TESTS unitaires en python

https://docs.python.org/fr/3/library/unittest.html

Le fichier testsExtraction.py qui se trouve dans le dossier extraction/test_extraction implémente un test avec le module unittest de python.

Ce test permet de s'assurer que la fonction qui récupère la donnée dans les tables des bases de données ne nécessitant pas de jointure est correctement effectuée et stocké dans le format attendu (ndjson).

D'autres tests devraient idéalement être implémentés. Ce premier test sert donc d'exemple à l'implémentation de tests futurs.

Le module unittest permet la mise en place facile de test.

Conseils :

```
class Test_Extract(unittest.TestCase):
```

- Définir une class distincte pour chaque script python contenant des fonctions à tester en suivant la syntaxe ci-dessus.

```
    def test_get_data_Njoin(self):
```

- Toujours nommer les fonctions de test en commençant par test_nomDeLaFonctionDeTests
- La méthode MagicMock de unittest.mock permet de forcer le résultat que retourne une fonction. Cette méthode est très utile car elle évite de devoir lancer tout le pipeline de fonctions permettant d'aboutir à la fonction que l'on souhaite tester. Par exemple, pour le test unitaire que nous avons implémenter pour get_data_Njoin ; on force le renvoie de données synthétique par query_mariadb afin de ne pas avoir à réellement se connecter à une base de données et en extraire la donnée.

```
self.assertEqual(result, expected_result)
```

- Les self.assert sont divers et permettent aussi bien de vérifier l'égalité de deux paramètres self.assrtEqual que de s'assurer qu'un paramètre est plus grand qu'un autre par exemple `self.assertGreater()`.

TESTS « réalisés à la main »

Afin de s'assure que l'on récupère bien l'intégralité de la donnée, entre autre, nous avons été amenés à réaliser des tests manuellement (par manque de temps pour les implémenter). Ces tests consistaient notamment à s'assure que le nombre d'items générés dans les ndjson

correspondaient bien au nombre d'enregistrements dans les deux bases de données sur lesquelles nous avons travaillées (mouleconnected et meteo1).

Pour les BDD sans jointure (meteo1) nous nous sommes assuré qu'il y avait autant d'item dans le dictionnaire que de résultats en effectuant la requête SQL suivante : SELECT COUNT(*) FROM table

Pour les BD avec jointure comme mouleconnected nous avons exécuté les deux requêtes suivantes :

SELECT COUNT(*) FROM data GROUP BY ConvTimeStamp

SELECT COUNT(*) FROM data2 GROUP BY ConvTimeStamp

Et nous avons sommés les résultats de ces requêtes car lorsque l'on génère le ndjson on récupère un item par timestamp chaque item pouvant donc regrouper plusieurs enregistrements.