Loading the packages

In [1]:

```
pip install plotly==4.7.1
```

```
Requirement already satisfied: plotly==4.7.1 in c:\users\welcome\anaconda3\lib\site-packages
(4.7.1)
Requirement already satisfied: six in c:\users\welcome\anaconda3\lib\site-packages (from
plotly==4.7.1) (1.14.0)
Requirement already satisfied: retrying>=1.3.3 in c:\users\welcome\anaconda3\lib\site-packages
(from plotly==4.7.1) (1.3.3)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
pip install catboost
```

```
Requirement already satisfied: catboost in c:\users\welcome\anaconda3\lib\site-packages (0.23.1)
Requirement already satisfied: graphviz in c:\users\welcome\anaconda3\lib\site-packages (from
catboost) (0.14)
Requirement already satisfied: pandas>=0.24.0 in c:\users\welcome\anaconda3\lib\site-packages
(from catboost) (1.0.1)
Requirement already satisfied: plotly in c:\users\welcome\anaconda3\lib\site-packages (from
catboost) (4.7.1)
Requirement already satisfied: matplotlib in c:\users\welcome\anaconda3\lib\site-packages (from
catboost) (3.1.3)
Requirement already satisfied: scipy in c:\users\welcome\anaconda3\lib\site-packages (from
catboost) (1.4.1)
Requirement already satisfied: numpy>=1.16.0 in c:\users\welcome\anaconda3\lib\site-packages (from
catboost) (1.18.1)
Requirement already satisfied: six in c:\users\welcome\anaconda3\lib\site-packages (from catboost)
(1.14.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\welcome\anaconda3\lib\site-packages (from
pandas>=0.24.0->catboost) (2019.3)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\welcome\anaconda3\lib\site-
packages (from pandas>=0.24.0->catboost) (2.8.1)
Requirement already satisfied: retrying>=1.3.3 in c:\users\welcome\anaconda3\lib\site-packages
(from plotly->catboost) (1.3.3)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\welcome\anaconda3\lib\site-packages
(from matplotlib->catboost) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\welcome\anaconda3\lib\site-packages (from matplotlib->catboost) (2.4.6)
Requirement already satisfied: cycler>=0.10 in c:\users\welcome\anaconda3\lib\site-packages (from
matplotlib->catboost) (0.10.0)
Requirement already satisfied: setuptools in c:\users\welcome\anaconda3\lib\site-packages (from
kiwisolver>=1.0.1->matplotlib->catboost) (45.2.0.post20200210)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:

```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\welcome\anaconda3\lib\site-packages (2.3.1)
Requirement already satisfied: scikit-learn in c:\users\welcome\anaconda3\lib\site-packages (from
lightgbm) (0.22.1)
Requirement already satisfied: scipy in c:\users\welcome\anaconda3\lib\site-packages (from
lightgbm) (1.4.1)
Requirement already satisfied: numpy in c:\users\welcome\anaconda3\lib\site-packages (from
lightgbm) (1.18.1)
Requirement already satisfied: joblib>=0.11 in c:\users\welcome\anaconda3\lib\site-packages (from
scikit-learn->lightgbm) (0.14.1)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\welcome\anaconda3\lib\site-packages (1.1.0)
Requirement already satisfied: numpy in c:\users\welcome\anaconda3\lib\site-packages (from
xgboost) (1.18.1)
Requirement already satisfied: scipy in c:\users\welcome\anaconda3\lib\site-packages (from
xgboost) (1.4.1)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly import tools
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)


import gc
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from catboost import CatBoostClassifier
from sklearn import svm
import lightgbm as lgb
from lightgbm import LGBMClassifier
import xgboost as xgb

pd.set_option('display.max_columns', 100)


RFC_METRIC = 'gini'  #metric used for RandomForrestClassifier
NUM_ESTIMATORS = 100 #number of estimators used for RandomForrestClassifier
NO_JOBS = 4 #number of parallel jobs used for RandomForrestClassifier


#TRAIN/VALIDATION/TEST SPLIT
#VALIDATION
VALID_SIZE = 0.20 # simple validation using train_test_split
TEST_SIZE = 0.20 # test size using_train_test_split

#CROSS-VALIDATION
NUMBER_KFOLDS = 5 #number of KFolds for cross-validation



RANDOM_STATE = 2018

MAX_ROUNDS = 1000 #lgb iterations
EARLY_STOP = 50 #lgb early stop
OPT_ROUNDS = 1000  #To be adjusted based on best validation rounds
VERBOSE_EVAL = 50 #Print out metric result

IS_LOCAL = False

import os
```

Reading the data

In [7]:

```
dataset=pd.read_csv('creditcard.csv.zip')
dataset
```

Out[7]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	0.6
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.1
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.5
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.7
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.9
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	0.9
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	0.0

284807 rows × 31 columns

Checking the data

In [8]:

```
print("Credit Card Fraud Detection data -  rows:",dataset.shape[0]," columns:", dataset.shape[1])
```

```
Credit Card Fraud Detection data -  rows: 284807  columns: 31
```

Glimpse the data

In [9]:

```
dataset.head()
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	0.99
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.48
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.71
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.50
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.34

In [10]:

```
dataset.describe()
```

Out[10]:

	Time	V1	V2	V3	V4	V5	V6	V7	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+

Check the missing data

In [11]:

```
total = dataset.isnull().sum().sort_values(ascending = False)
percent = (dataset.isnull().sum()/dataset.isnull().count()*100).sort_values(ascending = False)
pd.concat([total, percent], axis=1, keys=['Total', 'Percent']).transpose()
```

Out[11]:

	Class	V14	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V15	Amount	V16	V17	V18	V19	V20	V21	V2
Total	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
Percent	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

Data unbalance

In [12]:

```
temp = dataset["Class"].value_counts()
df = pd.DataFrame({'Class': temp.index,'values': temp.values})

trace = go.Bar(
    x = df['Class'],y = df['values'],
    name="Credit Card Fraud Class - data unbalance (Not fraud = 0, Fraud = 1)",
    marker=dict(color="Red"),
    text=df['values']
)
data = [trace]
layout = dict(title = 'Credit Card Fraud Class - data unbalance (Not fraud = 0, Fraud = 1)',
          xaxis = dict(title = 'Class', showticklabels=True),
          yaxis = dict(title = 'Number of transactions'),
          hovermode = 'closest',width=600
         )
fig = dict(data=data, layout=layout)
iplot(fig, filename='class')
```

Data exploration:

Transactions in time

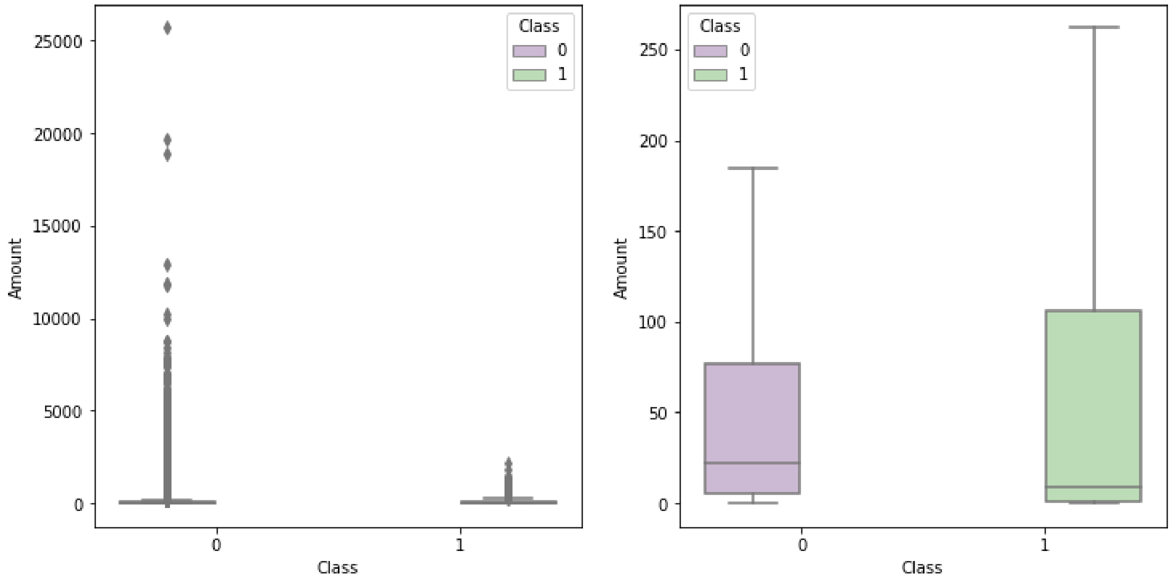
In [13]:

```
class_0 = dataset.loc[dataset['Class'] == 0]["Time"]
class_1 = dataset.loc[dataset['Class'] == 1]["Time"]
#plt.figure(figsize = (14,4))
#plt.title('Credit Card Transactions Time Density Plot')
#sns.set_color_codes("pastel")
#sns.distplot(class_0,kde=True,bins=480)
#sns.distplot(class_1,kde=True,bins=480)
#plt.show()
hist_data = [class_0, class_1]
group_labels = ['Not Fraud', 'Fraud']

fig = ff.create_distplot(hist_data, group_labels, show_hist=False, show_rug=False)
fig['layout'].update(title='Credit Card Transactions Time Density Plot', xaxis=dict(title='Time
[s]'))
iplot(fig, filename='dist_only')
```

Transactions amount

In [14]:

```
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))
s = sns.boxplot(ax = ax1, x="Class", y="Amount", hue="Class",data=dataset,
palette="PRGn",showfliers=True)
s = sns.boxplot(ax = ax2, x="Class", y="Amount", hue="Class",data=dataset,
palette="PRGn",showfliers=False)
plt.show();
```



In [15]:

```
tmp = dataset[['Amount','Class']].copy()
class_0 = tmp.loc[tmp['Class'] == 0]['Amount']
class_1 = tmp.loc[tmp['Class'] == 1]['Amount']
class_0.describe()
```

Out[15]:

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

In [16]:

```
class_1.describe()
```

Out[16]:

```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

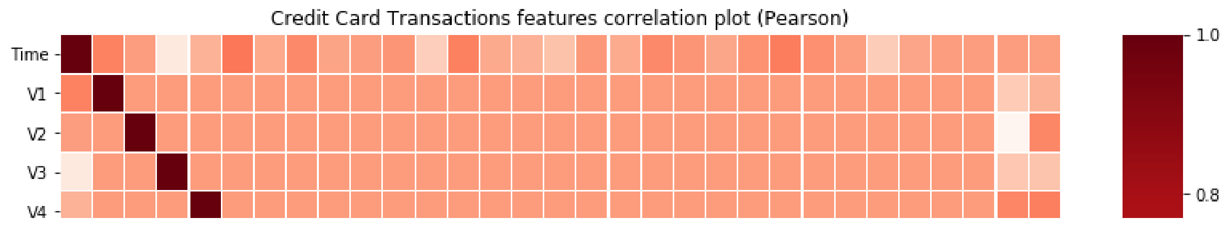
In [17]:

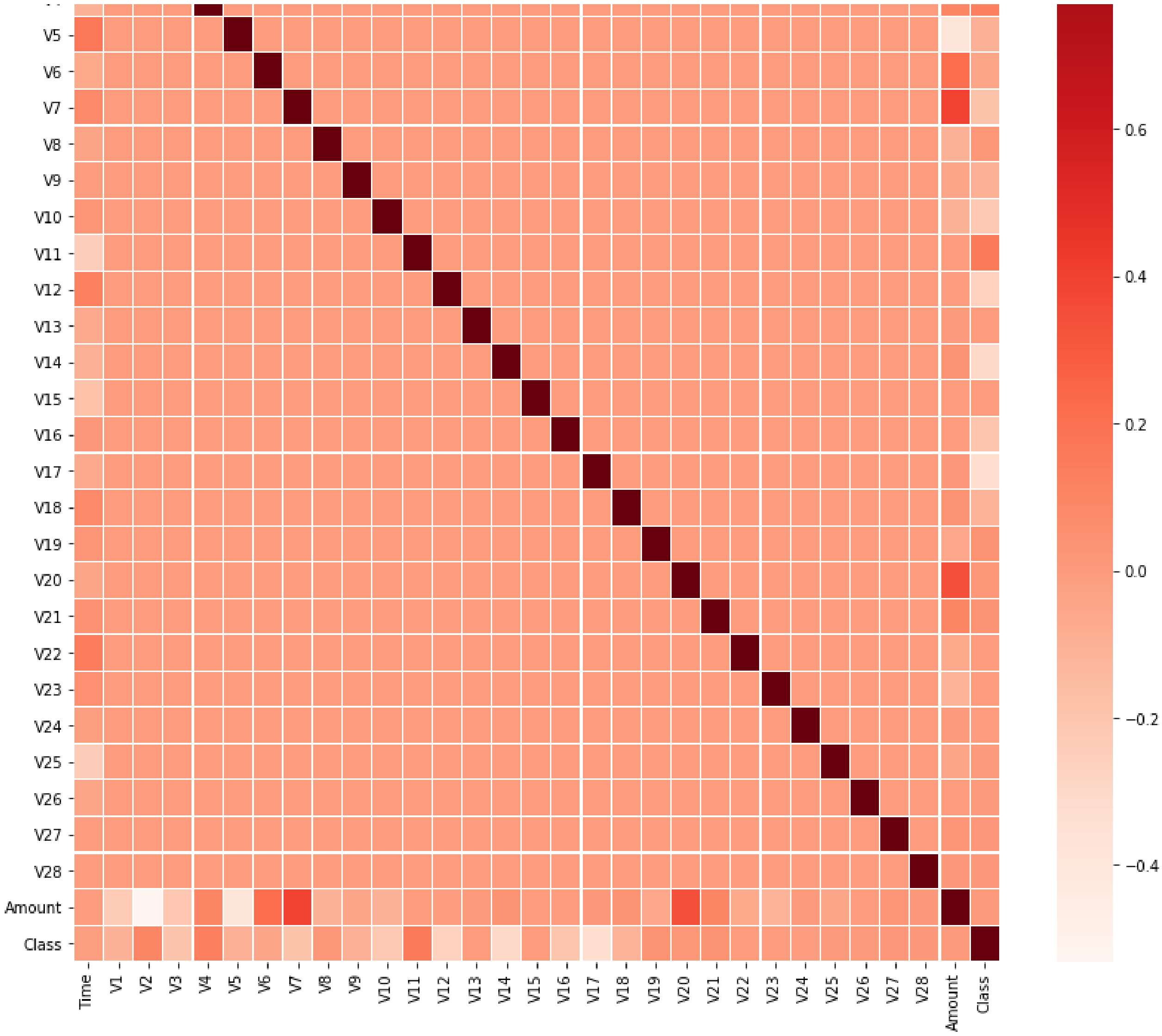
```
fraud = dataset.loc[dataset['Class'] == 1]

trace = go.Scatter(
    x = fraud['Time'],y = fraud['Amount'],
    name="Amount",
     marker=dict(
                color='rgb(238,23,11)',
                line=dict(
                    color='red',
                    width=1),
                opacity=0.5,
            ),
    text= fraud['Amount'],
    mode = "markers"
)
data = [trace]
layout = dict(title = 'Amount of fraudulent transactions',
          xaxis = dict(title = 'Time [s]', showticklabels=True),
          yaxis = dict(title = 'Amount'),
          hovermode='closest'
         )
fig = dict(data=data, layout=layout)
iplot(fig, filename='fraud-amount')
```

Features correlation

In [18]:

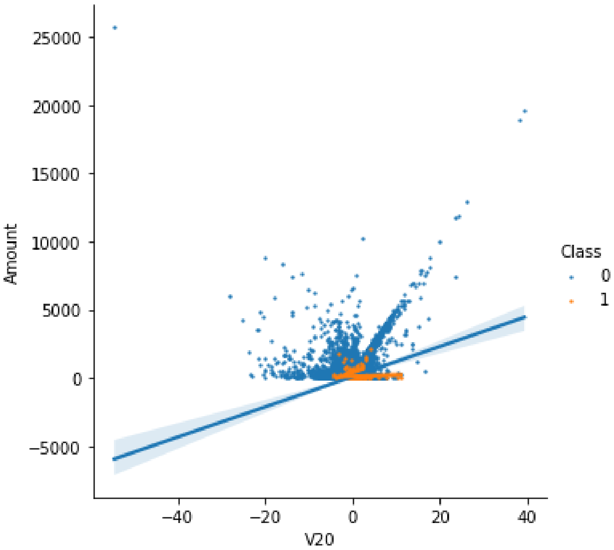
```
plt.figure(figsize = (14,14))
plt.title('Credit Card Transactions features correlation plot (Pearson)')
corr = dataset.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,cmap="Reds")
plt.show()
```

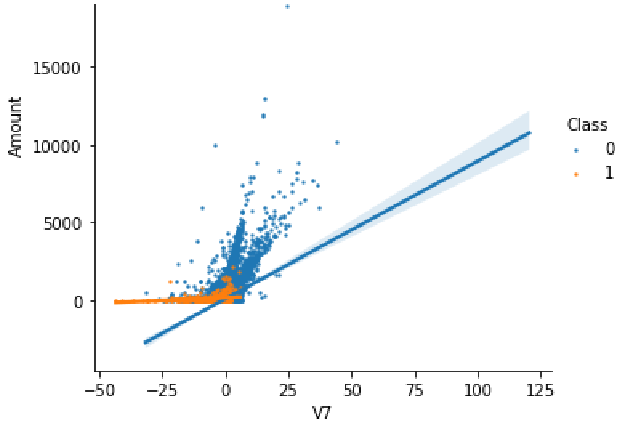




In [19]:

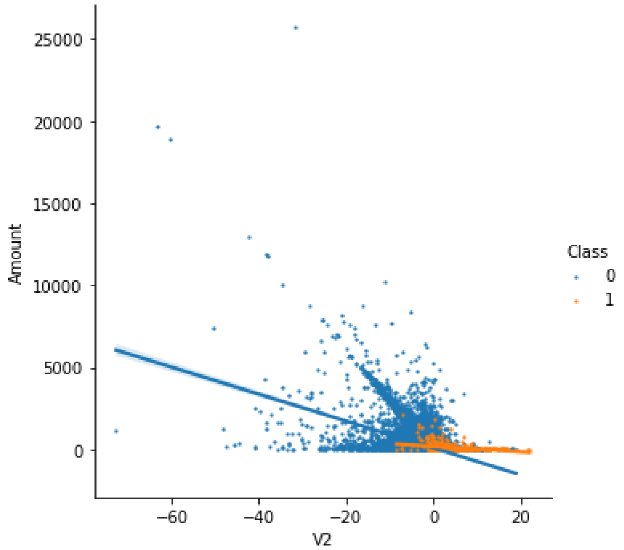
```
s = sns.lmplot(x='V20', y='Amount',data=dataset, hue='Class', fit_reg=True,scatter_kws={'s':2})
s = sns.lmplot(x='V7', y='Amount',data=dataset, hue='Class', fit_reg=True,scatter_kws={'s':2})
plt.show()
```

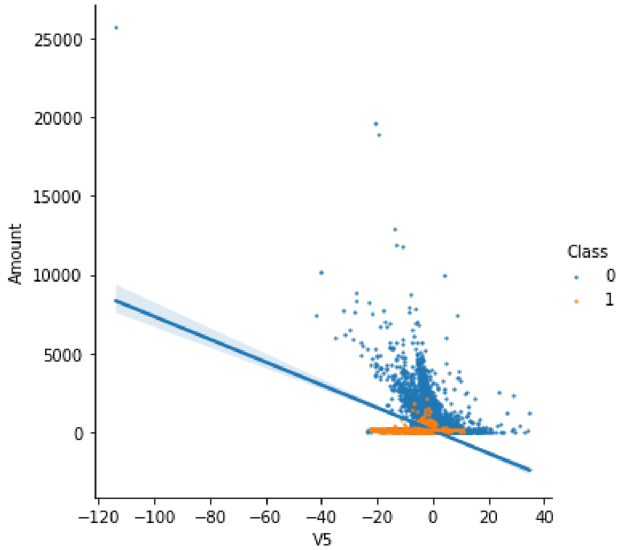




In [20]:

```
s = sns.lmplot(x='V2', y='Amount',data=dataset, hue='Class', fit_reg=True,scatter_kws={'s':2})
s = sns.lmplot(x='V5', y='Amount',data=dataset, hue='Class', fit_reg=True,scatter_kws={'s':2})
plt.show()
```





Features density plot

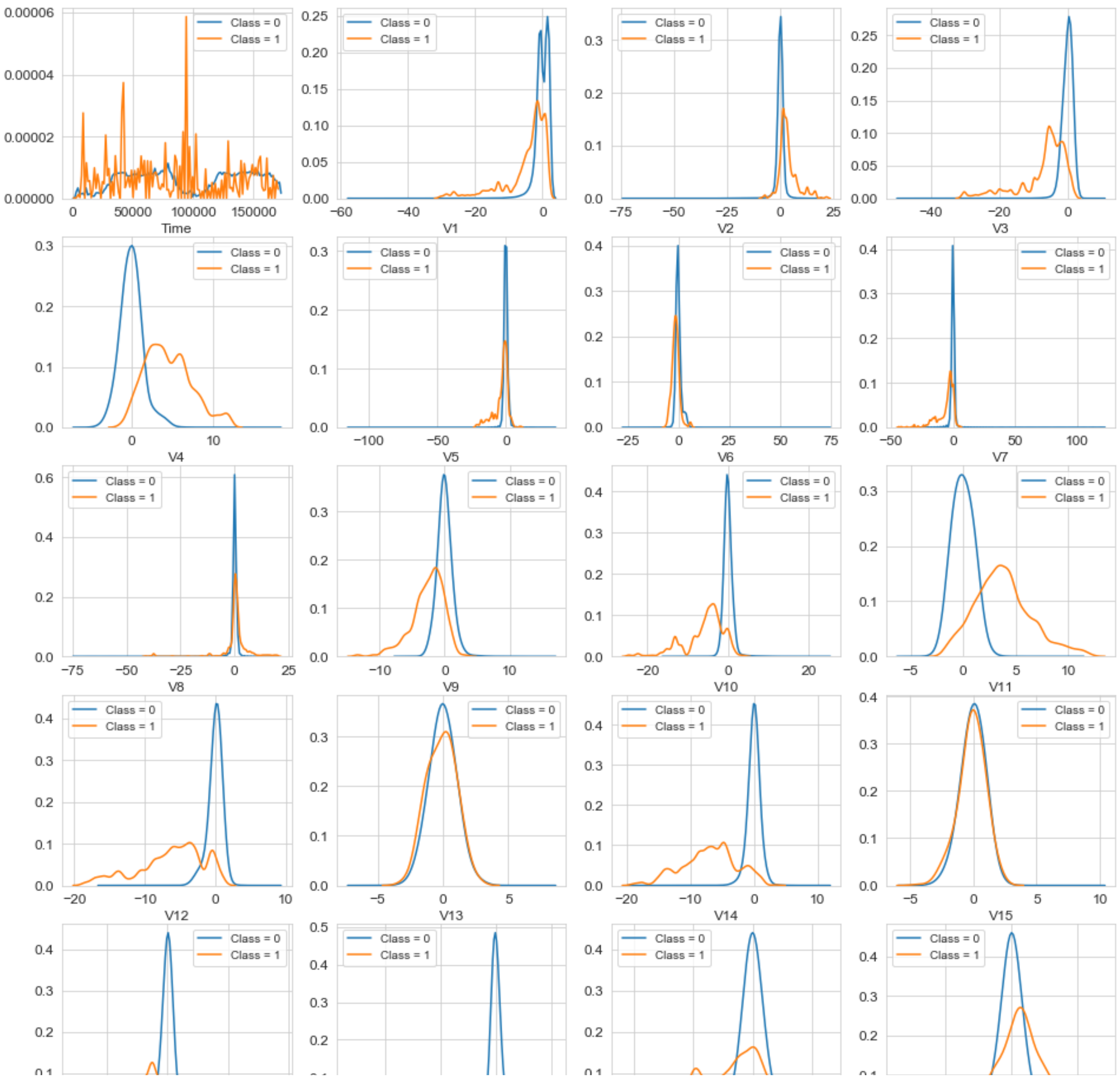
In [21]:

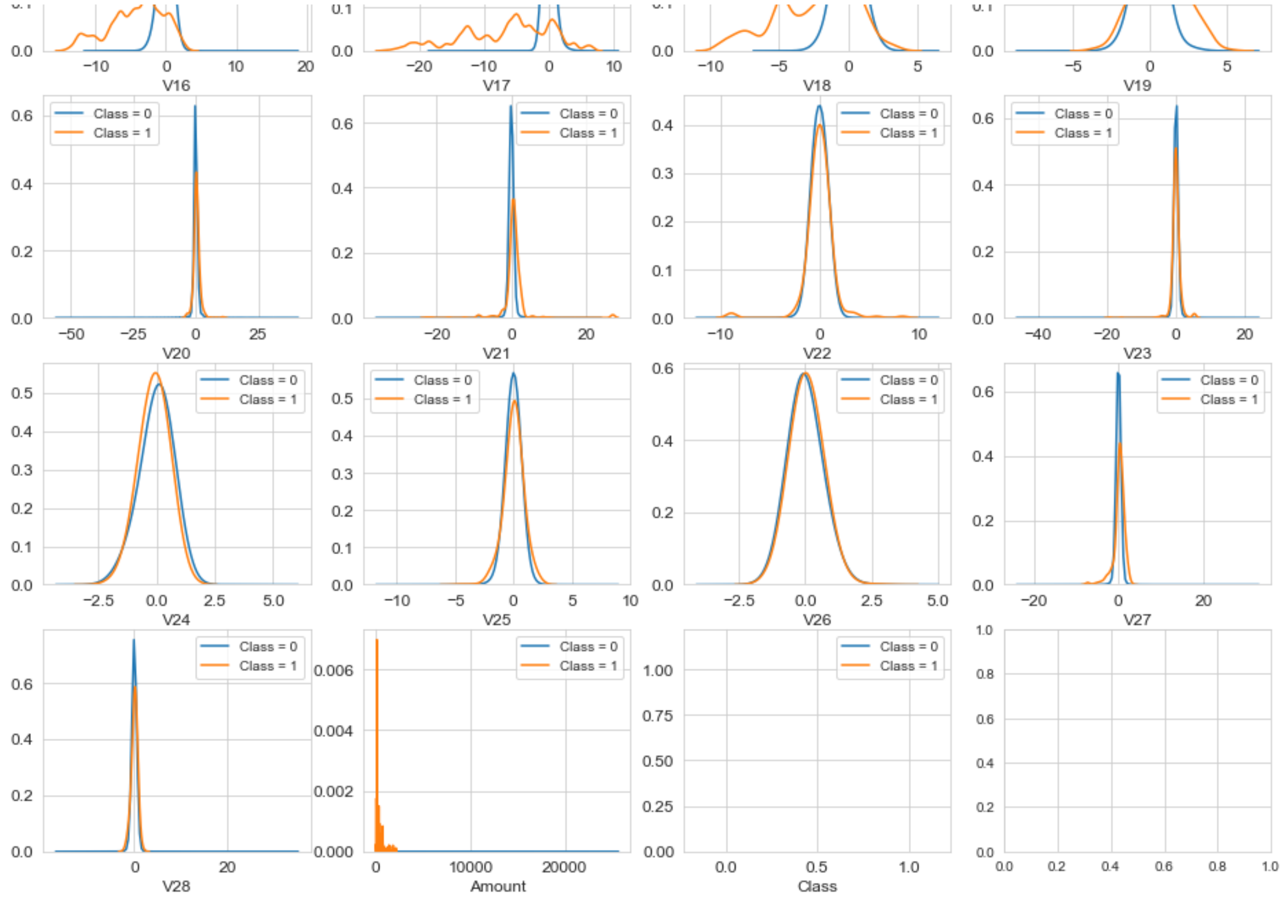
```
var = dataset.columns.values
```

```python
i = 0
t0 = dataset.loc[dataset['Class'] == 0]
t1 = dataset.loc[dataset['Class'] == 1]

sns.set_style('whitegrid')
plt.figure()
fig, ax = plt.subplots(8,4,figsize=(16,28))

for feature in var:
    i += 1
    plt.subplot(8,4,i)
    sns.kdeplot(t0[feature], bw=0.5,label="Class = 0")
    sns.kdeplot(t1[feature], bw=0.5,label="Class = 1")
    plt.xlabel(feature, fontsize=12)
    locs, labels = plt.xticks()
    plt.tick_params(axis='both', which='major', labelsize=12)
plt.show();
```

```
C:\Users\Welcome\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWarning:

Data must have variance to compute a kernel density estimate.

C:\Users\Welcome\anaconda3\lib\site-packages\seaborn\distributions.py:288: UserWarning:

Data must have variance to compute a kernel density estimate.
```

```
<Figure size 432x288 with 0 Axes>
```





Predictive models

In [22]:

```
target = 'Class'
predictors = ['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',\
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',\
       'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28',\
       'Amount']
```

In [23]:

```
train_df, test_df = train_test_split(dataset, test_size=TEST_SIZE, random_state=RANDOM_STATE,
shuffle=True )
train_df, valid_df = train_test_split(train_df, test_size=VALID_SIZE, random_state=RANDOM_STATE, sh
uffle=True )
```

In [24]:

```
train_df
```

Out[24]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V
46038	42612.0	-0.489771	0.319345	2.053837	0.729095	0.116019	1.800484	0.665996	0.010833	0.963039	-0.069473	-0.186374	0.687
257265	158079.0	-0.545212	0.491961	1.224502	-0.347668	0.211771	0.332727	0.397083	0.194546	0.538196	-1.601399	-1.337658	0.091
282877	171205.0	-1.312171	2.341898	-1.540839	-0.237345	-0.025731	-1.138582	-0.206301	-1.061977	-0.079198	-0.684883	-0.694481	0.691
226150	144511.0	2.029973	0.344478	-2.765664	1.145446	1.469254	-0.068542	0.508288	-0.076486	0.063214	-0.172802	0.091639	0.090
278800	168443.0	1.918104	0.002786	-1.765175	1.238956	0.573995	-0.626490	0.578274	-0.248337	0.000044	0.413549	0.387121	0.762

...	...	...	...	...	...	...	...	...	...	...	...	...	
	Time	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**	**V10**	**V11**	**V**
64434	51145.0	-0.898731	-0.064732	1.184601	0.196442	2.435108	4.041210	-0.932042	1.237936	-0.058540	-0.389705	-0.618639	0.075
164469	116737.0	1.128220	-1.701569	-0.892817	0.501977	-0.503090	0.880431	-0.268431	0.254330	1.373592	-0.511491	-0.190614	0.557
256083	157531.0	1.953158	-0.532874	-0.044464	0.619915	-1.195339	-0.880123	-0.726678	-0.015986	1.429566	0.044273	-0.821282	0.177
217751	141018.0	-1.024344	0.830423	-1.837306	1.455775	0.399188	-0.898847	2.087792	-0.376099	-1.753964	-0.622154	0.063710	0.078
166836	118338.0	1.936193	0.178625	-1.432951	1.032515	0.611245	-0.430855	0.412651	-0.176755	-0.301373	0.437408	1.169071	1.288

182276 rows × 31 columns

In [25]:

```
test_df
```

Out[25]:

	Time	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**	**V10**	**V11**	**V**
132514	80015.0	-1.130231	0.716230	0.560581	-0.818383	0.012762	-1.456391	0.989268	0.128520	-0.853739	-1.039166	-0.713705	0.243
231874	146958.0	0.887172	-2.177127	-2.365565	0.406678	-0.194711	-0.116013	0.812617	-0.257809	0.189188	0.061726	0.434397	0.353
240972	150826.0	-7.752743	6.396263	-5.797765	-0.932329	-3.551210	-2.022925	-2.709679	4.139447	2.034678	3.275483	-2.088108	1.225
91983	63715.0	1.039534	-1.071582	0.434023	-0.460506	-1.106821	-0.103705	-0.562238	0.076647	-0.714621	0.558690	1.487741	0.438
225669	144345.0	-9.742090	-8.480402	-3.582175	1.337566	-3.465308	4.292190	5.893034	-1.190614	1.133534	-0.307708	1.259998	0.441
...	...	...	...	...	...	...	...	...	...	...	...	...	
257165	158039.0	0.002789	-0.246172	0.432183	-2.192380	0.321394	0.594958	-0.165724	-0.147374	-0.493200	0.775867	0.188588	0.043
115726	73987.0	0.759530	-1.499729	-0.140199	-0.103248	-1.434009	-1.148393	0.170556	-0.366259	-0.767968	0.490421	-0.545246	0.232
171040	120468.0	2.259764	-0.734422	-1.389639	-1.033465	-0.418561	-0.846343	-0.532008	-0.239758	-0.353969	0.878250	-0.917697	1.044
8497	11395.0	-0.926167	0.980114	1.304831	-0.753279	-0.437823	-0.813154	0.223754	0.312315	1.492322	-1.458244	-0.146915	2.031
193254	130059.0	2.064870	0.043989	-1.725045	0.429677	0.302219	-0.845856	0.068676	-0.126801	0.718099	-0.342274	-0.752043	0.623

56962 rows × 31 columns

In [26]:

```
valid_df
```

Out[26]:

	Time	**V1**	**V2**	**V3**	**V4**	**V5**	**V6**	**V7**	**V8**	**V9**	**V10**	**V11**	**V**
155469	105519.0	-0.168435	0.601389	-1.322941	-0.794262	2.896167	3.723954	0.435124	0.809965	1.158062	-1.714128	1.078275	2.702
281294	170066.0	2.092300	0.203841	-2.046065	0.122669	0.789151	-1.062546	0.717175	-0.439611	-0.178991	0.187959	0.916904	1.391
56434	47425.0	-1.408212	-0.882836	0.390427	0.574399	1.269029	-0.063044	-0.731943	0.236889	-1.546301	0.895903	-1.608263	0.031
72882	54896.0	-4.041806	-5.119987	-2.640510	2.247336	-6.579783	3.314741	8.047611	-0.241093	-1.662153	-2.195497	0.555256	0.544
154497	101749.0	2.094108	0.277972	-1.964250	0.702926	0.868111	-0.324271	0.010622	-0.291076	1.974727	-1.082931	-0.121012	2.080
...	...	...	...	...	...	...	...	...	...	...	...	...	
130480	79363.0	1.204442	[illegible]	0.904010	[illegible]	[illegible]	[illegible]	[illegible]	0.046710	1.661364	[illegible]	[illegible]	1.375

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V
157649	110199.0	1.751363	0.337812	0.795262	4.312732	-0.681349	-0.126233	-0.566591	-0.031344	0.919523	0.975130	0.435764	2.215
166897	118373.0	1.706178	-0.022064	-0.371768	3.903360	0.137307	0.913564	-0.274172	0.272305	-0.430786	1.438444	-0.102608	0.313
173301	121432.0	-1.245628	0.797699	1.524232	-0.621050	-1.224624	-0.017876	-0.303114	0.563961	1.126998	-0.727933	-1.349329	0.155
188984	128221.0	-0.702233	0.473078	0.815668	-1.825901	0.046981	1.060628	-0.955869	-2.656679	-1.246511	-0.013146	-0.017544	0.625

45569 rows × 31 columns

a

RandomForestClassifier :

In [27]:

```
clf = RandomForestClassifier(n_jobs=NO_JOBS,
                             random_state=RANDOM_STATE,
                             criterion=RFC_METRIC,
                             n_estimators=NUM_ESTIMATORS,
                             verbose=False)
```

In [28]:

```
clf.fit(train_df[predictors], train_df[target].values)
```

Out[28]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=4,
                       oob_score=False, random_state=2018, verbose=False,
                       warm_start=False)
```

In [29]:

```
preds = clf.predict(valid_df[predictors])
```

In [30]:

```
from joblib import dump
dump(clf,'clf.save')
```

Out[30]:

```
['clf.save']
```

In [31]:

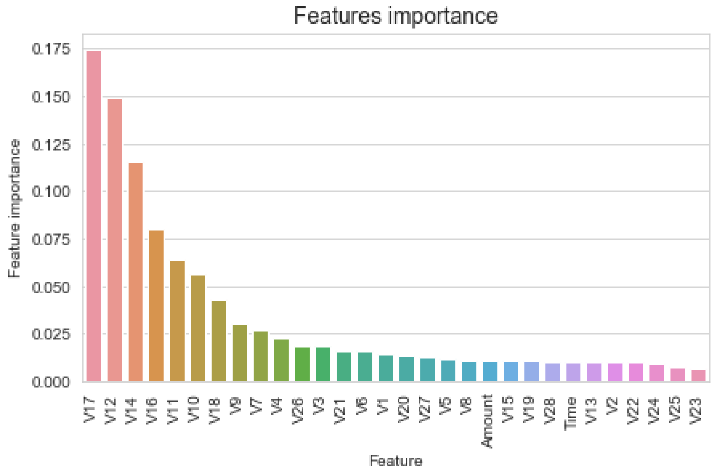
```
import pickle
pickle.dump(clf,open('clf.pkl','wb'))
```

Features importance

In [32]:

```
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.feature_importances_})
tmp = tmp.sort_values(by='Feature importance',ascending=False)
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)
```

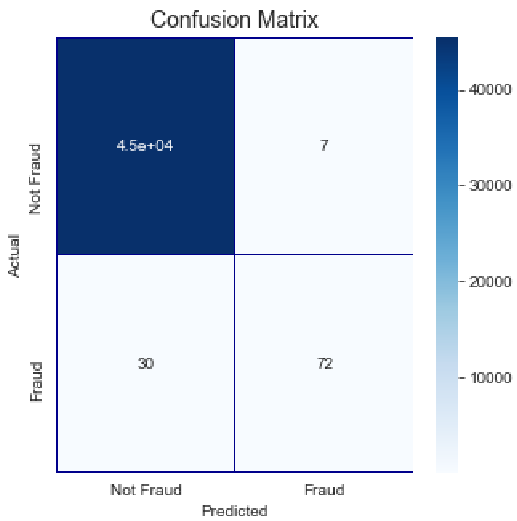
```
s.set_xticklabels(s.get_xticklabels(),rotation=90)
plt.show()
```



Confusion matrix

In [33]:

```
cm = pd.crosstab(valid_df[target].values, preds, rownames=['Actual'], colnames=['Predicted'])
fig, (ax1) = plt.subplots(ncols=1, figsize=(5,5))
sns.heatmap(cm,
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'],
            annot=True,ax=ax1,
            linewidths=.2,linecolor="Darkblue", cmap="Blues")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



Area under curve

In [34]:

```
roc_auc_score(valid_df[target].values, preds)
```

Out[34]:

```
0.8528641975628091
```

AdaBoostClassifier :

In [35]:

```
clf = AdaBoostClassifier(random_state=RANDOM_STATE,    #Prepare the model
                         algorithm='SAMME.R',
                         learning_rate=0.8,
                             n_estimators=NUM_ESTIMATORS)
```

Fit the model

In [36]:

```
clf.fit(train_df[predictors], train_df[target].values)
```

Out[36]:

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=0.8,
                   n_estimators=100, random_state=2018)
```

Predict the target values

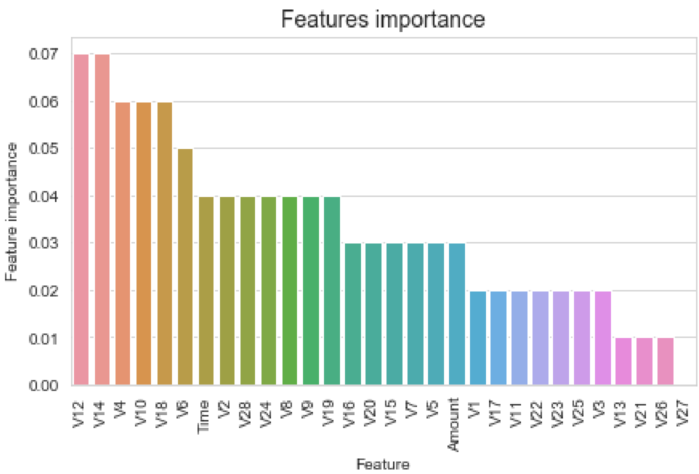
In [37]:

```
preds = clf.predict(valid_df[predictors])
```

Features Importance

In [38]:

```
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.feature_importances_})
tmp = tmp.sort_values(by='Feature importance',ascending=False)
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)
s.set_xticklabels(s.get_xticklabels(),rotation=90)
plt.show()
```

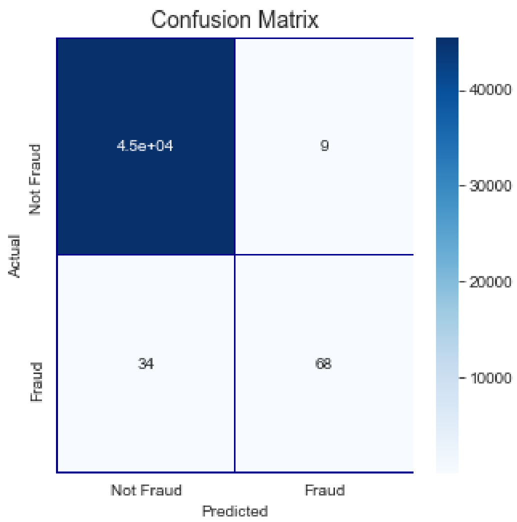


Confusion matrix

In [39]:

```
cm = pd.crosstab(valid_df[target].values, preds, rownames=['Actual'], colnames=['Predicted'])
fig, (ax1) = plt.subplots(ncols=1, figsize=(5,5))
sns.heatmap(cm,
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'],
```

```
            annot=True,ax=ax1,
            linewidths=.2,linecolor="Darkblue", cmap="Blues")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



Area under curve

In [40]:

```
roc_auc_score(valid_df[target].values, preds)
```

Out[40]:

```
0.8332343604519027
```

CatBoostClassifier :

In [41]:

```
clf = CatBoostClassifier(iterations=500,                            #Prepare the model
                             learning_rate=0.02,
                             depth=12,
                             eval_metric='AUC',
                             random_seed = RANDOM_STATE,
                             bagging_temperature = 0.2,
                             od_type='Iter',
                             metric_period = VERBOSE_EVAL,
                             od_wait=100)
```

In [42]:

```
clf.fit(train_df[predictors], train_df[target].values,verbose=True)
```

```
0: total: 1.41s remaining: 11m 43s
50: total: 51.2s remaining: 7m 30s
100: total: 1m 37s remaining: 6m 26s
150: total: 2m 22s remaining: 5m 28s
200: total: 3m 9s remaining: 4m 42s
250: total: 3m 54s remaining: 3m 52s
300: total: 4m 42s remaining: 3m 6s
350: total: 5m 30s remaining: 2m 20s
400: total: 6m 19s remaining: 1m 33s
450: total: 7m 6s remaining: 46.4s
499: total: 7m 55s remaining: 0us
```

Out[42]:

```
<catboost.core.CatBoostClassifier at 0x1aab606e288>
```

Predict the target values

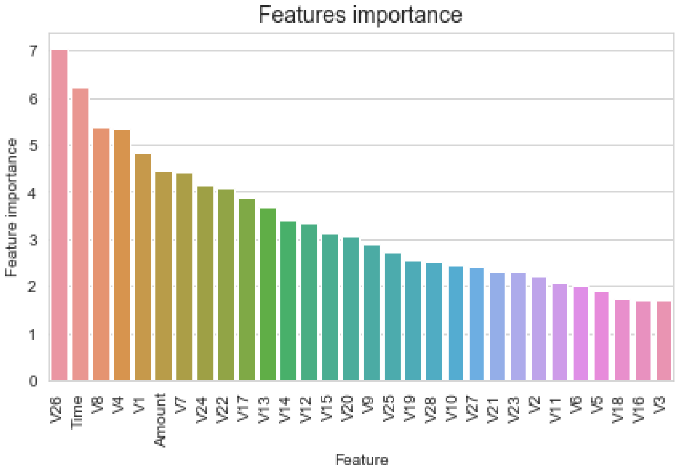
In [43]:

```
preds = clf.predict(valid_df[predictors])
```

Features importance

In [44]:

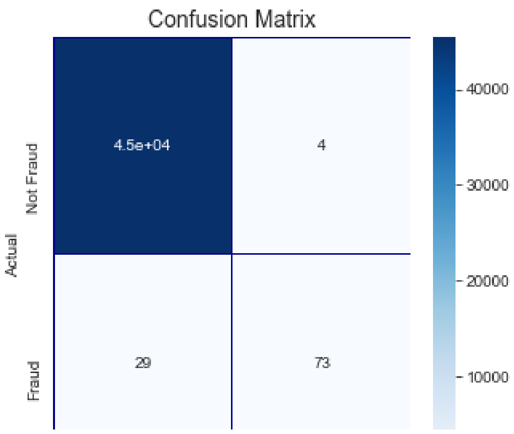
```
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.feature_importances_})
tmp = tmp.sort_values(by='Feature importance',ascending=False)
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)
s.set_xticklabels(s.get_xticklabels(),rotation=90)
plt.show()
```

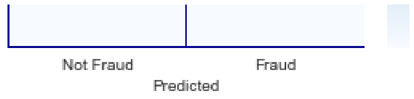


Confusion matrix

In [45]:

```
cm = pd.crosstab(valid_df[target].values, preds, rownames=['Actual'], colnames=['Predicted'])
fig, (ax1) = plt.subplots(ncols=1, figsize=(5,5))
sns.heatmap(cm,
            xticklabels=['Not Fraud', 'Fraud'],
            yticklabels=['Not Fraud', 'Fraud'],
            annot=True,ax=ax1,
            linewidths=.2,linecolor="Darkblue", cmap="Blues")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```





Area under curve

In [46]:

```
roc_auc_score(valid_df[target].values, preds)
```

Out[46]:

```
0.8577991493075996
```

XGBoost :

In [47]:

```
# Prepare the train and valid datasets
dtrain = xgb.DMatrix(train_df[predictors], train_df[target].values)
dvalid = xgb.DMatrix(valid_df[predictors], valid_df[target].values)
dtest = xgb.DMatrix(test_df[predictors], test_df[target].values)

#What to monitor (in this case, **train** and **valid**)
watchlist = [(dtrain, 'train'), (dvalid, 'valid')]

# Set xgboost parameters
params = {}
params['objective'] = 'binary:logistic'
params['eta'] = 0.039
params['silent'] = True
params['max_depth'] = 2
params['subsample'] = 0.8
params['colsample_bytree'] = 0.9
params['eval_metric'] = 'auc'
params['random_state'] = RANDOM_STATE
```

In [48]:

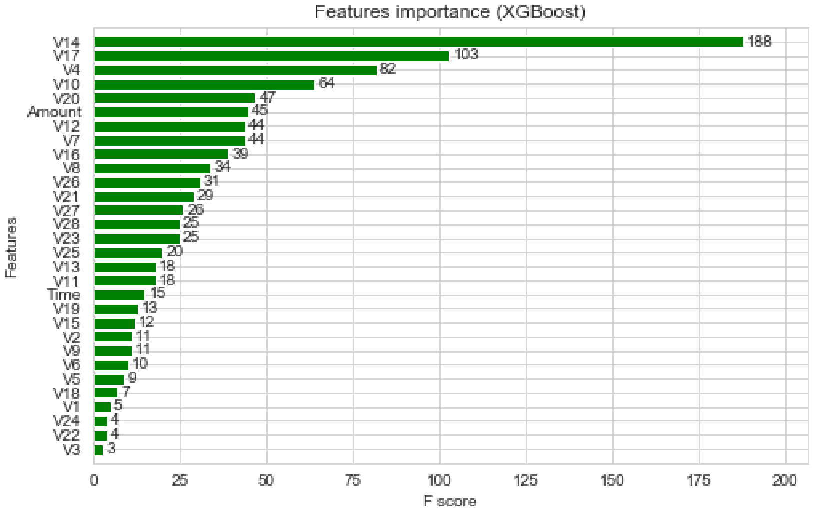
```
model = xgb.train(params,          #Train the model
                dtrain,
                MAX_ROUNDS,
                watchlist,
                early_stopping_rounds=EARLY_STOP,
                maximize=True,
                verbose_eval=VERBOSE_EVAL)
```

```
[11:52:31] WARNING: C:\Users\Administrator\workspace\xgboost-
win64_release_1.1.0\src\learner.cc:480:
Parameters: { silent } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[0] train-auc:0.89296 valid-auc:0.85272
Multiple eval metrics have been passed: 'valid-auc' will be used for early stopping.

Will train until valid-auc hasn't improved in 50 rounds.
[50] train-auc:0.93947 valid-auc:0.88200
[100] train-auc:0.94415 valid-auc:0.89094
[150] train-auc:0.97837 valid-auc:0.96362
[200] train-auc:0.99002 valid-auc:0.98397
[250] train-auc:0.99382 valid-auc:0.98592
[300] train-auc:0.99567 valid-auc:0.98667
Stopping. Best iteration:
[282] train-auc:0.99517 valid-auc:0.98706
```

Plot variable importance

In [49]:

```
fig, (ax) = plt.subplots(ncols=1, figsize=(8,5))
xgb.plot_importance(model, height=0.8, title="Features importance (XGBoost)", ax=ax, color="green")

plt.show()
```



Predict test set

In [50]:

```
preds = model.predict(dtest)
```

Area under curve

In [51]:

```
roc_auc_score(test_df[target].values, preds)
```

Out[51]:

```
0.9766700080897612
```

LightGBM :

In [52]:

```
params = {                                              #Defining model parameters
          'boosting_type': 'gbdt',
          'objective': 'binary',
          'metric':'auc',
          'learning_rate': 0.05,
          'num_leaves': 7,  # we should let it be smaller than 2^(max_depth)
          'max_depth': 4,  # -1 means no limit
          'min_child_samples': 100,  # Minimum number of data need in a child(min_data_in_leaf)
          'max_bin': 100,  # Number of bucketed bin for feature values
          'subsample': 0.9,  # Subsample ratio of the training instance.
          'subsample_freq': 1,  # frequence of subsample, <=0 means no enable
          'colsample_bytree': 0.7,  # Subsample ratio of columns when constructing each tree.
          'min_child_weight': 0,  # Minimum sum of instance weight(hessian) needed in a child(leaf)
          'min_split_gain': 0,  # lambda_l1, lambda_l2 and min_gain_to_split to regularization
```

```
            'nthread': 8,
            'verbose': 0,
            'scale_pos_weight':150, # because training data is extremely unbalanced
          }
```

In [53]:

```
dtrain = lgb.Dataset(train_df[predictors].values,      #Prepare the model
                     label=train_df[target].values,
                     feature_name=predictors)

dvalid = lgb.Dataset(valid_df[predictors].values,
                     label=valid_df[target].values,
                     feature_name=predictors)
```

In [54]:

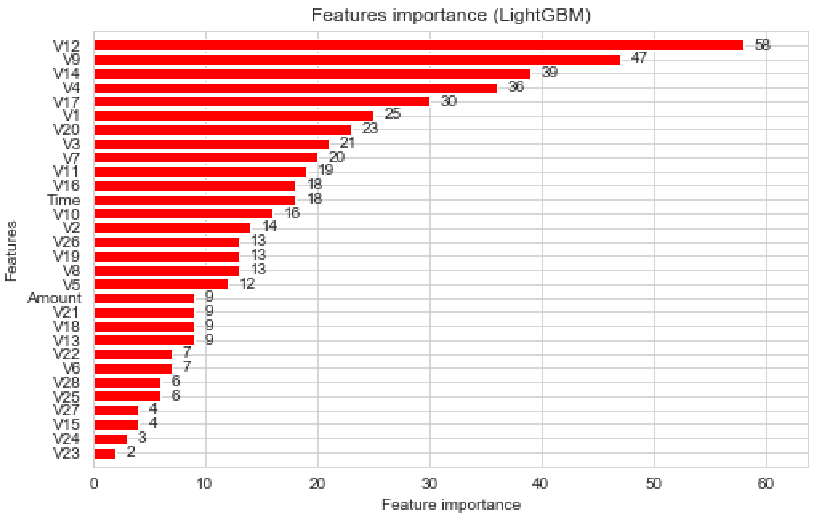
```
evals_results = {}                          #Run the model

model = lgb.train(params,
                  dtrain,
                  valid_sets=[dtrain, dvalid],
                  valid_names=['train','valid'],
                  evals_result=evals_results,
                  num_boost_round=MAX_ROUNDS,
                  early_stopping_rounds=2*EARLY_STOP,
                  verbose_eval=VERBOSE_EVAL,
                  feval=None)
```

```
Training until validation scores don't improve for 100 rounds
[50] train's auc: 0.97289 valid's auc: 0.967126
[100] train's auc: 0.987513 valid's auc: 0.972525
[150] train's auc: 0.988872 valid's auc: 0.93531
Early stopping, best iteration is:
[85] train's auc: 0.987093 valid's auc: 0.974528
```

Features importance

In [55]:

```
fig, (ax) = plt.subplots(ncols=1, figsize=(8,5))
lgb.plot_importance(model, height=0.8, title="Features importance (LightGBM)", ax=ax,color="red")
plt.show()
```



Predict test data

In [56]:

```
preds = model.predict(test_df[predictors])
```

Area under curve

In [57]:

```
roc_auc_score(test_df[target].values, preds)
```

Out[57]:

```
0.9459470296507333
```

Training and validation using cross-validation

In [58]:

```
kf = KFold(n_splits = NUMBER_KFOLDS, random_state = RANDOM_STATE, shuffle = True)

# Create arrays and dataframes to store results
oof_preds = np.zeros(train_df.shape[0])
test_preds = np.zeros(test_df.shape[0])
feature_importance_df = pd.DataFrame()
n_fold = 0
for train_idx, valid_idx in kf.split(train_df):
    train_x, train_y = train_df[predictors].iloc[train_idx],train_df[target].iloc[train_idx]
    valid_x, valid_y = train_df[predictors].iloc[valid_idx],train_df[target].iloc[valid_idx]

    evals_results = {}
    model =  LGBMClassifier(
                  nthread=-1,
                  n_estimators=2000,
                  learning_rate=0.01,
                  num_leaves=80,
                  colsample_bytree=0.98,
                  subsample=0.78,
                  reg_alpha=0.04,
                  reg_lambda=0.073,
                  subsample_for_bin=50,
                  boosting_type='gbdt',
                  is_unbalance=False,
                  min_split_gain=0.025,
                  min_child_weight=40,
                  min_child_samples=510,
                  objective='binary',
                  metric='auc',
                  silent=-1,
                  verbose=-1,
                  feval=None)
    model.fit(train_x, train_y, eval_set=[(train_x, train_y), (valid_x, valid_y)],
                eval_metric= 'auc', verbose= VERBOSE_EVAL, early_stopping_rounds= EARLY_STOP)

    oof_preds[valid_idx] = model.predict_proba(valid_x, num_iteration=model.best_iteration_)[:, 1]
    test_preds += model.predict_proba(test_df[predictors], num_iteration=model.best_iteration_)[:,
1] / kf.n_splits

    fold_importance_df = pd.DataFrame()
    fold_importance_df["feature"] = predictors
    fold_importance_df["importance"] = clf.feature_importances_
    fold_importance_df["fold"] = n_fold + 1

    feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)
    print('Fold %2d AUC : %.6f' % (n_fold + 1, roc_auc_score(valid_y, oof_preds[valid_idx])))
    del model, train_x, train_y, valid_x, valid_y
    gc.collect()
    n_fold = n_fold + 1
train_auc_score = roc_auc_score(train_df[target], oof_preds)
print('Full AUC score %.6f' % train_auc_score)
```

```
Training until validation scores don't improve for 50 rounds
[50] training's auc: 0.962157 valid_1's auc: 0.989338
Early stopping, best iteration is:
```

```
Early stopping, best iteration is:
[13] training's auc: 0.968109 valid_1's auc: 0.99314
Fold  1 AUC : 0.993140
Training until validation scores don't improve for 50 rounds
[50] training's auc: 0.981643 valid_1's auc: 0.95593
Early stopping, best iteration is:
[10] training's auc: 0.979098 valid_1's auc: 0.965326
Fold  2 AUC : 0.965326
Training until validation scores don't improve for 50 rounds
[50] training's auc: 0.979434 valid_1's auc: 0.943348
Early stopping, best iteration is:
[37] training's auc: 0.981891 valid_1's auc: 0.945099
Fold  3 AUC : 0.945099
Training until validation scores don't improve for 50 rounds
[50] training's auc: 0.972729 valid_1's auc: 0.989338
[100] training's auc: 0.97498 valid_1's auc: 0.994546
[150] training's auc: 0.976713 valid_1's auc: 0.994725
Early stopping, best iteration is:
[118] training's auc: 0.974884 valid_1's auc: 0.995364
Fold  4 AUC : 0.995364
Training until validation scores don't improve for 50 rounds
[50] training's auc: 0.974794 valid_1's auc: 0.987631
Early stopping, best iteration is:
[44] training's auc: 0.975142 valid_1's auc: 0.988285
Fold  5 AUC : 0.988285
Full AUC score 0.930928
```

In [59]:

```
pred = test_preds                 #The AUC score for the prediction from the test data was 0.93
                                  #We prepare the test prediction, from the averaged predictions fo
test over the 5 folds
```

In [ ]: