Bank_Customer_churn_Prediction using Deep Learning

```
#Importing Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error as mse, mean_absolute_error as mae, confusi
from sklearn.preprocessing import LabelEncoder,OneHotEncoder

import keras
import warnings
warnings.filterwarnings('ignore')
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

```
#import data
df=pd.read_csv('/content/drive/MyDrive/DL /Churn_Modelling.csv')
```

EDA - Exploratory Data Analysis

```
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

```
df.tail()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenur
9995	9996	15606229	Obijiaku	771	France	Male	39	
9996	9997	15569892	Johnstone	516	France	Male	35	1
9997	9998	15584532	Liu	709	France	Female	36	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	

```
df.shape
```

```
(10000, 14)
```

```
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.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Ba
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.0
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.8
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.4
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.0
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.0
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.5
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.2
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.0

```
# Check columns list and missing values
df.isnull().sum()
```

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

```
duplicate_data = df[df.duplicated()]
```

```
duplicate_data
```

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bal

```
# Get unique count for each variable
df.nunique()
```

```
RowNumber          10000
CustomerId         10000
Surname             2932
CreditScore          460
Geography              3
Gender                 2
Age                   70
Tenure                11
Balance             6382
NumOfProducts          4
HasCrCard              2
IsActiveMember         2
EstimatedSalary     9999
Exited                 2
dtype: int64
```

```
df.shape
```

```
(10000, 14)
```

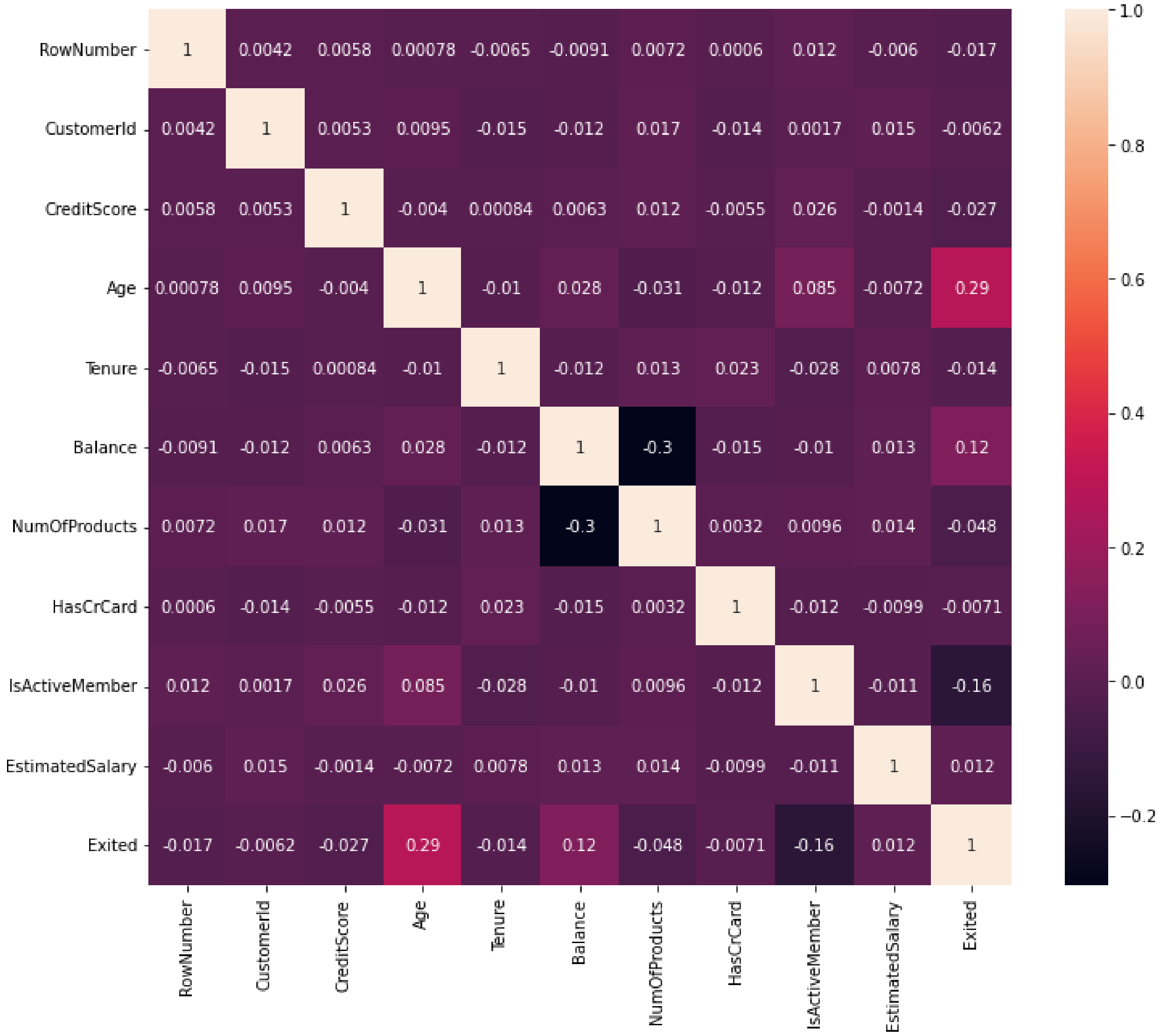
```
# Check variable data types
df.dtypes
```

```
RowNumber            int64
```

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

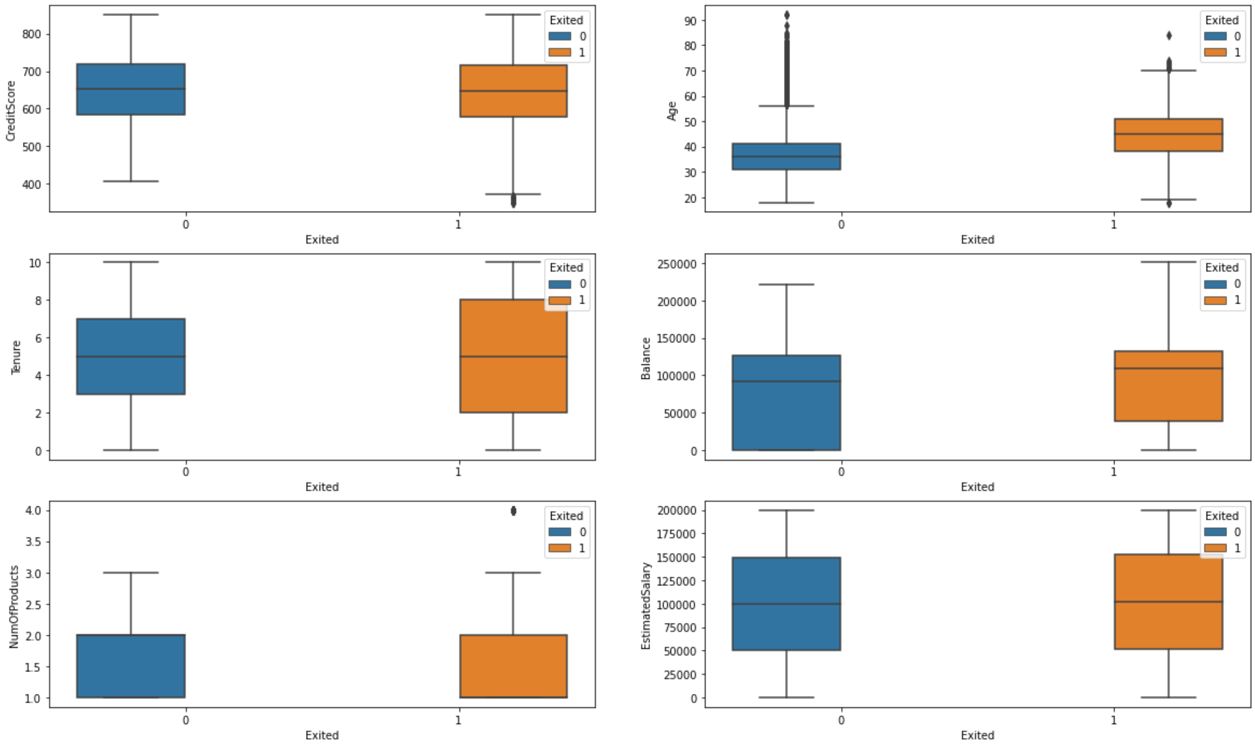
```
plt.figure(figsize=(12,10))
sns.heatmap(df.corr(), annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f535b14a3d0>
```



```
fig, axarr = plt.subplots(3, 2, figsize=(20, 12))
sns.boxplot(y='CreditScore',x = 'Exited', hue = 'Exited',data = df, ax=axarr[0][0])
sns.boxplot(y='Age',x = 'Exited', hue = 'Exited',data = df , ax=axarr[0][1])
sns.boxplot(y='Tenure',x = 'Exited', hue = 'Exited',data = df, ax=axarr[1][0])
sns.boxplot(y='Balance',x = 'Exited', hue = 'Exited',data = df, ax=axarr[1][1])
sns.boxplot(y='NumOfProducts',x = 'Exited', hue = 'Exited',data = df, ax=axarr[2][0])
sns.boxplot(y='EstimatedSalary',x = 'Exited', hue = 'Exited',data = df, ax=axarr[2][1])
```

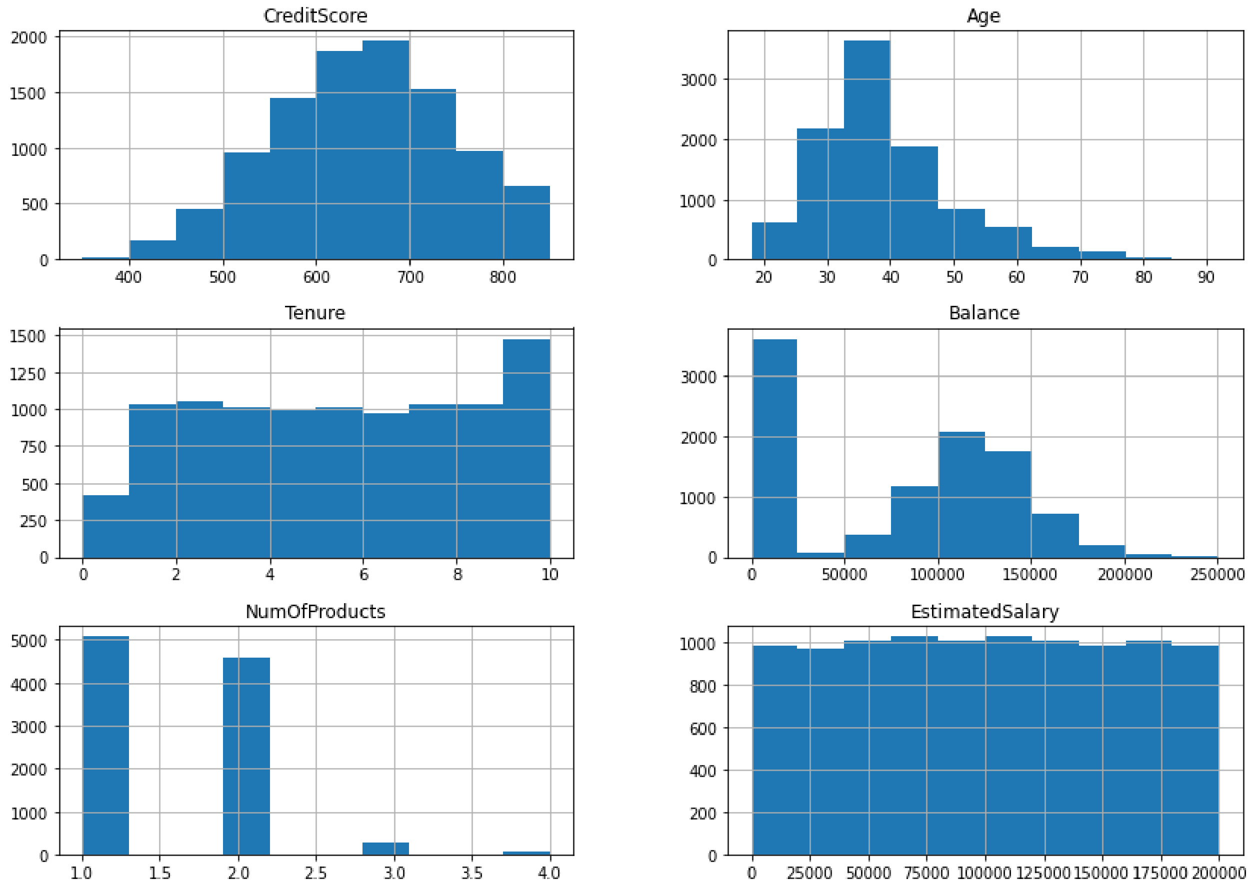
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f535aae1750>
```



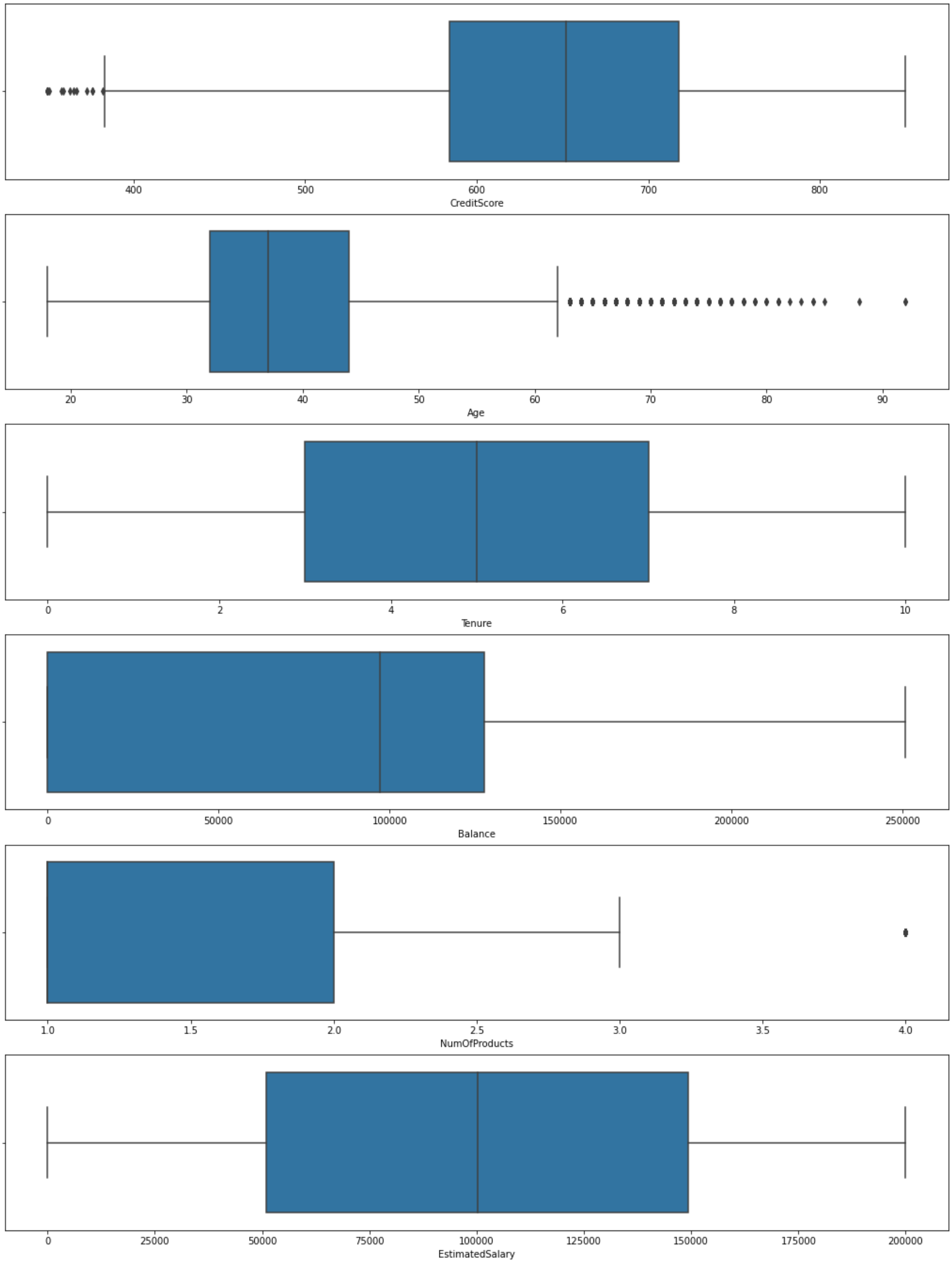
Check numberical data distribution

```
num_cols = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
df.hist(column=num_cols, figsize=(14,10))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5359fb21d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f535b234650>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f535abcd450>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f535aba5950>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f5361860e50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5361824390>]],
      dtype=object)
```




```
# Handling Outliers
fig, ax = plt.subplots(6, 1, figsize=(18,24))
for i in range(6):
    sns.boxplot(x = df[num_cols[i]], ax=ax[i])
```


400
500
600
700
800
CreditScore
20
30
40
50
60
70
80
90
Age
0
2
4
6
8
10
Tenure
0
50000
100000
150000
200000
250000
Balance
1.0
1.5
2.0
2.5
3.0
3.5
4.0
NumOfProducts
0
25000
50000
75000
100000
125000
150000
175000
200000
EstimatedSalary

Feature Engineering

```
#Drop identifier data column
df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis = 1)
```

```
#Move outliers values to the upper and lower bounds
for col in num_cols:
  Q1 = df[col].quantile(0.25)
  Q3 = df[col].quantile(0.75)
  IQR = Q3 - Q1
  S = 1.5*IQR
  LB = Q1 - S
  UB = Q3 + S
  df.loc[df[col] > UB,col] = UB
  df.loc[df[col] < LB,col] = LB
```

```
#Create new Gender column with value is 0 and 1
df['Gender New'] = pd.factorize(df.Gender)[0]
df = df.drop(['Gender'], axis = 1)
```

```
#One hot encode Geography column¶
dvcat_dummies = pd.get_dummies(df.Geography)
df=pd.concat([df, dvcat_dummies], axis=1)
```

```
df = df.drop(['Geography'], axis = 1)
```

```
df.head(10)
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
4	850	43	2	125510.82	1.0	1	1

```
# split feature, targer and train, test
X = df.drop(columns=['Exited'])
y = df['Exited'].values

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

```
#Feature scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_trf = scaler.fit_transform(X_train)
X_test_trf = scaler.transform(X_test)
```

```
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
```

```
y_train
```

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

```
y_test
```

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

Model Building

```
# Neural network

model = Sequential()

model.add(Dense(6, activation = 'relu', input_shape=(12,)))
model.add(keras.layers.Dropout(0.5))
model.add(Dense(6, activation = 'relu'))
model.add(keras.layers.Dropout(0.5))
model.add(Dense(1, activation = 'sigmoid'))
```

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

Model training

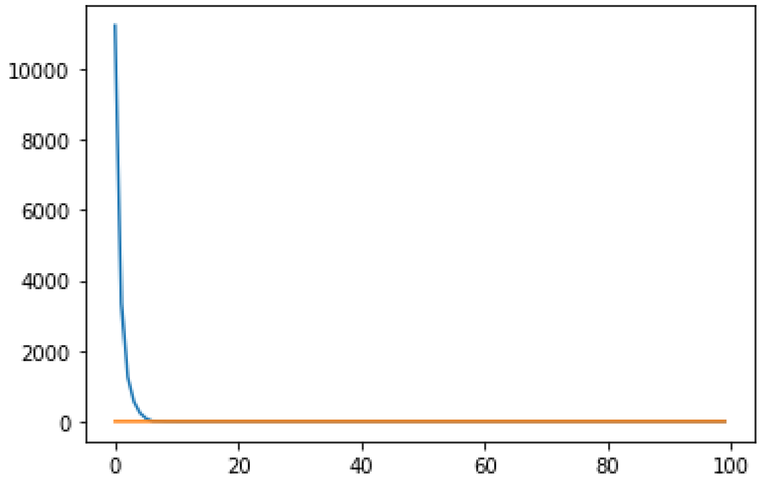
```
history = model.fit(X_train,y_train,batch_size=50,epochs=100,verbose=1,validation_split=0.
```

```
Epoch 1/100
```

```
128/128 [==============================] - 1s 5ms/step - loss: 11241.8672 - accura
Epoch 2/100
128/128 [==============================] - 0s 3ms/step - loss: 3341.2812 - accuracy
Epoch 3/100
128/128 [==============================] - 0s 3ms/step - loss: 1255.6195 - accuracy
Epoch 4/100
128/128 [==============================] - 0s 3ms/step - loss: 561.2086 - accuracy
Epoch 5/100
128/128 [==============================] - 0s 3ms/step - loss: 237.1535 - accuracy
Epoch 6/100
128/128 [==============================] - 0s 3ms/step - loss: 81.8563 - accuracy:
Epoch 7/100
128/128 [==============================] - 1s 5ms/step - loss: 10.8184 - accuracy:
Epoch 8/100
128/128 [==============================] - 1s 6ms/step - loss: 8.1663 - accuracy: (
Epoch 9/100
128/128 [==============================] - 1s 5ms/step - loss: 4.1229 - accuracy: (
Epoch 10/100
128/128 [==============================] - 1s 6ms/step - loss: 1.9319 - accuracy: (
Epoch 11/100
128/128 [==============================] - 1s 5ms/step - loss: 0.9040 - accuracy: (
Epoch 12/100
128/128 [==============================] - 1s 5ms/step - loss: 0.6583 - accuracy: (
Epoch 13/100
128/128 [==============================] - 1s 5ms/step - loss: 0.6699 - accuracy: (
Epoch 14/100
128/128 [==============================] - 1s 7ms/step - loss: 0.5520 - accuracy: (
Epoch 15/100
128/128 [==============================] - 1s 9ms/step - loss: 0.5662 - accuracy: (
Epoch 16/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5433 - accuracy: (
Epoch 17/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5489 - accuracy: (
Epoch 18/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5841 - accuracy: (
Epoch 19/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5232 - accuracy: (
Epoch 20/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5101 - accuracy: (
Epoch 21/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5131 - accuracy: (
Epoch 22/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5149 - accuracy: (
Epoch 23/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5084 - accuracy: (
Epoch 24/100
128/128 [==============================] - 0s 4ms/step - loss: 0.5058 - accuracy: (
Epoch 25/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5123 - accuracy: (
Epoch 26/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5181 - accuracy: (
Epoch 27/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5079 - accuracy: (
Epoch 28/100
128/128 [==============================] - 0s 3ms/step - loss: 0.5054 - accuracy: (
Epoch 29/100
```

```
plt.figure()
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['accuracy'])
plt.show()
```

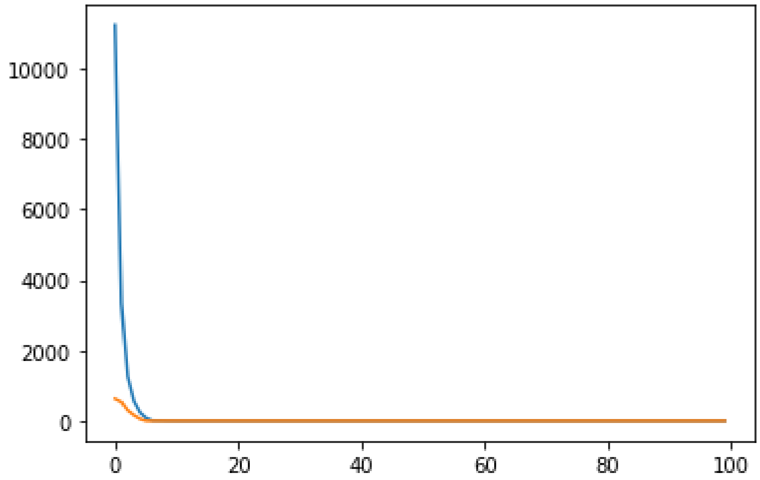



```
#Model evaluation

y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

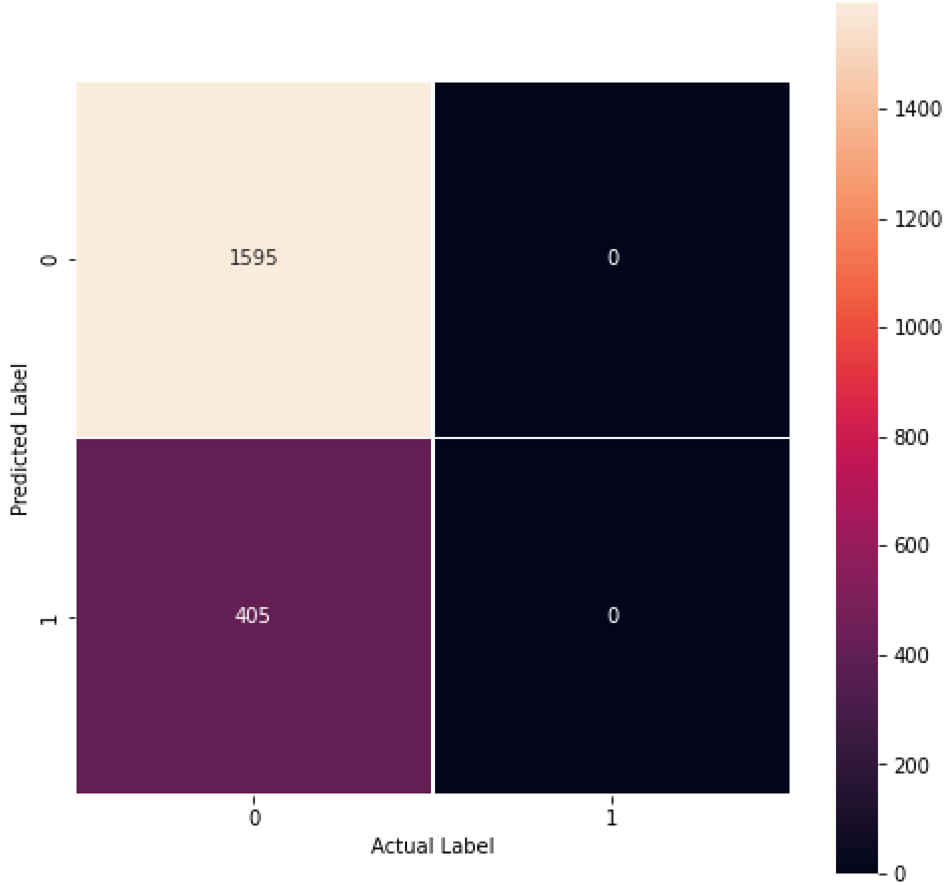
```
[<matplotlib.lines.Line2D at 0x7f5365332190>]
```




```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7f53652e7250>]
```


```
#Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,8))
sns.heatmap(cm, annot=True, fmt=".0f",linewidth=.5, square=True);
plt.xlabel('Actual Label');
plt.ylabel('Predicted Label');
```




```
#Classification report¶
score = metrics.accuracy_score(y_test,y_pred)
print("Accuracy:", score)

print("Report:",metrics.classification_report(y_test,y_pred))
```

```
Accuracy: 0.7975
Report:               precision    recall  f1-score   support

           0       0.80      1.00      0.89      1595
           1       0.00      0.00      0.00       405

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.44      2000
```

```
weighted avg       0.64      0.80      0.71      2000
```

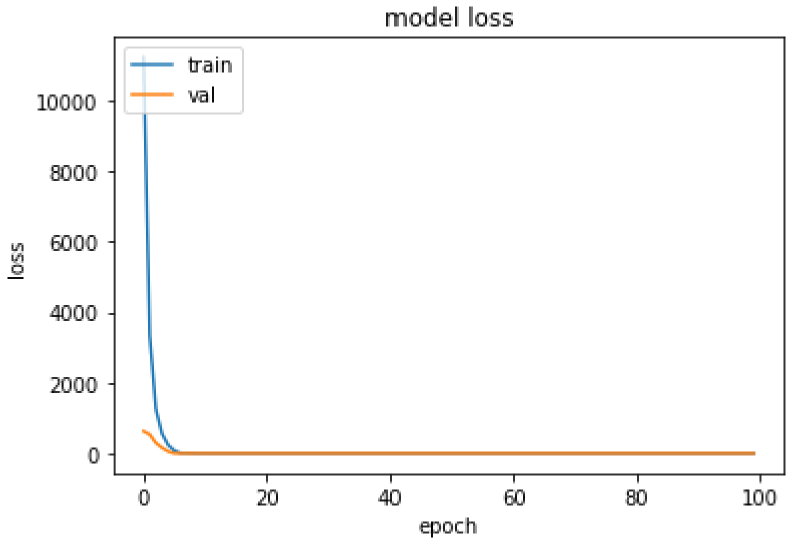
MAE, MSE & RMSE

```
MSE = mse(y_test, y_pred)
MAE = mae(y_test, y_pred)
print("MSE:",MSE)
print("RMSE:",np.sqrt(MSE))
print("MAE:",MAE)
```

```
MSE: 0.2025
RMSE: 0.45
MAE: 0.2025
```

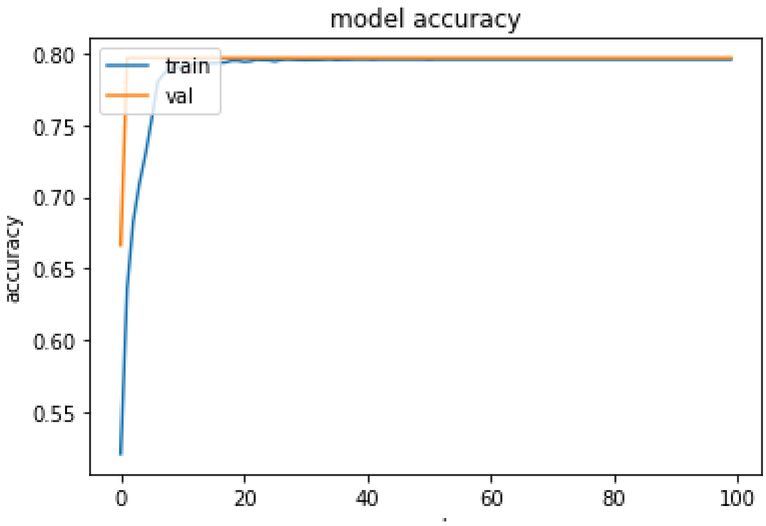
Model loss

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



Model accuracy

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```


model accuracy
train
val
accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0
20
40
60
80
100