Projet de blockchain ENSIBS Cyberdata S8

- Alexendre Lenfantin
- Axel Bacrie
- Mehdi Khalil

Enoncé

- **Projet** : Explorer un Ensemble de Données pour Détecter les Fraudes en Ethereum puis Développer une Application WEB3 Décentralisée (DApp)
- **Groups** : 3 élèves max.
- **Présentation** : Le dernier TD.
- **Durée** : 5 Séance de TDs.
- **Eléments attendus** : Présenter les réponses aux deux premières questions puis le code et le fonctionnement de l'application.

Première partie

Les deux questions suivantes concernent l'ensemble de donnée (Ethereum Fraud Detection Dataset https://www.kaggle.com/datasets/vagifa/ethereum-frauddetection-dataset):

- **Question 1** : Quelles sont les caractéristiques utilisées pour détecter les fraudes en Ethereum ?
- **Question 2** : Implémenter une technique pour supprimer les caractéristiques corrélées.

Les éléments de réponce sont donnés en annexe à la fin de ce document.

Deuxième partie

Implémenter une application DAPP, comme: ToDo list, voting... Les liens suivants présent des idées:

- Créer une DApp de A à Z | Foundry, NextJS, Wagmi, Viem, Typescript: https://www.youtube.com/watch?v=pTqYEretjs4&t=5127s
- Une application de vote-Ethereum for web developers, https://medium.com/@mvmurthy/ethereum-for-web-developers-890be23d1d0c
- ToDo List: How to Build Ethereum Dapp with React.js· Complete Step-By-Step Guide, https://www.dappuniversity.com/articles/ethereum-dapp-react-tutorial

Les étapes de développement:

1. Développer le contrat intelligent, ce sera le backend de l'application.

2. Déployer le contrat sur un nœud local.
3. Développer le frontend de l'application.

Notre projet : Une application de signature de documents

Un utilisateur doit pouvoir ajouter un document à signer, signer un document, vérifier qui à signé le document.
 Un document est représenté par son hash (fait dans le front).

Methode smart contract

- Ajout de document (hash avec id)
- Verification du document
- Signer un document
- Verification des signataires

Elements d'interface

On souhaite réaliser une application one page qui se découpe en plusieur pages :

- Déposer un document.
- Récupérer son hash.
- Ajouter le document à la blockchain pour être signé.
- Vérifier si le document est présent sur la blockchain.
- Vérifier si un wallet a signé un document.

Lancer le projet

Le projet nécessite nodejs, et ganache-cli.

Vous pouvez trouver le code dans le repo github : https://github.com/noxxou/blockchain

1. Compiler le smart contract :

```
npm run compile
```

1. Lancer ganache-cli :

```
ganache-cli
```

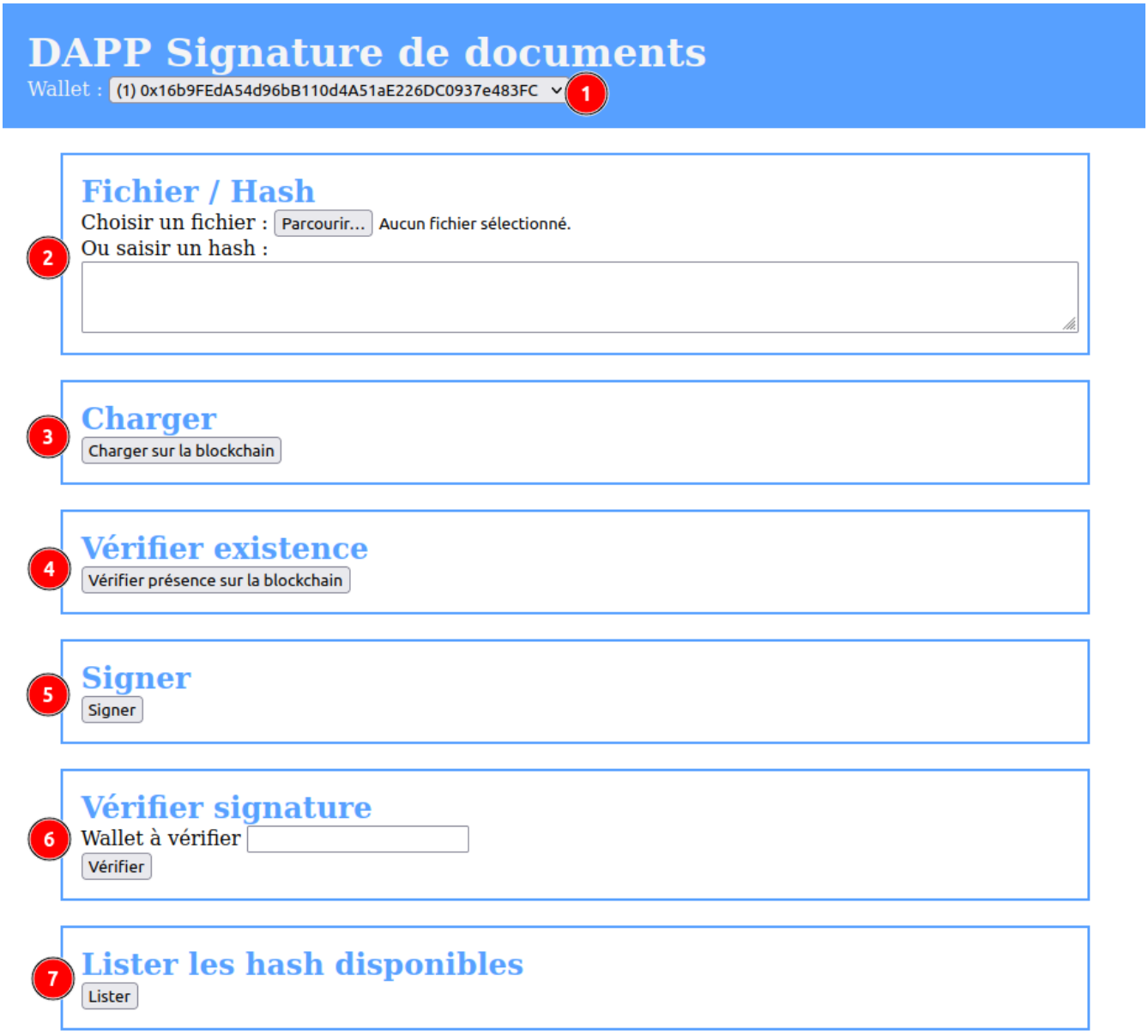
1. Initialiser l'application :

```
npm run startapp
```

1. Puis ouvrir le fichier app.html dans le navigateur.

L'application

Capture d'écran de l'application.



On retrouve différents éléments qui permettent d'utiliser les actions cités plu haut.

1. Un menu déroulant permettant de sélectionner le wallet avec lequel on travaille. Cela n'est possible que dans ce contexte où nous utilisons ganache-cli.
2. **Fichier** / **Hash** : permet de déposer un fichier (son hash directement donné en-dessous) ou de directement donner un hash. Ce hash sera pris en compte pour toutes les prochaines fonctionalitées.
3. **Charger** : permet de poster le hash sur la blockchain pour pouvoir le signer.

4. **Vérifier** : permet de vérifier si le hash à déjà été importé.
5. **Signer** : signe le hash avec le wallet actuel.
6. **Vérifier signature** : En donnant une adresse de wallet au format `0x···.`, on peut vérifier su cette adresse a signé le fichier.
7. **Lister** : permet de lister les hash sur la block chaine et indique le nombre de signataire.

Annexe : Export du jupyter notebook au de réponce aux questions 1 & 2 de la partie 1

Question 1 : Quelles sont les caractéristiques utilisées pour détecter les fraudes en Ethereum ?

Les caractéristiques utilisées sont les informations concernants les montants et les temps de transactions en ether et avec des tokens ERC20 ainsi qu'un flag indiquant si la transaction est une fraude et une ligne index correspondant à nombre propre à chaque ligne:

- Address: l'adresse du wallet ethereum
- Index: l'index propre à chaque ligne
- FLAG: flag indiquant s'il s'agit d'une fraude
- Avg min between sent tnx: temps moyen entre chaque transactions envoyées par ce wallet, en minute
- Avg_min_between_received_tnx: temps moyen entre chaque transactions reçues par ce wallet, en minute
- Time_Diff_between_first_and_last(Mins): différence de temps entre la première et la dernière transaction
- Sent_tnx: Nombre total de transactions envoyées
- Received_tnx: Nombre total de transactions reçues
- Number_of_Created_Contracts: Nombre total de transactions de contrat créées
- Unique_Received_From_Addresses: Nombre total d'adresses uniques à partir desquelles le compte a reçu des transactions
- Unique_Sent_To_Addresses20: Nombre total d'adresses uniques à partir desquelles le compte a envoyé des transactions
- Min_Value_Received: Valeur minimale en Ether jamais reçue
- Max_Value_Received: Valeur maximale en Ether jamais reçue
- Avg_Value_Received: Valeur moyenne en Ether jamais reçue
- Min_Val_Sent: Valeur minimale d'Ether jamais envoyée
- Max_Val_Sent: Valeur maximale d'Ether jamais envoyée

- Avg_Val_Sent: Valeur moyenne d'Ether jamais envoyée
- Min_Value_Sent_To_Contract: Valeur minimale d'Ether envoyée à un contrat
- Max_Value_Sent_To_Contract: Valeur maximale d'Ether envoyée à un contrat
- Avg_Value_Sent_To_Contract: Valeur moyenne d'Ether envoyée aux contrats
- Total_Transactions(Including_Tnx_to_Create_Contract): Nombre total de transactions (y compris celles pour créer des contrats)
- Total_Ether_Sent: Total d'Ether envoyé pour l'adresse du compte
- Total_Ether_Received: Total d'Ether reçu pour l'adresse du compte
- Total_Ether_Sent_Contracts: Total d'Ether envoyé aux adresses de contrat
- Total_Ether_Balance: Solde total d'Ether après les transactions effectuées
- Total_ERC20_Tnxs: Nombre total de transactions de transfert de jetons ERC20
- ERC20_Total_Ether_Received: Total des transactions de jetons ERC20 reçues en Ether
- ERC20_Total_Ether_Sent: Total des transactions de jetons ERC20 envoyées en Ether
- ERC20_Total_Ether_Sent_Contract: Total des transactions de transfert de jetons ERC20 vers d'autres contrats en Ether
- ERC20_Uniq_Sent_Addr: Nombre de transactions de jetons ERC20 envoyées à des adresses de compte uniques
- ERC20_Uniq_Rec_Addr: Nombre de transactions de jetons ERC20 reçues à partir d'adresses uniques
- ERC20_Uniq_Rec_Contract_Addr: Nombre de transactions de jetons ERC20 reçues à partir d'adresses de contrat uniques
- ERC20_Avg_Time_Between_Sent_Tnx: Temps moyen entre les transactions de jetons ERC20 envoyées en minutes
- ERC20_Avg_Time_Between_Rec_Tnx: Temps moyen entre les transactions de jetons ERC20 reçues en minutes
- ERC20_Avg_Time_Between_Contract_Tnx: Temps moyen entre les transactions de jetons ERC20 envoyées
- ERC20_Min_Val_Rec: Valeur minimale en Ether reçue des transactions de jetons ERC20 pour le compte
- ERC20_Max_Val_Rec: Valeur maximale en Ether reçue des transactions de jetons ERC20 pour le compte

- ERC20_Avg_Val_Rec: Valeur moyenne en Ether reçue des transactions de jetons ERC20 pour le compte
- ERC20_Min_Val_Sent: Valeur minimale en Ether envoyée des transactions de jetons ERC20 pour le compte
- ERC20_Max_Val_Sent: Valeur maximale en Ether envoyée des transactions de jetons ERC20 pour le compte
- ERC20_Avg_Val_Sent: Valeur moyenne en Ether envoyée des transactions de jetons ERC20 pour le compte
- ERC20_Uniq_Sent_Token_Name: Nombre de jetons ERC20 uniques transférés
- ERC20_Uniq_Rec_Token_Name: Nombre de jetons ERC20 uniques reçus
- ERC20_Most_Sent_Token_Type: Jeton le plus envoyé pour le compte via la transaction ERC20
- ERC20_Most_Rec_Token_Type: Jeton le plus reçu pour le compte via les transactions ERC20

Question 2 : Implémenter une technique pour supprimer les caractéristiques corrélées

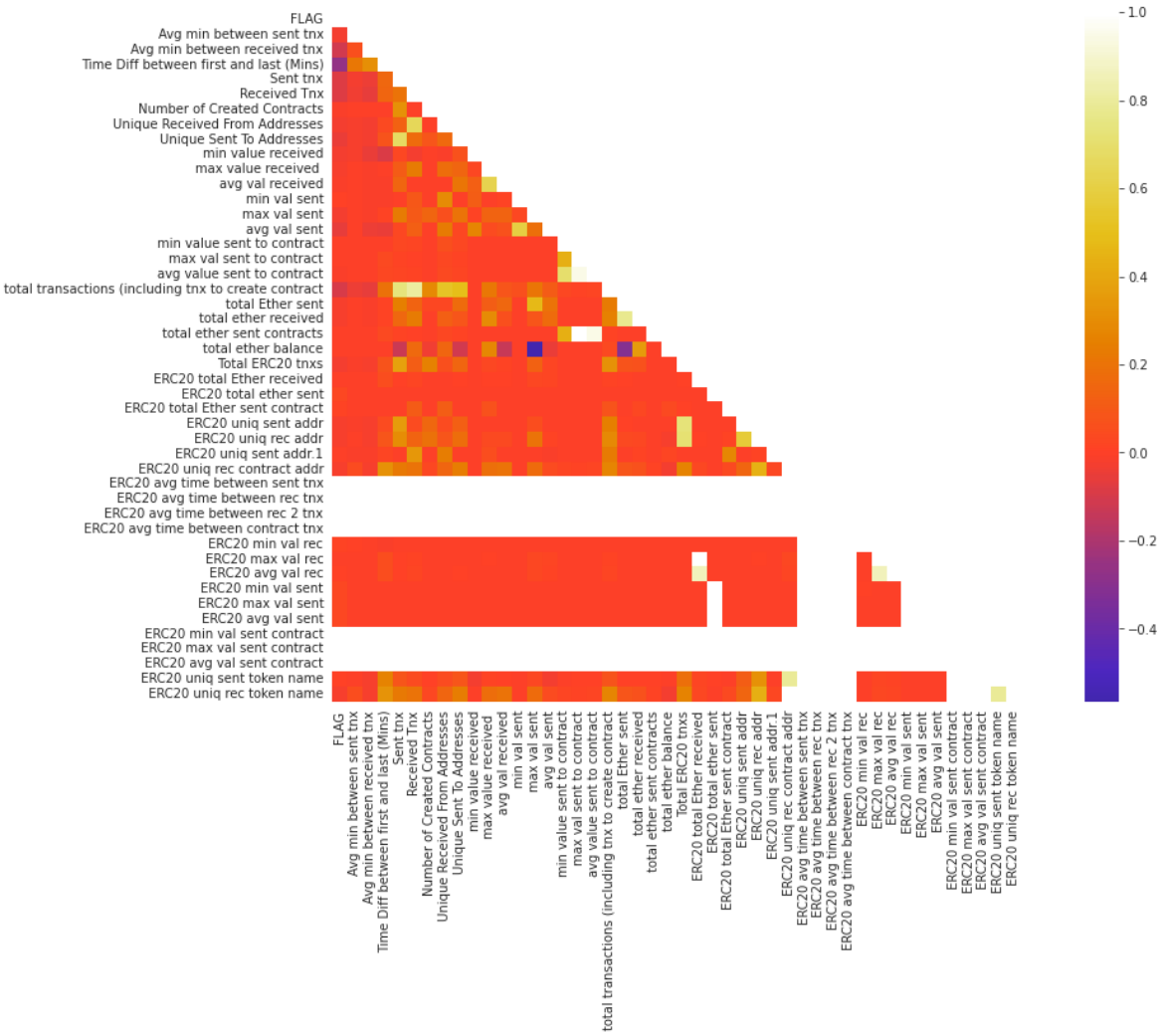
Basé sur le travail de CHITICARIU CRISTIAN

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [ ]:
```
df = pd.read_csv('transaction_dataset.csv', index_col=0)
print(df.shape)
df.head()

# supprimer les deux premières colonnes (Index, Adress)
df = df.iloc[:,2:]

#Récupération de la liste des objets sous forme de dataframe
categories = df.select_dtypes('O').columns.astype('category')
```

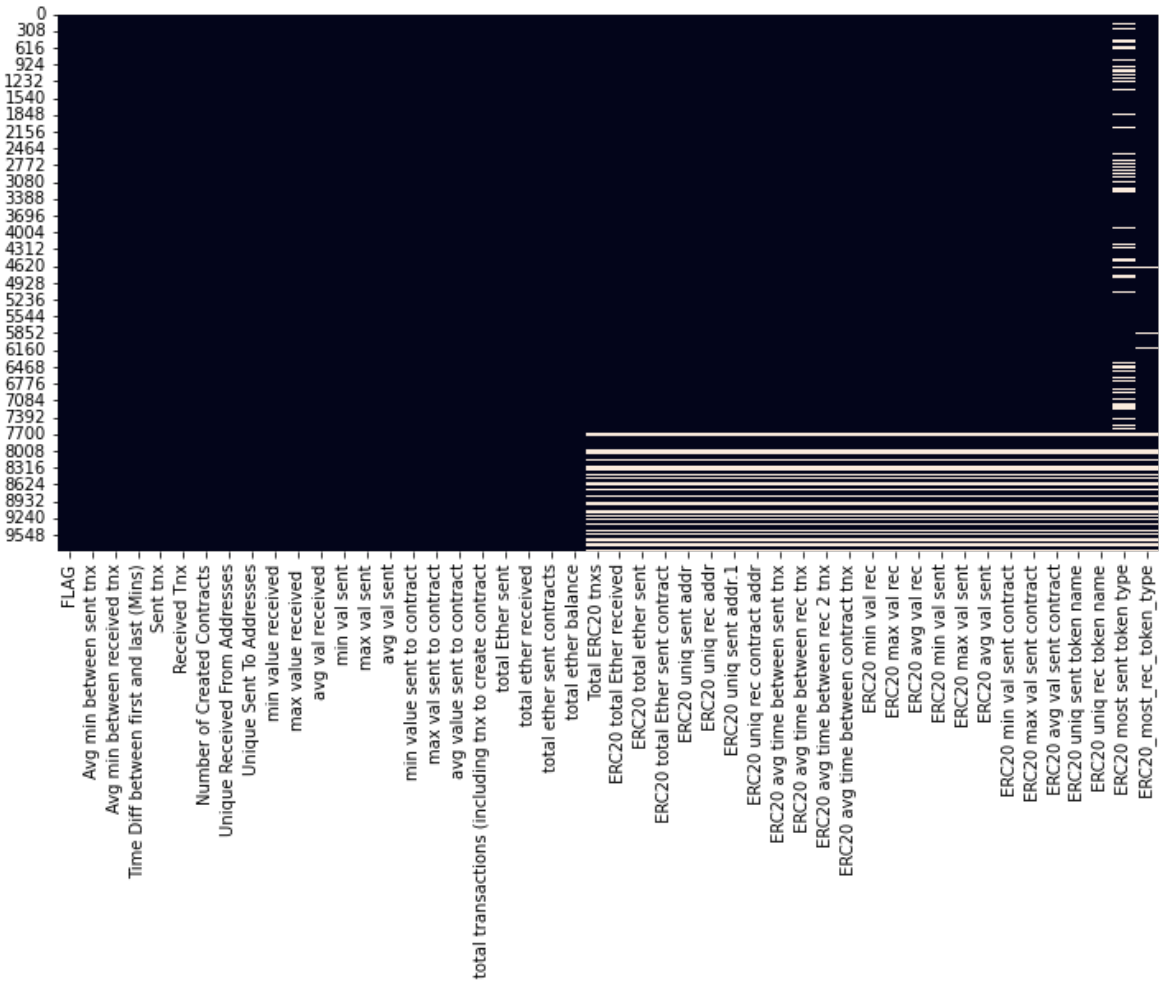
```
(9841, 50)
```

In [ ]:
```
numericals = df.select_dtypes(include=['float','int']).columns

corr = df.corr(numeric_only=True)

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)]=True
with sns.axes_style('white'):
    fig, ax = plt.subplots(figsize=(18,10))
    sns.heatmap(corr,  mask=mask, annot=False, cmap='CMRmap', center=0, s
```



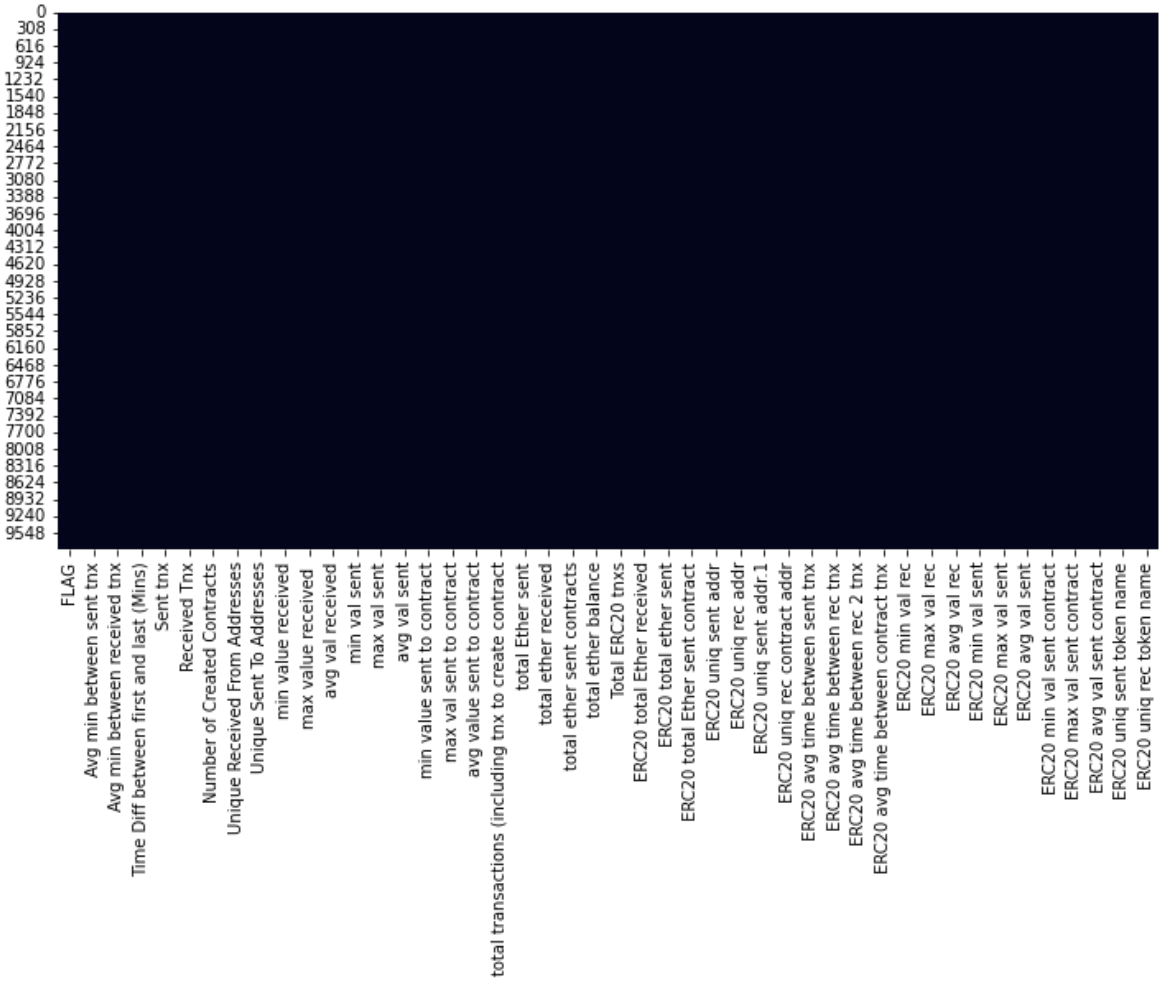
On peut constater qu'il y a des trous dans la heatmap, ce qui rend impossible pour l'instant la suppression des lignes fortement correlées. Il faut donc les filtrer.

In [ ]:
```
# Visualize missings pattern of the dataframe
plt.figure(figsize=(12,6))
sns.heatmap(df.isnull(), cbar=False)
plt.show()
# Drop the two categorical features
df.drop(df[categories], axis=1, inplace=True)
# Replace missings of numerical variables with median
df.fillna(df.median(), inplace=True)# Replace missings of numerical varia
df.fillna(df.median(), inplace=True)
```



On remplace les valeurs manquantes par la médiane de la colonne correspondante

In [ ]:
```
#Remplacement par la médiane des valeurs de la colonne
df.fillna(df.median(), inplace=True)

plt.figure(figsize=(12,6))
sns.heatmap(df.isnull(), cbar=False)
plt.show()
```

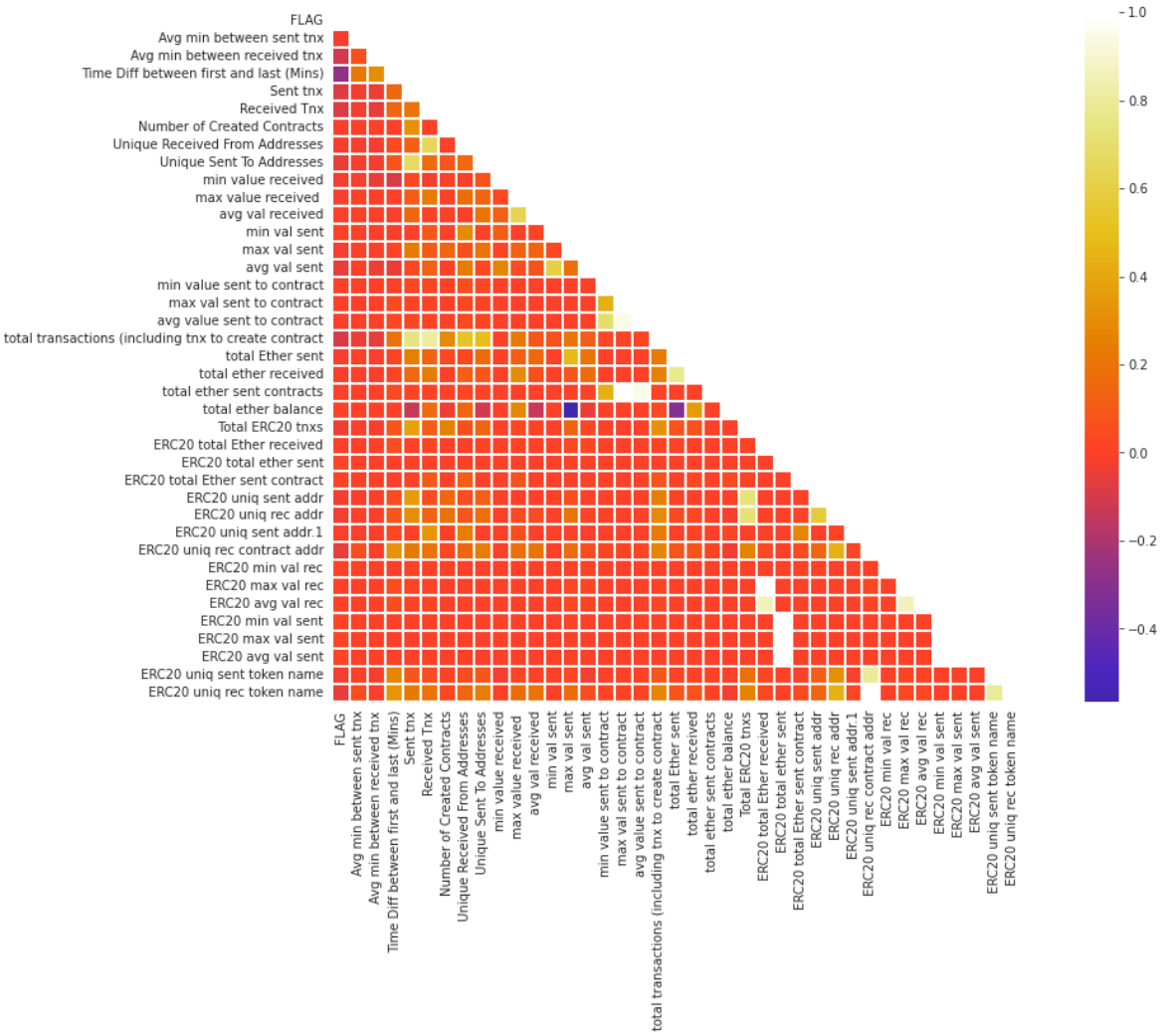


Il n'y a plus de colonne ayant des valeurs nulles. Il faut aussi s'assurer que toutes les colonnes n'aient pas de valeurs identique pour l'ensemble des lignes, c'est à dire une variance nulle, si c'est le cas, il faut supprimer la colonne

In [ ]:
```
# Filtering the features with 0 variance
no_var = df.var() == 0

# Drop features with 0 variance --- these features will not help in the p
df.drop(df.var()[no_var].index, axis = 1, inplace = True)
```

On peut désormer afficher une matrice de variance qui est utilisable :

In [ ]:
```
corr = df.corr()

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)]=True
with sns.axes_style('white'):
    fig, ax = plt.subplots(figsize=(18,10))
    sns.heatmap(corr,  mask=mask, annot=False, cmap='CMRmap', center=0, l
```

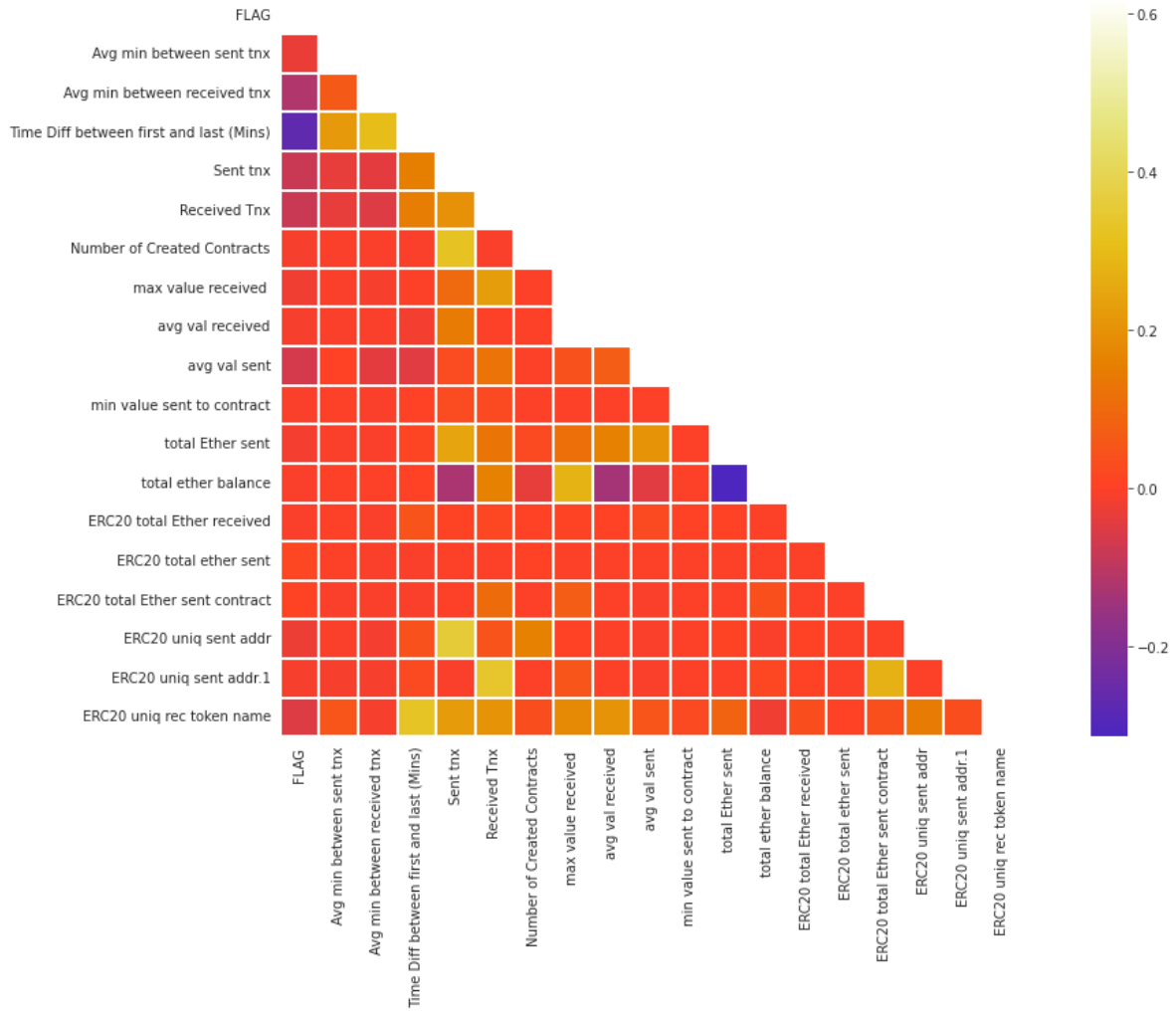


A partir de cette matrice, on relève les lignes ayant le plus de case rouge pour supprimer les features les plus corrélées.

In [ ]:
```
drop = ['total transactions (including tnx to create contract', 'total et
        ' ERC20 avg val rec',' ERC20 max val rec', ' ERC20 min val rec',
        ' ERC20 min val sent', ' ERC20 max val sent', ' Total ERC20 tnxs'
        'Unique Received From Addresses', 'total ether received', ' ERC20
df.drop(drop, axis=1, inplace=True)
```

On obtient la matrice de correlation suivante à la fin :

In [ ]:
```
corr = df.corr()

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)]=True
with sns.axes_style('white'):
    fig, ax = plt.subplots(figsize=(18,10))
    sns.heatmap(corr,  mask=mask, annot=False, cmap='CMRmap', center=0, l
```


FLAG
Avg min between sent tnx
Avg min between received tnx
Time Diff between first and last (Mins)
Sent tnx
Received Tnx
Number of Created Contracts
max value received
avg val received
avg val sent
min value sent to contract
total Ether sent
total ether balance
ERC20 total Ether received
ERC20 total ether sent
ERC20 total Ether sent contract
ERC20 uniq sent addr
ERC20 uniq sent addr.1
ERC20 uniq rec token name
0.6
0.4
0.2
0.0
−0.2