```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, accuracy_score

# pip install tensorflow ignore if already installed

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

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

dataset.head()
```

```
           CustomerId   Surname  CreditScore Geography  Gender  Age
Tenure  \
RowNumber

1            15634602  Hargrave          619    France  Female   42
2
2            15647311      Hill          608     Spain  Female   41
1
3            15619304      Onio          502    France  Female   42
8
4            15701354      Boni          699    France  Female   39
1
5            15737888  Mitchell          850     Spain  Female   43
2

              Balance  NumOfProducts  HasCrCard  IsActiveMember  \
RowNumber
1                0.00              1          1               1
2            83807.86              1          0               1
3           159660.80              3          1               0
4                0.00              2          0               0
5           125510.82              1          1               1

           EstimatedSalary  Exited
RowNumber
1                101348.88       1
2                112542.58       0
3                113931.57       1
4                 93826.63       0
5                 79084.10       0
```

```
# Features and target
X = dataset.iloc[:, 2:12]
X  # drop CustomerId and Surname
```

```
            CreditScore Geography  Gender  Age  Tenure    Balance  \
RowNumber
1                   619    France  Female   42       2       0.00
2                   608     Spain  Female   41       1   83807.86
3                   502    France  Female   42       8  159660.80
4                   699    France  Female   39       1       0.00
5                   850     Spain  Female   43       2  125510.82
...                 ...       ...     ...  ...     ...        ...
9996                771    France    Male   39       5       0.00
9997                516    France    Male   35      10   57369.61
9998                709    France  Female   36       7       0.00
9999                772   Germany    Male   42       3   75075.31
10000               792    France  Female   28       4  130142.79

            NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary
RowNumber
1                       1          1               1        101348.88
2                       1          0               1        112542.58
3                       3          1               0        113931.57
4                       2          0               0         93826.63
5                       1          1               1         79084.10
...                   ...        ...             ...              ...
9996                    2          1               0         96270.64
9997                    1          1               1        101699.77
9998                    1          0               1         42085.58
9999                    2          1               0         92888.52
10000                   1          1               0         38190.78

[10000 rows x 10 columns]
```

```
# Features and target
Y = dataset.iloc[:, 12].values  # Churn column
Y
```

```
array([1, 0, 1, ..., 1, 1, 0], shape=(10000,))
```

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

```
# Pipeline: One-hot encode categorical and scale
pipeline = Pipeline([
    ('preprocess', ColumnTransformer(
        transformers=[
            ('gender', OneHotEncoder(drop='first'), ['Gender']),
            ('geo', OneHotEncoder(drop='first'), ['Geography'])
        ],
        remainder='passthrough'
    )),
    ('scaler', StandardScaler())
])
```

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

#Spilt the data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.2, random_state = 0)

# Build ANN
classifier = Sequential()
classifier.add(Dense(6, activation='relu',
input_shape=(X_train.shape[1],)))
classifier.add(Dropout(0.1))
classifier.add(Dense(6, activation='relu'))
classifier.add(Dropout(0.1))
classifier.add(Dense(1, activation='sigmoid'))
```

```
C:\Users\vishw\AppData\Roaming\Python\Python313\site-packages\keras\
src\layers\core\dense.py:93: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in
the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
# Train ANN
history = classifier.fit(X_train, y_train, batch_size=32, epochs=100,
validation_split=0.1, verbose=2)
```

```
Epoch 1/100
225/225 - 3s - 13ms/step - accuracy: 0.7376 - loss: 0.5860 -
val_accuracy: 0.7950 - val_loss: 0.4805
Epoch 2/100
225/225 - 1s - 3ms/step - accuracy: 0.7969 - loss: 0.4791 -
val_accuracy: 0.7962 - val_loss: 0.4492
Epoch 3/100
225/225 - 1s - 4ms/step - accuracy: 0.7997 - loss: 0.4619 -
val_accuracy: 0.8000 - val_loss: 0.4358
Epoch 4/100
225/225 - 1s - 3ms/step - accuracy: 0.8011 - loss: 0.4548 -
val_accuracy: 0.8012 - val_loss: 0.4274
Epoch 5/100
225/225 - 1s - 3ms/step - accuracy: 0.8039 - loss: 0.4429 -
val_accuracy: 0.7987 - val_loss: 0.4211
Epoch 6/100
225/225 - 1s - 3ms/step - accuracy: 0.8036 - loss: 0.4396 -
val_accuracy: 0.8025 - val_loss: 0.4170
Epoch 7/100
225/225 - 1s - 3ms/step - accuracy: 0.8053 - loss: 0.4388 -
val_accuracy: 0.8050 - val_loss: 0.4138
Epoch 8/100
```

```
225/225 - 1s - 3ms/step - accuracy: 0.8049 - loss: 0.4343 -
val_accuracy: 0.8087 - val_loss: 0.4110
Epoch 9/100
225/225 - 1s - 3ms/step - accuracy: 0.8082 - loss: 0.4308 -
val_accuracy: 0.8100 - val_loss: 0.4072
Epoch 10/100
225/225 - 1s - 3ms/step - accuracy: 0.8058 - loss: 0.4287 -
val_accuracy: 0.8150 - val_loss: 0.4036
Epoch 11/100
225/225 - 1s - 3ms/step - accuracy: 0.8081 - loss: 0.4291 -
val_accuracy: 0.8138 - val_loss: 0.4001
Epoch 12/100
225/225 - 1s - 3ms/step - accuracy: 0.8110 - loss: 0.4197 -
val_accuracy: 0.8100 - val_loss: 0.3967
Epoch 13/100
225/225 - 1s - 3ms/step - accuracy: 0.8140 - loss: 0.4167 -
val_accuracy: 0.8125 - val_loss: 0.3931
Epoch 14/100
225/225 - 1s - 3ms/step - accuracy: 0.8158 - loss: 0.4138 -
val_accuracy: 0.8100 - val_loss: 0.3893
Epoch 15/100
225/225 - 1s - 3ms/step - accuracy: 0.8188 - loss: 0.4087 -
val_accuracy: 0.8050 - val_loss: 0.3866
Epoch 16/100
225/225 - 1s - 3ms/step - accuracy: 0.8157 - loss: 0.4071 -
val_accuracy: 0.8062 - val_loss: 0.3847
Epoch 17/100
225/225 - 1s - 3ms/step - accuracy: 0.8160 - loss: 0.4031 -
val_accuracy: 0.8075 - val_loss: 0.3818
Epoch 18/100
225/225 - 1s - 3ms/step - accuracy: 0.8196 - loss: 0.4023 -
val_accuracy: 0.8087 - val_loss: 0.3797
Epoch 19/100
225/225 - 1s - 3ms/step - accuracy: 0.8181 - loss: 0.4024 -
val_accuracy: 0.8138 - val_loss: 0.3764
Epoch 20/100
225/225 - 1s - 6ms/step - accuracy: 0.8215 - loss: 0.3960 -
val_accuracy: 0.8112 - val_loss: 0.3741
Epoch 21/100
225/225 - 1s - 3ms/step - accuracy: 0.8203 - loss: 0.3952 -
val_accuracy: 0.8100 - val_loss: 0.3723
Epoch 22/100
225/225 - 1s - 4ms/step - accuracy: 0.8194 - loss: 0.3945 -
val_accuracy: 0.8087 - val_loss: 0.3709
Epoch 23/100
225/225 - 1s - 3ms/step - accuracy: 0.8183 - loss: 0.3952 -
val_accuracy: 0.8087 - val_loss: 0.3693
Epoch 24/100
225/225 - 1s - 3ms/step - accuracy: 0.8203 - loss: 0.3938 -
```

```
val_accuracy: 0.8075 - val_loss: 0.3674
Epoch 25/100
225/225 - 1s - 3ms/step - accuracy: 0.8201 - loss: 0.3891 -
val_accuracy: 0.8087 - val_loss: 0.3649
Epoch 26/100
225/225 - 1s - 3ms/step - accuracy: 0.8208 - loss: 0.3884 -
val_accuracy: 0.8112 - val_loss: 0.3635
Epoch 27/100
225/225 - 1s - 3ms/step - accuracy: 0.8189 - loss: 0.3886 -
val_accuracy: 0.8100 - val_loss: 0.3620
Epoch 28/100
225/225 - 1s - 3ms/step - accuracy: 0.8186 - loss: 0.3883 -
val_accuracy: 0.8112 - val_loss: 0.3612
Epoch 29/100
225/225 - 1s - 3ms/step - accuracy: 0.8206 - loss: 0.3914 -
val_accuracy: 0.8112 - val_loss: 0.3596
Epoch 30/100
225/225 - 1s - 3ms/step - accuracy: 0.8190 - loss: 0.3877 -
val_accuracy: 0.8125 - val_loss: 0.3582
Epoch 31/100
225/225 - 1s - 3ms/step - accuracy: 0.8196 - loss: 0.3852 -
val_accuracy: 0.8112 - val_loss: 0.3572
Epoch 32/100
225/225 - 1s - 3ms/step - accuracy: 0.8196 - loss: 0.3853 -
val_accuracy: 0.8112 - val_loss: 0.3566
Epoch 33/100
225/225 - 1s - 3ms/step - accuracy: 0.8206 - loss: 0.3824 -
val_accuracy: 0.8112 - val_loss: 0.3548
Epoch 34/100
225/225 - 1s - 3ms/step - accuracy: 0.8193 - loss: 0.3843 -
val_accuracy: 0.8125 - val_loss: 0.3537
Epoch 35/100
225/225 - 1s - 3ms/step - accuracy: 0.8204 - loss: 0.3803 -
val_accuracy: 0.8100 - val_loss: 0.3524
Epoch 36/100
225/225 - 1s - 3ms/step - accuracy: 0.8199 - loss: 0.3818 -
val_accuracy: 0.8125 - val_loss: 0.3516
Epoch 37/100
225/225 - 1s - 3ms/step - accuracy: 0.8201 - loss: 0.3779 -
val_accuracy: 0.8475 - val_loss: 0.3513
Epoch 38/100
225/225 - 1s - 3ms/step - accuracy: 0.8326 - loss: 0.3773 -
val_accuracy: 0.8487 - val_loss: 0.3500
Epoch 39/100
225/225 - 1s - 3ms/step - accuracy: 0.8479 - loss: 0.3785 -
val_accuracy: 0.8475 - val_loss: 0.3487
Epoch 40/100
225/225 - 1s - 4ms/step - accuracy: 0.8462 - loss: 0.3771 -
val_accuracy: 0.8562 - val_loss: 0.3486
```

```
Epoch 41/100
225/225 - 1s - 6ms/step - accuracy: 0.8451 - loss: 0.3808 -
val_accuracy: 0.8537 - val_loss: 0.3489
Epoch 42/100
225/225 - 1s - 7ms/step - accuracy: 0.8468 - loss: 0.3801 -
val_accuracy: 0.8537 - val_loss: 0.3472
Epoch 43/100
225/225 - 1s - 6ms/step - accuracy: 0.8475 - loss: 0.3790 -
val_accuracy: 0.8550 - val_loss: 0.3472
Epoch 44/100
225/225 - 1s - 6ms/step - accuracy: 0.8489 - loss: 0.3800 -
val_accuracy: 0.8575 - val_loss: 0.3476
Epoch 45/100
225/225 - 1s - 6ms/step - accuracy: 0.8492 - loss: 0.3794 -
val_accuracy: 0.8512 - val_loss: 0.3467
Epoch 46/100
225/225 - 1s - 6ms/step - accuracy: 0.8483 - loss: 0.3768 -
val_accuracy: 0.8575 - val_loss: 0.3465
Epoch 47/100
225/225 - 1s - 6ms/step - accuracy: 0.8494 - loss: 0.3855 -
val_accuracy: 0.8575 - val_loss: 0.3460
Epoch 48/100
225/225 - 1s - 7ms/step - accuracy: 0.8476 - loss: 0.3807 -
val_accuracy: 0.8575 - val_loss: 0.3456
Epoch 49/100
225/225 - 1s - 6ms/step - accuracy: 0.8489 - loss: 0.3796 -
val_accuracy: 0.8562 - val_loss: 0.3452
Epoch 50/100
225/225 - 1s - 6ms/step - accuracy: 0.8493 - loss: 0.3768 -
val_accuracy: 0.8537 - val_loss: 0.3447
Epoch 51/100
225/225 - 1s - 5ms/step - accuracy: 0.8478 - loss: 0.3780 -
val_accuracy: 0.8600 - val_loss: 0.3461
Epoch 52/100
225/225 - 1s - 3ms/step - accuracy: 0.8490 - loss: 0.3771 -
val_accuracy: 0.8575 - val_loss: 0.3442
Epoch 53/100
225/225 - 1s - 3ms/step - accuracy: 0.8474 - loss: 0.3767 -
val_accuracy: 0.8550 - val_loss: 0.3435
Epoch 54/100
225/225 - 1s - 4ms/step - accuracy: 0.8493 - loss: 0.3771 -
val_accuracy: 0.8625 - val_loss: 0.3438
Epoch 55/100
225/225 - 1s - 3ms/step - accuracy: 0.8489 - loss: 0.3774 -
val_accuracy: 0.8587 - val_loss: 0.3430
Epoch 56/100
225/225 - 1s - 3ms/step - accuracy: 0.8497 - loss: 0.3814 -
val_accuracy: 0.8600 - val_loss: 0.3444
Epoch 57/100
```

```
225/225 - 1s - 4ms/step - accuracy: 0.8493 - loss: 0.3787 -
val_accuracy: 0.8612 - val_loss: 0.3431
Epoch 58/100
225/225 - 1s - 3ms/step - accuracy: 0.8499 - loss: 0.3732 -
val_accuracy: 0.8575 - val_loss: 0.3425
Epoch 59/100
225/225 - 1s - 3ms/step - accuracy: 0.8535 - loss: 0.3791 -
val_accuracy: 0.8575 - val_loss: 0.3426
Epoch 60/100
225/225 - 1s - 3ms/step - accuracy: 0.8494 - loss: 0.3774 -
val_accuracy: 0.8637 - val_loss: 0.3428
Epoch 61/100
225/225 - 1s - 3ms/step - accuracy: 0.8524 - loss: 0.3769 -
val_accuracy: 0.8612 - val_loss: 0.3420
Epoch 62/100
225/225 - 1s - 3ms/step - accuracy: 0.8486 - loss: 0.3801 -
val_accuracy: 0.8612 - val_loss: 0.3424
Epoch 63/100
225/225 - 1s - 3ms/step - accuracy: 0.8507 - loss: 0.3811 -
val_accuracy: 0.8600 - val_loss: 0.3435
Epoch 64/100
225/225 - 1s - 3ms/step - accuracy: 0.8537 - loss: 0.3798 -
val_accuracy: 0.8562 - val_loss: 0.3423
Epoch 65/100
225/225 - 1s - 3ms/step - accuracy: 0.8515 - loss: 0.3744 -
val_accuracy: 0.8587 - val_loss: 0.3427
Epoch 66/100
225/225 - 1s - 3ms/step - accuracy: 0.8511 - loss: 0.3787 -
val_accuracy: 0.8600 - val_loss: 0.3427
Epoch 67/100
225/225 - 1s - 3ms/step - accuracy: 0.8499 - loss: 0.3780 -
val_accuracy: 0.8587 - val_loss: 0.3428
Epoch 68/100
225/225 - 1s - 3ms/step - accuracy: 0.8510 - loss: 0.3767 -
val_accuracy: 0.8612 - val_loss: 0.3424
Epoch 69/100
225/225 - 1s - 3ms/step - accuracy: 0.8472 - loss: 0.3807 -
val_accuracy: 0.8600 - val_loss: 0.3427
Epoch 70/100
225/225 - 1s - 3ms/step - accuracy: 0.8494 - loss: 0.3739 -
val_accuracy: 0.8600 - val_loss: 0.3421
Epoch 71/100
225/225 - 1s - 3ms/step - accuracy: 0.8490 - loss: 0.3784 -
val_accuracy: 0.8600 - val_loss: 0.3424
Epoch 72/100
225/225 - 1s - 3ms/step - accuracy: 0.8508 - loss: 0.3779 -
val_accuracy: 0.8600 - val_loss: 0.3415
Epoch 73/100
225/225 - 1s - 3ms/step - accuracy: 0.8496 - loss: 0.3802 -
```

```
val_accuracy: 0.8587 - val_loss: 0.3425
Epoch 74/100
225/225 - 1s - 3ms/step - accuracy: 0.8506 - loss: 0.3785 -
val_accuracy: 0.8587 - val_loss: 0.3418
Epoch 75/100
225/225 - 1s - 3ms/step - accuracy: 0.8504 - loss: 0.3757 -
val_accuracy: 0.8600 - val_loss: 0.3409
Epoch 76/100
225/225 - 1s - 3ms/step - accuracy: 0.8490 - loss: 0.3778 -
val_accuracy: 0.8600 - val_loss: 0.3404
Epoch 77/100
225/225 - 1s - 3ms/step - accuracy: 0.8531 - loss: 0.3749 -
val_accuracy: 0.8637 - val_loss: 0.3399
Epoch 78/100
225/225 - 1s - 3ms/step - accuracy: 0.8472 - loss: 0.3771 -
val_accuracy: 0.8612 - val_loss: 0.3410
Epoch 79/100
225/225 - 1s - 3ms/step - accuracy: 0.8471 - loss: 0.3778 -
val_accuracy: 0.8600 - val_loss: 0.3411
Epoch 80/100
225/225 - 1s - 3ms/step - accuracy: 0.8515 - loss: 0.3762 -
val_accuracy: 0.8587 - val_loss: 0.3404
Epoch 81/100
225/225 - 1s - 3ms/step - accuracy: 0.8535 - loss: 0.3756 -
val_accuracy: 0.8600 - val_loss: 0.3401
Epoch 82/100
225/225 - 1s - 3ms/step - accuracy: 0.8479 - loss: 0.3772 -
val_accuracy: 0.8600 - val_loss: 0.3398
Epoch 83/100
225/225 - 1s - 3ms/step - accuracy: 0.8525 - loss: 0.3785 -
val_accuracy: 0.8587 - val_loss: 0.3400
Epoch 84/100
225/225 - 1s - 3ms/step - accuracy: 0.8494 - loss: 0.3747 -
val_accuracy: 0.8575 - val_loss: 0.3420
Epoch 85/100
225/225 - 1s - 3ms/step - accuracy: 0.8508 - loss: 0.3796 -
val_accuracy: 0.8600 - val_loss: 0.3401
Epoch 86/100
225/225 - 1s - 3ms/step - accuracy: 0.8490 - loss: 0.3788 -
val_accuracy: 0.8600 - val_loss: 0.3400
Epoch 87/100
225/225 - 1s - 3ms/step - accuracy: 0.8489 - loss: 0.3773 -
val_accuracy: 0.8612 - val_loss: 0.3411
Epoch 88/100
225/225 - 1s - 3ms/step - accuracy: 0.8512 - loss: 0.3753 -
val_accuracy: 0.8587 - val_loss: 0.3403
Epoch 89/100
225/225 - 1s - 3ms/step - accuracy: 0.8485 - loss: 0.3780 -
val_accuracy: 0.8612 - val_loss: 0.3410
```

```
Epoch 90/100
225/225 - 1s - 3ms/step - accuracy: 0.8508 - loss: 0.3762 -
val_accuracy: 0.8575 - val_loss: 0.3398
Epoch 91/100
225/225 - 1s - 3ms/step - accuracy: 0.8510 - loss: 0.3771 -
val_accuracy: 0.8600 - val_loss: 0.3403
Epoch 92/100
225/225 - 1s - 3ms/step - accuracy: 0.8504 - loss: 0.3724 -
val_accuracy: 0.8612 - val_loss: 0.3402
Epoch 93/100
225/225 - 1s - 3ms/step - accuracy: 0.8503 - loss: 0.3770 -
val_accuracy: 0.8625 - val_loss: 0.3400
Epoch 94/100
225/225 - 1s - 3ms/step - accuracy: 0.8508 - loss: 0.3744 -
val_accuracy: 0.8637 - val_loss: 0.3401
Epoch 95/100
225/225 - 1s - 3ms/step - accuracy: 0.8529 - loss: 0.3772 -
val_accuracy: 0.8650 - val_loss: 0.3404
Epoch 96/100
225/225 - 1s - 3ms/step - accuracy: 0.8511 - loss: 0.3750 -
val_accuracy: 0.8600 - val_loss: 0.3393
Epoch 97/100
225/225 - 1s - 3ms/step - accuracy: 0.8468 - loss: 0.3778 -
val_accuracy: 0.8637 - val_loss: 0.3393
Epoch 98/100
225/225 - 1s - 3ms/step - accuracy: 0.8497 - loss: 0.3785 -
val_accuracy: 0.8625 - val_loss: 0.3397
Epoch 99/100
225/225 - 1s - 3ms/step - accuracy: 0.8508 - loss: 0.3777 -
val_accuracy: 0.8637 - val_loss: 0.3402
Epoch 100/100
225/225 - 1s - 3ms/step - accuracy: 0.8489 - loss: 0.3810 -
val_accuracy: 0.8587 - val_loss: 0.3390

# Predict
y_pred = (classifier.predict(X_test) > 0.5).astype(int)

63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

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

63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
[[0.32752857]
 [0.3487643 ]
 [0.22522898]
 [0.08598081]
 [0.07558867]]
```

```python
#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]]
```

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

```
[[1545   50]
 [ 220  185]]
```

```python
#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
```