# **Mini-projet** : Web des données et web sémantique

## 1 Données, importation, conversion

Les données que j'ai choisi pour ce projet sont celles de la collection complète d'openedition, une plateforme ayant pour but de rassembler de multiples ressources traitant des sciences sociales. Elles comprennent une liste de 13145 livres, ainsi que de nombreuses informations à leur sujet[1].

Les données ont été récupérées directement sur le site de l'organisation, sous la forme d'un fichier tableur. Après avoir bien fait attention à sélectionner l'encodage adapté (ici UTF-8), nous les avons exportées en CSV afin de faciliter les traitements à venir.

## 2 Traitement sous OpenRefine

Après avoir importé les données sur le logiciel OpenRefine, la première chose a été d'effectuer les traitements de base de conversion de types. Ainsi, le nombre de pages, l'année de publication ainsi que les prix ont été transformés, du type *string* par défaut à des *ints* et *doubles* selon la donnée.

Un choix à faire quant à ces données porte sur la façon dont nous allons modéliser les variables pouvant prendre plusieurs valeurs. Dans nos données, les auteurs et index ISI peuvent être multiples (jusqu'à 42 auteurs !), et s'offre à nous plusieurs choix pour représenter cela :

- L'utilisation de listes RDF, qui présuppose un ordonnancement des valeurs
- L'utilisation de propriétés instanciées plusieurs fois

Nous avons choisi la seconde option, qui est à la fois plus proche sémantiquement de ce que l'on veut exprimer, et plus simple à mettre en place. Malheureusement, le langage GREL d'OpenRefine ne permet pas d'effectuer la transformation intégrale, à cause d'un problème d'échappement et d'objet Java.

En utilisant le langage GREL d'OpenRefine, nous pouvons utiliser la formule suivante :

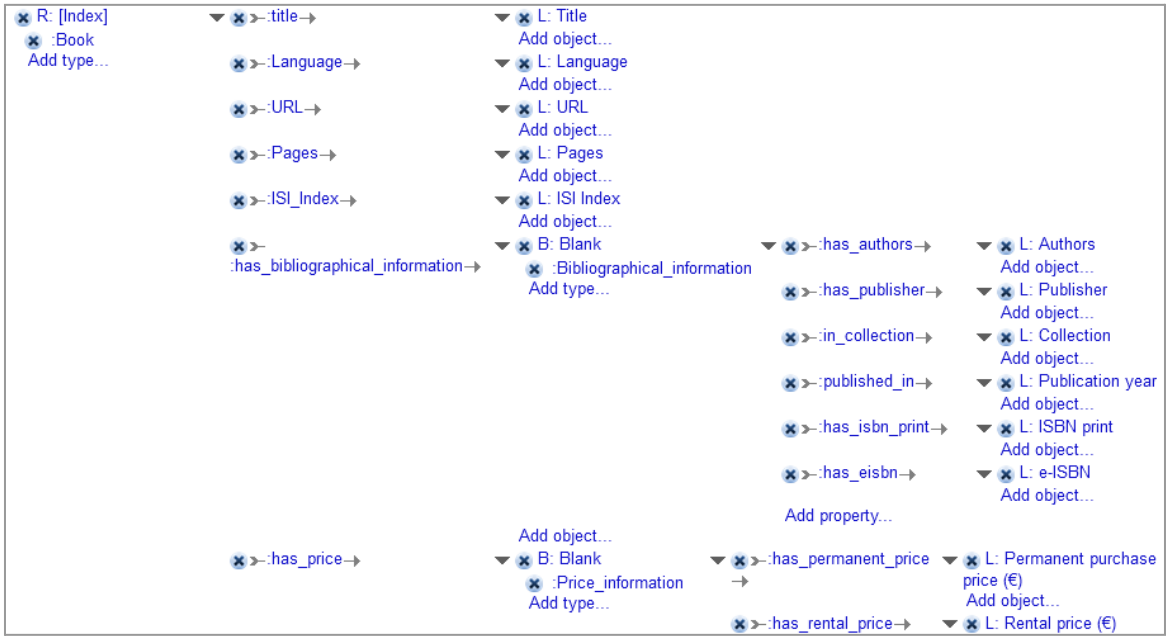
```
split(value.replace(/[^a-zA-Z0-9; ]/,""), ";").toString().replaceChars("[]", '').replace(', ', '\", \"')
```

Celle-ci nous permet de passer d'une liste d'auteurs séparés par des `;` à une liste séparée par des virgules entourées de `"`. Cela nous permet de grandement simplifier le

[1] Full list: Publisher, Title, Authors, Language, ISBN print, e-ISBN, Permanent purchase price (€), Rental price (€), URL, Collection, Number of pages, Publication year, ISI Index

traitement, et il nous faudra ainsi seulement ajouter des " de chaque côté de la chaîne finale pour transformer cela en accélérateur de propriétés multiples (nous verrons cela en détail dans la prochaine partie).

La dernière étape dans OpenRefine est de modéliser les données en RDF. Afin de créer de la structure dans nos données, nous avons créé deux nœuds blancs : Informations bibliographiques et Information de prix. Cela nous permet d'ajouter ces nouveaux types, et de regrouper certaines informations spécifiques sous ces nœuds. La modélisation finale est présentée dans l'image ci-dessous.




Une fois l'exportation réalisée, nous avons obtenu un fichier `.ttl`, que nous allions ensuite pouvoir utiliser pour continuer le projet. Avant cela, il nous a fallu définir nos classes et propriétés avec des triplets. Ainsi, nous avons défini la range et le domaine de nos propriétés, et nous avons typé toutes nos classes. Pour certaines, nous avons réutilisé des types d'ontologies déjà existantes (comme par exemple `dbo:Book`).

Nous avons également eu à utiliser un certain nombre de REGEX pour nettoyer le fichier et les quelques erreurs de formatage qui nous avaient échappé sur OpenRefine. Une fois cela réalisé, le fichier est enfin valide selon le RDF validator, et nous pouvons avancer.

*Les données finales peuvent être trouvées sur ce drive, afin d'alléger le git comportant le code source.*

# 3 Requête et visualisation

*Le code pour cette partie du projet peut être trouvé sur ce repository GitHub.*

## 3.1 Bibliothèque de composants de visualisation

Pour réaliser ce projet, nous avons utilisé plusieurs “frameworks” open source afin de nous aider dans notre travail.

d3sparql.js et d3.js sont deux bibliothèques de visualisation de données. La seconde, très populaire, permet de créer une multitude de graphiques avec un haut degré d’ajustement. La première, elle, est une surcouche permettant de simplifier la visualisation de requêtes sparql en d3.js.

mvp.css est un fichier de style minimaliste, qui nous a permi d’obtenir un site avec un design soigné sans passer trop de temps sur l’élaboration du css.

## 3.2 Exploration de nos propres données

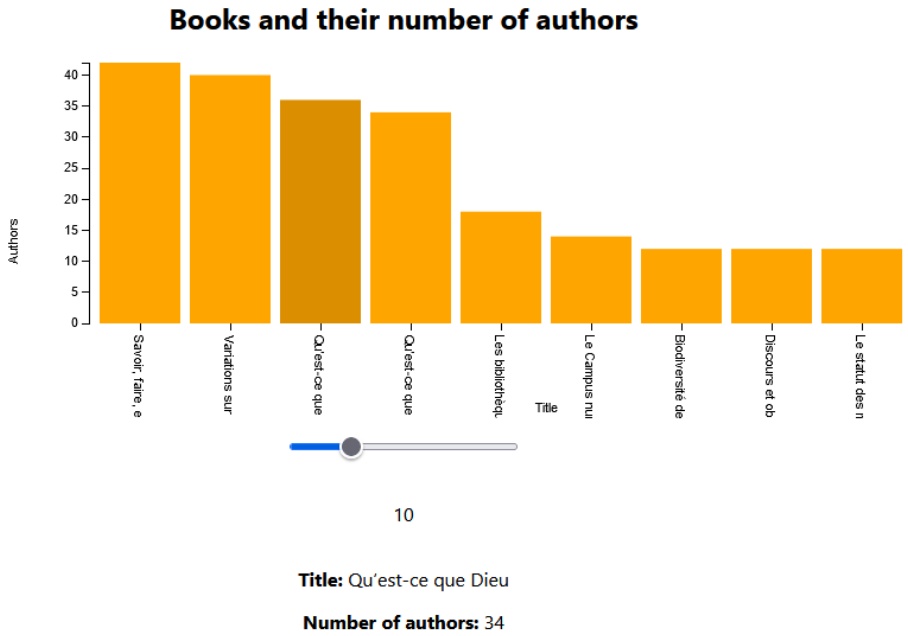
Dans le cadre du processus d'apprentissage de SparQL, nous commençons par des requêtes simples et courtes, traitant uniquement de nos données locales.

### 3.2.1 Statistiques sur le nombre d'auteurs

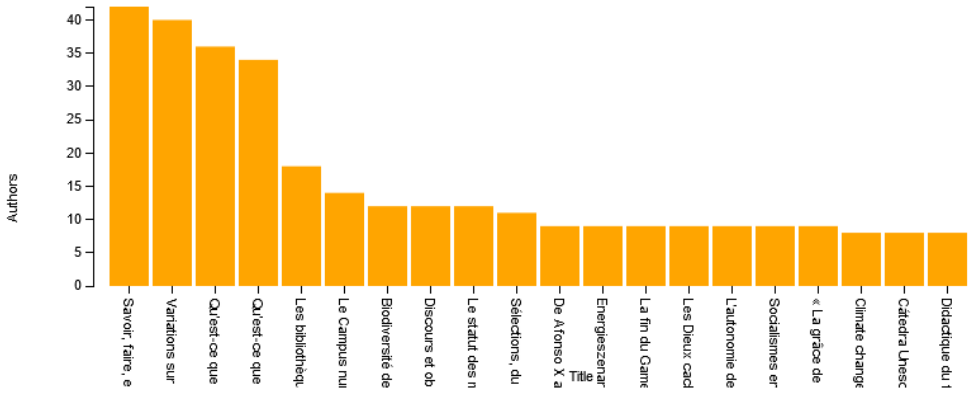
```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo:   <http://dbpedia.org/ontology/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://localhost:3333/>

SELECT ?t (COUNT(?p) as ?pCount) WHERE {
    ?b a dbo:Book.
    ?b dc:title ?t.
    ?b :has_bibliographical_information ?bb.
    ?bb :has_authors ?p.
}
GROUP BY ?t
HAVING (?pCount>${nb_authors})
ORDER BY DESC(?pCount)
```

La requête qui précède permet de récupérer tous les livres de nos données ayant plus d'auteurs que la variable `nb_authors`, puis de les ordonner par ordre décroissant. En injectant cette requête dans d3sparql.js, nous obtenons la visualisation suivante.




Pour ajouter de l'interaction, nous avons mis en place un slider permettant de montrer uniquement les livres ayant plus d'auteurs que la variable associée (on peut voir ci-dessus tous les livres ayant plus de 10 auteurs, et en dessous tous les livres ayant plus de 7 auteurs). Additionnellement, l'affichage du titre complet étant impossible avec cette visualisation, nous avons ajouté une fonctionnalité permettant de cliquer sur un livre pour afficher son titre et son nombre d'auteurs, afin de faciliter la lecture du graphique.



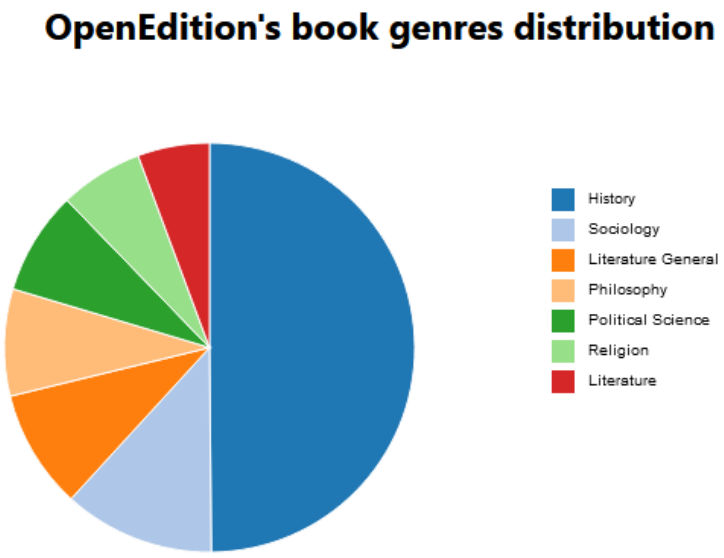
### 3.2.2 Statistiques sur le type de livre

Ensuite, nous avons compté les types de livres d'openEdition. Il y a `22520` différents types de balises dans openEdition. Et les 7 plus fréquentes sont :

| | Type | Times |
|---|---|---|
| 1 | History | "4188" |
| 2 | Sociology | "1000" |
| 3 | Literature General | "790" |
| 4 | Philosophy | "709" |
| 5 | Political Science | "685" |
| 6 | Religion | ”551“ |
| 7 | Literature | “475” |

Nous avons représenté les résultats de cette recherche sous la forme d'un pie chart.

```
SELECT ?booktype (count(?booktype) as ?tcount)
WHERE{
  ?book a dbo:Book.
  ?book :has_isi ?booktype
}
group by ?booktype
order by desc(?tcount)
limit 7 `
```



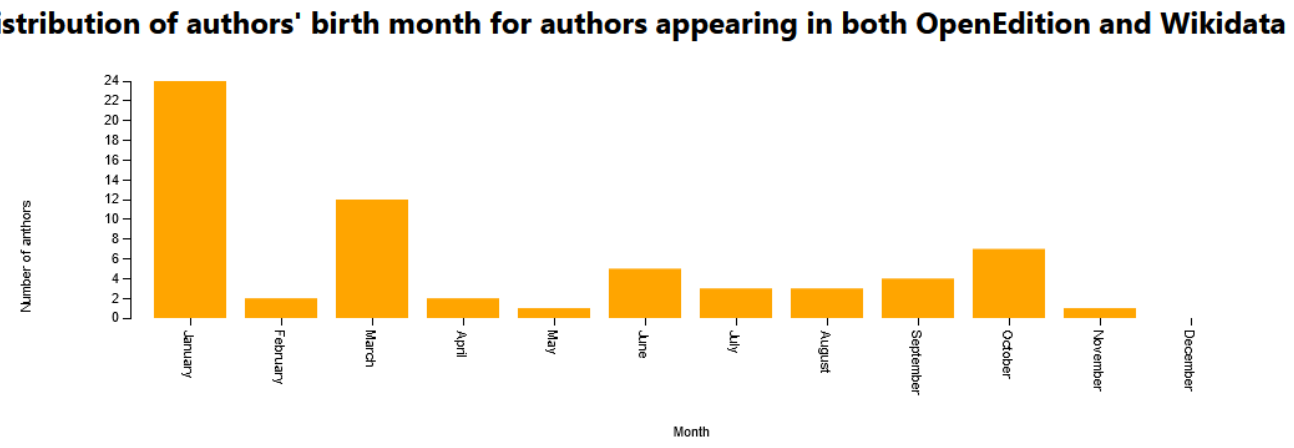
## 3.3 Connexion à des données externes

Après les plusieurs explorations ci-dessus sur nos ensemble de données, nous avons continué de connecter notre ensemble de données avec des données externes pour obtenir des informations plus riches et plus intéressantes.

### 3.3.1 Liens vers les wikidata

Nous avons d'abord essayé de rechercher les livres de notre rdf file dans wikidata. Cependant, nous avons constaté que nos données ne pouvaient pas être très liées à wikidata. Nous avons d'abord une compréhension macroscopique des données wikidata Nous avons trouvé 45588 entités dans les wikidata pour le livre (Q571). Mais seuls 20 000 d'entre eux ont un titre(P1476). Et il n'y a que 13 000 livres où le nom complet de l'auteur peut être trouvé. En outre, dans notre ensemble de données, le nom de l'auteur est une combinaison du nom de famille et du prénom, alors que la plupart des auteurs dans les wikidata contiennent le second prénom, etc. Cela a rendu notre recherche encore plus difficile.

Enfin, nous avons fusionné le nom de famille et prénom dans wikidata pour qu'ils correspondent au nom de l'auteur de openEdition. Nous avons finalement trouvé 64 auteurs qui apparaissent comme des entités dans les données de wikidata. Cela nous a permis de récupérer des informations supplémentaires sur eux. Ici, nous avons extrait les mois de naissance de ces auteurs et visualisé la distribution de ceux-ci.
Coordonnée horizontale : mois
Coordonnée verticale : nombre d'auteurs nés au cours de ce mois




(requête page suivante)

```
SELECT ?book_wiki
WHERE{
  # ?book_wiki is an entity of book
  ?book_wiki wdt:P31 wd:Q571.
}
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX ps: <http://www.wikidata.org/prop/statement/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX : <http://localhost:3333/>

SELECT  ?birth (count(?birth) as ?bcount)
WHERE {
  SERVICE <https://query.wikidata.org/sparql> {
    SELECT ?book_wiki ?title_wiki (concat(?family_name_label,'
',?given_name_label,' dir') as ?auth) (month(?birth_wiki) as ?birth)
    WHERE {
      # ?book_wiki is a book
      ?book_wiki wdt:P31 wd:Q571.
      ?book_wiki wdt:P1476 ?title_wiki.
      # ?book_wiki is author is auth_wiki
      ?book_wiki wdt:P50 ?auth_wiki.
      # the author has given_name
      ?auth_wiki wdt:P735 ?given_name.
      ?given_name wdt:P1705 ?given_name_label.
      # the author has family_name
      ?auth_wiki wdt:P734 ?family_name.
      ?family_name wdt:P1705 ?family_name_label.
      ?auth_wiki wdt:P569 ?birth_wiki
    }
   }
  ?book a dbo:Book.
  ?book :has_bibliographical_information ?bb.
  ?bb :has_authors ?auth.
}
group by ?birth
order by ?birth
```