In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
sns.set()
```

In [ ]:
```
dataset = pd.read_csv('7431_Churn_Modelling.csv', index_col = 'RowNumber')
dataset.head()
```

Out[ ]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	E
1	15634602	Hargrave	619	France	Female	42	2	
2	15647311	Hill	608	Spain	Female	41	1	8:
3	15619304	Onio	502	France	Female	42	8	15!
4	15701354	Boni	699	France	Female	39	1	
5	15737888	Mitchell	850	Spain	Female	43	2	12!

In [4]:
```
#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', 'NumOfProduct
s', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
['Exited']
```

In [5]:
```
X = dataset[X_columns].values
Y = dataset[Y_columns].values
```

In [6]:
```
#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])
```

In [7]:
```
#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

In [8]:
```
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
```

```
                ("Geography Label Encoder", OneHotEncoder(categories = 'auto', d
            ],
            remainder = 'passthrough', n_jobs = 1)),
        ('Normalizer', StandardScaler())
    ]
)
```

In [9]:
```
#Standardize the features
X = pipeline.fit_transform(X)
```

In [10]:
```
#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, randc
```

In [11]:
```
#Let us create the Neural Network
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

In [12]:
```
#Initialize ANN
classifier = Sequential()
```

In [13]:
```
#Add input layer and hidden layer
classifier.add(Dense(6, activation = 'relu', input_shape = (X_train.shape[1], ))
classifier.add(Dropout(rate = 0.1))
```

```
c:\Users\loken\Downloads\Lokendra ML\.venv\Lib\site-packages\keras\src\layers\cor
e\dense.py:92: UserWarning: Do not pass an `input_shape`/`input_dim` argument to
a layer. When using Sequential models, prefer using an `Input(shape)` object as t
he first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [14]:
```
#Add second layer
classifier.add(Dense(6, activation = 'relu'))
classifier.add(Dropout(rate = 0.1))
```

In [15]:
```
#Add output layer
classifier.add(Dense(1, activation = 'sigmoid'))
```

In [16]:
```
#Let us take a look at our network
classifier.summary()
```

Model: "sequential"

Layer (type)	Output Shape	I
dense (Dense)	(None, 6)	
dropout (Dropout)	(None, 6)	
dense_1 (Dense)	(None, 6)	
dropout_1 (Dropout)	(None, 6)	
dense_2 (Dense)	(None, 1)	

Total params: 121 (484.00 B)

Trainable params: 121 (484.00 B)

```
Non-trainable params: 0 (0.00 B)
```

In [17]:
```
#Optimize the weights
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = [
```

In [ ]:
```
#Fitting the Neural Network
history = classifier.fit(X_train, y_train, batch_size = 32, epochs = 200, valida
```

In [19]:
```
y_pred = classifier.predict(X_test)
print(y_pred[:5])
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
[[0.23755181]
 [0.28248632]
 [0.21702741]
 [0.10188767]
 [0.11005695]]
[[0.23755181]
 [0.28248632]
 [0.21702741]
 [0.10188767]
 [0.11005695]]
```

In [20]:
```
#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]]
```

In [21]:
```
#Making the Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1560   35]
 [ 232  173]]
```

In [22]:
```
#Accuracy of our NN
print(((cm[0][0] + cm[1][1])* 100) / len(y_test), '% of data was classified corr
```

```
86.65 % of data was classified correctly
```