OpenSchoolMaps: Data cleansing and integration Openrefine

OpenSchoolMaps.ch – Free learning materials on free geodata and maps and open source.

This is a worksheet with exercises for self-learners and students.

Overview

Goals and Objectives

This worksheet shows how you can use OpenRefine, which is a desktop tool for data pre-processing without programming. A lot of ways to deal with heterogeneous data are discussed. With data pre-processing we mean exploring, cleaning, integrating and enriching data.

The exercise part consists of some tasks that involve real-life examples of dealing with heterogeneous data and will try to show you what kind of approach we (as users) should be using when faced with potential messy data.

After completing this worksheet, you will be able to:

- Understand **OpenRefine**.
- Use OpenRefine to **explore** data.
- Understand the challenges that heterogeneous data can pose and use OpenRefine to **clean** data.
- Use OpenRefine to **integrate** and merge two datasets.
- Use OpenRefine to **enrich** data using a geocoding service.
- Use OpenRefine to **access structured data** using web scraping.

Time Required

The time required to complete this worksheet is about one hour for the reading part (without exercises), plus about five quarters of an hour for the exercises part—both depending on your previous knowledge and skills.

Prerequisites

In order to complete this worksheet, you need the following prerequisites:

- Hardware: At least 1 GB of free RAM available.
- Internet access (for software and data download and for the advanced chapters and exercises about services).
- Software: OpenRefine (available for Windows, Mac, Linux), installed as instructed below.
- Data: As denoted below.
- Basic understanding of how to work with data
- Basic knowledge of data types
- Basics in SQL

Installation of OpenRefine

In order to do the exercises, you need to have OpenRefine installed locally on your computer. If you are working on a Linux system you are required to install the Java JRE. Windows user can use the Windows kit with embedded Java. There is also a Mac kit which you can use right out of the box. Tested browsers are Firefox, Chrome and Safari; the browsers Internet Explorer and Edge are not supported. The installation instructions are located in OpenRefine's installation section of the manual. Follow the instructions there.

If you're having problems installing OpenRefine or an issue in general with OpenRefine, read first the FAQ in the manual, then follow the tips in the manual. See also this help page.

Introduction

Structure of this Worksheet

- In the next chapter, the basics of Data Integration and Data Enrichment will be explained.
- In chapters 3 and 4 we will explain the basics and functionality of OpenRefine.
- In chapter 5 we will be explaining what geocoding is and how it is done using OpenRefine.
- In chapter 6 we will scrape a Wikipedia web page using OpenRefine.
- The goal of the exercises in chapter 4, 5 and 6 is to strengthen the OpenRefine usage skills by going through a whole workflow using OpenRefine.

Data Integration and Data Enrichment

Data can be structured or unstructured. With **structured** data we mean data modeled in attributes having data types. An address list (encoded in a CSV file) typically represents structured data. **Unstructured** data is data stored in a plain text, like a novel. With this kind of data it's impossible to produce for example a serial letter. But even if the data is structured, it can contain heterogeneous and potential messy data. With **potential messy data**, we mean data with many missing data values and data with has `wrong` format or `wrong` structure - at least for our objective.

Data integration is the process of combining data from different sources into a single, unified view. The integration process begins with an import, and includes steps such as cleaning, schema mapping, and transformation. OpenRefine offers many features that are convenient for data integration (Chapters 3 and 4).

A typical integration process is when two address lists from different, independent sources are to be merged. We will practice this process in a task (more on this in Chapter 4).

Data enrichment is a term that applies to the process of enhancing and completing existing data. This can be done in different ways, like merging third-party data from an external source. Let's say you have some customer or friends data containing names and postal addresses. An example of this would be adding coordinates based on postal addresses. This process is called **geocoding** and allows addresses to be displayed on a map (more on this in Chapter 5).

Search engines use crawlers (or bots) to `scrape` the web. **Web scraping** is the process of extracting data/content from a website for analysis or other use. It involves reading the HTML file of a website and filtering it for data. An example of this occurs later as a task (see Chapter 6 Web scraping).

OpenRefine Basics

OpenRefine is an application tool that runs locally as a stand-alone application on your computer and uses a web browser as graphical user interface (GUI). It's written in Java and available as open-source and has it's roots in managing a knowledge base, like Wikidata. By the way, there are also open source alternatives like Workbench, which follows a similar concept but is written in Python.

OpenRefine reads and manages large amounts of tabular data, which is usually cluttered and unstructured. It doesn't require Internet access for most of its functions, since it runs locally. This means that your work environment with OpenRefine is private, since everything is run locally.

Referring to the generic tasks mentioned in chapter 2, these are the things OpenRefine can do:

- Explore the data, e.g. faceting and clustering.
- Clean potential messy data, e.g. unstructured (or semi-structured) text files.
- Data transformation - Bulk transformation of data, e.g. data normalization, data formatting.
- Data validation and deduplication.
- Data reconciliation with external services, such as Wikidata.
- Accessing web site data.

OpenRefine is a desktop application implemented as a client-server, with a browser as client. The principle of OpenRefine is that data is read from a source (file, service, database) and stored locally as a copy - i.e. as its own OpenRefine project.

The Graphical User Interface (GUI)

OpenRefine offers an interactive Graphical User Interface (GUI) that visualizes each step of working with your data set. It consists of a main grid (or spreadsheet) with the tabular data you are currently

working on.

This spreadsheet changes every time you perform an action such as filtering or creating different facets. The left part of the GUI has a tab for showing each facet/filter that is currently affecting your data set. This section offers the possibility of quickly changing facets and filters. Every time you perform such an action, the spreadsheet reflects the changes.

This spreadsheet changes every time you perform an action such as filtering or creating different facets. The left part of the user interface contains a tab that displays each facet/filter that is currently affecting your dataset. This section provides the possibility to quickly change facets and filters. Each time you perform such an action, the changes are displayed in the spreadsheet.

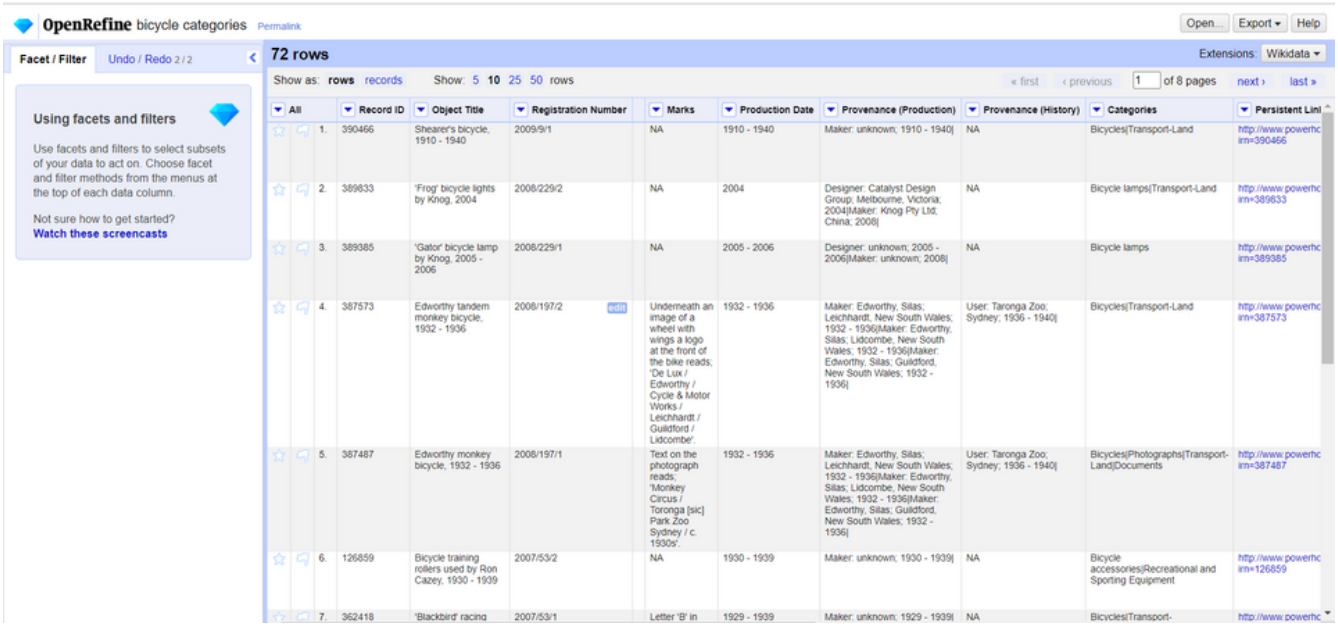


Figure 1. The OpenRefine GUI.

These are some important terms used in OpenRefine: Since OpenRefine uses a tabular data model, there are **columns** (similar terms: field, attributes, cells) and **rows** (similar terms: records, lines). Then there's **reconciliation** (similar terms: integration, conflation), the process of matching your dataset with that of an external source.

OpenRefine also has a powerful **Undo / Redo** - or history - feature which is a strong safety measure that allows you to work freely and experiment with your data as much as you want. Each action that you perform on the data set is tracked by OpenRefine, allowing you to comfortably make any transformations you want to do on your data set, with the possibility of quickly reverting those changes. However there are a few operations such as transformations on a single cell level that can't be exported as JSON.

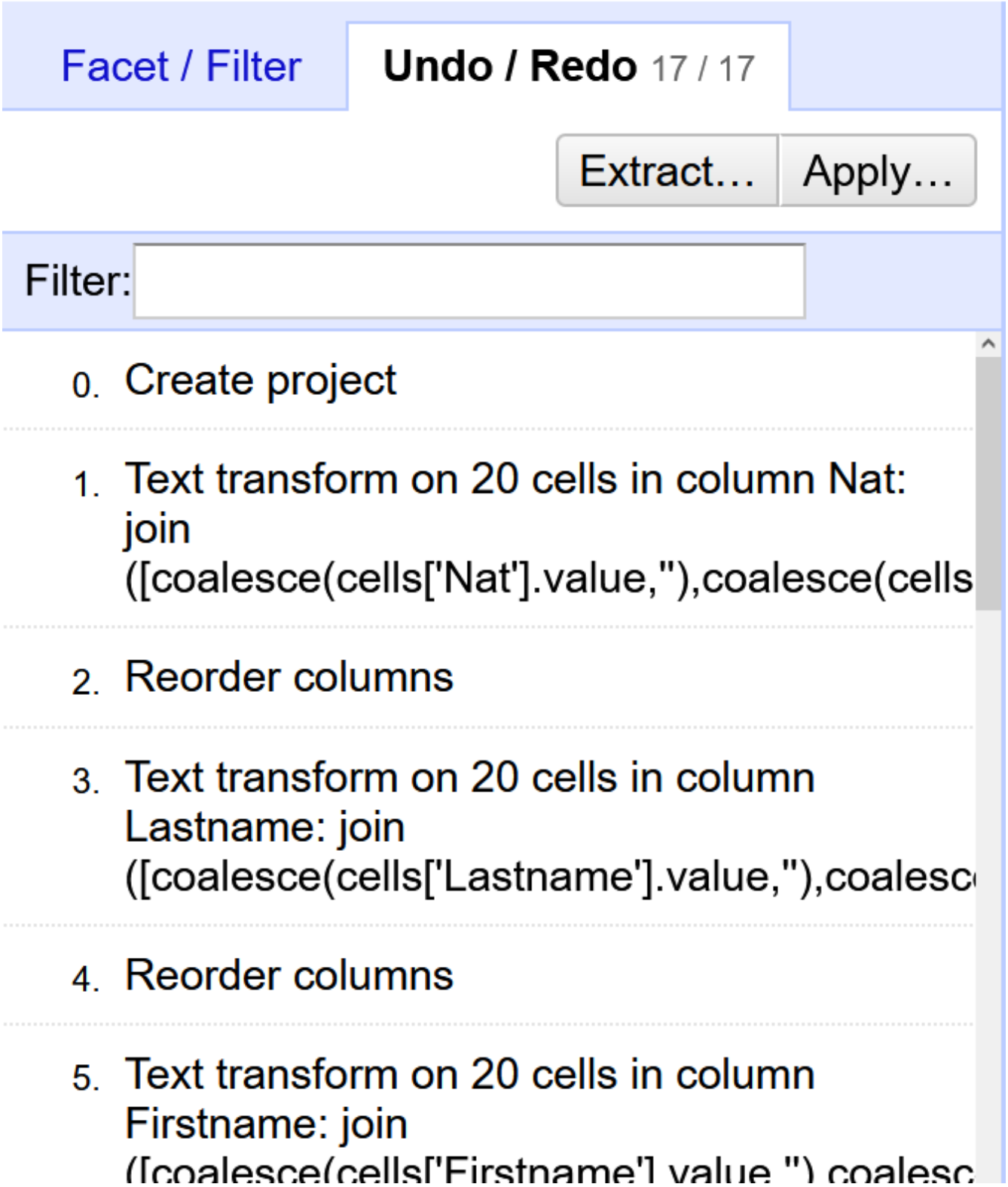
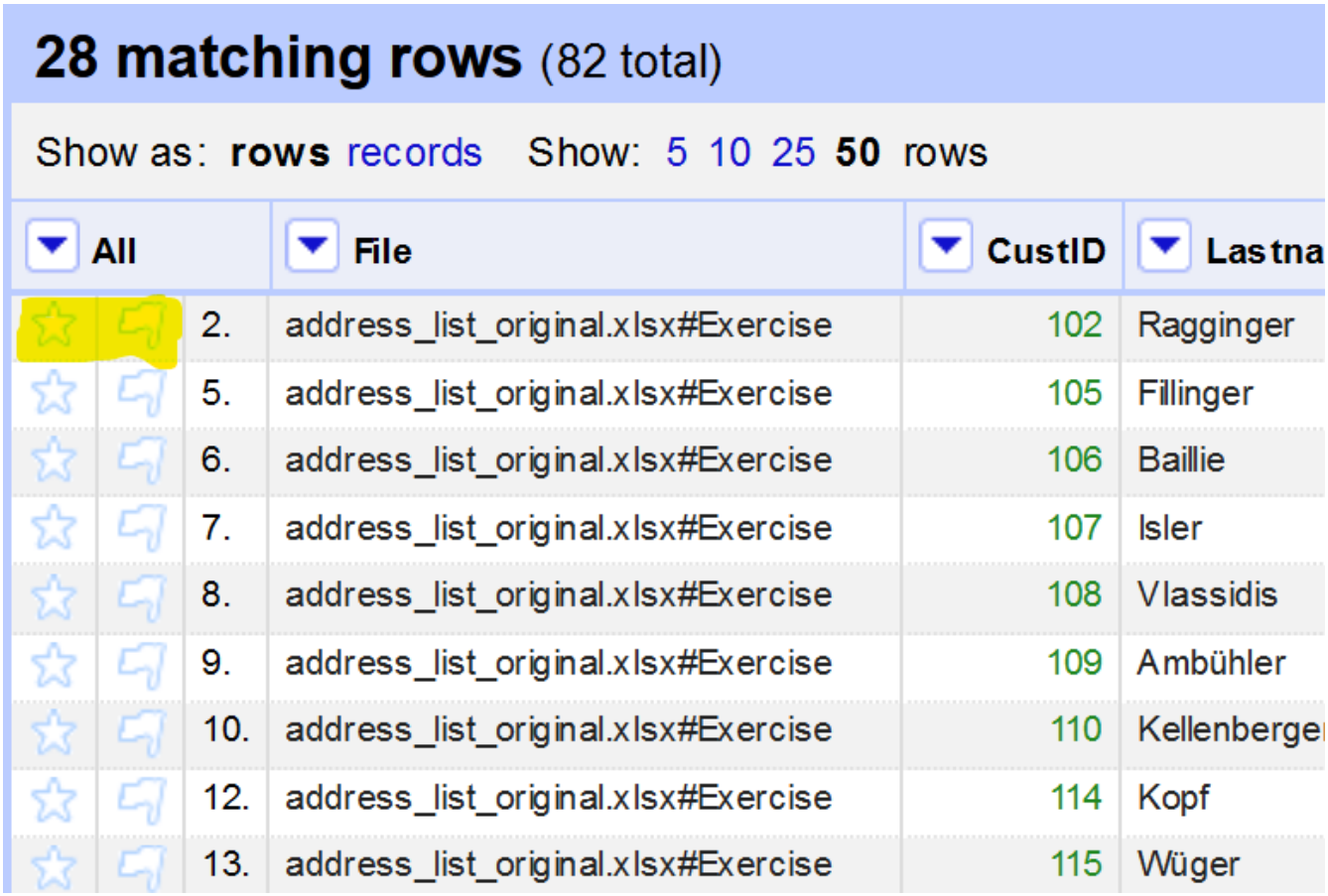


Figure 2. Undo/Redo feature of OpenRefine.

You can also extract all the project steps made or even apply a change set using a simple JSON file.

On the left of each row on your dataset is a flag and a star icon. Marking them persists even if you close/open your project and the idea of using these is marking your rows as starred or flagged for focusing later on them.

You can add stars/flags to your rows individually or you can apply them to all of the matching rows by using the special column *All*. Faceting (something you will learn later on in the worksheet) by stars/flags isalso possible.



28 matching rows (82 total)

Show as: **rows** records Show: 5 10 25 **50** rows

All		File	CustID	Lastna
2.		address_list_original.xlsx#Exercise	102	Ragginger
5.		address_list_original.xlsx#Exercise	105	Fillinger
6.		address_list_original.xlsx#Exercise	106	Baillie
7.		address_list_original.xlsx#Exercise	107	Isler
8.		address_list_original.xlsx#Exercise	108	Vlassidis
9.		address_list_original.xlsx#Exercise	109	Ambühler
10.		address_list_original.xlsx#Exercise	110	Kellenberge
12.		address_list_original.xlsx#Exercise	114	Kopf
13.		address_list_original.xlsx#Exercise	115	Wüger

Figure 3. The flag and star icon located on the left of each row of your dataset.

Projects

An OpenRefine **Project** is a `workspace` for your dataset within OpenRefine. It is created by importing some existing data into OpenRefine. First, import your data (in various ways), then configure the parsing options and proceed to create the project. Once you create the project, the data will be saved in a different file than your original source (in the said workspace of OpenRefine) and you won't have to worry about accidentally changing your original data.

OpenRefine stores a project into a separate directory after you create the project and doesn't ever modify your original file. For more information on how OpenRefine handles projects and on their storage, check out this documentation.

Exploring data within OpenRefine helps you learn more about your dataset. Typical questions while exploring are:

- Which attributes are text, discrete and which are continuous?
- What kind of values does each attribute have?
- How are the values distributed?

In OpenRefine there are different ways to display the data. Basically one distinguishes between **Rows** and **Records**. More information about this can be found on the OpenRefine documentation.

OpenRefine assigns a data type to each cell in the dataset, where some of the cells have a dedicated data type found by OpenRefine and others do not (they are initially set to String). Each data type has specific functions you can perform with it, but not necessarily with others, and many data types can be converted to other data types (if necessary).

Note that each cell is assigned its own data type. In this point OpenRefine is more similar to Excel than to a database, because in OpenRefine not all cells of a column necessarily have the same data type - although this is indispensable with regard to evaluations (and database tables). The solution for this is provided by the OpenRefine functions (so-called "facets"), which are described and practiced in the following.

OpenRefine Functionality

In this section, we will be first describe OpenRefine's reader/input and explore its functions. Then, transformation functions are presented. Finally, we will be describing how exportation of data is handled in OpenRefine.

OpenRefine's most valuable functions are explorations - facets which allow to quickly explore the values of a given column - and transformations which offer many options to manipulate the data. Understanding these features is key to understanding the whole lifecycle of an OpenRefine project and will also come in handy in the worksheet exercises.

Importing and Exploring Data

OpenRefine first imports a file from your computer or from the Internet and then creates a project from it. After data cleaning/manipulation, you can then export the data to a specific file format and use it.

Importing

Importing is the first step when working with OpenRefine. The following formats are supported for importing in OpenRefine: CSV, TSV, JSON, XML, Microsoft Excel spreadsheets (.xslx, .xls), HTML tables (.html), Google Spreadsheets (online).

OpenRefine also offers the possibility of importing data from an existing database source.

Facets

After you open an unknown data source, one of the first things you do is explore it. So you want to "see" the data content and also determine the data types of the columns. Data types are important because they allow for various operations. Some basic data types are text, number, date/time, boolean (true/false) and enumeration (e.g. red, yellow, green).

Data types are important. Programs like Superset/Tableau/PowerBI and even MS Excel have built-in functions to guess the type. In OpenRefine you will find the

> option **[Parse cell text into numbers, dates, ...]** when importing a CSV.

Facets are an important feature of OpenRefine that go beyond spreadsheets and traditional database tools. They show the data variance in a particular column. Faceting allows us to get a better picture of the entire data set and view the data from a broader perspective.

A facet groups values (e.g. of text or numeric data type) that are in a column and allows the user to filter and edit the values in different cells simultaneously, which works like a browsing mechanism.

Typically, facets are created for a particular column by clicking on the column, selecting the *Facet* option, and then clicking on one of the facet alternatives.

For example, if you select **[text facet]**, the entire contents of the column cells are compared to display general information about the values in that column. You can then either manually enter a new value for the facet entry or use the OpenRefine *clustering* option, which will automatically take care of the facet.

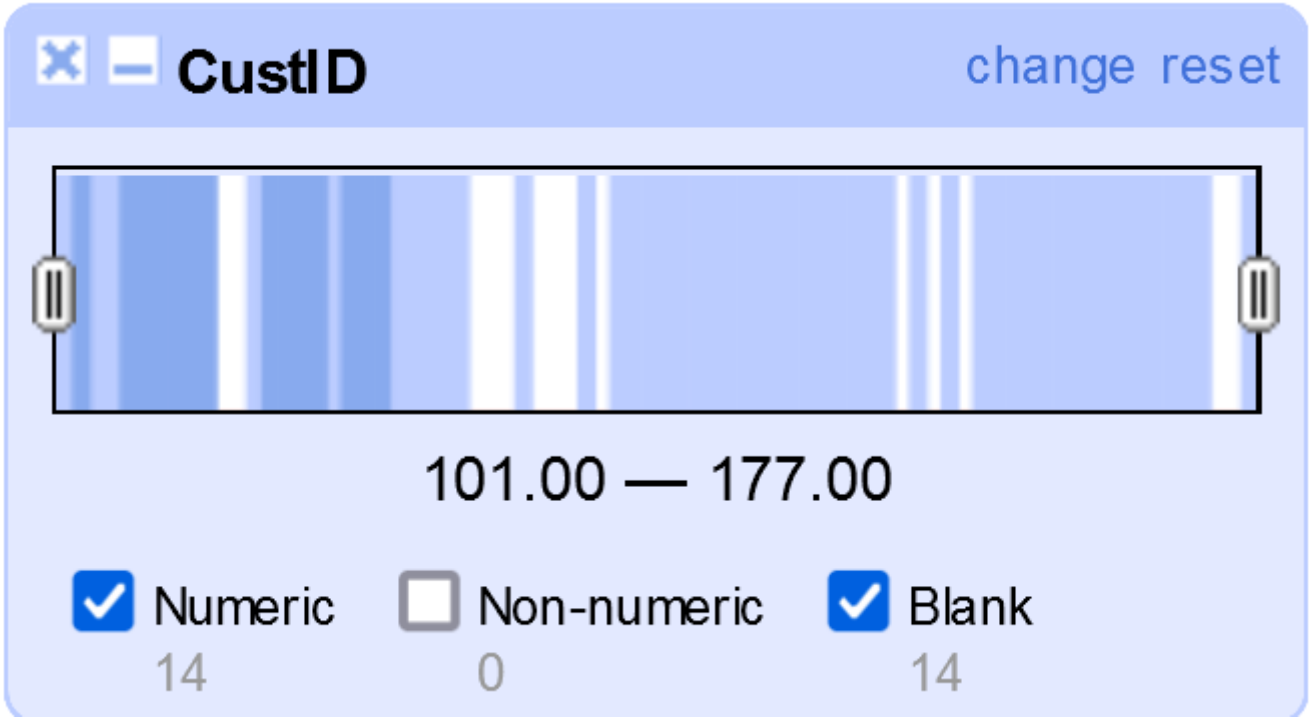


Figure 4. Example of a numeric facet on numeric type "customer_id".

Filtering

You can also filter specific column values in the data set by choosing the **[Text Filter]** option in the column dropdown menu. This creates a text box where you can insert the text you want to filter the column values.

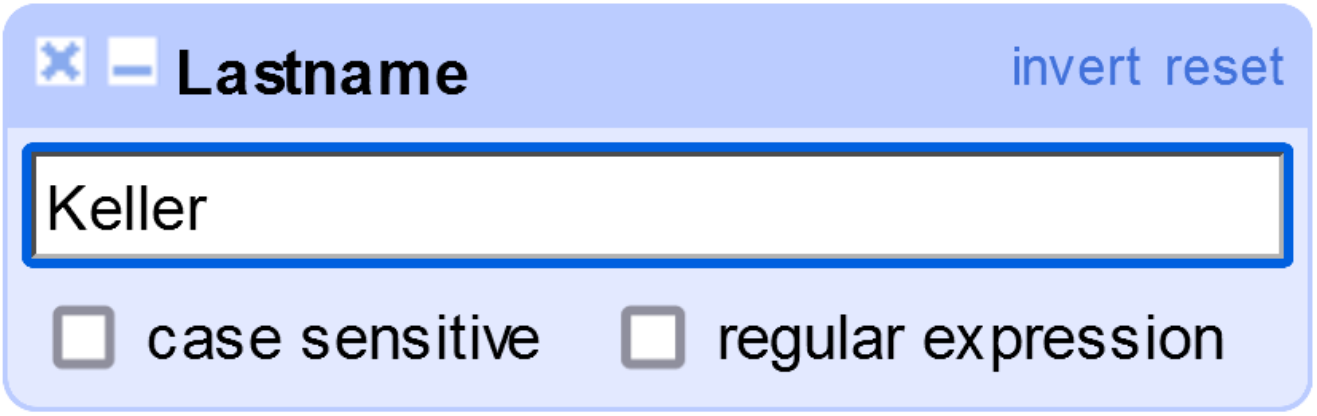


Figure 5. Example of a text filter on the `Lastname` column.

Clustering

Clustering is another important feature for grouping similar data which helps you identify data inconsistencies and misspellings. This is common in many data sets.

Clustering can be done by choosing a *Facet* on the column you want to cluster and then choosing the method you want to cluster your column values with.

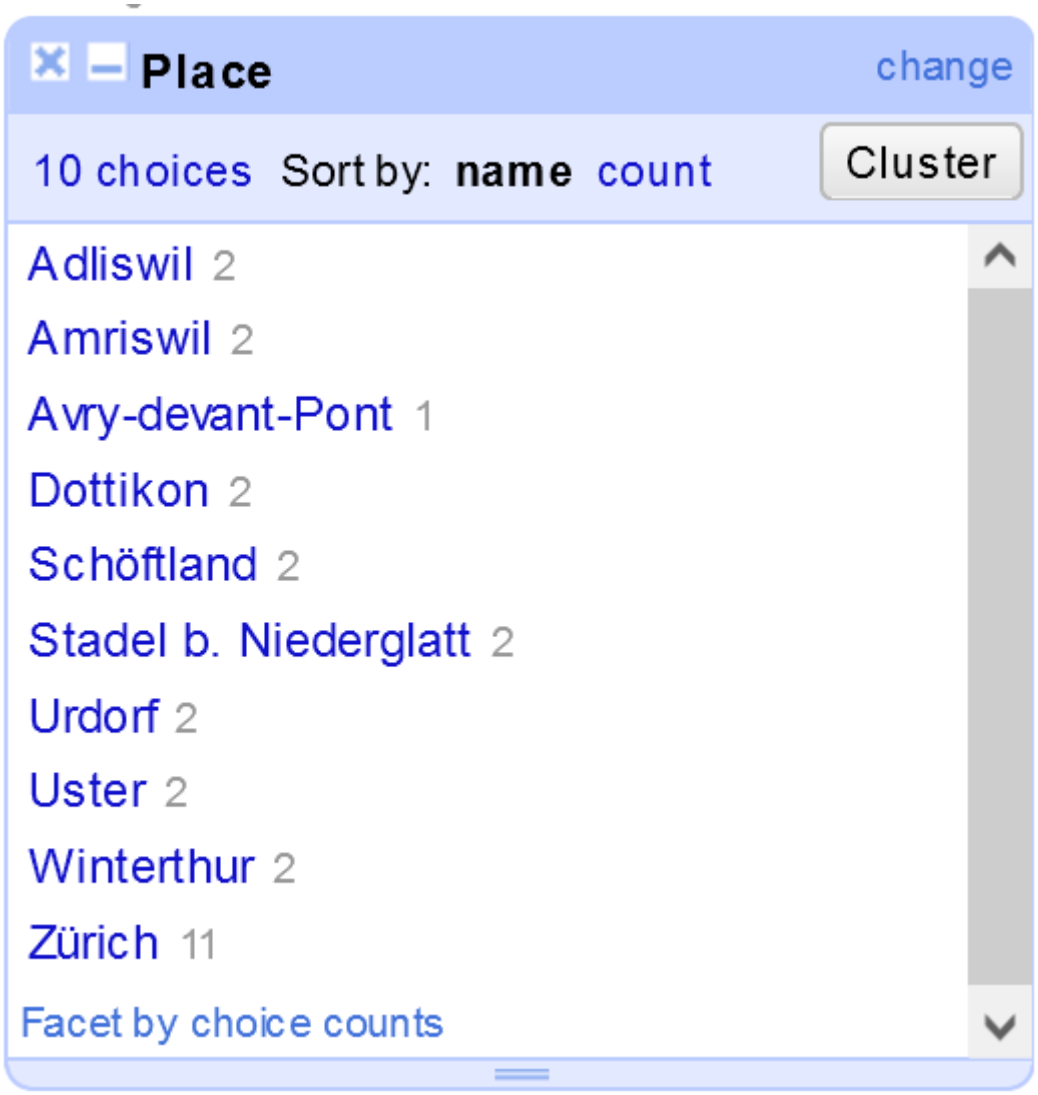


Figure 6. Finding the Cluster button on a Text Facet

Figure 7. Example of using the `nearest neighbor` clustering method.

As you can see in the above image, this is an example for the *Zurich* city where the city's name was purposefully misspelled a couple of times. The clustering function finds those misspellings and offers us options on how to treat these types of misspellings.

You can find out more about clustering in this OpenRefine documentation page

Transforming

OpenRefine offers some powerful features/functions for working with data. Some transformation functions include, among others:

- At value level:
  - String operations
- At field/column level:
  - Split and join multi-value cells
  - Calculations in fields
  - Adding constants
  - Joining (concatenating) fields

There are also other OpenRefine functions that can transform your whole data set (bulk edit) with only a few clicks, such as:

- Re-ordering columns:
  - Single re-order - Located on the dropdown of the **column › Edit Column › Move column to**

[direction]

- Multi re-order - Located on the dropdown of the **All › Edit Column › Re-order / remove columns...**

- Re-naming columns:
  - Located on the dropdown of the **column › Edit Column › Rename this column**
- Sorting data:
  - Located on the dropdown of the column **[Sort...]**, after that you have to choose the data type of the column values.

String Operations and Functions

OpenRefine provides some fast transformation operations that are useful when dealing with data. You can either use one of the preset transformations (ready to use) or use the GREL language to implement your own transformation function (requires light programming skills; see the separate chapter below).

Some of string operations include:

- Replacing quotes (common in potential messy data)
- Transforming text to other different data types
- Trimming whitespaces or other special characters
- Escaping/unescaping HTML characters

When using these operations correctly, one can quickly turn potential messy data into clean and machine-readable information.

Using *String Operations* on OpenRefine can be done by clicking the *Edit Cells* option on the column dropdown menu. When using string operations, you can choose one of the preset transformations offered by OpenRefine or write your own using the OpenRefine's GREL language, e.g. `value.toDate()` expression (see section below).

Splitting Fields

Another key feature of OpenRefine is the *Split* functionality that is useful for splitting column values into multiple columns. All you have to do is specify the character in-between the words and then OpenRefine will split those strings into multiple columns for you. This is very common when working with messy data, a lot of data within one column makes much more sense if split into multiple columns.

Splitting fields in OpenRefine is done by clicking on **[Edit Column]** and then **[Split into several columns...]**. This will let you specify a separator character(s) or a field length for separating your column values.

You can also **split multi-valued cells** (with extra options) using the **[Split multi-valued cells...]** option.

Joining Fields

Another option, which is sort of the opposite of splitting fields, is called *Join Fields* and is used when we want to join fields with a separator (or without). This is pretty self-explanatory.

The *joining of fields* in OpenRefine is done by clicking on **[edit column]** and then then **[Join columns...]**. You can then select the columns you want to join and set the separator between the contents of each column. You can write the results in the same column or create a new column.

You can also **join multi-valued cells** (just like splitting above) by using the **[Join multi-valued cells...]** option.

Joining datasets from different sources

A common scenario when working with data is to obtain data from multiple sources. Even if the incoming data ultimately serves a common purpose, the structure of the data in the different sources may not be identical. An example of this would be receiving two customer lists with the same attributes but different column names, for example. Although from a human point of view it looks as if they are the same data, this is not the case for computers. Different techniques have to be applied to integrate or rather "merge" these different sources of similar data.

There are usually three types of cases when joining datasets:

1. Extending the first dataset by another one which mostly has overlapping column names, a.k.a. **extend down vertically**
2. Enriching the first dataset by just one or few columns from another auxiliary dataset, a.k.a. **extend down horizontally by one or few columns**
3. Extend first dataset and it's rows by another complementing dataset, similar to the *SQL JOIN*, a.k.a. **extend horizontally**

In cases 1 and 2, a common column called "key" is required. Typical keys are identifiers such as postal codes, postal addresses (see section Geocoding further down the worksheet), country codes, municipality codes, ISBN, etc.

Some of the above cases will be applied in exercises 2 and 3. The first case will be applied in exercise 2 and the second case in exercise 3. For the third case, there is no exercise on this worksheet that demonstrates it. OpenRefine does not currently provide a direct solution for this case.

Validating the Dataset

OpenRefine also offers functions for validating your data against another data set. It can be done by creating a `Custom Text Facet` using the GREL `cell.cross()` function that matches values from two different columns. There is no native/straight-forward way for validating data sets but it can be achieved by using functions that behave in the same fashion as the GREL `cell.cross()` function.

Data Deduplication

For deduplicating the data, one of the ways you can achieve it is for example by using **Clustering** by

key collision to find similar data. Then you can manually assess duplicate data and choose what to do with them after the clustering process is finished. This is usually a pretty straight-forward and efficient way of deduplicating a data set.

Exporting

When you are done with working with the data and you have the data in the desired format, the last step is exporting that data into a specific file format.

The following formats are supported for exporting in OpenRefine: CSV, TSV, HTML table, Microsoft Excel spreadsheet (.xlsx, .xls), ODF spreadsheet (.ods), and some other custom exporters.

Exercises

Exercise 1: A first OpenRefine Workflow

In this exercise, we are going to demonstrate a typical workflow and lifecycle of an OpenRefine project, from its creation and until the data is ready to be exported.

Data

For this task, the data from the fill `address_list_original.xlsx` is used.

Step 1: Creating a Project

In order to create a project (the actual workspace where we work with the data), you need to:

1. Open OpenRefine on your local web browser.
2. Choose the file you want to work with (or retrieve it directly from the Web), in our case, choose the file you just downloaded from the above link.
3. Click on *Next*.
4. After you have chosen what file to import, you need to configure the parsing options. The `preferred` options will automatically be chosen by OpenRefine but you can customize them if you like (see figure below).
5. Name the project and click on *Create Project* to create it on OpenRefine and start working on it (also saved for future use).

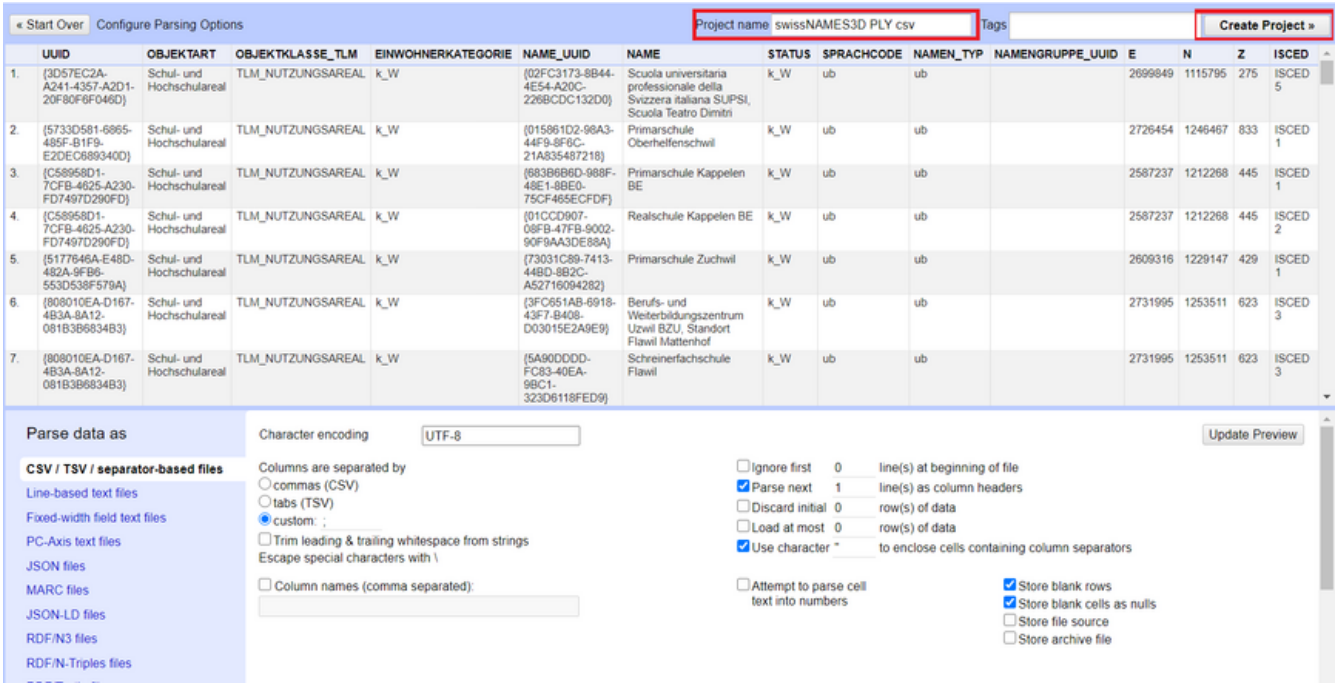


Figure 8. Project parsing options.

Step 2: Check data

We will refine the data set to filter out only customers from the Zurich canton and merge their addresses to a new column containing the full address.

1. Create a Text Facet on the `Place` column for filtering down to only customers living in the `ZH` Kanton
2. Merge the `Street`, `Place` and `ZipCd` columns into a new one, separated with the `,` character (except the `ZipCd` column)
3. Add the country entry code to their phone numbers. 3.1 Add `+41` as the country code (for Switzerland) 3.2 Remove the `0` prefix on their phone numbers 3.3 Trim the white spaces from the phone number (e.g. +41445308197)

One way of doing string manipulation (third step) is by applying the following GREL expression: `value.replace(value, "+41" + value.substring(1)).replace(" ", "")`. There are of course multiple ways, but it is encouraged for you to create your own.

Step 3: Exporting the data

After you have finished the above tasks, you can export your data set (project) by clicking on **Export › Excel 2007+** and OpenRefine will export your current data set into a new Excel spreadsheet.

Exercise 2: Integrating another Dataset

In this exercise we are going integrate a `source` data set into a given `target` one. The source dataset contains similar data like the target one but with different column names and structure. Scenarios like this are very common in real life where we receive a lot of data sources about practically the same data. Ultimately, all of these sources of data we receive will need to be integrated/merge into

one final data set containing all of the data from all of the sources.

We are going to use different methods and techniques in OpenRefine in order to successfully integrate (merge) two data sets into one. This will involve splitting/merging fields, mapping different attributes, deduplicating the data etc.

This exercise also helps in demonstrating the first case (*extend down vertically*) of the *Joining datasets from different sources* chapter that you read earlier in the worksheet.

Data

For this task the data from the files `address_list_original.xlsx` & `address_list_scrambled.xlsx` are used.

Step 1: Creating the project

When you have downloaded both of the excel spreadsheets, create the project by:

- Uploading both of the files as the project files (OpenRefine supports multiple files import).
- Select both of the files in the **[Select Files to Import page]**.
- Leave the default parsing options and click on **[Create Project]**.

Step 2: Integrating the data

The *target dataset* is the same as the one from the first exercise and the purpose here is to integrate another data set i.e. the *source dataset* (with similar column names and values) into the *target dataset* including correct mapping and successful data integration and deduplication.

After creating the project, you will see a new column called `File` corresponding to the file that the record belongs to. This column `File` is also called the **key** column when merging two datasets because it is the column we rely upon for differentiating where the data is coming from (see the first case in the Joining datasets from different sources chapter earlier in the worksheet).

The dataset shows a sum of all rows from both files containing all columns from both files. After the records from the first file (original/source record) are over, the display of the records from the second file (encrypted/source record) starts. You will also notice that some columns have the same name (on purpose) and therefore each record contains data about these columns.

Below are some screenshots that show what the data looks like and how the records evolve from the first to the second file.

82 rows

Extensions: Wikidata

Show as: **rows** records Show: 5 10 **25** 50 rows

« first ‹ previous **1 - 25** next › last »

	All	File	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	ZipCd	Place
1.		address_list_original.xlsx#Exercise	101	Hauser	Pascal	1973-05-19T23:00:00Z	CH	M	ZH	Röschibachstr. 77	8037	Zürich
2.		address_list_original.xlsx#Exercise	102	Ragginger	Jean-Pierre	1983-09-28T23:00:00Z	CH	M	ZH	Saatlenzelg 24	8050	Zürich
3.		address_list_original.xlsx#Exercise	103	Faist	Jenny	1969-10-07T23:00:00Z	CH	F	ZH	Köschenrütistr. 69	8052	Zürich
4.		address_list_original.xlsx#Exercise	104	Silberer	Lucas	1977-06-23T22:00:00Z	CH	M	AG	Oberbodenstr. 10	5415	Nussbaumen AG
5.		address_list_original.xlsx#Exercise	105	Fillinger	Claude	1975-07-20T23:00:00Z	CH	M	ZH	Rieterstr. 93	8002	Zürich
6.		address_list_original.xlsx#Exercise	106	Baillie	Marianne	Mon Aug 11 00:00:00 CEST 1980	F	F	TG	Zielweg 5	8580	Amriswil
7.		address_list_original.xlsx#Exercise	107	Isler	Ruth	1979-11-20T23:00:00Z	CH	F	ZH	Weihermattstrasse 48	8902	Urdorf
8.		address_list_original.xlsx#Exercise	108	Vlassidis	Stamatis	1970-12-21T23:00:00Z	GR	M	ZH	Brunnacherstr. 34	8174	Stadel b. Niedergla
9.		address_list_original.xlsx#Exercise	109	Ambühler	Urs	1977-07-28T22:00:00Z	CH	M	ZH	Gerechtigkeitsgasse 4	8002	Zürich
10.		address_list_original.xlsx#Exercise	110	Kellenberger	Kurt	1969-07-11T23:00:00Z	CH	M	ZH	Georgsstrasse 19	8055	Zürich
11.		address_list_original.xlsx#Exercise	113	Peters	Oliver	1975-10-16T23:00:00Z	CH	M	ZH	Ernastr. 26	8004	Zürich
12.		address_list_original.xlsx#Exercise	114	Kopf	Andreas	1981-02-02T23:00:00Z	CH	M	ZH	Soodring 19/20	8134	Adliswil
13.		address_list_original.xlsx#Exercise	115	Wüger	Heinrich	1977-09-19T22:00:00Z	CH	M	AG	Alte Hägglingerstr. 10	5605	Dottikon
14.		address_list_original.xlsx#Exercise	116	Hutter	Julien	1974-05-28T23:00:00Z	CH	M	AG	Dorfstr. 48	5040	Schöftland
15.		address_list_original.xlsx#Exercise	117	Schneider-Weber	Marianne	1972-08-28T23:00:00Z	CH	F	ZH	Bahnhofstr. 13	8001	Zürich
16.		address_list_original.xlsx#Exercise	118	Behrens	Jasmine	1976-10-02T23:00:00Z	D	F		Marktplatz 5	79196	Waldshut-Tiengen
17.		address_list_original.xlsx#Exercise	119	Casanova	Antonio	1970-06-28T23:00:00Z	I	M	ZH	Berninastr. 67	8057	Zürich
18.		address_list_original.xlsx#Exercise	120	Maurer	Elisabeth	Fri Jul 16 00:00:00 CEST 1982	CH	F	ZH	Brunnengasse 8	8400	Winterthur
19.		address_list_original.xlsx#Exercise	121	Walder	Karl	1979-03-01T23:00:00Z	CH	M	ZH	Zeltweg 12	8610	Uster
20.		address_list_original.xlsx#Exercise	122	Hedbom	Conrad	1975-04-23T23:00:00Z	NL	M	ZH	Im Sträler 5	8047	Zürich
21.		address_list_original.xlsx#Exercise	123	Mende	Rudolf	1973-11-11T23:00:00Z	D	M	ZH	Spielweg 7	8037	Zürich

Figure 9. Data in the table

82 rows

Extensions: Wikidata

Show as: **rows** records Show: 5 10 **25** 50 rows

« first ‹ previous **51 - 75** next › last »

	All	File	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	ZipCd	Place	Domicile_Count
						00:00:00 CET 1971							
57.		address_list_original.xlsx#Exercise	169	Bosshard	Greti	Sat Jul 03 00:00:00 CET 1976	CH	F	ZH	Im Holzerhurd 11/172	8046	Zürich	CH
58.		address_list_original.xlsx#Exercise	170	Wernli	Thomas	Wed Jun 11 00:00:00 CET 1975	CH	M	AG	Ziegelrain 18	5000	Aarau	CH
59.		address_list_original.xlsx#Exercise	171	Herget	Dieter	Sat May 02 00:00:00 CEST 1981	CH	M	ZH	Hubstr. 47	8303	Bassersdorf	CH
60.		address_list_original.xlsx#Exercise	172	Spieler	Erich	Sun Jan 26 00:00:00 CET 1975	CH	M	ZH	Eichrainstr. 13	8052	Zürich	CH
61.		address_list_original.xlsx#Exercise	173	Meier	Pia	Tue Jan 13 00:00:00 CET 1976	CH	F	TG	Seestr. 23	8596	Scherzingen	CH
62.		address_list_original.xlsx#Exercise	176	Dällenbach	Werner	Sat Dec 17 00:00:00 CET 1977	CH	M	ZH	Pflanzschulstr. 4	8400	Winterthur	CH
63.		address_list_scrambled.xlsx#Exercise							ZH	Saatlenzelg 24		Zürich	CH

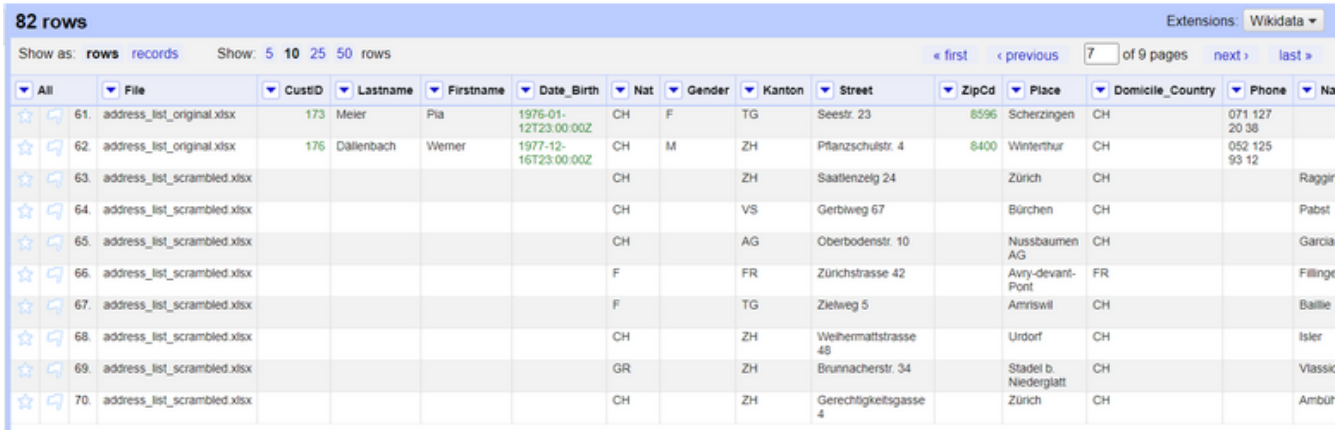
Figure 10. Data from the second table

Step 2.1 Joining and standardizing the columns from both files

First, we need to check the column names (from both files) and find out which ones of them belong to one-another. This part needs to be done manually.

When we find out the two columns corresponding to the same thing, we need to join them together. This will basically bring all the values from both of the files into a single column. Make sure that you leave the `separator` as an empty string when joining the columns.

You will see that in the columns that you joined, the first column will also take the values from the second column and we will have a value for each of our rows (from both of the files).



82 rows

Extensions: Wikidata

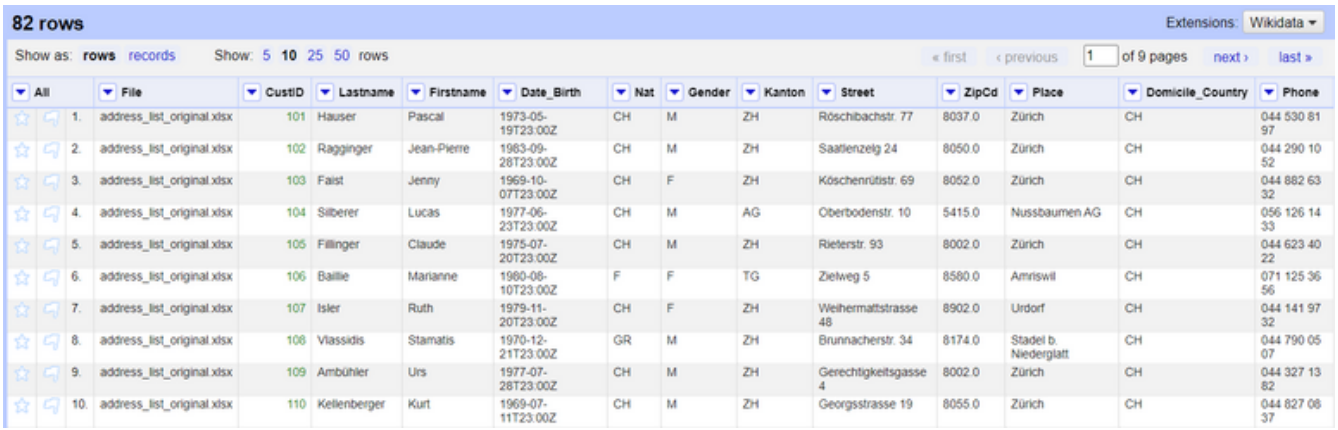
Show as: **rows** records Show: 5 **10** 25 50 rows « first ‹ previous 7 of 9 pages next › last »

	All	File	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	ZipCd	Place	Domicile_Country	Phone	Na
☆	61.	address_list_original.xlsx	173	Meier	Pia	1976-01-12T23:00:00Z	CH	F	TG	Seestr. 23	8596	Scherzingen	CH	071 127 20 38	
☆	62.	address_list_original.xlsx	176	Dällenbach	Werner	1977-12-16T23:00:00Z	CH	M	ZH	Pflanzschulstr. 4	8400	Winterthur	CH	052 125 93 12	
☆	63.	address_list_scrambled.xlsx					CH		ZH	Saatlenzelg 24		Zürich	CH		Raggi
☆	64.	address_list_scrambled.xlsx					CH		VS	Gerbiweg 67		Bürchen	CH		Pabst
☆	65.	address_list_scrambled.xlsx					CH		AG	Oberbodenstr. 10		Nussbaumen AG	CH		Garcia
☆	66.	address_list_scrambled.xlsx					F		FR	Zürichstrasse 42		Avry-devant-Pont	FR		Fillinge
☆	67.	address_list_scrambled.xlsx					F		TG	Zielweg 5		Amriswil	CH		Baillie
☆	68.	address_list_scrambled.xlsx					CH		ZH	Weihermattstrasse 48		Urdorf	CH		Isler
☆	69.	address_list_scrambled.xlsx					GR		ZH	Brunnacherstr. 34		Stadel b. Niederglatt	CH		Vlassi
☆	70.	address_list_scrambled.xlsx					CH		ZH	Gerechtigkeitsgasse 4		Zürich	CH		Ambüh

Figure 11. Joining the `Nat` and `Nationality` columns from both of the files.

Now do the same thing for all of the remaining columns and you will have a dataset with consistent columns.

When you have joined all the right columns, proceed by removing the other columns i.e. from the second file. This should leave you with a dataset containing all the data from both of the files, but with consistent and standardized column names. (Some columns can have the same name in both of the files, this was left on purpose).



82 rows

Extensions: Wikidata

Show as: **rows** records Show: 5 **10** 25 50 rows « first ‹ previous 1 of 9 pages next › last »

	All	File	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	ZipCd	Place	Domicile_Country	Phone
☆	1.	address_list_original.xlsx	101	Hauser	Pascal	1973-05-19T23:00Z	CH	M	ZH	Röschibachstr. 77	8037.0	Zürich	CH	044 530 81 97
☆	2.	address_list_original.xlsx	102	Ragginger	Jean-Pierre	1983-09-28T23:00Z	CH	M	ZH	Saatlenzelg 24	8050.0	Zürich	CH	044 290 10 52
☆	3.	address_list_original.xlsx	103	Faist	Jenny	1969-10-07T23:00Z	CH	F	ZH	Köschenrütistr. 69	8052.0	Zürich	CH	044 882 63 32
☆	4.	address_list_original.xlsx	104	Silberer	Lucas	1977-06-23T23:00Z	CH	M	AG	Oberbodenstr. 10	5415.0	Nussbaumen AG	CH	056 126 14 33
☆	5.	address_list_original.xlsx	105	Fillinger	Claude	1975-07-20T23:00Z	CH	M	ZH	Rieterstr. 93	8002.0	Zürich	CH	044 623 40 22
☆	6.	address_list_original.xlsx	106	Baillie	Marianne	1960-08-10T23:00Z	F	F	TG	Zielweg 5	8580.0	Amriswil	CH	071 125 36 56
☆	7.	address_list_original.xlsx	107	Isler	Ruth	1979-11-20T23:00Z	CH	F	ZH	Weihermattstrasse 48	8902.0	Urdorf	CH	044 141 97 32
☆	8.	address_list_original.xlsx	108	Vlassidis	Stamatis	1970-12-21T23:00Z	GR	M	ZH	Brunnacherstr. 34	8174.0	Stadel b. Niederglatt	CH	044 790 05 07
☆	9.	address_list_original.xlsx	109	Ambühler	Urs	1977-07-28T23:00Z	CH	M	ZH	Gerechtigkeitsgasse 4	8002.0	Zürich	CH	044 327 13 82
☆	10.	address_list_original.xlsx	110	Kellenberger	Kurt	1969-07-11T23:00Z	CH	M	ZH	Georgsstrasse 19	8055.0	Zürich	CH	044 827 08 37

Figure 12. Dataset after joining the columns from both of the files.

Step 2.2 Deduplicating the data

The next step is to deduplicate the data. Some data from the second file is also present on the first file and this means that they are duplicates and we need to remove them and only keep one of the entries. There is also other data on the second file which is not present on the first file (i.e. not duplicate) so we shouldn't remove these.

For a record to be considered duplicate, we will be considering the following condition: If the first name, last name and birth date are the same, then we are talking about the same person, i.e. duplicate.

In order to do this, we need to use the *Duplicates facet* feature which is located under **Facet › Customized facets › Duplicates facet**.

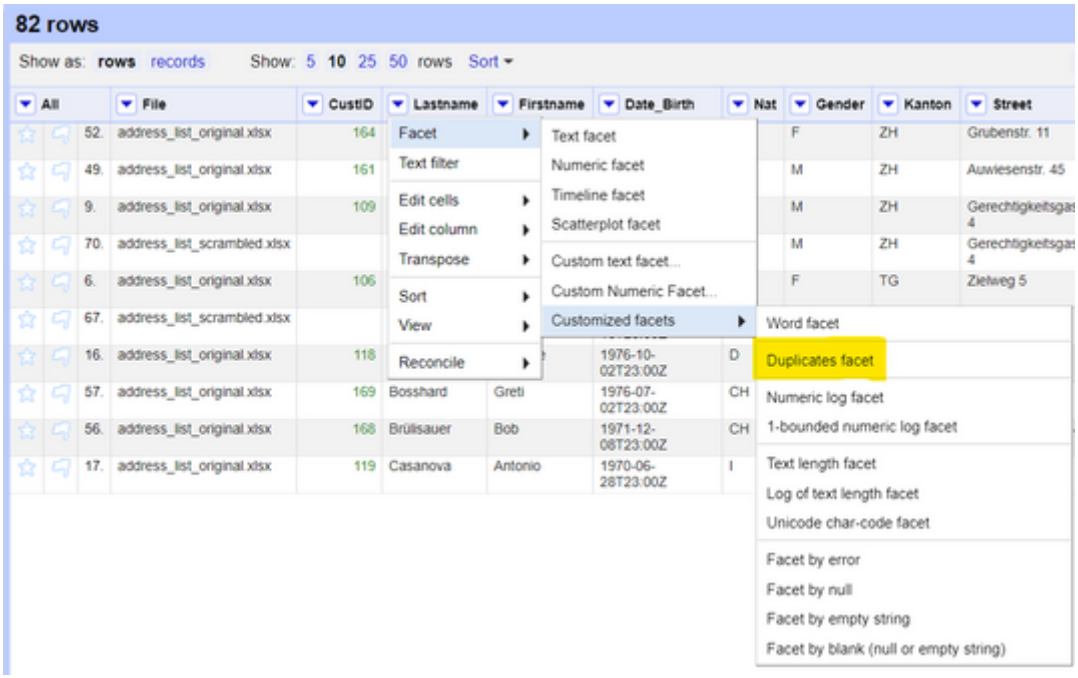


Figure 13. Duplicate facets option on OpenRefine.

This facet returns `false` if the value is not duplicate and `true` if the value is a duplicate. The idea in our case is to create three duplicate facets which check for duplicates on the *Firstname* AND *Lastname* AND *Date_Birth* column.

Go ahead and create three facets for the three columns and select the `true` value on each one of them. This will return the columns that are now duplicates on all three values of the selected columns (our target).

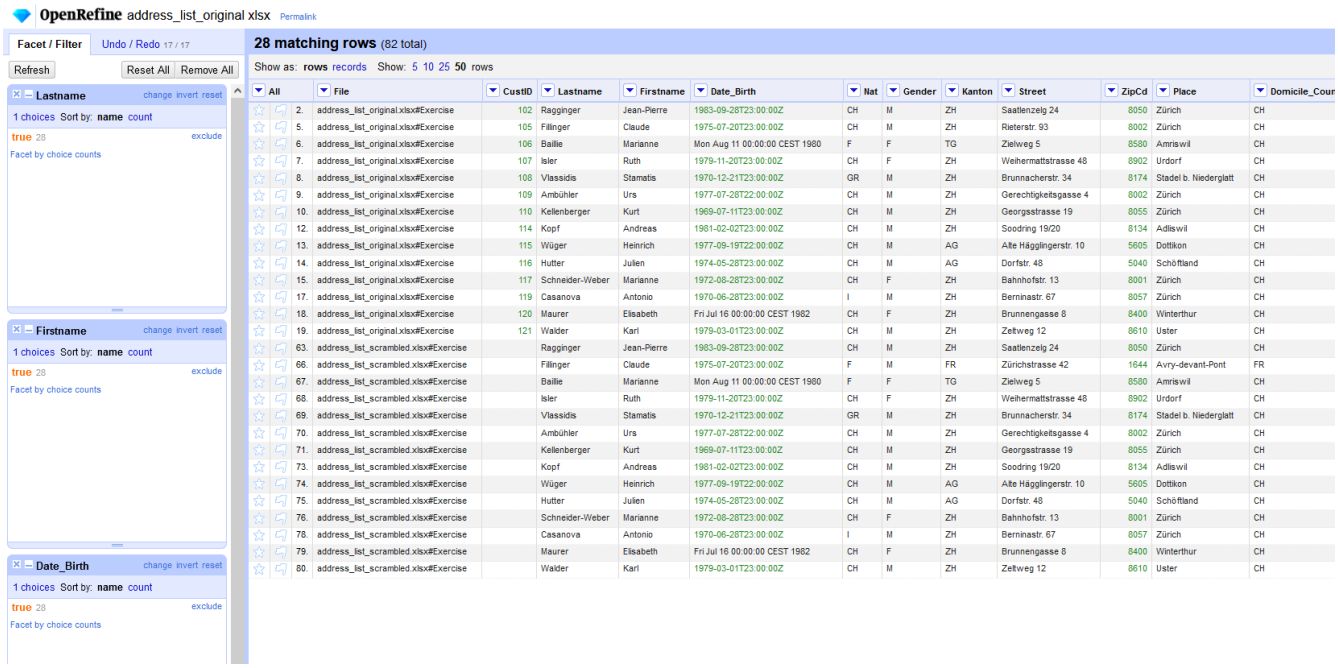


Figure 14. Applying all of the duplicate facets on the dataset.

Now all duplicate records remain, i.e. same first name, same last name and same date of birth. The next task is to sort the records alphabetically by one of the three columns (e.g. *Firstname*). After sorting the column, select **[Sort]** and then **[Reorder rows permanently]**. This is necessary to use the next function, which hides the duplicate records (except the first one).

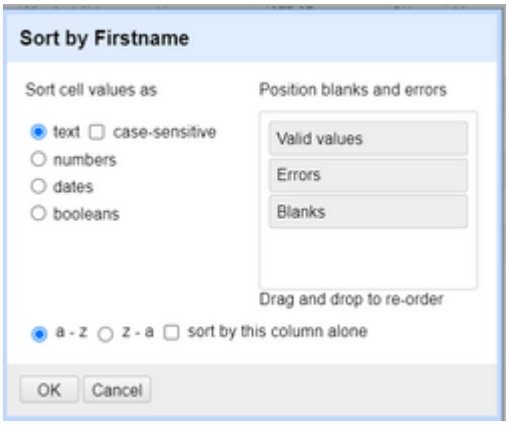


Figure 15. Sorting by the `Firstname` column.

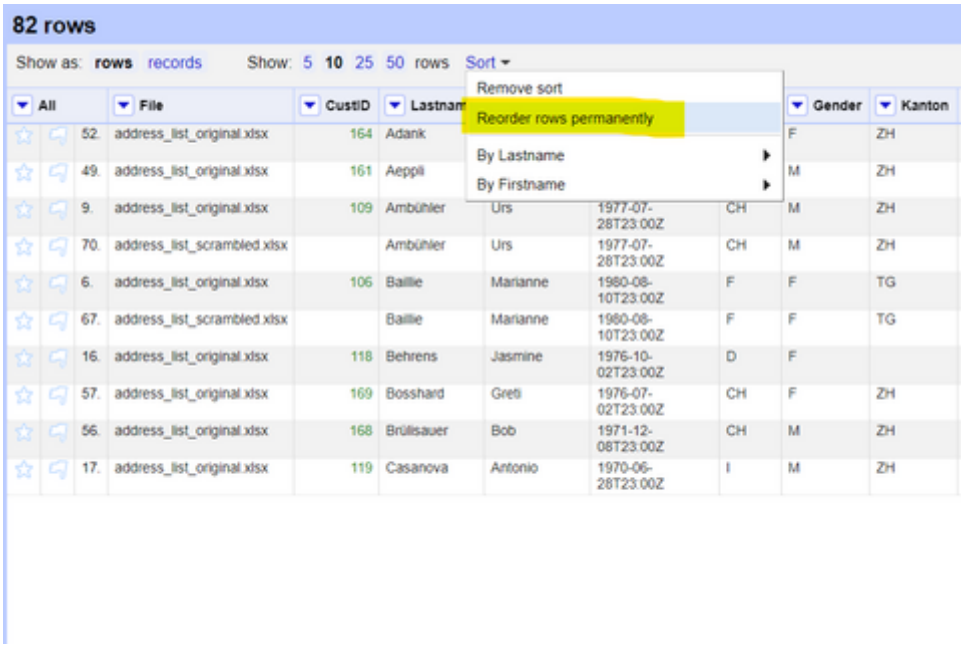


Figure 16. Persisting the sorted values.

When you have permanently re-ordered the rows (necessary step), go ahead and select the column you sorted and click on **Edit Cells › Blank down**. This will hide all the duplicate records (besides from the first one), which is exactly what we need.

Now you need to face the lines after spaces. This is a custom OpenRefine facet that you can find at **Facet › Customized Facets › Facet by blank (null or empty string)**.

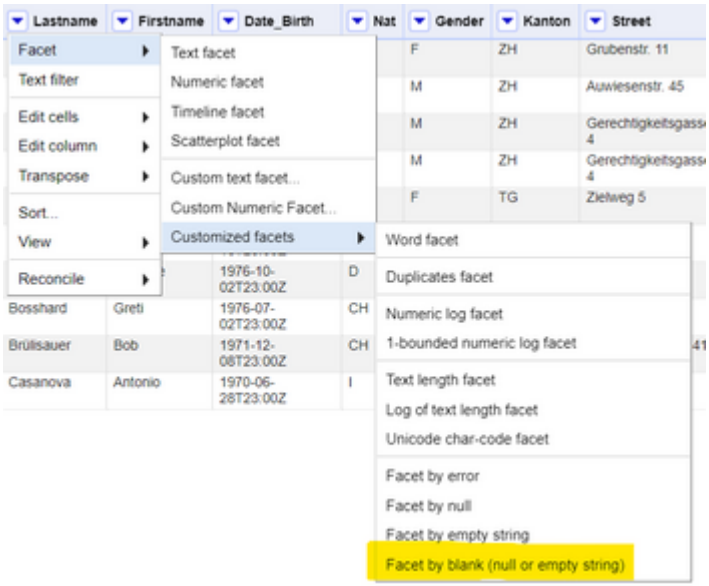
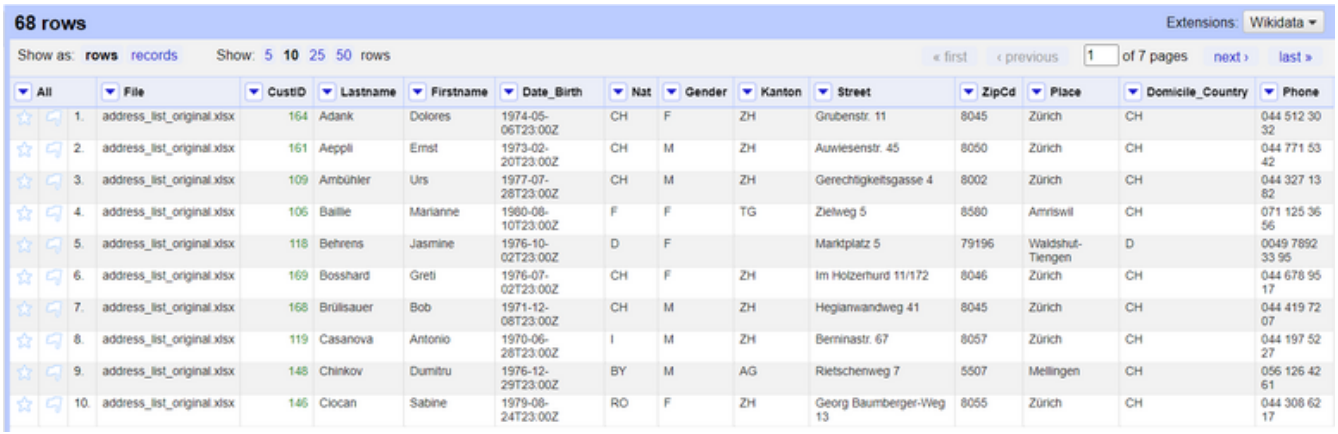


Figure 17. `Facet by blank` *option on OpenRefine.*

For the facet created with the **[facet by blank]** option, select `true` so that only the duplicate records (our target) are left. Now click on the *All* column and select **Edit Rows › Remove matching rows**. This will remove all rows from the current selection (the duplicate records).

Remove all facets by clicking on **[Remove All]** and you will have only unique records.



All	File	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	ZipCd	Place	Domicile_Country	Phone
1.	address_list_original.xlsx	164	Adank	Dolores	1974-05-06T23:00Z	CH	F	ZH	Grubenstr. 11	8045	Zürich	CH	044 512 30 32
2.	address_list_original.xlsx	161	Aeppli	Ernst	1973-02-20T23:00Z	CH	M	ZH	Auwiesenstr. 45	8050	Zürich	CH	044 771 53 42
3.	address_list_original.xlsx	109	Ambühler	Urs	1977-07-28T23:00Z	CH	M	ZH	Gerechtigkeitsgasse 4	8002	Zürich	CH	044 327 13 82
4.	address_list_original.xlsx	106	Baillie	Marianne	1960-08-10T23:00Z	F	F	TG	Zielweg 5	8580	Amriswil	CH	071 125 36 56
5.	address_list_original.xlsx	118	Behrens	Jasmine	1976-10-02T23:00Z	D	F		Marktplatz 5	79196	Waldshut-Tiengen	D	0049 7892 33 95
6.	address_list_original.xlsx	169	Bosshard	Greti	1976-07-02T23:00Z	CH	F	ZH	Im Holzerhurd 11/172	8046	Zürich	CH	044 678 95 17
7.	address_list_original.xlsx	168	Brülisauer	Bob	1971-12-08T23:00Z	CH	M	ZH	Hegianwandweg 41	8045	Zürich	CH	044 419 72 07
8.	address_list_original.xlsx	119	Casanova	Antonio	1970-06-28T23:00Z	I	M	ZH	Berninastr. 67	8057	Zürich	CH	044 197 52 27
9.	address_list_original.xlsx	148	Chinkov	Dumitru	1976-12-29T23:00Z	BY	M	AG	Rietschenweg 7	5507	Mellingen	CH	056 126 42 61
10.	address_list_original.xlsx	146	Ciocan	Sabine	1979-08-24T23:00Z	RO	F	ZH	Georg Baumberger-Weg 13	8055	Zürich	CH	044 308 62 17

Figure 18. Records after deduplication (and validation).

Step 3: Finishing up

On the *Phone* column (or *Phone_Number*) there are some phone numbers with country code and some without. There's also spaces on some of the numbers and not on others.

Write a text transform function using GREL on the *Phone* column that will standardize all it's column values. You can do this by either adding the country code to all of them (who don't have it) or by removing it from the ones who have it. You also need to remove spaces from the values that have them, or add spaces to the ones who don't, it's up to you.

After you finish this last task, you will have integrated, validated and deduplicated a source dataset into a target dataset successfully.

Step 4: Exporting the data

Export the data into the *MS Excel 2007+ (.xslx)* format in order to finish this exercise.

Enriching Data with Geocoding

Geocoding is the process of converting/transforming a human-readable description of a location, such as an address or a name of the place in the actual location of it in the world (geospatial data). Geocoding is an important asset to geospatial data and location analytics. The idea of geocoding is to input the description of a place and get back the exact location (e.g. longitude and latitude, `lat/lon`). **Reverse geocoding** is another concept (not as widely used) which is the opposite of geocoding, meaning, inputting the exact location of a place and having the address or place outputted back to you.

Geocoding can be done through online web applications or web services (APIs). There are several geocoding APIs that we can use and they usually come with a cost.

In this worksheet, we will be demonstrating geocoding using **Nominatim** (see Nominatim API, which relies on the open database OpenStreetMap. This is an example of a Nominatim API call for the address `Obere Bahnhofstrasse 32b, Rapperswil` (from the documentation): `https://nominatim.openstreetmap.org/search?format=xml&addressdetails=1&countryCodes=CH&format=geojson&limit=1&q=32b+Obere+Bahnhofstrasse,+Rapperswil`.

This results in the following GeoJSON (JSON) document (edited and shortened for demonstration purposes):


```
{
    "type": "FeatureCollection",
    "licence": "Data © OpenStreetMap contributors, ODbL 1.0.",
    "features": [{
        "type": "Feature",
        "properties": {
            "osm_id": 3124300001,
            "osm_type": "node",
            "importance": 0.42099999999999993
        },
        "geometry": {
            "type": "Point",
            "coordinates": [8.8190421, 47.2269746]
        }
    }]
}
```


You can find more about JSON at www.json.org.

The Nominatim Usage Policy for example states that a maximum of repeated requests has to be done one per second. This means that you have throttle calls

with a delay to 1000ms.

Exercise 3: Geocoding with OpenRefine

In this part you will use Nominatim as a geocoding service together with OpenRefine. Since our dataset contains information about the customer's address and country, geocoding will be applied to a dataset.

This exercise also helps in demonstrating the second case (*extend down horizontally by one or few columns*) of the *Joining datasets from different sources* chapter that you read earlier in the worksheet.

Data

For this task, the data from the file `address_list_original.xlsx` is used.

Step 1: Creating the Project

After you have the right dataset, open the OpenRefine program and create the project using the dataset you just downloaded.

With OpenRefine we will do the Nominatim API requests, fetch the data, parse the lat/lon out of it, then put it into the right columns.

Step 2: Joining the address columns

We need to concatenate address columns to one single column *Full_Address* in order to be ready for the geocoding call. We can do this using column joins and other functions, as follows:

1. Start by creating a new column which concatenates the *Street* and *Place* column. This will give us a new column which contains the street and the place.
2. Navigate to the *Street* column, clicking on the dropdown and selecting **Edit Column › Join columns...**. This will open a pop-up window where you can specify which columns you want to join and the way you want to join them.
3. Select the *Street* and *Address* columns on the left and specify the `,` separator between the content of each column on the right. Also make sure you select the **[Write result in new column named...]** radio button and specify the name as *Full_Address* because we want to create a new column out of these two.

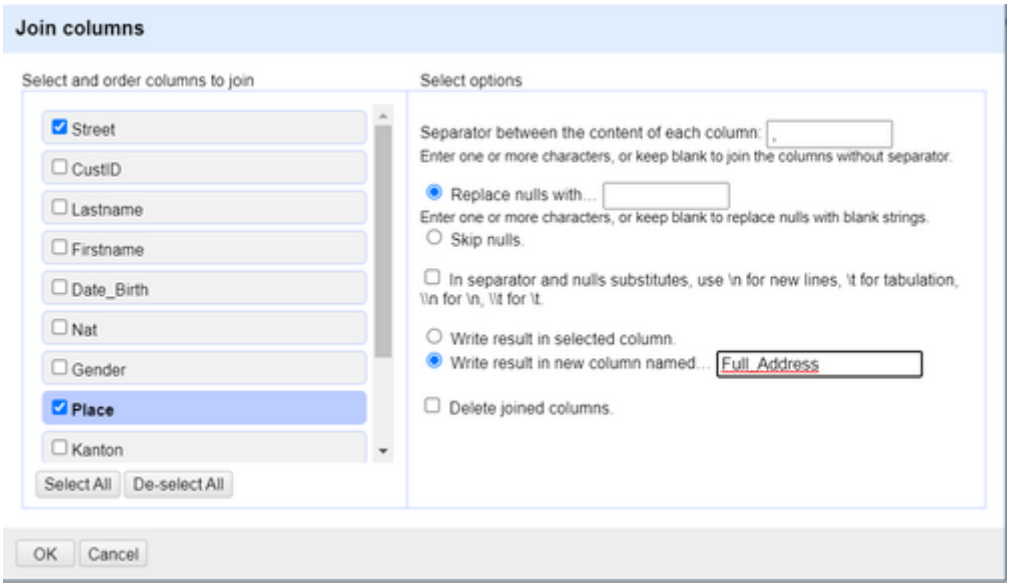
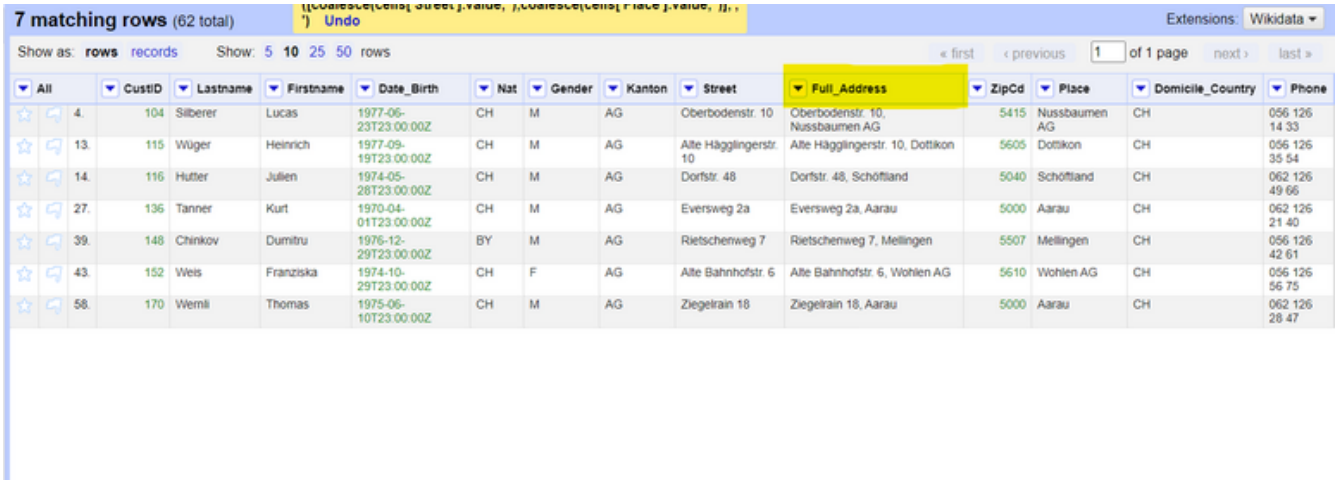


Figure 19. Joining/concatenating columns on OpenRefine.

1. Now you will have a new column called *Full_Address* which contains the address (with number) and the place part of the address. This will help the geocoding API with determining the location of the customer's addresses.



All	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	Full_Address	ZipCd	Place	Domicile_Country	Phone
4.	104	Silberer	Lucas	1977-06-23T23:00:00Z	CH	M	AG	Oberbodenstr. 10	Oberbodenstr. 10, Nussbaumen AG	5415	Nussbaumen AG	CH	056 126 14 33
13.	115	Wüger	Heinrich	1977-09-19T23:00:00Z	CH	M	AG	Alte Hägglingerstr. 10	Alte Hägglingerstr. 10, Dottikon	5605	Dottikon	CH	056 126 35 54
14.	116	Hutter	Julien	1974-05-28T23:00:00Z	CH	M	AG	Dorfstr. 48	Dorfstr. 48, Schöftland	5040	Schöftland	CH	062 126 49 66
27.	136	Tanner	Kurt	1970-04-01T23:00:00Z	CH	M	AG	Eversweg 2a	Eversweg 2a, Aarau	5000	Aarau	CH	062 126 21 40
39.	148	Chinkov	Dumitru	1976-12-29T23:00:00Z	BY	M	AG	Rietschenweg 7	Rietschenweg 7, Mellingen	5507	Mellingen	CH	056 126 42 61
43.	152	Weis	Franziska	1974-10-29T23:00:00Z	CH	F	AG	Alte Bahnhofstr. 6	Alte Bahnhofstr. 6, Wohlen AG	5610	Wohlen AG	CH	056 126 56 75
58.	170	Wemli	Thomas	1975-06-10T23:00:00Z	CH	M	AG	Ziegelrain 18	Ziegelrain 18, Aarau	5000	Aarau	CH	062 126 28 47

Figure 20. OpenRefine grid after joining the `Address` + `Place` columns.

Step 3: Using a Geocoding API with OpenRefine

Now, we have to call a Geocoding API using OpenRefine which will give us more information about our address, including lat/lon attributes.

1. We have to make a request to the Nominatim API where we will split the *Full_Address* column value and send it to the API and in return receive information about that location including the lat/lon (our objective) of it.
2. Start by clicking on the drop-down button of *Full_Address* column and select **Edit Column › Add column by fetching URLs...**.
3. Specify a new name for the column (*address_json* or *osm_json*, it doesn't really matter), change the *throttle delay* to 1000ms and write the expression as: `'https://nominatim.openstreetmap.org/search?street=' + escape(value.split(",")[0], 'url') + '&city=' + escape(value.split(",")[1], 'url') + '&format=json'` and click on **[OK]**. This code

will split the address column into two values (one contains the city and the other contains the place) and make an API request for receiving a GeoJSON object with the lat/lon coordinates of the location (the process of geocoding). We will also receive other information about the location but we are going to ignore those for now.

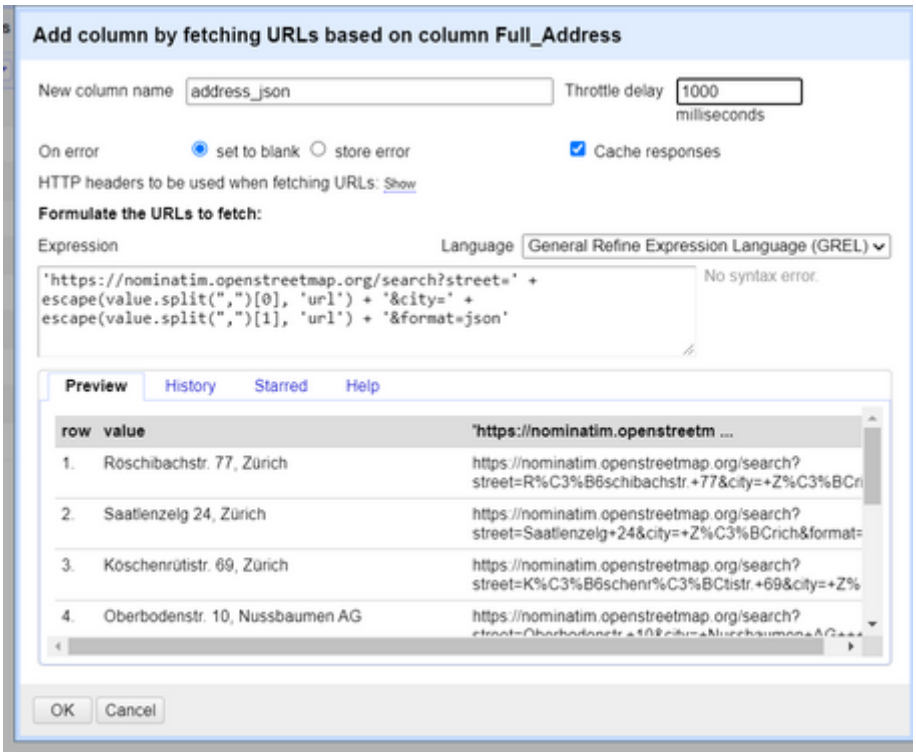


Figure 21. Adding a column by fetching an URL (API Call) using GREL.

4. Now you will have a new column containing the API response for the request we made, if you format/beautify this JSON code, you will see that it contains information about the location including lat/lon attributes, the bounding box, OpenStreetMap ID/Type etc. as shown above.

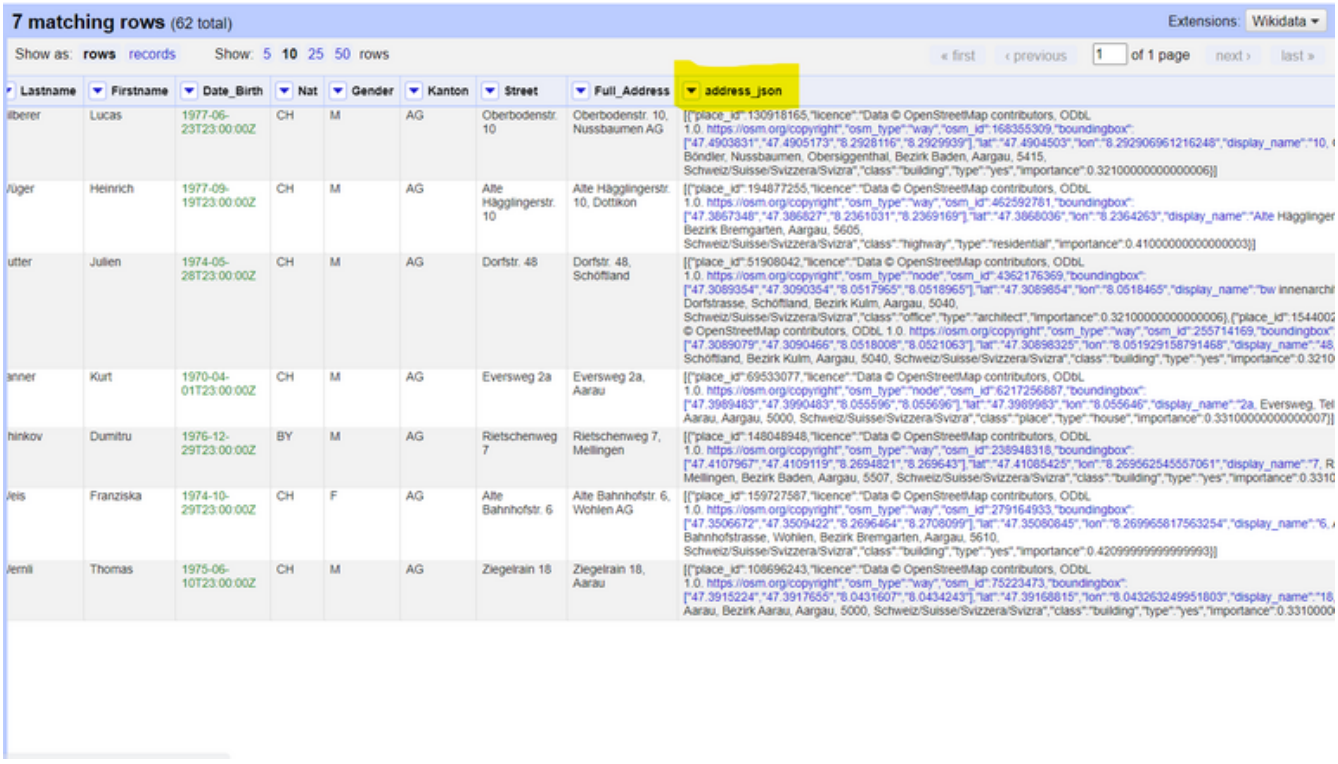


Figure 22. OpenRefine grid after the `address_json` column is added".

please be patient. If you want to complete the process faster, you can use facets to narrow down the dataset to less records, e.g. by filtering only the records with the value `AG` on the *Kanton* column.

Step 4: Parsing the JSON and creating the coordinate (lat/lon) columns

Now that we have the JSON object containing information about our customer's locations, we can use that JSON object to create new columns out of it, such as lat/lon.

In order to do this, we need to:

1. Click on the drop-down of the newly column you just created by calling the API.
2. Click on **Edit column › Add column based on this column...** (Insert picture here).
3. Specify a name for the new column(e.g. *lat* or *latitude*) and use the *Expression* box to parse the JSON and read the desired attribute by typing: `value.parseJson()[0].lat`. You can see the preview of the column after evaluating the expression. Click **[OK]** and you will see a new column for the latitude coordinate.

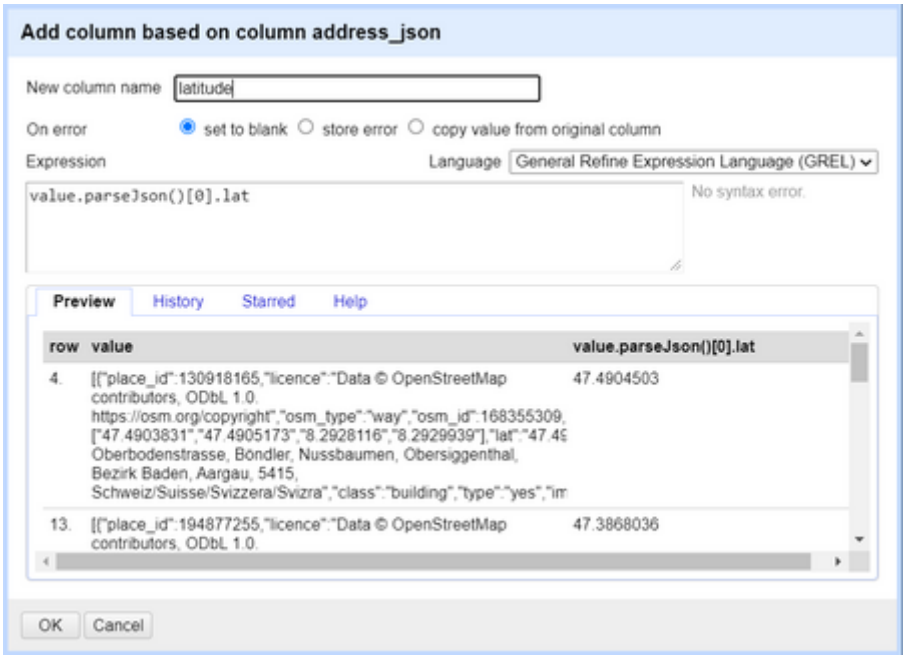
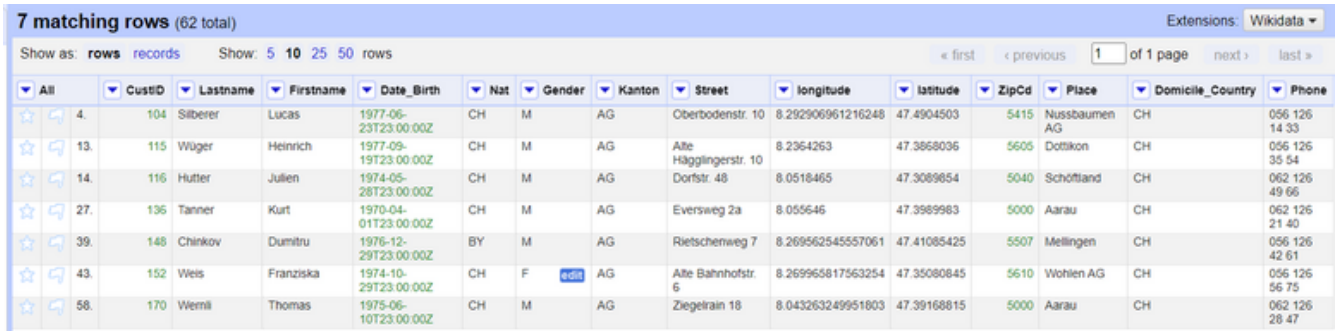


Figure 23. Extracting the `latitude` attribute from the json object we received.

4. Do the same thing for the longitude coordinate `lon`.
5. After you have the lat/lon columns, finish up by deleting the column with the full json string for a cleaner look. You can also remove the *Full_Address* column if you like, since it's not necessary anymore.

Now you have a clean dataset with *lat* and *lon* coordinates for the customer addresses retrieved using OpenRefine and geocoding APIs. There may be some locations that were not found by the API. In this case, you can use a facet to filter out the empty values, and then manually modify these addresses to find out what is wrong with them, and then make the API request again.



7 matching rows (62 total) Extensions: Wikidata

Show as: rows records Show: 5 10 25 50 rows « first ‹ previous 1 of 1 page next › last »

	All	CustID	Lastname	Firstname	Date_Birth	Nat	Gender	Kanton	Street	longitude	latitude	ZipCd	Place	Domicile_Country	Phone
☆	4.	104	Silberer	Lucas	1977-06-23T23:00:00Z	CH	M	AG	Oberbodenstr. 10	8.292906961216248	47.4904503	5415	Nussbaumen AG	CH	056 126 14 33
☆	13.	115	Wüger	Heinrich	1977-09-19T23:00:00Z	CH	M	AG	Alte Hägglingerstr. 10	8.2364263	47.3868036	5605	Dottikon	CH	056 126 35 54
☆	14.	116	Hutter	Julien	1974-05-28T23:00:00Z	CH	M	AG	Dorfstr. 48	8.0518465	47.3089854	5040	Schöftland	CH	062 126 49 66
☆	27.	136	Tanner	Kurt	1970-04-01T23:00:00Z	CH	M	AG	Eversweg 2a	8.055646	47.3989983	5000	Aarau	CH	062 126 21 40
☆	39.	148	Chinkov	Dumitru	1976-12-29T23:00:00Z	BY	M	AG	Rietschenweg 7	8.269562545557061	47.41085425	5507	Mellingen	CH	056 126 42 61
☆	43.	152	Weis	Franziska	1974-10-29T23:00:00Z	CH	F edit	AG	Alte Bahnhofstr. 6	8.269965817563254	47.35080845	5610	Wohlen AG	CH	056 126 56 75
☆	58.	170	Wernli	Thomas	1975-06-10T23:00:00Z	CH	M	AG	Ziegelrain 18	8.043263249951803	47.39168815	5000	Aarau	CH	062 126 28 47

Figure 24. Final look of the dataset after geocoding and cleaning it.

Web scraping

In this section, we are going to do some web scraping using OpenRefine. We can use functions to first fetch and parse a web page, which consists of HTML text, and then filter the data to the desired columns.

OpenRefine's GREL functions allow us to parse the HTML page into HTML text content and then use different methods for selecting the right tags, attributes, text nodes etc. .

Let's first explain a typical HTML structure and then what GREL is.

HTML and DOM

HTML is the standard markup language for web pages. It is not a programming language but rather an markup language which describes the structure of a web page.

Web pages are written in HyperText Markup Language (HTML). HTML is the standard language for documents designed to be displayed in a web browser.

The Document Object Model (DOM) is the data representation of the elements/objects that contribute to the content of a HTML document. The DOM plays a crucial part in web scraping since that can be used to access elements inside a web page (just like the DOM selector methods in JavaScript).

You can find more about HTML here and about Javascript DOM here.

General Refine Expression Language (GREL)

A key feature of OpenRefine is the `General Refine Expression Language`, short GREL. GREL is an expression language specific to OpenRefine, which is similar to JavaScript and is convenient for performing different functions such as:

- String operations
- Boolean operators
- Parsing HTML, JSON or XML
- Selecting HTML elements
- Iterating over elements etc.

GREL is going to be used in our case to access the HTML DOM and select the appropriate data out of the HTML page for scraping.

For more information on GREL and its functions, you can read the official documentation from OpenRefine on these pages: GREL and GREL functions.

Exercise 4: Web scraping with OpenRefine

For this exercise, we are going to

- Bring the HTML content into OpenRefine.
- Parse the HTML and its elements using OpenRefine's expression language GREL.
- Arrange the resulting columns.

We are going to scrape a Wikipedia page that contains a list of all the castles in the `Aargau` canton of Switzerland, like for example the Habsburg! First, we are going to bring the pure HTML content of that page into OpenRefine and then we are going to use different GREL HTML and text transform functions to parse out the correct data from the HTML content.

This is the Wikipedia web page we use in this exercise: `Liste von Burgen und Schlössern im Kanton Aargau`.

Step 1: Creating the `HTML` project

There exist multiple ways for bringing the HTML content of a web page into OpenRefine, however for this exercise, we are going to use the **[Clipboard]** option of OpenRefine. Proceed by copying the link of the web page and then paste it into the *Clipboard* text area of OpenRefine's initial page.

Figure 25. Using the `Clipboard` option to create a project in OpenRefine.

Configure your parsing options (leave them as default), name your project, click on **[Create Project]** and move on to the next step.

Step 2: Bringing the HTML content into OpenRefine

Now you will have one column and one cell simply containing the same link you copied into OpenRefine.

Figure 26. The single column containing the URL of the webpage we are going to scrape.

Now click on the dropdown of that column and select **Edit Column › Add column by fetching URLs...**. Simply give a name to the new column, e.g. *html* or *html_content*, leave the expression as `value` and click on **[OK]**.

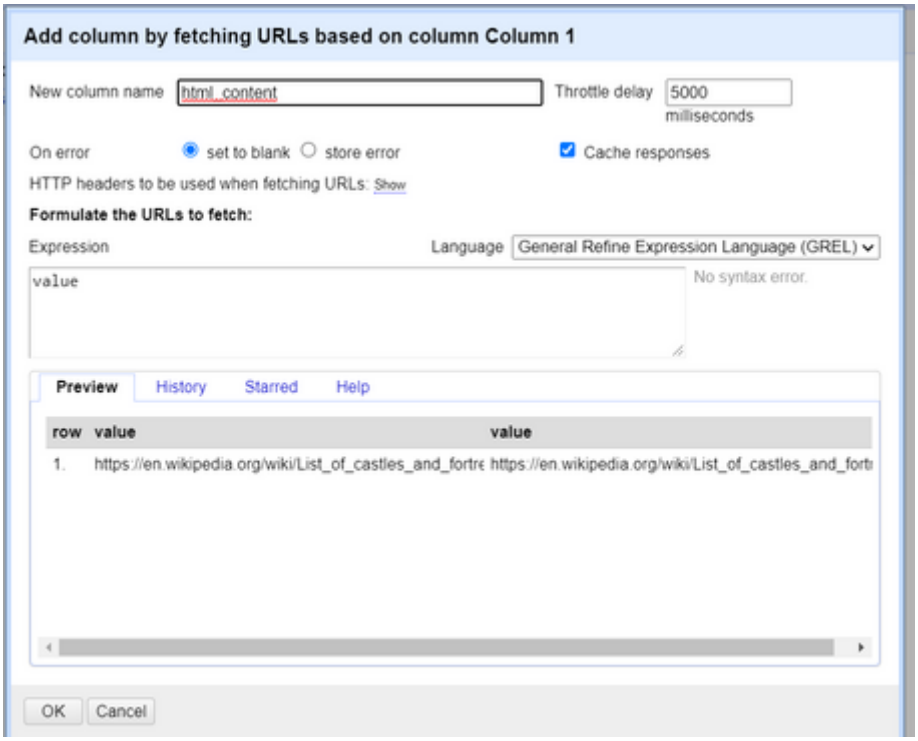


Figure 27. Adding the whole HTML content in a new column using OpenRefine's `Add column by fetching URL` *option.*

Now a new column will be created containing the whole pure HTML content of the web page we selected. We are going to use this column to extract the necessary information we need for the castles data.

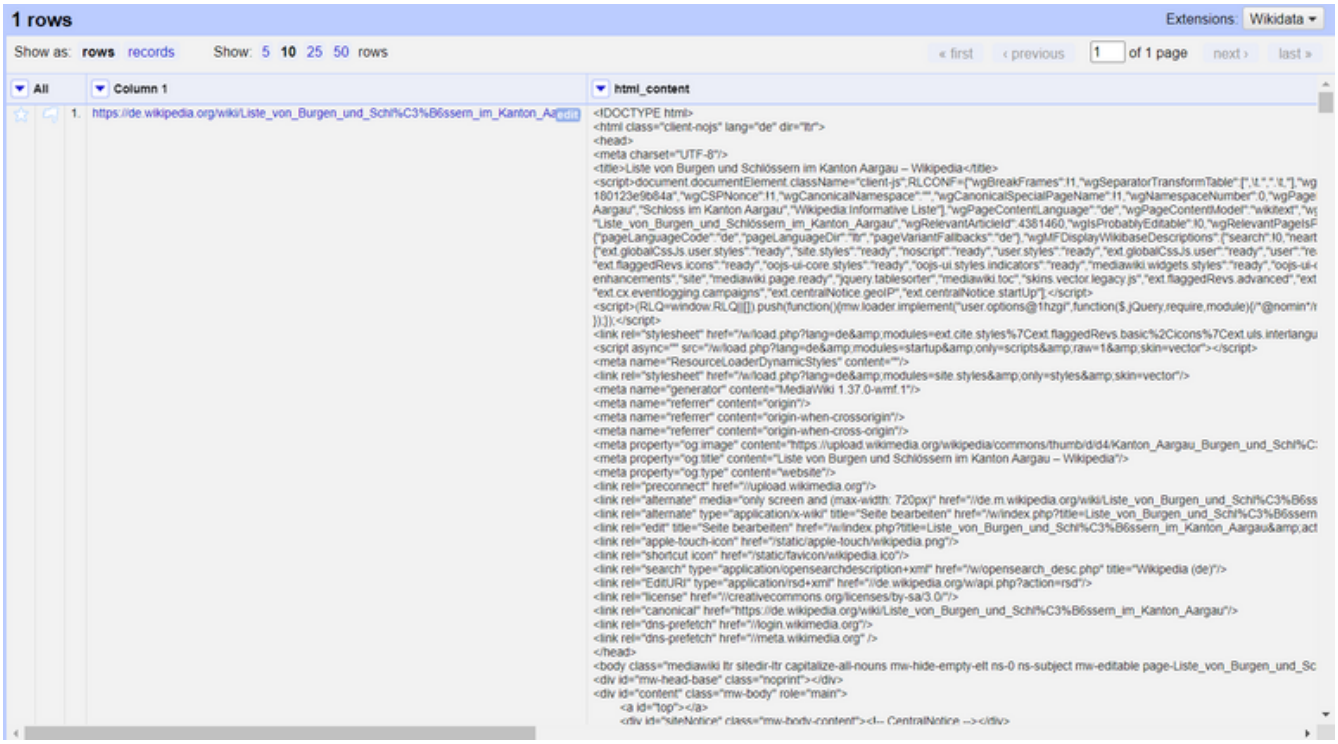


Figure 28. The `html_content` *column containing the full HTML content after adding the column by fetching the URL.*

Step 3: Parsing the HTML and its elements using GREL

Now we are going to use GREL's HTML parsing functions to parse the HTML and place it into the

appropriate columns.

We are going to use a *forEach* iterative loop from GREL to loop through the appropriate elements and only extract the data about the castles. In the loop, we are going to use GREL's HTML parsing functions and basic CSS selectors logic to select the needed HTML elements.

1. Click on the drop-down of the HTML content column you just created and click on **Edit column › Add column based on this column...**.
2. Place the following code into the expression box:

```
forEach(value.parseHtml().select("table.wikitable tbody tr td:first-child > a"), e,
e.ownText()).join("|").
```

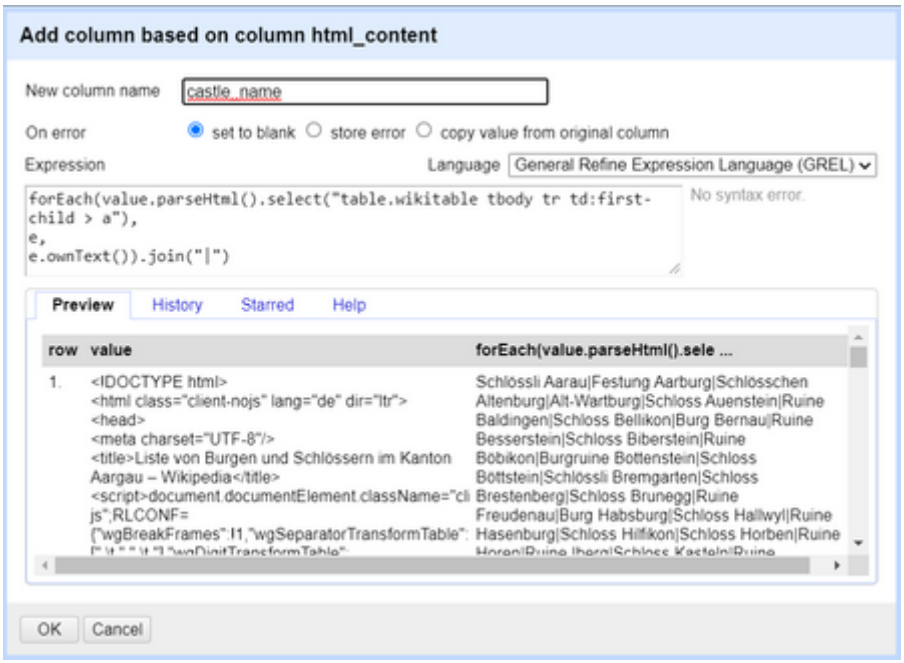


Figure 29. Using GREL to filter out the data we need from the HTML content.

3. This code loops through the elements specified in the first argument of the *forEach* loop using basic CSS selectors, gives a name to each element in the current iteration of the loop in the second argument (in our case: `e`) and then performs an action on each loop to that element (`e.ownText()`). After the loop is finished, we receive an array of those values, which we join with a `|` separator to then split into multiple cells.
4. We are left with a new column that contains a string of all the names of the castles, joined with a `|` character.
5. Now we have to split the string we received into multiple cells by separating them with the `|` separator (the reason why we did it in the first place). Click on the drop-down of the column you just created and click on **Edit Cells › Split multi-valued cells...**. Choose the `|` separator and click on **[OK]**.

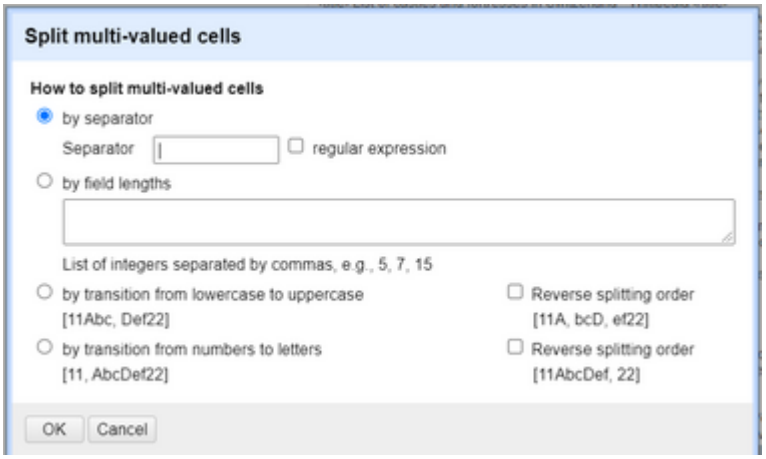


Figure 30. Using the `Split multi-valued cells` option to split the string we created before using GREL.

6. Now your new column should contain one row for each castle (containing the castle name).
7. Now proceed by extracting out the other necessary information castles such as: location, type, date, notes etc. It follows the same logic as above but you might have to change the CSS selectors and the action performed with the loop variable (e.g. `e.htmlAttr("title")` extracts the title attribute of the link etc).

When you have extracted all the necessary information for the castles, clean up your data set by removing the HTML content columns and other unnecessary data/columns. You will be left with a table containing the castles and fortresses of Switzerland, which you scraped from Wikipedia using OpenRefine.

There are multiple ways you can extract HTML content but as soon as you can receive all the information you need effectively, all of them are valid.

49 rows

Show as: **rows** records Show: 5 10 **25** 50 rows

All		name	ortschaft	jahr	typ	zustand
1.	Schlössli Aarau	Aarau	1240	Schloss	erhalten
2.	Festung Aarburg	Aarburg	1123	Festung	erhalten
3.	Schlösschen Altenburg	Altenburg bei Brugg	370	Kastell	erhalten
4.	Alt-Wartburg	Oftringen	1200	Burg	Ruine
5.	Schloss Auenstein	Auenstein AG	1307	Schloss	erhalten
6.	Ruine Baldingen	Baldingen AG	unbekannt	Burg	verfallen
7.	Schloss Bellikon	Bellikon	1430-1440	Schloss	erhalten
8.	Burg Bernau	Leibstadt	1157	Burg	Ruine
9.	Ruine Besserstein	Villigen	unbekannt	Burg	verfallen
10.	Schloss Biberstein	Biberstein	1280	Schloss	erhalten
11.	Ruine Böbikon	Böbikon	unbekannt	Burg	Ruine
12.	Burgruine Bottenstein	Zofingen	Mitte 13. Jahrhundert	Burg	Ruine
13.	Schloss Böttstein	Böttstein	1100-1200	Schloss	erhalten
14.	Schlössli Bremgarten	Bremgarten AG	1238	Schloss	erhalten
15.	Schloss Brestenberg	Seengen	1625	Schloss	erhalten
16.	Schloss Brunegg	Brunegg	1250	Schloss	erhalten
17.	Ruine Freudenau	Untersiggenthal	1240	Burg	Ruine
18.	Burg Habsburg	Habsburg AG	1030	Burg	erhalten
19.	Schloss Hallwyl	Seengen	1265	Schloss	erhalten
20.	Ruine Hasenburg	Bergdietikon	unbekannt	Burg	verfallen
21.	Schloss Hilfikon	Hilfikon	unbekannt	Schloss	erhalten
22.	Schloss Horben	Beinwil (Freiamt)	1700	Schloss	erhalten
23.	Ruine Horen	Küttigen	unbekannt	Burg	Ruine
24.	Ruine Iberg	Riniken	11. Jahrhundert	Burg	verfallen
25.	Schloss Kasteln	Oberflachs	1238	Schloss	erhalten

Figure 31. Castles of Switzerland dataset after scraping the web page, transforming it and cleaning the data in OpenRefine.

There are some GREL functions that are necessary for extracting some of the HTML information from the HTML content here. You can take a look at them at the section above about GREL.

Conclusion and Outlook

From the points discussed and explained above, you can see that OpenRefine is a handy tool for working with data, messy or not. However, it is important to know the right functions when working with data!

A lot of logical, technical and practical knowledge is required to work with data and turn it into something useful and meaningful. Hopefully, in the exercises you learned how to deal with messy data and which functions/tools to choose for different data problems.

What we have learned

- Creating a project in OpenRefine.
- Exploring a dataset and transforming it with OpenRefine, using facets and other transform functions.
- Using OpenRefine to clean/deduplicate and integrate a data set into another one.
- Using OpenRefine for geocoding and for web scraping.

What you haven't learned here are many other built-in features of OpenRefine - including, for example, date functions - and more complex analyses, such as statistical data descriptions (mean, median), and especially what data matching with OpenRefine is. To learn more, check out the many resources mentioned in the infobox below.

Recommended reading: The book `Using OpenRefine` by Verborgh & De Wilde (Packt Publishing Ltd, 2013) is freely availably online. And there are many online tutorials and videos, like for example the Library Carpentry OpenRefine lesson or the Introduction video by the University of Idaho Library.

Open questions? Feel free to contact OpenStreetMap Schweiz or Stefan Keller!