Churn Modelling Project

This data set contains details of a bank's customers and the target variable is a binary variable reflecting the fact whether the customer left the bank

(closed his account) or he continues to be a customer.

```
# import basic libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
# TO load dataset
df=pd.read_csv('/content/drive/MyDrive/Palak Deep Learning/Churn_Modelling.csv')
```

```
# To show first 5 record
df.head()
```

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

```
# To show the numbers of rows and columns
df.shape
```

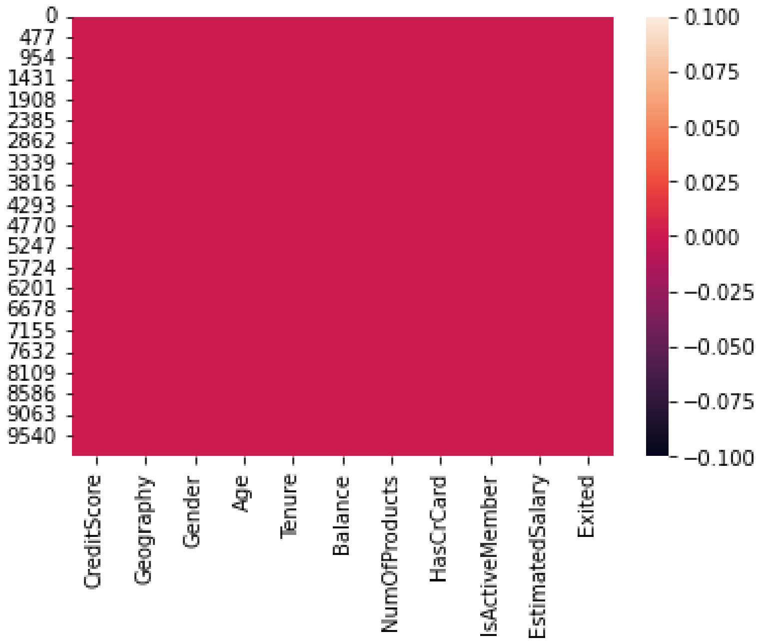
```
(10000, 14)
```

```
# To delete unwanted features permanently in dataset
df.drop(["RowNumber","CustomerId","Surname"],axis=1,inplace=True)
```

```
# TO show the null values
df.isnull().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
```

```
# To visualize the null value
sns.heatmap(df.isnull())
plt.show()
```




```
# To check the duplicates rows
df.duplicated().sum()
```

```
0
```

```
# To show the datatypes
df.dtypes
```

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
```

```
Exited              int64
dtype: object
```

```
# create 1st datafram df_cat to hold the categorical data
df_cat=df.select_dtypes(object)
```

```
# creating 2nd dataframe df_num to hold numeric data
df_num=df.select_dtypes(['int64','float64'])
```

```
# converting all the categorical data into numeric data
# we use labelEncoder to convert the data
from sklearn.preprocessing import LabelEncoder
# create object of Label Encoder
for col in df_cat:
   le=LabelEncoder()
   df_cat[col]=le.fit_transform(df_cat[[col]])
```

```
# concatenating of both the dataframe df_cat and df_new and hold into df_new
df_new=pd.concat([df_cat,df_num],axis=1)
```

```
# New dataframe
df_new.head()
```

	Geography	Gender	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActi
0	0	0	619	42	2	0.00	1	1	
1	2	0	608	41	1	83807.86	1	0	
2	0	0	502	42	8	159660.80	3	1	
3	0	0	699	39	1	0.00	2	0	
4	2	0	850	43	2	125510.82	1	1	

```
df_new.dtypes
```

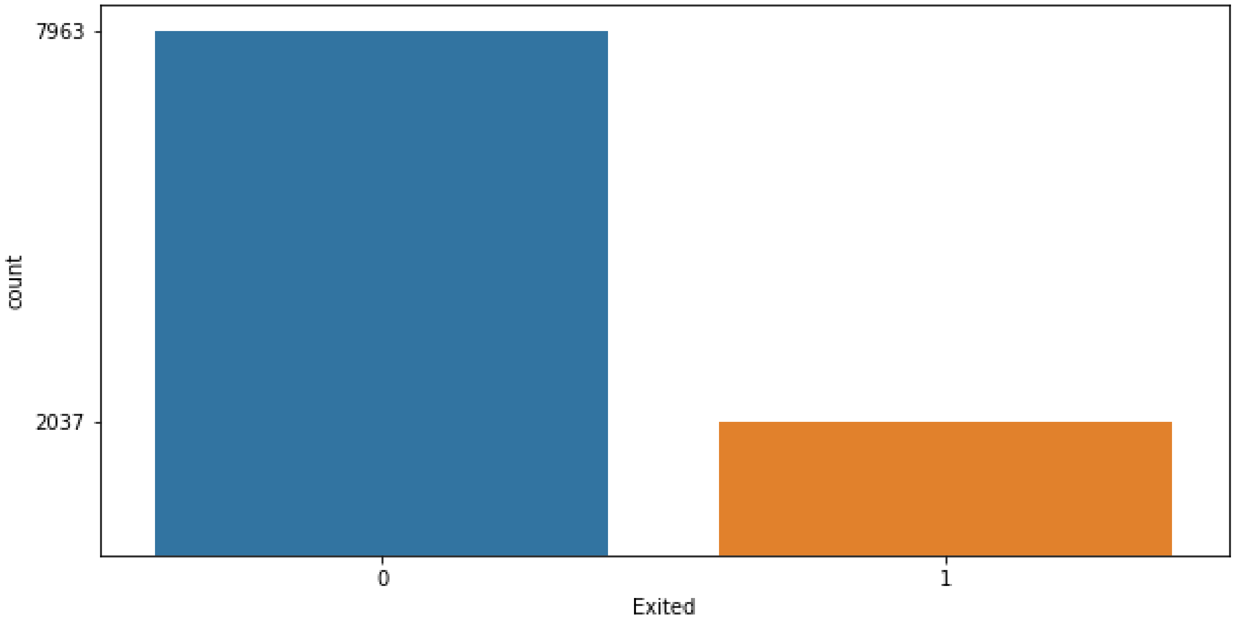
```
Geography            int64
Gender               int64
CreditScore          int64
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
# first check how many samples in both classes
df['Exited'].value_counts()
```

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

Here as we can see the data is not equaly distributed. we have to devide the data into equally part

```
# Visualization the unbalance data
plt.figure(figsize=(10,5))
sns.countplot(data=df,x='Exited')
f = df['Exited'].value_counts()
plt.yticks(f)
plt.show()
```




```
# select input and output
X=df_new.drop("Exited",axis=1) # input
Y=df_new["Exited"] # output
```

```
# Train Test split the data
from sklearn.model_selection import train_test_split
X_train ,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=1)
```

```
## Apply scaling (Standard Scaler)
from sklearn.preprocessing import StandardScaler
# create object of StandardScaler class
ss=StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.transform(X_test)
```

```
X.shape
```

```
(10000, 10)
```

Appling ' RandomOverSampler ' to make data equal.

From the above value_counts of 'Exited' Column and from the Graph There are about: '7963' out of '2037' Which means the Data is not properly distributed we have to divided the data qually part then, use RandomOverSampler First import RandomOverSampler

```
# Apply random over sampling : inbuild class
from imblearn.over_sampling import RandomOverSampler
```

```
# Create the object of RandomOverSampler
ros = RandomOverSampler(random_state=1)
```

```
# Before apply RandomOverSampler on training data
Y_train.value_counts()
```

```
0    5590
1    1410
Name: Exited, dtype: int64
```

```
# Applied OverSampler on Training data (70%)
X_train_ros,Y_train_ros = ros.fit_resample(X_train,Y_train)
```

```
# Check after apply RandomOverSampler
Y_train_ros.value_counts()
```

```
0    5590
1    5590
Name: Exited, dtype: int64
```

```
# Before apply RandomOverSampler on testing data
Y_test.value_counts()
```

```
0    2373
1     627
Name: Exited, dtype: int64
```

```
# Also apply RandomOverSampler on tesing data (30%)
X_test_ros,Y_test_ros = ros.fit_resample(X_test,Y_test)
```

```
Y_test_ros.value_counts()
```

```
0    2373
1    2373
Name: Exited, dtype: int64
```

Apply Neural Network

```
#First step when we take Neurons only 10,. means  Total Number of Neurons = Total  Nu
#  create a neural network
import tensorflow as tf
# create object of sequential class
model=tf.keras.Sequential([
       tf.keras.layers.Dense(units=10,activation='relu',input_shape=(X.shape[1],)),
       # hidden layer 1
       tf.keras.layers.Dense(units=10,activation='relu'), # second hidden layer
       tf.keras.layers.Dense(units=1,activation='sigmoid') # output layer
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 10)                110       
                                                                 
 dense_1 (Dense)             (None, 10)                110       
                                                                 
 dense_2 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 231
Trainable params: 231
Non-trainable params: 0
_________________________________________________________________
```

```
# Complie the model
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
```

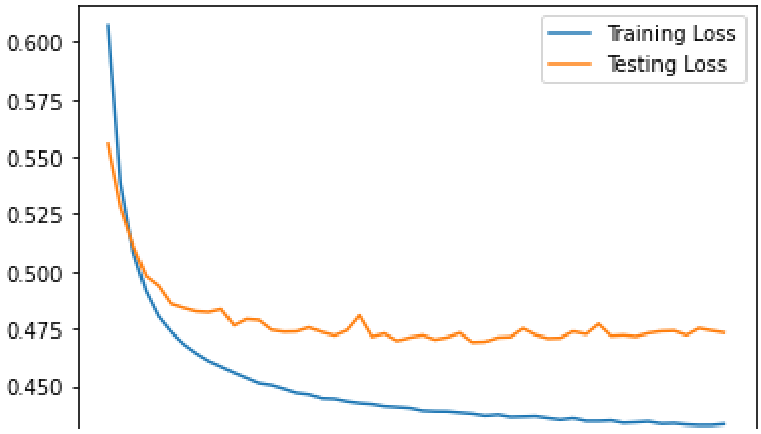
```
 #Create Early Stopping  means create a call back
from tensorflow.keras.callbacks import EarlyStopping
callback=EarlyStopping(
    monitor="val_loss",
    min_delta=0.00001,
    patience=20,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=False
)
```

```
# Train the Model
trained_model = model.fit(X_train_ros,Y_train_ros,batch_size=20,epochs=3500,validatic
```

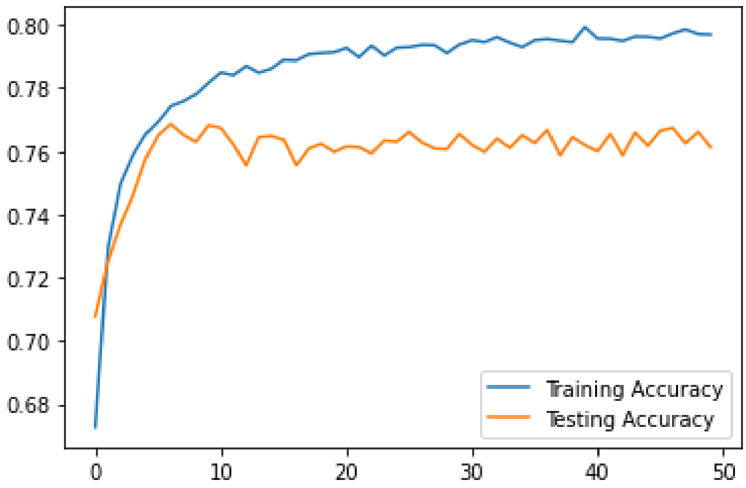
```
Epoch 1/3500
559/559 [==============================] - 2s 2ms/step - loss: 0.6072 - accuracy: 0.6727
Epoch 2/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5385 - accuracy: 0.7296
Epoch 3/3500
```

```
559/559 [==============================] - 2s 4ms/step - loss: 0.5086 - accuracy: 0.7497
Epoch 4/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4914 - accuracy: 0.7589
Epoch 5/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4803 - accuracy: 0.7654
Epoch 6/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4736 - accuracy: 0.7694
Epoch 7/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4683 - accuracy: 0.7742
Epoch 8/3500
559/559 [==============================] - 3s 5ms/step - loss: 0.4643 - accuracy: 0.7758
Epoch 9/3500
559/559 [==============================] - 3s 5ms/step - loss: 0.4609 - accuracy: 0.7780
Epoch 10/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4585 - accuracy: 0.7817
Epoch 11/3500
559/559 [==============================] - 3s 5ms/step - loss: 0.4558 - accuracy: 0.7850
Epoch 12/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4536 - accuracy: 0.7841
Epoch 13/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4513 - accuracy: 0.7870
Epoch 14/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4504 - accuracy: 0.7849
Epoch 15/3500
559/559 [==============================] - 2s 3ms/step - loss: 0.4488 - accuracy: 0.7861
Epoch 16/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4470 - accuracy: 0.7889
Epoch 17/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4463 - accuracy: 0.7887
Epoch 18/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4445 - accuracy: 0.7907
Epoch 19/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4443 - accuracy: 0.7911
Epoch 20/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4432 - accuracy: 0.7913
Epoch 21/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4424 - accuracy: 0.7927
Epoch 22/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4420 - accuracy: 0.7897
Epoch 23/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4411 - accuracy: 0.7934
Epoch 24/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4407 - accuracy: 0.7903
Epoch 25/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4402 - accuracy: 0.7928
Epoch 26/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4391 - accuracy: 0.7929
Epoch 27/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4388 - accuracy: 0.7936
Epoch 28/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4388 - accuracy: 0.7936
Epoch 29/3500
```

```
# Visualisation Training and Testing Loss
plt.plot(trained_model.history['loss'],label='Training Loss')
plt.plot(trained_model.history['val_loss'],label='Testing Loss')
plt.legend()
plt.show()
```




```
# Visualise
plt.plot(trained_model.history['accuracy'],label='Training Accuracy')
plt.plot(trained_model.history['val_accuracy'],label='Testing Accuracy')
plt.legend()
plt.show()
```




```
#Here we can see gap between Accuracy and loss, Which shows the model is Overfit
```

Conclusion

To reduce overfitting we will use Regularization: Regularization is a technique to prevent the model from overfitting. There are two type of technique of Regularization 1.Ridge L2 2.Lasso L1 here we use Ridge L2 Regularization .

```
# To create overfitting, use regularation : Ridge means L2
# create a neural network
# Create object of Sequential Class
# To reduce overfitting , use Regularation : Ridge means L2
# Create a Neural Network

from keras import regularizers
model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(units=10,activation='relu',input_shape=(X.shape[1],),ke
        tf.keras.layers.Dense(units=10,activation='relu'),
        tf.keras.layers.Dense(units=1,activation='sigmoid',kernel_regularizer=regular
])
```

```
# Compile the model
model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])


#Train the Model
trained_model = model.fit(X_train_ros,Y_train_ros,batch_size=20,epochs=3500,validatic
```

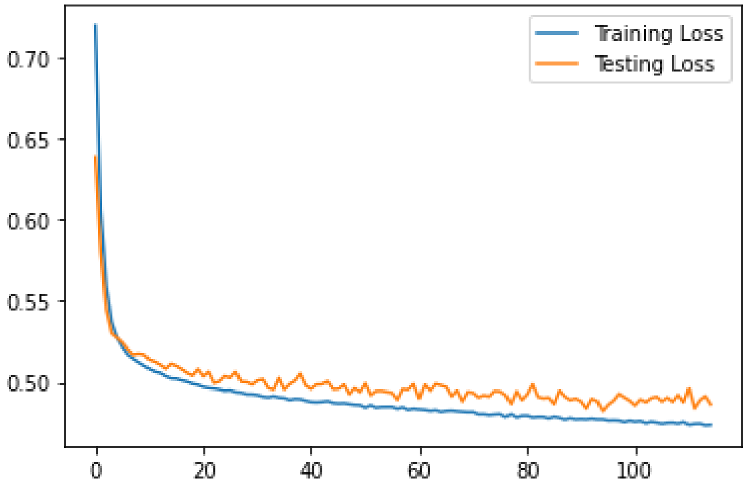
```
Epoch 1/3500
559/559 [==============================] - 2s 2ms/step - loss: 0.7193 - accuracy: 0.6258
Epoch 2/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.6057 - accuracy: 0.7202
Epoch 3/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5590 - accuracy: 0.7436
Epoch 4/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5370 - accuracy: 0.7514
Epoch 5/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5275 - accuracy: 0.7541
Epoch 6/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5218 - accuracy: 0.7577
Epoch 7/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5171 - accuracy: 0.7588
Epoch 8/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5146 - accuracy: 0.7618
Epoch 9/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5123 - accuracy: 0.7595
Epoch 10/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5103 - accuracy: 0.7601
Epoch 11/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5082 - accuracy: 0.7625
Epoch 12/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5066 - accuracy: 0.7645
Epoch 13/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5057 - accuracy: 0.7630
Epoch 14/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5038 - accuracy: 0.7623
Epoch 15/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5025 - accuracy: 0.7628
Epoch 16/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5022 - accuracy: 0.7656
Epoch 17/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5012 - accuracy: 0.7653
Epoch 18/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.5004 - accuracy: 0.7659
Epoch 19/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4992 - accuracy: 0.7643
Epoch 20/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4984 - accuracy: 0.7648
Epoch 21/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4972 - accuracy: 0.7676
Epoch 22/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4966 - accuracy: 0.7686
Epoch 23/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4962 - accuracy: 0.7655
Epoch 24/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4956 - accuracy: 0.7682
Epoch 25/3500
559/559 [==============================] - 1s 3ms/step - loss: 0.4948 - accuracy: 0.7652
Epoch 26/3500
559/559 [==============================] - 2s 4ms/step - loss: 0.4951 - accuracy: 0.7668
Epoch 27/3500
559/559 [==============================] - 2s 3ms/step - loss: 0.4939 - accuracy: 0.7676
```

```
Epoch 28/3500
559/559 [==============================] - 1s 2ms/step - loss: 0.4935 - accuracy: 0.7684
Epoch 29/3500
```

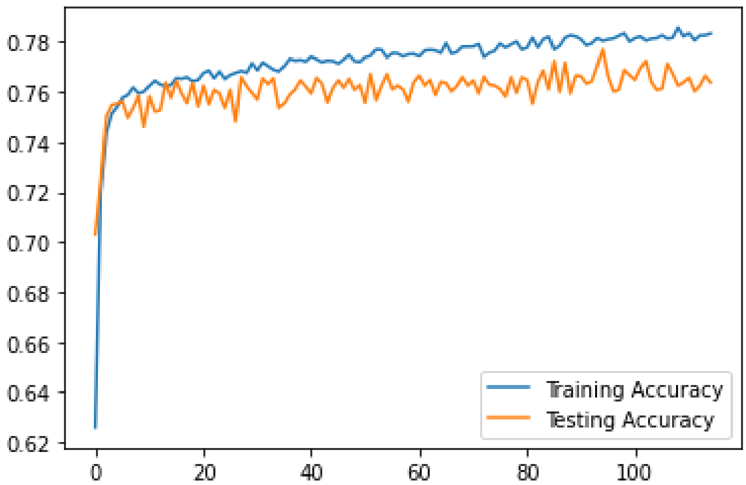
```
print("Training Loss and Training Accuracy:",model.evaluate(X_train_ros,Y_train_ros))
print("Testing Loss and Testing Accuracy:",model.evaluate(X_train_ros,Y_train_ros))
```

```
350/350 [==============================] - 0s 1ms/step - loss: 0.4691 - accuracy: 0.7841
Training Loss and Training Accuracy: [0.4690917432308197, 0.7840787172317505]
350/350 [==============================] - 0s 1ms/step - loss: 0.4691 - accuracy: 0.7841
Testing Loss and Testing Accuracy: [0.4690917432308197, 0.7840787172317505]
```

```
# Visualisation Training and Testing Loss
plt.plot(trained_model.history['loss'],label='Training Loss')
plt.plot(trained_model.history['val_loss'],label='Testing Loss')
plt.legend()
plt.show()
```




```
# Visualise
plt.plot(trained_model.history['accuracy'],label='Training Accuracy')
plt.plot(trained_model.history['val_accuracy'],label='Testing Accuracy')
plt.legend()
plt.show()
```



Conclusion

Now After using Regularization we got good accuracy, but we will still try to get more accuracy.

Then now we will increase the numbers of neurons and if we increase the number of neurons we will use dropout . Dropout is a technique that drops the number of neurons from the neural network or 'ignores' them during training.

Its means dropout handle the huge amount of number of neurons.

Dropout always 20,30,and 50% not more than 50%

```
# Create a Neural Network

from keras import regularizers
from keras.layers import Dropout
model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(units=1000,activation='relu',input_shape=(X.shape[1],),
        tf.keras.layers.Dense(units=1000,activation='relu',kernel_regularizer=regular
        tf.keras.layers.Dense(units=1,activation='sigmoid',kernel_regularizer=regular
])
```

```
# Compile the model

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

```
# Train the Model

trained_model= model.fit(X_train_ros,Y_train_ros,batch_size=32,epochs=3500,
                          validation_data=(X_test_ros,Y_test_ros),callbacks=callback)
```

```
Epoch 1/3500
350/350 [==============================] - 7s 18ms/step - loss: 1.2342 - accuracy: 0.721
Epoch 2/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.6222 - accuracy: 0.742
Epoch 3/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.6106 - accuracy: 0.745
Epoch 4/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.6103 - accuracy: 0.738
Epoch 5/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.6069 - accuracy: 0.742
Epoch 6/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.6033 - accuracy: 0.742
Epoch 7/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.6037 - accuracy: 0.743
Epoch 8/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.6043 - accuracy: 0.742
Epoch 9/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.6014 - accuracy: 0.743
Epoch 10/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.6003 - accuracy: 0.742
Epoch 11/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.5999 - accuracy: 0.743
Epoch 12/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.6018 - accuracy: 0.744
Epoch 13/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5993 - accuracy: 0.746
Epoch 14/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5993 - accuracy: 0.743
```

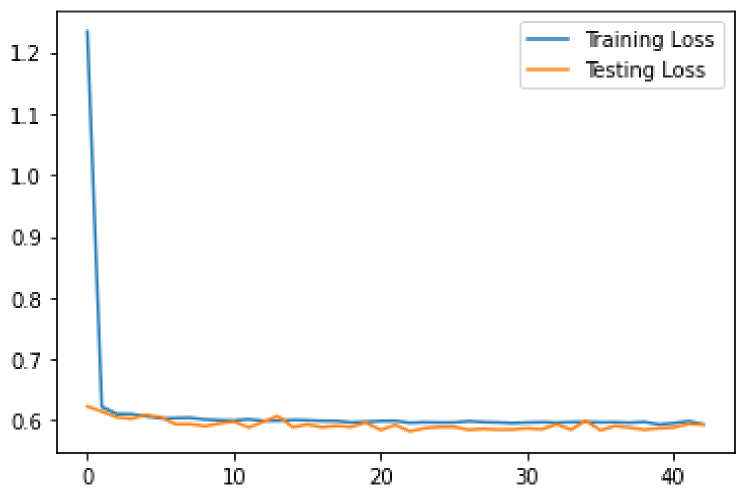
```
Epoch 15/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.6006 - accuracy: 0.742
Epoch 16/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.6002 - accuracy: 0.743
Epoch 17/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5991 - accuracy: 0.748
Epoch 18/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5990 - accuracy: 0.747
Epoch 19/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5965 - accuracy: 0.744
Epoch 20/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.5975 - accuracy: 0.745
Epoch 21/3500
350/350 [==============================] - 7s 19ms/step - loss: 0.5989 - accuracy: 0.743
Epoch 22/3500
350/350 [==============================] - 6s 18ms/step - loss: 0.5992 - accuracy: 0.742
Epoch 23/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5959 - accuracy: 0.746
Epoch 24/3500
350/350 [==============================] - 6s 16ms/step - loss: 0.5968 - accuracy: 0.745
Epoch 25/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.5964 - accuracy: 0.746
Epoch 26/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.5964 - accuracy: 0.747
Epoch 27/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.5985 - accuracy: 0.744
Epoch 28/3500
350/350 [==============================] - 6s 17ms/step - loss: 0.5972 - accuracy: 0.744
Epoch 29/3500
```

```
print("Training Loss and Training Accuracy:",model.evaluate(X_train_ros,Y_train_ros))
print("Testing Loss and Testing Accuracy:",model.evaluate(X_train_ros,Y_train_ros))
```

```
350/350 [==============================] - 2s 7ms/step - loss: 0.5902 - accuracy: 0.7485
Training Loss and Training Accuracy: [0.5901646018028259, 0.748479425907135]
350/350 [==============================] - 1s 4ms/step - loss: 0.5902 - accuracy: 0.7485
Testing Loss and Testing Accuracy: [0.5901646018028259, 0.748479425907135]
```

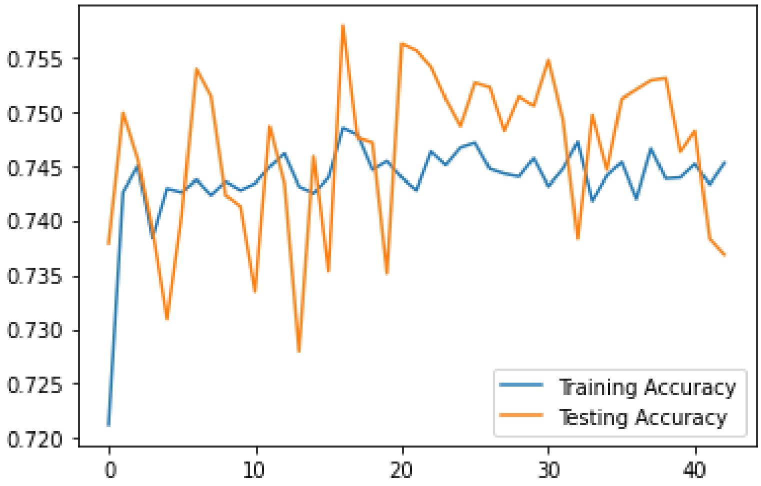
```
# Visualisation Training and Testing Loss

plt.plot(trained_model.history['loss'],label='Training Loss')
plt.plot(trained_model.history['val_loss'],label='Testing Loss')
plt.legend()
plt.show()
```



```
# Visualisation Training and Testing Accuracy

plt.plot(trained_model.history['accuracy'],label='Training Accuracy')
plt.plot(trained_model.history['val_accuracy'],label='Testing Accuracy')
plt.legend()
plt.show()
```



Conclusion

We can see that,After applying Dropout we got better percentage as what we had applied using Regularization.