As per the given employee turnover, we are going to predict customer churn using dataset

Import Libraries

In [1]:

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

In [2]:

```
# tensorflow is designed to efficiently handle large-scale numerical computations using
import tensorflow as tf
tf.__version__
```

Out[2]:

```
'2.10.0'
```

In [3]:

```
data = pd.read_csv(r"D:\Sabina\ANN\30th,31st\ANN_ 1st\Churn_Modelling.csv")
```

In [4]:

```
# top 5 data
data.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8380
2	3	15619304	Onio	502	France	Female	42	8	15966
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12551

In [5]:

```
data.columns
```

Out[5]:

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

In [6]:

```
data.shape
```

Out[6]:

```
(10000, 14)
```

In [7]:

```
# Checking the data types of all the columns
data.dtypes
```

Out[7]:

```
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
```

In [8]:

```
data.describe()
```

Out[8]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

In [9]:

```
# Checking any null/missing values

data.isnull().sum()
```

Out[9]:

```
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
```

no missing value

In [10]:

```
data['Exited'].value_counts()
```

Out[10]:

```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

In [11]:

```
100*data['Exited'].value_counts()/len(data['Exited'])
```

Out[11]:

```
Exited
0    79.63
1    20.37
Name: count, dtype: float64
```

In [12]:

```
# first 3 columns are not imp which not afftected on result so remove it
X = data.iloc[:, 3:-1].values
X
```

Out[12]:

```
array([[619, 'France', 'Female', ..., 1, 1, 101348.88],
       [608, 'Spain', 'Female', ..., 0, 1, 112542.58],
       [502, 'France', 'Female', ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 'Female', ..., 0, 1, 42085.58],
       [772, 'Germany', 'Male', ..., 1, 0, 92888.52],
       [792, 'France', 'Female', ..., 1, 0, 38190.78]], dtype=object)
```

In [13]:

```
y = data.iloc[:, -1].values
y
```

Out[13]:

```
array([1, 0, 1, ..., 1, 1, 0], dtype=int64)
```

Encoding categorical data to numerical data by LabelEncoder

In [14]:

```
# Label Encoding the "Gender" column
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

In [15]:

```
X
```

Out[15]:

```
array([[619, 'France', 0, ..., 1, 1, 101348.88],
       [608, 'Spain', 0, ..., 0, 1, 112542.58],
       [502, 'France', 0, ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 0, ..., 0, 1, 42085.58],
       [772, 'Germany', 1, ..., 1, 0, 92888.52],
       [792, 'France', 0, ..., 1, 0, 38190.78]], dtype=object)
```

so gender column converted into numerical as female = 0 and male = 1

One Hot Encoding

In [16]:

```
# One Hot Encoding the "Geography" column
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [17]:

```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='pass
```

In [18]:

```
X =np.array(ct.fit_transform(X))
X
```

Out[18]:

```
array([[1.0, 0.0, 0.0, ..., 1, 1, 101348.88],
       [0.0, 0.0, 1.0, ..., 0, 1, 112542.58],
       [1.0, 0.0, 0.0, ..., 1, 0, 113931.57],
       ...,
       [1.0, 0.0, 0.0, ..., 0, 1, 42085.58],
       [0.0, 1.0, 0.0, ..., 1, 0, 92888.52],
       [1.0, 0.0, 0.0, ..., 1, 0, 38190.78]], dtype=object)
```

for each category, a separate binary column is created, and a value of 1 is placed in the corresponding column while the rest are filled with zeros

Feature Scaling

In [19]:

```
# scale down all values between 0-1
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [20]:

```
sc
```

Out[20]:

```
StandardScaler()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [21]:

```
X = sc.fit_transform(X)
```

In [22]:

```
X
```

Out[22]:

```
array([[ 0.99720391, -0.57873591, -0.57380915, ...,  0.64609167,
         0.97024255,  0.02188649],
       [-1.00280393, -0.57873591,  1.74273971, ..., -1.54776799,
         0.97024255,  0.21653375],
       [ 0.99720391, -0.57873591, -0.57380915, ...,  0.64609167,
        -1.03067011,  0.2406869 ],
       ...,
       [ 0.99720391, -0.57873591, -0.57380915, ..., -1.54776799,
         0.97024255, -1.00864308],
       [-1.00280393,  1.72790383, -0.57380915, ...,  0.64609167,
        -1.03067011, -0.12523071],
       [ 0.99720391, -0.57873591, -0.57380915, ...,  0.64609167,
        -1.03067011, -1.07636976]])
```

In [23]:

```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
```

In [24]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
```

Part 2 - Building the ANN

```
# Initializing the ANN
```

In [25]:

```
#Layers are stacked one after another in a linear motion
ann = tf.keras.models.Sequential()
```

In [26]:

```
# Adding the input layer and the first hidden layer

# dense =i/p & hidden layer connection where each neuron is connected to every neuron in
#units = represents the number of neurons (nodes)
# relu (Rectified Linear Unit) it is a multiclass activation function(range==0 to infini

ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

In [27]:

```
# Adding the second hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

In [28]:

```
# Adding the output layer
# units=1 for binary classification (0 or 1/ yes or no)value represent the probability o
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Part 3 - Training the ANN

In [29]:

```
# Compiling the ANN
# adam (Adaptive Moment Estimation)
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [30]:

```
# Convert the NumPy array to a TensorFlow tensor
try:
    tf_tensor = tf.convert_to_tensor(data)
except ValueError as e:
    print("Error:", e)
```

```
Error: Failed to convert a NumPy array to a Tensor (Unsupported object typ
e int).
```

In [31]:

```
# Training the ANN on the Training set
ann.fit(X_train, y_train,batch_size = 6,epochs = 20)
```

```
Epoch 1/20
1334/1334 [==============================] - 2s 943us/step - loss: 0.5006
- accuracy: 0.7839
Epoch 2/20
1334/1334 [==============================] - 1s 922us/step - loss: 0.4327
- accuracy: 0.8037
Epoch 3/20
1334/1334 [==============================] - 1s 938us/step - loss: 0.4222
- accuracy: 0.8154
Epoch 4/20
1334/1334 [==============================] - 1s 957us/step - loss: 0.4136
- accuracy: 0.8214
Epoch 5/20
1334/1334 [==============================] - 1s 937us/step - loss: 0.3997
- accuracy: 0.8304
Epoch 6/20
1334/1334 [==============================] - 1s 941us/step - loss: 0.3767
- accuracy: 0.8418
Epoch 7/20
1334/1334 [==============================] - 1s 929us/step - loss: 0.3587
- accuracy: 0.8518
Epoch 8/20
1334/1334 [==============================] - 1s 960us/step - loss: 0.3520
- accuracy: 0.8531
Epoch 9/20
1334/1334 [==============================] - 1s 967us/step - loss: 0.3483
- accuracy: 0.8591
Epoch 10/20
1334/1334 [==============================] - 1s 945us/step - loss: 0.3463
- accuracy: 0.8583
Epoch 11/20
1334/1334 [==============================] - 1s 951us/step - loss: 0.3434
- accuracy: 0.8595
Epoch 12/20
1334/1334 [==============================] - 1s 947us/step - loss: 0.3421
- accuracy: 0.8590
Epoch 13/20
1334/1334 [==============================] - 1s 971us/step - loss: 0.3417
- accuracy: 0.8596
Epoch 14/20
1334/1334 [==============================] - 1s 974us/step - loss: 0.3408
- accuracy: 0.8595
Epoch 15/20
1334/1334 [==============================] - 1s 949us/step - loss: 0.3402
- accuracy: 0.8610
Epoch 16/20
1334/1334 [==============================] - 1s 968us/step - loss: 0.3402
- accuracy: 0.8601
Epoch 17/20
1334/1334 [==============================] - 1s 974us/step - loss: 0.3399
- accuracy: 0.8606
Epoch 18/20
1334/1334 [==============================] - 1s 951us/step - loss: 0.3391
- accuracy: 0.8593
Epoch 19/20
1334/1334 [==============================] - 1s 954us/step - loss: 0.3391
- accuracy: 0.8599
Epoch 20/20
1334/1334 [==============================] - 1s 952us/step - loss: 0.3384
- accuracy: 0.8615
```

Out[31]:

```
<keras.callbacks.History at 0x1f654312aa0>
```

In [32]:

```
# Predicting the Test set results
y_pred = ann.predict(X_test)
```

```
63/63 [==============================] - 0s 726us/step
```

In [33]:

```
# If predicted value is greater than 5 then its true
y_pred = (y_pred > 0.5)
y_pred
```

Out[33]:

```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [34]:

```
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

In [35]:

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

```
[[1515   80]
 [ 200  205]]
```

In [36]:

```
from sklearn.metrics import accuracy_score
ac = accuracy_score(y_test, y_pred)
print(ac)
```

```
0.86
```

predicting a single new observation

In [37]:

```
"""Predict if the customer with the following informations will leave the bank:
Geography: France
Credit Score: 600
Gender: Male
Age: 40
Tenure: 3
Balance: 60000
Number of Products: 2
Has Credit Card: Yes
Is Active Member: Yes
Estimated Salary: 50000"""
```

Out[37]:

```
'Predict if the customer with the following informations will leave the ba
nk:\nGeography: France\nCredit Score: 600\nGender: Male\nAge: 40\nTenure:
3\nBalance: 60000\nNumber of Products: 2\nHas Credit Card: Yes\nIs Active
Member: Yes\nEstimated Salary: 50000'
```

In [38]:

```
new_pred = ann.predict(sc.transform(np.array([[0.0,1.0, 0, 600, 1, 40, 3, 60000, 2, 1, 1
new_pred = (new_pred > 0.5)
```

```
1/1 [==============================] - 0s 21ms/step
```

In [39]:

```
print(new_pred)
```

```
[[False]]
```

In [40]:

```
"""Predict if the customer with the following informations will stay:
Geography: Spain
Credit Score: 850
Gender: Male
Age: 32
Tenure: 2
Balance: 2000
Number of Products: 4
Has Credit Card: Yes
Is Active Member: Yes
Estimated Salary: 145797"""
```

Out[40]:

```
'Predict if the customer with the following informations will stay:\nGeogr
aphy: Spain\nCredit Score: 850\nGender: Male\nAge: 32\nTenure: 2\nBalance:
2000\nNumber of Products: 4\nHas Credit Card: Yes\nIs Active Member: Yes\n
Estimated Salary: 145797'
```

In [41]:

```
new_pred = ann.predict(sc.transform(np.array([[0.0,0.0, 1.0, 850, 1, 32, 2, 2000, 4, 1,
new_pred2 = (new_pred > 0.5)
```

```
1/1 [==============================] - 0s 25ms/step
```

In [42]:

```
print(new_pred2)
```

```
[[ True]]
```

finally here model is trained & predict if the customer stays or exit the bank according to input value

In [ ]: