Creditcard_Frauddetection

January 1, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
import matplotlib.pyplot as plt
import scipy
from sklearn.metrics import accuracy_score,classification_report
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14 ,8
RANDOM_SEED = 42
LABELS = ['Normal','Fraud']
```

[2]:
```
data = pd.read_csv('creditcard.csv',sep=',')
data.head()
```

[2]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
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       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

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

```
[5 rows x 31 columns]
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

[4]:
```
data.describe()
```

[4]:

```
                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
```

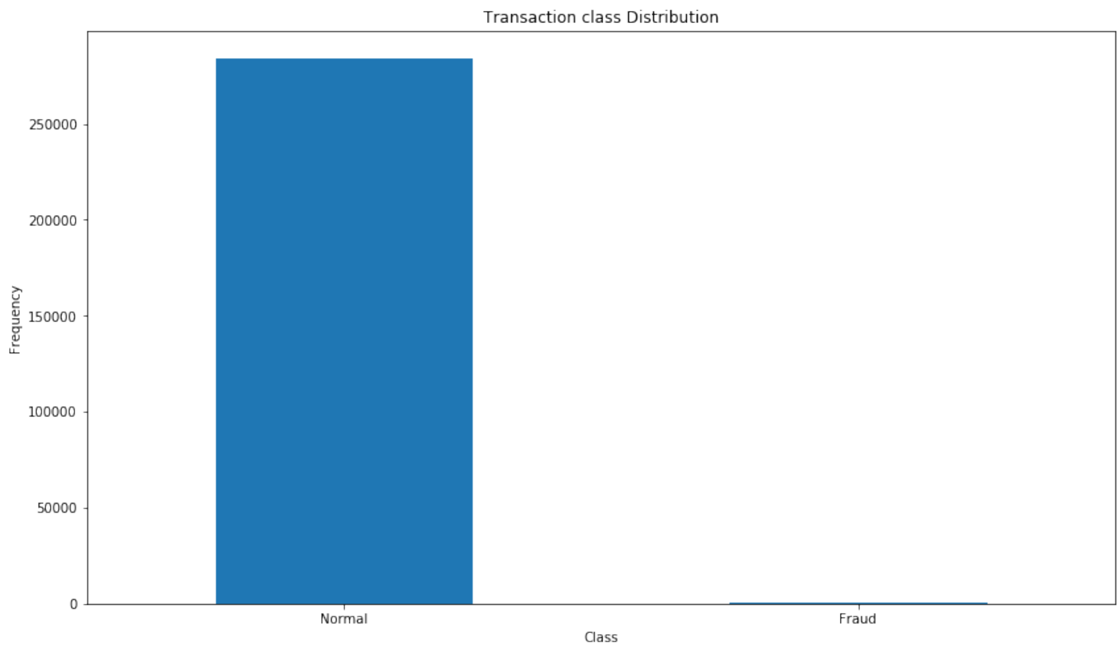
```
75%          0.000000
max          1.000000

[8 rows x 31 columns]
```

[5]:
```
data.isnull().values.any()
```

[5]:
```
False
```

[6]:
```
count_classes = pd.value_counts(data['Class'],sort = True)
count_classes.plot(kind = 'bar',rot=0)
plt.title('Transaction class Distribution')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.xticks(range(2),LABELS)
```

[6]:
```
([<matplotlib.axis.XTick at 0x22e2d82b108>,
  <matplotlib.axis.XTick at 0x22e2d81c708>],
 <a list of 2 Text xticklabel objects>)
```



[7]:
```
Fraud = data[data['Class']==1]
Normal =data[data['Class']==0]
```

[8]:
```
print(Fraud.shape,Normal.shape)
```

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

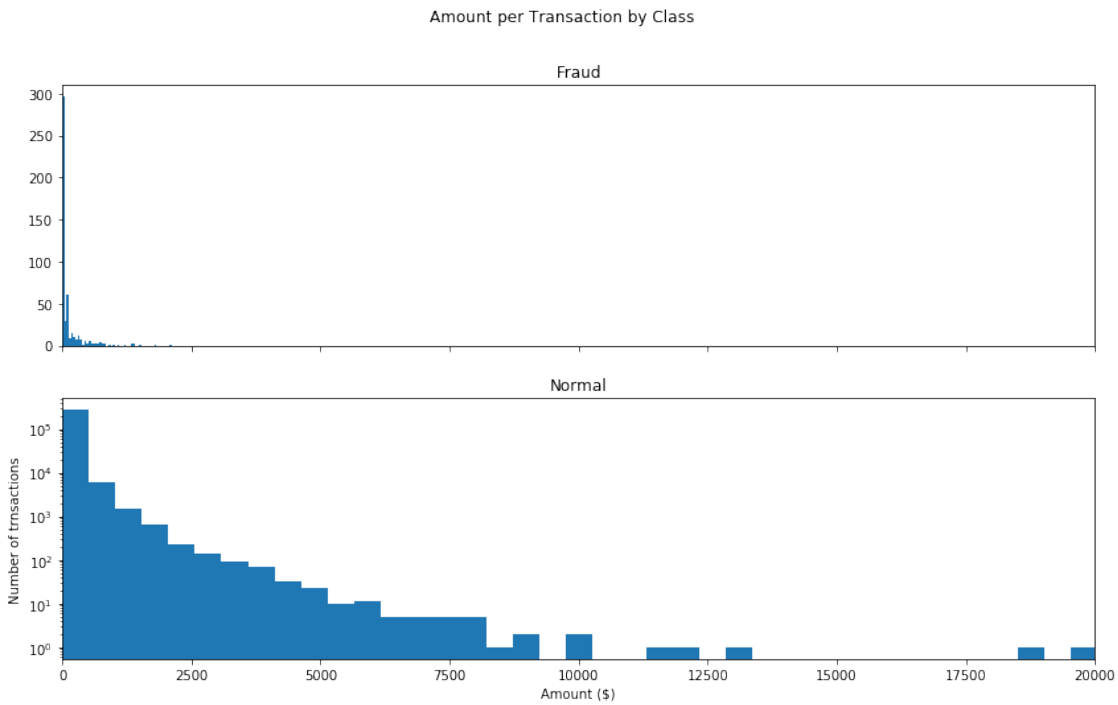
[9]:
```
Fraud.Amount.describe()
```

[9]:
```
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
```

[10]:
```
Normal.Amount.describe()
```

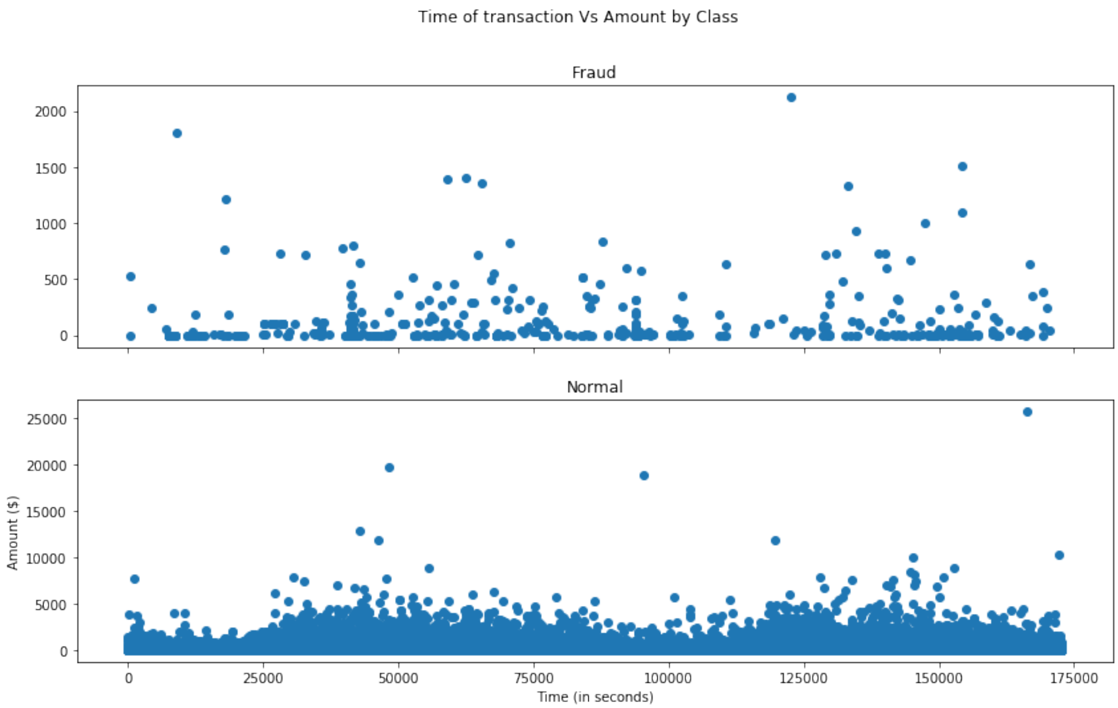
[10]:
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

[11]:
```
## We Will check Do fraudulent transactions occur more often during certain␣
↪time␣frame . Let us find out with a visual representation.

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



[12]:
```
f, (ax1,ax2) = plt.subplots(2,1, sharex=True)
f.suptitle('Time of transaction Vs Amount by Class')
plt.xlabel("Time (in seconds)")
plt.ylabel('Amount ($)')
ax1.set_title("Fraud")
ax2.set_title("Normal")
ax1.scatter(Fraud.Time,Fraud.Amount)
ax2.scatter(Normal.Time,Normal.Amount)
plt.show
```

[12]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[13]:
```
##Lets take some sample of data
data1 = data.sample(frac =0.1 , random_state = 1)
data1.shape
```

[13]: (28481, 31)

[14]:
```
data.shape
```

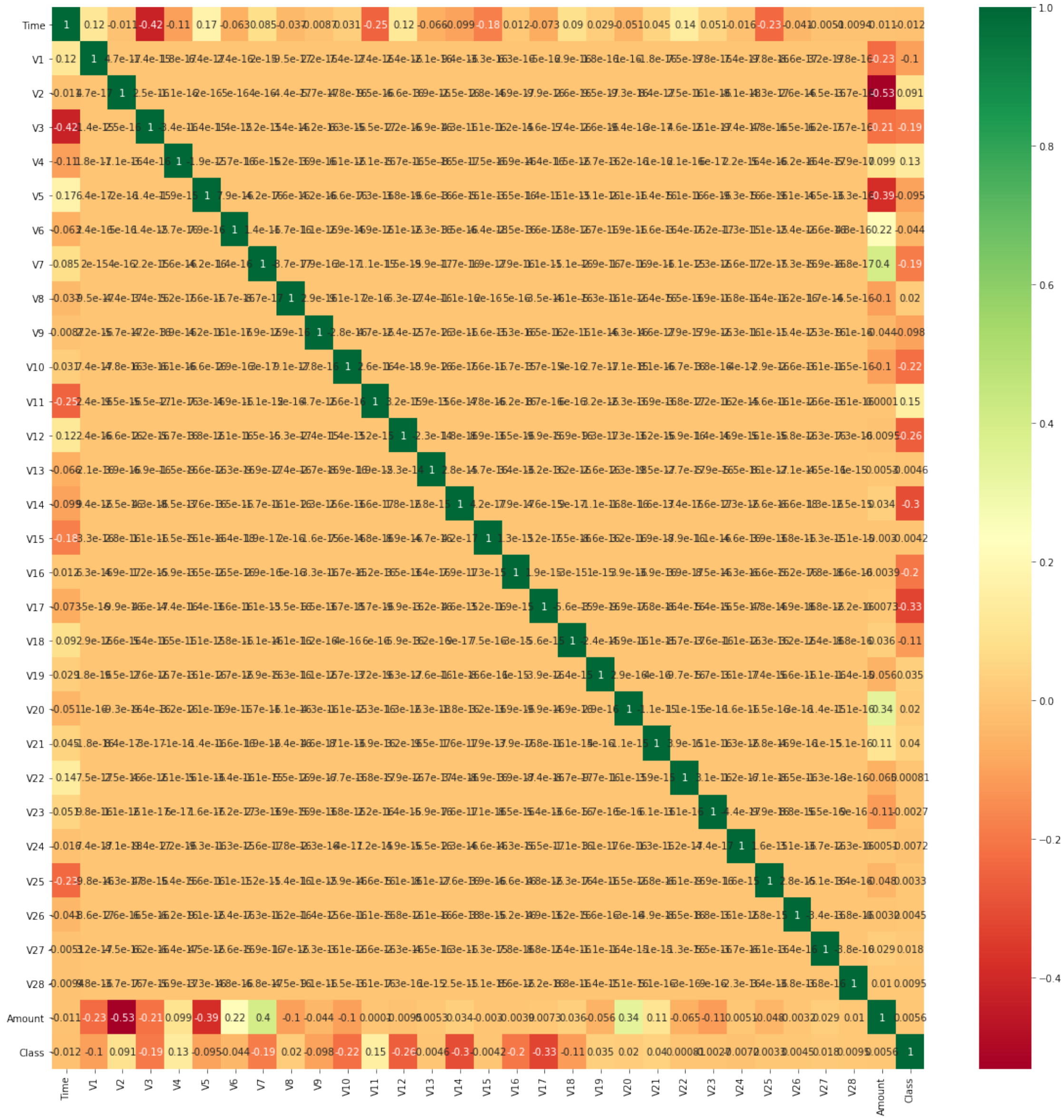
[14]: (284807, 31)

[15]:
```
##Determine the number of fraud and valid transactions in the dataset


Fraud = data1[data1['Class']==1]
Valid= data1[data1['Class']==0]
outlier_fraction=len(Fraud)/float(len(Valid))
```

[16]:
```
print(outlier_fraction)
print("Fraud cases:{}".format(len(Fraud)))
print("Valid cases:{}".format(len(Valid)))
```

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

[17]:
```
##Correlation
import seaborn as sns
corrmat = data1.corr()
top_corr_features =corrmat.index
plt.figure(figsize=(20,20))
g=sns.heatmap(data[top_corr_features].corr(),annot= True,cmap='RdYlGn')
```



[18]:
```
## Create independent and Dependent Features
columns= data1.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ['Class']]
```

```
target='Class'
state= np.random.RandomState(42)
X=data1[columns]
Y=data1[target]
X_outliers=state.uniform(low=0,high=1,size=(X.shape[0],X.shape[1]))
```

[19]:
```
print(X.shape)
print(Y.shape)
```

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

[20]:
```
classifiers = {
"Isolation Forest":IsolationForest(n_estimators=100,␣
↪max_samples=len(X),contamination=outlier_fraction,random_state=state,␣
↪verbose=0),
"Local Outlier Factor":LocalOutlierFactor(n_neighbors=20,␣
↪algorithm='auto',leaf_size=30, metric='minkowski',p=2, metric_params=None,
contamination=outlier_fraction),
"Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.
↪05,max_iter=-1)
}
```

[21]:
```
type(classifiers)
```

[21]:
```
dict
```

[22]:
```
n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    elif clf_name == "Support Vector Machine":
        clf.fit(X)
        y_pred = clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud␣
↪transactions
    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))
```

```python
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
    print("Classification Report :")
    print(classification_report(Y,y_pred))
```

```
Isolation Forest: 73
Accuracy Score :
0.9974368877497279
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.26      0.27      0.26        49

    accuracy                           1.00     28481
   macro avg       0.63      0.63      0.63     28481
weighted avg       1.00      1.00      1.00     28481

Local Outlier Factor: 97
Accuracy Score :
0.9965942207085425
Classification Report :
              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

Support Vector Machine: 8516
Accuracy Score :
0.7009936448860644
Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.70      0.82     28432
           1       0.00      0.37      0.00        49

    accuracy                           0.70     28481
   macro avg       0.50      0.53      0.41     28481
weighted avg       1.00      0.70      0.82     28481
```

Observations : • Isolation Forest detected 73 errors versus Local Outlier Factor detecting 97 errors vs. SVM detecting 8516 errors • Isolation Forest has a 99.74% more accurate than LOF of 99.65% and SVM of 70.09 • When comparing error precision & recall for 3 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 27 % versus LOF detection rate of just 2 % and SVM of 0%. • So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%. • We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

[ ]:

```

```