```
# https://towardsdatascience.com/anomaly-detection-using-autoencoders-5b032178a1ea
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_scor
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
```

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

```
Mounted at /content/drive
```

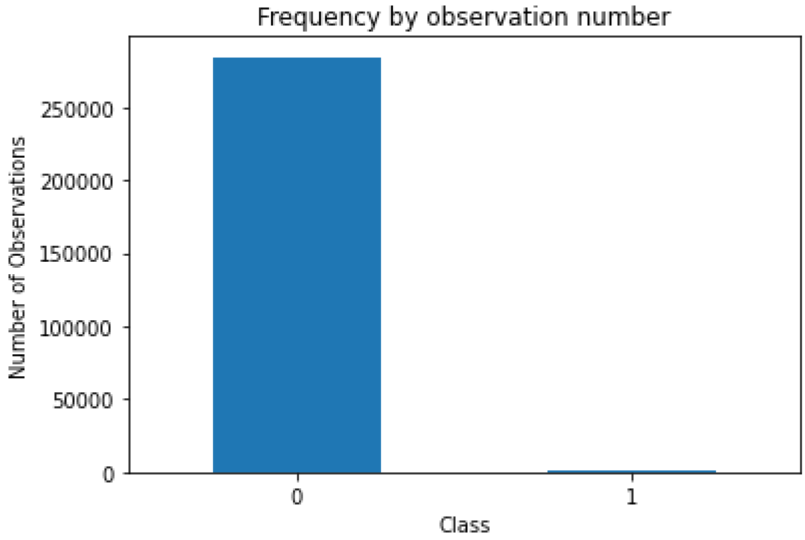
```
dataset = pd.read_csv("/content/drive/MyDrive/DL  assignment/DL csv folder/creditcard.csv"
```

New Section

```
#check for any  nullvalues
print("Any nulls in the dataset ",dataset.isnull().values.any() )
print('-------')
print("No. of unique labels ", len(dataset['Class'].unique()))
print("Label values ",dataset.Class.unique())
#0 is for normal credit card transaction
#1 is for fraudulent credit card transaction
print('-------')
print("Break down of the Normal and Fraud Transactions")
print(pd.value_counts(dataset['Class'], sort = True) )
```

```
Any nulls in the dataset  False
-------
No. of unique labels  2
Label values  [0 1]
-------
Break down of the Normal and Fraud Transactions
0    284315
1       492
Name: Class, dtype: int64
```

```
#Visualizing the imbalanced dataset
#plotting the number of normal and fraud transactions in the dataset.
count_classes = pd.value_counts(dataset['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(dataset['Class'].unique())), dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```




```
#The last column in the dataset is our target variable.
raw_data = dataset.values
# The last element contains if the transaction is normal which is represented by a 0 and i
labels = raw_data[:, -1]
# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=2021
)
```

```
#Normalize the data to have a value between 0 and 1
min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)
train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)
train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)
```

```
# Use only normal transactions to train the Autoencoder.
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
#creating normal and fraud datasets
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print(" No. of records in Fraud Train Data=",len(fraud_train_data))
print(" No. of records in Normal Train data=",len(normal_train_data))
```

```
print(" No. of records in Fraud Test Data=",len(fraud_test_data))
print(" No. of records in Normal Test data=",len(normal_test_data))
```

```
 No. of records in Fraud Train Data= 389
 No. of records in Normal Train data= 227456
 No. of records in Fraud Test Data= 103
 No. of records in Normal Test data= 56859
```

```
#Set the training parameter values
nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1] #num of columns, 30
encoding_dim = 14
hidden_dim_1 = int(encoding_dim / 2) #
hidden_dim_2=4
learning_rate = 1e-7
```

```
#Create the Autoencoder

#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))
#Encoder
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh",activity_regularizer=tf.ke
encoder=tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_2, activation=tf.nn.leaky_relu)(encoder)
# Decoder
decoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(encoder)
decoder=tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim, activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim, activation='tanh')(decoder)
#Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 30)]              0

 dense (Dense)               (None, 14)                434

 dropout (Dropout)           (None, 14)                0

 dense_1 (Dense)             (None, 7)                 105

 dense_2 (Dense)             (None, 4)                 32

 dense_3 (Dense)             (None, 7)                 35

 dropout_1 (Dropout)         (None, 7)                 0

 dense_4 (Dense)             (None, 14)                112
```

```
 dense_5 (Dense)             (None, 30)                450

=================================================================
Total params: 1,168
Trainable params: 1,168
Non-trainable params: 0
_________________________________________________________________
```

```
# Define the callbacks for checkpoints and early stopping

cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",
                               mode='min', monitor='val_loss', verbose=2, save_best_only=T

# define our early stopping
early_stop = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=0.0001,
    patience=10,
    verbose=1,
    mode='min',
    restore_best_weights=True)
```

```
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')
```

```
history = autoencoder.fit(normal_train_data, normal_train_data,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(test_data, test_data),
                    verbose=1,
                    callbacks=[cp, early_stop]
                    ).history
```

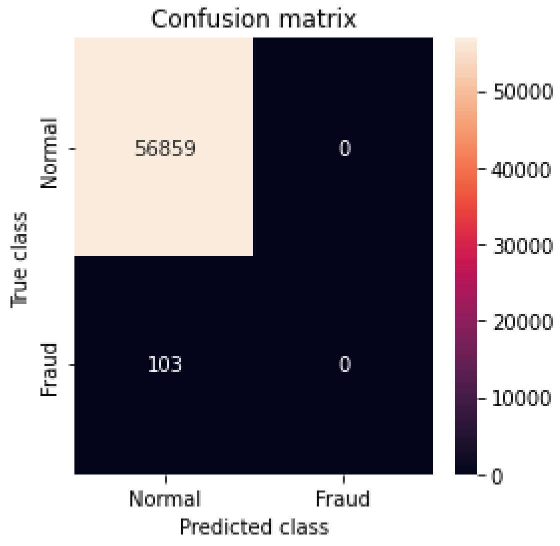
```
Epoch 1/50
3551/3554 [============================>.] - ETA: 0s - loss: 3.6159e-04 - accuracy: 0
Epoch 1: val_loss improved from inf to 0.00005, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 10s 3ms/step - loss: 3.6132e-04 - accura
Epoch 2/50
3549/3554 [============================>.] - ETA: 0s - loss: 3.8823e-05 - accuracy: 0
Epoch 2: val_loss did not improve from 0.00005
3554/3554 [==============================] - 9s 3ms/step - loss: 3.8809e-05 - accurac
Epoch 3/50
3552/3554 [============================>.] - ETA: 0s - loss: 3.1846e-05 - accuracy: 0
Epoch 3: val_loss did not improve from 0.00005
3554/3554 [==============================] - 9s 3ms/step - loss: 3.1843e-05 - accurac
Epoch 4/50
3535/3554 [============================>.] - ETA: 0s - loss: 2.9633e-05 - accuracy: 0
Epoch 4: val_loss did not improve from 0.00005
3554/3554 [==============================] - 9s 3ms/step - loss: 2.9654e-05 - accurac
Epoch 5/50
3543/3554 [============================>.] - ETA: 0s - loss: 2.5502e-05 - accuracy: 0
```

```
Epoch 5: val_loss did not improve from 0.00005
3554/3554 [==============================] - 9s 3ms/step - loss: 2.5484e-05 - accurac
Epoch 6/50
3547/3554 [============================>.] - ETA: 0s - loss: 2.5053e-05 - accuracy: 0
Epoch 6: val_loss did not improve from 0.00005
3554/3554 [==============================] - 9s 3ms/step - loss: 2.5034e-05 - accurac
Epoch 7/50
3532/3554 [============================>.] - ETA: 0s - loss: 2.3355e-05 - accuracy: 0
Epoch 7: val_loss did not improve from 0.00005
3554/3554 [==============================] - 12s 3ms/step - loss: 2.3293e-05 - accura
Epoch 8/50
3541/3554 [============================>.] - ETA: 0s - loss: 2.2482e-05 - accuracy: 0
Epoch 8: val_loss did not improve from 0.00005
3554/3554 [==============================] - 9s 3ms/step - loss: 2.2503e-05 - accurac
Epoch 9/50
3551/3554 [============================>.] - ETA: 0s - loss: 2.2096e-05 - accuracy: 0
Epoch 9: val_loss did not improve from 0.00005
3554/3554 [==============================] - 12s 3ms/step - loss: 2.2102e-05 - accura
Epoch 10/50
3540/3554 [============================>.] - ETA: 0s - loss: 2.1491e-05 - accuracy: 0
Epoch 10: val_loss did not improve from 0.00005
3554/3554 [==============================] - 13s 4ms/step - loss: 2.1454e-05 - accura
Epoch 11/50
3551/3554 [============================>.] - ETA: 0s - loss: 2.1228e-05 - accuracy: 0
Epoch 11: val_loss did not improve from 0.00005
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 10s 3ms/step - loss: 2.1229e-05 - accura
Epoch 11: early stopping
```

```
# Detect Anomalies on test data
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2), axis=1)
error_df = pd.DataFrame({'Reconstruction_error': mse,
                        'True_class': test_labels})
```

```
1781/1781 [==============================] - 2s 1ms/step
```

```
threshold_fixed =52
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_error.values]
error_df['pred'] =pred_y
conf_matrix = confusion_matrix(error_df.True_class, pred_y)
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```


Confusion matrix
True class
Normal
Fraud
56859
0
103
0
Normal
Fraud
Predicted class
50000
40000
30000
20000
10000
0