Introduction à OpenRefine

Vous pouvez le télécharger à cette addresse http://openrefine.org/ (http://openrefine.org/)

Pourquoi utiliser OpenRefine?

1. pour garder une trace de ce qu'on a fait à nos données
2. parce qu'on peut annuler une action facilement
3. parce qu'OpenRefine ne modifie pas le fichier original, il crée une copie
4. parce qu'on peut sauvegarder des routines et le appliquer à d'autres fichiers
5. parce qu'il contient des algorithmes d'aggrégation puissants

Quelques notes sur OpenRefine

1. OpenRefine a été initialement conçu par Google sous le nom de **Google Refine** et lorsque le financement a été épuisé pour le projet, il est devenu un projet ouvert sous le nom d'OpenRefine. Si vous cherchez de l'aide en ligne, les deux noms peuvent vous retourner de l'information utile.
2. OpenRefine est une application Java et **une *sorte* de Java** a été développé spécialement pour OpenRefine. Il s'agit d'un langage appelé ***GREL (General Refine Expression Language)***.
3. Il existe des **groupes en ligne** axés sur l'utilisation d'Open refine:

https://groups.google.com/forum/?hl=en#!forum/openrefine (https://groups.google.com/forum/?hl=en#!forum/openrefine)

> **LA DOC EST PAS SUPER - ça vaut la peine de regarder les groupes de discussion**

1. OpenRefine est relativement performant jusqu'à une concurrence de **100 000 lignes**. Il est peu recommandé d'utiliser OpenRefine sur des fichiers de millions de lignes pour des questions de performance.
2. Comme le projet est *open source*, il existe plusieurs versions modifiées d'OpenRefine qui permette notamment d'utiliser des fichiers de millions de lignes ou de faire les manipulations en parallèle.

Installer OpenRefine

Il n'y a pas vraiment d'application à installer pour utiliser OpenRefine. Cependant, assurez-vous que **Java** est installé et à jour.

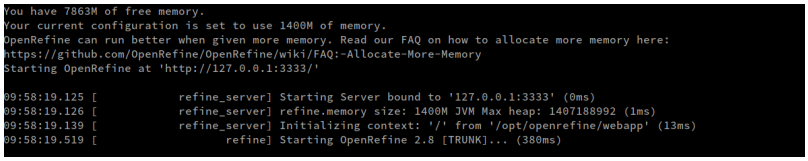
Vous devriez avoir téléchargé OpenRefine et extrait les fichiers à un emplacement sur votre ordinateur. Sous **Windows**, vous pouvez vous rendre à l'endroit où les fichiers sont extraits et double-cliquer "openrefine.exe". Sous **Mac et Linux**, utilisez un terminal pour vous rendre à l'endroit où sont placés les fichiers et exécutez

```
./refine
```

Sous **Linux et hypothétiquement sous Mac avec Homebrew**, il se pourrait que vous puissiez installer OpenRefine et ensuite l'exécuter dans le terminal avec la commande

```
openrefine
```

Lorsqu'OpenRefine est lancé, vous devriez avoir un terminal (peu importe votre système d'exploitation).



Une page web devrait s'ouvrir. Si ce n'est pas le cas, ouvrez un navigateur et allez à l'addresse http://127.0.0.1:3333/ (http://127.0.0.1:3333/).

Utiliser OpenRefine

Récupération des données

Nous allons maintenant récupérer des données provenant **de la base de données ouvertes de la ville de Montréal**

http://donnees.ville.montreal.qc.ca/dataset/declarations-exterminations-punaises-de-lit (http://donnees.ville.montreal.qc.ca/dataset/declarations-exterminations-punaises-de-lit)

Ce sont des données de déclarations de punaises de lit sur l'île de Montréal. Notez l'information mentionnée sur concernant le jeu de données:

> Déclarations des gestionnaires de parasites. Les formulaires de déclaration ont été soumis depuis le 5 juillet 2011. Les données ont un faible degré de fiabilité car elles sont consignées manuellement par des tiers, soit les gestionnaires de parasites et ne font l'objet d'aucune validation de la part de la Ville de Montréal.

Ce sont donc des données parfaites pour s'exercer! Vous devriez avoir **extrait le fichier csv sur votre ordinateur.**

Test d'importation des données dans R

Voyons voir rapidement de quoi a l'air un jeu de données lorsqu'importé directement dans R. Essayons avec punaises_mtl.csv. str() nous permet de constater qu'il y a des problèmes majeurs et il y a fort à partier que nous aurions quelques jours à investir dans le nettoyage des données.

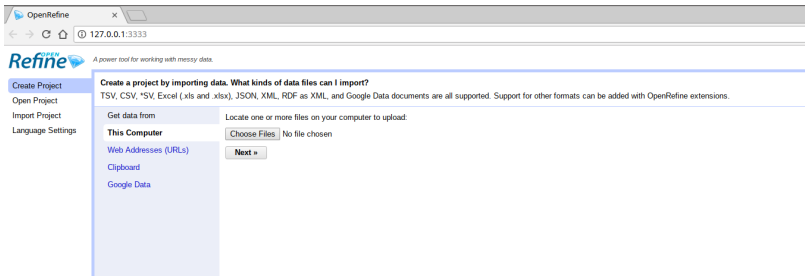
Si nous essayons de produire un graphique qui nous donnerait une indication du nombre d'exterminations qui sont effectuées en moyenne suite à une première visite d'inspection selon la date, on obtient un graphique peu satisfaisant.

Création d'un projet

OpenRefine peut importer plusieurs types de fichiers: **tsv (tab separated), csv (comma separated), xls, xlsx, JSON, XML, RDF as XML, Google Spreadsheets**, etc.

1. Dans l'onglet **"Create Project"** (on pourrait choisir un fichier provenant du web, mais utilisons le csv téléchargé)
2. Cliquez **"Choose Files"** pour chercher le fichier csv sur votre ordinateur
3. Cliquez **"Next"**.

4. Vous devriez voir un **aperçu du fichier**.
5. Vous pouvez **changer l'encodage (change encoding)**, peut être intéressant si vous travaillez dans windows(UTF-8).
6. Jouez dans le bas avec les types de séparation de fichiers pour voir les résultats.
7. Si tout vous semble correct, donnez un nom **à la copie du fichier** et cliquez **"Create Project"**.
8. Jetez un coup d'oeil aux noms des colonnes.
9. Dans OpenRefine, les modifications de font le plus souvent sur les **colonnes**. On peut aussi voir qu'il y a des **lignes** et chaque information est contenue dans une **cellule**.



Filtres de texte

Chaque colonne a un menu. Vous pouvez commencer par faire un **"Text filter"** sur la colonne NOM_ARROND. On peut chercher plein de mots.

Application de regroupements (*facetting*)

Cette étape est la plus importante d'OpenRefine. Elle vous permettra d'explorer et de vous familiariser avec les données. C'est particulièrement utile lorsqu'on a d'assez gros jeux de données. Le facetting nous permet d'avoir une **vue d'ensemble de jeux de données complexes** et de commencer à les explorer plus en détail.

Les filtres (***facets***) vous permettent d'agglomérer certaines données et d'effectuer des modifications sur ces groupes de données.

Text Facets

Prenons la **première colonne avec le nom des arrondissements**. Si on clique la flèche et **Facets** puis **Text Facets**, une boîte s'ouvre avec les différentes écritures des noms d'arrondissements.

> Comme les entrées ont été faites par de multiples personnes et avec des accents, c'est un peu le bordel.

Si on ***maintient le mot dans la boîte de filtre text***, on peut avoir un *facetting* juste pour ce mot. Si on enlève la recherche texte, on a toute la colonne arrondissements. On peut avoir le compte pour chaque arrondissement sur le côté et ça nous donne un aperçu rapide.

> Si on changeait un nom directement dans une cellule, on verrait le changement dans *facetting*.

Édition

On peut **cliquer directement dans une cellule pour changer manuellement du texte**. Changeons un seul mot et regardons ce qui arrive dans notre *facet*. On peut aussi cliquer "edit" sur une *facet* et changer toutes les cellules qui font partie de la *facet*.

On peut aussi le faire par ***ligne de code***, surtout lorque ça implique plusieurs modifications similaires sur un grand nombre de cellules. Par exemple, il est courant de travailler avec des jeux de données sans accent. Ça permet de partager les données avec des gens pour lesquels la locale ne permettrait pas l'interprétation de certains caractères spéciaux.

Exercice

Par exemple, avec GREL (le langage d'OpenRefine), l'expression **value.replace("é", "e")** permet de remplacer tous les "é" par des "e" dans toutes les cellules de la colonne sélectionnée. On peu enregistrer la modification et voir ce que ça donne.

> On pourrait aussi écrire replace(value, "é", "e") et c'est certainement plus intuitif pour quelqu'un qui utilise Excel ou R, mais ce ne serait pas la méthode standard de l'écrire dans GREL, qui parle plus à des utilisateurs de Python par exemple.

On peut ajouter des séries de modifications en ajoutant un point et une autre modification, par exemple:

```
value.replace("é", "e").replace("Ville-Marie", "Ville-Mario").replace("mot_orig", "nouveau_mot").modification(arguments)
```

> Un exemple que j'apprécie personnellement

```
value.toLowercase.replace("à","a").replace("â","a").replace("á", "a").replace("ä","a").replace("è","e").replace("ê","e").replace("ë","e").replace("é","e").re
-","-").replace("  ", "")
```

Vous pouvez sélectionner ***"retransform up to N times"*** pour créer des boucles qui vont tenter de retransformer un nombre X de fois. Ex: utile si plusieurs "- - -", "- -", "-"...

Réutiliser des commandes passées

On peut reprendre des commandes déjà exécutées dans l'onglet *historique* de l'interface de GREL. C'est très pratique pour réutiliser des commandes dont on prévoit se servir fréquemment.

Common transformations

Trim leading and trailing white spaces

Quand on travaille avec du texte, il y a souvent des cellules avec des espaces blancs au début et à la fin. Toujours, toujours le faire!

> C'est autre exemple de modification qui peut être faite avec du code et qui coûte rien:

```
value.trim()
```

Annuler/Répéter

Cliquer sur le **Undo/Redo** et **cliquer sur l'étape** ou on a commis une erreur. On peut remonter dans le temps jusqu'au point qu'on veut enlever. On peut aussi avancer dans le temps si on est finalement d'accord avec notre choix. On ne peut pas enlever seulement une étape dans le milieu parce qu'il se pourrait que ça ait des répercussions sur les étapes suivantes.

Autres Facets

On peut utiliser d'autres *facets* comme :

1. **Numérique** (montre un graphique, inclue une option "drag-and-drop")
2. **Timeline** (pour des dates)
3. **Scatterplot** (montre un graphique)
4. Définies par l'utilisateur. Exemples par défaut: 4.1. "Words" éclate un texte et compte les occurences de mots 4.2. "Duplicates" 4.3. "Text length" compte le nombre de caractères dans une cellule. Peut être utile pour dénicher des commentaires entrés dans une cellule Oui/Non 4.4. "Blanks"

Regroupements (*clustering*)

Il est possible dans Openrefine d'identifier les erreurs de frappe et les mots écrits au son. Des algorithmes automatiques sont à votre disposition dans le menu **flèche/cluster and edit**

Différentes options s'offrent à vous, qui sont décrites ici
https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth
(https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth)

1. key collision

   Groupe de méthodes les plus rapides car la complexité est linéaire. Méthodes qui créent une représentation alternative d'une valeur (ou une *clé*) qui contient seulement le coeur important d'une chaîne de caractères. Compare ensuite les clés entre elles (*collision*) pour trouver celles qui sont identiques.

   1.1. fingerprint

   Produit le moins de faux positifs. Comprend une série d'étapes qui sont décrites ici:
   https://github.com/OpenRefine/OpenRefine/blob/master/main/src/com/google/refine/clustering/binning/Fingerp
   (https://github.com/OpenRefine/OpenRefine/blob/master/main/src/com/google/refine/clustering/binning/Finger

   1.2. ngram-fingerprint (ngram size)

   Similaire à fingerprint. Choisir une taille de n-gram très élevée n'a pas beaucoup d'avantage, mais une valeur de 1 ou 2 peut trouver des combinaisons que fingerprint ne trouve pas, tout en produisant plus de faux positifs.

   1.3. metaphone3

   Pour l'anglais. Utilise la prononciation des mots.

1.4. cologne-phonetic

Pour l'allemand. Utilise la prononciation des mots.

1. nearest neighbor (kNN)

   Permet de définir une valeur de distance entre les paires de chaînes de caractères. Si deux chaînes de charactères tombent à l'intérieur de cette distance, elle seront agglomérées. Peut prendre énormément de temps!

   2.1. levenshtein (radius, block chars)

   Mesure le nombre d'opérations de modification nécessaires pour passer d'une chaîne de caractères à l'autre. Bon pour les typos, mais mettre une longue distance maximale peut ralentir le calcul considérablement.

   2.2. ppm (radius, block chars)

   Implémentation d'un code permettant de comparer des chaînes d'ADN. Méthode produisant généralement beaucoup de faux positifs.

Si vous identifiez un regroupement probable, vous pouvez accepter le remplacement par défaut ou cliquer sur une des façon d'écrire si le remplacement par défaut de correspond pas à celle que vous préférez. S'il y a plusieurs merges similaires possibles, on peut tous les faire d'un coup.

> Le visuel sur le côté peut être utile s'il y a beaucoup beaucoup de clusters et permet de naviguer selon les tailles de cluster.

Exercice

Dans la colonne NOM_QR, modifiez 3 cellules "Beaurivage" de cette façon: Borivage Baurivage Borrivage

Toujours dans la boîte des noms d'arrondissements, cliquez sur **"Cluster"**. Cluster utilise différents algorithmes de clustering de texte.

Jouez avec les différents modèles et les différentes options. Fingerprint = précis, Phonetica = pas précis, mais magique.

Vous pouvez ensuite cocher **"merge"** et cliquer **"Merge selected and recluster"** pour voir si le changement influence quelque chose, ou bien **"Merge and close"** si vous êtes satisfait des modifications.

Créer/Renommer/Diviser des colonnes

Lorsqu'on fait des modifications drastiques à une colonne, c'est possible de vouloir garder la colonne d'origine intacte pour pouvoir évaluer si on a fait les bonnes modifications.

`edit column/add column based on this column.`

> Il faut changer de nom. On peut même modifier les cellules directement.

Exercice

Sur la colonne _NOM*QR* **edit column/add column based on this column** Nommer la nouvelle colonne _NOM*QR2* Sur la colonne _NOM*QR2* **split column into several colums**, separator -, split into 2 columns

Si on veut remettre les colonnes ensemble on peut utiliser l'option **join columns**

Ou bien le faire à la main **facet by blank sur la colonne 2** et sur la **colonne 3**. On va voir qu'il y a plein de blanks dans la colonne 3. On va **sélectionner "False"** dans le facet by blank. On va seulement avoir ceux qui n'ont pas de blank. On peut maintenant faire *add column based on this column* sur la colonne _NOM*QR2 1* et taper **value + "-"+ cells["NOM_QR2 2"].value**

Si après coup vous vouliez renommer la colonne... **Changer les noms** des colonnes: **flèche/edit column/rename column**

Changer l'ordre des colonnes, enlever les colonnes

`Flèche à côté de ALL, edit columns/re-order remove columns`

> On peut réagencer ou enlever des colonnes maintenant inutiles.

Organiser par plusieurs colonnes

Sort: Essayer avec les coordonnées pour montrer que des fois les données sont trop nombreuses pour faire un facet et on veut juste les blanks

---

Transpose

Un peu le même principe que dans Excel, vous pouvez transposer les colonnes et les lignes. Nous ne passerons pas plus de temps là-dessus.

Enlever les cellules vides

Un problème récurrent dans les jeux de données sales est la présence de cellules vides. Sont-elles des erreurs de saisie? Des données manquantes? Une absence de résultat? À vous de décider. Pour ce qui est d'OpenRefine, il prend généralement mal en charge les cellules vides et les NA. Idéalement, il faudrait avoir fait ce traitement avant la manipulation des données dans OpenRefine. Cependant, quelques options s'offrent à vous si ce n'est pas le cas.

> Ex: La Côte Saint-Luc, c'est la Côte Saint-Luc. On pourrait si on veut remplacer les cases vides par Côte Saint-Luc pour fins d'analyse si on le désire. Également, il existe des lignes sans début/fin de traitement. À nous de choisir ce que ça veut dire. Finalement il n'y avait pas de contamination? Ils n'ont jamais été traités? L'inspecteur était trop lâche? Allez savoir...

Solution 1: delete

`Cliquer facet by blank`

`Setter à True`

`All/edit rows/remove all matching rows`

> On a maintenant un plus petit dataset.

Montrer les flags+stars

Solution: remplacer par NA

Les cellules vides qui sont en fait des **NA sont assez problématiques dans OpenRefine**. Idéalement, vous voudriez avoir fait cette étape avant d'utiliser OpenRefine.

`text facet par colonne`

`editer le facet blank pour mettre des NA`

Inclure/exclure des entrées

On peut décider de seulement travailler sur les données de certains arrondissements.

`cliquer sur Ville-Marie pour le faire apparaître en orange`

`cliquer sur include pour Saint-Léonard et Villeray`

> On a maintenant un subset des données qu'on pourra exporter quand on aura finit.

Nombres

À la base, OpenRefine lit toutes les colonnes comme du **texte**. Si on veut faire des manipulations sur des nombres, il faut les convertir. Plus facile à dire qu'à faire quand on a du texte et des nombres mélangés.

`edit cell/transform`

peut aider à régler certains problèmes. Ddns la colonne _DATE*DECLARATION*, on a un mélange. On peut faire **edit cell / transform** pour faire un maximum de changements automatiques, mais il se pourrait que vous ayez à faire des modifications manuelles si votre jeu de données est trop complexe/sale.

Facets numériques

Dans ce jeu de dponnées, le nombre d'exterminations est le seul réel élément numéérique, mais nous pouvons aussi voir comment ça pourrait fonctionner avec le numéro de déclarations ou le numéro du quartier.

Facets dates

Il est possible d'organiser les données selon la date plutôt que numériquement. Tout comme les facets numériques, il faut avoir préalablement changé le format de la colonne pour date pour pouvoir faire le *facetting*.

Attention: Il est possible que la conversion au format date fonctionne de façon aléatoire pour vous. Il semblerait qu'il s'agisse d'un problème avec Java et le fuseau horaire de votre ordinateur. Toujours vérifier que la conversion a produit le résultat escompté!

Exercice

Remplacer un 1 par un I dans la colonne _DATE*FINTRAIT* Transformer la colonne en date. Identifier la cellule demeurée en texte dans la colonne. Modifier la cellule et sélectionner le format *date*

(vous pouvez également utiliser value.toDate("format") pour convertir manuellement si la méthode par défaut ne donne pas les résultats escomptés. value.toDate("y") extraira l'année.)

On peut diviser les colonnes pour extraire le début de la date de déclaration et enlever l'heure.

`edit column/split into several columns`

> normalement l'idéal c'est d'avoir un diviseur. Ici on peut prendre le "T" et 2 colonnes maximum.

`facetter et cliquer seulement sur date`

`autre facet sur la colonne avec les noms`

`clusters pour corriger les erreurs`

Si on veut remettre les colonnes ensemble

`facet by blank sur la colonne 2 et sur la colonne 3`

> On va voir qu'il y a plein de blanks dans la colonne 3.

`sélectionner "False" dans le facet by blank.`

> On va seulement avoir ceux qui n'ont pas de blank.

`value + " " + cells["col2 3].value dans la colonne 1`

> Split [0] élément 1, [1] élément 2, [-1] dernier élément

`Changer les noms des colonnes: flèche/edit column/rename column`

Bonification des données

fetch URLs

On peut bonifier le jeu de données en ajoutant des colonnes contenant des informations trouvées automatiquement en ligne. Par exemple, nous pourrions vouloir utiliser le nom de l'arrondissement pour trouver une géolocalisation ou une géolocalisation pour trouver le type de bâtiment ou le nom de la rue. Une des options les plus connues pour accéder à des informations de géolocalisation est l'API de Google maps

https://cloud.google.com/maps-platform/ (https://cloud.google.com/maps-platform/)

Cependant, l'API est maintenant gratuite de façon limitée (avec un nombre de crédits offerts à l'inscription), mais il peut en devenir coûteux d'interroger leur API. Une option gratuite est OpenStreetMap

https://wiki.openstreetmap.org/wiki/API (https://wiki.openstreetmap.org/wiki/API)

Pour l'exercice, on peut vouloir utiliser les données de géolocalisation des inspections pour déterminer l'élévation de l'endroit où l'inspection a eu lieu. Pour ce faire, nous utiliseront une API qui est uniquement dédiée à l'élévation selon la géolocalisation:

https://elevation-api.io/ (https://elevation-api.io/)

cette api nous permet de faire un nombre illimité de requêtes à 1km de précision et ensuite il y a une facturation pour des précisions plus élevées.

Que nous allons interroger à l'aide des données de latitude et longitude dont nous disposons déjà.

> il serait peut-être judicieux de faire d'abord un *facet* numérique sur les coordonnées pour s'assurer qu'il n'y a pas eu d'erreur de frappe et des "points dans l'océan".

```
sur "latitude", add column based on column

nommer "query_api"

remplacer "value" par "https://elevation-api.io/api/elevation?points=(" + value + "," + cells.LONGITUDE.value + ")"
```

> ceci nous donne un URL qui nous permettra d'aller chercher les informations que l'API nous permet d'obtenir sur les coordonnées que nous avons sélectionnées, c'est-à-dire l'élévation dans ce cas.

Lorsque c'est fait, nous avons une colonne nommée query_api qui sera la base pour interroger l'API. Pour ce faire,

```
Sur la colonne "query_api: edit column/ add column by fetching URL

nommer query_result

throttle: 500 milliseconds
```

> pour faire une pause. Sans contrôle de votre part, Openrefine va se connecter frénétiquement sur l'API pour obtenir des données d'élévation. Il est donc important de limiter le nombre de requêtes par unité de temps.

On laisse ensuite la magie se faire et quelques secondes/minutes/heures plus tard on devrait avoit une nouvelle colonne contenant du texte (presque) indéchiffrable:

```
{"elevations":
[{"lat":45.5563940269788,"lon":-73.6459326267949,"elevation":34.0}],"resolution":"1000m"}
```

Comme nous ne voulons garder que la valeur pour "elevation", soit dans ce cas 34.0, nous devons indiquer à OpenRefine ce qu'il doit faire avec la colonne _query*result*.

`Sur la colonne "query_result": edit column/add column based on this column`

`value.parseJson().elevations[0]['elevation']`

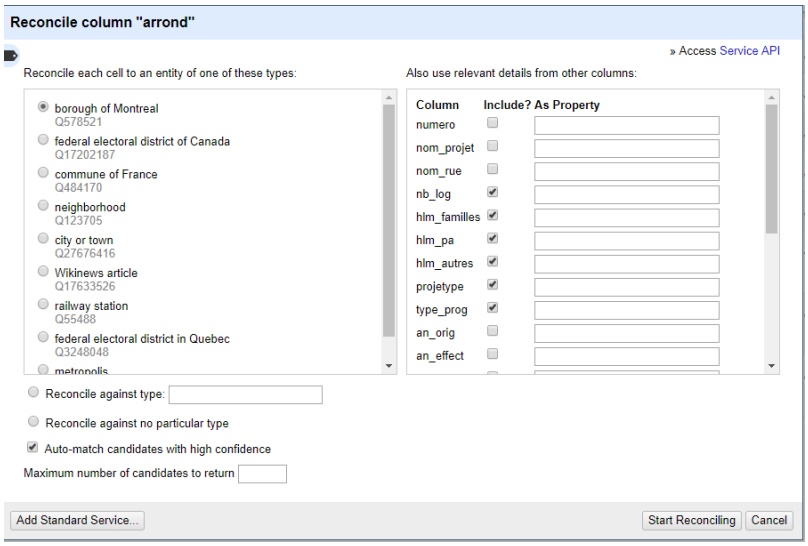
Reconcile

La réconciliation est utile lorsqu'on sait que certaines bases de donnéées existantes permettent de bonifier les données. Il suffit de choisir une colonne qui contiendrait des identifiants uniques qui pourraient être croisés avec les bases de données et choisir: `reconcile/start reconciling`

`choisir Wikidata reconciliation for openrefine`

> si vous ne trouvez pas les bases de données que vous désirez, vous pouvez les ajouter à wikidata

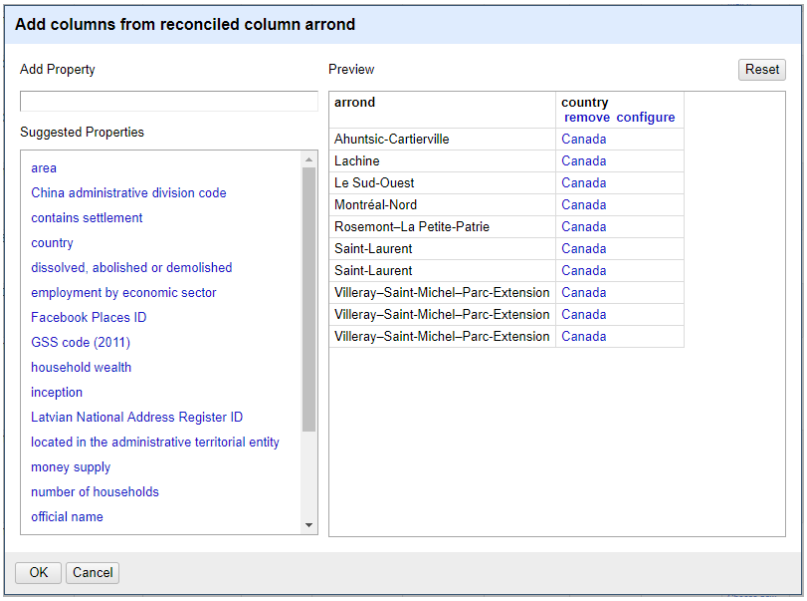
`choisir le type nécessaire, ici Q578521`



Le processus est semi automatisé parce que dans certains cas OpenRefine ne réussira pas à réconcilier les cellules avec la base de données et il faudra le faire à la main. Normalement, c'est assez rapide parce que wikidata permet environ 3 requêtes par seconde. Une fois les données réconciliées, on peut aller chercher les données supplémentaires disponibles via la base de données.

`edit column/add columns from reconciled values`

et faire des tests. Dans notre cas, on a seulement que "Area" et "country" à aller chercher. Mais on pourrait choisir une base de données différente pour avoir des résultats différents. Il faut aussi considérer qu'on a pas particulièrement un jeu de données adapté à la réconciliation.



exporter

L'icône **Export** mène à export project. Il y a plein de formats, mais un format intéressant est **"templating"**

as Template

Ça crée un **template en format JSON** qui peut être utilisé pour l'exportation dans des formats qui ne sont pas encore supportés par OpenRefine. Le prefix et suffix sont utiles lorsqu'on veut conserver le template JSON. Sinon, on peut se limiter à mettre du texte. La partie *jsonize* est aussi inutile à moins de vouloir utiliser JSON. Ce template peut être utilisé pour partager les données dans un contexte qui ne requiert pas le jeu de données complet, mais plutôt un format agréable pour la lecture.

OpenRefine offre un exemple pour exmporter en YAML: https://github.com/OpenRefine/OpenRefine/wiki/Export-As-YAML (https://github.com/OpenRefine/OpenRefine/wiki/Export-As-YAML)

Réutiliser les routines dans le futur

Dans l'onglet **undo/redo**, on peut cliquer extract et ça va nous donner un fichier JSON qui pourra servir à la documentation ou à la réutilisation. On peut sélectionner les parties qu'on veut garder de notre routine.

copier coller dans un fichier texte (pas de Word!)

et on peut ouvrir un nouveau projet, un nouveau fichier, retourner à undo/redo, cliquer apply, copier-coller le