Credit Card Fraud Detection

Presented by Sadhna

Throughout the financial sector, machine learning algorithms are being developed to detect fraudulent transactions. In this project, that is exactly what we are going to be doing as well. Using a dataset of of nearly 28,500 credit card transactions and multiple unsupervised anomaly detection algorithms, we are going to identify transactions with a high probability of being credit card fraud. In this project, we will build and deploy the following two machine learning algorithms:

Local Outlier Factor (LOF) Isolation Forest Algorithm

Furthermore, using metrics suchs as precision, recall, and F1-scores, we will investigate why the classification accuracy for these algorithms can be misleading.

In addition, we will explore the use of data visualization techniques common in data science, such as parameter histograms and correlation matrices, to gain a better understanding of the underlying distribution of data in our data set. Let's get started!

1. Importing Necessary Libraries

To start, let's print out the version numbers of all the libraries we will be using in this project. This serves two purposes - it ensures we have installed the libraries correctly and ensures that this tutorial will be reproducible.

In [1]:

```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy



#print('Python:{}',format(sys.version))
#print('Numpy:{}',format(np.__version__))
#print('pandas:{}',format(pd.__version__))
#print('matplotlib:{}',format(plt.__version__))
#print('seaborn:{}',format(sns.__version__))
#print('scipy:{}',format(scipy.__version__))#/
```

2. The Data Set A

In the following cells, we will import our dataset from a .csv file as a Pandas DataFrame. Furthermore, we will begin exploring the dataset to gain an understanding of the type, quantity, and distribution of data in our dataset. For this purpose, we will use Pandas' built-in describe feature, as well as parameter histograms and a correlation matrix.

In [2]:

```
# Load the dataset from the csv file using pandas

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

Out[2]:

```
<bound method NDFrame.head of              Time         V1         V2         V3         V4         V5
0             0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1             0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2             1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3             1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4             2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
5             2.0  -0.425966   0.960523  1.141109 -0.168252  0.420987
6             4.0   1.229658   0.141004  0.045371  1.202613  0.191881
7             7.0  -0.644269   1.417964  1.074380 -0.492199  0.948934
8             7.0  -0.894286   0.286157 -0.113192 -0.271526  2.669599
9             9.0  -0.338262   1.119593  1.044367 -0.222187  0.499361
```

```
10          10.0   1.449044  -1.176339  0.913860 -1.375667 -1.971383
11          10.0   0.384978   0.616109 -0.874300 -0.094019  2.924584
12          10.0   1.249999  -1.221637  0.383930 -1.234899 -1.485419
13          11.0   1.069374   0.287722  0.828613  2.712520 -0.178398
14          12.0  -2.791855  -0.327771  1.641750  1.767473 -0.136588
15          12.0  -0.752417   0.345485  2.057323 -1.468643 -1.158394
16          12.0   1.103215  -0.040296  1.267332  1.289091 -0.735997
17          13.0  -0.436905   0.918966  0.924591 -0.727219  0.915679
18          14.0  -5.401258  -5.450148  1.186305  1.736239  3.049106
19          15.0   1.492936  -1.029346  0.454795 -1.438026 -1.555434
20          16.0   0.694885  -1.361819  1.029221  0.834159 -1.191209
21          17.0   0.962496   0.328461 -0.171479  2.109204  1.129566
22          18.0   1.166616   0.502120 -0.067300  2.261569  0.428804
23          18.0   0.247491   0.277666  1.185471 -0.092603 -1.314394
24          22.0  -1.946525  -0.044901 -0.405570 -1.013057  2.941968
25          22.0  -2.074295  -0.121482  1.322021  0.410008  0.295198
26          23.0   1.173285   0.353498  0.283905  1.133563 -0.172577
27          23.0   1.322707  -0.174041  0.434555  0.576038 -0.836758
28          23.0  -0.414289   0.905437  1.727453  1.473471  0.007443
29          23.0   1.059387  -0.175319  1.266130  1.186110 -0.786002
...          ...        ...        ...       ...       ...       ...
284777  172764.0   2.079137  -0.028723 -1.343392  0.358000 -0.045791
284778  172764.0  -0.764523   0.588379 -0.907599 -0.418847  0.901528
284779  172766.0   1.975178  -0.616244 -2.628295 -0.406246  2.327804
284780  172766.0  -1.727503   1.108356  2.219561  1.148583 -0.884199
284781  172766.0  -1.139015  -0.155510  1.894478 -1.138957  1.451777
284782  172767.0  -0.268061   2.540315 -1.400915  4.846661  0.639105
284783  172768.0  -1.796092   1.929178 -2.828417 -1.689844  2.199572
284784  172768.0  -0.669662   0.923769 -1.543167 -1.560729  2.833960
284785  172768.0   0.032887   0.545338 -1.185844 -1.729828  2.932315
284786  172768.0  -2.076175   2.142238 -2.522704 -1.888063  1.982785
284787  172769.0  -1.029719  -1.110670 -0.636179 -0.840816  2.424360
284788  172770.0   2.007418  -0.280235 -0.208113  0.335261 -0.715798
284789  172770.0  -0.446951   1.302212 -0.168583  0.981577  0.578957
284790  172771.0  -0.515513   0.971950 -1.014580 -0.677037  0.912430
284791  172774.0  -0.863506   0.874701  0.420358 -0.530365  0.356561
284792  172774.0  -0.724123   1.485216 -1.132218 -0.607190  0.709499
284793  172775.0   1.971002  -0.699067 -1.697541 -0.617643  1.718797
284794  172777.0  -1.266580  -0.400461  0.956221 -0.723919  1.531993
284795  172778.0 -12.516732  10.187818 -8.476671 -2.510473 -4.586669
284796  172780.0   1.884849  -0.143540 -0.999943  1.506772 -0.035300
284797  172782.0  -0.241923   0.712247  0.399806 -0.463406  0.244531
284798  172782.0   0.219529   0.881246 -0.635891  0.960928 -0.152971
284799  172783.0  -1.775135  -0.004235  1.189786  0.331096  1.196063
284800  172784.0   2.039560  -0.175233 -1.196825  0.234580 -0.008713
284801  172785.0   0.120316   0.931005 -0.546012 -0.745097  1.130314
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  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278
5      -0.029728  0.476201  0.260314 -0.568671  ... -0.208254 -0.559825
6       0.272708 -0.005159  0.081213  0.464960  ... -0.167716 -0.270710
7       0.428118  1.120631 -3.807864  0.615375  ...  1.943465 -1.015455
8       3.721818  0.370145  0.851084 -0.392048  ... -0.073425 -0.268092
9      -0.246761  0.651583  0.069539 -0.736727  ... -0.246914 -0.633753
10     -0.629152 -1.423236  0.048456 -1.720408  ... -0.009302  0.313894
11      3.317027  0.470455  0.538247 -0.558895  ...  0.049924  0.238422
12     -0.753230 -0.689405 -0.227487 -2.094011  ... -0.231809 -0.483285
13      0.337544 -0.096717  0.115982 -0.221083  ... -0.036876  0.074412
14      0.807596 -0.422911 -1.907107  0.755713  ...  1.151663  0.222182
15     -0.077850 -0.608581  0.003603 -0.436167  ...  0.499625  1.353650
16      0.288069 -0.586057  0.189380  0.782333  ... -0.024612  0.196002
17     -0.127867  0.707642  0.087962 -0.665271  ... -0.194796 -0.672638
18     -1.763406 -1.559738  0.160842  1.233090  ... -0.503600  0.984460
19     -0.720961 -1.080664 -0.053127 -1.978682  ... -0.177650 -0.175074
20      1.309109 -0.878586  0.445290 -0.446196  ... -0.295583 -0.571955
21      1.696038  0.107712  0.521502 -1.191311  ...  0.143997  0.402492
22      0.089474  0.241147  0.138082 -0.989162  ...  0.018702 -0.061972
23     -0.150116 -0.946365 -1.617935  1.544071  ...  1.650180  0.200454
```

```
24       2.955053 -0.063063  0.855546  0.049967  ... -0.579526 -0.799229
25      -0.959537  0.543985 -0.104627  0.475664  ... -0.403639 -0.227404
26      -0.916054  0.369025 -0.327260 -0.246651  ...  0.067003  0.227812
27      -0.831083 -0.264905 -0.220982 -1.071425  ... -0.284376 -0.323357
28      -0.200331  0.740228 -0.029247 -0.593392  ...  0.077237  0.457331
29       0.578435 -0.767084  0.401046  0.699500  ...  0.013676  0.213734
...           ...       ...       ...       ...  ...       ...       ...
284777 -1.345452  0.227476 -0.378355  0.665911  ...  0.235758  0.829758
284778 -0.760802  0.758545  0.414698 -0.730854  ...  0.003530 -0.431876
284779  3.664740 -0.533297  0.842937  1.128798  ...  0.086043  0.543613
284780  0.793083 -0.527298  0.866429  0.853819  ... -0.094708  0.236818
284781  0.093598  0.191353  0.092211 -0.062621  ... -0.191027 -0.631658
284782  0.186479 -0.045911  0.936448 -2.419986  ... -0.263889 -0.857904
284783  3.123732 -0.270714  1.657495  0.465804  ...  0.271170  1.145750
284784  3.240843  0.181576  1.282746 -0.893890  ...  0.183856  0.202670
284785  3.401529  0.337434  0.925377 -0.165663  ... -0.266113 -0.716336
284786  3.732950 -1.217430 -0.536644  0.272867  ...  2.016666 -1.588269
284787 -2.956733  0.283610 -0.332656 -0.247488  ...  0.353722  0.488487
284788 -0.751373 -0.458972 -0.140140  0.959971  ... -0.208260 -0.430347
284789 -0.605641  1.253430 -1.042610 -0.417116  ...  0.851800  0.305268
284790 -0.316187  0.396137  0.532364 -0.224606  ... -0.280302 -0.849919
284791 -1.046238  0.757051  0.230473 -0.506856  ... -0.108846 -0.480820
284792 -0.482638  0.548393  0.343003 -0.226323  ...  0.414621  1.307511
284793  3.911336 -1.259306  1.056209  1.315006  ...  0.188758  0.694418
284794 -1.788600  0.314741  0.004704  0.013857  ... -0.157831 -0.883365
284795 -1.394465 -3.632516  5.498583  4.893089  ... -0.944759 -1.565026
284796 -0.613638  0.190241 -0.249058  0.666458  ...  0.144008  0.634646
284797 -1.343668  0.929369 -0.206210  0.106234  ... -0.228876 -0.514376
284798 -1.014307  0.427126  0.121340 -0.285670  ...  0.099936  0.337120
284799  5.519980 -1.518185  2.080825  1.159498  ...  0.103302  0.654850
284800 -0.726571  0.017050 -0.118228  0.435402  ... -0.268048 -0.717211
284801 -0.235973  0.812722  0.115093 -0.204064  ... -0.314205 -0.808520
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  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
5      -0.026398 -0.371427 -0.232794  0.105915  0.253844  0.081080    3.67
6      -0.154104 -0.780055  0.750137 -0.257237  0.034507  0.005168    4.99
7       0.057504 -0.649709 -0.415267 -0.051634 -1.206921 -1.085339   40.80
8      -0.204233  1.011592  0.373205 -0.384157  0.011747  0.142404   93.20
9      -0.120794 -0.385050 -0.069733  0.094199  0.246219  0.083076    3.68
10      0.027740  0.500512  0.251367 -0.129478  0.042850  0.016253    7.80
11      0.009130  0.996710 -0.767315 -0.492208  0.042472 -0.054337    9.99
12      0.084668  0.392831  0.161135 -0.354990  0.026416  0.042422  121.50
13     -0.071407  0.104744  0.548265  0.104094  0.021491  0.021293   27.50
14      1.020586  0.028317 -0.232746 -0.235557 -0.164778 -0.030154   58.80
15     -0.256573 -0.065084 -0.039124 -0.087086 -0.180998  0.129394   15.99
16      0.013802  0.103758  0.364298 -0.382261  0.092809  0.037051   12.99
17     -0.156858 -0.888386 -0.342413 -0.049027  0.079692  0.131024    0.89
18      2.458589  0.042119 -0.481631 -0.621272  0.392053  0.949594   46.80
19      0.040002  0.295814  0.332931 -0.220385  0.022298  0.007602    5.00
20     -0.050881 -0.304215  0.072001 -0.422234  0.086553  0.063499  231.71
21     -0.048508 -1.371866  0.390814  0.199964  0.016371 -0.014605   34.09
22     -0.103855 -0.370415  0.603200  0.108556 -0.040521 -0.011418    2.28
23     -0.185353  0.423073  0.820591 -0.227632  0.336634  0.250475   22.75
24      0.870300  0.983421  0.321201  0.149650  0.707519  0.014600    0.89
25      0.742435  0.398535  0.249212  0.274404  0.359969  0.243232   26.43
26     -0.150487  0.435045  0.724825 -0.337082  0.016368  0.030041   41.88
27     -0.037710  0.347151  0.559639 -0.280158  0.042335  0.028822   16.00
28     -0.038500  0.642522 -0.183891 -0.277464  0.182687  0.152665   33.00
29      0.014462  0.002951  0.294638 -0.395070  0.081461  0.024220   12.99
...          ...       ...       ...       ...       ...       ...     ...
284777 -0.002063  0.001344  0.262183 -0.105327 -0.022363 -0.060283    1.00
284778  0.141759  0.587119 -0.200998  0.267337 -0.152951 -0.065285   80.00
284779 -0.032129  0.768379  0.477688 -0.031833  0.014151 -0.066542   25.00
284780 -0.204280  1.158185  0.627801 -0.399981  0.510818  0.233265   30.00
284781 -0.147249  0.212931  0.354257 -0.241068 -0.161717 -0.149188   13.00
284782  0.235172 -0.681794 -0.668894  0.044657 -0.066751 -0.072447   12.82
284783  0.084783  0.721269 -0.529906 -0.240117  0.129126 -0.080620   11.46
```

```
284784 -0.373023  0.651122  1.073823  0.844590 -0.286676 -0.187719   40.00
284785  0.108519  0.688519 -0.460220  0.161939  0.265368  0.090245    1.79
284786  0.588482  0.632444 -0.201064  0.199251  0.438657  0.172923    8.95
284787  0.293632  0.107812 -0.935586  1.138216  0.025271  0.255347    9.99
284788  0.416765  0.064819 -0.608337  0.268436 -0.028069 -0.041367    3.99
284789 -0.148093 -0.038712  0.010209 -0.362666  0.503092  0.229921   60.50
284790  0.300245  0.000607 -0.376379  0.128660 -0.015205 -0.021486    9.81
284791 -0.074513 -0.003988 -0.113149  0.280378 -0.077310  0.023079   20.32
284792 -0.059545  0.242669 -0.665424 -0.269869 -0.170579 -0.030692    3.99
284793  0.163002  0.726365 -0.058282 -0.191813  0.061858 -0.043716    4.99
284794  0.088485 -0.076790 -0.095833  0.132720 -0.028468  0.126494    0.89
284795  0.890675 -1.253276  1.786717  0.320763  2.090712  1.232864    9.87
284796 -0.042114 -0.053206  0.316403 -0.461441  0.018265 -0.041068   60.00
284797  0.279598  0.371441 -0.559238  0.113144  0.131507  0.081265    5.49
284798  0.251791  0.057688 -1.508368  0.144023  0.181205  0.215243   24.05
284799 -0.348929  0.745323  0.704545 -0.127579  0.454379  0.130308   79.99
284800  0.297930 -0.359769 -0.315610  0.201114 -0.080826 -0.075071    2.68
284801  0.050343  0.102800 -0.435870  0.124079  0.217940  0.068803    2.69
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
0           0
1           0
2           0
3           0
4           0
5           0
6           0
7           0
8           0
9           0
10          0
11          0
12          0
13          0
14          0
15          0
16          0
17          0
18          0
19          0
20          0
21          0
22          0
23          0
24          0
25          0
26          0
27          0
28          0
29          0
...       ...
284777      0
284778      0
284779      0
284780      0
284781      0
284782      0
284783      0
284784      0
284785      0
284786      0
284787      0
284788      0
284789      0
284790      0
284791      0
284792      0
284793      0
284794      0
284795      0
284796      0
284797      0
```

```
284798      0
284799      0
284800      0
284801      0
284802      0
284803      0
284804      0
284805      0
284806      0

[284807 rows x 31 columns]>
```

In [9]:

```python
# Start exploring the dataset

print(data.columns)
```

```
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 [10]:

```python
print(data.shape)
```

```
(284807, 31)
```

In [11]:

```python
print(data.describe())
```

```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.165980e-15  3.416908e-16 -1.373150e-15  2.086869e-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   9.604066e-16  1.490107e-15 -5.556467e-16  1.177556e-16 -2.406455e-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   ...  1.656562e-16 -3.444850e-16  2.578648e-16  4.471968e-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   5.340915e-16  1.687098e-15 -3.666453e-16 -1.220404e-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]
```

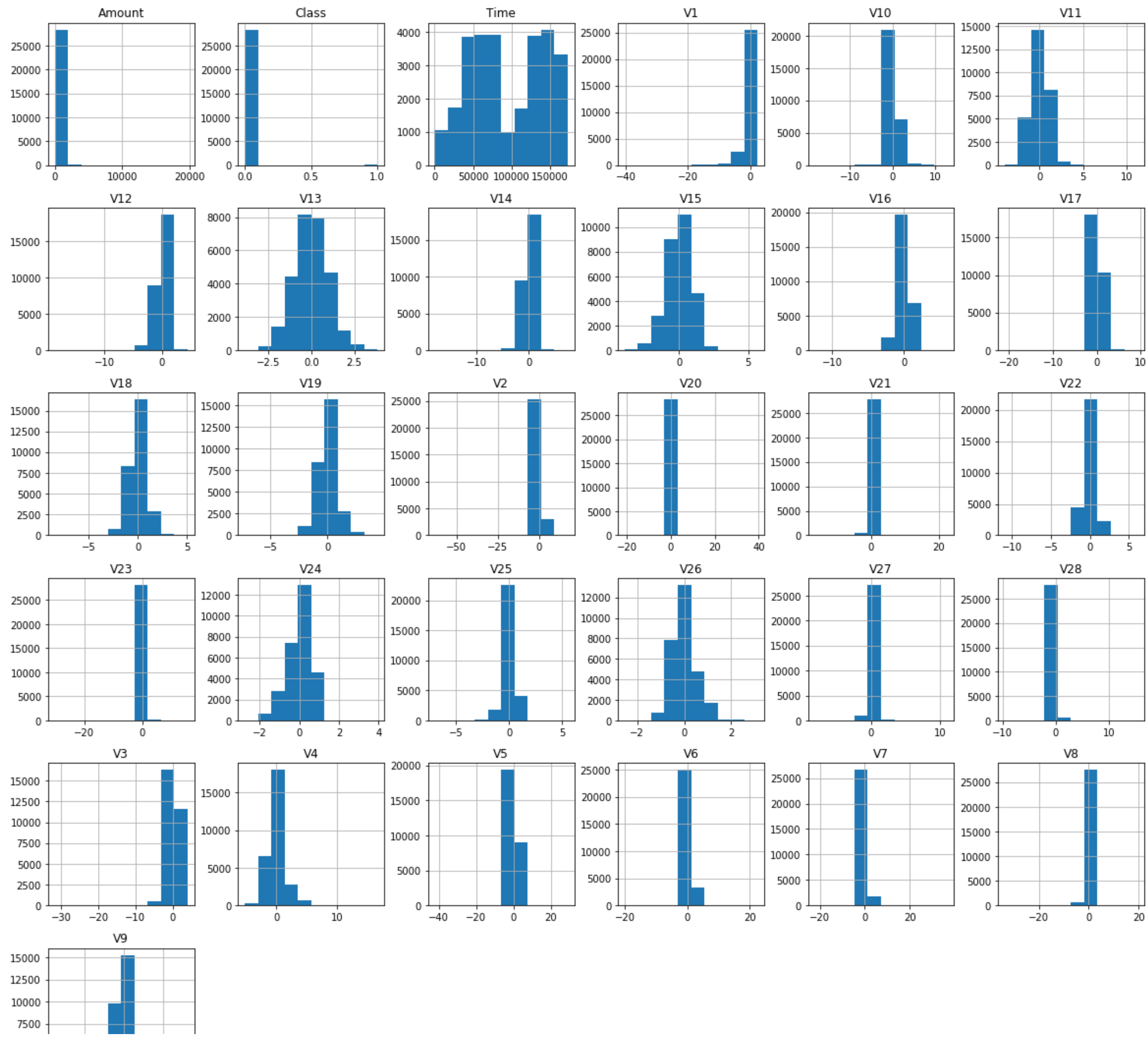
In [12]:

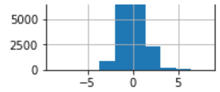
```python
# Print the shape of the data

data=data.sample(frac=0.1,random_state=1)
print(data.shape)
```

```
(28481, 31)
```

In [13]:

```python
# Plot histograms of each parameter

data.hist(figsize=(20,20))
plt.show()
```





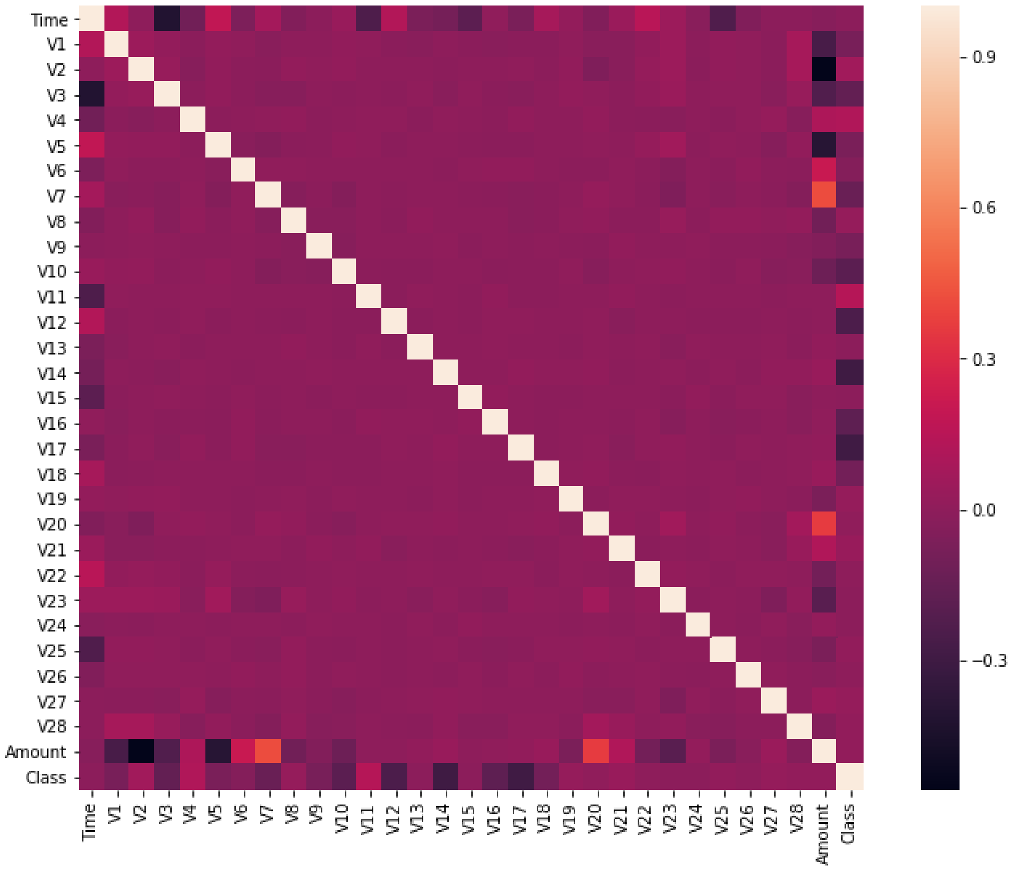
In [14]:

```
# Determine number of fraud cases in dataset

Fraud= data[data['Class']==1]
valid= data[data['Class']==0]


outliyar_fraction = len(Fraud)/float(len(valid))
print(outliyar_fraction)

print('Fraud cases:()',format(len(Fraud)))
print('Valid cases:()',format(len(valid)))
```

```
0.0017234102419808666
Fraud cases:() 49
Valid cases:() 28432
```

In [15]:

```
# Correlation matrix

corrat=data.corr()
fig=plt.figure(figsize=(13,9))


sns.heatmap(corrat,square=True)
plt.show()
```



In [16]:

```
# Get all the columns from the dataFrame

columns=data.columns.tolist()
```

```
# Filter the columns to remove data we do not want
columns=[c for c in columns if c not in['Class']]

# Store the variable we'll be predicting on
target='Class'
x=data[columns]
y=data[target]
# Print shapes
print(x.shape)
print(y.shape)
```

```
(28481, 30)
(28481,)
```

3. Unsupervised Outlier Detection

Now that we have processed our data, we can begin deploying our machine learning algorithms. We will use the following techniques:

Local Outlier Factor (LOF)

The anomaly score of each sample is called Local Outlier Factor. It measures the local deviation of density of a given sample with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood.

Isolation Forest Algorithm

The IsolationForest ‘isolates’ observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.

This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.

Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

In [17]:

```
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

#define random states
state=1

# define outlier detection tools to be compared
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(x), contamination=outliyar_fraction,random_
state=state),

    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=20,contamination=outliyar_fraction)}
```

In [18]:

```
# fit the data and tag outliers
n_outliers = len(Fraud)
for i, (clf_name, clf) in enumerate(classifiers.items()):
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(x)
        scores_pred = clf.negative_outlier_factor_
    else:
        clf.fit(x)
        scores_pred = clf.decision_function(x)
        y_pred = clf.predict(x)


    # 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).sum()

    # Run classification metrics
    print('{}: {}'.format(clf_name, n_errors))
    print(accuracy_score(y, y_pred))
    print(classification_report(y, y_pred))
   # plt.show()
```

```
C:\Users\ganesh chaurasiya\Desktop\python3.7\lib\site-packages\sklearn\ensemble\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\ganesh chaurasiya\Desktop\python3.7\lib\site-packages\sklearn\ensemble\iforest.py:415: DeprecationWarning: threshold_ attribute is deprecated in 0.20 and will be removed in 0.22.
  " be removed in 0.22.", DeprecationWarning)
```

```
Isolation Forest: 71
0.99750711000316
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.28      0.29      0.28        49

    accuracy                           1.00     28481
   macro avg       0.64      0.64      0.64     28481
weighted avg       1.00      1.00      1.00     28481

Local Outlier Factor: 97
0.9965942207085425
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.02      0.02      0.02        49

    accuracy                           1.00     28481
   macro avg       0.51      0.51      0.51     28481
weighted avg       1.00      1.00      1.00     28481
```

In [19]:

```
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
cc = pd.read_csv("creditcard.csv")
```

In [20]:

```
cc.columns
```

Out[20]:

```
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 [21]:

```
cc_train= cc.drop('Class', 1)
```

In [22]:

```
from sklearn.ensemble import IsolationForest
clf = IsolationForest(n_estimators=100, max_samples=200)
```

In [23]:

```
#Train the model with the data.
#clf.fit(cc_train)
```

In [24]:

```
# The Anomaly scores are calclated for each observation and stored in 'scores_pred'
#scores_pred = clf.decision_function(cc_train)

#verify the length of scores and number of obersvations.
#print(len(scores_pred))
print(len(cc))
```

```
284807
```

In [25]:

```
cc= cc.rename(columns={'Class': 'Category'})
```

In [26]:

```
# Based on (Liu and Ting, 2008), anomalous observation is scored close to 1
# and non anamolous observations are scored close to zero.
# I have written a simple loop that will count the number of observation that has score more than
0.5 and is actually anomalous.
counter =0
for n in range(len(cc)):
   # if (cc['Category'][n]== 1 and cc['scores'][n] >=0.5):
        counter= counter+1
print (counter)
```

```
284807
```

In [27]:

```
# For convinience, divide the dataframe cc based on two labels.
avg_count_0 = cc.loc[cc.Category==0]    #Data frame with normal observation
avg_count_1 = cc.loc[cc.Category==1]    #Data frame with anomalous observation
```

In [28]:

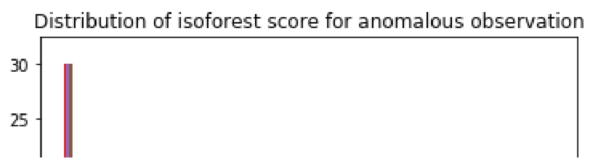
```

```

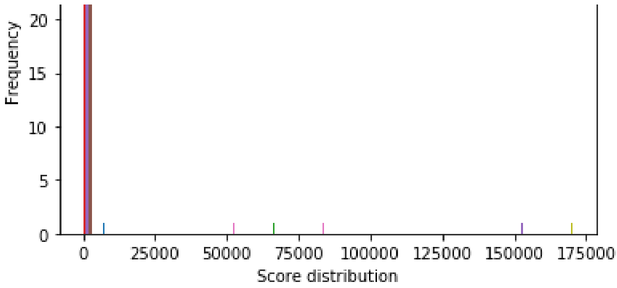
In [30]:

```
#Plot the combined distribution of the scores
%matplotlib inline
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
#%pylab.inline
normal = plt.hist(avg_count_1, 50,)
plt.xlabel('Score distribution')
plt.ylabel('Frequency')
plt.title("Distribution of isoforest score for anomalous observation")
#plt.show()
```

Out[30]:

```
Text(0.5, 1.0, 'Distribution of isoforest score for anomalous observation')
```




Frequency
20
15
10
5
0
0
25000
50000
75000
100000
125000
150000
175000
Score distribution