```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn import preprocessing
```

Loading the Dataset

First we load the dataset and find out the number of columns, rows, NULL values, etc.

```
df = pd.read_csv('churn_modelling.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Esti
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	

Next steps: Generate code with df | View recommended plots | New interactive sheet

Cleaning

```
df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], inplace=True)
```

```
df.isna().sum()
```

	0
CreditScore	0
Geography	0
Gender	0
Age	0
Tenure	0
Balance	0
NumOfProducts	0
HasCrCard	0
IsActiveMember	0
EstimatedSalary	0
Exited	0

```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

Separating the features and the labels

```
X=df.iloc[:, :df.shape[1]-1].values       #Independent Variables
y=df.iloc[:, -1].values                   #Dependent Variable
X.shape, y.shape
```

```
((10000, 10), (10000,))
```

Encoding categorical (string based) data.

```
print(X[:8,1], '... will now become: ')

label_X_country_encoder = LabelEncoder()
X[:,1] = label_X_country_encoder.fit_transform(X[:,1])
print(X[:8,1])
```

```
['France' 'Spain' 'France' 'France' 'Spain' 'Spain' 'France' 'Germany'] ... will now become:
[0 2 0 0 2 2 0 1]
```

```
print(X[:6,2], '... will now become: ')

label_X_gender_encoder = LabelEncoder()
X[:,2] = label_X_gender_encoder.fit_transform(X[:,2])
print(X[:6,2])
```

```
['Female' 'Female' 'Female' 'Female' 'Female' 'Male'] ... will now become:
[0 0 0 0 0 1]
```

Split the countries into respective dimensions. Converting the string features into their own dimensions.

```
transform = ColumnTransformer([("countries", OneHotEncoder(), [1])], remainder="passthrough") # 1 is the country column
X = transform.fit_transform(X)
X
```

```
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)
```

Dimensionality reduction. A 0 on two countries means that the country has to be the one variable which wasn't included

```
X = X[:,1:]
X.shape
```

```
(10000, 11)
```

Splitting the Dataset

Training and Test Set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Normalize the train and test data

`['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']`

```
sc=StandardScaler()
X_train[:,np.array([2,4,5,6,7,10])] = sc.fit_transform(X_train[:,np.array([2,4,5,6,7,10])])
X_test[:,np.array([2,4,5,6,7,10])] = sc.transform(X_test[:,np.array([2,4,5,6,7,10])])
```

```
sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train
```

```
array([[-0.5698444 ,  1.74309049,  0.16958176, ...,  0.64259497,
        -1.03227043,  1.10643166],
       [ 1.75486502, -0.57369368, -2.30455945, ...,  0.64259497,
         0.9687384 , -0.74866447],
       [-0.5698444 , -0.57369368, -1.19119591, ...,  0.64259497,
        -1.03227043,  1.48533467],
       ...,
       [-0.5698444 , -0.57369368,  0.9015152 , ...,  0.64259497,
        -1.03227043,  1.41231994],
       [-0.5698444 ,  1.74309049, -0.62420521, ...,  0.64259497,
         0.9687384 ,  0.84432121],
       [ 1.75486502, -0.57369368, -0.28401079, ...,  0.64259497,
        -1.03227043,  0.32472465]])
```

Initialize & build the model

`INPUT = Number columns (Independet ) HIDDEN - AF HIDDEN -AF . . . N OUTPUT (1,2) -Sigmoid`

```
from tensorflow.keras.models import Sequential

# Initializing the ANN
classifier = Sequential()
```

```
from tensorflow.keras.layers import Dense

# The amount of nodes (dimensions) in hidden layer should be the average of input and output layers, in this case 6.
# This adds the input layer (by specifying input dimension) AND the first hidden layer (units)
classifier.add(Dense(activation = 'relu', input_dim = 11, units=256, kernel_initializer='uniform'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Adding the hidden layer
classifier.add(Dense(activation = 'relu', units=512, kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=256, kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=128, kernel_initializer='uniform'))


# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output (stay with the bank or not)
# We use the sigmoid because we want probability outcomes
classifier.add(Dense(activation = 'sigmoid', units=1, kernel_initializer='uniform'))


# Create optimizer with default learning rate
# sgd_optimizer = tf.keras.optimizers.SGD()
# Compile the model
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


classifier.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	3,072
dense_1 (Dense)	(None, 512)	131,584
dense_2 (Dense)	(None, 256)	131,328
dense_3 (Dense)	(None, 128)	32,896
dense_4 (Dense)	(None, 1)	129

Total params: 299,009 (1.14 MB)
Trainable params: 299,009 (1.14 MB)
Non-trainable params: 0 (0.00 B)

```
classifier.fit(
    X_train, y_train,
    validation_data=(X_test,y_test),
    epochs=20,
    batch_size=32
)
```

```
Epoch 1/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.7998 - loss: 0.4757 - val_accuracy: 0.8500 - val_loss: 0.3747
Epoch 2/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8476 - loss: 0.3693 - val_accuracy: 0.8585 - val_loss: 0.3537
Epoch 3/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8585 - loss: 0.3455 - val_accuracy: 0.8605 - val_loss: 0.3418
Epoch 4/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8597 - loss: 0.3438 - val_accuracy: 0.8660 - val_loss: 0.3360
Epoch 5/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.8626 - loss: 0.3355 - val_accuracy: 0.8675 - val_loss: 0.3397
Epoch 6/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8625 - loss: 0.3375 - val_accuracy: 0.8505 - val_loss: 0.3506
Epoch 7/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8605 - loss: 0.3407 - val_accuracy: 0.8605 - val_loss: 0.3379
Epoch 8/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.8565 - loss: 0.3308 - val_accuracy: 0.8660 - val_loss: 0.3420
Epoch 9/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.8687 - loss: 0.3196 - val_accuracy: 0.8595 - val_loss: 0.3446
Epoch 10/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8683 - loss: 0.3183 - val_accuracy: 0.8600 - val_loss: 0.3394
Epoch 11/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8723 - loss: 0.3121 - val_accuracy: 0.8590 - val_loss: 0.3411
Epoch 12/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8793 - loss: 0.3070 - val_accuracy: 0.8600 - val_loss: 0.3690
Epoch 13/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.8772 - loss: 0.3069 - val_accuracy: 0.8630 - val_loss: 0.3303
Epoch 14/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.8758 - loss: 0.3108 - val_accuracy: 0.8530 - val_loss: 0.3652
Epoch 15/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8822 - loss: 0.2981 - val_accuracy: 0.8555 - val_loss: 0.3535
Epoch 16/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.8794 - loss: 0.2998 - val_accuracy: 0.8605 - val_loss: 0.3591
```

```
Epoch 17/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.8768 - loss: 0.3016 - val_accuracy: 0.8555 - val_loss: 0.3811
Epoch 18/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.8863 - loss: 0.2831 - val_accuracy: 0.8600 - val_loss: 0.3627
Epoch 19/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.8948 - loss: 0.2637 - val_accuracy: 0.8620 - val_loss: 0.3642
Epoch 20/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.8934 - loss: 0.2682 - val_accuracy: 0.8595 - val_loss: 0.3675
<keras.src.callbacks.history.History at 0x7c254bd1a1a0>
```

Predict the results using 0.5 as a threshold

```
y_pred = classifier.predict(X_test)
y_pred
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
array([[0.38601685],
       [0.2562771 ],
       [0.05166654],
       ...,
       [0.01091674],
       [0.2623498 ],
       [0.15966687]], dtype=float32)
```

```
# To use the confusion Matrix, we need to convert the probabilities that a customer will leave the bank into the form true or false.
# So we will use the cutoff value 0.5 to indicate whether they are likely to exit or not.
y_pred = (y_pred > 0.5)
y_pred
```

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

Print the Accuracy score and confusion matrix

```
from sklearn.metrics import confusion_matrix,classification_report

cm1 = confusion_matrix(y_test, y_pred)
cm1
```

```
array([[1511,   84],
       [ 197,  208]])
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      1595
           1       0.71      0.51      0.60       405

    accuracy                           0.86      2000
   macro avg       0.80      0.73      0.76      2000
weighted avg       0.85      0.86      0.85      2000
```

```
accuracy_model1 = ((cm1[0][0]+cm1[1][1])*100)/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0])
print (accuracy_model1, '% of testing data was classified correctly')
```

```
85.95 % of testing data was classified correctly
```