NEELIMA_B180632CS

April 30, 2021

1 CS 4038D Data Mining - Assignment

Dataset - Pima Indians Diabetes Database (https://www.kaggle.com/datasets)

DESCRIPTION - The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. In particular, all patients here are females, at least 21 years old of *Pima Indian heritage*. The datasets consist of several medical predictor (independent) variables and one target (dependent) variable, Outcome.

PROBLEM STATEMENT - To accurately predict whether or not the patients in the dataset have diabetes

No. of samples in dataset - 768

No. of classes - 2 [0,1]

COLUMNS -

1) Pregnancies - Number of times pregnant
2) Glucose - Plasma glucose concentration
3) BloodPressure - Diastolic blood pressure (mm Hg)
4) SkinThickness - Triceps skin fold thickness (mm)
5) Insulin - 2-Hour serum insulin (mu U/ml)
6) BMI - Body mass index
7) DiabetesPedigreeFunction - Diabetes pedigree function
8) Age - Age (years)
9) Outcome - Class variable (0 or 1) - (Target variable) 268 of 768 are 1 (diabetic) , the others are 0 (not diabetic)

IMPORTING MODULES AND DATASET

[59]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings('ignore')
```

```
%matplotlib inline
```

[35]:
```
db = pd.read_csv('diabetes.csv',sep=',')
```

[36]:
```
db.head()
```

[36]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[37]:
```
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[38]:
```
db.describe()
```

[38]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
```

```
75%       6.000000  140.250000     80.000000     32.000000  127.250000
max      17.000000  199.000000    122.000000     99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Data contains '0' values. A value of zero does not make sense for these attributes and thus indicates missing value. Hence it needs to be cleaned. The '0' values are replaced with the mean value of the column.

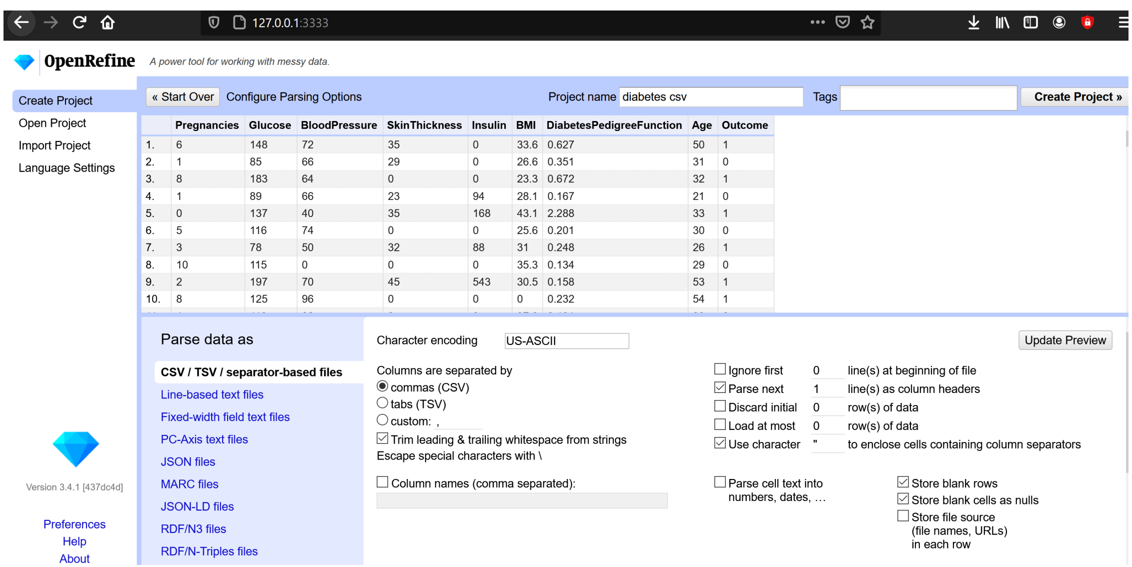
1.0.1 DATA CLEANING

DATA CLEANING IN OPENREFINE -

Firstly we import the data set and create a project

[49]:
```python
from IPython.display import Image
Image(filename='0_openrefine.png')
```

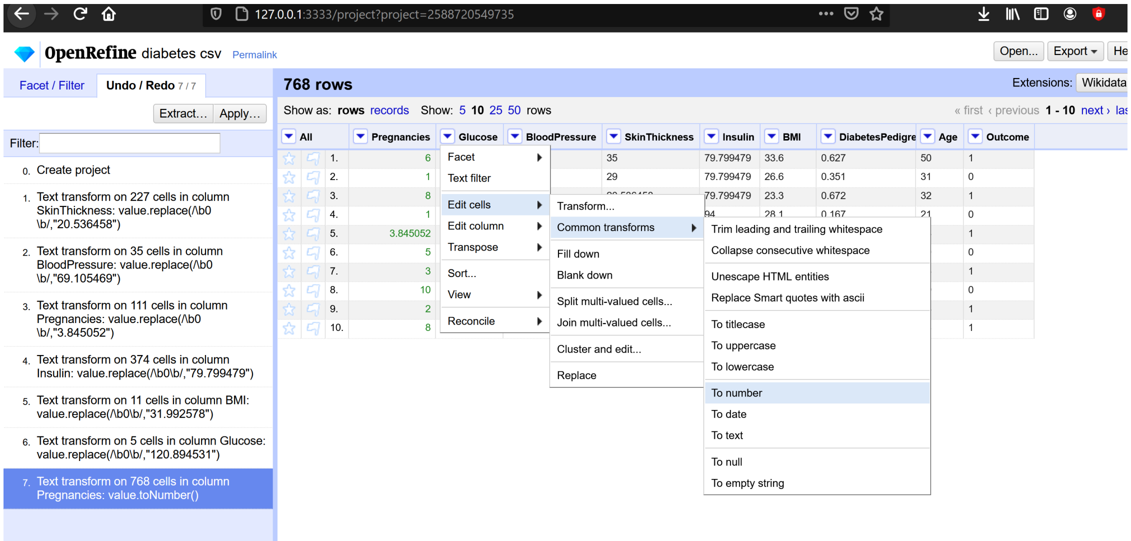
[49]:



Now we convert all values into numeric values

[50]:
```python
Image(filename='1_Convert_to_numeric.png')
```

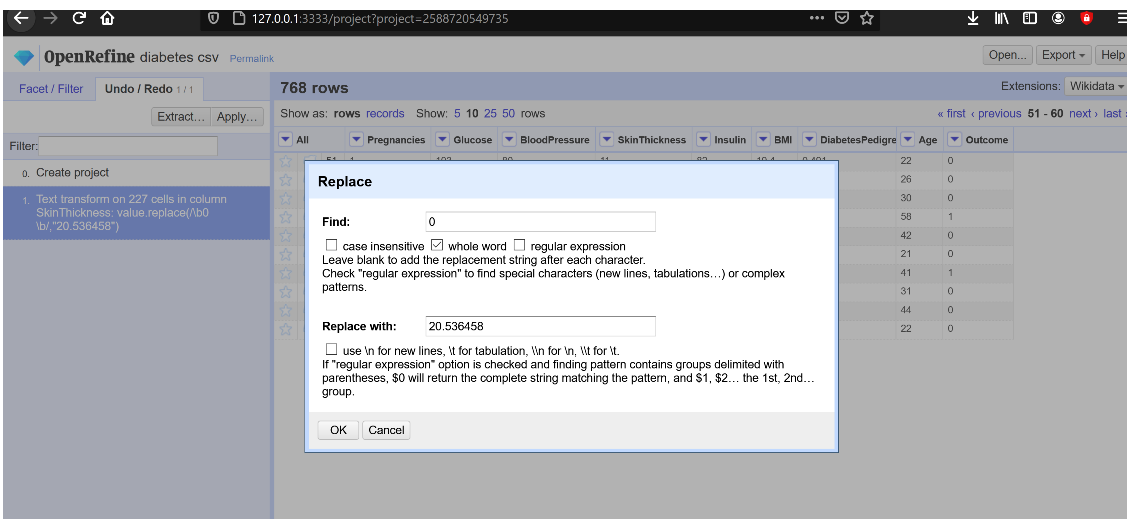
[50]:



Finally we replace the '0' values in the columns with the mean of the column

[48]:
```python
Image(filename='2_Removing_null.png')
```

[48]:



[39]:
```python
#Importing dataset after cleaning using OpenRefine
db = pd.read_csv('diabetes2.csv',sep=',')
```

[40]:
```python
db.head()
```

[40]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness    Insulin   BMI  \
0     6.000000    148.0           72.0      35.000000  79.799479  33.6
```

```
1     1.000000     85.0           66.0      29.000000    79.799479   26.6
2     8.000000    183.0           64.0      20.536458    79.799479   23.3
3     1.000000     89.0           66.0      23.000000    94.000000   28.1
4     3.845052    137.0           40.0      35.000000   168.000000   43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[41]:
```
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    float64
 1   Glucose                   768 non-null    float64
 2   BloodPressure             768 non-null    float64
 3   SkinThickness             768 non-null    float64
 4   Insulin                   768 non-null    float64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(7), int64(2)
memory usage: 54.1 KB
```

[42]:
```
db.describe()
```

[42]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      4.400782  121.681605      72.254807      26.606479  118.660163
std       2.984162   30.436016      12.115932       9.631241   93.080358
min       1.000000   44.000000      24.000000       7.000000   14.000000
25%       2.000000   99.750000      64.000000      20.536458   79.799479
50%       3.845052  117.000000      72.000000      23.000000   79.799479
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    32.450805                  0.471876   33.240885    0.348958
std      6.875374                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
```

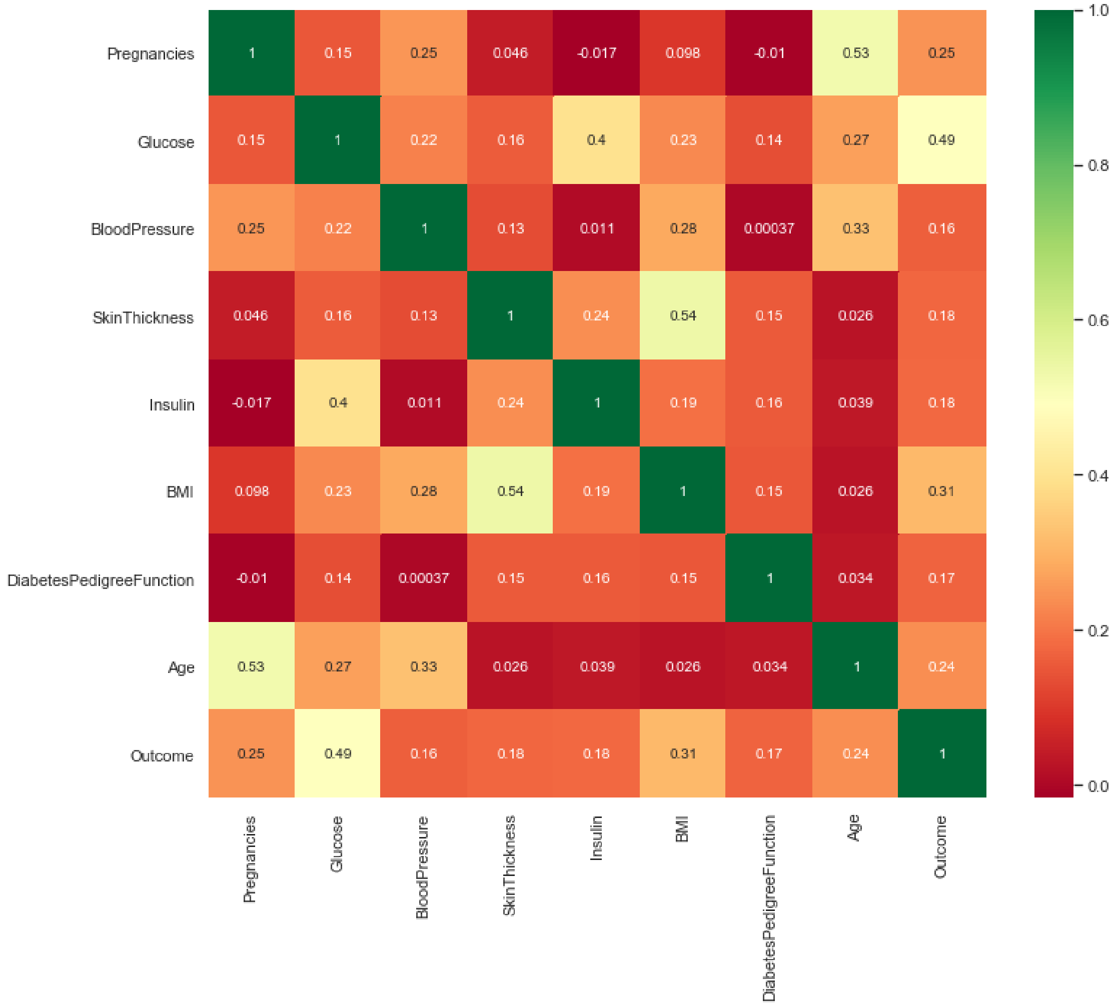
25%	27.500000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

```
Cleaning of data complete.
```

1.0.2 CORRELATION ANALYSIS

Generating a heatmap showing the correlation of the different columns

[52]:
```
plt.figure(figsize=(12,10))
p=sns.heatmap(db.corr(), annot=True,cmap ='RdYlGn')
```



INFERENCE - Here we see that all the variables have significant correlation to the 'outcome' and hence we do not need to drop any column before prediction

1.0.3 SPLITTING DATASET INTO TESTING AND TRAINING DATA

[70]:
```
#Import functions for Model, Dataset Splitting and Evaluation
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import linear_model
```

[556]:
```
X=db[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigre
 ↪values
y=db['Outcome']
```

Creating the training and test sets using 0.2 as test size (i.e 80% of data for training rest 20% for model testing)

[557]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

1.1 QUESTION 1 - Running the different models on the dataset

1.2 DECISION TREE ALGORITHM

[558]:
```
#creating classifier object for decision tree
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
```

[559]:
```
#predicting
y_pred = clf.predict(X_test)
```

[560]:
```
print("Accuracy on Test Set:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy on Test Set: 0.7662337662337663
```

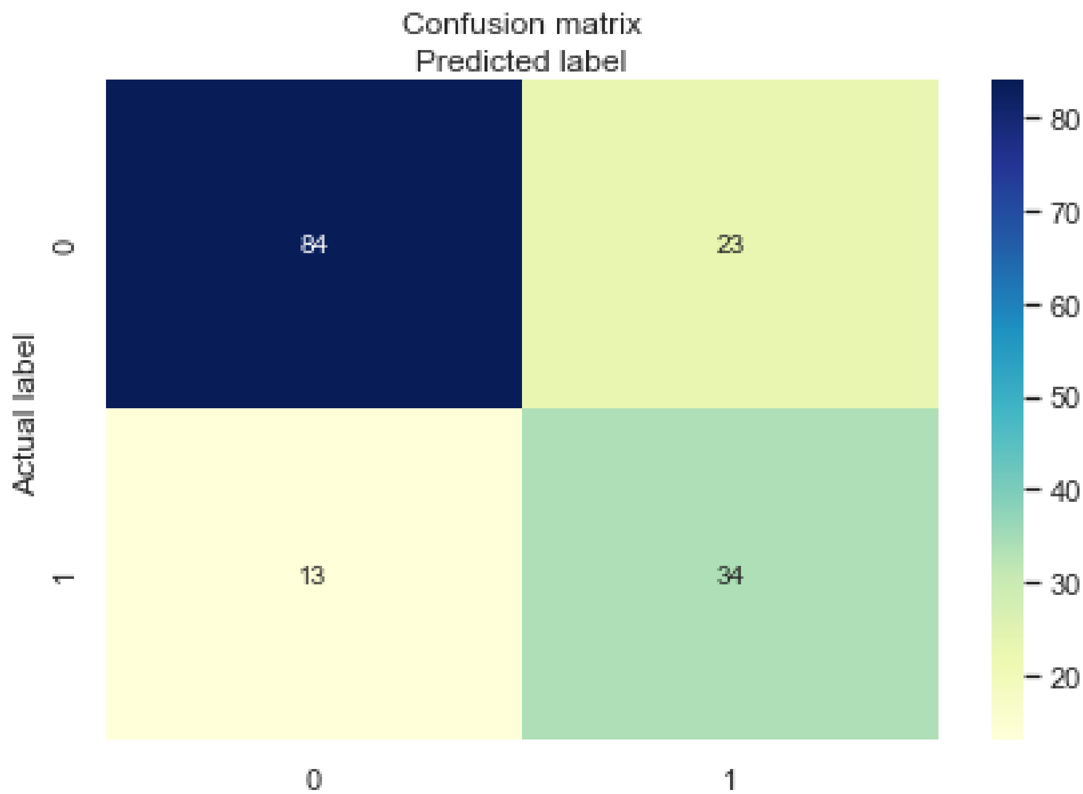
[561]:
```
# The confusion Matrix of the Model
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

[561]:
```
array([[84, 23],
       [13, 34]], dtype=int64)
```

[562]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
```

```python
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[562]: Text(0.5, 257.44, 'Predicted label')



Displaying a comprehensive Report of the Decision Tree Model On overall Dataset

[563]:
```python
print(metrics.classification_report(y_test, clf.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.87      0.79      0.82       107
           1       0.60      0.72      0.65        47

    accuracy                           0.77       154
   macro avg       0.73      0.75      0.74       154
weighted avg       0.78      0.77      0.77       154
```

1.3 NAIVE-BAYES CLASSIFIER MODEL

[564]:
```
from sklearn.naive_bayes import GaussianNB
```

[565]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[566]:
```
#creating classifier object
nb = GaussianNB()
nb.fit(X_train, y_train)

#predicting
nb_pred = nb.predict(X_test)
```

[567]:
```
print("Accuracy on Test Set:",metrics.accuracy_score(y_test, nb_pred))
```

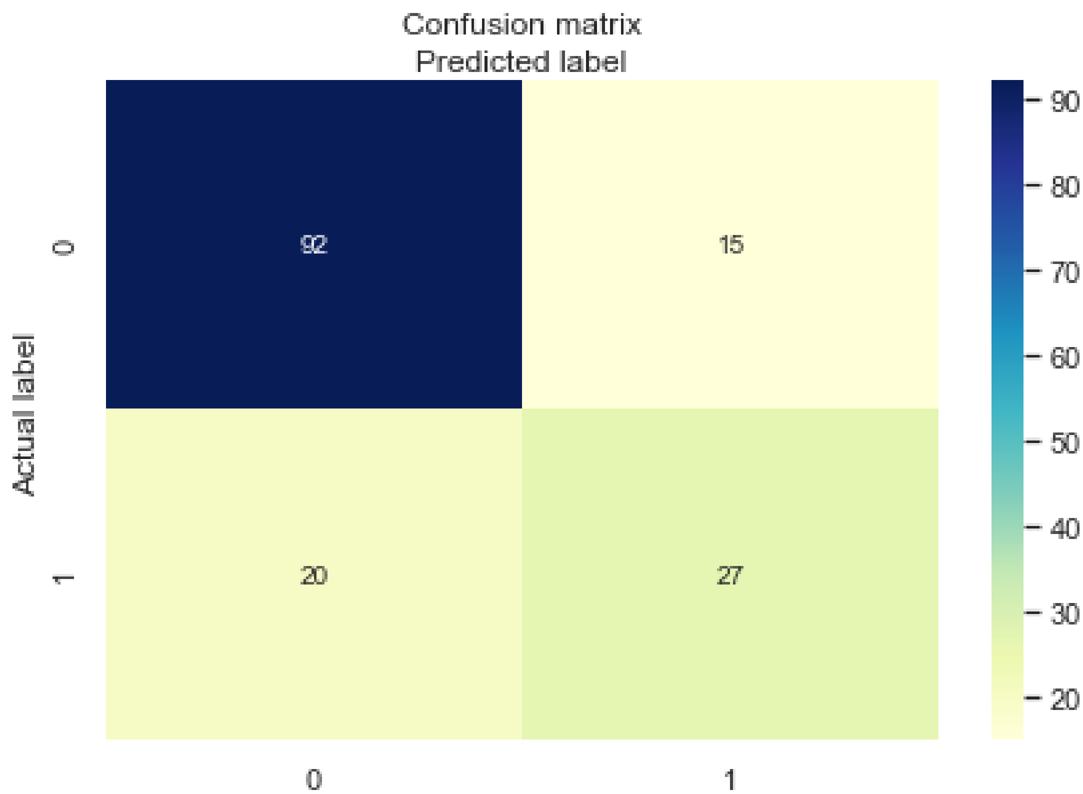
```
Accuracy on Test Set: 0.7727272727272727
```

[568]:
```
# The confusion Matrix of the Model
cnf_matrix = metrics.confusion_matrix(y_test, nb_pred)
cnf_matrix
```

[568]:
```
array([[92, 15],
       [20, 27]], dtype=int64)
```

[569]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[569]:
```
Text(0.5, 257.44, 'Predicted label')
```



Displaying a comprehensive Report of the Naive Bayes Model On overall Dataset

[570]:
```
print(metrics.classification_report(y_test, nb.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84       107
           1       0.64      0.57      0.61        47

    accuracy                           0.77       154
   macro avg       0.73      0.72      0.72       154
weighted avg       0.77      0.77      0.77       154
```

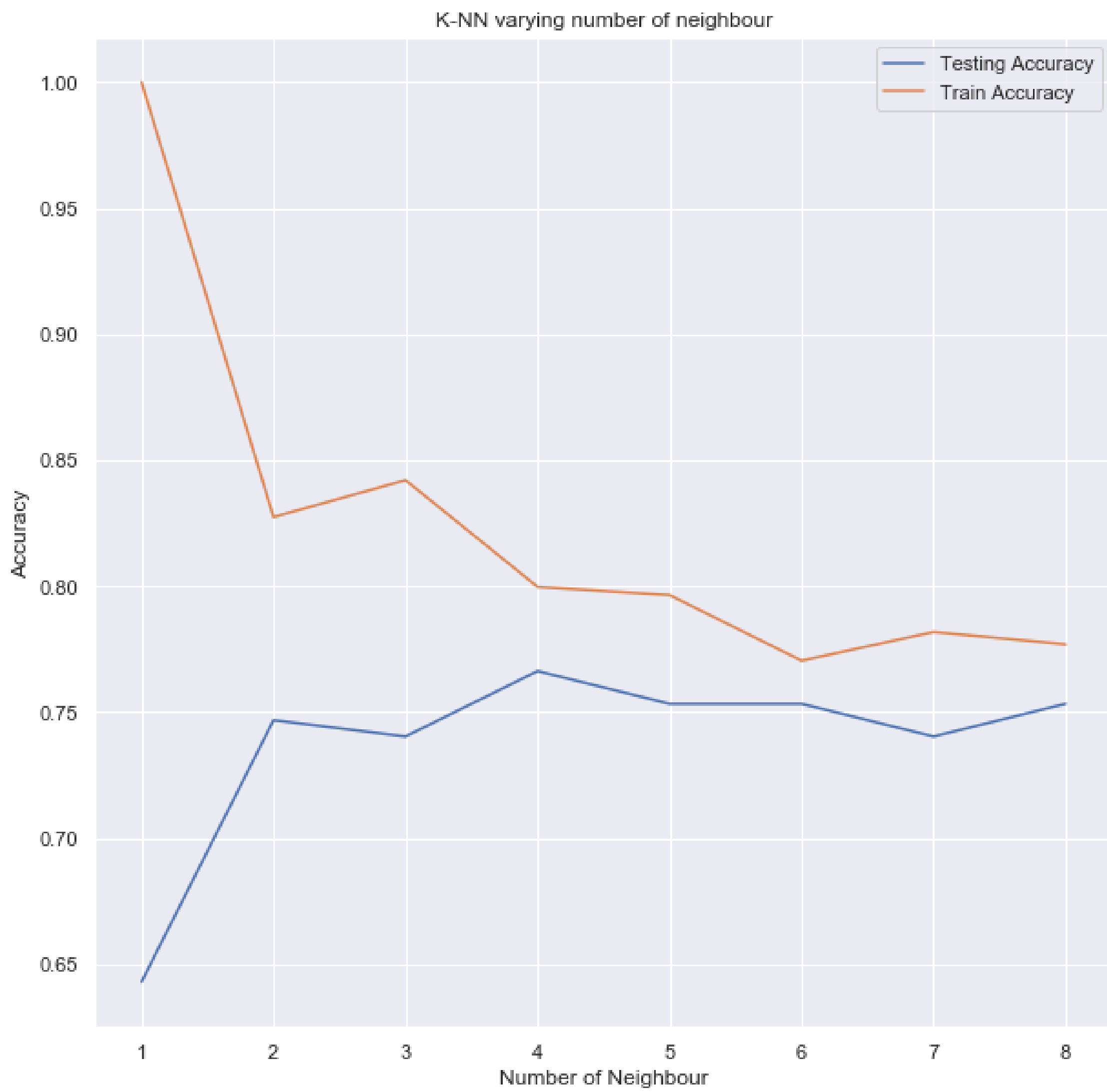
1.4 KNN MODEL

[571]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[572]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[573]:
```
neighbors=np.arange(1,9)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    knn=KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train, y_train)

    train_accuracy[i]=knn.score(X_train, y_train)

    test_accuracy[i]=knn.score(X_test, y_test)
```

[574]:
```
plt.figure(figsize=(10,10))
plt.title("K-NN varying number of neighbour")
plt.plot(neighbors, test_accuracy, label="Testing Accuracy")
plt.plot(neighbors, train_accuracy, label="Train Accuracy")
plt.legend()
plt.xlabel("Number of Neighbour")
plt.ylabel("Accuracy")
plt.show()
```



As test accuracy is highest at k=4, We adopt the KNeighborsClassifier with number of neighbours as 4

[575]:
```
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(X_train, y_train)
```

[575]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

[576]:
```
K_pred=knn.predict(X_test)
print("Accuracy on Test Set:",metrics.accuracy_score(y_test, K_pred))
```

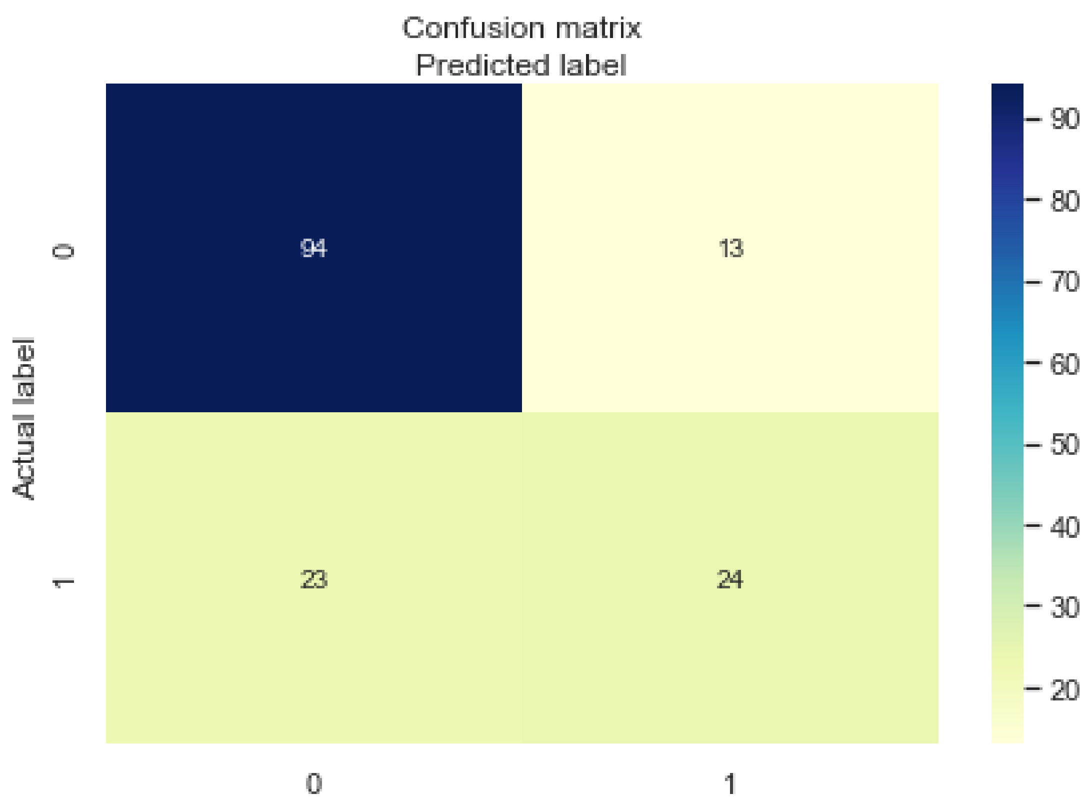
```
Accuracy on Test Set: 0.7662337662337663
```

[577]:
```
cnf_matrix = metrics.confusion_matrix(y_test, K_pred)
cnf_matrix
```

[577]:
```
array([[94, 13],
       [23, 24]], dtype=int64)
```

[578]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[578]:
```
Text(0.5, 257.44, 'Predicted label')
```



[579]:
```
print(metrics.classification_report(y_test, knn.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84       107
           1       0.65      0.51      0.57        47

    accuracy                           0.77       154
   macro avg       0.73      0.69      0.71       154
weighted avg       0.76      0.77      0.76       154
```

1.5 ANN MODEL

[117]:
```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
```

[134]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

[157]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[158]:
```
# compiling the keras model

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

[361]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[362]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 78.65
```

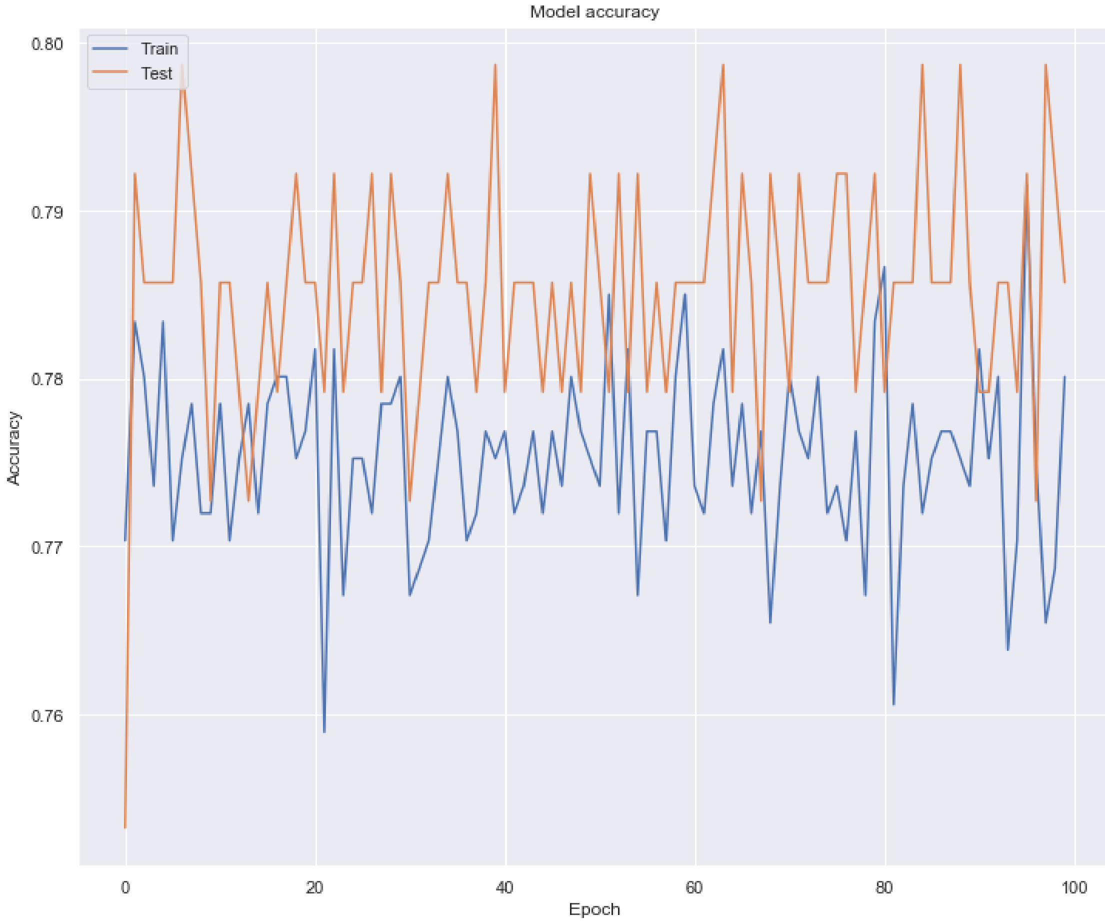
[166]:
```
# make class predictions with the model
predictions = model.predict_classes(X)
# summarize the first 5 cases
for i in range(10):
        print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], y[i]))
```

```
[6.0, 148.0, 72.0, 35.0, 79.799479, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 79.799479, 26.6, 0.35100000000000003, 31.0] => 0
```

```
(expected 0)
[8.0, 183.0, 64.0, 20.536458, 79.799479, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.16699999999999998, 21.0] => 0 (expected 0)
[3.8450519999999995, 137.0, 40.0, 35.0, 168.0, 43.1, 2.2880000000000003, 33.0]
=> 1 (expected 1)
[5.0, 116.0, 74.0, 20.536458, 79.799479, 25.6, 0.201, 30.0] => 0 (expected 0)
[3.0, 78.0, 50.0, 32.0, 88.0, 31.0, 0.248, 26.0] => 0 (expected 1)
[10.0, 115.0, 69.105469, 20.536458, 79.799479, 35.3, 0.134, 29.0] => 1 (expected
0)
[2.0, 197.0, 70.0, 45.0, 543.0, 30.5, 0.158, 53.0] => 1 (expected 1)
[8.0, 125.0, 96.0, 20.536458, 79.799479, 31.992578, 0.23199999999999998, 54.0]
=> 0 (expected 1)
```

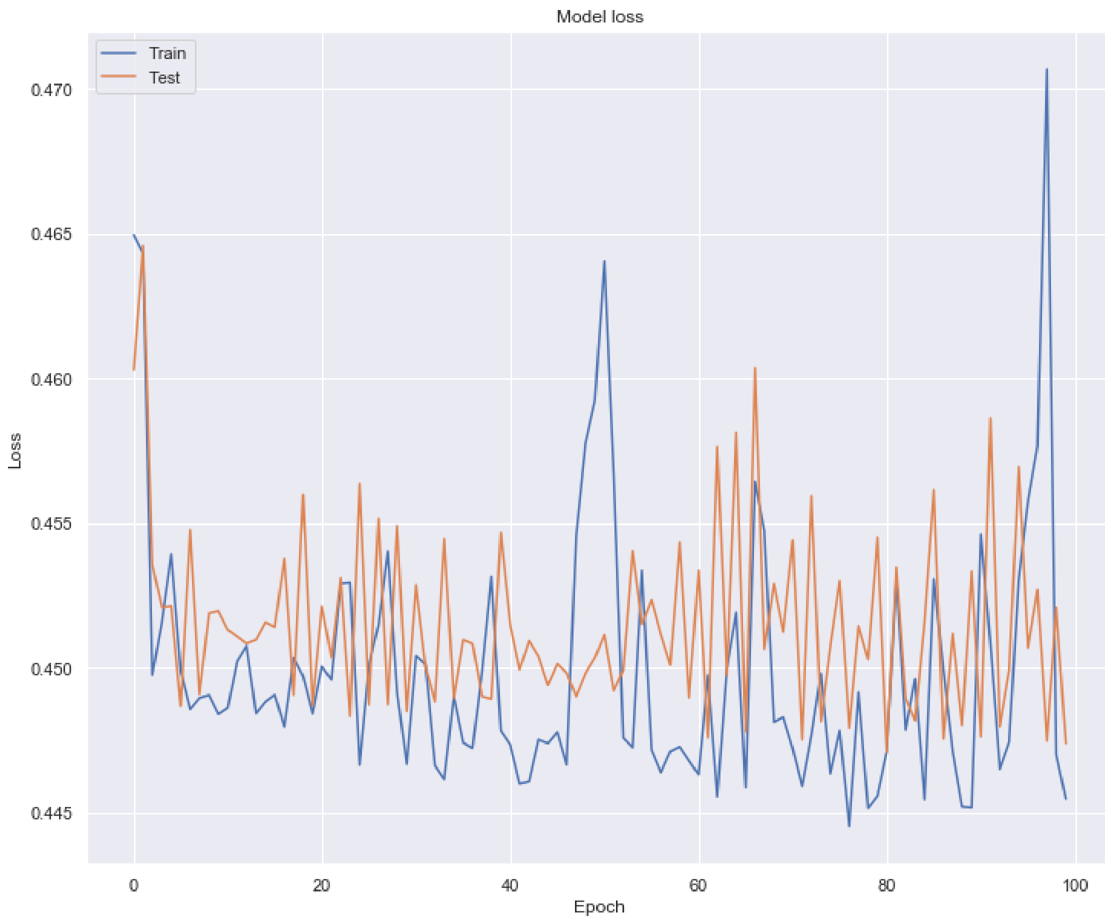
[ ]:
```
history = model.fit(X_train, y_train,validation_data = (X_test,y_test),␣
↪epochs=100, batch_size=64,verbose=0)
```

[197]:
```
#model accuracy
plt.figure(figsize=(12,10))
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'], loc='upper left')
plt.show()
```



1.6 QUESTION 2 - Plot loss function against epochs

[170]:
```
#model loss
plt.figure(figsize=(12,10))
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'], loc='upper left')
plt.show()
```



1.7 Comparing the performance of ANN with any three different activation functions.

1.7.1 RELU

[417]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[418]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(1, activation='relu'))
```

[419]:
```
# compiling the keras model

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

[420]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[421]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 65.10
```

1.7.2 TANH

[477]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[478]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='tanh'))
model.add(Dense(1, activation='tanh'))
```

[479]:
```
# compiling the keras model

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

[480]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[481]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 68.10
```

1.7.3 SIGMOID

[427]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[428]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

[429]:
```
# compiling the keras model

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

[430]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[431]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 68.36
```

INFERANCE - `ACCURACY for ANN using RELU activation function is 65.10`

`ACCURACY for ANN using TANH activation function is 68.10`

`ACCURACY for ANN using SIGMOID activation function is 68.36`

CONCLUSION - SIGMOID activation function gave most accuracy and RELU activation function gave least accuracy

1.8 QUESTION 3 - Comparing and plotting ANN MODEL accuracy for different numbers of hidden nodes, like 1, 2, 3, square root of number of features = 3 (rounded), and half of the number of features = 4.

1.8.1 No. of Hidden nodes = 1

[506]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[507]:
```
model = Sequential()
model.add(Dense(1, input_dim=8, activation='relu')) #1 HIDDEN NODE
model.add(Dense(1, activation='relu'))
```

[508]:
```
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[509]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[510]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 65.10
```

1.8.2 No. of Hidden nodes = 2

[467]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[468]:
```
model = Sequential()
model.add(Dense(2, input_dim=8, activation='relu'))
```

```
model.add(Dense(1, activation='relu'))
```

[469]:
```
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[470]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[471]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 63.54
```

1.8.3 No. of Hidden nodes = 3

[511]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[512]:
```
model = Sequential()
model.add(Dense(3, input_dim=8, activation='relu'))
#model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='relu'))
```

[513]:
```
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[514]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)
```

[515]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 64.84
```

1.8.4 No. of Hidden nodes = 4

[497]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=0)
```

[498]:
```
model = Sequential()
model.add(Dense(4, input_dim=8, activation='relu'))
model.add(Dense(1, activation='relu'))
```

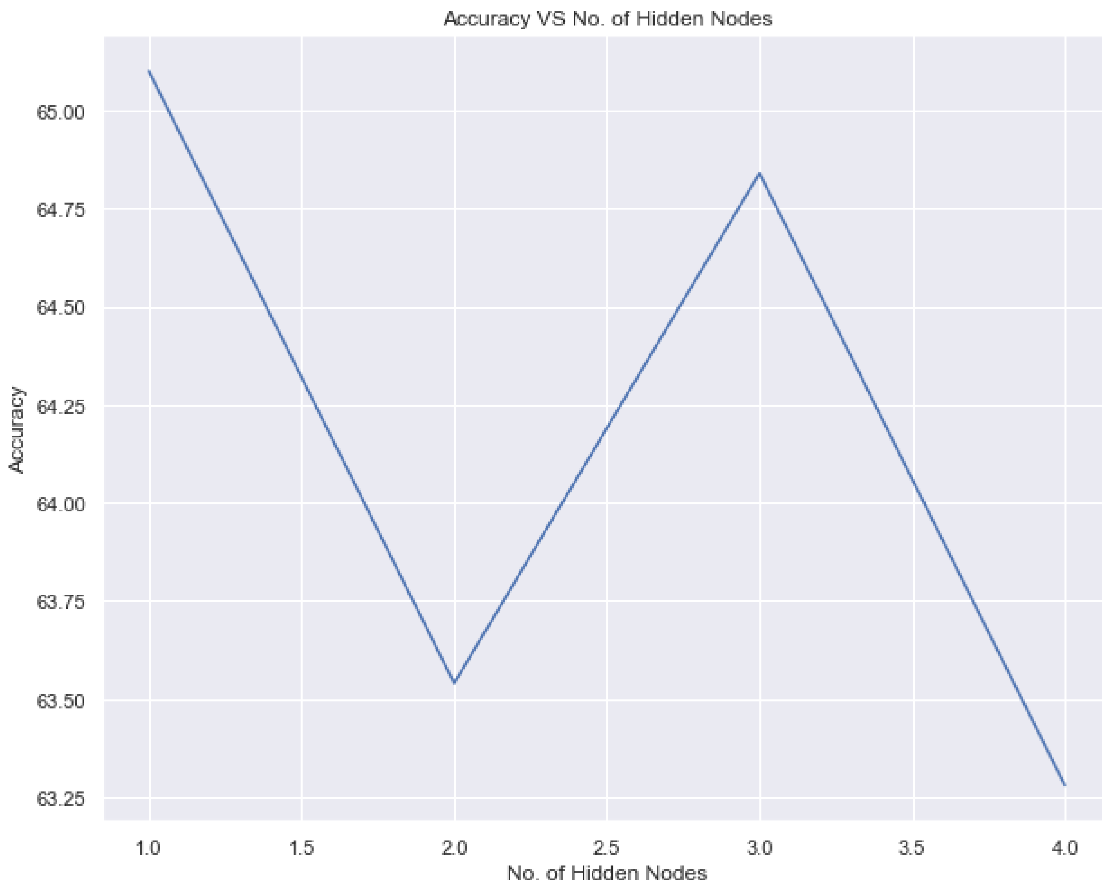
[499]:
```
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[500]:
```
# fit the keras model on the dataset
history = model.fit(X_train, y_train, epochs=150, batch_size=10,verbose=0)

# evaluating the keras model
_, accuracy = model.evaluate(X, y, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 63.28
```

1.8.5 Plotting the different accuracies for comparion

[581]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,8))
# x axis values
x = [1,2,3,4]
# corresponding y axis values
y = [65.10,63.54,64.84,63.28]

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('No. of Hidden Nodes')
# naming the y axis
plt.ylabel('Accuracy')

# giving a title to my graph
plt.title('Accuracy VS No. of Hidden Nodes')

# function to show the plot
plt.show()
```



No significant in

1.9 QUESTION 4 - APPLYING MODELS ON DIVIDED DATASETS D1, D2, D3

Data sets have been manually divided and will now be imported.

[433]:
```
#importing divided datasets
db1 = pd.read_csv('D1.csv',sep=',')
db2 = pd.read_csv('D2.csv',sep=',')
db3 = pd.read_csv('D3.csv',sep=',')
```

[435]:
```
db1.shape #D1
```

[435]: (192, 9)

[436]:
```
db2.shape #D2
```

[436]: (384, 9)

[437]:
```
db3.shape #D3
```

[437]: (576, 9)

[438]:
```
db.shape #D
```

[438]: (768, 9)

1.10 Decision tree

1.10.1 Splitting into training and testing

[600]:
```
X1=db1[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedig
 ↪values
y1=db1['Outcome']
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,␣
 ↪random_state=0)

X2=db2[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedig
 ↪values
y2=db2['Outcome']
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2,␣
 ↪random_state=0)

X3=db3[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedig
 ↪values
y3=db3['Outcome']
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2,␣
 ↪random_state=0)
```

[601]:
```
#creating classifier object for decision tree
clf1 = DecisionTreeClassifier()
clf1 = clf1.fit(X1_train,y1_train)

clf2 = DecisionTreeClassifier()
clf2 = clf2.fit(X2_train,y2_train)

clf3 = DecisionTreeClassifier()
clf3 = clf3.fit(X3_train,y3_train)
```

[602]:
```
y1_pred = clf1.predict(X1_test)
y2_pred = clf2.predict(X2_test)
y3_pred = clf3.predict(X3_test)
```

[603]:
```
# The confusion Matrix of the Model
cnf1_matrix = metrics.confusion_matrix(y1_test, y1_pred)
cnf1_matrix
```

[603]:
```
array([[19, 10],
       [ 4,  6]], dtype=int64)
```

[604]:
```
cnf2_matrix = metrics.confusion_matrix(y2_test, y2_pred)
cnf2_matrix
```

[604]:
```
array([[38, 19],
       [ 7, 13]], dtype=int64)
```

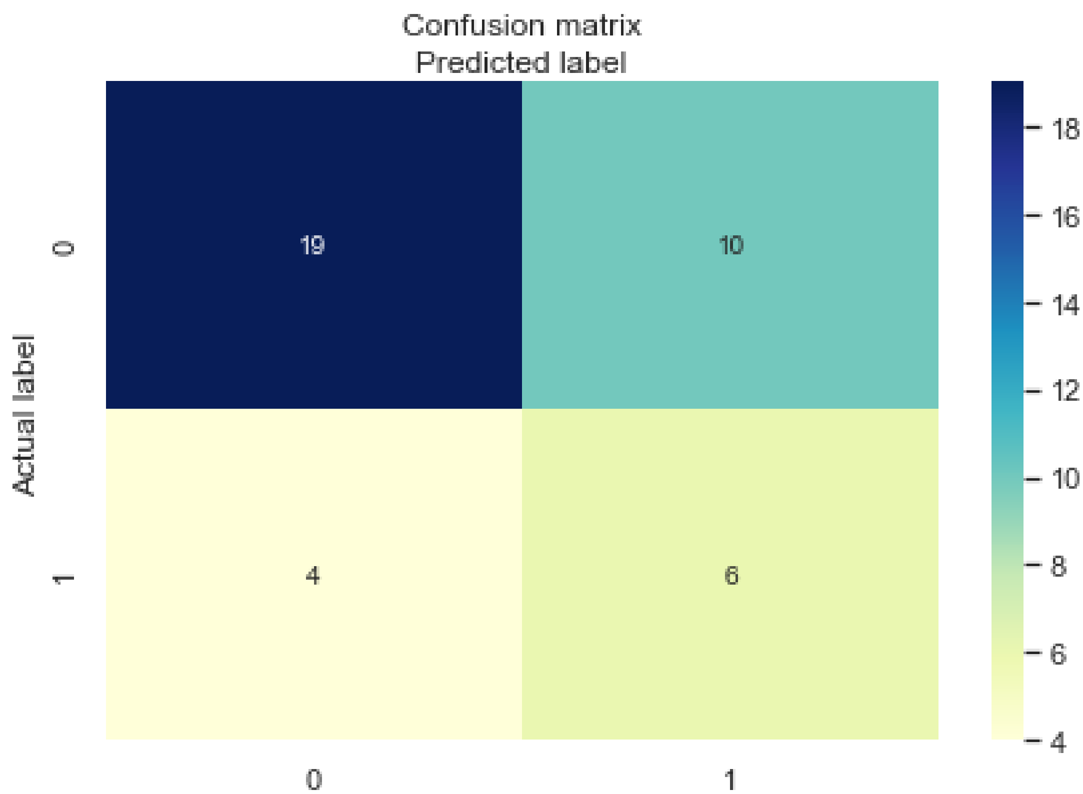
[605]:
```
cnf3_matrix = metrics.confusion_matrix(y3_test, y3_pred)
cnf3_matrix
```

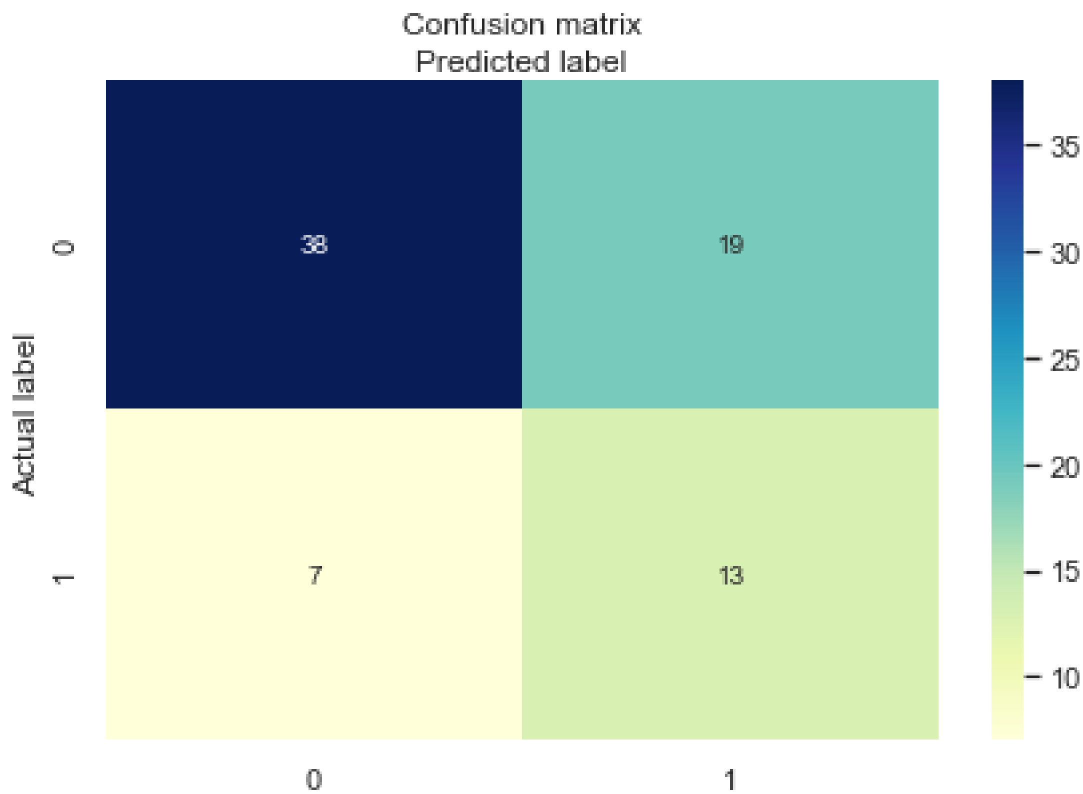
[605]:
```
array([[61, 15],
       [17, 23]], dtype=int64)
```

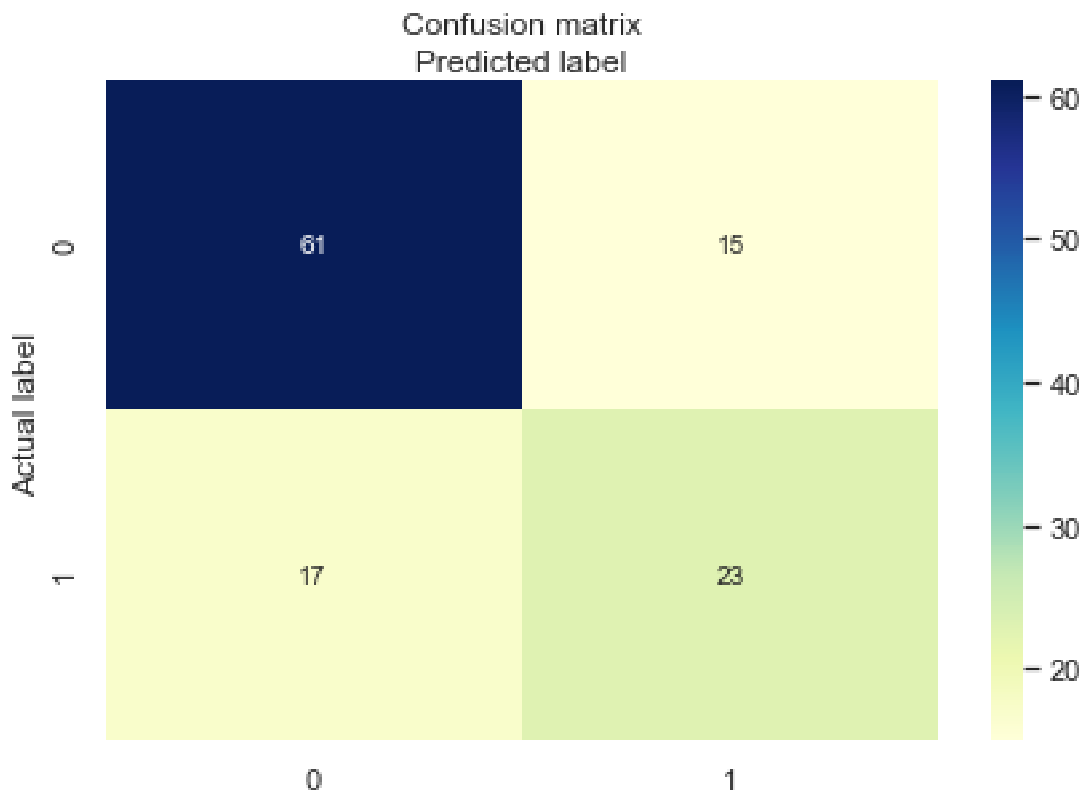
[606]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf1_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf2_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
```

```
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf3_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[606]: Text(0.5, 257.44, 'Predicted label')




Confusion matrix
Predicted label
Actual label
0
1
38
19
7
13
0
1
35
30
25
20
15
10



[609]:
```
print("FOR D1 - \n")
print(metrics.classification_report(y1_test, y1_pred))
print("FOR D2 - \n")
print(metrics.classification_report(y2_test, y2_pred))
print("FOR D3 - \n")
print(metrics.classification_report(y3_test, y3_pred))
```

```
FOR D1 -

              precision    recall  f1-score   support

           0       0.83      0.66      0.73        29
           1       0.38      0.60      0.46        10

    accuracy                           0.64        39
   macro avg       0.60      0.63      0.60        39
weighted avg       0.71      0.64      0.66        39

FOR D2 -

              precision    recall  f1-score   support
```

```
              0       0.84      0.67      0.75        57
              1       0.41      0.65      0.50        20

       accuracy                           0.66        77
      macro avg       0.63      0.66      0.62        77
   weighted avg       0.73      0.66      0.68        77
```

```
FOR D3 -
```

```
              precision    recall  f1-score   support

              0       0.78      0.80      0.79        76
              1       0.61      0.57      0.59        40

       accuracy                           0.72       116
      macro avg       0.69      0.69      0.69       116
   weighted avg       0.72      0.72      0.72       116
```

[610]:

```
print("Accuracy on Test Set1:",metrics.accuracy_score(y1_test, y1_pred))
print("Accuracy on Test Set2:",metrics.accuracy_score(y2_test, y2_pred))
print("Accuracy on Test Set3:",metrics.accuracy_score(y3_test, y3_pred))
print("Accuracy on Test Set for main data set: 0.7597402597402597")
```

```
Accuracy on Test Set1: 0.6410256410256411
Accuracy on Test Set2: 0.6623376623376623
Accuracy on Test Set3: 0.7241379310344828
Accuracy on Test Set for main data set: 0.7597402597402597
```

INFERENCE - With increase in number of samples , the accurancy of the model seems to increase. Irregularities and exceptions in this observation could be because of the uneven distribution of classes in the target variable of the data sets after splitting

1.11 Naive bayes

[611]:

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,␣
 ↪random_state=0)
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2,␣
 ↪random_state=0)
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2,␣
 ↪random_state=0)
```

[612]:

```
#creating classifier object
nb1 = GaussianNB()
nb1.fit(X1_train, y1_train)

#predicting
```

```
nb1_pred = nb1.predict(X1_test)

#creating classifier object
nb2 = GaussianNB()
nb2.fit(X2_train, y2_train)

#predicting
nb2_pred = nb2.predict(X2_test)

#creating classifier object
nb3 = GaussianNB()
nb3.fit(X3_train, y3_train)

#predicting
nb3_pred = nb3.predict(X3_test)
```

[613]:
```
# The confusion Matrix of the Model for D1
cnf1_matrix = metrics.confusion_matrix(y1_test, nb1_pred)
cnf1_matrix
```

[613]:
```
array([[26,  3],
       [ 5,  5]], dtype=int64)
```

[614]:
```
# The confusion Matrix of the Model for D2
cnf2_matrix = metrics.confusion_matrix(y2_test, nb2_pred)
cnf2_matrix
```

[614]:
```
array([[45, 12],
       [13,  7]], dtype=int64)
```

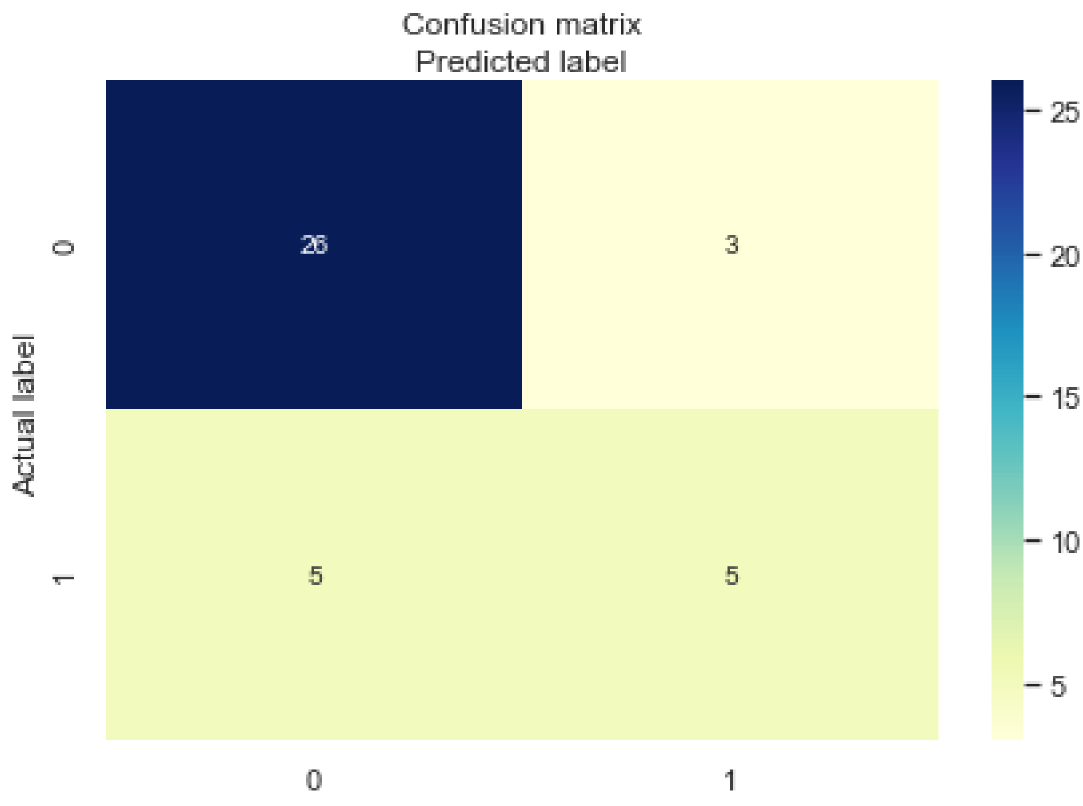
[615]:
```
# The confusion Matrix of the Model for D3
cnf3_matrix = metrics.confusion_matrix(y3_test, nb3_pred)
cnf3_matrix
```

[615]:
```
array([[67,  9],
       [18, 22]], dtype=int64)
```

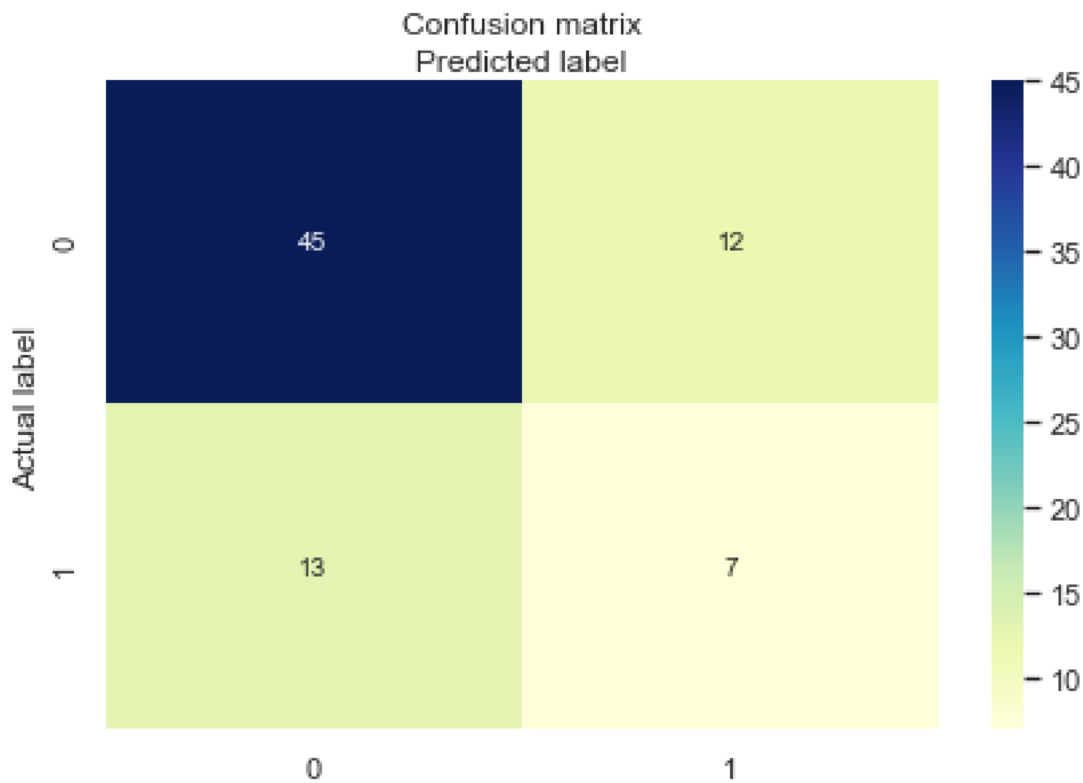
[616]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf1_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
```

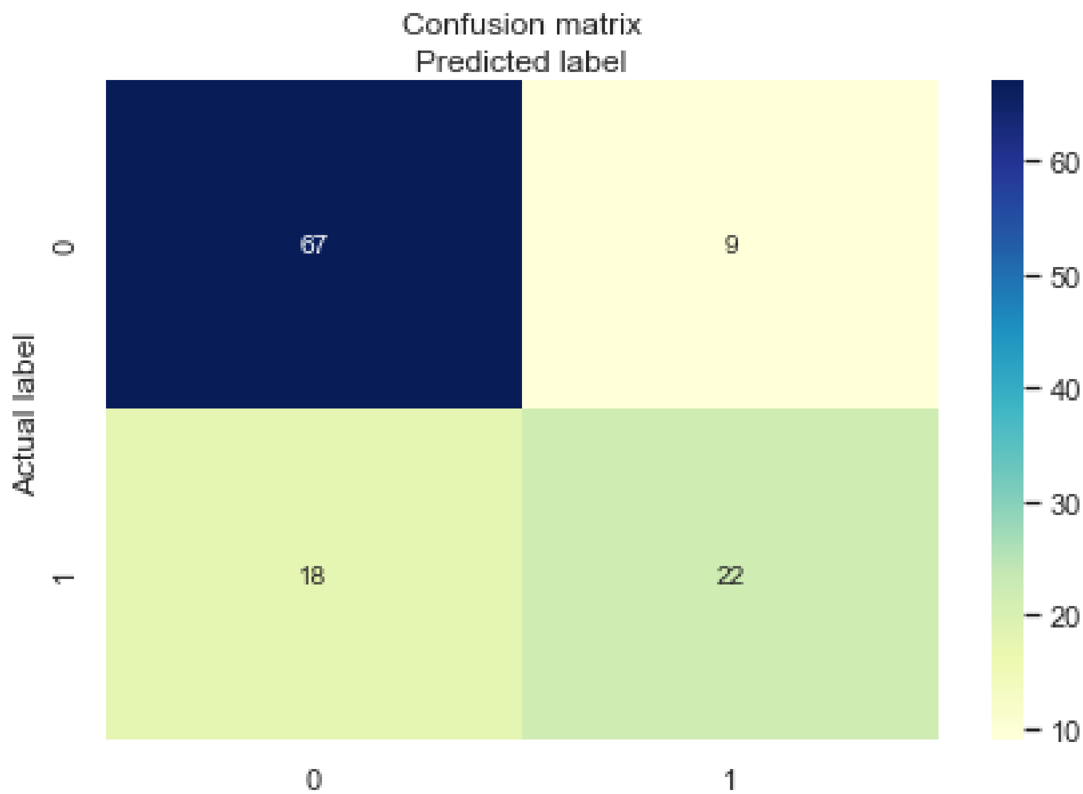
```python
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf2_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf3_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[616]:
```
Text(0.5, 257.44, 'Predicted label')
```



Confusion matrix
Predicted label
Actual label
0
1
26
3
5
5
0
1
25
20
15
10
5


Confusion matrix
Predicted label
Actual label
0
1
45
12
13
7
0
1
45
40
35
30
25
20
15
10



[617]:
```
print("FOR D1 - \n")
print(metrics.classification_report(y1_test, nb1.predict(X1_test)))
print("\nFOR D2 - \n")
print(metrics.classification_report(y2_test, nb2.predict(X2_test)))
print("\nFOR D3 - \n")
print(metrics.classification_report(y3_test, nb3.predict(X3_test)))
```

```
FOR D1 -

              precision    recall  f1-score   support

           0       0.84      0.90      0.87        29
           1       0.62      0.50      0.56        10

    accuracy                           0.79        39
   macro avg       0.73      0.70      0.71        39
weighted avg       0.78      0.79      0.79        39


FOR D2 -

              precision    recall  f1-score   support
```

```
              0       0.78      0.79      0.78        57
              1       0.37      0.35      0.36        20

       accuracy                           0.68        77
      macro avg       0.57      0.57      0.57        77
   weighted avg       0.67      0.68      0.67        77


FOR D3 -

              precision    recall  f1-score   support

              0       0.79      0.88      0.83        76
              1       0.71      0.55      0.62        40

       accuracy                           0.77       116
      macro avg       0.75      0.72      0.73       116
   weighted avg       0.76      0.77      0.76       116
```

[618]:
```
print("Accuracy on Test Set1:",metrics.accuracy_score(y1_test, nb1_pred))
print("Accuracy on Test Set2:",metrics.accuracy_score(y2_test, nb2_pred))
print("Accuracy on Test Set3:",metrics.accuracy_score(y3_test, nb3_pred))
print("Accuracy on Test Set for main data set: 0.7727272727272727")
```

```
Accuracy on Test Set1: 0.7948717948717948
Accuracy on Test Set2: 0.6753246753246753
Accuracy on Test Set3: 0.7672413793103449
Accuracy on Test Set for main data set: 0.7727272727272727
```

```
INFERANCE - For this model, all the accuracies are in the same range.  Only
dataset - D2 seems to be lesser than the range and this can be due to uneven
distribution of classes in the target variable of the training dataset upon
spliting the main dataset
```

1.12 KNN MODEL

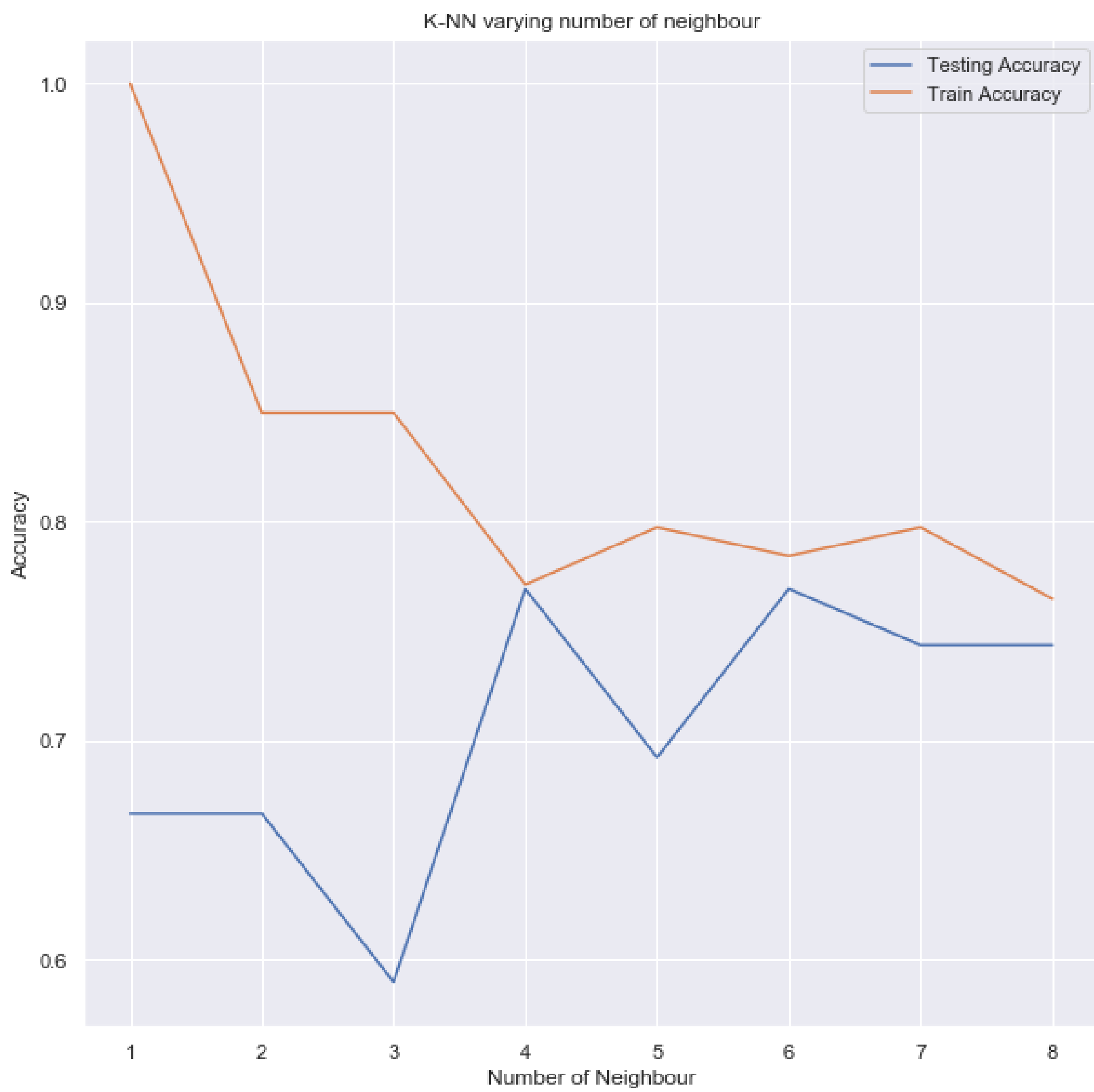
Applying on D1 -

[527]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,␣
 ↪random_state=0)
```

[528]:
```
neighbors=np.arange(1,9)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    knn1=KNeighborsClassifier(n_neighbors=k)
```

```
    knn1.fit(X1_train, y1_train)

    train_accuracy[i]=knn1.score(X1_train, y1_train)

    test_accuracy[i]=knn1.score(X1_test, y1_test)
```

[529]:
```
plt.figure(figsize=(10,10))
plt.title("K-NN varying number of neighbour")
plt.plot(neighbors, test_accuracy, label="Testing Accuracy")
plt.plot(neighbors, train_accuracy, label="Train Accuracy")
plt.legend()
plt.xlabel("Number of Neighbour")
plt.ylabel("Accuracy")
plt.show()
```



As test accracy is highest at k=4, We adopt the KNeighborsClassifier with number of neighbours as 4

[530]:
```
knn1=KNeighborsClassifier(n_neighbors=4)
knn1.fit(X1_train, y1_train)
```

[530]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                     weights='uniform')
```

[531]:
```
K1_pred=knn.predict(X1_test)
print("Accuracy on Test Set:",metrics.accuracy_score(y1_test, K1_pred))
```

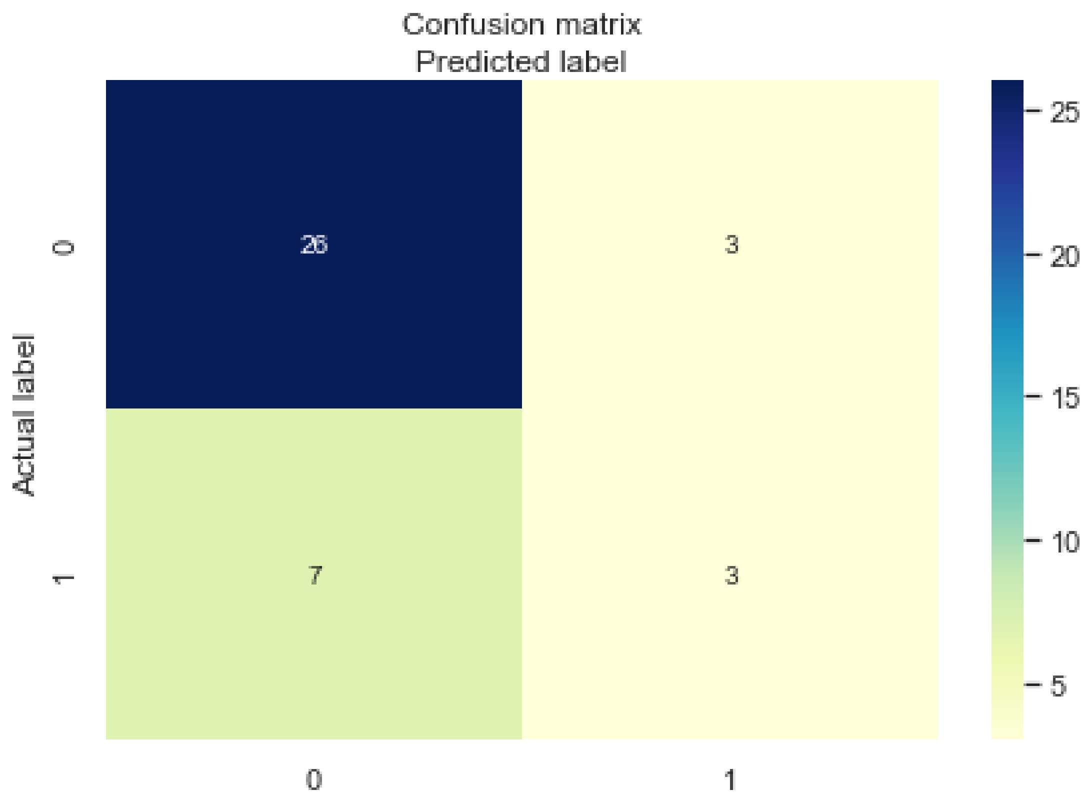
```
Accuracy on Test Set: 0.7435897435897436
```

[532]:
```
cnf1_matrix = metrics.confusion_matrix(y1_test, K1_pred)
cnf1_matrix
```

[532]:
```
array([[26,  3],
       [ 7,  3]], dtype=int64)
```

[533]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf1_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[533]:
```
Text(0.5, 257.44, 'Predicted label')
```



[535]:
```
print(metrics.classification_report(y1_test, knn1.predict(X1_test)))
```

```
              precision    recall  f1-score   support

           0       0.79      0.93      0.86        29
           1       0.60      0.30      0.40        10

    accuracy                           0.77        39
   macro avg       0.70      0.62      0.63        39
weighted avg       0.74      0.77      0.74        39
```

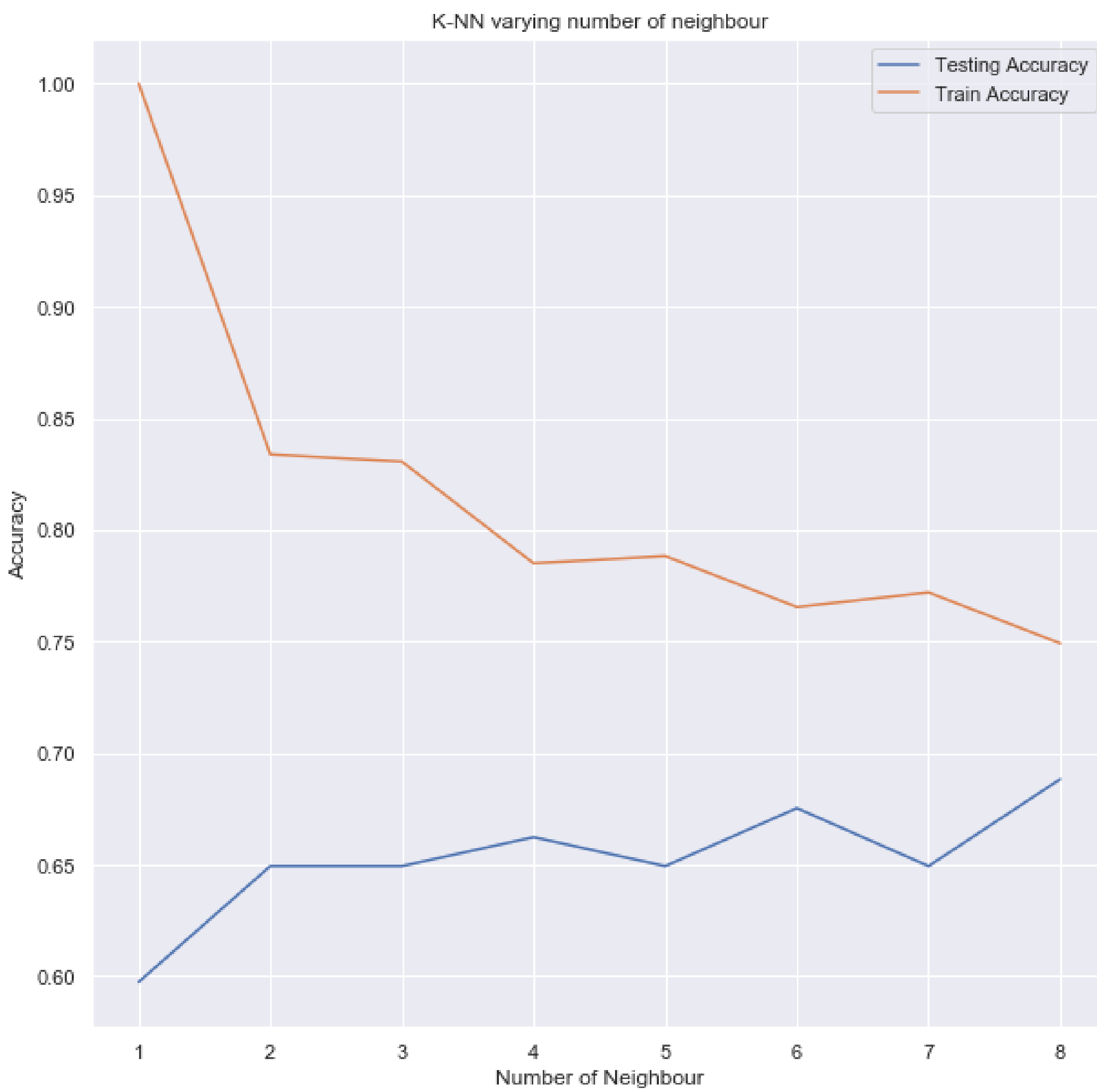
Applying on D2

[536]:
```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2,␣
↪random_state=0)
```

[537]:
```
neighbors=np.arange(1,9)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))

for i,k in enumerate(neighbors):
```

```
    knn2=KNeighborsClassifier(n_neighbors=k)

    knn2.fit(X2_train, y2_train)

    train_accuracy[i]=knn2.score(X2_train, y2_train)

    test_accuracy[i]=knn2.score(X2_test, y2_test)
```

[538]:
```
plt.figure(figsize=(10,10))
plt.title("K-NN varying number of neighbour")
plt.plot(neighbors, test_accuracy, label="Testing Accuracy")
plt.plot(neighbors, train_accuracy, label="Train Accuracy")
plt.legend()
plt.xlabel("Number of Neighbour")
plt.ylabel("Accuracy")
plt.show()
```



As test accracy is highest at k=8, We adopt the KNeighborsClassifier with number of neighbours as 8

[539]:
```
knn2=KNeighborsClassifier(n_neighbors=8)
knn2.fit(X2_train, y2_train)
```

[539]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=8, p=2,
                     weights='uniform')
```

[540]:
```
K2_pred=knn2.predict(X2_test)
print("Accuracy on Test Set:",metrics.accuracy_score(y2_test, K2_pred))
```

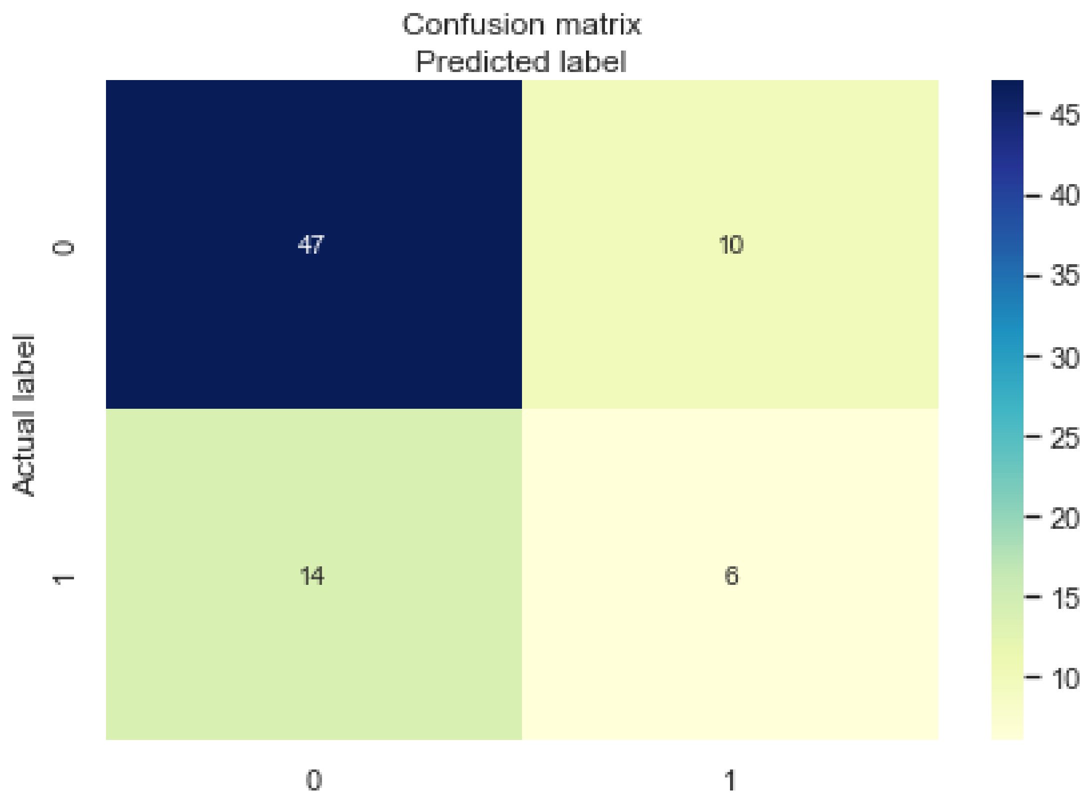
```
Accuracy on Test Set: 0.6883116883116883
```

[542]:
```
cnf2_matrix = metrics.confusion_matrix(y2_test, K2_pred)
cnf2_matrix
```

[542]:
```
array([[47, 10],
       [14,  6]], dtype=int64)
```

[543]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf2_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[543]:
```
Text(0.5, 257.44, 'Predicted label')
```



[544]:
```
print(metrics.classification_report(y2_test, knn2.predict(X2_test)))
```

```
              precision    recall  f1-score   support

           0       0.77      0.82      0.80        57
           1       0.38      0.30      0.33        20

    accuracy                           0.69        77
   macro avg       0.57      0.56      0.56        77
weighted avg       0.67      0.69      0.68        77
```

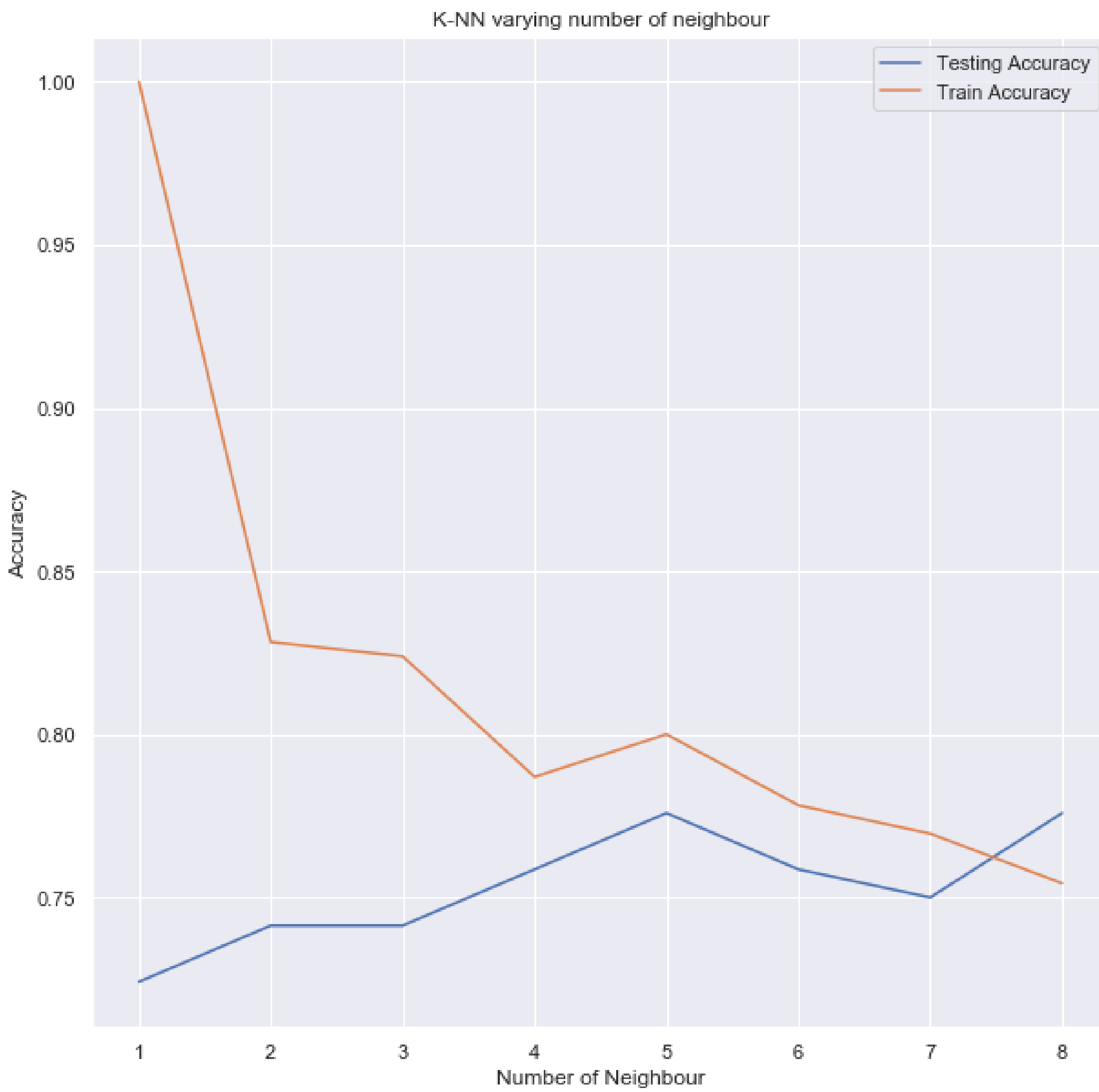
Applying on D3

[545]:
```
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2,␣
↪random_state=0)
```

[546]:
```
neighbors=np.arange(1,9)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))

for i,k in enumerate(neighbors):
```

```
    knn3=KNeighborsClassifier(n_neighbors=k)

    knn3.fit(X3_train, y3_train)

    train_accuracy[i]=knn3.score(X3_train, y3_train)

    test_accuracy[i]=knn3.score(X3_test, y3_test)
```

[547]:
```
plt.figure(figsize=(10,10))
plt.title("K-NN varying number of neighbour")
plt.plot(neighbors, test_accuracy, label="Testing Accuracy")
plt.plot(neighbors, train_accuracy, label="Train Accuracy")
plt.legend()
plt.xlabel("Number of Neighbour")
plt.ylabel("Accuracy")
plt.show()
```



As test accracy is highest at k=5, We adopt the KNeighborsClassifier with number of neighbours as 5

[548]:
```
knn3=KNeighborsClassifier(n_neighbors=5)
knn3.fit(X3_train, y3_train)
```

[548]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[549]:
```
K3_pred=knn3.predict(X3_test)
print("Accuracy on Test Set:",metrics.accuracy_score(y3_test, K3_pred))
```

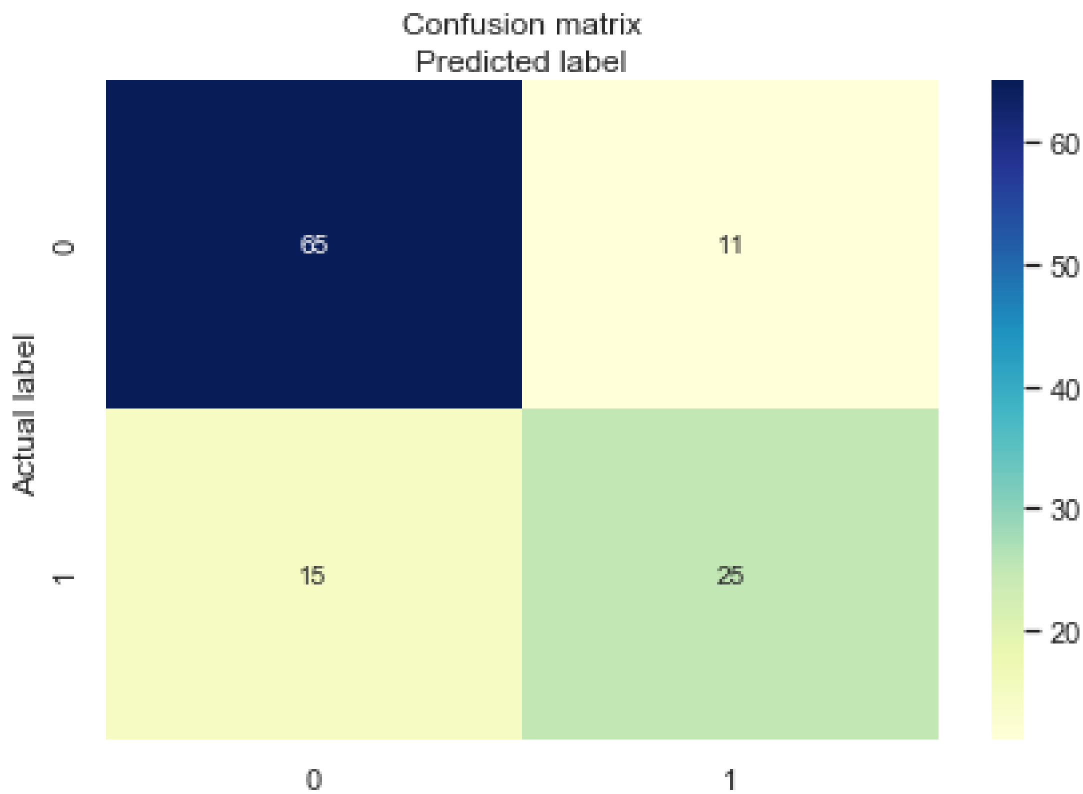
```
Accuracy on Test Set: 0.7758620689655172
```

[550]:
```
cnf3_matrix = metrics.confusion_matrix(y3_test, K3_pred)
cnf3_matrix
```

[550]:
```
array([[65, 11],
       [15, 25]], dtype=int64)
```

[551]:
```
# Plot the Confusion Matrix as a HeatMap
class_names=[0,1] # Name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf3_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

[551]:
```
Text(0.5, 257.44, 'Predicted label')
```



[554]:
```
print(metrics.classification_report(y3_test, knn3.predict(X3_test)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83        76
           1       0.69      0.62      0.66        40

    accuracy                           0.78       116
   macro avg       0.75      0.74      0.75       116
weighted avg       0.77      0.78      0.77       116
```

```
INFERENCE -

ACCURACY on test set for D1 is 0.7435897435897436

ACCURACY on test set for D2 is 0.6883116883116883

ACCURACY on test set for D3 is 0.7758620689655172

ACCURACY on test set for D (Main set) is 0.7662337662337663

INFERANCE - For this model, all the accuracies are in the same range. Only
dataset - D2 seems to be lesser than the range and this can be due to uneven
```

distribution of classes in the target variable of the training dataset upon spliting the main dataset

1.12.1 ANN MODEL

Applying on D1

[330]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,␣
↪random_state=0)
```

[331]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[332]:
```
# compiling the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[354]:
```
# fit the keras model on the dataset
history = model.fit(X1_train, y1_train, epochs=150, batch_size=10,verbose=0)
```

[355]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X1, y1, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

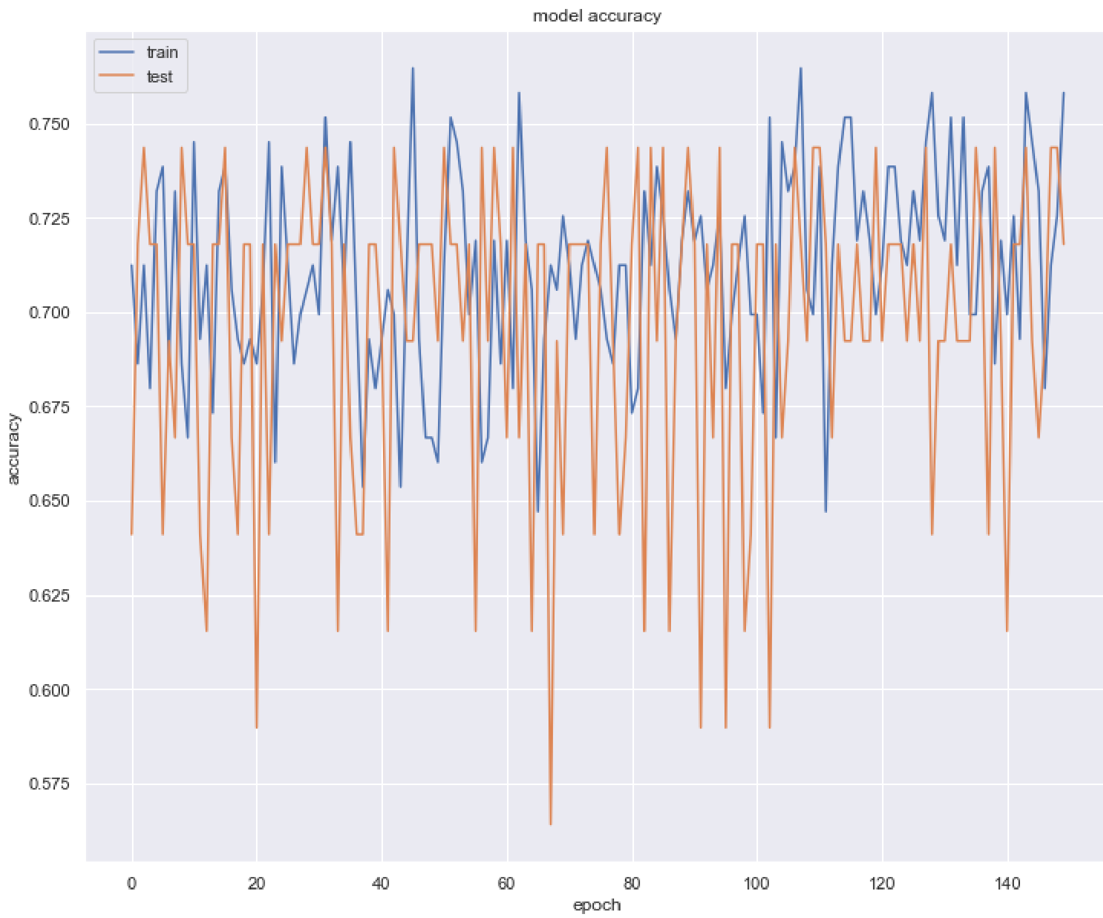
```
Accuracy: 77.60
```

[335]:
```
# make class predictions with the model
predictions = model.predict_classes(X1)
# summarize the first 5 cases
for i in range(10):
        print('%s => %d (expected %d)' % (X1[i].tolist(), predictions[i],␣
↪y1[i]))
```

```
[6.0, 148.0, 72.0, 35.0, 79.799479, 33.6, 0.627, 50.0] => 0 (expected 1)
[1.0, 85.0, 66.0, 29.0, 79.799479, 26.6, 0.35100000000000003, 31.0] => 0
(expected 0)
[8.0, 183.0, 64.0, 20.536458, 79.799479, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.16699999999999998, 21.0] => 0 (expected 0)
[3.8450519999999995, 137.0, 40.0, 35.0, 168.0, 43.1, 2.2880000000000003, 33.0]
=> 1 (expected 1)
[5.0, 116.0, 74.0, 20.536458, 79.799479, 25.6, 0.201, 30.0] => 0 (expected 0)
[3.0, 78.0, 50.0, 32.0, 88.0, 31.0, 0.248, 26.0] => 0 (expected 1)
[10.0, 115.0, 69.105469, 20.536458, 79.799479, 35.3, 0.134, 29.0] => 1 (expected
0)
[2.0, 197.0, 70.0, 45.0, 543.0, 30.5, 0.158, 53.0] => 1 (expected 1)
[8.0, 125.0, 96.0, 20.536458, 79.799479, 31.992578, 0.23199999999999998, 54.0]
=> 0 (expected 1)
```

[336]:
```
history = model.fit(X1_train, y1_train,validation_data = (X1_test,y1_test),␣
↪epochs=150, batch_size=10,verbose=0)
```

[337]:
```
#model accuracy
plt.figure(figsize=(12,10))
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'], loc='upper left')
plt.show()
```



Applying on D2

[338]:
```
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2,␣
↪random_state=0)
```

[339]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[340]:
```
# compiling the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[356]:
```
# fit the keras model on the dataset
history = model.fit(X2_train, y2_train, epochs=150, batch_size=10,verbose=0)
```

[357]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X2, y2, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

```
Accuracy: 78.12
```

[343]:
```
# make class predictions with the model
predictions = model.predict_classes(X2)
# summarize the first 5 cases
for i in range(10):
        print('%s => %d (expected %d)' % (X2[i].tolist(), predictions[i],␣
↪y2[i]))
```

```
[6.0, 148.0, 72.0, 35.0, 79.799479, 33.6, 0.627, 50.0] => 0 (expected 1)
[1.0, 85.0, 66.0, 29.0, 79.799479, 26.6, 0.35100000000000003, 31.0] => 0
(expected 0)
[8.0, 183.0, 64.0, 20.536458, 79.799479, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.16699999999999998, 21.0] => 0 (expected 0)
[3.8450519999999995, 137.0, 40.0, 35.0, 168.0, 43.1, 2.2880000000000003, 33.0]
=> 1 (expected 1)
[5.0, 116.0, 74.0, 20.536458, 79.799479, 25.6, 0.201, 30.0] => 0 (expected 0)
[3.0, 78.0, 50.0, 32.0, 88.0, 31.0, 0.248, 26.0] => 0 (expected 1)
[10.0, 115.0, 69.105469, 20.536458, 79.799479, 35.3, 0.134, 29.0] => 1 (expected
0)
[2.0, 197.0, 70.0, 45.0, 543.0, 30.5, 0.158, 53.0] => 1 (expected 1)
[8.0, 125.0, 96.0, 20.536458, 79.799479, 31.992578, 0.23199999999999998, 54.0]
=> 0 (expected 1)
```

[344]:
```
history = model.fit(X2_train, y2_train,validation_data = (X2_test,y2_test),␣
↪epochs=150, batch_size=10,verbose=0)
```

[345]:
```
#model accuracy
plt.figure(figsize=(12,10))
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'], loc='upper left')
plt.show()
```

Applying on D3

[346]:
```
X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2,␣
↪random_state=0)
```

[347]:
```
# defining the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

[348]:
```
# compiling the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

[358]:
```
# fit the keras model on the dataset
history = model.fit(X3_train, y3_train, epochs=150, batch_size=10,verbose=0)
```

[359]:
```
# evaluating the keras model
_, accuracy = model.evaluate(X3, y3, verbose=0)
print('Accuracy: %.2f' % (accuracy*100))
```

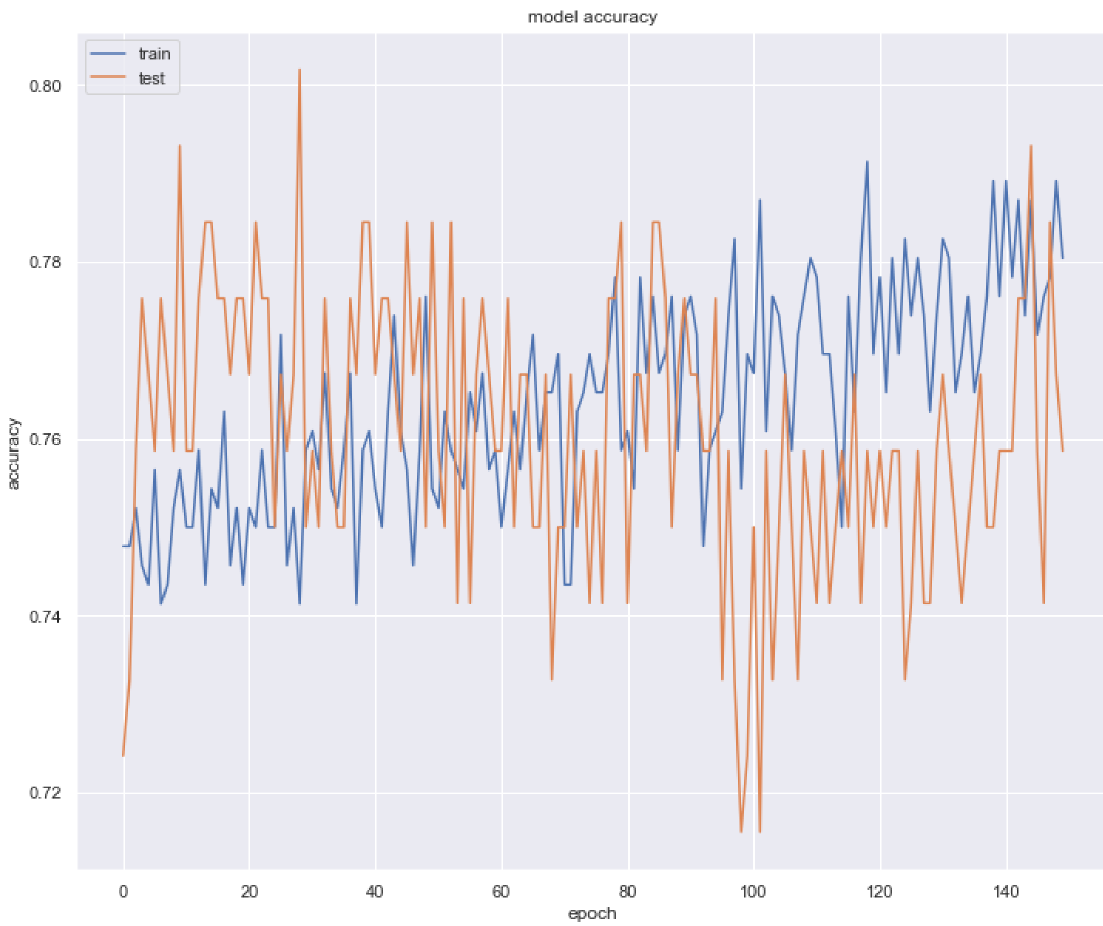
```
Accuracy: 79.34
```

[360]:
```
# make class predictions with the model
predictions = model.predict_classes(X3)
# summarize the first 5 cases
for i in range(10):
        print('%s => %d (expected %d)' % (X3[i].tolist(), predictions[i],␣
↪y3[i]))
```

```
[6.0, 148.0, 72.0, 35.0, 79.799479, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 79.799479, 26.6, 0.35100000000000003, 31.0] => 0
(expected 0)
[8.0, 183.0, 64.0, 20.536458, 79.799479, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.16699999999999998, 21.0] => 0 (expected 0)
[3.8450519999999995, 137.0, 40.0, 35.0, 168.0, 43.1, 2.2880000000000003, 33.0]
=> 1 (expected 1)
[5.0, 116.0, 74.0, 20.536458, 79.799479, 25.6, 0.201, 30.0] => 0 (expected 0)
[3.0, 78.0, 50.0, 32.0, 88.0, 31.0, 0.248, 26.0] => 0 (expected 1)
[10.0, 115.0, 69.105469, 20.536458, 79.799479, 35.3, 0.134, 29.0] => 0 (expected
0)
[2.0, 197.0, 70.0, 45.0, 543.0, 30.5, 0.158, 53.0] => 1 (expected 1)
[8.0, 125.0, 96.0, 20.536458, 79.799479, 31.992578, 0.23199999999999998, 54.0]
=> 0 (expected 1)
```

[352]:
```
history = model.fit(X3_train, y3_train,validation_data = (X3_test,y3_test),␣
↪epochs=150, batch_size=10,verbose=0)
```

[353]:
```
#model accuracy
plt.figure(figsize=(12,10))
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'], loc='upper left')
plt.show()
```




```
INFERENCE -

ACCURACY on test set for D1 is 0.7760

ACCURACY on test set for D2 is 0.7812

ACCURACY on test set for D3 is 0.7934

ACCURACY on test set for D (Main) is 0.7865
```

INFERENCE - For ANN model, all the accuracies seem to be in the same range.

[ ]: