Guide d'utilisation des données du recensement de la population au format `Parquet`

2023-12-22

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).

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.

L'ensemble des codes utilisés pour produire cette note est disponible sur le dépôt `Github` InseeFrLab/exemples-recensement-parquet au format `Quarto Markdown`. Une version plus ergonomique et présentant des éléments complémentaires (`DuckDB` par le biais d'`Observable` et `Quarto`, visualisations réactives), est disponible sur le blog du réseau des *data scientists* de la statistique publique.

> **Note**
>
> Cette note au format `PDF` présente de manière linéaire les exemples de code `R` et `Python` pour exploiter les fichiers détails du recensement au format `Parquet`.
> Pour bénéficier d'une expérience plus ergonomique et d'exemples supplémentaires liés au langage `Javascript`, il est possible de consulter la documentation publiée sur le blog du SSP Hub.

1 Initialisation

Les pages d'informations sur les données, où sont notamment disponibles la documentation de celles-ci, se retrouvent sur le site `insee.fr` aux adresses suivantes:

- Fichier détail individuel
- Fichier détail logement

Ces pages présentent aussi les données détaillées au format `CSV`. Néanmoins, le format `Parquet` est plus intéressant pour le traitement de celles-ci comme expliqué par Dondon and Lamarche (2023). Les données au format `Parquet` sont mises à disposition sur le site `data.gouv` aux adresses suivantes:

- Fichier détail individus
- Fichier détail logement

Ces fichiers peuvent être téléchargés par la biais de `Python` ou de `R`. Dans la suite de ce guide, il sera fait l'hypothèse que les données sont téléchargées par le biais du code ci-dessous et stockées dans le dossier de travail utilisé par `Python` ou `R`.

Exemple `Python`

```python
import requests
import os

def download_file(url, filename):
    if not os.path.exists(filename):
        response = requests.get(url)
        with open(filename, 'wb') as f:
            f.write(response.content)

url_table_logement = (
    "https://static.data.gouv.fr/resources/"
    "recensement-de-la-population-fichiers-detail-logements-ordinaires-en-2020-1/"
    "20231023-123618/fd-logemt-2020.parquet"
)
url_table_individu = (
    "https://static.data.gouv.fr/resources/"
    "recensement-de-la-population-fichiers-detail-individus"
    "-localises-au-canton-ou-ville-2020-1"
    "/20231023-122841/fd-indcvi-2020.parquet"
)
url_doc_logement = (
```

```
    "https://www.data.gouv.fr/fr/datasets/r/"
    "c274705f-98db-4d9b-9674-578e04f03198"
)
url_doc_individu = (
    "https://www.data.gouv.fr/fr/datasets/r/"
    "1c6c6ab2-b766-41a4-90f0-043173d5e9d1"
)

download_file(url_table_logement, "FD_LOGEMT_2020.parquet")
download_file(url_table_individu, "FD_INDCVI_2020.parquet")
download_file(url_doc_logement, "dictionnaire_variables_logemt_2020.csv")
download_file(url_doc_individu, "dictionnaire_variables_indcvi_2020.csv")
```

Exemple `R`

```
url_table_logement <- paste("https://static.data.gouv.fr/resources/",
    "recensement-de-la-population-fichiers-detail-logements-ordinaires-en-2020-1/",
    "20231023-123618/fd-logemt-2020.parquet"
url_table_individu <- paste("https://static.data.gouv.fr/resources",
    "recensement-de-la-population-fichiers-detail-individus-",
    "localises-au-canton-ou-ville-2020-1/20231023-122841/fd-indcvi-2020.parquet"
url_doc_logement <- paste("https://www.data.gouv.fr/fr/datasets/r/",
    "c274705f-98db-4d9b-9674-578e04f03198"
url_doc_individu <- paste("https://www.data.gouv.fr/fr/datasets/r/",
    "1c6c6ab2-b766-41a4-90f0-043173d5e9d1"

options(timeout = max(300, getOption("timeout")))

if (!file.exists("FD_LOGEMT_2020.parquet")){
    download.file(url_table_logement, "FD_LOGEMT_2020.parquet")
}
if (!file.exists("FD_INDCVI_2020.parquet")){
    download.file(url_table_individu, "FD_INDCVI_2020.parquet")
}
if (!file.exists("dictionnaire_variables_logemt_2020.csv")){
    download.file(url_doc_logement, "dictionnaire_variables_logemt_2020.csv")
}
if (!file.exists("dictionnaire_variables_indcvi_2020.csv")){
    download.file(url_doc_individu, "dictionnaire_variables_indcvi_2020.csv")
}
```

Il est proposé, pour initialiser la connexion entre les données `Parquet` et le langage client (`R`

ou `Python`) d'utiliser des vues. Ceci permet de faire référence de manière répétée à la même source de données par le biais d'un alias (`table_logement` ou `table_individu`).

Exemple `Python`

```
import duckdb

duckdb.sql('''
  CREATE OR REPLACE VIEW table_individu
  AS SELECT * FROM read_parquet("FD_INDCVI_2020.parquet")
'''
)

duckdb.sql('''
  CREATE OR REPLACE VIEW table_logement
  AS SELECT * FROM read_parquet("FD_LOGEMT_2020.parquet")
'''
)

duckdb.sql('''
  CREATE OR REPLACE VIEW documentation_indiv
  AS 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,
     LONG_VAR as longueur_variable
  FROM read_csv_auto("dictionnaire_variables_indcvi_2020.csv", header=true)
'''
)

duckdb.sql('''
  CREATE OR REPLACE VIEW documentation_logement
  AS 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,
     LONG_VAR as longueur_variable
  FROM read_csv_auto("dictionnaire_variables_logemt_2020.csv", header=true)
'''
)
```

Exemple R

```r
library(duckdb)
library(glue)

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

path_data_sql <- DBI::SQL(path_data)


renommage_documentation <-  DBI::SQL(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,",
"LONG_VAR as longueur_variable"
))


dbExecute(
  con,
  glue_sql(
    'CREATE OR REPLACE VIEW table_individu AS ',
    'SELECT * FROM read_parquet("{path_data_sql}/FD_INDCVI_2020.parquet")',
    .con=con
  )
)

dbExecute(
  con,
  glue_sql(
    'CREATE OR REPLACE VIEW table_logement AS ',
    'SELECT * FROM read_parquet("{path_data_sql}/FD_LOGEMT_2020.parquet")',
    .con=con
  )
)

dbExecute(
  con,
```

```
  glue_sql(
  'CREATE OR REPLACE VIEW documentation_indiv AS ',
  '{renommage_documentation} FROM ',
  'read_csv_auto("{path_data_sql}/dictionnaire_variables_indcvi_2020.csv", header=true)',
  .con=con
  )
)

dbExecute(
  con,
  glue_sql(
  'CREATE OR REPLACE VIEW documentation_logement AS ',
  '{renommage_documentation} FROM ',
  'read_csv_auto("{path_data_sql}/dictionnaire_variables_logemt_2020.csv", header=true)',
  .con=con
  )
)
```

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

Exemple `Python`

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

schema_table_logement = duckdb.sql(
  "SELECT * FROM documentation_logement"
  ).to_df()
display(schema_table_logement.head(2))
```

Exemple `R`

```
schema_table_individu <- dbGetQuery(
  con,
  "SELECT * FROM documentation_indiv"
)
print(head(schema_table_individu))
```

```r
schema_table_logement <- dbGetQuery(
  con,
  "SELECT * FROM documentation_logement"
)
print(head(schema_table_logement))
```

	code_modalite	description_modalite	nom_variable
0	1	Avant 1919	ACHLR
1	2	De 1919 à 1945	ACHLR

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 toutes les modalités des variables dont la description contient le terme *"catégorie"*:

Exemple `Python`

```python
query = \
"""
    SELECT * FROM documentation_logement
    WHERE CONTAINS(description_variable, 'Catégorie')
"""
duckdb.sql(query)
```

Exemple `R`

```r
query <- paste(
    "SELECT * FROM documentation_logement ",
    "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        X              "IRIS de moins de 200 habitants (fichier individus "localisat
CATIRIS        Z              Commune non découpée en IRIS
CATL           1              Résidences principales
CATL           Z              Hors logement ordinaire
CATPC          0              Population des ménages
CATPC          1              Population des communautés
CATPC          2              Population des habitations mobiles, sans abri, bateliers
CS1            1              Agriculteurs exploitants
CS1            2              Artisans, commerçants et chefs d'entreprise
CS1            3              Cadres et professions intellectuelles supérieures
CS1            4              Professions Intermédiaires
CS1            5              Employés
CS1            6              Ouvriers
CS1            7              Retraités
CS1            8              Autres personnes sans activité professionnelle

18 rows
```

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 :

Exemple `Python`

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

Exemple `R`

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

```
nom_variable   code_modalite            description_modalite
  varchar         varchar                      varchar

ACHLR          Z              Hors logement ordinaire
AEMMR          Z              Hors logement ordinaire
ANEMR          ZZ             Hors logement ordinaire
APAF           Z              Hors logement ordinaire
ARM            ZZZZZ          Sans objet
ASCEN          Z              Hors logement ordinaire
BAIN           Z              Logement ordinaire France métropolitaine
BATI           Z              Logement ordinaire France métropolitaine
CATIRIS        Z              Commune non découpée en IRIS
CATL           Z              Hors logement ordinaire
.              .                  .
.              .                  .
.              .                  .
TP             Z              Sans objet
TRANS          Z              Sans objet
TRIRIS         ZZZZZZ         Commune non découpée en TRIRIS
TYPC           Z              Hors logement ordinaire
TYPFC          Z              Hors famille ou hors logement ordinaire
TYPL           Z              Hors logement ordinaire
TYPMC          Z              Hors logement ordinaire
TYPMR          ZZ             Hors logement ordinaire
VOIT           Z              Hors logement ordinaire
WC             Z              Logement ordinaire France métropolitaine

72 rows (20 shown)                                          3 columns
```

2 Lecture et affichage de quelques valeurs

Pour visualiser un nombre limité de valeurs, par exemple 5, deux approches sont possibles :

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

Pour les premières lignes, la commande à utiliser est `LIMIT`.

Exemple `Python`

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

Exemple `R`

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

COMMUNE	ARM	IRIS	TYPC	TYPL
varchar	varchar	varchar	varchar	varchar
01001	ZZZZZ	ZZZZZZZZZ	1	1
01001	ZZZZZ	ZZZZZZZZZ	2	1
01001	ZZZZZ	ZZZZZZZZZ	2	1
01001	ZZZZZ	ZZZZZZZZZ	1	1
01001	ZZZZZ	ZZZZZZZZZ	Y	6

Pour un échantillon aléatoire, la commande à utiliser est `USING SAMPLE`.

Exemple `Python`

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

Exemple `R`

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

COMMUNE	ARM	IRIS	TYPC	TYPL
varchar	varchar	varchar	varchar	varchar
76497	ZZZZZ	764970102	3	2
31499	ZZZZZ	314990101	1	1
44184	ZZZZZ	441840103	1	1

```
27528     ZZZZZ     ZZZZZZZZZ   1         1
93051     ZZZZZ     930510106   3         2
```

3 Sélectionner des observations ou des variables

3.1 Requêtes sur les colonnes (`SELECT`)

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

Exemple `Python`

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

Exemple `R`

```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 propose également des fonctionnalités pour extraire des colonnes à travers des expressions régulières. De nombreux exemples peuvent être trouvés sur cette page.

Exemple Python

```python
duckdb.sql("SELECT IPONDI AS poids, COLUMNS('.*AGE.*') FROM table_individu LIMIT 10")
```

Exemple R

```r
dbGetQuery(
  con,
  "SELECT IPONDI AS poids, COLUMNS('.*AGE.*') FROM table_individu LIMIT 10"
)
```

```
      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
```

3.2 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), de la Haute-Garonne (31) et de l'Hérault (34).

Exemple Python

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

Exemple `R`

```
dbGetQuery(
  con,
  "SELECT * FROM table_individu WHERE DEPT IN ('11', '31', '34')"
)
```

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

Exemple `Python`

```
con = duckdb.connect()

con.execute('''
  CREATE OR REPLACE VIEW table_individu
  AS SELECT * FROM read_parquet("FD_INDCVI_2020.parquet")
'''
)

liste_regions = ["11", "31", "34"]

dep_slots = ", ".join(["?" for _ in liste_regions])
query = "SELECT * FROM table_individu WHERE DEPT IN ({})".format(dep_slots)
liste_regions_sql = ", ".join([f"'{dep}'" for dep in liste_regions])
con.execute(query, liste_regions).fetchdf()
```

Exemple `R`

```
liste_regions <- c("11", "31", "34")
liste_regions_sql <- glue_sql_collapse(
  lapply(
    liste_regions, function(dep) glue_sql("'{`dep`}'", .con=con)
    ),
  ", "
)
query <- glue_sql(
  "SELECT * FROM table_individu WHERE DEPT IN ({liste_regions_sql})",
  .con=con
```

```
)
dbGetQuery(con, query)
```

CANTVILLE	NUMMI	ACHLR	DEPT
varchar	varchar	varchar	varchar
1101	1	1	11
1101	1	1	11
1101	2	3	11
1101	2	3	11
1101	3	6	11
1101	3	6	11
1101	3	6	11
1101	4	1	11
1101	4	1	11
1101	5	6	11
.	.	.	.
.	.	.	.
.	.	.	.
1105	1666	5	11
1105	1666	5	11
1105	1667	1	11
1105	1668	6	11
1105	1668	6	11
1105	1668	6	11
1105	1669	1	11
1105	1670	5	11
1105	1670	5	11
1105	1670	5	11

? rows (>9999 rows, 20 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 :

Exemple `Python`

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

Exemple R

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

```
COMMUNE     ARM       IRIS      AEMM
varchar   varchar    varchar    int32

06088     ZZZZZ     060880101    2022
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
06088     ZZZZZ     060880101    2021
  .         .           .          .
  .         .           .          .
  .         .           .          .
06088     ZZZZZ     060883701    2021
06088     ZZZZZ     060883701    2021
06088     ZZZZZ     060883701    2021
06088     ZZZZZ     060883701    2021
06088     ZZZZZ     060883701    2021
06088     ZZZZZ     060883801    2022
06088     ZZZZZ     060883801    2021
06088     ZZZZZ     060883801    2021
06088     ZZZZZ     060883801    2021
06088     ZZZZZ     060883801    2022

2692 rows (20 shown)        4 columns
```

4 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.

La fonction `DISTINCT` appliquée à la variable `ARM` permet d'extraire la liste des codes arrondissements présents dans la base de données.

Exemple `Python`

```python
query = "SELECT DISTINCT(ARM) " +\
    "FROM table_logement " +\
    "WHERE NOT CONTAINS(ARM, 'ZZZZZ') " +\
    "ORDER BY ARM"
duckdb.sql(query)
```

Exemple `R`

```r
query <- glue_sql(
    "SELECT DISTINCT(ARM) ",
    "FROM table_logement ",
    "WHERE NOT CONTAINS(ARM, 'ZZZZZ') ",
    "ORDER BY ARM",
    .con=con
)
dbGetQuery(con, query)
```

```
   ARM
 varchar

13201
13202
13203
13204
13205
13206
13207
13208
13209
13210
  .
  .
  .
75111
75112
75113
75114
75115
75116
75117
75118
75119
75120

 45 rows
(20 shown)
```

Il est possible d'extraire des statistiques beaucoup plus raffinées par le biais d'une requête SQL plus complexe. Par exemple pour calculer le nombre d'habitants de Toulouse qui ont changé de logement en un an:

Exemple `Python`

```
query = \
"""
SELECT CAST(
  SUM(IPONDL*CAST(INPER AS INT)) AS INT
```

```
) AS habitants_toulouse_demenagement
FROM table_logement
WHERE COMMUNE == '31555' AND IRANM NOT IN ('1', 'Z') AND INPER != 'Y'
"""
duckdb.sql(query).df()
```

Exemple `R`

```
query <- paste(
  "SELECT CAST(SUM(IPONDL*CAST(INPER AS INT)) AS INT) ",
  "AS habitants_toulouse_demenagement",
  "FROM table_logement",
  "WHERE COMMUNE == '31555' AND IRANM NOT IN ('1', 'Z') AND INPER != 'Y'",
  sep = " ")
dbGetQuery(con, query)
```

	habitants_toulouse_demenagement
0	86364

Pour représenter la pyramide des âges recensés dans ces trois départements, il est possible de procéder en deux étapes

- Effectuer une agrégation par le biais de `DuckDB` et transformer ces résultats sous forme de *dataframe*
- Utiliser ce *dataframe* avec un *package* d'analyse graphique pour représenter la pyramide des âges.

> **Note**
>
> Pour illustrer le parallélisme possible entre les codes `R` et `Python`, l'exemple de représentation graphique ci-dessus s'appuie sur le *package* `plotnine` - dont la syntaxe reproduit celle du *package* `R` `ggplot2`, plutôt que sur `matplotlib` ou `seaborn`.

Exemple `Python`

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

pyramide_ages = duckdb.sql(
```

```
"""
SELECT
  SUM(IPONDI) AS individus,
  CAST(AGED AS int) AS 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(xintercept = 18, color = "grey", linetype = "dashed") +
    facet_wrap('departement', scales = "free_y", nrow = 3) +
    theme_minimal() +
    labs(y = "Individus recensés", x = "Âge")
)
```

Exemple R

```
library(labeling)
library(ggplot2)

query <- paste(
  "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",
  sep = " "
)
pyramide_ages <- dbGetQuery(con, query)

ggplot(pyramide_ages, aes(x = AGED, y = individus)) +
  geom_bar(aes(fill = departement), stat = "identity") +
  geom_vline(xintercept = 18, color = "grey", linetype = "dashed") +
```

```
  facet_wrap(~departement, scales = "free_y", nrow = 3) +
  theme_minimal() +
  labs(y = "Individus recensés", x = "Âge")
```

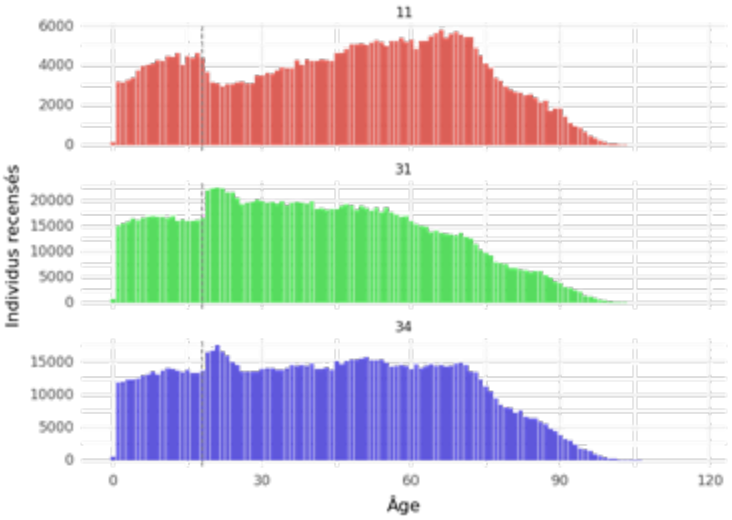

Figure 1: Un exemple de représentation graphique produite à partir du recensement de la population

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

Exemple `Python`

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

Exemple `R`

```r
query <- paste(
  "SELECT SUM(IPONDI) AS individus_recenses, DEPT",
  "FROM table_individu",
  "WHERE AGED >= 100",
  "GROUP BY DEPT",
  "ORDER BY individus_recenses DESC",
  sep = " "
)
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   04
                23   70
                23   08
                21   15
                21   05
                20   23
                17   55
                14   90
                12   973
                11   48

    100 rows (20 shown)
```

5 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`.

L'association de sources de données passe généralement par un `JOIN`. Pour illustrer cette clause, il est possible d'associer les agrégats de la table logement à un niveau communal avec celles du COG grâce au code commune.

Exemple `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)
```

```
duckdb.sql(
  'CREATE OR REPLACE VIEW cog2023 AS ' +\
  'SELECT * FROM read_csv_auto("cog.csv", header=true)'
)
```

```
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;
"""
)
```

Exemple `R`

```r
url <- "https://www.insee.fr/fr/statistiques/fichier/6800675/v_commune_2023.csv"
download.file(url, "cog.csv")

dbExecute(
  con,
  glue_sql(
    "CREATE OR REPLACE VIEW cog2023 AS ",
    "SELECT * FROM read_csv_auto("cog.csv", header=true)",
    .con=con
  )
)

query <- paste(
  "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",
  sep = " "
)
dbGetQuery(con, query)
```

NCCENR	recenses
varchar	int32
Cumières-le-Mort-Homme	1
Épécamps	3
Leménil-Mitry	3
Maroncourt	4
Rouvroy-Ripont	4
Ornes	4
Aingoulaincourt	5
Casterets	6
Bâtie-des-Fonds	6
Molring	7
.	.
.	.
.	.
Gimbrède	151
Bach	151
Montliot-et-Courcelles	151
Senargent-Mignafans	151

```
Saint-Denis-de-Vaux          151
Saint-Mard-de-Vaux           151
Morchain                     151
Vevy                         151
Champmotteux                 151
Montségur                    151

  ? rows (>9999 rows, 20 shown)
```

Références

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/.