DSC680 Project2: Credit Card Fraud Detection

Assignment 7.1

Name: Madhuri Basava

Date: 07/17/2023

In [1]:
```
# Import the necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Set the style of matplotlib
%matplotlib inline
#  plt.style.use('fivethirtyeight')

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, roc_curve, confusion_matrix, ConfusionMa
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
```

In [2]:

```
# Load the Credit card dataset into the data frame
credit_card_df = pd.read_csv('creditcard.csv')
credit_card_df
```

Out[2]:

	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	0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2
...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.3
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.2
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.7
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.6
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.4

284807 rows × 31 columns

In [3]:

```
# Display the information to understand the dataset
credit_card_df.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 [4]:

```
# Check for any missing values
credit_card_df.isna().sum()
```

Out[4]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

There are no missing values in the dataset

In [5]:
```
# Check if there are duplicate rows in the data set
credit_card_df.duplicated().sum()
```

Out[5]:
```
1081
```

In [6]:
```
# Removing the duplicate rows

credit_card_df.drop_duplicates(inplace=True)
credit_card_df.duplicated().sum()
```

Out[6]:
```
0
```

Exploratory Data Analysis

In [7]:
```
credit_card_df.describe()
```

Out[7]:

	Time	V1	V2	V3	V4	V5
count	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000
mean	94811.077600	0.005917	-0.004135	0.001613	-0.002966	0.001828
std	47481.047891	1.948026	1.646703	1.508682	1.414184	1.377008
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	-0.689830
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	-0.053468
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	0.612218
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666

In [8]:

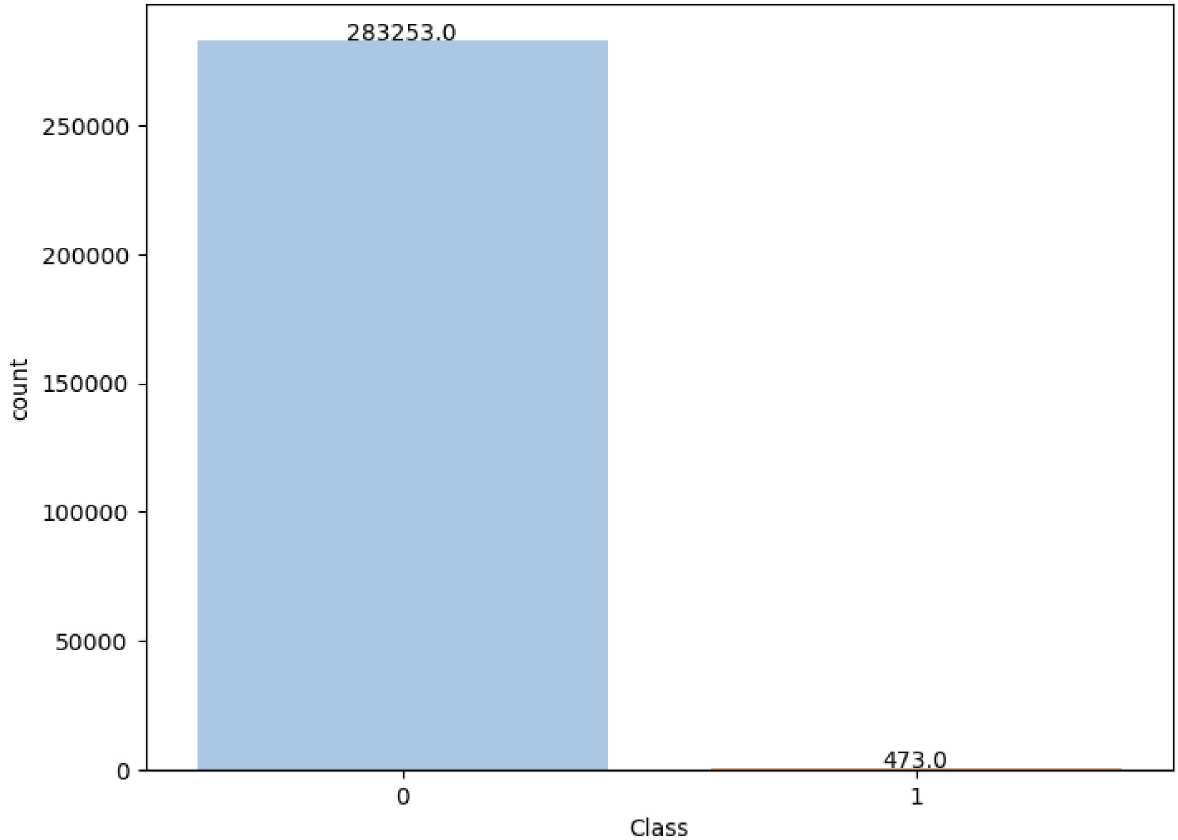
```python
# Print the number of unique values for each column

for col in credit_card_df.columns:
    print(f'{col} has {credit_card_df[col].nunique()} values')
```

```
Time has 124592 values
V1 has 275663 values
V2 has 275663 values
V3 has 275663 values
V4 has 275663 values
V5 has 275663 values
V6 has 275663 values
V7 has 275663 values
V8 has 275663 values
V9 has 275663 values
V10 has 275663 values
V11 has 275663 values
V12 has 275663 values
V13 has 275663 values
V14 has 275663 values
V15 has 275663 values
V16 has 275663 values
V17 has 275663 values
V18 has 275663 values
V19 has 275663 values
V20 has 275663 values
V21 has 275663 values
V22 has 275663 values
V23 has 275663 values
V24 has 275663 values
V25 has 275663 values
V26 has 275663 values
V27 has 275663 values
V28 has 275663 values
Amount has 32767 values
Class has 2 values
```

In [9]:

```python
# Target distribution
```

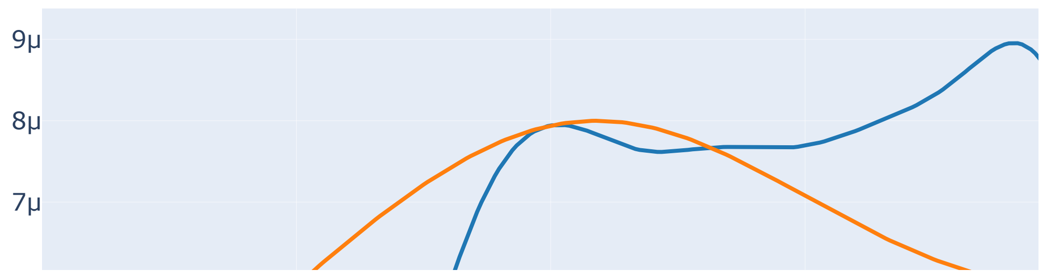
```python
# Set the figure size and create a count plot
plt.figure(figsize=(8, 6))
ax = sns.countplot(x=credit_card_df['Class'], palette='pastel')

# Add labels to each bar in the plot
for p in ax.patches:
    ax.text(p.get_x()+p.get_width()/2, p.get_height()+3, f'{p.get_height()}', ha="c

plt.show()
```



Only 473 of transactions are fraudulent. So, the data is highly unbalanced with respect with target variable Class.

In [10]:
```python
class_0 = credit_card_df.loc[credit_card_df['Class'] == 0]["Time"]
class_1 = credit_card_df.loc[credit_card_df['Class'] == 1]["Time"]

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
iplot(fig, filename='dist_only')
```

Credit Card Transactions Time Density Plot



Distribution of fraudulent transactions are more even than valid transactions.

Let's look into more details to the time distribution of both classes transaction, as well as to aggregated values of transaction count and amount, per hour. We assume that the time unit is second

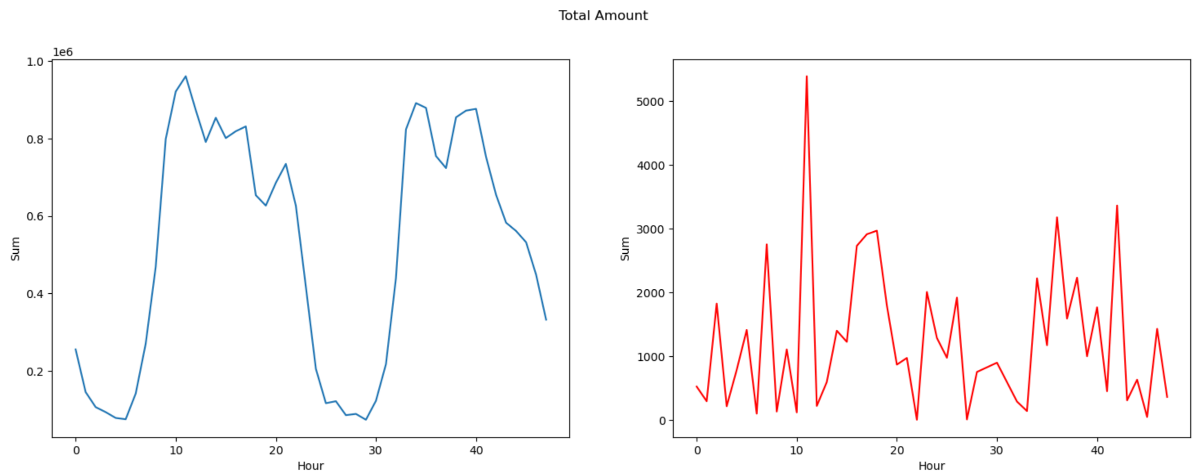
In [11]:
```
credit_card_df['Hour'] = credit_card_df['Time'].apply(lambda x: np.floor(x / 3600))

tmp = credit_card_df.groupby(['Hour', 'Class'])['Amount'].aggregate(['min', 'max',
df = pd.DataFrame(tmp)
df.columns = ['Hour', 'Class', 'Min', 'Max', 'Transactions', 'Sum', 'Mean', 'Median
df.head()
```

Out[11]:

	Hour	Class	Min	Max	Transactions	Sum	Mean	Median	Var
0	0.0	0	0.0	7712.43	3929	255825.95	65.112230	12.990	45961.838558
1	0.0	1	0.0	529.00	2	529.00	264.500000	264.500	139920.500000
2	1.0	0	0.0	1769.69	2211	145744.59	65.917951	23.000	20085.295527
3	1.0	1	59.0	239.93	2	298.93	149.465000	149.465	16367.832450
4	2.0	0	0.0	4002.88	1552	106983.39	68.932597	17.985	45434.509936

In [12]:

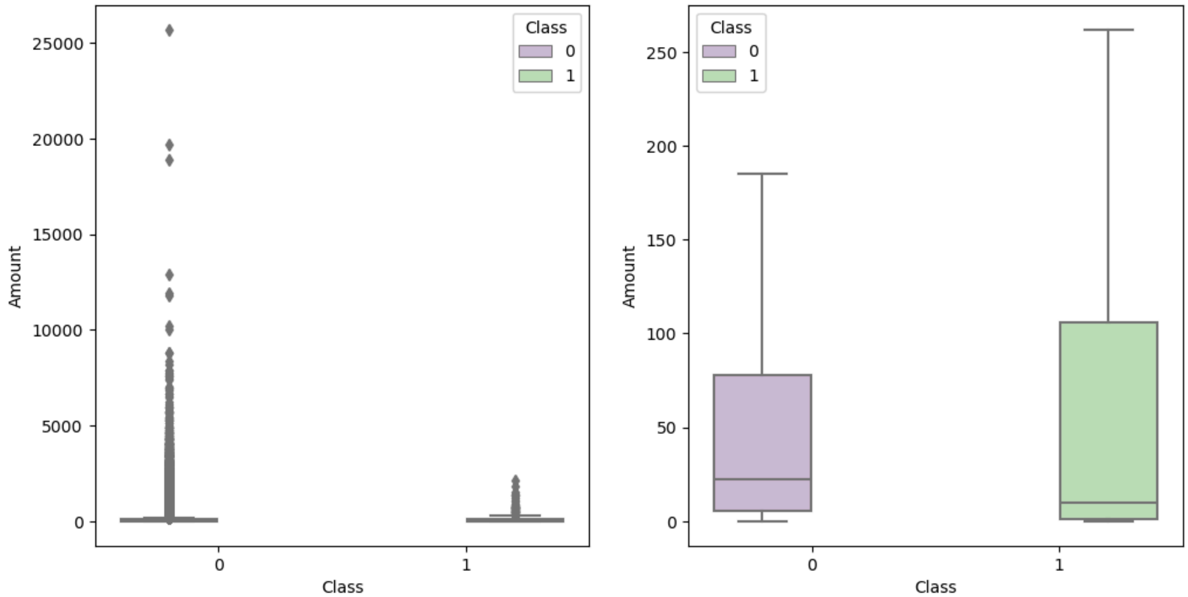
```
# Total Amount
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Sum", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Sum", data=df.loc[df.Class==1], color="red"
plt.suptitle("Total Amount")
plt.show();
```



In [13]:

```
# Total Number of Transactions
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18,6))
s = sns.lineplot(ax = ax1, x="Hour", y="Transactions", data=df.loc[df.Class==0])
s = sns.lineplot(ax = ax2, x="Hour", y="Transactions", data=df.loc[df.Class==1], co
plt.suptitle("Total Number of Transactions")
plt.show();
```

In [14]:
```
# Transcation Amount
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))
s = sns.boxplot(ax = ax1, x="Class", y="Amount", hue="Class",data=credit_card_df, p
s = sns.boxplot(ax = ax2, x="Class", y="Amount", hue="Class",data=credit_card_df, p
plt.show();
```



In [15]:
```
tmp = credit_card_df[['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    283253.000000
mean         88.413575
std         250.379023
min           0.000000
25%           5.670000
50%          22.000000
75%          77.460000
max       25691.160000
Name: Amount, dtype: float64
```

In [16]:
```
class_1.describe()
```

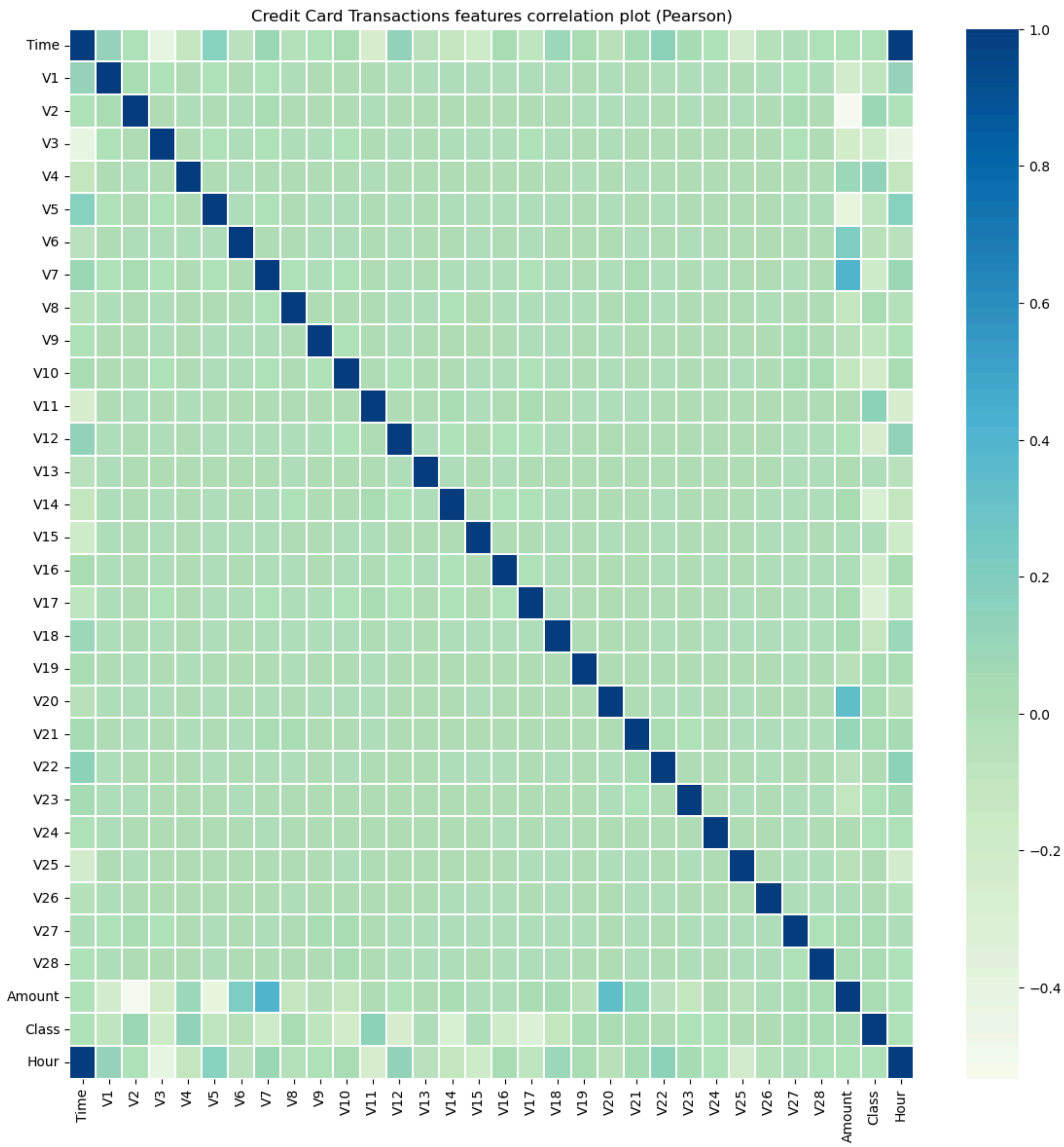
Out[16]:
```
count     473.000000
mean      123.871860
std       260.211041
min         0.000000
25%         1.000000
50%         9.820000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

In [17]:
```
# Amount of fraudulfraudulent transactions
fraud = credit_card_df.loc[credit_card_df['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')
```

In [18]:
```
# Correlation Matrix
plt.figure(figsize = (14,14))
plt.title('Credit Card Transactions features correlation plot (Pearson)')
```

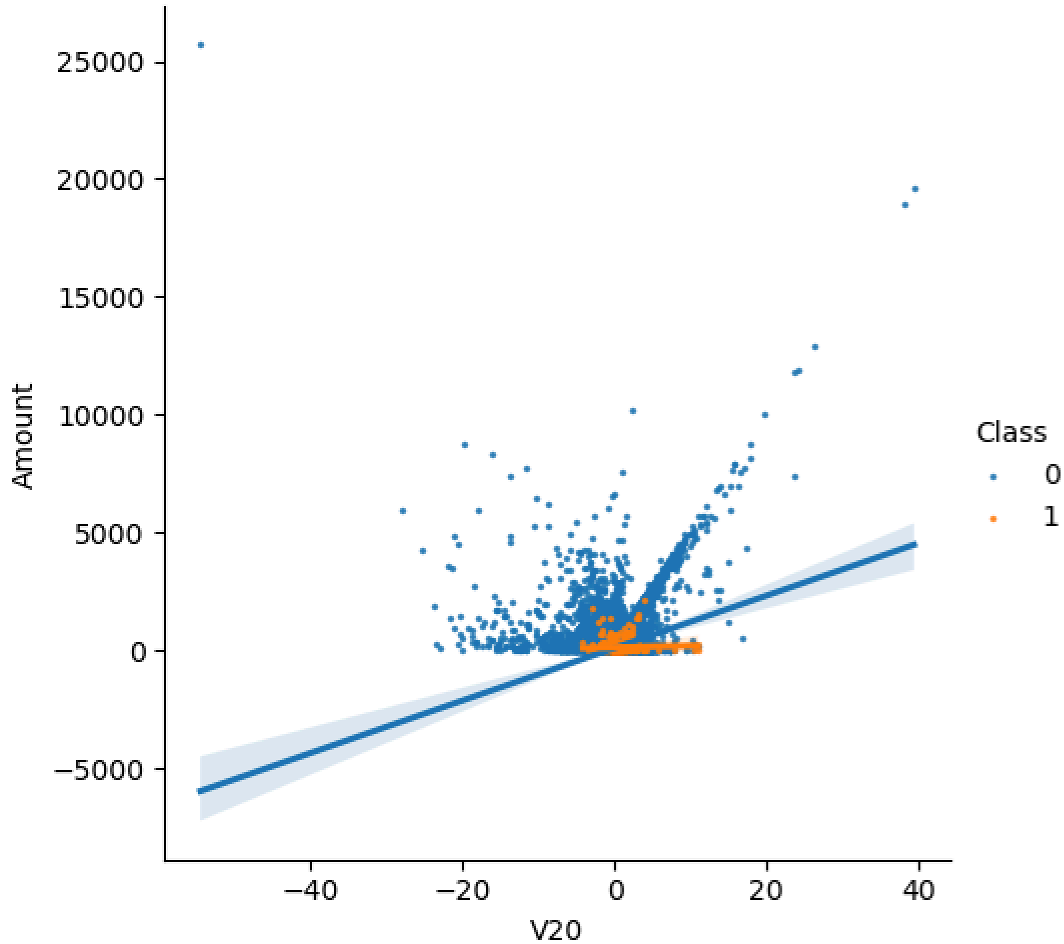
```
corr = credit_card_df.corr()
sns.heatmap(corr,xticklabels=corr.columns,yticklabels=corr.columns,linewidths=.1,cm
plt.show()
```

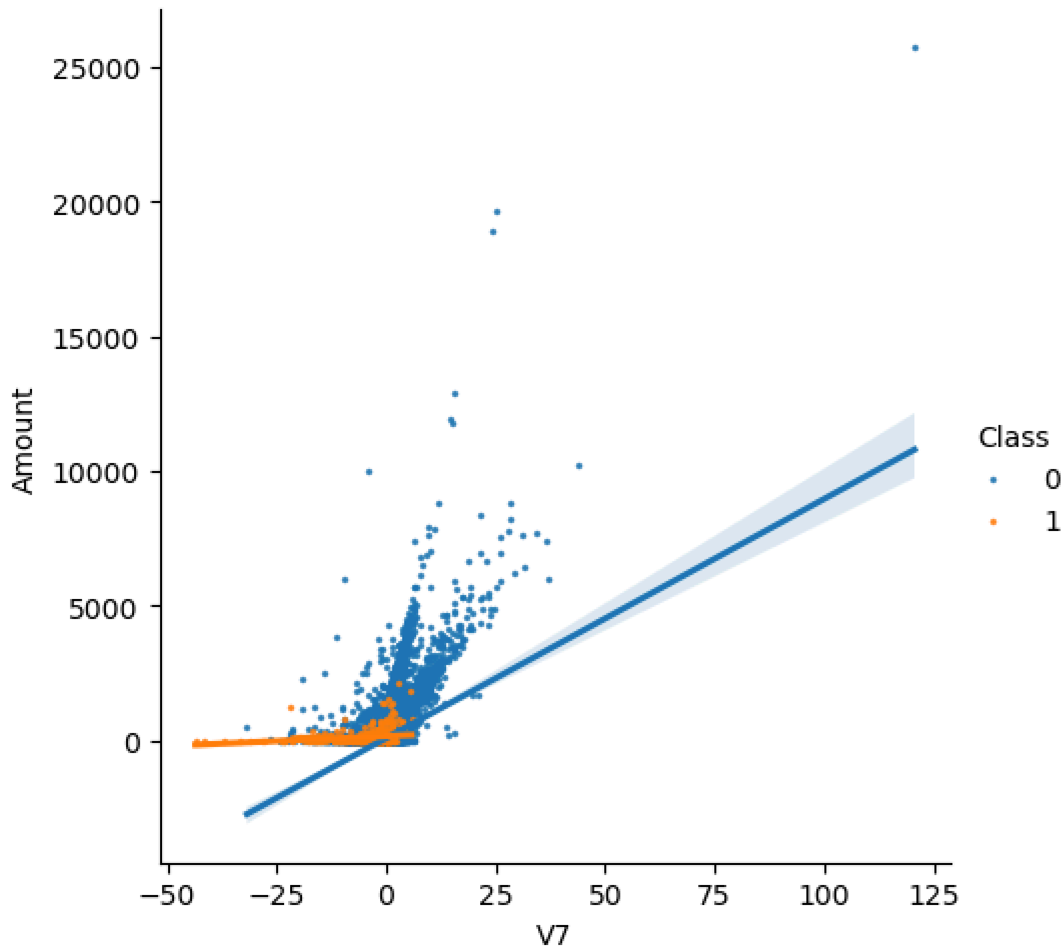


From the above heat map, we can conclude that there is no notable correlation between features V1-V28. There are certain correlations between some of these features and Time (inverse correlation with V3) and Amount (direct correlation with V7 and V20, inverse correlation with V1 and V5).

Now, Let's start with the direct correlated values: (V20;Amount) and (V7;Amount) .

In [19]:
```
s = sns.lmplot(x='V20', y='Amount',data=credit_card_df, hue='Class', fit_reg=True,s
s = sns.lmplot(x='V7', y='Amount',data=credit_card_df, hue='Class', fit_reg=True,sc
plt.show()
```

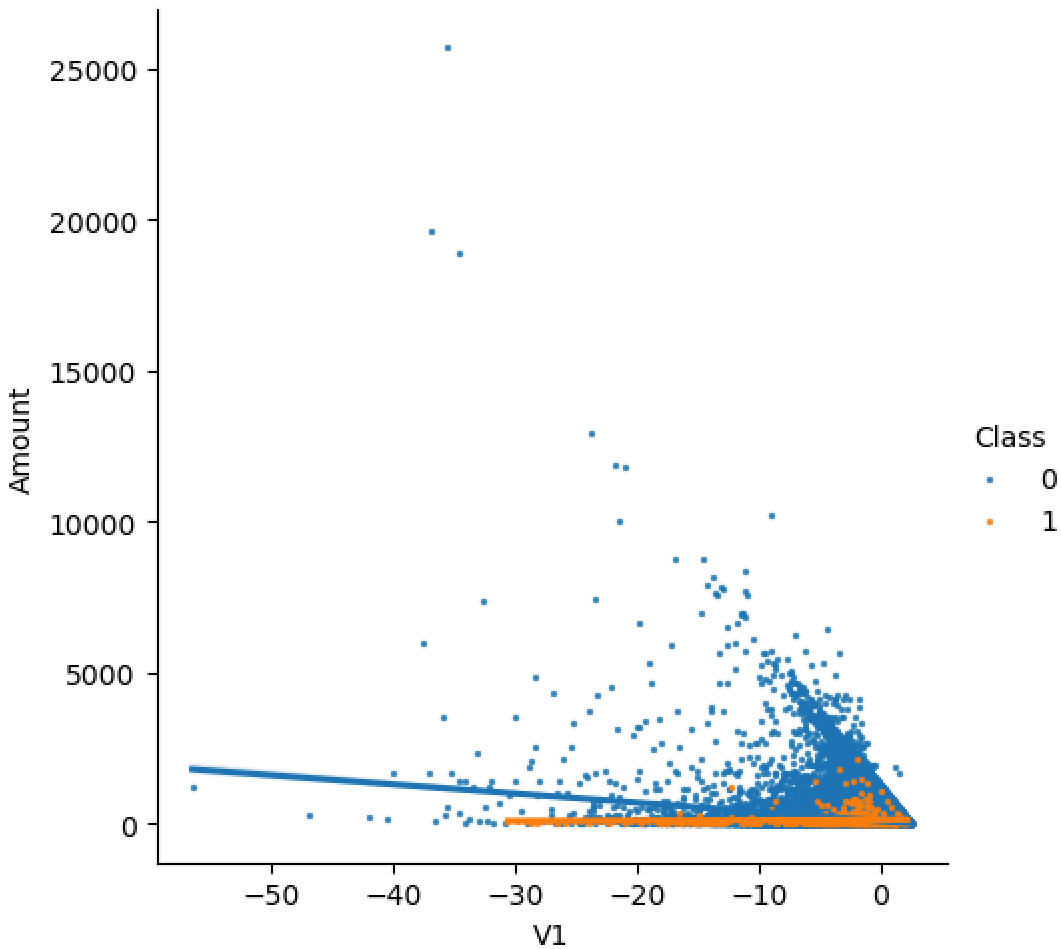

25000
20000
15000
10000
5000
0
−5000
Amount
−40
−20
0
20
40
V20
Class
0
1



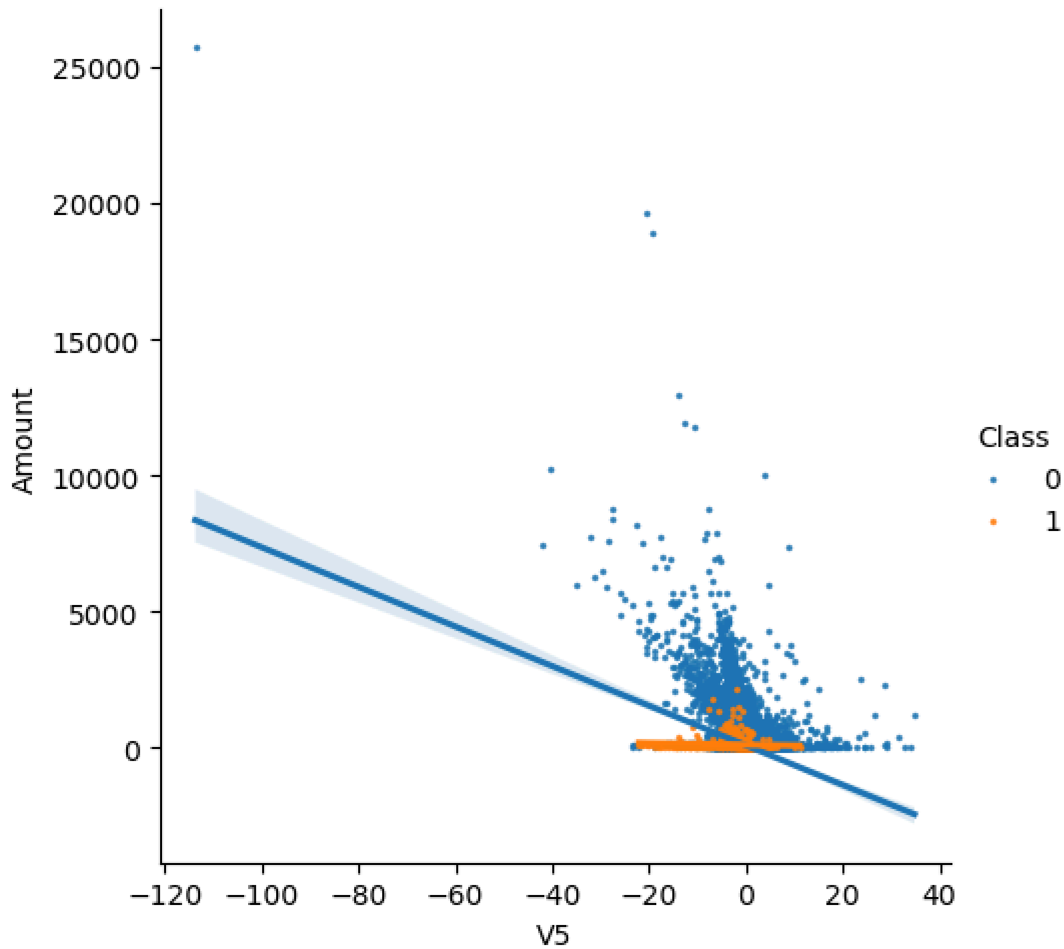
From the above graphs, we can confirm that the two couples of features are correlated (the regression lines for Class = 0 have a positive slope, whilst the regression line for Class = 1 have a smaller positive slope).

Now, let's plot now the inverse correlated values: (V1;Amount) and (V2;Amount).

In [20]:
```
s = sns.lmplot(x='V1', y='Amount',data=credit_card_df, hue='Class', fit_reg=True,sc
s = sns.lmplot(x='V5', y='Amount',data=credit_card_df, hue='Class', fit_reg=True,sc
plt.show()
```



25000
20000
15000
10000
5000
0
Amount
−50
−40
−30
−20
−10
0
V1
Class
0
1



From the above graphs, we can confirm that the two couples of features are inverse correlated (the regression lines for Class = 0 have a negative slope while the regression lines for Class = 1 have a very small negative slope).

Predictive Modelling with 5 models

In [21]:
```
# Predictors and target variables
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 [22]:
```
# Split data in train, test and validation set
train_df, test_df = train_test_split(credit_card_df, test_size=TEST_SIZE, random_st
train_df, valid_df = train_test_split(train_df, test_size=VALID_SIZE, random_state=
```

1) Random Forest Classifier

In [23]:
```
clf = RandomForestClassifier(n_jobs=NO_JOBS,
                             random_state=RANDOM_STATE,
                             criterion=RFC_METRIC,
```

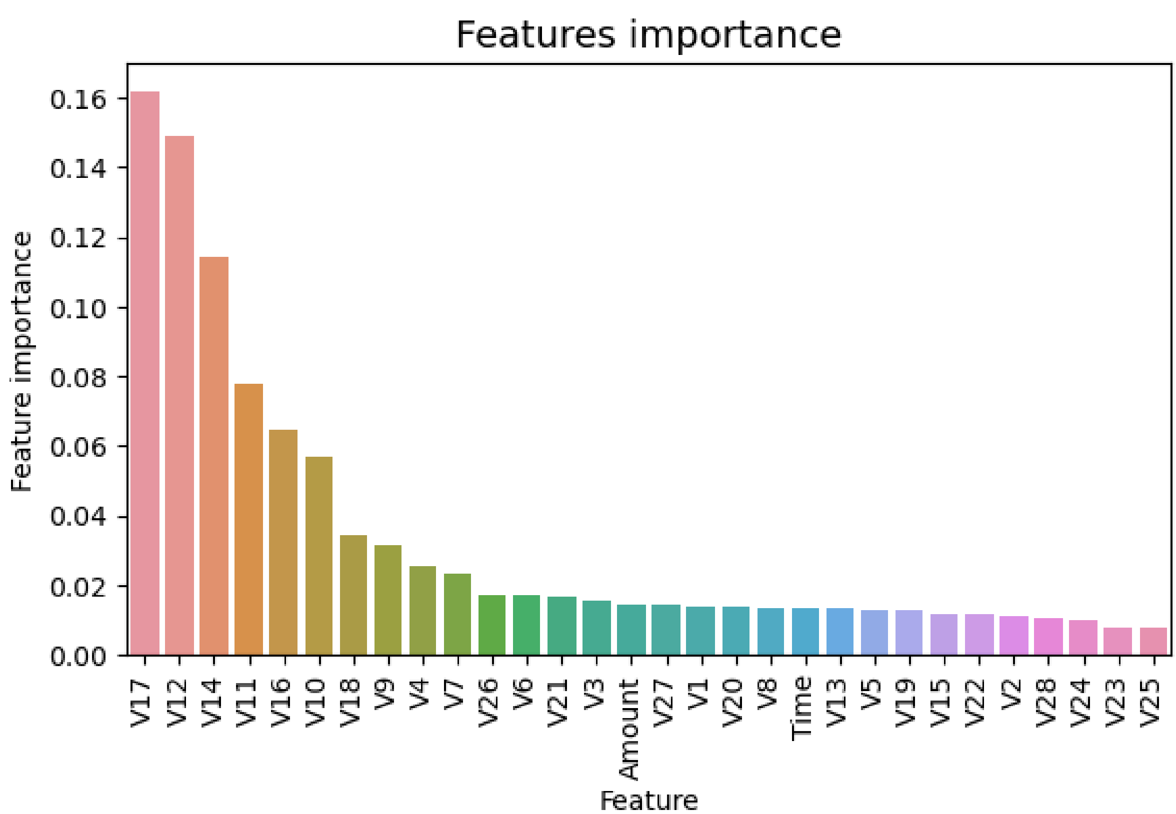
```
                               n_estimators=NUM_ESTIMATORS,
                               verbose=False)
```

In [24]:
```
# Train the model
clf.fit(train_df[predictors], train_df[target].values)
```

Out[24]:
```
RandomForestClassifier
RandomForestClassifier(n_jobs=4, random_state=2018, verbose=False)
```

In [25]:
```
# Predict the model
preds = clf.predict(valid_df[predictors])
```

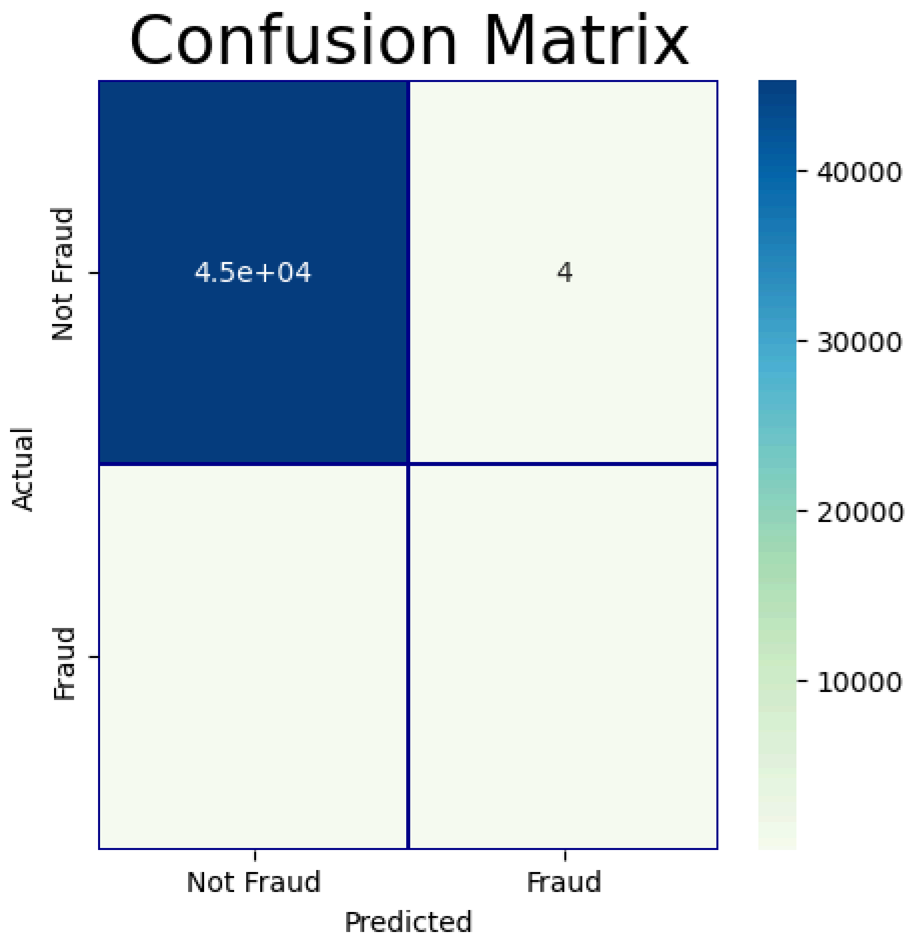
In [26]:
```
# Feature Importance
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.feature_import
tmp = tmp.sort_values(by='Feature importance',ascending=False)
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)
s.set_xticklabels(s.get_xticklabels(),rotation=90)
plt.show()
```



The most important features are V17, V12, V14, V11, V16, V10.

In [27]:
```
# Confusion Matrix

cm = pd.crosstab(valid_df[target].values, preds, rownames=['Actual'], colnames=['Pr
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="GnBu")
plt.title('Confusion Matrix', fontsize=24)
plt.show()
```



In [28]:
```
### Lets calculate the Area Under the Curve
roc_auc_score(valid_df[target].values, preds)
```

Out[28]: 0.8749558654779769

The ROC-AUC score obtained with Random Forest Classifier is 0.87.

2) ADA Boost Classifier

In [29]:
```
# Prepare the model
clf = AdaBoostClassifier(random_state=RANDOM_STATE,
                         algorithm='SAMME.R',
                         learning_rate=0.8,
                             n_estimators=NUM_ESTIMATORS)
```

In [30]:
```
# Fit the model
```

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

Out[30]:

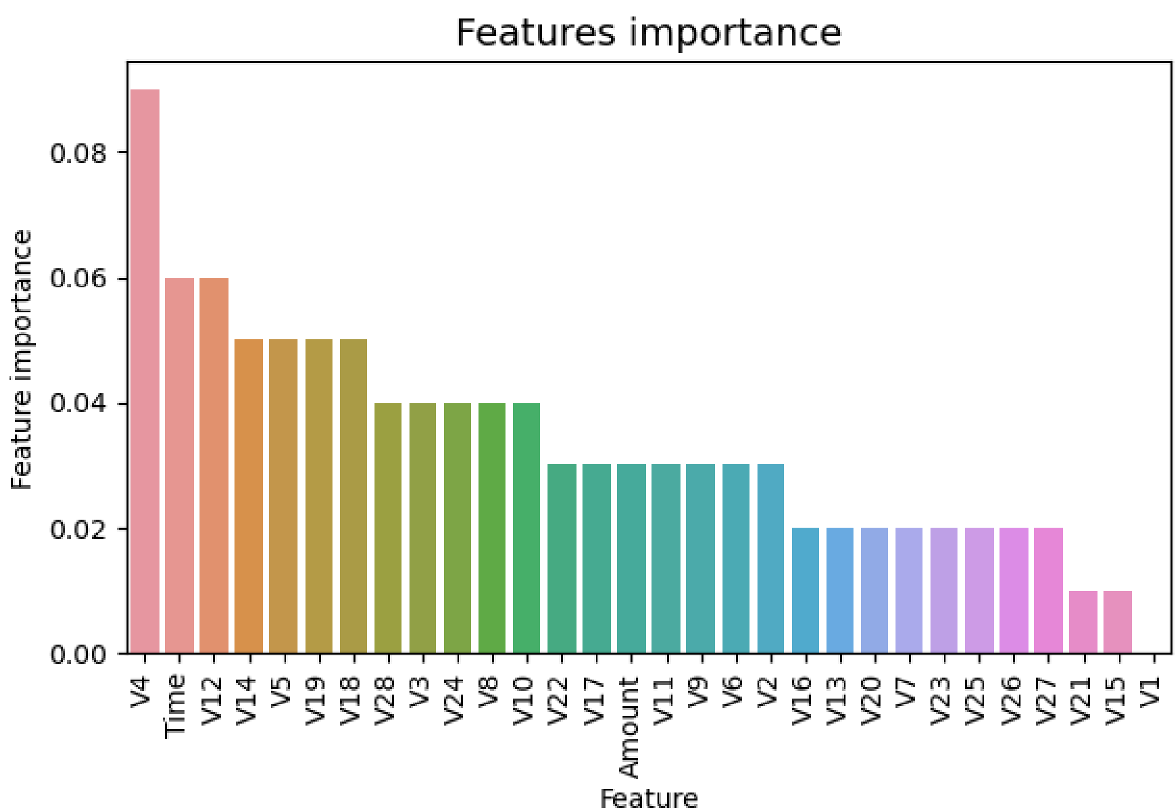
```
AdaBoostClassifier

AdaBoostClassifier(learning_rate=0.8, n_estimators=100, random_state=2018)
```

In [31]:

```
# Pedict the target variable
preds = clf.predict(valid_df[predictors])
```

In [32]:

```
# Feature importance
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.feature_import
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()
```

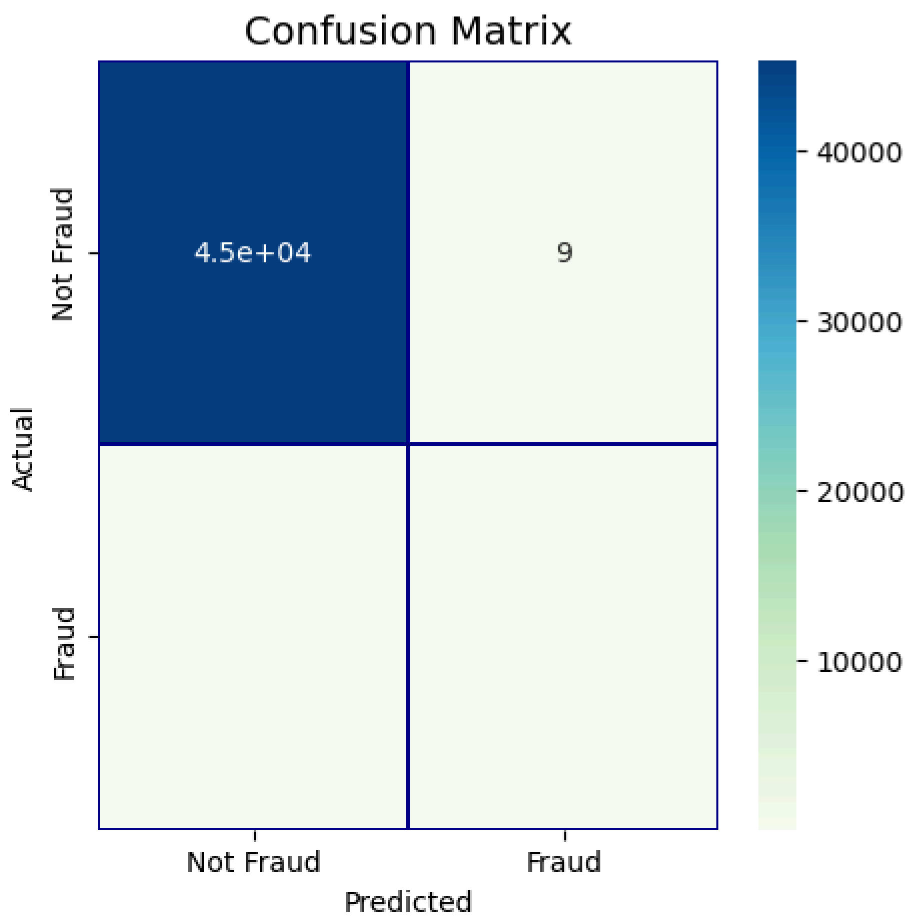


The most important features are V4, Time, V12, V5, V19, V18.

In [33]:

```
# Confusion Matrix
cm = pd.crosstab(valid_df[target].values, preds, rownames=['Actual'], colnames=['Pr
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="GnBu")
```

```
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



In [34]:
```
# Now, lets calculate the Area under curve
roc_auc_score(valid_df[target].values, preds)
```

Out[34]: 0.8561506973254479

The ROC-AUC score obtained with ADA Boost Classifier is 0.86.

3) Cat Boost Classifier

In [35]:
```
# Prepare the model
clf = CatBoostClassifier(iterations=500,
                             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 [36]:
```
# Fit the model
clf.fit(train_df[predictors], train_df[target].values,verbose=True)
```

```
0:      total: 630ms    remaining: 5m 14s
50:     total: 20.1s    remaining: 2m 57s
100:    total: 38.7s    remaining: 2m 32s
150:    total: 55.9s    remaining: 2m 9s
200:    total: 1m 13s   remaining: 1m 49s
250:    total: 1m 30s   remaining: 1m 29s
300:    total: 1m 47s   remaining: 1m 11s
350:    total: 2m 5s    remaining: 53.3s
400:    total: 2m 23s   remaining: 35.4s
450:    total: 2m 40s   remaining: 17.4s
499:    total: 2m 57s   remaining: 0us
```

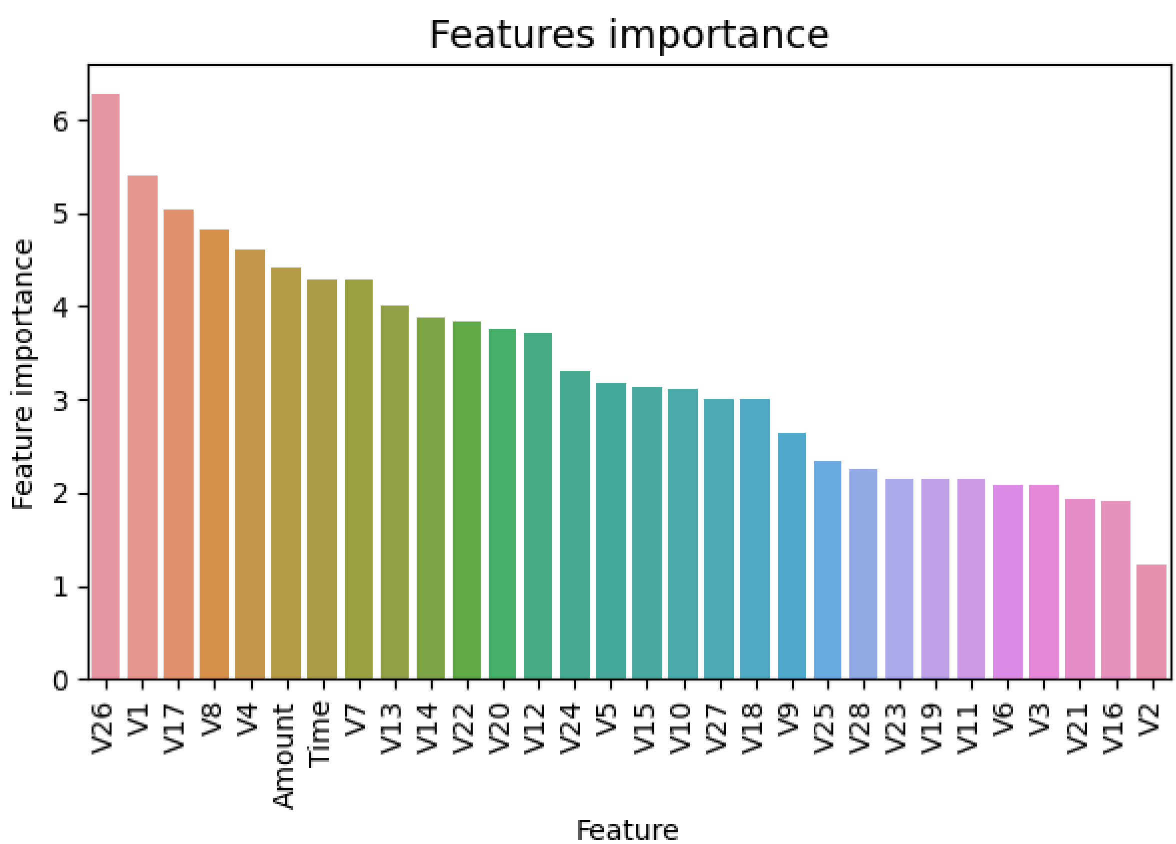
Out[36]: <catboost.core.CatBoostClassifier at 0x2a1aa232940>

In [37]:

```
# Predict the Target Variables
preds = clf.predict(valid_df[predictors])
```

In [38]:

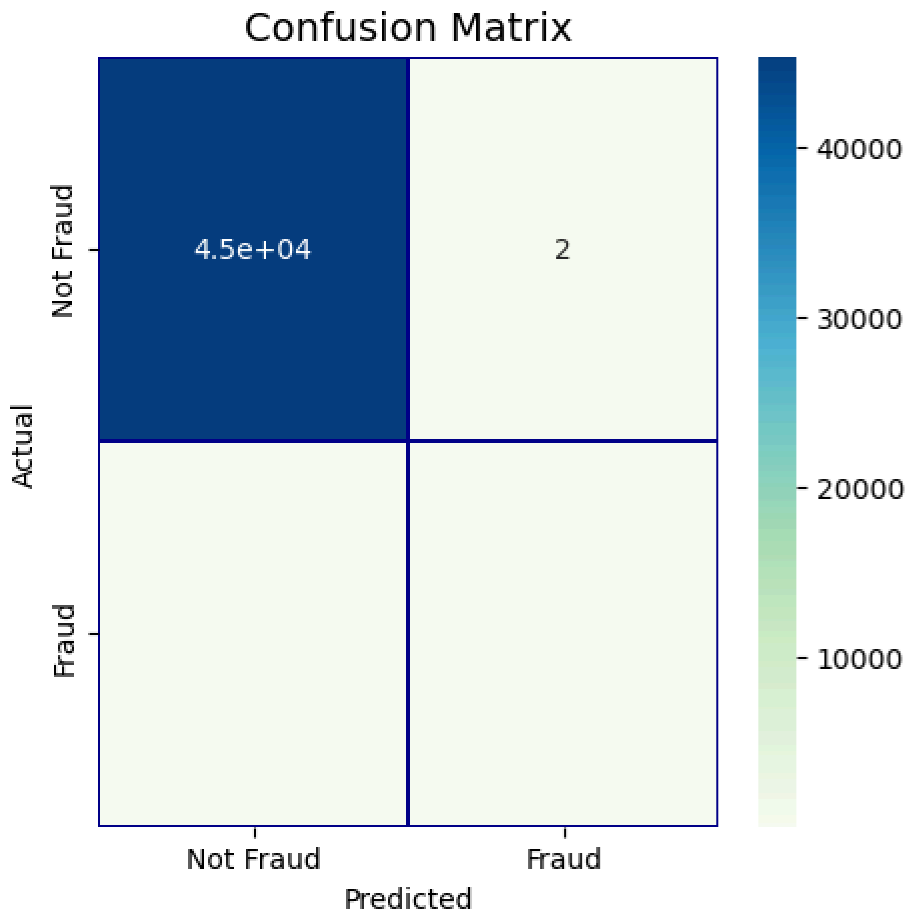
```
# Features Importance
tmp = pd.DataFrame({'Feature': predictors, 'Feature importance': clf.feature_import
tmp = tmp.sort_values(by='Feature importance',ascending=False)
plt.figure(figsize = (7,4))
plt.title('Features importance',fontsize=14)
s = sns.barplot(x='Feature',y='Feature importance',data=tmp)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



In [39]:

```
cm = pd.crosstab(valid_df[target].values, preds, rownames=['Actual'], colnames=['Pr
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="GnBu")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



In [40]:

```
# Now lets Calculate the  ROC-AUC
roc_auc_score(valid_df[target].values, preds)
```

Out[40]: 0.8874779327389885

The ROC-AUC score obtained with CatBoostClassifier is 0.89.

4) XG Boost Classifier

In [41]:

```
# Prepare the model
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)

watchlist = [(dtrain, 'train'), (dvalid, 'valid')]

# Set xgboost parameters
params = {}
params['objective'] = 'binary:logistic'
params['eta'] = 0.039
```

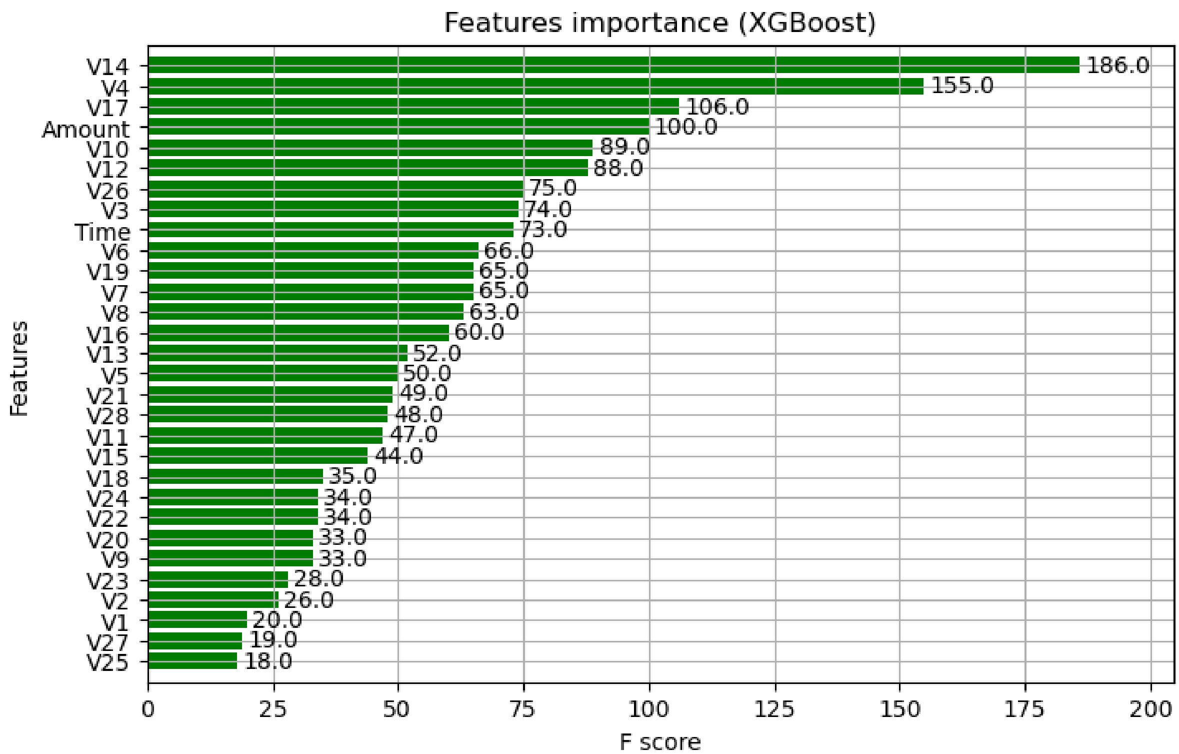
```
params['silent'] = True
params['max_depth'] = 2
params['subsample'] = 0.8
params['colsample_bytree'] = 0.9
params['eval_metric'] = 'auc'
params['random_state'] = RANDOM_STATE
```

In [42]:
```
# Train the model
model = xgb.train(params,
                dtrain,
                MAX_ROUNDS,
                watchlist,
                early_stopping_rounds=EARLY_STOP,
                maximize=True,
                verbose_eval=VERBOSE_EVAL)
```

```
[0]	train-auc:0.91740	valid-auc:0.91810
[50]	train-auc:0.92153	valid-auc:0.92958
[100]	train-auc:0.96205	valid-auc:0.97771
[150]	train-auc:0.98758	valid-auc:0.98501
[200]	train-auc:0.99166	valid-auc:0.98680
[250]	train-auc:0.99373	valid-auc:0.98774
[300]	train-auc:0.99520	valid-auc:0.98924
[350]	train-auc:0.99609	valid-auc:0.98936
[400]	train-auc:0.99688	valid-auc:0.99048
[450]	train-auc:0.99755	valid-auc:0.99055
[500]	train-auc:0.99800	valid-auc:0.99127
[550]	train-auc:0.99844	valid-auc:0.99169
[600]	train-auc:0.99870	valid-auc:0.99134
[625]	train-auc:0.99880	valid-auc:0.99150
```

The best validation score (ROC-AUC) was 0.99

In [43]:
```
# Feature Importance
fig, (ax) = plt.subplots(ncols=1, figsize=(8,5))
xgb.plot_importance(model, height=0.8, title="Features importance (XGBoost)", ax=ax
plt.show()
```



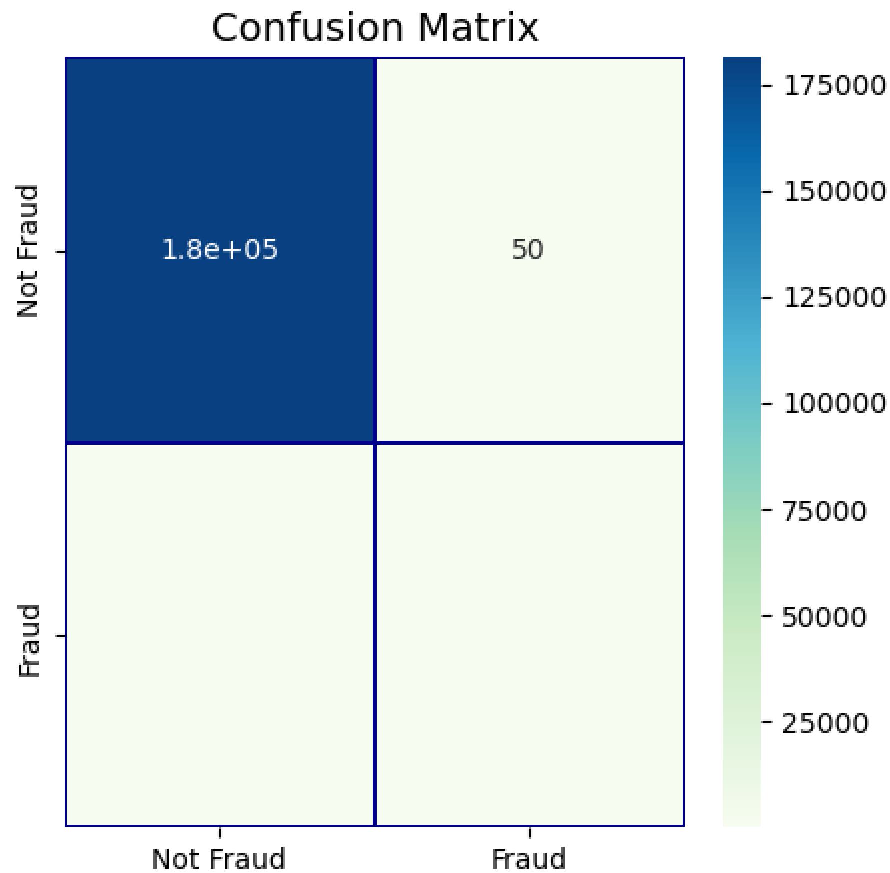
In [44]:

```
# Predict the model
preds = model.predict(dtest)
```

In [45]:

```
import xgboost
X_val = train_df[predictors]
y_val =  train_df[target].values
xgb_clf = xgboost.XGBRFClassifier(max_depth=4, random_state=1)
xgb_clf.fit(X_val,y_val)

y_pred=pd.DataFrame(xgb_clf.predict_proba(X_val))[1].values
roc_auc_score(y_val,y_pred)

cm = confusion_matrix(y_val, xgb_clf.predict(X_val))


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

cm = confusion_matrix(y_val, xgb_clf.predict(X_val))

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="GnBu")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```

```
0:	total: 447ms	remaining: 3m 42s
50:	total: 19.5s	remaining: 2m 51s
100:	total: 37.2s	remaining: 2m 26s
150:	total: 54.5s	remaining: 2m 5s
200:	total: 1m 11s	remaining: 1m 46s
250:	total: 1m 28s	remaining: 1m 28s
300:	total: 1m 46s	remaining: 1m 10s
350:	total: 2m 3s	remaining: 52.4s
400:	total: 2m 21s	remaining: 34.9s
450:	total: 2m 40s	remaining: 17.4s
499:	total: 3m 2s	remaining: 0us
```



In [46]:
```
#Lets calculate the Area Under the curve
roc_auc_score(y_val,y_pred)
```

Out[46]: 0.9814884101161215

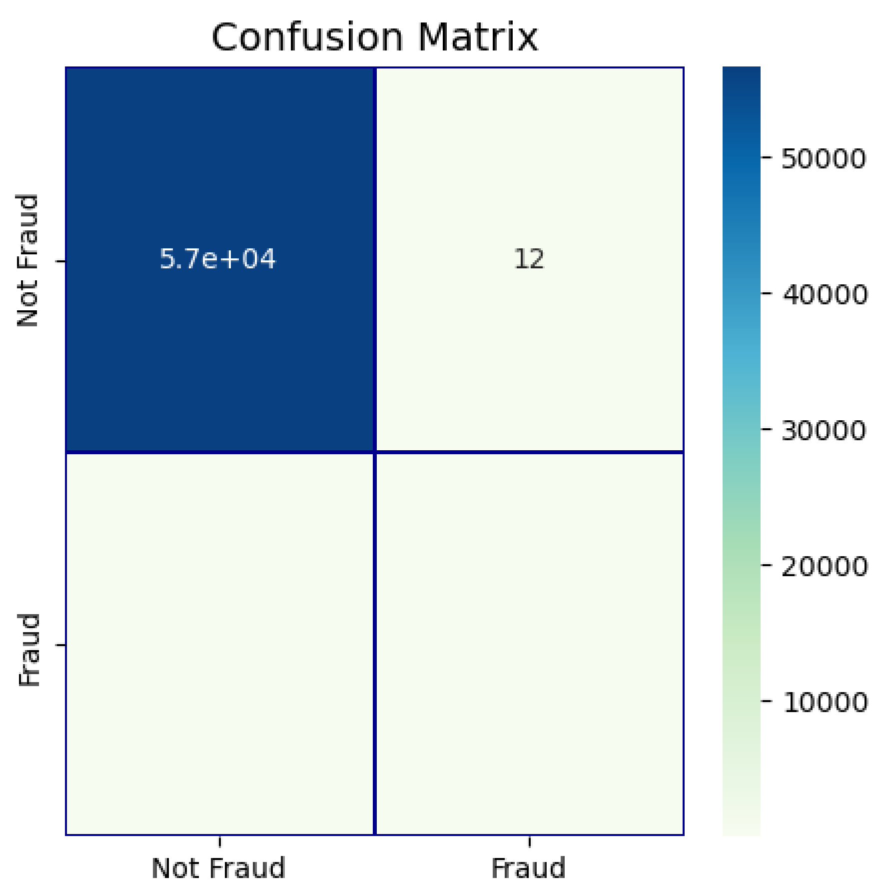
The ROC-AUC score obtained with XG Boost Classifier is 0.98.

5) Logistic Regression

In [48]:
```
from sklearn.linear_model import LogisticRegression
x_test = test_df[predictors]
y_test  = test_df[target].values

logreg=LogisticRegression()
```

```
logreg.fit(X_val,y_val)
y_pred=logreg.predict(x_test)
```

In [49]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)

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="GnBu")
plt.title('Confusion Matrix', fontsize=14)
plt.show()
```



In [50]:
```
#Lets calculate the Area Under the curve
y_pred_prob_yes=logreg.predict_proba(x_test)
roc_auc_score(y_test,y_pred_prob_yes[:,1])
```

Out[50]: 0.9405809765914627

The ROC-AUC score obtained with Logistic Regression is 0.94.