Final Project

Dataset: UCI Heart disease dataset.

Report created by: Keyur Unadkat

Description of the dataset:

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Target" field refers to the presence of heart disease in the patient. It is integer valued 0 or 1.

Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (value 1) from absence (value 0).

Attribute Information:

1. age
2. sex (male==1 and female==0)
3. chest pain type (4 values) (**cp**)
    -- Value 1: typical angina
    -- Value 2: atypical angina
    -- Value 3: non-anginal pain
    -- Value 4: asymptomatic
4. Resting blood pressure (**trestbps**)
5. Serum cholesterol in mg/dl (**chol**)
6. Fasting blood sugar > 120 mg/dl (**fbs)**
7. Resting electrocardiographic results (values 0,1,2) (**restecg**)
   -- Value 0: normal.
   -- Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
   -- Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria.
8. Maximum heart rate achieved (**thalach**)
9. Exercise induced angina (**exang**)
10. Oldpeak = ST depression induced by exercise relative to rest **(oldpeak)**
11. Slope of the peak exercise ST segment. (**slope**)
    -- Value 1: upsloping
    -- Value 2: flat
    -- Value 3: down sloping

12. Number of major vessels (0-3) coloured by fluoroscopy (**ca**)
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect (**thal**)

For the given classification problem, we would leverage the power of the deep neural networks to predict the

EDA and Feature Engineering:

Exploratory Data Analysis (EDA) is an open-ended process where we calculate statistics and make figures to find trends, anomalies, patterns, or relationships within the data. The goal of EDA is to learn what our data can tell us. It generally starts out with a high-level overview, then narrows in to specific areas as we find intriguing areas of the data. The findings may be interesting in their own right, or they can be used to inform our modelling choices, such as by helping us decide which features to use.

→importing libraries:

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

→Overview of the data:

```
data=pd.read_csv("/content/heart.csv")
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.shape
```

```
(303, 14)
```

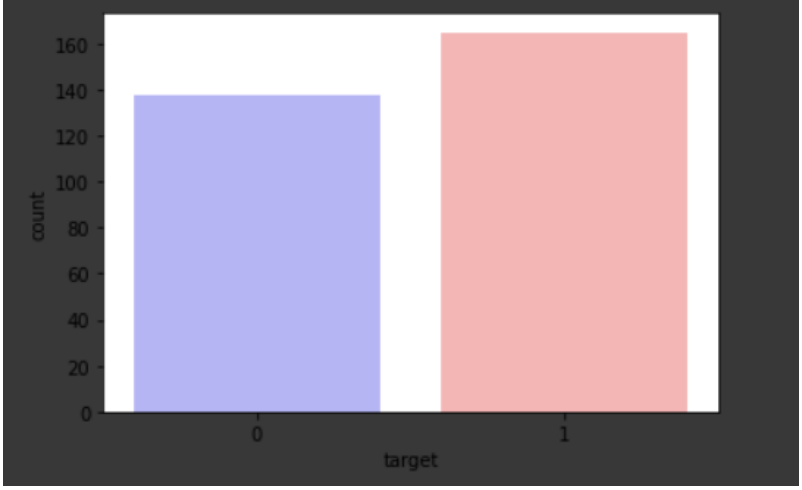
So, there are just 303 rows of the data with us, along with 13 features.

Let's see the distribution of the "target" variable:

```
data.target.value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

```
sns.countplot(x="target", data=data, palette="bwr")
plt.show()
```



The data seems to be fairly distributed. Both the classes have comparable number of occurrences.

The datatypes of the given features are:

```
[10] 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
```

Luckily all the given features are “numerical”.

We now need to see if “null” values are present in our data or not.

```
[14] data.isna().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
```

There are no null values in the given dataset, this makes our work easy.

The statistical summary of the data is as follows:

```
data.describe()
```

	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

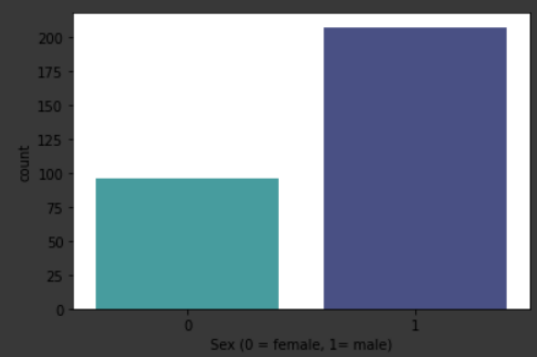
thalach	exang	oldpeak	slope	ca	thal	target
303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

Insight: The given data is widely spread on different scales and units, we would need to scale our data using either MInMaxScaler or StandardScaler.

Now, let us study the spread of the data to get some insights.

Disease Vs Gender (how likely a sex is to have a heart disease)

```
sns.countplot(x='sex', data=data, palette="mako_r")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
countFemale = len(data[data.sex == 0])
countMale = len(data[data.sex == 1])
print("Percentage of Female Patients: {:.2f}%".format((countFemale / (len(data.sex))*100)))
print("Percentage of Male Patients: {:.2f}%".format((countMale / (len(data.sex))*100)))
```

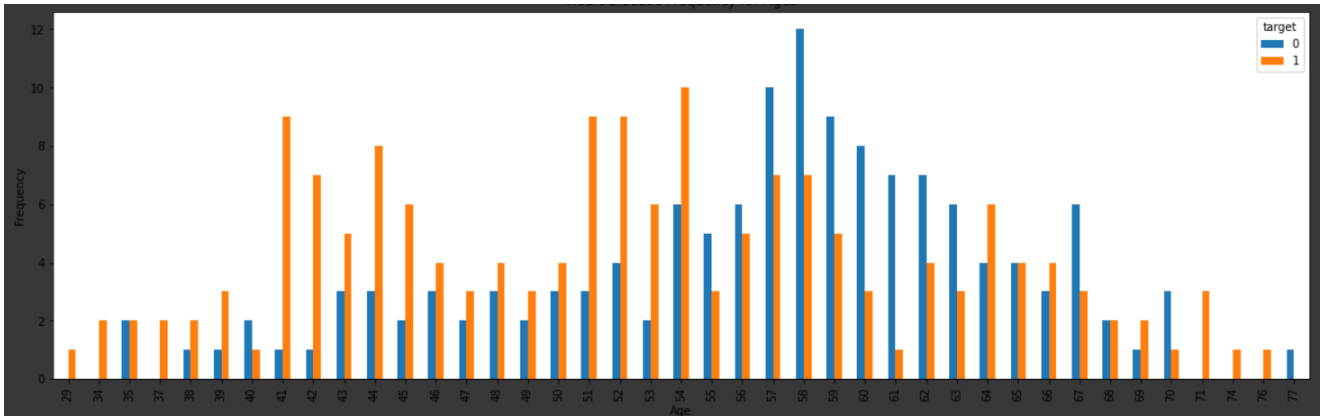



```
Percentage of Female Patients: 31.68%
Percentage of Male Patients: 68.32%
```

The given plot shows us that males are more probable to have a disease than a female.

Age Vs Disease frequency

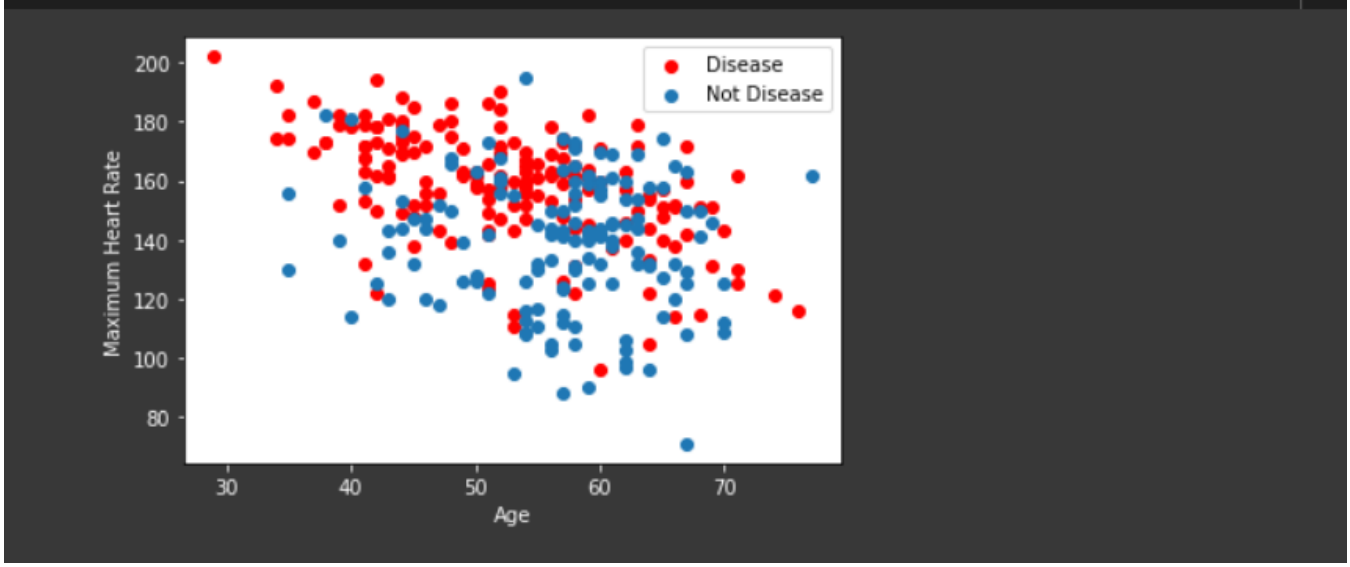
```
[24] pd.crosstab(data.age,data.target).plot(kind="bar",figsize=(20,6))
     plt.title('Heart Disease Frequency for Ages')
     plt.xlabel('Age')
     plt.ylabel('Frequency')
     plt.show()
```



We can see that the age group of 50-60 is more likely to have disease than other age groups.

Heartrate Vs Disease (scatter plot)

```
[29] plt.scatter(x=data.age[data.target==1], y=data.thalach[(data.target==1)], c="red")
     plt.scatter(x=data.age[data.target==0], y=data.thalach[(data.target==0)])
     plt.legend(["Disease", "Not Disease"])
     plt.xlabel("Age",color='white')
     plt.ylabel("Maximum Heart Rate",color='white')
     plt.tick_params(colors='white')
     plt.show()
```



From the given graph, we can say that there is some relationship between heartrate and the disease. Higher the heartrate, higher is the probability to have disease. This plot also shows some relationship between age and heartrate.

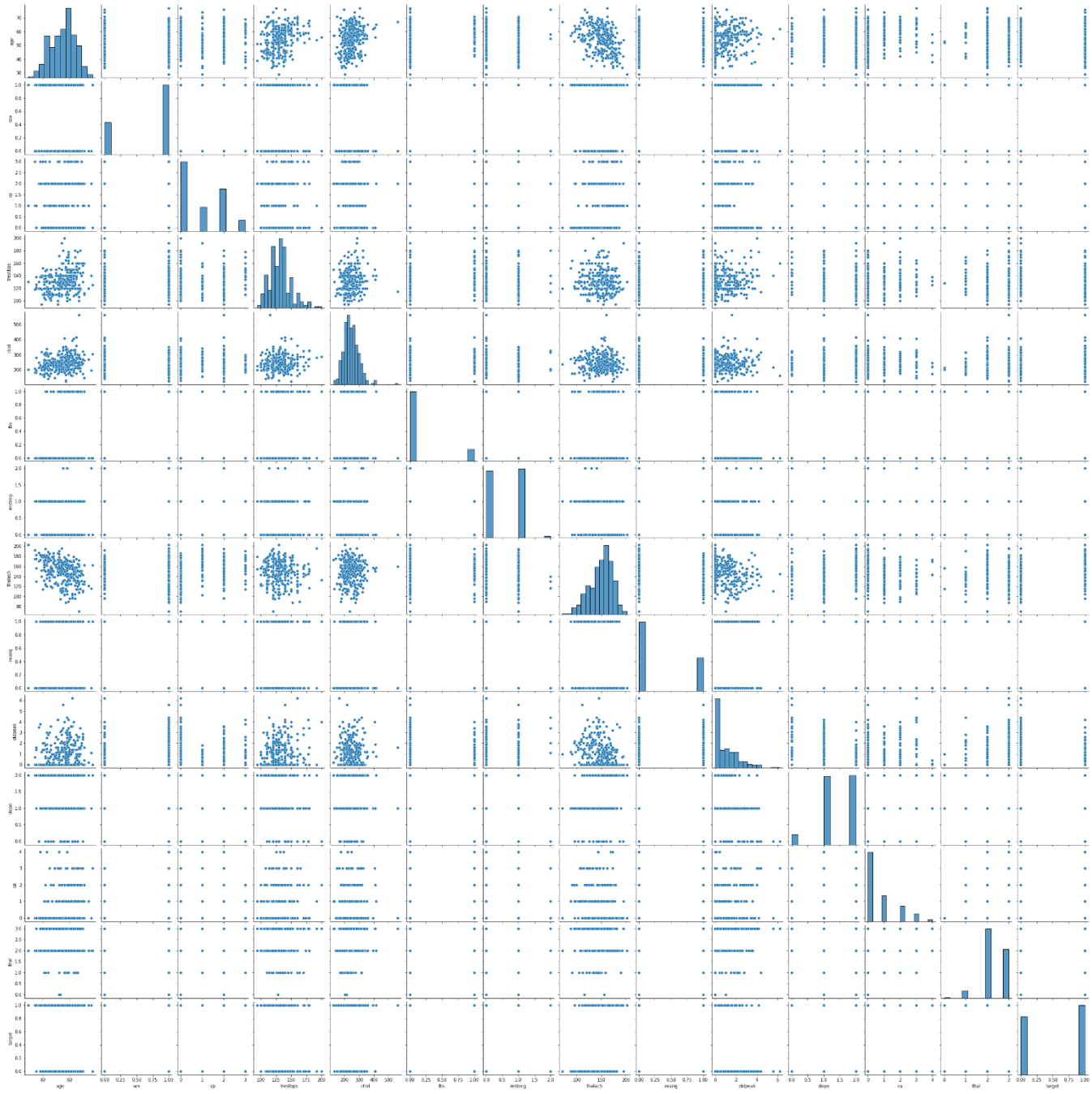
Let us now study the correlation among the features, some of the highly correlated pairs include:

```
[34] correlations=data.corr()
     for i in range(correlations.shape[0]):
       correlations.iloc[i,i]=0.0
     correlations.abs().idxmax()
```

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

Insight: As expected, age and heartrate are highly correlated. Blood sugar is also very much related to heartrate.

→pair plots for the given data.



Feature engineering:

There is not much to do with the data given to us as it is already clean, we would need to one hot encode some of the features. Those are **CP** (chest pain), **Thal** and **Slope**.

```
[51] data=pd.get_dummies(data,columns=["cp","thal","slope"],drop_first=True)

[52] data.shape

     (303, 19)

[53] data.head()
```

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	target	cp_1	cp_2	cp_3	thal_1	thal_2	thal_3	slope_1	slope_2
0	63	1	145	233	1	0	150	0	2.3	0	1	0	0	1	1	0	0	0	0
1	37	1	130	250	0	1	187	0	3.5	0	1	0	1	0	0	1	0	0	0
2	41	0	130	204	0	0	172	0	1.4	0	1	1	0	0	0	1	0	0	1
3	56	1	120	236	0	1	178	0	0.8	0	1	1	0	0	0	1	0	0	1
4	57	0	120	354	0	1	163	1	0.6	0	1	0	0	0	0	1	0	0	1

Approach1: Setting a benchmark using Logistic Regression

Setting up libraries and the data

```
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedShuffleSplit
```

```
y=data.target.values
X=data.drop(['target'],axis=1)

[55] (X.shape,y.shape)

     ((303, 18), (303,))
```

Here we would use stratified shuffle split with test size equal to 30% of the total data.

```
[77] strat_shuffle_split=StratifiedShuffleSplit(n_splits=1,test_size=0.3,random_state=42)
     train_idx,test_idx=next(strat_shuffle_split.split(X,y))

     X_test=X.iloc[test_idx,:]
     X_train=X.iloc[train_idx,:]
     y_test=y.iloc[test_idx]
     y_train=y.iloc[train_idx]
```

We would evaluate accuracy score as well as roc_auc_score for the model.

```
lrcv=LogisticRegressionCV(Cs=10,penalty='l1',solver='liblinear').fit(X_train,y_train)
y_predict=lrcv.predict(X_test)
auc=roc_auc_score(y_test,y_predict)
acc=accuracy_score(y_predict,y_test)
(auc,acc)
```

```
(0.7636585365853658, 0.7692307692307693)
```

We have accuracy score of **0.769** and roc_auc_score of **0.763**

Approach2 Using Neural Networks

We would now like to evaluate a simple neural network model.

→But before feeding the data into the model, we would like to scale our data using MinMaxScaler.

```
scaler=MinMaxScaler()
X_train_sc=scaler.fit_transform(X_train)
X_test_sc=scaler.transform(X_test)
```

→Building a model using Sequential API

```
model_1 = Sequential()
model_1.add(Dense(36,input_shape = (18,),activation = 'sigmoid'))
model_1.add(Dense(36,input_shape=(36,),activation='sigmoid'))
model_1.add(Dense(1,activation='sigmoid'))
```

```
[89] model_1.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 36)                684
_________________________________________________________________
dense_1 (Dense)              (None, 36)                1332
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 37
=================================================================
Total params: 2,053
Trainable params: 2,053
Non-trainable params: 0
```

→Running through 200 epochs with learning rate of 0.003, with Stochastic Gradient Descent.

```
model_1.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
run_hist_1 = model_1.fit(X_train_sc, y_train, validation_data=(X_test_sc, y_test), epochs=200)
```

→Final accuracy score

```
Epoch 197/200
7/7 [==============================] - 0s 5ms/step - loss: 0.6853 - accuracy: 0.5425 - val_loss: 0.6852 - val_accuracy: 0.5495
Epoch 198/200
7/7 [==============================] - 0s 5ms/step - loss: 0.6852 - accuracy: 0.5425 - val_loss: 0.6852 - val_accuracy: 0.5495
Epoch 199/200
7/7 [==============================] - 0s 5ms/step - loss: 0.6852 - accuracy: 0.5425 - val_loss: 0.6852 - val_accuracy: 0.5495
Epoch 200/200
7/7 [==============================] - 0s 4ms/step - loss: 0.6852 - accuracy: 0.5425 - val_loss: 0.6851 - val_accuracy: 0.5495
```

So, we have a final accuracy of 0.5495 on validation set

Let's check the AUC score:

```
y_pred_class_nn_1 = model_1.predict_classes(X_test_sc)
y_pred_prob_nn_1 = model_1.predict(X_test_sc)
print(roc_auc_score(y_test,y_pred_prob_nn_1))
```

```
WARNING:tensorflow:From <ipython-input-100-cf0256598c0f>:1: Sequential.predict
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model
0.7268292682926829
```

Which is equal to 0.7268 (not better than Logistic regression)

Approach3 Neural Network with ReLu and optimisers.

→Initializing the model_2

```
[102] model_2 = Sequential()
      model_2.add(Dense(12, input_shape=(18,), activation="relu"))
      model_2.add(Dense(12,  activation="relu"))
      model_2.add(Dense(1, activation="sigmoid"))

      model_2.compile(SGD(lr = .002), "binary_crossentropy", metrics=["accuracy"])
      run_hist_2 = model_2.fit(X_train_sc, y_train, validation_data=(X_test_sc, y_test), epochs=600)
```

→The final score after 600 epochs

```
Epoch 596/600
7/7 [==============================] - 0s 6ms/step - loss: 0.4011 - accuracy: 0.8349 - val_loss: 0.5305 - val_accuracy: 0.7582
Epoch 597/600
7/7 [==============================] - 0s 5ms/step - loss: 0.4009 - accuracy: 0.8349 - val_loss: 0.5305 - val_accuracy: 0.7582
Epoch 598/600
7/7 [==============================] - 0s 6ms/step - loss: 0.4007 - accuracy: 0.8349 - val_loss: 0.5305 - val_accuracy: 0.7582
Epoch 599/600
7/7 [==============================] - 0s 5ms/step - loss: 0.4006 - accuracy: 0.8349 - val_loss: 0.5305 - val_accuracy: 0.7582
Epoch 600/600
7/7 [==============================] - 0s 5ms/step - loss: 0.4004 - accuracy: 0.8349 - val_loss: 0.5305 - val_accuracy: 0.7582
```

We have now a score of 0.8349 on the set and 0.758 on validation set.

AUC score:

```
y_pred_class_nn_2 = model_2.predict_classes(X_test_sc)
y_pred_prob_nn_2 = model_2.predict(X_test_sc)
auc2=roc_auc_score(y_test,y_pred_prob_nn_2)
auc2
```

```
0.8126829268292683
```

Which is 0.8126 (better!!)

Next Steps:

We can get more accuracy and better AUC score with more layers and some optimizers. We can see that introducing a ReLU activation helped significantly to get better score. Slower learning rate, with RMSProp would be beneficial as well. This would then require more epochs so that we can plateau accuracy score.