```
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
```

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

df.info()

df.head()
```

```
<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
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenu
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	

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

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

```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProduc
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.0000
mean	650.528800	38.921800	5.012800	76485.889288	1.5302
std	96.653299	10.487806	2.892174	62397.405202	0.5816
min	350.000000	18.000000	0.000000	0.000000	1.0000
25%	584.000000	32.000000	3.000000	0.000000	1.0000
50%	652.000000	37.000000	5.000000	97198.540000	1.0000
75%	718.000000	44.000000	7.000000	127644.240000	2.0000
max	850.000000	92.000000	10.000000	250898.090000	4.0000

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

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

```
['France' 'Spain' 'France' 'France' 'Spain' 'Spain' 'France' 'Germany'] ...
```

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

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])
```

```
[0 2 0 0 2 2 0 1]
['Female' 'Female' 'Female' 'Female' 'Female' 'Male'] ... will now become:
[0 0 0 0 0 1]
```

```
transform = ColumnTransformer([("countries", OneHotEncoder(), [1])], remainder="
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)
```

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

```
(10000, 11)
```

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

```
sc=StandardScaler()
X_train[:,np.array([2,4,5,6,7,10])] = sc.fit_transform(X_train[:,np.array([2,4,5
X_test[:,np.array([2,4,5,6,7,10])] = sc.transform(X_test[:,np.array([2,4,5,6,7,1
```

```
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 -
```

```
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 inpu
# This adds the input layer (by specifying input dimension) AND the first hidden
classifier.add(Dense(activation = 'relu', input_dim = 11, units=256, kernel_init
```

```
# 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
# We use the sigmoid because we want probability outcomes
classifier.add(Dense(activation = 'sigmoid', units=1, kernel_initializer='unifor
```

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

classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               3072

 dense_1 (Dense)             (None, 512)               131584

 dense_2 (Dense)             (None, 256)               131328

 dense_3 (Dense)             (None, 128)               32896

 dense_4 (Dense)             (None, 1)                 129

=================================================================
Total params: 299009 (1.14 MB)
Trainable params: 299009 (1.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
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 - loss: 0.4234 - acc
Epoch 2/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3626 - acc
Epoch 3/20
250/250 [==============================] - 2s 9ms/step - loss: 0.3500 - acc
Epoch 4/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3430 - ac
Epoch 5/20
250/250 [==============================] - 3s 12ms/step - loss: 0.3395 - ac
Epoch 6/20
250/250 [==============================] - 3s 10ms/step - loss: 0.3337 - ac
Epoch 7/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3294 - acc
Epoch 8/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3272 - acc
Epoch 9/20
250/250 [==============================] - 2s 9ms/step - loss: 0.3242 - acc
Epoch 10/20
250/250 [==============================] - 4s 14ms/step - loss: 0.3243 - ac
Epoch 11/20
250/250 [==============================] - 2s 8ms/step - loss: 0.3188 - acc
Epoch 12/20
250/250 [==============================] - 2s 9ms/step - loss: 0.3138 - acc
Epoch 13/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3113 - ac
Epoch 14/20
250/250 [==============================] - 2s 9ms/step - loss: 0.3094 - acc
Epoch 15/20
250/250 [==============================] - 3s 11ms/step - loss: 0.3030 - ac
Epoch 16/20
250/250 [==============================] - 2s 8ms/step - loss: 0.2989 - acc
Epoch 17/20
250/250 [==============================] - 3s 12ms/step - loss: 0.2988 - ac
Epoch 18/20
250/250 [==============================] - 3s 14ms/step - loss: 0.2940 - ac
Epoch 19/20
250/250 [==============================] - 5s 19ms/step - loss: 0.2853 - ac
Epoch 20/20
250/250 [==============================] - 3s 13ms/step - loss: 0.2780 - ac
<keras.src.callbacks.History at 0x786149a80af0>
```

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

```
63/63 [==============================] - 0s 2ms/step
array([[0.43192458],
       [0.21820262],
```

```
           [0.10865229],
           ...,
           [0.22263762],
           [0.25179225],
           [0.2263274 ]], dtype=float32)
```

```
# To use the confusion Matrix, we need to convert the probabilities that a custo
# So we will use the cutoff value 0.5 to indicate whether they are likely to exi
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([[1486,  109],
           [ 185,  220]])
```

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

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

```
                  precision    recall  f1-score   support

               0       0.89      0.93      0.91      1595
               1       0.67      0.54      0.60       405

        accuracy                           0.85      2000
       macro avg       0.78      0.74      0.75      2000
    weighted avg       0.84      0.85      0.85      2000

    85.3 % of testing data was classified correctly
```