In [1]:

```
#Importing required librairies

import pandas as pd
import numpy as np

#importing svm from scikit learn
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn import svm

import itertools

#import matplotlib library to plot the charts
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

#this is the library for statistic data visualization
import seaborn

%matplotlib inline
```

In [2]:

```
data = pd.read_csv("C:\\Users\\asims\\Documents\\Machine Learning and Deep Learning\\cr
```

In [3]:

```
dFrame = pd.DataFrame(data) #Dataframe to PandaDataFrame
dFrame.describe()
```

Out[3]:

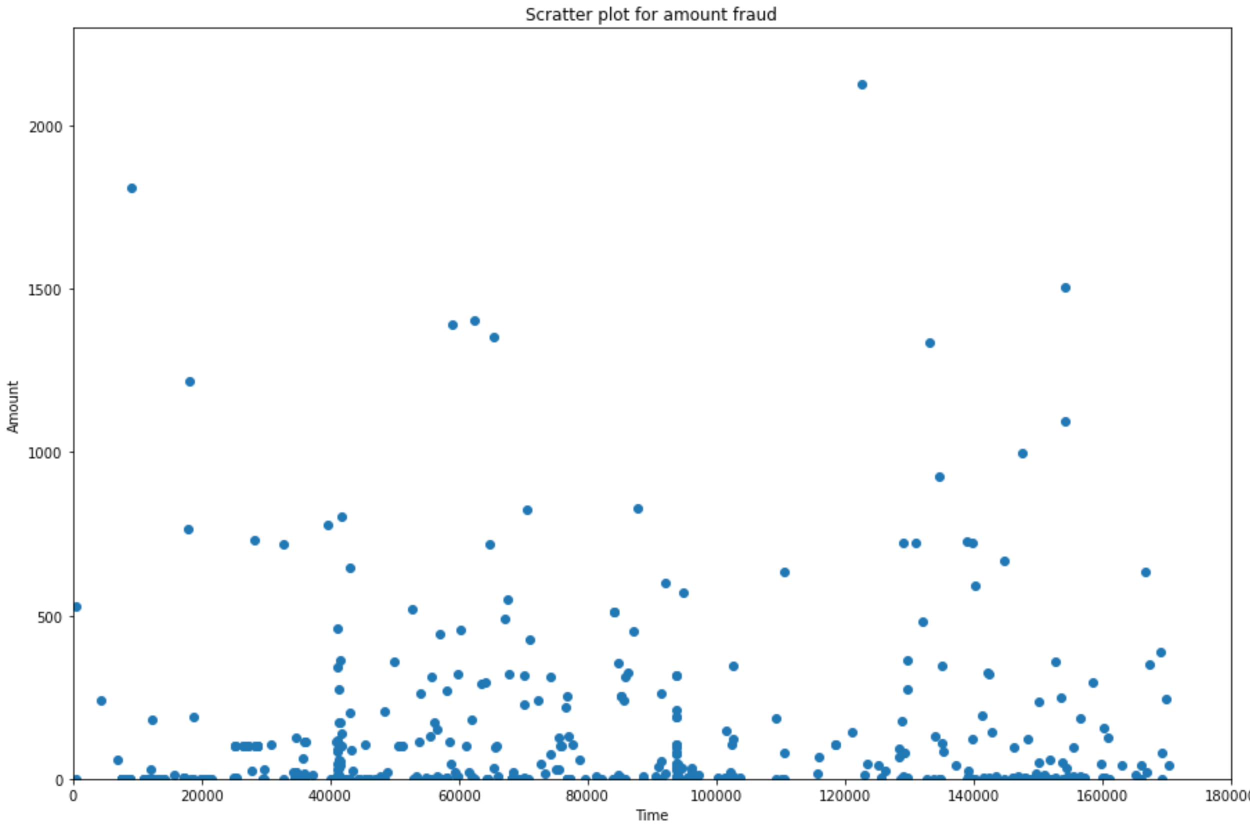
	Time	V1	V2	V3	V4	V5	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.84
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.0
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.33
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.61
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.6
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.7
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.9
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.33

8 rows × 31 columns

In [4]:

```
dFrame_fraud = dFrame[dFrame['Class'] == 1] #recovering fraud data
plt.figure(figsize=(15,10)) #assigning figuresize
plt.scatter(dFrame_fraud['Time'], dFrame_fraud['Amount']) #showing fraud amount with re
plt.title('Scatter plot for amount fraud')
plt.xlabel('Time')
plt.ylabel('Amount')
plt.xlim([0,180000])
```

```
plt.ylim([0,2300])
plt.show()
```



In [6]:

```
biggerFraud = dFrame_fraud[dFrame_fraud['Amount'] > 1000].shape[0] #lets do the recover
print('There are only '+ str(biggerFraud) + ' frauds in total that were bigger than 100
```

```
There are only 9 frauds in total that were bigger than 1000 among 492 frauds
```

In [9]:

```
numOfFrauds = len(data[data.Class == 1])
numOfNoFrauds= len(data[data.Class == 0])
print('There are ' + str(numOfFrauds) + ' frauds in the original dataset, even though t
print("\nNow the Accuracy of classifier: "+ str((284315-492)/284315))
```

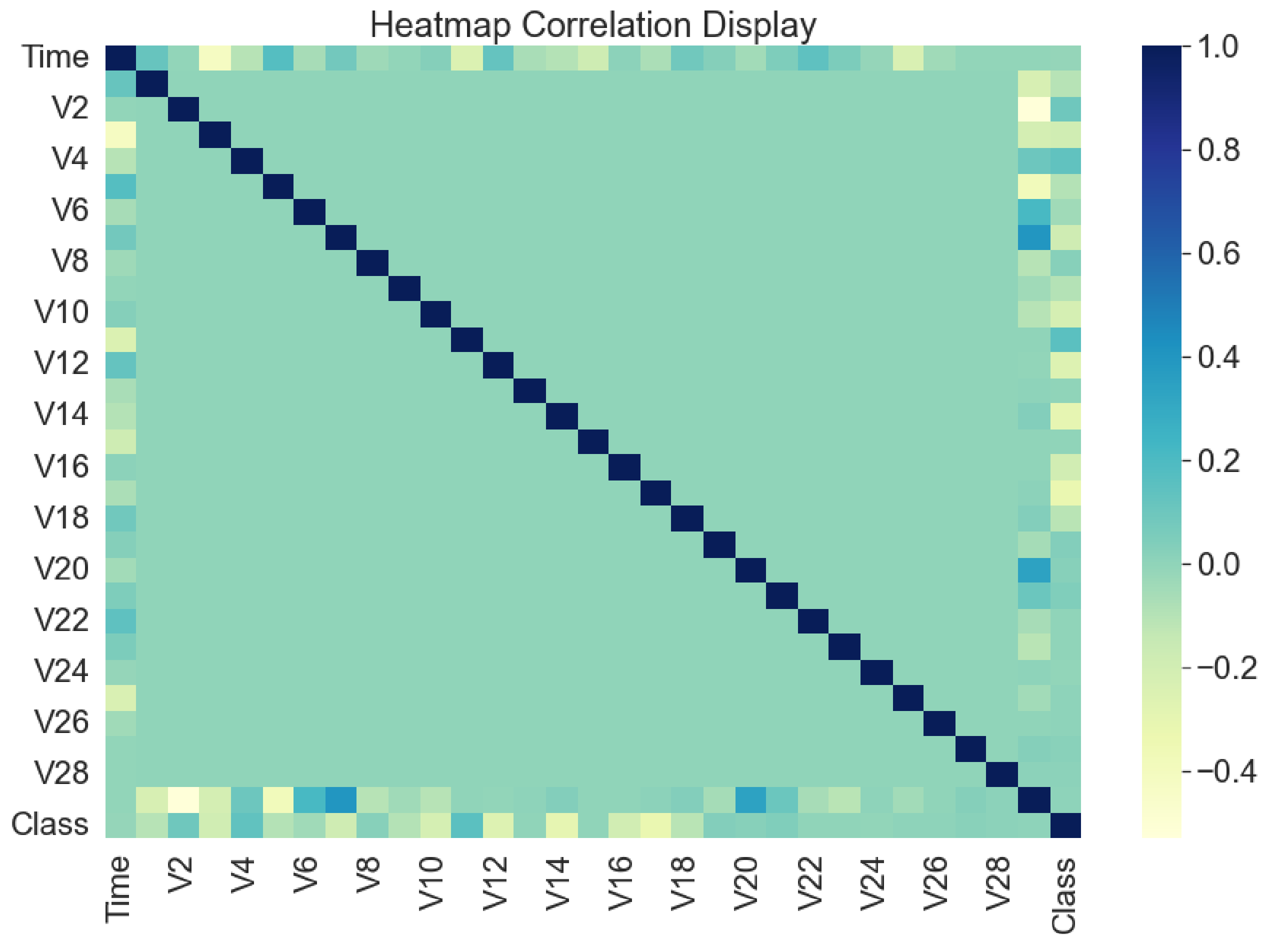
```
There are 492 frauds in the original dataset, even though there are 284315 frauds.

Now the Accuracy of classifier: 0.998269524998681
```

In [10]:

```
dFrame_corr = dFrame.corr() #Correlation coefficients calculation in pairs with the def
```

In [14]:

```
plt.figure(figsize=(15,10)) #setting the figure size
seaborn.heatmap(dFrame_corr, cmap="YlGnBu") #heatmap correlation display
seaborn.set(font_scale=2,style='white')
plt.title('Heatmap Correlation Display')
plt.show()
```



In [15]:

```
rank = dFrame_corr['Class'] #Retrieving correlation coefficients as w.r.t feature class
dFrame_rank = pd.DataFrame(rank)
dFrame_rank = np.abs(dFrame_rank).sort_values(by='Class',ascending=False) #ranking abso
dFrame_rank.dropna(inplace=True) #removing the missing data but not number
```

In [16]:

```
#we have to divide data in two groups- train dataset & test dataset

#Now build train dataset
dFrame_train_all = dFrame[0:150000] # start to separate original dataset into frauds &
dFrameTrainds_1 = dFrame_train_all[dFrame_train_all['Class'] == 1]
dFrameTrainds_0 = dFrame_train_all[dFrame_train_all['Class'] == 0]
print('In this dataset, we have ' + str(len(dFrameTrainds_1)) +" frauds so we need to t

dFrame_sample=dFrameTrainds_0.sample(300)
dFrame_train = dFrameTrainds_1.append(dFrame_sample) #collecting frauds along with no f
dFrame_train = dFrame_train.sample(frac=1) #Mixing the dataset
```

```
In this dataset, we have 293 frauds so we need to take a similar number of non-fraud
```

In [19]:

```
XTrainD = dFrame_train.drop(['Time', 'Class'],axis=1) # We drop the features Time
YTrainD = dFrame_train['Class'] #class as label
XTrainD = np.asarray(XTrainD)
YTrainD = np.asarray(YTrainD)
```

In [20]:

```
#To check if whether the model learn correctly with respect to dataset
TestAll_dFrame = dFrame[150000:]
```

```
TestAll_X = TestAll_dFrame.drop(['Time', 'Class'],axis=1)
TestAll_Y = TestAll_dFrame['Class']
TestAll_X = np.asarray(TestAll_X)
TestAll_Y = np.asarray(TestAll_Y)
```

In [21]:

```
XTrainD_rank = dFrame_train[dFrame_rank.index[1:11]] # 1 to 11 takes only 10 features
XTrainD_rank = np.asarray(XTrainD_rank)
```

In [22]:

```
#To check if whether the model learn correctly with respect to dataset
TestAll_X_rank = TestAll_dFrame[dFrame_rank.index[1:11]]
TestAll_X_rank = np.asarray(TestAll_X_rank)
TestAll_Y = np.asarray(TestAll_Y)
```

In [23]:

```
class_names=np.array(['0','1']) #FYI Class = 1 is fraud and Class = 0 is no fraud, usin
```

In [25]:

```
# Create Plot Confusion Matirx Method
def plot_Cmrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    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'
    threshold = cm.max() / 2.0
    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] > threshold else "black")

    plt.tight_layout()
    plt.ylabel('True-Label')
    plt.xlabel('Predicted-Label')
```

In [26]:

```
classifier_SVM = svm.SVC(kernel='linear') # We set a SVM classifier, the default SVM Cl
classifier_SVM.fit(XTrainD, YTrainD) # Then we train our model, with our balanced data
```

Out[26]:

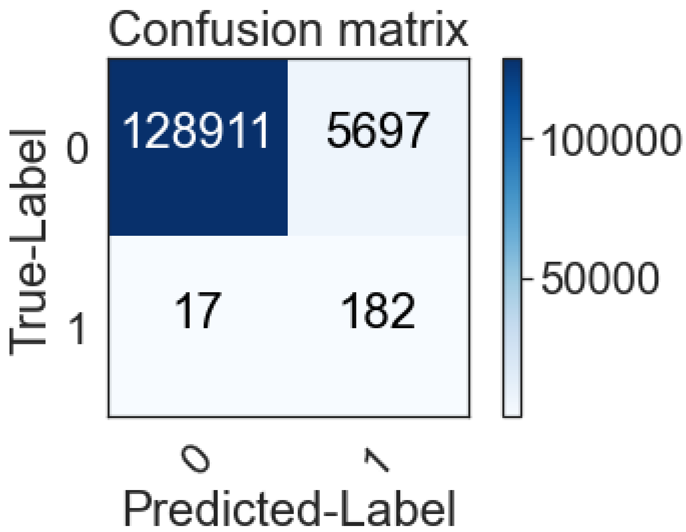
```
SVC(kernel='linear')
```

In [27]:

```
prediction_SVM_all = classifier_SVM.predict(TestAll_X) #And finally, we predict our dat
```

In [28]:

```
cm = confusion_matrix(TestAll_Y, prediction_SVM_all)
plot_Cmrix(cm,class_names)
```



In [29]:

```
print('Criterion Result that we got is '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]
```

```
Criterion Result that we got is 0.9231809868662785
```

In [30]:

```
print('So We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) +
print('\nThe probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("\nAccuracy -> "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

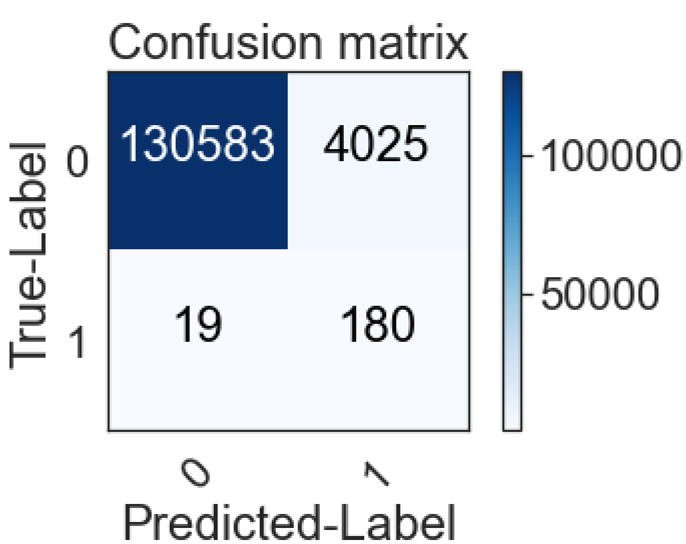
```
So We have detected 182 frauds / 199 total frauds.

The probability to detect a fraud is 0.914572864321608

Accuracy -> 0.9576134770449606
```

In [31]:

```
classifier_SVM.fit(XTrainD_rank, YTrainD) #here we train the model with balanced data t
prediction_SVM = classifier_SVM.predict(TestAll_X_rank) # Now predict data test.
```

In [32]:

```
cm = confusion_matrix(TestAll_Y, prediction_SVM)
plot_Cmrix(cm,class_names)
```



In [33]:

```
print('Criterion Result that we got is '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]
```

```
Criterion Result that we got is 0.917618402008783
```

In [34]:

```
print('So We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) +
print('\nThe probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("\nAccuracy -> "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

```
So We have detected 180 frauds / 199 total frauds.

The probability to detect a fraud is 0.9045226130653267

Accuracy -> 0.9700015577826078
```

In [35]:

```
classifier_SVM_b = svm.SVC(kernel='linear',class_weight={0:0.6, 1:0.4})
```

In [36]:

```
classifier_SVM_b.fit(XTrainD, YTrainD) # Then we train our model, with our balanced dat
```

Out[36]:

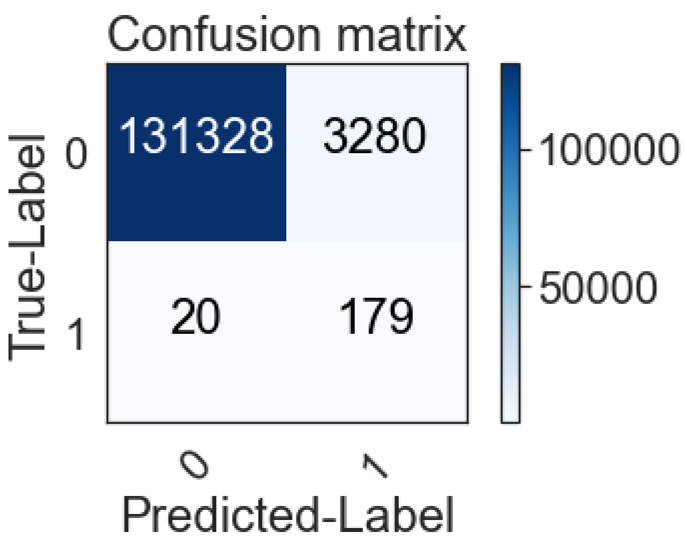
```
SVC(class_weight={0: 0.6, 1: 0.4}, kernel='linear')
```

In [37]:

```
prediction_SVM_b_all = classifier_SVM_b.predict(TestAll_X) #We predict all the data set
```

In [38]:

```
cm = confusion_matrix(TestAll_Y, prediction_SVM_b_all)
plot_Cmrix(cm,class_names)
```



In [39]:

```
print('Criterion Result that we got is '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]
```

```
Criterion Result that we got is 0.9147021017540318
```

In [40]:

```
print('So We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) +
print('\nThe probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("\nAccuracy -> "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

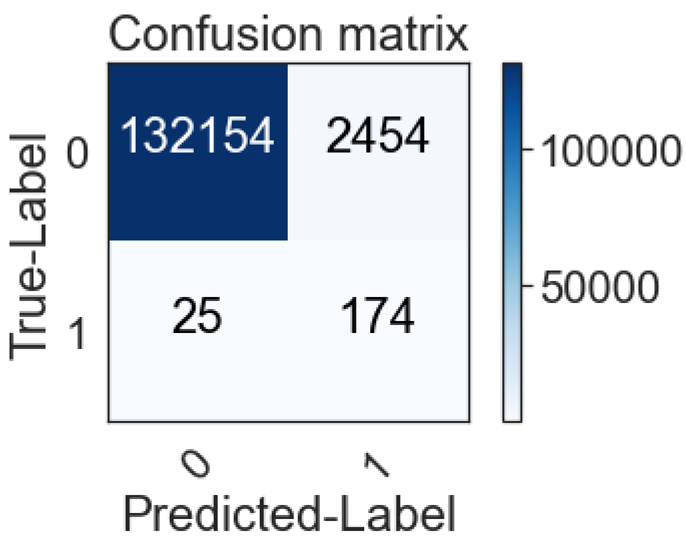
```
So We have detected 179 frauds / 199 total frauds.

The probability to detect a fraud is 0.8994974874371859

Accuracy -> 0.9755205590214158
```

In [41]:

```
classifier_SVM_b.fit(XTrainD_rank, YTrainD) # Now train the model with balanced train d
prediction_SVM = classifier_SVM_b.predict(TestAll_X_rank) #Now predict data test.
```

In [42]:

```
cm = confusion_matrix(TestAll_Y, prediction_SVM)
plot_Cmrix(cm,class_names)
```



In [43]:

```
print('Criterion Result that we got is '
      + str( ( (cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1])) + 4 * cm[1][1]/(cm[1][0]
```

```
Criterion Result that we got is 0.8958196368804641
```

In [44]:

```
print('So We have detected ' + str(cm[1][1]) + ' frauds / ' + str(cm[1][1]+cm[1][0]) +
print('\n The probability to detect a fraud is ' + str(cm[1][1]/(cm[1][1]+cm[1][0])))
print("\nAccuracy -> "+str((cm[0][0]+cm[1][1]) / (sum(cm[0]) + sum(cm[1]))))
```

```
So We have detected 174 frauds / 199 total frauds.

 The probability to detect a fraud is 0.8743718592964824

Accuracy -> 0.9816107472163909
```

In [ ]: