# Updates02072018

February 7, 2018

## 1 CREDIT CARD FRAUD MILESTONE REPORT

### 1.1 Overview

This capstone project aims at using machine learning models to learn and differentiate between fraud and legal transactions. By analyzing a variety of information such as time and transation amount from real credit card data, the model hopes to give accurate fraud predictions and therefore enhances the security of credit transactions for both customers and banks.

### 1.2 The Problem

With ecommerce and online transactions increasing each day, credit card fraud has become a real concern for consumers as well as financial companies. According to Forbes news(1), card fraud losses actually hit $21.84 billion worldwide in 2015, which was truly alarming.

So the question I am going solve here is: given data, how accurately can I detect a fraudulent transaction?

### 1.3 The Client

The real-world clients could be financial companies like Chase, Capital One, American Express, Discover etc.
* For company themselves, the early prediction of potential frauds will help prevent unnecessary financial losses by allowing them to take appropriate early actions. * For customers, the detection features could be built into credit card apps so that when a potential fraud is detected, the customers could be alerted of the situation by texts or calls. They could then respond and communicate with card companies to solve the issues together.

After all, who doesn't like a reliable model/method that helps guard your bank accounts and prevent unnecessary financial losses?

### 1.4 The Data

The dataset is already available on Kaggle.(https://www.kaggle.com/dalpozz/creditcardfraud)

It contains transactions made by credit cards in September 2013 by European cardholders. The transactions all happened within 2 days, where there are 492 frauds out of 284, 807 transactions.(2) It's in the format of .csv tabular data which has the following information: * V1-V28: The numerical principal components obtained through PCA transformation. Original features and backgroud information are not provided due to confidentiality issues. * Time: The seconds elapsed between

each transaction and the first transaction in the dataset; * Amount: Transaction amount; * Class: The classification class. 1: fraud; 0: legal transaction.

### 1.5 Exploratory Data Analyisis (EDA)

```
In [20]: #Capstone1: Credit Card Fraud
         #1.Import libraries
         import pandas as pd
         import numpy as np

         import matplotlib.pyplot as plt
         import seaborn as sns

         from datetime import datetime, timedelta

         from statsmodels.stats.weightstats import ztest
         from scipy import stats

         %matplotlib inline
```

```
In [2]: #2.1 Import data
        fn='creditcard.csv'
        #Uncomment this in github fn='/Data/creditcard.csv'
        df = pd.read_csv(fn)
```

First we want to get a general idea of what the dataset looks like, whether it has any missing values, the data format, the dimensions, simple summary stats and sample data in the beginning and at the end to decide on future steps od data wrangling.

```
In [3]: #2.2 Summary stats
        # Check missing data
        df.isnull().any().sum()
```

```
Out[3]: 0
```

```
In [4]: print(df.info())
        print(df.shape)
        print(df.columns)
        print(df.describe())
        print(df.head(5))
        print(df.tail(5))
```

```
<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
None
(284807, 31)
Index([u'Time', u'V1', u'V2', u'V3', u'V4', u'V5', u'V6', u'V7', u'V8', u'V9',
       u'V10', u'V11', u'V12', u'V13', u'V14', u'V15', u'V16', u'V17', u'V18',
       u'V19', u'V20', u'V21', u'V22', u'V23', u'V24', u'V25', u'V26', u'V27',
       u'V28', u'Amount', u'Class'],
      dtype='object')
                Time             V1             V2             V3             V4  \
count  284807.000000  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
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5             V6             V7             V8             V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
```

```
mean  -1.552563e-15  2.010663e-15 -1.694249e-15 -1.927028e-16 -3.137024e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

           ...                V21           V22           V23           V24  \
count      ...       2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean       ...       1.537294e-16  7.959909e-16  5.367590e-16  4.458112e-15
std        ...       7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min        ...      -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%        ...      -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%        ...      -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%        ...       1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max        ...       2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

[8 rows x 31 columns]
   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
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...         V21       V22       V23       V24  \
0  0.098698  0.363787  ...   -0.018307  0.277838 -0.110474  0.066928
```

```
1  0.085102 -0.255425  ...   -0.225775 -0.638672  0.101288 -0.339846
2  0.247676 -1.514654  ...    0.247998  0.771679  0.909412 -0.689281
3  0.377436 -1.387024  ...   -0.108300  0.005274 -0.190321 -1.175575
4 -0.270533  0.817739  ...   -0.009431  0.798278 -0.137458  0.141267

        V25       V26       V27       V28  Amount  Class
0  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1  0.167170  0.125895 -0.008983  0.014724    2.69      0
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3  0.647376 -0.221929  0.062723  0.061458  123.50      0
4 -0.206010  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
           Time         V1         V2        V3        V4        V5  \
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...         V21       V22  \
284802 -2.606837 -4.918215  7.305334  1.914428  ...    0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...    0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...    0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...    0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...    0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
284802  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

        Class
284802      0
284803      0
284804      0
284805      0
284806      0

[5 rows x 31 columns]
```

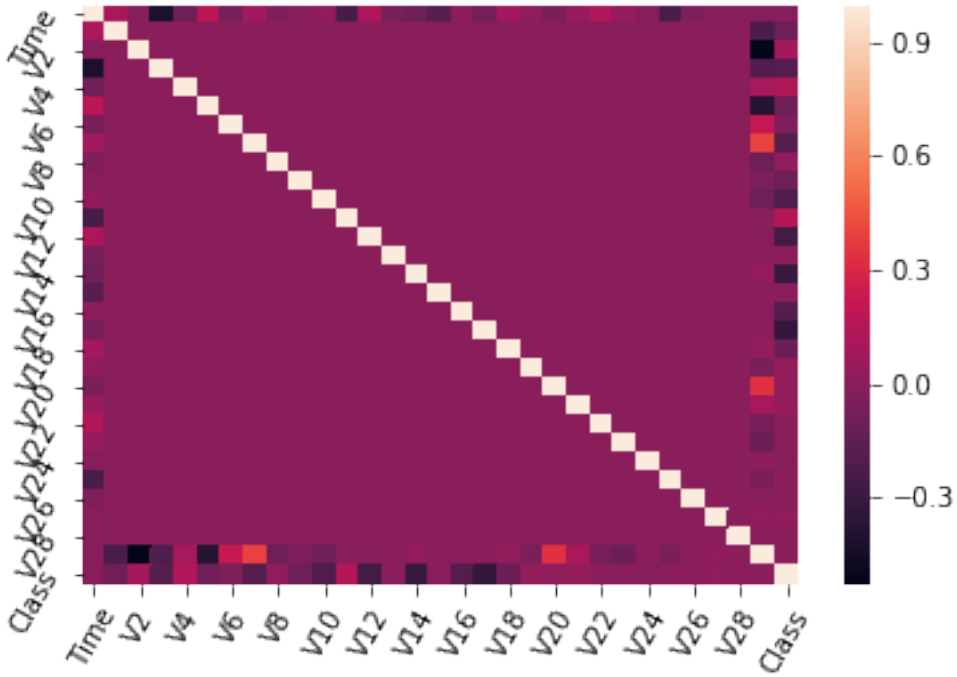
**Summary**:

We can see that, from initial exploratory analysis: * None of the columns have missing data, so we don't need to clean the data. For the next part, we only need to focus our attention on dealing with outliers, if any. * It has 284807 records in total, which is consistent with the description on

Kaggle. All of the features are numerical, with only 'Class' in the integer format, others being float. * Further .shape attribute is used to confirm the dimensions of (284807, 31); and .columns attribute to confirm the column names. * Finally, .describe(), .head() and .tail() methods are used to inspect summary stats, and general data structures of the beginning, and the end of the data frame respectively. * Time is not unique for each entry, which means there are multiple transactions happening at the same time in the dataset.

Next, we are plotting a heat map to see majorly the correlations of V1-V28. Since they were pre-processed by PCA, they are supposed to be a set of linearly uncorrelated variables.

```
In [5]: #Visualize correlations between all variables
        sns.heatmap(df.corr())
        plt.xticks(rotation=60)
        plt.yticks(rotation=60)
        plt.show()
```

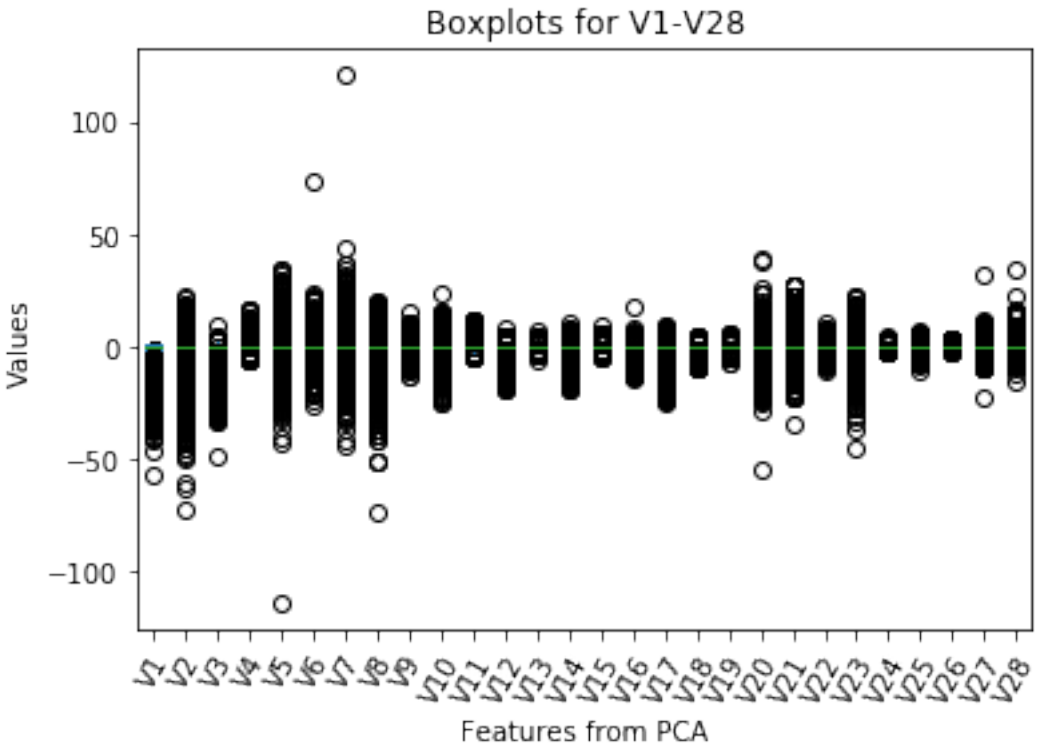



The above graph confirms our previous assumption, as V1-V28 are completely uncorrelated (zero value, purple).

For the following sections, we are going to explore how these variables: PCA features, time, amount and class, distribute separately for the whole dataset. * PCA features: boxplot * time: distribution plot * amount: boxplot * class: summary statistics

```
In [6]: #By different categories: PCA features, Time, Amount, and Class(fraud or not)
        #Visual EDA, PCA features plot
        df_pca = df.iloc[:,1:29]
        df_pca.plot(kind='Box')
```

```
plt.xticks(rotation=60)
plt.title('Boxplots for V1-V28')
plt.xlabel('Features from PCA')
plt.ylabel('Values')
plt.show()
```

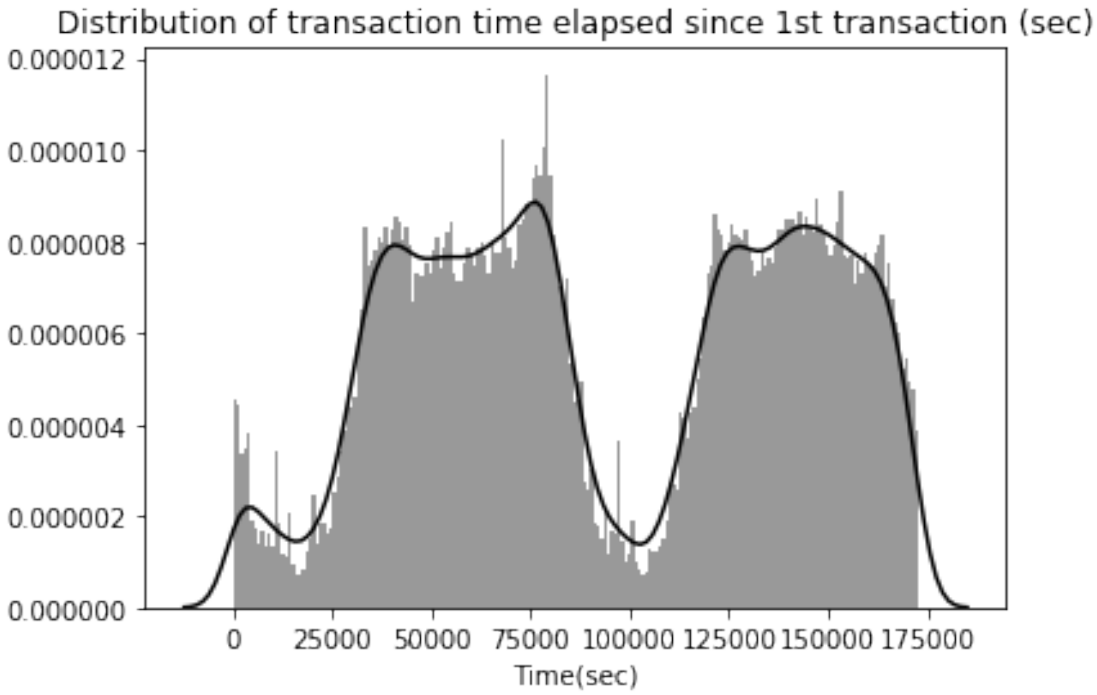



We can see from the plot that in general, these 28 features are on the same scale of within -50 to 50. Several particularly noticeable outliers are in V5, V6, V7. Considering this is the box plot for all data including legal and fraud transactions, we will examine further later in the analysis to see where these outliers are from.

Next, we use .value_counts() on 'Time' column to confirm that Time is not unique, as the length reveals that there are only 124,592 unique values (as compared to 284, 807 total records). Further we are drawing a distribution plot to see whether there are any patterns in these transaction times.

```
In [7]: #EDA and visual EDA for time
        print(df['Time'].describe())
        print(len(df['Time'].value_counts()))
        plt.figure()
        sns.distplot(df['Time'], bins = 200, norm_hist = True, color = 'black')
        plt.title('Distribution of transaction time elapsed since 1st transaction (sec)')
        plt.xlabel('Time(sec)')
        plt.show()

count    284807.000000
mean      94813.859575
```

```
std        47488.145955
min            0.000000
25%        54201.500000
50%        84692.000000
75%       139320.500000
max       172792.000000
Name: Time, dtype: float64
124592
```



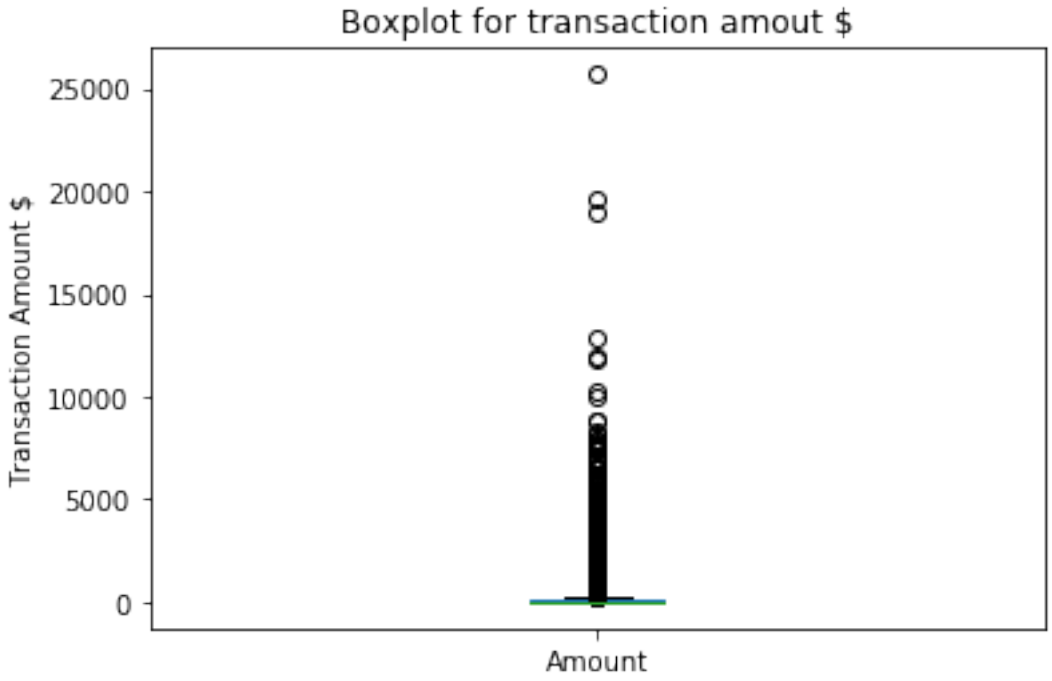

The plot shows us that there were certain time periods that transactions happened more frequently, considering this variable is expressed in seconds, we are creating a new column of datetime objects and another of hour for later analysis.

```
In [8]: #Convert time into datetime obejct
        def GetTime(time):
            sec = timedelta(seconds=time)
            d = datetime(1,1,1) + sec
            return d
            #"%d:%d:%d:%d" % (d.day-1, d.hour, d.minute, d.second)
        df_newtime = [GetTime(time) for ind, time in df['Time'].iteritems()]
        df['Newtime'] = df_newtime
        #Extract hour information
        df['hour'] = [(GetTime(time)).hour for ind, time in df['Time'].iteritems()]
```

Next, we are using a boxplot to visualize the distribution of transaction amounts.

```
In [9]: #EDA and visual EDA for amount
        print(df['Amount'].describe())
        plt.figure()
        df.plot(y='Amount', kind='box')
        plt.title('Boxplot for transaction amout $')
        plt.ylabel('Transaction Amount $')
        plt.show()
```

```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64


<matplotlib.figure.Figure at 0x107e62490>
```




We can see there are some outliers with really big values in this feature, and we'll examine it further in the analysis.

For 'Class', since Class == 1 denotes the fraud transaction, we can use .sum() to confirm the total number of frauds within the dataset. As it turns out, there are 492 frauds in the whole dataset, which constitutes only 0.172% of all transactions, making it a very unbalanced dataset.

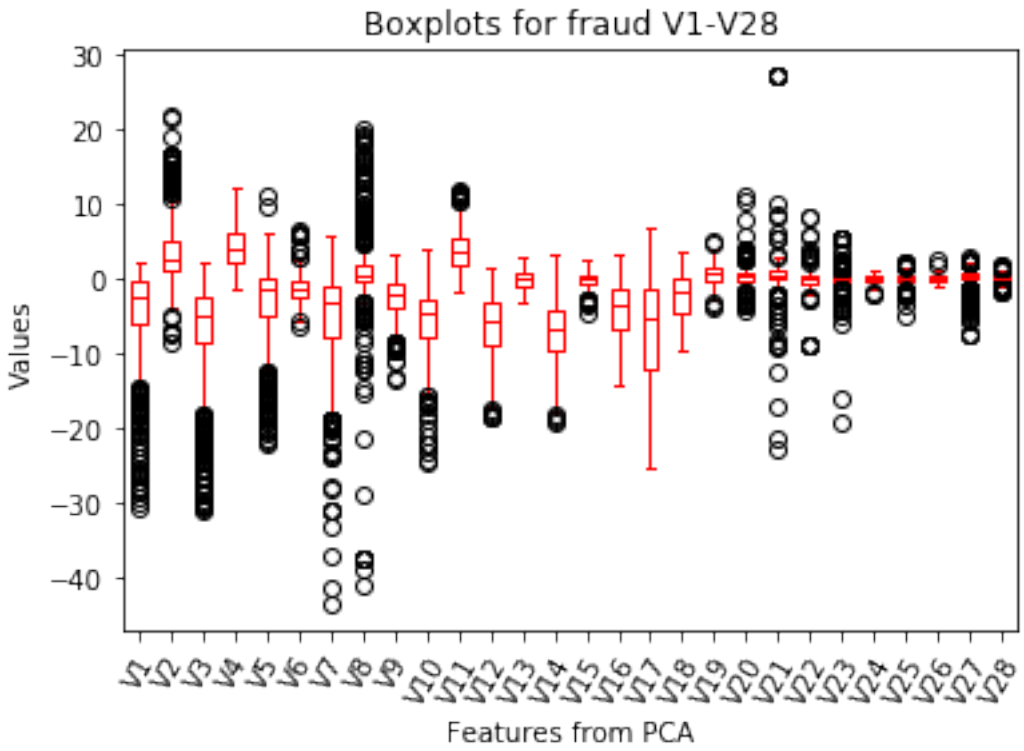
```
In [10]: #Class ==1:fraud. Confirm the number of frauds in the dataset.(Expected: Inbalanced)
         fnum = df['Class'].sum()
         fperc = float(fnum)/len(df['Class'])
         print(fperc)
```

```
0.00172748563062
```

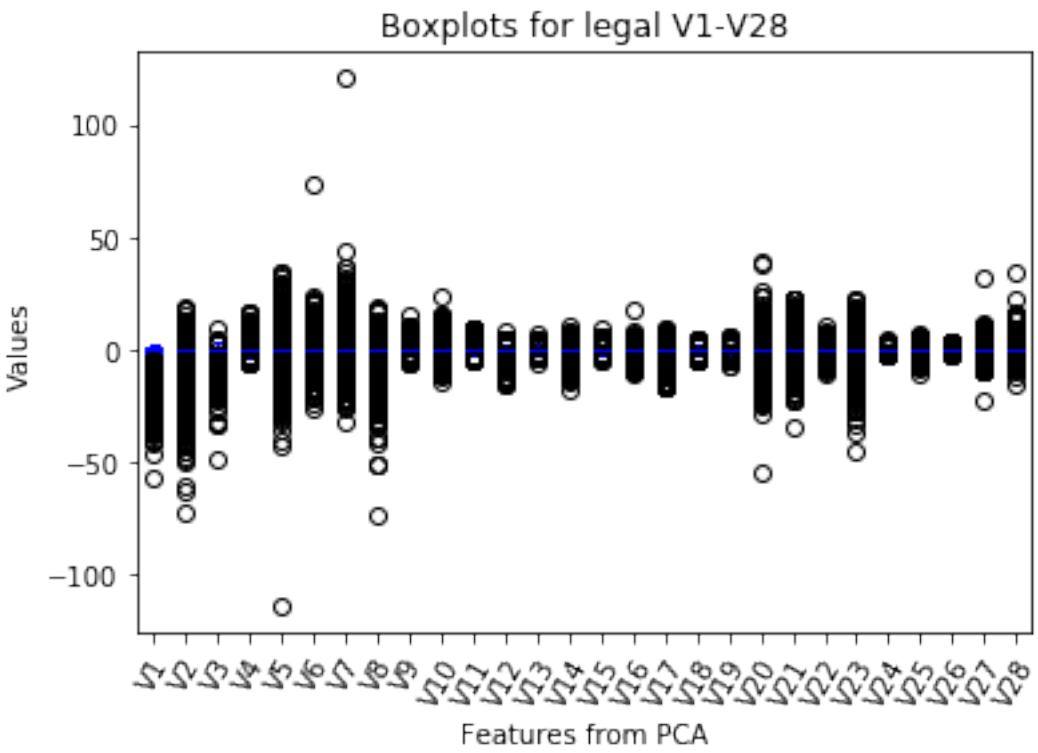
It will make sense to investigate all variables again separately based on the fraud/legal status, as they will certainly contain features that help distinguish a fraud transaction and a legal one.

```
In [11]: #2.3 EDA comparing fraud vs. legal transactions
         frauds=df[df['Class']==1]
         legals=df[df['Class']==0]
```

```
In [12]: #V1-V28, boxplots for fraud and legal transactions
         frauds.iloc[:,1:29].plot(kind='Box', color = 'red')
         plt.xticks(rotation=60)
         plt.title('Boxplots for fraud V1-V28')
         plt.xlabel('Features from PCA')
         plt.ylabel('Values')
         plt.show()
         plt.savefig('Boxplots for fraud V1-V28.png')
         legals.iloc[:,1:29].plot(kind='Box', color = 'blue')
         plt.xticks(rotation=60)
         plt.title('Boxplots for legal V1-V28')
         plt.xlabel('Features from PCA')
         plt.ylabel('Values')
         plt.show()
```




```
<matplotlib.figure.Figure at 0x1a116eef50>
```

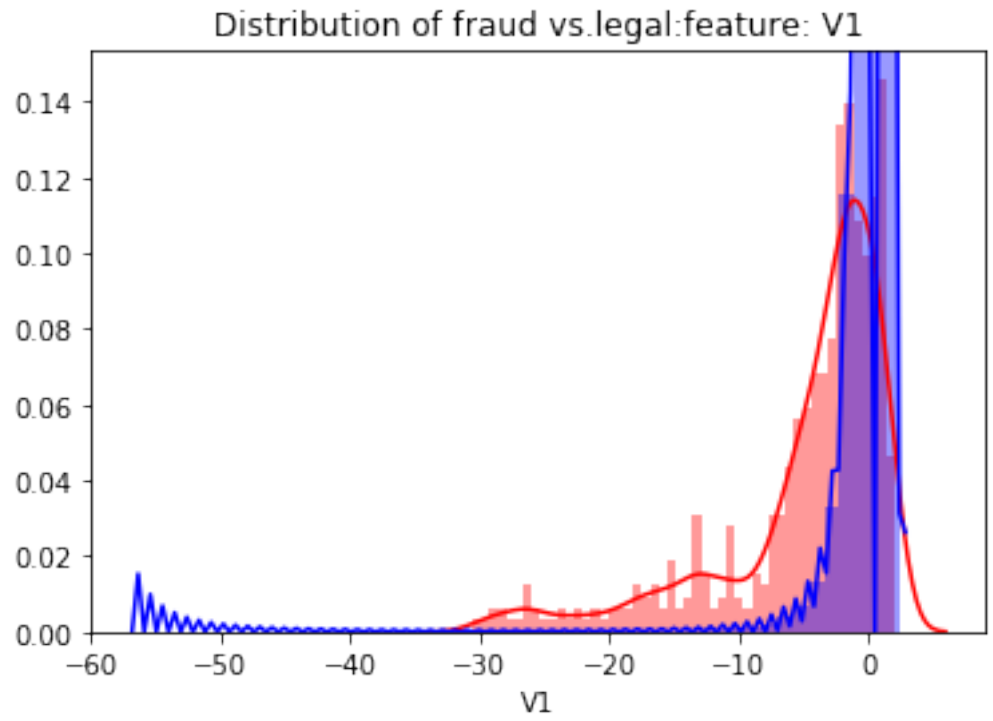


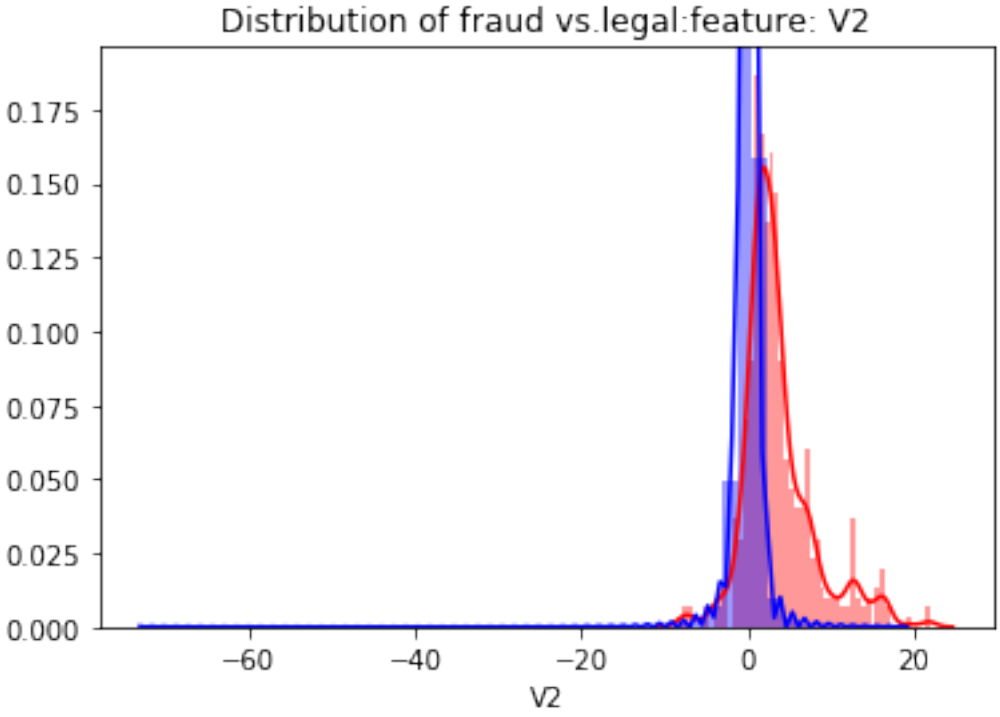

As V1-V28 of fraud transactions show relatively (but not absolutely) more variations and therefore more dispersed winthin itself than legal transactions. However, as we can see here, the extremely big outliers that we spotted earlier in the whole dataset(V5, V6, V7) are actually from legal transactions.

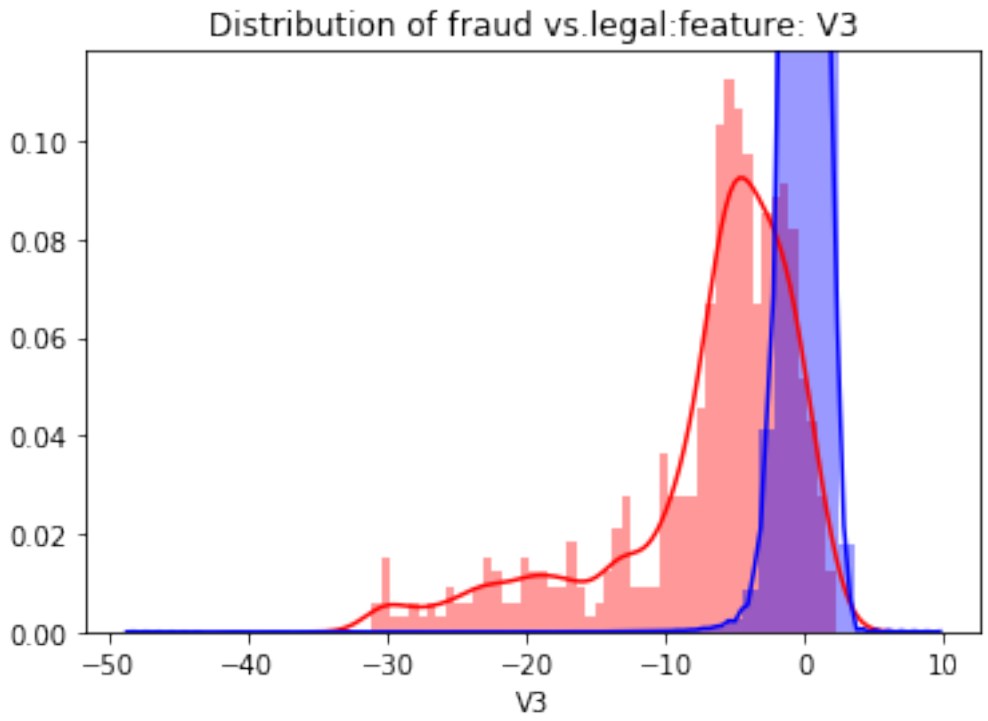
Further, we are plotting each of the PCA features in an individual distribution plot to see visually how the feature differ between fraud and legal transactions, with red indicating 'fraud' and blue 'legal'.

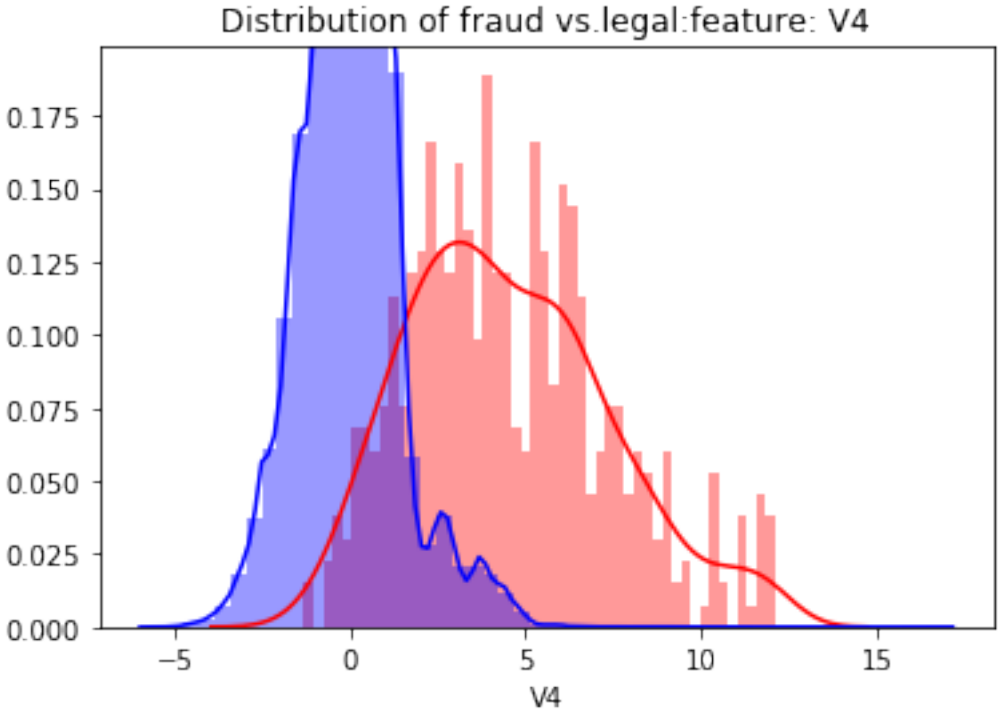
```
In [13]: #Iterate through the 28 features and compare distibution for each fraud&legal pair
         pca_names = df_pca.columns
         print(pca_names)
         for i, V in enumerate(df[pca_names]):
             plt.figure(i+1)
             sns.distplot(frauds[V], bins = 50, norm_hist = True, color = 'red')
             sns.distplot(legals[V], bins = 50, norm_hist = True, color = 'blue')
             plt.title('Distribution of fraud vs.legal:feature: ' + str(V))
             plt.show()
             plt.savefig(str(V)+'.png')
             plt.close()
```

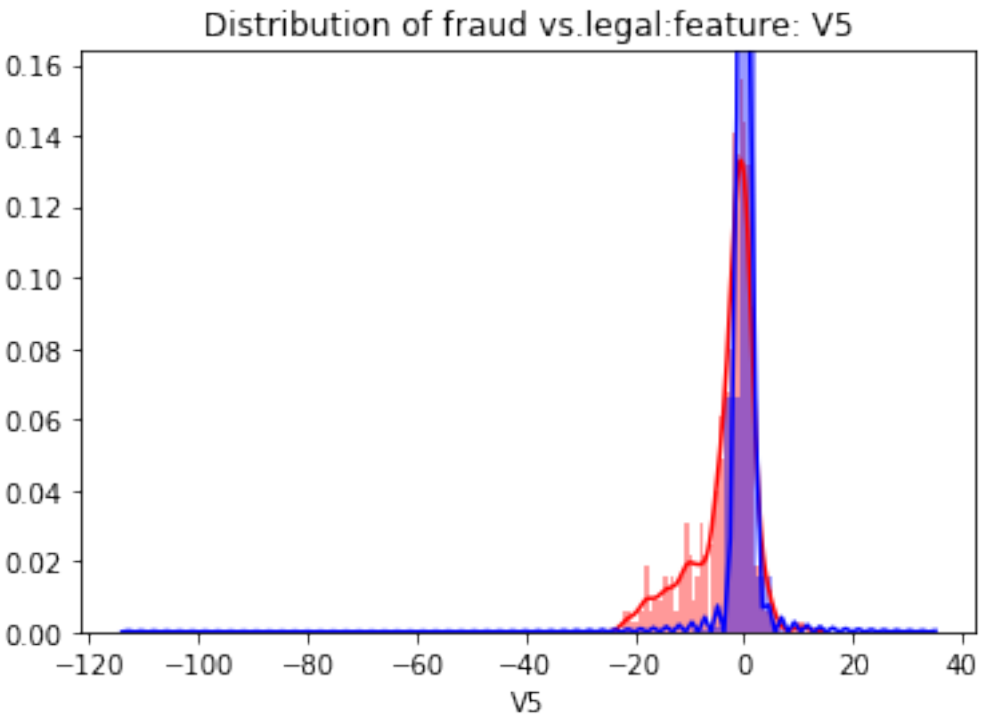
```
Index([u'V1', u'V2', u'V3', u'V4', u'V5', u'V6', u'V7', u'V8', u'V9', u'V10',
       u'V11', u'V12', u'V13', u'V14', u'V15', u'V16', u'V17', u'V18', u'V19',
       u'V20', u'V21', u'V22', u'V23', u'V24', u'V25', u'V26', u'V27', u'V28'],
      dtype='object')
```

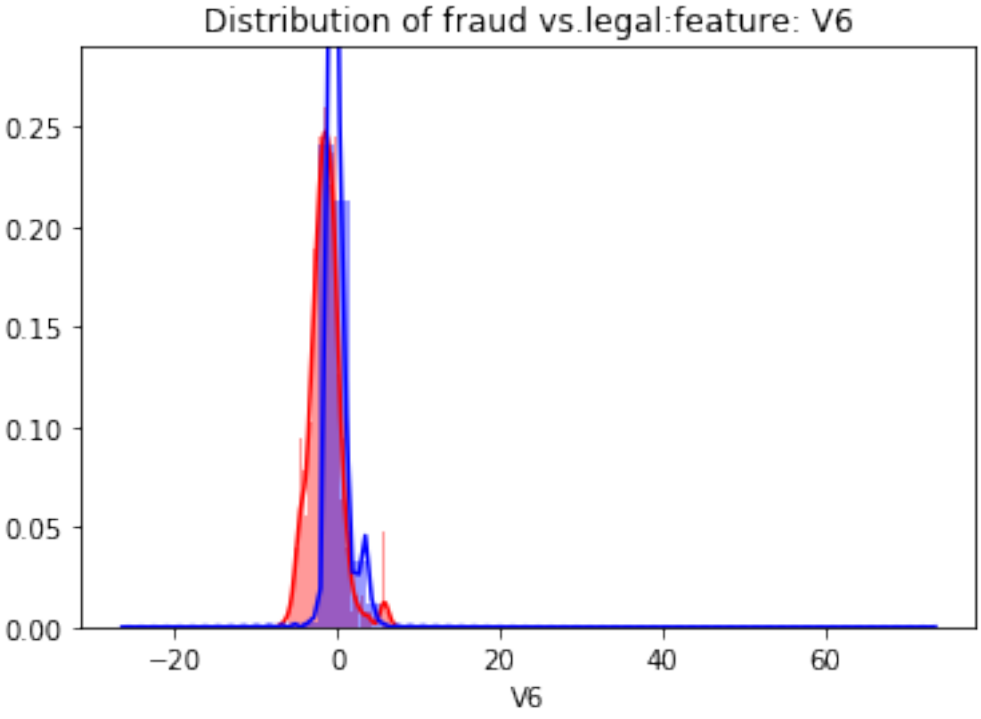

Distribution of fraud vs.legal:feature: V1
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−60
−50
−40
−30
−20
−10
0
V1

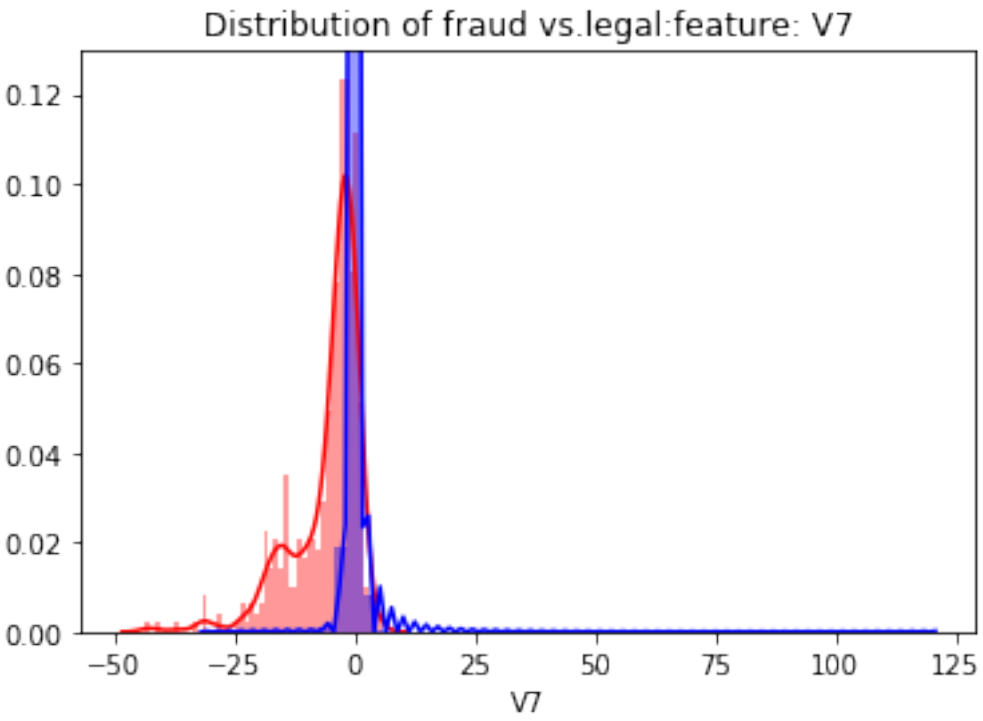

Distribution of fraud vs.legal:feature: V2
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−60
−40
−20
0
20
V2

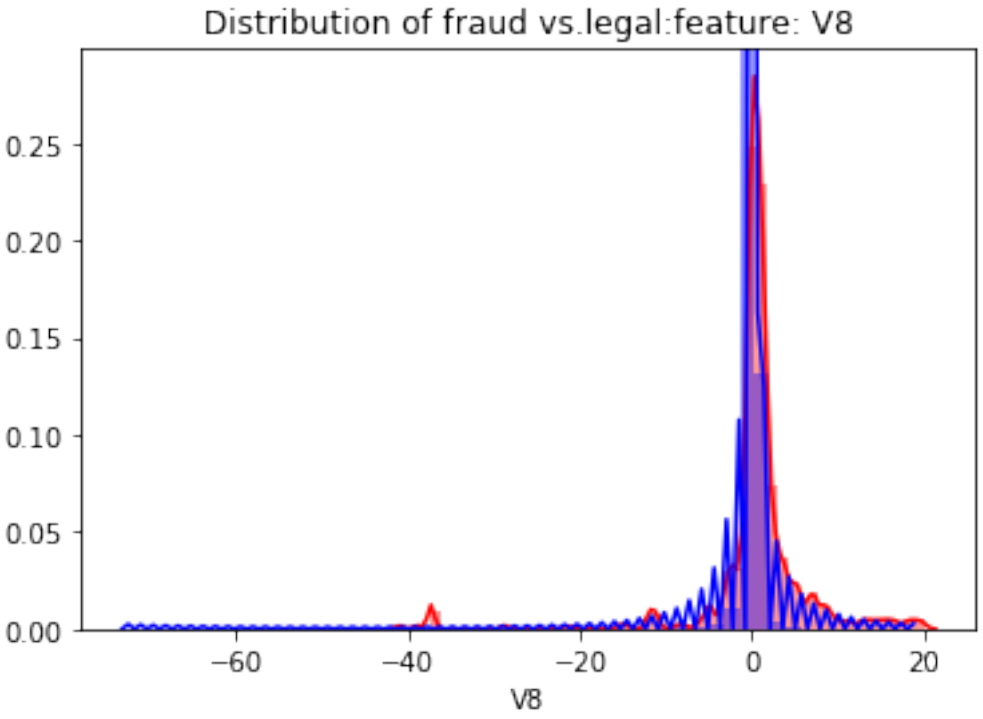

Distribution of fraud vs.legal:feature: V3
0.10
0.08
0.06
0.04
0.02
0.00
−50
−40
−30
−20
−10
0
10
V3

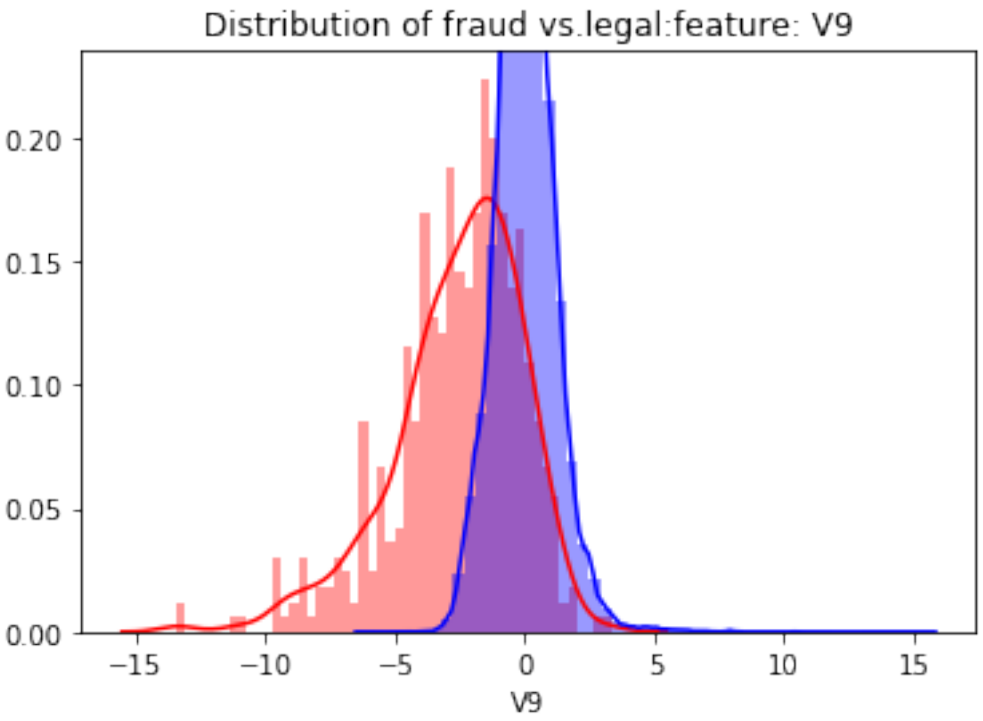

Distribution of fraud vs.legal:feature: V4
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−5
0
5
10
15
V4

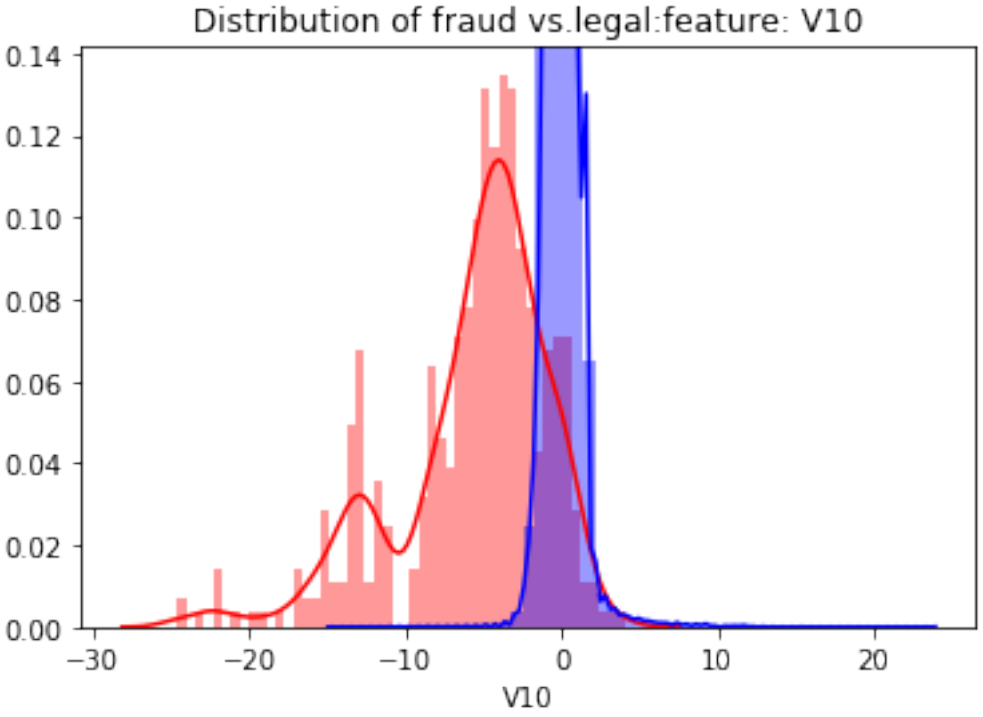

Distribution of fraud vs.legal:feature: V5
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−120
−100
−80
−60
−40
−20
0
20
40
V5

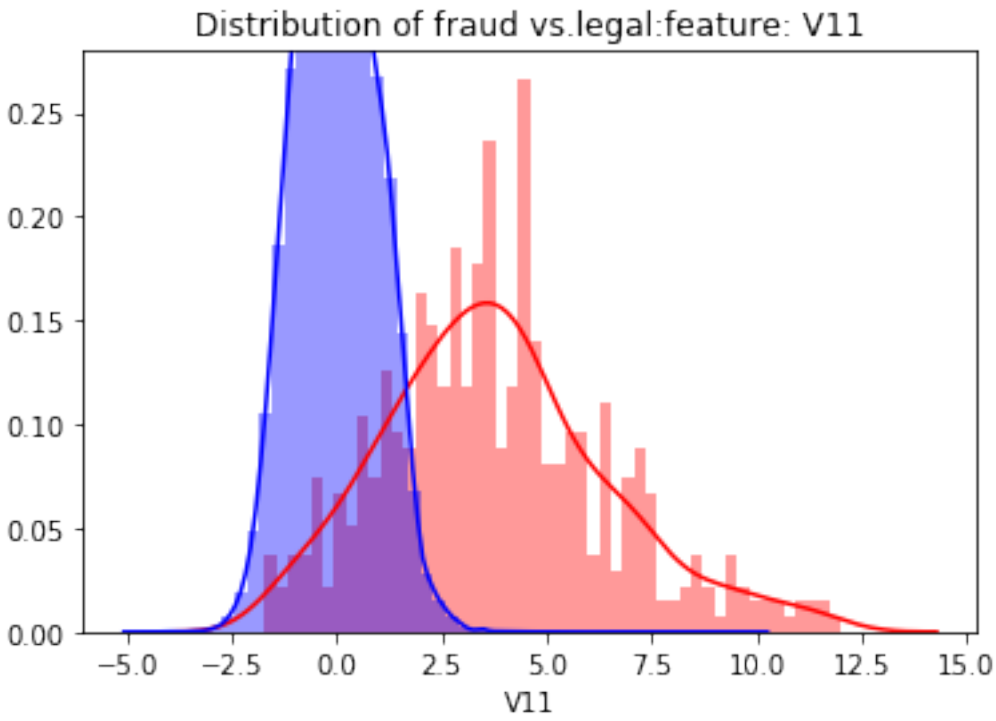

Distribution of fraud vs.legal:feature: V6
0.25
0.20
0.15
0.10
0.05
0.00
−20
0
20
40
60
V6

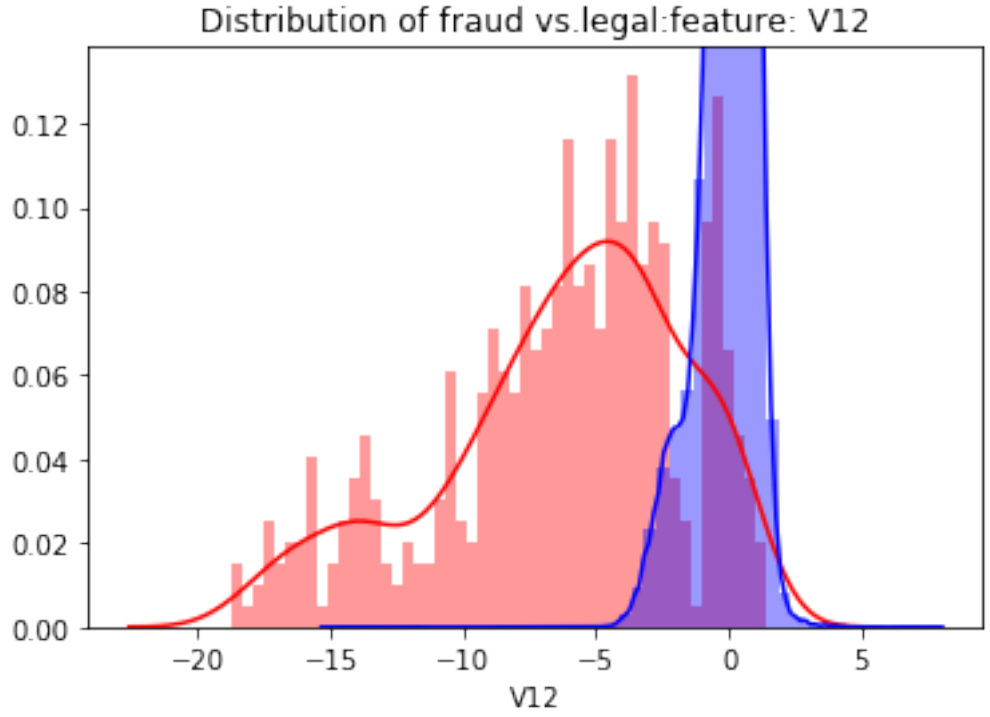

Distribution of fraud vs.legal:feature: V7
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−50
−25
0
25
50
75
100
125
V7

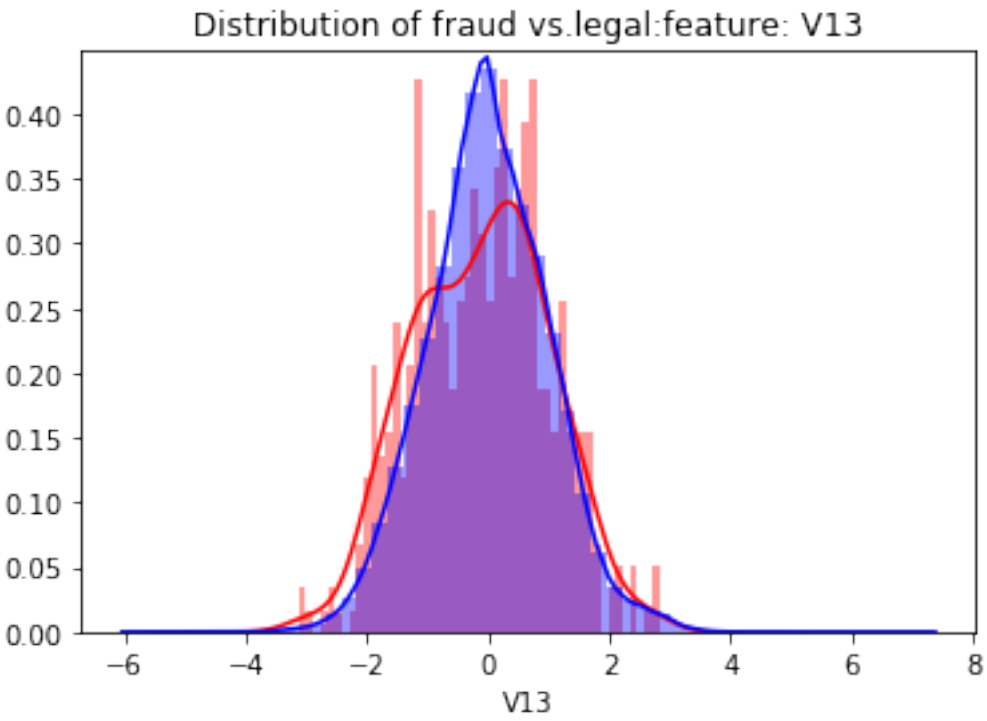

Distribution of fraud vs.legal:feature: V8
0.25
0.20
0.15
0.10
0.05
0.00
−60
−40
−20
0
20
V8

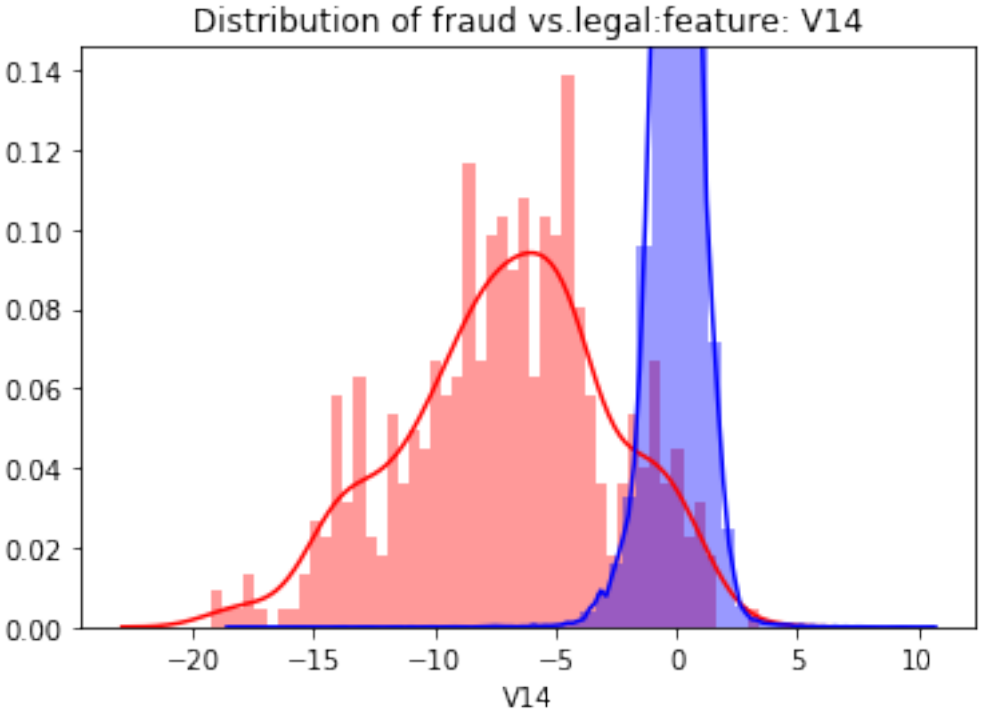

Distribution of fraud vs.legal:feature: V9
0.20
0.15
0.10
0.05
0.00
−15
−10
−5
0
5
10
15
V9

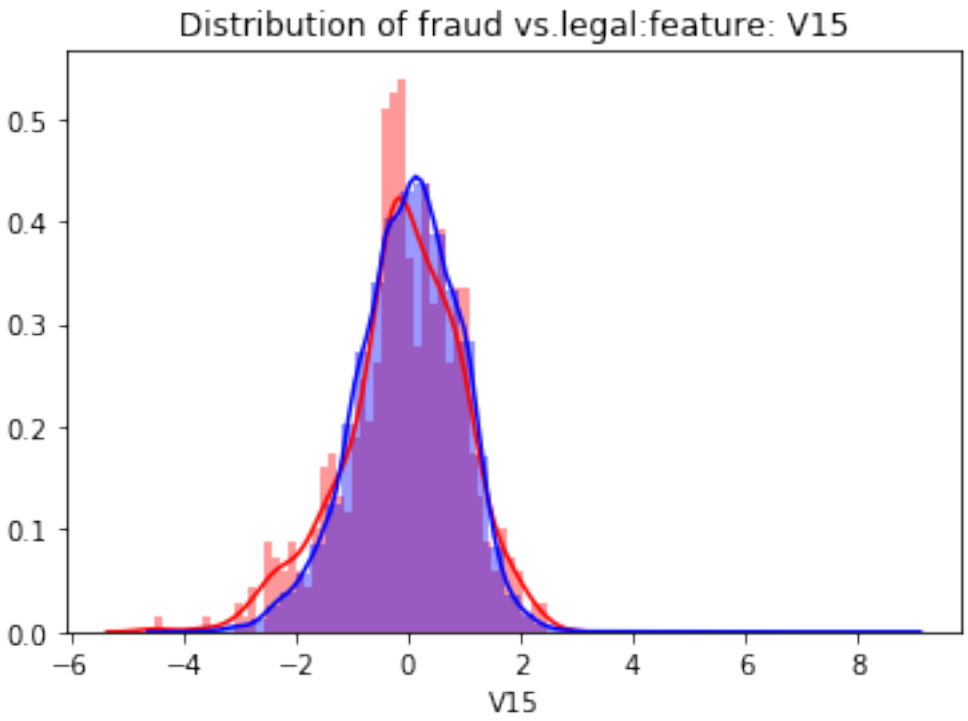

Distribution of fraud vs.legal:feature: V10
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−30
−20
−10
0
10
20
V10

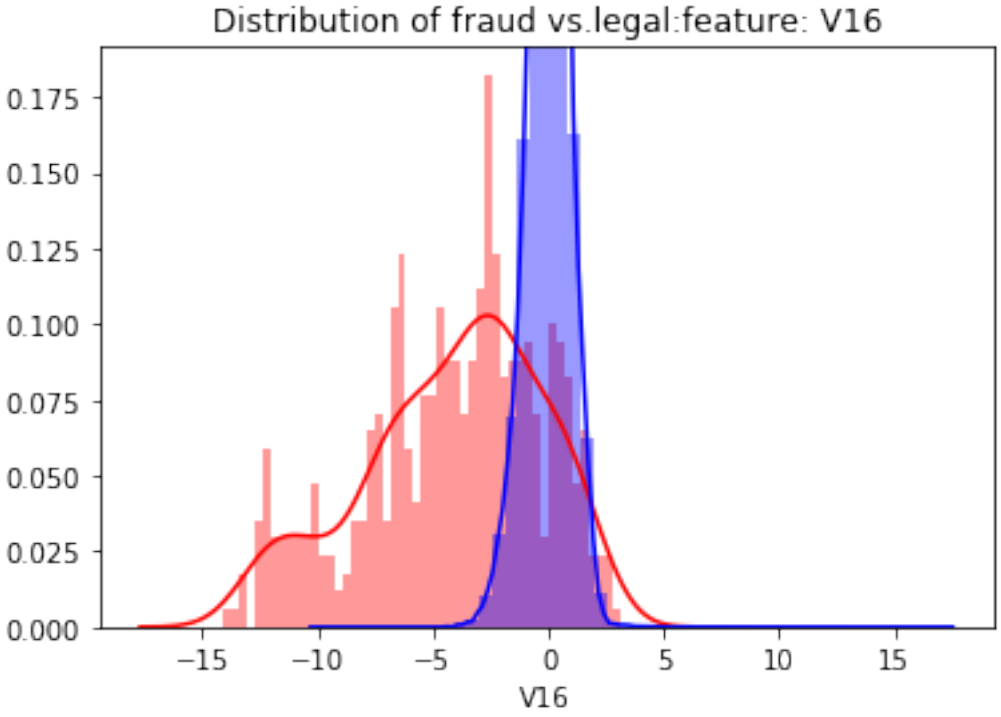

Distribution of fraud vs.legal:feature: V11
0.25
0.20
0.15
0.10
0.05
0.00
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
V11

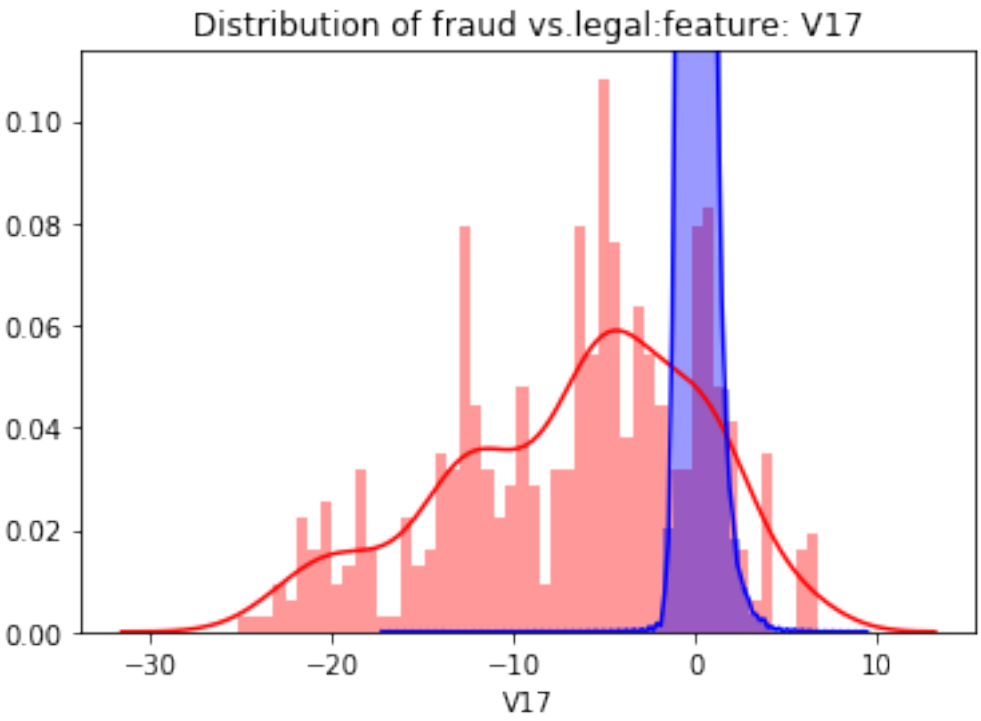

Distribution of fraud vs.legal:feature: V12
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−20
−15
−10
−5
0
5
V12

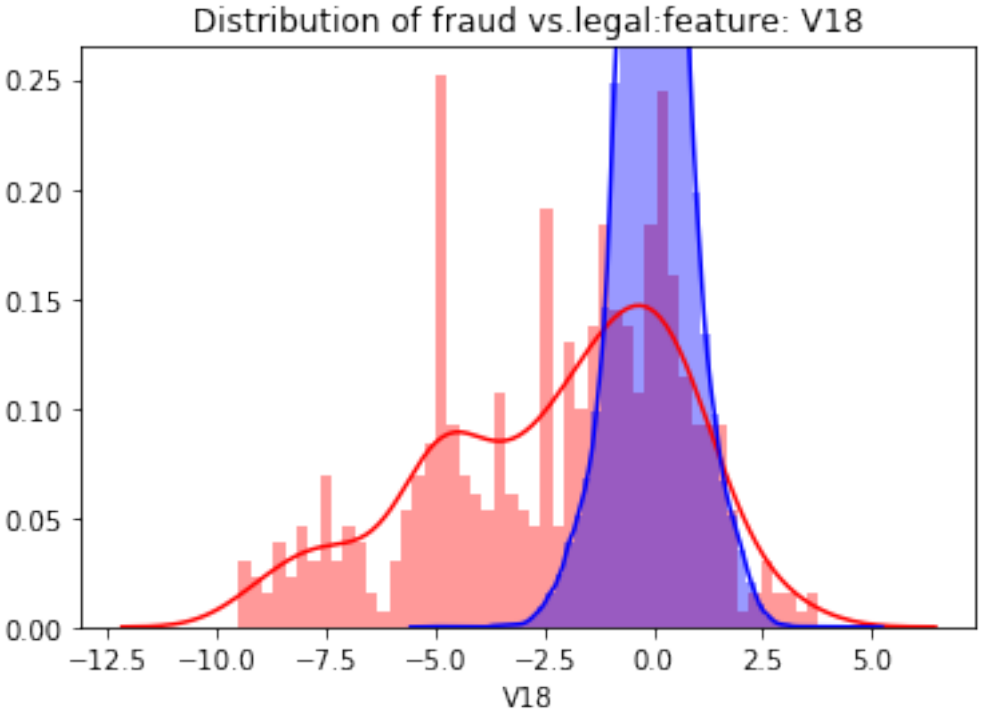

Distribution of fraud vs.legal:feature: V13
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−6
−4
−2
0
2
4
6
8
V13

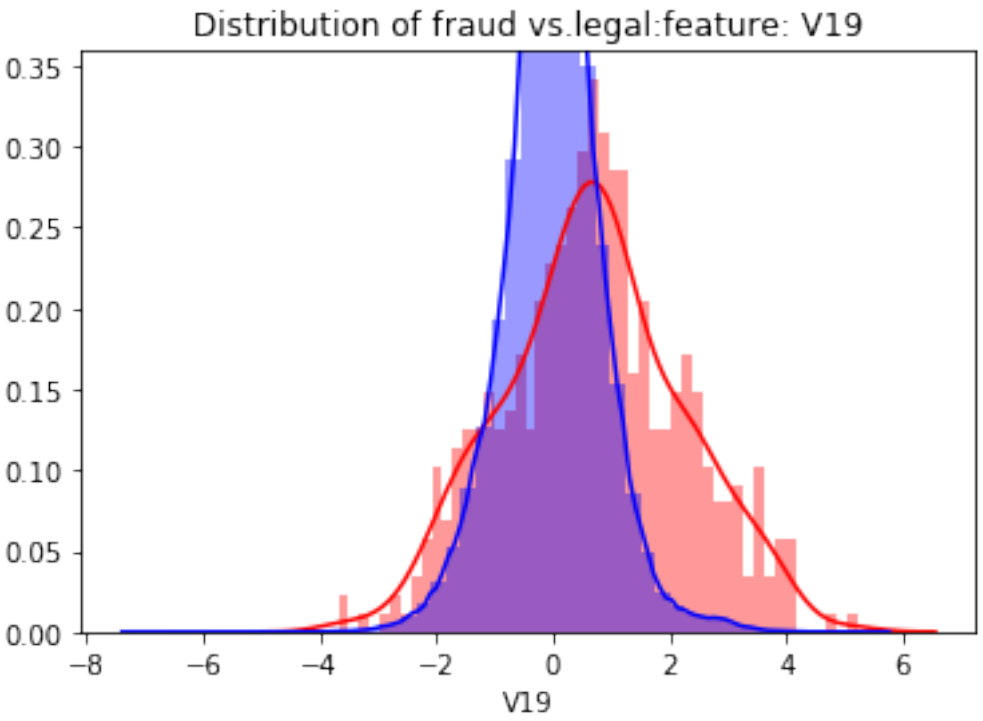

Distribution of fraud vs.legal:feature: V14
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−20
−15
−10
−5
0
5
10
V14

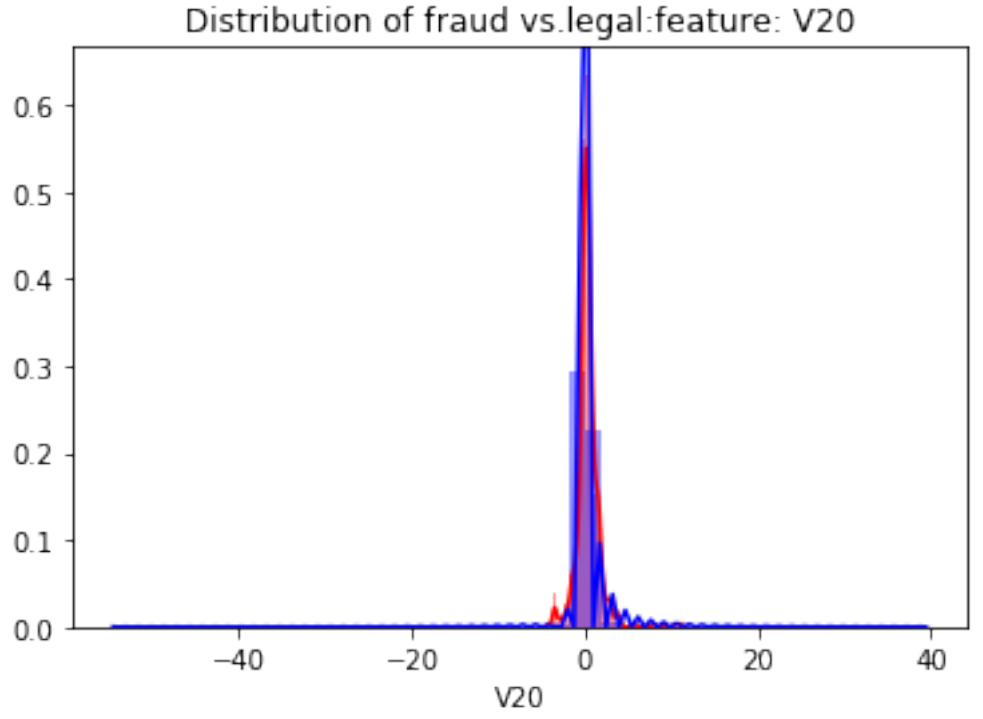

Distribution of fraud vs.legal:feature: V15
0.5
0.4
0.3
0.2
0.1
0.0
−6
−4
−2
0
2
4
6
8
V15

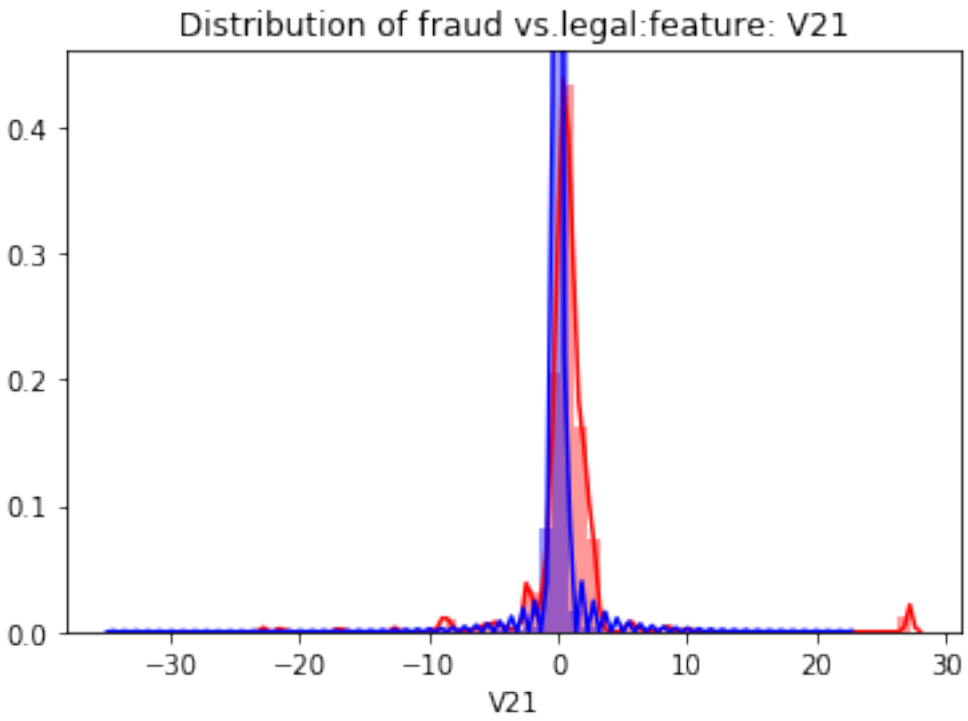

Distribution of fraud vs.legal:feature: V16
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−15
−10
−5
0
5
10
15
V16

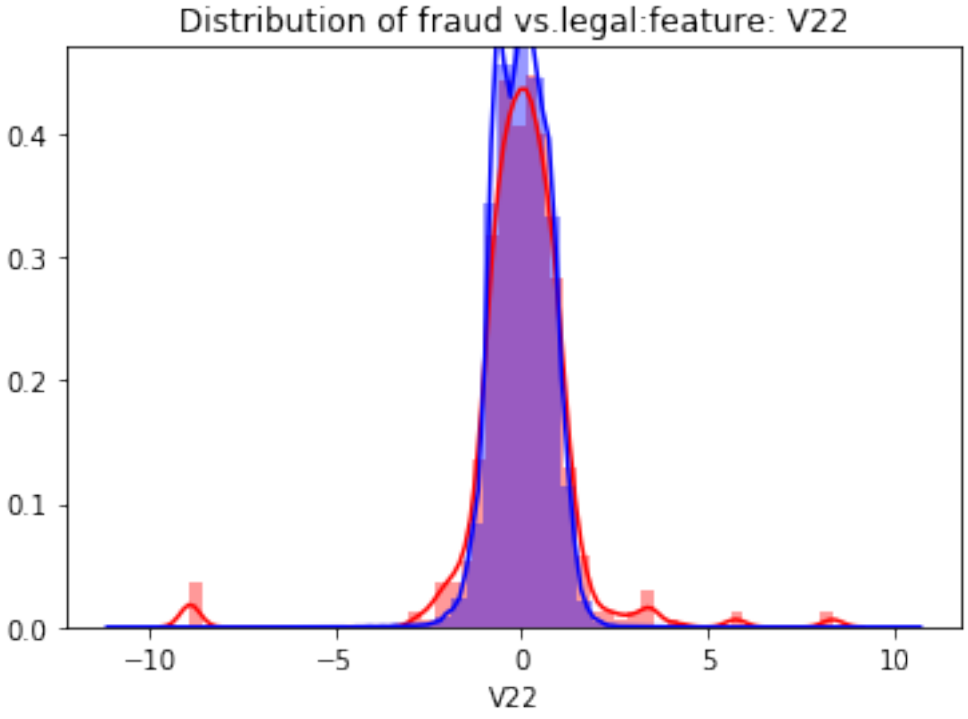

Distribution of fraud vs.legal:feature: V17
0.10
0.08
0.06
0.04
0.02
0.00
−30
−20
−10
0
10
V17

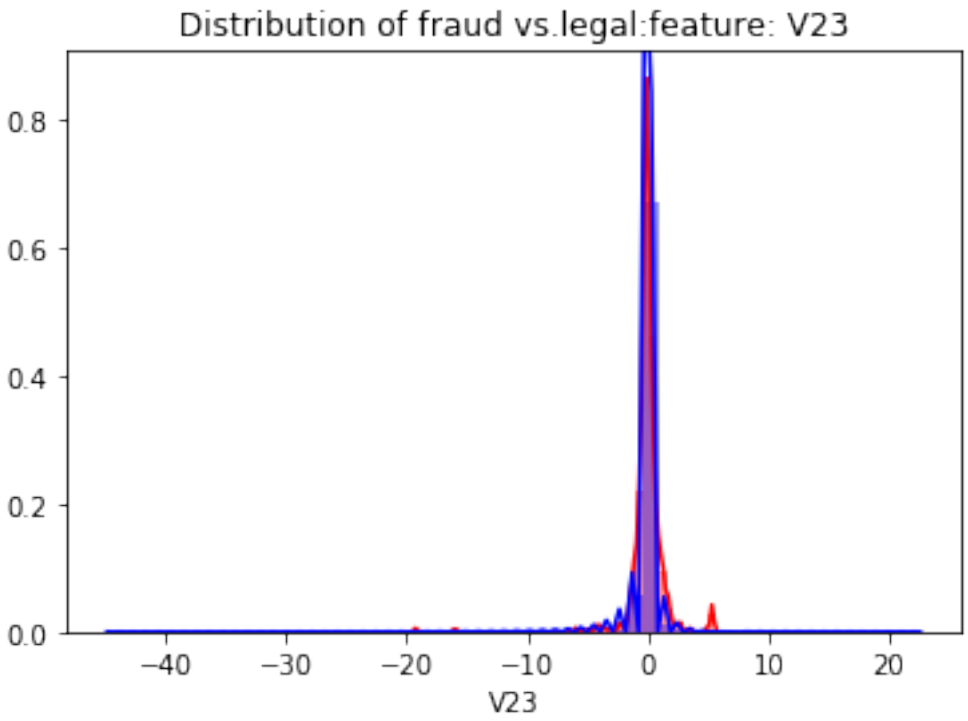

Distribution of fraud vs.legal:feature: V18
0.25
0.20
0.15
0.10
0.05
0.00
−12.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
V18

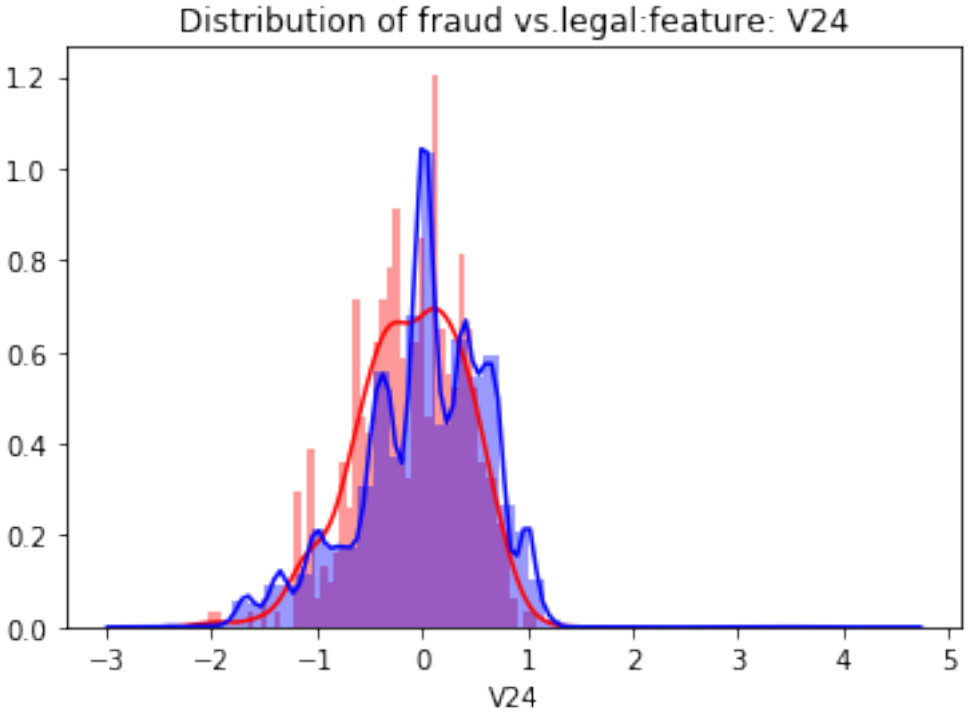

Distribution of fraud vs.legal:feature: V19
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−8
−6
−4
−2
0
2
4
6
V19

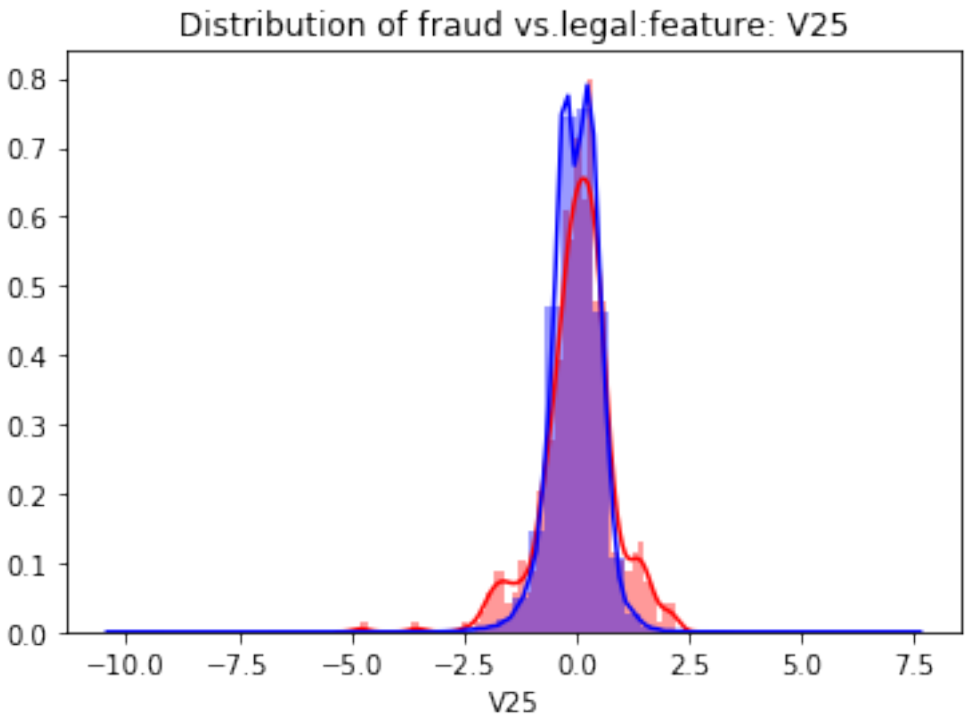

Distribution of fraud vs.legal:feature: V20
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−40
−20
0
20
40
V20

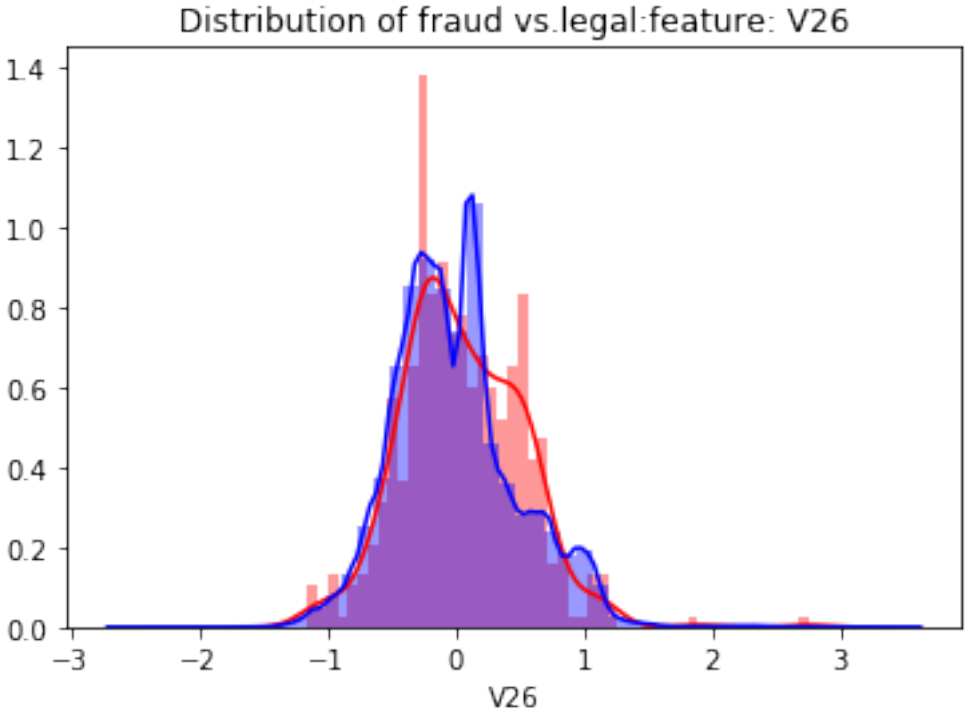

Distribution of fraud vs.legal:feature: V21
0.4
0.3
0.2
0.1
0.0
−30
−20
−10
0
10
20
30
V21

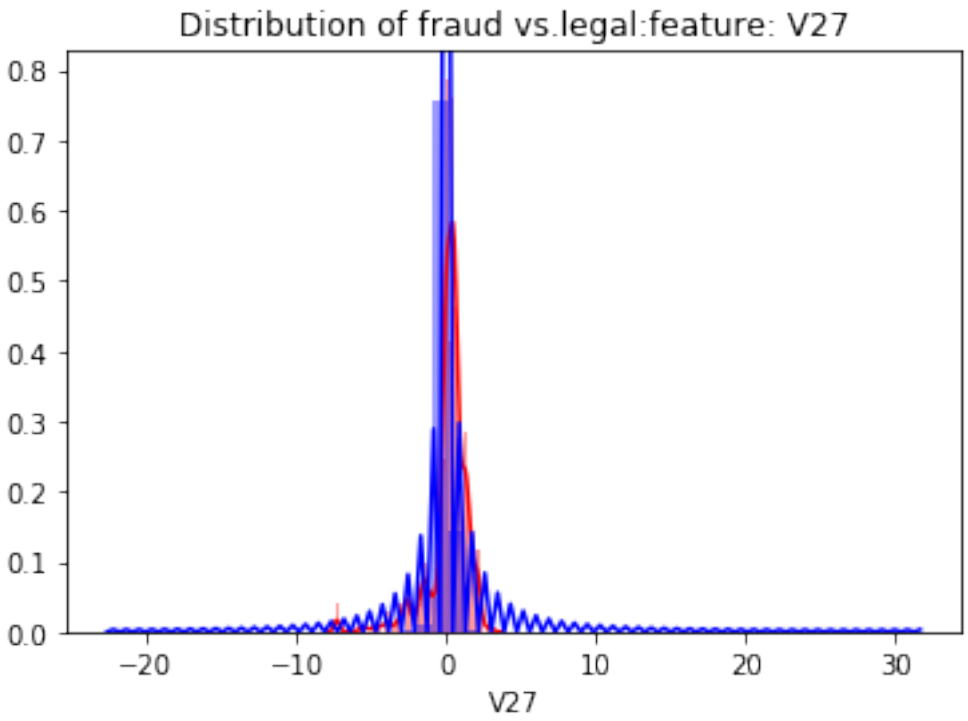

Distribution of fraud vs.legal:feature: V22
0.4
0.3
0.2
0.1
0.0
−10
−5
0
5
10
V22

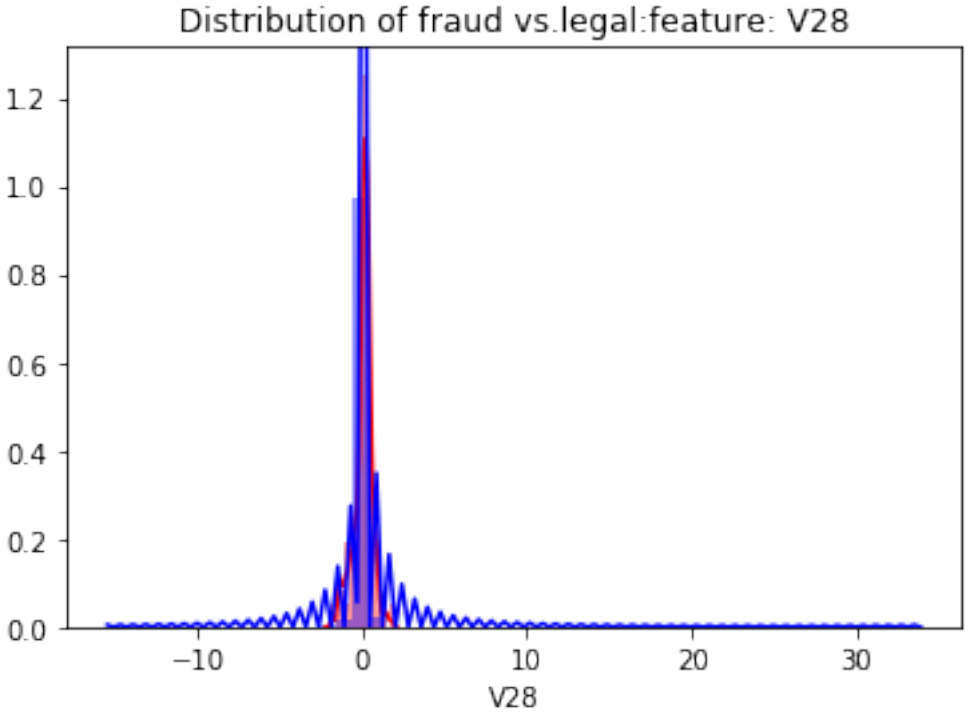

Distribution of fraud vs.legal:feature: V23
0.8
0.6
0.4
0.2
0.0
−40
−30
−20
−10
0
10
20
V23


Distribution of fraud vs.legal:feature: V24
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3
4
5
V24


Distribution of fraud vs.legal:feature: V25
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
V25


Distribution of fraud vs.legal:feature: V26
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−3
−2
−1
0
1
2
3
V26

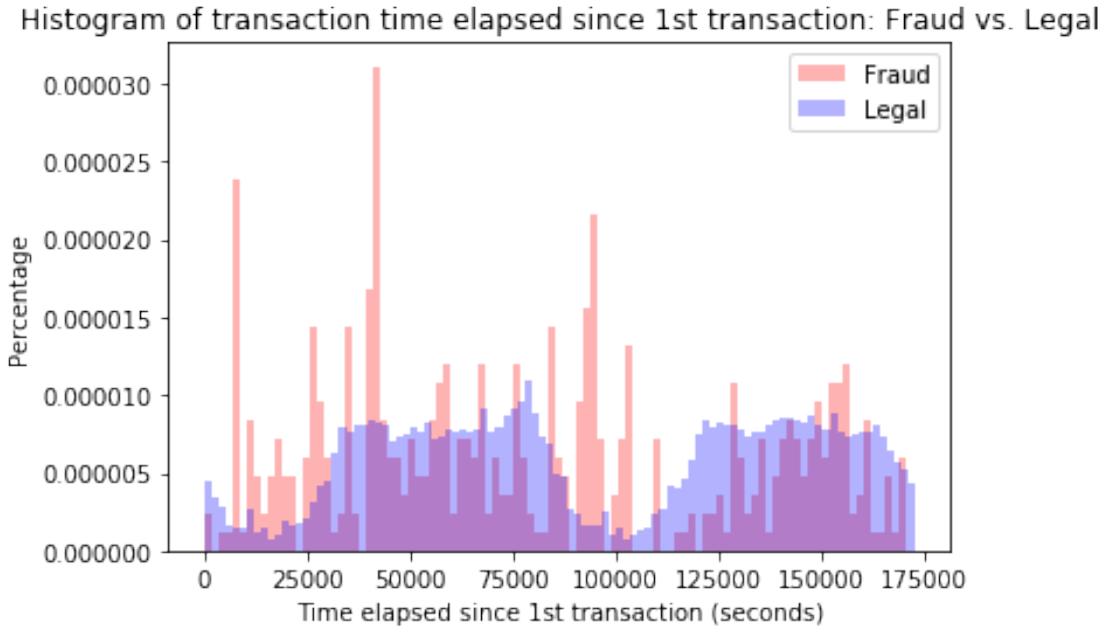

Distribution of fraud vs.legal:feature: V27
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−20
−10
0
10
20
30
V27


Distribution of fraud vs.legal:feature: V28
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−10
0
10
20
30
V28

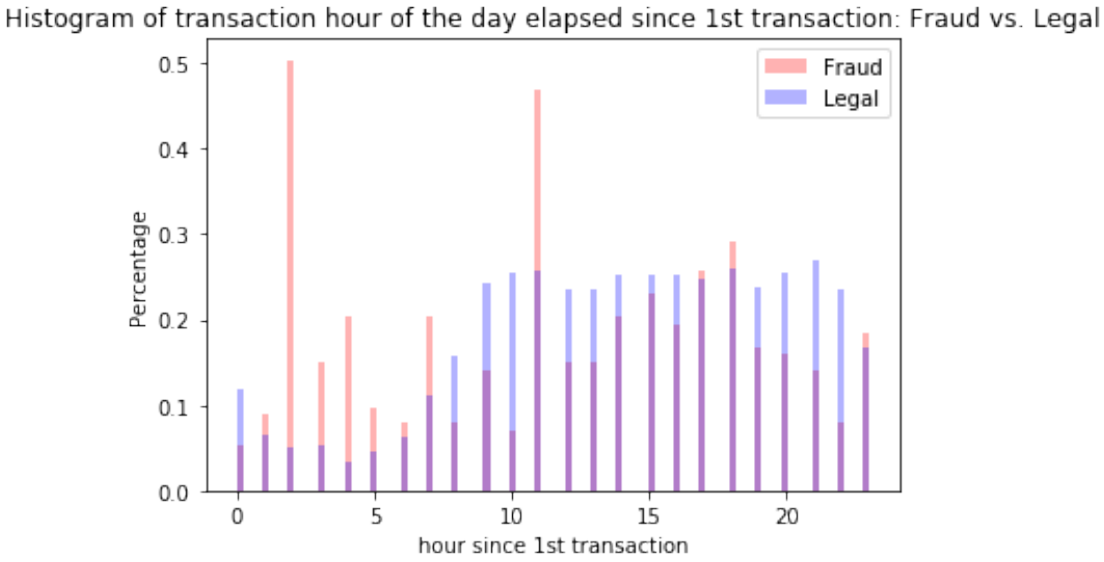
These plots confirm our previous conclusion that, while legal transactions spread over a larger range in general, it has relatively much narrower peak range.

Transaction time histograms are also compared between fraud (red) and legal (blue) in original 'Time' variable, and in 'hour' variable that we created earlier.

```
In [14]: #Time, distribution plots for fraud vs. legal
         plt.figure()
         frauds.Time.plot(kind='hist', bins=100, color = 'red', normed= True,label= 'Fraud', alp
         legals.Time.plot(kind='hist', bins=100, color = 'blue', normed= True, label= 'Legal',al
         plt.legend(loc='upper right')
         plt.title('Histogram of transaction time elapsed since 1st transaction: Fraud vs. Legal
         plt.xlabel('Time elapsed since 1st transaction (seconds)')
         plt.ylabel('Percentage')
         plt.show()
         plt.savefig('Fraud vs. Legal Transaction Time.png')
         #Check the hours
         plt.figure()
         frauds.hour.plot(kind='hist', bins=100, color = 'red', normed= True,label= 'Fraud', alp
         legals.hour.plot(kind='hist', bins=100, color = 'blue', normed= True, label= 'Legal',al
         plt.legend(loc='upper right')
         plt.title('Histogram of transaction hour of the day elapsed since 1st transaction: Frau
         plt.xlabel('hour since 1st transaction')
         plt.ylabel('Percentage')
         plt.show()
```




```
<matplotlib.figure.Figure at 0x10625b6d0>
```

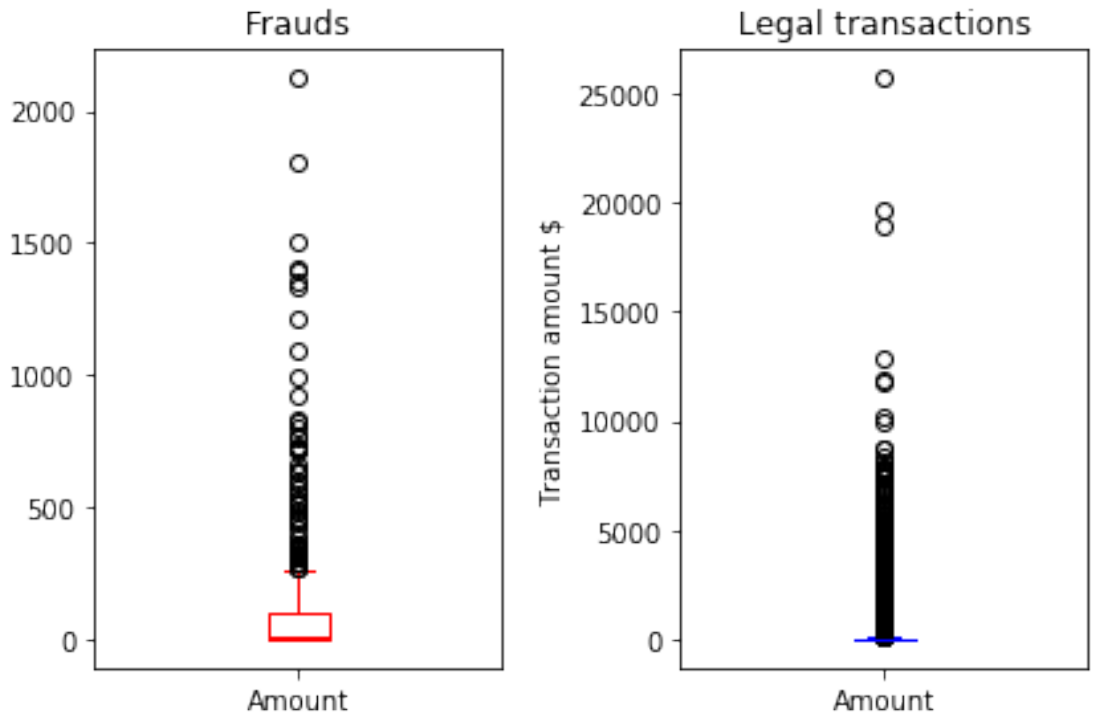



Both of the plots tell us that fraud transactions peak when the legal ones are low. We don't have detailed time (24-hour clock) of the day of when the first transaction happened to calculate these transaction hours, but it would be logical to assume that these fraud transactions probably happen more frequently at late night and/or early morning when legal transactions are normally low.

Finally, we compare the transaction amounts for fraud and legal transactions in both .describe() function and a side by side graph.

```
In [15]: #Amount,summary stats and plots for fraud vs. legal
         print(frauds.Amount.describe())
         print(legals.Amount.describe())
         plt.figure()
         plt.subplot(1,2,1)
         frauds.Amount.plot(kind='Box',color = 'red')
         plt.title('Frauds')
         plt.subplot(1,2,2)
         legals.Amount.plot(kind='Box',color = 'blue')
         plt.title('Legal transactions')
         plt.ylabel('Transaction amount $')
         plt.tight_layout()
         plt.show()

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
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
```




Fraud transactions are all much higher than zero (which is logical), and disperse throughout around 250 to 2000. Further, even though legal transactions have some extremely large values, the fraud transactions has a higher average (122 vs. 88) as can be obtained from the summary stats.

### 1.6 Inferential Statistics

Further, we carry out hypothesis testing comparing statstics between fraud and legal transactions. The specific tests will be performed are: * z-tests for V1-V28. The null hypothesis would be h0: fraud and legal transactions have the same mean PCA. * K-S test for Time. The null hypothesis would be h0: fraud and legal time are from the same distribution. * Bootstrap test for Amount. The null hypothesis would be h0: fraud and legal transactions have the same mean amount.

```
In [16]: #2.4 Hypothesis testing
         #z-test for V1-V28 features
         #legal vs. fraud
         #significant level 99%. Null Hypothesis: Vi fraud = Vi legal
         #any p< 0.10 will reject the null hypothesis
         for col in pca_names:
             p1 =  frauds[col]
             p0 = legals[col]
             z_cal, p_cal= ztest(x1=p1, x2=p0, value=0, alternative='two-sided', usevar='pooled'
             print(col,'statistically significant difference' if p_cal < 0.01 else 'statisticall
```

```
('V1', 'statistically significant difference')
('V2', 'statistically significant difference')
('V3', 'statistically significant difference')
('V4', 'statistically significant difference')
('V5', 'statistically significant difference')
('V6', 'statistically significant difference')
('V7', 'statistically significant difference')
('V8', 'statistically significant difference')
('V9', 'statistically significant difference')
('V10', 'statistically significant difference')
('V11', 'statistically significant difference')
('V12', 'statistically significant difference')
('V13', 'statistically insignificant difference')
('V14', 'statistically significant difference')
('V15', 'statistically insignificant difference')
('V16', 'statistically significant difference')
('V17', 'statistically significant difference')
('V18', 'statistically significant difference')
('V19', 'statistically significant difference')
('V20', 'statistically significant difference')
('V21', 'statistically significant difference')
('V22', 'statistically insignificant difference')
('V23', 'statistically insignificant difference')
('V24', 'statistically significant difference')
('V25', 'statistically insignificant difference')
('V26', 'statistically insignificant difference')
('V27', 'statistically significant difference')
('V28', 'statistically significant difference')
```

```
In [21]: #K-S test for time(both Time and hour)
         ks_cal1, p_cal1=stats.ks_2samp(frauds.Time, legals.Time)
         print('K-S statistics for Time:', ks_cal1, 'p-value:',p_cal1)
         ks_cal2, p_cal2=stats.ks_2samp(frauds.hour, legals.hour)
         print('K-S statistics for hour:', ks_cal2, 'p-value:',p_cal2)
```

```
('K-S statistics for Time:', 0.16938909937434854, 'p-value:', 8.357813751103828e-13)
```

```
('K-S statistics for hour:', 0.1935411727716982, 'p-value:', 1.3815502491489748e-16)
```

For both (original) Time and hour, the p-values are small enough to reject the null hypothesis that the frauds and legals have the same distribution. The results are consistant with what we've obtained earlier using visual EDA.

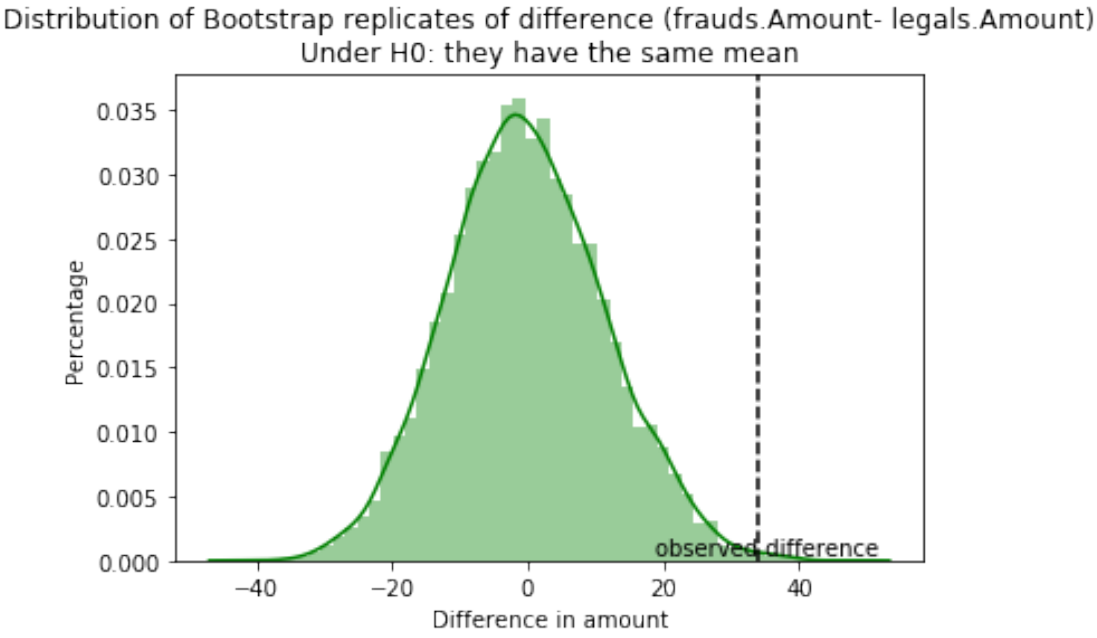
We then carry out a bootstrap test for amount as well to see if the difference between frauds mean and legals mean is purely due to chance. Therefore our null hypothesis h0: the mean of fraud amount = legal amount.

In [18]:
```
#bootstrapping for Amount
#legal vs. fraud
#first define relevant functions:
def bootstrap_replicate_1d(data, func):
    """Generate bootstrap replicate of 1D Data"""
    bs_sample = np.random.choice(data.values, len(data))

    return func(bs_sample)

def draw_bs_reps(data, func, size=1):
    """Draw bootstrap replicates."""
    bs_replicates = np.empty(size)
    for i in range(size):
        bs_replicates[i] = bootstrap_replicate_1d(data,func)

    return bs_replicates
```

In [19]:
```
#Hypothesis testing: significant level alpha =0.01, Null Hypthesis: they have the same
mean_amount = np.mean(df.Amount.values)
obs_diff_mean = np.mean(frauds.Amount.values) - np.mean(legals.Amount.values)

frauds_shifted = frauds.Amount - np.mean(frauds.Amount.values) + mean_amount
legals_shifted = legals.Amount - np.mean(legals.Amount.values) + mean_amount

bs_replicates_frauds = draw_bs_reps(frauds_shifted, np.mean, 10000)
bs_replicates_legals = draw_bs_reps(legals_shifted, np.mean, 10000)

bs_replicates = bs_replicates_frauds - bs_replicates_legals

# Compute and print p-value: p
p = np.sum(bs_replicates >= obs_diff_mean) / len(bs_replicates)        #because obs_dif
print('p-value =', format(p, '.6f'))

#visualize the bootstrap replicates results
sns.distplot(bs_replicates, bins = 50, norm_hist = True, color = 'green')
plt.axvline(x=obs_diff_mean, color='k', linestyle='--')
plt.text(obs_diff_mean-15, 0.0005, 'observed difference')
plt.xlabel('Difference in amount')
```

```
        plt.ylabel('Percentage')
        plt.title('Distribution of Bootstrap replicates of difference (frauds.Amount- legals.Am
        plt.show()
```

```
('p-value =', '0.000000')
```




As we can see here, the difference in means of fraud transactions and legal transactions is highly unlikely due to chance, as p-value for getting a observed difference of ~33.92 or more extreme is 0.000000.

**Summary:**

This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. * * Highly unbalanced, frauds (class ==1) only account for 0.172% of all transactions. * No missing value is present as concluded from EDA, and therefore there is no need for cleaning the data. * V1-V28 were preprocessed by PCA to maintain confidentiality of sensitive information and therfore PCA features outliers, if any, are not completely meaningless. For the outliers in the amount, we see that outliers almost exclusively come from legal transactions. Depending on linearity of models built later, I will decide on whether to remove the outliers or not on a case-by-case basis. * V1-V28, time(hour) and amount in general, all demonstrate different stats properties (mean, std, distribution etc.) in fraud and legal transactions, so it would be reasonable to include them in the prediction model.

The seperate code could also be found at https://github.com/chocolocked/Capstone_1_CreditCardFraud

## 1.7 References

*(1) https://www.forbes.com/sites/rogeraitken/2016/10/26/us-card-fraud-losses-could-exceed-12bn-by-2020/#53aa8baed243

*(2) K.R., Seeja & Zareapoor, Masoumeh. (2014). FraudMiner: A Novel Credit Card Fraud Detection Model Based on Frequent Itemset Mining. TheScientificWorldJournal. 2014. 252797. 10.1155/2014/252797.

TO DO (based on readings) Re-scale the features? Select optimum feature numbers for different models? Some drop the time, some don't. Using metrics: precision, recall, f1-score, and ROC AUC