In [3]:
```python
# Import all Necessary libraries and files from libraries
# We have used here Pandas,Numpy,scipy,seaborn,tensorflow,keras,sklears all labraries we are utilising for Model Building
# Pandas for Dataframe
# Numpy for numerical operation, Scipy for generation of Statistical data
# Tensorflow is used for Neural Network Model Building
# Keras Library used for Autoencoder and building Model on several Iterations.
# Also used Regularazation,PCA method.

import os
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
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
```

```
Using TensorFlow backend.
```

In [4]:
```python
# Load the Working Directory, where my Data file has stored.
os.chdir("D:\My ML Simulations\Linear Regression")
```

In [5]:
```python
# Load the data set, It is availabel on Kaggle for free.
carddata = pd.read_csv('creditcard.csv')
```

In [6]:
```python
# See the data set with Limit on only first 9 Rows.
carddata.head(n=9)
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373

9 rows × 31 columns

In [7]:
```python
carddata.shape
```

Out[7]:
```
(284807, 31)
```

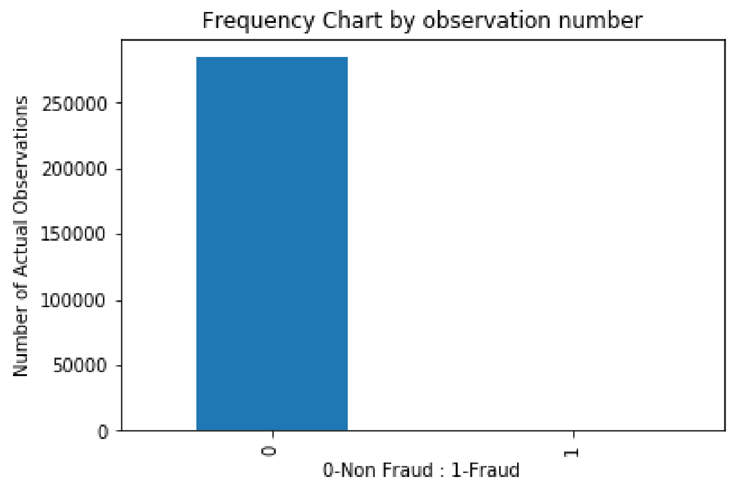
In [8]:
```python
# Is Null Method is used to find out any Null values in Data Set?
carddata.isnull().values.any()
```

Out[8]:
```
False
```

In [9]:
```python
# As we are building Model on Fraud Detection, Assign 1:Fraudulent Behavious and 0: Non(Normal Transaction)

pd.value_counts(carddata['Class'], sort = True)
```

Out[9]:
```
0    284315
1       492
Name: Class, dtype: int64
```

```
In [10]: # Graphical Represantation of same data as Its easy way to understand.

         count_classes = pd.value_counts(carddata['Class'], sort = True)
         count_classes.plot(kind = 'bar', rot=90)
         plt.xticks(range(2))
         plt.title("Frequency Chart by observation number")
         plt.xlabel("0-Non Fraud : 1-Fraud")
         plt.ylabel("Number of Actual Observations");
```




```
In [11]: normal_carddata = carddata[carddata.Class == 0]
```

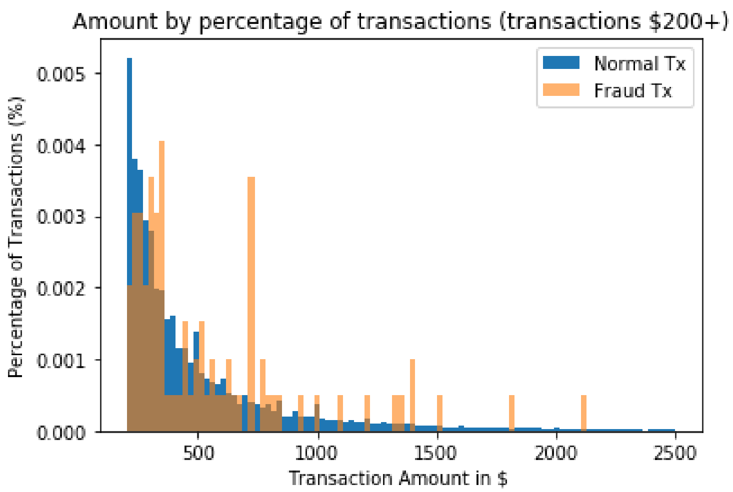
```
In [12]: Fraudulant_carddata = carddata[carddata.Class == 1]
```

```
In [13]: # summary statistics differences between fraud and normal transactions.
         # the mean is a little higher in the fraud transactions, it is certainly within a standard deviation and
         # so is unlikely to be easy to discriminate in a highly precise manner between the classes with pure statistical methods.

         normal_carddata.Amount.describe()
```

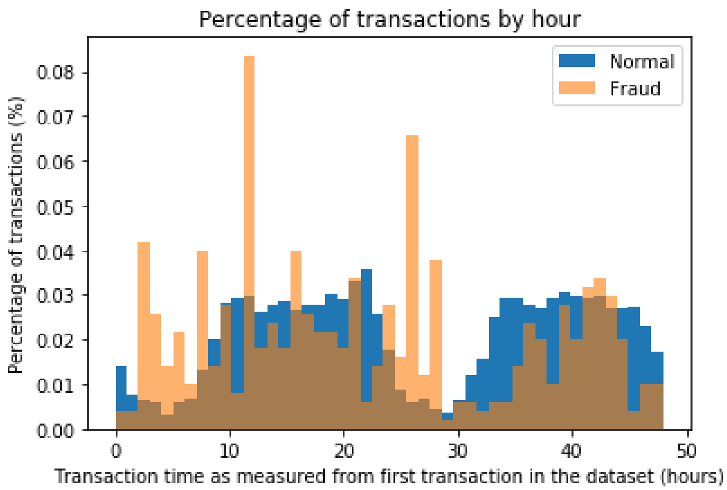
```
Out[13]: count    284315.000000
         mean         88.291022
         std         250.105092
         min           0.000000
         25%           5.650000
         50%          22.000000
         75%          77.050000
         max       25691.160000
         Name: Amount, dtype: float64
```

In [14]:
```
Fraudulant_carddata.Amount.describe()
```
Out[14]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

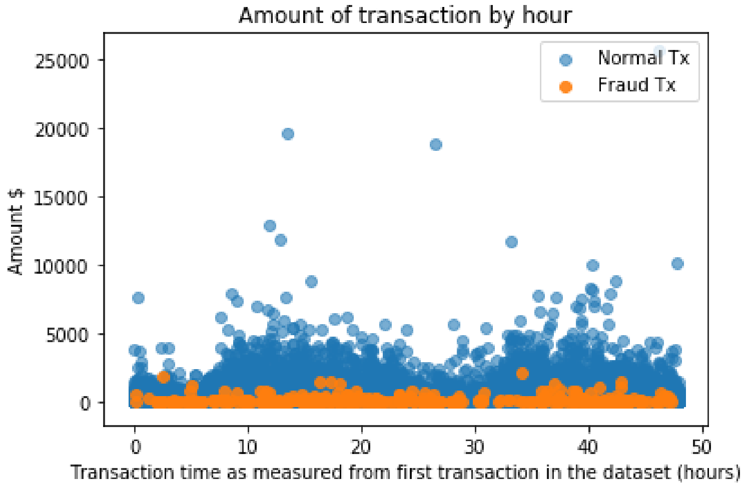
In [15]:
```
bins = np.linspace(200, 2500, 100)
plt.hist(normal_carddata.Amount, bins, alpha=1, density=True, label='Normal Tx')
plt.hist(Fraudulant_carddata.Amount, bins, alpha=0.6, density=True, label='Fraud Tx')
plt.legend(loc='upper right')
plt.title("Amount by percentage of transactions (transactions \$200+)")
plt.xlabel("Transaction Amount in $ ")
plt.ylabel("Percentage of Transactions (%)");
plt.show()

# the fraud cases are relatively few in number compared to bin size,
# It would be hard to differentiate fraud from normal transactions by transaction amount alone.
```



In [18]:
```
# The transaction amount does not look very informative. Let's look at the time of day:

bins = np.linspace(0, 48, 48) #48 hours
plt.hist((normal_carddata.Time/(60*60)), bins, alpha=1, density=True, label='Normal')
plt.hist((Fraudulant_carddata.Time/(60*60)), bins, alpha=0.6,density=True, label='Fraud')
plt.legend(loc='upper right')
plt.title("Percentage of transactions by hour")
plt.xlabel("Transaction time as measured from first transaction in the dataset (hours)")
plt.ylabel("Percentage of transactions (%)");
#plt.hist((df.Time/(60*60)),bins)
plt.show()

#  Fraud tends to occur at higher rates during the night. Statistical tests could be used to give evidence for this fact.
```



In [59]:
```
plt.scatter((normal_carddata.Time/(60*60)), normal_carddata.Amount, alpha=0.6, label='Normal Tx')
plt.scatter((Fraudulant_carddata.Time/(60*60)), Fraudulant_carddata.Amount, alpha=0.9, label='Fraud Tx')
plt.title("Amount of transaction by hour")
plt.xlabel("Transaction time as measured from first transaction in the dataset (hours)")
plt.ylabel('Amount $')
plt.legend(loc='upper right')
plt.show()
```



In [61]:
```
x = carddata
```

In [73]:
```
train_x, test_x = train_test_split(carddata_norm, test_size=0.2, random_state=0)
train_x = train_x[train_x.Class == 0]
train_y = train_x.drop(['Class'], axis=1)
```

In [63]:
```
test_y = test_x['Class']
test_x = test_x.drop(['Class'], axis=1)
```

In [64]:
```
train_x = train_x.values
test_x = test_x.values
```

In [74]:
```
train_x.shape
```

Out[74]:
```
(227454, 31)
```

In [66]:
```
import tensorflow as tf
```

In [67]:
```
print("Size of training set: ",train_x.shape)
```

```
Size of training set:  (227454, 31)
```

In [69]:
```
input_dim = train_x.shape[1]
encoding_dim = 14
input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="tanh",
                activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="relu")(encoder)
decoder = Dense(int(encoding_dim / 2), activation='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

In [75]:
```python
nb_epoch = 100
batch_size = 31
autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy'])
checkpointer = ModelCheckpoint(filepath="model.h5",
                               verbose=0,
                               save_best_only=True)
tensorboard = TensorBoard(log_dir='./logs',
                          histogram_freq=0,
                          write_graph=True,
                          write_images=True)
history = autoencoder.fit(train_x, train_x,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(test_x, test_x),
                    verbose=1,
                    callbacks=[checkpointer, tensorboard]).history
```

```
WARNING:tensorflow:From C:\Users\nilesh\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:1033: The name tf.assign
_add is deprecated. Please use tf.compat.v1.assign_add instead.

Train on 227454 samples, validate on 56962 samples
WARNING:tensorflow:From C:\Users\nilesh\Anaconda3\lib\site-packages\keras\callbacks.py:1122: The name tf.summary.merge_all is de
precated. Please use tf.compat.v1.summary.merge_all instead.

WARNING:tensorflow:From C:\Users\nilesh\Anaconda3\lib\site-packages\keras\callbacks.py:1125: The name tf.summary.FileWriter is d
eprecated. Please use tf.compat.v1.summary.FileWriter instead.

Epoch 1/100
227454/227454 [==============================] - 16s 71us/step - loss: 0.7811 - acc: 0.5809 - val_loss: 0.7881 - val_acc: 0.6401
Epoch 2/100
227454/227454 [==============================] - 14s 63us/step - loss: 0.7175 - acc: 0.6451 - val_loss: 0.7751 - val_acc: 0.6430
Epoch 3/100
227454/227454 [==============================] - 14s 64us/step - loss: 0.7083 - acc: 0.6537 - val_loss: 0.7702 - val_acc: 0.6534
Epoch 4/100
227454/227454 [==============================] - 14s 63us/step - loss: 0.7034 - acc: 0.6611 - val_loss: 0.7648 - val_acc: 0.6720
Epoch 5/100
227454/227454 [==============================] - 15s 65us/step - loss: 0.6988 - acc: 0.6704 - val_loss: 0.7589 - val_acc: 0.6798
Epoch 6/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6947 - acc: 0.6749 - val_loss: 0.7590 - val_acc: 0.6841
Epoch 7/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6928 - acc: 0.6782 - val_loss: 0.7553 - val_acc: 0.6836
Epoch 8/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6919 - acc: 0.6793 - val_loss: 0.7545 - val_acc: 0.6782
Epoch 9/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6909 - acc: 0.6810 - val_loss: 0.7523 - val_acc: 0.6759
Epoch 10/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6901 - acc: 0.6814 - val_loss: 0.7516 - val_acc: 0.6868
Epoch 11/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6896 - acc: 0.6835 - val_loss: 0.7534 - val_acc: 0.6839
Epoch 12/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6890 - acc: 0.6845 - val_loss: 0.7512 - val_acc: 0.6848
Epoch 13/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6886 - acc: 0.6856 - val_loss: 0.7502 - val_acc: 0.6843
Epoch 14/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6881 - acc: 0.6867 - val_loss: 0.7490 - val_acc: 0.6869
Epoch 15/100
227454/227454 [==============================] - 16s 69us/step - loss: 0.6877 - acc: 0.6870 - val_loss: 0.7477 - val_acc: 0.6880
Epoch 16/100
227454/227454 [==============================] - 16s 72us/step - loss: 0.6873 - acc: 0.6872 - val_loss: 0.7484 - val_acc: 0.6883
Epoch 17/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6873 - acc: 0.6888 - val_loss: 0.7478 - val_acc: 0.6944
Epoch 18/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6868 - acc: 0.6886 - val_loss: 0.7484 - val_acc: 0.6904
Epoch 19/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6867 - acc: 0.6900 - val_loss: 0.7490 - val_acc: 0.6957
Epoch 20/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6860 - acc: 0.6921 - val_loss: 0.7467 - val_acc: 0.6957
Epoch 21/100
227454/227454 [==============================] - 17s 73us/step - loss: 0.6858 - acc: 0.6932 - val_loss: 0.7462 - val_acc: 0.6970
```

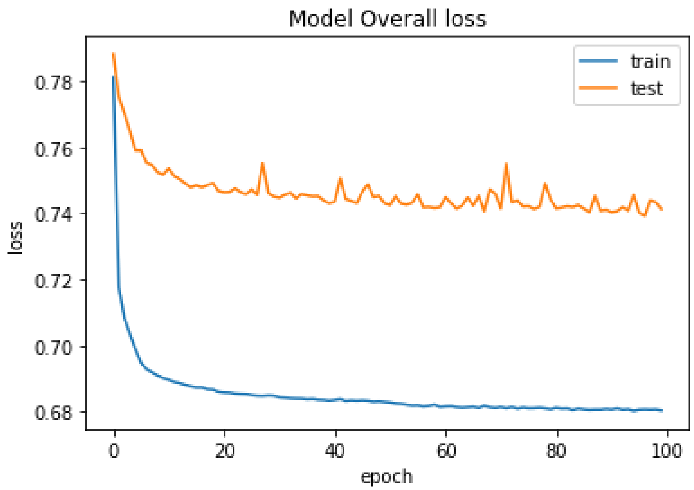
```
Epoch 22/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6857 - acc: 0.6937 - val_loss: 0.7463 - val_acc: 0.6859
Epoch 23/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6854 - acc: 0.6945 - val_loss: 0.7475 - val_acc: 0.7005
Epoch 24/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6853 - acc: 0.6940 - val_loss: 0.7463 - val_acc: 0.6975
Epoch 25/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6853 - acc: 0.6933 - val_loss: 0.7457 - val_acc: 0.6934
Epoch 26/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6850 - acc: 0.6940 - val_loss: 0.7471 - val_acc: 0.6927
Epoch 27/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6848 - acc: 0.6943 - val_loss: 0.7455 - val_acc: 0.6894
Epoch 28/100
227454/227454 [==============================] - 15s 66us/step - loss: 0.6847 - acc: 0.6950 - val_loss: 0.7551 - val_acc: 0.6808
Epoch 29/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6849 - acc: 0.6947 - val_loss: 0.7460 - val_acc: 0.6890
Epoch 30/100
227454/227454 [==============================] - 16s 72us/step - loss: 0.6848 - acc: 0.6951 - val_loss: 0.7450 - val_acc: 0.6935
Epoch 31/100
227454/227454 [==============================] - 15s 64us/step - loss: 0.6843 - acc: 0.6945 - val_loss: 0.7446 - val_acc: 0.7000
Epoch 32/100
227454/227454 [==============================] - 14s 63us/step - loss: 0.6842 - acc: 0.6954 - val_loss: 0.7455 - val_acc: 0.6937
Epoch 33/100
227454/227454 [==============================] - 15s 64us/step - loss: 0.6840 - acc: 0.6955 - val_loss: 0.7462 - val_acc: 0.6876
Epoch 34/100
227454/227454 [==============================] - 14s 63us/step - loss: 0.6840 - acc: 0.6957 - val_loss: 0.7444 - val_acc: 0.6989
Epoch 35/100
227454/227454 [==============================] - 14s 61us/step - loss: 0.6840 - acc: 0.6953 - val_loss: 0.7457 - val_acc: 0.7017
Epoch 36/100
227454/227454 [==============================] - 15s 64us/step - loss: 0.6837 - acc: 0.6961 - val_loss: 0.7453 - val_acc: 0.6986
Epoch 37/100
227454/227454 [==============================] - 16s 71us/step - loss: 0.6838 - acc: 0.6948 - val_loss: 0.7451 - val_acc: 0.6973
Epoch 38/100
227454/227454 [==============================] - 16s 70us/step - loss: 0.6836 - acc: 0.6953 - val_loss: 0.7451 - val_acc: 0.6979
Epoch 39/100
227454/227454 [==============================] - 17s 74us/step - loss: 0.6835 - acc: 0.6960 - val_loss: 0.7438 - val_acc: 0.7011
Epoch 40/100
227454/227454 [==============================] - 15s 66us/step - loss: 0.6833 - acc: 0.6966 - val_loss: 0.7430 - val_acc: 0.6937
Epoch 41/100
227454/227454 [==============================] - 15s 66us/step - loss: 0.6835 - acc: 0.6961 - val_loss: 0.7434 - val_acc: 0.6959
Epoch 42/100
227454/227454 [==============================] - 15s 66us/step - loss: 0.6838 - acc: 0.6960 - val_loss: 0.7505 - val_acc: 0.6956
Epoch 43/100
227454/227454 [==============================] - 15s 64us/step - loss: 0.6832 - acc: 0.6963 - val_loss: 0.7442 - val_acc: 0.6904
Epoch 44/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6834 - acc: 0.6959 - val_loss: 0.7436 - val_acc: 0.7011
Epoch 45/100
227454/227454 [==============================] - 16s 71us/step - loss: 0.6833 - acc: 0.6957 - val_loss: 0.7430 - val_acc: 0.6970
Epoch 46/100
227454/227454 [==============================] - 13s 58us/step - loss: 0.6834 - acc: 0.6953 - val_loss: 0.7465 - val_acc: 0.6815
Epoch 47/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6833 - acc: 0.6965 - val_loss: 0.7487 - val_acc: 0.6899
Epoch 48/100
```

```
227454/227454 [==============================] - 15s 67us/step - loss: 0.6830 - acc: 0.6956 - val_loss: 0.7448 - val_acc: 0.6978
Epoch 49/100
227454/227454 [==============================] - 13s 56us/step - loss: 0.6831 - acc: 0.6953 - val_loss: 0.7452 - val_acc: 0.6923
Epoch 50/100
227454/227454 [==============================] - 19s 85us/step - loss: 0.6829 - acc: 0.6956 - val_loss: 0.7430 - val_acc: 0.7075
Epoch 51/100
227454/227454 [==============================] - 19s 81us/step - loss: 0.6828 - acc: 0.6924 - val_loss: 0.7424 - val_acc: 0.6921
Epoch 52/100
227454/227454 [==============================] - 18s 80us/step - loss: 0.6824 - acc: 0.6928 - val_loss: 0.7451 - val_acc: 0.6859
Epoch 53/100
227454/227454 [==============================] - 18s 81us/step - loss: 0.6823 - acc: 0.6920 - val_loss: 0.7429 - val_acc: 0.6918
Epoch 54/100
227454/227454 [==============================] - 19s 83us/step - loss: 0.6821 - acc: 0.6919 - val_loss: 0.7425 - val_acc: 0.6853
Epoch 55/100
227454/227454 [==============================] - 17s 76us/step - loss: 0.6818 - acc: 0.6916 - val_loss: 0.7432 - val_acc: 0.6973
Epoch 56/100
227454/227454 [==============================] - 15s 66us/step - loss: 0.6819 - acc: 0.6922 - val_loss: 0.7456 - val_acc: 0.6947
Epoch 57/100
227454/227454 [==============================] - 13s 59us/step - loss: 0.6816 - acc: 0.6918 - val_loss: 0.7417 - val_acc: 0.6980
Epoch 58/100
227454/227454 [==============================] - 14s 61us/step - loss: 0.6817 - acc: 0.6920 - val_loss: 0.7419 - val_acc: 0.6927
Epoch 59/100
227454/227454 [==============================] - 17s 76us/step - loss: 0.6821 - acc: 0.6919 - val_loss: 0.7416 - val_acc: 0.6830
Epoch 60/100
227454/227454 [==============================] - 19s 84us/step - loss: 0.6814 - acc: 0.6923 - val_loss: 0.7418 - val_acc: 0.6986
Epoch 61/100
227454/227454 [==============================] - 17s 74us/step - loss: 0.6816 - acc: 0.6935 - val_loss: 0.7448 - val_acc: 0.6979
Epoch 62/100
227454/227454 [==============================] - 16s 72us/step - loss: 0.6816 - acc: 0.6934 - val_loss: 0.7429 - val_acc: 0.6872
Epoch 63/100
227454/227454 [==============================] - 16s 68us/step - loss: 0.6814 - acc: 0.6941 - val_loss: 0.7414 - val_acc: 0.6885
Epoch 64/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6812 - acc: 0.6949 - val_loss: 0.7423 - val_acc: 0.6835
Epoch 65/100
227454/227454 [==============================] - 14s 63us/step - loss: 0.6813 - acc: 0.6953 - val_loss: 0.7447 - val_acc: 0.6978
Epoch 66/100
227454/227454 [==============================] - 14s 62us/step - loss: 0.6815 - acc: 0.6952 - val_loss: 0.7422 - val_acc: 0.6947
Epoch 67/100
227454/227454 [==============================] - 16s 72us/step - loss: 0.6811 - acc: 0.6949 - val_loss: 0.7452 - val_acc: 0.6723
Epoch 68/100
227454/227454 [==============================] - 14s 62us/step - loss: 0.6818 - acc: 0.6954 - val_loss: 0.7406 - val_acc: 0.6979
Epoch 69/100
227454/227454 [==============================] - 15s 66us/step - loss: 0.6813 - acc: 0.6954 - val_loss: 0.7472 - val_acc: 0.6924
Epoch 70/100
227454/227454 [==============================] - 15s 68us/step - loss: 0.6811 - acc: 0.6963 - val_loss: 0.7458 - val_acc: 0.6841
Epoch 71/100
227454/227454 [==============================] - 16s 68us/step - loss: 0.6814 - acc: 0.6971 - val_loss: 0.7414 - val_acc: 0.6937
Epoch 72/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6810 - acc: 0.6978 - val_loss: 0.7550 - val_acc: 0.6736
Epoch 73/100
227454/227454 [==============================] - 15s 65us/step - loss: 0.6814 - acc: 0.6970 - val_loss: 0.7432 - val_acc: 0.6955
Epoch 74/100
227454/227454 [==============================] - 14s 60us/step - loss: 0.6809 - acc: 0.6984 - val_loss: 0.7438 - val_acc: 0.6945
```

```
Epoch 75/100
227454/227454 [==============================] - 14s 61us/step - loss: 0.6813 - acc: 0.6979 - val_loss: 0.7420 - val_acc: 0.6977
Epoch 76/100
227454/227454 [==============================] - 13s 59us/step - loss: 0.6810 - acc: 0.6982 - val_loss: 0.7421 - val_acc: 0.6940
Epoch 77/100
227454/227454 [==============================] - 16s 69us/step - loss: 0.6811 - acc: 0.6982 - val_loss: 0.7412 - val_acc: 0.6971
Epoch 78/100
227454/227454 [==============================] - 17s 77us/step - loss: 0.6812 - acc: 0.6980 - val_loss: 0.7419 - val_acc: 0.7008
Epoch 79/100
227454/227454 [==============================] - 14s 60us/step - loss: 0.6810 - acc: 0.6986 - val_loss: 0.7489 - val_acc: 0.6875
Epoch 80/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6807 - acc: 0.6988 - val_loss: 0.7441 - val_acc: 0.6951
Epoch 81/100
227454/227454 [==============================] - 14s 61us/step - loss: 0.6812 - acc: 0.6991 - val_loss: 0.7414 - val_acc: 0.6954
Epoch 82/100
227454/227454 [==============================] - 14s 62us/step - loss: 0.6809 - acc: 0.6980 - val_loss: 0.7417 - val_acc: 0.7003
Epoch 83/100
227454/227454 [==============================] - 14s 60us/step - loss: 0.6810 - acc: 0.6996 - val_loss: 0.7421 - val_acc: 0.6947
Epoch 84/100
227454/227454 [==============================] - 14s 61us/step - loss: 0.6805 - acc: 0.6996 - val_loss: 0.7418 - val_acc: 0.6955
Epoch 85/100
227454/227454 [==============================] - 13s 57us/step - loss: 0.6809 - acc: 0.6995 - val_loss: 0.7424 - val_acc: 0.6977
Epoch 86/100
227454/227454 [==============================] - 13s 56us/step - loss: 0.6807 - acc: 0.6996 - val_loss: 0.7414 - val_acc: 0.6993
Epoch 87/100
227454/227454 [==============================] - 14s 62us/step - loss: 0.6805 - acc: 0.6998 - val_loss: 0.7403 - val_acc: 0.7038
Epoch 88/100
227454/227454 [==============================] - 13s 59us/step - loss: 0.6806 - acc: 0.6995 - val_loss: 0.7452 - val_acc: 0.6948
Epoch 89/100
227454/227454 [==============================] - 14s 59us/step - loss: 0.6806 - acc: 0.7008 - val_loss: 0.7407 - val_acc: 0.6935
Epoch 90/100
227454/227454 [==============================] - 13s 59us/step - loss: 0.6808 - acc: 0.7001 - val_loss: 0.7410 - val_acc: 0.6999
Epoch 91/100
227454/227454 [==============================] - 14s 60us/step - loss: 0.6807 - acc: 0.7007 - val_loss: 0.7403 - val_acc: 0.7057
Epoch 92/100
227454/227454 [==============================] - 15s 67us/step - loss: 0.6810 - acc: 0.6999 - val_loss: 0.7406 - val_acc: 0.7050
Epoch 93/100
227454/227454 [==============================] - 13s 58us/step - loss: 0.6806 - acc: 0.7000 - val_loss: 0.7418 - val_acc: 0.6954
Epoch 94/100
227454/227454 [==============================] - 13s 57us/step - loss: 0.6807 - acc: 0.7004 - val_loss: 0.7408 - val_acc: 0.6924
Epoch 95/100
227454/227454 [==============================] - 14s 60us/step - loss: 0.6802 - acc: 0.7007 - val_loss: 0.7455 - val_acc: 0.6950
Epoch 96/100
227454/227454 [==============================] - 13s 59us/step - loss: 0.6806 - acc: 0.7007 - val_loss: 0.7401 - val_acc: 0.7005
Epoch 97/100
227454/227454 [==============================] - 13s 58us/step - loss: 0.6807 - acc: 0.7005 - val_loss: 0.7392 - val_acc: 0.7007
Epoch 98/100
227454/227454 [==============================] - 13s 58us/step - loss: 0.6806 - acc: 0.7008 - val_loss: 0.7438 - val_acc: 0.7026
Epoch 99/100
227454/227454 [==============================] - 13s 56us/step - loss: 0.6807 - acc: 0.7008 - val_loss: 0.7432 - val_acc: 0.7023
Epoch 100/100
227454/227454 [==============================] - 13s 59us/step - loss: 0.6804 - acc: 0.7010 - val_loss: 0.7412 - val_acc: 0.6993
```

In [76]:
```
autoencoder = load_model('model.h5')
```

In [93]:
```
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('Model Overall loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right');
```

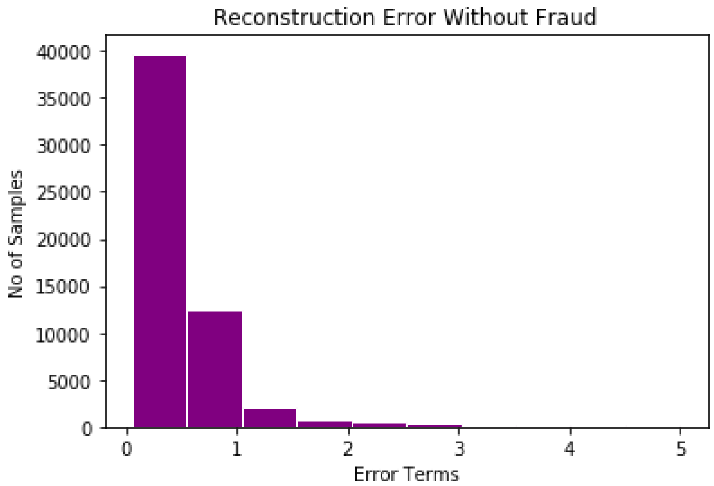


In [78]:
```
predictions = autoencoder.predict(test_x)
mse = np.mean(np.power(test_x - predictions, 2), axis=1)
error_carddata = pd.DataFrame({'reconstruction_error': mse,
                        'true_class': test_y})
error_carddata.describe()
```

Out[78]:

	reconstruction_error	true_class
count	56962.000000	56962.000000
mean	0.730099	0.001773
std	6.908029	0.042071
min	0.049395	0.000000
25%	0.243083	0.000000
50%	0.386764	0.000000
75%	0.604005	0.000000
max	1452.679951	1.000000

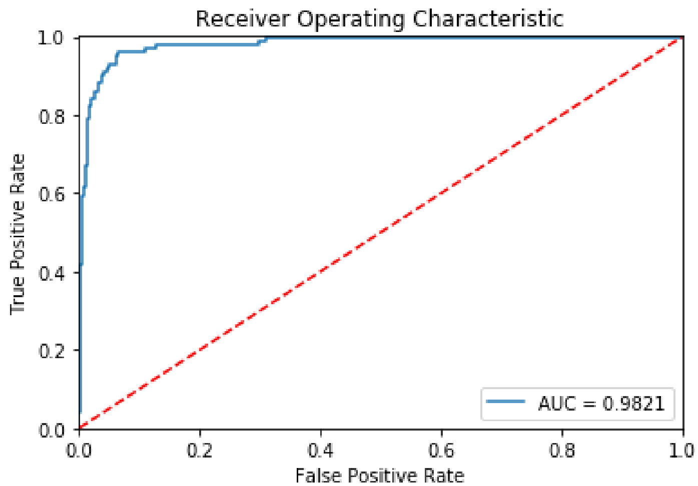
In [99]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
normal_error_carddata = error_carddata[(error_carddata['true_class']== 0) & (error_carddata['reconstruction_error'] < 5)]
_ = ax.hist(normal_error_carddata.reconstruction_error.values, color = 'purple',edgecolor = 'white',bins = 10)
plt.title('Reconstruction Error Without Fraud')
plt.ylabel('No of Samples')
plt.xlabel('Error Terms')
plt.show()
```



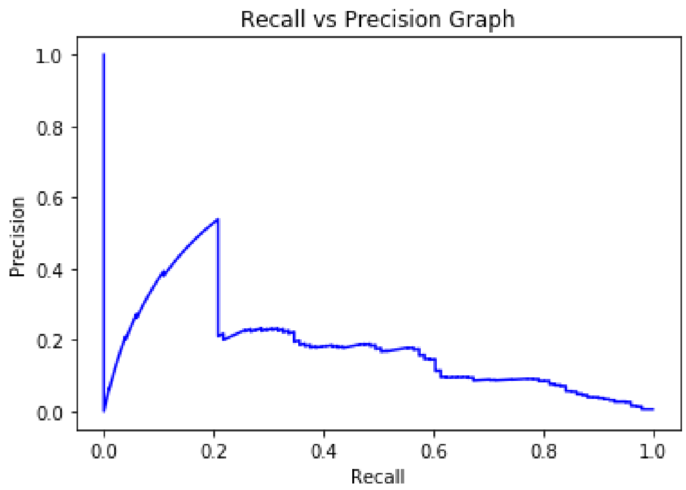
In [100]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
fraud_error_carddata = error_carddata[error_carddata['true_class'] == 1]
_ = ax.hist(fraud_error_carddata.reconstruction_error.values, color = 'red',edgecolor = 'white',bins = 10)
plt.title('Reconstruction Error With Fraud')
plt.ylabel('No of Samples')
plt.xlabel('Error Terms')
plt.show()
```

In [101]:
```
from sklearn.metrics import (confusion_matrix, precision_recall_curve, auc,
                             roc_curve, recall_score, classification_report, f1_score,
                             precision_recall_fscore_support)
```

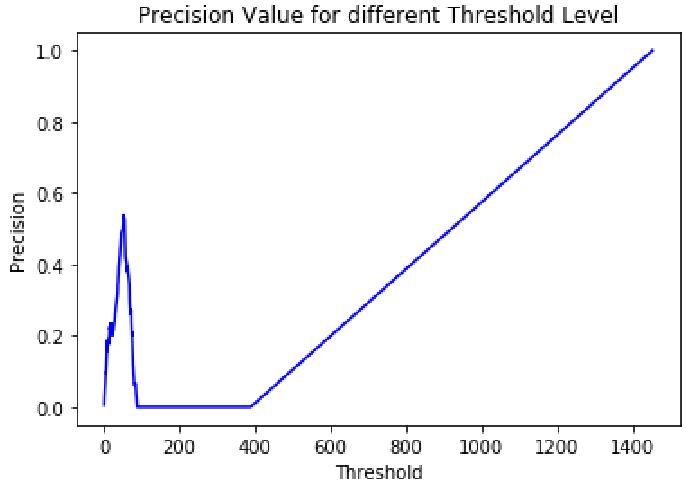
In [103]:
```python
fpr, tpr, thresholds = roc_curve(error_carddata.true_class, error_carddata.reconstruction_error)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='AUC = %0.4f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.001, 1])
plt.ylim([0, 1.001])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show();

# Receiver operating characteristic curves are an expected output of most binary classifiers.
# Since we have an imbalanced data set they are somewhat less useful.
#  Basically, we want the blue line to be as close as possible to the upper left corner.
```



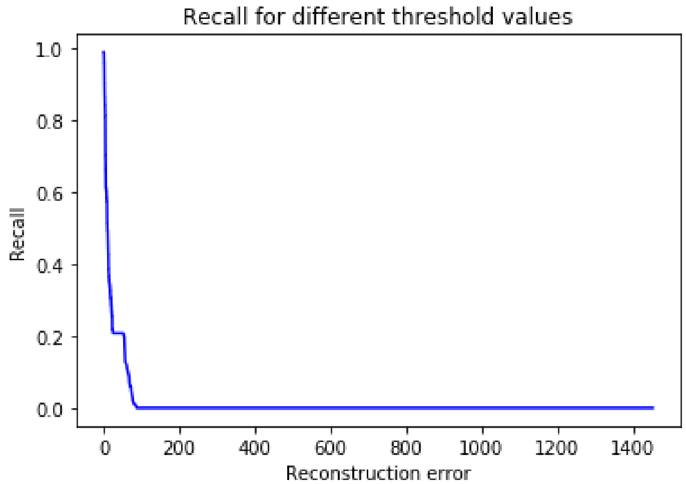
In [105]:
```
precision, recall, th = precision_recall_curve(error_carddata.true_class, error_carddata.reconstruction_error)
plt.plot(recall, precision, 'b', label='Precision-Recall curve')
plt.title('Recall vs Precision Graph')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()

#  High recall but low precision means many results, most of which has low or no relevancy.
#  When precision is high but recall is low we have the opposite — few returned results with very high relevancy.
# Ideally, you would want high precision and high recall — many results with that are highly relevant.
```



In [107]:
```
plt.plot(th, precision[1:], 'b', label='Threshold-Precision curve')
plt.title('Precision Value for different Threshold Level')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.show()

# You can see that as the reconstruction error increases our precision rises as
```

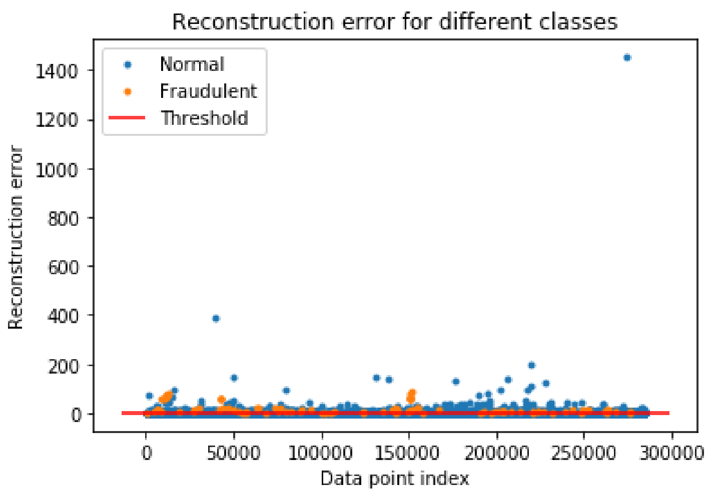


In [108]:
```
plt.plot(th, recall[1:], 'b', label='Threshold-Recall curve')
plt.title('Recall for different threshold values')
plt.xlabel('Reconstruction error')
plt.ylabel('Recall')
plt.show()

# Here, we have the exact opposite situation. As the reconstruction error increases the recall decreases.
```



In [112]:
```
threshold = 3.1
```

In [122]:
```
groups = error_carddata.groupby('true_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.reconstruction_error, marker='o', ms=3, linestyle='',
            label= "Fraudulent" if name == 1 else "Normal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction Error For Different Classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



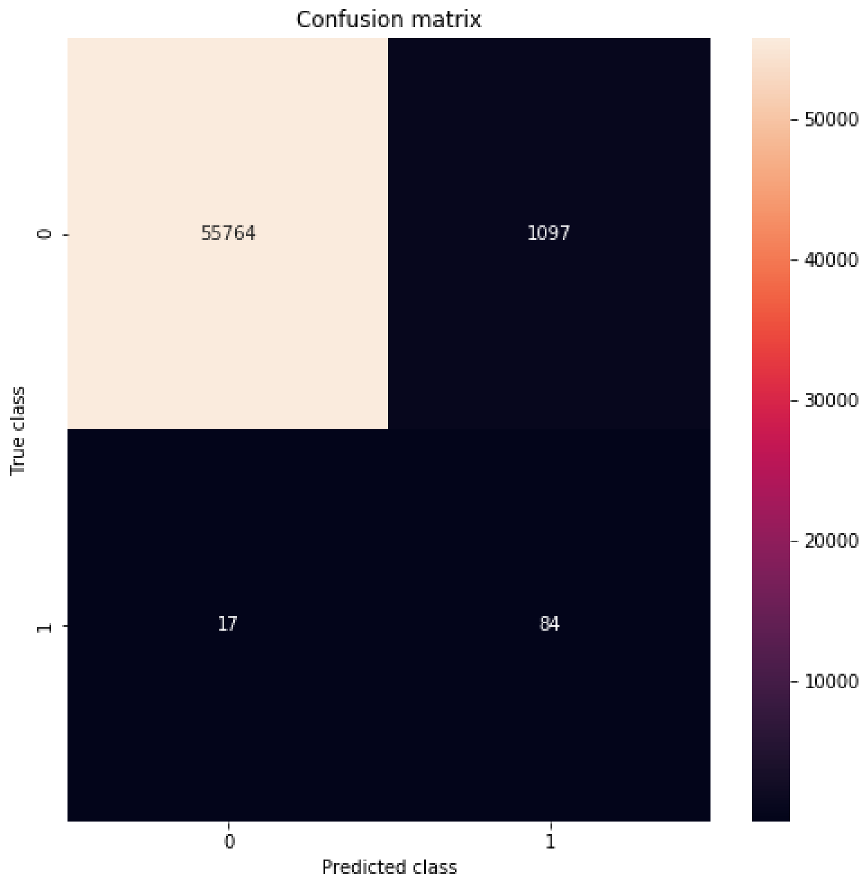
In [133]:
```
y_pred = [1 if e > threshold else 0 for e in error_carddata.reconstruction_error.values]
conf_matrix = confusion_matrix(error_carddata.true_class, y_pred)
```

In [136]:
```
print(conf_matrix)
```

```
[[55764  1097]
 [   17    84]]
```

In [151]:

```
plt.figure(figsize=(8, 8))
sns.heatmap(conf_matrix,xticklabels='auto', yticklabels='auto',robust=False, annot= True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



In [ ]:

```
# Our model seems to catch a lot of the fraudulent cases. Of course, there is a catch.
# We might want to increase or decrease the value of the threshold, depending on the problem.
# I presented the business case for card payment fraud detection and provided a brief overview of the algorithms in use.
#
```