Credit Card Fraud

May 3, 2018

1 Credit Card Fraud

1.0.1 About Dataset

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

1.0.2 Dataset at a Glance

```
In [1]: %matplotlib inline

        import itertools
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.gridspec as gridspec
        from sklearn.preprocessing import StandardScaler
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import cross_validate
        from sklearn.metrics import precision_recall_curve
        from sklearn.metrics import classification_report
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import average_precision_score
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestClassifier
        from imblearn.over_sampling import SMOTE
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Dropout

        pd.set_option("display.max_columns", 31)

        transactions = pd.read_csv("creditcard.csv")
        transactions.describe()
```

```
/home/firabby/pyML/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion o
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

```
Out[1]:              Time            V1            V2            V3            V4  \
        count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
        mean    94813.859575  1.165980e-15  3.416908e-16 -1.373150e-15  2.086869e-15
        std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
        min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
        25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
        50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
        75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
        max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                         V5            V6            V7            V8            V9  \
        count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
        mean   9.604066e-16  1.490107e-15 -5.556467e-16  1.177556e-16 -2.406455e-15
        std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
        min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
        25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
        50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
        75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
        max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

                        V10           V11           V12           V13           V14  \
        count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
        mean   2.239751e-15  1.673327e-15 -1.254995e-15  8.176030e-16  1.206296e-15
        std    1.088850e+00  1.020713e+00  9.992014e-01  9.952742e-01  9.585956e-01
        min   -2.458826e+01 -4.797473e+00 -1.868371e+01 -5.791881e+00 -1.921433e+01
        25%   -5.354257e-01 -7.624942e-01 -4.055715e-01 -6.485393e-01 -4.255740e-01
        50%   -9.291738e-02 -3.275735e-02  1.400326e-01 -1.356806e-02  5.060132e-02
        75%    4.539234e-01  7.395934e-01  6.182380e-01  6.625050e-01  4.931498e-01
        max    2.374514e+01  1.201891e+01  7.848392e+00  7.126883e+00  1.052677e+01

                        V15           V16           V17           V18           V19  \
        count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
        mean   4.913003e-15  1.437666e-15 -3.800113e-16  9.572133e-16  1.039817e-15
        std    9.153160e-01  8.762529e-01  8.493371e-01  8.381762e-01  8.140405e-01
        min   -4.498945e+00 -1.412985e+01 -2.516280e+01 -9.498746e+00 -7.213527e+00
        25%   -5.828843e-01 -4.680368e-01 -4.837483e-01 -4.988498e-01 -4.562989e-01
        50%    4.807155e-02  6.641332e-02 -6.567575e-02 -3.636312e-03  3.734823e-03
        75%    6.488208e-01  5.232963e-01  3.996750e-01  5.008067e-01  4.589494e-01
        max    8.877742e+00  1.731511e+01  9.253526e+00  5.041069e+00  5.591971e+00

                        V20           V21           V22           V23           V24  \
        count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
        mean   6.406703e-16  1.656562e-16 -3.444850e-16  2.578648e-16  4.471968e-15
```

```
std    7.709250e-01  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min   -5.449772e+01 -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%   -2.117214e-01 -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%   -6.248109e-02 -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    1.330408e-01  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    3.942090e+01  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.340915e-16  1.687098e-15 -3.666453e-16 -1.220404e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
```

In [2]:
```
transactions.head()
```

Out[2]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9       V10       V11       V12       V13       V14  \
0  0.098698  0.363787  0.090794 -0.551600 -0.617801 -0.991390 -0.311169
1  0.085102 -0.255425 -0.166974  1.612727  1.065235  0.489095 -0.143772
2  0.247676 -1.514654  0.207643  0.624501  0.066084  0.717293 -0.165946
3  0.377436 -1.387024 -0.054952 -0.226487  0.178228  0.507757 -0.287924
4 -0.270533  0.817739  0.753074 -0.822843  0.538196  1.345852 -1.119670

        V15       V16       V17       V18       V19       V20       V21  \
0  1.468177 -0.470401  0.207971  0.025791  0.403993  0.251412 -0.018307
1  0.635558  0.463917 -0.114805 -0.183361 -0.145783 -0.069083 -0.225775
2  2.345865 -2.890083  1.109969 -0.121359 -2.261857  0.524980  0.247998
3 -0.631418 -1.059647 -0.684093  1.965775 -1.232622 -0.208038 -0.108300
```

```
4  0.175121 -0.451449 -0.237033 -0.038195  0.803487  0.408542 -0.009431

        V22       V23       V24       V25       V26       V27       V28  \
0  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153

   Amount  Class
0  149.62      0
1    2.69      0
2  378.66      0
3  123.50      0
4   69.99      0
```

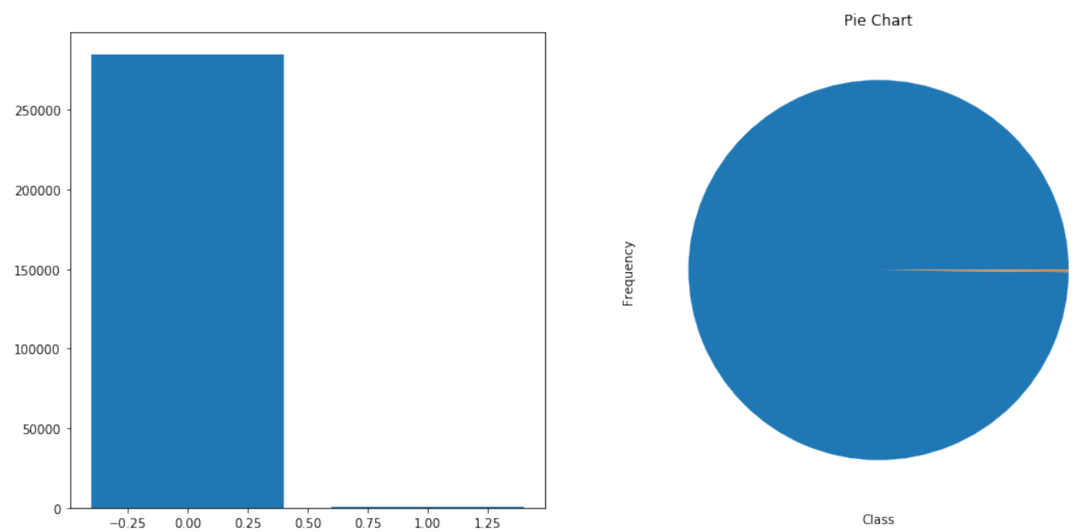
In [3]:
```
# check if there is any missing value
transactions.isnull().sum()
```

Out[3]:
```
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
```

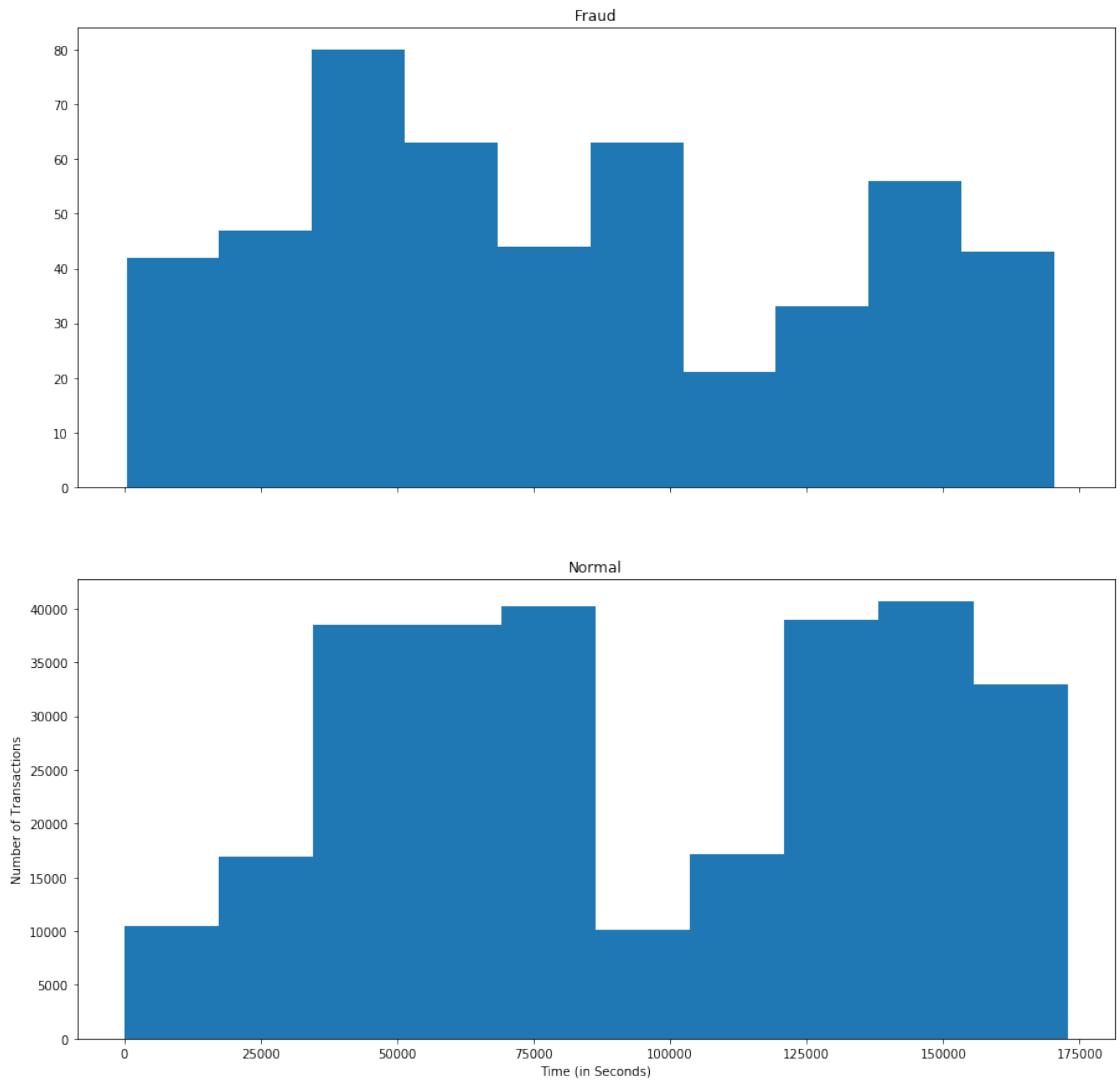
In [4]:
```
transactions_count = pd.value_counts(transactions.Class)
figure, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 7))
ax1.bar(transactions_count.index, transactions_count.values)
plt.title("Bar Chart")
plt.xlabel("Class")
plt.ylabel("Frequency")

ax2.pie(transactions_count)
plt.title("Pie Chart")
```

Out[4]:
```
Text(0.5,1,'Pie Chart')
```



From these charts it is clear that the data is very imbalanced.
Let's see how time compares to both fraud and normal transactions.

In [5]:
```
figure, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15, 15))

ax1.hist(transactions.Time[transactions.Class == 1])
ax1.set_title('Fraud')

ax2.hist(transactions.Time[transactions.Class == 0])
ax2.set_title('Normal')

plt.xlabel('Time (in Seconds)')
plt.ylabel('Number of Transactions')
plt.show()
```

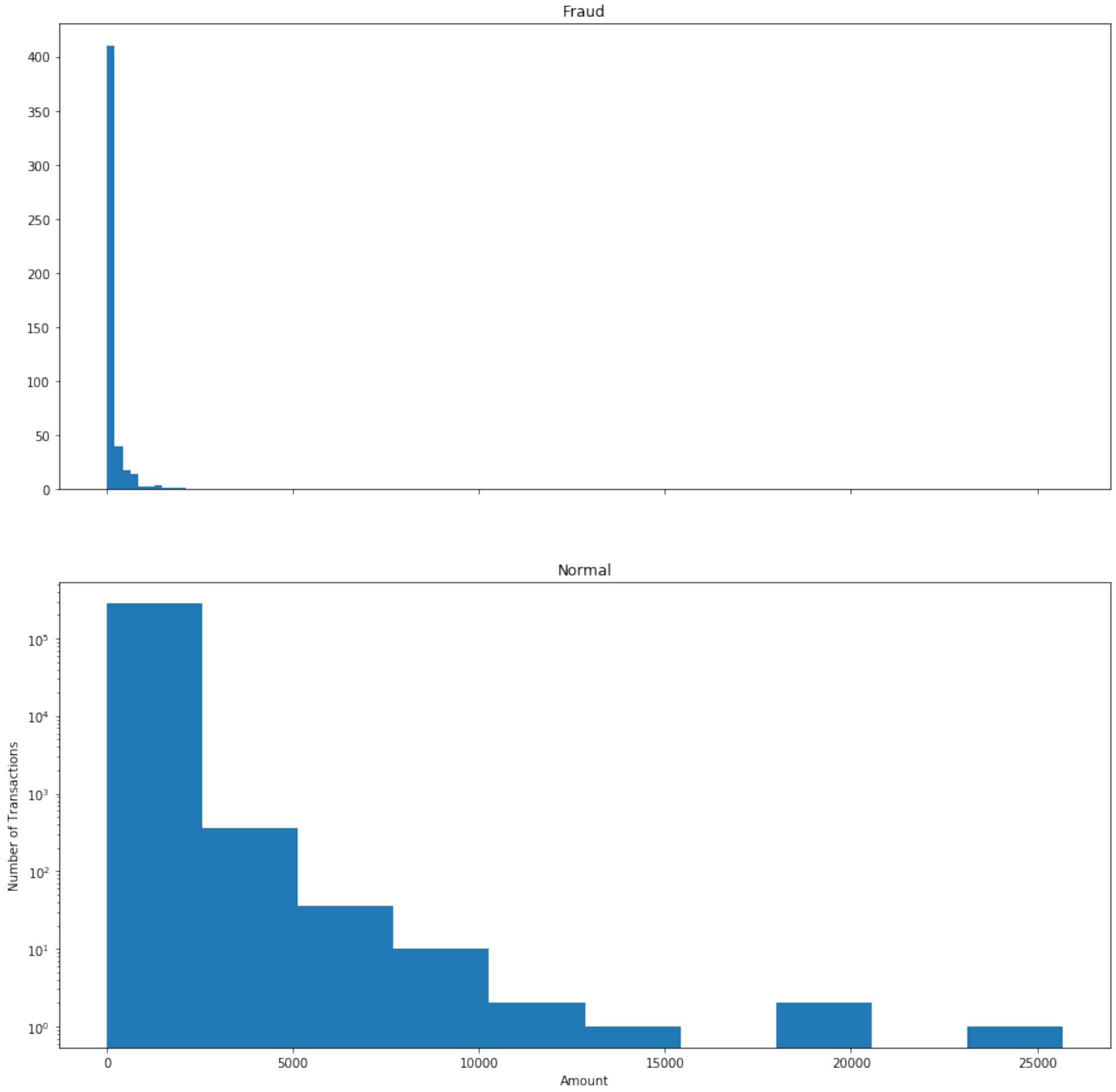


We can see that fraud transaction were still being made a little after the peak hour. This information can help us detect fraud transaction during off-peak hour.

Let us see if we can find any relation in amount for fraud and normal transaction.

```
In [6]: figure, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(15, 15))

        ax1.hist(transactions.Amount[transactions.Class == 1])
        ax1.set_title('Fraud')

        ax2.hist(transactions.Amount[transactions.Class == 0])
        ax2.set_title('Normal')

        plt.xlabel('Amount')
        plt.ylabel('Number of Transactions')
        plt.yscale('log')
```

```
plt.show()
```



We can see that fraud transactions were made in smaller amount.

1.0.3 Normalize the Data

Only the amounts are not normalized. So, we need to normalize the amounts.

```
In [7]: transactions["normalizedAmount"] = StandardScaler().fit_transform(transactions.Amount.v
```

Drop time and amount column, as we no longer need them.

```
In [8]: transactions = transactions.drop(["Amount", "Time"], axis=1)
```

Let's look at our new dataset.

In [9]: transactions.describe()

Out[9]:
```
                 V1            V2            V3            V4            V5  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.165980e-15  3.416908e-16 -1.373150e-15  2.086869e-15  9.604066e-16
std    1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00  1.380247e+00
min   -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00 -1.137433e+02
25%   -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01 -6.915971e-01
50%    1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02 -5.433583e-02
75%    1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01  6.119264e-01
max    2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01  3.480167e+01

                 V6            V7            V8            V9           V10  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.490107e-15 -5.556467e-16  1.177556e-16 -2.406455e-15  2.239751e-15
std    1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00  1.088850e+00
min   -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01 -2.458826e+01
25%   -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01 -5.354257e-01
50%   -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02 -9.291738e-02
75%    3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01  4.539234e-01
max    7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01  2.374514e+01

                V11           V12           V13           V14           V15  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.673327e-15 -1.254995e-15  8.176030e-16  1.206296e-15  4.913003e-15
std    1.020713e+00  9.992014e-01  9.952742e-01  9.585956e-01  9.153160e-01
min   -4.797473e+00 -1.868371e+01 -5.791881e+00 -1.921433e+01 -4.498945e+00
25%   -7.624942e-01 -4.055715e-01 -6.485393e-01 -4.255740e-01 -5.828843e-01
50%   -3.275735e-02  1.400326e-01 -1.356806e-02  5.060132e-02  4.807155e-02
75%    7.395934e-01  6.182380e-01  6.625050e-01  4.931498e-01  6.488208e-01
max    1.201891e+01  7.848392e+00  7.126883e+00  1.052677e+01  8.877742e+00

                V16           V17           V18           V19           V20  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.437666e-15 -3.800113e-16  9.572133e-16  1.039817e-15  6.406703e-16
std    8.762529e-01  8.493371e-01  8.381762e-01  8.140405e-01  7.709250e-01
min   -1.412985e+01 -2.516280e+01 -9.498746e+00 -7.213527e+00 -5.449772e+01
25%   -4.680368e-01 -4.837483e-01 -4.988498e-01 -4.562989e-01 -2.117214e-01
50%    6.641332e-02 -6.567575e-02 -3.636312e-03  3.734823e-03 -6.248109e-02
75%    5.232963e-01  3.996750e-01  5.008067e-01  4.589494e-01  1.330408e-01
max    1.731511e+01  9.253526e+00  5.041069e+00  5.591971e+00  3.942090e+01

                V21           V22           V23           V24           V25  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   1.656562e-16 -3.444850e-16  2.578648e-16  4.471968e-15  5.340915e-16
std    7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01  5.212781e-01
min   -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00 -1.029540e+01
25%   -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01 -3.171451e-01
```

```
50%   -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02  1.659350e-02
75%    1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01  3.507156e-01
max    2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00  7.519589e+00

                V26           V27           V28          Class  \
count  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.687098e-15 -3.666453e-16 -1.220404e-16       0.001727
std    4.822270e-01  4.036325e-01  3.300833e-01       0.041527
min   -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.269839e-01 -7.083953e-02 -5.295979e-02       0.000000
50%   -5.213911e-02  1.342146e-03  1.124383e-02       0.000000
75%    2.409522e-01  9.104512e-02  7.827995e-02       0.000000
max    3.517346e+00  3.161220e+01  3.384781e+01       1.000000

       normalizedAmount
count      2.848070e+05
mean       2.913952e-17
std        1.000002e+00
min       -3.532294e-01
25%       -3.308401e-01
50%       -2.652715e-01
75%       -4.471707e-02
max        1.023622e+02
```

In [10]:
```
transactions.head()
```

Out[10]:
```
         V1        V2        V3        V4        V5        V6        V7  \
0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9       V10       V11       V12       V13       V14  \
0  0.098698  0.363787  0.090794 -0.551600 -0.617801 -0.991390 -0.311169
1  0.085102 -0.255425 -0.166974  1.612727  1.065235  0.489095 -0.143772
2  0.247676 -1.514654  0.207643  0.624501  0.066084  0.717293 -0.165946
3  0.377436 -1.387024 -0.054952 -0.226487  0.178228  0.507757 -0.287924
4 -0.270533  0.817739  0.753074 -0.822843  0.538196  1.345852 -1.119670

        V15       V16       V17       V18       V19       V20       V21  \
0  1.468177 -0.470401  0.207971  0.025791  0.403993  0.251412 -0.018307
1  0.635558  0.463917 -0.114805 -0.183361 -0.145783 -0.069083 -0.225775
2  2.345865 -2.890083  1.109969 -0.121359 -2.261857  0.524980  0.247998
3 -0.631418 -1.059647 -0.684093  1.965775 -1.232622 -0.208038 -0.108300
4  0.175121 -0.451449 -0.237033 -0.038195  0.803487  0.408542 -0.009431

        V22       V23       V24       V25       V26       V27       V28  \
```

```
0  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053
1 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724
2  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752
3  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458
4  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153

   Class  normalizedAmount
0      0          0.244964
1      0         -0.342475
2      0          1.160686
3      0          0.140534
4      0         -0.073403
```

1.0.4 Splitting Data for Train and Test Set with and without Resampling

Data without resampling

```
In [11]: X = transactions.iloc[:, transactions.columns != 'Class'].values
         y = transactions.iloc[:, transactions.columns == 'Class'].values
```

Undersampling Normal Transactions

```
In [12]: fraud_transactions_count = len(transactions[transactions.Class == 1])

         fraud_indices = np.array(transactions[transactions.Class == 1].index)
         normal_indices = transactions[transactions.Class == 0].index

         random_normal_indices = np.random.choice(normal_indices, fraud_transactions_count, rep


         undersampled_indices = np.concatenate([fraud_indices,random_normal_indices])

         undersampled_transactions = transactions.iloc[undersampled_indices,:]

         X_undersampled = undersampled_transactions.iloc[:, undersampled_transactions.columns
         y_undersampled = undersampled_transactions.iloc[:, undersampled_transactions.columns =


         print("Normal transactions: ", len(undersampled_transactions[undersampled_transactions
         print("Fraud transactions: ", len(undersampled_transactions[undersampled_transactions
         print("Total transactions in resampled data: ", len(undersampled_transactions))
```

```
Normal transactions:  0.5
Fraud transactions:  0.5
Total transactions in resampled data:  984
```

Oversampling using SMOTE

```
In [13]: sm = SMOTE(kind="regular")
```

```
In [14]: X_oversampled, y_oversampled = sm.fit_sample(X, y.ravel())
```

```
In [15]: oversampled_transactions = np.hstack((X_oversampled, y_oversampled.reshape(-1, 1)))
         print("Normal transactions: ", len(oversampled_transactions[oversampled_transactions[
         print("Fraud transactions: ", len(oversampled_transactions[oversampled_transactions[:
         print("Total transactions in resampled data: ", len(oversampled_transactions))
```

```
Normal transactions:  0.5
Fraud transactions:  0.5
Total transactions in resampled data:  568630
```

1.0.5 Splitting into Train and Test Data

```
In [16]: X_train, X_test, y_train, y_test = train_test_split(X, y.ravel(), test_size=0.2, rand
         X_undersampled_train, X_undersampled_test, y_undersampled_train, y_undersampled_test =
         X_oversampled_train, X_oversampled_test, y_oversampled_train, y_oversampled_test = tra
```

```
In [17]: def get_cv_scores(clf, x, y):
             scoring = ['precision_micro', 'recall_micro', 'precision_macro', 'recall_macro']
             scores = cross_validate(clf, x, y, scoring=scoring, cv=5, return_train_score=False
             scores_mean = {}
             print("Average Training Time: ", scores["fit_time"].mean(), "(+/- %0.2f)" %(scores
             print("Average Training Time: ", scores["score_time"].mean(), "(+/- %0.2f)" %(sco
             print("Average Test Precision Macro: ", scores["test_precision_macro"].mean(), "(
             print("Average Test Precision Micro: ", scores["test_precision_micro"].mean(), "(
             print("Average Test Recall Macro: ", scores["test_recall_macro"].mean(), "(+/- %0
             print("Average Test Recall Micro: ", scores["test_recall_micro"].mean(), "(+/- %0


             scores_mean["test_precision_macro"] = scores["test_precision_macro"].mean()
             scores_mean["test_precision_micro"] = scores["test_precision_micro"].mean()
             scores_mean["test_recall_macro"] = scores["test_recall_macro"].mean()
             scores_mean["test_recall_micro"] = scores["test_recall_micro"].mean()

             return scores_mean

         clf = LogisticRegression(C=0.01, random_state=42)

         print("Original Data")
         print("=======================================================\n")
         org_lr_score = get_cv_scores(clf, X_train, y_train)
         print()

         print("Undersampled Data")
         print("=======================================================\n")
         und_lr_score = get_cv_scores(clf, X_undersampled_train, y_undersampled_train)
```

```
        print()

        print("Oversampled Data")
        print("=======================================================\n")
        ovr_lr_score = get_cv_scores(clf, X_oversampled_train, y_oversampled_train)
```

```
Original Data
=======================================================

Average Training Time:  1.173475170135498 (+/- 0.10)
Average Training Time:  0.04214701652526855 (+/- 0.01)
Average Test Precision Macro:  0.9458818378730246 (+/- 0.01)
Average Test Precision Micro:  0.9991573221523449 (+/- 0.00)
Average Test Recall Macro:  0.7919377991122797 (+/- 0.03)
Average Test Recall Micro:  0.9991573221523449 (+/- 0.00)

Undersampled Data
=======================================================

Average Training Time:  0.0049957275390625 (+/- 0.00)
Average Training Time:  0.002244901657104492 (+/- 0.00)
Average Test Precision Macro:  0.9192994880497036 (+/- 0.01)
Average Test Precision Micro:  0.9173814441088488 (+/- 0.01)
Average Test Recall Macro:  0.9173482635507952 (+/- 0.01)
Average Test Recall Micro:  0.9173814441088488 (+/- 0.01)

Oversampled Data
=======================================================

Average Training Time:  4.735983037948609 (+/- 0.22)
Average Training Time:  0.09955897331237792 (+/- 0.00)
Average Test Precision Macro:  0.9467523161299045 (+/- 0.00)
Average Test Precision Micro:  0.9450763222483586 (+/- 0.00)
Average Test Recall Macro:  0.9450611497209722 (+/- 0.00)
Average Test Recall Micro:  0.9450763222483586 (+/- 0.00)
```

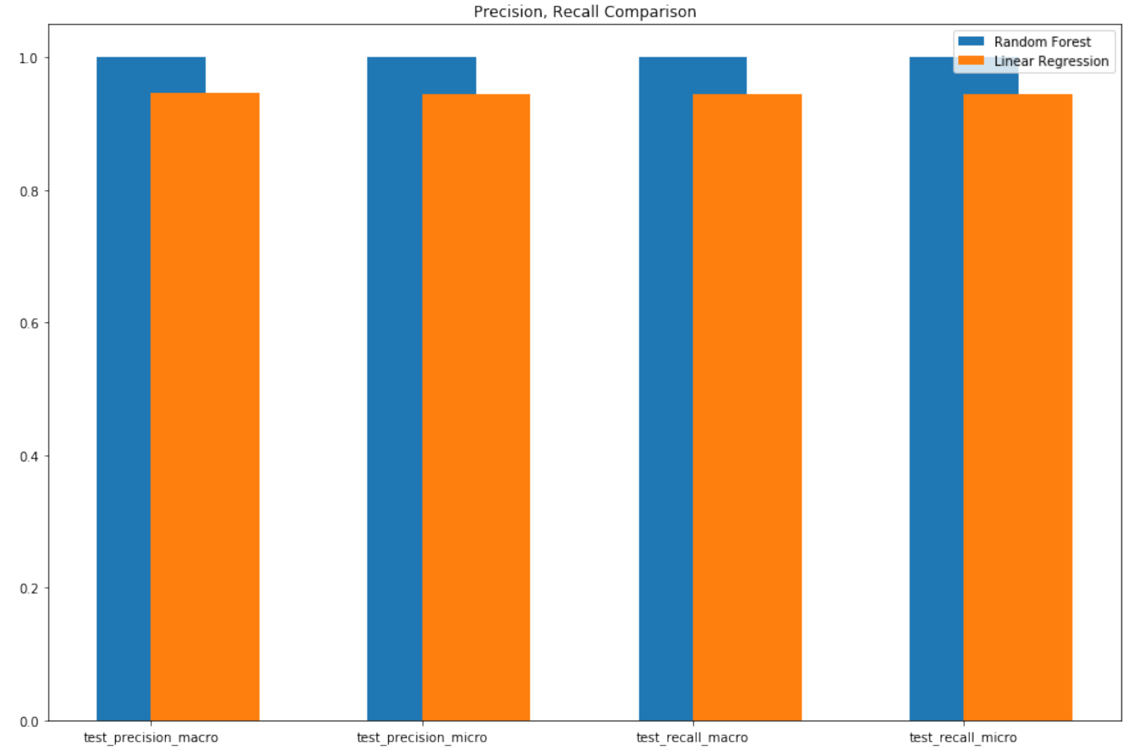
```
In [18]: clf = RandomForestClassifier(random_state=42)

         print("Original Data")
         print("=======================================================\n")
         org_rf_score = get_cv_scores(clf, X_train, y_train)
         print()

         print("Undersampled Data")
         print("=======================================================\n")
         und_rf_score = get_cv_scores(clf, X_undersampled_train, y_undersampled_train)
         print()
```

```
print("Oversampled Data")
print("=======================================================\n")
ovr_rf_score = get_cv_scores(clf, X_oversampled_train, y_oversampled_train)
```

```
Original Data
=======================================================

Average Training Time:  14.825970077514649 (+/- 1.42)
Average Training Time:  0.17570128440856933 (+/- 0.01)
Average Test Precision Macro:  0.9721289734137869 (+/- 0.02)
Average Test Precision Micro:  0.9994952711967147 (+/- 0.00)
Average Test Recall Macro:  0.8769646505148264 (+/- 0.03)
Average Test Recall Micro:  0.9994952711967147 (+/- 0.00)

Undersampled Data
=======================================================

Average Training Time:  0.01947441101074219 (+/- 0.00)
Average Training Time:  0.004582881927490234 (+/- 0.00)
Average Test Precision Macro:  0.927227247385666 (+/- 0.01)
Average Test Precision Micro:  0.9237429116611573 (+/- 0.01)
Average Test Recall Macro:  0.9236773774748459 (+/- 0.01)
Average Test Recall Micro:  0.9237429116611573 (+/- 0.01)

Oversampled Data
=======================================================

Average Training Time:  27.69242305755615 (+/- 0.86)
Average Training Time:  0.47842750549316404 (+/- 0.01)
Average Test Precision Macro:  0.9998175263441713 (+/- 0.00)
Average Test Precision Micro:  0.9998175438875245 (+/- 0.00)
Average Test Recall Macro:  0.9998175757121663 (+/- 0.00)
Average Test Recall Micro:  0.9998175438875245 (+/- 0.00)
```

Random forest takes longer time to train as data increases.

In [22]:
```
plt.figure(figsize=(15, 10))
plt.bar(range(len(ovr_rf_score)), list(ovr_rf_score.values()), align='center', width=(
plt.xticks(range(len(ovr_rf_score)), list(ovr_rf_score.keys()))

plt.bar(np.arange(len(ovr_lr_score))+0.2, list(ovr_lr_score.values()), align='center'
plt.xticks(range(len(ovr_lr_score)), list(ovr_lr_score.keys()))
plt.legend(["Random Forest", "Linear Regression"])
plt.title("Precision, Recall Comparison")
plt.show()
```

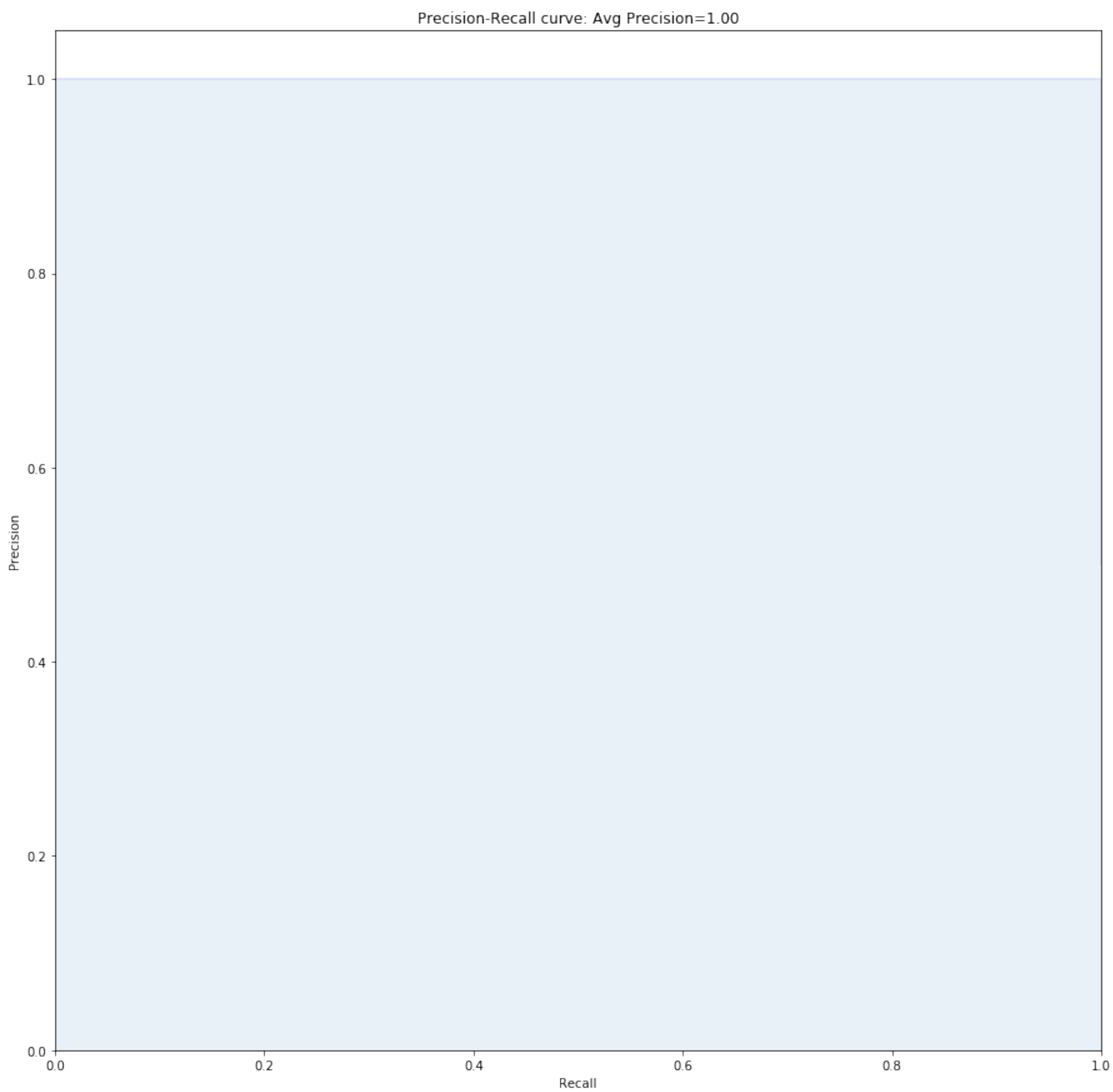


From the above experiments, we can see that oversampled data with Synthetic Minority Oversampling Technique, we get the best precision and recall using Random Forest Classifier. Let's train the classifier and evaluate the model.

In [20]:
```
clf.fit(X_oversampled_train, y_oversampled_train)
pred = clf.predict(X_oversampled_test)
class_names = ["Normal", "Fraud"]
report = classification_report(y_oversampled_test, pred, target_names = class_names)
print(report)
print("Accuracy Score: ", accuracy_score(y_oversampled_test, pred))

average_precision = average_precision_score(y_oversampled_test, pred)

precision, recall, _ = precision_recall_curve(y_oversampled_test, pred)

plt.figure(figsize=(15, 15))
plt.step(recall, precision, color='b', alpha=0.1, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.1)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve: Avg Precision={0:0.2f}'.format(
          average_precision))
```

```
             precision    recall  f1-score   support

     Normal       1.00      1.00      1.00     56750
      Fraud       1.00      1.00      1.00     56976

avg / total       1.00      1.00      1.00    113726

Accuracy Score:  0.9998329317834093
```

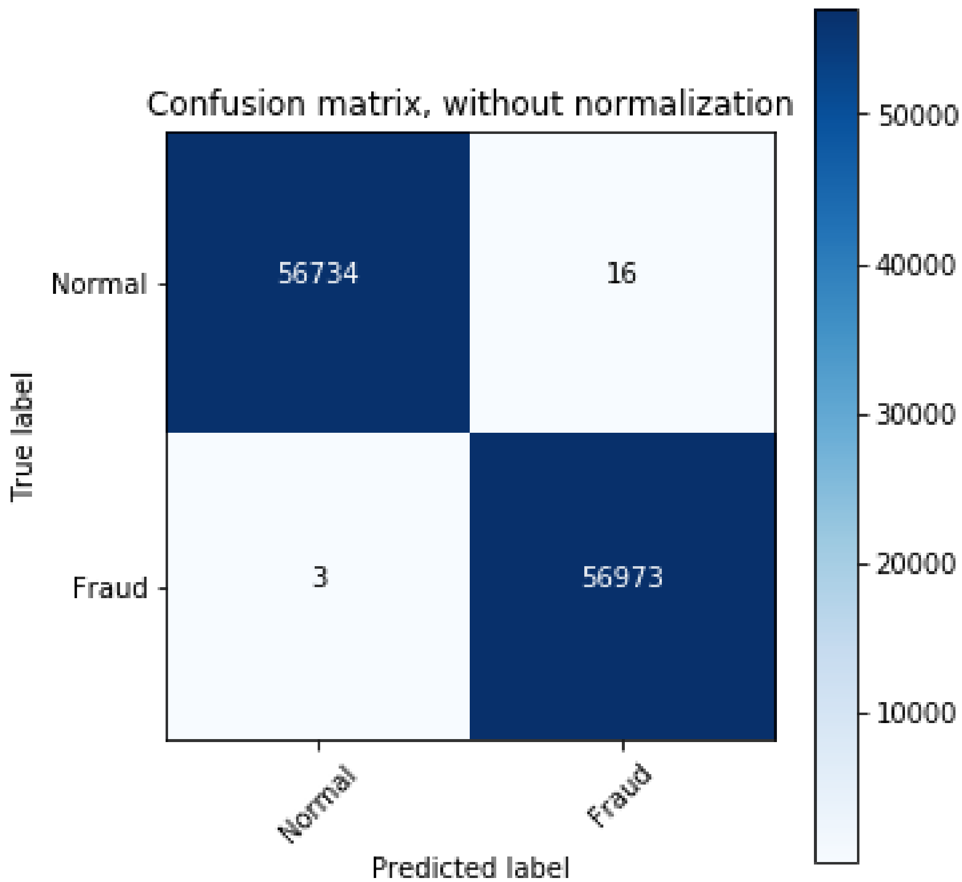
Out[20]: Text(0.5,1,'Precision-Recall curve: Avg Precision=1.00')

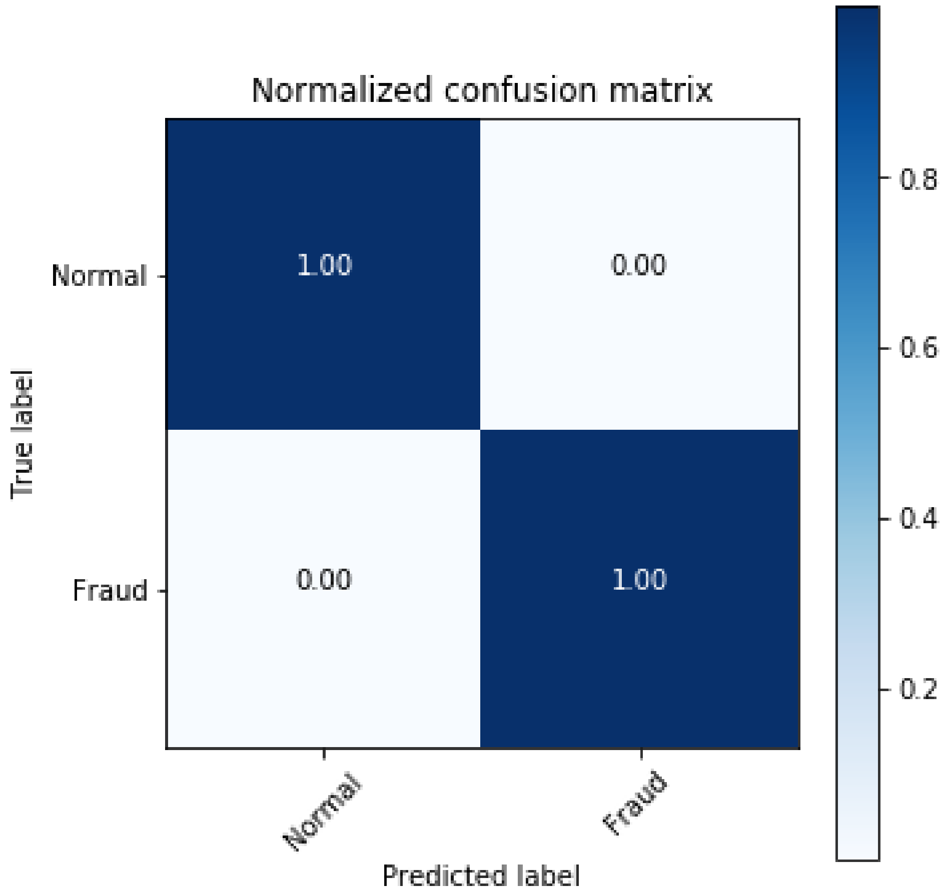



```
In [25]: def plot_confusion_matrix(cm, classes,
                                   normalize=False,
```

```
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


cnf_matrix = confusion_matrix(y_oversampled_test, pred)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure(figsize=(5, 5))
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')

# Plot normalized confusion matrix
plt.figure(figsize=(5, 5))
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```

```
Confusion matrix, without normalization
[[56734    16]
 [    3 56973]]
Normalized confusion matrix
[[1.00e+00 2.82e-04]
 [5.27e-05 1.00e+00]]
```




Normalized confusion matrix
Normal
Fraud
True label
1.00
0.00
0.00
1.00
Normal
Fraud
Predicted label
0.8
0.6
0.4
0.2