In [1]:

```
import pandas as pd
import numpy as np
import keras

np.random.seed(2)
```

```
Using TensorFlow backend.
```

In [2]:

```
# Read the dataset

data = pd.read_csv("creditcard.csv")
```

In [3]:

```
# To view the top part of the credit card dataset we are working with
data.head()
```

Out[3]:

	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

5 rows × 31 columns

Data Preprocessing: This prepares the data before we proceed with the building of the machine learning models.

In [4]:

```
# This drops the amount data and replaced it with the normalizedAmount,
# Since the amount column is not in line with the anonimised features
from sklearn.preprocessing import StandardScaler
data['normalizedAmount'] = StandardScaler().fit_transform(data['Amount'].values.reshape
(-1,1))
data = data.drop(['Amount'], axis = 1)
```

In [5]:

```
# This drops the Time variable as well
data = data.drop(['Time'], axis =1)
```

In [6]:

```
# To view the new data
data.head()
```

Out[6]:

	V1	V2	V3	V4	V5	V6	V7	V
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.27053

5 rows × 30 columns

In [7]:

```
## Splitting the dataset into dependent and independent variables
X = data.iloc[:, data.columns != 'Class']
y = data.iloc[:, data.columns == 'Class']
```

In [8]:

```
# Splitting the dataset into training and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state =
 0)
```

In [9]:

```
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

In [10]:

```
# To view the dimension of the training and test dataset
X_train.shape
```

Out[10]:

```
(199364, 29)
```

In [11]:

```
X_test.shape
```

Out[11]:

```
(85443, 29)
```

Deep Neural Network Using Keras

In [12]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

In [13]:

```
model = Sequential([
    Dense(units = 16, input_dim = 29, activation = "relu"),
    Dense(units = 24, activation = "relu"),
    Dropout(0.5),
    Dense(20, activation = "relu"),
    Dense(24, activation = "relu"),
    Dense(1,activation = "sigmoid"),
])
```

In [14]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                480
_________________________________________________________________
dense_2 (Dense)              (None, 24)                408
_________________________________________________________________
dropout_1 (Dropout)          (None, 24)                0
_________________________________________________________________
dense_3 (Dense)              (None, 20)                500
_________________________________________________________________
dense_4 (Dense)              (None, 24)                504
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 25
=================================================================
Total params: 1,917
Trainable params: 1,917
Non-trainable params: 0
_________________________________________________________________
```

Training the model

In [15]:

```
model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ['accuracy'])
model.fit(X_train,y_train,batch_size = 15, epochs = 5)
```

```
Epoch 1/5
199364/199364 [==============================] - 22s 111us/step - loss: 0.
0099 - acc: 0.9979
Epoch 2/5
199364/199364 [==============================] - 20s 99us/step - loss: 0.0
039 - acc: 0.9994
Epoch 3/5
199364/199364 [==============================] - 21s 107us/step - loss: 0.
0035 - acc: 0.9993
Epoch 4/5
199364/199364 [==============================] - 22s 108us/step - loss: 0.
0034 - acc: 0.9994
Epoch 5/5
199364/199364 [==============================] - 23s 114us/step - loss: 0.
0032 - acc: 0.9993
```

Out[15]:

```
<keras.callbacks.History at 0x24490d70668>
```

In [16]:

```
# To get the score of the accuracy of the test dataset
score = model.evaluate(X_test,y_test)
```

```
85443/85443 [==============================] - 2s 21us/step
```

In [17]:

```
print(score)
```

```
[0.004360657058588501, 0.9993562960102056]
```

In [35]:

```
# To plot the confusion matrix

import itertools
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.metrics import confusion_matrix

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

    print("the recall for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[
1,0]))
    print("TP",cnf_matrix[1,1,]) # no of fraud transaction which are predicted fraud
    print("TN",cnf_matrix[0,0]) # no. of normal transaction which are predited normal
    print("FP",cnf_matrix[0,1]) # no of normal transaction which are predicted fraud
    print("FN",cnf_matrix[1,0]) # no of fraud Transaction which are predicted normal
```

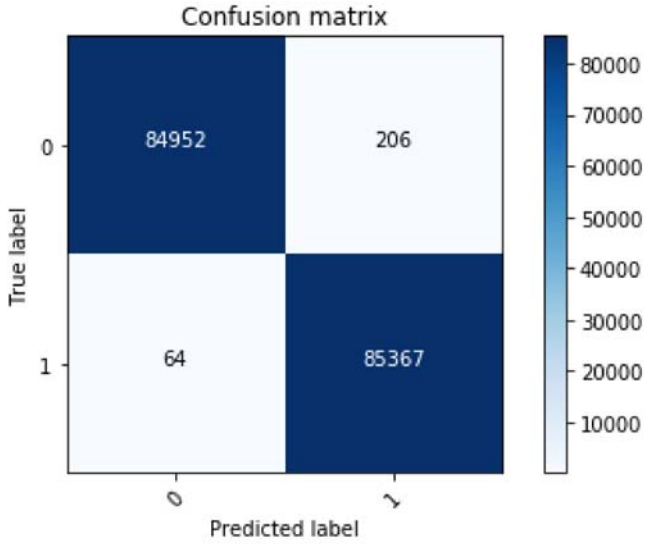
In [36]:

```
y_pred = model.predict(X_test)
y_test = pd.DataFrame(y_test)
cnf_matrix = confusion_matrix(y_test, y_pred.round())
```

In [37]:

```
plot_confusion_matrix(cnf_matrix, classes = [0,1])
```

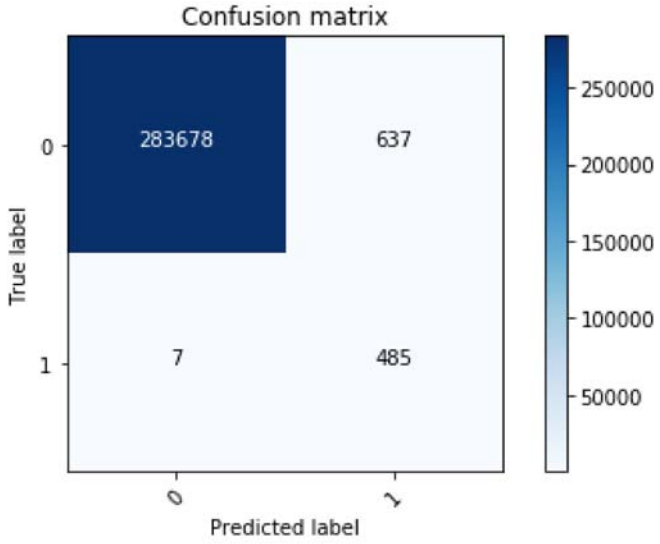
```
Confusion matrix, without normalization
[[84952   206]
 [   64 85367]]
the recall for this model is : 0.9992508574170968
TP 85367
TN 84952
FP 206
FN 64
```



In [38]:

```
y_pred = model.predict(X)
y_expected = pd.DataFrame(y)
cnf_matrix = confusion_matrix(y_expected, y_pred.round())
plot_confusion_matrix(cnf_matrix, classes = [0,1])
```

```
Confusion matrix, without normalization
[[283678    637]
 [     7    485]]
the recall for this model is : 0.9857723577235772
TP 485
TN 283678
FP 637
FN 7
```



SMOTE: Synthetic Minority Oversampling Technique

In [23]:

```
from imblearn.over_sampling import SMOTE
X_resample, y_resample = SMOTE().fit_sample(X,y.values.ravel())
```

In [24]:

```
X_resample = pd.DataFrame(X_resample)
y_resample = pd.DataFrame(y_resample)
```

In [25]:

```
X_train,X_test,y_train,y_test = train_test_split(X_resample,y_resample,test_size = 0.3)
```

In [26]:

```
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
```

In [27]:

```
model.compile(optimizer = "adam", loss = "binary_crossentropy",metrics = ["accuracy"])
model.fit(X_train,y_train,batch_size = 15, epochs =5)
```

```
Epoch 1/5
398041/398041 [==============================] - 40s 100us/step - loss: 0.
0359 - acc: 0.9878
Epoch 2/5
398041/398041 [==============================] - 40s 100us/step - loss: 0.
0142 - acc: 0.9962
Epoch 3/5
398041/398041 [==============================] - 40s 101us/step - loss: 0.
0119 - acc: 0.9970
Epoch 4/5
398041/398041 [==============================] - 41s 103us/step - loss: 0.
0098 - acc: 0.9976
Epoch 5/5
398041/398041 [==============================] - 41s 103us/step - loss: 0.
0089 - acc: 0.9978
```

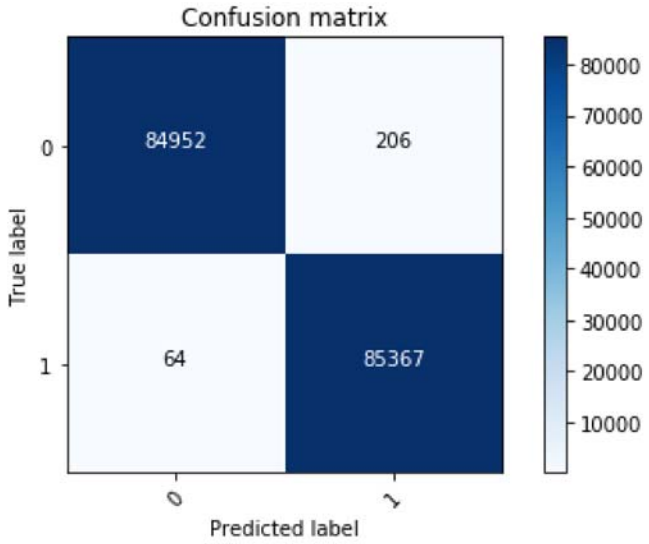
Out[27]:

```
<keras.callbacks.History at 0x24494337cf8>
```

In [28]:

```
y_pred = model.predict(X_test)
y_expected = pd.DataFrame(y_test)
cnf_matrix = confusion_matrix(y_expected,y_pred.round())
plot_confusion_matrix(cnf_matrix,classes = [0,1])
```

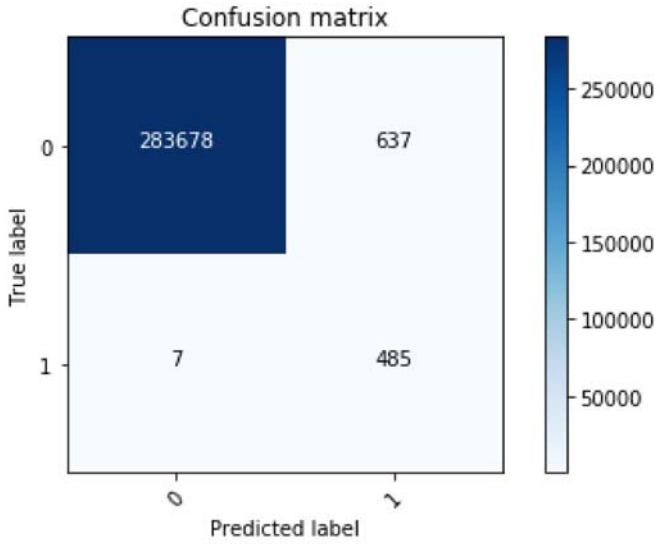
```
Confusion matrix, without normalization
[[84952   206]
 [   64 85367]]
```



In [29]:

```
y_pred = model.predict(X)
y_expected = pd.DataFrame(y)
cnf_matrix = confusion_matrix(y_expected,y_pred.round())
plot_confusion_matrix(cnf_matrix,classes = [0,1])
```

```
Confusion matrix, without normalization
[[283678    637]
 [     7    485]]
```



In [34]:

```
print("the recall for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[1,0
]))
print("TP",cnf_matrix[1,1,]) # no of fraud transaction which are predicted fraud
print("TN",cnf_matrix[0,0]) # no. of normal transaction which are predited normal
print("FP",cnf_matrix[0,1]) # no of normal transaction which are predicted fraud
print("FN",cnf_matrix[1,0]) # no of fraud Transaction which are predicted normal
```

```
the recall for this model is : 0.9857723577235772
TP 485
TN 283678
FP 637
FN 7
```