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

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

```
    Mounted at /content/drive
```

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

```
df.head()
```

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

```
# 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 **MNN**

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

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

# Adding the fourth 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 = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
#training the ANN with training set with epoch 120
ann.fit(X_train, y_train, batch_size = 45, epochs = 130)
```

```
    Epoch 114/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3534 - accuracy:
    Epoch 115/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3534 - accuracy:
    Epoch 116/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3527 - accuracy:
    Epoch 117/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3532 - accuracy:
    Epoch 118/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3528 - accuracy:
    Epoch 119/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3528 - accuracy:
    Epoch 120/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3521 - accuracy:
    Epoch 121/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3525 - accuracy:
    Epoch 122/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3523 - accuracy:
    Epoch 123/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3521 - accuracy:
    Epoch 124/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3520 - accuracy:
    Epoch 125/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3519 - accuracy:
    Epoch 126/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3517 - accuracy:
    Epoch 127/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3515 - accuracy:
    Epoch 128/130
    167/167 [==============================] - 0s 3ms/step - loss: 0.3515 - accuracy:
    Epoch 129/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3510 - accuracy:
    Epoch 130/130
    167/167 [==============================] - 0s 2ms/step - loss: 0.3513 - accuracy:
    <keras.callbacks.History at 0x7ae4f43097e0>
```

nobel barua
04:07 Today

Take 2 minute for epoch 120 and MNN

```
# 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 1ms/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)
```

```
    [[1883  108]
     [ 261  248]]
```

✓ 0s completed at 04:39