Title : Credit Card Fraud Detection System

ABSTRACT

It is vital that credit card companies are able to identify fraudulent credit card transactions so that customers are not charged for items that they did not purchase. Such problems can be tackled with Data Science and its importance, along with Machine Learning, cannot be overstated. This project intends to illustrate the modelling of a data set using machine learning with Credit Card Fraud Detection. The Credit Card Fraud Detection Problem includes modelling past credit card transactions with the data of the ones that turned out to be fraud. This model is then used to recognize whether a new transaction is fraudulent or not. Our objective here is to detect the fraudulent transactions while minimizing the incorrect fraud classifications. Credit Card Fraud Detection is a typical sample of classification. In this process, we have focused on analysing and pre-processing data sets as well as the deployment of multiple anomaly detection algorithms such as Local Outlier Factor and Isolation Forest algorithm on the PCA transformed Credit Card Transaction data.

Keywords— Credit card fraud, applications of machine learning, data science, isolation forest algorithm, local outlier factor, automated fraud detection.

1. INTRODUCTION

In Today’s world high dependency on internet technology has increased credit `card transactions but credit card frauds are also accelerated as online and offline transaction have increased. As credit card transactions have become a widespread mode of payment, focus has been given to recent computational methodologies to handle the credit card fraud problem. There are many fraud detection solutions and software which prevent frauds in businesses such as credit card, retail, e-commerce, insurance, and industries. Credit card fraud detection is a very popular but also a difficult problem to solve.

Firstly, due to issue of having only a limited amount of data, credit card makes it challenging to match a pattern for dataset. Secondly, there can be many entries in dataset with transactions of fraudsters which also will fit a pattern of legitimate behaviour. Also, the problem has many constraints. Avoiding these all problems, In this project model can find Fraud Transactions using Python, or Machine Learning Algorithms, and analyses fraud data using Plotted graph or tables.

2. ARCHITECTURAL DESIGN AND MODELING

This data is fit into a model and the following outlier detection modules are applied on it:

• Local Outlier Factor

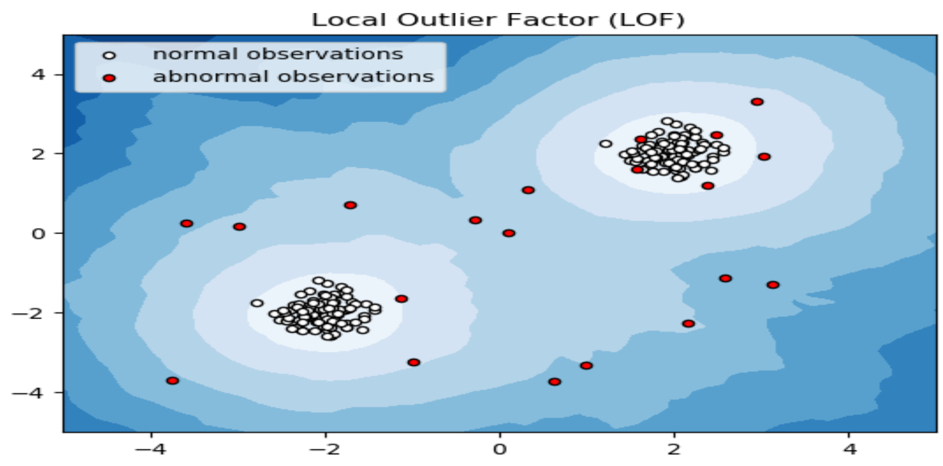
• Isolation Forest Algorithm

These algorithms are a part of sklearn. The ensemble module in the sklearn package includes ensemble-based methods and functions for the classification, regression and outlier detection. This free and open-source Python library is built using NumPy, SciPy and matplotlib modules which provides a lot of simple and efficient tools which can be used for data analysis and machine learning.

A. Local Outlier Factor

It is an Unsupervised Outlier Detection algorithm. 'Local Outlier Factor' refers to the anomaly score of each sample. It measures the local deviation of the sample data with respect to its neighbours. More precisely, locality is given by k-nearest neighbours, whose distance is used to estimate the local data.

Example Image of Local Outlier Factor algorithm:



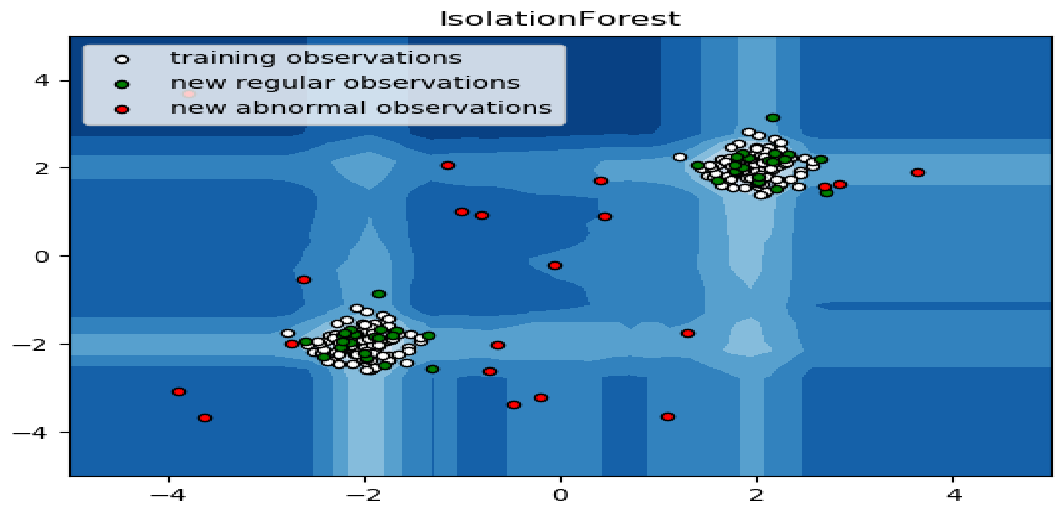
By comparing the local values of a sample to that of its neighbours, one can identify samples that are substantially lower than their neighbours. These values are quite amanous and they are considered as outliers.
As the dataset is very large, we used only a fraction of it in out tests to reduce processing times.

Outlier are patterns in data that do not conform to the expected behaviour. The main motive of the algorithm is to find such patterns in the data and consider them as anomalies. Then in our project , these anomalies may belong to fraudulent cases. Hence we separate out such cases to further check these cases and send it to the next algorithm called Isolation Forest Algorithm

B. The Isolation Forest

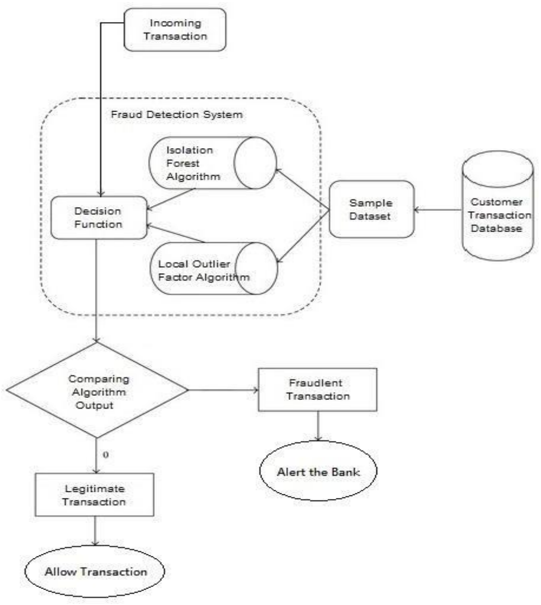
Isolation Forest explicitly identifies anomalies instead of profiling normal data points. So here we are checking the transaction entities such as transaction duration, last transaction time and location, no of credential retries etc. In This Algorithm the found Anomalies are Isolated (Separated) as Successful Transactions or suspicious Transactions. It isolates observations by arbitrarily selecting a feature and then randomly selecting a split value between the maximum and minimum values of the designated feature.The analysis done on these transactions are shown further in graphical format.

Example Image of Isolation Forest algorithm :



.

FLOWCHART



3. IMPLEMENTATION AND EVALUATION

Fraud act as the unlawful or criminal deception intended to result in financial or personal benefit. It is a deliberate act that is against the law, rule or policy with an aim to attain unauthorized financial benefit. Numerous literatures pertaining to anomaly or fraud detection in this domain have been published already and are available for public usage.

Firstly, due to issue of having only a limited amount of data, credit card makes it challenging to match a pattern for dataset. Secondly, there can be many entries in dataset with transactions of fraudsters which also will fit a pattern of legitimate behaviour. Also, the problem has many constraints. Avoiding these all problems, In this project model can find Fraud Transactions using Python, or Machine Learning Algorithms, and analyses fraud data using Plotted graph or tables.

The data is taken from kaggle.com which provides dataset for different attributes and developing purpose. The data set is in a huge amount so to determine the fraudlent cases some factors are needed to be taken into consideration.So factors like as transaction duration, last transaction time and location, no of credential retries ,number of times the pin is entered etc are taken into consideration to determine whether a particular case is fraudlent or the case is non-fraudlent.

The main goal is to determine the number of fraudlent cases so for that we have implemented the local outlier factor and isolation forest algorithm.In our project we are going through the Credit card transaction cases. However, after inspecting the data it turned out that some Credit card transactions were representing abnormal behaviour — they were outliers.Outlier are patterns in data that do not conform to the expected behaviour. The main motive of the algorithm is to find such patterns in the data and consider them as anomalies. In Isolation Forest Algorithm the found Anomalies are Isolated (Separated) as Successful Transactions or suspicious Transactions. It provides accuracy finding out the fraudulent transactions and minimizing the number of false alerts.

We observed that the module can be used for Banking & Business Sectors for fraud detections. The module is developed in low cost. In this system it detects Anomalies using Machine learning Algorithms. The solution is that it is implementable and very useful for the user.

4. DATA DESCRIPTION

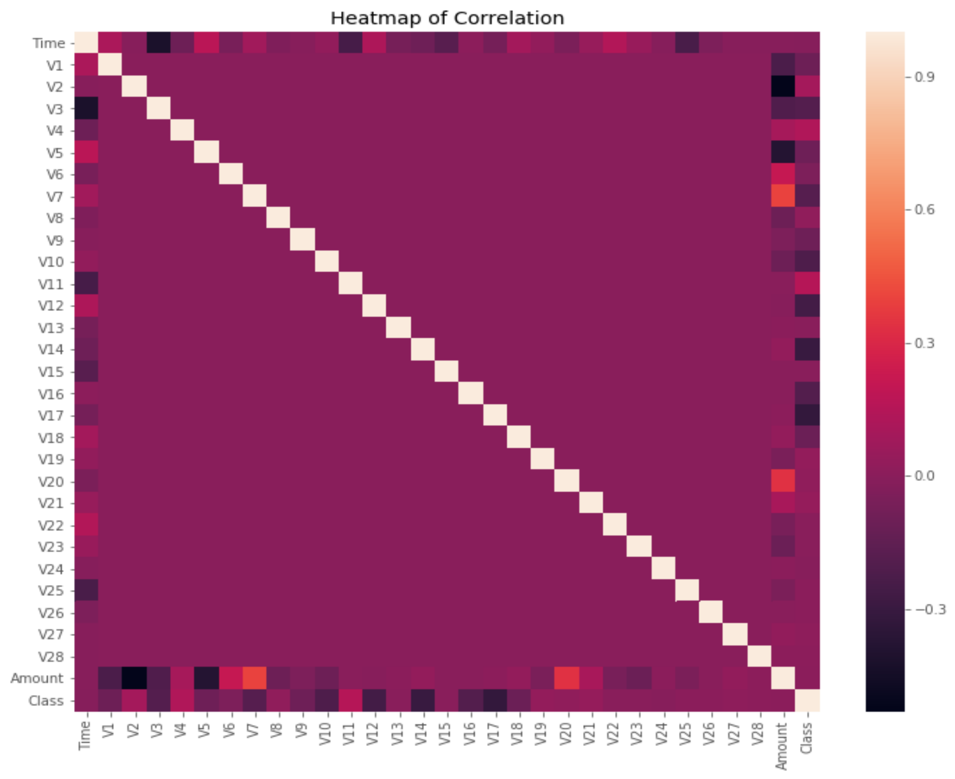
Dataset Name-credit.csv

The data is been taken from kaggle.com which provides dataset for different attributes and developing purpose.

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

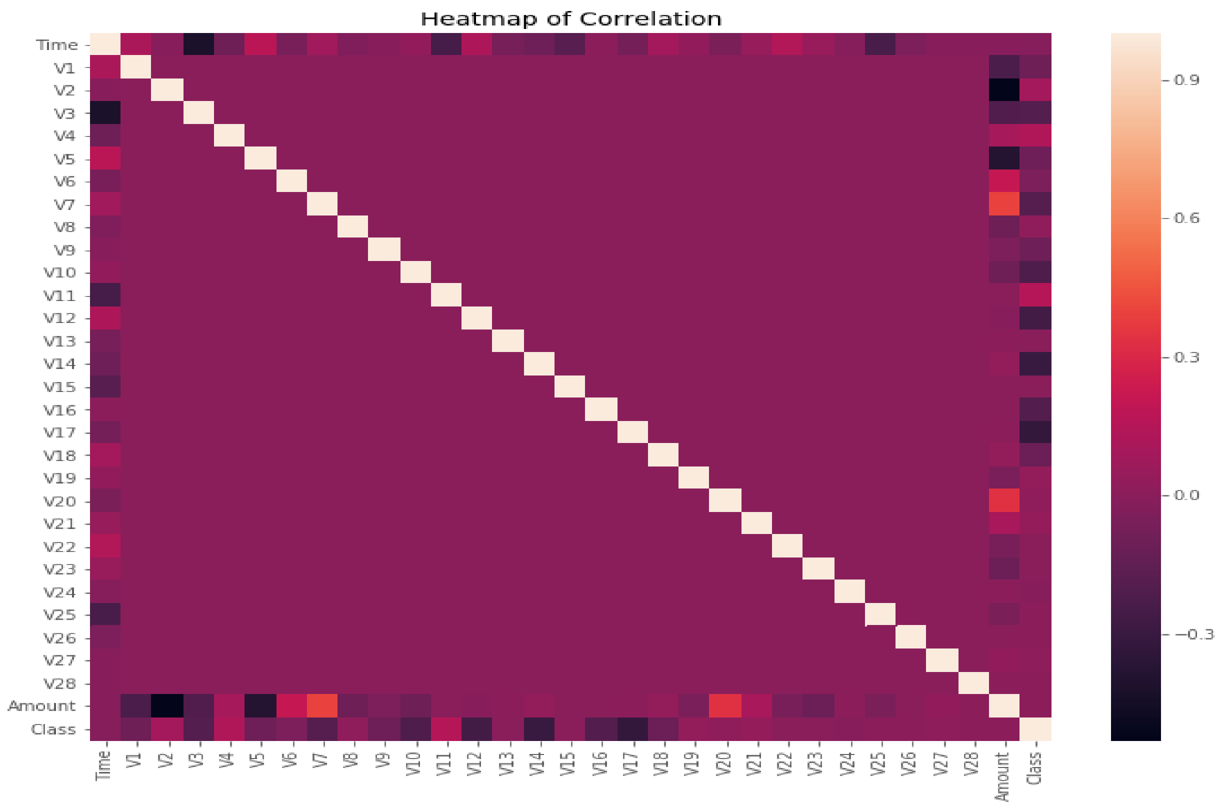
5. FEATURE ENGINEERING

- **Correlation Analysis Visualizations**

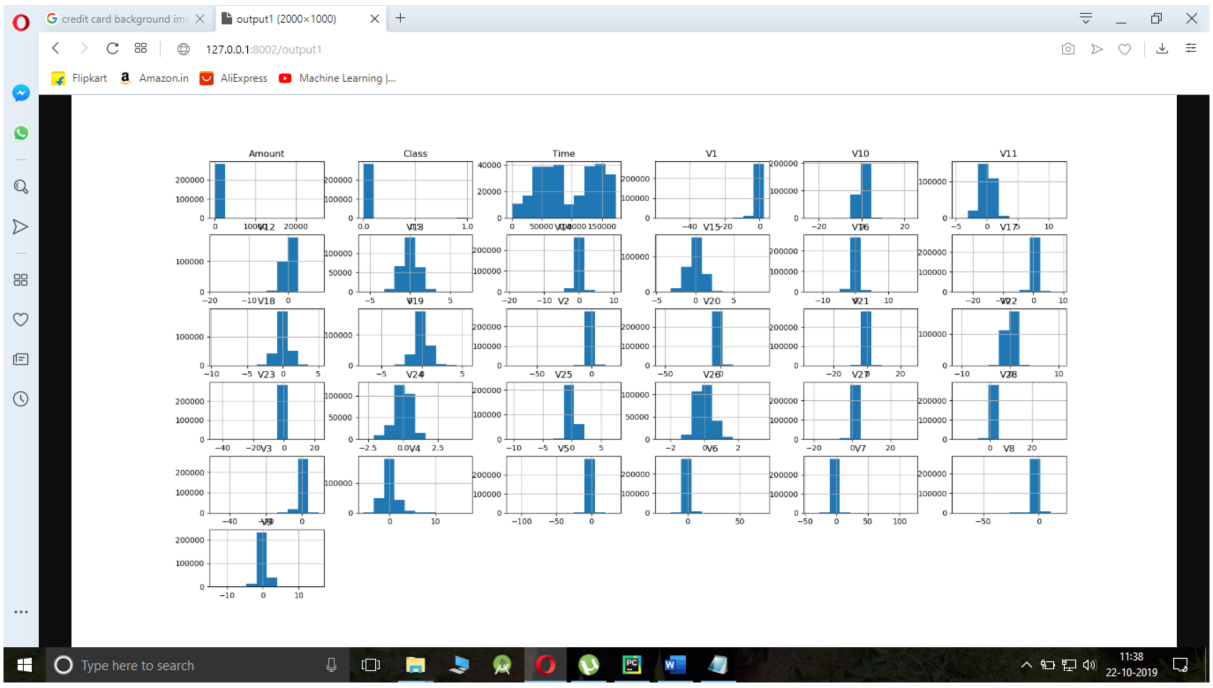


First of all, we obtained our dataset from Kaggle, a data analysis website which provides datasets.Inside this dataset, there are 31 columns out of which 28 are named as v1-v28 to protect sensitive data.The other columns represent Time, Amount and Class. Time shows the time gap between the first transaction and the following one. Amount is the amount of money transacted. Class 0 represents a valid transaction and 1 represents a fraudulent one.

After this analysis, we plot a heatmap to get a coloured representation of the data and to study the correlation between out predicting variables and the class variable. This heatmap is shown below:



After checking this dataset, we plot a histogram for every column. This is done to get a graphical representation of the dataset which can be used to verify that there are no missing any values in the dataset. This is done to ensure that we don't require any missing value imputation and the machine learning algorithms can process the dataset smoothly.



Histogram

6. IMPLEMENTATION OF MODELS

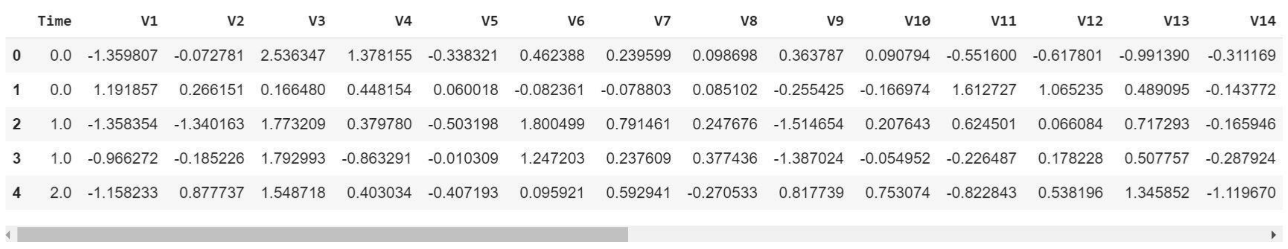
Importing all the necessary Libraries

```
# import the necessary packages
import pandas as pd
import sys
import scipy
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

#importing csv using pandas
```

Loading the Data

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

Understanding the Data



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

data.head()

Entries in DataSet

print(data.shape)

printing useful information

print(data.describe())

Describing the Data

```
# Print the shape of the data
data=data.sample(frac = 0.1,random_state = 1)
print(data.shape)

print(data.describe())
```

Output

```
(284807, 31)
                  Time            V1  ...         Amount          Class

count  284807.000000  2.848070e+05  ...  284807.000000  284807.000000

mean    94813.859575  3.919560e-15  ...      88.349619       0.001727

std     47488.145955  1.958696e+00  ...     250.120109       0.041527

min         0.000000 -5.640751e+01  ...       0.000000       0.000000

25%     54201.500000 -9.203734e-01  ...       5.600000       0.000000

50%     84692.000000  1.810880e-02  ...      22.000000       0.000000

75%    139320.500000  1.315642e+00  ...      77.165000       0.000000

max    172792.000000  2.454930e+00  ...   25691.160000       1.000000
```

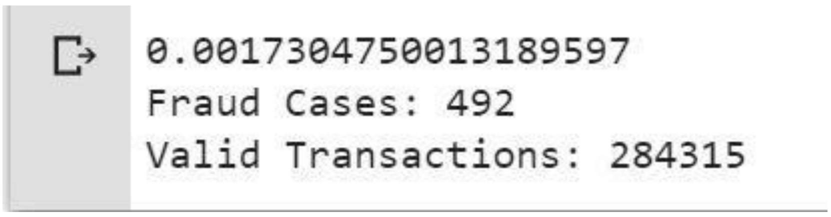
```
[8 rows x 31 columns]
```

Plotting Histogram of each parameter

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

Imbalance in the data

```
# 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
```

Print the amount details for Fraudulent Transaction

```
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

```
Amount details of the fraudulent transaction
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
```

Print the amount details for Normal Transaction

print(“details of valid transaction”)

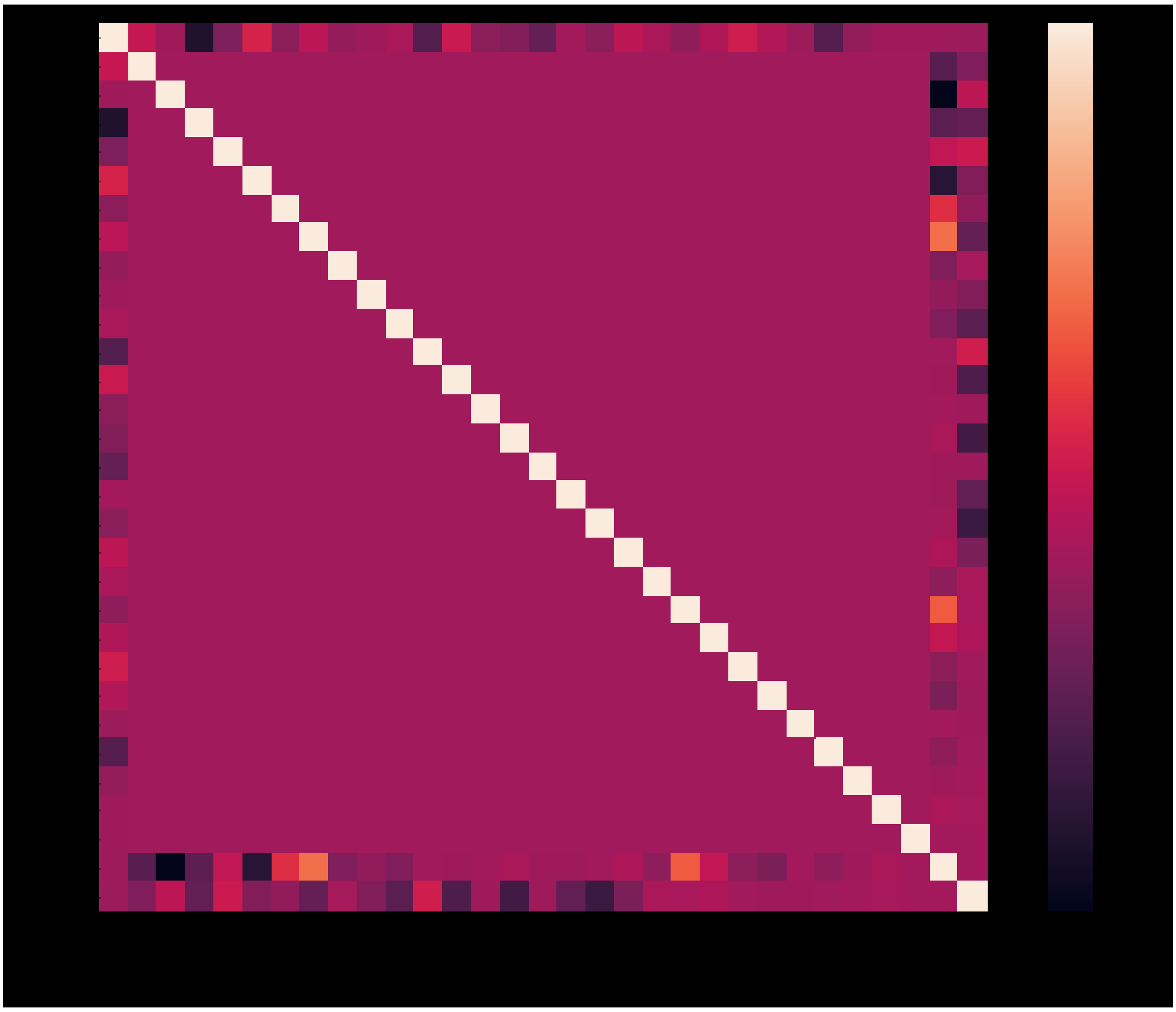
valid.Amount.describe()

```
Amount details of valid transaction
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
```

Plotting the Correlation Matrix

#Correlation matrix

corrmat = data.corr()
#pyplot fig
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.

Separating the X and the Y values

```
#get all columns from 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)
```

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

Training and Testing Data Bifurcation

```
# Using Skicit-learn to split data into training and testing sets

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

# define random states
state = 1
```

For Isolation Forest and Local Outlier Factor

```
# define outlier detection tools to be compared
classifiers = {
    "Isolation Forest": IsolationForest(max_samples=len(X),
                                       contamination=outlier_fraction,
                                       random_state=state),
    "Local Outlier Factor": LocalOutlierFactor(
        n_neighbors=20,
        contamination=outlier_fraction)}

#Fit the model
plt.figure(figsize=(9, 7))
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_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()
```

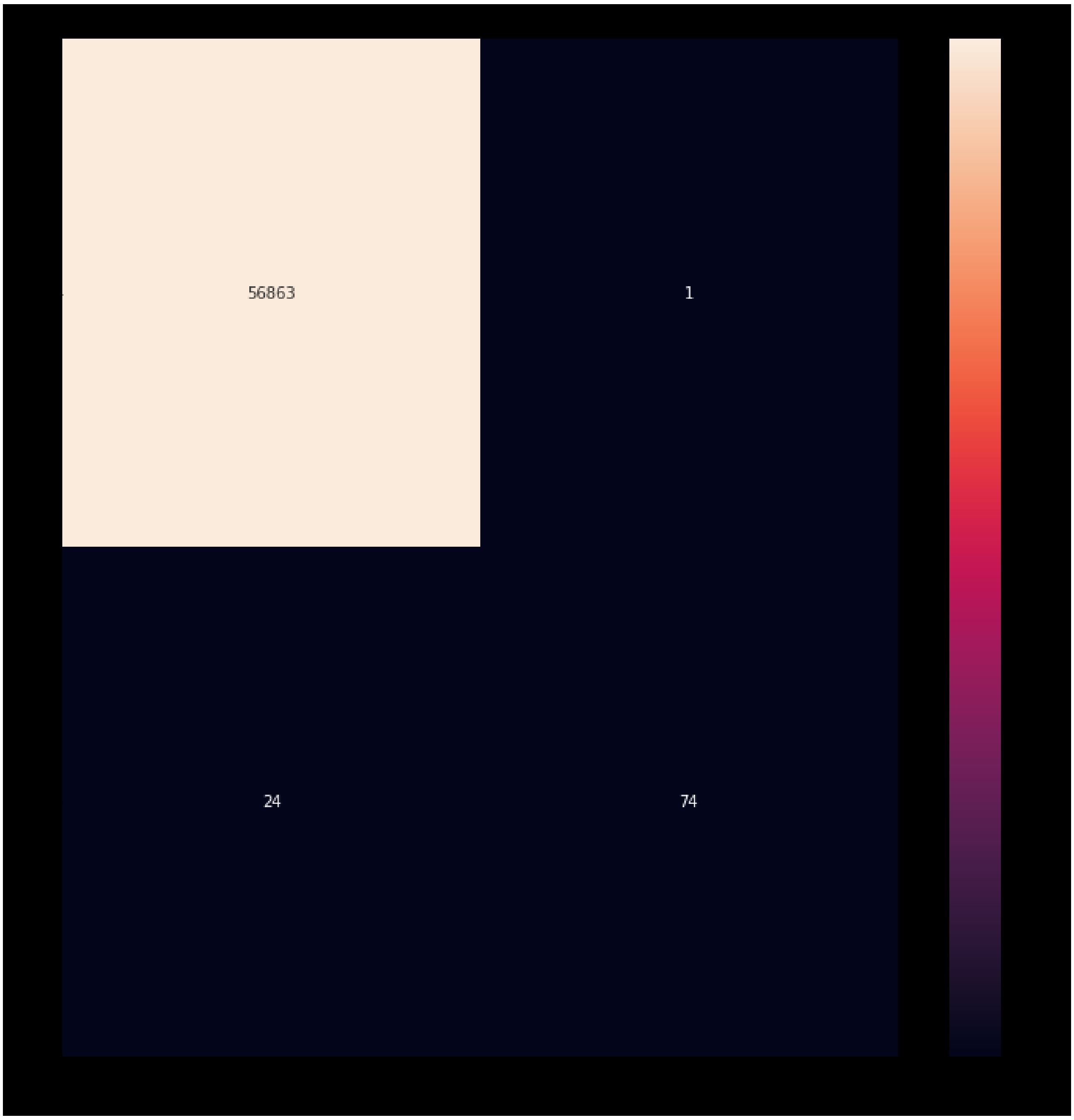
classification metrics

```
    print('{}: {}'.format(clf_name, n_errors))
    print(accuracy_score(Y, y_pred))
    print(classification_report(Y, y_pred))
```

Visulalizing the Confusion Matrix

```
# printing the confusion matrix

LABELS = ['Normal', 'Fraud']

conf_matrix = confusion_matrix(yTest, yPred)

plt.figure(figsize =(12, 12))

sns.heatmap(conf_matrix, xticklabels = LABELS,

                        yticklabels = LABELS, annot = True, fmt ="d");

plt.title("Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()
```

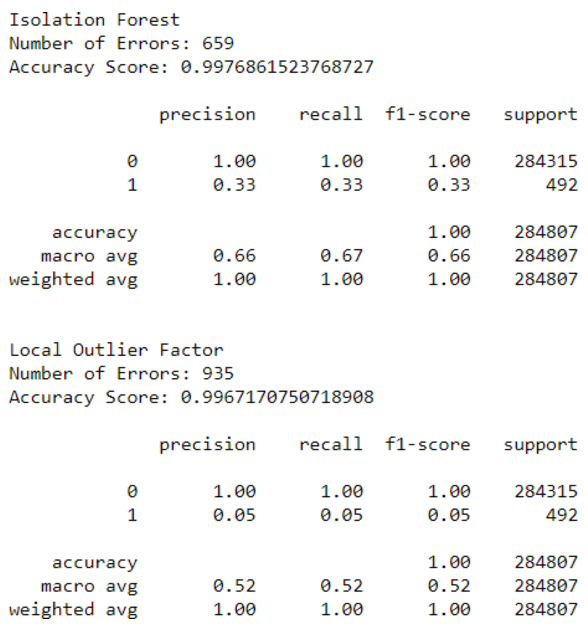
Output :



7.RESULTS

The code prints out the number of false positives it detected and compares it with the actual values. This is used to calculate the accuracy score and precision of the algorithms. The fraction of data we used for faster testing is 10% of the entire dataset. The complete dataset is also used at the end and both the results are printed.
These results along with the classification report for each algorithm is given in the output as follows, where class 0 means the transaction was determined to be valid and 1 means it was determined as a fraud transaction.

This result matched against the class values to check for false positives.



```
Isolation Forest
Number of Errors: 659
Accuracy Score: 0.9976861523768727

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.33      0.33      0.33       492

    accuracy                           1.00    284807
   macro avg       0.66      0.67      0.66    284807
weighted avg       1.00      1.00      1.00    284807

Local Outlier Factor
Number of Errors: 935
Accuracy Score: 0.9967170750718908

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    284315
           1       0.05      0.05      0.05       492

    accuracy                           1.00    284807
   macro avg       0.52      0.52      0.52    284807
weighted avg       1.00      1.00      1.00    284807
```

8.CONCLUSION

Credit card fraud is without a doubt an act of criminal dishonesty. This Projet has listed out the most common methods of fraud along with their detection methods and reviewed recent findings in this field. This Report has also explained in detail, how machine learning can be applied to get better results in fraud detection along with the algorithm, pseudocode, explanation its implementation and experimentation results.
While the algorithm does reach over 99.6% accuracy, its precision remains only at 28% when a tenth of the data set is taken into consideration. However, when the entire dataset is fed into the algorithm, the precision rises to 33%. This high percentage of accuracy is to be expected due to the huge imbalance between the number of valid and number of genuine transactions if this project were to be used on a commercial scale. Being based on machine learning algorithms, the program will only increase its efficiency over time as more data is put into it.

9. REFERENCES

1) A"Credit Card Fraud Detection | Kaggle."
https://www.kaggle.com › mlg-ulb › creditcardfraud".

2) Python Libraries Used:

Sklearn:http//scikit-learn.org/

Pandas: http//pandas.pydata.org/

Numpy: http//www.numpy.org/

Matplotlib: http//matplotlib.org/