In [75]:
```
!pip install seaborn
!pip install tensorflow --user
!pip install keras
!pip install daytime
!pip install torch
```

```
Defaulting to user installation because normal site-packages is not
writeable
Requirement already satisfied: seaborn in /home/student/.local/lib/
python3.10/site-packages (0.13.2)
Requirement already satisfied: pandas>=1.2 in /home/student/.local/
lib/python3.10/site-packages (from seaborn) (2.0.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /home/stu
dent/.local/lib/python3.10/site-packages (from seaborn) (3.7.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /home/studen
t/.local/lib/python3.10/site-packages (from seaborn) (1.24.3)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3/di
st-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (9.0.1)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in /usr/lib/py
thon3/dist-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in /home/studen
t/.local/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=
3.4->seaborn) (1.4.4)
Requirement already satisfied: cycler>=0.10 in /home/student/.loca
l/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seabo
```

In [76]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
```

In [77]:
```
dataset = pd.read_csv("creditcard.csv")
```

In [78]:
```
#dataset.head
print(dataset.columns)
dataset.describe()
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9'
, 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19
', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amo
unt',
       'Class'],
      dtype='object')
```

Out[78]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e

8 rows × 31 columns

In [ ]:

In [79]:
```
#checking for null value
dataset.isnull().sum()
```

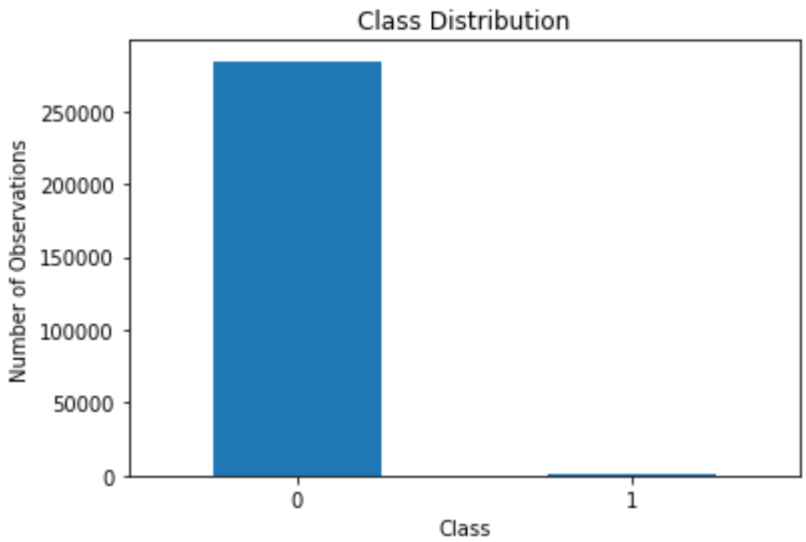
Out[79]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

In [80]:
```
print("No. of unique labels ", len(dataset['Class'].unique()))
print("Label values ", dataset.Class.unique())
```

```
No. of unique labels  2
Label values  [0 1]
```

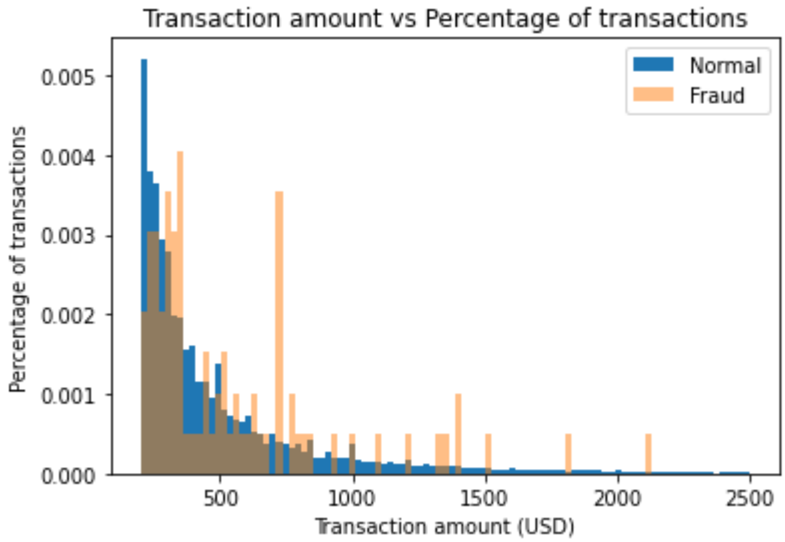
In [81]:
```
#0 is for normal credit card transaction
#1 is for fraudulent credit card transaction
```

In [82]:
```
import matplotlib.pyplot as plt

# Count occurrences of each class
count_classes = dataset['Class'].value_counts()

# Plotting
count_classes.plot(kind='bar')
plt.title("Class Distribution")
plt.xlabel("Class")
plt.ylabel("Number of Observations")
plt.xticks(rotation=0)
plt.show()
```



In [83]:
```
# Save the normal and fradulent transactions in separate dataframe

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

In [84]:
```
#Visualize transactionamounts for normal and fraudulent transactions
bins = np.linspace(200, 2500, 100)
plt.hist(normal_dataset.Amount, bins=bins, alpha=1, density=True, lab
plt.hist(fraud_dataset.Amount, bins=bins, alpha=0.5, density=True, la
plt.legend(loc='upper right')
plt.title("Transaction amount vs Percentage of transactions")
plt.xlabel("Transaction amount (USD)")
plt.ylabel("Percentage of transactions");
plt.show()
```



In [85]:
```
#Time and Amount are the columns that are not scaled, so applying Sta
#Normalizing the values between 0 and 1 did not work great for the da
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1,
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values.reshape
```

In [86]:
```

#The last column in the dataset is our target variable.

from sklearn.model_selection import train_test_split
raw_data = dataset.values

# The last element contains if the transaction is normal which is rep
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(d
```

In [ ]:

In [87]:
```
#normalizing the training and testing data to scale the values betwee

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)
```

In [88]:
```
#Use only normal transactions to train the Autoencoder.
#Normal data has a value of 0 in the target variable. Using the targe
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= 402
 No. of records in Normal Train data= 227443
 No. of records in Fraud Test Data= 90
 No. of records in Normal Test data= 56872
```

In [89]:
```
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
```

In [90]:
```
#input Layer
input_layer = tf.keras.layers.Input(shape=(input_dim, ))
```

In [91]:
```
#Encoder
encoder = tf.keras.layers.Dense(encoding_dim, activation="tanh",activ
encoder=tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(enco
encoder = tf.keras.layers.Dense(hidden_dim_2, activation=tf.nn.leaky_
```

In [92]:
```
# Decoder
decoder = tf.keras.layers.Dense(hidden_dim_1, activation='relu')(enco
decoder=tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim, activation='relu')(deco
decoder = tf.keras.layers.Dense(input_dim, activation='tanh')(decoder
```

In [93]:
```
#Autoencoder
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 30)]              0

 dense_6 (Dense)             (None, 14)                434

 dropout_2 (Dropout)         (None, 14)                0

 dense_7 (Dense)             (None, 7)                 105

 dense_8 (Dense)             (None, 4)                 32

 dense_9 (Dense)             (None, 7)                 35

 dropout_3 (Dropout)         (None, 7)                 0

 dense_10 (Dense)            (None, 14)                112

 dense_11 (Dense)            (None, 30)                450

=================================================================
Total params: 1168 (4.56 KB)
Trainable params: 1168 (4.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [94]:
```
#Define the callbacks for checkpoints and early stopping
cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.h
```

In [95]:
```
# 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)
```

In [96]:
```
#Compile the Autoencoder

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

In [97]:
```
#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
```

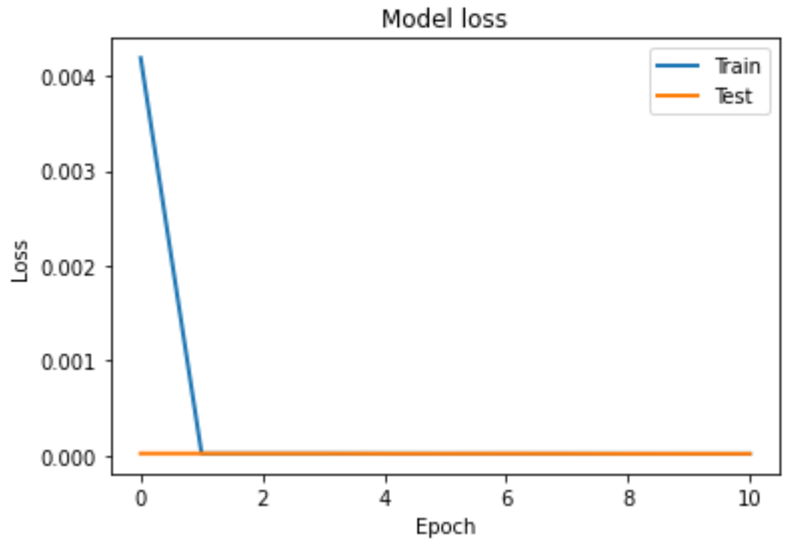
```
Epoch 1/50
3513/3554 [============================>.] - ETA: 0s - loss: 0.0042
- accuracy: 0.0325
Epoch 1: val_loss improved from inf to 0.00002, saving model to aut
oencoder_fraud.h5
3554/3554 [==============================] - 4s 865us/step - loss:
0.0042 - accuracy: 0.0326 - val_loss: 2.0033e-05 - val_accuracy:
0.1247
Epoch 2/50
 213/3554 [>.............................] - ETA: 2s - loss: 1.8245
e-05 - accuracy: 0.0352

/home/student/.local/lib/python3.10/site-packages/keras/src/engine/
training.py:3000: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g. `model.save
('my_model.keras')`.
  saving_api.save_model(
```

```
3483/3554 [============================>.] - ETA: 0s - loss: 1.9319
e-05 - accuracy: 0.0616
Epoch 2: val_loss improved from 0.00002 to 0.00002, saving model to
autoencoder_fraud.h5
3554/3554 [==============================] - 3s 833us/step - loss:
1.9307e-05 - accuracy: 0.0618 - val_loss: 1.9989e-05 - val_accuracy
: 0.0110
Epoch 3/50
3486/3554 [============================>.] - ETA: 0s - loss: 1.9314
e-05 - accuracy: 0.0658
Epoch 3: val_loss did not improve from 0.00002
3554/3554 [==============================] - 3s 828us/step - loss:
1.9352e-05 - accuracy: 0.0658 - val_loss: 2.0114e-05 - val_accuracy
: 0.2171
Epoch 4/50
3490/3554 [============================>.] - ETA: 0s - loss: 1.9375
e-05 - accuracy: 0.0692
Epoch 4: val_loss did not improve from 0.00002
3554/3554 [==============================] - 3s 825us/step - loss:
1.9409e-05 - accuracy: 0.0689 - val_loss: 2.0084e-05 - val_accuracy
: 0.0418
Epoch 5/50
3514/3554 [============================>.] - ETA: 0s - loss: 1.8848
e-05 - accuracy: 0.1712
Epoch 5: val_loss improved from 0.00002 to 0.00002, saving model to
autoencoder_fraud.h5
3554/3554 [==============================] - 3s 840us/step - loss:
1.8851e-05 - accuracy: 0.1723 - val_loss: 1.8681e-05 - val_accuracy
: 0.3232
Epoch 6/50
3513/3554 [============================>.] - ETA: 0s - loss: 1.8233
e-05 - accuracy: 0.2195
Epoch 6: val_loss improved from 0.00002 to 0.00002, saving model to
autoencoder_fraud.h5
3554/3554 [==============================] - 3s 825us/step - loss:
1.8233e-05 - accuracy: 0.2187 - val_loss: 1.8656e-05 - val_accuracy
: 0.2306
Epoch 7/50
3550/3554 [============================>.] - ETA: 0s - loss: 1.8183
e-05 - accuracy: 0.1444
Epoch 7: val_loss did not improve from 0.00002
3554/3554 [==============================] - 3s 832us/step - loss:
1.8182e-05 - accuracy: 0.1445 - val_loss: 1.9488e-05 - val_accuracy
: 0.2188
Epoch 8/50
3525/3554 [============================>.] - ETA: 0s - loss: 1.8202
e-05 - accuracy: 0.1272
Epoch 8: val_loss did not improve from 0.00002
3554/3554 [==============================] - 3s 819us/step - loss:
1.8193e-05 - accuracy: 0.1276 - val_loss: 1.8851e-05 - val_accuracy
: 0.2266
Epoch 9/50
3521/3554 [============================>.] - ETA: 0s - loss: 1.7961
e-05 - accuracy: 0.1892
Epoch 9: val_loss improved from 0.00002 to 0.00002, saving model to
autoencoder_fraud.h5
3554/3554 [==============================] - 3s 826us/step - loss:
1.7952e-05 - accuracy: 0.1898 - val_loss: 1.7849e-05 - val_accuracy
: 0.2472
Epoch 10/50
```

```
3519/3554 [============================>.] - ETA: 0s - loss: 1.7411
e-05 - accuracy: 0.2809
Epoch 10: val_loss improved from 0.00002 to 0.00002, saving model t
o autoencoder_fraud.h5
3554/3554 [==============================] - 3s 823us/step - loss:
1.7418e-05 - accuracy: 0.2811 - val_loss: 1.7601e-05 - val_accuracy
: 0.3152
Epoch 11/50
3505/3554 [============================>.] - ETA: 0s - loss: 1.7296
e-05 - accuracy: 0.2915
Epoch 11: val_loss improved from 0.00002 to 0.00002, saving model t
o autoencoder_fraud.h5
Restoring model weights from the end of the best epoch: 1.
3554/3554 [==============================] - 3s 829us/step - loss:
1.7306e-05 - accuracy: 0.2916 - val_loss: 1.7454e-05 - val_accuracy
: 0.3497
Epoch 11: early stopping
```

In [98]:

```
#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()
```

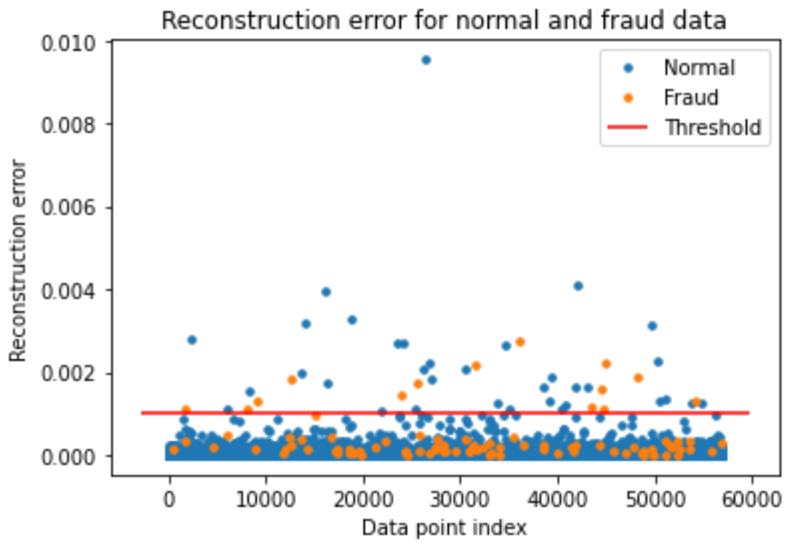


In [99]:

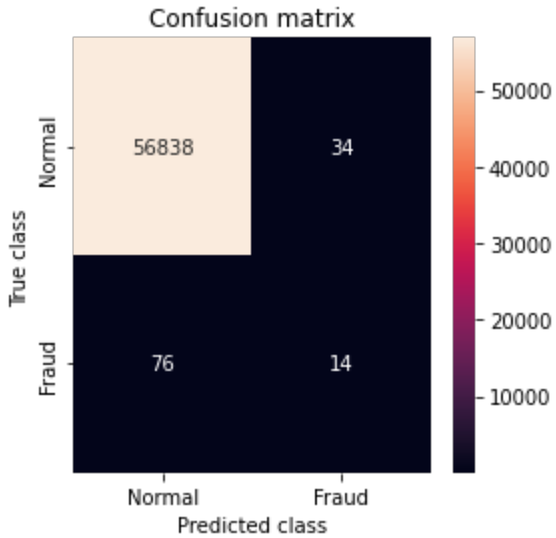
```
#Detect Anomalies on test data
'''
Anomalies are data points where the reconstruction loss is higher
To calculate the reconstruction loss on test data,predict the test
data and calculate the mean square error between the test data and th
'''
test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2), axis=1)
error_df = pd.DataFrame({'Reconstruction_error': mse,'True_class': te
```

```
1781/1781 [==============================] - 1s 458us/step
```

In [100]:
```
#Plotting the test data points and their respective reconstruction er
#if the threshold value needs to be adjusted.
threshold_fixed = 0.001
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
ax.hlines(threshold_fixed, ax.get_xlim()[0], ax.get_xlim()[1], colors
ax.legend()
plt.title("Reconstruction error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



In [106]:
```
'''Detect anomalies as points where the reconstruction loss is greate
a fixed threshold.
Here we see that a value of 52 for the threshold will be good.
Evaluating the performance of the anomaly detection'''

from sklearn.metrics import confusion_matrix, recall_score, accuracy_
import seaborn as sns

threshold_fixed =0.001
pred_y = [1 if e > threshold_fixed else 0 for e in error_df.Reconstru
error_df['pred'] =pred_y

conf_matrix = confusion_matrix(error_df.True_class, pred_y)

plt.figure(figsize=(4, 4))

LABELS = ["Normal","Fraud"]

sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, anno
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
# print Accuracy, precision and recall
print(" Accuracy: ",accuracy_score(error_df['True_class'], error_df[
print(" Recall: ",recall_score(error_df['True_class'], error_df['pred
print(" Precision: ",precision_score(error_df['True_class'], error_df
```




```
Accuracy:  0.9980688880306169
Recall:  0.15555555555555556
Precision:  0.2916666666666667
```

In [ ]: