```
import pandas as pd
import numpy as np
```

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

```
df
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	838
2	3	15619304	Onio	502	France	Female	42	8	1596
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1255
...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	573
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	750
9999	10000	15628319	Walker	792	France	Female	28	4	130

10000 rows × 14 columns

```
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.duplicated().sum()
```

```
0
```

```
df['Exited'].value_counts()
```

```
0    7963
1    2037
Name: Exited, dtype: int64
```

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

```
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1

```
df['Geography'].value_counts()
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

Geography & Gender are two categorical columns we have to enhotencoding

```
df=pd.get_dummies(df, columns=['Geography','Gender'],drop_first=True)
```

```
df
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germany	Geograph
0	619	42	2	0.00	1	1	1	101348.88	1	0	
1	608	41	1	83807.86	1	0	1	112542.58	0	0	
2	502	42	8	159660.80	3	1	0	113931.57	1	0	
3	699	39	1	0.00	2	0	0	93826.63	0	0	
4	850	43	2	125510.82	1	1	1	79084.10	0	0	
...	...	...	...	...	...	...	...	...	...	...	
9995	771	39	5	0.00	2	1	0	96270.64	0	0	
9996	516	35	10	57369.61	1	1	1	101699.77	0	0	
9997	709	36	7	0.00	1	0	1	42085.58	1	0	
9998	772	42	3	75075.31	2	1	0	92888.52	1	1	
9999	792	28	4	130142.79	1	1	0	38190.78	0	0	

10000 rows × 12 columns

Now all data is in numeric format Now we have to scale it

```
X =df.drop(columns = ['Exited'])
y = df['Exited']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.37, random_state = 1)
```

```
X
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germany	Geography_Spain
0	619	42	2	0.00	1	1	1	101348.88	0	0
1	608	41	1	83807.86	1	0	1	112542.58	0	1
2	502	42	8	159660.80	3	1	0	113931.57	0	0

```
y
```

```
0       1
1       0
2       1
3       0
4       0
       ..
9995    0
9996    0
9997    1
9998    1
9999    0
Name: Exited, Length: 10000, dtype: int64
```

10000 rows × 11 columns

```
X_train.shape
```

```
(6300, 11)
```

Scaling

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled
X_train_scaled
```

```
array([[ 0.70682183, -1.22769729,  0.68227257, ...,  1.71751077,
        -0.57026295,  0.91502478],
       [ 0.35484626, -0.3702992 , -0.70294084, ...,  1.71751077,
        -0.57026295,  0.91502478],
       [-0.64931934,  0.6776318 ,  1.37487928, ...,  1.71751077,
        -0.57026295,  0.91502478],
       ...,
       [ 0.23061959,  0.58236535,  1.37487928, ..., -0.58223798,
        -0.57026295, -1.09286658],
       [ 0.13744958,  0.01076662,  1.02857592, ..., -0.58223798,
        -0.57026295, -1.09286658],
       [ 1.17267185,  0.29656599,  0.33596922, ...,  1.71751077,
        -0.57026295,  0.91502478]])
```

```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

Adding Layer

```
model = Sequential()
model.add(Dense(3,activation = 'sigmoid', input_dim =11))###input layer(11 inputcolumn thay why input_dem =11,3 perseptrons
model.add(Dense(1,activation = 'sigmoid'))###output layer
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 3)                 36

 dense_5 (Dense)             (None, 1)                 4
```

```
=================================================================
Total params: 40
Trainable params: 40
Non-trainable params: 0
_________________________________________________________________
```

Compaling

```
model.compile(loss='binary_crossentropy', optimizer ='adam')
```

Last stage - Model fitting

```
model.fit(X_train_scaled, y_train, epochs =100)### epochs =10 mhnaje 10 veles ikd tikde firun weight manage krun loss kami krun result
```

```
197/197 [==============================] - 0s 2ms/step - loss: 0.3834
Epoch 71/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3829
Epoch 72/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3826
Epoch 73/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3822
Epoch 74/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3819
Epoch 75/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3816
Epoch 76/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3811
Epoch 77/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3809
Epoch 78/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3806
Epoch 79/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3802
Epoch 80/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3800
Epoch 81/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3796
Epoch 82/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3793
Epoch 83/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3791
Epoch 84/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3788
Epoch 85/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3786
Epoch 86/100
197/197 [==============================] - 1s 3ms/step - loss: 0.3783
Epoch 87/100
197/197 [==============================] - 1s 3ms/step - loss: 0.3780
Epoch 88/100
197/197 [==============================] - 1s 3ms/step - loss: 0.3777
Epoch 89/100
197/197 [==============================] - 1s 3ms/step - loss: 0.3775
Epoch 90/100
197/197 [==============================] - 1s 3ms/step - loss: 0.3772
Epoch 91/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3769
Epoch 92/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3767
Epoch 93/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3765
Epoch 94/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3763
Epoch 95/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3761
Epoch 96/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3759
Epoch 97/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3756
Epoch 98/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3754
Epoch 99/100
197/197 [==============================] - 0s 2ms/step - loss: 0.3752
```

Now The Model is trained

```
model.layers[0].get_weights()### weights ani baises che values kuth strore ahe ky ahe te baghnya sathi ithe 1st layers cha value varch weight
```

```
[array([[-0.08282412,  0.14011528, -0.05184836],
        [ 0.4905517 , -0.28554642, -4.1824164 ],
        [ 0.18129183,  0.13490632, -0.18391083],
        [ 1.4684751 , -0.8974749 , -0.39438874],
        [ 4.0245447 , -0.96995765, -0.78182447],
        [ 0.1215133 , -0.0399824 , -0.02228319],
        [ 0.6813985 ,  1.5501534 , -0.43020228],
        [-0.09740294,  0.01135234, -0.04295088],
        [-1.1847771 , -0.55903536,  0.08964276],
        [ 0.16218469, -0.37382394,  0.06361698],
        [ 0.02013901,  0.7753729 ,  0.20163739]], dtype=float32),
 array([2.084026  , 0.79584706, 0.5450206 ], dtype=float32)]
```

```
model.layers[1].get_weights()###Second layes che 3 weights ani tych 1 bais
```

```
[array([[-2.0268068],
        [-1.7711706],
        [-2.3128417]], dtype=float32), array([2.006123], dtype=float32)]
```

Now we are doing prediction

```
model.predict(X_train_scaled)### apn test sathi je thevle hote tyche sagly che result pahayla
```

```
197/197 [==============================] - 0s 1ms/step
array([[0.2508253 ],
       [0.1269836 ],
       [0.6790319 ],
       ...,
       [0.39025342],
       [0.07834914],
       [0.42484897]], dtype=float32)
```

Ithe output 1 kiva zero nahiye karn apn "sigmoid" use krto ani "sigmoid" ch output he 0 to 1 cha probably mdhe ast

Values 0 ani 1 form mdhe anych asel tr "threshod" decide krva lagto..mhanje.. 0.5 = "threshod" tr value 0.4 asel tr op 0 ani 0.5 cha vr asel tr 1

```
y_log = model.predict(X_train_scaled) ###y_log veriable mdhe prediction odel save kel
```

```
197/197 [==============================] - 0s 1ms/step
```

```
np.where(y_log>0.5,1,0)###y_log 0.5 peksha modh asel tr op 1 nasel tr 0
```

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

```
y_pred = np.where(y_log>0.5,1,0)### y_pred variable mdhesave kel
```

Ata Model chi accuracy check kraychi

```
from sklearn.metrics import accuracy_score
```

"pd.DataFrame.from_records" hi commond ya problem mdhech run keli karn ""ValueError: Found input variables with inconsistent numbers of samples: [1, 29]"" ha error yet hot Reason -- Given your data, it looks like you have 6 features. In that case, try to convert your X to have 29 rows and 6 columns. Then pass that dataframe to train_test_split. You can convert your list to dataframe using pd.DataFrame.from_records. Answer --Thanks for the help Sal! You're right, I just had to convert it to the same lengths. My X.shape was (1, 6, 29) and Y.shape was (29,). I just had to reshape them and it all worked fine for me :)

```
pd.DataFrame.from_records###You can convert your list to dataframe using pd.DataFrame.from_records.
```

```
<bound method DataFrame.from_records of <class 'pandas.core.frame.DataFrame'>>
```

```
accuracy_score(y_train,y_pred)
```

```
0.8326984126984127
```

Accuracy increase krnya sathi ky kru skto 1) epoches vathavu skto 2) activation function'relu' dhevu shkto 3) no. of nodes padhvu shkto ithe 3 ahe 10, 30 as accuracy vatvu shkto 4) no. of hidden layer vadhvaycha

```
import matplotlib.pyplot as plt
```

0s completed at 4:52 PM