```
#import required libraries
import pandas as pd
import numpy as np
from scipy import stats
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_split
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, TensorBoard
from keras import regularizers
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score
```

```
sns.set(style='whitegrid', palette='muted', font_scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

```
#upload the dataset

df = pd.read_csv("/content/creditcard.csv")
```

```
df.shape
```

```
(284807, 31)
```

```
df.isnull().values.any()
```

```
False
```

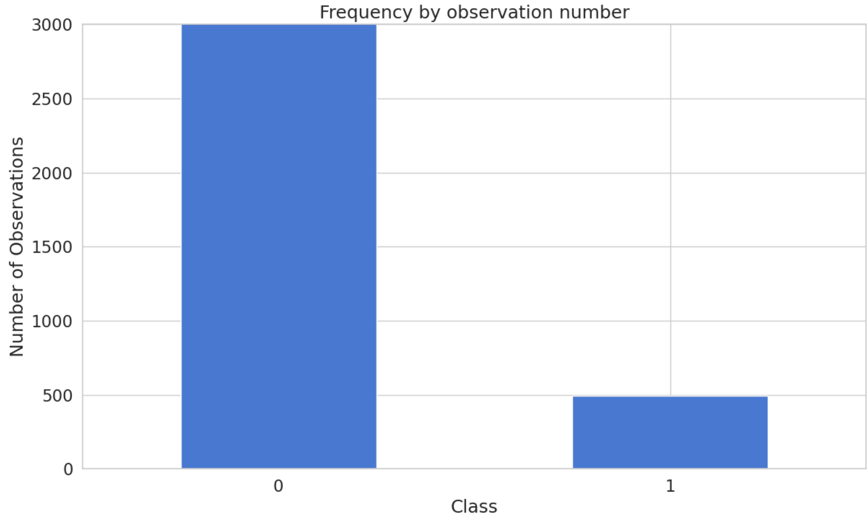
```
print(list(df.columns))
df.describe()
```

```
['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', '
```

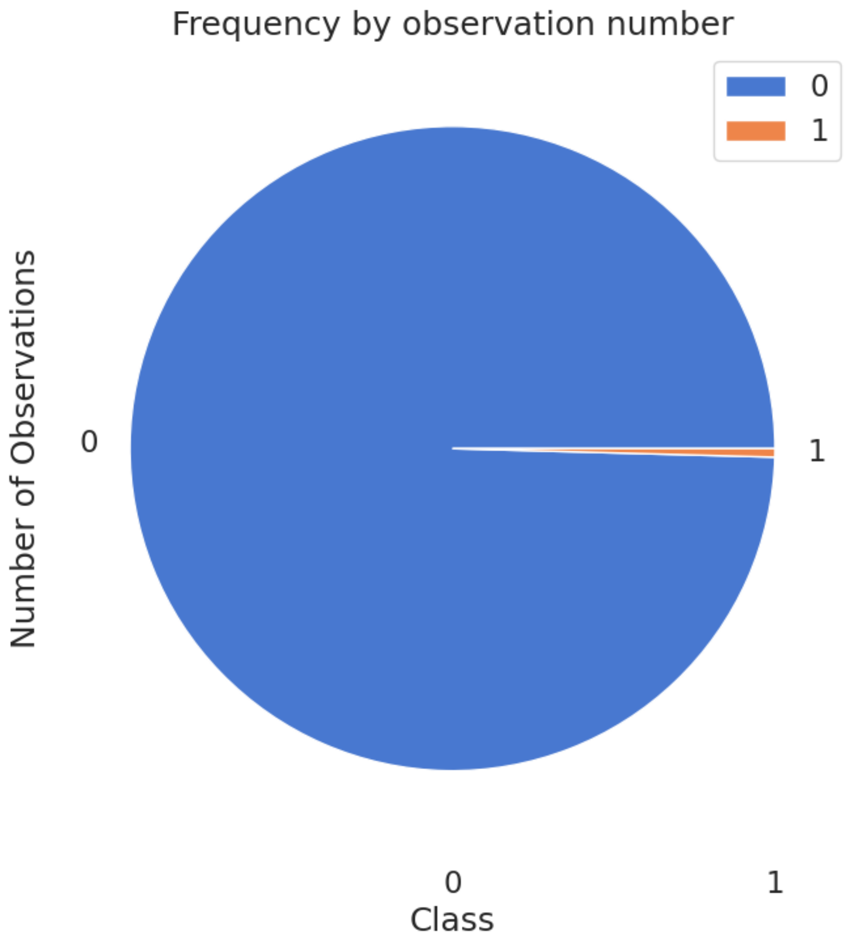
	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.8480
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604(
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.3802
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.1374
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915(
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433(
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119(
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.4801

8 rows × 31 columns

```
#Visualizing the imbalanced dataset
count_classes = pd.value_counts(df['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.xticks(range(len(df['Class'].unique())), df.Class.unique())
plt.ylim(0,3000)
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```




```
#Visualizing the imbalanced dataset
sub=df.iloc[:20000]
count_classes = pd.value_counts(sub['Class'], sort = True)
count_classes.plot(kind = 'pie', rot=0)
plt.xticks(range(len(sub['Class'].unique())), sub.Class.unique())
plt.legend()
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations");
```




```
sc=StandardScaler()
df['Time'] = sc.fit_transform(df['Time'].values.reshape(-1, 1))
df['Amount'] = sc.fit_transform(df['Amount'].values.reshape(-1, 1))
```

```
raw_data = df.values
# The last element contains if the transaction is normal which is represented by a 0 and if fraud then 1
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)


train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
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
```

```
raw_data = df.values
# The last element contains if the transaction is normal which is represented by a 0 and if fraud then 1
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)


train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
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
```

```
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


#Encoder converts it into latent representation
#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))
#Encoder
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh",
activity_regularizer=tf.keras.regularizers.l2(learning_rate))(input_layer)
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 networks convert it back to the original
# 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: 1168 (4.56 KB)
Trainable params: 1168 (4.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min', monitor='val_loss', verbose=2, save_best_only=True)


# 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)


#Compile the Autoencoder

autoencoder.compile(metrics=['accuracy'],
loss='mean_squared_error',

optimizer='adam')

#Train the Autoencoder
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
3535/3554 [============================>.] - ETA: 0s - loss: 0.0048 - accuracy: 0.0450
Epoch 1: val_loss improved from inf to 0.00002, saving model to autoencoder_fraud.h5
3554/3554 [==============================] - 10s 2ms/step - loss: 0.0048 - accuracy: 0.0451 - val_loss: 1.9893e-05 - val_accuracy: (
Epoch 2/50
  89/3554 [..............................] - ETA: 6s - loss: 1.9877e-05 - accuracy: 0.0571/usr/local/lib/python3.10/dist-packages/k
  saving_api.save_model(
3546/3554 [============================>.] - ETA: 0s - loss: 1.9335e-05 - accuracy: 0.0695
Epoch 2: val_loss did not improve from 0.00002
3554/3554 [==============================] - 10s 3ms/step - loss: 1.9331e-05 - accuracy: 0.0696 - val_loss: 2.0112e-05 - val_accura
Epoch 3/50
3552/3554 [============================>.] - ETA: 0s - loss: 1.9447e-05 - accuracy: 0.0616
```

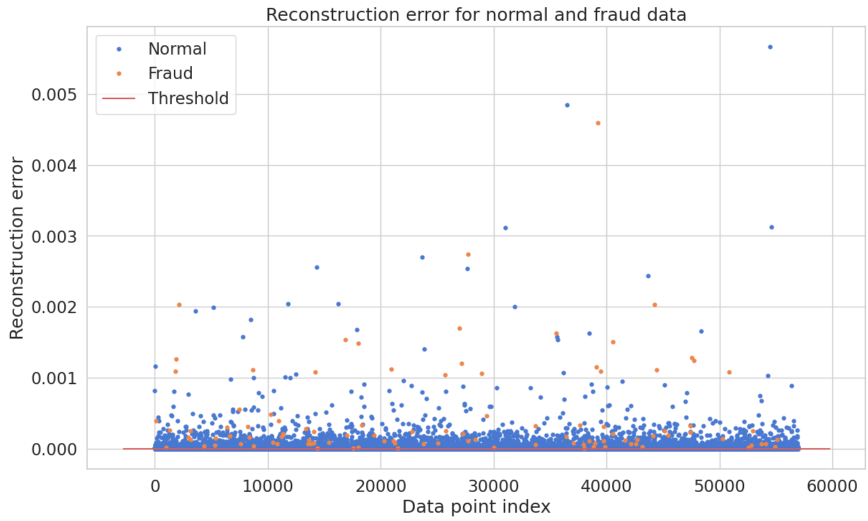
```
Epoch 3: val_loss did not improve from 0.00002
3554/3554 [==============================] - 9s 3ms/step - loss: 1.9445e-05 - accuracy: 0.0616 - val_loss: 2.0113e-05 - val_accuracy
Epoch 4/50
3540/3554 [============================>.] - ETA: 0s - loss: 1.9542e-05 - accuracy: 0.0636
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9550e-05 - accuracy: 0.0636 - val_loss: 2.0032e-05 - val_accuracy
Epoch 5/50
3545/3554 [============================>.] - ETA: 0s - loss: 1.9540e-05 - accuracy: 0.0609
Epoch 5: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9545e-05 - accuracy: 0.0609 - val_loss: 2.0122e-05 - val_accuracy
Epoch 6/50
3551/3554 [============================>.] - ETA: 0s - loss: 1.9547e-05 - accuracy: 0.0587
Epoch 6: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9549e-05 - accuracy: 0.0587 - val_loss: 2.0195e-05 - val_accuracy
Epoch 7/50
3544/3554 [============================>.] - ETA: 0s - loss: 1.9520e-05 - accuracy: 0.0594
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 7s 2ms/step - loss: 1.9533e-05 - accuracy: 0.0594 - val_loss: 2.0377e-05 - val_accuracy
Epoch 8/50
3535/3554 [============================>.] - ETA: 0s - loss: 1.9518e-05 - accuracy: 0.0596
Epoch 8: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9527e-05 - accuracy: 0.0600 - val_loss: 2.0062e-05 - val_accuracy
Epoch 9/50
3554/3554 [==============================] - ETA: 0s - loss: 1.9528e-05 - accuracy: 0.0581
Epoch 9: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9528e-05 - accuracy: 0.0581 - val_loss: 2.0419e-05 - val_accuracy
Epoch 10/50
3541/3554 [============================>.] - ETA: 0s - loss: 1.9514e-05 - accuracy: 0.0601
Epoch 10: val_loss did not improve from 0.00002
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9513e-05 - accuracy: 0.0600 - val_loss: 2.0367e-05 - val_accuracy
Epoch 11/50
3537/3554 [============================>.] - ETA: 0s - loss: 1.9505e-05 - accuracy: 0.0618
Epoch 11: val_loss did not improve from 0.00002
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 8s 2ms/step - loss: 1.9508e-05 - accuracy: 0.0617 - val_loss: 2.0001e-05 - val_accuracy
Epoch 11: early stopping
```

```
#Plot training and test loss
plt.plot(history['loss'], linewidth=2, label='Train')
plt.plot(history['val_loss'], linewidth=2, label='Test')
plt.legend(loc='upper right')
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
plt.show()
```

```
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})
```

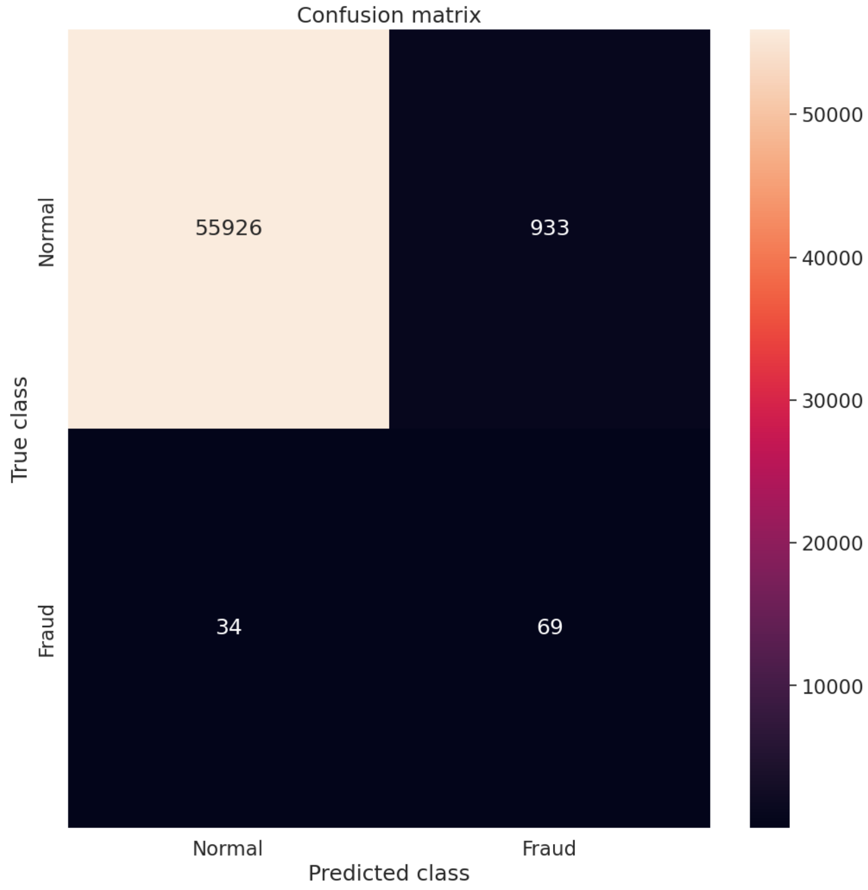
```
1781/1781 [==============================] - 2s 1ms/step
```

```
threshold_fixed = 0
groups = error_df.groupby('True_class')
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.index, group.Reconstruction_error, marker='o', ms=3.5, linestyle='',label= "Fraud" if name == 1 else "Normal")
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```




```
threshold_fixed = 0.0001
y_pred = [1 if e > threshold_fixed else 0 for e in error_df.Reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.True_class, y_pred)

plt.figure(figsize=(12, 12))
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
55926
933
34
69
Normal
Fraud
Normal
Fraud
True class
Predicted class
50000
40000
30000
20000
10000