# Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: [https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling](https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling) Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

```
df = pd.read_csv("Churn_Modelling.csv")
```

# Preprocessing.

```
df.head()
```

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | Has |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | 1 | |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660.80 | 3 | |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 | 2 | |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | |

```
df.shape
```

```
(10000, 14)
```

```
df.describe()
```

| | RowNumber | CustomerId | CreditScore | Age | Tenure | Balance | NumOfProducts | Has( |
|---|---|---|---|---|---|---|---|---|
| **count** | 10000.00000 | 1.000000e+04 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000 |
| **mean** | 5000.50000 | 1.569094e+07 | 650.528800 | 38.921800 | 5.012800 | 76485.889288 | 1.530200 | 0 |
| **std** | 2886.89568 | 7.193619e+04 | 96.653299 | 10.487806 | 2.892174 | 62397.405202 | 0.581654 | 0 |
| **min** | 1.00000 | 1.556570e+07 | 350.000000 | 18.000000 | 0.000000 | 0.000000 | 1.000000 | 0 |
| **25%** | 2500.75000 | 1.562853e+07 | 584.000000 | 32.000000 | 3.000000 | 0.000000 | 1.000000 | 0 |

```
1 df.isnull()
```

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | H |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False | False | False | False | False | |
| **1** | False | False | False | False | False | False | False | False | False | False | |
| **2** | False | False | False | False | False | False | False | False | False | False | |
| **3** | False | False | False | False | False | False | False | False | False | False | |
| **4** | False | False | False | False | False | False | False | False | False | False | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **9995** | False | False | False | False | False | False | False | False | False | False | |
| **9996** | False | False | False | False | False | False | False | False | False | False | |
| **9997** | False | False | False | False | False | False | False | False | False | False | |
| **9998** | False | False | False | False | False | False | False | False | False | False | |
| **9999** | False | False | False | False | False | False | False | False | False | False | |

10000 rows × 14 columns

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

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
```

```
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.dtypes
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary columns
```

```
df.head()
```

| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | Estimate |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 10 |
| **1** | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | 11 |
| **2** | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | 11 |
| **3** | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 9 |
| **4** | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | 7 |

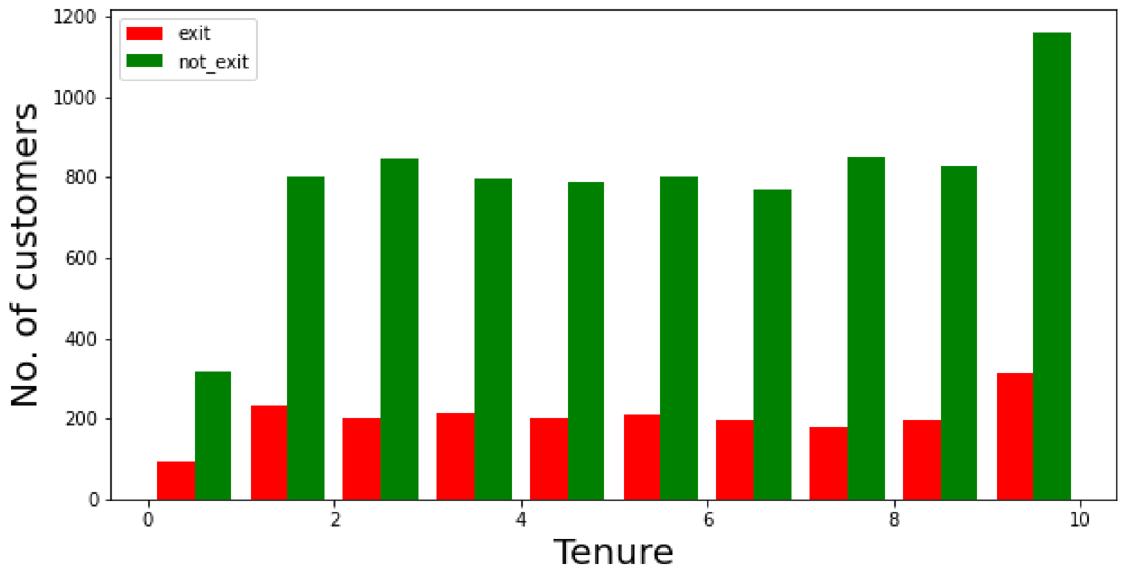
# Visualization

```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
```

```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

```
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecationWarning: Creating an
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationWarning: Creating
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```




```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

```
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```


## Converting the Categorical Variables

```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedS
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
```

```

df = pd.concat([df,gender,states], axis = 1)
```

## Splitting the training and testing Dataset

```
df.head()
```

| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | Estimate |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | 10 |
| **1** | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | 11 |
| **2** | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | 11 |
| **3** | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | 9 |
| **4** | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | 7 |

```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','M
```

```
y = df['Exited']
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

## Normalizing the values with mean as 0 and Standard Deviation as 1

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

```
X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
X_train
```

```
array([[ 0.33912606, -1.41493093,  1.38251425, ..., -1.08628092,
        -0.56679212,  1.72941551],
```

```
       [-1.58539179,  2.10250049,  0.68783334, ...,  0.92057219,
         1.7643153 , -0.57823004],
       [ 0.67022591, -0.84453664, -1.04886894, ..., -1.08628092,
         1.7643153 , -0.57823004],
       ...,
       [ 1.77734102, -0.17907664, -0.00684757, ...,  0.92057219,
        -0.56679212, -0.57823004],
       [-0.44723607, -1.03466807, -0.35418802, ...,  0.92057219,
         1.7643153 , -0.57823004],
       [-0.77833592,  0.67651478, -0.35418802, ...,  0.92057219,
         1.7643153 , -0.57823004]])
```

```
X_test
```

```
array([[ 0.2563511 , -0.36920807, -1.74354985, ...,  0.92057219,
        -0.56679212,  1.72941551],
       [-1.1301295 , -1.31986521,  1.0351738 , ...,  0.92057219,
         1.7643153 , -0.57823004],
       [ 0.93924453,  0.01105478,  1.0351738 , ...,  0.92057219,
        -0.56679212, -0.57823004],
       ...,
       [-1.34741378, -0.36920807,  0.34049289, ...,  0.92057219,
        -0.56679212, -0.57823004],
       [ 1.01167262, -1.2247995 ,  0.34049289, ..., -1.08628092,
        -0.56679212, -0.57823004],
       [ 0.14253553,  0.1061205 , -1.39620939, ..., -1.08628092,
        -0.56679212,  1.72941551]])
```

# Building the Classifier Model using Keras

```
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors initially.
```

```
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

```
classifier = Sequential()
```

```
#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "uniform"))
```

```
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform"))   #Adding second hidden laye
```

```
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform")) #Final neuron will be hav
```

```
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy']) #To compile the Artific
```

```
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 6)                 72

 dense_7 (Dense)             (None, 6)                 42

 dense_8 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training dataset
```

```
Epoch 1/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4939 - accuracy: 0.7953
Epoch 2/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4256 - accuracy: 0.7973
Epoch 3/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4215 - accuracy: 0.7973
Epoch 4/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4179 - accuracy: 0.7973
Epoch 5/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4144 - accuracy: 0.8163
Epoch 6/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4121 - accuracy: 0.8270
Epoch 7/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4099 - accuracy: 0.8311
Epoch 8/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4081 - accuracy: 0.8313
Epoch 9/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4062 - accuracy: 0.8321
Epoch 10/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4056 - accuracy: 0.8330
Epoch 11/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4042 - accuracy: 0.8357
Epoch 12/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4031 - accuracy: 0.8354
Epoch 13/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4020 - accuracy: 0.8374
Epoch 14/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4018 - accuracy: 0.8369
Epoch 15/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4013 - accuracy: 0.8354
Epoch 16/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4005 - accuracy: 0.8360
Epoch 17/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3999 - accuracy: 0.8379
Epoch 18/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3990 - accuracy: 0.8366
Epoch 19/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3987 - accuracy: 0.8377
Epoch 20/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3990 - accuracy: 0.8369
Epoch 21/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3981 - accuracy: 0.8389
Epoch 22/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3979 - accuracy: 0.8373
Epoch 23/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3977 - accuracy: 0.8381
```

```
Epoch 24/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3968 - accuracy: 0.8379
Epoch 25/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3972 - accuracy: 0.8376
Epoch 26/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3966 - accuracy: 0.8376
Epoch 27/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3965 - accuracy: 0.8381
Epoch 28/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3964 - accuracy: 0.8384
Epoch 29/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3964 - accuracy: 0.8370
```

```
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

```
94/94 [==============================] - 0s 1ms/step
```

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

```
cm = confusion_matrix(y_test,y_pred)
```

```
cm
```

```
array([[2314,   68],
       [ 443,  175]])
```

```
accuracy = accuracy_score(y_test,y_pred)
```

```
accuracy
```

```
0.8296666666666667
```

```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Text(69.0, 0.5, 'Truth')
```


```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.97      0.90      2382
           1       0.72      0.28      0.41       618

    accuracy                           0.83      3000
   macro avg       0.78      0.63      0.65      3000
weighted avg       0.81      0.83      0.80      3000
```