Credit Card Fraudulent Detection

Context:

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Content:

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation.Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Importing The Libraries

In [17]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

Data Importing

In [31]:

```
data=pd.read_csv('creditcard.csv')
data.head()
```

Out[31]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns

Data Exploration

In [3]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [4]:

```
data.isnull().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
```

```
V28       0
Amount    0
Class     0
dtype: int64
```

In [5]:

```
data.isnull().values.any()
```

Out[5]:

```
False
```

Zero null values found in the dataset.

In [6]:

```
data.size
```

Out[6]:

```
8829017
```

In [7]:

```
data.shape
```

Out[7]:

```
(284807, 31)
```

The dataset contains rows=284807 and columns=31

In [8]:

```
data.columns
```

Out[8]:

```
Index(['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',
       'Class'],
      dtype='object')
```

In [9]:

```
Fraud = data[data['Class']==1]

Normal = data[data['Class']==0]
```

In [10]:

```
print(Fraud.shape,Normal.shape)
```

```
(492, 31) (284315, 31)
```

There are 492 Fraud cases present in the dataset.

In [11]:

```
Fraud.Amount.describe()
```

Out[11]:

```
count     492.000000
mean      122.211321
```

```
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 [12]:

```
Normal.Amount.describe()
```

Out[12]:

```
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
```

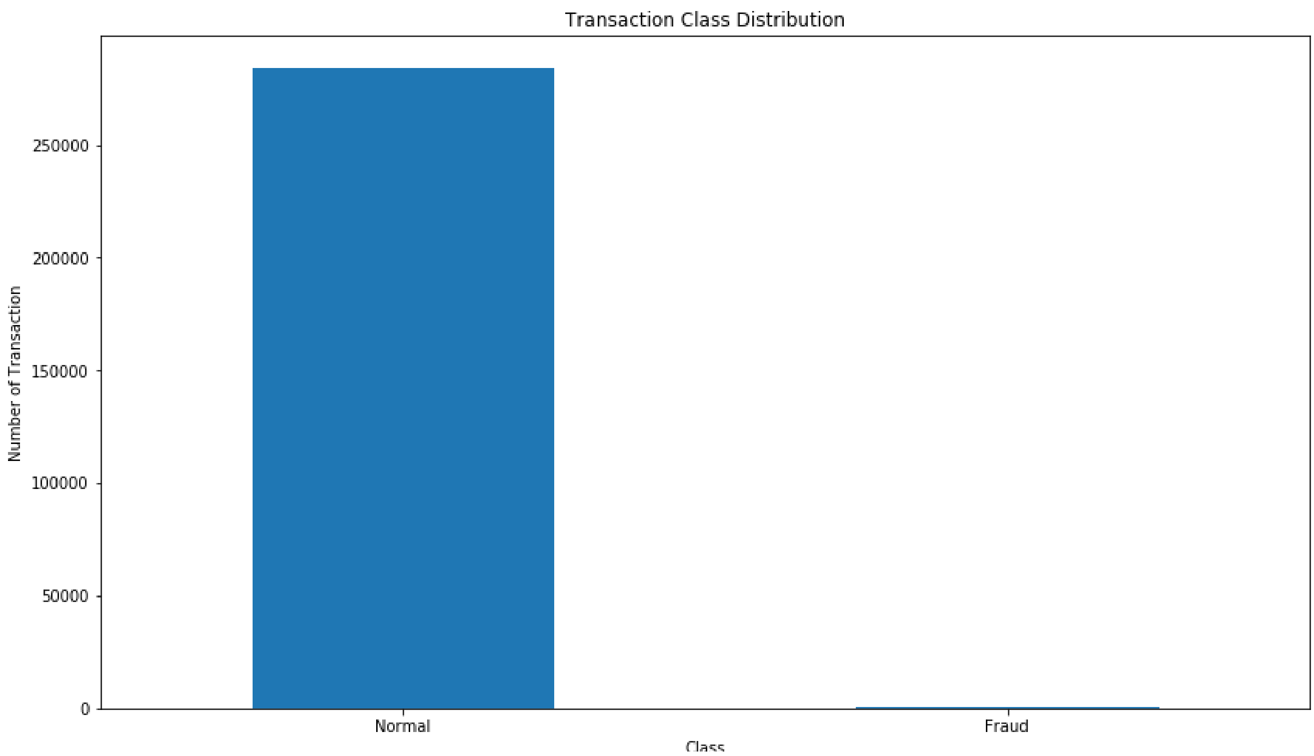
Data Visualization

In [18]:

```
Classes = pd.value_counts(data['Class'])

Classes.plot(kind = 'bar', rot=0)

plt.title("Transaction Class Distribution")

plt.xticks(range(2), ['Normal','Fraud'])

plt.xlabel("Class")

plt.ylabel("Number of Transaction")
```

Out[18]:

```
Text(0, 0.5, 'Number of Transaction')
```

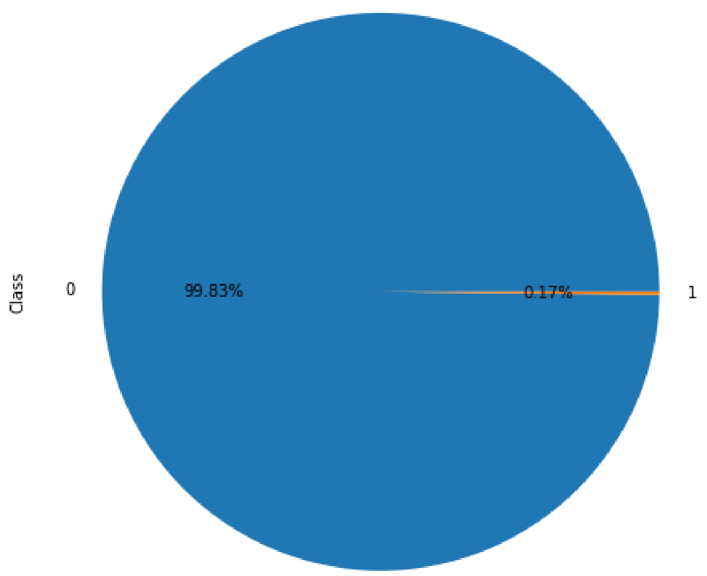


In [19]:

```python
data['Class'].value_counts().plot(kind='pie',autopct='%1.2f%%')
```

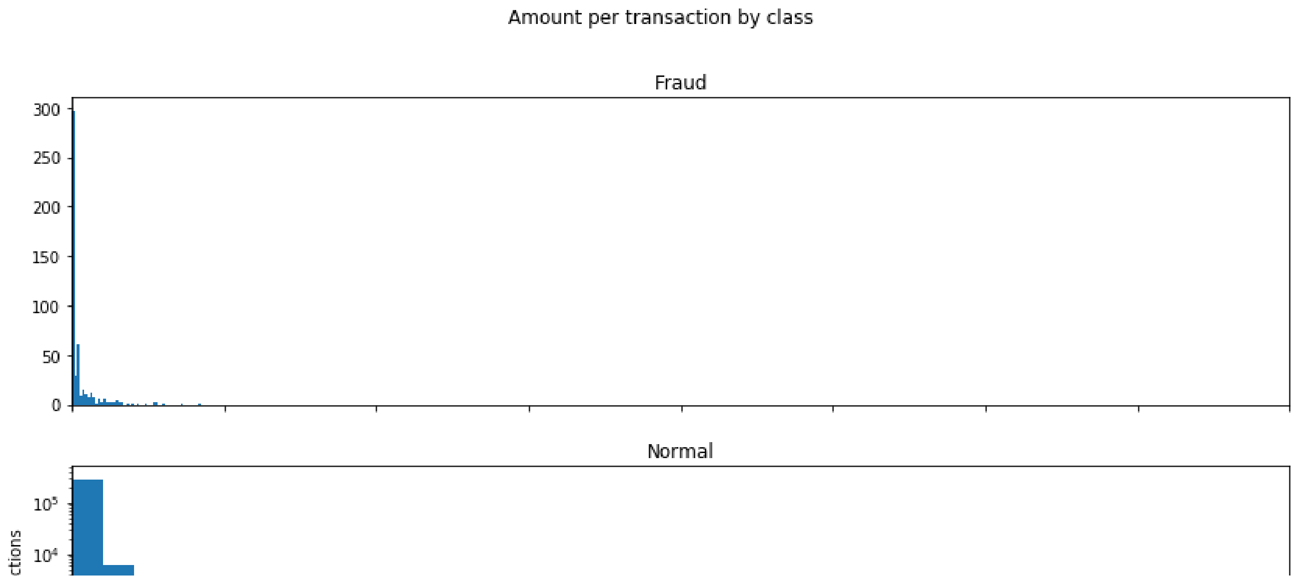
Out[19]:

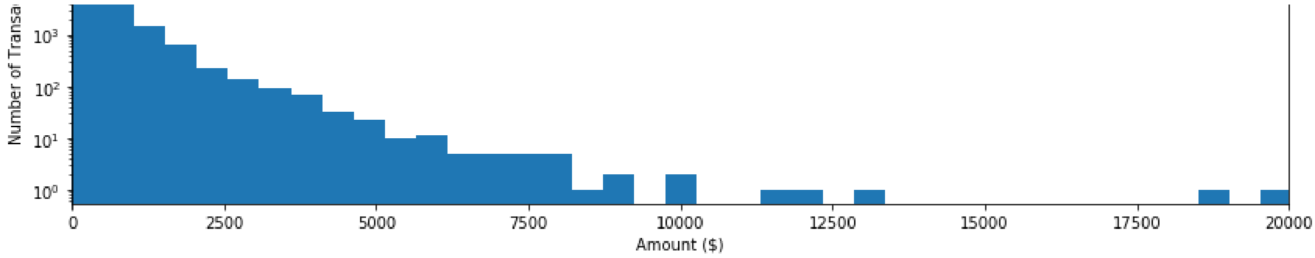
```
<matplotlib.axes._subplots.AxesSubplot at 0x28bcfa396d8>
```



In [20]:

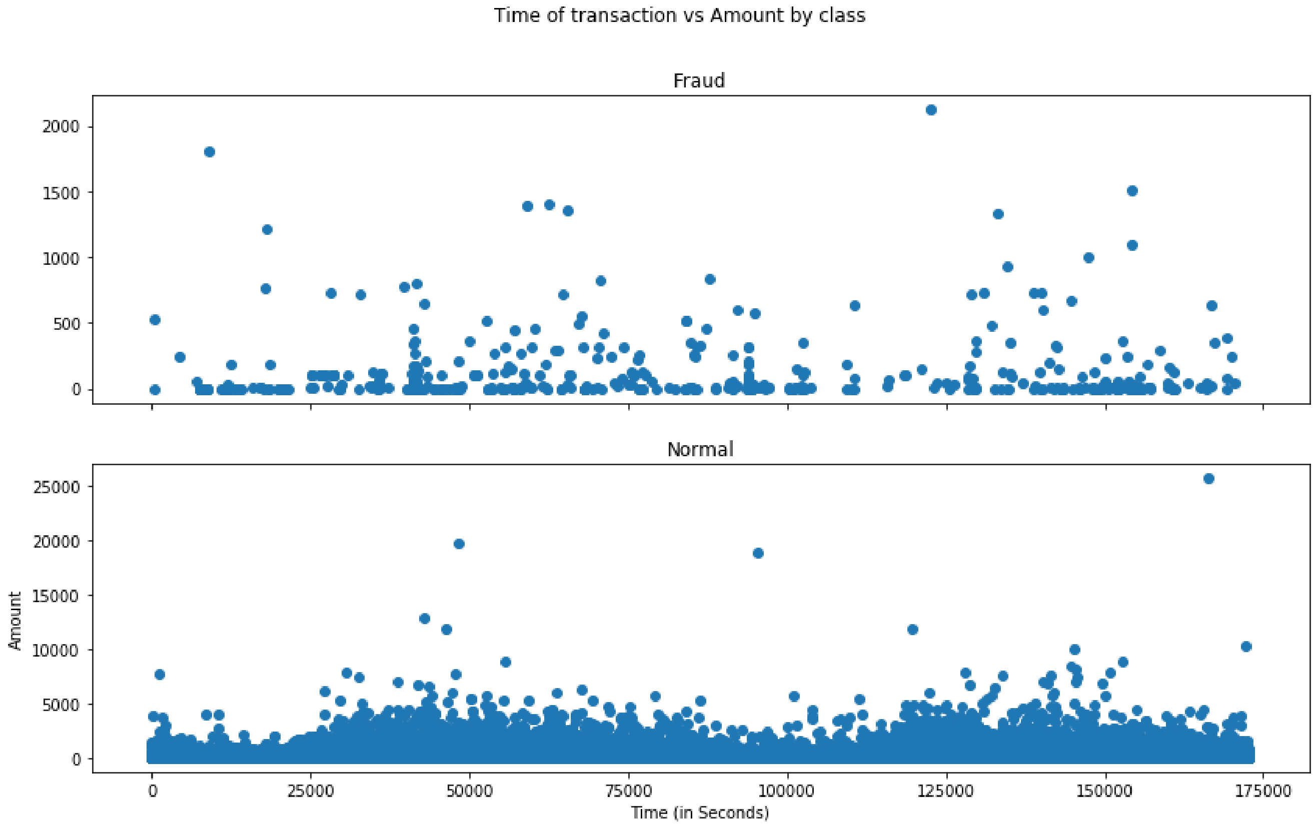
```python
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(Fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(Normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```





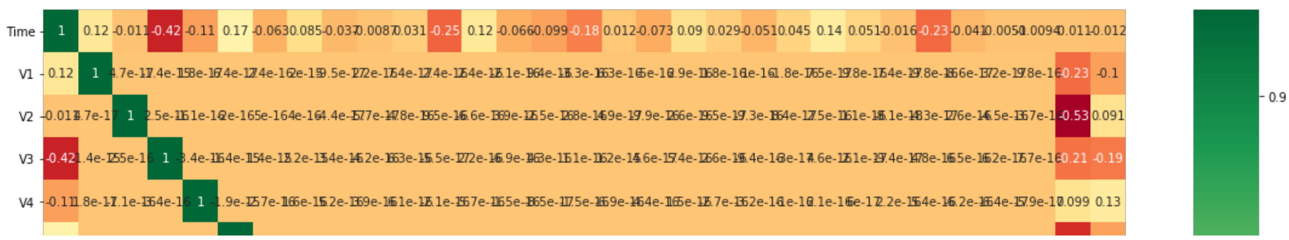
In [21]:

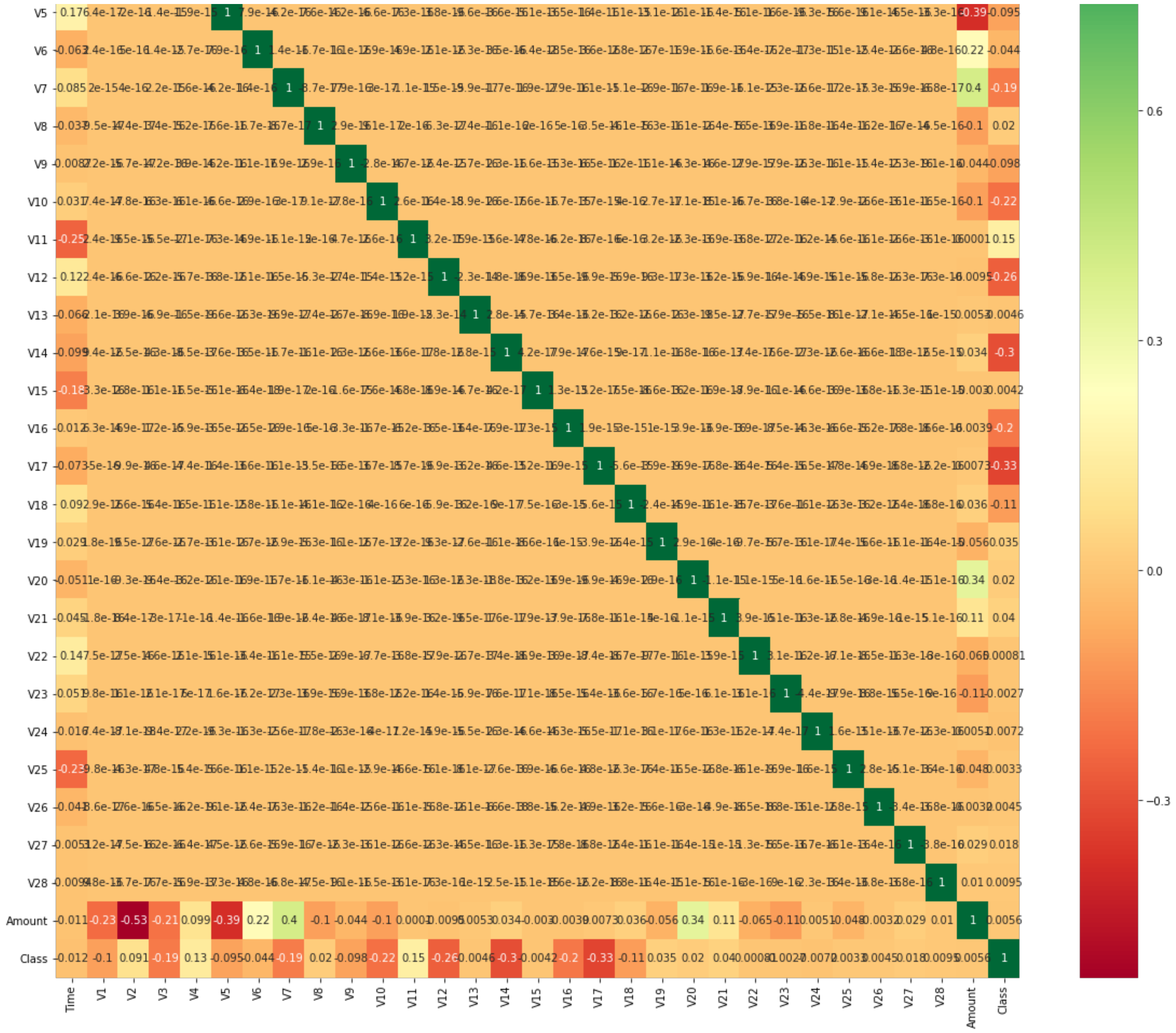
```python
# We Will check Do fraudulent transactions occur more often during certain time frame.

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(Fraud.Time, Fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```



In [22]:

```python
#Get correlations of each features in dataset
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```





Data Preprocessing:

Lets take the 20% of the dataset and apply different algorithms and check the accuracy.

In [23]:

```
data1= data.sample(frac = 0.2,random_state=1)

data1.shape
```

Out[23]:

```
(56961, 31)
```

In [24]:

```
X=data1.iloc[:,1:-1].values
y=data1.iloc[:,-1].values
```

In [25]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0,test_size=0.2)
```

Model Prediction:

Now it is time to start building the model . The types of algorithms we are going to use to try to do anomaly detection on this dataset are as follows:

1) Logistic Regression:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression[1] (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value).

In [26]:

```
classifier1=LogisticRegression(random_state=0)
classifier1.fit(X_train,y_train)
```

```
C:\Users\NILESH\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: D
efault solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[26]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=0, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

2) Support Vector Classifier:

In machine learning, support-vector machines (SVMs, also support-vector networks[1]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In [27]:

```
classifier2=SVC(kernel='rbf',random_state=0)
classifier2.fit(X_train,y_train)
```

```
C:\Users\NILESH\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default va
lue of gamma will change from 'auto' to 'scale' in version 0.22 to account better for unscaled fea
tures. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[27]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=0,
  shrinking=True, tol=0.001, verbose=False)
```

3) Random Forest Classifier:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.[1][2] Random decision forests correct for decision trees' habit of overfitting to their training set. Parameters: 1)n_estimators :default=100 The number of trees in the forest. 2)criterion{“gini”, “entropy”}, default="gini” The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Note: this parameter is tree-specific.

In [28]:

```
classifier3=RandomForestClassifier(n_estimators=100,criterion='gini',random_state=0)
classifier3.fit(X_train,y_train)
```

Out[28]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            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=0, verbose=0, warm_start=False)
```

In [29]:

```
y_pred1=classifier1.predict(X_test)
y_pred2=classifier2.predict(X_test)
y_pred3=classifier3.predict(X_test)
```

Classification Report and Accuracy Check:

In [30]:

```
print("1.Logistic Regression:")
print("Number of Errors in Logistic Regression:",(y_pred1 != y_test).sum())
print("Accuracy Score:",accuracy_score(y_test,y_pred1))
print("Classification Report :")
print(classification_report(y_test,y_pred1))

print("2.Support Vector Classifier:")
print("Number of Errors in SVC:",(y_pred2 != y_test).sum())
print("Accuracy Score:",accuracy_score(y_test,y_pred2))
print("Classification Report :")
print(classification_report(y_test,y_pred2))

print("3.Random Forest Classifier:")
print("Number of Errors in Random Forest Classifier:",(y_pred3 != y_test).sum())
print("Accuracy Score:",accuracy_score(y_test,y_pred3))
print("Classification Report :")
print(classification_report(y_test,y_pred3))
```

```
1.Logistic Regression:
Number of Errors in Logistic Regression: 9
Accuracy Score: 0.9992100412534012
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11376
           1       0.79      0.65      0.71        17

   micro avg       1.00      1.00      1.00     11393
   macro avg       0.89      0.82      0.85     11393
weighted avg       1.00      1.00      1.00     11393

2.Support Vector Classifier:
Number of Errors in SVC: 16
Accuracy Score: 0.9985956288949355
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11376
           1       1.00      0.06      0.11        17

   micro avg       1.00      1.00      1.00     11393
   macro avg       1.00      0.53      0.56     11393
weighted avg       1.00      1.00      1.00     11393
```

```
3.Random Forest Classifier:
Number of Errors in Random Forest Classifier: 5
Accuracy Score: 0.9995611340296673
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     11376
           1       0.93      0.76      0.84        17

   micro avg       1.00      1.00      1.00     11393
   macro avg       0.96      0.88      0.92     11393
weighted avg       1.00      1.00      1.00     11393
```

Observation and Conclusion:

1)Number of errors in logistic regression is 9 while in svm is 16 and randomforest contains only 5 errors 2)accuracy of random forest is slightly better than svm and logistic regression. 3)When comparing error precision & recall for 3 models , random forest is better than svm and logistic regression. 4)So random forest classifier is better at predicting the credit card fraudalent detection. 5)We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases