Guide d'utilisation du recensement au format parquet

Ce guide présente quelques exemples d'utilisation des données du recensement de la population diffusées au format `Parquet`.

Pour plus d'informations sur le format `Parquet`, dans un contexte de statistique publique, se référer à Dondon and Lamarche (2023). Pour un exemple sur la différence entre format `CSV` et `Parquet` illustré sur les données du recensement de la population, voir Mauvière (2022).

Initialisation

Ce guide propose d'utiliser DuckDB à travers plusieurs langages pour effectuer des traitements sur les fichiers détails du recensement. Par rapport à d'autres approches, DuckDB a été choisi pour son efficacité ainsi que pour son universalité puisque peu de différences

On va supposer que les données sont stockées dans un dossier conservé sous le nom `path_data` (par exemple `path_data = 'C:/MesDocuments/dossierpersonnel'`). Des vues sont créées afin de faciliter les requêtes ultérieures.

Python

```
import duckdb

renommage_documentation = "SELECT " +\
  "COD_VAR AS nom_variable, " +\
  "LIB_VAR AS description_variable, " +\
  "TYPE_VAR AS type_variable, "  +\
  "COD_MOD AS code_modalite, " +\
  "LIB_MOD AS description_modalite"
```

```
duckdb.sql(
    f'CREATE OR REPLACE VIEW table_individu AS SELECT * FROM read_parquet("{path_data}/FD_
    )
duckdb.sql(
    f'CREATE OR REPLACE VIEW table_logement AS SELECT * FROM read_parquet("{path_data}/FD_
    )
duckdb.sql(
    f'CREATE OR REPLACE VIEW documentation AS {renommage_documentation} FROM read_csv_auto
    )
```

R

```
library(duckdb)
library(glue)

# Pour créer une base de données en mémoire
con <- dbConnect(duckdb())


renommage_documentation <- paste(
"SELECT",
"COD_VAR AS nom_variable,",
"LIB_VAR AS description_variable,",
"TYPE_VAR AS type_variable,",
"COD_MOD AS code_modalite,",
"LIB_MOD AS description_modalite"
)


dbExecute(
  con,
  glue(
    'CREATE OR REPLACE VIEW table_logement AS SELECT * FROM read_parquet("{path_data}/FD_I
  )
)

dbExecute(
  con,
  glue(
    'CREATE OR REPLACE VIEW table_logement AS SELECT * FROM read_parquet("{path_data}/FD_L
```

```
  )
)

dbExecute(
  con,
  glue(
  'CREATE OR REPLACE VIEW documentation AS {renommage_documentation} FROM read_csv_auto("{
  )
)
```

Pour rapidement avoir une idée des informations présentes dans ces données, le code ci-dessous peut être utilisé :

Python

```
schema_table_individu = duckdb.sql(
  "SELECT * FROM documentation"
  ).to_df()
schema_table_individu.head(2)
```

R

```
schema_table_individu <- dbGetQuery(
  con,
  "SELECT column_name AS colnames, data_type AS types
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'table_individu'"
)
print(schema_table_individu)
```

	nom_variable	description_variable	type_variable	code_modalite	descri
0	ACHL	Période d'achèvement de la construction de la ...	CHAR	A11	Avant
1	ACHL	Période d'achèvement de la construction de la ...	CHAR	A12	De 19

```
<IPython.core.display.HTML object>
```

```
documentation = transpose(documentation_raw)
```

Pour découvrir les informations présentes dans la base, il est possible d'utiliser les fonctions pré-implémentées de `DuckDB` pour la manipulation de données textuelles. Par exemple, pour extraire les variables dont la description contient le terme *"catégorie"*:

Python

```python
duckdb.sql("SELECT * FROM documentation WHERE CONTAINS(description_variable, 'Catégorie')"
```

R

```r
query <- "SELECT * FROM documentation WHERE CONTAINS(description_variable, 'Catégorie')"
dbGetQuery(con, query)
```

```
nom_variable   …   code_modalite   description_modalite
  varchar              varchar            varchar

CATIRIS        …   A               Activité
CATIRIS        …   D               Divers
CATIRIS        …   H               Habitat
CATIRIS        …   Z               Commune non découp…
CATL           …   1               Résidences princip…
CATL           …   2               Logements occasion…
CATL           …   3               Résidences seconda…
CATL           …   4               Logements vacants

8 rows                               5 columns (3 shown)
```

Cette approche peut permettre de récupérer les modalités d'une variable. Dans cette base de données, les valeurs `Z` sont à part. Il est possible d'avoir du détail sur celles-ci avec la requête suivante

Python

```
duckdb.sql(
  "SELECT * FROM documentation WHERE CONTAINS(code_modalite, 'Z')"
)
```

R

```
query <- "SELECT * FROM documentation WHERE CONTAINS(code_modalite, 'Z')"
dbGetQuery(con, query)
```

```
nom_variable   …   code_modalite   description_modalite
  varchar              varchar            varchar

ARM              …   ZZZZZ            Sans objet
BAIN             …   Z                Logement ordinaire…
BATI             …   Z                Logement ordinaire…
CATIRIS          …   Z                Commune non découp…
CHAU             …   Z                Logement ordinaire…
CHFL             …   Z                Logement ordinaire…
CHOS             …   Z                Logement ordinaire…
CLIM             …   Z                Logement ordinaire…
CMBL             …   Z                Logement ordinaire…
CUIS             …   Z                Logement ordinaire…
 .               .   .                          .
 .               .   .                          .
 .               .   .                          .
ILETUDM          …   Z                Sans objet (pas d'…
ILTM             …   Z                Sans objet (sans e…
IRIS             …   ZZZZZZZZZ        Commune non découp…
RECHM            …   Z                Sans objet (en emp…
SANI             …   Z                Logement ordinaire…
SANIDOM          …   ZZ               Logement ordinaire…
TPM              …   Z                Sans objet (sans e…
TRANSM           …   Z                Sans objet
TRIRIS           …   ZZZZZZ           Commune non découp…
WC               …   Z                Logement ordinaire…

25 rows (20 shown)                    5 columns (3 shown)
```

Lecture et affichage de quelques valeurs

Pour visualiser la table, deux approches :

- Sélectionner un échantillon restreint sur les premières lignes du `Parquet`, par exemple les 5 premières lignes ;
- Sélectionner un échantillon aléatoire.

Python

```
duckdb.sql("SELECT * FROM table_logement LIMIT 5")
```

R

```
dbGetQuery(
  con,
  "SELECT * FROM table_logement LIMIT 5"
)
```

```
┌─────────┬─────────┬────────────┬───┬─────────┬─────────┬─────────┬─────────┐
│ COMMUNE │   ARM   │    IRIS    │ … │  TYPC   │  TYPL   │  VOIT   │   WC    │
│ varchar │ varchar │  varchar   │   │ varchar │ varchar │ varchar │ varchar │
├─────────┼─────────┼────────────┼───┼─────────┼─────────┼─────────┼─────────┤
│ 01001   │ ZZZZZ   │ ZZZZZZZZZ  │ … │ 1       │ 1       │ 2       │ Z       │
│ 01001   │ ZZZZZ   │ ZZZZZZZZZ  │ … │ 2       │ 1       │ 2       │ Z       │
│ 01001   │ ZZZZZ   │ ZZZZZZZZZ  │ … │ 2       │ 1       │ 2       │ Z       │
│ 01001   │ ZZZZZ   │ ZZZZZZZZZ  │ … │ 1       │ 1       │ 1       │ Z       │
│ 01001   │ ZZZZZ   │ ZZZZZZZZZ  │ … │ Y       │ 6       │ X       │ Z       │
└─────────┴─────────┴────────────┴───┴─────────┴─────────┴─────────┴─────────┘
5 rows                                                  69 columns (7 shown)
```

Python

```
duckdb.sql("SELECT * FROM table_logement USING SAMPLE 5")
```

R

```
dbGetQuery(
  con,
  "SELECT * FROM table_logement USING SAMPLE 5"
)
```

```
FloatProgress(value=0.0, layout=Layout(width='auto'), style=ProgressStyle(bar_color='black'))
```

```
┌─────────┬─────────┬──────────┬───┬─────────┬─────────┬─────────┐
│ COMMUNE │   ARM   │   IRIS   │ … │  TYPC   │  TYPL   │  VOIT   │
│ varchar │ varchar │ varchar  │   │ varchar │ varchar │ varchar │
├─────────┼─────────┼──────────┼───┼─────────┼─────────┼─────────┤
│ 87001   │ ZZZZZ   │ 870010101 │ … │ 1       │ 1       │ 1       │
│ 59507   │ ZZZZZ   │ 595070101 │ … │ 3       │ 2       │ 0       │
│ 83016   │ ZZZZZ   │ 830160101 │ … │ 3       │ 2       │ 1       │
│ 40184   │ ZZZZZ   │ 401840101 │ … │ 1       │ 1       │ 2       │
│ 33422   │ ZZZZZ   │ 334220102 │ … │ 1       │ 1       │ 2       │
└─────────┴─────────┴──────────┴───┴─────────┴─────────┴─────────┘
5 rows                                          69 columns (7 shown)
```

Sélectionner des observations ou des variables

Requêtes sur les colonnes (`SELECT`)

La liste des colonnes à extraire du fichier peut être renseignée avec le verbe `SELECT`. Celles-ci peuvent être renommées en appliquant au passage `AS`

Python

```
duckdb.sql("SELECT IPONDI AS poids, AGED, VOIT FROM table_individu LIMIT 10")
```

R

```
dbGetQuery(
  con,
  "SELECT IPONDI AS poids, AGED, VOIT FROM table_individu LIMIT 10"
)
```

```
      poids         AGED     VOIT
     double         int32   varchar

0.865065781333387      74   0
 4.98793114865391      39   1
 4.98793114865391       9   1
 5.01354653482522      55   2
 3.47836794972965       1   0
 3.47836794972965      43   0
 3.47836794972965      38   0
 1.00358526972379      46   1
 1.00358526972379      16   1
0.984086854919485      59   2

10 rows                   3 columns
```

DuckDB offre également des fonctionnalités pour extraire des colonnes à travers des expressions régulières. Plus d'exemples peuvent être trouvés sur cette page.

```
      poids         AGED    AGER20    AGEREV   AGEREVQ
     double         int32   varchar   int32    varchar

0.865065781333387      74   79            73   70
 4.98793114865391      39   39            38   35
 4.98793114865391       9   10             8   5
 5.01354653482522      55   54            54   50
 3.47836794972965       1   2              0   0
 3.47836794972965      43   54            42   40
 3.47836794972965      38   39            37   35
 1.00358526972379      46   54            45   45
 1.00358526972379      16   17            15   15
```

```
0.984086854919485      59   64            58   55

10 rows                                        5 columns
```

Requêtes sur les lignes (`WHERE`)

Pour extraire un sous-échantillon des données complètes, la clause `WHERE` permet d'appliquer des filtres à partir de conditions logiques. Par exemple, il est possible de ne conserver, du fichier national, que les données de l'Aude (11), la Haute-Garonne (31) et l'Hérault (34).

`Python`

```python
duckdb.sql("SELECT * FROM table_individu WHERE DEPT IN ('11', '31', '34')")
```

Il est également possible de formatter cette liste telle qu'attendue par SQL à partir d'une liste `Python` plus classique. Pour cela, le code suivant peut servir de modèle:

```python
liste_regions = ["11", "31", "34"]
liste_regions_sql = ", ".join([f"'{dep}'" for dep in liste_regions])
duckdb.sql(f"SELECT * FROM table_individu WHERE DEPT IN ({liste_regions_sql})")
```

`R`

```r
liste_regions <- c("11", "31", "34")
liste_regions_sql <- glue_collapse(lapply(liste_regions, function(dep) glue("'{dep}'")), "
query <- glue("SELECT * FROM table_individu WHERE DEPT IN ({liste_regions_sql})")
dbGetQuery(con, query)
```

```
FloatProgress(value=0.0, layout=Layout(width='auto'), style=ProgressStyle(bar_color='black'))
```

CANTVILLE	NUMMI	ACHLR	AEMMR	…	TYPMR	VOIT	WC
varchar	varchar	varchar	varchar		varchar	varchar	varchar
1101	1	1	6	…	32	2	Z
1101	1	1	6	…	32	2	Z
1101	2	3	8	…	32	2	Z

1101	2	3	8	…	32	2	Z
1101	3	6	9	…	41	3	Z
1101	3	6	9	…	41	3	Z
1101	3	6	9	…	41	3	Z
1101	4	1	7	…	20	2	Z
1101	4	1	7	…	20	2	Z
1101	5	6	9	…	41	2	Z
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
1105	1666	5	9	…	32	0	Z
1105	1666	5	9	…	32	0	Z
1105	1667	1	4	…	12	1	Z
1105	1668	6	9	…	41	2	Z
1105	1668	6	9	…	41	2	Z
1105	1668	6	9	…	41	2	Z
1105	1669	1	9	…	11	1	Z
1105	1670	5	9	…	41	2	Z
1105	1670	5	9	…	41	2	Z
1105	1670	5	9	…	41	2	Z

? rows (>9999 rows, 20 shown) 88 columns (7 shown)

Les filtres sur les observations peuvent être faits à partir de critères sur plusieurs colonnes. Par exemple, pour ne conserver que les observations de la ville de Nice où la date d'emménagement est postérieure à 2020, la requête suivante peut être utilisée:

Python

```
duckdb.sql("SELECT * FROM table_logement WHERE COMMUNE = '06088' and AEMM > 2020")
```

R

```
dbGetQuery(
  con,
  "SELECT * FROM table_logement WHERE COMMUNE = '06088' and AEMM > 2020"
)
```

Statistiques agrégées

Le langage `SQL` permet d'exécuter de manière très efficace des requêtes complexes afin de construire, à partir de données fines, des statistiques agrégées.

Cette partie illustre d'abord ceci avec deux exemples de statistiques agrégées renvoyant une unique statistique :

- Extraire la liste des codes arrondissements de Paris, Lyon, Marseille où au moins une personne a été recensée ;
- Reproduire l'exemple de (Mauvière 2022) permettant de calculer le nombre d'habitants de Toulouse qui ont changé de logement en un an;

Ensuite, des statistiques plus fines sont construites par le biais d'agrégations par groupe:

- Calculer le nombre de personnes recensées par cohorte pour les départements de l'Aude (11), de la Haute-Garonne (31) et de l'Hérault (34) ;
- Calculer le nombre de centenaires recensés par département.

Python

```
duckdb.sql("SELECT DISTINCT(ARM) FROM table_logement WHERE ARM IS NOT NULL")
```

R

```
dbGetQuery(
  con,
  "SELECT DISTINCT(ARM) FROM table_logement WHERE ARM IS NOT NULL"
)
```

```
  ARM
varchar

75112
75109
69389
13215
75103
69381
```

```
 69383
 13212
 13214
 13208
   .
   .
   .
 75115
 75116
 69385
 69387
 13202
 13204
 13206
 13211
 13213
 75120

  46 rows
 (20 shown)
```

Python

```python
duckdb.sql("SELECT CAST(SUM(IPONDL) AS INT) AS habitants_toulouse_demenagement FROM table_
```

R

```r
query <- "SELECT CAST(SUM(IPONDL) AS INT) AS habitants_toulouse_demenagement FROM table_lo
dbGetQuery(con, query)
```

	habitants_toulouse_demenagement
0	91195

Python

```
import matplotlib.pyplot as plt
from plotnine import *

pyramide_ages = duckdb.sql(
"""
SELECT
  SUM(IPONDI) AS individus,
  AGED,
  DEPT AS departement
FROM table_individu
  WHERE DEPT IN ('11', '31', '34')
GROUP BY AGED, DEPT ORDER BY DEPT, AGED
"""
).to_df()


(
    ggplot(pyramide_ages, aes(x = "AGED", y = "individus")) +
    geom_bar(aes(fill = "departement"), stat = "identity", show_legend=False) + geom_vline
    theme_minimal() +
    labs(y = "Individus recensés", x = "Âge")
)
```

R

```
library(ggplot)

query <- "SELECT SUM(IPONDI) AS individus, AGED, DEPT AS departement FROM table_individu W
dbGetQuery(con, query)

(
    ggplot(pyramide_ages, aes(x = "AGED", y = "individus")) +
    geom_bar(aes(fill = "departement"), stat = "identity", show_legend=False) + geom_vline
    theme_minimal() +
    labs(y = "Individus recensés", x = "Âge")
)
```

Si on s'intéresse plus spécifiquement au nombre de centenaires recensés par département et qu'on désire les classer par ordre décroissant

Python

```
duckdb.sql(
"""
SELECT
  SUM(IPONDI) AS individus_recenses,
  DEPT
FROM table_individu
  WHERE AGED >= 100
GROUP BY DEPT
ORDER BY individus_recenses DESC
"""
)
```

R

```
query <- "SELECT SUM(IPONDI) AS individus_recenses, DEPT FROM table_individu WHERE AGED >=
dbGetQuery(con, query)
```

individus_recenses	DEPT
int64	varchar
470	75
322	13
281	06
261	69
241	92
239	83
216	33
183	31
183	44
177	59
.	.
.	.
.	.
25	46
23	70
23	08
21	15

```
21   05
20   23
17   55
14   90
12   973
11   48

100 rows (20 shown)
```

Associer à d'autres sources de données

Le *code officiel géographique* (COG) est utile pour illuster l'ajout d'information annexe. Le code commune va être utilisé pour associer les deux bases de données. Cette variable porte des noms différents dans les deux bases, ce qui n'est pas un problème.

Il est proposé, ci-dessous, de télécharger les données de manière reproductible via une fonction adaptée (ici à travers le *package* `requests` pour `Python` ou via `download.file` en `R`). Bien que `DuckDB` permette l'import direct depuis un *url*, ceci implique l'installation en amont de l'extension `httpfs`.

Python

```python
import requests
import os

url_cog = "https://www.insee.fr/fr/statistiques/fichier/6800675/v_commune_2023.csv"
if os.path.exists("cog.csv") is False:
  response = requests.get(url_cog)
  with open("cog.csv", mode="wb") as file:
      file.write(response.content)
```

```python
duckdb.sql('CREATE OR REPLACE VIEW cog2023 AS SELECT * FROM read_csv_auto("cog.csv", heade
```

```python
duckdb.sql(
"""
SELECT cog2023.NCCENR, CAST(SUM(table_logement.IPONDL) AS INT) AS recenses
FROM table_logement
LEFT OUTER JOIN cog2023 ON table_logement.COMMUNE = cog2023.COM
GROUP BY cog2023.NCCENR
```

```
ORDER BY recenses;
"""
)
```

R

```
query <- "SELECT cog2023.NCCENR, CAST(SUM(table_logement.IPONDL) AS INT) AS recenses FROM
dbGetQuery(con, query)
```

NCCENR	recenses
varchar	int32
Cumières-le-Mort-Homme	1
Épécamps	3
Leménil-Mitry	3
Rouvroy-Ripont	4
Maroncourt	4
Ornes	4
Aingoulaincourt	5
Bâtie-des-Fonds	6
Casterets	6
Molring	7
.	.
.	.
.	.
Plessis-Lastelle	151
Saint-Outrille	151
Peyrouse	151
Dieulivol	151
Genettes	151
Formentin	151
Hôtellerie	151
Rotherens	151
Thennelières	151
Frasnois	151

? rows (>9999 rows, 20 shown)

Dondon, Alexis, and Pierre Lamarche. 2023. “Quels Formats Pour Quelles Données?” *Courrier Des Statistiques*, no. 9.

Mauvière, Éric. 2022. “Parquet Devrait Remplacer Le Format CSV.” Post de blog [Consulté le 12 octobre 2023]. 2022. https://www.icem7.fr/cartographie/parquet-devrait-remplacer-le-format-csv/.