REGRESSION

Accuracy using ANN

In [90]:
```
# importing requered libraries
import numpy as np
import pandas as pd
import tensorflow as tf
```

In [91]:
```
##reading the dataset into a variable
dataset = pd.read_excel('Folds5x2_pp.xlsx')
```

In [92]:
```
#printing first five coloumns of dataset
dataset.head()
```

Out[92]:

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37
2	5.11	39.40	1012.16	92.14	488.56
3	20.86	57.32	1010.24	76.64	446.48
4	10.82	37.50	1009.23	96.62	473.90

In [93]:
```
#checking the dataset for any Null values
dataset.isnull().sum()
```

Out[93]:
```
AT    0
V     0
AP    0
RH    0
PE    0
dtype: int64
```

In [94]:
```
#fetching information of each row in the dataset
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9568 entries, 0 to 9567
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   AT      9568 non-null   float64
 1   V       9568 non-null   float64
 2   AP      9568 non-null   float64
 3   RH      9568 non-null   float64
 4   PE      9568 non-null   float64
dtypes: float64(5)
memory usage: 373.9 KB
```

In [95]:
```
#getting the column and row size of data
dataset.shape
```

Out[95]:
```
(9568, 5)
```

In [96]:
```
#splitting the rows into input and output
X = dataset.iloc[:, :-1].values
```

```
y = dataset.iloc[:, -1].values
```

In [97]:
```
# Separating the data into train and test.
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, shuffle=True)

X_train, X_val, y_train, y_val = train_test_split(X_train,y_train, test_size = 0.2, shuf

X_train.shape, y_train.shape,X_test.shape,y_test.shape
```

Out[97]:
```
((6123, 4), (6123,), (1914, 4), (1914,))
```

In [98]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [99]:
```
#designing the model using tenserflow
ann = Sequential()

ann.add(Dense(units=4, activation='relu'))#input layer

ann.add(Dense(units=6, activation='relu'))#hidden layer

ann.add(Dense(units=1))#output layer
```

In [118…
```
#using adam optimizer to compile the model
ann.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [102…
```
model = ann.fit(X_train, y_train, batch_size = 32, epochs = 25,validation_data=(X_val, y
              shuffle=True)
```

```
Epoch 1/25
192/192 [==============================] - 1s 3ms/step - loss: 1343.2537 - val_loss: 15
9.7437
Epoch 2/25
192/192 [==============================] - 0s 2ms/step - loss: 147.2841 - val_loss: 149.
7636
Epoch 3/25
192/192 [==============================] - 0s 2ms/step - loss: 136.6754 - val_loss: 137.
9041
Epoch 4/25
192/192 [==============================] - 0s 2ms/step - loss: 125.2263 - val_loss: 124.
8189
Epoch 5/25
192/192 [==============================] - 0s 2ms/step - loss: 112.7543 - val_loss: 111.
8021
Epoch 6/25
192/192 [==============================] - 0s 3ms/step - loss: 100.1540 - val_loss: 98.4
151
Epoch 7/25
192/192 [==============================] - 1s 3ms/step - loss: 88.3959 - val_loss: 86.20
34
Epoch 8/25
192/192 [==============================] - 0s 3ms/step - loss: 77.8604 - val_loss: 75.36
09
Epoch 9/25
192/192 [==============================] - 1s 3ms/step - loss: 67.8400 - val_loss: 65.58
31
Epoch 10/25
192/192 [==============================] - 1s 3ms/step - loss: 59.5815 - val_loss: 57.90
41
Epoch 11/25
192/192 [==============================] - 0s 3ms/step - loss: 52.9429 - val_loss: 50.76
07
Epoch 12/25
192/192 [==============================] - 0s 3ms/step - loss: 47.1470 - val_loss: 45.25
06
Epoch 13/25
192/192 [==============================] - 1s 3ms/step - loss: 42.9707 - val_loss: 41.66
32
Epoch 14/25
192/192 [==============================] - 1s 3ms/step - loss: 39.1026 - val_loss: 37.50
48
Epoch 15/25
192/192 [==============================] - 1s 4ms/step - loss: 36.5159 - val_loss: 34.83
86
Epoch 16/25
192/192 [==============================] - 1s 3ms/step - loss: 34.4723 - val_loss: 32.75
09
Epoch 17/25
192/192 [==============================] - 0s 2ms/step - loss: 32.8541 - val_loss: 31.59
13
Epoch 18/25
192/192 [==============================] - 0s 2ms/step - loss: 31.6036 - val_loss: 30.38
34
Epoch 19/25
192/192 [==============================] - 0s 3ms/step - loss: 31.4974 - val_loss: 29.50
35
Epoch 20/25
192/192 [==============================] - 0s 3ms/step - loss: 30.1345 - val_loss: 28.41
50
Epoch 21/25
192/192 [==============================] - 0s 2ms/step - loss: 30.3752 - val_loss: 30.68
11
Epoch 22/25
```

```
192/192 [==============================] - 0s 2ms/step - loss: 29.7382 - val_loss: 28.26
50
Epoch 23/25
192/192 [==============================] - 0s 2ms/step - loss: 29.5514 - val_loss: 27.88
38
Epoch 24/25
192/192 [==============================] - 0s 3ms/step - loss: 28.8701 - val_loss: 27.10
42
Epoch 25/25
192/192 [==============================] - 1s 3ms/step - loss: 28.9305 - val_loss: 26.62
37
```

In [103…
```
#Predicting the results of the Test set
y_pred = ann.predict(X_test)
```

```
60/60 [==============================] - 0s 2ms/step
```

In [104…
```
#calculating accuracy
from sklearn.metrics import r2_score

Accuracy=r2_score(y_test,y_pred)
```

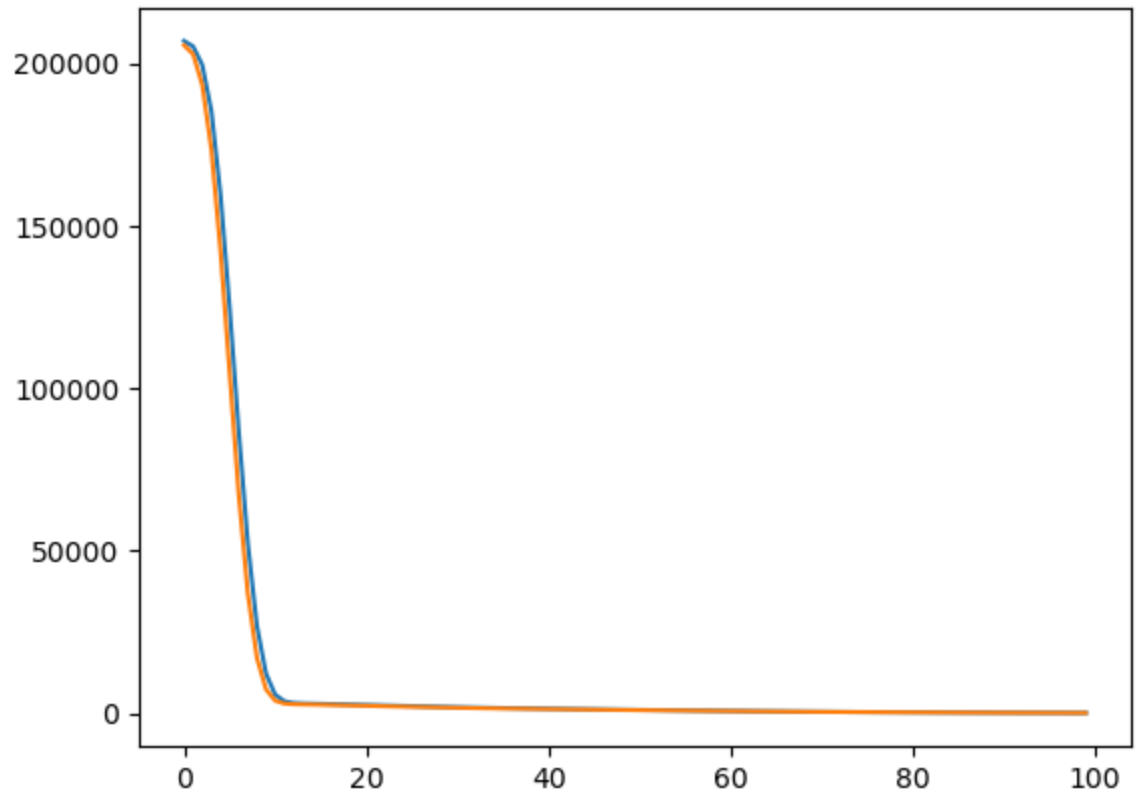
In [105…
```
Accuracy
```

Out[105]:
```
0.9051268626316156
```

In [106…
```
import matplotlib.pyplot as plt
```

In [107…
```
#plotting training and validation loss

plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.show()
```



Accuracy using Linear Regression

In [108…
```
# importing requered libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [109…
```
data = pd.read_excel('Folds5x2_pp.xlsx')
```

In [110…
```
data
```

Out[110]:

	AT	V	AP	RH	PE
0	14.96	41.76	1024.07	73.17	463.26
1	25.18	62.96	1020.04	59.08	444.37
2	5.11	39.40	1012.16	92.14	488.56
3	20.86	57.32	1010.24	76.64	446.48
4	10.82	37.50	1009.23	96.62	473.90
...	...	...	...	...	...
9563	16.65	49.69	1014.01	91.00	460.03
9564	13.19	39.18	1023.67	66.78	469.62
9565	31.32	74.33	1012.92	36.48	429.57
9566	24.48	69.45	1013.86	62.39	435.74
9567	21.60	62.52	1017.23	67.87	453.28

9568 rows × 5 columns

In [111…
```
data.isnull().sum()
```

Out[111]:
```
AT    0
V     0
AP    0
RH    0
PE    0
dtype: int64
```

In [112…
```
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
```

In [113…
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.2)


X_train.shape, y_train.shape,X_test.shape,y_test.shape
```

Out[113]:
```
((7654, 4), (7654,), (1914, 4), (1914,))
```

In [114…
```
#creating model using linear regression algorithm
from sklearn.linear_model import LinearRegression
```

```
from sklearn.metrics import mean_squared_error

lin_model = LinearRegression()
lin_model.fit(X_train, y_train)
```

Out[114]:

```
▼ LinearRegression
LinearRegression()
```

In [115…

```
#predicting and calculating the accuracy
from sklearn.metrics import r2_score

y_test_predict = lin_model.predict(X_test)

b=r2_score(y_test,y_test_predict)
```

In [116…

```
b
```

Out[116]:

```
0.9235606731016073
```

Accuracy score

ANN: 90.51% ~ 91%

Linear Regression: 92.35% ~ 92%

DATASET

https://www.kaggle.com/datasets/gauravduttakiit/combined-cycle-power-plant

Classification

Accuracy using ANN

In [70]:
```
# importing requered libraries
import numpy as np
import pandas as pd
```

In [71]:
```
#reading the dataset into a variable
data = pd.read_csv("heart.csv")
```

In [72]:
```
data.shape
```

Out[72]:
```
(303, 14)
```

In [73]:
```
data.head()
```

Out[73]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
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

In [74]:
```
#checking the data information
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   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [75]:
```
#Cheaking for duplicate values
data.duplicated().sum()
```

Out[75]:
```
1
```

In [76]:
```
#Removing duplicate values
print(data[~data.duplicated()])
```

```
     age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  \
0     63    1   3     145   233    1        0       150     0      2.3    0
1     37    1   2     130   250    0        1       187     0      3.5    0
2     41    0   1     130   204    0        0       172     0      1.4    2
3     56    1   1     120   236    0        1       178     0      0.8    2
4     57    0   0     120   354    0        1       163     1      0.6    2
..   ...  ...  ..     ...   ...  ...      ...       ...   ...      ...  ...
298   57    0   0     140   241    0        1       123     1      0.2    1
299   45    1   3     110   264    0        1       132     0      1.2    1
300   68    1   0     144   193    1        1       141     0      3.4    1
301   57    1   0     130   131    0        1       115     1      1.2    1
302   57    0   1     130   236    0        0       174     0      0.0    1

     caa  thall  output
0      0      1       1
1      0      2       1
2      0      2       1
3      0      2       1
4      0      2       1
..   ...    ...     ...
298    0      3       0
299    0      3       0
300    2      3       0
301    1      3       0
302    1      2       0

[302 rows x 14 columns]
```

In [77]:
```
#removing the unnecessary coloumns
data.drop(columns=['slp','caa','thall'],inplace=True)
```

In [78]:
```
#spliting the data into output
x = data.drop(columns=['output'])
y = data['output']
```

In [79]:
```
from sklearn.model_selection import train_test_split
# Separating the data into train and test.
train_x,test_x,train_y,test_y = train_test_split(x,y,test_size=0.2,random_state=108)
```

In [80]:
```
train_x.shape,test_x.shape,train_y.shape,test_y.shape
```

Out[80]:
```
((242, 10), (61, 10), (242,), (61,))
```

In [81]:
```
from sklearn.preprocessing import StandardScaler

scal = StandardScaler()

train_x_scal = scal.fit_transform(train_x)
test_x_scal = scal.fit_transform(test_x)
```

In [82]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

In [83]:
```
#creating the model
model = Sequential()
```

```
model.add(Dense(5,activation='sigmoid',input_dim=10))
#hidden layer
model.add(Dense(6, activation='relu'))
#output layer
model.add(Dense(1,activation='relu'))
```

In [84]:
```
#printing the summary for model
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 5)                 55

 dense_19 (Dense)            (None, 6)                 36

 dense_20 (Dense)            (None, 1)                 7

=================================================================
Total params: 98 (392.00 Byte)
Trainable params: 98 (392.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [85]:
```
#binary_crossentropy or can say log loss
model.compile(loss='binary_crossentropy',optimizer='Adam')
```

In [86]:
```
hist = model.fit(train_x_scal, train_y,epochs=25)
```

```
Epoch 1/25
8/8 [==============================] - 1s 4ms/step - loss: 2.3175
Epoch 2/25
8/8 [==============================] - 0s 3ms/step - loss: 1.8992
Epoch 3/25
8/8 [==============================] - 0s 3ms/step - loss: 1.6756
Epoch 4/25
8/8 [==============================] - 0s 5ms/step - loss: 1.4593
Epoch 5/25
8/8 [==============================] - 0s 6ms/step - loss: 1.3454
Epoch 6/25
8/8 [==============================] - 0s 6ms/step - loss: 1.2260
Epoch 7/25
8/8 [==============================] - 0s 4ms/step - loss: 1.1282
Epoch 8/25
8/8 [==============================] - 0s 5ms/step - loss: 1.0802
Epoch 9/25
8/8 [==============================] - 0s 4ms/step - loss: 1.0413
Epoch 10/25
8/8 [==============================] - 0s 7ms/step - loss: 1.0047
Epoch 11/25
8/8 [==============================] - 0s 6ms/step - loss: 0.9727
Epoch 12/25
8/8 [==============================] - 0s 6ms/step - loss: 0.9457
Epoch 13/25
8/8 [==============================] - 0s 7ms/step - loss: 0.9218
Epoch 14/25
8/8 [==============================] - 0s 13ms/step - loss: 0.8976
Epoch 15/25
8/8 [==============================] - 0s 11ms/step - loss: 0.8768
Epoch 16/25
8/8 [==============================] - 0s 8ms/step - loss: 0.8579
Epoch 17/25
8/8 [==============================] - 0s 6ms/step - loss: 0.8395
Epoch 18/25
8/8 [==============================] - 0s 7ms/step - loss: 0.8231
Epoch 19/25
8/8 [==============================] - 0s 6ms/step - loss: 0.7658
Epoch 20/25
8/8 [==============================] - 0s 5ms/step - loss: 0.6994
Epoch 21/25
8/8 [==============================] - 0s 3ms/step - loss: 0.6764
Epoch 22/25
8/8 [==============================] - 0s 7ms/step - loss: 0.6622
Epoch 23/25
8/8 [==============================] - 0s 6ms/step - loss: 0.6501
Epoch 24/25
8/8 [==============================] - 0s 6ms/step - loss: 0.6400
Epoch 25/25
8/8 [==============================] - 0s 6ms/step - loss: 0.6304
```

In [88]:
```
y_pred = np.where(model.predict(test_x_scal)>0.5,1,0)
```

```
2/2 [==============================] - 0s 3ms/step
```

In [89]:
```
from sklearn.metrics import accuracy_score
#Claculating acuracy score
accuracy_score(test_y,y_pred)
```

Out[89]:
```
0.6885245901639344
```

Accuracy using Decision Tree

In [90]:
```
# importing data and requered libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

In [91]:
```
df = pd.read_csv("heart.csv")
```

In [92]:
```
df.head()
```

Out[92]:

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
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

In [93]:
```
df.shape
```

Out[93]:
```
(303, 14)
```

In [94]:
```
df.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   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [95]:
```
y=df.output.values
X=df.drop(columns="output",axis=1)
```

In [96]:
```
# Separating the data into train and test.(%20 Test set / %80 Train set
train_X,test_X,train_y,test_y=train_test_split(X,y,test_size=0.2,random_state=42,shuffle
```

In [97]:
```
# We fit our decision tree train set and generate a prediction with our test set
reeClassifier()
```

```
dtc.fit(train_X,train_y)
predictions=dtc.predict(test_X)
```

In [98]:
```
# Calculating accuracy score
print("Accuracy Score :",accuracy_score(test_y,predictions))
print("Classification Report \n",classification_report(test_y,predictions))
```

```
Accuracy Score : 0.6557377049180327
Classification Report
               precision    recall  f1-score   support

           0       0.64      0.57      0.60        28
           1       0.67      0.73      0.70        33

    accuracy                           0.66        61
   macro avg       0.65      0.65      0.65        61
weighted avg       0.65      0.66      0.65        61
```

Accuracy score

ANN: 68.85% ~ 69%

Decision Tree: 65.57% ~ 66%

DATASET

https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset