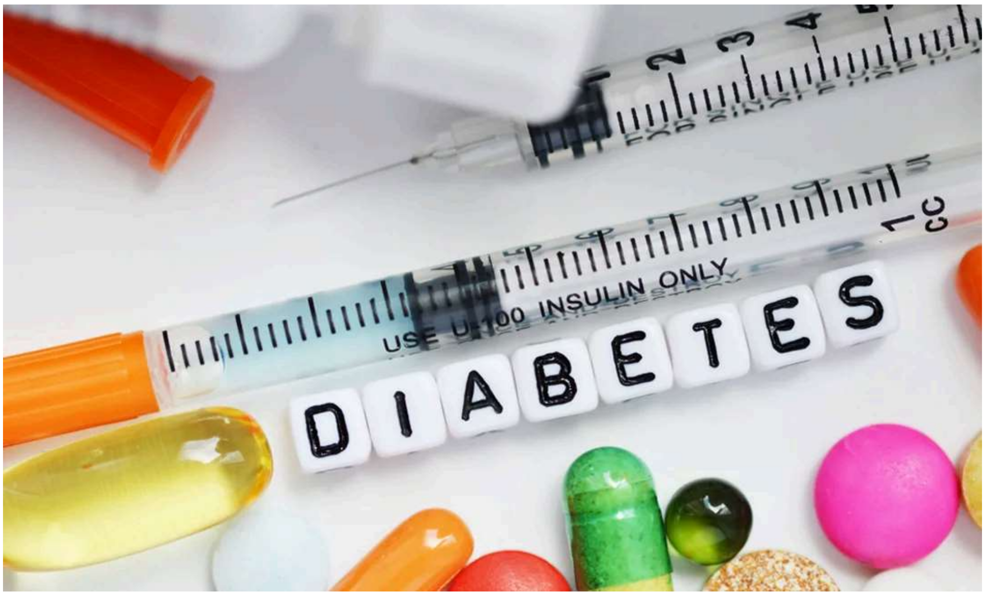
Diabetes_Prediction_with_ANN

Mayank Srivastava

- [(https://www.linkedin.com/in/mayank-srivastava-6a8421105/)](https://www.linkedin.com/in/mayank-srivastava-6a8421105/)

In [ ]:

```

```



Pima Indians Diabetes Database

Problem:

- Predict the onset of diabetes based on diagnostic measures

Data Source:

- This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. 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. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Content

- The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

In [1]:

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

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

from sklearn.metrics import recall_score,confusion_matrix,roc_auc_score, roc_curve,auc

```

In [2]:

```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout

```

In [3]:

```
# Set random seeds for reproducibility
import random
seed = 42
tf.random.set_seed(seed)     # Sets the random seed for TensorFlow operations.
np.random.seed(seed)         # Sets the random seed for NumPy operations.
random.seed(seed)            # Sets the random seed for Python's built-in random module.

```

Loading the Dataset

In [4]:

```
df=pd.read_csv('diabetes.csv')
df.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:

```
# checking the info
df.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
```

In [6]:

```
# checking the descriptive stats fo all features
df.describe()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [7]:

```
# lets check for 0s in each column
df.eq(0).sum()
```

Out[7]:

```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

Observations:

1. Pregnancies and Outcome can have zero values based on patient's parameters
2. But for other variables having ZEROES is not possible

In [8]:

```
# we will replace 0s with np.NaN
np.NaN
```

Out[8]: nan

In [9]:

```
df.columns
```

Out[9]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [10]:

```
df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age',]]=df[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age',]].replace({0:np.NaN})
```

In [11]:

```
df.isnull().sum()
```

Out[11]:

```
Pregnancies                   0
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                       0
dtype: int64
```

Observations:

1. Pregnancies and Outcome can have zeroe values based on patient's parameters
2. But other values having ZEROES is not possible

In [12]:

```
# we will compute all above NaN with column mean values
df.mean()
```

Out[12]:

```
Pregnancies                   3.845052
Glucose                     121.686763
BloodPressure                72.405184
SkinThickness                29.153420
Insulin                     155.548223
BMI                          32.457464
DiabetesPedigreeFunction      0.471876
Age                          33.240885
Outcome                       0.348958
dtype: float64
```

In [13]:

```
#imputing NaN values
df=df.fillna(df.mean())
```

Observations:

- Dataset has 768 rows x 9 columns
- Zeroes have been imputed with mean. There are no null/missing values in the dataset.
- Target column: 'Outcome'
- All input features are numerical

In [14]:

```
# Descriptive stats
df.describe()
```

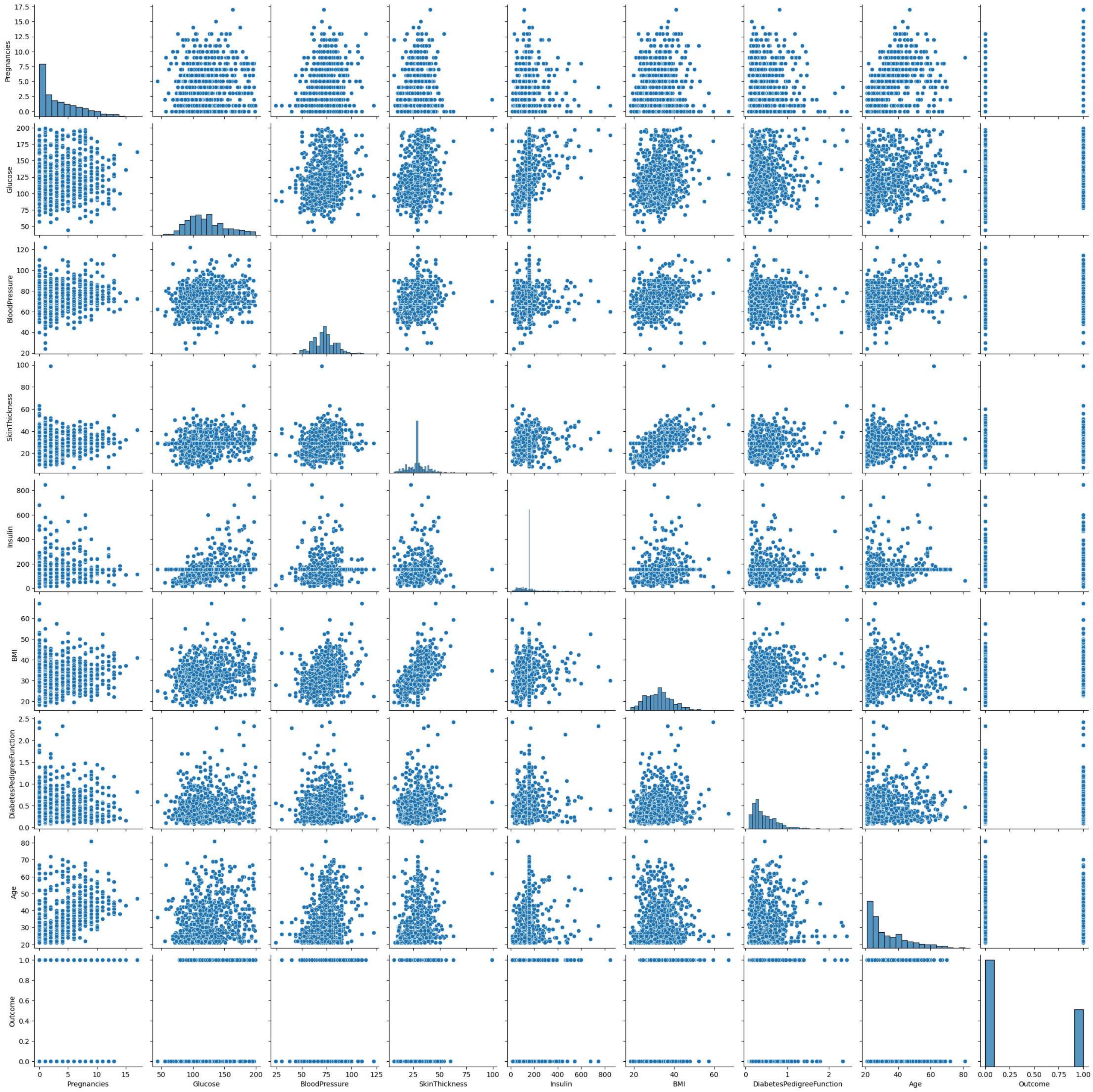
Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.686763	72.405184	29.153420	155.548223	32.457464	0.471876	33.240885	0.348958
std	3.369578	30.435949	12.096346	8.790942	85.021108	6.875151	0.331329	11.760232	0.476951
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.078000	21.000000	0.000000
25%	1.000000	99.750000	64.000000	25.000000	121.500000	27.500000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.202592	29.153420	155.548223	32.400000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	155.548223	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [15]:

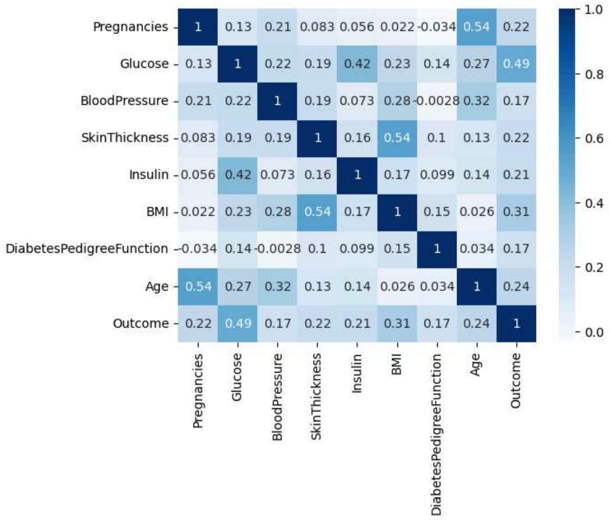
```
# pairplot
sns.pairplot(df)
plt.show()
```

```
E:\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [16]:
```python
corr= df.corr(numeric_only = True)
sns.heatmap(corr, annot = True, cmap ="Blues")
```

Out[16]: <Axes: >



In [17]:
```python
df.corr().nlargest(4,'Outcome')['Outcome']
```

Out[17]:
```
Outcome    1.000000
Glucose    0.492928
BMI        0.311924
Age        0.238356
Name: Outcome, dtype: float64
```

Splitting the Dataset

In [18]:
```python
x= df.drop('Outcome', axis =1)
y=df.Outcome
```

In [19]:
```python
# checking for imbalance
df.Outcome.value_counts()
```

Out[19]:
```
Outcome
0    500
1    268
Name: count, dtype: int64
```

In [20]:
```python
df.Outcome.value_counts(normalize = True)
```

Out[20]:
```
Outcome
0    0.651042
1    0.348958
Name: proportion, dtype: float64
```

In [21]:
```python
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest= train_test_split(x,y, test_size =0.2, random_state =42,stratify =y)
```

We have used **stratify =y**, to ensure same proportion of positives and negatives in test and train sets, as in the original datset

In [22]:
```python
ytrain.value_counts(normalize=True)
```

Out[22]:
```
Outcome
0    0.651466
1    0.348534
Name: proportion, dtype: float64
```

In [23]:
```python
ytest.value_counts(normalize=True)
```

Out[23]:
```
Outcome
0    0.649351
1    0.350649
Name: proportion, dtype: float64
```

Scaling the data

In [24]:
```python
from sklearn.preprocessing import StandardScaler, MinMaxScaler
scaler=MinMaxScaler()
xtrain=scaler.fit_transform(xtrain)
xtest=scaler.transform(xtest)
```

In [25]:
```python
xtrain.shape, xtest.shape
```

Out[25]: ((614, 8), (154, 8))

In [26]:
```python
xtrain
```

Out[26]:
```
array([[0.05882353, 0.23776224, 0.3877551 , ..., 0.18404908, 0.22093541,
        0.05      ],
       [0.29411765, 0.48951049, 0.55102041, ..., 0.23312883, 0.15812918,
        0.31666667],
       [0.11764706, 0.34265734, 0.34693878, ..., 0.34151329, 0.06280624,
        0.06666667],
       ...,
       [0.05882353, 0.28671329, 0.46938776, ..., 0.40695297, 0.0596882 ,
        0.15      ],
       [0.58823529, 0.38461538, 0.46938776, ..., 0.19018405, 0.02538976,
        0.31666667],
       [0.23529412, 0.61538462, 0.34693878, ..., 0.23108384, 0.09042316,
        0.26666667]])
```

Balancing the data

In [27]:
```python
# Imbalanced --> Balanced
# pip install git+https://github.com/scikit-learn-contrib/imbalanced-learn.git@master
```

In [28]:
```python
# removing the imbalance
from imblearn.over_sampling import RandomOverSampler
```

In [29]:
```python
smote=RandomOverSampler()
smote_xtrain, smote_ytrain = smote.fit_resample(xtrain, ytrain)
```

In [30]:
```python
smote_ytrain.value_counts(normalize =True)
#y-class in trianing data is now balanced
```

Out[30]:
```
Outcome
0    0.5
1    0.5
Name: proportion, dtype: float64
```

In [31]:
```python
xtrain.shape, ytrain.shape
```

Out[31]: `((614, 8), (614,))`

In [32]:
```python
smote_xtrain.shape, smote_ytrain.shape
```

Out[32]: `((800, 8), (800,))`

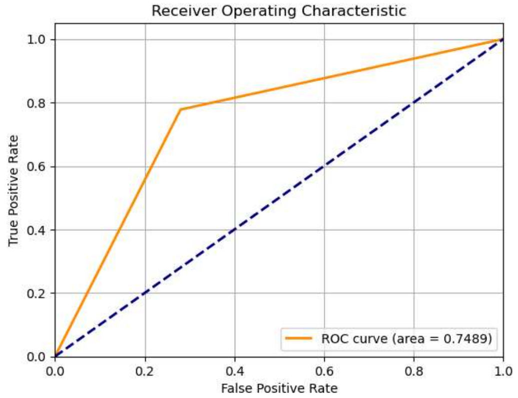
our y-class in now balanced, using over-sampling

In [33]:
```python
# Lets try a ML algorithm, say SVM
from sklearn.svm import SVC
svc=SVC(kernel='rbf')
#'rbf', 'linear', 'precomputed', 'sigmoid', 'poly'
svc.fit(smote_xtrain,smote_ytrain)
train_pred_svc= svc.predict(xtrain)
test_pred_svc =svc.predict(xtest)

from sklearn.metrics import accuracy_score, recall_score
print('Training metrics')
print('accuracy_score: ',accuracy_score(ytrain, train_pred_svc))
print('recall_score: ',recall_score(ytrain, train_pred_svc))
print('Testing metrics')
print('accuracy_score: ',accuracy_score(ytest, test_pred_svc))
print('recall_score: ',recall_score(ytest, test_pred_svc))
```

```
Training metrics
accuracy_score:  0.8061889250814332
recall_score:  0.8271028037383178
Testing metrics
accuracy_score:  0.7402597402597403
recall_score:  0.7777777777777778
```

In [34]:
```python
# plotting the ROC-curve

from sklearn.metrics import roc_auc_score, roc_curve, auc
fpr, tpr, thresholds = roc_curve(ytest, test_pred_svc)
roc_auc = auc(fpr, tpr)
```

In [35]:
```python
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.4f})' )
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



In [36]:
```python
auc(fpr,tpr)
```

Out[36]: `0.7488888888888888`

In [37]:
```python
# # Lets try a ML algorithm, say DT
# from sklearn.tree import DecisionTreeClassifier
# dt=DecisionTreeClassifier()
# dt.fit(xtrain,ytrain)
# train_pred_dt= dt.predict(xtrain)
# test_pred_dt =dt.predict(xtest)

# from sklearn.metrics import accuracy_score, recall_score
# print('Training metrics')
# print('accuracy_score: ',accuracy_score(ytrain, train_pred_dt))
# print('recall_score: ',recall_score(ytrain, train_pred_dt))
# print('Testing metrics')
# print('accuracy_score: ',accuracy_score(ytest, test_pred_dt))
# print('recall_score: ',recall_score(ytest, test_pred_dt))
```

ANN_Model_Architecture

In [38]:

```python
# Starting with a simple model

model= Sequential()
model.add(Dense(32, activation ='relu', input_dim=8)) # input layer
# model.add(Dropout(0.5))
model.add(Dense (1, activation ='sigmoid'))
```

```
E:\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [39]:

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

In [40]:

```python
# Basic model without HP-tuning
# default batch_size if not specified is 32
from tensorflow.keras.callbacks import ModelCheckpoint
# Define ModelCheckpoint callback to save the best weights
checkpoint = ModelCheckpoint(r'best_model.keras',
                             monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)

# Train the model with ModelCheckpoint callback
history=model.fit(smote_xtrain,smote_ytrain, epochs =100, validation_data=(xtest,ytest), shuffle =False,\
                 callbacks=[checkpoint])

# Load the best weights
model.load_weights('best_model.keras')

```

```
Epoch 22: val_accuracy did not improve from 0.76623
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7060 - loss: 0.6219 - val_accuracy: 0.7532 - val_loss: 0.6147
Epoch 23/100
12/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6880 - loss: 0.6234
Epoch 23: val_accuracy did not improve from 0.76623
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7071 - loss: 0.6185 - val_accuracy: 0.7532 - val_loss: 0.6111
Epoch 24/100
24/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6993 - loss: 0.6164
Epoch 24: val_accuracy did not improve from 0.76623
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7019 - loss: 0.6151 - val_accuracy: 0.7532 - val_loss: 0.6076
Epoch 25/100

 1/25 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step - accuracy: 0.6250 - loss: 0.6258
Epoch 25: val_accuracy did not improve from 0.76623
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7014 - loss: 0.6119 - val_accuracy: 0.7468 - val_loss: 0.6041
Epoch 26/100
23/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7006 - loss: 0.6102
Epoch 26: val_accuracy did not improve from 0.76623
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7040 - loss: 0.6085 - val_accuracy: 0.7468 - val_loss: 0.6006
Epoch 27/100
```

In [41]:

```python
# model.evaluate returns the loss value and metrics value for the model in test mode.
# loss, metrics['accuracy']
model.evaluate(xtrain,ytrain)

```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7002 - loss: 0.6323
```

Out[41]:

```
[0.6293267607688904, 0.7133550643920898]
```

In [42]:

```python
model.evaluate(xtest,ytest)

# loss, metrics['accuracy']
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7676 - loss: 0.6330
```

Out[42]:

```
[0.6299659609794617, 0.7662337422370911]
```

In [43]:

```python
train_pred = [1 if i>0.5 else 0 for i in model.predict(xtrain)]
test_pred =[1 if i>0.5 else 0 for i in model.predict(xtest)]
print('Training metrics')
print('accuracy_score: ',accuracy_score(ytrain, train_pred))
print('recall_score: ',recall_score(ytrain, train_pred))
print('Testing metrics')
print('accuracy_score: ',accuracy_score(ytest, test_pred))
print('recall_score: ',recall_score(ytest, test_pred))
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Training metrics
accuracy_score:  0.7133550488599348
recall_score:  0.8504672897196262
Testing metrics
accuracy_score:  0.7662337662337663
recall_score:  0.8888888888888888
```

In [44]:

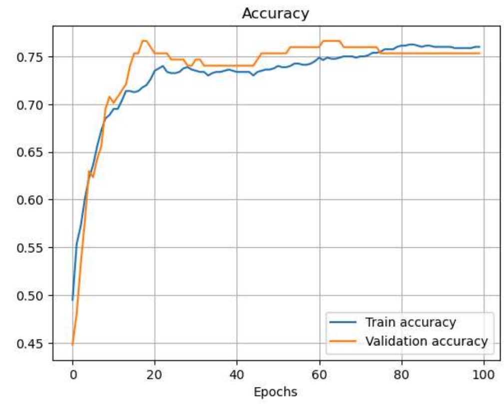
```python
fpr, tpr, thresholds = roc_curve(ytest, test_pred)
roc_auc = auc(fpr, tpr)
roc_auc
```

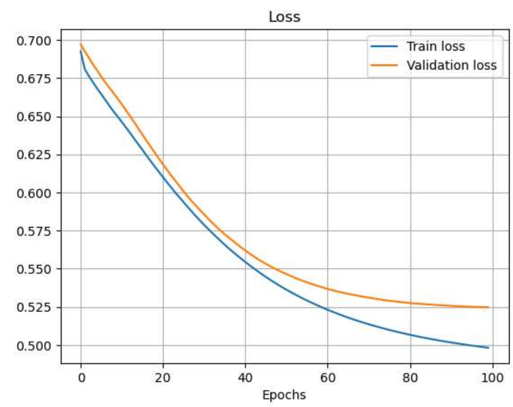
Out[44]:

```
0.7944444444444444
```

In [45]:

```python
# Plot accuracy and loss
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'], label='Train accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.legend()
plt.title('Accuracy')
plt.xlabel("Epochs")
plt.grid()
plt.show()

# Similar plot for loss
plt.plot(history.history['loss'], label='Train loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.legend()
plt.title("Loss")
plt.xlabel("Epochs")
plt.grid()
plt.show()
```





In [46]:

```python
pd.DataFrame({'ytest':ytest,'pred':test_pred})
```

268	0	0
635	1	1
87	0	0
75	0	0
537	0	0
38	1	0
298	1	1
115	1	1
86	0	1
32	0	0
637	0	0
593	0	0
425	1	1

Using library KERAS_TUNER to optimize all parametrs

In [47]:

```python
import keras_tuner as kt
```

Steps:

1. create a function
2. create a tuner object and pass the function and objective in the tuner object
3. train the tuner object with training data to find best model

hp is the hyper paramater object which provide followinf methods:

- hp.Int()
- hp.Float()
- hp.Boolean()
- hp.Choice()

to handle various situations while tuning the parameters

Keras Tuner to Hypertune:

- How to select appropriate optimizer
- learning rate of optimizer
- No. of neurons in a layer
- How to select no. of layers
- All in one model

In [48]:
```python
def final_model(hp):
    model=Sequential()

    model.add(Dense(hp.Int('units: ', 32,512,step =8), input_dim=8, activation ='relu'))
    if hp.Boolean('dropout'):
        model.add(Dropout(hp.Choice('drop_rate: ', values=[i/10 for i in range(1,10)])))

    model.add(Dense(1, activation = 'sigmoid'))
    model.compile(optimizer =hp.Choice('optimizer', values =['adam','adamax','adadelta','nadam','rmsprop','sgd','lion']),
                  loss='binary_crossentropy', metrics =['accuracy'])

    return model
```

In [49]:
```python
tuner=kt.RandomSearch(final_model, objective ='val_accuracy', max_trials =10)
tuner.search(smote_xtrain, smote_ytrain, validation_data =(xtest, ytest), epochs =10, shuffle = False)
```

```
Reloading Tuner from .\untitled_project\tuner0.json
```

In [50]:
```python
# tuner.results_summary()
```

In [51]:
```python
tuner.get_best_hyperparameters()[0].values
```

Out[51]:
```
{'units: ': 400, 'dropout': False, 'optimizer': 'nadam', 'drop_rate: ': 0.8}
```

In [52]:
```python
best_model=tuner.get_best_models(num_models =1)[0]
```

```
WARNING:tensorflow:From E:\anaconda3\Lib\site-packages\keras\src\backend\common\global_state.py:82: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default
_graph instead.


E:\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using
an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
E:\anaconda3\Lib\site-packages\keras\src\saving\saving_lib.py:576: UserWarning: Skipping variable loading for optimizer 'nadam', because it has 2 variables whereas the saved optimizer
has 11 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
```

In [53]:
```python
# best_model's currrent performance, with initial 10 epochs of tuner
```

In [55]:
```python
train_pred = [1 if i>0.5 else 0 for i in best_model.predict(xtrain)]
test_pred =[1 if i>0.5 else 0 for i in best_model.predict(xtest)]
print('Training metrics')
print('accuracy_score: ',accuracy_score(ytrain, train_pred))
print('recall_score: ',recall_score(ytrain, train_pred))
print('Testing metrics')
print('accuracy_score: ',accuracy_score(ytest, test_pred))
print('recall_score: ',recall_score(ytest, test_pred))
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Training metrics
accuracy_score:  0.7133550488599348
recall_score:  0.8644859813084113
Testing metrics
accuracy_score:  0.7792207792207793
recall_score:  0.8888888888888888
```

In [56]:
```python
best_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 400)	3,600
dense_1 (Dense)	(None, 1)	401

Total params: 4,001 (15.63 KB)

Trainable params: 4,001 (15.63 KB)

Non-trainable params: 0 (0.00 B)

Since the last achived accuracy with ANN Simple architecture was better, there is No benefit of tuning the params

In [57]:
```python
fpr, tpr, thresholds = roc_curve(ytest, test_pred)
roc_auc = auc(fpr, tpr)
roc_auc
```

Out[57]:
```
0.8044444444444444
```

In [58]:
```python
# best_model's currrent performance, with initial 10 epochs of tuner, and further training for 100 epochs
checkpoint = ModelCheckpoint(r'best_model_1.keras',
                             monitor='val_accuracy', save_best_only=True, mode='max', verbose=1)

# Train the model with ModelCheckpoint callback
history=best_model.fit(smote_xtrain,smote_ytrain, epochs =100, initial_epoch=10, validation_data=(xtest,ytest), shuffle =False,\
                 callbacks=[checkpoint])

# Load the best weights
best_model.load_weights('best_model_1.keras')
```

```
Epoch 36/100
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7318 - loss: 0.5231
Epoch 36: val_accuracy did not improve from 0.79221

25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7321 - loss: 0.5224 - val_accuracy: 0.7857 - val_loss: 0.5258
Epoch 37/100
20/25 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.7327 - loss: 0.5249
Epoch 37: val_accuracy did not improve from 0.79221
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7335 - loss: 0.5212 - val_accuracy: 0.7922 - val_loss: 0.5251
Epoch 38/100
 1/25 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.7188 - loss: 0.5532
Epoch 38: val_accuracy did not improve from 0.79221
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7383 - loss: 0.5202 - val_accuracy: 0.7857 - val_loss: 0.5250
Epoch 39/100
 1/25 ━━━━━━━━━━━━━━━━━━━━ 1s 62ms/step - accuracy: 0.7188 - loss: 0.5533
Epoch 39: val_accuracy did not improve from 0.79221
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7383 - loss: 0.5192 - val_accuracy: 0.7857 - val_loss: 0.5243
Epoch 40/100
24/25 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7400 - loss: 0.5197
Epoch 40: val_accuracy did not improve from 0.79221
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7406 - loss: 0.5182 - val_accuracy: 0.7857 - val_loss: 0.5240
```

In [59]:

```
train_pred = [1 if i>0.5 else 0 for i in best_model.predict(xtrain)]
test_pred =[1 if i>0.5 else 0 for i in best_model.predict(xtest)]
print('Training metrics')
print('accuracy_score: ',accuracy_score(ytrain, train_pred))
print('recall_score: ',recall_score(ytrain, train_pred))
print('Testing metrics')
print('accuracy_score: ',accuracy_score(ytest, test_pred))
print('recall_score: ',recall_score(ytest, test_pred))
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Training metrics
accuracy_score:  0.757328990228013
recall_score:  0.794392523364486
Testing metrics
accuracy_score:  0.7922077922077922
recall_score:  0.8333333333333334
```

In [60]:

```
fpr, tpr, thresholds = roc_curve(ytest, test_pred)
roc_auc = auc(fpr, tpr)
roc_auc
```

Out[60]: 0.8016666666666667

The performance thorugh various methods is as follows

	SVC (kernel ='rbf')	ANN, optimizer =adam, Layers (2) = Dense, Dense; 32-relu,1-sigmoid	Tuned(nadam,Layers =2 Dense Dense,400,1	
Training metrics				
accuracy_score:	0.806188925	0.713355049	0.713355049	0.75732899
recall_score:	0.827102804	0.85046729	0.864485981	0.794392523
Testing metrics				
accuracy_score:	0.74025974	0.766233766	0.779220779	0.792207792
recall_score:	0.777777778	0.888888889	0.888888889	0.833333333
ROC_AUC	0.748888889	0.794444444	0.804444444	0.801666667
			with initial_epoch only	after training tuned best model till 100 epochs