In [ ]:
```
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_score

RANDOM_SEED=2021
TEST_PCT=0.3
LABELS=["Normal","Fraud"]
```

```
2022-11-01 14:44:03.985975: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is opti
mized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-cri
tical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-11-01 14:44:04.214580: E tensorflow/stream_executor/cuda/cuda_blas.cc:2981] Unable to register cuBLAS fact
ory: Attempting to register factory for plugin cuBLAS when one has already been registered
2022-11-01 14:44:04.952565: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dyna
mic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open shared object file: No such file or direct
ory
2022-11-01 14:44:04.952624: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dyna
mic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7: cannot open shared object file: No such
file or directory
2022-11-01 14:44:04.952633: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot dlop
en some TensorRT libraries. If you would like to use Nvidia GPU with TensorRT, please make sure the missing lib
raries mentioned above are installed properly.
```

In [ ]:
```
dataset=pd.read_csv("creditcard.csv")
```

In [ ]:
```
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())
```

```
Any nulls in the dataset False
---------
No of unique labels 2
Label values [0 1]
```

In [ ]:
```
print("--------")
```

```
--------
```

In [ ]:
```
print("--------")
print("Break Down of Normal and Fraud Transactions")
print(pd.value_counts(dataset['Class'],sort=True))
```

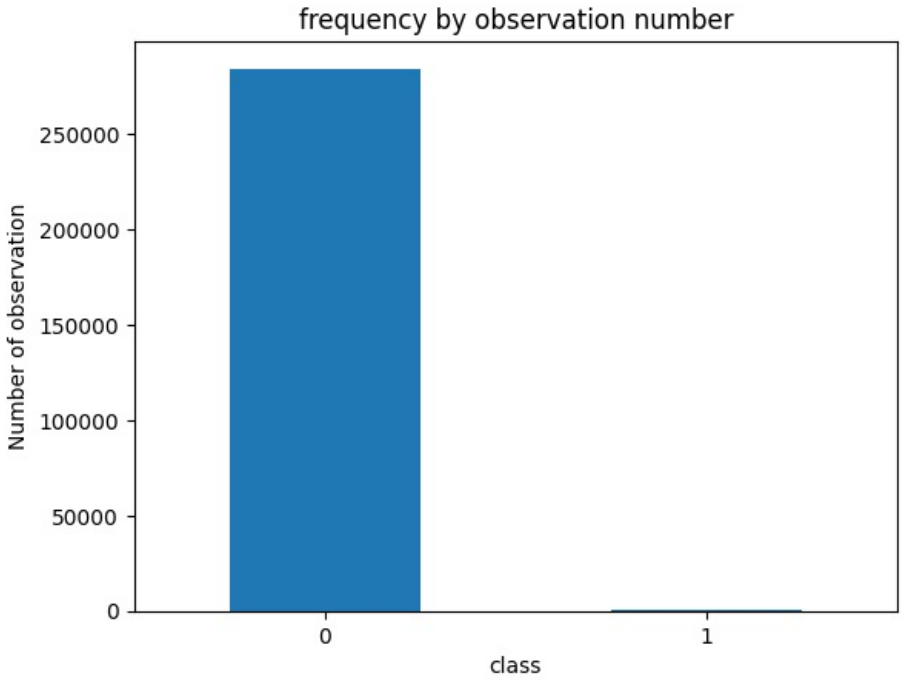
```
--------
Break Down of Normal and Fraud Transactions
0    284315
1       492
Name: Class, dtype: int64
```

In [ ]:
```
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())


print("--------")
print("Break Down of Normal and Fraud Transactions")
print(pd.value_counts(dataset['Class'],sort=True))
```

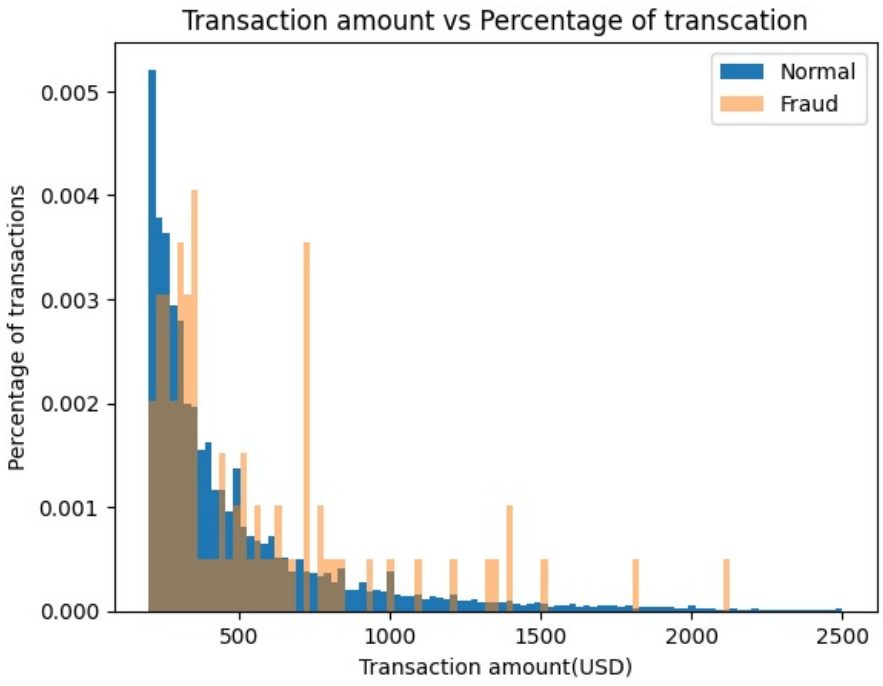
```
Any nulls in the dataset False
---------
No of unique labels 2
Label values [0 1]
--------
Break Down of Normal and Fraud Transactions
0    284315
1       492
Name: Class, dtype: int64
```

In [ ]:
```
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 observation")
```

Out[ ]: Text(0, 0.5, 'Number of observation')



In [ ]:
```
normal_dataset=dataset[dataset.Class ==0]
fraud_dataset =dataset[dataset.Class==1]


bins=np.linspace(200,2500,100)
plt.hist(normal_dataset.Amount,bins=bins,alpha=1,density=True,label='Normal')
plt.hist(fraud_dataset.Amount,bins=bins,alpha=0.5,density=True,label='Fraud')
plt.legend(loc='upper right')
plt.title("Transaction amount vs Percentage of transcation")
plt.xlabel("Transaction amount(USD)")
plt.ylabel("Percentage of transactions")
plt.show()
```



In [ ]:
```
sc=StandardScaler()
dataset['Time']=sc.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset['Amount']=sc.fit_transform(dataset['Amount'].values.reshape(-1,1))
raw_data=dataset.values
labels=raw_data[:,-1]

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
```

In [ ]:
```
min_val=tf.reduce_min(train_data)
max_val=tf.reduce_max(train_data)
```

```
train_data=(train_data-min_val)/(max_val - min_val)

train_data =tf.cast(train_data,tf.float32)

test_data=tf.cast(test_data,tf.float32)
```

```
2022-11-01 14:44:09.009721: E tensorflow/stream_executor/cuda/cuda_driver.cc:265] failed call to cuInit: CUDA_E
RROR_NO_DEVICE: no CUDA-capable device is detected
2022-11-01 14:44:09.009760: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not a
ppear to be running on this host (turing-machine): /proc/driver/nvidia/version does not exist
2022-11-01 14:44:09.011067: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is opti
mized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-cri
tical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-11-01 14:44:09.013227: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 54682800 exceed
s 10% of free system memory.
2022-11-01 14:44:09.088044: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 54682800 exceed
s 10% of free system memory.
2022-11-01 14:44:09.138783: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 54682800 exceed
s 10% of free system memory.
2022-11-01 14:44:09.181182: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 54682800 exceed
s 10% of free system memory.
2022-11-01 14:44:09.227936: W tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 54682800 exceed
s 10% of free system memory.
```

In [ ]:
```
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
```

In [ ]:
```
nb_epoch=50
batch_size =64
input_dim=normal_train_data.shape[1]

encoding_dim=14
hidden_dim_1=int(encoding_dim/2)
hidden_dim_2=4

learning_rate=1e-7
```

In [ ]:
```
input_layer =tf.keras.layers.Input(shape=(input_dim,))
encoder=tf.keras.layers.Dense(encoding_dim,activation="tanh", activity_regularizer=tf.keras.regularizers.l2(lea

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=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=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
_________________________________________________________________
```

In [ ]:
```
cp=tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h5",mode='min',monitor='val_loss',verbose=2,sa
early_stop=tf.keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=0.0001,patience=10,
                                            verbose=1,mode='min',restore_best_weights=True)
```

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

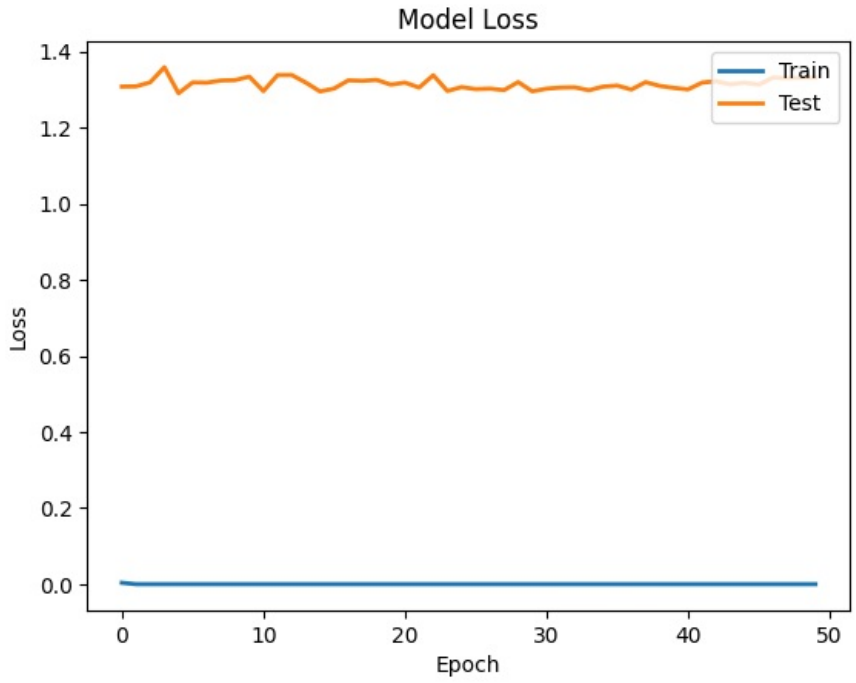
In [ ]:
```
history=autoencoder.fit(normal_train_data,normal_train_data,epochs=nb_epoch,batch_size=batch_size,shuffle=True,v
                        ).history
```

```
Epoch 1/50
3554/3554 [==============================] - 5s 1ms/step - loss: 0.0037 - accuracy: 0.0367 - val_loss: 1.3082 -
val_accuracy: 0.0244
Epoch 2/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9514e-05 - accuracy: 0.0609 - val_loss: 1.30
87 - val_accuracy: 0.0304
Epoch 3/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9459e-05 - accuracy: 0.0650 - val_loss: 1.31
93 - val_accuracy: 0.0477
Epoch 4/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9292e-05 - accuracy: 0.0814 - val_loss: 1.35
89 - val_accuracy: 0.0414
Epoch 5/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9619e-05 - accuracy: 0.0605 - val_loss: 1.29
06 - val_accuracy: 0.0578
Epoch 6/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9511e-05 - accuracy: 0.0598 - val_loss: 1.31
91 - val_accuracy: 0.0414
Epoch 7/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9564e-05 - accuracy: 0.0593 - val_loss: 1.31
85 - val_accuracy: 0.0387
Epoch 8/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9514e-05 - accuracy: 0.0614 - val_loss: 1.32
42 - val_accuracy: 0.1398
Epoch 9/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9497e-05 - accuracy: 0.0607 - val_loss: 1.32
51 - val_accuracy: 0.0340
Epoch 10/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9453e-05 - accuracy: 0.0573 - val_loss: 1.33
44 - val_accuracy: 0.0377
Epoch 11/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9434e-05 - accuracy: 0.0575 - val_loss: 1.29
63 - val_accuracy: 0.0280
Epoch 12/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9465e-05 - accuracy: 0.0587 - val_loss: 1.33
84 - val_accuracy: 0.0266
Epoch 13/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9480e-05 - accuracy: 0.0596 - val_loss: 1.33
86 - val_accuracy: 0.1413
Epoch 14/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9400e-05 - accuracy: 0.0604 - val_loss: 1.31
85 - val_accuracy: 0.0237
Epoch 15/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9403e-05 - accuracy: 0.0592 - val_loss: 1.29
53 - val_accuracy: 0.0591
Epoch 16/50
```

```
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9436e-05 - accuracy: 0.0593 - val_loss: 1.30
30 - val_accuracy: 0.0673
Epoch 17/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9354e-05 - accuracy: 0.0616 - val_loss: 1.32
46 - val_accuracy: 0.0571
Epoch 18/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9342e-05 - accuracy: 0.0649 - val_loss: 1.32
33 - val_accuracy: 0.0302
Epoch 19/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9388e-05 - accuracy: 0.0633 - val_loss: 1.32
57 - val_accuracy: 0.0675
Epoch 20/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9487e-05 - accuracy: 0.0619 - val_loss: 1.31
33 - val_accuracy: 0.0738
Epoch 21/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9400e-05 - accuracy: 0.0601 - val_loss: 1.31
87 - val_accuracy: 0.0388
Epoch 22/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9336e-05 - accuracy: 0.0594 - val_loss: 1.30
56 - val_accuracy: 0.0541
Epoch 23/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9349e-05 - accuracy: 0.0588 - val_loss: 1.33
80 - val_accuracy: 0.0267
Epoch 24/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9343e-05 - accuracy: 0.0629 - val_loss: 1.29
66 - val_accuracy: 0.0339
Epoch 25/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9451e-05 - accuracy: 0.0629 - val_loss: 1.30
67 - val_accuracy: 0.0348
Epoch 26/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9399e-05 - accuracy: 0.0621 - val_loss: 1.30
15 - val_accuracy: 0.0275
Epoch 27/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9411e-05 - accuracy: 0.0614 - val_loss: 1.30
28 - val_accuracy: 0.1872
Epoch 28/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9422e-05 - accuracy: 0.0643 - val_loss: 1.29
89 - val_accuracy: 0.0648
Epoch 29/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9344e-05 - accuracy: 0.0597 - val_loss: 1.32
00 - val_accuracy: 0.0319
Epoch 30/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9393e-05 - accuracy: 0.0607 - val_loss: 1.29
56 - val_accuracy: 0.0293
Epoch 31/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9357e-05 - accuracy: 0.0641 - val_loss: 1.30
27 - val_accuracy: 0.0910
Epoch 32/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9385e-05 - accuracy: 0.0612 - val_loss: 1.30
55 - val_accuracy: 0.0334
Epoch 33/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9310e-05 - accuracy: 0.0617 - val_loss: 1.30
62 - val_accuracy: 0.0336
Epoch 34/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9329e-05 - accuracy: 0.0623 - val_loss: 1.29
85 - val_accuracy: 0.0349
Epoch 35/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9302e-05 - accuracy: 0.0620 - val_loss: 1.30
80 - val_accuracy: 0.0562
Epoch 36/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9403e-05 - accuracy: 0.0634 - val_loss: 1.31
08 - val_accuracy: 0.0367
Epoch 37/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.9351e-05 - accuracy: 0.0617 - val_loss: 1.30
04 - val_accuracy: 0.0416
Epoch 38/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.9288e-05 - accuracy: 0.0615 - val_loss: 1.31
98 - val_accuracy: 0.0735
Epoch 39/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.8613e-05 - accuracy: 0.0828 - val_loss: 1.30
99 - val_accuracy: 0.1673
Epoch 40/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.8243e-05 - accuracy: 0.1361 - val_loss: 1.30
46 - val_accuracy: 0.1422
Epoch 41/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7948e-05 - accuracy: 0.2058 - val_loss: 1.30
06 - val_accuracy: 0.0358
Epoch 42/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7911e-05 - accuracy: 0.2422 - val_loss: 1.31
86 - val_accuracy: 0.0858
Epoch 43/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7456e-05 - accuracy: 0.2771 - val_loss: 1.32
21 - val_accuracy: 0.0818
```

```
Epoch 44/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7293e-05 - accuracy: 0.2954 - val_loss: 1.31
42 - val_accuracy: 0.0767
Epoch 45/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7212e-05 - accuracy: 0.2993 - val_loss: 1.31
85 - val_accuracy: 0.0723
Epoch 46/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7182e-05 - accuracy: 0.2975 - val_loss: 1.31
32 - val_accuracy: 0.0610
Epoch 47/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7300e-05 - accuracy: 0.2882 - val_loss: 1.33
18 - val_accuracy: 0.0610
Epoch 48/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7143e-05 - accuracy: 0.2970 - val_loss: 1.33
01 - val_accuracy: 0.0567
Epoch 49/50
3554/3554 [==============================] - 5s 1ms/step - loss: 1.7271e-05 - accuracy: 0.2763 - val_loss: 1.33
14 - val_accuracy: 0.0540
Epoch 50/50
3554/3554 [==============================] - 4s 1ms/step - loss: 1.7051e-05 - accuracy: 0.2967 - val_loss: 1.33
15 - val_accuracy: 0.0496
```

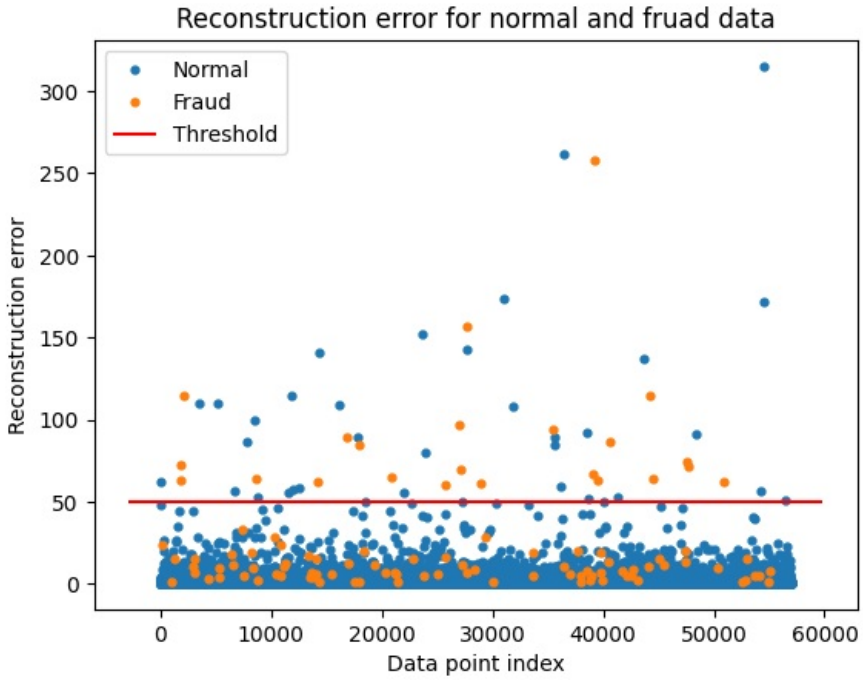
In [ ]:
```
plt.plot(history['loss'],linewidth=2,label='Train')
plt.plot(history['val_loss'],linewidth=2,label='Test')
plt.legend(loc='upper right')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
#plt.ylim(ymin=0.70,ymax=1)
plt.show()
```



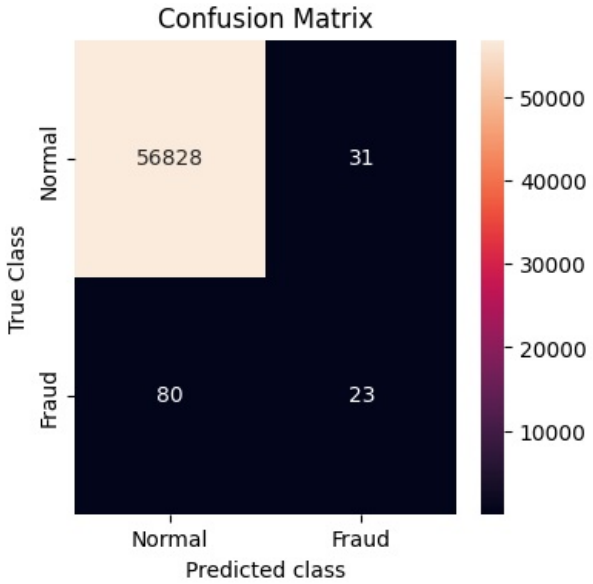
In [ ]:
```
test_x_predictions=autoencoder.predict(test_data)
mse=np.mean(np.power(test_data-test_x_predictions,2),axis=1)
error_df=pd.DataFrame({'Reconstruction_error':mse, 'True_class':test_labels})
```

```
1781/1781 [==============================] - 2s 892us/step
```

In [ ]:
```
threshold_fixed=50
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
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 fruad data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
```



In [ ]:
```
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()


print("Accuracy:", accuracy_score(error_df['True_class'],error_df['pred']))
print("REcall:", recall_score(error_df['True_class'],error_df['pred']))
print("precision:", precision_score(error_df['True_class'], error_df['pred']))
```




```
Accuracy: 0.9980513324672589
REcall: 0.22330097087378642
precision: 0.42592592592592593
```