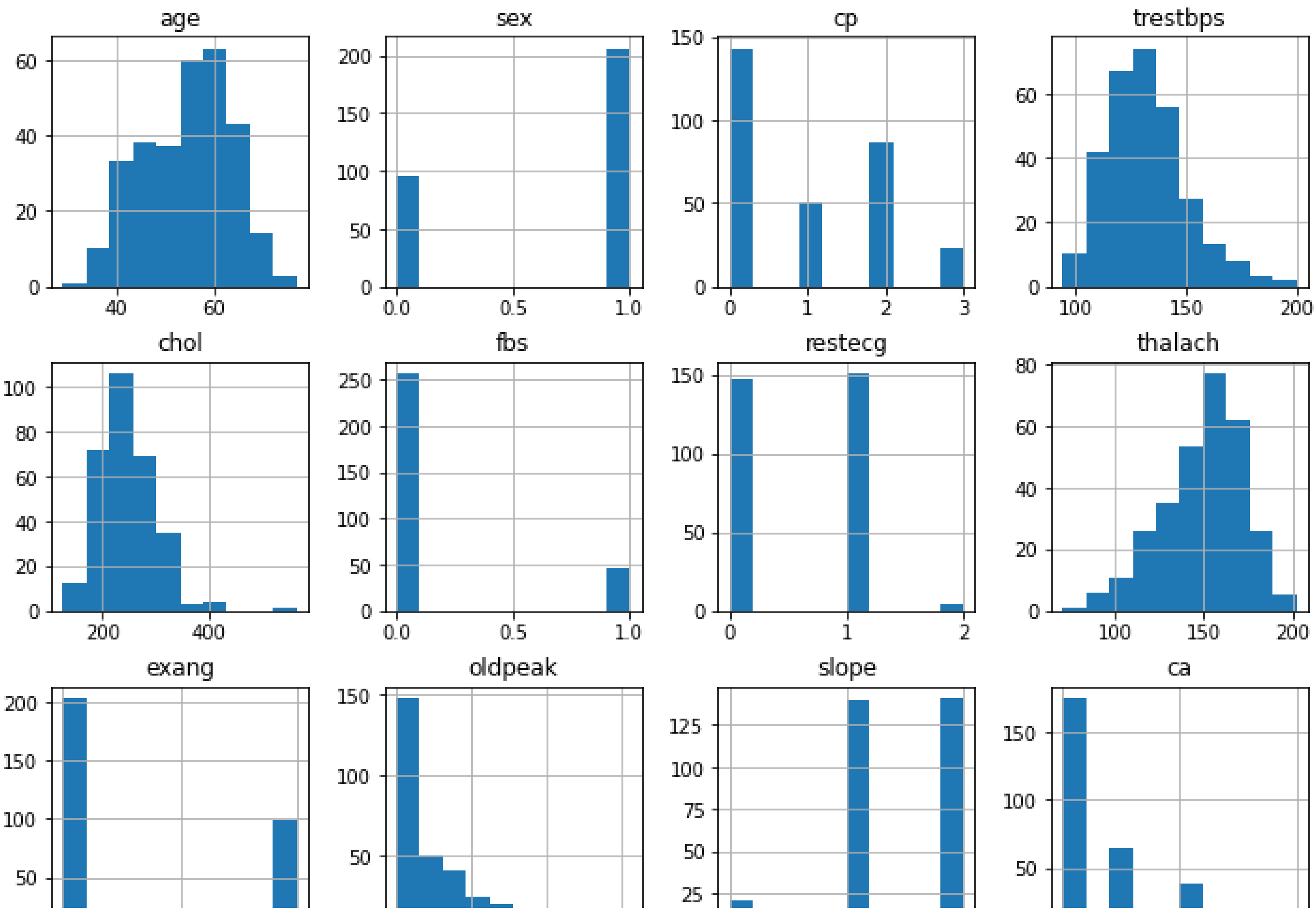
```
     age           int64
     sex           int64
     cp            int64
     trestbps      int64
     chol          int64
     fbs           int64
     restecg       int64
     thalach       int64
     exang         int64
     oldpeak     float64
     slope         int64
     ca            int64
     thal          int64
     target        int64
     dtype: object
```

```
data.isnull().sum()
```

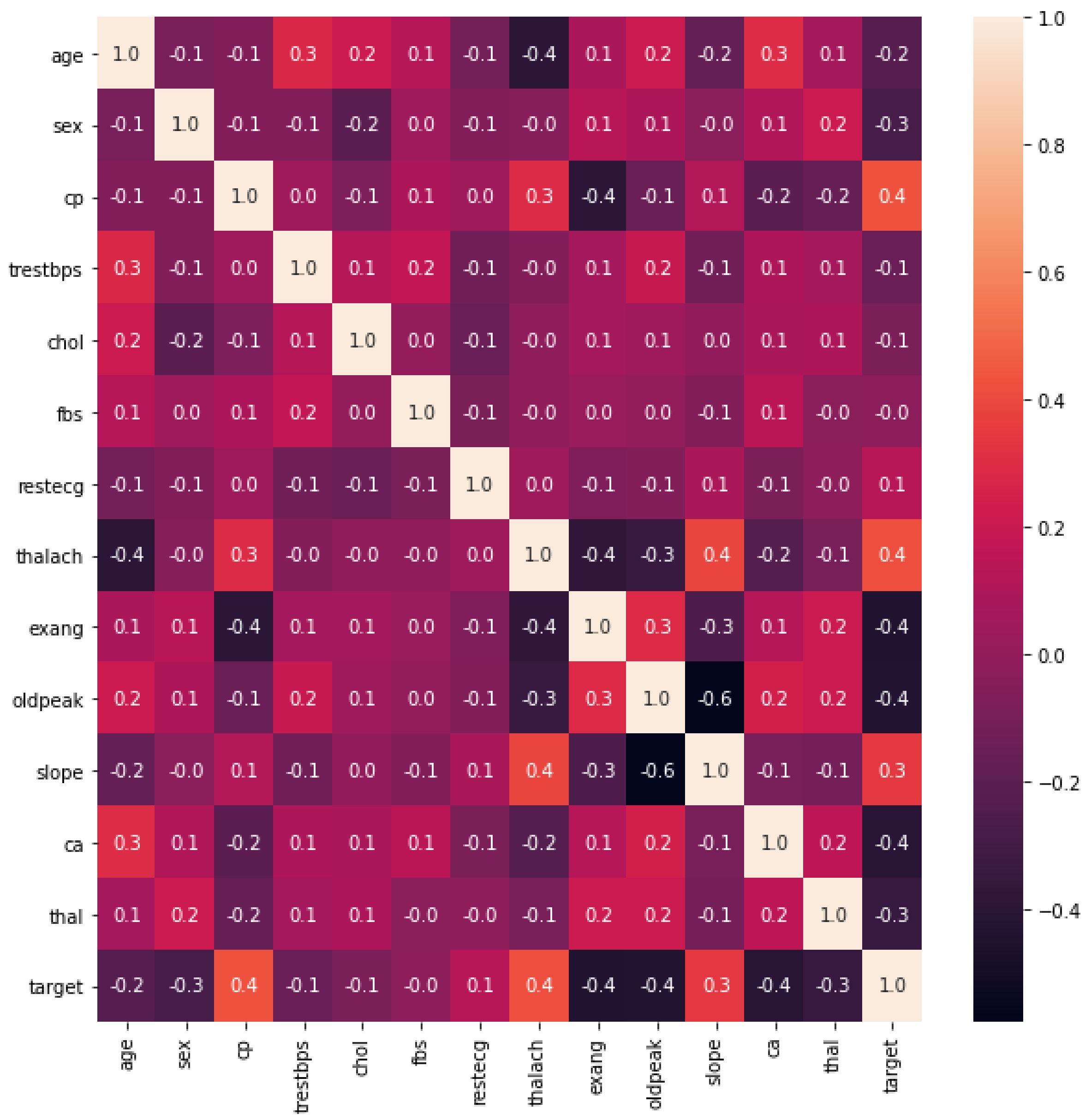
```
     age         0
     sex         0
     cp          0
     trestbps    0
     chol        0
     fbs         0
     restecg     0
     thalach     0
     exang       0
     oldpeak     0
     slope       0
     ca          0
     thal        0
     target      0
     dtype: int64
```

```
data.drop_duplicates(inplace=True)
```

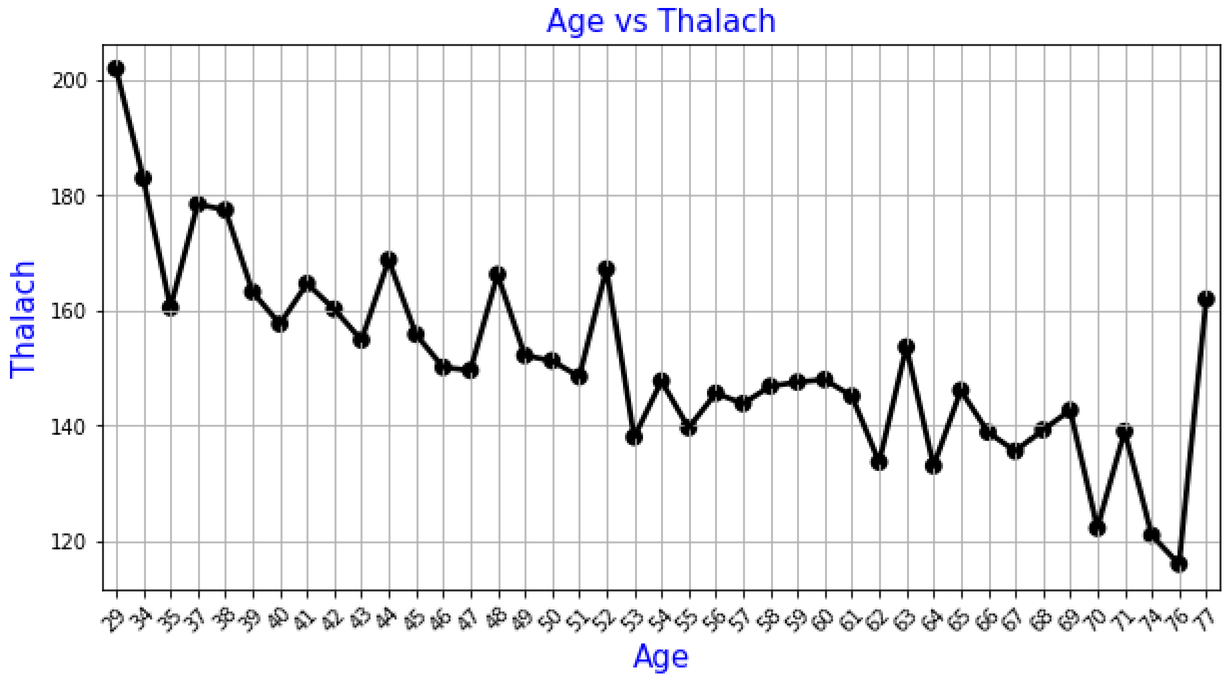
```
data.hist(figsize = (12, 12))
plt.show()
```



```
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True,fmt='.1f')
plt.show()
```



```
age_unique=sorted(data.age.unique())
age_thalach_values=data.groupby('age')['thalach'].count().values
mean_thalach=[]
for i,age in enumerate(age_unique):
    mean_thalach.append(sum(data[data['age']==age].thalach)/age_thalach_values[i])

plt.figure(figsize=(10,5))
sns.pointplot(x=age_unique,y=mean_thalach,color='black',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()
```



Train and test

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


#Scaling data
from sklearn.preprocessing import StandardScaler
X_copy = X.copy()
y_copy = y.copy()

mean = X.mean(axis=0)
X -= mean
std = X.std(axis=0)
X /= std


#Test train split
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,y,stratify=y, random_state=42, test_size = 0.2)


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])
```

```
    (241, 2)
    [[0. 1.]
     [1. 0.]
     [1. 0.]
     [1. 0.]
     [0. 1.]
     [0. 1.]
     [0. 1.]
     [0. 1.]
     [1. 0.]
     [0. 1.]]
```

Building and training network

```
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(0.001), activation=
    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(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())
```

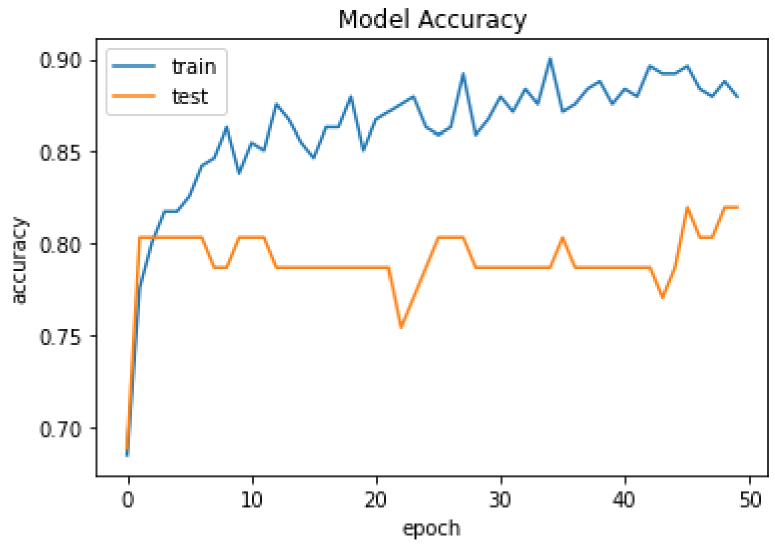
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                224
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout_1 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 18
=================================================================
Total params: 378
Trainable params: 378
Non-trainable params: 0
_________________________________________________________________
None
```

```
history=model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=50, batch_size=10)
```

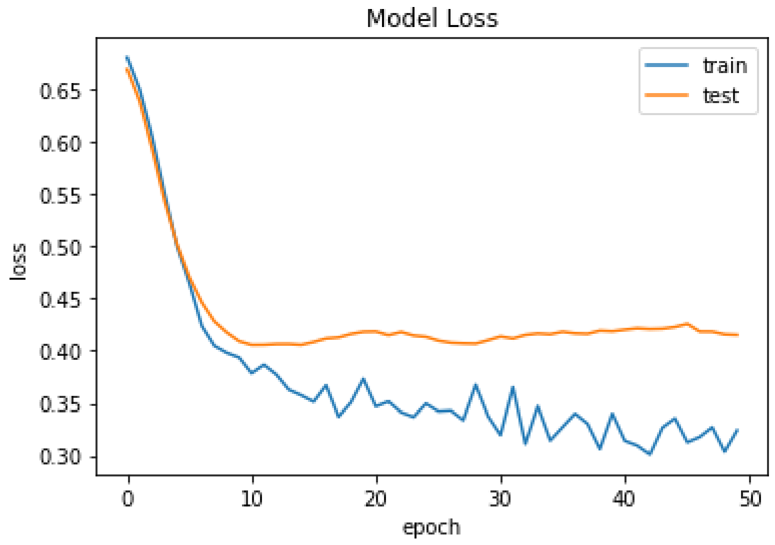
```
25/25 [==============================] - 0s 2ms/step - loss: 0.3469 - accuracy: 0.8672 - val_loss: 0.4184 - val_ac
Epoch 22/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3516 - accuracy: 0.8714 - val_loss: 0.4151 - val_ac
Epoch 23/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3411 - accuracy: 0.8755 - val_loss: 0.4180 - val_ac
Epoch 24/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3367 - accuracy: 0.8797 - val_loss: 0.4146 - val_ac
Epoch 25/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3496 - accuracy: 0.8631 - val_loss: 0.4134 - val_ac
Epoch 26/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3421 - accuracy: 0.8589 - val_loss: 0.4097 - val_ac
Epoch 27/50

25/25 [==============================] - 0s 2ms/step - loss: 0.3428 - accuracy: 0.8631 - val_loss: 0.4078 - val_ac
Epoch 28/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3336 - accuracy: 0.8921 - val_loss: 0.4071 - val_ac
Epoch 29/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3676 - accuracy: 0.8589 - val_loss: 0.4069 - val_ac
Epoch 30/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3373 - accuracy: 0.8672 - val_loss: 0.4102 - val_ac
Epoch 31/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3195 - accuracy: 0.8797 - val_loss: 0.4137 - val_ac
Epoch 32/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3653 - accuracy: 0.8714 - val_loss: 0.4118 - val_ac
Epoch 33/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3111 - accuracy: 0.8838 - val_loss: 0.4152 - val_ac
Epoch 34/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3472 - accuracy: 0.8755 - val_loss: 0.4166 - val_ac
Epoch 35/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3143 - accuracy: 0.9004 - val_loss: 0.4159 - val_ac
Epoch 36/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3272 - accuracy: 0.8714 - val_loss: 0.4182 - val_ac
Epoch 37/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3399 - accuracy: 0.8755 - val_loss: 0.4167 - val_ac
Epoch 38/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3304 - accuracy: 0.8838 - val_loss: 0.4163 - val_ac
Epoch 39/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3063 - accuracy: 0.8880 - val_loss: 0.4193 - val_ac
Epoch 40/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3401 - accuracy: 0.8755 - val_loss: 0.4187 - val_ac
Epoch 41/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3142 - accuracy: 0.8838 - val_loss: 0.4202 - val_ac
Epoch 42/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3096 - accuracy: 0.8797 - val_loss: 0.4216 - val_ac
Epoch 43/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3013 - accuracy: 0.8963 - val_loss: 0.4209 - val_ac
Epoch 44/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3267 - accuracy: 0.8921 - val_loss: 0.4212 - val_ac
Epoch 45/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3355 - accuracy: 0.8921 - val_loss: 0.4228 - val_ac
Epoch 46/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3127 - accuracy: 0.8963 - val_loss: 0.4258 - val_ac
Epoch 47/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3178 - accuracy: 0.8838 - val_loss: 0.4184 - val_ac
```

```
Epoch 48/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3270 - accuracy: 0.8797 - val_loss: 0.4184 - val_ac
Epoch 49/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3037 - accuracy: 0.8880 - val_loss: 0.4157 - val_ac
Epoch 50/50
```

```
%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()
```



```
#Loss
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()
```



Improving model

```
Y_train_binary = y_train.copy()
Y_test_binary = y_test.copy()
#converting it into a binary problem
Y_train_binary[Y_train_binary > 0] = 1
Y_test_binary[Y_test_binary > 0] = 1
```

```
#New Model Creation
def create_new_model():
    model = Sequential()
    model.add(Dense(16, input_dim=13, kernel_initializer='normal',  kernel_regularizer=regularizers.l2(0.001),activation=
    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_new_model()

print(binary_model.summary())
```

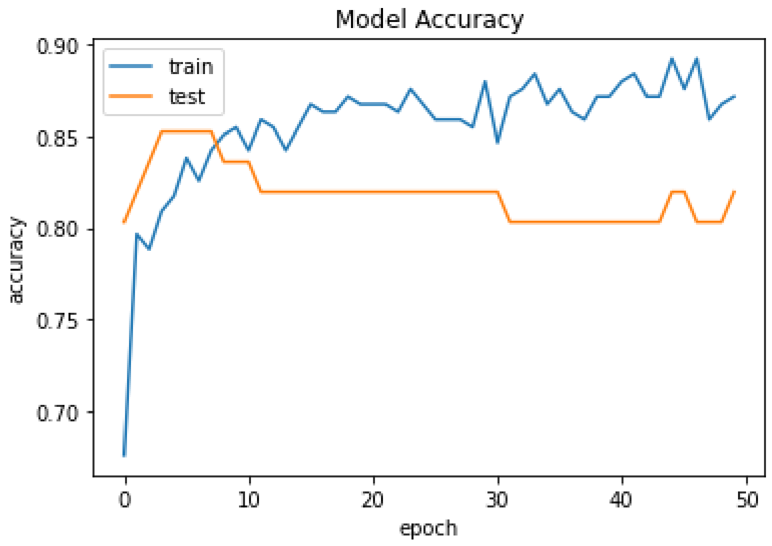
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 16)                224       
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136       
_________________________________________________________________
dropout_3 (Dropout)          (None, 8)                 0         
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 9         
=================================================================
Total params: 369
Trainable params: 369
Non-trainable params: 0
_________________________________________________________________
None
```

```
history=binary_model.fit(X_train, Y_train_binary, validation_data=(X_test, Y_test_binary), epochs=50, batch_size=10)
```

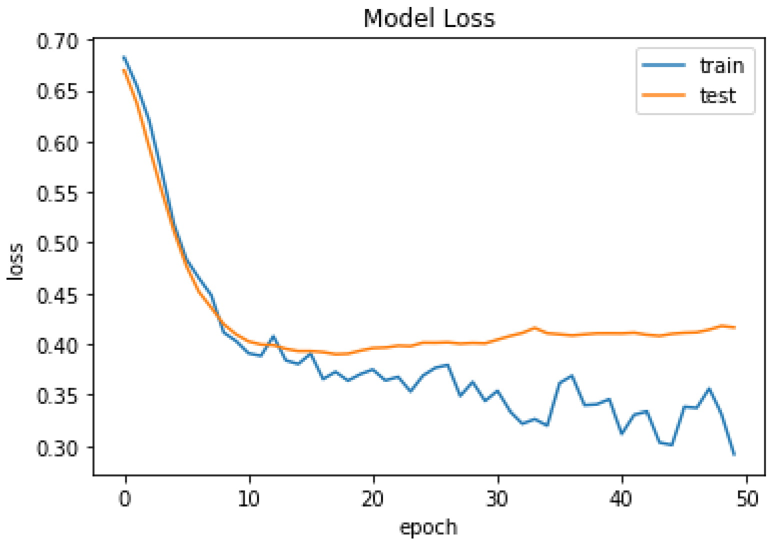
```
25/25 [==============================] - 0s 2ms/step - loss: 0.3748 - accuracy: 0.8672 - val_loss: 0.3959 - val_ac
Epoch 22/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3641 - accuracy: 0.8672 - val_loss: 0.3964 - val_ac
Epoch 23/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3675 - accuracy: 0.8631 - val_loss: 0.3983 - val_ac
Epoch 24/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3530 - accuracy: 0.8755 - val_loss: 0.3978 - val_ac
Epoch 25/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3688 - accuracy: 0.8672 - val_loss: 0.4013 - val_ac
Epoch 26/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3765 - accuracy: 0.8589 - val_loss: 0.4012 - val_ac
Epoch 27/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3793 - accuracy: 0.8589 - val_loss: 0.4016 - val_ac
Epoch 28/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3486 - accuracy: 0.8589 - val_loss: 0.4003 - val_ac
Epoch 29/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3625 - accuracy: 0.8548 - val_loss: 0.4010 - val_ac
Epoch 30/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3437 - accuracy: 0.8797 - val_loss: 0.4006 - val_ac
Epoch 31/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3537 - accuracy: 0.8465 - val_loss: 0.4043 - val_ac
Epoch 32/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3339 - accuracy: 0.8714 - val_loss: 0.4078 - val_ac
Epoch 33/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3217 - accuracy: 0.8755 - val_loss: 0.4108 - val_ac
Epoch 34/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3263 - accuracy: 0.8838 - val_loss: 0.4158 - val_ac
Epoch 35/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3198 - accuracy: 0.8672 - val_loss: 0.4106 - val_ac
Epoch 36/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3614 - accuracy: 0.8755 - val_loss: 0.4095 - val_ac
Epoch 37/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3686 - accuracy: 0.8631 - val_loss: 0.4084 - val_ac
Epoch 38/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3396 - accuracy: 0.8589 - val_loss: 0.4095 - val_ac
Epoch 39/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3404 - accuracy: 0.8714 - val_loss: 0.4104 - val_ac
Epoch 40/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3455 - accuracy: 0.8714 - val_loss: 0.4105 - val_ac
Epoch 41/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3116 - accuracy: 0.8797 - val_loss: 0.4104 - val_ac
Epoch 42/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3306 - accuracy: 0.8838 - val_loss: 0.4111 - val_ac
Epoch 43/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3336 - accuracy: 0.8714 - val_loss: 0.4091 - val_ac
Epoch 44/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3032 - accuracy: 0.8714 - val_loss: 0.4079 - val_ac
Epoch 45/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3007 - accuracy: 0.8921 - val_loss: 0.4100 - val_ac
Epoch 46/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3380 - accuracy: 0.8755 - val_loss: 0.4110 - val_ac
Epoch 47/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3369 - accuracy: 0.8921 - val_loss: 0.4115 - val_ac
Epoch 48/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3560 - accuracy: 0.8589 - val_loss: 0.4141 - val_ac
Epoch 49/50
25/25 [==============================] - 0s 2ms/step - loss: 0.3308 - accuracy: 0.8672 - val_loss: 0.4178 - val_ac
```

```
Epoch 50/50
25/25 [==============================] - 0s 2ms/step - loss: 0.2916 - accuracy: 0.8714 - val_loss: 0.4163 - val_ac
```

```
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()
```



```
# 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()
```



Metrics

```
from sklearn.metrics import classification_report, accuracy_score

categorical_pred = np.argmax(model.predict(X_test), axis=1)
print("Results for 1st model used")
print(accuracy_score(y_test, categorical_pred))
print(classification_report(y_test, categorical_pred))
```

```
Results for 1st model used
0.819672131147541
              precision    recall  f1-score   support

           0       0.81      0.79      0.80        28
           1       0.82      0.85      0.84        33

    accuracy                           0.82        61
   macro avg       0.82      0.82      0.82        61
weighted avg       0.82      0.82      0.82        61
```

```
from sklearn.metrics import classification_report, accuracy_score
```

```
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
0.819672131147541
              precision    recall  f1-score   support

           0       0.90      0.68      0.78        28
           1       0.78      0.94      0.85        33

    accuracy                           0.82        61
   macro avg       0.84      0.81      0.81        61
weighted avg       0.83      0.82      0.82        61
```