```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

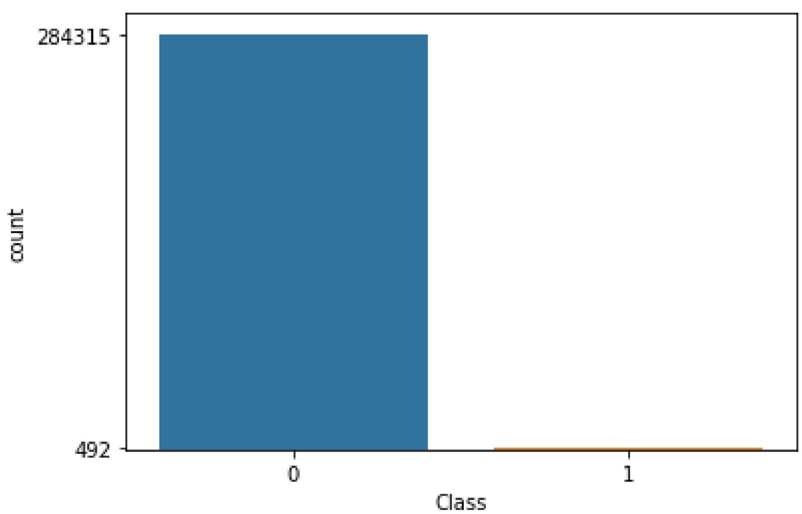
```
df=pd.read_csv("/content/drive/MyDrive/Deep learning/creditcard (1).csv")
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.09869
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.08510
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.24767
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.37743
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

```
print(df["Class"].value_counts()) #we want to predict Class ,this is target variable

#Visualise
import seaborn as sns
sns.countplot(data=df,x="Class")
c=df["Class"].value_counts()
plt.yticks(c)
plt.show()
```

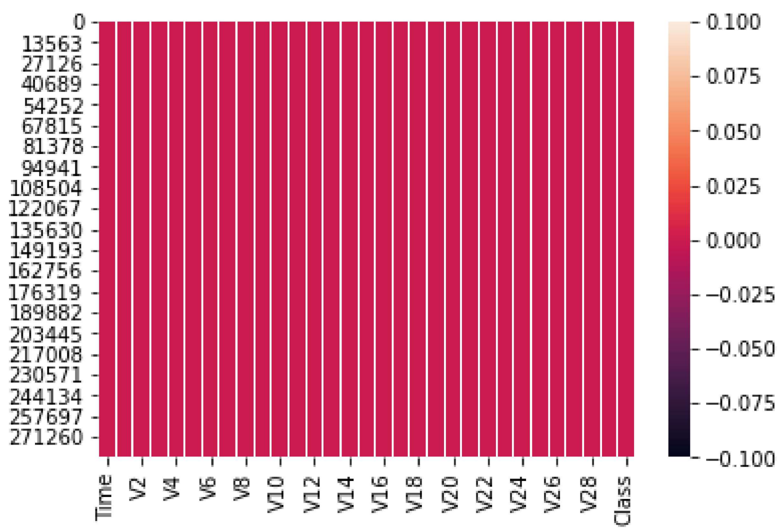
```
0    284315
1       492
Name: Class, dtype: int64
```




```
#Check null value
print(df.isnull().sum())
```

```
#Visualise
sns.heatmap(df.isnull())
plt.show()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```




```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
#apply Label Encoder on species target column : - means to convert object type data into nume
from sklearn.preprocessing import LabelEncoder
#Create object of LabelEncoder class
le=LabelEncoder()
df["Class"]=le.fit_transform(df["Class"])
#check
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
```

```
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
#Separate input and output from dataset
X=df.drop("Class",axis=1)
Y=df["Class"]
```

```
#train test split : 70%-30%
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)
```

```
#here split the data randomly

print(X_train.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(199364, 30)
(199364,)
(85443,)
```

```
print(Y_train.value_counts())
print(Y_test.value_counts())
```

```
0    199007
```

```
1       357
Name: Class, dtype: int64
0    85308
1      135
Name: Class, dtype: int64
```

```
from sklearn.model_selection import train_test_split
#spit the data same when we write stratify=Y
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=1,stratify=Y)
```

```
print(Y_train.value_counts())
print(Y_test.value_counts())
```

```
0    199020
1       344
Name: Class, dtype: int64
0    85295
1      148
Name: Class, dtype: int64
```

```
#apply scaling on X_train and X_test data
from sklearn.preprocessing import StandardScaler
#create the object of StandardScaler class
ss=StandardScaler()
X_train=ss.fit_transform(X_train)
X_test=ss.transform(X_test)
```

```
X_train
```

```
array([[-0.52692701,  0.58190778, -0.40724787, ...,  0.21343269,
         0.01335408, -0.35455213],
       [-0.23805364, -0.92002219,  0.68738658, ..., -0.10567826,
        -0.17931826,  0.21465265],
       [ 0.35862596, -0.35300868,  0.41750434, ..., -2.67637576,
        -1.54343883, -0.2122206 ],
       ...,
       [ 0.69977506,  1.18780475, -0.65454452, ..., -0.19435428,
        -0.26339329, -0.14018524],
       [ 1.03558089,  1.01651558,  0.06450567, ...,  0.18067623,
        -0.01803595, -0.32664399],
       [ 0.87139092,  0.96955575, -0.12216971, ...,  0.12206464,
        -0.10146095, -0.3236105 ]])
```

```
#create a neural network
import tensorflow as tf

model=tf.keras.Sequential([
                tf.keras.layers.Dense(32,activation='relu',input_shape=(X.shape[1],)), #first
                  tf.keras.layers.Dense(32,activation='relu'), #Second hidden layer
                  tf.keras.layers.Dense(1,activation='sigmoid')   #output class
])
```

```
model.summary()
#30 i/p *33(neuron)+33bias=992
#32(neuron(i/p))*32(neuron)+32bias=1056
#32(neuron(i/p))*1(neuron)+1(bias)=33
#Total param= 2,081
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                992
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,081
Trainable params: 2,081
Non-trainable params: 0
_________________________________________________________________
```

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

```
#Train the model and also check model is overfit or not then use validation_data parameter an
#the value of 30% testing data (input and output)
trained_model=model.fit(X_train, Y_train,batch_size=32, epochs=100,validation_data=(X_test,Y_
```

```
Epoch 1/100
6231/6231 [==============================] - 13s 2ms/step - loss: 0.0090 - accuracy:
Epoch 2/100
6231/6231 [==============================] - 12s 2ms/step - loss: 0.0032 - accuracy:
Epoch 3/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0029 - accuracy:
Epoch 4/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0026 - accuracy:
Epoch 5/100
6231/6231 [==============================] - 12s 2ms/step - loss: 0.0025 - accuracy:
Epoch 6/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0023 - accuracy:
Epoch 7/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0021 - accuracy:
Epoch 8/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0021 - accuracy:
Epoch 9/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0020 - accuracy:
Epoch 10/100
6231/6231 [==============================] - 12s 2ms/step - loss: 0.0019 - accuracy:
Epoch 11/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0017 - accuracy:
Epoch 12/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0017 - accuracy:
```

```
Epoch 13/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0017 - accuracy:
Epoch 14/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0016 - accuracy:
Epoch 15/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0015 - accuracy:
Epoch 16/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0015 - accuracy:
Epoch 17/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0014 - accuracy:
Epoch 18/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0014 - accuracy:
Epoch 19/100
6231/6231 [==============================] - 10s 2ms/step - loss: 0.0012 - accuracy:
Epoch 20/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0014 - accuracy:
Epoch 21/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0012 - accuracy:
Epoch 22/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0011 - accuracy:
Epoch 23/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0012 - accuracy:
Epoch 24/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0011 - accuracy:
Epoch 25/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0012 - accuracy:
Epoch 26/100
6231/6231 [==============================] - 11s 2ms/step - loss: 9.3054e-04 - accura
Epoch 27/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0011 - accuracy:
Epoch 28/100
6231/6231 [==============================] - 11s 2ms/step - loss: 0.0011 - accuracy:
Epoch 29/100
6231/6231 [==============================] - 11s 2ms/step - loss: 9.6387e-04 - accura
```

```
#here training_error =0.12 which is less than testing error=0.26 means model is overfit
#means training's error< testing error so model is overfit
#or accuracy of training data >accuracy of testing data means model is overfit
print("Testing Error and Accuracy of Testing Data : ",model.evaluate(X_test, Y_test) )
```

```
2671/2671 [==============================] - 3s 955us/step - loss: 0.0110 - accuracy: 0
Testing Error and Accuracy of Testing Data :  [0.011039292439818382, 0.9994616508483887]
```

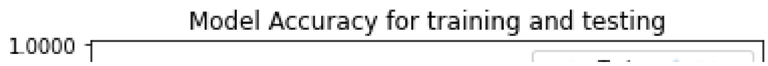
```
#visualise training error(loss) and testing error (loss)
plt.plot(trained_model.history['loss']) #training's loss means error
plt.plot(trained_model.history['val_loss']) #testing's loss means error
plt.title('Model loss for training and testing')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val means test'], loc='upper right')#loc means location
plt.show()
```


```
#here training_error =0.12 which is less than testing error=0.26 means model is overfit
#means training's error< testing error so model is overfit
#or accuracy of training data >accuracy of testing data means model is overfit
print("Testing Error and Accuracy of Testing Data : ",model.evaluate(X_test, Y_test) )

print("Training Error and Accuracy of Testing Data : ",model.evaluate(X_train, Y_train) )
```

```
2671/2671 [==============================] - 3s 962us/step - loss: 0.0110 - accuracy: 0
Testing Error and Accuracy of Testing Data :  [0.011039292439818382, 0.9994616508483887]
6231/6231 [==============================] - 6s 951us/step - loss: 2.4026e-04 - accuracy
Training Error and Accuracy of Testing Data :  [0.0002402560057817027, 0.99993479251861
```

```
#visualise training Accuracy and testing accuracy
plt.plot(trained_model.history['accuracy']) #training's loss means error
plt.plot(trained_model.history['val_accuracy']) #testing's loss means error
plt.title('Model Accuracy for training and testing')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val means test'], loc='upper right')#loc means location
plt.show()
```




```
#we can see , our model is overfit
#find prediction
Y_pred=model.predict(X_test)  #give probability value Y_pred=1/(1+exp(-X))
print(Y_pred)
```

```
[[4.0228051e-22]
 [5.4024568e-18]
 [1.4521355e-23]
 ...
 [3.4406151e-09]
 [2.4998188e-04]
 [1.1828717e-09]]
```

```
Y_pred=np.where(Y_pred>=0.5,1,0)
print(Y_pred)
```

```
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
```

```
#generate report
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.83      0.86      0.85       148

    accuracy                           1.00     85443
   macro avg       0.92      0.93      0.92     85443
weighted avg       1.00      1.00      1.00     85443

[[85269    26]
 [   20   128]]
```

```
#score is good but not better .will do much better
#reason : model is overfit
#apply regularisation means to reduce overfit
#1. L1 means Lasso and L2 means Ridge and Dropout
```

```
from keras.layers import Dropout
```

```
from keras import regularizers
#apply regularisation and model2 user defined object of Sequential class
model2 = tf.keras.Sequential([
tf.keras.layers.Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01), inpu
    Dropout(0.5),  #50% neuron deactivate
  tf.keras.layers.Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),#h
    Dropout(0.5),
    tf.keras.layers.Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.5),
    tf.keras.layers.Dense(1000, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    tf.keras.layers.Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01))
])


model2.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])


#Train the model

trained_model1 = model2.fit(X_train, Y_train,batch_size=32, epochs=50,validation_data=(X_test
```

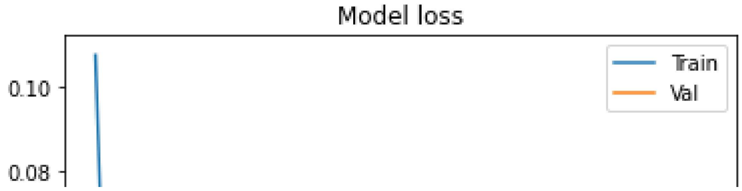
```
Epoch 1/50
6231/6231 [==============================] - 267s 43ms/step - loss: 0.1077 - accuracy
Epoch 2/50
6231/6231 [==============================] - 268s 43ms/step - loss: 0.0165 - accuracy
Epoch 3/50
6231/6231 [==============================] - 269s 43ms/step - loss: 0.0158 - accuracy
Epoch 4/50
6231/6231 [==============================] - 269s 43ms/step - loss: 0.0156 - accuracy
Epoch 5/50
6231/6231 [==============================] - 270s 43ms/step - loss: 0.0154 - accuracy
Epoch 6/50
6231/6231 [==============================] - 267s 43ms/step - loss: 0.0153 - accuracy
Epoch 7/50
6231/6231 [==============================] - 269s 43ms/step - loss: 0.0152 - accuracy
Epoch 8/50
6231/6231 [==============================] - 270s 43ms/step - loss: 0.0154 - accuracy
Epoch 9/50
6231/6231 [==============================] - 279s 45ms/step - loss: 0.0154 - accuracy
Epoch 10/50
6231/6231 [==============================] - 273s 44ms/step - loss: 0.0153 - accuracy
Epoch 11/50
6231/6231 [==============================] - 264s 42ms/step - loss: 0.0152 - accuracy
Epoch 12/50
6231/6231 [==============================] - 262s 42ms/step - loss: 0.0153 - accuracy
Epoch 13/50
6231/6231 [==============================] - 265s 42ms/step - loss: 0.0150 - accuracy
Epoch 14/50
6231/6231 [==============================] - 266s 43ms/step - loss: 0.0150 - accuracy
Epoch 15/50
6231/6231 [==============================] - 287s 46ms/step - loss: 0.0148 - accuracy
Epoch 16/50
6231/6231 [==============================] - 315s 51ms/step - loss: 0.0146 - accuracy
Epoch 17/50
```

```
6231/6231 [==============================] - 323s 52ms/step - loss: 0.0144 - accuracy
Epoch 18/50
6231/6231 [==============================] - 308s 49ms/step - loss: 0.0143 - accuracy
Epoch 19/50
6231/6231 [==============================] - 302s 49ms/step - loss: 0.0143 - accuracy
Epoch 20/50
6231/6231 [==============================] - 326s 52ms/step - loss: 0.0139 - accuracy
Epoch 21/50
6231/6231 [==============================] - 310s 50ms/step - loss: 0.0138 - accuracy
Epoch 22/50
6231/6231 [==============================] - 341s 55ms/step - loss: 0.0137 - accuracy
Epoch 23/50
6231/6231 [==============================] - 337s 54ms/step - loss: 0.0135 - accuracy
Epoch 24/50
6231/6231 [==============================] - 279s 45ms/step - loss: 0.0134 - accuracy
Epoch 25/50
6231/6231 [==============================] - 277s 44ms/step - loss: 0.0133 - accuracy
Epoch 26/50
6231/6231 [==============================] - 273s 44ms/step - loss: 0.0132 - accuracy
Epoch 27/50
6231/6231 [==============================] - 274s 44ms/step - loss: 0.0131 - accuracy
Epoch 28/50
6231/6231 [==============================] - 274s 44ms/step - loss: 0.0130 - accuracy
Epoch 29/50
```

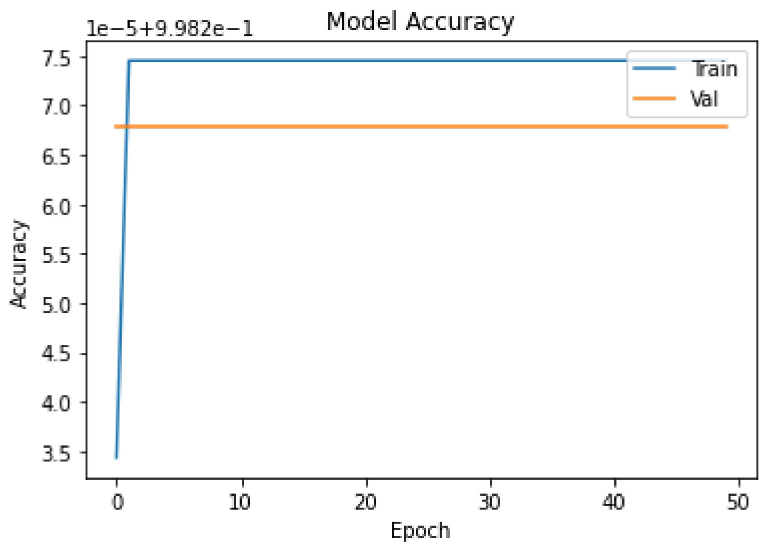
```
print("Testing Error and Accuracy of Testing Data : ",model2.evaluate(X_test, Y_test) )
print("Training Error and Accuracy of Training Data : ",model2.evaluate(X_train, Y_train) )
```

```
2671/2671 [==============================] - 30s 11ms/step - loss: 0.0128 - accuracy: 0
Testing Error and Accuracy of Testing Data :  [0.012750618159770966, 0.9982678294181824
6231/6231 [==============================] - 67s 11ms/step - loss: 0.0127 - accuracy: 0
Training Error and Accuracy of Training Data :  [0.012708255089819431, 0.99827450513839
```

```
plt.plot(trained_model1.history['loss']) #training
plt.plot(trained_model1.history['val_loss'])#testing
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
plt.plot(trained_model1.history['accuracy']) #training score
plt.plot(trained_model1.history['val_accuracy'])#testing score
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```




```
Y_pred=model2.predict(X_test)
Y_pred=np.where(Y_pred>=0.5,1,0)

#Generate Classification report and confusion matrix
print(classification_report(Y_test,Y_pred))
print(confusion_matrix(Y_test,Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     85295
           1       0.00      0.00      0.00       148

    accuracy                           1.00     85443
   macro avg       0.50      0.50      0.50     85443
weighted avg       1.00      1.00      1.00     85443

[[85295     0]
 [  148     0]]
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
```