In [26]:

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

np.random.seed(2)
```

In [27]:

```
# Read the dataset

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

In [28]:

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

Out[28]:

	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 [29]:

```
# 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 [30]:

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

In [31]:

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

In [32]:

```
# 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 [33]:

```
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 [34]:

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

Out[34]:

```
(199364, 29)
```

In [35]:

```
X_test.shape
```

Out[35]:

```
(85443, 29)
```

Random Forest

In [36]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [37]:

```
random_forest = RandomForestClassifier(n_estimators = 100)
```

In [38]:

```
random_forest.fit(X_train,y_train.ravel())
```

Out[38]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [39]:

```
y_pred =random_forest.predict(X_test)
```

In [40]:

```
# 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
```

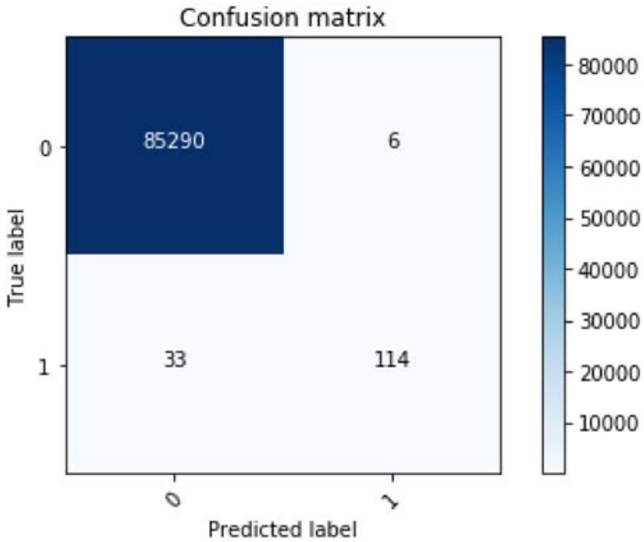
In [41]:

```
cnf_matrix = confusion_matrix(y_test, y_pred.round())
```

In [42]:

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

```
Confusion matrix, without normalization
[[85290     6]
 [   33   114]]
the recall for this model is : 0.7755102040816326
TP 114
TN 85290
FP 6
FN 33
```



In [43]:

```
y_pred =random_forest.predict(X)
```

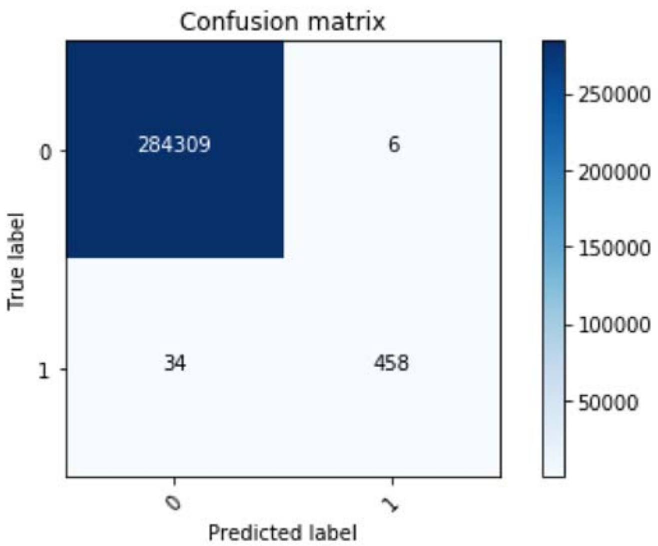
In [44]:

```
cnf_matrix = confusion_matrix(y, y_pred.round())
```

In [45]:

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

```
Confusion matrix, without normalization
[[284309      6]
 [    34    458]]
the recall for this model is : 0.9308943089430894
TP 458
TN 284309
FP 6
FN 34
```



In [46]:

```
random_forest.score(X_test,y_test)
```

Out[46]:

```
0.9995435553526912
```