In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries instal
led
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/d
ocker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will lis
t all files under the input directory



# You can write up to 20GB to the current directory (/kaggle/working/) that ge
ts preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
outside of the current session
```

In [2]:
```
# load the already done pca on the dataset
data = pd.read_csv(r"C:\Users\Piyush\Desktop\Data Science\Projects\Credit Card
- Anomaly\creditcard.csv")
data.head()
```

Out[2]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [3]:
```
data.shape
```

Out[3]: (284807, 31)

In [4]:
```
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score, confusion_matrix,roc_auc_score, precision_recall_curve, roc_curve, auc, average_precision_score,plot_roc_curve
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

Exploratory Data Analysis

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [6]:
```
# checking for NAN values
data.isnull().values.any()
```

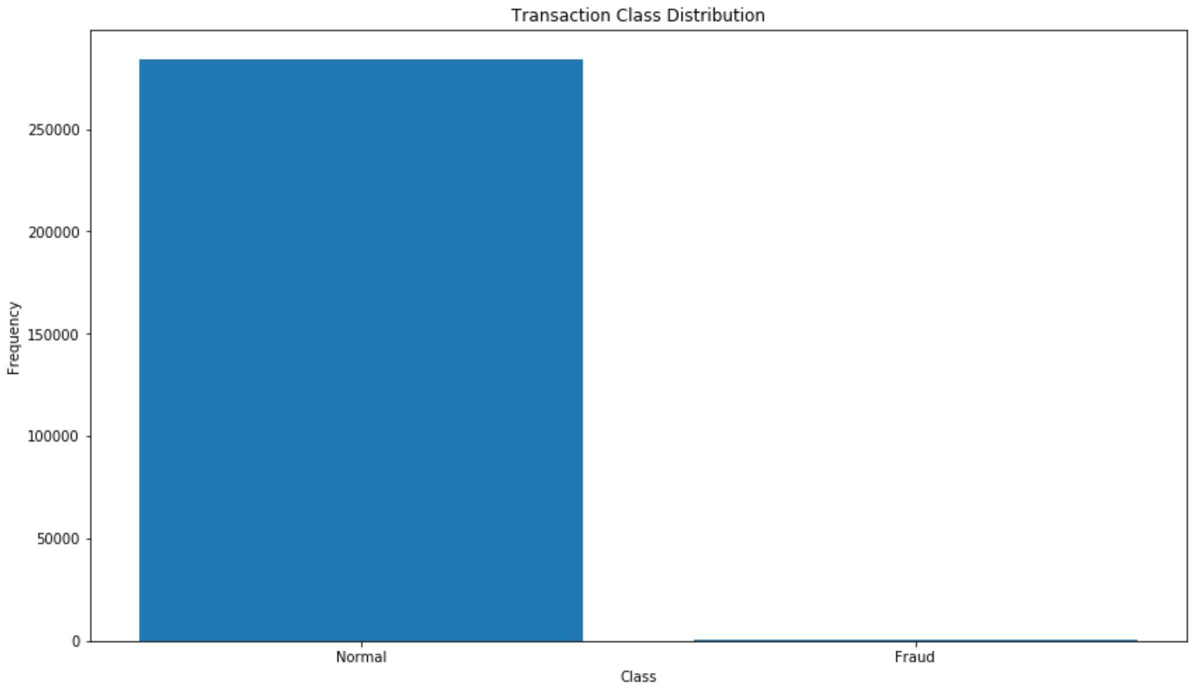
Out[6]:
```
False
```

In [7]:
```
# plot bar graph showing the distribution of anmomolous to non-anomolous trans
actions
count_classes = pd.value_counts(data['Class'], sort = True)
count_classes
# got  - 492 fraud trnasactions
```

Out[7]:
```
0    284315
1       492
Name: Class, dtype: int64
```

In [8]:
```
names = ["Normal","Fraud"]
values = count_classes
plt.title("Transaction Class Distribution")
plt.xlabel("Class")
plt.ylabel("Frequency")
plt.bar(names,values)
```

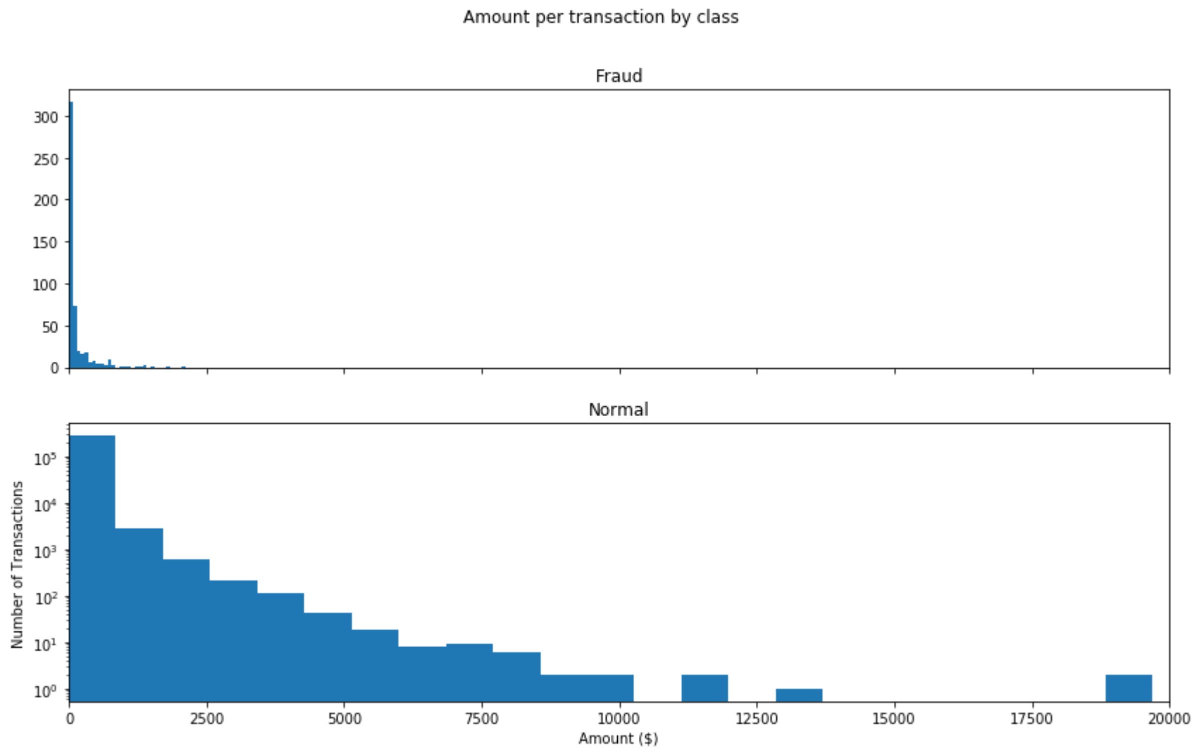
Out[8]: <BarContainer object of 2 artists>



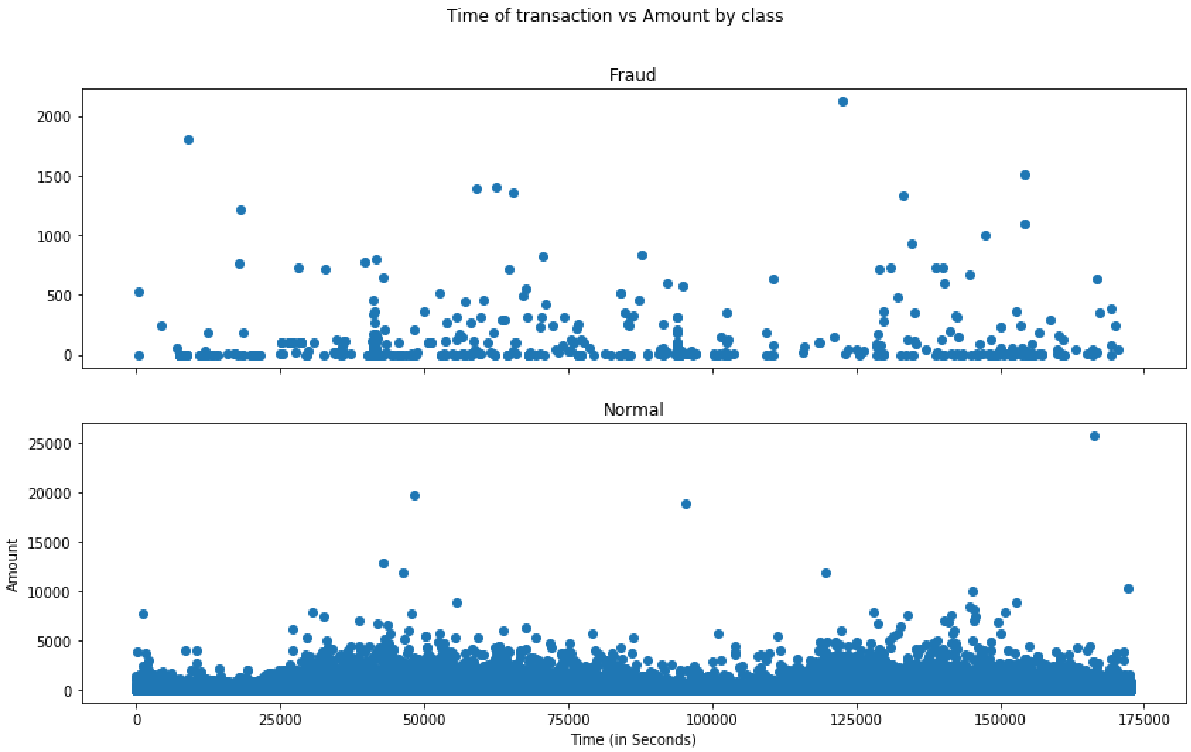
In [9]:
```
# separating the fraud and normal data
normal = data[data['Class']==0]
fraud = data[data['Class']==1]
print(normal.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

In [10]:
```
# Amount per transaction distrbution
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 30
ax1.hist(fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```

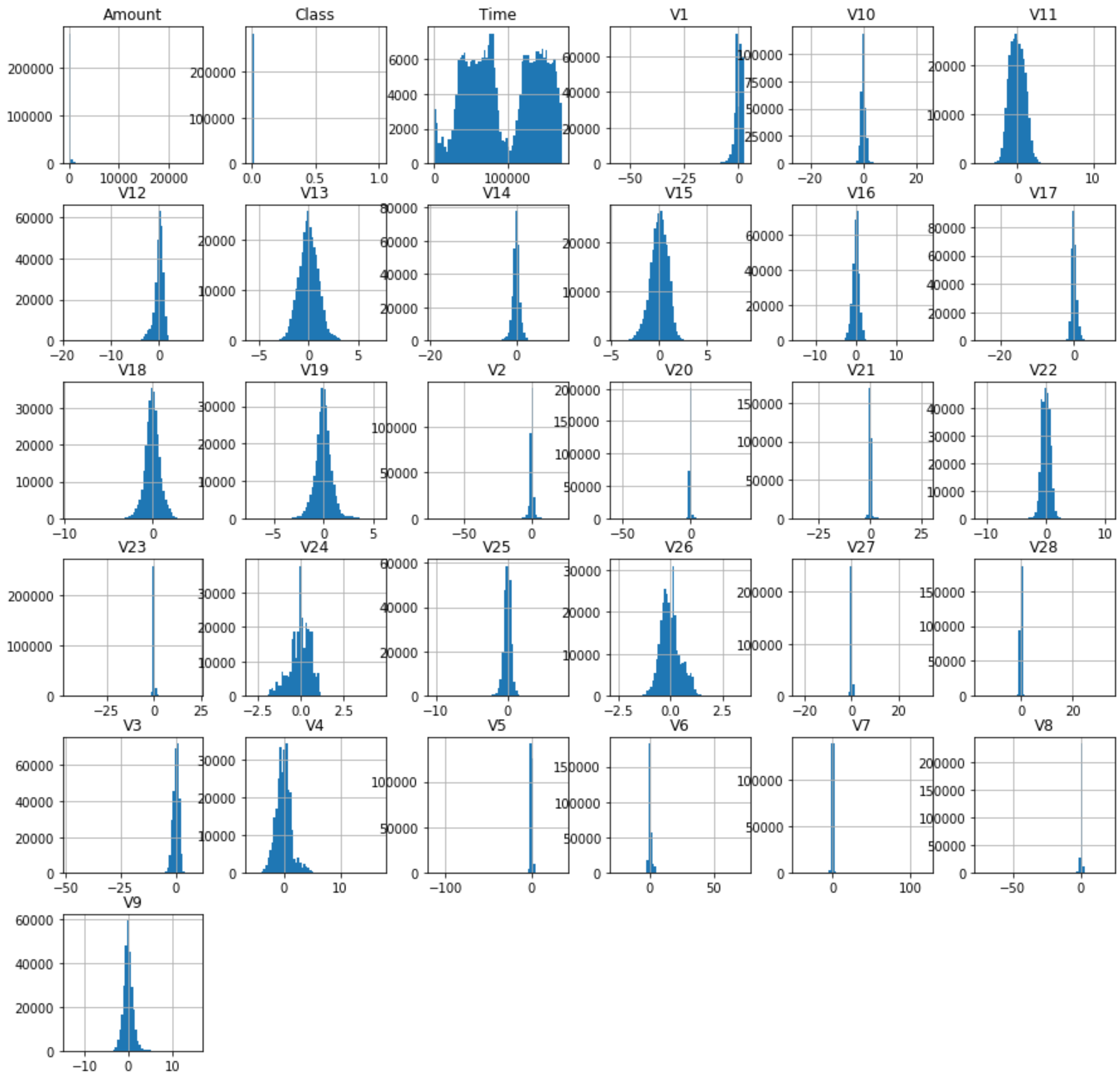


In [11]:
```
# time of trnasaction vs amount by class
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(fraud.Time, fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(normal.Time, normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```



From the second plot, we can observe that fraudulent transactions occur at the same time as normal transaction, making time an irrelevant factor. From the first plot, we can see that most of the fraudulent transactions are small amount transactions. This is however not a huge differentiating feature since majority of normal transactions are also small amount transactions.

In [12]:
```
data.hist(figsize=(15,15), bins = 64)
plt.show()
```



In [13]:
```
#data.drop(['Time', 'V1', 'V24'], axis=1, inplace=True)
data.drop(['Time', 'V24'], axis=1, inplace=True)
```

In [14]:
```
# lets reduce our dataset to say 30% as it is a huge dataset with more than 28
4k+ objects
df= data.sample(frac = 0.2,random_state=1)
df.shape
```

Out[14]: (56961, 29)

In [15]:
```
data.shape
# you see the difference , original data had 284k examples while the reduced h
ave 85k
```

Out[15]: (284807, 29)

In [16]:
```
# now lets see the distribution again of normal vs fraud transaction
Fraud = df[df['Class']==1]
Normal = df[df['Class']==0]
print(Fraud.shape,Normal.shape)
# you see about 135 fraud cases now
```

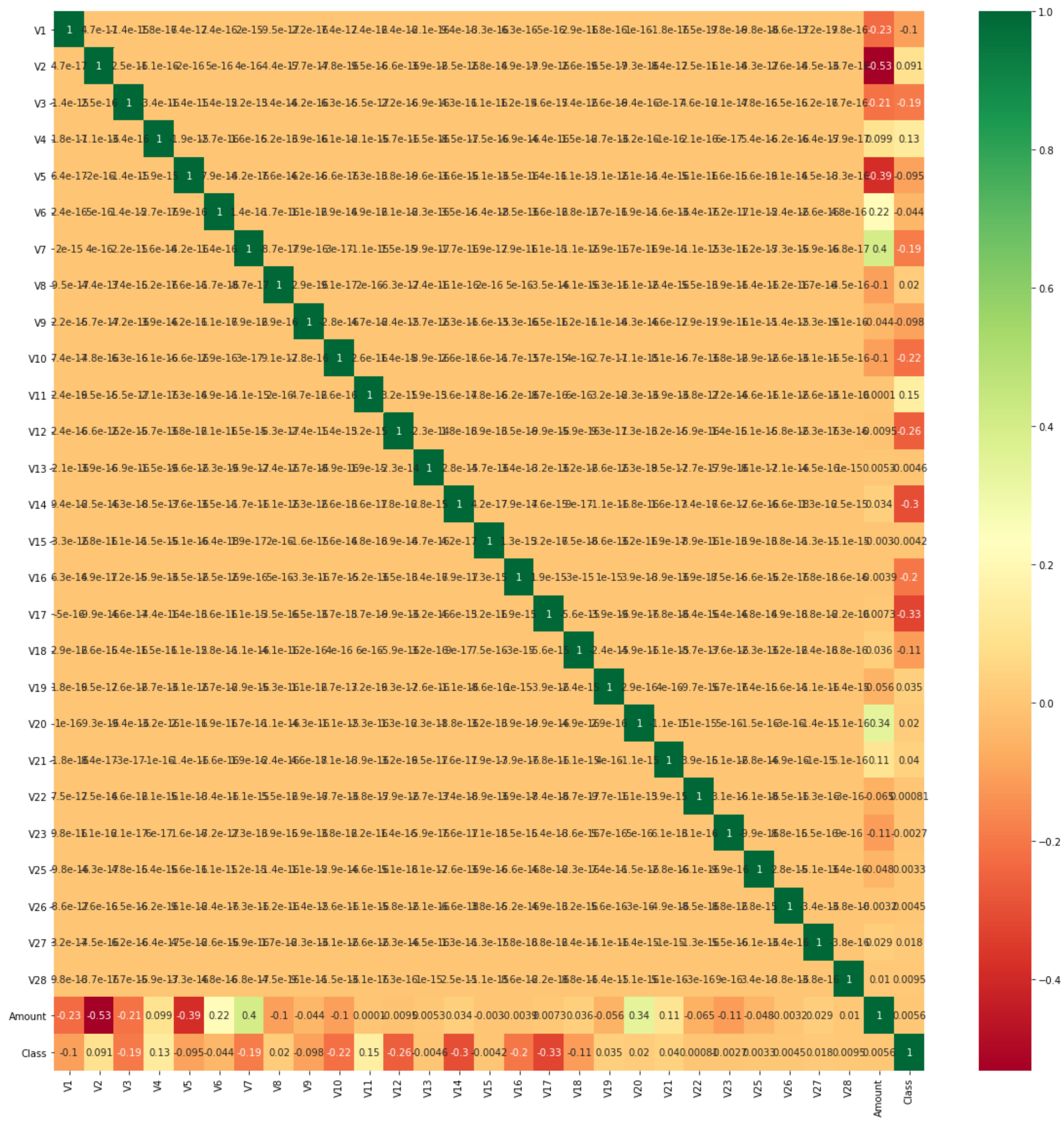
```
(87, 29) (56874, 29)
```

We have just 0.16% fraudulent transactions in the dataset. This means that a random guess by the model should yield 0.16% accuracy for fraudulent transactions

In [17]:
```
outlier_fraction = len(Fraud)/float(len(Normal))
outlier_fraction
```

Out[17]:
```
0.0015296972254457222
```

In [18]:
```
#Correlation using heatmap
import seaborn as sns
#get correlations of each features in dataset
corrmat =df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



Building Models and Model Prediction

In [19]:
```
#Create independent and Dependent Features
columns = df.columns.tolist()  # all columns
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]   # removing "Class" from
our columns list
# Store the variable we are predicting
target = "Class"
# Define a random state
state = np.random.RandomState(42)
X = df[columns]
Y = df[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(56961, 28)
(56961,)
```

In [20]:
```
# Train_test Split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, rand
om_state=42)
```

In [21]:
```
print(y_test)
```

```
150340    0
130939    0
150066    0
284285    0
196462    0
         ..
196504    0
39304     0
183807    0
37174     0
138361    0
Name: Class, Length: 18798, dtype: int64
```

Isolation Forest Algorithm :

One of the newest techniques to detect anomalies is called Isolation Forests. The algorithm is based on the fact that anomalies are data points that are few and different. As a result of these properties, anomalies are susceptible to a mechanism called isolation.

This method is highly useful and is fundamentally different from all existing methods. It introduces the use of isolation as a more effective and efficient means to detect anomalies than the commonly used basic distance and density measures. Moreover, this method is an algorithm with a low linear time complexity and a small memory requirement. It builds a good performing model with a small number of trees using small sub-samples of fixed size, regardless of the size of a data set.

Typical machine learning methods tend to work better when the patterns they try to learn are balanced, meaning the same amount of good and bad behaviors are present in the dataset.

How Isolation Forests Work The Isolation Forest algorithm isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. The logic argument goes: isolating anomaly observations is easier because only a few conditions are needed to separate those cases from the normal observations. On the other hand, isolating normal observations require more conditions. Therefore, an anomaly score can be calculated as the number of conditions required to separate a given observation.

The way that the algorithm constructs the separation is by first creating isolation trees, or random decision trees. Then, the score is calculated as the path length to isolate the observation.

Local Outlier Factor(LOF) Algorithm

The LOF algorithm is an unsupervised outlier detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outlier samples that have a substantially lower density than their neighbors.

The number of neighbors considered, (parameter n_neighbors) is typically chosen 1) greater than the minimum number of objects a cluster has to contain, so that other objects can be local outliers relative to this cluster, and 2) smaller than the maximum number of close by objects that can potentially be local outliers. In practice, such informations are generally not available, and taking n_neighbors=20 appears to work well in general.

In [22]:
```
n_outliers = len(Fraud)
n_outliers
```

Out[22]:
```
87
```

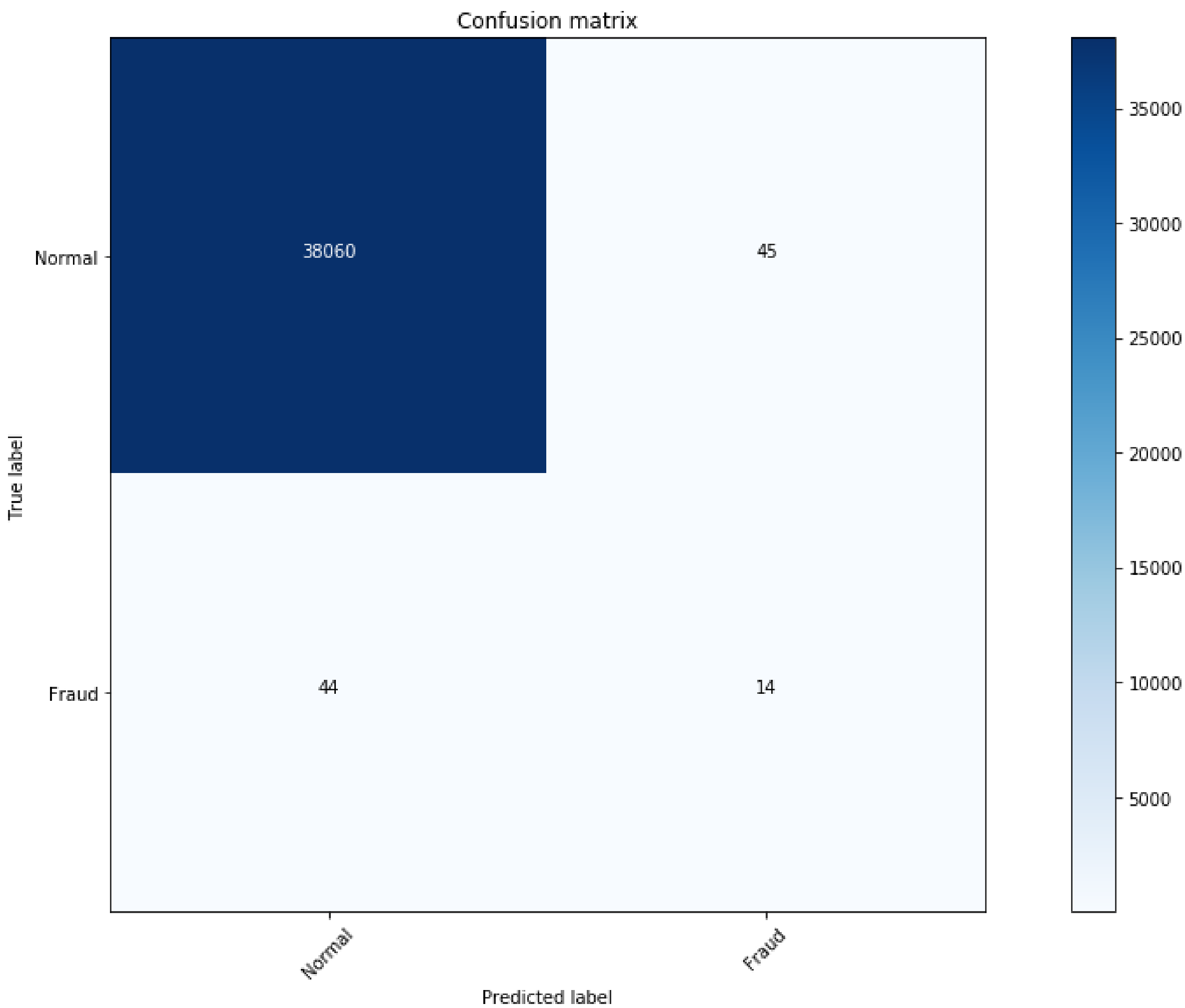
In [23]:
```
#plotting roc curve
def plot_roc(y_test,preds):
    fpr, tpr, threshold = roc_curve(y_test, preds)
    roc_auc = auc(fpr, tpr)

    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()
```

Local Outlier factor

In [24]:
```
clf = LocalOutlierFactor(n_neighbors=20, algorithm='auto', leaf_size=30, metri
c='minkowski',p=2, metric_params=None, contamination=outlier_fraction)
y_train_pred = clf.fit_predict(X_train)
#print(y_pred)
scores_prediction = clf.negative_outlier_factor_
y_train_pred[y_train_pred == 1] = 0
y_train_pred[y_train_pred == -1] = 1

# on test data
y_test_pred = clf.fit_predict(X_test)
y_test_pred[y_test_pred == 1] = 0
y_test_pred[y_test_pred == -1] = 1
```

In [25]:
```
import itertools
classes = np.array(['0','1'])
def plot_confusion_matrix(cm, classes,title='Confusion matrix', cmap=plt.cm.Bl
ues):
    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 = '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')
```

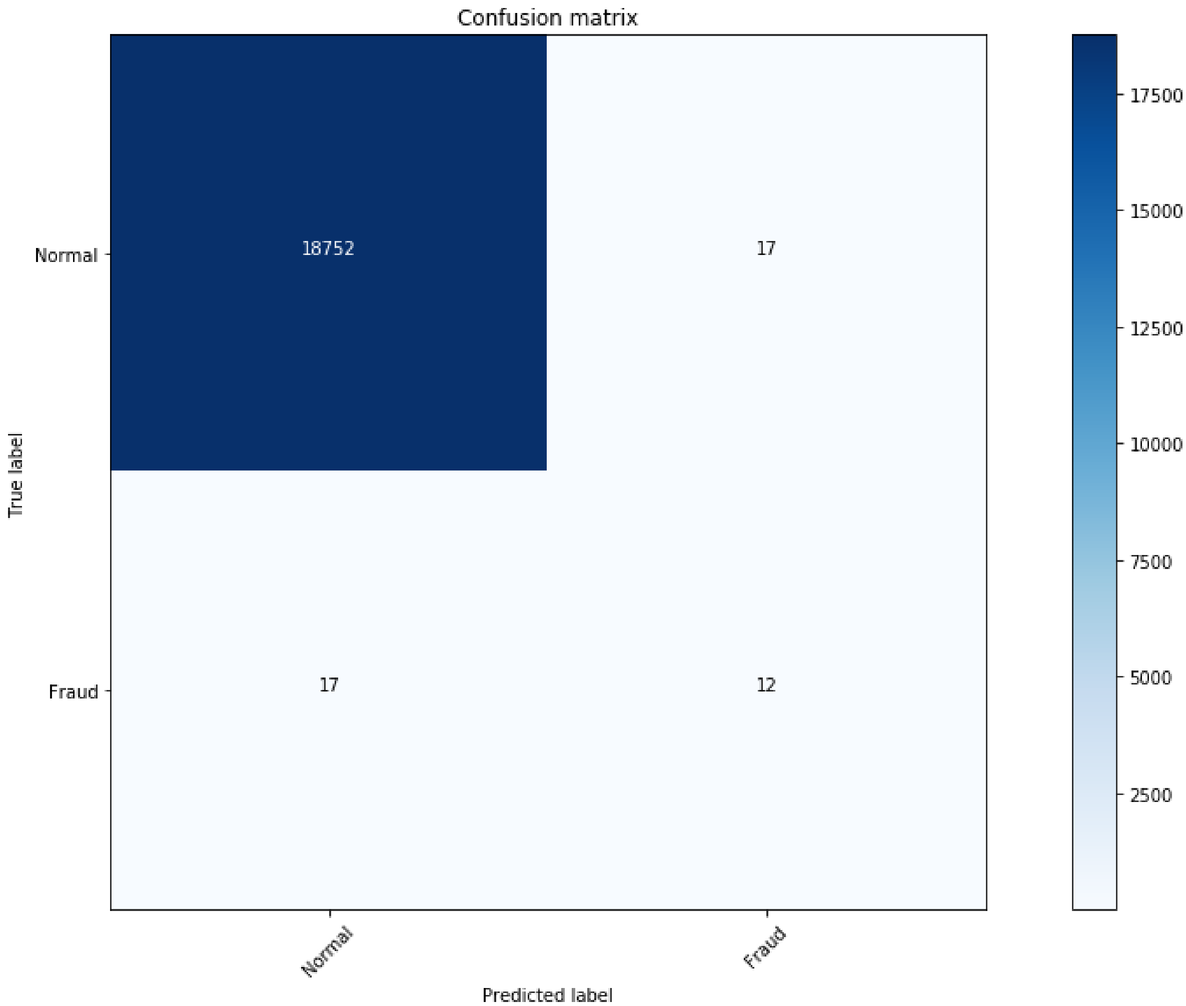
In [26]:
```
cm_train = confusion_matrix(y_train, y_train_pred)
plot_confusion_matrix(cm_train,["Normal", "Fraud"])
```



In [27]:
```
print('Total fraudulent transactions detected in training set: ' + str(cm_train[1][1]) + ' / ' + str(cm_train[1][1]+cm_train[1][0]))
print('Total non-fraudulent transactions detected in training set: ' + str(cm_train[0][0]) + ' / ' + str(cm_train[0][1]+cm_train[0][0]))

print('Probability to detect a fraudulent transaction in the training set: ' + str(cm_train[1][1]/(cm_train[1][1]+cm_train[1][0])))
print('Probability to detect a non-fraudulent transaction in the training set: ' + str(cm_train[0][0]/(cm_train[0][1]+cm_train[0][0])))

print("Accuracy of unsupervised anomaly detection model on the training set: "+str(100*(cm_train[0][0]+cm_train[1][1]) / (sum(cm_train[0]) + sum(cm_train[1]))) + "%")
```

```
Total fraudulent transactions detected in training set: 14 / 58
Total non-fraudulent transactions detected in training set: 38060 / 38105
Probability to detect a fraudulent transaction in the training set: 0.2413793103448276
Probability to detect a non-fraudulent transaction in the training set: 0.9988190526177667
Accuracy of unsupervised anomaly detection model on the training set: 99.76678982260304%
```

In [28]:
```
cm_test = confusion_matrix(y_test,y_test_pred)
plot_confusion_matrix(cm_test,["Normal", "Fraud"])
```



In [29]:
```
print('Total fraudulent transactions detected in test set: ' + str(cm_test[1][
1]) + ' / ' + str(cm_test[1][1]+cm_test[1][0]))
print('Total non-fraudulent transactions detected in test set: ' + str(cm_test
[0][0]) + ' / ' + str(cm_test[0][1]+cm_test[0][0]))

print('Probability to detect a fraudulent transaction in the test set: ' + str
(cm_test[1][1]/(cm_test[1][1]+cm_test[1][0])))
print('Probability to detect a non-fraudulent transaction in the test set: ' +
str(cm_test[0][0]/(cm_test[0][1]+cm_test[0][0])))

print("Accuracy of unsupervised anomaly detection model on the test set: "+str
(100*(cm_test[0][0]+cm_test[1][1]) / (sum(cm_test[0]) + sum(cm_test[1]))) +
"%")
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, y_test_pred)))
```

```
Total fraudulent transactions detected in test set: 12 / 29
Total non-fraudulent transactions detected in test set: 18752 / 18769
Probability to detect a fraudulent transaction in the test set: 0.41379310344
827586
Probability to detect a non-fraudulent transaction in the test set: 0.9990942
511588258
Accuracy of unsupervised anomaly detection model on the test set: 99.81912969
464837%
ROC_AUC_score : 0.706444
```

In [30]:
```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18769
           1       0.41      0.41      0.41        29

    accuracy                           1.00     18798
   macro avg       0.71      0.71      0.71     18798
weighted avg       1.00      1.00      1.00     18798
```

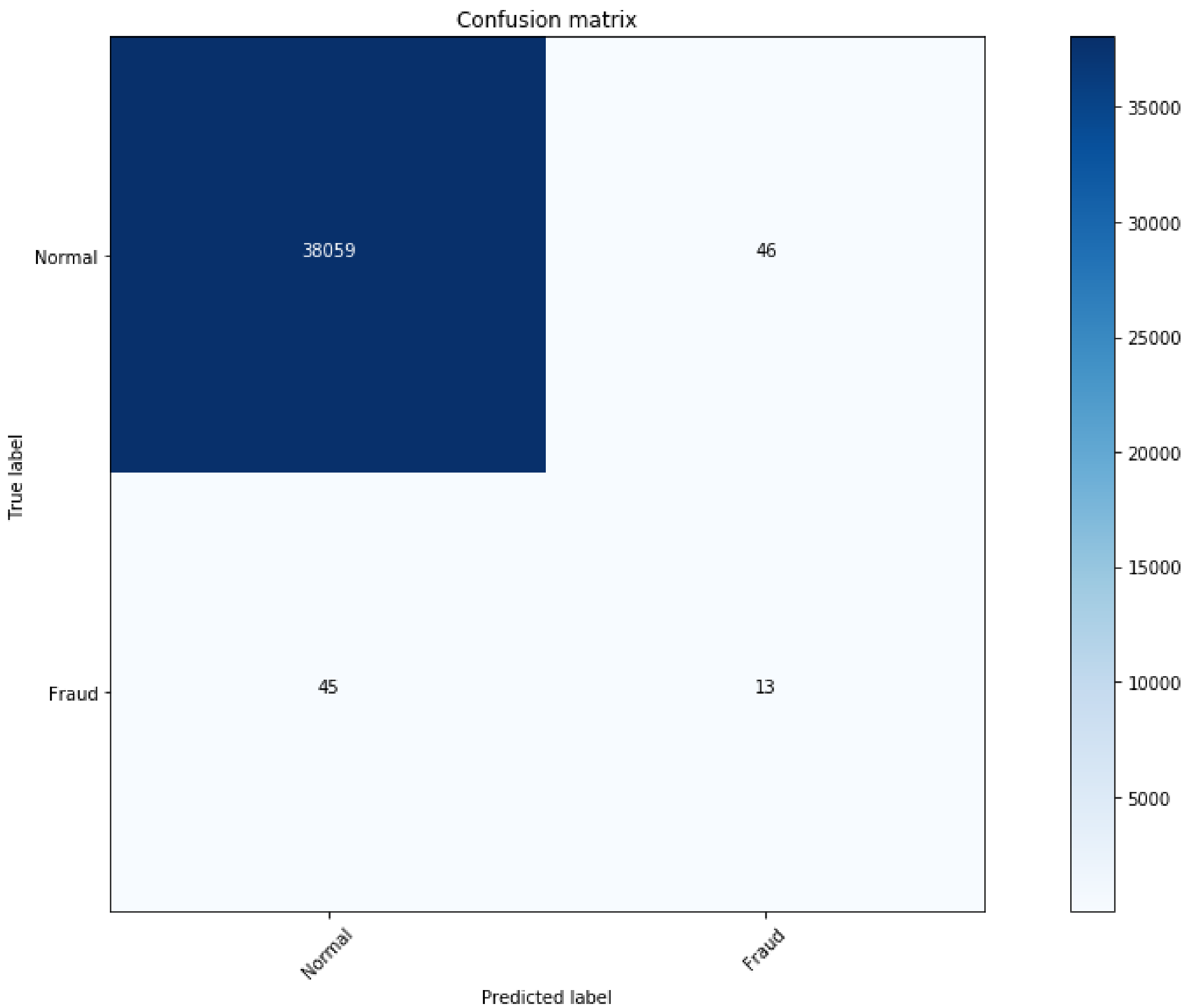
In [31]:
```
#plot_roc_curve(clf,X_test,y_test_pred)
```

Isolation Forest

In [32]:
```
clf = IsolationForest(n_estimators=100, max_samples=len(X),contamination=outli
er_fraction,random_state=state, verbose=0)
clf.fit(X_train)
#scores_prediction = clf.decision_function(X)
y_train_pred = clf.predict(X_train)
y_train_pred[y_train_pred == 1] = 0
y_train_pred[y_train_pred == -1] = 1

# On test set
y_test_pred = clf.fit_predict(X_test)
y_test_pred[y_test_pred == 1] = 0
y_test_pred[y_test_pred == -1] = 1
```

```
C:\Users\Piyush\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:281:
UserWarning: max_samples (56961) is greater than the total number of samples
(38163). max_samples will be set to n_samples for estimation.
  % (self.max_samples, n_samples))
C:\Users\Piyush\anaconda3\lib\site-packages\sklearn\ensemble\_iforest.py:281:
UserWarning: max_samples (56961) is greater than the total number of samples
(18798). max_samples will be set to n_samples for estimation.
  % (self.max_samples, n_samples))
```

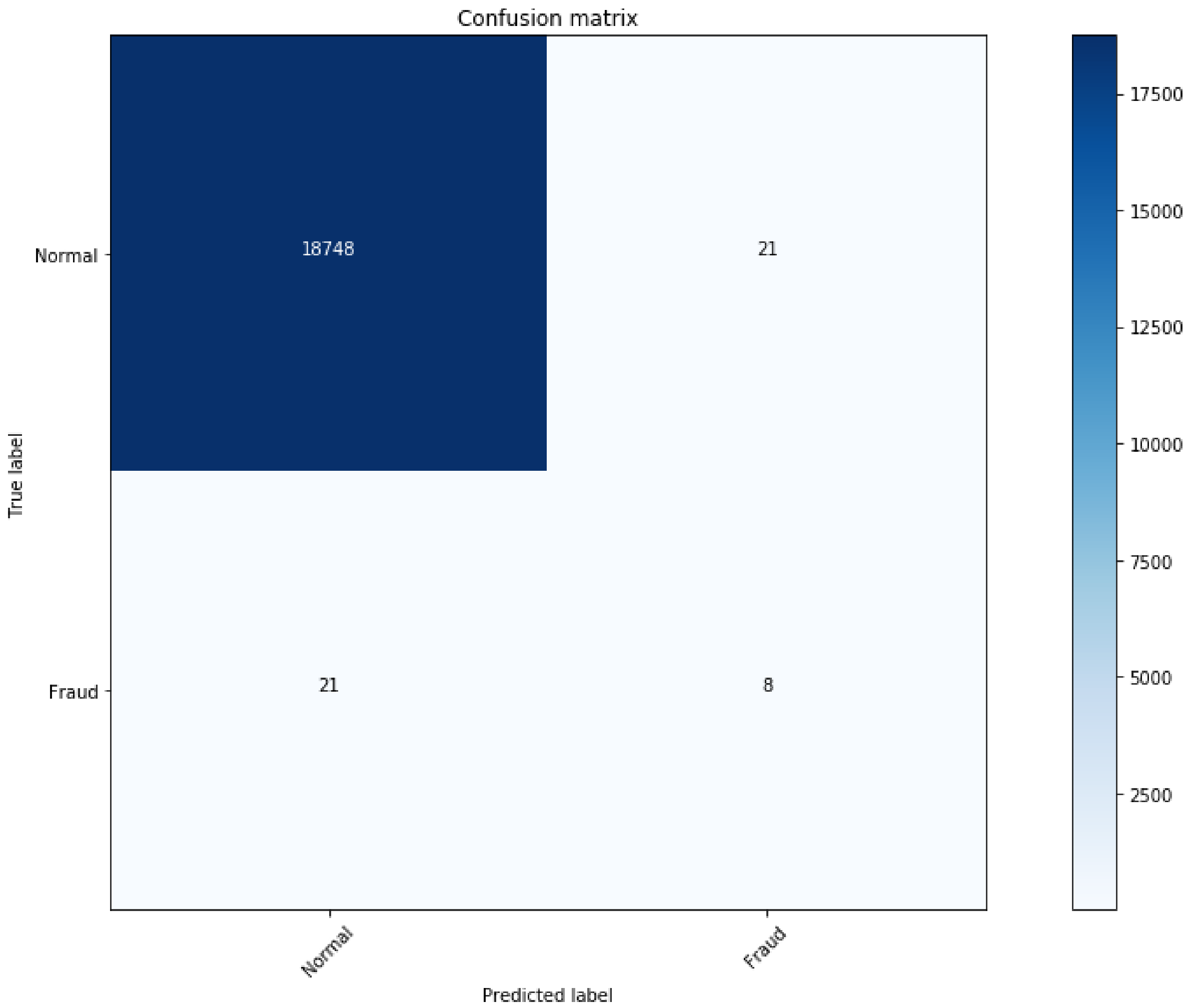
In [33]:
```
cm_train = confusion_matrix(y_train, y_train_pred)
plot_confusion_matrix(cm_train,["Normal", "Fraud"])
```



In [34]:
```
print('Total fraudulent transactions detected in training set: ' + str(cm_train[1][1]) + ' / ' + str(cm_train[1][1]+cm_train[1][0]))
print('Total non-fraudulent transactions detected in training set: ' + str(cm_train[0][0]) + ' / ' + str(cm_train[0][1]+cm_train[0][0]))

print('Probability to detect a fraudulent transaction in the training set: ' + str(cm_train[1][1]/(cm_train[1][1]+cm_train[1][0])))
print('Probability to detect a non-fraudulent transaction in the training set: ' + str(cm_train[0][0]/(cm_train[0][1]+cm_train[0][0])))

print("Accuracy of unsupervised anomaly detection model on the training set: "+str(100*(cm_train[0][0]+cm_train[1][1]) / (sum(cm_train[0]) + sum(cm_train[1]))) + "%")
```

```
Total fraudulent transactions detected in training set: 13 / 58
Total non-fraudulent transactions detected in training set: 38059 / 38105
Probability to detect a fraudulent transaction in the training set: 0.22413793103448276
Probability to detect a non-fraudulent transaction in the training set: 0.998792809342606
Accuracy of unsupervised anomaly detection model on the training set: 99.7615491444593%
```

In [35]:
```
cm_test = confusion_matrix( y_test,y_test_pred)
plot_confusion_matrix(cm_test,["Normal", "Fraud"])
```



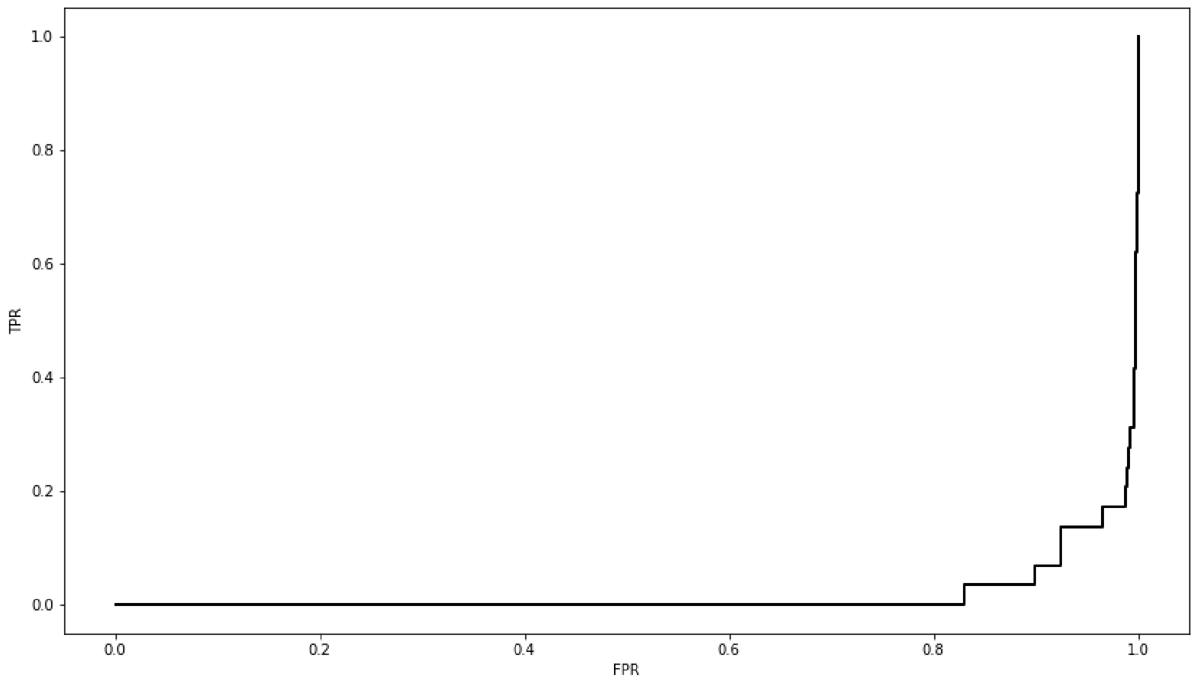
In [36]:
```
print('Total fraudulent transactions detected in test set: ' + str(cm_test[1][
1]) + ' / ' + str(cm_test[1][1]+cm_test[1][0]))
print('Total non-fraudulent transactions detected in test set: ' + str(cm_test
[0][0]) + ' / ' + str(cm_test[0][1]+cm_test[0][0]))

print('Probability to detect a fraudulent transaction in the test set: ' + str
(cm_test[1][1]/(cm_test[1][1]+cm_test[1][0])))
print('Probability to detect a non-fraudulent transaction in the test set: ' +
str(cm_test[0][0]/(cm_test[0][1]+cm_test[0][0])))

print("Accuracy of unsupervised anomaly detection model on the test set: "+str
(100*(cm_test[0][0]+cm_test[1][1]) / (sum(cm_test[0]) + sum(cm_test[1]))) +
"%")
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, y_test_pred)))
```

```
Total fraudulent transactions detected in test set: 8 / 29
Total non-fraudulent transactions detected in test set: 18748 / 18769
Probability to detect a fraudulent transaction in the test set: 0.27586206896
551724
Probability to detect a non-fraudulent transaction in the test set: 0.9988811
337844318
Accuracy of unsupervised anomaly detection model on the test set: 99.77657197
57421%
ROC_AUC_score : 0.637372
```

In [37]:
```
print(classification_report(y_test,y_test_pred))
```

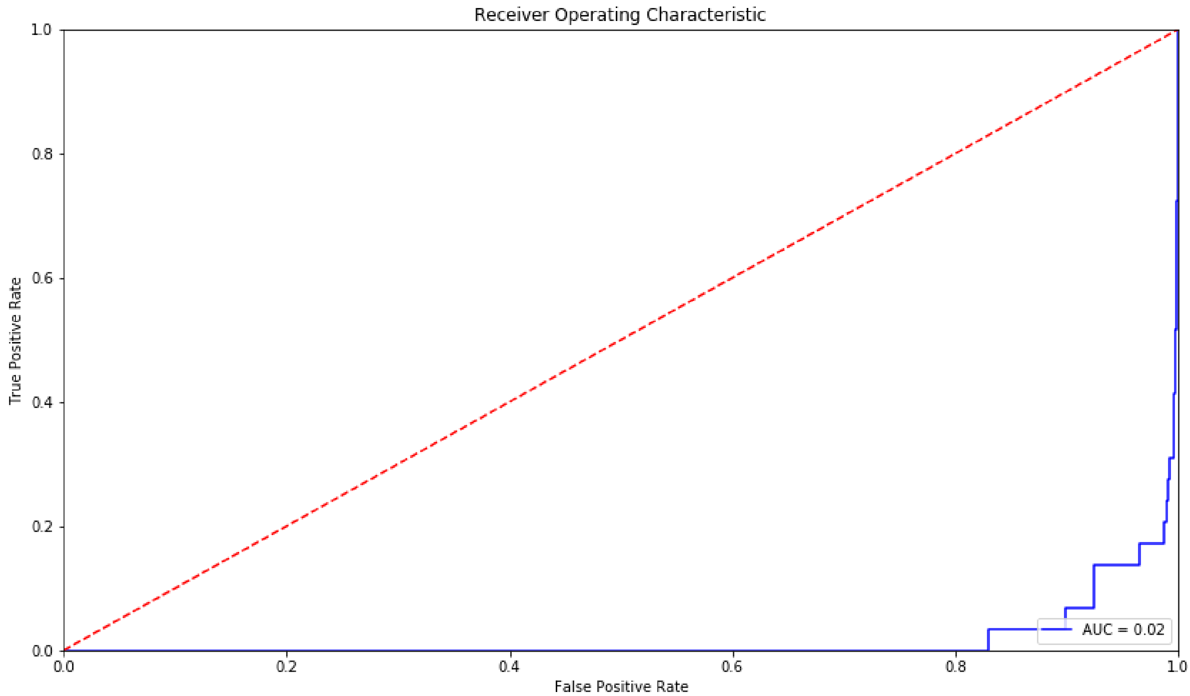
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18769
           1       0.28      0.28      0.28        29

    accuracy                           1.00     18798
   macro avg       0.64      0.64      0.64     18798
weighted avg       1.00      1.00      1.00     18798
```

In [38]:
```
y_pred = clf.decision_function(X_test)

from sklearn.metrics import roc_curve,roc_auc_score
fpr, tpr, thresholds = roc_curve(y_test,y_pred)
import matplotlib.pyplot as plt
plt.plot(fpr, tpr, 'k-', lw=2)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, y_pred)))
```




```
ROC_AUC_score : 0.018295
```

In [39]:
```
plot_roc(y_test,y_pred)
```



The results we've got through this model are far from ideal. We have not been able to classify fraudulent transactions efficiently despite having a high accuracy (which is not a good metric to measure performance on a skewed dataset anyways). Supervised learning for anomaly detection is the move fot this dataset since we have the labels. One reason why unsupervised learning did not perform well enough is because most of the fraudulent transactions did not have much unusual characteristics regarding them which can be well separated from normal transactions and I feel that's the main reason they provided us with a labelled dataset. Anyways, this notebook represents how unsupervised learning captures anomalies. The accuracy of detecting anomalies on the test set is 25%, which is way better than a random guess (the fraction of anomalies in the dataset is < 0.1%). I have also implemented the supervised learning model for this dataset, which works extremely well.

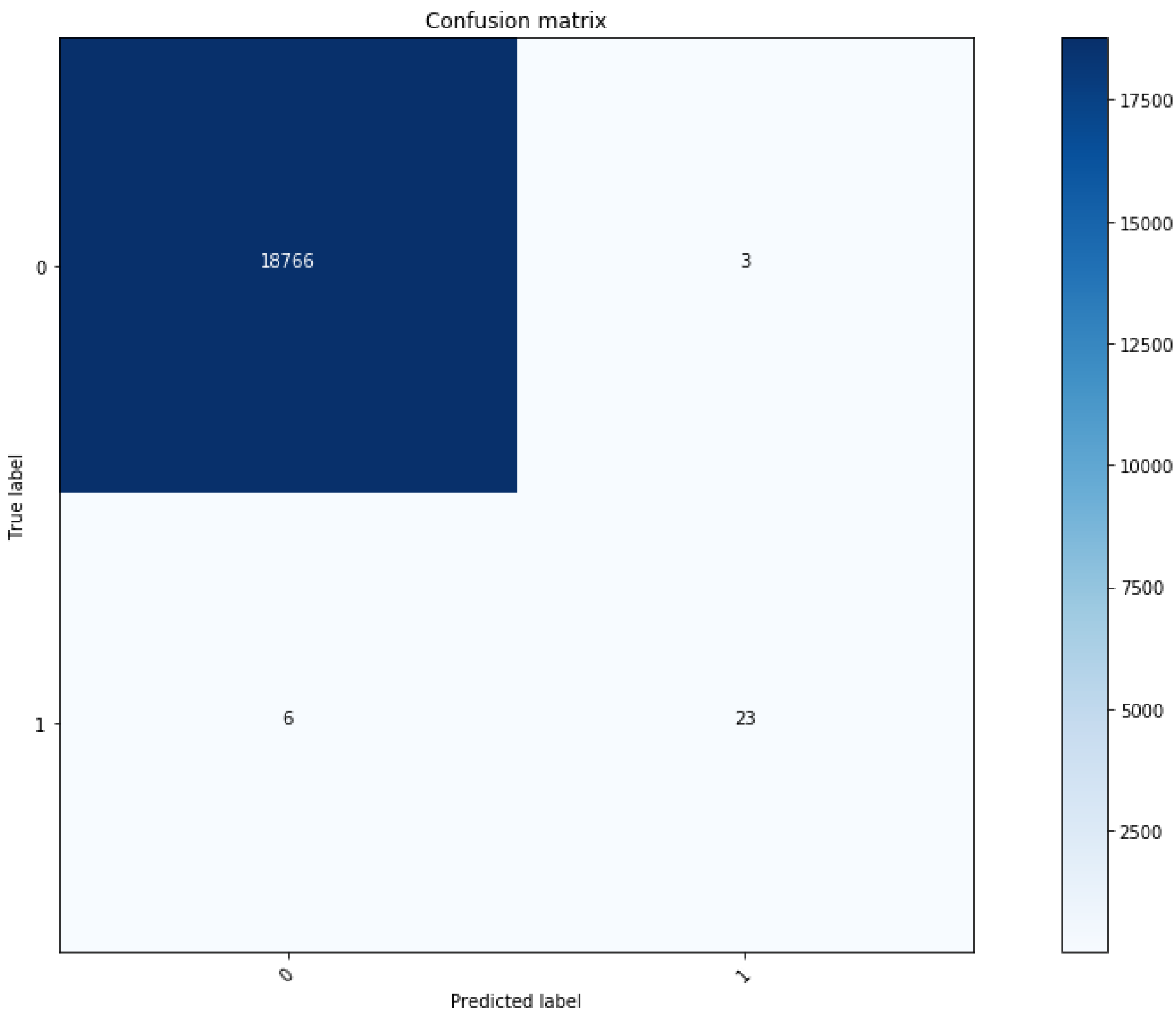
Supervised SVM

In [40]:
```
from sklearn import preprocessing, svm
classifier = svm.SVC(kernel='linear')
classifier.fit(X_train, y_train)
```

Out[40]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [41]:
```
predictions = classifier.predict(X_test)
```

In [42]:
```
cm = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cm,classes)
```



In [43]:
```
print('Total fraudulent transactions detected: ' + str(cm[1][1]) + ' / ' + str
(cm[1][1]+cm[1][0]))
print('Total non-fraudulent transactions detected: ' + str(cm[0][0]) + ' / ' +
str(cm[0][1]+cm[0][0]))

print('Probability to detect a fraudulent transaction: ' + str(cm[1][1]/(cm[1]
[1]+cm[1][0])))
print('Probability to detect a non-fraudulent transaction: ' + str(cm[0][0]/(c
m[0][1]+cm[0][0])))

print("Accuracy of the Logistic Regression model : "+str(100*(cm[0][0]+cm[1][1
]) / (sum(cm[0]) + sum(cm[1]))) + "%")
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, predictions)))
```

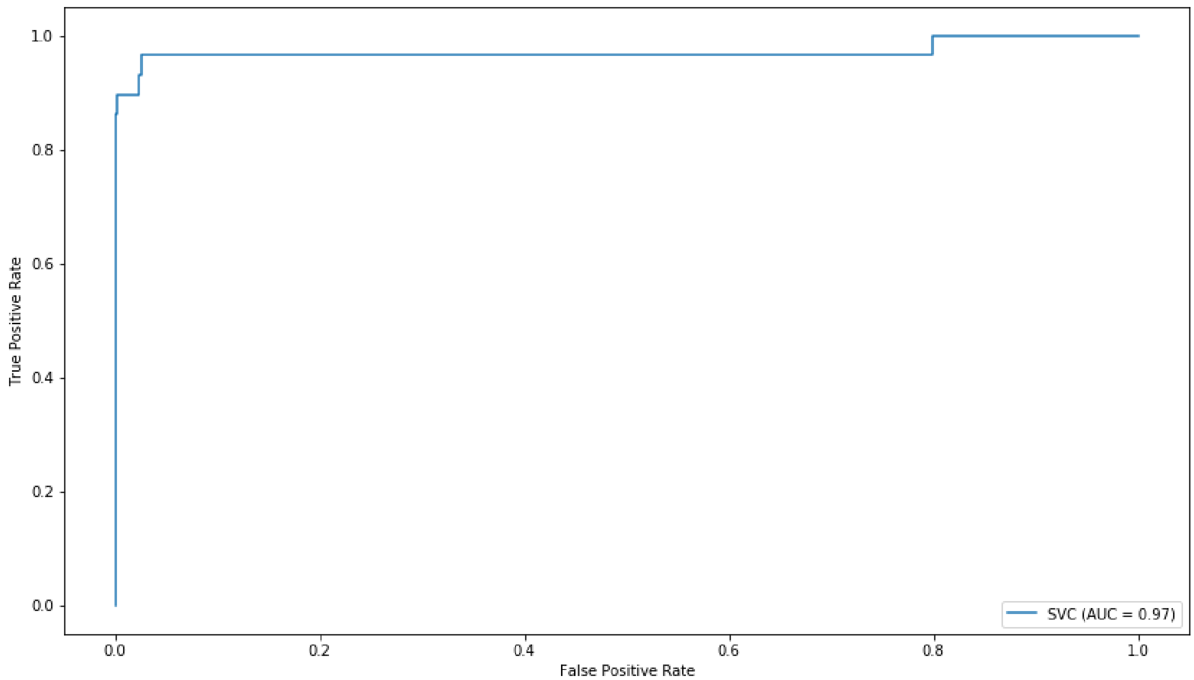
```
Total fraudulent transactions detected: 23 / 29
Total non-fraudulent transactions detected: 18766 / 18769
Probability to detect a fraudulent transaction: 0.7931034482758621
Probability to detect a non-fraudulent transaction: 0.9998401619692046
Accuracy of the Logistic Regression model : 99.95212256623044%
ROC_AUC_score : 0.896472
```

In [44]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18769
           1       0.88      0.79      0.84        29

    accuracy                           1.00     18798
   macro avg       0.94      0.90      0.92     18798
weighted avg       1.00      1.00      1.00     18798
```

In [45]:
```
plot_roc_curve(classifier,X_test,y_test)
```

Out[45]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1f0a5946c08>



Logistic Regression

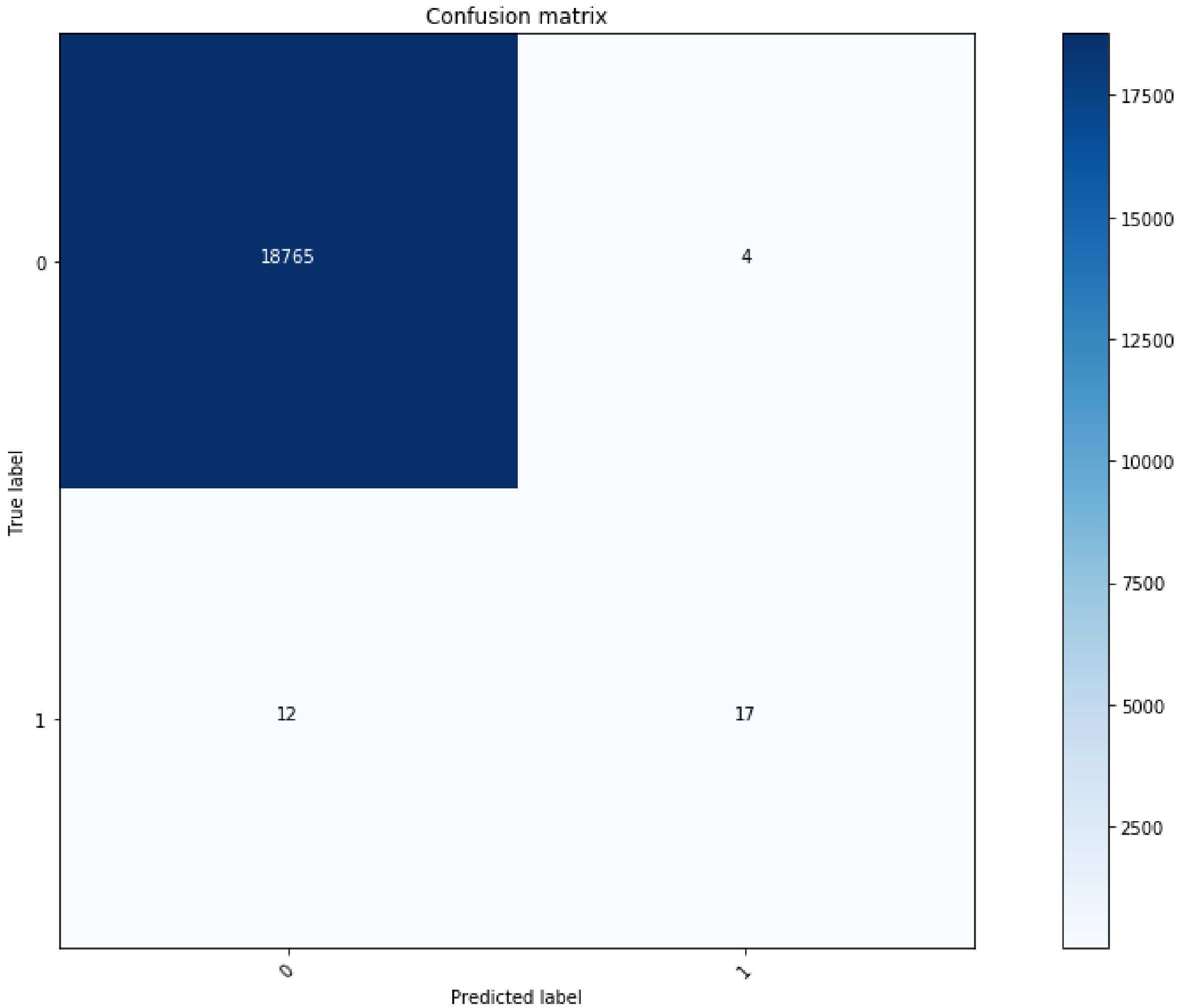
In [46]:
```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
```

```
C:\Users\Piyush\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:940: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

Out[46]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [47]:
```
predictions = classifier.predict(X_test)
```

In [48]:
```
cm = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cm,classes)
```



In [49]:
```
print('Total fraudulent transactions detected: ' + str(cm[1][1]) + ' / ' + str
(cm[1][1]+cm[1][0]))
print('Total non-fraudulent transactions detected: ' + str(cm[0][0]) + ' / ' +
str(cm[0][1]+cm[0][0]))

print('Probability to detect a fraudulent transaction: ' + str(cm[1][1]/(cm[1]
[1]+cm[1][0])))
print('Probability to detect a non-fraudulent transaction: ' + str(cm[0][0]/(c
m[0][1]+cm[0][0])))

print("Accuracy of the Logistic Regression model : "+str(100*(cm[0][0]+cm[1][1
]) / (sum(cm[0]) + sum(cm[1]))) + "%")
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, predictions)))
```

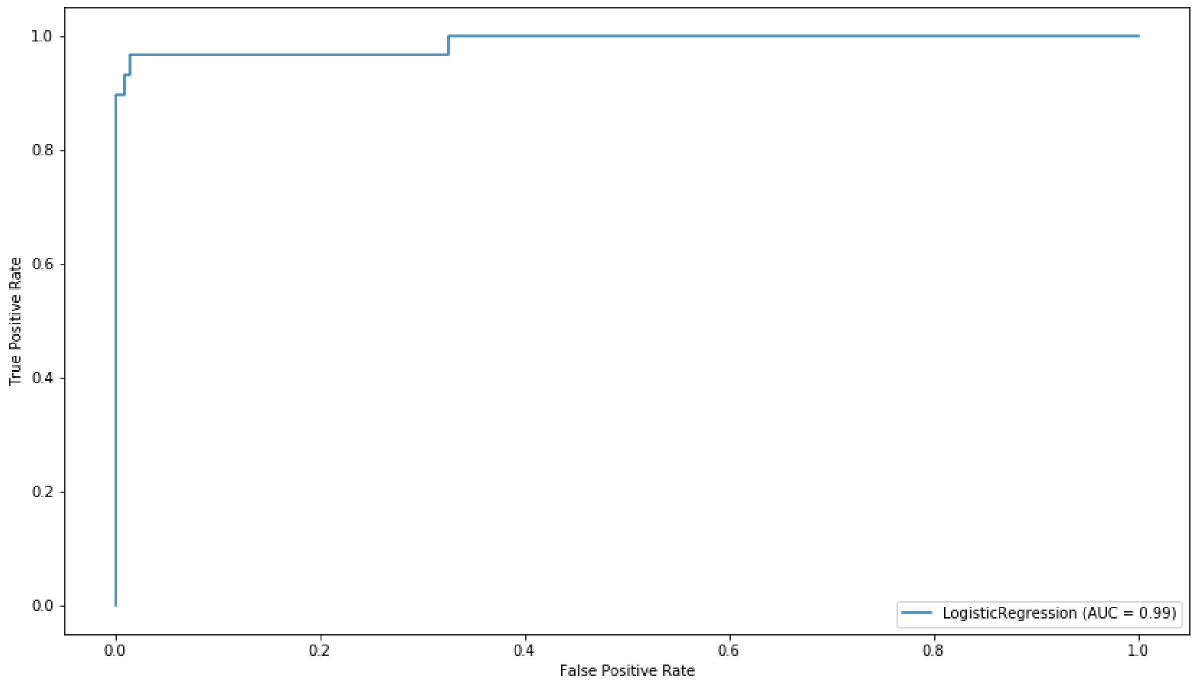
```
Total fraudulent transactions detected: 17 / 29
Total non-fraudulent transactions detected: 18765 / 18769
Probability to detect a fraudulent transaction: 0.5862068965517241
Probability to detect a non-fraudulent transaction: 0.9997868826256O6
Accuracy of the Logistic Regression model : 99.91488456218747%
ROC_AUC_score : 0.792997
```

In [50]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18769
           1       0.81      0.59      0.68        29

    accuracy                           1.00     18798
   macro avg       0.90      0.79      0.84     18798
weighted avg       1.00      1.00      1.00     18798
```

In [51]:
```
plot_roc_curve(classifier,X_test,y_test)
```

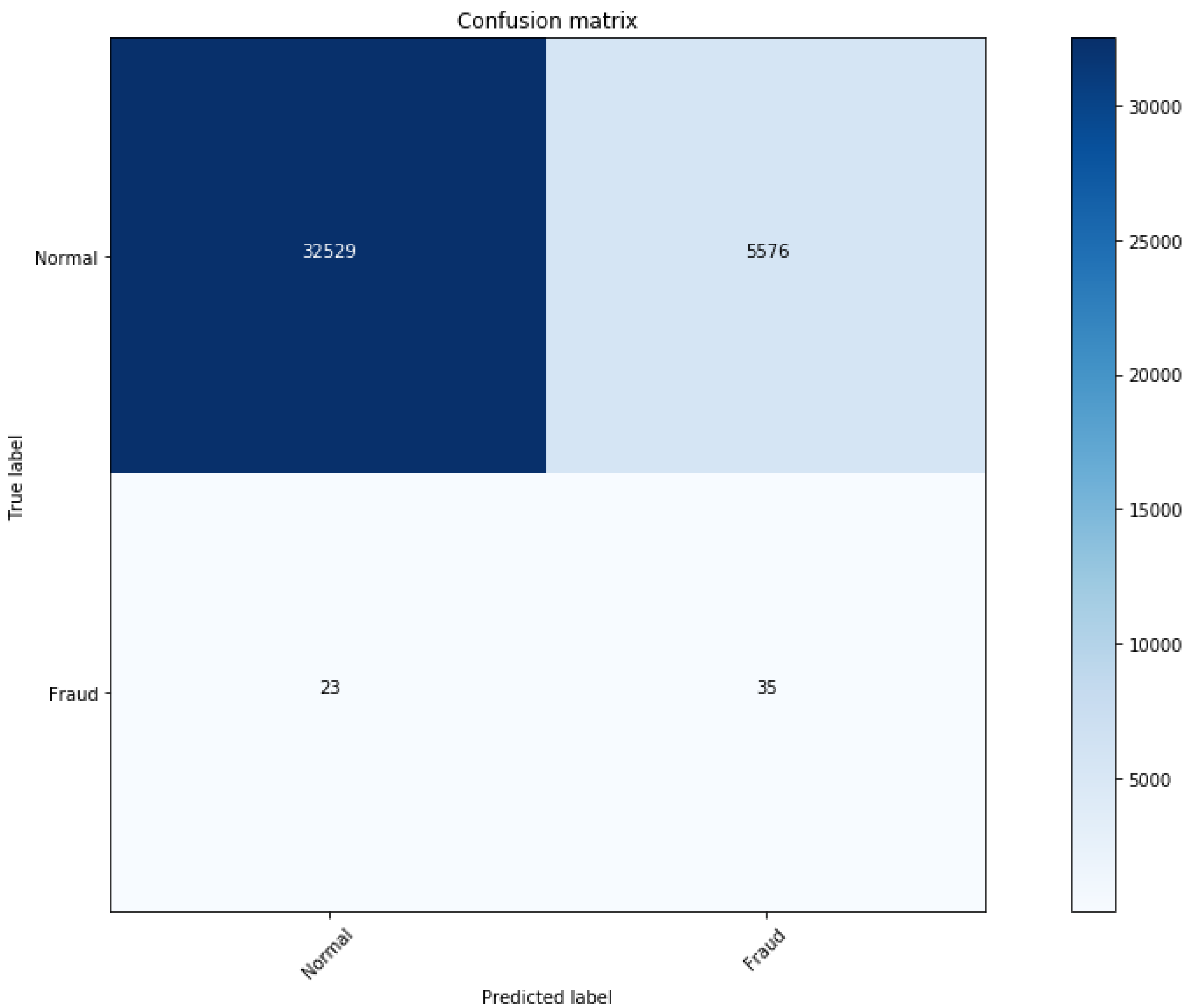
Out[51]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1f0a5e36188>
```



One Class SVM

In [52]:
```
clf = OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.05, max_iter=-1)
clf.fit(X_train)
#scores_prediction = clf.decision_function(X)
y_train_pred = clf.predict(X_train)
y_train_pred[y_train_pred == 1] = 0
y_train_pred[y_train_pred == -1] = 1

# On test set
y_test_pred = clf.fit_predict(X_test)
y_test_pred[y_test_pred == 1] = 0
y_test_pred[y_test_pred == -1] = 1
```

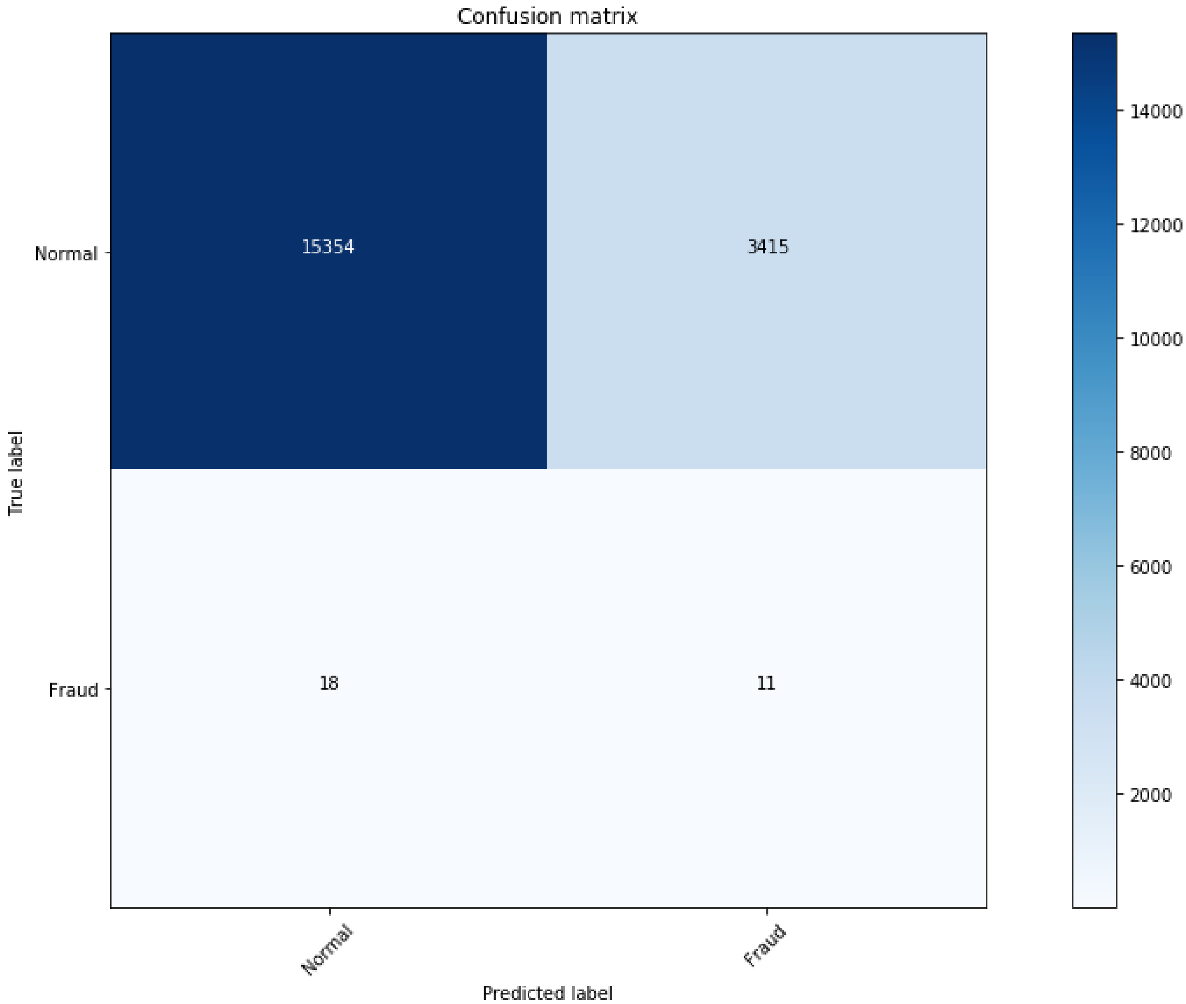
In [53]:
```
cm_train = confusion_matrix(y_train, y_train_pred)
plot_confusion_matrix(cm_train,["Normal", "Fraud"])
```



In [54]:
```
print('Total fraudulent transactions detected in training set: ' + str(cm_train[1][1]) + ' / ' + str(cm_train[1][1]+cm_train[1][0]))
print('Total non-fraudulent transactions detected in training set: ' + str(cm_train[0][0]) + ' / ' + str(cm_train[0][1]+cm_train[0][0]))

print('Probability to detect a fraudulent transaction in the training set: ' + str(cm_train[1][1]/(cm_train[1][1]+cm_train[1][0])))
print('Probability to detect a non-fraudulent transaction in the training set: ' + str(cm_train[0][0]/(cm_train[0][1]+cm_train[0][0])))

print("Accuracy of unsupervised anomaly detection model on the training set: "+str(100*(cm_train[0][0]+cm_train[1][1]) / (sum(cm_train[0]) + sum(cm_train[1]))) + "%")
```

```
Total fraudulent transactions detected in training set: 35 / 58
Total non-fraudulent transactions detected in training set: 32529 / 38105
Probability to detect a fraudulent transaction in the training set: 0.603448275862069
Probability to detect a non-fraudulent transaction in the training set: 0.8536674977037134
Accuracy of unsupervised anomaly detection model on the training set: 85.32872153656683%
```

In [55]:
```
cm_test = confusion_matrix(y_test,y_test_pred )
plot_confusion_matrix(cm_test,["Normal", "Fraud"])
```



In [56]:
```
print('Total fraudulent transactions detected in test set: ' + str(cm_test[1][
1]) + ' / ' + str(cm_test[1][1]+cm_test[1][0]))
print('Total non-fraudulent transactions detected in test set: ' + str(cm_test
[0][0]) + ' / ' + str(cm_test[0][1]+cm_test[0][0]))

print('Probability to detect a fraudulent transaction in the test set: ' + str
(cm_test[1][1]/(cm_test[1][1]+cm_test[1][0])))
print('Probability to detect a non-fraudulent transaction in the test set: ' +
str(cm_test[0][0]/(cm_test[0][1]+cm_test[0][0])))

print("Accuracy of unsupervised anomaly detection model on the test set: "+str
(100*(cm_test[0][0]+cm_test[1][1]) / (sum(cm_test[0]) + sum(cm_test[1]))) +
"%")
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, y_test_pred)))
```

```
Total fraudulent transactions detected in test set: 11 / 29
Total non-fraudulent transactions detected in test set: 15354 / 18769
Probability to detect a fraudulent transaction in the test set: 0.37931034482
75862
Probability to detect a non-fraudulent transaction in the test set: 0.8180510
416111674
Accuracy of unsupervised anomaly detection model on the test set: 81.73741887
434834%
ROC_AUC_score : 0.598681
```

In [57]:
```
print(classification_report(y_test,y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.82      0.90     18769
           1       0.00      0.38      0.01        29

    accuracy                           0.82     18798
   macro avg       0.50      0.60      0.45     18798
weighted avg       1.00      0.82      0.90     18798
```

One class SVM doesnt work well because it works on decison spearating bpundary

In [58]:
```
#plot_roc_curve(clf,X_test,y_test)
```

Multivariate Gaussian Anomaly detection

In [59]:
```
def covariance_matrix(X):
    m, n = X.shape
    tmp_mat = np.zeros((n, n))
    mu = X.mean(axis=0)
    for i in range(m):
        tmp_mat += np.outer(X[i] - mu, X[i] - mu)
    return tmp_mat / m
```

In [60]:
```
y_test
```

Out[60]:
```
150340    0
130939    0
150066    0
284285    0
196462    0
         ..
196504    0
39304     0
183807    0
37174     0
138361    0
Name: Class, Length: 18798, dtype: int64
```

In [61]:
```
cov_mat = covariance_matrix(np.array(X_train))
cov_mat
```

Out[61]:
```
array([[ 3.76983157e+00,  5.69398438e-02, -1.08915251e-01,
         1.39361900e-02, -1.25841357e-01,  5.09740584e-02,
         3.76852701e-02, -4.74898142e-02, -6.96788550e-03,
        -2.69690314e-02,  1.89724048e-02, -4.10063310e-02,
        -1.10822011e-02, -3.18423285e-02, -1.59334145e-03,
        -1.60992897e-02, -5.05926075e-02, -2.93453073e-02,
         1.00823486e-02, -7.64231919e-02, -5.13399425e-02,
         2.51336812e-02,  2.17613086e-02,  5.14124630e-03,
        -7.74853407e-04, -1.76959755e-02,  8.30014233e-03,
        -1.17708414e+02],
       [ 5.69398438e-02,  2.80234992e+00,  2.89273765e-03,
        -3.98528034e-02, -9.04445297e-02,  6.65076074e-02,
         1.22359772e-01,  2.97771244e-03,  1.18336515e-02,
         2.79667132e-02,  2.33974242e-03,  3.30202416e-02,
         5.75754933e-03,  8.62543733e-03,  1.16782569e-02,
         2.10451164e-02,  3.05374301e-02, -7.27917453e-03,
         7.82874261e-03, -1.06772600e-01, -2.26868451e-02,
         2.94055111e-03,  1.23216210e-02,  2.90494984e-03,
        -2.24932267e-03,  1.16388540e-02,  2.02930379e-02,
        -2.32940560e+02],
       [-1.08915251e-01,  2.89273765e-03,  2.18764188e+00,
         4.45775160e-02, -5.91685757e-02,  2.97134548e-02,
        -3.98756707e-02, -3.89424302e-02, -1.00716110e-02,
        -3.94104602e-02,  2.38724193e-02, -3.84931392e-02,
         1.76552440e-03, -4.23193302e-02, -3.43693391e-03,
        -2.37758373e-02, -5.63867800e-02, -2.18752102e-02,
         1.67512163e-02, -3.25295153e-02, -2.43877549e-02,
         1.27720135e-02,  1.89223397e-02,  2.36519681e-04,
        -8.93581528e-04,  2.20827332e-03, -2.02936342e-03,
        -7.81214884e+01],
       [ 1.39361900e-02, -3.98528034e-02,  4.45775160e-02,
         1.98394674e+00,  2.69051214e-02, -2.42139661e-02,
         1.30978662e-02,  2.37576742e-02,  2.82601886e-02,
         9.81574724e-03, -5.72581762e-03,  3.25898170e-02,
        -7.78799326e-03,  2.23902549e-02,  6.20565864e-03,
        -8.80168499e-03,  3.15406076e-02,  1.19855470e-02,
         2.38766881e-03,  1.35670092e-02,  4.61505572e-03,
        -1.44747375e-02, -2.04712916e-03, -3.14856560e-03,
        -2.55560282e-03,  9.18095212e-03, -6.90122068e-03,
         3.84002706e+01],
       [-1.25841357e-01, -9.04445297e-02, -5.91685757e-02,
         2.69051214e-02,  1.88976586e+00, -1.82842414e-03,
        -2.25789537e-02, -3.40420187e-02, -5.93858729e-03,
        -3.32803833e-02,  2.87600891e-02, -2.99783859e-02,
         5.91343970e-03, -1.14664381e-02, -3.03053605e-03,
        -5.78126530e-03, -3.81312401e-02, -2.76987450e-02,
         9.56199896e-03, -3.31134252e-02, -2.42209170e-02,
         6.93113518e-03, -1.72110817e-03, -1.82321190e-03,
         4.12872000e-03,  1.62165961e-02, -1.73135666e-02,
        -1.22865511e+02],
       [ 5.09740584e-02,  6.65076074e-02,  2.97134548e-02,
        -2.42139661e-02, -1.82842414e-03,  1.76249398e+00,
        -1.37915887e-02,  1.77092209e-02, -3.69200635e-03,
        -5.16031143e-03, -2.51430929e-03,  3.20849760e-03,
         1.46957499e-04, -1.27438135e-02, -2.72989597e-03,
        -1.04646552e-02, -4.49632428e-03, -5.54287983e-03,
         2.19224281e-03,  1.68173466e-02,  3.26475231e-02,
```

```
         1.20534225e-03,  1.24114131e-02,  2.61067830e-03,
         1.63099737e-03, -1.62011154e-02,  3.74766238e-03,
         6.19335774e+01],
       [ 3.76852701e-02,  1.22359772e-01, -3.98756707e-02,
         1.30978662e-02, -2.25789537e-02, -1.37915887e-02,
         1.41968243e+00, -3.17472205e-02, -3.68409760e-02,
        -6.57465043e-02,  5.00241893e-03, -3.48721484e-02,
        -1.68143355e-02, -1.46274099e-02, -4.10788800e-03,
        -4.63275113e-02, -4.30115255e-02, -2.14999472e-02,
         4.77720058e-03,  7.02668369e-02, -1.63166087e-03,
         1.07776383e-02,  3.08625626e-02,  7.48736968e-04,
        -2.38282657e-03, -3.08368916e-02,  4.35088040e-03,
         1.10898424e+02],
       [-4.74898142e-02,  2.97771244e-03, -3.89424302e-02,
         2.37576742e-02, -3.40420187e-02,  1.77092209e-02,
        -3.17472205e-02,  1.36111579e+00, -1.02038593e-02,
        -1.74381234e-02, -1.17793913e-02, -3.75017565e-03,
        -5.79440581e-04, -5.07948468e-04, -2.89957979e-04,
         1.66836238e-03, -2.96366469e-03, -1.04917265e-02,
        -4.51854616e-03,  1.61303217e-02, -4.97213341e-02,
         1.49585765e-02,  2.58223472e-02,  3.33082291e-03,
         8.43506790e-03, -4.18062582e-03, -3.70233810e-03,
        -3.01519428e+01],
       [-6.96788550e-03,  1.18336515e-02, -1.00716110e-02,
         2.82601886e-02, -5.93858729e-03, -3.69200635e-03,
        -3.68409760e-02, -1.02038593e-02,  1.18570828e+00,
        -3.27667678e-02, -3.69609852e-03, -1.32435840e-02,
        -1.46883141e-03, -3.38352762e-03, -1.07703842e-02,
        -3.30894473e-03, -1.54190068e-02, -2.56890132e-03,
        -2.59656543e-03,  5.00425058e-03, -1.01867362e-03,
        -1.95676662e-03,  5.89431130e-03, -1.25336057e-03,
         6.50010191e-04,  6.79530563e-03,  1.99583903e-03,
        -1.17890070e+01],
       [-2.69690314e-02,  2.79667132e-02, -3.94104602e-02,
         9.81574724e-03, -3.32803833e-02, -5.16031143e-03,
        -6.57465043e-02, -1.74381234e-02, -3.27667678e-02,
         1.13110244e+00,  9.88926176e-03, -2.68967858e-02,
        -9.75936766e-03, -1.23415497e-02,  2.87483857e-04,
        -3.08305612e-02, -3.29629591e-02, -1.39643401e-02,
         6.00443894e-03, -6.03702088e-03, -1.51532170e-03,
         5.53032673e-03,  7.19383868e-03,  1.89048541e-03,
         3.14646817e-03,  1.14631512e-02,  5.48420889e-03,
        -2.97216199e+01],
       [ 1.89724048e-02,  2.33974242e-03,  2.38724193e-02,
        -5.72581762e-03,  2.87600891e-02, -2.51430929e-03,
         5.00241893e-03, -1.17793913e-02, -3.69609852e-03,
         9.88926176e-03,  1.03171098e+00,  1.52074234e-02,
         2.06187718e-03,  3.00761917e-02,  5.33074610e-03,
         1.89296339e-02,  1.47675328e-02,  5.69357248e-03,
        -3.57125953e-03, -5.85001854e-04,  3.80717160e-04,
         1.88372232e-03, -7.65856371e-03, -3.52060643e-03,
        -1.73299075e-03,  2.18183316e-03, -4.63306486e-04,
        -2.05554844e+00],
       [-4.10063310e-02,  3.30202416e-02, -3.84931392e-02,
         3.25898170e-02, -2.99783859e-02,  3.20849760e-03,
        -3.48721484e-02, -3.75017565e-03, -1.32435840e-02,
        -2.68967858e-02,  1.52074234e-02,  9.67314089e-01,
```

```
        -9.69375420e-03, -3.20838819e-02,  1.30444608e-03,
        -2.74472917e-02, -3.47480054e-02, -1.34078587e-02,
         4.68084495e-03,  8.02458031e-03, -1.68193045e-03,
         8.89278783e-04,  1.84959165e-04,  2.81193407e-04,
         4.18575092e-03, -6.55466694e-04, -2.95222248e-04,
        -1.31983952e+00],
       [-1.10822011e-02,  5.75754933e-03,  1.76552440e-03,
        -7.78799326e-03,  5.91343970e-03,  1.46957499e-04,
        -1.68143355e-02, -5.79440581e-04, -1.46883141e-03,
        -9.75936766e-03,  2.06187718e-03, -9.69375420e-03,
         9.97269712e-01,  3.68470145e-03, -8.62652575e-03,
         1.71964212e-04,  4.65056386e-04,  2.60931141e-03,
         1.81115051e-03,  7.20000309e-03,  3.60362193e-03,
         1.84756051e-03, -3.35245165e-04, -1.85044722e-03,
        -2.32453187e-03,  1.33039573e-03, -1.17055112e-03,
         7.20870197e-01],
       [-3.18423285e-02,  8.62543733e-03, -4.23193302e-02,
         2.23902549e-02, -1.14664381e-02, -1.27438135e-02,
        -1.46274099e-02, -5.07948468e-04, -3.38352762e-03,
        -1.23415497e-02,  3.00761917e-02, -3.20838819e-02,
         3.68470145e-03,  8.79387931e-01,  2.38962857e-03,
        -1.42125854e-02, -2.68601841e-02, -7.13787749e-03,
         4.02517124e-03,  3.10598771e-03,  6.71744144e-03,
        -7.89072773e-03,  3.35186320e-03,  1.48755818e-03,
        -3.91356550e-04, -4.48618826e-03, -7.63530234e-04,
         1.00496904e+01],
       [-1.59334145e-03,  1.16782569e-02, -3.43693391e-03,
         6.20565864e-03, -3.03053605e-03, -2.72989597e-03,
        -4.10788800e-03, -2.89957979e-04, -1.07703842e-02,
         2.87483857e-04,  5.33074610e-03,  1.30444608e-03,
        -8.62652575e-03,  2.38962857e-03,  8.32946981e-01,
         3.32602992e-03,  1.29377133e-03, -7.50461033e-03,
        -1.12029405e-02,  4.93582022e-03,  1.13245002e-03,
        -4.32072234e-03, -3.08858960e-03, -4.37099281e-03,
         2.97316175e-03,  2.17313422e-03, -2.84219285e-03,
         3.33771921e-01],
       [-1.60992897e-02,  2.10451164e-02, -2.37758373e-02,
        -8.80168499e-03, -5.78126530e-03, -1.04646552e-02,
        -4.63275113e-02,  1.66836238e-03, -3.30894473e-03,
        -3.08305612e-02,  1.89296339e-02, -2.74472917e-02,
         1.71964212e-04, -1.42125854e-02,  3.32602992e-03,
         7.57580431e-01, -3.26881106e-02, -1.22066947e-02,
         4.33544429e-03,  1.15641530e-02,  2.99863156e-03,
         3.94811214e-03,  6.19252898e-04, -2.27423078e-03,
        -6.74535466e-03, -5.13421363e-03, -1.25006739e-03,
        -2.57778653e+00],
       [-5.05926075e-02,  3.05374301e-02, -5.63867800e-02,
         3.15406076e-02, -3.81312401e-02, -4.49632428e-03,
        -4.30115255e-02, -2.96366469e-03, -1.54190068e-02,
        -3.29629591e-02,  1.47675328e-02, -3.47480054e-02,
         4.65056386e-04, -2.68601841e-02,  1.29377133e-03,
        -3.26881106e-02,  6.71671379e-01, -1.82552641e-02,
         3.74265054e-03,  3.32776050e-03, -3.50823219e-04,
         6.16479375e-05,  7.07749044e-03, -3.05192654e-03,
         9.76540519e-04, -1.76543699e-03,  1.30088728e-03,
         2.19609281e+00],
       [-2.93453073e-02, -7.27917453e-03, -2.18752102e-02,
```

```
 1.19855470e-02, -2.76987450e-02, -5.54287983e-03,
-2.14999472e-02, -1.04917265e-02, -2.56890132e-03,
-1.39643401e-02,  5.69357248e-03, -1.34078587e-02,
 2.60931141e-03, -7.13787749e-03, -7.50461033e-03,
-1.22066947e-02, -1.82552641e-02,  7.02276963e-01,
 6.67921008e-03,  4.45445184e-03, -4.49543751e-03,
-3.15187243e-04,  7.65111383e-03, -6.11963952e-04,
 7.68344464e-04, -1.82673218e-03, -6.64209031e-04,
 9.25530226e+00],
[ 1.00823486e-02,  7.82874261e-03,  1.67512163e-02,
 2.38766881e-03,  9.56199896e-03,  2.19224281e-03,
 4.77720058e-03, -4.51854616e-03, -2.59656543e-03,
 6.00443894e-03, -3.57125953e-03,  4.68084495e-03,
 1.81115051e-03,  4.02517124e-03, -1.12029405e-02,
 4.33544429e-03,  3.74265054e-03,  6.67921008e-03,
 6.66865664e-01, -5.90360435e-04,  2.79505439e-04,
-1.93317406e-03,  1.64305044e-03,  8.72145502e-04,
 3.69638671e-03,  1.94949531e-03, -4.37552125e-03,
-1.32928932e+01],
[-7.64231919e-02, -1.06772600e-01, -3.25295153e-02,
 1.35670092e-02, -3.31134252e-02,  1.68173466e-02,
 7.02668369e-02,  1.61303217e-02,  5.00425058e-03,
-6.03702088e-03, -5.85001854e-04,  8.02458031e-03,
 7.20000309e-03,  3.10598771e-03,  4.93582022e-03,
 1.15641530e-02,  3.32776050e-03,  4.45445184e-03,
-5.90360435e-04,  5.89706284e-01,  4.24288853e-03,
-2.83128413e-03,  1.50494483e-02,  7.56341046e-03,
-4.68948093e-03, -4.29732550e-03,  1.46749315e-02,
 8.22846064e+01],
[-5.13399425e-02, -2.26868451e-02, -2.43877549e-02,
 4.61505572e-03, -2.42209170e-02,  3.26475231e-02,
-1.63166087e-03, -4.97213341e-02, -1.01867362e-03,
-1.51532170e-03,  3.80717160e-04, -1.68193045e-03,
 3.60362193e-03,  6.71744144e-03,  1.13245002e-03,
 2.99863156e-03, -3.50823219e-04, -4.49543751e-03,
 2.79505439e-04,  4.24288853e-03,  5.38345618e-01,
-1.06558528e-03,  3.57941826e-03,  1.54193647e-03,
 6.18353389e-04, -4.87327089e-03,  1.45425008e-03,
 2.64036915e+01],
[ 2.51336812e-02,  2.94055111e-03,  1.27720135e-02,
-1.44747375e-02,  6.93113518e-03,  1.20534225e-03,
 1.07776383e-02,  1.49585765e-02, -1.95676662e-03,
 5.53032673e-03,  1.88372232e-03,  8.89278783e-04,
 1.84756051e-03, -7.89072773e-03, -4.32072234e-03,
 3.94811214e-03,  6.16479375e-05, -3.15187243e-04,
-1.93317406e-03, -2.83128413e-03, -1.06558528e-03,
 5.25173962e-01,  9.51548531e-03,  1.71110562e-03,
-2.08877679e-04,  1.24356864e-03, -2.32134711e-03,
-1.52435910e+01],
[ 2.17613086e-02,  1.23216210e-02,  1.89223397e-02,
-2.04712916e-03, -1.72110817e-03,  1.24114131e-02,
 3.08625626e-02,  2.58223472e-02,  5.89431130e-03,
 7.19383868e-03, -7.65856371e-03,  1.84959165e-04,
-3.35245165e-04,  3.35186320e-03, -3.08858960e-03,
 6.19252898e-04,  7.07749044e-03,  7.65111383e-03,
 1.64305044e-03,  1.50494483e-02,  3.57941826e-03,
 9.51548531e-03,  4.62263452e-01,  1.43295988e-02,
```

```
 -4.92191455e-04, -7.26407237e-03, -1.24728228e-02,
 -2.78304983e+01],
[ 5.14124630e-03,  2.90494984e-03,  2.36519681e-04,
 -3.14856560e-03, -1.82321190e-03,  2.61067830e-03,
  7.48736968e-04,  3.33082291e-03, -1.25336057e-03,
  1.89048541e-03, -3.52060643e-03,  2.81193407e-04,
 -1.85044722e-03,  1.48755818e-03, -4.37099281e-03,
 -2.27423078e-03, -3.05192654e-03, -6.11963952e-04,
  8.72145502e-04,  7.56341046e-03,  1.54193647e-03,
  1.71110562e-03,  1.43295988e-02,  2.72619474e-01,
 -1.11123462e-03, -2.04529732e-03, -4.84905042e-03,
 -8.01236386e+00],
[-7.74853407e-04, -2.24932267e-03, -8.93581528e-04,
 -2.55560282e-03,  4.12872000e-03,  1.63099737e-03,
 -2.38282657e-03,  8.43506790e-03,  6.50010191e-04,
  3.14646817e-03, -1.73299075e-03,  4.18575092e-03,
 -2.32453187e-03, -3.91356550e-04,  2.97316175e-03,
 -6.74535466e-03,  9.76540519e-04,  7.68344464e-04,
  3.69638671e-03, -4.68948093e-03,  6.18353389e-04,
 -2.08877679e-04, -4.92191455e-04, -1.11123462e-03,
  2.33234293e-01,  4.22776601e-04, -2.35299710e-03,
 -1.35732569e+00],
[-1.76959755e-02,  1.16388540e-02,  2.20827332e-03,
  9.18095212e-03,  1.62165961e-02, -1.62011154e-02,
 -3.08368916e-02, -4.18062582e-03,  6.79530563e-03,
  1.14631512e-02,  2.18183316e-03, -6.55466694e-04,
  1.33039573e-03, -4.48618826e-03,  2.17313422e-03,
 -5.13421363e-03, -1.76543699e-03, -1.82673218e-03,
  1.94949531e-03, -4.29732550e-03, -4.87327089e-03,
  1.24356864e-03, -7.26407237e-03, -2.04529732e-03,
  4.22776601e-04,  1.55103124e-01,  3.73418580e-03,
 -1.84722110e-01],
[ 8.30014233e-03,  2.02930379e-02, -2.02936342e-03,
 -6.90122068e-03, -1.73135666e-02,  3.74766238e-03,
  4.35088040e-03, -3.70233810e-03,  1.99583903e-03,
  5.48420889e-03, -4.63306486e-04, -2.95222248e-04,
 -1.17055112e-03, -7.63530234e-04, -2.84219285e-03,
 -1.25006739e-03,  1.30088728e-03, -6.64209031e-04,
 -4.37552125e-03,  1.46749315e-02,  1.45425008e-03,
 -2.32134711e-03, -1.24728228e-02, -4.84905042e-03,
 -2.35299710e-03,  3.73418580e-03,  9.04210303e-02,
  1.42158207e+00],
[-1.17708414e+02, -2.32940560e+02, -7.81214884e+01,
  3.84002706e+01, -1.22865511e+02,  6.19335774e+01,
  1.10898424e+02, -3.01519428e+01, -1.17890070e+01,
 -2.97216199e+01, -2.05554844e+00, -1.31983952e+00,
  7.20870197e-01,  1.00496904e+01,  3.33771921e-01,
 -2.57778653e+00,  2.19609281e+00,  9.25530226e+00,
 -1.32928932e+01,  8.22846064e+01,  2.64036915e+01,
 -1.52435910e+01, -2.78304983e+01, -8.01236386e+00,
 -1.35732569e+00, -1.84722110e-01,  1.42158207e+00,
  6.55978068e+04]])
```

In [62]:
```
cov_mat_inv = np.linalg.pinv(cov_mat)
cov_mat_det = np.linalg.det(cov_mat)
def multi_gauss(x):
    n = len(cov_mat)
    #print(x)
    return (np.exp(-0.5 * np.dot(x, np.dot(cov_mat_inv, x.transpose())))
            / (2. * np.pi)**(n/2.)
            / np.sqrt(cov_mat_det))
```

In [63]:
```
X_test = np.array(X_test)
y_test = np.array(y_test)
y_test
```

Out[63]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [64]:
```
from sklearn.metrics import confusion_matrix

def stats(X_test, y_test, eps):
    predictions = np.array([multi_gauss(x) <= eps for x in X_test], dtype=bool
)
    #print("fk")
    y_test = np.array(y_test, dtype=bool)
    #print("fk")
    #print(y_test)
    #print(predictions)
    tn, fp, fn, tp = confusion_matrix(y_test, predictions).ravel()
    #print("fk")
    recall = tp / (tp + fn)
    prec = tp / (tp + fp)
    F1 = 2 * recall * prec / (recall + prec)
    return recall, prec, F1
```

In [65]:
```
eps = 0.000000000002
```

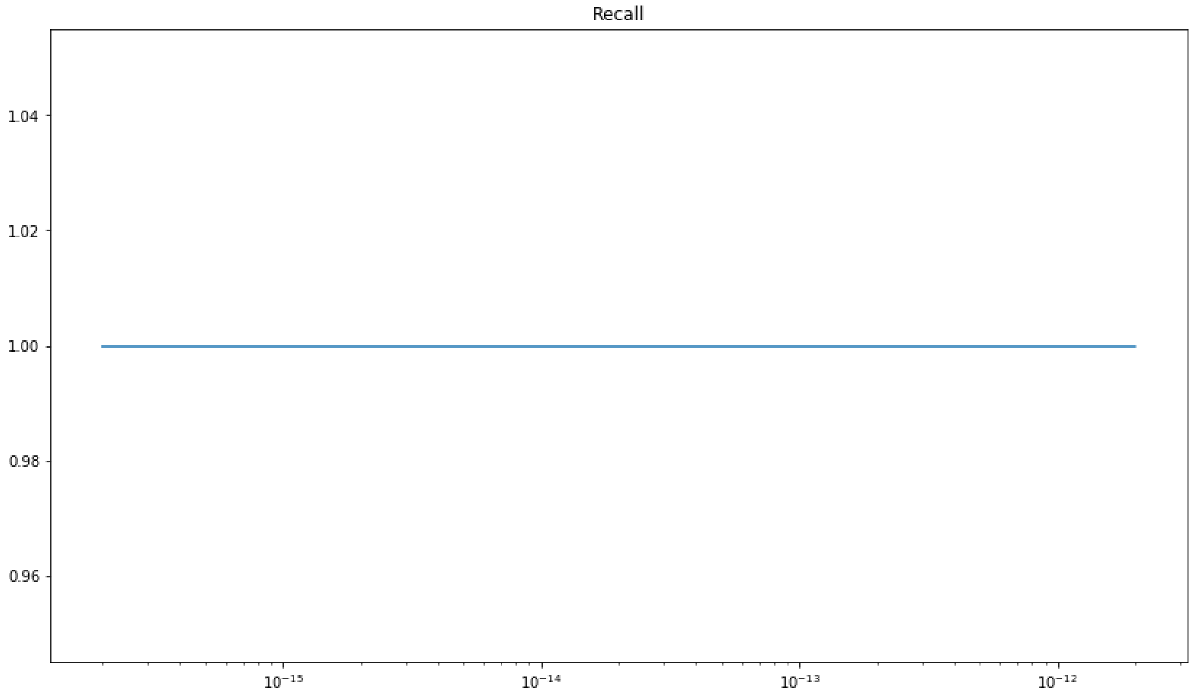
In [66]:
```
#print(y_test)
recall, prec, F1 = stats(X_test, y_test, eps)
print("For a boundary of:", eps)
print("Recall:", recall)
print("Precision:", prec)
print("F1-score:", F1)
```

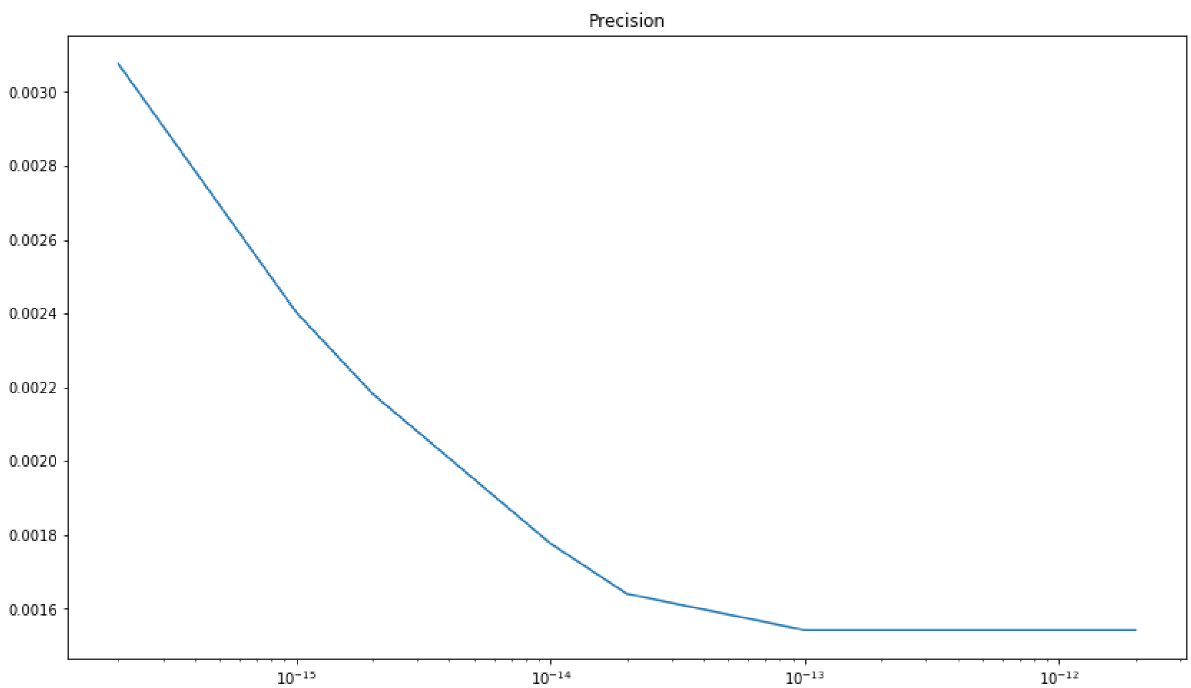
```
For a boundary of: 2e-12
Recall: 1.0
Precision: 0.001542717310352165
F1-score: 0.003080681999256387
```

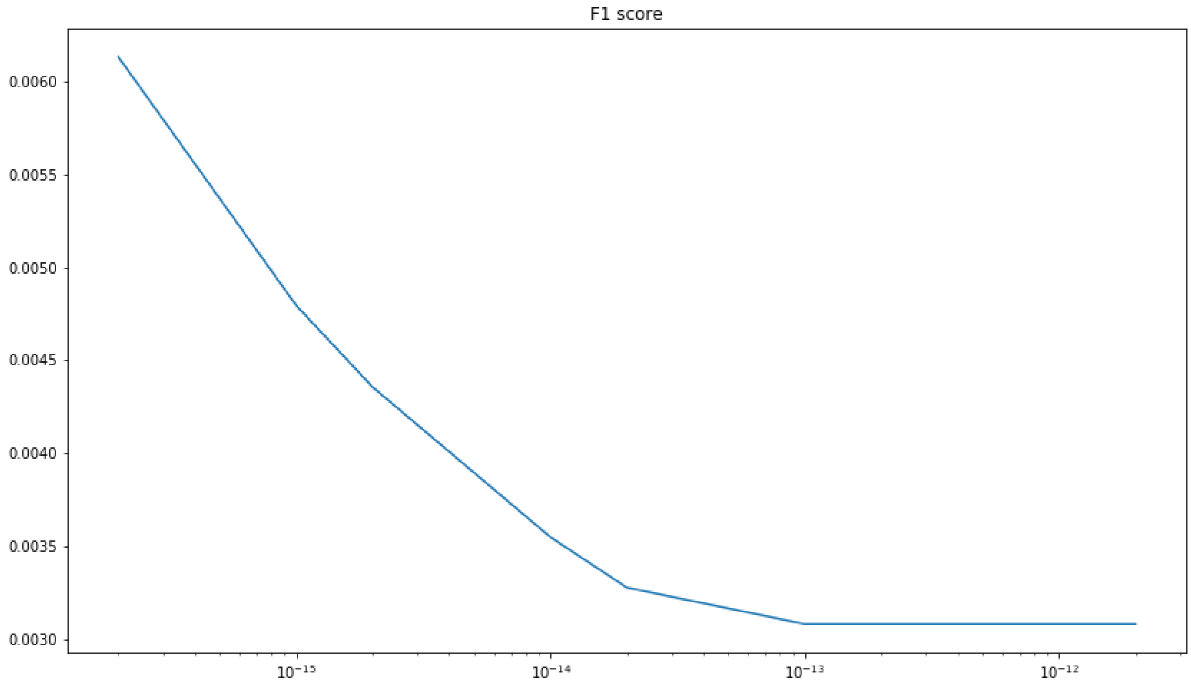
In [67]:
```
validation = []
print(X_test)
print(y_test)
for thresh in np.array([1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001
]) * eps:
    recall, prec, F1 = stats(X_test, y_test, thresh)
    validation.append([thresh, recall, prec, F1])
```

```
[[-2.00352998e+00  2.17175749e+00  1.63056588e-01 ... -3.81730874e-01
  -8.77567419e-02  1.62700000e+01]
 [-8.01828442e-01  3.77739279e-01  2.58826668e+00 ...  1.81501353e-01
  -9.50373581e-02  1.00000000e+00]
 [ 1.50217591e+00 -1.22822172e+00  2.96940079e-01 ... -4.77335933e-02
   2.51610419e-03  2.38800000e+02]
 ...
 [ 1.13241622e+00 -3.04147959e+00 -5.57578299e-01 ...  5.24909234e-03
   5.00017408e-02  5.06640000e+02]
 [-8.81068584e-01  8.17336277e-02  2.43029614e+00 ... -2.26184111e-02
   9.98425163e-02  1.23980000e+02]
 [ 8.72839078e-01 -1.04666626e+00  5.49059518e-01 ... -5.61891006e-02
   3.31845223e-02  1.91720000e+02]]
[0 0 0 ... 0 0 0]
```

In [68]:
```
x = np.array(validation)[:, 0]
y1 = np.array(validation)[:, 1]
y2 = np.array(validation)[:, 2]
y3 = np.array(validation)[:, 3]
plt.plot(x, y1)
plt.title("Recall")
plt.xscale('log')
plt.show()
plt.plot(x, y2)
plt.title("Precision")
plt.xscale('log')
plt.show()
plt.plot(x, y3)
plt.title("F1 score")
plt.xscale('log')
plt.show()
```


Recall
1.04
1.02
1.00
0.98
0.96
10^{-15}
10^{-14}
10^{-13}
10^{-12}


Precision
0.0030
0.0028
0.0026
0.0024
0.0022
0.0020
0.0018
0.0016
10^{-15}
10^{-14}
10^{-13}
10^{-12}



Classification Using Neural Networks

In [69]:
```
import tensorflow
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
C:\Users\Piyush\anaconda3\lib\site-packages\tensorflow\python\framework\dtype
s.py:516: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorflow\python\framework\dtype
s.py:517: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorflow\python\framework\dtype
s.py:518: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorflow\python\framework\dtype
s.py:519: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorflow\python\framework\dtype
s.py:520: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorflow\python\framework\dtype
s.py:525: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is
deprecated; in a future version of numpy, it will be understood as (type,
(1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stu
b\dtypes.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stu
b\dtypes.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stu
b\dtypes.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stu
b\dtypes.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stu
b\dtypes.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of
type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Piyush\anaconda3\lib\site-packages\tensorboard\compat\tensorflow_stu
b\dtypes.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of
```

```
type is deprecated; in a future version of numpy, it will be understood as (t
ype, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
Using TensorFlow backend.
```

In [70]:
```
X_train.shape
```

Out[70]:
```
(38163, 28)
```

In [71]:
```
# Building our model with 2 hidden layers
model = Sequential()
model.add(Dense(32,kernel_initializer = 'he_uniform', input_dim=X_train.shape[
1], activation='relu'))
model.add(Dense(32, kernel_initializer = 'he_uniform', activation='relu'))
model.add(Dense(1, kernel_initializer = 'glorot_uniform',activation='sigmoid'
))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
# compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=20, batch_size=10)
```

```
WARNING:tensorflow:From C:\Users\Piyush\anaconda3\lib\site-packages\tensorflo
w\python\ops\nn_impl.py:180: add_dispatch_support.<locals>.wrapper (from tens
orflow.python.ops.array_ops) is deprecated and will be removed in a future ve
rsion.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\Piyush\anaconda3\lib\site-packages\keras\bac
kend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. P
lease use tf.compat.v1.global_variables instead.

Epoch 1/50
38163/38163 [==============================] - 12s 325us/step - loss: 0.1341
- accuracy: 0.9926
Epoch 2/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0384
- accuracy: 0.9986
Epoch 3/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0407
- accuracy: 0.9987
Epoch 4/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0206
- accuracy: 0.9990
Epoch 5/50
38163/38163 [==============================] - 12s 308us/step - loss: 0.0203
- accuracy: 0.9988
Epoch 6/50
38163/38163 [==============================] - 11s 279us/step - loss: 0.0144
- accuracy: 0.9988
Epoch 7/50
38163/38163 [==============================] - 12s 301us/step - loss: 0.0159
- accuracy: 0.9991
Epoch 8/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0095
- accuracy: 0.9991
Epoch 9/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0099
- accuracy: 0.9993
Epoch 10/50
38163/38163 [==============================] - 11s 296us/step - loss: 0.0104
- accuracy: 0.9991
Epoch 11/50
38163/38163 [==============================] - 10s 250us/step - loss: 0.0085
- accuracy: 0.9991
Epoch 12/50
38163/38163 [==============================] - 12s 315us/step - loss: 0.0108
- accuracy: 0.9991
Epoch 13/50
38163/38163 [==============================] - 12s 320us/step - loss: 0.0056
- accuracy: 0.9993
Epoch 14/50
38163/38163 [==============================] - 12s 319us/step - loss: 0.0059
- accuracy: 0.9994
Epoch 15/50
38163/38163 [==============================] - 11s 295us/step - loss: 0.0059
- accuracy: 0.9995
Epoch 16/50
38163/38163 [==============================] - 12s 304us/step - loss: 0.0076
```

```
- accuracy: 0.9994
Epoch 17/50
38163/38163 [==============================] - 12s 304us/step - loss: 0.0052
- accuracy: 0.9994
Epoch 18/50
38163/38163 [==============================] - 12s 306us/step - loss: 0.0058
- accuracy: 0.9994
Epoch 19/50
38163/38163 [==============================] - 12s 304us/step - loss: 0.0042
- accuracy: 0.9994
Epoch 20/50
38163/38163 [==============================] - 12s 305us/step - loss: 0.0060
- accuracy: 0.9994
Epoch 21/50
38163/38163 [==============================] - 10s 273us/step - loss: 0.0044
- accuracy: 0.9995
Epoch 22/50
38163/38163 [==============================] - 10s 269us/step - loss: 0.0042
- accuracy: 0.9996
Epoch 23/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0044
- accuracy: 0.9994
Epoch 24/50
38163/38163 [==============================] - 11s 293us/step - loss: 0.0048
- accuracy: 0.9995
Epoch 25/50
38163/38163 [==============================] - 11s 295us/step - loss: 0.0054
- accuracy: 0.9995
Epoch 26/50
38163/38163 [==============================] - 11s 289us/step - loss: 0.0056
- accuracy: 0.9995
Epoch 27/50
38163/38163 [==============================] - 12s 303us/step - loss: 0.0079
- accuracy: 0.9994
Epoch 28/50
38163/38163 [==============================] - 11s 292us/step - loss: 0.0038
- accuracy: 0.9994
Epoch 29/50
38163/38163 [==============================] - 12s 307us/step - loss: 0.0061
- accuracy: 0.9994
Epoch 30/50
38163/38163 [==============================] - 12s 304us/step - loss: 0.0051
- accuracy: 0.9995
Epoch 31/50
38163/38163 [==============================] - 12s 302us/step - loss: 0.0030
- accuracy: 0.9996
Epoch 32/50
38163/38163 [==============================] - 11s 300us/step - loss: 0.0032
- accuracy: 0.9996
Epoch 33/50
38163/38163 [==============================] - 8s 207us/step - loss: 0.0030 -
accuracy: 0.9996
Epoch 34/50
38163/38163 [==============================] - 6s 165us/step - loss: 0.0042 -
accuracy: 0.9996
Epoch 35/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0032 -
```

```
accuracy: 0.9997
Epoch 36/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0095 -
accuracy: 0.9995
Epoch 37/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0044 -
accuracy: 0.9995
Epoch 38/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0048 -
accuracy: 0.9996
Epoch 39/50
38163/38163 [==============================] - 6s 163us/step - loss: 0.0054 -
accuracy: 0.9996
Epoch 40/50
38163/38163 [==============================] - 6s 163us/step - loss: 0.0071 -
accuracy: 0.9995
Epoch 41/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0057 -
accuracy: 0.9996
Epoch 42/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0048 -
accuracy: 0.9996
Epoch 43/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0041 -
accuracy: 0.9996
Epoch 44/50
38163/38163 [==============================] - 6s 165us/step - loss: 0.0035 -
accuracy: 0.9996
Epoch 45/50
38163/38163 [==============================] - 6s 165us/step - loss: 0.0034 -
accuracy: 0.9995
Epoch 46/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0052 -
accuracy: 0.9996
Epoch 47/50
38163/38163 [==============================] - 6s 163us/step - loss: 0.0035 -
accuracy: 0.9995
Epoch 48/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0045 -
accuracy: 0.9997
Epoch 49/50
38163/38163 [==============================] - 6s 164us/step - loss: 0.0021 -
accuracy: 0.9997
Epoch 50/50
38163/38163 [==============================] - 6s 163us/step - loss: 0.0040 -
accuracy: 0.9996
```

Out[71]:
```
<keras.callbacks.callbacks.History at 0x1f0c934e7c8>
```

In [72]:
```
# evaluate the keras model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy*100))
```

```
18798/18798 [==============================] - 1s 28us/step
Accuracy: 99.94
```

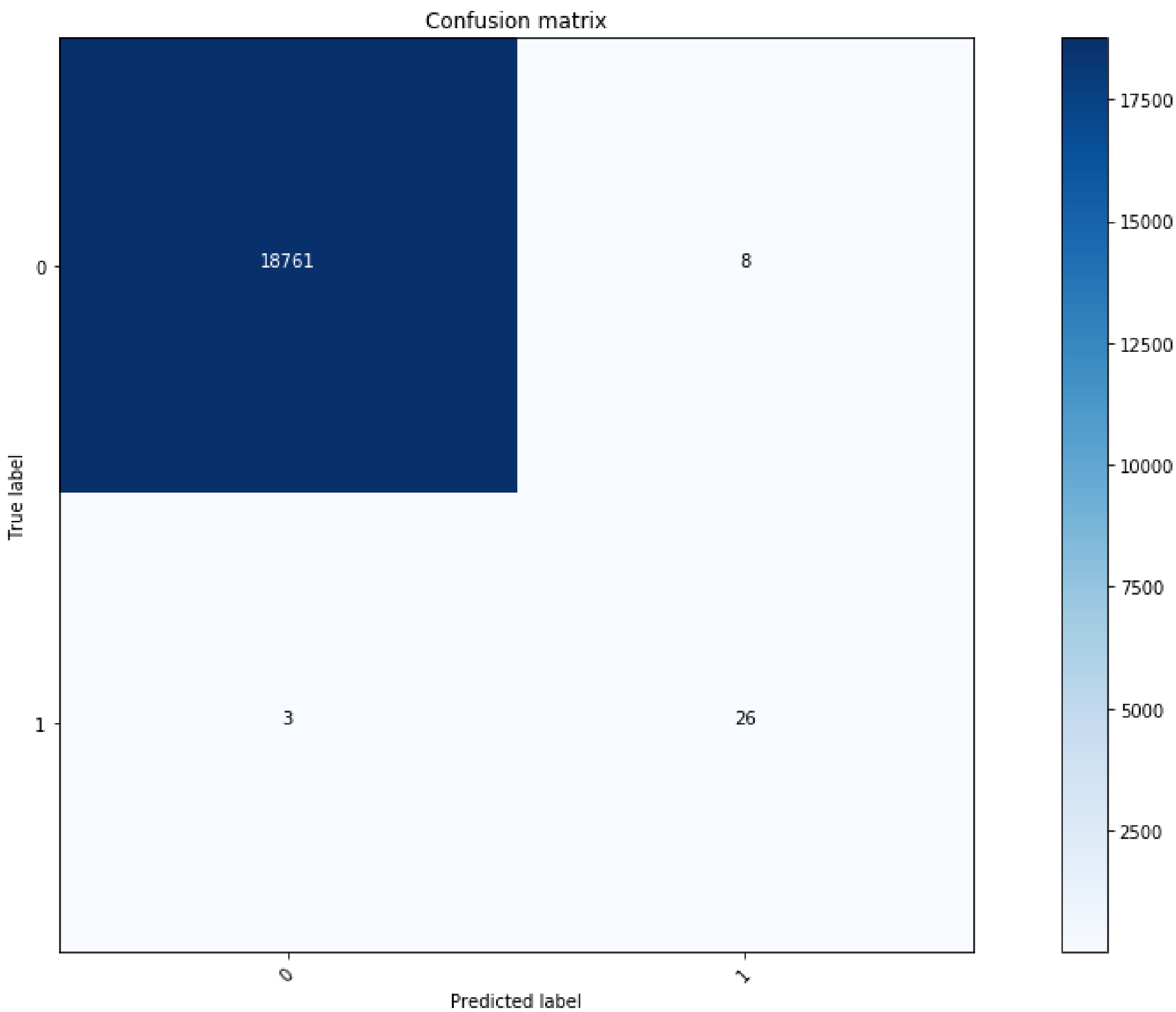
In [73]:
```
# make probability predictions with the model
predictions = model.predict(X_test)
# round predictions
rounded = [round(x[0]) for x in predictions]
rounded = np.array(rounded)
rounded.shape
```

Out[73]: (18798,)

In [74]:
```
# make class predictions with the model
predictions = model.predict_classes(X_test)
predictions.shape
```

Out[74]: (18798, 1)

In [75]:
```
cm = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cm,classes)
```



In [76]:
```
print('Total fraudulent transactions detected: ' + str(cm[1][1]) + ' / ' + str
(cm[1][1]+cm[1][0]))
print('Total non-fraudulent transactions detected: ' + str(cm[0][0]) + ' / ' +
str(cm[0][1]+cm[0][0]))

print('Probability to detect a fraudulent transaction: ' + str(cm[1][1]/(cm[1]
[1]+cm[1][0])))
print('Probability to detect a non-fraudulent transaction: ' + str(cm[0][0]/(c
m[0][1]+cm[0][0])))

print("Accuracy of the Neural Network model : "+str(100*(cm[0][0]+cm[1][1]) /
(sum(cm[0]) + sum(cm[1]))) + "%")
print("ROC_AUC_score : %.6f" % (roc_auc_score(y_test, predictions)))
```

```
Total fraudulent transactions detected: 26 / 29
Total non-fraudulent transactions detected: 18761 / 18769
Probability to detect a fraudulent transaction: 0.896551724137931
Probability to detect a non-fraudulent transaction: 0.9995737652512121
Accuracy of the Neural Network model : 99.94148313650388%
ROC_AUC_score : 0.948063
```

In [77]:
```
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18769
           1       0.76      0.90      0.83        29

    accuracy                           1.00     18798
   macro avg       0.88      0.95      0.91     18798
weighted avg       1.00      1.00      1.00     18798
```

In [ ]: