How can a credit card Fraud happen?

Some of the most common ways it may happen are:

1. Firstly and most ostensibly when your card details are overseen by some other person.
2. When your card is lost or stolen and the person possessing it knows how to get things done.
3. Fake phone call convincing you to share the details.
4. And lastly and most improbably, a high-level hacking of the bank account details.

Main challenges involved in credit card fraud detection are:

1. Enormous Data is processed every day and the model build must be fast enough to respond to the scam in time.
2. Imbalanced Data i.e most of the transactions(99.8%) are not fraudulent which makes it really hard for detecting the fraudulent ones
3. Data availability as the data is mostly private.
4. Misclassified Data can be another major issue, as not every fraudulent transaction is caught and reported.
5. And last but not the least, Adaptive techniques used against the model by the scammers.

How to tackle these challenges?

1. The model used must be simple and fast enough to detect the anomaly and classify it as a fraudulent transaction as quickly as possible.
2. Imbalance can be dealt with by properly using some methods which we will talk about in the next paragraph
3. For protecting the privacy of the user the dimensionality of the data can be reduced.
4. A more trustworthy source must be taken which double-check the data, at least for training the model.
5. We can make the model simple and interpretable so that when the scammer adapts to it with just some tweaks we can have a new model up and running to deploy.

Dealing with Imbalance

We will see in the later parts of the article that the data we received is highly imbalanced i.e only 0.17% of the total Credit Card transaction is fraudulent. Well, a class imbalance is a very common problem in real life and needs to be handled before applying any algorithm to it.

There are three common ways to deal with the imbalance of Data

1. Undersampling- One-sided sampling by Kubat and Matwin(ICML 1997)
2. Oversampling-SMOTE(Synthetic Minority Oversampling Technique)
3. Combining the above two.

For those of you who are wondering if the fraudulent transaction is so rare why even bother, well here is another fact. The amount of money involved in the fraudulent transaction reaches Billions of USD and by increasing the specificity to 0.1% we can save Millions of USD. Whereas higher Sensitivity means fewer people harassed

Import all Necessary Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec
import pickle
```

Load the Dataset

In [2]:
```
data = pd.read_csv('C:/Users/sundara.rao.ext/Desktop/SUNDAR/Data Science/Socia
l Prachar Material/Codes With Examples - Python/creditcard.csv')
```

Understanding the Data

In [3]:
```
# view the first 5 values
data.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [4]:
```
# view the last 5 values
data.tail()
```

Out[4]:

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

5 rows × 31 columns

In [5]:
```
# View the shape of data set
data.shape
```

Out[5]:
```
(284807, 31)
```

In [10]:
```
# view the type of data
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
```

Check if any missing values in the dataset

In [11]:
```
data.isnull().any()
```

Out[11]:
```
Time      False
V1        False
V2        False
V3        False
V4        False
V5        False
V6        False
V7        False
V8        False
V9        False
V10       False
V11       False
V12       False
V13       False
V14       False
V15       False
V16       False
V17       False
V18       False
V19       False
V20       False
V21       False
V22       False
V23       False
V24       False
V25       False
V26       False
V27       False
V28       False
Amount    False
Class     False
dtype: bool
```

In [12]:
```
data.isnull().sum()
```

Out[12]:
```
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
```

Exploratory Data Analysis (EDA)

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

Out[7]:

	Time	V1	V2	V3	V4	V
count	8.000000	8.000000	8.000000	8.000000	8.000000	8.00000
mean	109764.375691	35594.427437	35594.782996	35596.236238	35602.435362	35591.1630
std	88923.633614	100697.087720	100696.945914	100696.356440	100693.850245	100698.4141
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.7433
25%	52523.161489	-0.230093	-0.149637	-0.222591	-0.227045	-0.2136
50%	89752.929788	0.666875	0.434605	0.603521	0.371671	0.3059
75%	147688.375000	2.082754	6.752914	3.482831	5.280737	9.7356
max	284807.000000	284807.000000	284807.000000	284807.000000	284807.000000	284807.00000

8 rows × 31 columns

In [9]:

```python
# EDA using Datasist library
import datasist as ds
ds.structdata.describe(data)
```

```
First five data points
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

```
Last five data points
```

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

5 rows × 31 columns

```
Shape of  data set: (284807, 31)


Size of  data set: 8829017


Data Types
Note: All Non-numerical features are identified as objects in pandas
```

	Data Type
Time	float64
V1	float64
V2	float64
V3	float64
V4	float64
V5	float64
V6	float64
V7	float64
V8	float64
V9	float64
V10	float64
V11	float64
V12	float64
V13	float64
V14	float64
V15	float64
V16	float64
V17	float64
V18	float64
V19	float64
V20	float64
V21	float64
V22	float64
V23	float64
V24	float64
V25	float64
V26	float64
V27	float64
V28	float64
Amount	float64
Class	int64

```
Numerical Features in Data set
['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']
```

```
Statistical Description of Columns
```

	Time	V1	V2	V3	V4	V5
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01

8 rows × 31 columns

```
Categorical Features in Data set
[]
```

```
Unique class Count of Categorical features
```

Feature	Unique Count

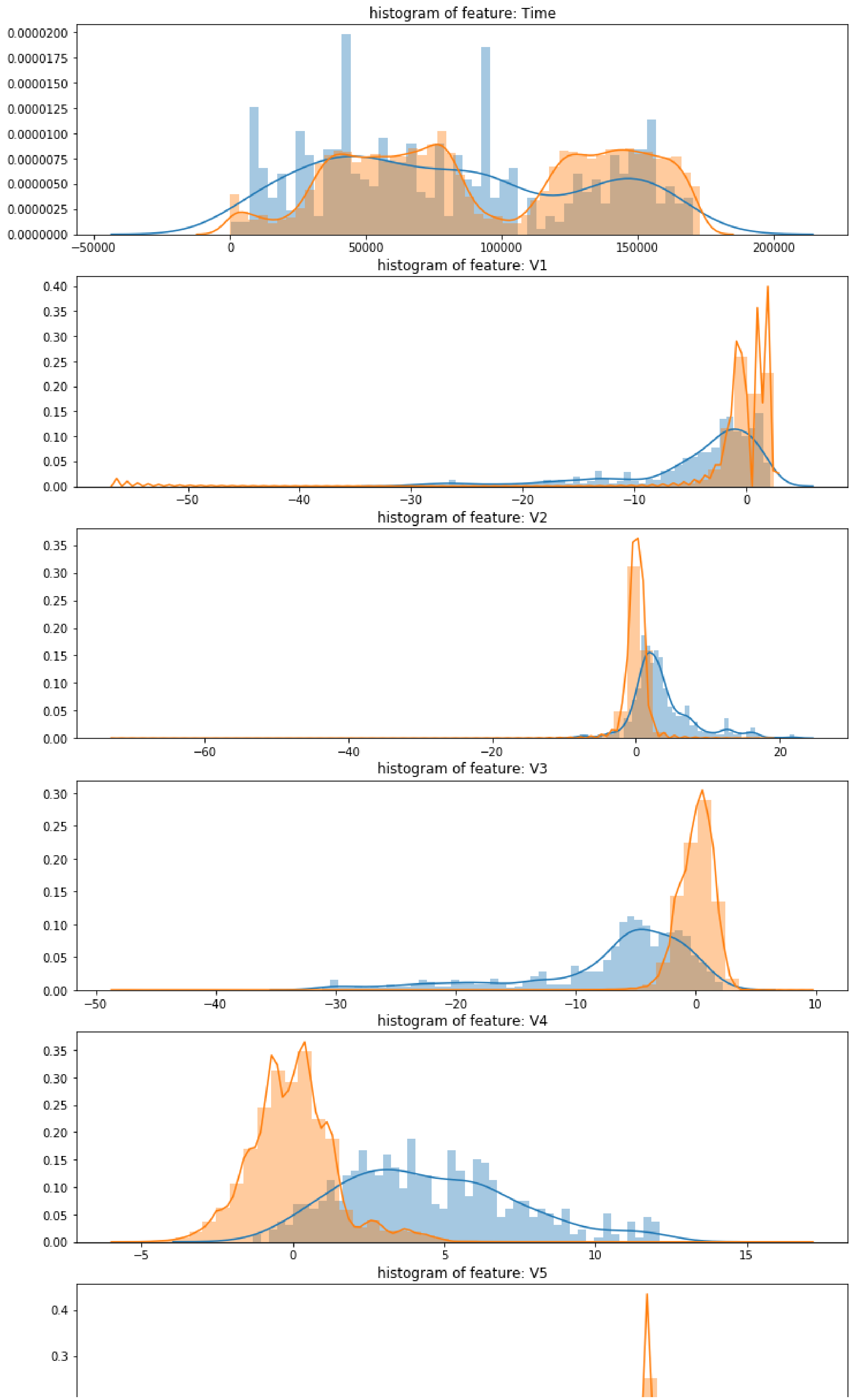
```
Missing Values in Data
```

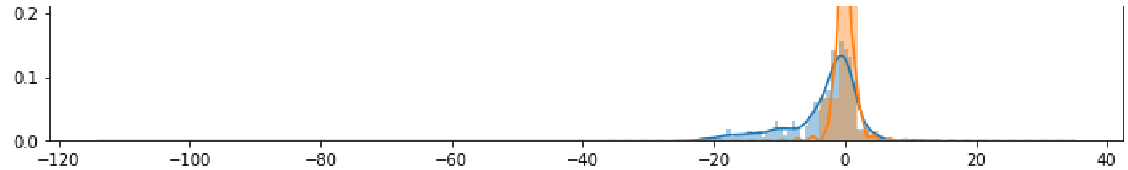
	features	missing_counts	missing_percent
0	Time	0	0.0
1	V1	0	0.0
2	V2	0	0.0
3	V3	0	0.0
4	V4	0	0.0
5	V5	0	0.0
6	V6	0	0.0
7	V7	0	0.0
8	V8	0	0.0
9	V9	0	0.0
10	V10	0	0.0
11	V11	0	0.0
12	V12	0	0.0
13	V13	0	0.0
14	V14	0	0.0
15	V15	0	0.0
16	V16	0	0.0
17	V17	0	0.0
18	V18	0	0.0
19	V19	0	0.0
20	V20	0	0.0
21	V21	0	0.0
22	V22	0	0.0
23	V23	0	0.0
24	V24	0	0.0
25	V25	0	0.0
26	V26	0	0.0
27	V27	0	0.0
28	V28	0	0.0
29	Amount	0	0.0
30	Class	0	0.0

Data Visulizations

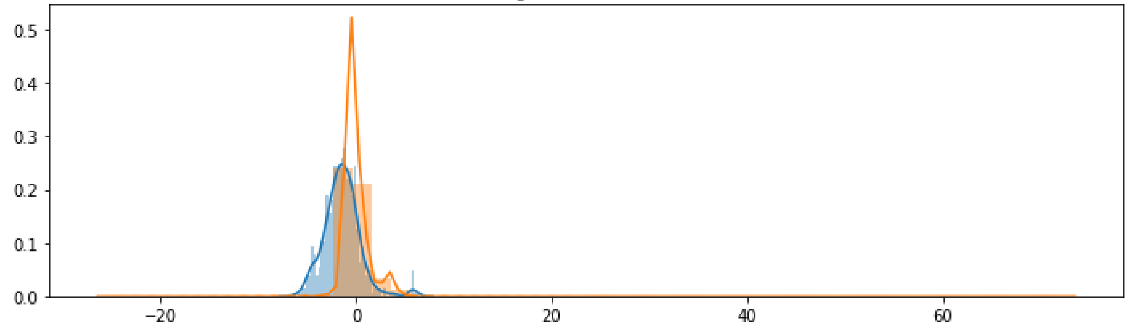
In [15]:

```
# distribution of anomalous features
features = data.iloc[:,0:28].columns
plt.figure(figsize=(12,28*4))
gs = gridspec.GridSpec(28, 1)
for i, c in enumerate(data[features]):
 ax = plt.subplot(gs[i])
 sns.distplot(data[c][data.Class == 1], bins=50)
 sns.distplot(data[c][data.Class == 0], bins=50)
 ax.set_xlabel('')
 ax.set_title('histogram of feature: ' + str(c))
plt.show()
```

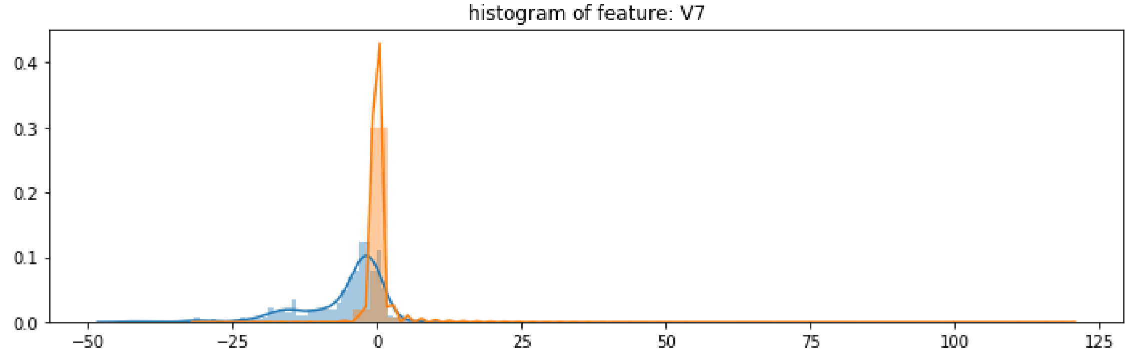

histogram of feature: Time
0.0000200
0.0000175
0.0000150
0.0000125
0.0000100
0.0000075
0.0000050
0.0000025
0.0000000
−50000
0
50000
100000
150000
200000
histogram of feature: V1
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−50
−40
−30
−20
−10
0
histogram of feature: V2
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20
histogram of feature: V3
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−50
−40
−30
−20
−10
0
10
histogram of feature: V4
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15
histogram of feature: V5
0.4
0.3



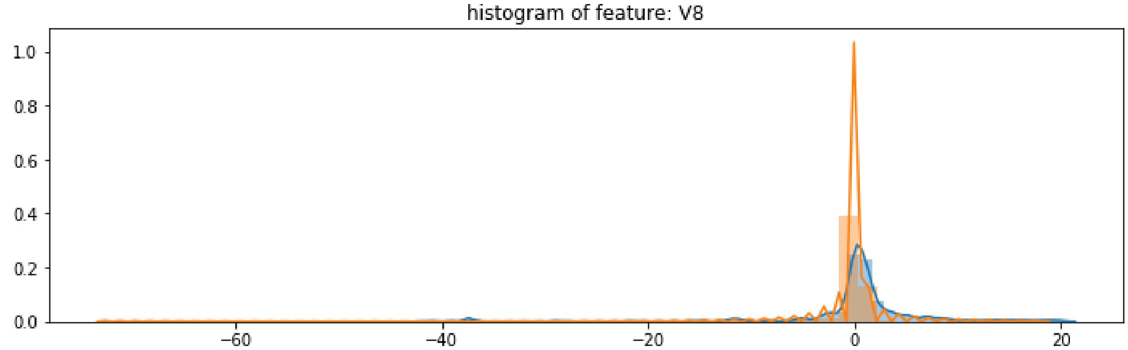
histogram of feature: V6



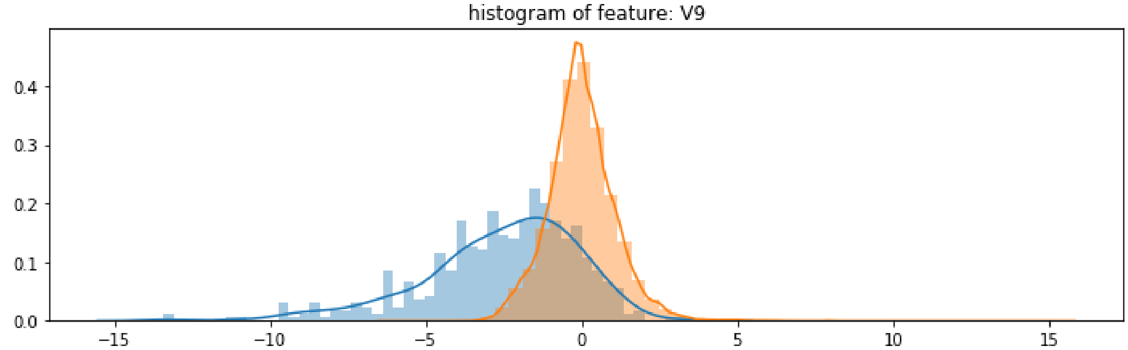
histogram of feature: V7



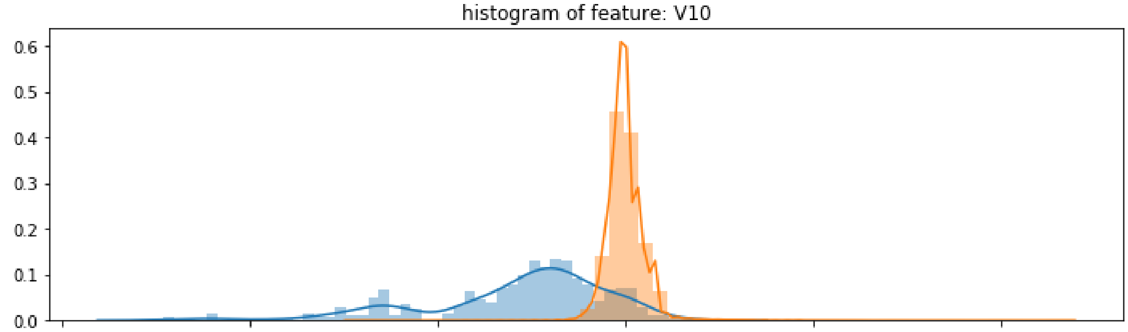
histogram of feature: V8



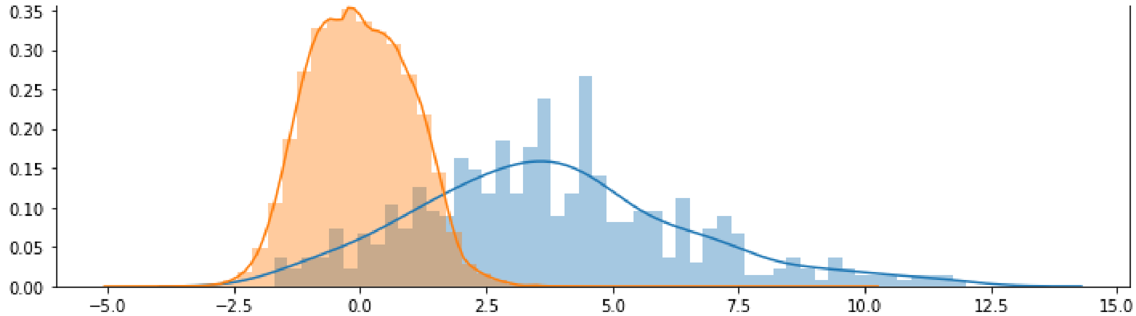
histogram of feature: V9



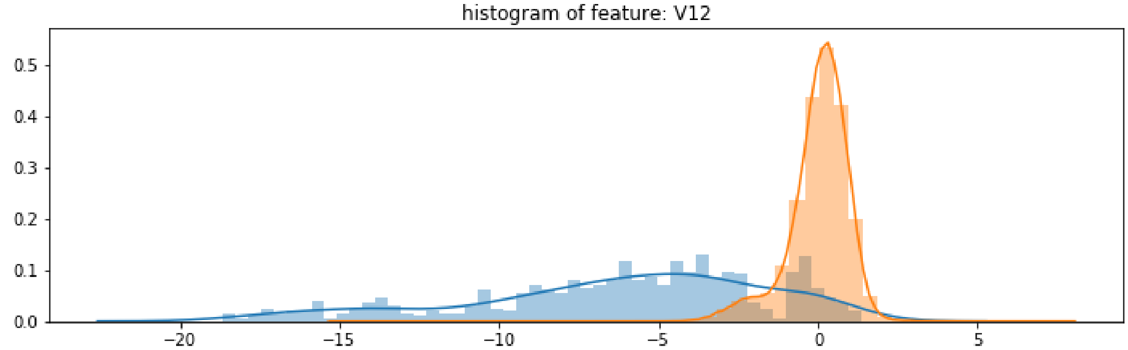
histogram of feature: V10



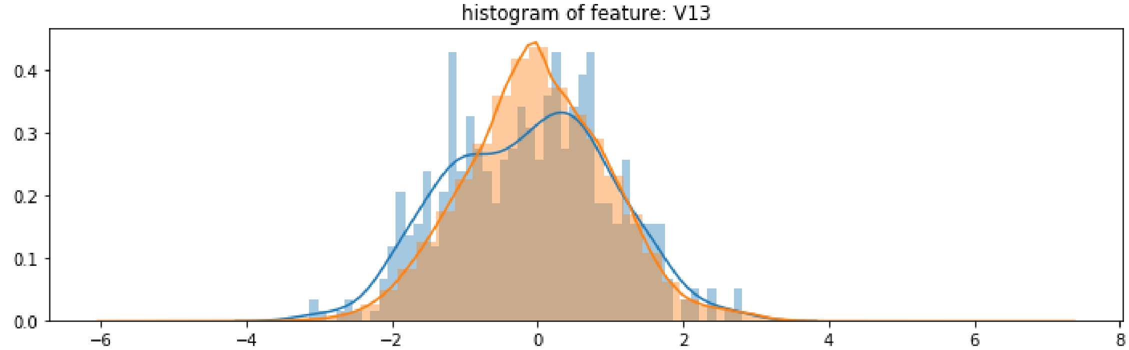
histogram of feature: V11



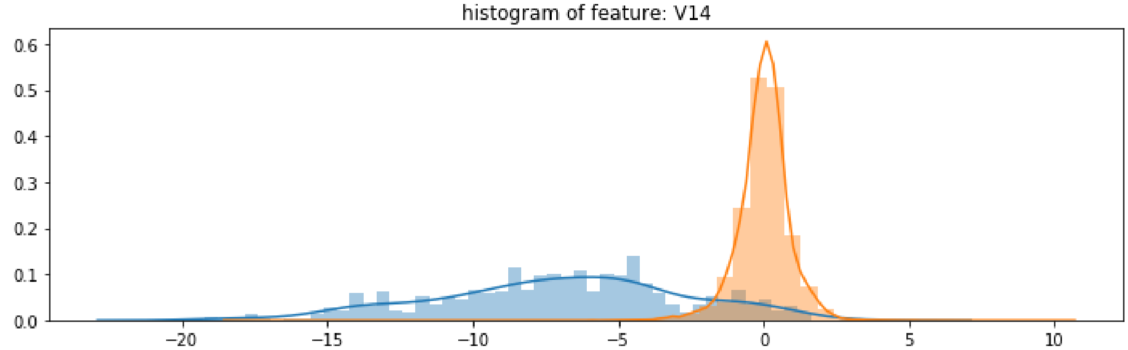
histogram of feature: V12



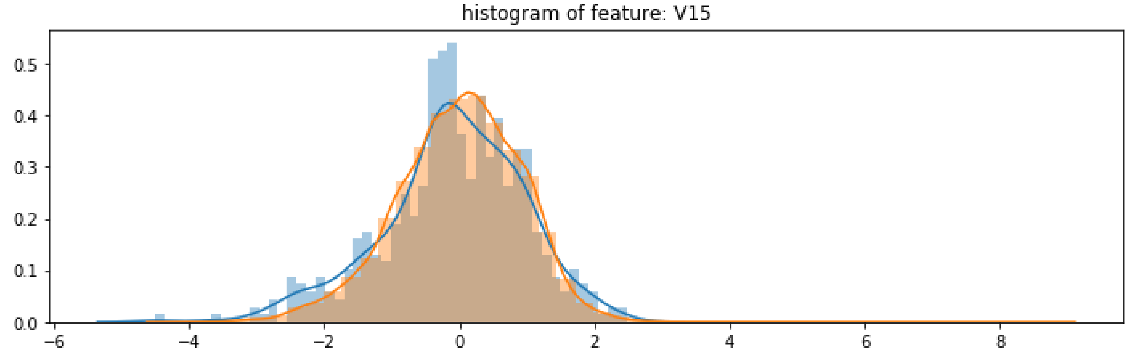
histogram of feature: V13



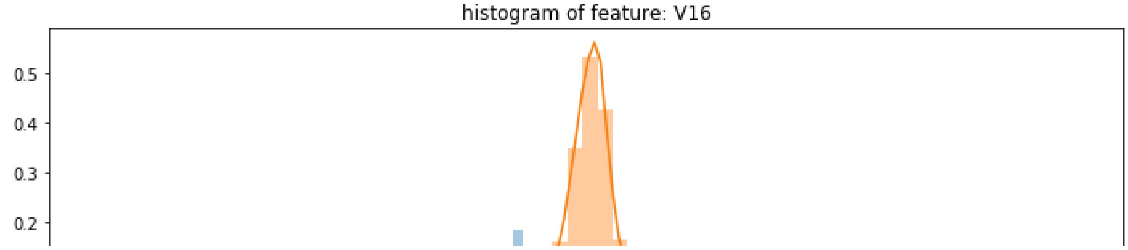
histogram of feature: V14

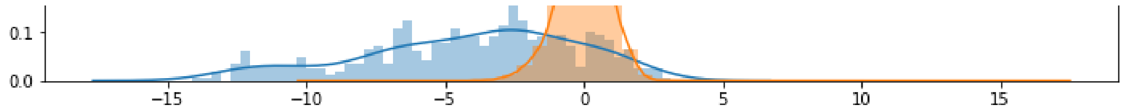


histogram of feature: V15

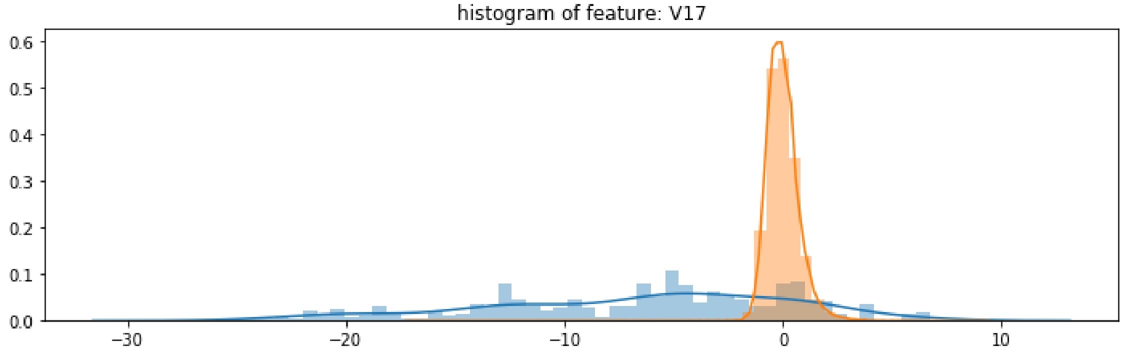


histogram of feature: V16

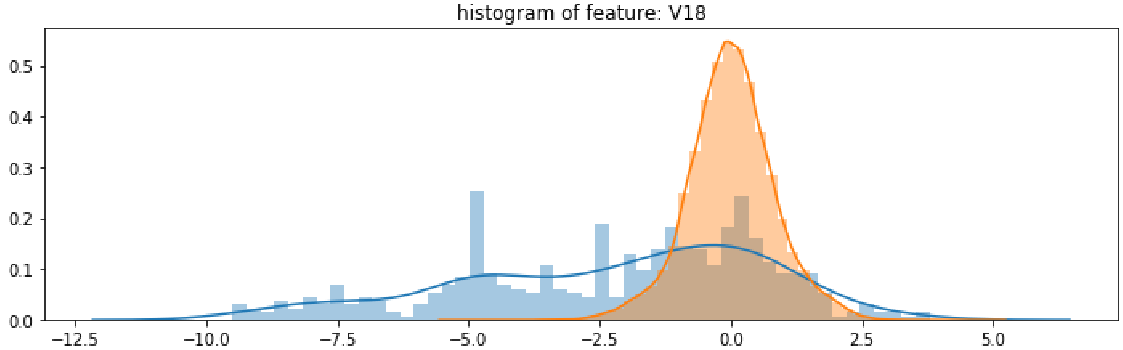




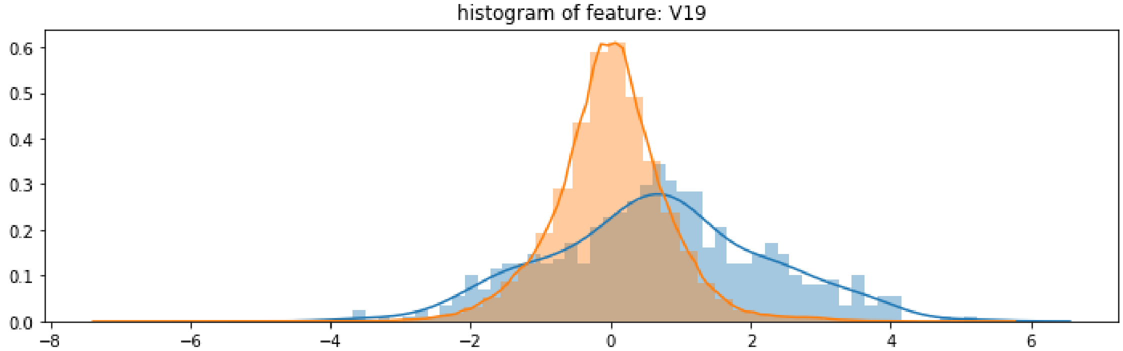
histogram of feature: V17



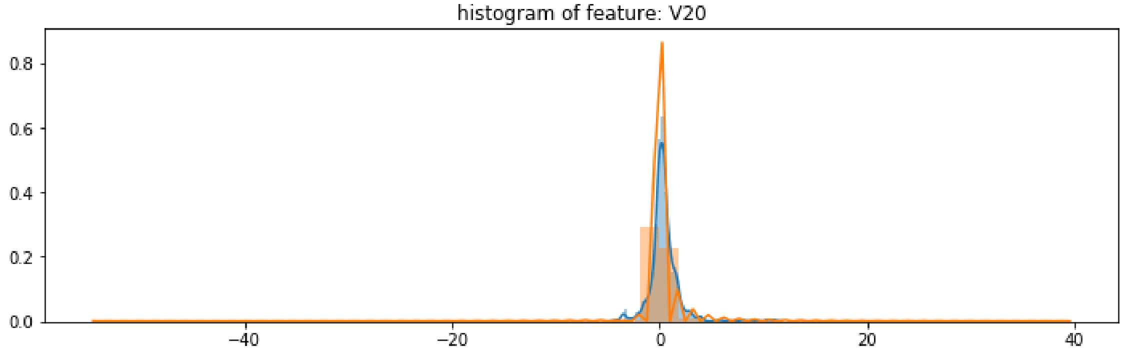
histogram of feature: V18



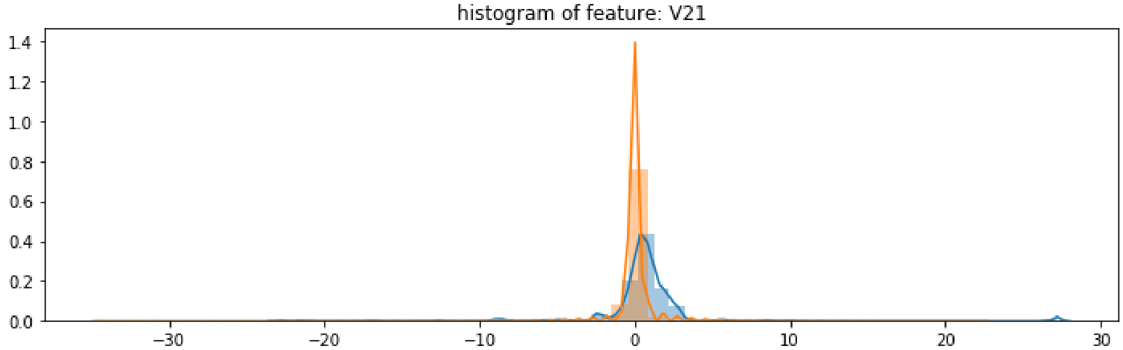
histogram of feature: V19



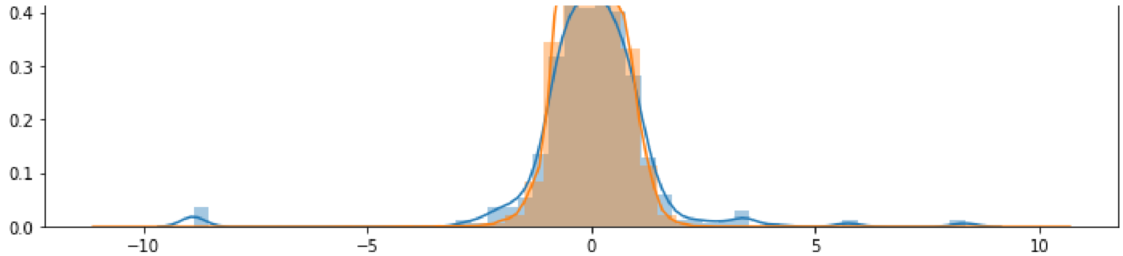
histogram of feature: V20



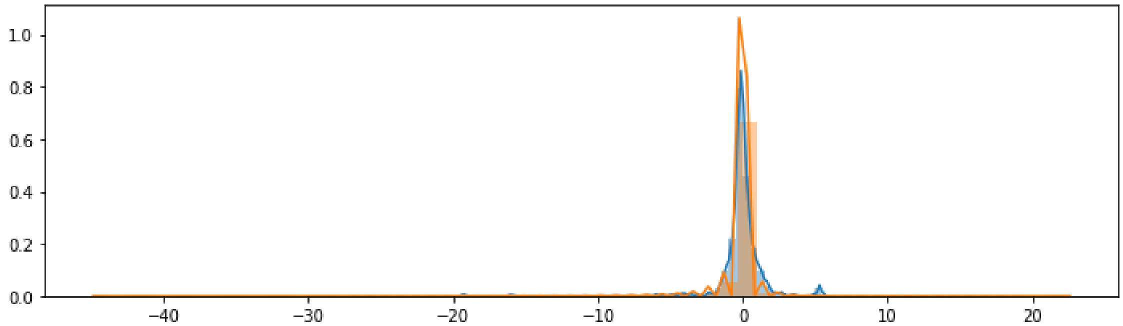
histogram of feature: V21



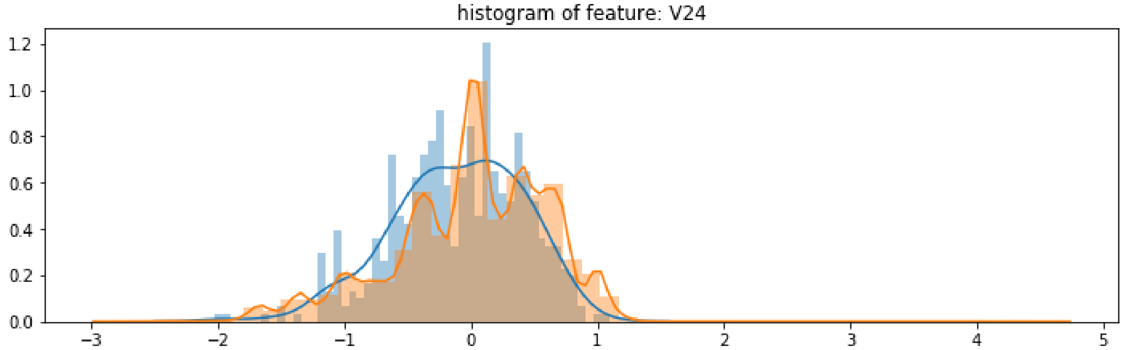
histogram of feature: V22



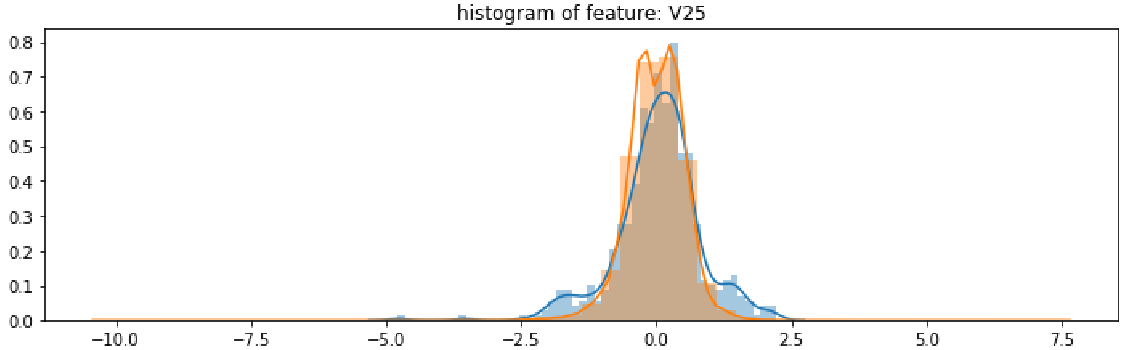
histogram of feature: V23



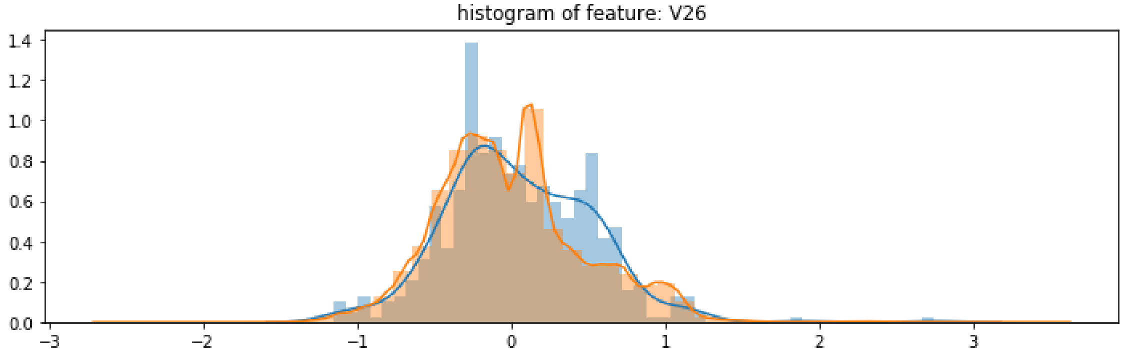
histogram of feature: V24



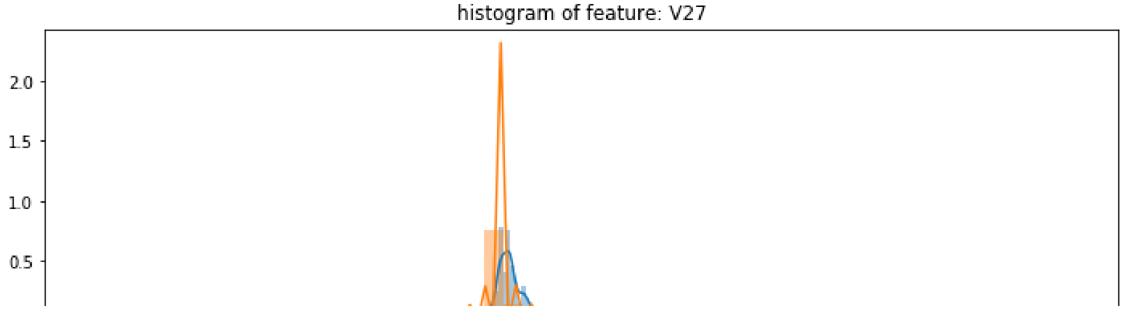
histogram of feature: V25

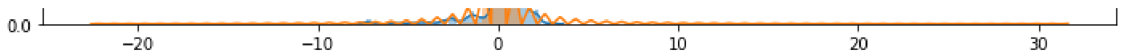


histogram of feature: V26



histogram of feature: V27





Let’s separate the Fraudulent cases from the authentic ones and compare their occurrences in the dataset.

In [16]:
```
# Determine number of fraud cases in dataset
Fraud = data[data['Class'] == 1]
Valid = data[data['Class'] == 0]
outlier_fraction = len(Fraud)/float(len(Valid))
print(outlier_fraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

```
0.0017304750013189597
Fraud Cases: 492
Valid Transactions: 284315
```

fraud There is only 0.17% fraudulent transaction out all the transactions. The data is highly Unbalanced. Lets first apply our models without balancing it and if we don’t get a good accuracy then we can find a way to balance this dataset.

In [19]:
```
print("Amount details of fraudulent transaction")
Fraud.Amount.describe()
```

```
Amount details of fraudulent transaction
```

Out[19]:
```
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 [20]:
```
print("details of valid transaction")
Valid.Amount.describe()
```

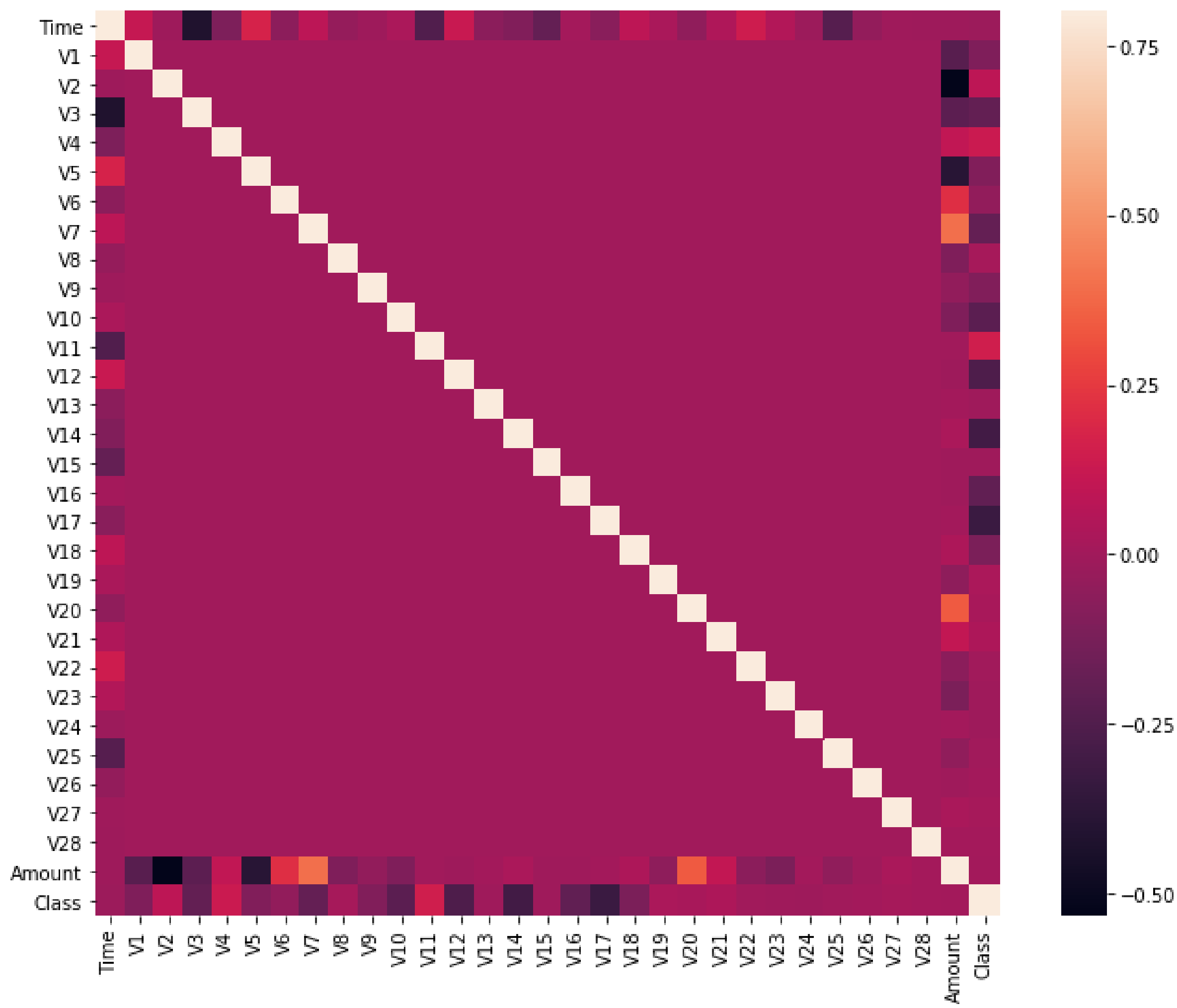
```
details of valid transaction
```

Out[20]:
```
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
```

As we can clearly notice from this, the average Money transaction for the fraudulent ones are more. This makes this problem crucial to deal with.

Correlation matrix graphically gives us an idea of how features correlate with each other and can help us predict what are the features that are most relevant for the prediction.

In [21]:
```python
# Correlation matrix
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```



In the HeatMap we can clearly see that most of the features do not correlate to other features but there are some features that either has a positive or a negative correlation with each other. For example “V2” and “V5” are highly negatively correlated with the feature called “Amount”. We also see some correlation with “V20” and “Amount”. This gives us a deeper understanding of the Data available to us.

With that out of the way let’s proceed with dividing the data values into Features and Target.

```
In [23]: #dividing the X and the Y from the dataset
X=data.drop(['Class'], axis=1)
Y=data["Class"]
print(X.shape)
print(Y.shape)
#getting just the values for the sake of processing (its a numpy array with no
columns)
X_data=X.values
Y_data=Y.values
```

```
(284807, 30)
(284807,)
```

Using Skicit learn to split the data into Training and Testing.

```
In [24]: # Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size
= 0.2, random_state = 42)
```

Building the Isolation Forest Model

Isolation forest is generally used for Anomaly detection. Feel free to have a look at this video if you want to learn more about this Algorithm.

```
In [25]: #Building another model/classifier ISOLATION FOREST
from sklearn.ensemble import IsolationForest
ifc=IsolationForest(max_samples=len(X_train),contamination=outlier_fraction,ra
ndom_state=1)
ifc.fit(X_train)
scores_pred = ifc.decision_function(X_train)
y_pred = ifc.predict(X_test)
```

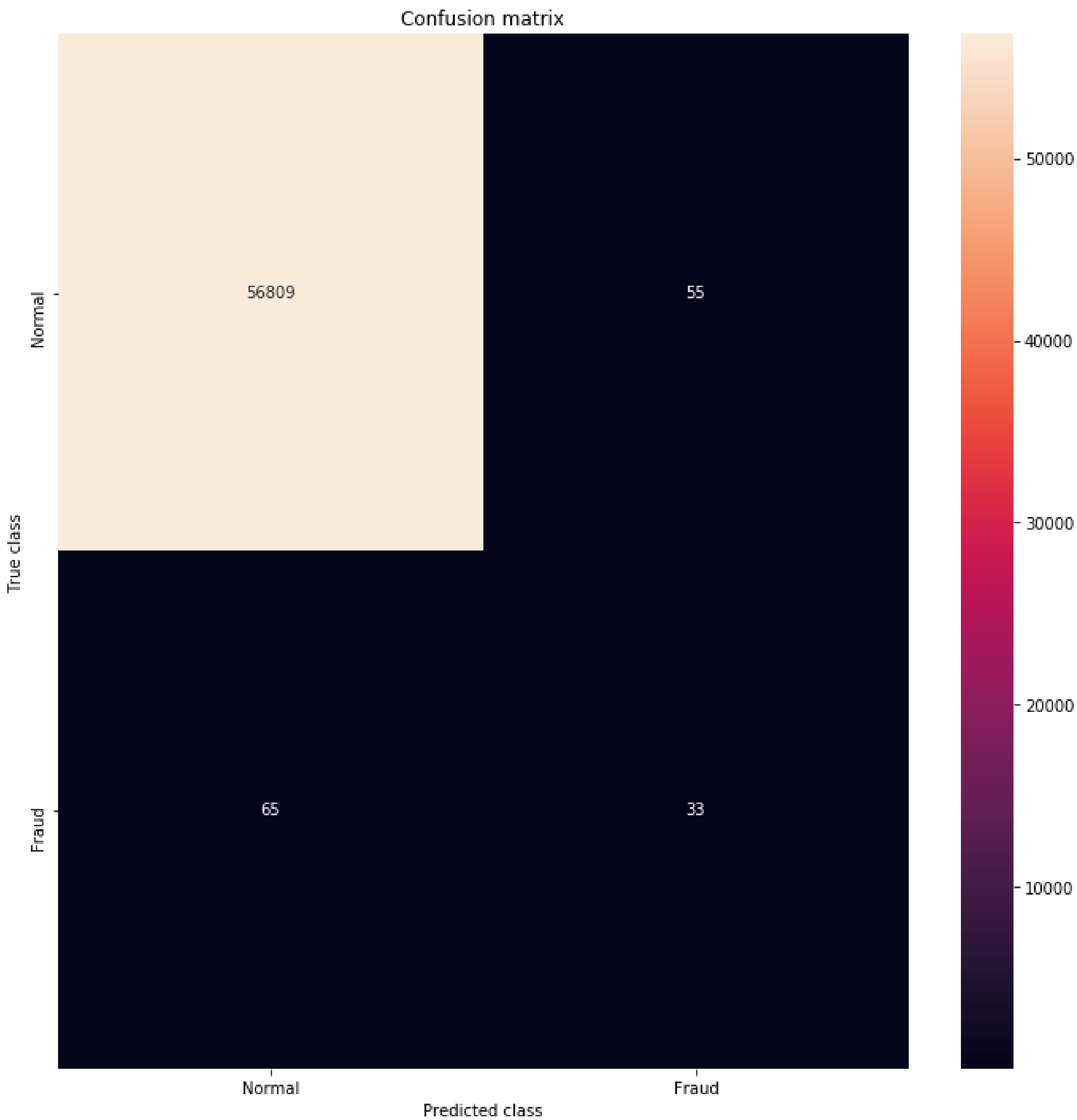
```
C:\Users\sundara.rao.ext\AppData\Local\python\lib\site-packages\sklearn\ensem
ble\iforest.py:247: FutureWarning: behaviour="old" is deprecated and will be
removed in version 0.22. Please use behaviour="new", which makes the decision
_function change to match other anomaly detection algorithm API.
  FutureWarning)
C:\Users\sundara.rao.ext\AppData\Local\python\lib\site-packages\sklearn\ensem
ble\iforest.py:415: DeprecationWarning: threshold_ attribute is deprecated in
0.20 and will be removed in 0.22.
  " be removed in 0.22.", DeprecationWarning)
```

Building an Evaluation Matrix on Test Set

In [26]:
```
# Reshape the prediction values to 0 for valid, 1 for fraud.
y_pred[y_pred == 1] = 0
y_pred[y_pred == -1] = 1
n_errors = (y_pred != Y_test).sum()
```

Visualizing the Confusion Matrix for this model.

In [38]:
```
#printing the confusion matrix
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score,re
call_score,f1_score,matthews_corrcoef
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(Y_test, y_pred)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS,yticklabels=LABELS, annot=True, fm
t="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



Let's see how to Evaluate the Model and print the results. We will be calculating the Accuracy, Precision, Recall, F1-Score and the Matthews correlation coefficient for the sake of totality.

In [39]:
```
#evaluation of the model
#printing every score of the classifier
#scoring in any thing
n_outliers = len(Fraud)
print("the Model used is {}".format("Isolation Forest"))
acc= accuracy_score(Y_test,y_pred)
print("The accuracy is {}".format(acc))
prec= precision_score(Y_test,y_pred)
print("The precision is {}".format(prec))
rec= recall_score(Y_test,y_pred)
print("The recall is {}".format(rec))
f1= f1_score(Y_test,y_pred)
print("The F1-Score is {}".format(f1))
MCC=matthews_corrcoef(Y_test,y_pred)
print("The Matthews correlation coefficient is{}".format(MCC))
```

```
the Model used is Isolation Forest
The accuracy is 0.9978933323970366
The precision is 0.375
The recall is 0.336734693877551
The F1-Score is 0.3548387096774193
The Matthews correlation coefficient is0.3543008067850027
```

As you can clearly see this model is not doing as great as expected, so let’s build some other model to get a better result.

Building the Random Forest Model

Lets us Build a Random Forest to increase the performance of the Detector. I thought of using a Decision Tree Model but as we know Random Forest is like an army of Decision Trees, then why to even bother trying and failing. You can think Random Forest to be the Ensembling applied to the Decision Tree

In [40]:
```
# Building the Random Forest Classifier (RANDOM FOREST)
from sklearn.ensemble import RandomForestClassifier
# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(X_train,Y_train)
# predictions
y_pred = rfc.predict(X_test)
```

```
C:\Users\sundara.rao.ext\AppData\Local\python\lib\site-packages\sklearn\ensem
ble\forest.py:245: FutureWarning: The default value of n_estimators will chan
ge from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Building an Evaluation matrix on test set

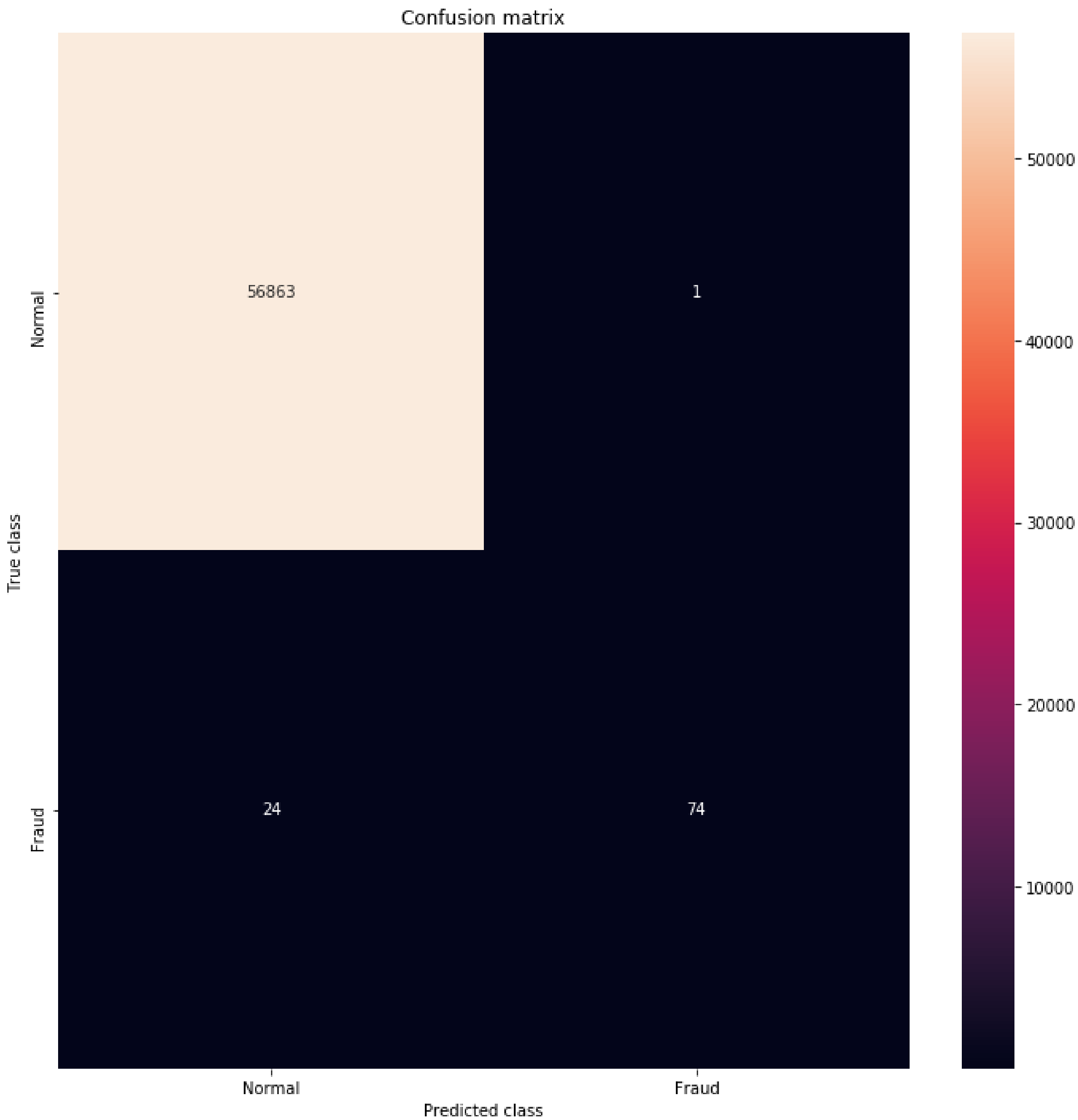
We will be calculating the Accuracy, Precision, Recall, F1-Score, and the Matthews correlation coefficient.

In [41]:
```
#Evaluating the classifier
#printing every score of the classifier
#scoring in any thing
from sklearn.metrics import classification_report, accuracy_score,precision_sc
ore,recall_score,f1_score,matthews_corrcoef
from sklearn.metrics import confusion_matrix
n_outliers = len(Fraud)
n_errors = (y_pred != Y_test).sum()
print("The model used is Random Forest classifier")
acc= accuracy_score(Y_test,y_pred)
print("The accuracy is {}".format(acc))
prec= precision_score(Y_test,y_pred)
print("The precision is {}".format(prec))
rec= recall_score(Y_test,y_pred)
print("The recall is {}".format(rec))
f1= f1_score(Y_test,y_pred)
print("The F1-Score is {}".format(f1))
MCC=matthews_corrcoef(Y_test,y_pred)
print("The Matthews correlation coefficient is{}".format(MCC))
```

```
The model used is Random Forest classifier
The accuracy is 0.9995611109160493
The precision is 0.9866666666666667
The recall is 0.7551020408163265
The F1-Score is 0.8554913294797689
The Matthews correlation coefficient is0.8629589216367891
```

Visualizing the confusion matrix as well.

In [43]:
```
#printing the confusion matrix
LABELS = ['Normal', 'Fraud']
conf_matrix = confusion_matrix(Y_test, y_pred)
plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, f
mt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



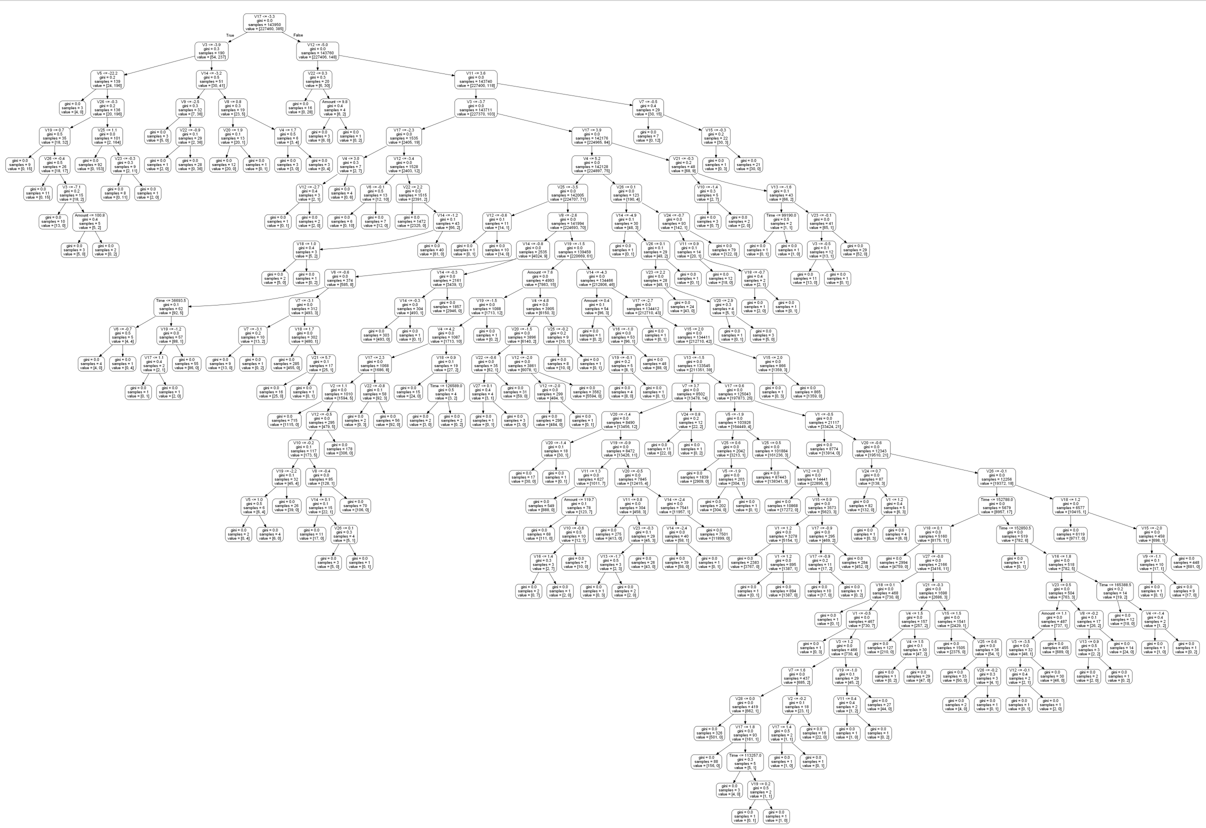
Visualizing the Forest

A single tree from the forest is taken randomly and then visualized for the sake of knowing how the Algorithm is taking its decision and this will help in changing the model easily if a countermeasure is taken by the scammers. For that, you have to import some Tools from Sklearn library and IPython library to display it in your Notebook.

In [46]:
```
pip install pydot
```

```
Collecting pydot
  Using cached https://files.pythonhosted.org/packages/33/d1/b1479a770f66d962
f545c2101630ce1d5592d90cb4f083d38862e93d16d2/pydot-1.4.1-py2.py3-none-any.whl
Requirement already satisfied: pyparsing>=2.1.4 in c:\users\sundara.rao.ext\a
ppdata\local\python\lib\site-packages (from pydot) (2.4.0)
Installing collected packages: pydot
Successfully installed pydot-1.4.1
Note: you may need to restart the kernel to use updated packages.
```

In [47]:
```
#visualizing the random tree
feature_list = list(X.columns)
# Import tools needed for visualization
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydot
#pulling out one tree from the forest
tree = rfc.estimators_[5]
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rou
nded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
# Write graph to a png file
display(Image(graph.create_png()))
```



Conclusion

Our Random Forest result in most cases exceeds the previously reported results with a Matthews Correlation Coefficient of 0.8629. Other performance characteristics are also satisfactory so now we don't need to apply some other model to this.

As you can clearly see that our model or any model, in general, have a low Recall Value, which is precisely the reason why you get harassed by so many confirmation messages after a transaction. But with more and more advancements in Machine Learning Models, we are slowly but steadily dealing with that problem without compromising your account's security.

The model is fast, it is definitely simple and most importantly easily interpretable as shown in the Decision Tree diagram. The privacy of the user is still intact, as the data used had its dimensionality reduced in the beginning. Well, we still have not managed to deal with the unbalancing of data, but I think we have done pretty fine without it. It is, actually a big milestone covered for all of us. There is and will always be a long way to go but this sounds like a good start to me. Hope you enjoyed reading this article as much as I enjoyed writing it. Honestly, I was a bit skeptical about it at first, especially when the Isolation Forest did not produce a good result but now having seen the result from the Random Forest, I am feeling pretty gratified after finishing it with this kind of result.

This field needs so much more research and this is one of the topics where an increase in specificity by 0.1% will save Millions, if not Billions of Dollars.

Save the Model

In [3]:
```
pickle.dump(regressor , open('creditcared.pkl' ,'wb'))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-0b2c77dcbe96> in <module>
----> 1 pickle.dump(regressor , open('creditcared.pkl' ,'wb'))

NameError: name 'regressor' is not defined
```

In [ ]:
```

```