```
import numpy as np
import pandas as pd
import tensorflow as tf
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remoun
```

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

```
df.head()
```

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

```
# for X, take 4th column to untill last second column as independent variable and exited as dependent variable
X = df.iloc[:, 3:-1].values
y = df.iloc[:, -1].values
```

```
X
```

```
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)
```

```
y
```

```
array([1, 0, 1, ..., 1, 1, 0])
```

```
#lebel encodding for gender column
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

```
#one hot encoding for country column, for this first import all necesary moddule
#Then, define the transformer function which has list of transformer which transform value into one hot
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

```
#define column transfer function where instruct which column need to apply the function
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])], remainder='passthrough')
```

```
#apply the CT to X
X = np.array(ct.fit_transform(X))
```

```
#feature scaling with std_scaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
X
```

```
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]])
```

```
#spliting the dataset intotrain and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

Double-click (or enter) to edit

Building **ANN**

```
ann = tf.keras.models.Sequential()
```

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

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

```
#adding output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

Adam stands for Adaptive Moment Estimation, and it's an extension of the stochastic gradient descent (SGD) algorithm. Adam adapts the learning rates of each parameter based on the past gradients and their squared gradients.

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

```
#training the ANN with training set with epoch 120
ann.fit(X_train, y_train, batch_size = 32, epochs = 120)
```

```
Epoch 1/120
235/235 [==============================] - 3s 4ms/step - loss: 0.5634 - accuracy: 0.7853
Epoch 2/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4856 - accuracy: 0.8003
Epoch 3/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4570 - accuracy: 0.8048
Epoch 4/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4432 - accuracy: 0.8108
Epoch 5/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4350 - accuracy: 0.8144
Epoch 6/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4285 - accuracy: 0.8160
Epoch 7/120
235/235 [==============================] - 1s 2ms/step - loss: 0.4232 - accuracy: 0.8176
Epoch 8/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4188 - accuracy: 0.8211
Epoch 9/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4150 - accuracy: 0.8236
Epoch 10/120
235/235 [==============================] - 0s 2ms/step - loss: 0.4108 - accuracy: 0.8268
Epoch 11/120
235/235 [==============================] - 1s 3ms/step - loss: 0.4065 - accuracy: 0.8299
Epoch 12/120
235/235 [==============================] - 1s 3ms/step - loss: 0.4024 - accuracy: 0.8331
Epoch 13/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3978 - accuracy: 0.8349
Epoch 14/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3927 - accuracy: 0.8373
Epoch 15/120
235/235 [==============================] - 1s 4ms/step - loss: 0.3875 - accuracy: 0.8392
Epoch 16/120
235/235 [==============================] - 1s 6ms/step - loss: 0.3818 - accuracy: 0.8431
Epoch 17/120
235/235 [==============================] - 1s 5ms/step - loss: 0.3764 - accuracy: 0.8432
Epoch 18/120
235/235 [==============================] - 1s 3ms/step - loss: 0.3712 - accuracy: 0.8452
Epoch 19/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3654 - accuracy: 0.8467
```

```
Epoch 20/120
235/235 [==============================] - 1s 2ms/step - loss: 0.3606 - accuracy: 0.8520
Epoch 21/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3566 - accuracy: 0.8533
Epoch 22/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3533 - accuracy: 0.8564
Epoch 23/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3504 - accuracy: 0.8567
Epoch 24/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3482 - accuracy: 0.8579
Epoch 25/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3462 - accuracy: 0.8585
Epoch 26/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3440 - accuracy: 0.8599
Epoch 27/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3425 - accuracy: 0.8597
Epoch 28/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3411 - accuracy: 0.8593
Epoch 29/120
235/235 [==============================] - 0s 2ms/step - loss: 0.3402 - accuracy: 0.8605
```

```
# predict with x test and see the orginal y test and ypred line by line
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
79/79 [==============================] - 0s 3ms/step
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1878  113]
 [ 231  278]]
```