Projet d'Architecture Big-Data: Machine Learning
Détection de Fraude par Carte de Crédit
Moussa Diakho & Nango Fofana

Le but de ce projet est d'établir si une transaction effectuée par carte de crédit est frauduleuse ou pas.

L'ensemble de données contient les transactions effectuées par carte de crédit en septembre 2013 par les titulaires de carte européens.

Cet ensemble de données présente les transactions qui ont eu lieu en deux jours, où nous avons 492 fraudes sur 284 807 transactions. L'ensemble de données est très déséquilibré, la classe positive (fraudes) représente 0,172% de toutes les transactions (donc, à fortiriori on va faire une réduction de dimension pour rééquilibrer nos données).

Il ne contient que des variables d'entrée numériques qui sont le résultat d'une transformation PCA (analyse en composante principale). Les fonctionnalités V1, V2,… V28 sont les principaux composants obtenus avec PCA, les seules fonctionnalités qui n'ont pas été transformées avec PCA sont 'Time' et 'Amount'. La variable «Time» contient les secondes écoulées entre chaque transaction et la première transaction de l'ensemble de données. La fonction «amount» est le montant de la transaction, cette variable peut être utilisée pour un apprentissage dépendant des coûts, par exemple. La caractéristique «Class» est la variable de réponse et prend la valeur 1 en cas de fraude et 0 sinon.

Malheureusement en raison de confidentialité, les fonctionnalités d'origines et les informations générales relatives aux données (notamment les colonnes v1 à v28) ne seront pas fournit. cela ne nous empêchera pas de réaliser nos modèles de machines learning pour prédir si une transaction est frauduleuse ou non.

La méthodologie utilisé pour l'étude de ce projet est établi ci-dessous

In [1]:
```
from IPython.display import Image
Image(filename = "image4.png", width=1300, height=1300)
```

In [2]:
```
## Initialisation ##
# Import des librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from math import *
plt.style.use("ggplot")
```

In [3]:
```
## Import des données ##
df = pd.read_csv("C:/Users/mouss/Downloads/creditcard.csv", sep=",")
```

1. Analyse des données

La première étape de ce projet consiste à analyser et comprendre les données. Cette étape es cruciale pour améliorer la qualité des données en complétant les valeurs manquantes, selectionnant seulement les variables pertinentes à l'étude ou encore créant de nouvelles variables pertinentes à partir de celles dont on dispose. Toutes ces transformations vont contribuer à une meilleure qualité de prédiction des modèles de machine learning que nous utiliserons plus tard, mais aussi à interpréter notre modèle et savoir si les résultats semblent cohérents vis à vis des données et des problématiques de détection de fraude.

1.1. Exploration du data-frame

In [4]:
```
print(df.columns)
print("\n-----------------------------------------------------------------------------------------------\n")
print(df.info())
print("\n-----------------------------------------------------------------------------------------------\n")
print(df.describe())
print("\n-----------------------------------------------------------------------------------------------\n")
print("Ce data frame comporte {} lignes et {} colonnes.".format(df.shape[0], df.shape[1]))
```

```
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')

-----------------------------------------------------------------------------------------------

<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
None

-----------------------------------------------------------------------------------------------

                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]

-----------------------------------------------------------------------------------------------

Ce data frame comporte 284807 lignes et 31 colonnes.
```

In [5]:
```
# On peut regarder les 10 premières lignes de notre data-frame
df.head(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
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	-0.110474
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	0.101288
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	0.909412
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	-0.190321
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	-0.137458
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794

10 rows × 31 columns

In [6]:
```
# Les 10 dernières lignes de notre data-frame
df.tail(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
284797	172782.0	-0.241923	0.712247	0.399806	-0.463406	0.244531	-1.343668	0.929369	-0.206210	0.106234	...	-0.228876	-0.514376
284798	172782.0	0.219529	0.881246	-0.635891	0.960928	-0.152971	-1.014307	0.427126	0.121340	-0.285670	...	0.099936	0.337120
284799	172783.0	-1.775135	-0.004235	1.189786	0.331096	1.196063	5.519980	-1.518185	2.080825	1.159498	...	0.103302	0.654850
284800	172784.0	2.039560	-0.175233	-1.196825	0.234580	-0.008713	-0.726571	0.017050	-0.118228	0.435402	...	-0.268048	-0.717211
284801	172785.0	0.120316	0.931005	-0.546012	-0.745097	1.130314	-0.235973	0.812722	0.115093	-0.204064	...	-0.314205	-0.808520
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078

10 rows × 31 columns

In [7]:
```
print("Classe : ",df['Class'].unique())
print("V1 : ", df['V1'].unique())
print("Montant", df['Amount'].unique())
print("Temps : ", df['Time'].unique())
```

```
Classe :  [0 1]
V1 :  [-1.35980713  1.19185711 -1.35835406 ...  1.91956501 -0.24044005
 -0.53341252]
Montant [149.62   2.69 378.66 ... 381.05 337.54  95.63]
Temps :  [0.00000e+00 1.00000e+00 2.00000e+00 ... 1.72787e+05 1.72788e+05
 1.72792e+05]
```

On constate que notre data-frame ne comporte que des variable de type numérique, c'est à dire que les valeurs des colonnes sont des réelles.

De plus la colonne "Class" est la variable de réponse et prend la valeur 0 ou 1, donc c'est un problème de classification binaire.

1.2. Traiter les données manquantes

In [8]:
```
df.isnull().any()
# On remarque qu'il n'y a pas de valeurs manquantes
```

```
Time      False
V1        False
V2        False
V3        False
V4        False
V5        False
V6        False
V7        False
V8        False
V9        False
V10       False
V11       False
V12       False
V13       False
V14       False
V15       False
V16       False
V17       False
V18       False
V19       False
V20       False
V21       False
V22       False
V23       False
V24       False
V25       False
V26       False
V27       False
V28       False
Amount    False
Class     False
dtype: bool
```

In [9]:
```
df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

On constate qu'il n'y a pas de donnée manquante ce qui va faciliter le traitement des données.

Résumé numérique --> Uniquement les colonnes d'intérêt non anonymisées.

C'est à dire les les colonnes "Time", "Amount" et "Class".

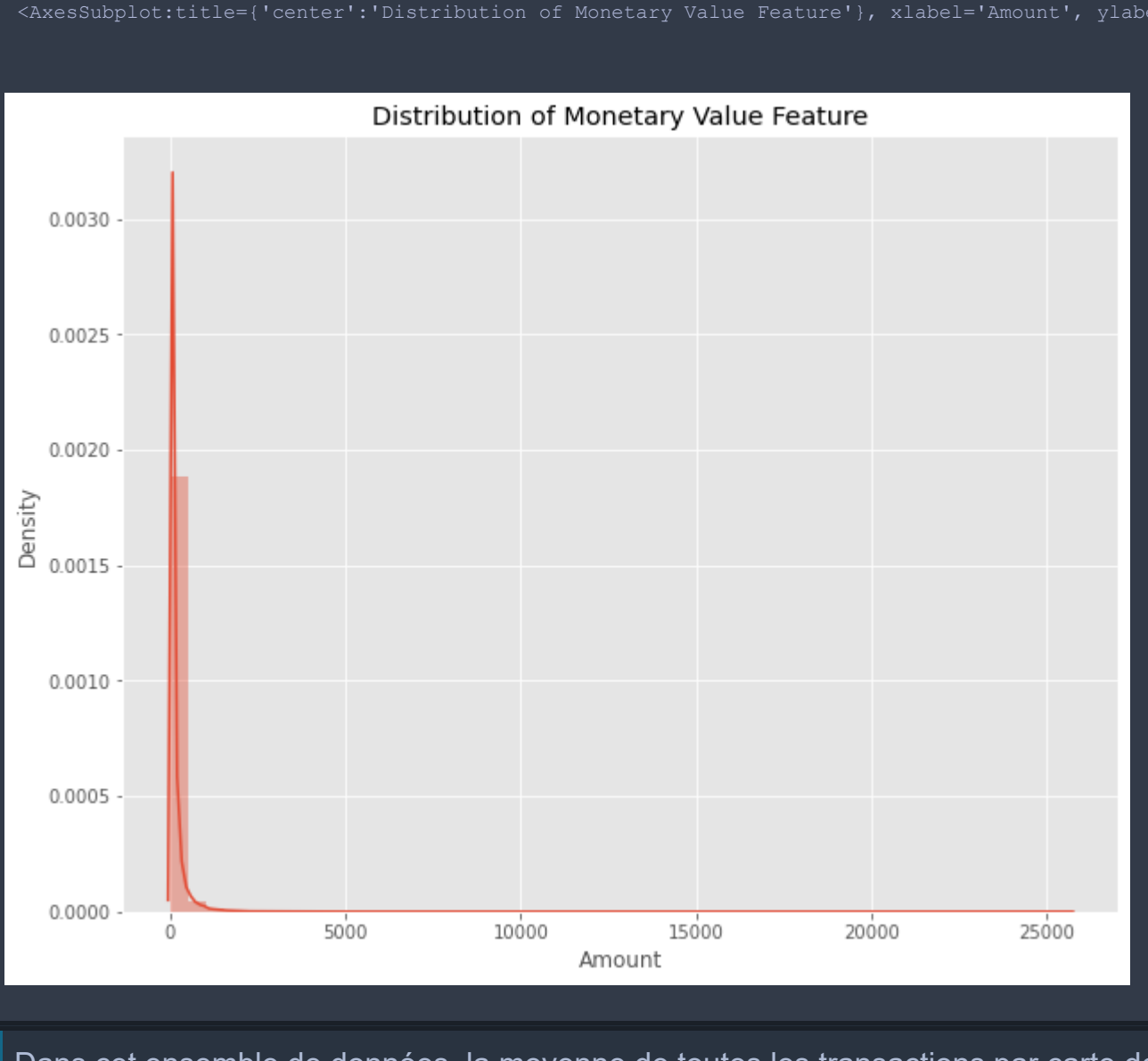
In [10]:
```
pd.set_option('precision',3)
df.loc[:, ['Time', 'Amount','Class']].describe()
```

	Time	Amount	Class
count	284807.000	284807.000	284807.000
mean	94813.860	88.350	0.002
std	47488.146	250.120	0.042
min	0.000	0.000	0.000
25%	54201.500	5.600	0.000
50%	84692.000	22.000	0.000
75%	139320.500	77.165	0.000
max	172792.000	25691.160	1.000

1.3. Visualisation

In [11]:
```
## Visualisation du temps et du montant
plt.figure(figsize=(10,8))
plt.title('Distribution of Monetary Value Feature')
sns.distplot(df.Amount)
```

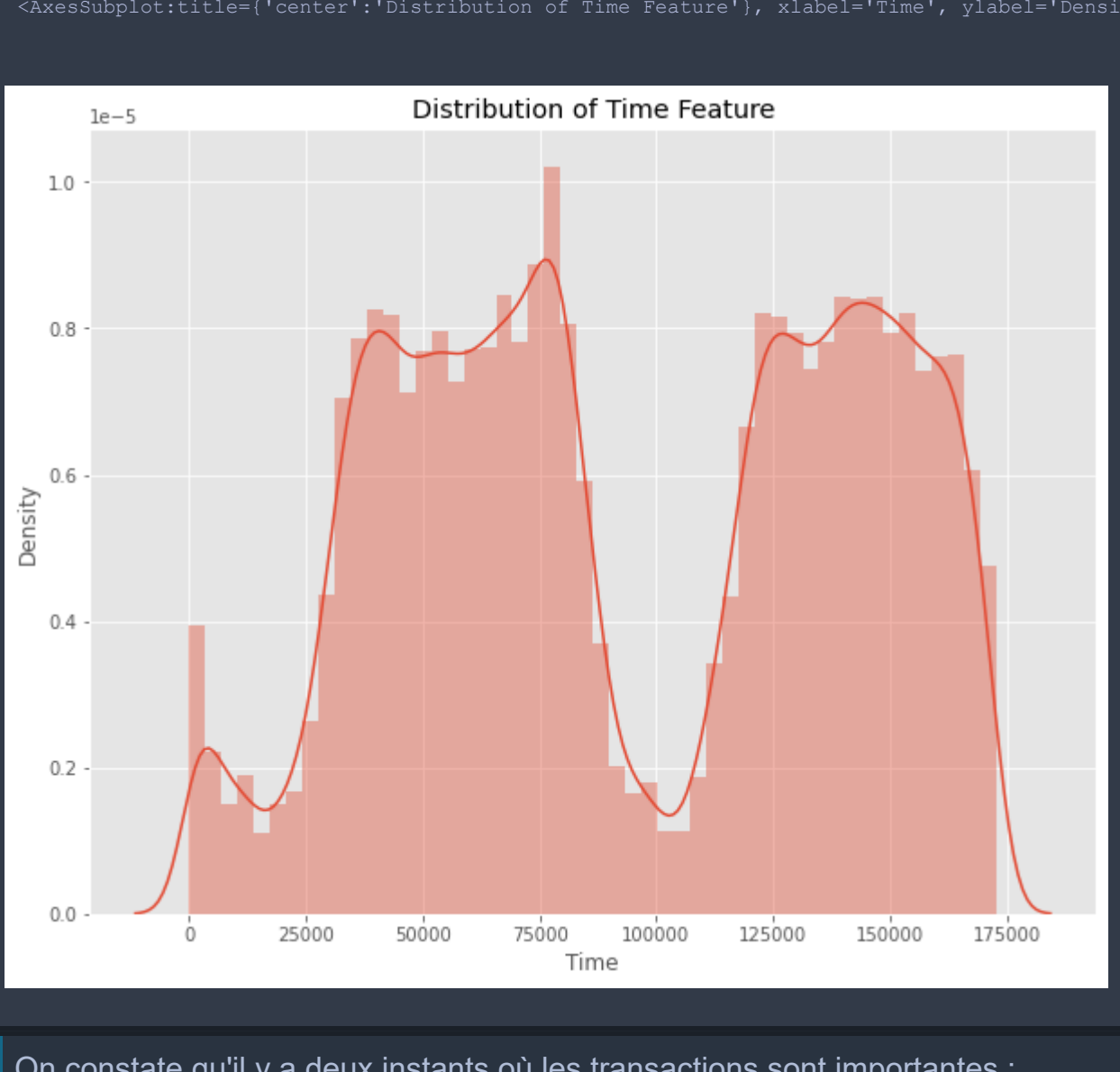
```
C:\Users\mouss\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)

<AxesSubplot:title={'center':'Distribution of Monetary Value Feature'}, xlabel='Amount', ylabel='Density'>
```



Dans cet ensemble de données, la moyenne de toutes les transactions par carte de crédit est d'environs 88 dollars. La plus grosse transaction avait une valeur monétaire d'environs 25691 dollars.

In [12]:
```
plt.figure(figsize=(10,8))
plt.title('Distribution of Time Feature')
sns.distplot(df.Time)
```

```
C:\Users\mouss\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)

<AxesSubplot:title={'center':'Distribution of Time Feature'}, xlabel='Time', ylabel='Density'>
```



On constate qu'il y a deux instants où les transactions sont importantes :

- Le premier intant est lorsque time aux environs de 85000 secondes.
- Le second instant est lorsque time est aux ernvirons de 145000 secondes

In [13]:
```
count = df.Class.value_counts()
print(count)
```

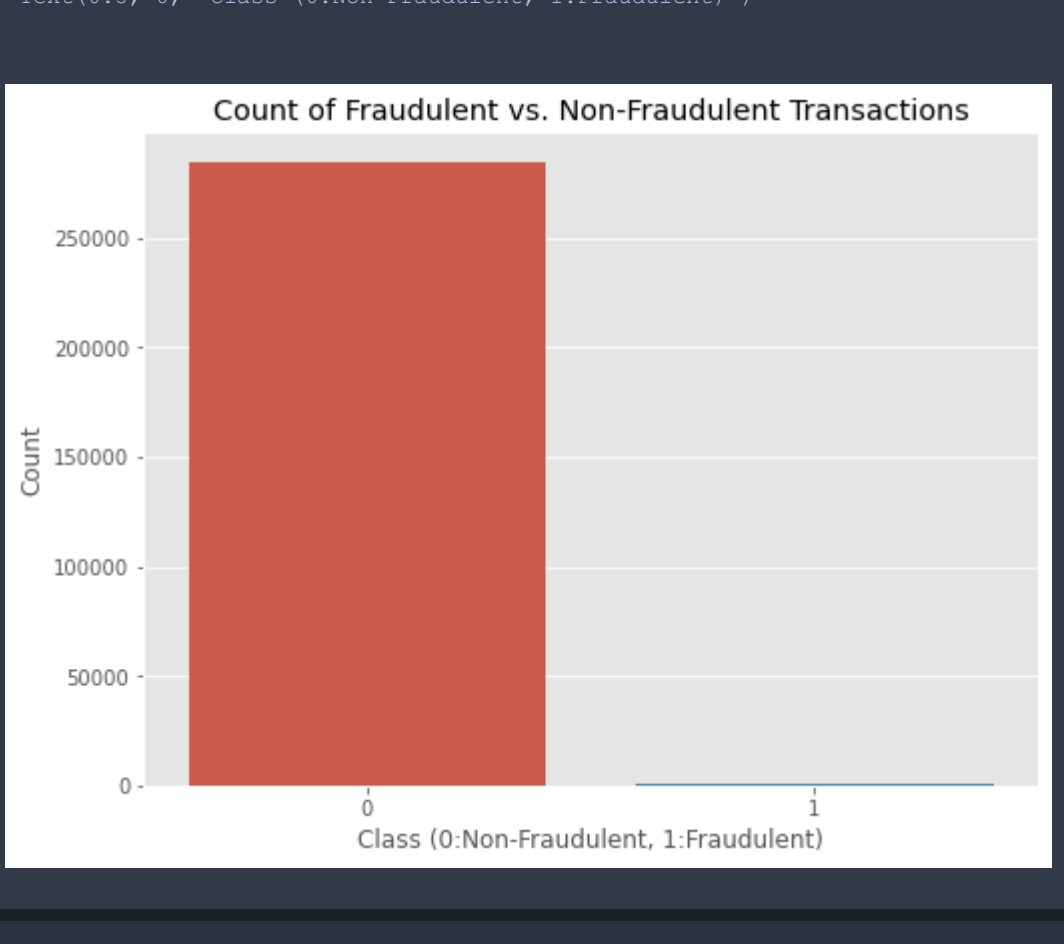
```
0    284315
1       492
Name: Class, dtype: int64
```

In [14]:
```
# On calcule le nombre de transaction normale et de transaction frauduleuse en pourcentage
transac_normal = count[0]
transac_frauduleuse = count[1]
pour_tr_norm = (transac_normal/(transac_normal+transac_frauduleuse))*100 # % de transaction
 normale
pour_tr_fraud = (transac_frauduleuse/(transac_normal+transac_frauduleuse))*100 # % de trans
action frauduleuse
print("Il y a {} transactions non frauduleuses ({:.3f}%) et {} transactions frauduleuses (
{:.3f}%).".format(transac_normal,pour_tr_norm,transac_frauduleuse,pour_tr_fraud))
```

```
Il y a 284315 transactions non frauduleuses (99.827%) et 492 transactions frauduleuses (0.173%).
```

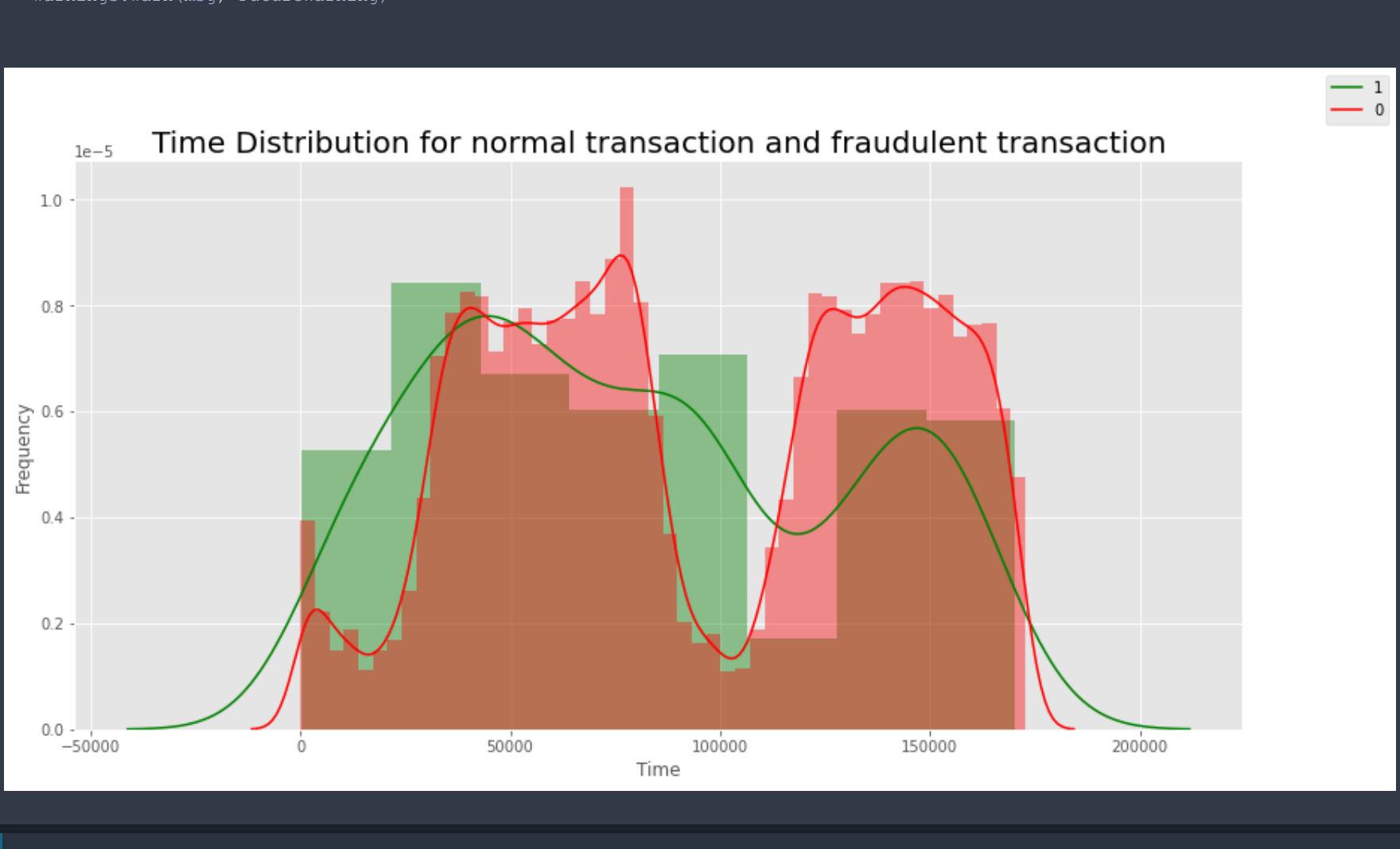
In [15]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=count.index, y=count)
plt.title('Count of Fraudulent vs. Non-Fraudulent Transactions')
plt.ylabel('Count')
plt.xlabel('Class (0:Non-Fraudulent, 1:Fraudulent)')
```

```
Text(0.5, 0, 'Class (0:Non-Fraudulent, 1:Fraudulent)')
```



In [16]:
```
fig, ax = plt.subplots(figsize=(14,7))
g = sns.distplot(df[df["Class"]==1]["Time"], ax=ax, color='g')
g = sns.distplot(df[df["Class"]==0]["Time"], ax=ax, color='r')
g.set_xlabel("Time", fontsize=12)
g.set_ylabel("Frequency", fontsize=12)
g.set_title("Time Distribution for normal transaction and fraudulent transaction", fontsize
=20)
fig.legend(labels=[1,0])
plt.show()
```

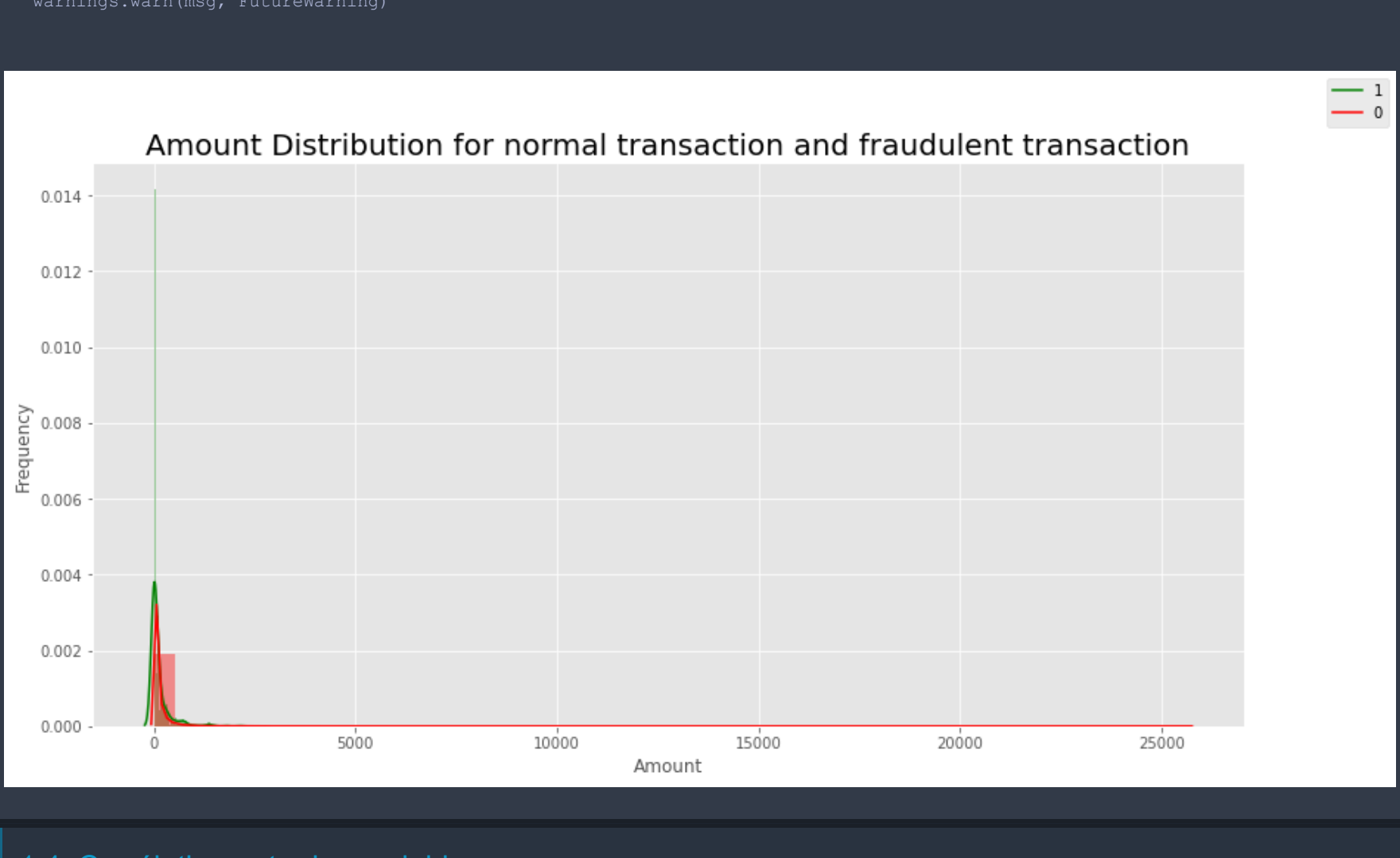
```
C:\Users\mouss\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\mouss\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



On constate que pour $Time \leq 50000$ secondes il y a plus de de fraude que sur le reste.

In [17]:
```
# On peut faire de même pour les montants
fig, ax = plt.subplots(figsize=(14,7))
g = sns.distplot(df[df["Class"]==1]["Amount"], ax=ax, color='g')
g = sns.distplot(df[df["Class"]==0]["Amount"], ax=ax, color='r')
g.set_xlabel("Amount", fontsize=12)
g.set_ylabel("Frequency", fontsize=12)
g.set_title("Amount Distribution for normal transaction and fraudulent transaction", fontsi
ze=20)
fig.legend(labels=[1,0])
plt.show()
```

```
C:\Users\mouss\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\mouss\Anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



1.4. Corrélation entre les variables

Nous avons 30 varaibles candidates. On doit sélectionner les variables qui nous permettrait au mieux de prédire la variable "Class" qui est le vecteur des transactions frauduleuses (1 pour frauduleuse) ou non frauduleuse (0 non frauduleuse).

La correlation permet la mesure de la force de liaison entre deux variables. Elle nous permettra de voir quels variables sont fortement liées à class avec un seuil significative de 0.5%

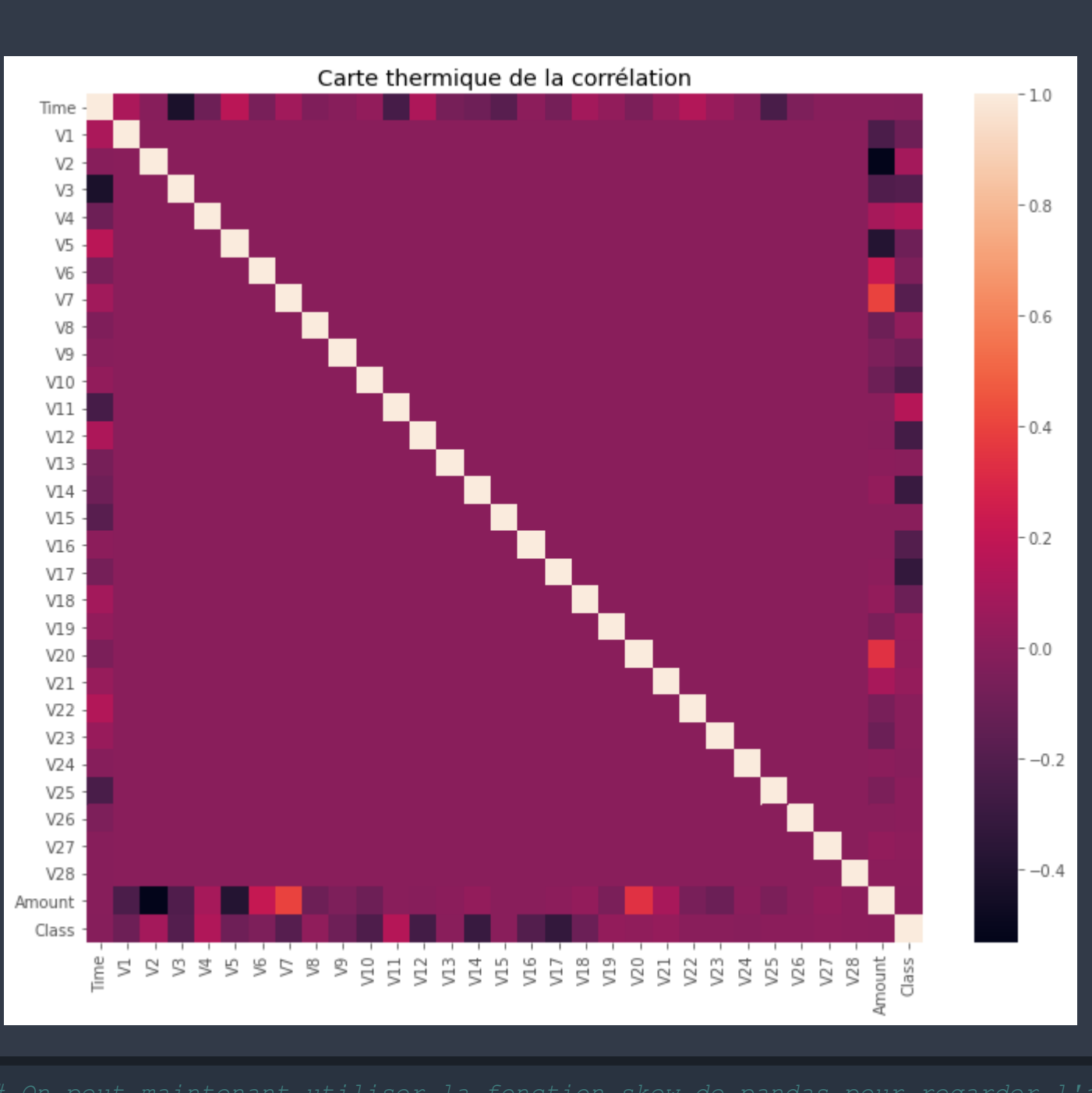
In [18]:
```
# corrélation
coor = df.corr()
coor
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
Time	1.000	1.174e-01	-1.059e-02	-4.196e-01	-1.053e-01	1.731e-01	-6.302e-02	8.471e-02	-3.695e-02	-8.660e-03	...	4.474e-02	1.441e-01
V1	0.117	1.000e+00	4.697e-17	-1.424e-15	1.755e-17	6.391e-17	2.398e-16	1.992e-15	-9.491e-17	2.170e-16	...	-1.755e-16	7.477e-17
V2	-0.011	4.697e-17	1.000e+00	2.512e-16	-1.126e-16	-2.040e-16	5.025e-16	3.966e-16	-4.414e-17	-5.729e-17	...	8.444e-17	2.501e-16
V3	-0.420	-1.424e-15	2.512e-16	1.000e+00	-3.417e-16	-1.437e-15	1.432e-15	2.169e-15	3.433e-16	-4.234e-16	...	-2.972e-17	4.648e-16
V4	-0.105	1.755e-17	-1.126e-16	-3.417e-16	1.000e+00	-1.941e-15	-2.713e-16	1.556e-16	5.196e-16	3.860e-16	...	-9.977e-17	2.100e-16
V5	0.173	6.391e-17	-2.040e-16	-1.437e-15	-1.941e-15	1.000e+00	7.926e-16	-4.210e-16	7.589e-16	4.205e-16	...	-1.369e-16	5.060e-16
V6	-0.063	2.398e-16	5.025e-16	1.432e-15	-2.713e-16	7.926e-16	1.000e+00	1.429e-16	-1.707e-16	1.114e-16	...	-1.576e-16	-3.363e-16
V7	0.085	1.992e-15	3.966e-16	2.169e-15	1.556e-16	-4.210e-16	1.429e-16	1.000e+00	-8.692e-17	7.933e-16	...	1.939e-16	-1.058e-15
V8	-0.037	-9.491e-17	-4.414e-17	3.433e-16	5.196e-16	7.589e-16	-1.707e-16	-8.692e-17	1.000e+00	2.901e-16	...	-2.412e-16	5.476e-16
V9	-0.009	2.170e-16	-5.729e-17	-4.234e-16	3.860e-16	4.205e-16	1.114e-16	7.933e-16	2.901e-16	1.000e+00	...	4.578e-17	2.872e-17
V10	0.031	7.434e-17	-4.782e-16	6.289e-16	6.055e-16	-6.602e-16	2.851e-16	3.043e-17	9.052e-17	-2.772e-16	...	8.090e-16	-6.708e-16
V11	-0.248	2.439e-16	9.469e-16	-5.502e-17	-2.084e-16	7.343e-16	4.866e-16	-1.084e-15	1.955e-16	4.682e-16	...	-3.912e-16	-3.812e-17
V12	0.124	2.422e-16	-6.588e-16	2.207e-16	-5.658e-16	3.761e-16	2.141e-16	1.510e-15	-6.266e-17	-2.445e-15	...	3.230e-16	-5.904e-16
V13	-0.066	-2.115e-16	3.855e-16	-6.883e-16	-1.506e-16	-9.579e-16	-2.268e-16	-9.892e-17	-2.383e-16	-2.650e-16	...	9.499e-17	-2.660e-17
V14	-0.099	9.353e-16	-2.541e-16	4.271e-16	-8.522e-17	-3.635e-16	3.453e-16	-1.729e-16	-1.131e-16	2.343e-16	...	1.634e-17	3.440e-16
V15	-0.183	-3.252e-16	2.831e-16	1.123e-16	-1.508e-16	-5.133e-16	-6.368e-18	1.937e-17	2.021e-16	-1.588e-15	...	1.947e-17	-8.937e-16
V16	0.012	6.309e-16	4.934e-17	1.183e-15	-6.939e-16	-3.517e-16	-2.478e-16	2.894e-16	5.027e-16	-3.252e-16	...	-3.927e-16	3.878e-17
V17	-0.073	-5.012e-16	-9.883e-16	4.577e-17	-4.398e-16	1.426e-16	3.568e-16	1.150e-15	-3.509e-16	6.536e-16	...	-7.754e-16	-8.390e-17
V18	0.090	2.870e-16	2.637e-16	5.428e-16	1.494e-16	1.110e-15	2.811e-16	-1.117e-16	-4.094e-16	1.204e-16	...	-1.141e-15	-8.663e-17
V19	0.029	1.818e-16	9.528e-17	2.577e-16	-2.657e-16	-3.138e-16	2.717e-16	-2.874e-16	-5.340e-16	1.121e-16	...	4.033e-16	-9.690e-16
V20	-0.051	1.037e-16	-9.310e-16	-9.429e-16	-3.223e-16	2.076e-16	1.899e-16	1.744e-16	-1.096e-16	-4.341e-16	...	-1.121e-15	1.106e-15
V21	0.045	-1.755e-16	8.444e-17	-2.972e-17	-9.977e-17	-1.369e-16	-1.576e-16	1.939e-16	-2.412e-16	4.578e-17	...	1.000e+00	3.906e-15
V22	0.144	7.477e-17	2.501e-16	4.648e-16	2.100e-16	5.060e-16	-3.363e-16	-1.058e-15	5.476e-16	2.872e-17	...	3.906e-15	1.000e+00
V23	0.051	9.809e-16	1.060e-16	2.115e-17	6.003e-17	1.638e-16	-7.232e-17	2.328e-16	3.897e-16	5.929e-16	...	6.127e-16	3.131e-16
V24	-0.016	7.354e-17	-8.142e-18	-9.352e-17	2.230e-16	-9.286e-16	-1.262e-15	-2.590e-17	-1.803e-16	-2.346e-16	...	1.298e-16	1.151e-17
V25	-0.233	-9.805e-16	-4.262e-17	4.771e-16	5.395e-16	5.625e-16	1.082e-15	1.174e-15	-1.391e-16	1.100e-15	...	-2.826e-16	-6.079e-16
V26	-0.041	-8.622e-17	2.602e-16	6.522e-16	-6.180e-16	9.145e-16	-2.378e-16	-7.335e-16	-1.210e-16	-1.389e-15	...	-4.907e-16	-8.477e-16
V27	-0.005	3.208e-17	-4.478e-16	6.240e-16	-6.403e-17	4.466e-16	-2.624e-16	-5.887e-16	1.734e-16	-2.287e-16	...	-1.033e-15	-1.295e-16
V28	-0.009	9.821e-16	-3.676e-16	7.727e-16	-5.864e-17	-3.299e-16	4.813e-16	-6.837e-17	-4.484e-16	9.147e-16	...	5.132e-16	-3.021e-16
Amount	-0.011	-2.277e-01	-5.314e-01	-2.109e-01	9.873e-02	-3.864e-01	2.160e-01	3.973e-01	-1.031e-01	-4.425e-02	...	1.060e-01	-6.480e-02
Class	-0.012	-1.013e-01	9.129e-02	-1.930e-01	1.334e-01	-9.497e-02	-4.364e-02	-1.873e-01	1.988e-02	-9.773e-02	...	4.041e-02	8.053e-04

31 rows × 31 columns

In [19]:
```
# La carte thermique de la corrélation est donnée par
plt.figure(figsize=(12,10))
heat_matrix = sns.heatmap(data=coor)
plt.title("Carte thermique de la corrélation")
```

```
Text(0.5, 1.0, 'Carte thermique de la corrélation')
```



In [20]:
```
# On peut maintenant utiliser la fonction skew de pandas pour regarder l'asymétrie entre le
s variables
# Cette fonction renvoie une asymétrie sur l'axe demandé renormalisé par N-1 avec N la tail
le du data-frame
# c'est une mesure de l'asymétrie de la distribution de probabilité d'une variable aléatoir
e à valeur
# réelle par rapport à sa moyenne
sk = df.skew()
# On peut checker ce que ça donne et on a
sk
```

```
Time      -0.036
V1        -3.281
V2        -4.625
V3        -2.240
V4         0.676
V5        -2.426
V6         1.827
V7         2.554
V8        -8.522
V9         0.555
V10        1.187
V11        0.357
V12       -2.278
V13        0.065
V14       -1.995
V15       -0.308
V16       -1.101
V17       -3.845
V18       -0.260
V19        0.109
V20       -2.037
V21        3.593
V22       -0.213
V23       -5.875
V24       -0.552
V25       -0.416
V26        0.577
V27       -1.170
V28       11.192
Amount    16.978
Class     23.998
dtype: float64
```

2. Prétraitement des données

On commenc d'abord à faire une mise en échelle de colonnes des deux features qui nous intéresse c'est à dire "Amount" et "Time"

In [21]:
```
from sklearn.preprocessing import StandardScaler
scaler1 = StandardScaler()
scaler2 = StandardScaler()
```

```
In [22]: # Mise à l'échelle de la colonne "Time"
scale_Time = scaler1.fit_transform(df[["Time"]])
fl_liste1 = [item for sublist in scale_Time.tolist() for item in sublist]
scale_Time = pd.Series(fl_liste1)
scale_Time
```

```
0        -1.997
1        -1.997
2        -1.997
3        -1.997
4        -1.997
          ...
284802    1.642
284803    1.642
284804    1.642
284805    1.642
284806    1.642
Length: 284807, dtype: float64
```

```
In [23]: # Mise à l'échelle de la colonne "Amount"
scale_Amount = scaler2.fit_transform(df[["Amount"]])
fl_liste2 = [item for sublist in scale_Amount.tolist() for item in sublist]
scale_Amount = pd.Series(fl_liste2)
scale_Amount
```

```
0         0.245
1        -0.342
2         1.161
3         0.141
4        -0.073
          ...
284802   -0.350
284803   -0.254
284804   -0.082
284805   -0.313
284806    0.514
Length: 284807, dtype: float64
```

On peut maintenat concantener les colonnes nouvellement créées dans le data-frame de départ i.e df

```
In [24]: df = pd.concat([df, scale_Amount.rename('scale_Amount'), scale_Time.rename('scale_Time')], axis=1)
df.sample(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V23	V24	V25	V26	V27	V28	Amoun
118066	74936.0	1.335	0.331	-2.058	-0.346	2.583	2.854	-0.188	0.685	-0.287	...	-0.114	0.916	0.730	0.384	-0.032	0.030	0.76
173342	121447.0	1.986	-1.247	0.164	-0.298	-1.745	-0.064	-1.748	0.180	1.070	...	0.101	-0.039	-0.325	-0.029	0.094	0.002	46.99
56313	47364.0	1.343	0.036	-0.376	-0.167	0.270	-0.114	0.100	-0.095	-0.020	...	-0.248	-0.730	0.692	1.144	-0.095	-0.028	2.00
95789	65467.0	0.883	-1.926	1.129	-0.080	-2.103	0.440	-1.344	0.277	0.952	...	-0.341	-0.077	0.402	-0.041	0.029	0.060	243.50
156305	107812.0	0.217	1.331	-0.834	0.186	0.419	-1.410	0.502	-0.006	1.045	...	-0.105	-0.068	-0.278	-0.193	-0.089	-0.052	1.50
94926	65092.0	-0.326	0.692	1.422	-0.681	0.157	-0.964	1.205	-0.466	-0.219	...	0.081	0.510	-0.425	0.507	-0.293	-0.397	19.98
199780	133134.0	1.693	-0.750	-0.476	0.425	-0.719	-0.531	-0.224	-0.131	0.839	...	0.377	1.165	-0.704	0.154	-0.067	-0.004	161.93
236514	148842.0	0.514	1.390	-1.091	1.605	0.461	-0.382	-0.599	-2.722	-0.567	...	0.130	0.789	0.910	-0.692	-0.033	0.097	16.70
227755	145206.0	2.170	-0.338	-2.580	-0.766	0.665	-1.009	0.510	-0.373	0.820	...	-0.083	0.102	0.367	1.090	-0.161	-0.092	27.83
131039	79486.0	1.061	0.079	0.202	1.326	-0.249	-0.593	0.296	-0.118	0.143	...	-0.128	0.423	0.695	-0.273	0.014	0.023	63.96

10 rows × 33 columns

Ainsi on peut supprimer les anciennes colonnes "Amount" et "Time" que seront remplacées par les colonnes nouvellement créées ("scale-Amount" et "scale_Time")

```
In [25]: df.drop(['Amount', 'Time'], axis=1, inplace=True)
df.sample(5)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Cla
281514	0.121	-3.452	-2.203	0.879	-1.104	-0.223	1.198	-0.341	0.807	-0.485	...	-0.716	-0.698	-0.283	-0.428	-0.192	-0.217	0.101	0
160805	2.092	0.505	-2.706	0.600	0.862	-1.310	0.294	-0.294	0.674	-1.184	...	0.306	-0.126	-1.011	0.319	-0.048	0.017	-0.002	0
109954	1.248	0.324	-0.966	1.320	2.355	3.708	-0.436	0.865	-0.866	0.760	...	-0.168	-0.106	1.001	0.720	0.083	-0.009	0.014	0
219021	-1.332	0.326	-0.941	0.402	-0.648	-1.219	0.448	0.519	-0.320	-1.387	...	0.260	0.653	0.347	-0.769	0.335	-0.083	-0.361	0
234426	-4.066	3.084	-0.166	-0.987	-0.571	3.084	-3.952	-9.208	1.418	-0.380	...	-1.958	1.288	-1.205	-0.215	-0.015	0.593	0.166	0

5 rows × 31 columns

A présent, on peut diviser le dataset en deux : le train et le test. Le fichier d'entrainement doit être assez volimineux pour obtenir des résultats statistiques significatifs. On choisis alors de sélectionner un échantillon aléatoire à hauteur de 80% des données du fichier de base. Si le fichier est totalement aléatoire on risque d'avoir un fichier d'entrainement contenant peu de transaction frauduleuse car ces dennières ne constitue que 0.17% du dataset df, par conséquent notre modèle risque de ne pas être significatif. On décide alors de créer un noveau fichier df2 qui servirat de fichier de base d'échantillonage contenant 100% des transactions frauduleuses de df soit 50% de ces données sont frauduleuses et 50% de ces données sont non frauduleuses.(voir histogramme ci-dessous).

Ainsi on a autant de chance de sélectionner aléatoirement une transaction frauduleuse qu'une transaction. Ensuite à partir de df2 on effectue un échantillonage aléatoire pour obtenir un fichier d'entrainement et un fichier test.

```
In [26]: # On commence d'abord créer le train et le test manuellement
DivCol = np.random.rand(len(df))<0.9 # un vecteur de booléen de taille len(df)
train = df[DivCol]
test = df[~DivCol]
print('forme du train: {}\nforme du test: {}'.format(train.shape, test.shape))
```

```
forme du train: (256062, 31)
forme du test: (28745, 31)
```

```
In [27]: train.reset_index(drop=True, inplace=True) # faire que les indices commence de 0 à len(train)
test.reset_index(drop=True, inplace=True) # faire que les indices commence de 0 à len(test)
```

```
In [28]: test
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Clas
0	1.230	0.141	0.045	1.203	0.192	0.273	-0.005	0.081	0.465	-0.099	...	-0.271	-0.154	-0.780	0.750	-0.257	0.035	0.005	0
1	-2.792	-0.328	1.642	1.767	-0.137	0.808	-0.423	-1.907	0.756	1.151	...	0.222	1.021	0.028	-0.233	-0.236	-0.165	-0.030	0
2	0.962	0.328	-0.171	2.109	1.130	1.696	0.108	0.522	-1.191	0.724	...	0.402	-0.049	-1.372	0.391	0.200	0.016	-0.015	0
3	0.247	0.278	1.185	-0.093	-1.314	-0.150	-0.946	-1.618	1.544	-0.830	...	0.200	-0.185	0.423	0.821	-0.228	0.337	0.250	0
4	-1.452	1.765	0.612	1.177	-0.446	0.247	-0.258	1.092	-0.608	0.047	...	0.326	-0.069	0.021	-0.045	-0.243	0.149	0.121	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28740	-0.828	0.650	1.844	0.191	-0.605	0.580	-0.614	0.922	0.259	-0.725	...	1.052	-0.167	-0.445	-0.408	-0.605	0.104	0.101	0
28741	-0.765	0.588	-0.908	-0.419	0.902	-0.761	0.759	0.415	-0.731	-1.245	...	-0.432	0.142	0.587	-0.201	0.267	-0.153	-0.065	0
28742	1.975	-0.616	-2.628	-0.406	2.328	3.665	-0.533	0.843	1.129	-0.221	...	0.544	-0.032	0.768	0.478	-0.032	0.014	-0.067	0
28743	1.920	-0.301	-3.250	-0.558	2.631	3.031	-0.297	0.708	0.432	-0.485	...	0.578	-0.038	0.640	0.266	-0.087	0.004	-0.027	0
28744	-0.240	0.530	0.703	0.690	-0.378	0.624	-0.686	0.679	0.392	-0.399	...	0.800	-0.163	0.123	-0.569	0.547	0.109	0.105	0

28745 rows × 31 columns

On va créer un ensemble de données de sous échantillon de df avec des distributions de classe équilibrées.

De ce fait nous avons besoins de savoir combien d'échantillons aléatoires de transactions normales nous avons besoin.

```
In [29]: nbr_de_fraude = train.Class.value_counts()[1]
print("Il y'a {} transactions frauduleuses dans les données train.".format(nbr_de_fraude))
```

```
Il y'a 444 transactions frauduleuses dans les données train.
```

```
In [30]: # On définit les vecteurs non_fraude et fraude comportant
non_fraude = train[train['Class']==0]
fraude = train[train['Class']==1]
```

La variable Class est une variable qualitative binaire qui a été codé de sorte à avoir deux modalités : 0 (non-fraude) et 1 (fraude), on dit aussi qu'elle est dichotomique. Le reste des variables du dataset sont toutes quantitatives continues. Notre choix de classifieur va surement se faire pour la réfression logistique qui est méthode de régression permettant la prédiction d'une variable qualitative binaire par des variables quantitatives. Ce modèle respecte le type de notre variavble d'intérêt et permet l'usage du reste des variables du dataset.

```
In [31]: # On peut sélectionner aléatoire 442 transactions non frauduleuses
selection = non_fraude.sample(nbr_de_fraude)
selection.head(10)
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Cla
50310	-1.473	0.468	2.587	3.263	-0.557	2.122	-0.858	1.336	-0.398	0.193	...	-0.067	0.031	-0.293	0.010	0.284	0.276	0.020	0
77604	-0.389	1.106	1.271	0.010	0.244	-0.499	0.672	0.041	-0.463	-0.570	...	-0.541	0.016	0.034	-0.218	0.100	0.264	0.096	0
248147	-1.297	-0.095	0.843	-0.682	1.625	-0.794	1.184	-0.663	-0.043	0.217	...	-0.297	0.034	-0.302	-0.072	0.207	-0.604	-0.229	0
190759	1.613	-0.670	-2.110	0.495	0.173	-0.742	0.426	-0.298	0.999	-0.960	...	-0.643	-0.101	-0.809	-0.060	-0.068	-0.046	0.012	0
170831	0.110	-4.958	-2.813	-1.074	-1.394	1.133	0.235	-0.036	-1.711	1.253	...	0.404	-0.879	-1.591	-0.521	0.078	-0.178	0.099	0
115224	1.270	-0.222	0.410	0.187	-0.412	0.182	-0.451	0.062	-1.359	0.897	...	-0.523	0.042	-0.337	0.368	-0.434	0.068	0.018	0
172905	2.420	-0.961	-1.816	-1.881	-0.212	-0.634	-0.504	-0.426	-1.935	1.624	...	-0.544	0.229	0.027	-0.004	-0.307	-0.017	-0.056	0
143775	-1.553	-0.996	1.618	0.816	1.557	0.520	-0.014	0.216	-0.208	-0.068	...	0.506	-0.100	-1.487	0.759	-0.217	-0.151	-0.131	0
86469	1.292	-0.247	-0.985	-0.290	1.889	3.655	-0.724	0.902	0.497	-0.148	...	-0.314	-0.118	1.063	0.745	0.454	-0.013	0.003	0
195201	-0.517	0.527	0.503	-0.736	1.201	0.234	1.352	-0.260	0.374	0.126	...	0.244	0.159	-0.584	-1.103	0.174	0.246	-0.110	0

10 rows × 31 columns

```
In [32]: # On remet à jour les indices de selection et de fraude. ainsi les deux tableaux auront comme indice de
# départ 0 et vont jusqu'à respectivement len(selection) et len(fraude)
selection.reset_index(drop=True, inplace=True)
fraude.reset_index(drop=True, inplace=True)
```

```
In [33]: # Concaténation des deux en un ensemble de données de sous-échantillon avec une distribution de classe
# égale
sous_ensemble = pd.concat([selection,fraude])
len(sous_ensemble)
```

```
888
```

```
In [34]: # Remaniement de l'ensemble de nos données
sous_ensemble = sous_ensemble.sample(frac=1).reset_index(drop=True)
sous_ensemble.head(10)
```

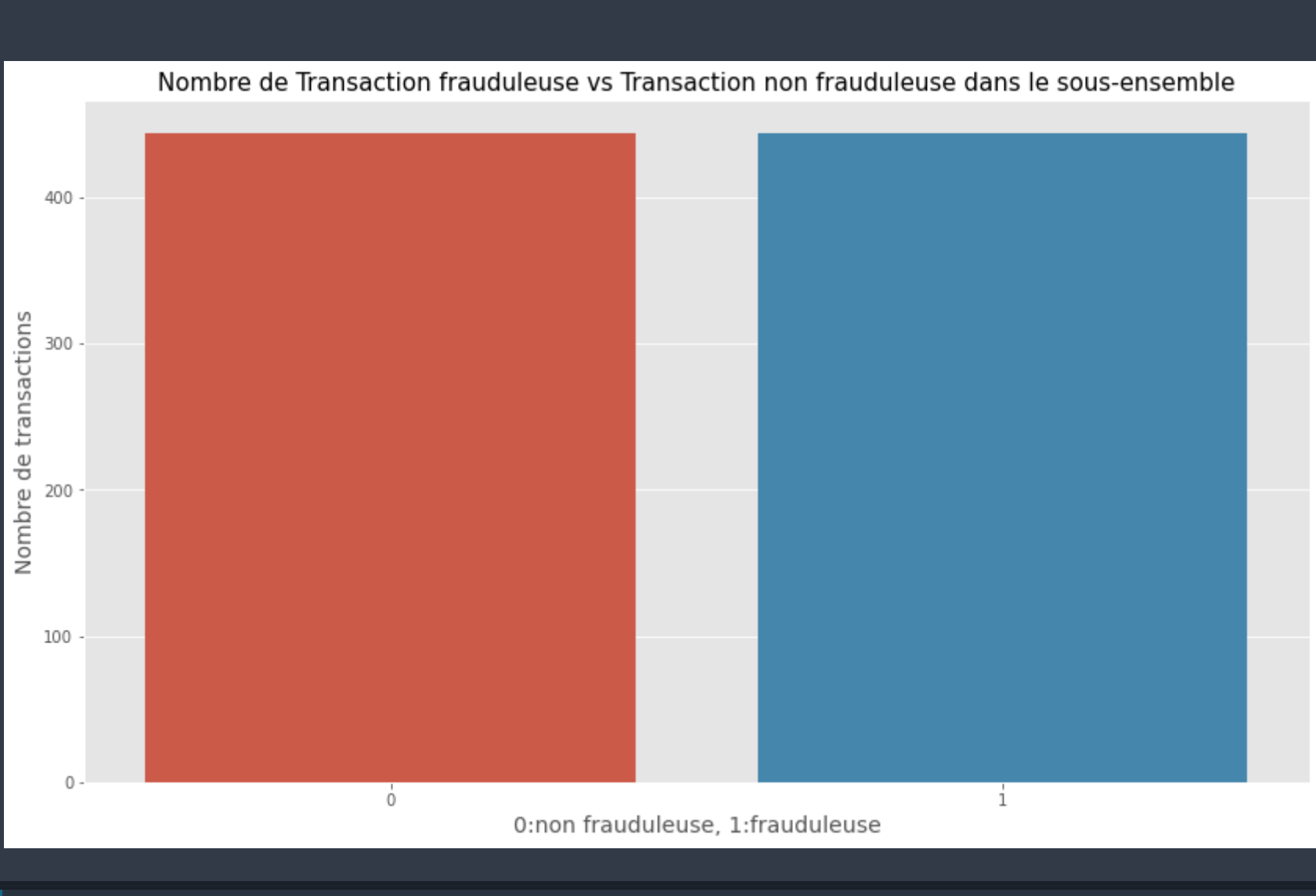
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Class
0	-3.975	0.582	-1.880	4.319	-3.024	1.241	-1.910	0.661	-2.753	-3.550	...	1.397	1.045	-0.304	0.005	0.235	0.962	-0.674	1
1	-5.188	6.968	-13.511	8.618	-11.214	0.672	-9.463	5.329	-4.897	-11.787	...	0.760	0.717	-0.647	-1.617	0.172	0.627	-0.170	1
2	-4.414	-1.867	-1.000	0.179	1.752	-1.530	0.520	-0.118	0.453	-0.359	...	-0.879	1.095	0.479	0.299	0.168	-1.349	1.778	0
3	-1.748	-0.639	-0.436	-0.379	-1.932	-0.157	2.947	-0.030	-0.628	-1.378	...	0.294	1.498	0.421	-0.166	0.812	-0.034	0.211	0
4	1.326	-0.558	-0.063	-0.755	-0.407	-0.092	-0.395	-0.027	-1.090	0.681	...	0.694	-0.212	-0.227	0.743	-0.037	0.003	-0.008	0
5	-1.675	-0.777	1.447	0.725	-1.128	-0.094	-0.334	0.628	-1.221	0.214	...	0.296	0.399	0.374	-0.157	-0.173	0.292	0.029	0
6	1.143	-0.414	1.019	-0.673	-0.955	0.008	-0.874	0.204	2.935	-1.313	...	0.198	-0.009	-0.035	0.350	-0.712	0.074	0.016	0
7	-1.928	1.126	-4.518	1.749	-1.566	-2.010	-0.883	0.697	-2.065	-5.588	...	-0.319	0.639	-0.295	0.538	0.788	0.293	0.148	1
8	-0.724	0.808	1.650	-0.320	0.469	-1.199	0.693	-0.103	-0.583	-0.997	...	-0.836	-0.109	0.339	0.320	0.012	-0.081	-0.010	0
9	0.007	1.210	0.377	1.317	0.114	0.451	-0.879	-2.655	-0.334	-1.345	...	-0.421	-0.138	-0.560	0.913	-0.344	0.140	0.268	0

10 rows × 31 columns

Si les variables selectionnées sont trop corrélées entre elles, cela peut fausser notre modèle de regressions nous allons donc etudier les correlations entre les variables selectionnées et en supprimer en cas de corrélation trop importante. On appelle cela un problème de multicolinéarité dans la regression. Néanmoins étant donné qu'on a des variables issu de l'AFC nous savons que chaque composante principale est indépendante et possède une information propre à elle. Il n'y a donc pas de suppression necessaire.

```
In [35]: nbr_liste = sous_ensemble.Class.value_counts()
fig, ax = plt.subplots(figsize=(14,8))
sns.barplot(x=nbr_liste.index,y=nbr_liste)
plt.title("Nombre de Transaction frauduleuse vs Transaction non frauduleuse dans le sous-ensemble", size = 15)
plt.ylabel("Nombre de transactions", size = 14)
plt.xlabel("0:non frauduleuse, 1:frauduleuse", size=14)
```

```
Text(0.5, 0, '0:non frauduleuse, 1:frauduleuse')
```



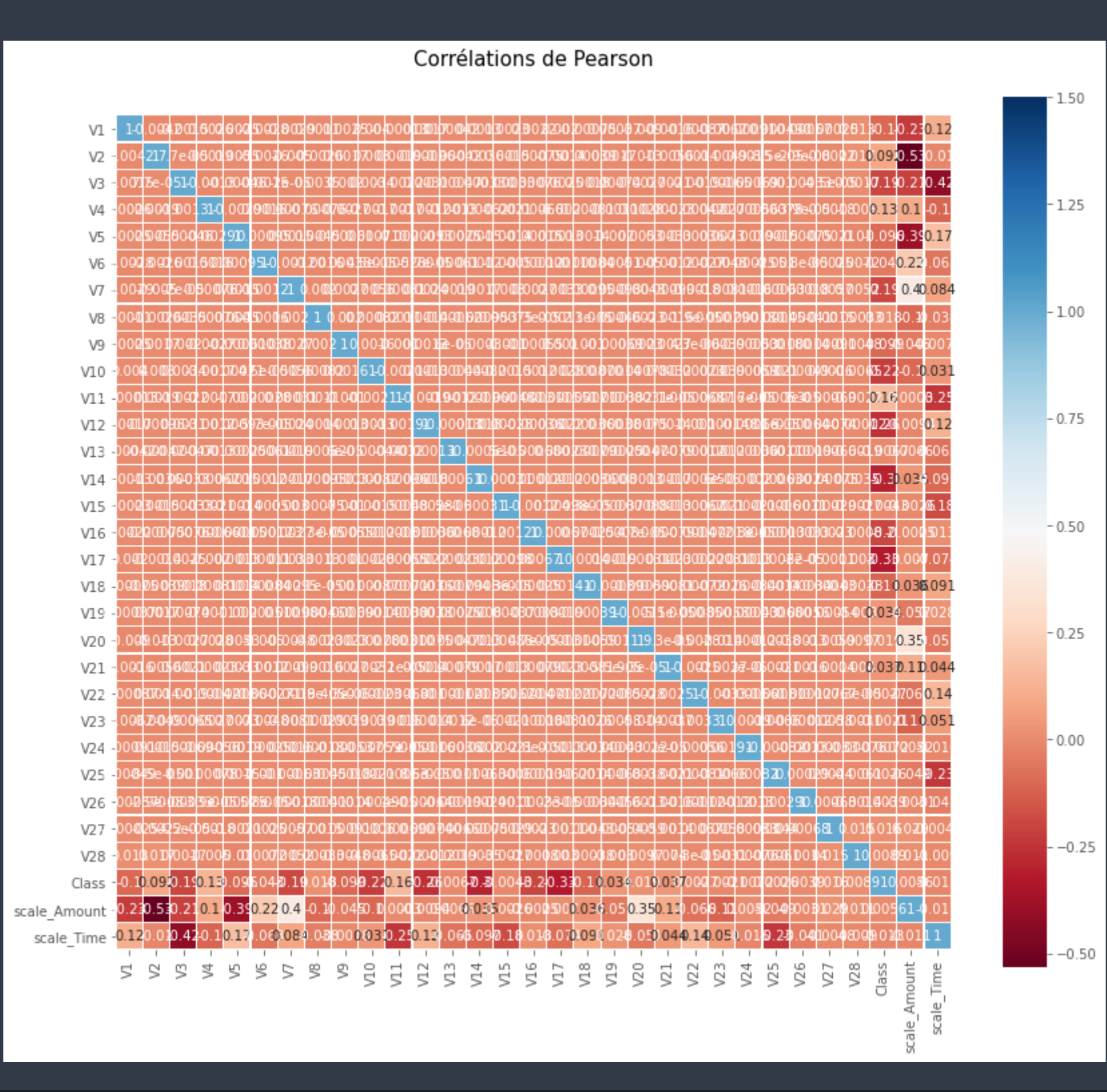
Une fois de plus on peut regarder la corrélation entre les variables, mais cette fois ci dans le sous ensemble.

```
In [36]: coor = sous_ensemble.corr()
coor = coor[['Class']]
coor
```

	Class
V1	-0.426
V2	0.489
V3	-0.562
V4	0.697
V5	-0.390
V6	-0.388
V7	-0.476
V8	0.049
V9	-0.552
V10	-0.615
V11	0.679
V12	-0.673
V13	-0.092
V14	-0.739
V15	-0.050
V16	-0.600
V17	-0.556
V18	-0.463
V19	0.252
V20	0.140
V21	0.117
V22	0.049
V23	-0.034
V24	-0.076
V25	0.050
V26	0.067
V27	0.073
V28	0.073
Class	1.000
scale_Amount	0.047
scale_Time	-0.152

```
In [64]: colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Corrélations de Pearson', y=1.05, size=15)
sns.heatmap(train.astype(float).corr(),linewidths=0.1,vmax=1.5, square=True, cmap=colormap,
 linecolor='white', annot=True)
```

```
<AxesSubplot:title={'center':'Corrélations de Pearson'}>
```

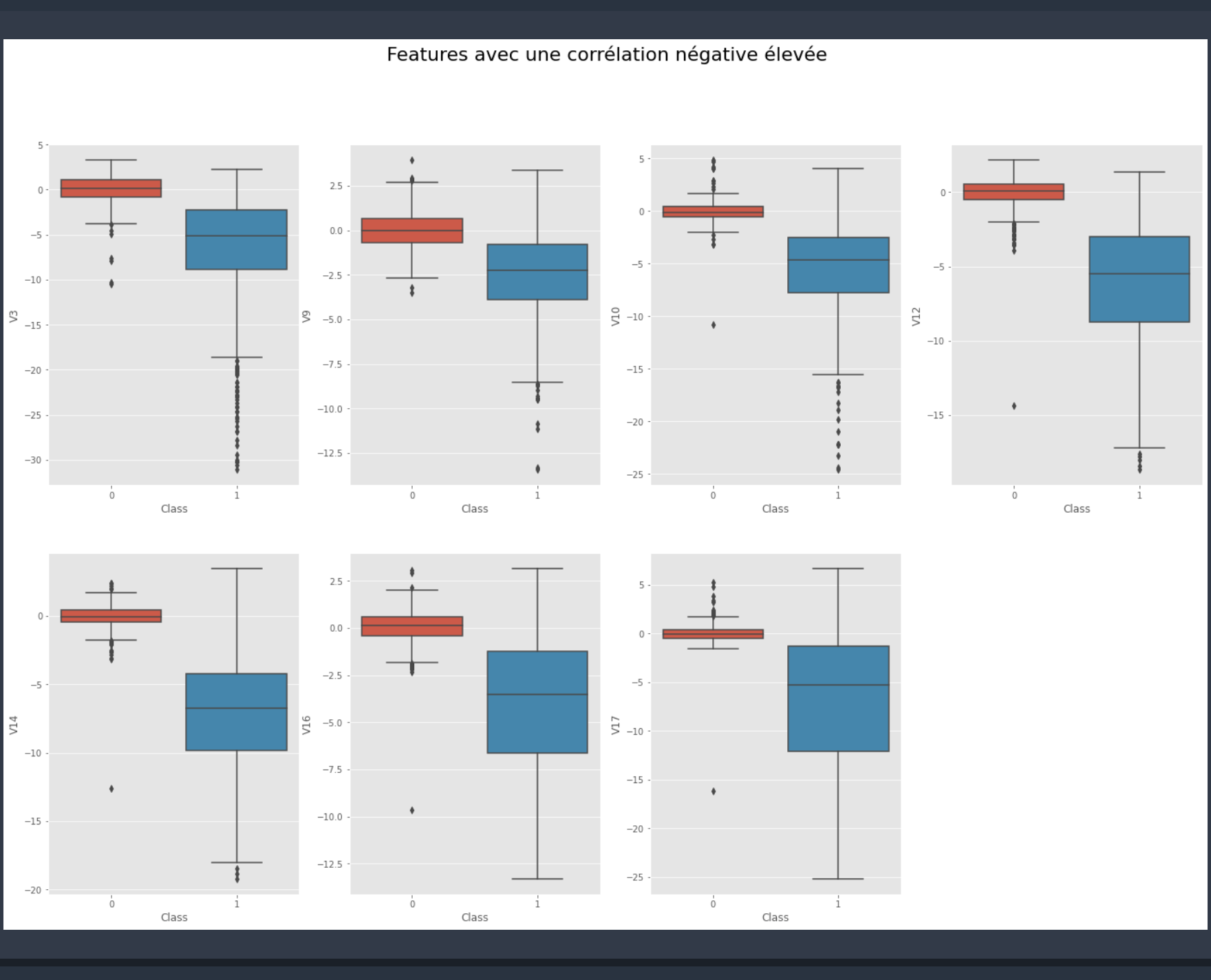


On observe ainsi une forte corrélation entre "scale_Time" et "V3" et de même une forte corrrélation entre "scale_Amount" et "V2". Ce qui est cohérant en regardant la carte thermique établi précedemment.

```
In [38]: # Corrélation négative plus petit que -0.5
coor[coor.Class < -0.5]
```

	Class
V3	-0.562
V9	-0.552
V10	-0.615
V12	-0.673
V14	-0.739
V16	-0.600
V17	-0.556

```
In [39]: # On peut tracer les boxplots pour les corrélations négatives strictement inférieure à -0.5
fig, ax = plt.subplots(nrows=2,ncols=4,figsize=(24,16))
fig.suptitle('Features avec une corrélation négative élevée', size=22)
sns.boxplot(x="Class",y="V3",data=sous_ensemble, ax=ax[0,0])
sns.boxplot(x="Class",y="V9",data=sous_ensemble, ax=ax[0,1])
sns.boxplot(x="Class",y="V10",data=sous_ensemble, ax=ax[0,2])
sns.boxplot(x="Class",y="V12",data=sous_ensemble, ax=ax[0,3])
sns.boxplot(x="Class",y="V14",data=sous_ensemble, ax=ax[1,0])
sns.boxplot(x="Class",y="V16",data=sous_ensemble, ax=ax[1,1])
sns.boxplot(x="Class",y="V17",data=sous_ensemble, ax=ax[1,2])
fig.delaxes(ax[1,3])
```

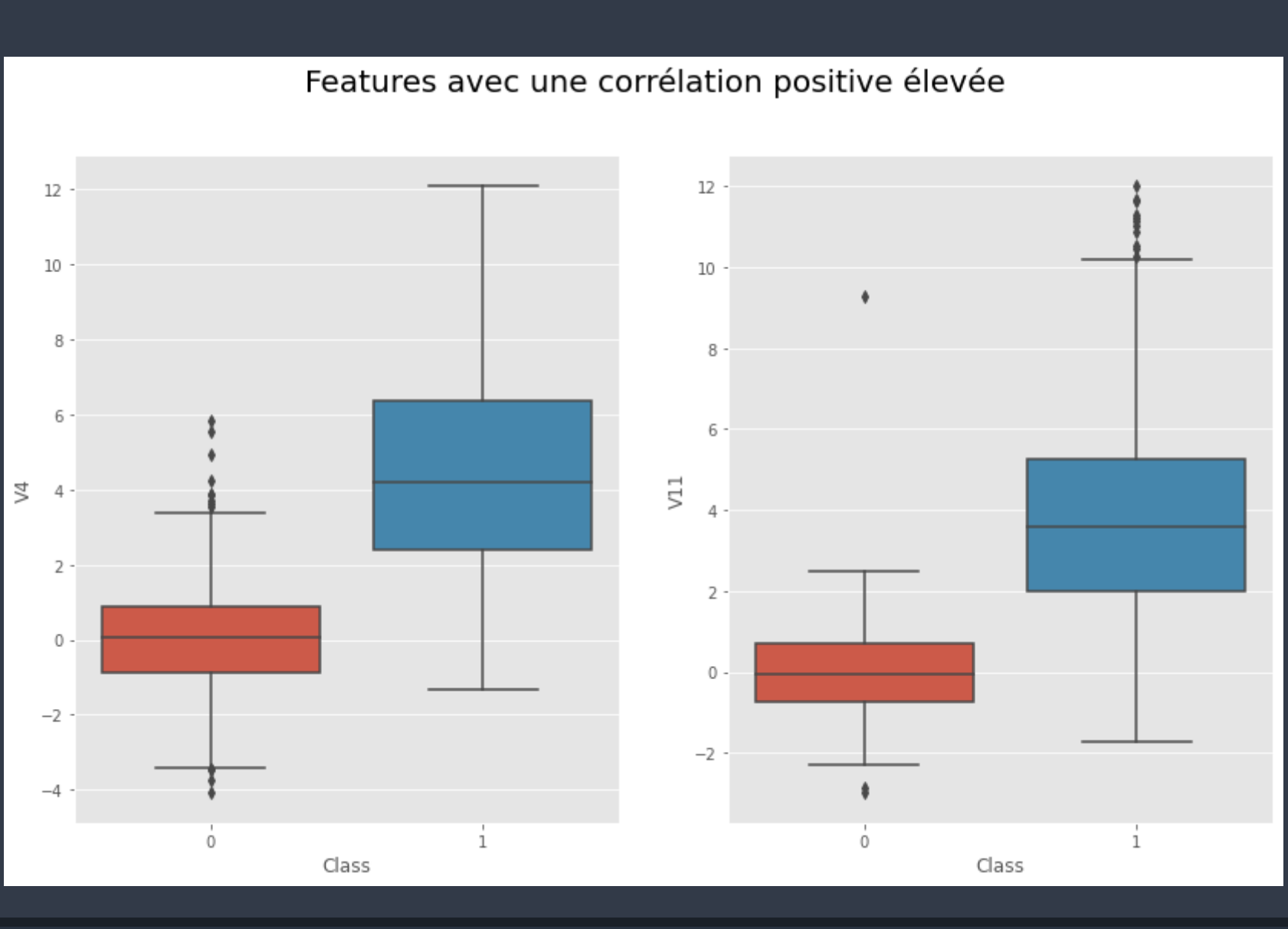



```
In [40]: # Corrélation positive plus grand que 0.5
coor[coor.Class > 0.5]
```

	Class
V4	0.697
V11	0.679
Class	1.000

```
In [41]: # On peut tracer les boxplot pour les corrélations positive supérieur à 0.5
fig, ax = plt.subplots(nrows=1, ncols=2,figsize = (14,8))
fig.suptitle('Features avec une corrélation positive élevée', size=20)
sns.boxplot(x="Class",y="V4",data=sous_ensemble, ax=ax[0])
sns.boxplot(x="Class",y="V11",data=sous_ensemble, ax=ax[1])
```

```
<AxesSubplot:xlabel='Class', ylabel='V11'>
```




```
In [42]: #On va supprimer que les cas extrêmes
q1 = sous_ensemble.quantile(0.25)
q2 = sous_ensemble.quantile(0.75)
IQR = q2-q1
# On peut maintenat définir notre nouveau data-frame noté df2
df2 = sous_ensemble[~((sous_ensemble < (q1 - 2.5 * IQR)) |(sous_ensemble > (q2 + 2.5 * IQR))).any(axis=1)]
```

```
In [43]: df2.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V22	V23	V24	V25	V26	V27	V28	Class
4	1.326	-0.558	-0.063	-0.755	-0.407	-0.092	-0.395	-0.027	-1.090	0.681	...	0.694	-0.212	-0.227	0.743	-0.037	0.003	-0.008	0
5	-1.675	-0.777	1.447	0.725	-1.128	-0.094	-0.334	0.628	-1.221	0.214	...	0.296	0.399	0.374	-0.157	-0.173	0.292	0.029	0
6	1.143	-0.414	1.019	-0.673	-0.955	0.008	-0.874	0.204	2.935	-1.313	...	0.198	-0.009	-0.035	0.350	-0.712	0.074	0.016	0
8	-0.724	0.808	1.650	-0.320	0.469	-1.199	0.693	-0.103	-0.583	-0.997	...	-0.836	-0.109	0.339	0.320	0.012	-0.081	-0.010	0
9	0.007	1.210	0.377	1.317	0.114	0.451	-0.879	-2.655	-0.334	-1.345	...	-0.421	-0.138	-0.560	0.913	-0.344	0.140	0.268	0

5 rows × 31 columns

```
In [44]: print("Nous avons réduit la taille de nos données de {} transactions à {} transactions, soit {} transactions de moins.".format(len(sous_ensemble),len(sous_ensemble)-len(df2),len(df2)))
```

```
Nous avons réduit la taille de nos données de 888 transactions à 268 transactions, soit 620 transactions de moins.
```

A présent on peut réduire la dimension.

```
In [45]: from sklearn.manifold import TSNE

X = df2.drop('Class', axis=1)
y = df2['Class']
```

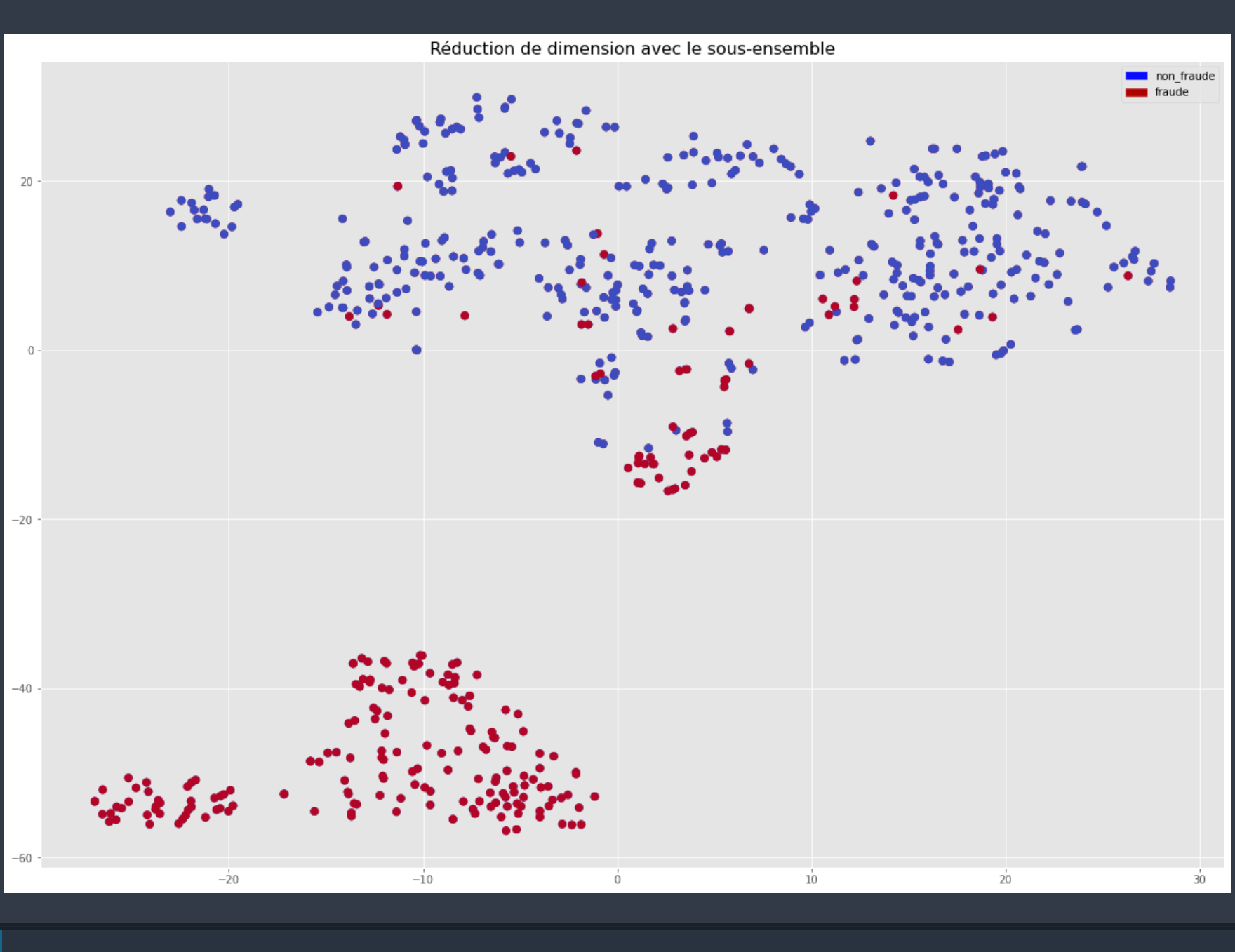
```
In [46]: #t-SNE
X_reduced_tsne = TSNE(n_components=2, random_state=42).fit_transform(X.values)
X_reduced_tsne
```

```
array([[ -6.2814803,  22.152905 ],
       [ 27.350624 ,   8.189682 ],
       [  3.4269648,  23.088364 ],
       ...,
       [  6.7797594,  -1.5833898],
       [  1.4590074,  20.208658 ],
       [-14.155357 ,   5.0237617]], dtype=float32)
```

On peut afficher le nuage de point de la réduction de dimension

```
In [47]: import matplotlib.patches as mpatches

fig, ax = plt.subplots(figsize = (20,16))
blue_patch = mpatches.Patch(color='#0A0AFF', label='non_fraude')
red_patch = mpatches.Patch(color='#AF0000', label='fraude')
ax.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 0),cmap='coolwarm',label='non_fraude', linewidths=2)
ax.scatter(X_reduced_tsne[:,0], X_reduced_tsne[:,1], c=(y == 1),cmap='coolwarm', label='fraude', linewidths=2)
ax.set_title('Réduction de dimension avec le sous-ensemble', fontsize=16)
ax.grid(True)
ax.legend(handles=[blue_patch, red_patch])
```

```
<matplotlib.legend.Legend at 0x28e8491ee48>
```



3. Algorithme de Classification

```
In [48]: # D'abord on commence par diviser le sous ensemble en train et test pour appliquer les algorithme
# de classification
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
In [49]: from sklearn.model_selection import cross_val_score
# Cross Validation to test and anticipate overfitting problem
def crossvalidate(clf, X,y):
    precision = cross_val_score(clf, X, y, cv=10, scoring='precision')
    recall = cross_val_score(clf, X, y, cv=10, scoring='recall')
    f1 = cross_val_score(clf, X, y, cv=10, scoring='f1')
    roc_auc = cross_val_score(clf, X, y, cv=10, scoring='roc_auc')
    accuracy = cross_val_score(clf, X, y, cv=10, scoring='accuracy')
    # The mean score and standard deviation of the score estimate
    print("Cross Validation Precision: %0.2f (+/- %0.2f)" % (precision.mean(), precision.std()))
    print("Cross Validation Recall: %0.2f (+/- %0.2f)" % (recall.mean(), recall.std()))
    print("Cross Validation f1 score: %0.2f (+/- %0.2f)" % (f1.mean(), f1.std()))
    print("Cross Validation roc_auc: %0.2f (+/- %0.2f)" % (roc_auc.mean(), roc_auc.std()))
    print("Cross Validation Accuracy: %0.2f (+/- %0.2f)" % (accuracy.mean(), accuracy.std()))
```

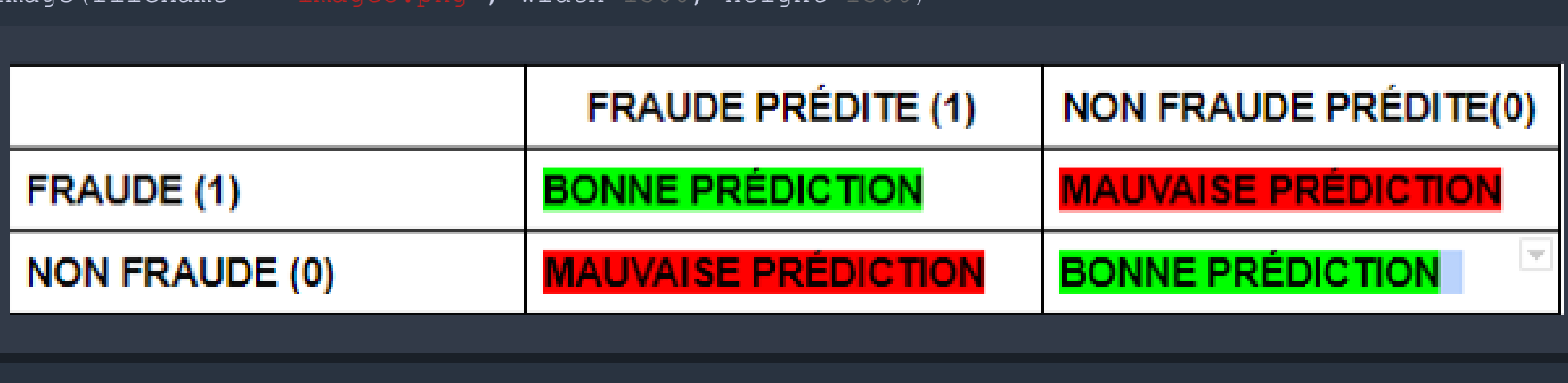
```
In [50]: # Import module
from sklearn.metrics import confusion_matrix
def plot_confusion_matrix(clf, X, y):
  # Predict the test data using the best model
  y_pred = clf.predict(X)
  # Create the confusion matrix
  confmat = confusion_matrix(y_true=y, y_pred=y_pred)
  # Plot the confusion matrix in a fancy way
  fig, ax = plt.subplots(figsize=(5, 5))
  ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
  for i in range(confmat.shape[0]):
      for j in range(confmat.shape[1]):
          ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')

  plt.xlabel('predicted label')
  plt.ylabel('true label')

  plt.tight_layout()
  plt.show()
```

```
In [51]: %%latex
On appelle matrice de confusion, la matrice définit par :
    \begin{align}
    MC := \begin{pmatrix}
    \displaystyle \sum_{1=1}^n \mathbb{1}_{\{y_i = \tilde{y}_i = 0 \}} & \displaystyle \sum_{1=1}^n \mathbb{1}_{\{y_i = 0 \} \cap \{\tilde{y}_i = 1 \}} \\
    \displaystyle \sum_{1=1}^n \mathbb{1}_{\{y_i = 1 \} \cap \{\tilde{y}_i = 0 \}} & \displaystyle \sum_{1=1}^n \mathbb{1}_{\{y_i = \tilde{y}_i = 1 \}}
\end{pmatrix} \\
      &= \begin{pmatrix}
    Nombre \:de \:0 \: prédit, \:0 \: en \: réalité & Nombre\: de \:1 \:prédit, \:0 \:en \:réalité \\
    Nombre \:de \:0 \:prédit, \:1 \:en \:réalité & Nombre \:de \:1 \:prédit, \:1 \:en \:réalité
\end{pmatrix}
\end{align}
```

On appelle matrice de confusion, la matrice définit par :

$$MC = \begin{pmatrix} \displaystyle\sum_{1=1}^n 1_{\{y_i=\tilde{y}_i=0\}} & \displaystyle\sum_{1=1}^n 1_{\{y_i=0\}\cap\{\tilde{y}_i=1\}} \\ \displaystyle\sum_{1=1}^n 1_{\{y_i=1\}\cap\{\tilde{y}_i=0\}} & \displaystyle\sum_{1=1}^n 1_{\{y_i=\tilde{y}_i=1\}} \end{pmatrix} = \begin{pmatrix} \text{Nombre de 0 prédit, 0 en réalité} & \text{Nombre de 1 prédit, 0 en réalité} \\ \text{Nombre de 0 prédit, 1 en réalité} & \text{Nombre de 1 prédit, 1 en réalité} \end{pmatrix}$$

```
In [52]: from IPython.display import Image
Image(filename = "image3.png", width=1300, height=1300)
```



	FRAUDE PRÉDITE (1)	NON FRAUDE PRÉDITE(0)
FRAUDE (1)	BONNE PRÉDICTION	MAUVAISE PRÉDICTION
NON FRAUDE (0)	MAUVAISE PRÉDICTION	BONNE PRÉDICTION

```
In [53]: # Import Modules
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.decomposition import TruncatedSVD

# Create a default model
pipe_rf = Pipeline([('svd', TruncatedSVD(n_components=2)),
                    ('clf', RandomForestClassifier(n_estimators=10))])

# Create a list of choices for each hyperparameter
param_range_svd = [2, 5, 10, 15, 20] # dimension of the data after dimensionality reduction

param_range_clf = [5, 10, 20, 30] # number of trees in RandomForest

# Create the grid of possible combinations
param_grid = [{'svd__n_components': param_range_svd,
               'clf__n_estimators': param_range_clf,
               'clf__max_features': param_range_clf}]

# Prepare the Grid Search Fine Tuning
gs = GridSearchCV(estimator=pipe_rf, # the model
                  param_grid=param_grid, # its hyperparameters possibilities
                  scoring='precision', # precision is the evaluation metric
                  cv=5, # We use 5 folds in cross validation
                  n_jobs=-1)

# Test all the possible hyperparamters
gs = gs.fit(X_train, y_train)

# The best hyperparameters
print("The optimal precision is {}".format(gs.best_score_))
print("The optimal hyperparameters are {}".format(gs.best_params_))
```

```
The optimal precision is 0.9785185185185185
The optimal hyperparameters are {'clf__max_features': 5, 'clf__n_estimators': 10, 'svd__n_components': 10}
```

```
In [54]: # Optimal model
best_rf_svd = gs.best_estimator_
# Evaluation performances
crossvalidate(best_rf_svd, X_train, y_train)
# Confusion Matrix
plot_confusion_matrix(best_rf_svd, X_test, y_test)
```

```
Cross Validation Precision: 0.95 (+/- 0.05)
Cross Validation Recall: 0.86 (+/- 0.07)
Cross Validation f1 score: 0.91 (+/- 0.03)
Cross Validation roc_auc: 0.95 (+/- 0.03)
Cross Validation Accuracy: 0.93 (+/- 0.03)
```

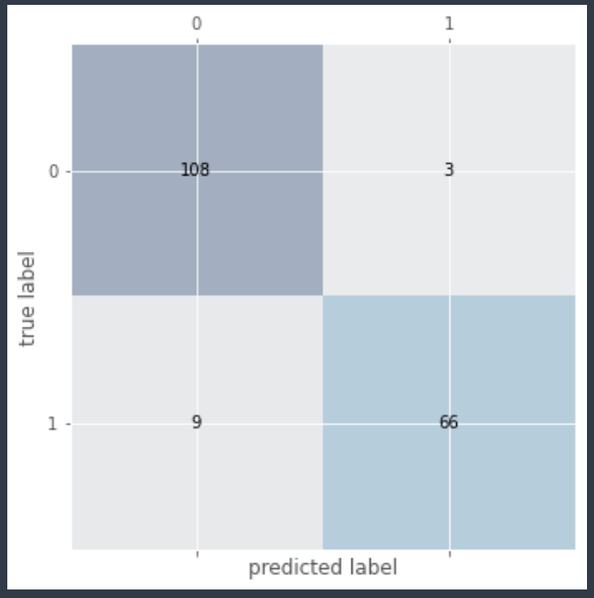
On constate qu'on a seulement 6 et 8 éléments qui sont mal étiquetés par le classificateur contre 123 et 54 élément diagonaux qui sont bien étiquetés ce qui est suffisamment grand par-raport à nos données du sous-ensemble. donc on peut dire que notre paramètre de régularisation est bon et peut-être meilleure.

In [55]:
```
# Import Modules
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV


# Create a default model
pipe_lr = Pipeline([('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(penalty='l2',
                                               random_state=1,
                                               solver='lbfgs'))])

# Create a list of choices for each hyperparameter
param_range_pca = [2, 5, 10, 5, 20] # dimension of the data after dimensionality reduction
param_clf = ['l1', 'l2']

# Create the grid of possible combinations
param_grid = [{'pca__n_components': param_range_pca,
               'clf__penalty': param_clf}]

# Prepare the Grid Search Fine Tuning
gs = GridSearchCV(estimator=pipe_lr, # the model
                  param_grid=param_grid, # its hyperparameters possibilities
                  scoring='precision', # precision is the evaluation metric
                  cv=5, # We use 5 folds in cross validation
                  n_jobs=-1)

# Test all the possible hyperparamters
gs = gs.fit(X_train, y_train)

# The best hyperparameters
print("The optimal precision is {}".format(gs.best_score_))
print("The optimal hyperparameters are {}".format(gs.best_params_))
```

```
The optimal precision is 0.9861409796893668
The optimal hyperparameters are {'clf__penalty': 'l2', 'pca__n_components': 10}
```

In [56]:
```
# Optimal model
best_lr_svd = gs.best_estimator_
# Evaluation performances
crossvalidate(best_lr_svd, X_train, y_train)
# Confusion Matrix
plot_confusion_matrix(best_lr_svd, X_test, y_test)
```

```
Cross Validation Precision: 0.98 (+/- 0.03)
Cross Validation Recall: 0.86 (+/- 0.08)
Cross Validation f1 score: 0.91 (+/- 0.04)
Cross Validation roc_auc: 0.96 (+/- 0.03)
Cross Validation Accuracy: 0.94 (+/- 0.03)
```

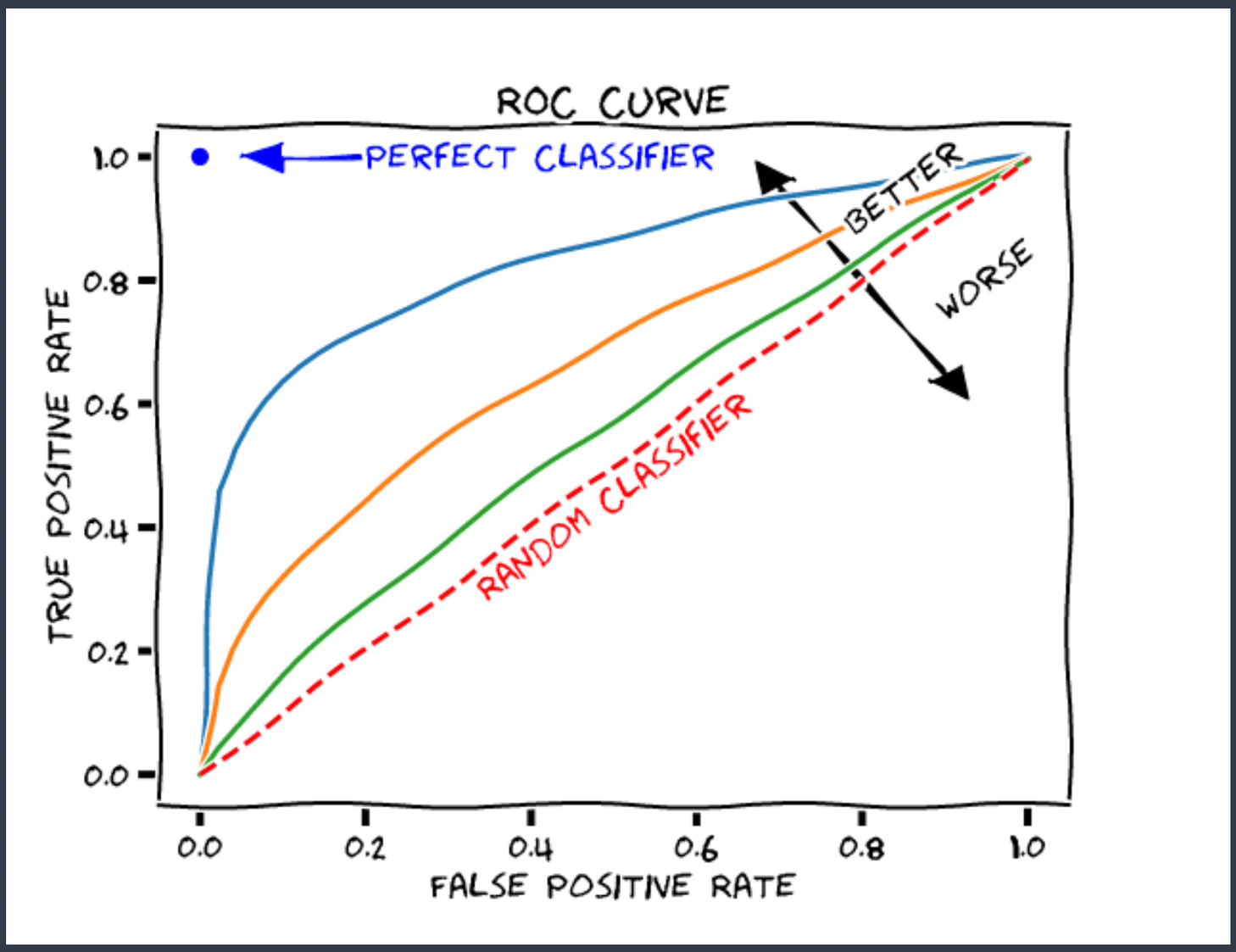


Ici, on constate que les éléments hors diagaunaux sont réduits par-rapport à la matrice de confusion précédente. Donc le paramètre de régularisation est mieux qu'avant. Ainsi notre classifieur est meilleure que celui d'avant.

La courbe ROC est une courbe qui nous permettra de determiner le seuil de décision adéquat pour notre modèle. Chaque point de la courbe represente le ratio sensibilité / specificité pour un seul de décision spécifique. L'aire en dessous de la courbe ROC et nommé AUC et est comprise entre 0 et 1. L'AUC donne une mesure de la capacité du modèle à discriminer les cas positifs des cas négatifs.

Le schema ci dessous nous montre quel est la forme idéal et standard de la courbe.

In [57]:
```
from IPython.display import Image
Image(filename = "image2.png", width=1300, height=1300)
```



In [58]:
```python
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

In [59]:
```python
#On peut créer les models par défauts
modeles = []
modeles.append(('LDA', LinearDiscriminantAnalysis()))
modeles.append(('KNN', KNeighborsClassifier()))
modeles.append(('CART', DecisionTreeClassifier()))
modeles.append(('SVM', SVC()))
modeles.append(('RF', RandomForestClassifier()))
modeles.append(('LR', LogisticRegression()))
```

In [60]:
```python
modeles
```

```
[('LDA', LinearDiscriminantAnalysis()),
 ('KNN', KNeighborsClassifier()),
 ('CART', DecisionTreeClassifier()),
 ('SVM', SVC()),
 ('RF', RandomForestClassifier()),
 ('LR', LogisticRegression())]
```

In [61]:
```python
# On peut tester les modèles
resultats = []
list_modeles = []
for i, modele in modeles:
    kfold = KFold(n_splits=10, random_state=42)
    cv_resultats = cross_val_score(modele, X_train, y_train, cv=kfold, scoring='roc_auc')
    resultats.append(cv_resultats)
    list_modeles.append(i)
    message = '%s: %f (%f)' % (i, cv_resultats.mean(), cv_resultats.std())
    print(message)
```

```
C:\Users\mouss\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no e
ffect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuf
fle=True.
  FutureWarning
C:\Users\mouss\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no e
ffect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuf
fle=True.
  FutureWarning


LDA: 0.964921 (0.029941)
KNN: 0.957825 (0.036498)


C:\Users\mouss\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no e
ffect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuf
fle=True.
  FutureWarning


CART: 0.891568 (0.049818)
SVM: 0.964193 (0.031255)


C:\Users\mouss\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no e
ffect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuf
fle=True.
  FutureWarning
C:\Users\mouss\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no e
ffect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuf
fle=True.
```

```
  FutureWarning


RF: 0.971546 (0.027922)


C:\Users\mouss\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:297: FutureWarning: Setting a random_state has no e
ffect since shuffle is False. This will raise an error in 0.24. You should leave random_state to its default (None), or set shuf
fle=True.
  FutureWarning


LR: 0.963628 (0.029673)
```

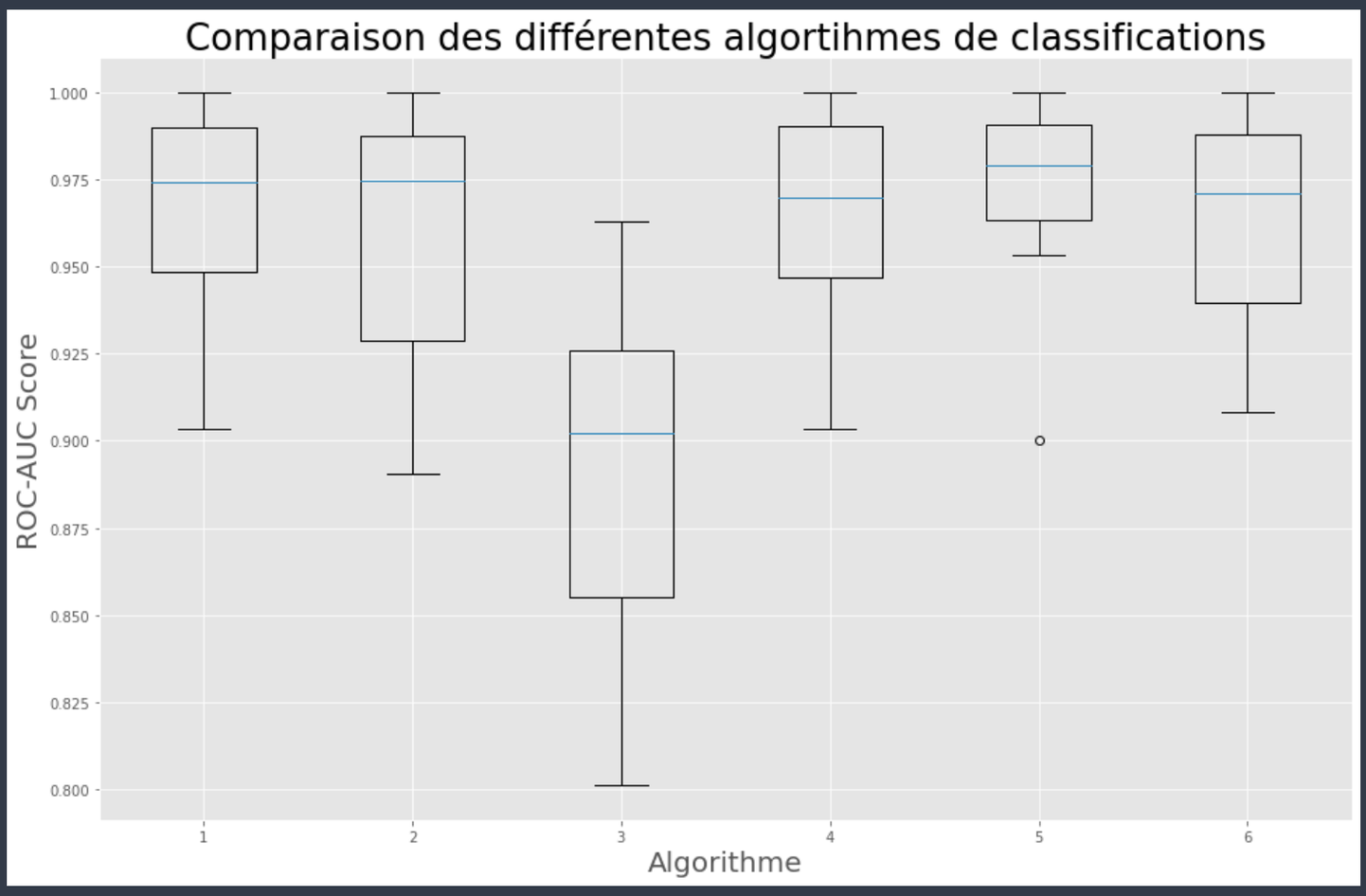
In [ ]:

On ainsi comparer les algorithmes de classifications en faisant un boxplot des modèles

In [62]:

```
fig = plt.subplots(figsize = (16,10))
plt.title("Comparaison des différentes algortihmes de classifications", size=26)
ax.set_xticklabels(list_modeles)
plt.boxplot(resultats)
plt.xlabel("Algorithme", size=20)
plt.ylabel("ROC-AUC Score", size =20)
```

```
C:\Users\mouss\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: UserWarning: FixedFormatter should only be used together wit
h FixedLocator
  This is separate from the ipykernel package so we can avoid doing imports until


Text(0, 0.5, 'ROC-AUC Score')
```



On remarque le meilleur classifieur est RandomForestClassifier.

In [ ]: