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

```
dataset = pd.read_csv('Churn_Modelling.csv', index_col = 'RowNumber')
dataset.head()
```

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

```
#Customer ID and Surname would not be relevant as features
X_columns = dataset.columns.tolist()[2:12]
Y_columns = dataset.columns.tolist()[-1:]
print(X_columns)
print(Y_columns)
```

```
['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
['Exited']
```

```
X = dataset[X_columns].values
Y = dataset[Y_columns].values
```

```
#We need to encode categorical variables such as geography and gender
from sklearn.preprocessing import LabelEncoder
X_column_transformer = LabelEncoder()
X[:, 1] = X_column_transformer.fit_transform(X[:, 1])
```

```
#Lets Encode gender now
X[:, 2] = X_column_transformer.fit_transform(X[:, 2])
```

```
#We are treating countries with ordinal values(0 < 1 < 2) but they are incomparable.
#To solve this we can use one hot encoding.
#We will perform some standardization
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

pipeline = Pipeline(
    [
        ('Categorizer', ColumnTransformer(
            [
                ("Gender Label Encoder", OneHotEncoder(categories = 'auto', drop = 'first'), [2]),
                ("Geography Label Encoder", OneHotEncoder(categories = 'auto', drop = 'first'), [1])
            ],
            remainder = 'passthrough', n_jobs = 1)),
        ('Normalizer', StandardScaler())
    ]
)
```

```
#Standardize the features
X = pipeline.fit_transform(X)
```

```
#Spilt the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
```

```
#Let us create the Neural Network
from keras.models import Sequential
from keras.layers import Dense, Dropout


#Initialize ANN
classifier = Sequential()


#Add input layer and hidden layer
classifier.add(Dense(6, activation = 'relu', input_shape = (X_train.shape[1], )))
classifier.add(Dropout(rate = 0.1))


#Add second layer
classifier.add(Dense(6, activation = 'relu'))
classifier.add(Dropout(rate = 0.1))


#Add output layer
classifier.add(Dense(1, activation = 'sigmoid'))


#Let us take a look at our network
classifier.summary()
```

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

 dropout (Dropout)           (None, 6)                 0

 dense_1 (Dense)             (None, 6)                 42

 dropout_1 (Dropout)         (None, 6)                 0

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121 (484.00 Byte)
Trainable params: 121 (484.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#Optimize the weights
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])


#Fitting the Neural Network
history = classifier.fit(X_train, y_train, batch_size = 32, epochs = 200, validation_split = 0.1, verbose = 2)
```

```
225/225 - 0s - loss: 0.3553 - accuracy: 0.8562 - val_loss: 0.3191 - val_accuracy: 0.8725 - 373ms/epoch - 2ms/step
Epoch 187/200
225/225 - 0s - loss: 0.3599 - accuracy: 0.8519 - val_loss: 0.3201 - val_accuracy: 0.8763 - 391ms/epoch - 2ms/step
Epoch 188/200
225/225 - 0s - loss: 0.3638 - accuracy: 0.8479 - val_loss: 0.3208 - val_accuracy: 0.8725 - 410ms/epoch - 2ms/step
Epoch 189/200
225/225 - 0s - loss: 0.3615 - accuracy: 0.8535 - val_loss: 0.3217 - val_accuracy: 0.8725 - 396ms/epoch - 2ms/step
Epoch 190/200
225/225 - 1s - loss: 0.3591 - accuracy: 0.8533 - val_loss: 0.3209 - val_accuracy: 0.8737 - 508ms/epoch - 2ms/step
Epoch 191/200
225/225 - 1s - loss: 0.3620 - accuracy: 0.8508 - val_loss: 0.3210 - val_accuracy: 0.8750 - 576ms/epoch - 3ms/step
Epoch 192/200
225/225 - 1s - loss: 0.3600 - accuracy: 0.8518 - val_loss: 0.3208 - val_accuracy: 0.8725 - 564ms/epoch - 3ms/step
Epoch 193/200
225/225 - 1s - loss: 0.3567 - accuracy: 0.8558 - val_loss: 0.3216 - val_accuracy: 0.8725 - 557ms/epoch - 2ms/step
Epoch 194/200
225/225 - 1s - loss: 0.3552 - accuracy: 0.8535 - val_loss: 0.3195 - val_accuracy: 0.8700 - 531ms/epoch - 2ms/step
Epoch 195/200
225/225 - 0s - loss: 0.3557 - accuracy: 0.8562 - val_loss: 0.3204 - val_accuracy: 0.8700 - 395ms/epoch - 2ms/step
Epoch 196/200
225/225 - 0s - loss: 0.3583 - accuracy: 0.8514 - val_loss: 0.3203 - val_accuracy: 0.8763 - 379ms/epoch - 2ms/step
Epoch 197/200
225/225 - 0s - loss: 0.3624 - accuracy: 0.8517 - val_loss: 0.3193 - val_accuracy: 0.8763 - 360ms/epoch - 2ms/step
Epoch 198/200
225/225 - 0s - loss: 0.3584 - accuracy: 0.8531 - val_loss: 0.3192 - val_accuracy: 0.8763 - 390ms/epoch - 2ms/step
Epoch 199/200
225/225 - 0s - loss: 0.3653 - accuracy: 0.8490 - val_loss: 0.3208 - val_accuracy: 0.8750 - 345ms/epoch - 2ms/step
Epoch 200/200
225/225 - 0s - loss: 0.3624 - accuracy: 0.8512 - val_loss: 0.3213 - val_accuracy: 0.8737 - 341ms/epoch - 2ms/step
```

```
y_pred = classifier.predict(X_test)
print(y_pred[:5])
```

```
63/63 [==============================] - 0s 1ms/step
[[0.26967978]
 [0.23207372]
 [0.16839103]
 [0.08661045]
 [0.09321575]]
```

```
#Let us use confusion matrix with cutoff value as 0.5
y_pred = (y_pred > 0.5).astype(int)
print(y_pred[:5])
```

```
[[0]
 [0]
 [0]
 [0]
 [0]]
```

```
#Making the Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1525   70]
 [ 200  205]]
```

```
#Accuracy of our NN
print(((cm[0][0] + cm[1][1])* 100) / len(y_test), '% of data was classified correctly')
```

```
86.5 % of data was classified correctly
```