Compte rendu TD1

Auteur : Hugo Soleau

Sommaire

Introduction.....
Note sur l'exécution du programme.....
Exercice 1.....
Exercice 2.....
Exercice 3.....
Exercice 4.....
Exercice 5.....

Introduction

Afin de mener à bien ce TD de façon pratique dans le stockage nos données, nous utiliserons la bibliothèque *pandas* et la bibliothèques *requests* pour effectuer nos requêtages.

```
import requests
import pandas as pd
```

Dans ce TD, nous ferons régulièrement appel à des cryptomonnaies à partir de leur identifiant. Par conséquent, afin de nous simplifier la tâche dans nos traitements, nous concevons la fonction *get_details* qui à partir d'un identifiant de monnaie nous renvoie le fichier json associé sur le site coingecko.

```
def get_details(coin_id) :

    detailed_url = f"https://api.coingecko.com/api/v3/coins/{coin_id}"
    detailed_response = requests.get(detailed_url)
    return detailed_response.json()
```

Note sur l'exécution du programme

Le programme TD1.py s'exécute en lançant la fonction *execute_ex* dont le paramètre d'entrée *n* correspond au numéro de l'exercice qu'on souhaite exécuter.
Il est à noter que le site coingecko limitant les accès pour les comptes gratuits, il est souvent nécessaire d'exécuter plusieurs fois la fonction *execute_ex* avant d'obtenir le résultat correct.
Afin que le lecteur est une idée de si la sortie est correcte, nous donnons un exemple de sortie de chaque exécution fonctionnelle à la fin de nos différents paragraphes.

Exercice 1

Dans cet exercice, on récupère les prix moyens du Bitcoin et de l'Ethereum en EUR, USD, USDC et USDT ainsi que leur volume journalier moyen.
Pour ceci on passe par l'adresse suivante : https://api.coingecko.com/api/v3/coins où on ajoute l'indication de la monnaie et du cryptocoin qui nous intéressent.
Il est donc nécessaire de vérifier qu'une valeur pour les prix est notamment renseignée. En effet, on observe pour le prix en USDT que la valeur n'est pas toujours renseignée.

```
list_coin = pd.DataFrame({'id' : ["bitcoin", "ethereum"]})
```

Exemple de sortie

```
Exercice 1

Bitcoin
Valeur USD : 35373 USD
Valeur EUR : 33159 EUR
Valeur USDC : 18.759285 USDC
Volume : 12825442739 USD
Ethereum
Valeur USD : 1885.66 USD
Valeur EUR : 1768.07 EUR
Valeur USDC : 1.0 USDC
Volume : 13490955336 USD
```

Exercice 2

Ici, on effectue le même travail que précédemment en récupérant les informations sur les 10 cryptocoins les plus liquides via la section "markets" de l'URL précédent.

```
url_tick = "https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd"
response_tick = requests.get(url_tick)
cryptcoin = response_tick.json()

df_tick = pd.DataFrame(cryptcoin[:10])
list_coin_2 = df_tick[['id']]
```

Pour simplifier notre tâche, nous définissons une fonction nommée *print_info* qui effectue la synthèse des informations dans les questions 1 et 2 pour un dataframe donné.

La syntaxe est la suivante :

```
def print_info(list_coin) :

    for index, row in list_coin.iterrows() :
```

```
        coin_id = row['id']
        coin_details = get_details(coin_id)

        # Imprimer des informations clés
        print(coin_details['name'])
        if 'usd' in coin_details['market_data']['current_price'] :
            print("Valeur USD :", coin_details['market_data']['current_price']
['usd'], "USD")
        if 'eur' in coin_details['market_data']['current_price'] :
            print("Valeur EUR :", coin_details['market_data']['current_price']
['eur'], "EUR")
        if 'eth' in coin_details['market_data']['current_price'] :
            print("Valeur USDC :", coin_details['market_data']['current_price']
['eth'], "USDC")
        if 'usdt' in coin_details['market_data']['current_price'] :
            print("Valeur USDC :", coin_details['market_data']['current_price']
['usdt'], "USDT")
        print("Volume :", coin_details['market_data']['total_volume']['usd'],
"USD")
```

Exemple de sortie

```
Exercice 2

Bitcoin
Valeur USD : 35370 USD
Valeur EUR : 33156 EUR
Valeur USDC : 18.758995 USDC
Volume : 13937440788 USD
Ethereum
Valeur USD : 1885.61 USD
Valeur EUR : 1767.6 EUR
Valeur USDC : 1.0 USDC
Volume : 13474259956 USD
Tether
Valeur USD : 1.0 USD
Valeur EUR : 0.937967 EUR
Valeur USDC : 0.00053051 USDC
Volume : 30882299253 USD
...
```

Exercice 3

Dans cet exercice, on cherche à obtenir les 3 cryptomonnaies les plus liquides par rapport au Bitcoin ou à l'Ethereum. Autrement dit, il s'agit pour chaque cryptomonnaie de récupérer son volume en Bitcoin et en Ethereum, de garder la valeur la plus grande des deux puis de la stocker dans un dataframe qu'on triera par ordre décroissant pour n'en garde que la tête composée des 3 premiers actifs.

Nous procédons en définissant une fonction *get_volume* qui à partir d'un identifiant de cryptomonnaie récupère la liste de ses volumes en conversion dans les différents actifs renseignés.

```
def get_volume(coin_id) :

   coin_details = get_details(coin_id)
   if 'market_data' in coin_details:
        coin_details = coin_details['market_data']['total_volume']
        coin_dict = {"id": list(coin_details.keys()), "volume":
list(coin_details.values())}
        return pd.DataFrame(coin_dict)
   else:
        return pd.DataFrame(columns=["id", "volume"])
```

Ensuite, afin d'exploiter de façon pertinente les volumes d'un actif dans les devises qui nous intéressent, nous définissons la fonction *pick_volume* qui à un identifiant de cryptomonnaie et au symbole d'une cryptomonnaie renvoie le volume dans la seconde monnaie de la première.

```
def pick_volume(coin_id, coin):

    temp_coin = get_volume(coin_id)
    volume = temp_coin.loc[temp_coin['id'] == coin, 'volume']
    if volume.empty :
        return 0
    else :
        return volume.iloc[0]
```

Enfin, il s'agit de définir une méthode de construction de notre liste regroupant les actifs dans les monnaies que nous souhaitons et de trier cette même liste.
Ces tâches sont effectuées dans la fonction *sort_coin_vs_crypt* qui prend en entrée la liste des monnaies dont on veut récupérer le volume et un seuil (*threshold*) ici définit à 3 qui indique la longueur de la tête d'éléments à conserver.

```
def sort_coin_vs_crypt(df_coin, threshold) :
    ndf_coin = pd.DataFrame(0, index=df_coin.index, columns=["id", "name",
"volume", "currency"])

    for index, row in df_coin.iterrows() :
        coin_id = row['id']
        ndf_coin.at[index, 'id'] = coin_id
        ndf_coin.at[index, 'name'] = row['name']

        vol_btc = pick_volume(coin_id, 'btc')
        vol_eth = pick_volume(coin_id, 'eth')

        if vol_btc > vol_eth :
            ndf_coin.at[index, 'volume'] = vol_btc
            ndf_coin.at[index, 'currency'] = 'btc'
        else:
            ndf_coin.at[index, 'volume'] = vol_eth
```

```
            ndf_coin.at[index, 'currency'] = 'eth'

    ndf_coin = ndf_coin.sort_values(by='volume', ascending=False)
    ndf_coin = ndf_coin.reset_index(drop=True)

    ndf_coin = ndf_coin.iloc[:threshold]

    return ndf_coin
```

NB : Pour une exécution rapide du code, nous reprenons la liste des 100 actifs les plus échangés utilisés dans l'exercice 2. L'hypothèse que nos 3 actifs les plus échangés en volume se trouvent parmi les 100 plus gros actifs semble pertienente au regard de la conversion de nos monnaies et du non arbitrage entre elles.

```
url_tick = "https://api.coingecko.com/api/v3/coins/markets?vs_currency=usd"
response_tick = requests.get(url_tick)
cryptcoin = response_tick.json()

df_coin_speed = pd.DataFrame(cryptcoin)
df_coin_speed = df_coin_speed[["id", "symbol", "name"]]
```

Exemple de sortie

```
Exercice 3

1. Tether : 16377731 usdt/eth
2. Bitcoin : 8341159 btc/eth
3. Ethereum : 7090189 eth/eth
```

Exercice 4

Nous procédons de la même façon que dans l'exercice 3 en considérant cette fois les valeurs données en USD, USDT et USDC.
Toutefois, ici puisque nous voulons afficher les volumes dans les différentes conversions, nous les conservons en mémoire dans notre dataframe et nous les trions via un attribut *max_volume* qui enregistre la conversion la plus grande dans les différentes monnaies. Les arguments de la fonction *sort_coin_vs_curr* sont les mêmes que dans la fonction *sort_coin_vs_crypt* (**cf. Exercice 3**)

```
def sort_coin_vs_curr(df_curr, threshold) :
    ndf_curr = pd.DataFrame(0, index=df_curr.index, columns=["id", "name",
"volume_usd", "volume_usdc", "volume_tusd", "max_volume"])

    for index, row in df_curr.iterrows() :
        curr_id = row['id']
        ndf_curr.at[index, 'id'] = curr_id
        ndf_curr.at[index, 'name'] = row['name']
```

```
            vol_usd = pick_volume(curr_id, 'usd')
            vol_usdc = pick_volume(curr_id, 'eth')
            vol_tusd = pick_volume(curr_id, 'tusd')

            ndf_curr.at[index, 'volume_usd'] = vol_usd
            ndf_curr.at[index, 'volume_usdc'] = vol_usdc
            ndf_curr.at[index, 'volume_tusd'] = vol_tusd
            ndf_curr.at[index, 'max_volume'] = max(vol_usd, vol_usdc, vol_tusd)

        ndf_curr = ndf_curr.sort_values(by='max_volume', ascending=False)
        ndf_curr = ndf_curr.reset_index(drop=True)

        ndf_curr = ndf_curr.iloc[:threshold]

        return ndf_curr
```

NB : A noter qu'ici aussi nous employons la même liste que pour la question 4 pour les mêmes raisons.

Exemple de sortie

```
Exercice 4

1. Tether : 30882299253 usdt/USD, 16377731 usdt/USDC, 0 usdt/TUSD
2. Bitcoin : 16320088247 btc/USD, 8655269 btc/USDC, 0 btc/TUSD
3. Ethereum : 13353872513 eth/USD, 7083258 eth/USDC, 0 eth/TUSD
```

Exercice 5

Nous avons choisi d'utiliser les données de marché car elles sont plus complètes et permet d'accéder aux différentes valeurs à des périodes différentes. Ici nous avons choisi de traiter les seules données historiques des dernières 24h afin de pouvoir éventuellement effectuer un suivi journalier des volumes échangés en les stockant dans un fichier.