



**UNIVERSIDAD DE CÓRDOBA**

Departamento de Informática y Análisis Numérico

Ingeniería del Software, Conocimiento y Bases de Datos

**MASTERES UNIVERSITARIOS-UNIVERSIDAD DE CÓRDOBA**

*ASIGNATURA:*

Análisis, Diseño y Procesamiento de Datos Aplicados a las Ciencias y a las Tecnologías

**Nombre :** Lebbihi

**Apellido:** Mohammed

**Profesores :** 1- Gonzalo  
Cerreuila Garcia

2- Domingo Ortiz Boyer

3- Juan Antonio Romero

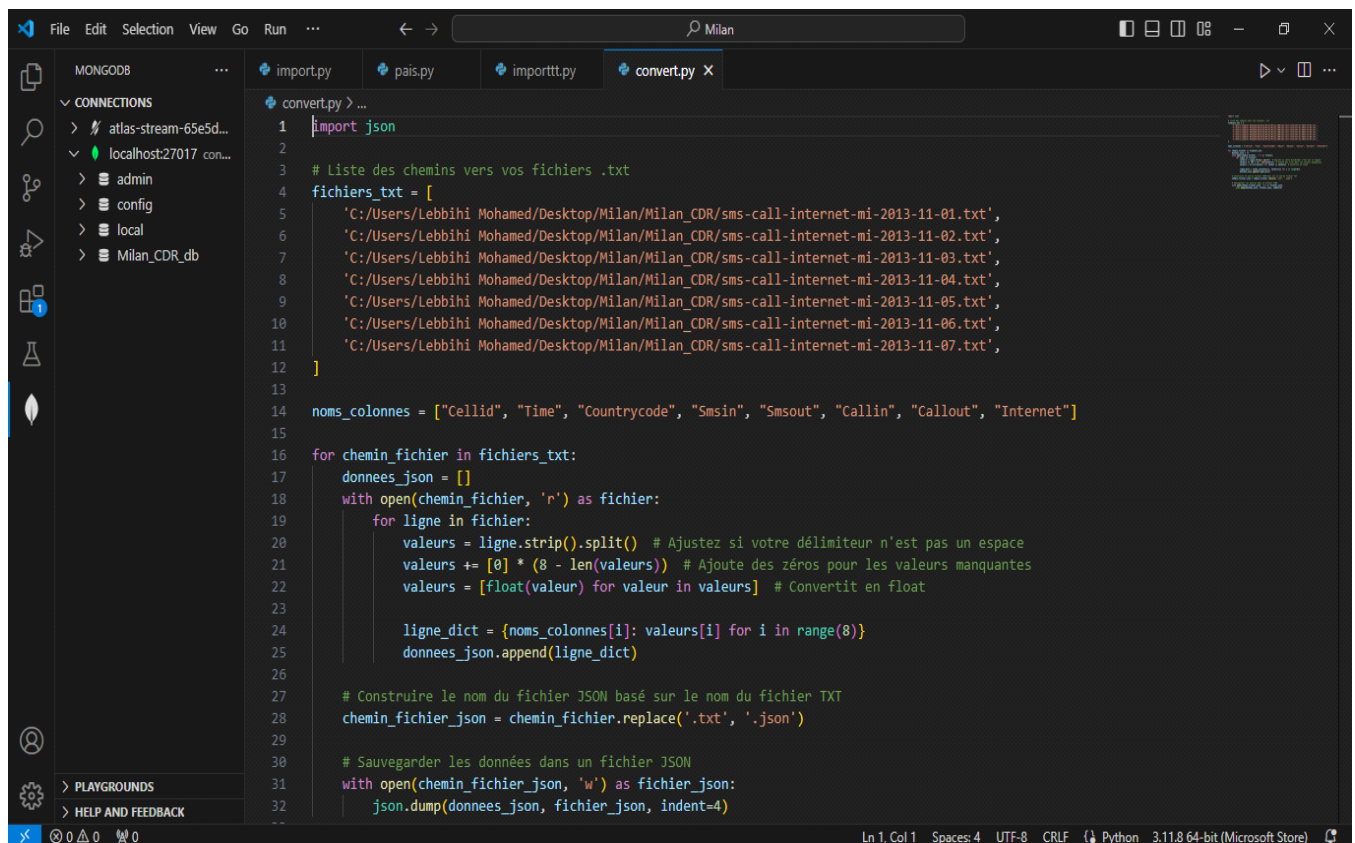
**PRÁCTICAS (APARTADO III)**

Trabajo CDR's de Milán

### convertir archivos .txt a Json:

Este script hace lo siguiente:

- 1- Itère sur chaque chemin de fichier dans la liste fichiers\_txt.
- 2- Para cada archivo, lee las líneas, procesa cada línea para crear una lista de diccionarios donde cada diccionario representa una línea del archivo con ceros añadidos para las columnas que faltan.
- 3- Convierte esta lista de diccionarios en JSON y guarda el resultado en un nuevo archivo .json. El nombre del archivo JSON se deriva del nombre del archivo .txt original (sustituyendo .txt por .json).
- 4- Este enfoque procesa automáticamente los 7 archivos .txt especificados en la lista files\_txt y crea un archivo .json correspondiente para cada uno de ellos.



```
1 import json
2
3 # Liste des chemins vers vos fichiers .txt
4 fichiers_txt = [
5     'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-01.txt',
6     'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-02.txt',
7     'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-03.txt',
8     'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-04.txt',
9     'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-05.txt',
10    'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-06.txt',
11    'C:/Users/Lebbihi Mohamed/Desktop/Milan/Milan_CDR/sms-call-internet-mi-2013-11-07.txt',
12 ]
13
14 noms_colonnes = ["Cellid", "Time", "Countrycode", "Smsin", "Smsout", "Callin", "Callout", "Internet"]
15
16 for chemin_fichier in fichiers_txt:
17     donnees_json = []
18     with open(chemin_fichier, 'r') as fichier:
19         for ligne in fichier:
20             valeurs = ligne.strip().split() # Ajustez si votre délimiteur n'est pas un espace
21             valeurs += [0] * (8 - len(valeurs)) # Ajoute des zéros pour les valeurs manquantes
22             valeurs = [float(valeur) for valeur in valeurs] # Convertit en float
23
24             ligne_dict = {noms_colonnes[i]: valeurs[i] for i in range(8)}
25             donnees_json.append(ligne_dict)
26
27 # Construire le nom du fichier JSON basé sur le nom du fichier TXT
28 chemin_fichier_json = chemin_fichier.replace('.txt', '.json')
29
30 # Sauvegarder les données dans un fichier JSON
31 with open(chemin_fichier_json, 'w') as fichier_json:
32     json.dump(donnees_json, fichier_json, indent=4)
```

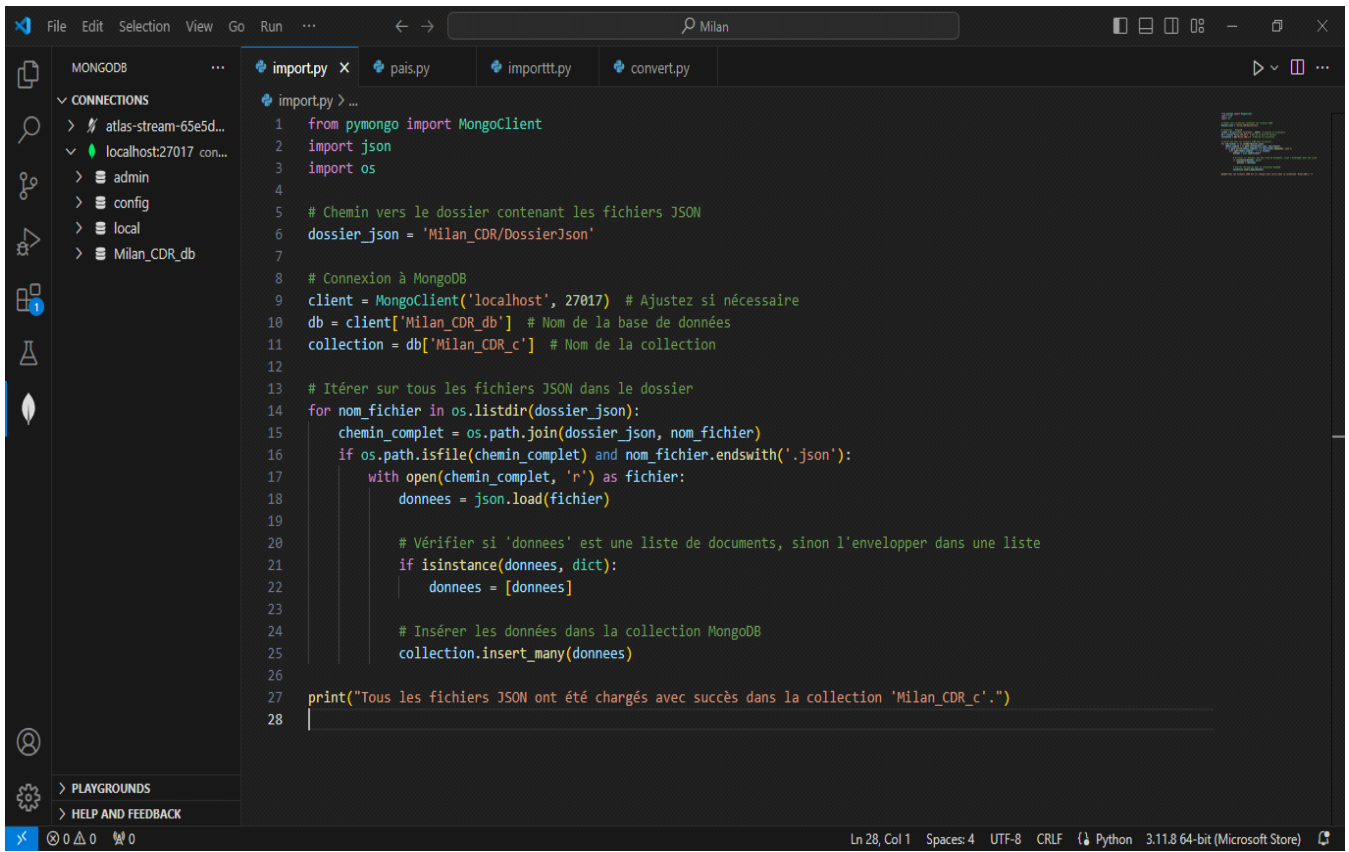
### 1) Importación de archivos a mongodb:

Este script hace lo siguiente:

- 1- Se conecta a MongoDB y especifica la base de datos y la colección de destino.
- 2- Utiliza os.listdir para obtener todos los archivos de la carpeta especificada.

3- Para cada archivo, compruebe si se trata de un archivo JSON (que termina en .json).

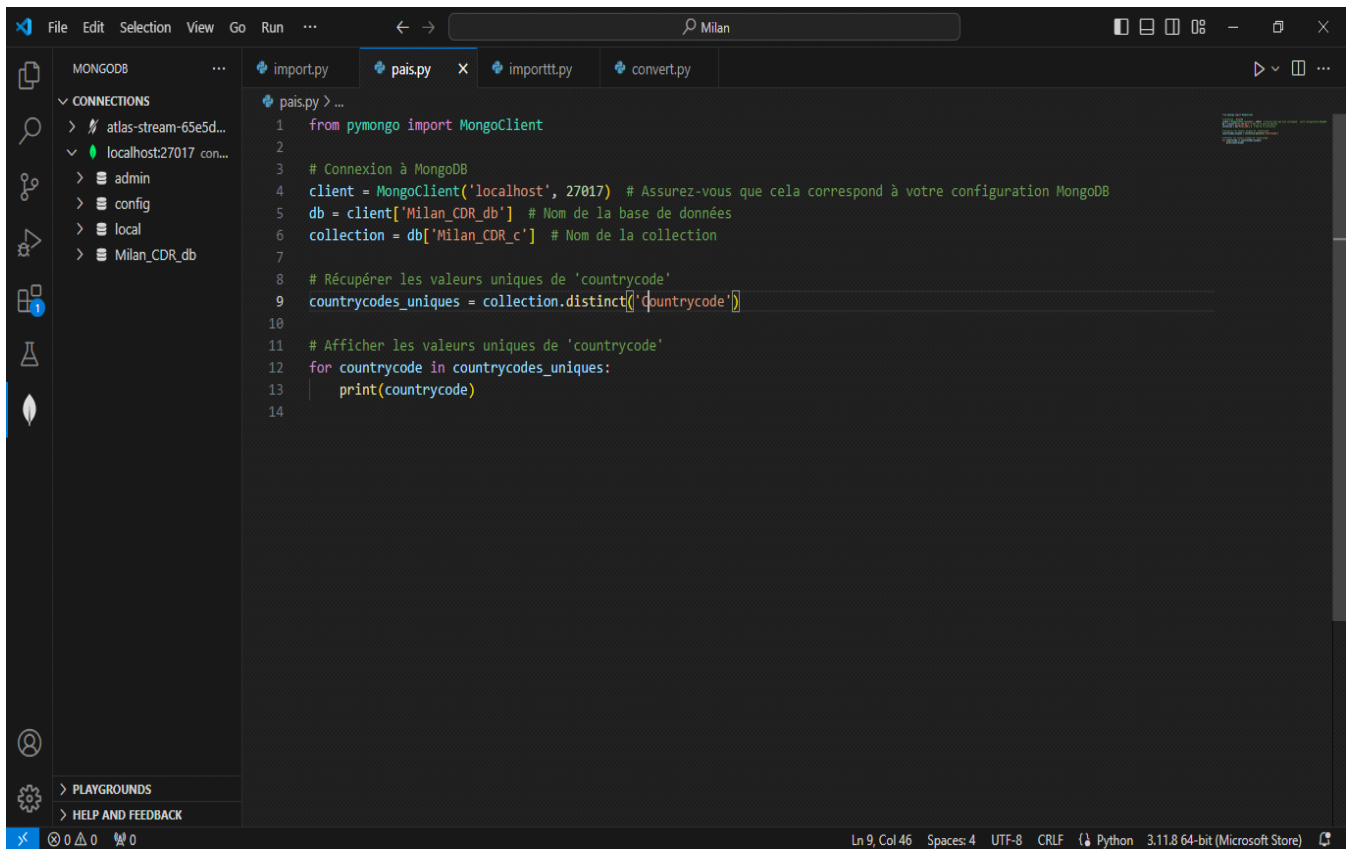
4- Abre cada archivo JSON, carga su contenido e inserta los datos en la colección MongoDB.



```
1 from pymongo import MongoClient
2 import json
3 import os
4
5 # Chemin vers le dossier contenant les fichiers JSON
6 dossier_json = 'Milan_CDR/DossierJson'
7
8 # Connexion à MongoDB
9 client = MongoClient('localhost', 27017) # Ajustez si nécessaire
10 db = client['Milan_CDR_db'] # Nom de la base de données
11 collection = db['Milan_CDR_c'] # Nom de la collection
12
13 # Itérer sur tous les fichiers JSON dans le dossier
14 for nom_fichier in os.listdir(dossier_json):
15     chemin_complet = os.path.join(dossier_json, nom_fichier)
16     if os.path.isfile(chemin_complet) and nom_fichier.endswith('.json'):
17         with open(chemin_complet, 'r') as fichier:
18             donnees = json.load(fichier)
19
20     # Vérifier si 'donnees' est une liste de documents, sinon l'envelopper dans une liste
21     if isinstance(donnees, dict):
22         donnees = [donnees]
23
24     # Insérer les données dans la collection MongoDB
25     collection.insert_many(donnees)
26
27 print("Tous les fichiers JSON ont été chargés avec succès dans la collection 'Milan_CDR_c'.")
28
```

## 2) Encuentra los países con los que se interactúa :

Este script utiliza la función `distinct` para recuperar una matriz de todos los valores únicos del campo `countrycode` de la colección `Milan_CDR_c`. A continuación, itera sobre esta matriz y muestra cada valor único. A continuación, itera sobre esta matriz y muestra cada valor único. Este método garantiza que no se repitan valores de código de país en la pantalla.



```
1 from pymongo import MongoClient
2
3 # Connexion à MongoDB
4 client = MongoClient('localhost', 27017) # Assurez-vous que cela correspond à votre configuration MongoDB
5 db = client['Milan_CDR_db'] # Nom de la base de données
6 collection = db['Milan_CDR_c'] # Nom de la collection
7
8 # Récupérer les valeurs uniques de 'countrycode'
9 countrycodes_uniques = collection.distinct('countrycode')
10
11 # Afficher les valeurs uniques de 'countrycode'
12 for countrycode in countrycodes_uniques:
13     print(countrycode)
14
```

### **3) Encuentra que país es con el que más se interactúa ademas de Italia:**

Este script realiza las siguientes operaciones:

- 1- Agrupación: Agrupa los documentos por código de país sin excluir primero el valor 0, y cuenta el número de apariciones de cada código de país.
- 2- Ordenar: Ordena los grupos por número de apariciones en orden descendente, de forma que los valores más frecuentes aparezcan al principio de la lista.
- 3- Recuento de 0: Utiliza count\_documents con un filtro específico para countrycode igual a 0 para calcular directamente su ocurrencia sin tener que filtrar este valor en la agregación.

```
1 from pymongo import MongoClient
2
3 # Connexion à MongoDB
4 client = MongoClient('localhost', 27017) # Assurez-vous que cela correspond à votre configuration MongoDB
5 db = client['Milan_CDR_db'] # Nom de la base de données
6 collection = db['Milan_CDR_c'] # Nom de la collection
7 # Compter les occurrences de la valeur 39 pour countrycode
8 occurrences_39 = collection.count_documents({"Countrycode": 39})
9 print(f"Nombre d'occurrences de la valeur 39 pour Countrycode: {occurrences_39}")
10 # Pipeline d'agrégation pour trouver le countrycode le plus répété (à l'exception de 39)
11 pipeline = [
12     {"$match": {"Countrycode": {"$ne": 39}}}, # Exclure les documents où countrycode est 39
13     {"$group": {"_id": "$Countrycode", "count": {"$sum": 1}}}, # Grouper par countrycode et compter
14     {"$sort": {"count": -1}}, # Trier par count décroissant
15     {"$limit": 1} # Limiter à la valeur la plus répétée
16 ]
17 # Exécuter l'agrégation
18 resultat = list(collection.aggregate(pipeline))
19 if resultat:
20     # Afficher le countrycode le plus fréquent (à l'exception de 39) et son nombre d'occurrences
21     print(f"Countrycode le plus répété (à l'exception de 39): {resultat[0]['_id']}, Nombre d'occurrences: {resultat[0]['count']}")
22 else:
23     print("Aucun countrycode trouvé ou tous sont 39.")
24
```

ihl Mohamed/Desktop/Milan/paisleplusappeler.py  
Nombre d'occurrences de la valeur 39 pour Countrycode: 10079821  
Countrycode le plus répété (à l'exception de 39): 0.0, Nombre d'occurrences: 7955616  
PS C:\Users\Lebbihl Mohamed\Desktop\Milan>

#### 4 -Celda comunica más con el extranjero:

Este script utiliza un proceso de agregación que comienza con \$match para excluir los documentos en los que countrycode es igual a 39. El resto del proceso no cambia y sigue agrupando los documentos por cellid, contando las ocurrencias, ordenando los resultados por número de ocurrencias en orden descendente y limitando el resultado al cellid más frecuente.

```
1 from pymongo import MongoClient
2
3 # Connexion à MongoDB
4 client = MongoClient('localhost', 27017)
5 db = client['Milan_CDR_db']
6 collection = db['Milan_CDR_c']
7
8 # Pipeline d'agrégation pour trouver le 'cellid' le plus fréquent sans prendre en compte countrycode = 39
9 pipeline = [
10     {"$match": {"Countrycode": {"$ne": 39}}}, # Exclure les documents où countrycode est 39
11     {"$group": {"_id": "$Cellid", "count": {"$sum": 1}}}, # Grouper par 'cellid' et compter
12     {"$sort": {"count": -1}}, # Trier par le nombre d'occurrences en ordre décroissant
13     {"$limit": 1} # Prendre le 'cellid' le plus fréquent
14 ]
15
16 # Exécuter l'agrégation
17 resultat = list(collection.aggregate(pipeline))
18
19 if resultat:
20     print(f"La Cellid qui communique le plus avec l'etranger: {resultat[0]['_id']}, Nombre d'occurrences: {resultat[0]['count']}")
21 else:
22     print("Aucun Cellid trouvé ou tous ont un Countrycode de 39.")
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Lebbihi Mohamed\Desktop\Milan> & "C:/Users/Lebbihi Mohamed/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Lebbihi Mohamed/Desktop/Milan/Cellid.py"

La Cellid qui communique le plus avec l'etranger: 6064.0, Nombre d'occurrences: 16984

PS C:\Users\Lebbihi Mohamed\Desktop\Milan>

## 5) la celda con más actividad de smsin, smsout callin, callout, internet y la total:

El script que compartí anteriormente realiza una operación de agregación en una colección MongoDB. Agrupa los documentos por Cellid y cuenta el número de entradas de las columnas Smsin, Smsout, Callin, Callout e Internet que no son cero.

```
1 from pymongo import MongoClient
2 # Connexion à MongoDB
3 client = MongoClient('localhost', 27017)
4 db = client['Milan_CDR_db']
5 collection = db['Milan_CDR_c']
6 # Pipeline d'agrégation pour compter les valeurs non nulles et trouver le maximum dans chaque catégorie
7 pipeline = [
8     {
9         "$group": {
10             "_id": "$Cellid",
11             "Smsin_Count": {"$sum": {"$cond": [{"$ne": ["$Smsin", 0]}, 1, 0]}},
12             "Smsout_Count": {"$sum": {"$cond": [{"$ne": ["$Smsout", 0]}, 1, 0]}},
13             "Callin_Count": {"$sum": {"$cond": [{"$ne": ["$Callin", 0]}, 1, 0]}},
14             "Callout_Count": {"$sum": {"$cond": [{"$ne": ["$Callout", 0]}, 1, 0]}},
15             "Internet_Count": {"$sum": {"$cond": [{"$ne": ["$Internet", 0]}, 1, 0]}}
16         }
17     },
18     # je montre comment récupérer la valeur la plus élevée de toutes les catégories.
19     {
20         "$project": {
21             "Cellid": "$_id",
22             "Smsin_Count": 1,
23             "Smsout_Count": 1,
24             "Callin_Count": 1,
25             "Callout_Count": 1,
26             "Internet_Count": 1,
27             "Total_Count": {"$add": ["$Smsin_Count", "$Smsout_Count", "$Callin_Count", "$Callout_Count", "$Internet_Count"]}
28         }
29     },
30     {"$sort": {"Total_Count": -1}},
31     {"$limit": 1}
32 ]
```

```
29     },
30     {"$sort": {"Total_Count": -1}},
31     {"$limit": 1}
32 ]
33 # Exécuter l'agrégation
34 resultat = list(collection.aggregate(pipeline))
35 # Affichage du résultat
36 if resultat:
37     print(f"Cellid avec le plus d'activité non-nulle: {resultat[0]['Cellid']}")
38     print(f"Smsin non-zero count: {resultat[0]['Smsin_Count']}")
39     print(f"Smsout non-zero count: {resultat[0]['Smsout_Count']}")
40     print(f"Callin non-zero count: {resultat[0]['Callin_Count']}")
41     print(f"Callout non-zero count: {resultat[0]['Callout_Count']}")
42     print(f"Internet non-zero count: {resultat[0]['Internet_Count']}")
43     print(f"Total non-zero count: {resultat[0]['Total_Count']}")
44 else:
45     print("Aucun résultat trouvé.")
46
```

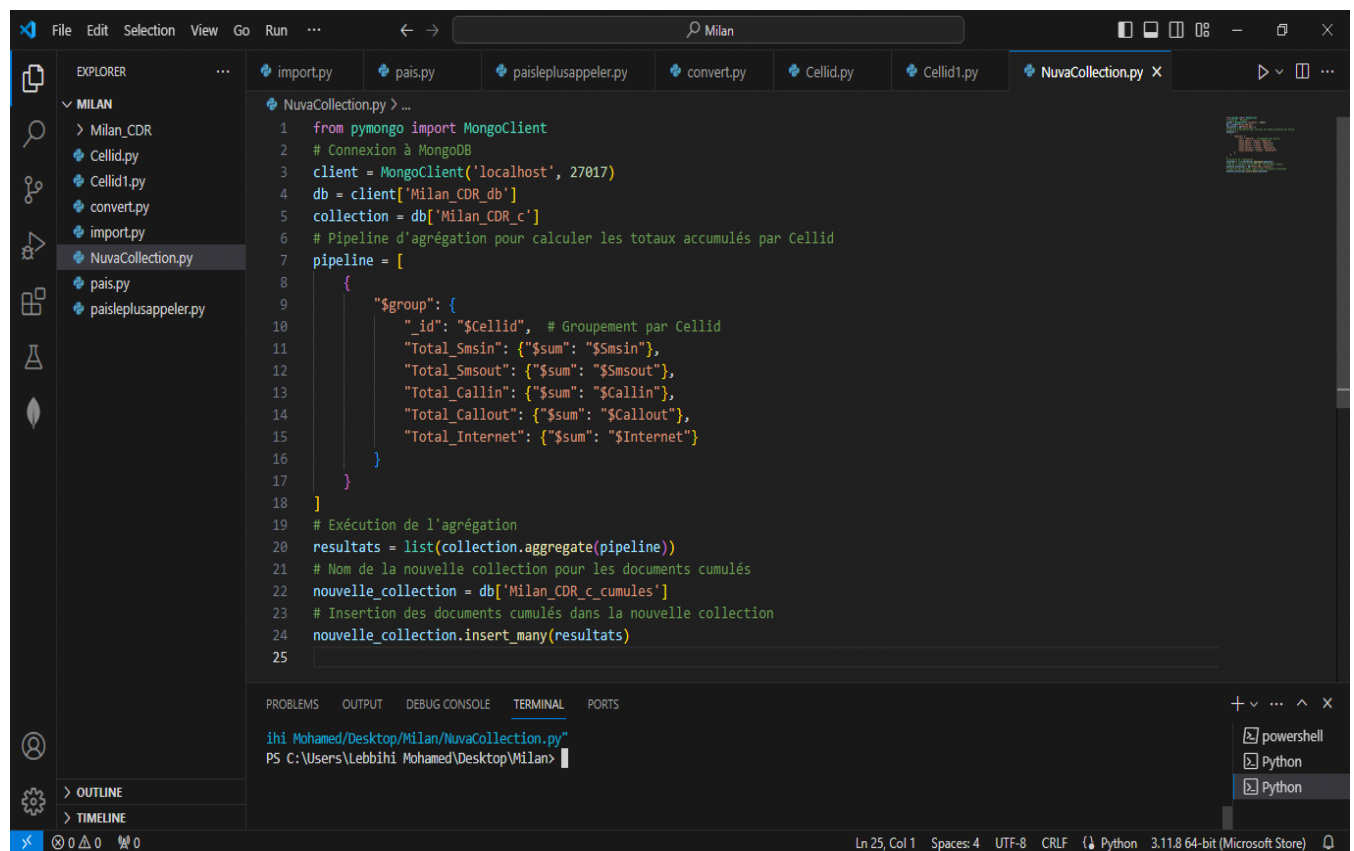
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Lebbihi Mohamed\Desktop\Milan> & "C:/Users/Lebbihi Mohamed/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "c:/Users/Lebbihi Mohamed/Desktop/Milan/Cellid1.py"
Cellid avec le plus d'activité non-nulle: 6064.0
Smsin non-zero count: 17912
Smsout non-zero count: 7045
Callin non-zero count: 3239
Callout non-zero count: 1608
Internet non-zero count: 974
Total non-zero count: 30778
PS C:\Users\Lebbihi Mohamed\Desktop\Milan>
```

## 6) Une collection avec un document par cellule en el que aparezcen los acumulados de los diferentes campos:

Este script realizará las siguientes acciones:

- 1- Se conecta a la base de datos Milan\_CDR\_db y accede a la colección existente Milan\_CDR\_c.
- 2- Define un pipeline de agregación que agrupa los documentos por Cellid y calcula la suma de los campos Smsin, Smsout, Callin, Callout y Internet para cada Cellid.
- 3- Ejecuta la operación de agregación y guarda los resultados en una variable resultados.
- 4- Crea una nueva colección llamada Milan\_CDR\_c\_accumulated en la base de datos.
- 5- Inserta los documentos de resultados acumulados en la nueva colección.

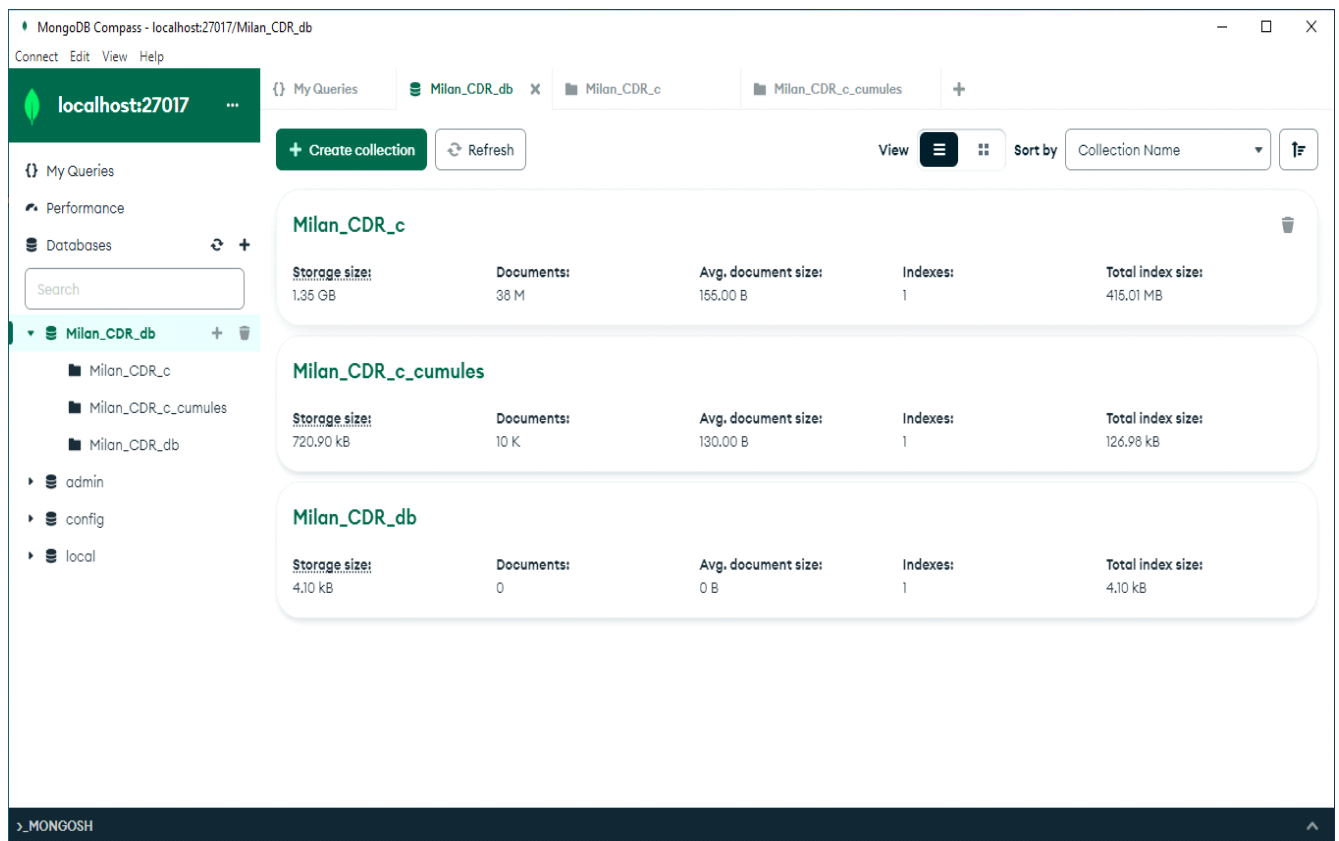


The screenshot shows a Visual Studio Code editor window with a Python script named `NuvaCollection.py` open. The script is designed to connect to a MongoDB database, perform an aggregation pipeline, and save the results to a new collection. The Explorer sidebar on the left shows a project named 'MILAN' with several files, including `NuvaCollection.py`. The script code is as follows:

```
1 from pymongo import MongoClient
2 # Connexion à MongoDB
3 client = MongoClient('localhost', 27017)
4 db = client['Milan_CDR_db']
5 collection = db['Milan_CDR_c']
6 # Pipeline d'agrégation pour calculer les totaux accumulés par Cellid
7 pipeline = [
8     {
9         "$group": {
10             "_id": "$Cellid", # Groupement par Cellid
11             "Total_Smsin": {"$sum": "$Smsin"},
12             "Total_Smsout": {"$sum": "$Smsout"},
13             "Total_Callin": {"$sum": "$Callin"},
14             "Total_Callout": {"$sum": "$Callout"},
15             "Total_Internet": {"$sum": "$Internet"}
16         }
17     }
18 ]
19 # Exécution de l'agrégation
20 resultats = list(collection.aggregate(pipeline))
21 # Nom de la nouvelle collection pour les documents cumulés
22 nouvelle_collection = db['Milan_CDR_c_cumules']
23 # Insertion des documents cumulés dans la nouvelle collection
24 nouvelle_collection.insert_many(resultats)
25
```

The bottom of the editor shows the TERMINAL panel with the command prompt indicating the file path and the current directory.





**7) Una colección con una documento por celda y hora en el que aparezcan los acumulados de los diferentes campos:**

El script está diseñado para realizar agregación en MongoDB, donde agrupa documentos por Cellid y una hipotética columna de tiempo, calculando la suma de campos específicos(Smsin, Smsout, Callin, Callout, e Internet) para cada grupo único de Cellid y tiempo.

```
File Edit Selection View Go Run ... Milan
EXPLORER
  MILAN
    Milan_CDR
      7NuvaCollection.py
      Cellid1.py
      Cellid.py
      convert.py
      import.py
      NuvaCollection.py
      pais.py
      paisleplusappeler.py
  OUTLINE
  TIMELINE
  0 0 0 0

7NuvaCollection.py > ...
1 from pymongo import MongoClient
2 # Connexion à MongoDB
3 client = MongoClient('localhost', 27017)
4 db = client['Milan_CDR_db']
5 collection = db['Milan_CDR_c']
6 # Pipeline d'agrégation pour calculer les totaux accumulés par Cellid et heure extrait de timestamp
7 pipeline = [
8     {
9         "$addFields": {
10             "Heure": {"$hour": "$timestamp"} # Ajoute un champ Heure en extrayant l'heure du champ timestamp
11         }
12     },
13     {
14         "$group": {
15             "_id": {
16                 "Cellid": "$Cellid",
17                 "Heure": "$Heure" # Groupe par Cellid et Heure
18             },
19             "Total_Smsin": {"$sum": "$Smsin"},
20             "Total_Smsout": {"$sum": "$Smsout"},
21             "Total_Callin": {"$sum": "$Callin"},
22             "Total_Callout": {"$sum": "$Callout"},
23             "Total_Internet": {"$sum": "$Internet"}
24         }
25     }
26 ]
27 # Exécution de l'agrégation
28 resultats = list(collection.aggregate(pipeline))
29 # Nom de la nouvelle collection pour les documents cumulés par Cellid et heure
30 nouvelle_collection = db['Milan_CDR_c_cumules_par_heure']
31 # Insertion des documents cumulés dans la nouvelle collection
32 nouvelle_collection.insert_many(resultats)
```

MongoDB Compass - localhost:27017/Milan\_CDR\_db.Milan\_CDR\_c\_cumules\_par\_heure

Connect Edit View Collection Help

localhost:27017

{ } My Queries Milan\_CDR\_db Milan\_CDR\_c Milan\_CDR\_c\_cumules Milan\_CDR\_c\_cumules\_par\_h...

### Milan\_CDR\_db.Milan\_CDR\_c\_cumules\_par\_heure

10.0k 1 DOCUMENTS INDEXES

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or [Generate query](#) Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE 1 - 20 of 10000

```
{ "_id": "Object", "Cellid": 3203, "Heure": null, "Total_Smsin": 168.19768732517713, "Total_Smsout": 206.98438530328414, "Total_Callin": 250.99342069299416, "Total_Callout": 353.1585398902743, "Total_Internet": 1534.3678309123845 }
```

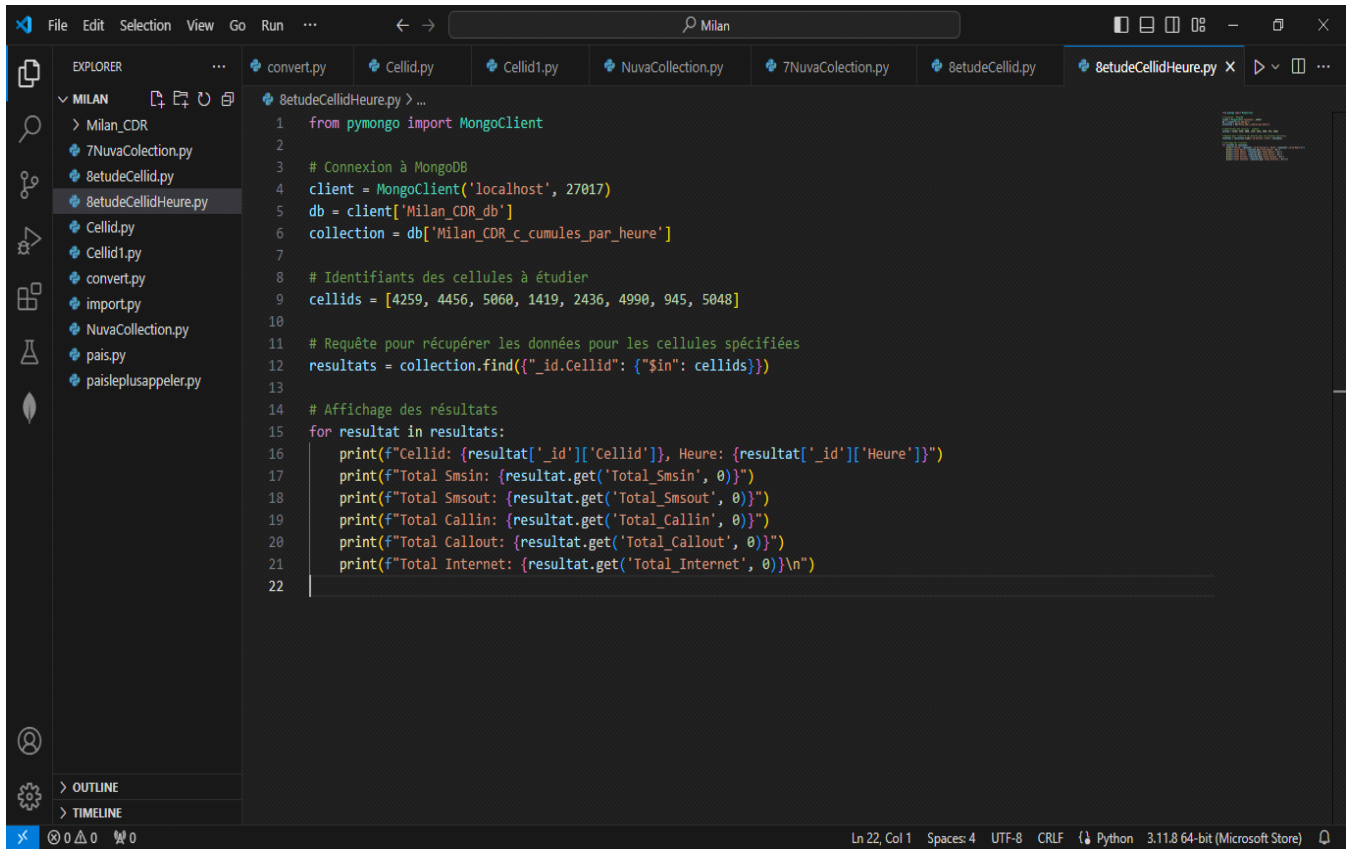
```
{ "_id": "Object", "Cellid": 1599, "Heure": null, "Total_Smsin": 2737.500123212235, "Total_Smsout": 1894.295020688834, "Total_Callin": 2769.1720879516424, "Total_Callout": 2807.759990895204, "Total_Internet": 24862.04343211312 }
```

> MONGOSH

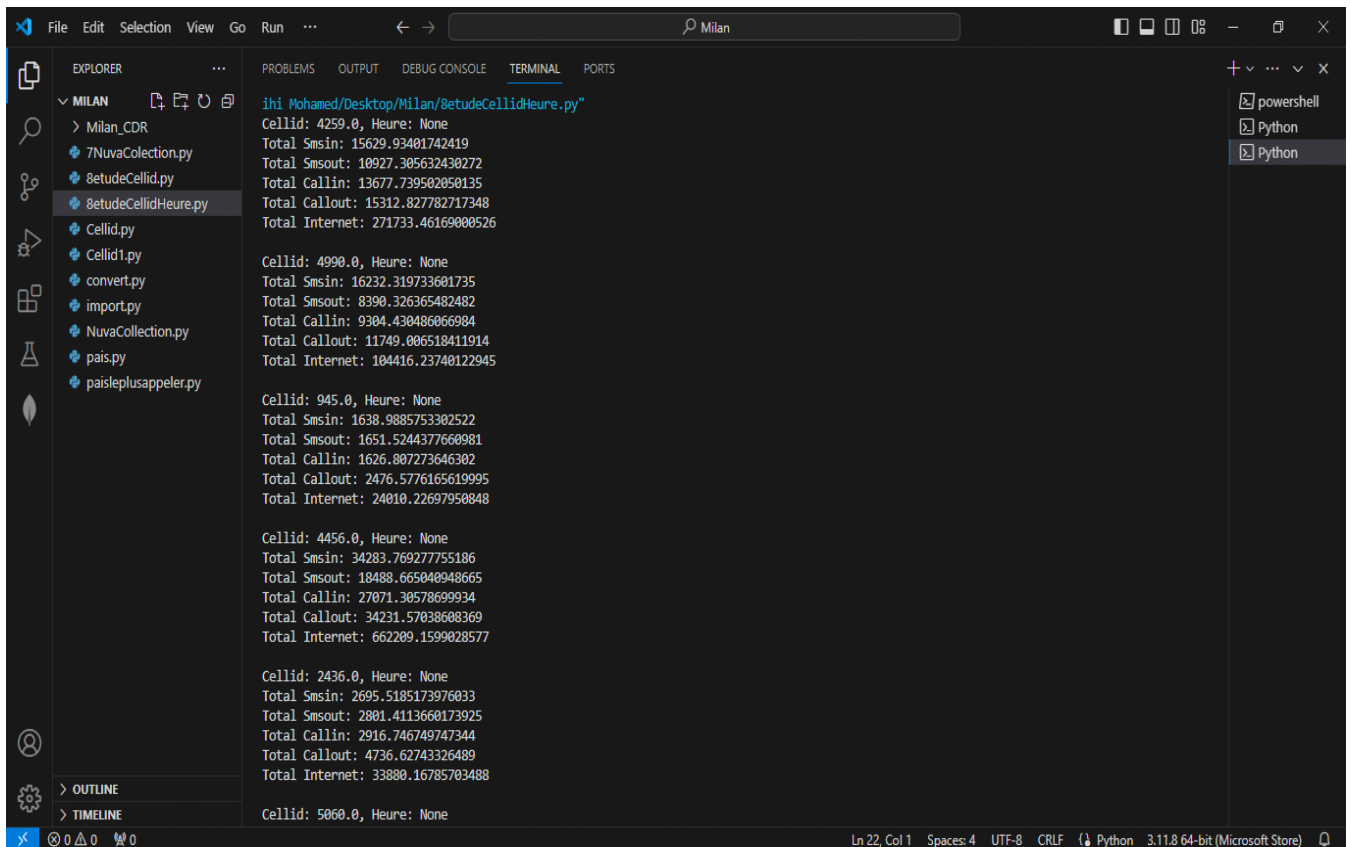
**8) Estudio de las celdas 4259 (Bocconi), 4456 (Navigli), 5060 (Duomo), 1419 (terreno agrícola), 2436 (zona industrial), 4990 (aeropuerto de Linate), 945 (residencial aislado) y 5048 (residencial central):**

Este script :

- 1- Se conecta a la base de datos MongoDB y accede a la colección Milan\_CDR\_c\_cumules\_per\_hour.
- 2- Define un array con el Cellid de las celdas de interés.
- 3- Utiliza find con un criterio de búsqueda para seleccionar los documentos que coincidan con el Cellid especificado. Aquí, "\_id.Cellid" se refiere a la estructura del documento donde \_id es un objeto que contiene Cellid y potencialmente otros campos como Hora.
- 4- Recorre los documentos devueltos y muestra la información acumulada, incluidos los totales de Smsin, Smsout, Callin, Callout e Internet, así como el Cellid y la hora, si están disponibles.



```
1 from pymongo import MongoClient
2
3 # Connexion à MongoDB
4 client = MongoClient('localhost', 27017)
5 db = client['Milan_CDR_db']
6 collection = db['Milan_CDR_c_cumules_par_heure']
7
8 # Identifiants des cellules à étudier
9 cellids = [4259, 4456, 5060, 1419, 2436, 4990, 945, 5048]
10
11 # Requête pour récupérer les données pour les cellules spécifiées
12 resultats = collection.find({"_id.Cellid": {"$in": cellids}})
13
14 # Affichage des résultats
15 for resultat in resultats:
16     print(f"Cellid: {resultat['_id']['Cellid']}, Heure: {resultat['_id']['Heure']}")
17     print(f"Total Smsin: {resultat.get('Total_Smsin', 0)}")
18     print(f"Total Smsout: {resultat.get('Total_Smsout', 0)}")
19     print(f"Total Callin: {resultat.get('Total_Callin', 0)}")
20     print(f"Total Callout: {resultat.get('Total_Callout', 0)}")
21     print(f"Total Internet: {resultat.get('Total_Internet', 0)}\n")
22
```



```
ihi Mohamed/Desktop/Milan/8etudeCellidHeure.py"
Cellid: 4259.0, Heure: None
Total Smsin: 15629.93401742419
Total Smsout: 10927.305632430272
Total Callin: 13677.739502050135
Total Callout: 15312.827782717348
Total Internet: 271733.46169000526

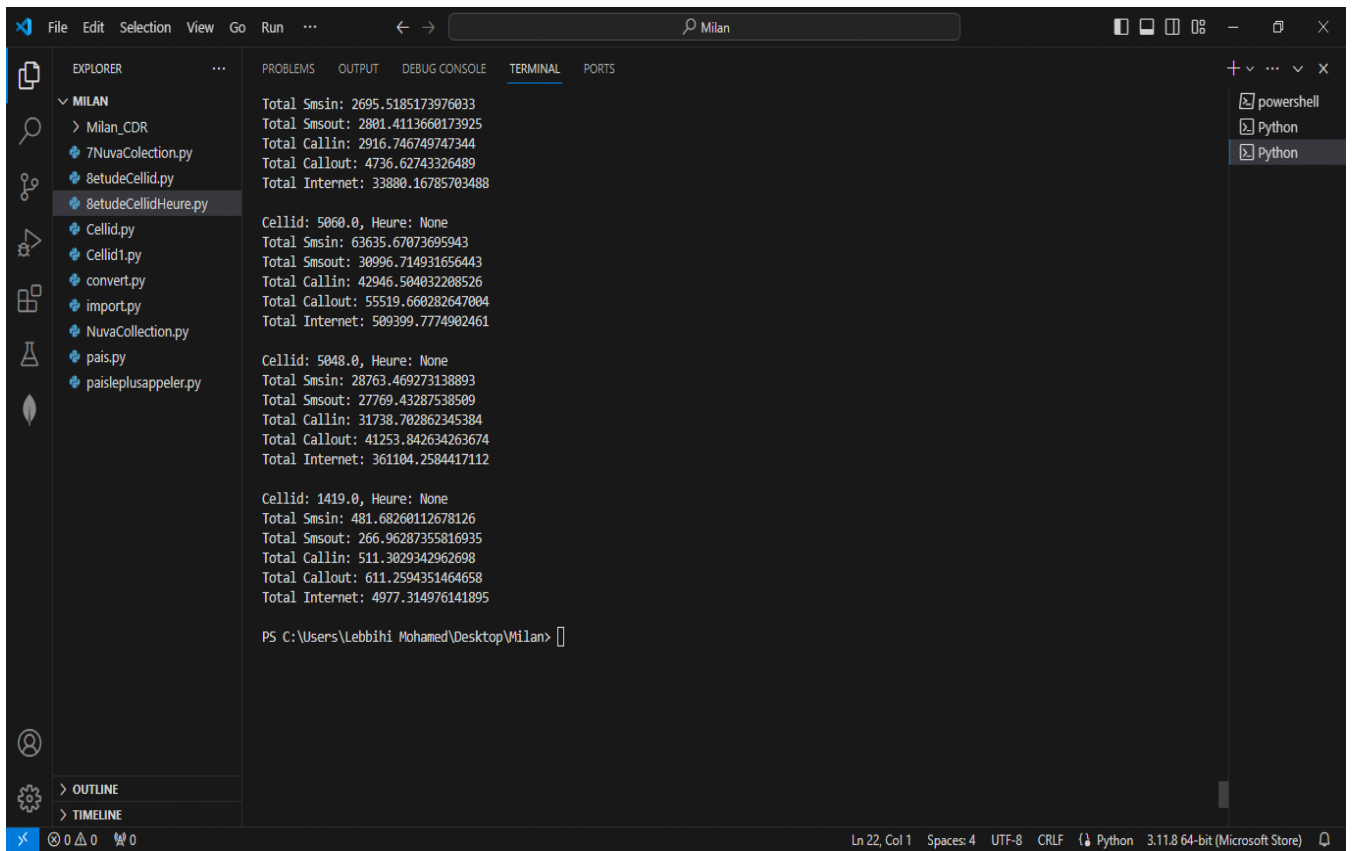
Cellid: 4990.0, Heure: None
Total Smsin: 16232.319733601735
Total Smsout: 8390.326365482482
Total Callin: 9304.430486066984
Total Callout: 11749.006518411914
Total Internet: 104416.23740122945

Cellid: 945.0, Heure: None
Total Smsin: 1638.9885753302522
Total Smsout: 1651.5244377660981
Total Callin: 1626.807273646302
Total Callout: 2476.5776165619995
Total Internet: 24010.22697950848

Cellid: 4456.0, Heure: None
Total Smsin: 34283.769277755186
Total Smsout: 18488.665040948665
Total Callin: 27071.30578699934
Total Callout: 34231.57038608369
Total Internet: 662209.1599028577

Cellid: 2436.0, Heure: None
Total Smsin: 2695.5185173976033
Total Smsout: 2801.4113660173925
Total Callin: 2916.746749747344
Total Callout: 4736.62743326489
Total Internet: 33880.16785703488

Cellid: 5060.0, Heure: None
```



utilicé mongoddbcompass porque no podía conectar atlas