Nelson Lefebvre
NUID : 03137910

Course Project AI CSCI 8110: Advanced Topics Artificial Intelligence

Introduction :

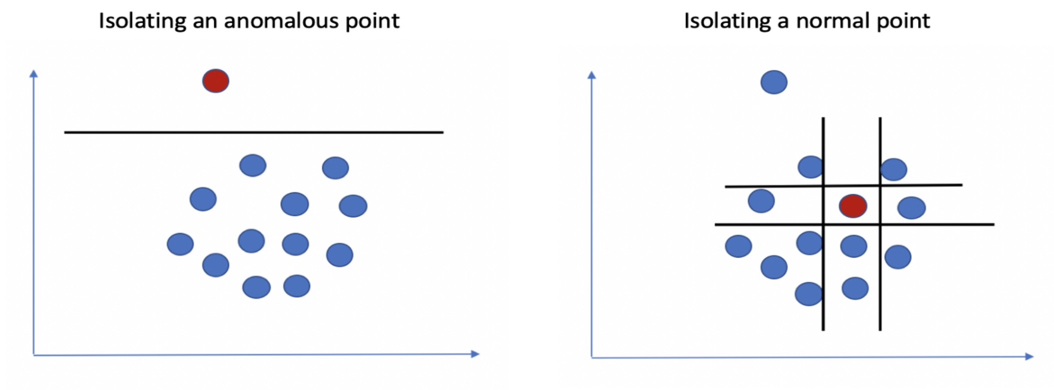
For my final project I chose to focus on the subject of anomaly detection because being in data-sciences as well as artificial intelligence I think it's an interesting subject.

Related work :

The goal of the approach is to find an outlier with a data point that is significantly different from other data points in a given data set. Large real-world data sets can have very complex patterns that are difficult to detect by simply observing the data. For this reason, the study of anomaly detection is an extremely important application of machine learning. The areas of possibility are immense whether in medicine (ecg, brain tumor..), banking, trading etc.. This algorithm is strong to analyze datasets of very large volume (many variables).

Methodology:

In the field of error detection we use algorithms with unsupervised matching which consists in detecting in a dataset the samples whose X characteristics are very far from those of the other samples. For this there are several ways to perform this algorithm. We can calculate the mean and the standard deviation of our data to determine a probability density function, and use this function to calculate the probability of existence of a given sample. When this probability is below the only given one then the sample is considered as abnormal.



From :
https://towardsdatascience.com/how-to-perform-anomaly-detection-with-the-isolation-forest-algorithm-e8c8372520bc

But for my part I'm going to focus on one algorithm in particular during my writing, "isolation forest". But what is it? In our dataset we will make a series of random splits and we will count the number of splits we have to make to isolate our samples. The smaller the number of splits the higher the chance that the sample is an anomaly.

For this I will use two libraries sklearn and keras to demonstrate how it works. We need to define the percentage of data we want to filter (contamination rate). Example Isolation(contamination = 0.02).

fit(X) -> then train the model

predict(X) -> (+1 normal, -1 abnormal)

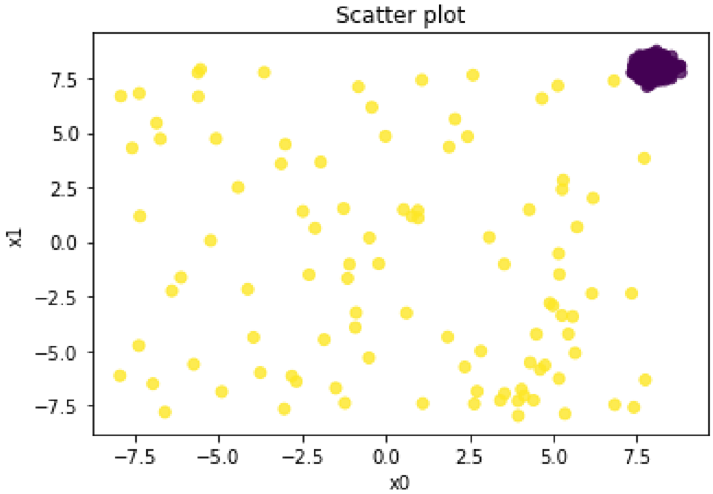
Experiment :

In this experimentation i used the database : creditcard.csv (available in the source)
But also the sklearn dataset to use number pictures.

In the first time I will explain how the algorithm is working and in a second time I will experiment with real world data.

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pyod.utils.data import generate_data
from pyod.utils.utility import standardizer
from pyod.models.iforest import IForest
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
n_train = 1000       # number of training points
n_test = 1000        # number of testing points
n_features = 10      # number of features
X_train, X_test, y_train, y_test = generate_data(
    n_train=n_train,
    n_test=n_test,
    n_features= n_features,
    random_state=1998)

X_train_pd = pd.DataFrame(X_train)
X_train_pd.head()

# Let's plot the graph to show the isolation
plt.scatter(X_train_pd[0], X_train_pd[1], c=y_train, alpha=0.8)
plt.title('Scatter plot')
plt.xlabel('x0')
plt.ylabel('x1')
plt.show()

X_train_pd.head()
```



Out[ ]:

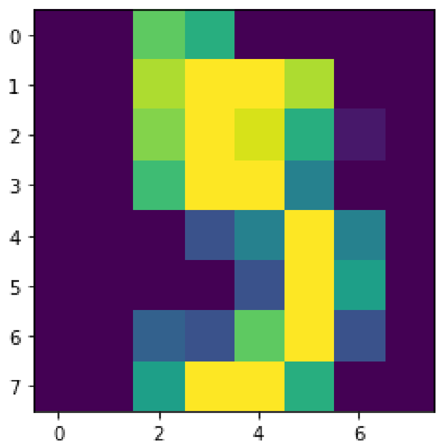
	0	1	2	3	4	5	6	7	8	9
0	8.074749	8.204974	7.805534	7.549014	8.198251	7.726323	7.883649	8.132128	7.962474	7.776526
1	7.415854	8.107022	8.132627	7.868063	7.746653	8.152237	7.700637	7.585423	8.133098	8.013190
2	7.936377	7.564359	8.422605	8.200313	8.559978	7.878968	7.941599	7.773176	8.069690	8.022218
3	7.761050	8.118319	7.818403	8.076272	7.869649	7.976227	7.736197	7.965967	8.239789	8.358154
4	7.389331	8.063779	7.529773	8.258147	7.935390	8.406474	7.730541	8.050230	8.162081	8.087700

In [ ]:

In [ ]:

```
#Let's try in a concrete case to explain how is it working.
from sklearn.datasets import load_digits
from sklearn.ensemble import IsolationForest

digits = load_digits()
images = digits.images
X = digits.data
y = digits.target

print("Shape of X", X.shape)

#For this exemple let's try to print() the number 5
plt.imshow(images[5])

#Then let's predict the image with the number 5 and let's plot it
model = IsolationForest(random_state=0, contamination=0.05)
model.fit(X)

#And let's show that the array is composed of 1 (normal) and -1 (not normal) as we discus
ses before
print(model.predict(X))
outliers = model.predict(X) == -1
print(outliers)
```

```
Shape of X (1797, 64)
[1 1 1 ... 1 1 1]
[False False False ... False False False]
```



In [ ]:

```
#Lets inject outliers in image
images[outliers]
```

Out[ ]:

```
array([[[ 0.,  6., 13., ...,  8.,  1.,  0.],
        [ 0.,  8., 16., ..., 16.,  6.,  0.],
        [ 0.,  6., 16., ...,  4.,  0.,  0.],
        ...,
        [ 0.,  0.,  0., ...,  9.,  0.,  0.],
        [ 0.,  1.,  8., ...,  3.,  0.,  0.],
        [ 0.,  4., 16., ...,  0.,  0.,  0.]],

       [[ 0.,  0.,  0., ..., 15.,  4.,  0.],
        [ 0.,  0.,  0., ..., 16., 12.,  0.],
        [ 0.,  0.,  0., ..., 16., 12.,  0.],
        ...,
        [ 0.,  9., 16., ...,  1.,  0.,  0.],
        [ 0.,  3.,  8., ...,  9.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16., 12.,  0.]],

       [[ 0.,  0.,  0., ..., 15.,  8.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        [ 0.,  0.,  3., ..., 16., 10.,  0.],
        ...,
        [ 0.,  0.,  0., ..., 16.,  1.,  0.],
```

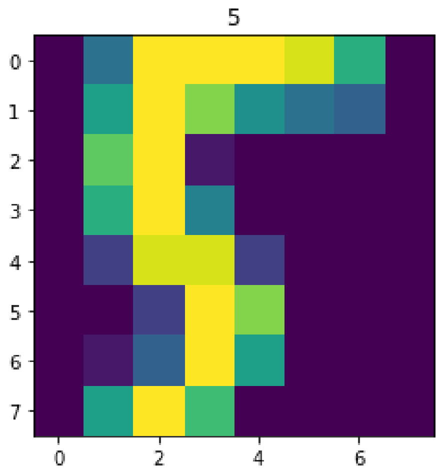
```
 [ 0.,   0.,   0., ..., 16.,   1.,   0.],
 [ 0.,   0.,   0., ..., 14.,   0.,   0.]],

...,

[[ 0.,   2., 16., ...,  0.,   0.,   0.],
 [ 0.,   7., 16., ...,  0.,   0.,   0.],
 [ 0.,   3., 10., ...,  0.,   0.,   0.],
 ...,
 [ 0.,   0.,   8., ..., 12.,   5.,   0.],
 [ 0.,   2., 16., ..., 16.,  15.,   2.],
 [ 0.,   2., 15., ..., 12.,   7.,   0.]],

[[ 0.,   0.,   0., ...,  2.,   0.,   0.],
 [ 0.,   0.,   0., ...,  0.,   0.,   0.],
 [ 0.,   0.,   0., ...,  0.,   6.,   0.],
 ...,
 [ 1.,  15., 16., ..., 16.,   3.,   0.],
 [ 0.,   3.,   7., ..., 11.,   0.,   0.],
 [ 0.,   0.,   0., ...,  3.,   0.,   0.]],

[[ 0.,   0.,   0., ...,  8.,   0.,   0.],
 [ 0.,   0.,   0., ...,  3.,   0.,   0.],
 [ 0.,   0.,   0., ...,  2.,   9.,   0.],
 ...,
 [ 1.,  15., 16., ..., 16.,   4.,   0.],
 [ 0.,   4.,   4., ..., 14.,   0.,   0.],
 [ 0.,   0.,   0., ...,  7.,   0.,   0.]]])
```

In [ ]:

```
#Now let's print() the result with matplot lib
plt.imshow(images[outliers][5])
plt.title(y[outliers][0])
```

Out[ ]:

```
Text(0.5, 1.0, '5')
```



In [1]:

```
#Let's try with a concrete case of the daily newspaper. Here the banking data. But there
are many fields of application. For exemple in the factory, medical analysis etc..
from google.colab import files
#Let's charge the creditcard file with some error inside

uploaded = files.upload()
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

In [ ]:

```python
import pandas as pd
import io
# Let's see if the data is imported well
df = pd.read_csv(io.BytesIO(uploaded['creditcard.csv']))
df.head()
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	0.

5 rows × 31 columns

In [ ]:

```python
df.isnull().values.any()
```

Out[ ]:

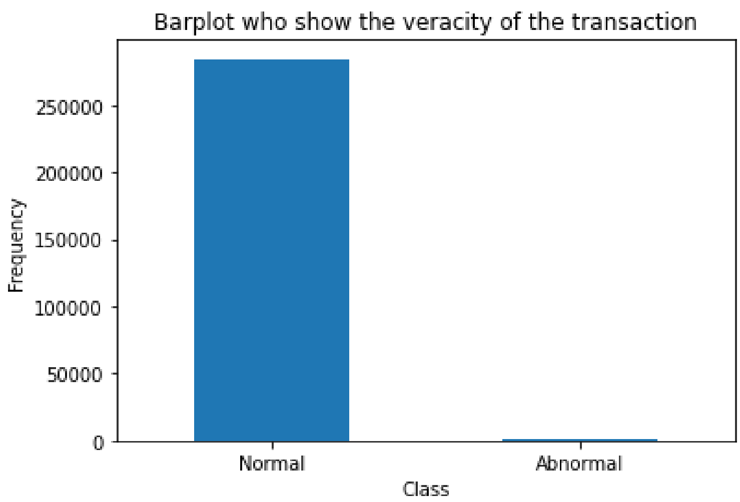
```
False
```

In [ ]:

```python
plotting = pd.value_counts(df['Class'], sort = True)
plotting.plot(kind = 'bar', rot=0)

plt.title("Barplot who show the veracity of the transaction")
plt.xticks(range(2), ["Normal", "Abnormal"])
plt.xlabel("Class")
plt.ylabel("Frequency")
```

Out[ ]:

```
Text(0, 0.5, 'Frequency')
```



In [ ]:

```python
## Let's implement the abnormal and the normal dataset
normal = df[df['Class']==0]
abnormal = df[df['Class']==1]
print(normal.head(),abnormal.head())
```

```
   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]         Time        V1        V2        V3        V4        V5
V6  \
541    406.0 -2.312227  1.951992 -1.609851  3.997906 -0.522188 -1.426545
623    472.0 -3.043541 -3.157307  1.088463  2.288644  1.359805 -1.064823
4920  4462.0 -2.303350  1.759247 -0.359745  2.330243 -0.821628 -0.075788
6108  6986.0 -4.397974  1.358367 -2.592844  2.679787 -1.128131 -1.706536
6329  7519.0  1.234235  3.019740 -4.304597  4.732795  3.624201 -1.357746

            V7        V8        V9  ...       V21       V22       V23  \
541  -2.537387  1.391657 -2.770089  ...  0.517232 -0.035049 -0.465211
623   0.325574 -0.067794 -0.270953  ...  0.661696  0.435477  1.375966
4920  0.562320 -0.399147 -0.238253  ... -0.294166 -0.932391  0.172726
6108 -3.496197 -0.248778 -0.247768  ...  0.573574  0.176968 -0.436207
6329  1.713445 -0.496358 -1.282858  ... -0.379068 -0.704181 -0.656805

           V24       V25       V26       V27       V28  Amount  Class
541   0.320198  0.044519  0.177840  0.261145 -0.143276    0.00      1
623  -0.293803  0.279798 -0.145362 -0.252773  0.035764  529.00      1
4920 -0.087330 -0.156114 -0.542628  0.039566 -0.153029  239.93      1
6108 -0.053502  0.252405 -0.657488 -0.827136  0.849573   59.00      1
6329 -1.632653  1.488901  0.566797 -0.010016  0.146793    1.00      1

[5 rows x 31 columns]
```

In [ ]:

```
#Now lets see the shape of the sorted data
print("Shape of the normal data",normal.shape)
print("Shape of the abnormal data",abnormal.shape)

#Also we need to understand how is compose the abnormal data so we are checking the descr
iption of it more precisely
print("Description of the abnormal information")
abnormal.Amount.describe()
```

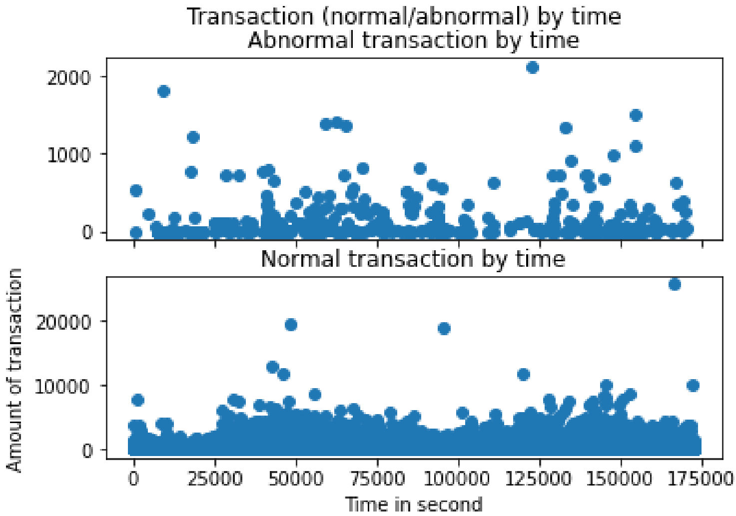
```
Shape of the normal data (284315, 31)
Shape of the abnormal data (492, 31)
Description of the abnormal information
```

Out[ ]:

```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

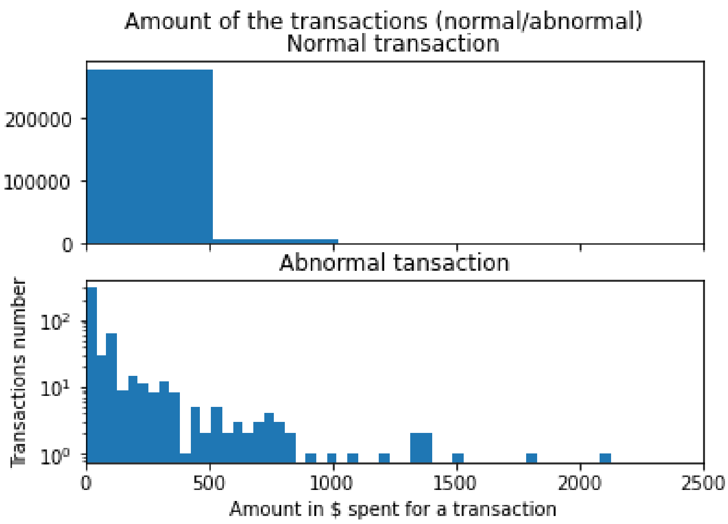
In [ ]:

```
# Now let's check the if the abnormal transactions occur more often during certain time f
rame
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Transaction (normal/abnormal) by time')
ax1.scatter(abnormal.Time, abnormal.Amount)
ax1.set_title('Abnormal transaction by time')

ax2.scatter(normal.Time,normal.Amount)
ax2.set_title('Normal transaction by time')
plt.xlabel('Time in second')
plt.ylabel('Amount of transaction')

plt.show()
```



In [ ]:

```
#One other interesting information is to know of much money transaction are made of
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)

f.suptitle('Amount of the transactions (normal/abnormal)')
ax1.hist(normal.Amount, bins = 50)
ax1.set_title('Normal transaction')
plt.xlabel('Amount in $ spent for a transaction')
plt.ylabel('Transactions number')
ax2.hist(abnormal.Amount, bins = 50)
ax2.set_title('Abnormal tansaction')
plt.xlabel('Amount in $ spent for a transaction')
plt.ylabel('Transactions number')

plt.xlim((0, 2500))
plt.yscale('log')
plt.show();
```



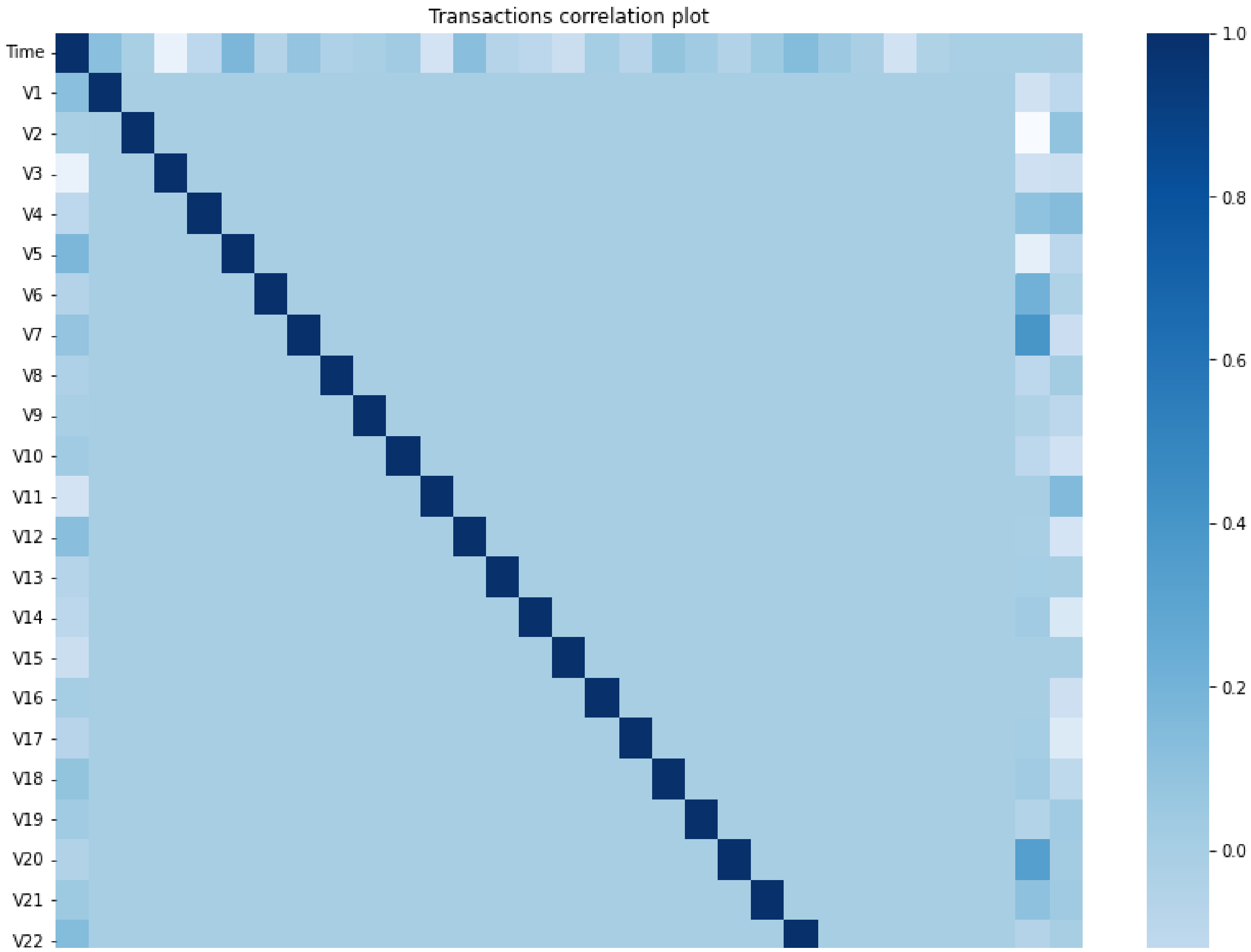
In [ ]:

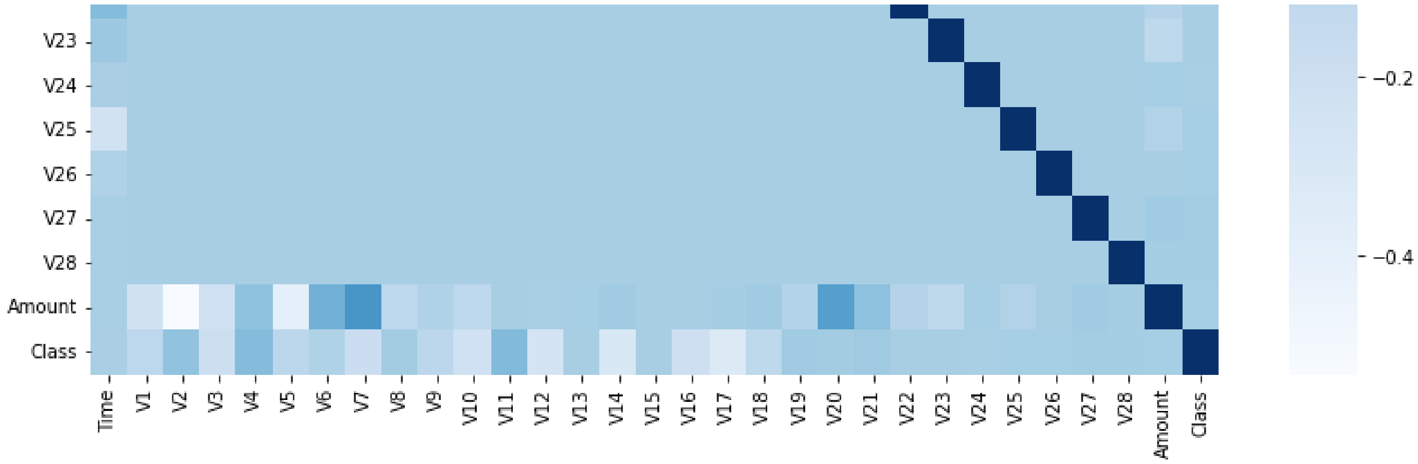
```
# Interesting visual analysis, it's seems like the abnormal transaction are proportional with the normal transaction. But also that the huge trasaction are detected as anomaly.
# Lets take some sample of the data
sample = df.sample(frac = 0.2,random_state=1)

#And now let's go more deeply and lets determine the number of fraud and valid transactions in the dataset

Abnormal = sample[sample['Class']==1]
Normal = sample[sample['Class']==0]
# And lets calculate the portion of the abnormal data between the normal data (mean)
Meaner = len(Abnormal)/float(len(Normal))
print("Pourcent of abnormal data in the transaction: ", Meaner*100, "%")
print("Abnormal transaction: ",format(len(Abnormal)))
print("Valid transaction: ",format(len(Normal)))
```

```
Pourcent of abnormal data in the transaction:  0.15296972254457222 %
Abnormal transaction:  87
Valid transaction:  56874
```

In [ ]:

```
## Lets import seaborn library to plot the heatmap
import seaborn as sns

#And lets creat a figure to show with a heat map the correlation of the abnormal and normal
plt.figure(figsize = (14,14))
plt.title('Transactions correlation plot')
correlation = df.corr()

sns.heatmap(correlation,xticklabels=correlation.columns,yticklabels=correlation.columns,cmap="Blues")

plt.show()
```





In [ ]:

```
#Create independent and Dependent Features
columns = df.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 are predicting
target = "Class"
# Define a random state
state = np.random.RandomState()
X = df[columns]
Y = df[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))

classifiers = {
    "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(X),contamination
=0.02,random_state=state, verbose=0)
}

n_outliers = len(Abnormal)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    if clf_name == "Isolation Forest":
        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))
    print("Accuracy Score :",accuracy_score(Y,y_pred))
    print("Classification Report :",classification_report(Y,y_pred))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have
valid feature names, but IsolationForest was fitted with feature names
  "X does not have valid feature names, but"
```

```
Isolation Forest: 5385
Accuracy Score : 0.9810924591038844
Classification Report :               precision    recall  f1-score   support

           0       1.00      0.98      0.99    284315
           1       0.07      0.82      0.13       492

    accuracy                           0.98    284807
   macro avg       0.54      0.90      0.56    284807
weighted avg       1.00      0.98      0.99    284807
```

Conclusion :

To conclude with the anomaly detection, I explored the possibility of using the "isolation forest" algorithm which is a very efficient algorithm. It has a strong point which is to be able to analyze datasets of very large volume (many variables).

Thing to know: There is another kind of algorithm, the "Local Outlier Factor". It is based on the nearest neighbors and allows novelty detection. This, like outlier detection like forest isolation which is based on the dataset, allows to detect anomalies in future data.

Source : https://machinelearnia.com/
https://towardsdatascience.com/anomaly-detection-with-autoencoder-b4cdce4866a6
https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
https://www.kaggle.com/code/chocojh/starter-credit-card-fraud-detection-249b5984-4
https://www.kaggle.com/code/prayasgautam/credit-card-fraud-detection