Università degli Studi di Modena e Reggio Emilia

DIPARTIMENTO DI INGEGNERIA ENZO FERRARI
Corso di Laurea in Ingegneria Informatica

Analisi del tool per la preparazione dei dati OpenRefine

Analysis of the OpenRefine data preparation tool

Relatore
Prof.ssa Sonia Bergamaschi

Correlatore
Dott. Luca Zecchini

Candidato
Enrico De Luca

Anno Accademico 2019/2020

Sintesi dei Contenuti

- **Introduzione**

Nel presente elaborato si introducono e si definiscono gli aspetti principali della preparazione dei dati (*data preparation*) e si analizza il software *OpenRefine* per valutarne le potenzialità come strumento per effettuare la preparazione dei dati.

- **Capitolo 1: La preparazione dei dati**

Innanzitutto, è opportuno definire le motivazioni che rendono necessaria la preparazione dei dati. Il numero di fonti che producono dati sta crescendo velocemente, così come la quantità di applicazioni che hanno bisogno di dati pronti per essere utilizzati. Questi dati pronti all'uso sono indicati come *prepared data* (dati preparati) e si presentano in una forma organizzata, differenziandosi per questa ragione dai dati prodotti dalle fonti, chiamati invece *raw data* (dati grezzi).

La trasformazione che consente di ottenere dati preparati a partire dai dati grezzi è composta da molteplici fasi operative che si susseguono in vario ordine, anche ripetutamente. La più importante di queste fasi è la preparazione dei dati.

La necessità di effettuare la *data preparation* non è un problema recente; bisogna però notare che in passato questa operazione era maggiormente complessa, in quanto non esistevano software dedicati. Coloro che avevano bisogno di *prepared data* dovevano avvalersi dell'aiuto di colleghi esperti programmatori per ottenerli a partire dai *raw data*. Di conseguenza, il processo di lavoro era lungo, complicato e spesso non produceva i risultati attesi. Questo perché chi aveva bisogno dei dati doveva riuscire a spiegare al programmatore cosa questi descrivessero e cosa volesse ottenere; il programmatore doveva capire quanto spiegato e trovare il modo di generare il risultato richiesto, individuando una serie di operazioni da compiere sui dati di partenza per ottenerlo; se non corretto, il risultato ottenuto doveva essere scartato, ripartendo da

zero. Da questa problematica nascono i software per la preparazione dei dati, che sono rivolti direttamente a chi deve utilizzare quei dati. Nel lungo elenco dei software utilizzabili per la *data preparation* si trova anche *OpenRefine*, lo strumento analizzato nel presente elaborato.

La preparazione dei dati può essere a sua volta suddivisa in moltissime piccole operazioni, le quali sono svolte dai cosiddetti *preparators* (preparatori). Questi preparatori possono essere raggruppati in sei famiglie/categorie, che sono:

- "Data discovery" (esplorazione): ne fanno parte tutti i preparatori che permettono di navigare tra i dati per poter scoprire cosa contengono e cosa ci sia bisogno di fare per pulirli.
- "Data validation" (validazione): ne fanno parte tutti i preparatori che permettono di controllare la validità dei dati, in particolare la correttezza e la completezza.
- "Data structuring" (strutturazione): ne fanno parte tutti i preparatori che permettono di effettuare trasformazioni sui dati in modo da riorganizzarli in maniera consistente in una tabella.
- "Data enrichment" (arricchimento): ne fanno parte tutti i preparatori che permettono di aggiungere informazioni all'insieme iniziale dei dati, permettendoci di avere dei dati più completi e significativi.
- "Data filtering" (filtraggio): ne fanno parte tutti i preparatori che permettono di creare dei sottoinsiemi dei dati, seguendo criteri definiti.
- "Data cleaning" (pulizia): ne fanno parte tutti i preparatori che permettono di modificare i dati conoscendone il significato.

Prima di procedere è necessario chiarire che nonostante la pulizia dei dati (*data cleaning*) si possa considerare come una fase separata dalla preparazione dei dati, nelle applicazioni reali queste due fasi di fatto si fondono. In particolare, la pulizia opera sui dati a livello semantico, mentre la preparazione si concentra sulla loro struttura.

Un'altra importante distinzione deve essere fatta tra *missing values*, *unknown values* e *zero values*. I valori *missing* definisco l'assenza di un valore e sono anche indicati con NA o NULL. I valori *unknown* indicano che il dato è presente ma il valore è sconosciuto. Infine, i valori *zero* indicano che il dato è presente con valore zero.

I dati vengono migliorati un po' alla volta e assumono le seguenti forme:

- Dati grezzi (*raw*): sono i dati originali, così come prodotti dalle fonti, impossibili da usare.
- Dati tecnicamente corretti (*technically-correct*): ad ogni valore è assegnato il corretto tipo.
- Dati consistenti (*consistent*): tutti gli errori sono stati corretti.
- Dati puliti (*tidy*): i dati sono organizzati secondo una struttura standardizzata.
- Dati aggregati (*aggregated*): spesso è utile aggregare i dati in modo da produrne un riassunto, riducendone il livello di dettaglio.
- Dati formattati (*formatted*): è la forma finale dei dati, pronti per essere utilizzati.

Il processo di preparazione dei dati offre notevoli benefici oltre ovviamente a quello di produrre dati utilizzabili per essere analizzati. In primo luogo, eventuali errori sono rilevati subito, impedendo che si propaghino alle fasi successive. Inoltre, creando una struttura ordinata riusciamo ad evidenziare il significato anche di un insieme di dati che nella sua forma grezza sembrava senza senso. Infine, la struttura organizzata ci permette ovviamente di gestire meglio i dati.

La preparazione dei dati è un'operazione che può richiedere molto tempo, quindi l'utilizzo di un software in grado di semplificare e velocizzare questo compito permette di ottimizzare notevolmente il processo da svolgere per ottenere il risultato desiderato. Questo aumento di velocità all'interno di un'azienda comporta inoltre un risparmio di denaro.

Bisogna comunque considerare che questa operazione richiede ancora uno sforzo manuale e i software disponibili accettano in input insiemi di dati che siano già stati

un minimo riorganizzati. Pertanto, una delle sfide future sarà quella di automatizzare al meglio anche questo compito.

- **Capitolo 2: OpenRefine**

  Si può ora introdurre il software che sarà oggetto di analisi: *OpenRefine*.

  OpenRefine è un software che nasce nel 2010 come *Freebase Gridworks*; Metaweb, l'azienda che lo ha prodotto, viene acquisita da Google nello stesso anno (*Google Refine*), ma poco dopo viene interrotto il supporto al progetto e questo è trasferito alla comunità open-source, prendendo il nome che ha attualmente.

  Il software è una applicazione desktop, compatibile con Linux, Mac e Windows, che gira localmente come web server e a cui si può accedere utilizzando la maggior parte dei browser. Le caratteristiche più importanti sono:

- Estensioni: forniscono funzionalità aggiuntive.
- Grande varietà di formati supportati, sia per importare che per esportare i dati. Un elenco non esaustivo comprende: CSV, TSV, file di testo, JSON, fogli di calcolo. Inoltre, è anche possibile importare da ed esportare verso un database sfruttando le query SQL.
- Privacy: girando localmente, tutti i dati rimangono sul nostro pc e per tanto non viene condiviso alcun dato personale. Bisogna comunque segnalare che esiste un piccolo set di funzionalità (che potremmo considerare opzionale nella maggior parte dei casi) che prevedono la comunicazione con fonti esterne.
- Installazione: semplice, veloce e simile per tutti i sistemi operativi.
- RAM: per funzionare in modo ottimale richiede di avere RAM riservata a disposizione; per alcuni utenti questo aspetto potrebbe essere problematico.
- Indirizzo di accesso al web server: *http://127.0.0.1:3333/* , che può essere modificato.

  Diretta conseguenza è la possibilità di farlo girare su una macchina a cui accederemo in remoto: questo potrebbe essere vantaggioso per creare un progetto

in comune su cui lavorano più persone. Questa soluzione prevede delle limitazioni in quanto non è prevista nell'uso originale del software.

- *Rows* e *records*: sono due termini fondamentali per comprendere il funzionamento di OpenRefine.

  Una row (riga) è una singola linea del progetto, che è composta da più celle, una per colonna.

  Un record è invece l'insieme di una o più righe, che condividono lo stesso valore nella prima colonna (identificatore).
- Filtri e *facets*: sono due elementi fondamentali per lo svolgimento di molte operazioni. In sostanza permettono di filtrare le righe visualizzate (filtri) o di creare sottoinsiemi (facets) che possono essere esplorati indipendentemente.
- *Expressions*: le azioni che trasformano i dati sono descritte da espressioni (da non confondere con le formule dei fogli di calcolo che sono memorizzate nelle celle e mostrano dinamicamente il contenuto). La maggior parte delle azioni è eseguibile utilizzando le funzionalità presenti nei vari menu, ma nei casi in cui si vogliano eseguire operazioni particolari è necessario saper utilizzare le espressioni.

  Le espressioni possono essere scritte in uno dei linguaggi supportati: GREL (General Refine Expression Language), che è il linguaggio di default, Jython e Clojure.

➢ **Capitolo 3: Valutazione sperimentale**

La valutazione da me condotta si basa sui risultati presentati nel *survey paper* "Data Preparation: A survey of Commercial Tools", di Mazhar Hameed e Felix Naumann (Hasso Plattner Institute, University of Potsdam) [1]. Nella ricerca vengono individuati sette software per la preparazione dei dati e si va a valutare se ciascuno di essi sia in grado o meno di effettuare ognuna delle 40 operazioni (Figure 3.2.1: List of

Preparators) che i ricercatori considerano maggiormente significative in un processo di preparazione dei dati.

Pertanto, per questa tesi ho analizzato la documentazione dei sette software presentati in modo da capire al meglio in cosa consistessero i preparatori individuati. Successivamente sono andato a verificare se e come OpenRefine sia in grado di eseguire le stesse operazioni.

In questo capitolo sono quindi riportate le descrizioni dei preparatori e di come le operazioni indicate possano essere eseguite utilizzando OpenRefine.

- **Conclusioni**

Al termine della valutazione sperimentale ho prodotto una tabella riassuntiva (Figure 3.10.1: Preparator List - Results and Comparison) dei risultati, che ho poi anche reso graficamente (Figure 3.10.2: Preparators - Comparison Graph) in modo da poter confrontare meglio OpenRefine con gli altri sette software.

In particolare, dal mio utilizzo di questo strumento sono emersi alcuni aspetti critici o comunque migliorabili del prodotto; sono riportate di seguito le principali criticità rilevate:

- Mancato supporto della funzionalità di assegnamento di un tipo di dato ad una colonna.
- Mancanza di una funzione di completamento automatico delle espressioni (sul modello di quanto accade ad esempio nella maggior parte degli IDE), che ne renderebbe più veloce l'utilizzo.
- Mancanza di scorciatoie rapide per l'utilizzo di alcune funzionalità, che invece al momento richiedono l'esecuzione di una serie di passaggi.
- Necessità di riorganizzare ed unificare la documentazione, attualmente divisa tra più siti.

Tali criticità non costituiscono però un fattore limitante nell'utilizzo del software e possono essere sottoposte al giudizio della community, valutando l'opportunità di implementarle o meno al fine di migliorare il software.

In conclusione, prendendo in considerazione i risultati ottenuti e le caratteristiche del software descritte nel Capitolo 2, concludo che, al netto delle criticità precedentemente elencate, OpenRefine sia una valida opzione per effettuare il processo di preparazione dei dati.

Contents

Introduction ..... **I**

Chapter 1: Data Preparation **1**

1.1 Motivation 1
1.2 Evolution of the Approach 2
1.3 Steps and Tasks 2
1.4 Let's Clarify Some Important Concepts 3
1.5 Data Classification 5
1.6 Benefits 6
1.7 Challenges 6

Chapter 2: OpenRefine **9**

2.1 History 9
2.2 Documentation and Community 9
2.3 Structure 10
2.4 Extensions 10
2.5 Import and Export 10
2.6 Privacy 12
2.7 Some Statistics 13
2.8 The Basics 13
2.9 Running as a Server 24

Chapter 3: Experimental Evaluation **25**

3.1 Background 25
3.2 Approach 25
3.3 Datasets 27
3.4 Category I: Data Discovery 28
3.5 Category II: Data Validation 33
3.6 Category III: Data Structuring 38
3.7 Category IV: Data Enrichment 47
3.8 Category V: Data Filtering 55
3.9 Category VI: Data Cleaning 59
3.10 Results 69

Conclusion **73**

References **75**

Appendix A **77**

Environment 77

Introduction

Data preparation is the starting point in the complex process of obtaining structured and useful information from raw data; it is therefore essential to carry out this operation correctly and effectively.

The following thesis aims to present data preparation and to analyze the software OpenRefine to provide an evaluation of its capabilities as a data preparation tool.

The thesis consists of the following chapters:

- Chapter 1: provides a description of what "data preparation" is, what the key points are, and what the issues are.
- Chapter 2: introduces the main characteristics of OpenRefine.
- Chapter 3: presents the evaluation process performed on OpenRefine and its results.
- Conclusion: concludes the thesis.

Chapter 1: Data Preparation

1.1 Motivation

The amount of available data, coming from more and more different sources (like logs, sensors, government data, online transactions, scientific research, citizen science, etc.), grows every day. In particular, this growth has enormously exploded in recent years; therefore, there is a huge amount of data that can and/or needs to be used. At the same time, there is a growing environment of applications that need data, and not simply *raw* data, but *prepared* data.

Different sources generate raw data, which is stored in so-called data lakes. The storing process does not provide any standardization or control; therefore, there will be duplicates, inconsistences, different encodings, missing or unwanted elements, wrong data types and so on. Raw data is, essentially, messy and unstructured.

It is impossible to use raw data, since any type of analysis would fail; therefore, a preliminary processing is needed in order to get it into a usable state, that consists of a structured, organized, and cleaned form.

This transformation is not a single-step process, but requires different steps, that can happen in various order, even iteratively. These steps typically are data exploration, data collection, data profiling, data preparation, data integration, and data cleaning. It has been found that data preparation is the most important one: studies [2] report that data preparation is considered critical by 72% of respondents, 88% say that it is at least important, and only 4% indicate that it is not important. Furthermore, data scientists and data analysts report that 80% of their time is spent doing data preparation, rather than analysis [3].

Data preparation is certainly the most critical step to make raw data usable in data analysis processes or by data-driven applications; it is therefore essential to carry out this operation correctly and effectively.

1.2 Evolution of the Approach

Originally, in addition to the complexity of the operation, there was a further complexity due to the fact that data preparation was performed using traditional programming approaches. In fact, capable analysts and people who know the data best might not be skilled enough in other fields: even if they have coding skills, they may lack the level of coding expertise needed to prepare data. Therefore, often the people who needed to perform data preparation had to ask for help to more experienced programmers; this required to be able to explain in a clear way what was in the data, and what needed to be done. This separation of roles made data preparation a long task that most of the time produced unsatisfactory results. Hence the need to have self-service, end-user-dedicated tools, enabling everyone to prepare data by himself. Over the years many tools have been developed, even for specific applications, to meet this request, and nowadays there are many options to choose from.

A long list of tools for data preparation can be found in the survey paper "Data Preparation: A Survey of Commercial Tools" [1], by Mazhar Hameed and Felix Naumann, which also reports the analysis of the following ones: Altair Monarch Data Preparation, Paxata Self Service Data Preparation, SAP Agile Data Preparation, SAS Data Preparation, Tableau Prep, Talend Data Preparation, Trifacta Wrangler. As stated in its title, the goal of this thesis is to analyze and evaluate OpenRefine, another widely used tool for data preparation.

1.3 Steps and Tasks

Data preparation is the combination of many individual steps, that are carried out by *preparators*. These preparators are grouped in six macro-categories, based on the performed task:

- Data discovery: first, we need to understand the data, what it contains, and what has to be done to make it usable. We need to answer questions about the structure, the presence of any metadata, the presence of missing values or

outliers, etc. Therefore, we have to navigate the dataset to answer these questions, and preparators in this category do that.

- Data validation: once that the content of the dataset is clear, we need to inspect the data to check for correctness, completeness, etc. This inspection is done according to a defined set of rules and constraints.
- Data structuring: we transform the data producing a structured and consistent view, that is organized in a table. Well-structured tables are essential for any analytics software. In addition, the user has a clearer view of the data and can understand them better. Structuring transformations include splitting column, changing column data type, and pivoting.
- Data enrichment: additional information needs to be added and connected to the original dataset, in order to get a much more complete view and context.
- Data filtering: it is an important task that simplifies inspection, producing a subset of the data based on some constraints. Using subsets, we can remove irregular rows or values.
- Data cleaning: it is the most complex part of data preparation. Analyzing the data semantic means that knowing the content of the dataset we can improve its consistency by removing errors, filling missing information, standardizing date format, etc.

However, this is not a fixed subdivision; in fact, some steps may belong to multiple categories. More details on preparators will be introduced in Chapter 3: .

1.4 Let's Clarify Some Important Concepts

➢ Data Preparation vs Data Cleaning

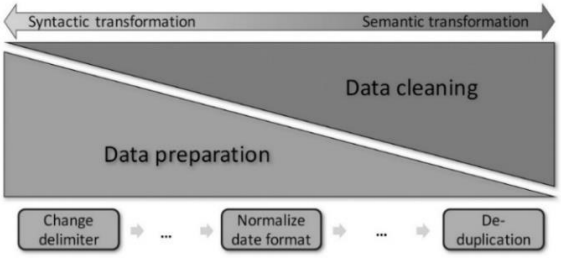


Figure 1.4.1: Data Preparation vs Data Cleaning

First, we described data cleaning as a separated step from data preparation; then, it is stated that it is a part of data preparation. As shown in Figure 1.4.1, in real applications data cleaning is strongly connected with data preparation and they are melted together. Data cleaning approaches given data from a different point of view. It analyzes the semantic, thus it is focused on the values of the dataset. Data preparation instead operates with transformations that alter the structure of the dataset.

- **Missing Values vs Unknown Values vs Zero Values**

Missing values are the most common and basic inconsistencies that could be present in a dataset. They are known as NAs or NULLs, stating the absence of a value, i.e., the information is not available.

On the contrary, an unknown value indicates that the information is present, but it is unknown.

Lastly, a zero value indicates that the information is present, and its value is zero.

Since those three concepts are frequently confused, the following example is provided to better explain the difference (Figure 1.4.2).

To subscribe to a photography contest, it is required to compile a form.
The following optional question is present:

"How many photos have you ever shot?"

(a) 0
(b) Less than 100
(c) Between 100 and 1000
(d) More than 1000
(e) Don't know

Each answer produces a different outcome:

Answer	Value stored
User doesn't pick any option	NA or NULL
a	0 (zero value)
e	String "unknown"
b, c, d	String from the selected option

Figure 1.4.2: Null, Zero, and Unknown Values

Dealing with missing values is a difficult task; depending on the case, we may decide to ignore records that contain them or to use imputation techniques. These techniques correct missing values by estimating them based on other information.

1.5 Data Classification

Since data preparation is not a single-step process, there is no direct transformation from raw data to structured data. Instead, there is a set of individual steps (covered in section 1.3) that improve the data quality at each step.

The following categories can be defined: raw data, technically-correct data, consistent data, tidy data, aggregated or compressed data, and formatted data. [4]

- Raw data: refers to the data as it is, containing all the errors and inconsistencies. It is impossible to use that data in any kind of application or analysis.
- Technically-correct data: each variable is stored consistently using the appropriate data type, according to the real-world domain. There are still errors or incomplete values.
- Consistent data: errors are fixed, and unknown values imputed.
- Tidy data [5]: standardized structure in which each variable forms a column, each observation forms a row, and each type of observational unit forms a table. The meaning of the dataset is linked to its structure in a standard way.
- Aggregated or compressed data: the granularity of the dataset is reduced. It is a common practice in data analysis to substitute data with a summary. Granularity is the level of detail, typically temporal, at which data is stored. For example, an industrial mixer produces every second a record containing the data about the raw materials flows, output flow, rotating speed, etc. This level of detail is not necessary in most analysis; therefore, we can compress these data replacing 3600 records (1 record/s * 60 s * 60 min = 3600 record/h) with only one that provides the hourly average.
- Formatted data: the final data version that is obtained and that is ready to be used.

1.6 Benefits

Data preparation is necessary even for apparently correct data, because it can detect unexpected errors that would produce bad analysis and consequently bad decisions.

That said, the benefits of data preparation are:

- Early error detection: errors that were present in raw data are immediately discovered and dealt with, avoiding error propagation to data processing.
- Any data becomes usable: that is a game changer, since data can be retrieved or extracted from any source and used, therefore the available data enormously increases. For example, data presented in an HTML page can be parsed, prepared, and used even without direct access to the original source.
- Meaning is revealed: by structuring the data, relationships are clarified and highlighted, thus an apparently meaningless dataset can reveal a lot of information instead.
- Better decisions and analysis: prepared data can be analyzed better, producing clearer and faster results, leading to better decisions.
- Easier managing: well-structured data are easier to maintain, access, update, and of course to use.

1.7 Challenges

Data preparation is a time-consuming thus money-consuming task. Consequently, a lot of businesses struggle (lack of budget) to implement it, because the volume and the variety of data cannot be estimated correctly.

Nowadays on the market there are a lot of different data preparation tools that make this arduous and ungrateful task easier. The biggest improvement is that they provide real-time data visualization and transformations, making this field more accessible.

Obviously, as shown in analysis and surveys, data preparation tools are not perfect and still lack many features. First, data preparation remains a mostly manual

task, lacking some useful automatisms that would speed up data preparation process. Available data preparation tools require pre-preparation of the dataset, such as making it a single-table file, removing preambles and footnotes, defining homogeneous escape symbols, etc.

Another consideration concerns the lack of tools (or integrated features in the existent tools) to perform preparation of unstructured data, such as stop-word removal or sentence breaking in textual data (descriptions, plot synopses, etc.).

Chapter 2: OpenRefine

2.1 History

OpenRefine is an open-source software for analyzing and enhancing data. Born in 2010 as Freebase Gridworks by Metaweb Technologies, soon later its producer was acquired by Google, and the project was renamed as Google Refine. In 2013 Google support stopped and the project was transferred to open-source community, finally becoming OpenRefine.

Google acquisition helped a lot both in development and marketing. However, during the Google management the product remained a standalone application, since Google did not integrate any of its services (e.g., Docs or Drive) because it was more interested in other technologies of the acquired business.

2.2 Documentation and Community

Before continuing, let's focus on the organization of the documentation about OpenRefine. At present, it is a bit scattered across multiple sources:

- Repository on GitHub (https://github.com/OpenRefine/OpenRefine) [6]: it is the core of the project. It stores the source code, and it is the main page where we can find information and documentation.
- Website (openrefine.org) [7]: at present, its main utility is to provide a user-friendly page to download the software and to show the available extensions and libraries. The other pages report just a small part of the information and links that are available on the GitHub page.
- User manual: the official user manual (docs.openrefine.org) [8] was released in January 2021, but it is still subject to modifications. Additional useful documentation can still be found on the GitHub page, which provides FAQs, a wiki, and solutions to common problems. At present, consulting the

documentation can be a tedious process, since the information is scattered across multiple sources (as explained above).

- Community: the community is very active on multiple platforms and anyone can contribute or ask for help; therefore, if facing any type of problem, it could be useful to also check its platforms in addition to the documentation. The community can be contacted on GitHub issues page (https://github.com/OpenRefine/OpenRefine/issues), StackOverflow (using the "openrefine" tag), Google discussion group (https://groups.google.com/g/openrefine and https://groups.google.com/g/openrefine-dev/, respectively for users and for developers), and Gitter chat (https://gitter.im/OpenRefine/OpenRefine).

2.3 Structure

OpenRefine is a standalone desktop application, that works with Linux, Mac, and Windows operating systems. Java is required, but luckily recent releases have already included it in order to make it a ready-to-use product. Once installed, it runs locally as a small web server, whose web user interface can be accessed using almost any browser, working best on browsers based on Webkit (Google Chrome, Chromium, Opera, Microsoft Edge). It has minor issues with Firefox, and Internet Explorer is not supported.

2.4 Extensions

Thanks to the OpenRefine community, extensions have been created to provide additional functionalities and shortcuts for common operations. These extensions are listed on a dedicated download page.

2.5 Import and Export

OpenRefine provides a wide range of formats and methodologies to import and export data, thus a valid solution can be found in almost any situation.

It should be noted that OpenRefine does not modify the original data, but it copies all the information in its own project file. This detail is extremely important because the user can work freely, without worrying about irremediably modifying or damaging the original data.

Now, let's look in more detail at the import and export options that OpenRefine offers.

- **Importing Data**

The supported formats are: CSV, TSV, text file, JSON, XML, OpenDocument or Excel spreadsheet, PC-Axis, MARC, RDF data, Wikitext, and even more formats are available thanks to extensions.

Data can be imported from different sources:

- Files on computer: in addition to the already mentioned formats, data can also be retrieved from archives (such as *.zip* or *.tar.gz*, not working with *.rar*).
- URLs.
- Any text pasted from clipboard.
- Databases: *.db* files or connections to database systems (PostgreSQL, MySQL, MariaDB, and SQLite are all supported). It retrieves data using SQL queries.
- Google Sheets: data can be in a linked Google Drive account or provided via a file with link-sharing turned on.

- **Exporting Data**

First of all, projects can be entirely (project and history) exported in a *.tar.gz* archive. This feature allows sharing projects between different machines or people.

Supported formats are: CSV, TSV, HTML-formatted table, OpenDocument and Excel spreadsheet, Google Sheets.

In addition, OpenRefine provides:

- Custom tabular exporter: it allows to choose which columns we want to export, how to order them and which separator must be used.
- SQL statement exporter: it allows to create a SQL table from the dataset, generating a *.sql* file containing the SQL statement for creation and insertion, that can be later executed on an existing database.
- Templating exporter: it is useful for recreating formats that are not supported or to produce a custom format. It generates a *.txt* file containing the data.

2.6 Privacy

Everything in OpenRefine stays locally, that means that projects, history, preferences, and data are never shared with anyone or duplicate anywhere.

There is no data encryption, but it can be enabled by setting up full disk encryption in the operating system.

Cookies are not stored, except for OAuth authentication when using Google Drive.

In essence, the user is the sole owner of everything.

Just a small set of operations involve external communications and therefore transmit some information to the other services. These operations are:

- Using a reconciliation service.
- Using URLs (either to import a file or to add a column).
- Importing from or exporting to Google Sheets or Google Drive.
- Importing from or exporting to a SQL database (borderline case because the user can also own the database).
- Uploading data to Wikidata.

An important consideration is that this set of operations can be considered optional, therefore if the user does not want to use them, no fundamental functionality is lost.

These features are enough to state that your privacy is safe using OpenRefine, therefore it is a tool that can be used without any major concern.

2.7 Some Statistics

A 2020 survey [9] conducted on OpenRefine community provides some important information about the users of the software and how it has been used over the years.

First of all, it shows that OpenRefine is mostly used by librarians (36.64%), semantic web users (12.92%), and also researchers (10.7%).

An important note goes to the fact that the experienced users are growing, since 57% of the user base stated that it had more than two years of experience. This type of user is very important, given the open-source and community-supported nature of OpenRefine. Having experienced users means more support available and more requests of features/improvements.

Another important consideration concerns the tasks that users accomplish using OpenRefine. It is reported that data normalization (e.g., fixing spelling mistakes) leads the rankings, followed by data preparation for datasets that will be later loaded in another system (database, cloud application, etc.).

2.8 The Basics

2.8.1 Installing

The installation process of OpenRefine is very fast and pretty much identical on all operating systems, basically consisting of a single step. Then it is useful to set the workspace directory, a desired directory in which OpenRefine stores data.

2.8.2 RAM

An important thing to keep in mind is that OpenRefine relies on having computer memory (RAM) available to work effectively. This detail could be a limiting factor for some users. By default, OpenRefine is set to operate with 1GB. However, if needed, the reserved memory can be easily increased. It is suggested to do so when using large datasets. Indicators that more memory is needed are: more than one million total cells, input file size greater than 50MB, more than 50 rows per record, OpenRefine is running slowly, or "out of memory" errors.

2.8.3 Starting and Exiting

Launching OpenRefine opens a command line window and then uses the default browser to load the interface. Once OpenRefine is running, the home screen can be accessed at any time and from any compatible browser, simply going to: *http://127.0.0.1:3333/*. The default IP and port can be modified before launching the software (see section 2.9 to learn about an important feature made possible by this modification). OpenRefine runs as long as the command line window is kept open.

To exit, close all the browser tabs first and then close the command line window pressing the combination *Ctrl+C* (*Cmd+Q* on Mac) to properly close the window and save any last changes.

2.8.4 History Feature

OpenRefine provides a history feature that tracks any activity that changes the data, consequently it is possible to undo and redo these activities. Furthermore, these activities can be reused in other projects. To do so, in the "Undo/Redo" tab in the first project click on "Extract..." button, select the desired operations, copy the generated text, then paste it in the "Undo/Redo" tab in the second project after clicking on "Apply..." button.

2.8.5 Saving

In OpenRefine there is no "save button" because it autosaves every five minutes, of course this frequency can be changed. History is the only thing that is saved; therefore, facets and filters are lost. The only option to save facets and filters is to click on the "Permalink" button that provides a new URL at which the project is reloaded including these elements.

2.8.6 Rows vs Records

Defining these two terms is fundamental to better understand and use OpenRefine:

- Row: a single line of the project. A row is made up of a series of cells, one cell per column. It must be noted that a single cell could contain multiple pieces of

information. One of the task of data preparation is in fact to split these parts in single cells.

- Record: a combination of one or multiple rows that share the same first column, identifying a unique object. Therefore, each record is identified by the value of the first column, that is the primary key.

In OpenRefine a record is created based on blank cells in the first column of the project. Consecutive rows with a blank cell in the first column are part of the same record, which is identified by the previous non-blank cell in the first column.

In OpenRefine, records are displayed using grey and white background alternately. Also, the third sub-column of the column "All" counts the number of records.

In the following example, each athlete has a unique identifier "ID", that works as a primary key. Knowing that, I applied some transformation to create records. In particular, by applying the definition of "record" I proceeded to blank down (keep the first unique value and erase the rest) the "ID" column. At the end of the process, we can switch to "records mode" and we will see the rows grouped in records.

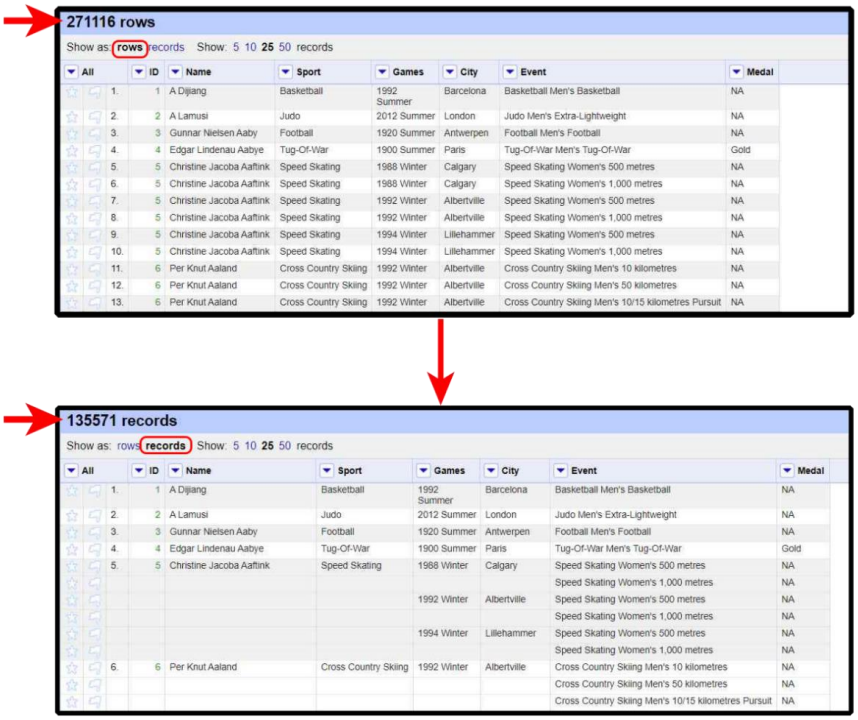


Figure 2.8.1: Rows vs Records

2.8.7 Facets and Filters

Facets and filters allow to display patterns and trends, allowing the user to better explore and modify subsets of the data. This is a very useful feature in data preparation, also considering that multiple filter and facets can be applied.

A filter is just a way to reduce the displayed rows based on whether a column includes a text string or pattern.

Facets create subsets that can be explored individually or combined. This possibility creates a more refined and powerful way to explore data. OpenRefine has several default facets, but offers the possibility of creating more specific custom facets. Once the facets are defined the subsets are created, and here comes the power of facets because we just select the subsets we want to display, without having to execute any additional expression.

Facets will be used in many sections of Chapter 3: Experimental Evaluation; therefore, we provide here an overview of the default facets, grouping them in categories:

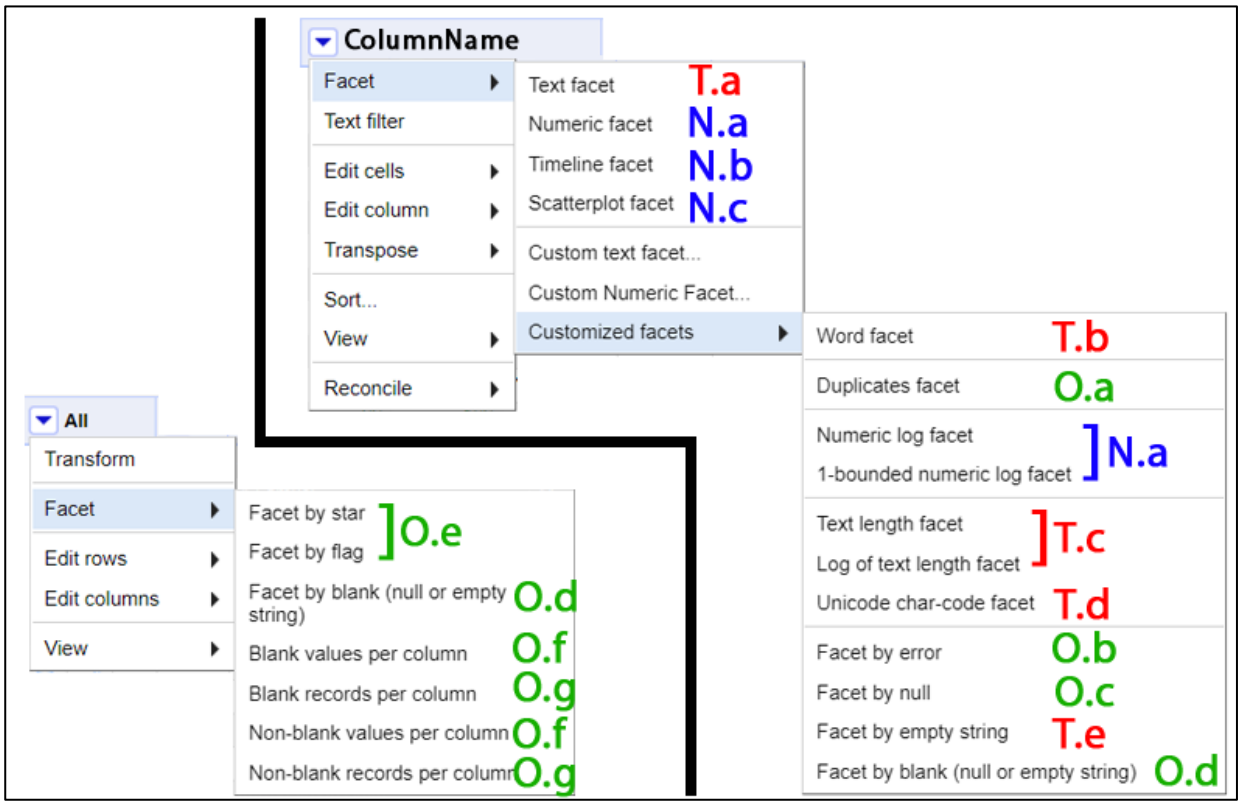


Figure 2.8.2: Default Facets in the Menus

T. Facets based on a textual content

a. “Text facet”

It takes the total content of the cells of the selected column and groups the exact matches.

b. “Word facet”

It splits up the content of the cells based on spaces, then acts like a “Text facet”. This facet can be useful to look at words used in cells containing long texts.

c. “Text length facet” and “Log of text length facet”

It calculates the length (that can also be expressed in logarithmic scale) for each cell and plots the resulting values using a numeric facet chart.

d. “Unicode char-code facet”

It identifies the Unicode value of each character contained in the cells of the selected column and plots these values using a numeric facet chart.

e. “Facet by empty string”

It identifies the cells that contains only an empty string, producing the “true” and “false” groups, “true” being empty. This type of cells can be a left over from other operations.

N. Facets based on a numeric content

a. “Numeric facet”, “Numeric log facet” and “1-bounded numeric log facet”

It sorts numbers by their range, providing a histogram that allows to modify the minimum and maximum limits of the range that we want to display. The facet will provide options to include blank, non-numeric, and error values in the visualization, appearing as “0” values. There are also two versions that use a logarithmic scale to compact wide ranges.

b. “Timeline facet”

It acts like a numeric facet, but it works on cells formatted as the supported date data type, sorting them chronologically.

c. “Scatterplot facet”

It visually represents two related sets of numeric data. An example can be found in section 3.5.2.

O. Facets based on other criteria

a. “Duplicates facet”

It will display only rows that have non-unique values in the selected column. In detail it creates a “true” and a “false” group, “true” being cells that are not unique.

b. “Facet by error”

Using expression to transform data, we may have that for some values an “error” is generated. OpenRefine allows us to store that error; therefore, we can apply a facet to check whether errors were generated during a transformation.

c. “Facet by null”

It will display rows that contain a null value in the selected column. In detail, it produces the “true” and “false” groups, “true” being null.

d. “Facet by blank”

It is the combination of the “Facet by empty string” and the “Facet by null”. If this facet is applied from the dropdown menu of the column “All”, then it will display the rows that contain a blank value in any column.

e. “Facet by star” and “Facet by flag”

In OpenRefine we can mark specific rows for further analysis with two labels: a “star” or a “flag”. Therefore, these two facets allow to display rows that belong to these groups.

f. “Blank values per column” and “Non-blank values per column”

“Facet by blank” simply shows rows that contain or not at least one blank value. These two facets are a detailed version because they group the rows by the columns in which the blank/non-blank value are present.

g. “Blank records per column” and “Non-blank records per column”

   These two facets allow to check for each record whether it has a column containing only blank values, grouping together records with the same completely blank column.

Specific applications of the facets can be found in Chapter 3: Experimental Evaluation; anyway, a short example can be found in Figure 2.8.3 and Figure 2.8.4 to show that we can combine multiple facets to modify the final subset that is displayed. For the purposes of this example, the “athlete_events.csv” dataset has been modified in order to provide a simple list of athletes with some attributes (“Sex”, “Year of Birth”, “Height”, “Weight”, and “NOC”) and facets have been applied on the columns “Year of Birth” and “NOC”. We can easily combine the subsets in order to display the desired final subset. In this example I started displaying the athletes from Italy (NOC value “ITA”) born in the ‘90s (1990-2000), then I added the athletes from Iceland (NOC value “ISL”). Lastly, I applied a filter to show only the athletes which name starts with the letter “E”.

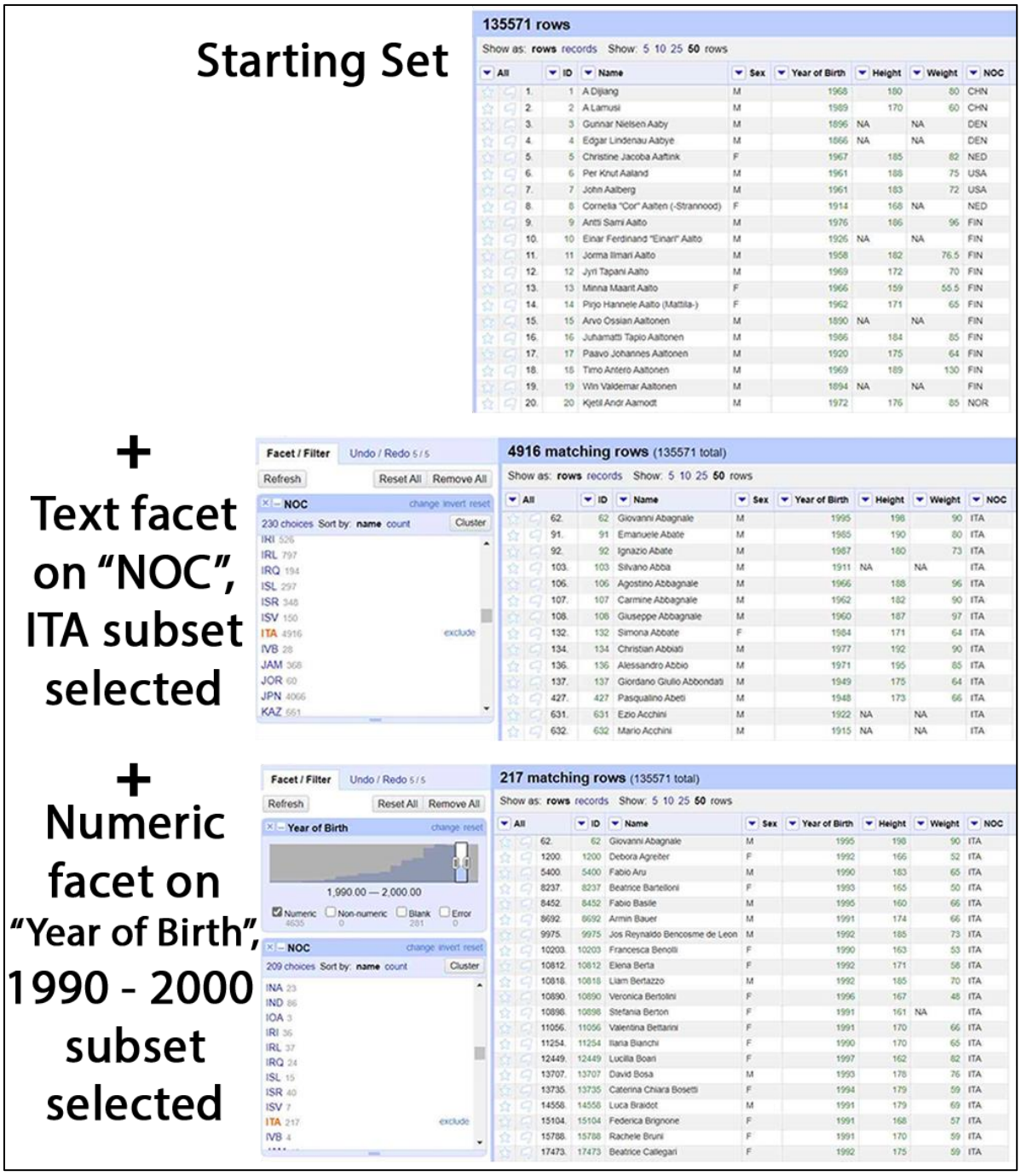


Figure 2.8.3: Example of Facets Applied on a Dataset - Part 1

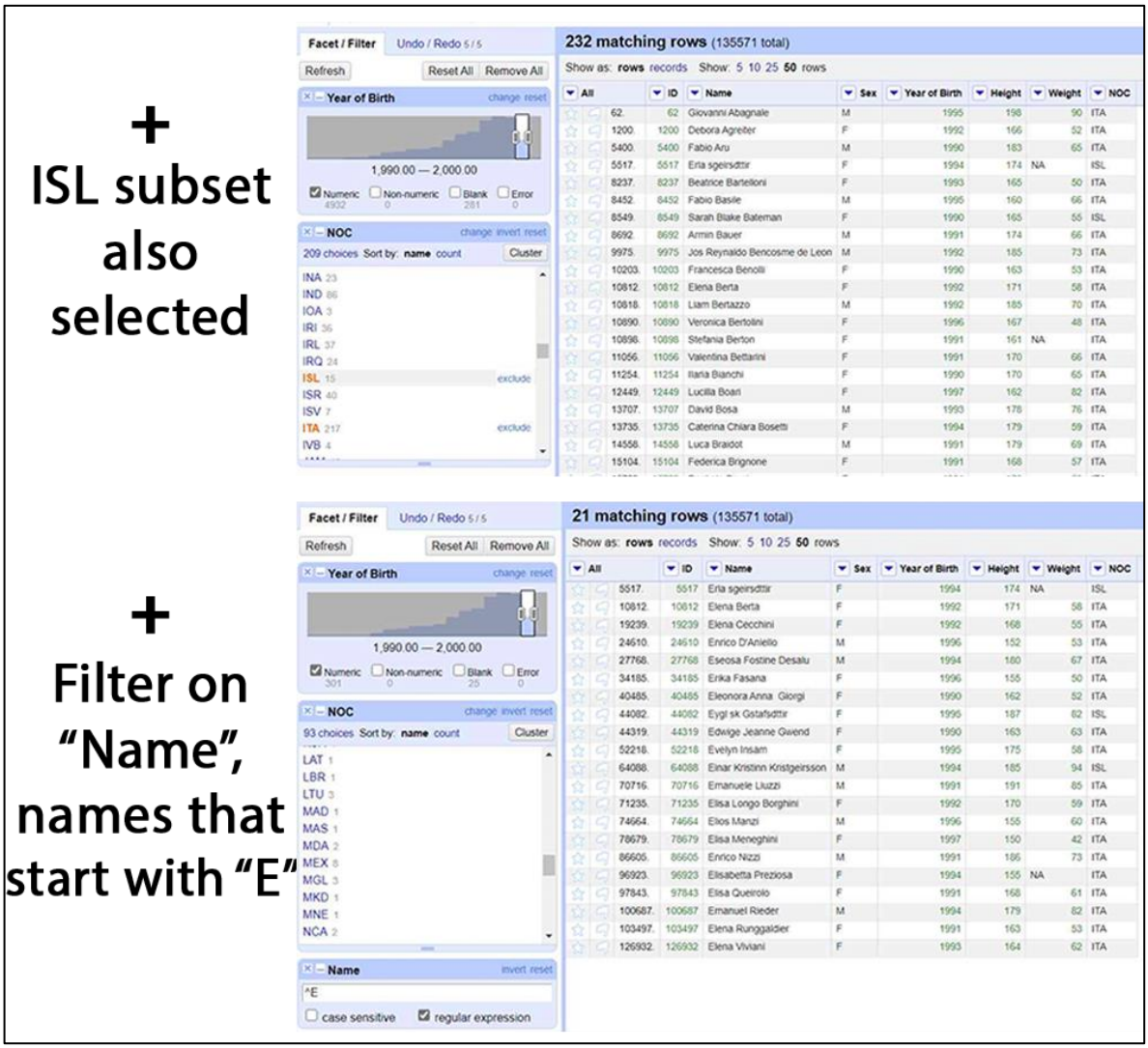


Figure 2.8.4: Example of Facets Applied on a Dataset - Part 2

2.8.8 Expressions and Supported Languages

Each action that transforms the data in OpenRefine is described by an expression. In detail, expressions are the description of one-time transformations that can change column contents or generate new columns. They must not be confused with spreadsheet formulas, that are stored in cells displaying the output dynamically.

All the basic and most common actions are directly accessible from various dropdown menu, without having to write anything. Instead, if we want to execute more complex operations, then we need to learn how to use expressions. Knowing how to use expressions is fundamental if we need to operate with a complex dataset that needs a lot of transformations.

Expressions can be used in the following operations:

- Facet: custom text facet, custom numeric facet, customized facets.
- Edit cells: transform, split multi-valued cells, join multi-valued cells.
- Edit column: split, join, add column based on this column, add column by fetching URLs.

The supported languages to write expressions are: GREL (General Refine Expression Language), Jython, and Clojure. Obviously, there are some syntax differences, but they support many of the same variables. GREL is the default (it was developed together with OpenRefine), but the last two are also offered with the installation package. It should be noted that extensions may provide additional languages.

Using an operation that accepts expressions will open a pop-up window containing the expression editor, Figure 2.8.5. It contains the following areas:

a. Expression text field: input the expression here.
b. Language selector.
c. Error console: it displays the errors present in the expression.
d. Additional tabs:
    - History: shows the expressions recently used across all the projects.
    - Starred: expressions from History that have been starred for reuse.
    - Help: a concise reference to GREL functions.
e. A list of the first values of the selected column.
f. A preview of the result obtained by applying the transformation.
g. Selector to decide what to do when an error is generated.

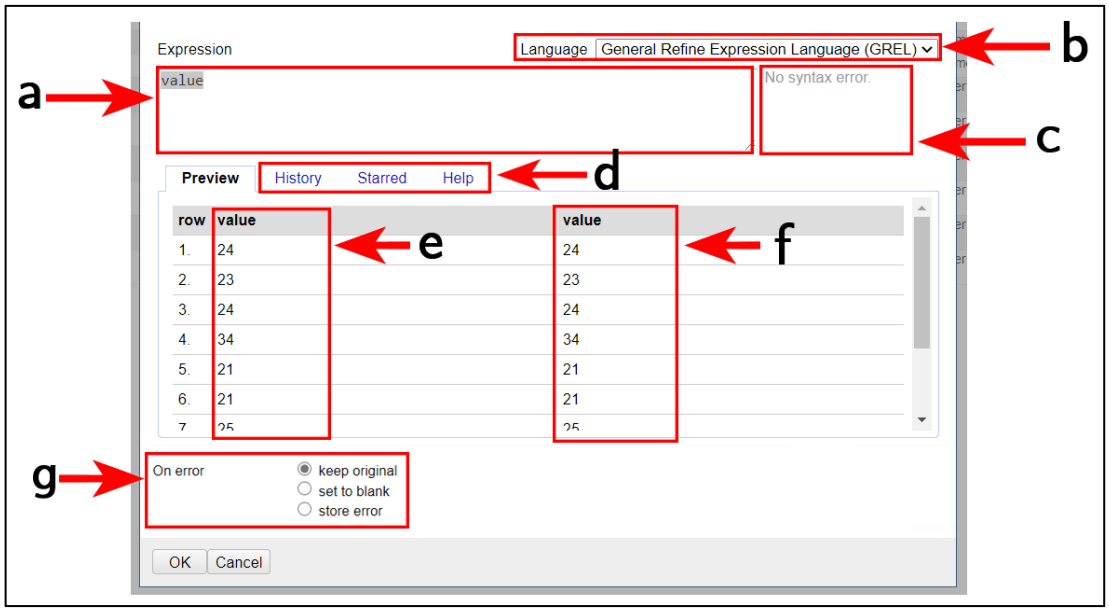


Figure 2.8.5: Expression Editor Pop-Up Window

A brief introduction to GREL must be made to conclude this section. GREL is designed to resemble JavaScript. The syntax is very easy to learn, and functions can be written in either one of the two following forms:

- `functionName(arg0, arg1, ...)` ← "Full Notation"
- `arg0.functionName(arg1, ...)` ← "Dot Notation"

The most used, easy to read and to understand is the "Dot Notation", in which the functions occur in the order of calling from left to right. This allows us to combine multiple functions in a readable order. Furthermore, the "Dot Notation" is also used to access the member fields of variables.

Variables make functions work, because they provide the access to the fields of the table that contains the data. The following list shows the variables available in OpenRefine:

- value: the value of the cell in the current column of the current row (equals to cell.value).
- row: the current row.
- cells: the cells of the current row (equals to row.cells).
- cell: the cell in the current column of the current row.

- rowIndex: the index value of the current row (starting from 0).
- columnName: the name of the current cell's column.

The complete list of functions and a more exhaustive description of the variables can be found in GREL manual.

2.9 Running as a Server

OpenRefine, by default and for security, is set to be accessible only from localhost (*127.0.0.1*) on port *3333*. Changing this default allows to make an OpenRefine instance accessible from remote. Therefore, this allows to run OpenRefine on a more powerful machine (more RAM) or to share that instance between more users. For example, OpenRefine could run on a university server and students/researchers can collaborate on the same projects.

However, it should be noted that there is no built-in security or version control for multi-user scenarios, therefore overwriting is a risk and must be self-managed by users. This small flaw could be fixed in future releases, if strongly desired by the community.

Chapter 3: Experimental Evaluation

3.1 Background

This thesis moves from the survey paper "Data Preparation: A Survey of Commercial Tools", written by Mazhar Hameed and Felix Naumann (Hasso Plattern Institute, University of Potsdam) [1]. This research, as the title says, proceeds to pick seven popular commercial tools for data preparation and to evaluate them by checking whether they have some features (specifically called preparators) or not. Preparators are identified by the researchers as the most useful features needed in a data preparation process.

3.2 Approach

First, it was necessary to provide an accurate description for each one of the preparators (Figure 3.2.1); therefore, I had to analyze the available documentation of the seven tools presented in the research. Then, I analyzed and tested OpenRefine in order to verify whether the functions described by the preparators can be executed in OpenRefine, with the ultimate aim of verifying whether it can be a valid tool for data preparation.

Categories	ID	Preparators
Data Discovery	I.a	Locate missing values (nulls)
	I.b	Locate outliers
	I.c	Search by pattern
	I.d	Sort data
Data Validation	II.a	Compare values
	II.b	Check data range
	II.c	Check permitted characters
	II.d	Check column uniqueness
	II.e	Find type-mismatched data
	II.f	Find data-mismatched datatypes
Data Structuring	III.a	Change column data type
	III.b	Delete column
	III.c	Detect & change encoding
	III.d	Pivot / unpivot
	III.e	Rename column
	III.f	Split column
	III.g	Transform by example
Data Enrichment	IV.a	Assign semantic data type
	IV.b	Calculate column using expressions
	IV.c	Discover & merge external data
	IV.d	Duplicate column
	IV.e	Generate primary key column
	IV.f	Join & union
	IV.g	Merge columns
	IV.h	Normalize numeric values
Data Filtering	V.a	Delete/keep filtered rows
	V.b	Delete empty and invalid rows
	V.c	Extract value parts
	V.d	Filter with regular expressions
Data Cleaning	VI.a	Change date & time format
	VI.b	Change letter case
	VI.c	Change number format
	VI.d	Deduplicate data
	VI.e	Delete by pattern
	VI.f	Edit & replace cell data
	VI.g	Fill empty cells
	VI.h	Remove extra whitespace
	VI.i	Remove diacritics
	VI.j	Standardize strings by pattern
	VI.k	Standardize values in clusters

Figure 3.2.1: List of Preparators

3.3 Datasets

The researchers, mentioned in section 3.1, have identified three public datasets to test each software. The chosen datasets have the characteristic of containing data organized in different ways, therefore providing various scenario that allows the testing of all preparators. I have tested OpenRefine on the same three datasets.

Follows the list of these datasets accompanied by a brief description:

- Kaggle – 120 years of Olympic history: athletes and results [10]

  It is a dataset about the modern Olympic Games, including Games from 1896 to 2016. It contains two files: "*athlete_events.csv*" (271116 rows and 15 columns) and "*noc_regions.csv*" (230 rows and 3 columns).

  In the first one, each row corresponds to an individual athlete competing in an individual Olympic event; athletes are uniquely identified by the ID column.

  The second one matches the NOC (National Olympic Committee) to the respective countries.

- IMDb – Data about movies [11]

  It is a vast archive containing a lot of different data about movies. Most of this data is completely disorganized and pretty much useless in that condition. Therefore, it is a perfect example of data that needs data preparation.

- UK government web archive – Research on how much time employees spend on training besides their 9-5 job [12]

  It is just one file "*11-708-data-nlss-2009.csv*" with no coherent structure. It is a stack of almost 1000 tables interleaved by preambles/comments, and even the actual tables contain inconsistencies. A lot of work is required to extract valuable data from this dataset.

I used these datasets to test the features of OpenRefine.

3.4 Category I: Data Discovery

3.4.1 [I.a] Locate missing values (nulls)

In order to decide what to do with missing values, we must be able to locate them first. Different datasets can store missing values with different representations (for example, using a particular string like "NA").

If missing values are stored as proper null values, then OpenRefine has a default facet for locating nulls (see Figure 3.4.1).

1.1 Click on the column dropdown menu arrow.

1.2 Choose "Facet" then "Customized facets" and click on "Facet by null".

Instead, for any other representation of missing values we need to create a customized text facet (see Figure 3.4.1).

2.1 Click on the column dropdown menu arrow.

2.2 Choose "Facet" and click on "Custom text facet...".

2.3 A pop-up window will appear to provide the expression needed to identify the missing value representation.

The expression (where *<nullRep>* must be substituted with the string that represents null values in the dataset) is:

```
value == "<nullRep>"
```

Then click "Ok" to create the facet.

2.4 From the facet tab on the left, select "true" to display the rows that contain null value in the previously chosen column.

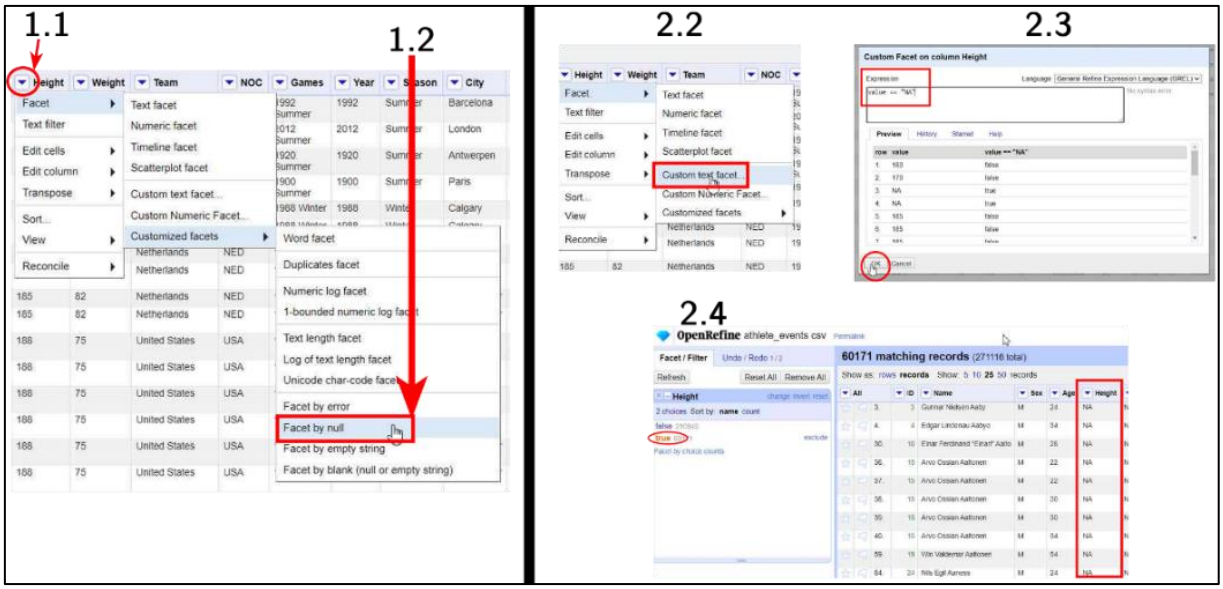


Figure 3.4.1: Steps to Locate Missing Values (nulls)

3.4.2 [I.b] Locate outliers

The ability to identify outlier patterns or values is an important feature because it allows us to spot errors and invalid data.

In OpenRefine this operation can be done using facets; there is one facet for each supported data type.

1. Data type: text → Facet: "Text length facet" (there is also the possibility of having the length expressed in log) or "Unicode char-code facet" (see section 3.6.3 for a description of usage possibilities).
2. Data type: numeric → Facet: "Numeric facet" (there is also the possibility of having a visualization in log scale).
3. Data type: date → Facet: "Timeline facet".

These facets produce a visualization of the values distribution; therefore, we can easily find strange values that need further analysis. Once the facet is applied, we can use the slider to select the area of the distribution that we want to display.

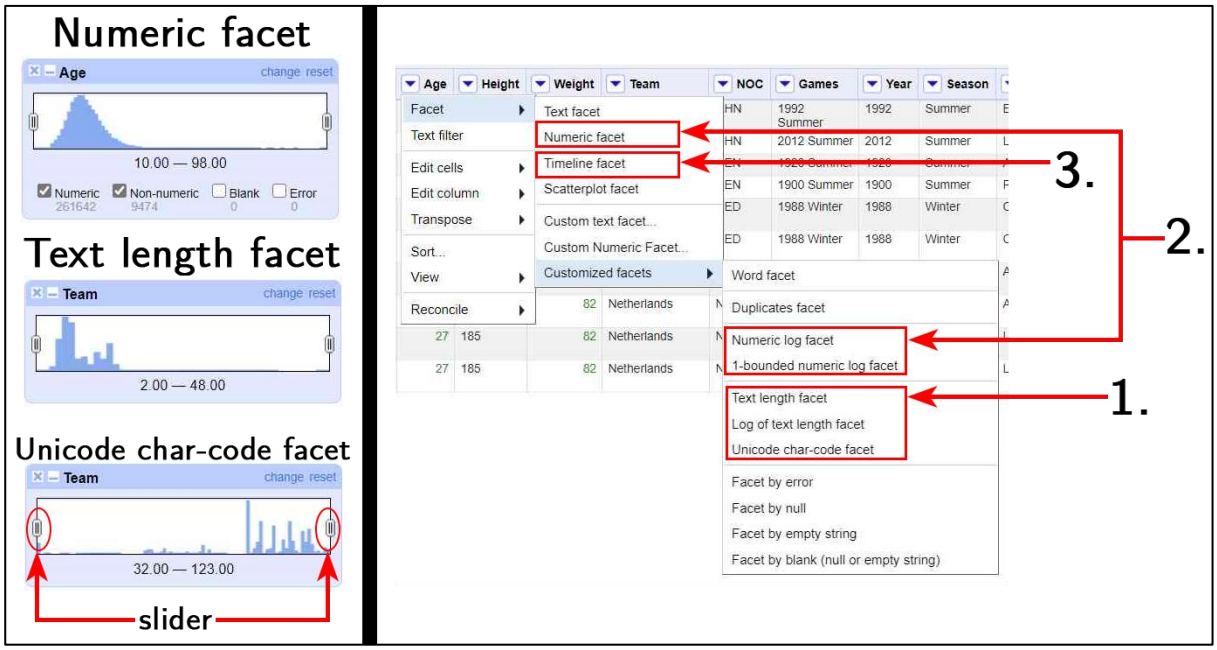


Figure 3.4.2: Locate Outliers

3.4.3 [I.c] Search by pattern

To accomplish this task, you need to use facets and filters combined with regular expressions. I have found that in OpenRefine this feature is merged with "3.8.4 [V.d] Filter with regular expressions".

3.4.4 [I.d] Sort data

Sorting data is a fundamental action to better understand the dataset. It allows us to organize the data. It is useful to highlight some characteristics (such as putting blanks first), and to navigate the dataset.

OpenRefine has this feature. In detail, the dataset can be sorted by each column, therefore the overall sorting is given by the combination of multiple sorting rules. The required steps are (see Figure 3.4.3):

1. Click on the column dropdown menu arrow.
2. Click on "Sort...".
3. A pop-up window will appear to select the criteria. In detail, select:
   - how to treat the values (as text, numbers, dates or booleans).
   - ascending or descending order.

- how to order groups of values (the groups are: valid values, errors, blanks).
- if a sorting rule is already applied then a checkbox will be present to choose whether to sort by that column only (deleting all the other sorting rules).

Then click on "Ok" to apply it.

4. The order of rows/records will change accordingly.

So far, it must be noted that only the viewing order has been modified. Therefore, to permanently change the order:

5. Click on the "Sort" label that appeared at the top of the table view.
6. Click on "Reorder rows permanently".
7. The operation will be executed. A message in the top-center will appear confirming it.

By clicking on the "Sort" label mentioned in step number 5, we can remove all the sorting rules ("Remove sort"), but also visualize and modify them.

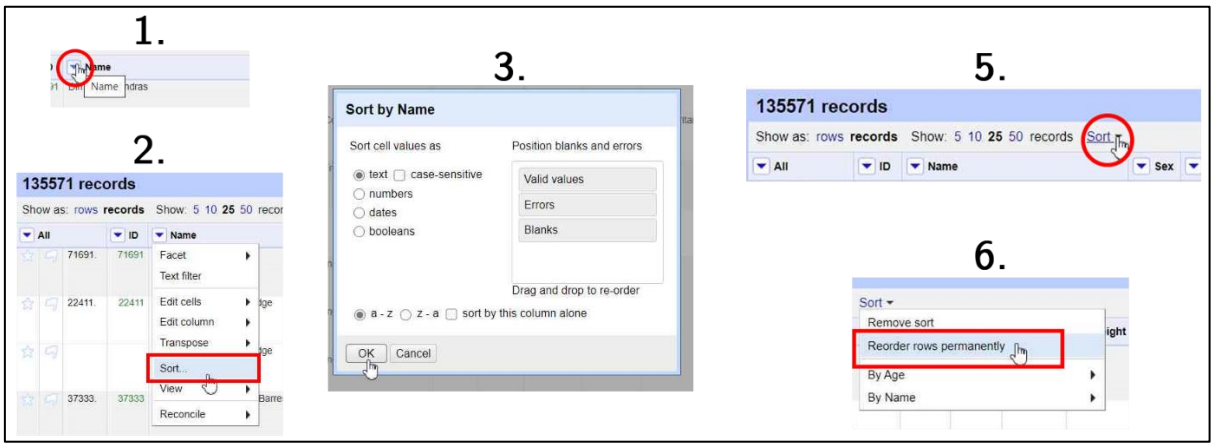


Figure 3.4.3: Steps to Sort Data

To provide an example, I applied it to the "athlete_events.csv" dataset, and I decided to sort it by "Age" and alphabetically by "Name".

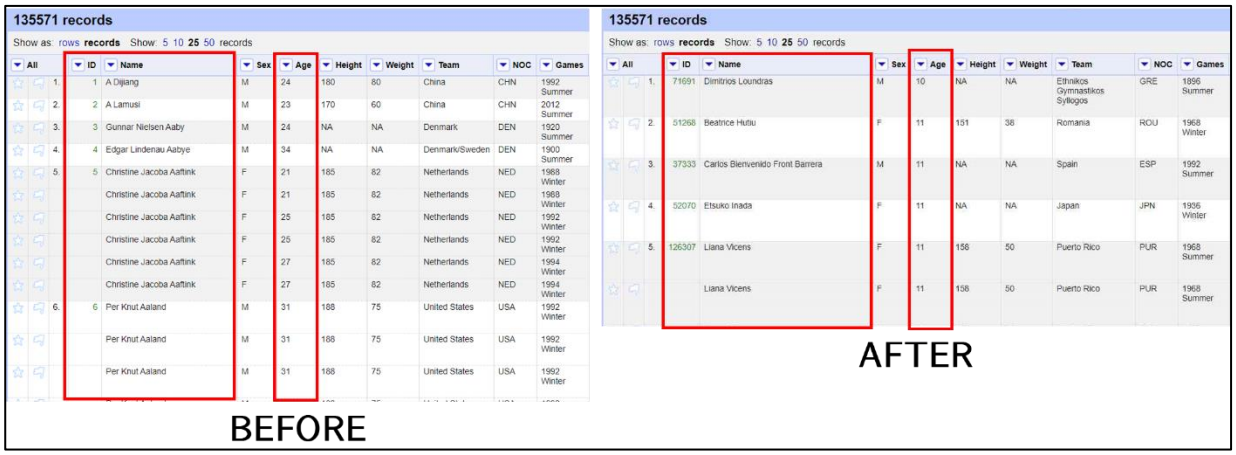


Figure 3.4.4: Sort Data - Before & After on a Dataset

3.5 Category II: Data Validation

3.5.1 [II.a] Compare values

Being able to compare values is a fundamental property in data validation, we cannot validate anything if we cannot compare with other values. Ideally, for each data type that is supported we need to be able to compare its values. What "compare" means exactly, depends on the data type:

- For numerical values the comparisons are of equality, majority and minority.
- For strings we may want to check whether they match or not, which one is longer/shorter, if a string is contained in another one, etc.
- For boolean values we can check whether they are equal or not.
- For dates we need to check whether they are the same; otherwise, we can determinate which one comes first/after, but also how many days/months/years/... have elapsed between the two.

The following list provides a sum up of how OpenRefine implements these features:

- Comparison Operators: ==, !=, <, >, <=, >= (equal to, not equal to, less than, greater than, less or equal, greater or equal).
- Numbers: we can use any of comparison operator.
- Boolean: we can use only "equal to" and "not equal to" operators.
- Date: to compare dates we need to use the function "diff" that returns the difference in a given time unit. Once we have this value, we can define the temporal relationship (0 means they are equal, a negative value means that the first date precedes the second one, a positive value means the first date comes after the second one).
- Strings: we can use the "equal to" operator to check if two strings match. Besides, there are a lot of various functions with self-explanatory names, such as: "startsWith", "contains", "find", "substring", etc.

Furthermore, it provides control structures for branching and looping ("if/else" and "for"), increasing the comparisons we can make.

3.5.2 [II.b] Check data range

Checking the range of values that a variable assumes in the dataset is an important task because it allows to check the validity and the consistency of the dataset. Furthermore, for numeric values and text length, it allows us to understand the minimum and maximum values that a variable assumes; therefore, we can decide how it needs to be stored when the dataset is exported in other formats. Lastly, we can check if the scatterplot of two related sets of numeric data complies with the previsions.

In OpenRefine we can perform this operation using the facets illustrated in section 3.4.2. Furthermore, there is the "Scatterplot facet" that can be recalled from any column dropdown menu, then a pop-up window will appear allowing to choose the pair of numeric values that we want to compare. Once selected, it produces a facet tab (on the left) that can be explored similarly to the other facets. From this tab we can export the scatter plot visualization to a PNG file by clicking on "export plot".

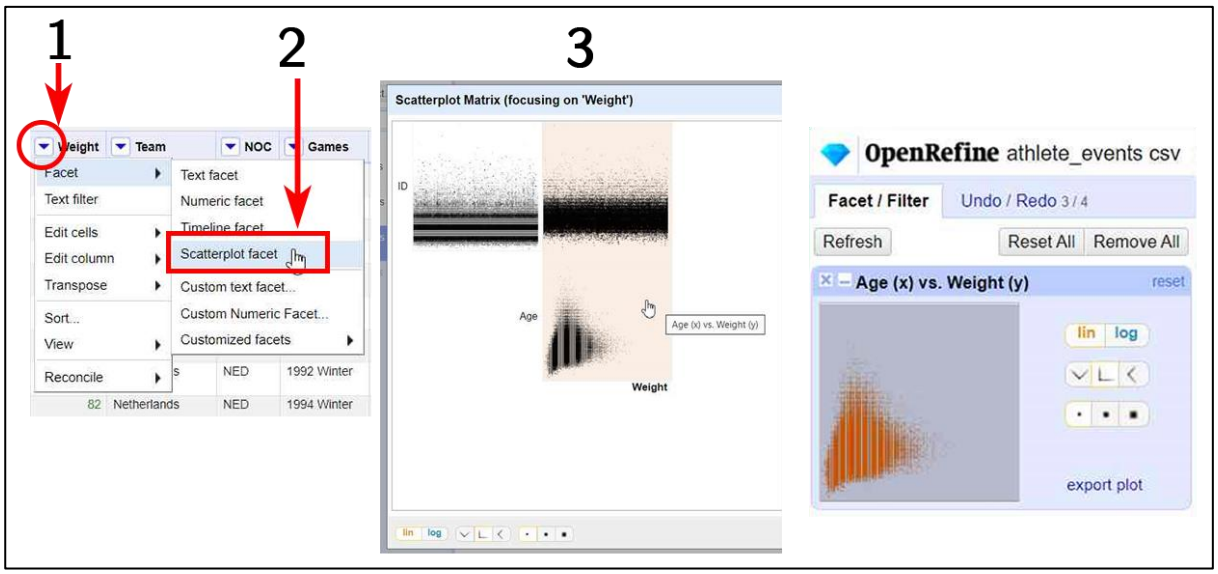


Figure 3.5.1: Check Data Range - Scatterplot Facet

3.5.3 [II.c] Check permitted characters

Text values need to be checked in order to verify whether they are made up with only the characters from the permitted set. This is a valuable operation because we can correct/delete characters that are not allowed and that can cause subsequent encoding problems.

In OpenRefine this operation can be executed in two ways. The first one is to apply the "Unicode char-code" facet (see section 3.4.2) to the column that needs to be checked. This way we can visualize the distribution of the characters and explore it to check if there is any character not allowed.

Anyway, the above descripted method is not very practical and effective. Therefore, the second way is to create a customized true/false facet using the function "contains" that allows to check permitted characters, not only in one column but even in multiple columns.

To better explain this second method, I applied it to the "athlete_events.csv" dataset. I wanted to check whether the cities and the names contain only alphabetic characters (A-Z and a-z) and the space character. The expression produced a true value for values containing only the permitted values. The expression is:

```
not(or(cells['Name'].value.contains(/[^A-Za-z ]/),
cells['City'].value.contains(/[^A-Za-z ]/)))
```

This expression has been built using the function "contains" to check whether invalid characters are present in either one of the cells of the column "Name" or "City", then the "not" function is applied to obtain the logic "true = only permitted char" and "false = otherwise".

3.5.4 [II.d] Check column uniqueness

To check for column uniqueness means to check whether all the values in a column are different.

In OpenRefine we can do it using facets and comparing the "choices counter" (see Figure 3.5.2) with the total rows/records, if the two matches then we have that each value in the column is unique.

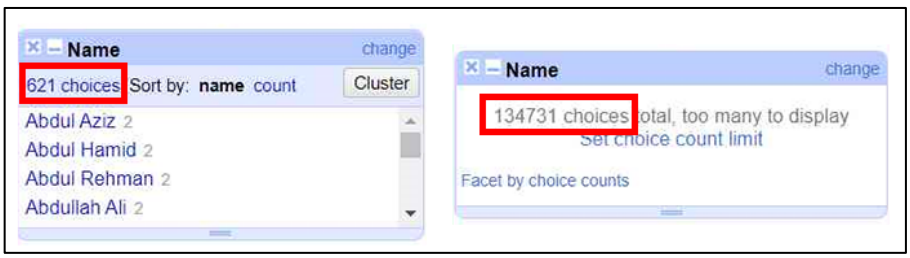


Figure 3.5.2: Choices Counter - Small Dataset (left) and Large Dataset (right)

To provide an example I have applied it to a revision of the "athlete_events.csv" dataset, in which I kept one row for each ID obtaining a list of the athletes. Applying the facet on the ID column it confirms that the IDs are unique (135571 choices = 135571 rows). Applying it on the "Name" column we can set that there are duplicated names (homonyms) because the choice counter has a value of 134731 and the rows are still 135571 (see Figure 3.5.3).

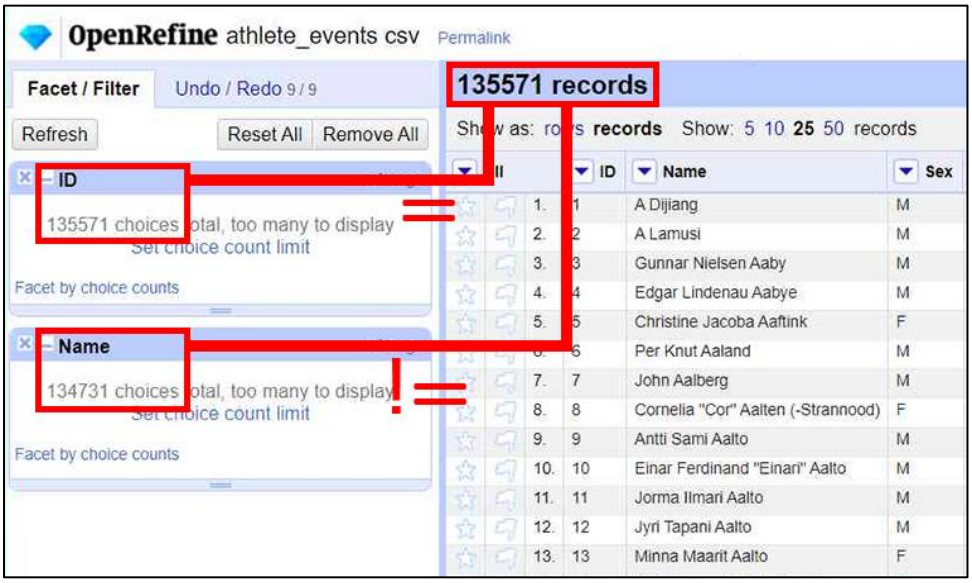


Figure 3.5.3: Check Column Uniqueness - Test on a Dataset

3.5.5 [II.e] Find type-mismatched data & [II.f] Find data-mismatched datatypes

These two features are discussed together because they are strictly connected. The first one refers to the ability to find data that is "out of context" compared to the known and assigned datatype, thus invalid data. The second one refers to the ability to find data which have assigned the wrong datatype. This second feature derives from the first one because a high number of type-mismatched data should trigger an alert about the correctness of the assigned datatype.

This description implies that the software allows to assign (automatically) a datatype to a column and to check the composition of its data against the datatype in order to evaluate the correctness of the assignment. Furthermore, it needs to support

a wide set of datatype and not just the basic ones (number, text, boolean, date) such as zip-codes, URLs, telephone numbers, ISBNs, etc.

To better explain these two concepts the following examples are provided.

- Type-mismatched data:

  Given a column with URL as datatype, the ability to find type-mismatched data will provide a set of all the values that are not recognized as a valid URL (i.e., "https://www.ingmo.unimore.it/site/home.html" is a valid URL, while "https://www.ingmo.unim o re .it/site/home.html" is not valid because it contains spaces).

- Data-mismatched datatype:

  Given a column containing values "0.23, 0.1, 1, 0, 0, 0, 1, 0.01, ...", the presence of a high number of zero and one only may conduct to wrongly assign boolean as the column datatype. The high percentage of mismatched values (the decimal ones) it is a signal of mis-assignment of the column datatype.

OpenRefine does not have a wide set of datatypes, and it is not capable of assigning a datatype to a column; therefore, it does not have these two features. It should be noted that in some cases we may be able to apply regular expressions to verify if the values in a given column complies with a pattern (i.e., email address structure). Anyway, this is only a workaround because it lacks automatism and intuitiveness.

3.6 Category III: Data Structuring

3.6.1 [III.a] Change column data type

To create a structured dataset, it is useful to assign a correct data type to each column; doing so, we can manage the values for what they really are. For example, treating two columns as numbers allows us to sum them, and this would not have been possible treating them as text.

OpenRefine does not assign a data type to each column and can manage the data type of single cells only. Nevertheless, it offers the possibility to apply the same data type to all the cells in the chosen column. In detail, the data types that we can apply are:

- string: one or more text characters.
- number: one or more characters of numbers only.
- date: ISO8601 format (*YYYY-MM-DDTHH:MM:SSZ*).

The required steps are (see Figure 3.6.2):

1. Choose the column and click on its dropdown menu arrow.
2. Choose "Edit cells" then "Common transforms" and click on one of the following options: "To number", "To date", "To text".

   Note that to correctly convert strings that contain dates formatted in different way you must click on "Transform..." and refer to the GREL function "toDate" to provide the input formats (see section 3.9.1).
3. The operation will be executed. A message in the top-center will appear showing how many rows have been transformed (non-valid values are skipped). Numbers, date or boolean values are now displayed in green.

Another useful feature of OpenRefine is that during import, cells text can be parsed into the appropriate data type. This operation is not suggested for large dataset because it enormously slows down the import process.

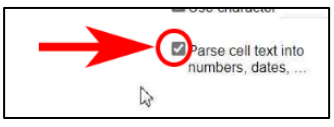


Figure 3.6.1: Change Column Data Type - Import Parse

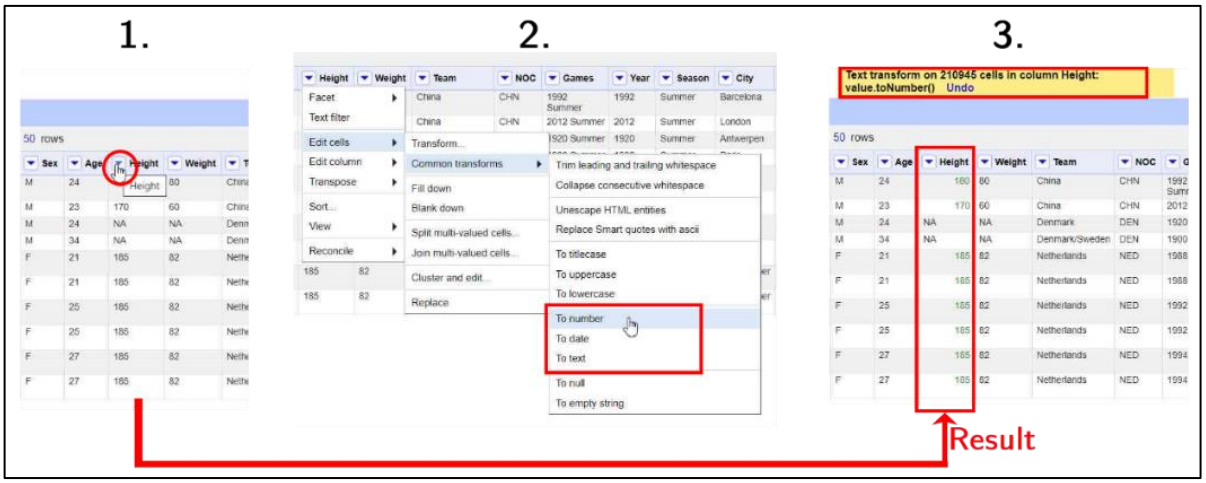


Figure 3.6.2: Steps to Change Column Data Type

OpenRefine also supports boolean data type but only when editing single cells. In OpenRefine boolean means a value between "true" and "false".

It should be noted that the community is discussing about boolean values, in detail:

- extend the values accepted as boolean, for example: 0/1, y/n, on/off.
- implement the possibility to apply boolean data type to all the cells in a column, just like with other types.

Moreover, using expressions to transform/generate columns we have that the data type of the future cells is set accordingly with the data type of the result produced by the expression.

3.6.2 [III.b] Delete column

If we are only interested in some aspects of the complete dataset, then we may decide to discard the data we do not need.

Obviously, this feature is also needed to delete temporary columns that we may have to create for some reasons, or to delete the columns containing any type of useless data (redundant or incorrect).

OpenRefine has this feature. The required steps are (see Figure 3.6.3):

1. Click on the column dropdown menu arrow.
2. Choose "Edit column", then click on "Remove this column".

3. The operation will be executed, then a message in the top-center will appear confirming the removal.

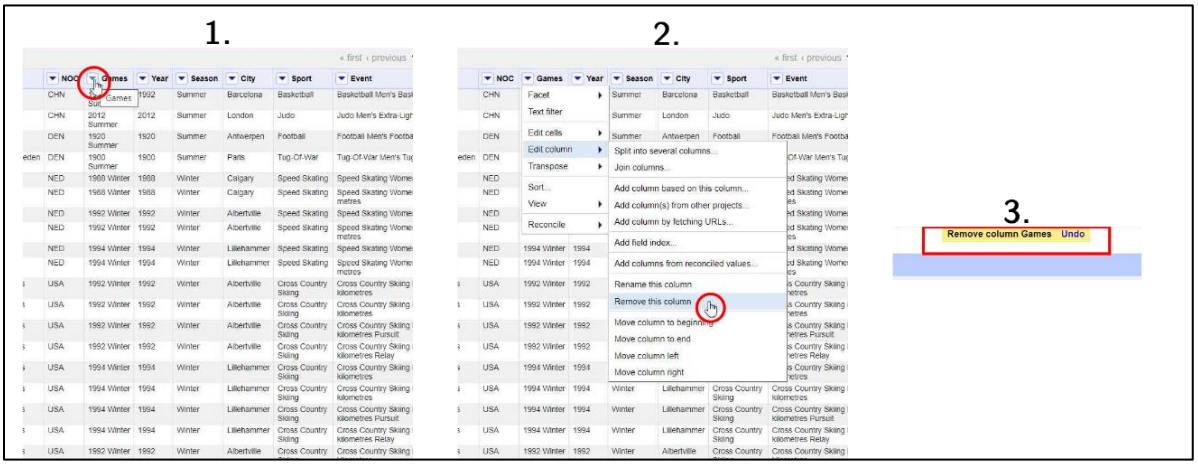


Figure 3.6.3: Steps to Delete Column

A quicker way to delete columns, that is very useful if we need to delete more than one column in just one step, is (see Figure 3.6.4):

1. Click the dropdown menu arrow of the first column ("All").
2. Choose "Edit columns" then click on "Re-order / remove columns...".
3. A pop-up window will appear: move columns from the left list to the right list and then click on "Ok" to remove them.

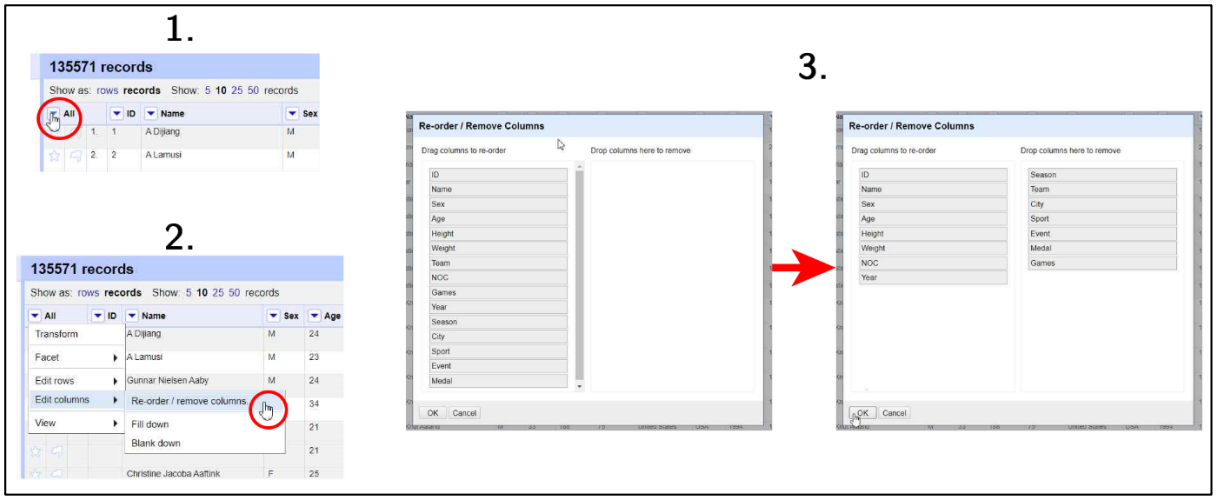


Figure 3.6.4: Steps to Delete More Than One Column

To provide an example, I applied it to the "athlete_events.csv" dataset, because I wanted to obtain a list containing only the attribute related to Olympic athletes, therefore I had to remove all the information related to the events.

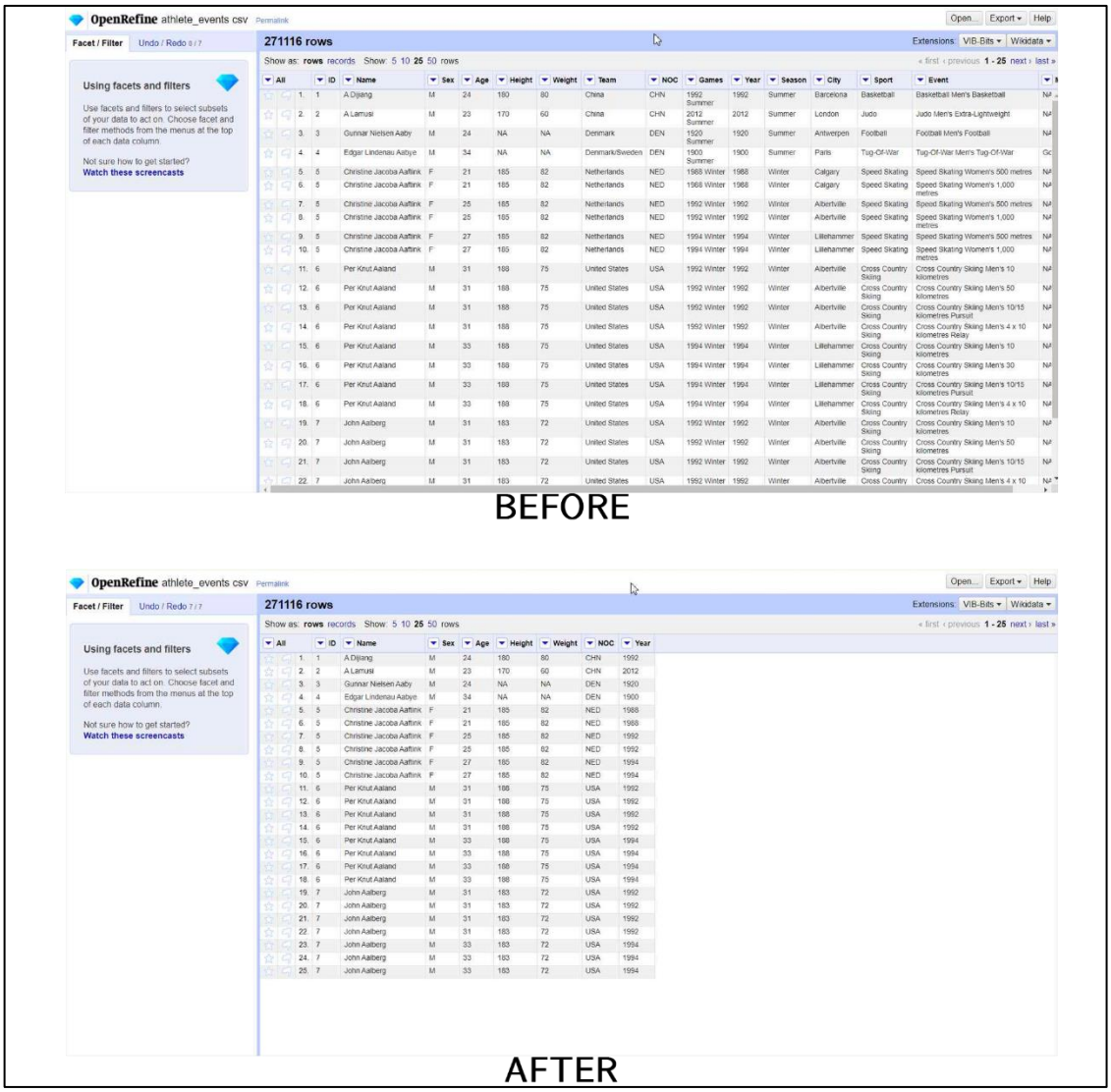


Figure 3.6.5: Delete Column - Before & After on a Dataset

3.6.3 [III.c] Detect & change encoding

This feature consists in auto-detecting the character encoding needed to interpret the data during the import phase, but also in giving the possibility to change this encoding at any time to fix any errors due to improper decoding.

OpenRefine has both characteristics. In detail, when a new project is started and data is loaded, OpenRefine identifies the character encoding needed. The user has the possibility to change the suggested encoding with anyone from the provided list, then to create the project by clicking on the corresponding button.

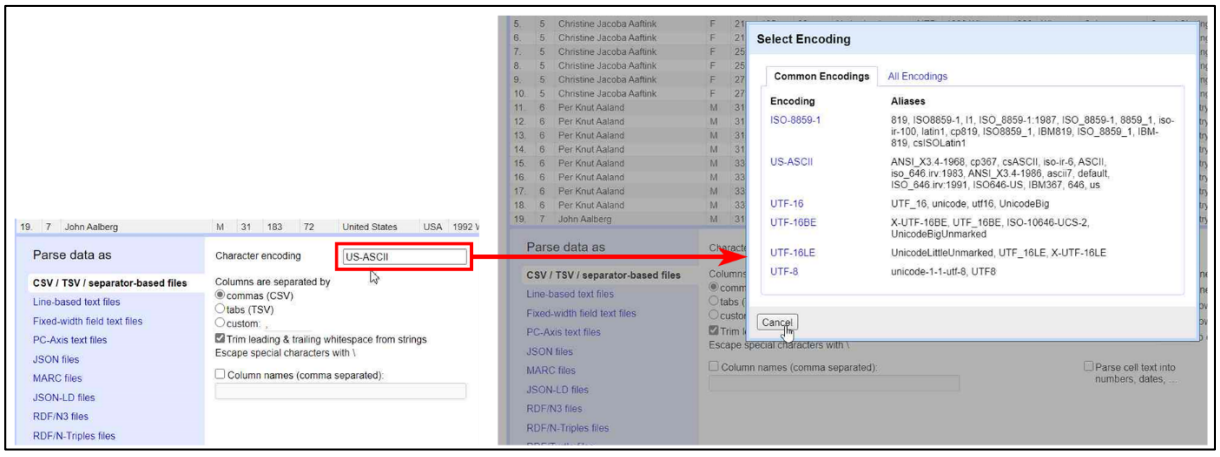


Figure 3.6.6: Detecting Encoding on Import

For fixing encoding errors or just changing the encoding, OpenRefine has the function “reinterpret” that accepts as parameter the string that identifies the encoding. The most common values for the string are: “utf-8” (most of the time fixes all the problems), “latin-1” (for European content), and “Big5” (for Chinese content").

In order to spot potential encoding errors, it is useful to utilize the facet “Unicode char-code facet” that generates the distribution of the Unicode characters used. Observing this distribution, we can spot outliers, that are infrequently used character, that may suggest an encoding error that needs to be inspected.

3.6.4 [III.d] Pivot / unpivot

In order to deal with these two operations, it is necessary to define them first. Therefore, a short and simple definition follows.

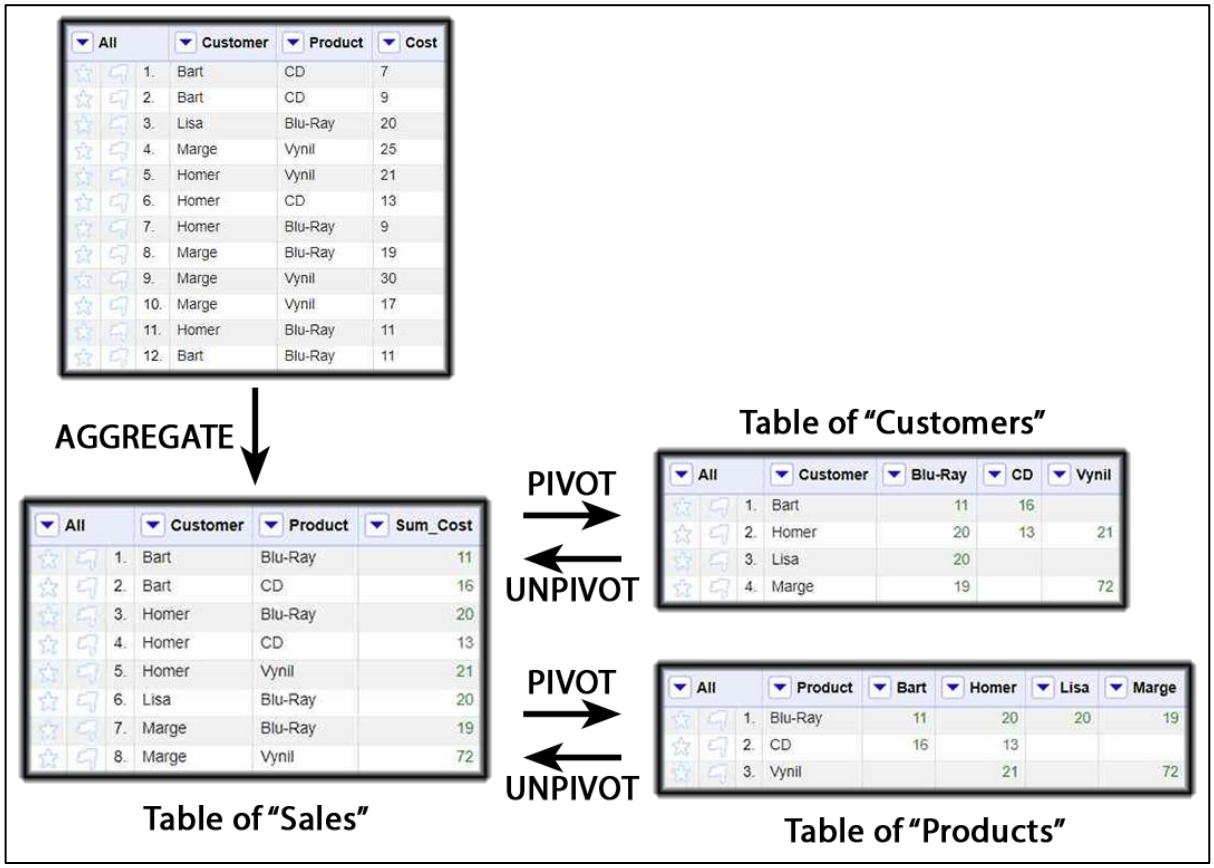
- Pivot: “transforms a series of rows into a series of fewer rows with additional columns. Data in one source column is used to determine the new column for a row, and another source column is used as the data for that new column” [13]. A much simpler definition would be that each unique value (key) in a column becomes a new column. We fill each new column with values from another column, for each one we take only the values which are on the same row of the key.

  Furthermore, in most cases this transformation is used to aggregate values that have the same key. We group the keys and for each group

we produce a summary of the same-key values. Typical summary operations are average and sum.

- Unpivot: transforms columns in key-value pairs, in which the key is the name of the transformed column and the values are the values in its cells.

These two operations are fundamental to rearrange data in order to draw attention on useful information. The following example should clarify these definitions and the usefulness of these operations.



All	Customer	Product	Cost
1.	Bart	CD	7
2.	Bart	CD	9
3.	Lisa	Blu-Ray	20
4.	Marge	Vynil	25
5.	Homer	Vynil	21
6.	Homer	CD	13
7.	Homer	Blu-Ray	9
8.	Marge	Blu-Ray	19
9.	Marge	Vynil	30
10.	Marge	Vynil	17
11.	Homer	Blu-Ray	11
12.	Bart	Blu-Ray	11

All	Customer	Product	Sum_Cost
1.	Bart	Blu-Ray	11
2.	Bart	CD	16
3.	Homer	Blu-Ray	20
4.	Homer	CD	13
5.	Homer	Vynil	21
6.	Lisa	Blu-Ray	20
7.	Marge	Blu-Ray	19
8.	Marge	Vynil	72

All	Customer	Blu-Ray	CD	Vynil
1.	Bart	11	16	
2.	Homer	20	13	21
3.	Lisa	20		
4.	Marge	19		72

All	Product	Bart	Homer	Lisa	Marge
1.	Blu-Ray	11	20	20	19
2.	CD	16	13		
3.	Vynil		21		72

Figure 3.6.7: Pivot and Unpivot Operations

In OpenRefine these operations can be executed, and we can find them under "Transpose" as "Columnize by key/value columns..." for pivot and as "Transpose cells across columns into rows..." for unpivot. The main drawback is the lack of an automatism for aggregation during pivot, so we have to perform it before pivoting. Other data preparation tools offer this capability instead.

3.6.5 [III.e] Rename column

Each column represents a variable/attribute; therefore, it can be useful to name them in a more meaningful way, resulting in a better organization of the dataset.

OpenRefine has this feature. The required steps are (see Figure 3.6.8):

1. Click on the column dropdown menu arrow.
2. Choose "Edit column" then click on "Rename this column".
3. An input box will appear; the new column name can be typed inside it.
4. The operation will be executed, then a message in the top-center will appear confirming the change.

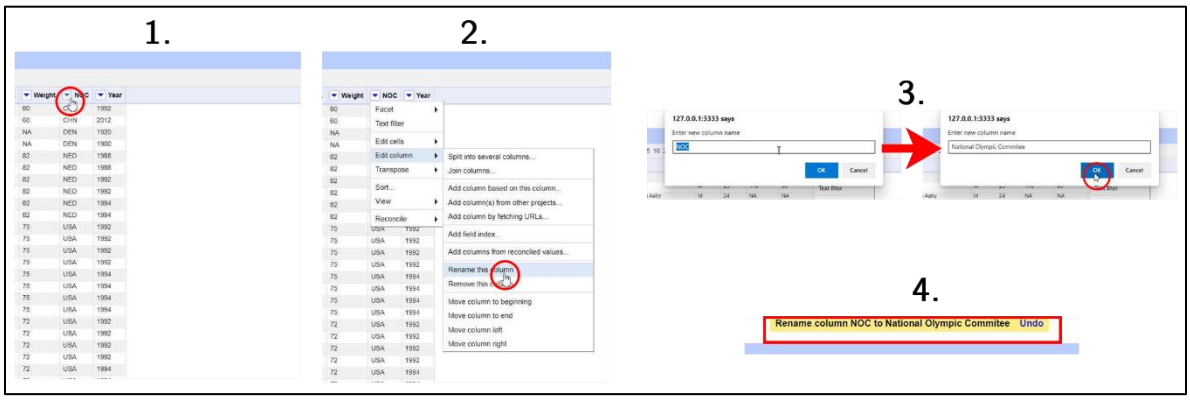


Figure 3.6.8: Steps to Rename Column

To provide an example, I applied it to the "athlete_events.csv" dataset to change the column named "NOC" with the extended denomination "National Olympic Committee".

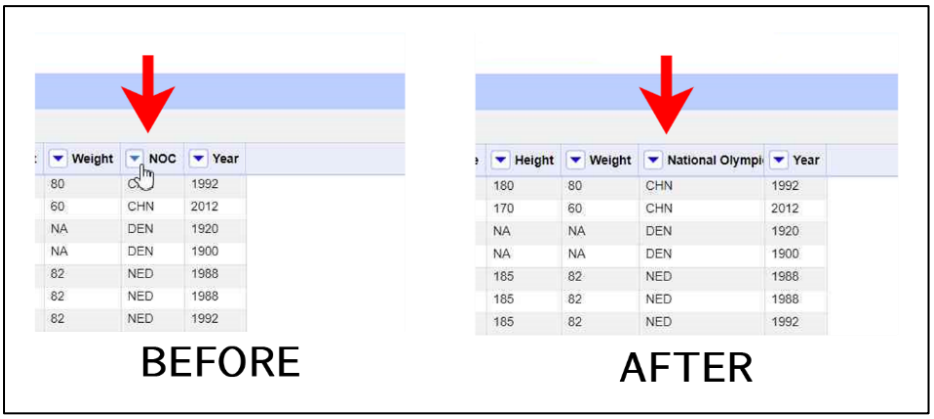


Figure 3.6.9: Rename Column - Before & After on a Dataset

3.6.6 [III.f] Split column

Data can include values that consists of multiple atomic parts separated by a constant separator of some sort. The presence of these multi-valued columns reduces the flexibility of handling data. Therefore, it is needed to split these columns into multiple columns in order to create a structured and more manageable dataset.

OpenRefine offers this feature, allowing us to split a column into several column by separator or by field lengths. To do so, the steps are:

1. Choose the multi-valued column you want to split and click on its dropdown menu arrow.
2. Choose "Edit column" then click on "Split into several columns...".
3. A pop-up window will appear. Fill it with the required information and then click on "Ok" to execute.

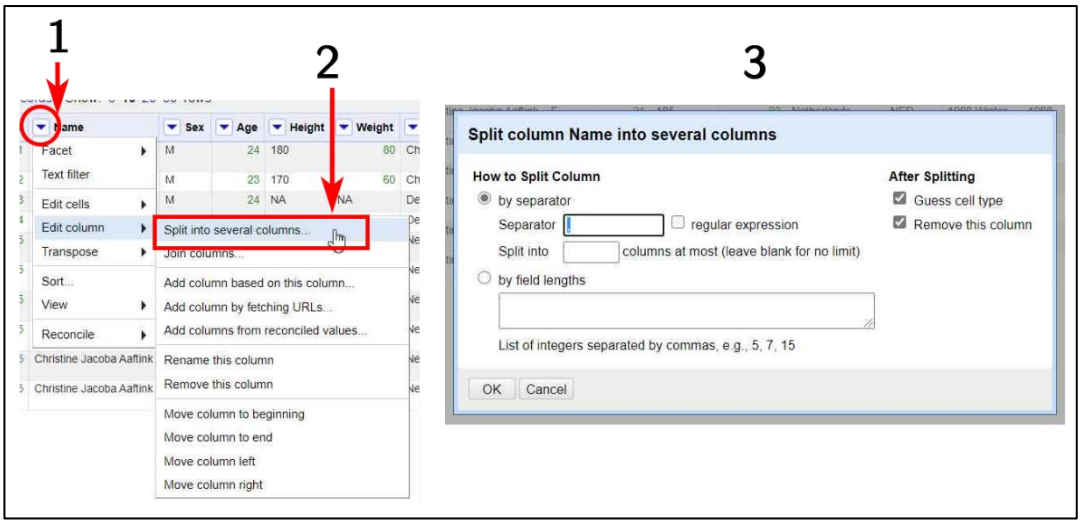


Figure 3.6.10: Steps to Split Column

Let's take a look at the two splitting options:

- Split by separator: each value is separated by a defined separator, that allows us to identify the splitting point.

  For example, this technique could be used to split a number into its integer and decimal part, if we know which character is used ad decimal separator. Otherwise, if we have an address in which each field (street,

civic number, CAP, city, province, state) is divided by a constant separator (e.g., ;) then we can split each field in a dedicated column.

- Split by field lengths: this type of splitting is useful when there is no separator to be used, but the value is composed of fields that have a fixed size. Therefore, knowing the size of each field we can easily split them in separated values, providing the list of field lengths.

  For example, the Italian IBAN is composed of the following fields: national identifier (2 letters), European CIN (2 digits), CIN (1 letter), ABI (5 digits), CAB (5 digits), account number (12 digits). Therefore, if we have a column of IBANs and we want to manage each field individually we can use the feature "split by field lengths" inserting the list "2, 2, 1, 5, 5, 12".

3.6.7 [III.g] Transform by example

This operation consists in "allowing the analyst to describe the transformation with a small input-output example pair, without being concerned with the transformation steps required to get there" [14]. This definition also describes the relevance of this operation in a tool dedicated to make data preparation easier.

Currently OpenRefine does not have this functionality.

3.7 Category IV: Data Enrichment

3.7.1 [IV.a] Assign semantic data type

This operation implicates being able to detect the content of the dataset and proposing/assigning the most appropriate data type for each column. OpenRefine, considering also that it cannot assign a data type to a column, does not have that feature.

3.7.2 [IV.b] Calculate column using expressions

It is one of the most useful features, because starting from the original dataset we can extract more specific information.

OpenRefine has this feature. The required steps are (see Figure 3.7.1):

1. Choose one of the columns that contain the data that will be used as base for the new column. Click on its dropdown menu arrow.
2. Choose "Edit column" then click on "Add column based on this column...".
3. A pop-up window will appear. The name for the new column is required. The "Expression" text field must contain a valid expression that describes how to calculate the new column; this description must be written in one of the supported expression languages (see section 2.8.8). This expression is compiled and evaluated in real-time, therefore errors can be seen on the right of the menu and a preview of the content of the new column can be seen at the bottom of the menu. Click "Ok" to execute.
4. The operation will be executed, then a message in the top-center will appear confirming it.

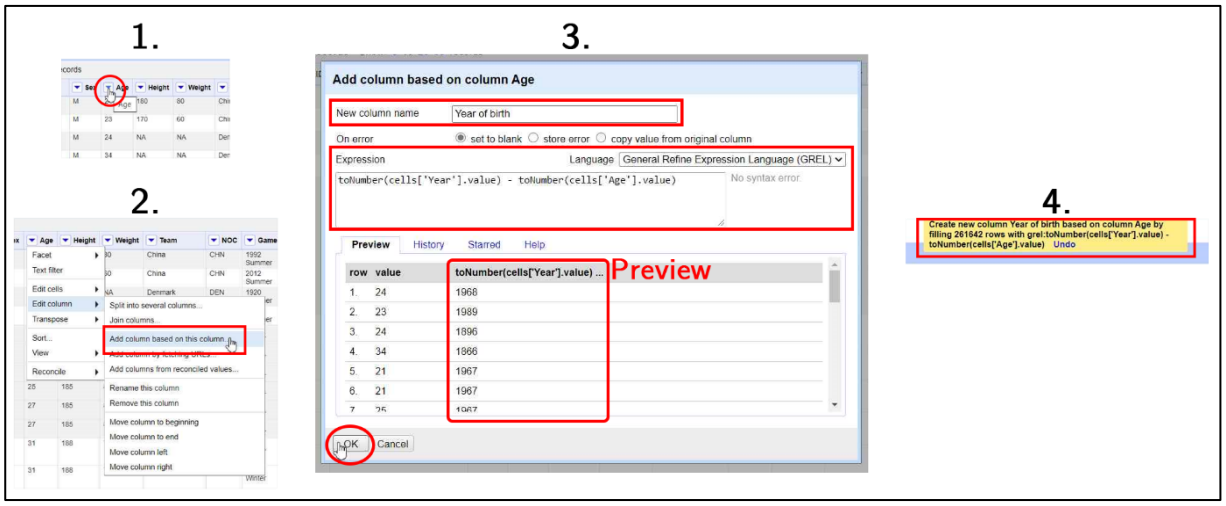


Figure 3.7.1: Steps to Calculate Column Using Expressions

To provide an example, I applied it to the "athlete_events.csv" dataset to calculate the year of birth given the year of the event and the age of the athletes at the moment of their participation in it.

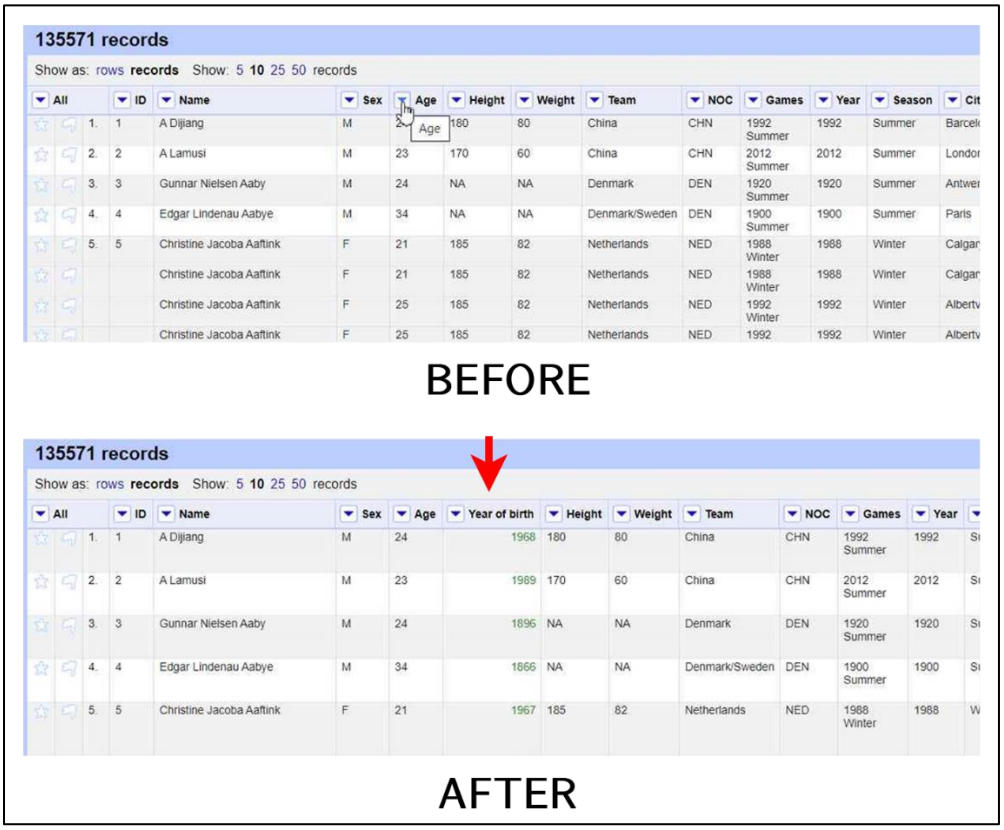


Figure 3.7.2: Calculate Column Using Expressions - Before & After on a Dataset

3.7.3 [IV.c] Discover & merge external data

This operation allows to retrieve additional data from external resources starting from our datasets. This operation goes from searching a set of values in an external dataset to retrieving data from URLs.

In OpenRefine this feature is called "Reconciliation". In detail, the software provides two options:

- "Add column by fetching URLs..."

  Useful for retrieving data from web APIs via simple HTTP requests.

  This feature can also be used for retrieving an entire HTML page that we will parse later in order to extract the information that we want.

- "Add columns from reconciled values..."

To use this functionality, we need to reconcile the values first. "Reconciliation is the process of matching your dataset with that of an external source" [8]. A long list of available reconciliation services and methods to reconcile against a local dataset are provided in the software reference. By default, OpenRefine comes with Wikidata reconciliation service (Wikidata is a vast user-maintained data source).

As an example, starting from reconciling the column "Games" of the "athlete_events.csv" dataset I was able to retrieve additional information on the Olympic Games, such as the country where they took place and the edition number.

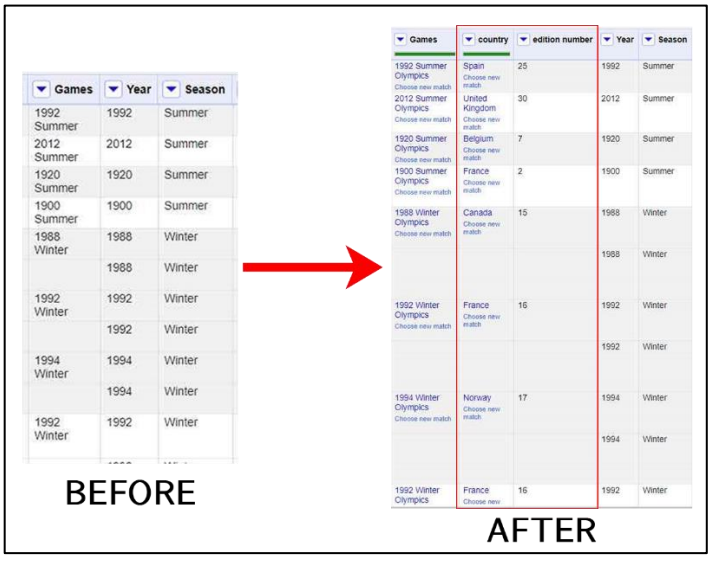


Figure 3.7.3: Retrieve External Data - OpenRefine Reconciliation Feature

3.7.4 [IV.d] Duplicate column

A more complex operation could require a duplication feature as one of the steps. For example, if we want to maintain the original column but also join it with another one, then duplication is needed.

OpenRefine has this feature. The required steps are (see Figure 3.7.4):

1. Choose the column to be duplicated and click on its dropdown menu arrow.
2. Choose "Edit column" then click on "Add column based on this column...".
3. A pop-up window will appear. The name for the new column is required, then click on "Ok". The text in the "Expression" text field must be "value" (it should be already there), meaning that a new column will be created and will be filled with the values of the current one.
4. The operation will be executed, then a message in the top-center will appear confirming the duplication.

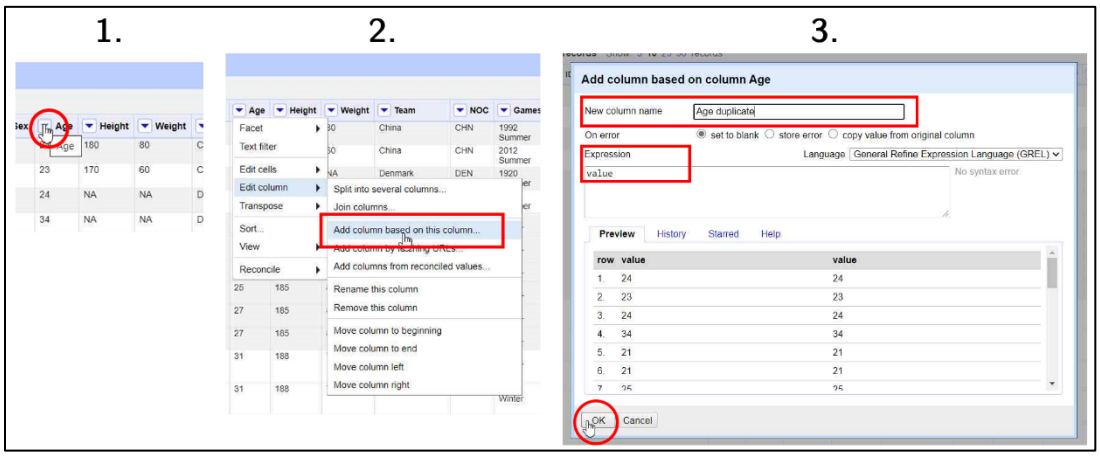


Figure 3.7.4: Steps to Duplicate Column

3.7.5 [IV.e] Generate primary key column

Primary key column can be created applying the concepts defined in section "2.8.6 Rows vs Records". The required steps are (see Figure 3.7.5):

1. Identify the column to use as primary key and click on its dropdown menu arrow.
2. Choose "Edit column" then click on "Move column to beginning".
3. Group the rows of the same record by applying an appropriate sorting rule.

4. Click on its dropdown menu arrow. Choose "Edit rows" then click on "Blank down".
5. Switch to "records mode" to display the dataset to better visualize the utility of the primary key column.

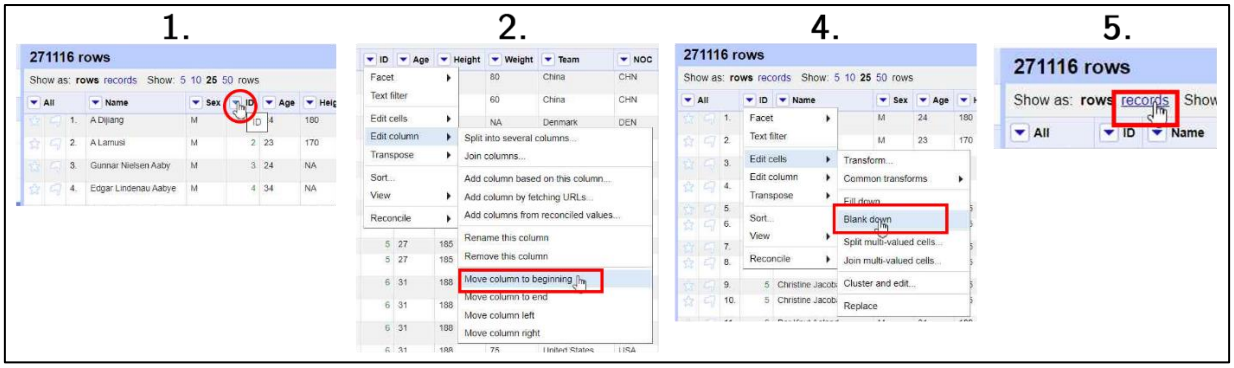


Figure 3.7.5: Steps to Generate Primary Key Column

3.7.6 [IV.f] Join & union

Being able to combine multiple datasets is an important part of data enrichment. Join and union are the two operations that perform this task.

- Union

This operation consists in concatenating multiple datasets together. In detail, starting from the first dataset we append all the rows from the others. Columns with matching names are collapsed in one; otherwise, empty values are inserted to fill cells in rows that do not have that column.

DATASET 1

Col1	Col2	Col3
Val1.1	Val2.1	Val3.1
Val1.2	Val2.2	Val3.2
Val1.3	Val2.3	Val3.3

DATASET 2

Col1	Col2	Col4
Val4.1	Val5.1	Val6.1
Val4.2	Val5.2	Val6.2
Val4.3	Val5.3	Val6.3

UNION DATASET

Col1	Col2	Col3	Col4
Val1.1	Val2.1	Val3.1	
Val1.2	Val2.2	Val3.2	
Val1.3	Val2.3	Val3.3	
Val4.1	Val5.1		Val6.1
Val4.2	Val5.2		Val6.2
Val4.3	Val5.3		Val6.3

Figure 3.7.6: Union Example

The example in Figure 3.7.6 shows that we have two columns ("Col1" and "Col2") that are in both datasets, but also "Col3" and "Col4" that are present only in "Dataset

1" and "Dataset 2" respectively. The union operation collapses the matching columns and fills with empty values the rows of columns without a match.

In OpenRefine this operation can be executed only at import time by loading more than one file. If we have already modified some datasets that now we want to unite, then the solution is to export each one of them and importing them together as a new project.

- Join

A join operation is useful for pulling in additional information from a second dataset, based on a column that appears in both datasets and contains the same unique values to identify the rows. Based on this key, the records from the two datasets are joined together. How the two datasets are joined is defined by the type of join: inner join, left join, right join, outer join.

In OpenRefine the join operation is accomplished by the GREL function "cross", which is used as follows:

1. Identify the two projects to join. Here we will call the project in which the data will be joined as *main* (main) and the project that provides the additional data as *add* (additional).
2. Note down the name of the *add* project and also the column name of the column containing the unique key that we will use.
3. In project *main* identify the column which contains matching key values. Click on its dropdown menu arrow.
4. Choose "Edit column" then click on "Add column based on this column...".
5. A pop-up window will appear. Insert the name for the new column. Fill the following expression and insert it in the expression text field:

```
cell.cross("addName", "addKeyCol")[0].cells["addImportCol"].value
```

   - addName is the name of the project *add*.
   - addKeyCol is the name of the column identified at "point 2".

- addImportCol is the name of the column of the project *add* that we want to import in project *main*.

6. Click on "Ok" to apply the join.

Note that we have to repeat these steps for every column that we want to import. The only type of join supported by OpenRefine is the left join. Each row in the main (left) dataset appears in the results, regardless of whether there are matches in the additional (right) dataset. For rows that have no match we have that the corresponding values in the result are empty.

As a final note, in order to execute the join operation faster and with a more user-friendly process we may want to install the "VIB-Bits" extension [15]. This extension among other useful features provides a user graphical interface for the GREL function "cross".

3.7.7 [IV.g] Merge columns

Depending on the scope of the dataset, we may need to combine data from multiple columns into one in order to produce a more usable and meaningful dataset. An example could be merging date parts spread across multiple columns (day, month, year, etc.) into just one column only that represents the complete date.

The merging operation can include tasks like inserting a separator between column values and deleting the original columns.

In OpenRefine this operation can be done using expressions just like in section 3.7.2. The only difference is that we have to transform each value in a string and to use the concatenate operator +. Advantageously, OpenRefine also presents a shortcut to this operation, making it more user-friendly and easier to execute. This shortcut is accessible following the steps (see Figure 3.7.7):

1. From any column click on its dropdown menu arrow.

   Note that the chosen column can be selected later as the column to store the merge result.

2. Choose "Edit column" then click on "Join columns...".

3. A pop-up window will appear. In the left panel, select and order the columns to join. In the right panel, select the merging options:
    a. Separator.
    b. Null handling: decide how to handle potential null values.
    c. Escape sequences: select the checkbox to interpret escape sequences for new line and tabulation.
    d. Columns handling: decide where to write the merge result, put it in the column selected in step 1 or create a new column. Decide if you want to delete the source columns.
4. Click on "Ok" to apply the merging.

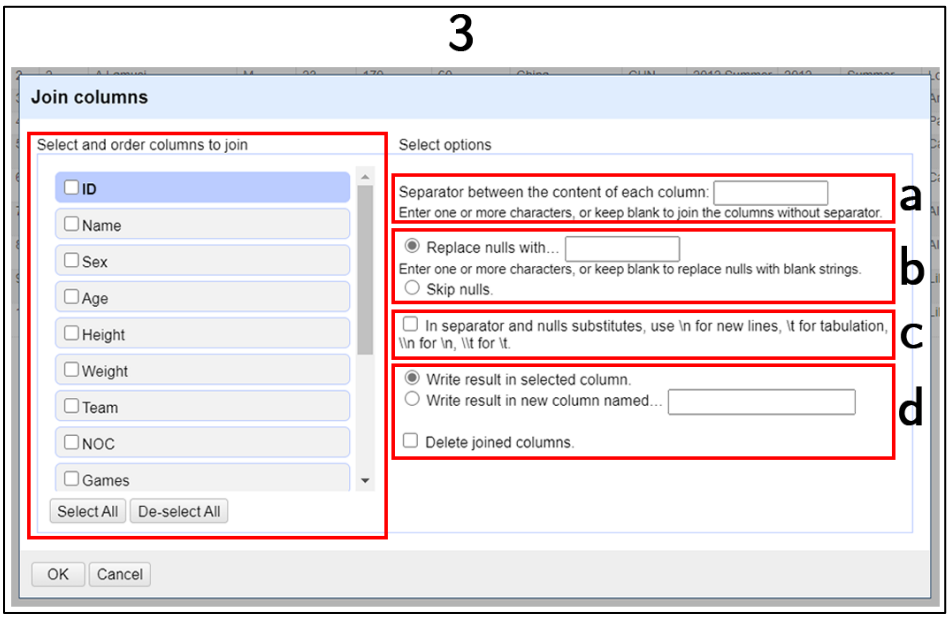


Figure 3.7.7: Steps to Merge Columns

3.7.8 [IV.h] Normalize numeric values

Numeric values may be stored in the original dataset with different level of precision; therefore, a useful operation is to standardize it. An example could be transforming numbers with different decimal precision into numbers with only n decimal digits.

OpenRefine does not have these capabilities. The only things that it offers are the classic mathematical functions for rounding, which are "ceil", "floor", and "round".

3.8 Category V: Data Filtering

3.8.1 [V.a] Delete/keep filtered rows

In order to create a subset from the original dataset it is fundamental to be able to keep only the rows that we are interested in.

OpenRefine has this feature. The required steps are (see Figure 3.8.1):

1. Apply the wanted filter/facet, remembering that the rows that are displayed are the ones that will be deleted.

   If a facet was applied, then select the subsets to display (thus to delete).

   If a filter was applied and the filtered rows are the ones to be kept, then it is necessary to click on "invert" in the filter tab on the left.

   More complex facets can be created using expressions.
2. Click the dropdown menu arrow of the first column ("All").
3. Choose "Edit column" then click on "Remove matching rows".
4. The operation will be executed, then a message in the top-center will appear confirming the delete operation.
5. The filters/facets can be removed by closing the respective tabs, displaying the updated dataset.

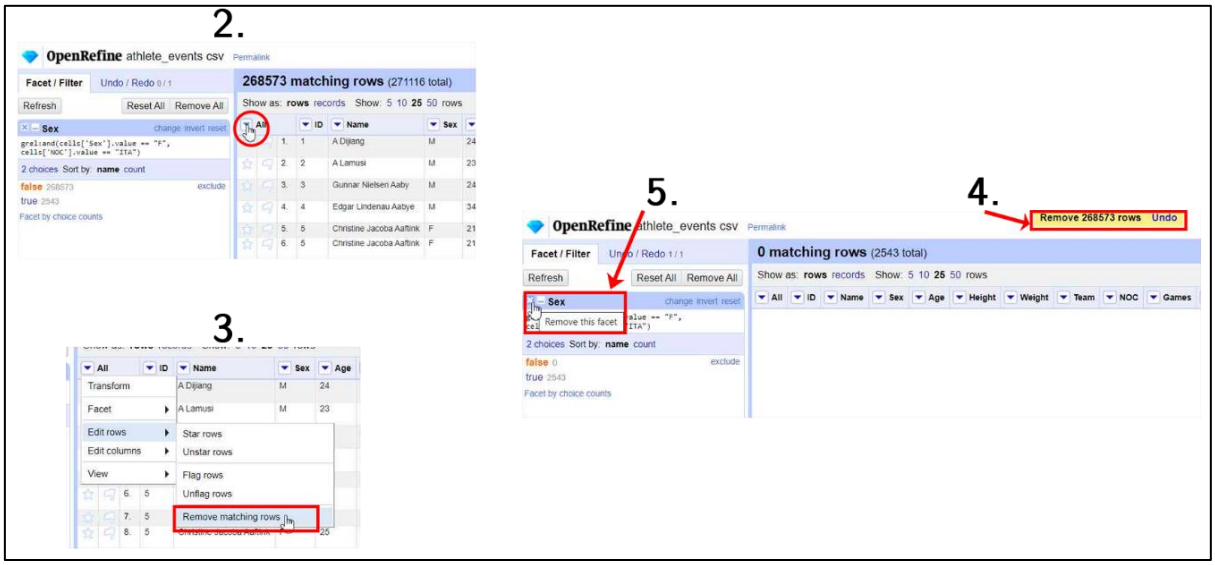


Figure 3.8.1: Steps to Delete/Keep Filtered Rows

To provide an example, I applied it to the "athlete_events.csv" dataset, in order to keep only the Italian female athletes subset. This operation could have been done in two steps (delete all the male athletes and delete all the non-Italian athletes) or just in one, like I did, using an expression to create a facet that describes the wanted subset.

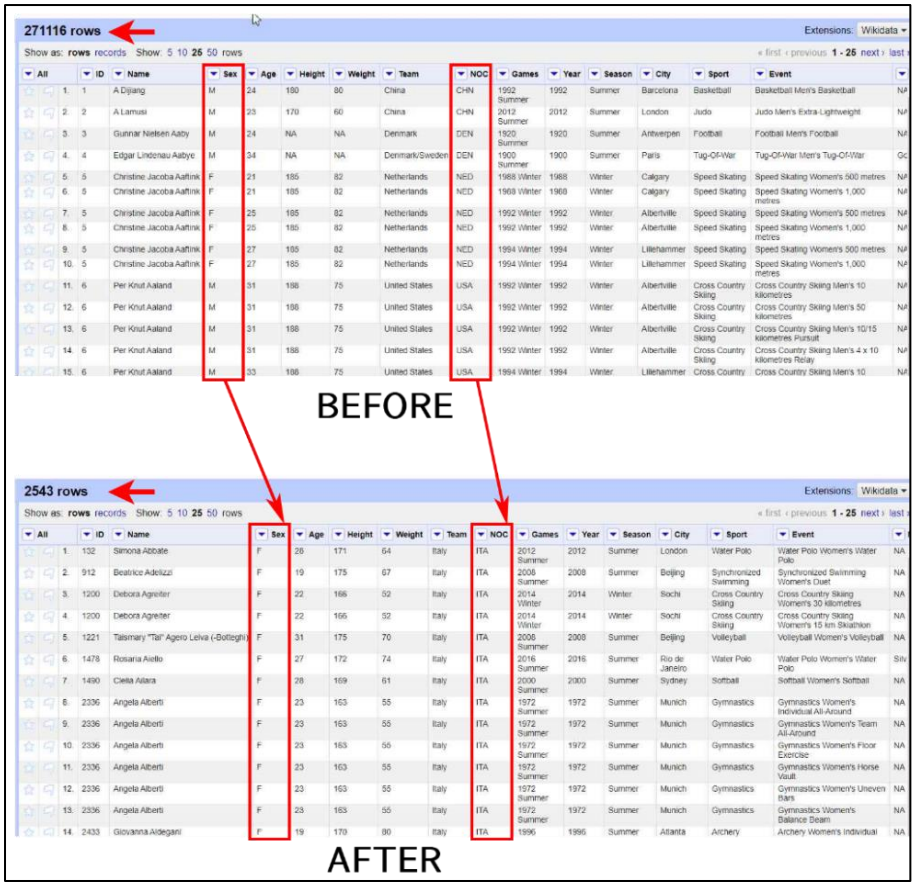


Figure 3.8.2: Delete/Keep Filtered Rows - Before & After on a Dataset

3.8.2 [V.b] Delete empty and invalid rows

First of all, empty rows can be ignored, in some data formats, at importing time by deselecting the option "Store blank rows" as in Figure 3.8.3.

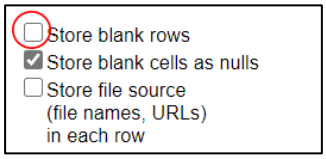


Figure 3.8.3: Ignore Empty Rows During Import

If we want to delete empty/invalid rows and the dataset has been already imported, then we have to proceed in a similar way as in 3.8.1. This time we have to create a filter/facet for empty or invalid rows.

One option is to manually flag the rows that we want to delete (by clicking on the symbol present at the start of each row) and then apply the default facet for flagged rows.

Another solution is to use expressions to create the facet for the unwanted rows. To detect empty rows, we need to create a custom text facet with the following expression:

```
(filter(row.columnNames,cn,isNonBlank(cells[cn].value)).length()==0).toString()
```

If we have defined a pattern for invalid rows, then it is easy to create a facet that uses an expression that describes it.

3.8.3 [V.c] Extract value parts

Being able to extract value parts it is a useful feature in order to split across multiple specific cells data that is condensed in one cell.

If we need to extract a pattern from a string, then OpenRefine can do this combining the GREL function "find" with a regular expression. An example could be extracting people names from a text string, in OpenRefine the expression would be:

```
value.find(/([A-Z][a-z]|[\u00C0-\u00FF])+)\s([A-Z][a-z]|[\u00C0-
\u00FF])+)/).join(" | ")
```

Another typical operation is to extract date part. In OpenRefine this is done using the GREL function "datePart" that has as parameters a date value and a string indicating the date part wanted. The complete list of valid strings can be found on the GREL manual. For example, using the ISO-8601-compliant date *1998-02-14T21:30:00Z* as value, we can extract:

- the year (the output is *1998*).

  ```
  value.datePart("year")
  ```

- the week of the month (the output is *2*).

  ```
  value.datePart("week")
  ```

- the day of the week (the output is *SATURDAY*).

  ```
  value.datePart("weekday")
  ```

Knowing how to extract value parts it is easy to create/fill other column with these extracted values.

3.8.4 [V.d] Filter with regular expressions

This is an important possibility in filtering operations because we can define a regular pattern and filter data that matches it.

OpenRefine can handle regular expression in multiple places, including filtering; in detail, regular expressions can be used both in filter and in facet:

- In filter tab input the regular expression and select the checkbox "regular expression".
- In facets, insert them in the "Expression" text field, combined with appropriate functions. The most common GREL function for this purpose is called "match".

To provide an example, I decided to apply this feature to the file "color-info.list.gz" contained in the "IMDb" archive. In this file, each row stores a tv-series/movie and the relative information about whether it is in color or in black and white. To produce an organized dataset, we need to split the information in dedicated columns, but we observe that the rows about tv-series store information differently from the rows referring to movies. Therefore, we need to be able to apply different rules to the two types of rows. Knowing that the title of tv-series is written between quotation marks unlike the title of movies, we can use a regular expression to filter the rows.

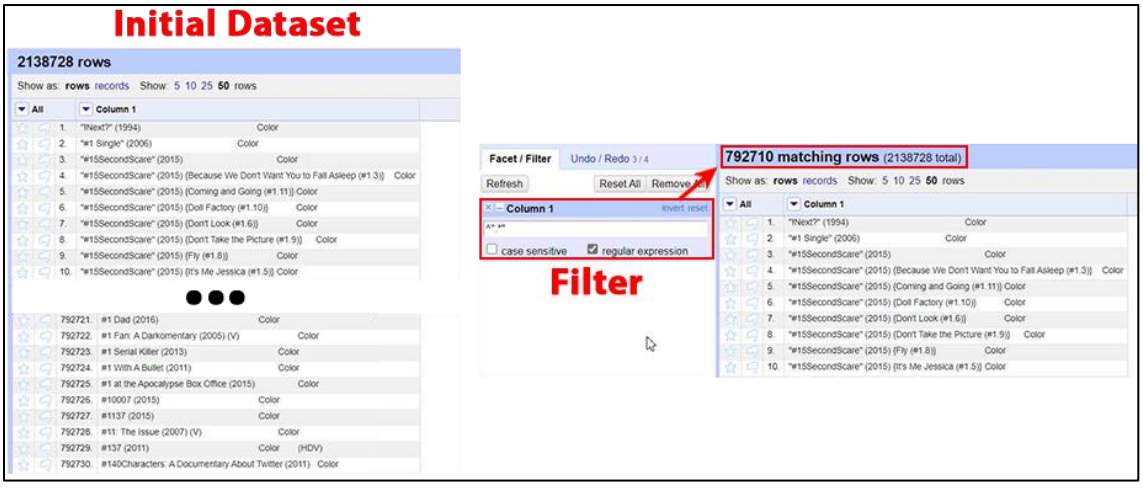


Figure 3.8.4: Filter with Regular Expression - Example on a Dataset

3.9 Category VI: Data Cleaning

3.9.1 [VI.a] Change date & time format

We may have datetime stored as strings and we can also have it represented in different formats. Therefore, it is needed to convert them to a standardize format.

OpenRefine offers this possibility using the GREL function "toDate". The following example should clarify its usage:

Input	Transformation	Output
1998/02/14	`value.toDate('yyyy/MM/dd', 'dd.MM.yyyy', 'yyyy, MMM dd')`	1998-02-14T00:00:00Z
14.02.1998		
1998, Feb 14		

Figure 3.9.1: Parse Different Date Formats - Example

Furthermore, we may need to convert a date and time format to another one.

The date data type in OpenRefine represents dates accordingly with the ISO8601 format. Therefore, we cannot properly change the date and time format. Anyway, we can use the GREL function "datePart" or "toString" to create a string that represents date in a custom format.

Furthermore, OpenRefine also offers the conversion from an ISO8601 string to an Epoch/Unix time formatted string (number of seconds that have elapsed since January 1, 1970 midnight UTC/GMT), and vice versa. These operations can be done as follows:

- ISO8602 string → Epoch/Unix time string

  We use the parameter "time" in the GREL function "datePart" obtaining the number of milliseconds between the input date and the Unix Epoch. In order to obtain the number of seconds, as required by the definition, we need to divide the obtained value by 1000:

  ```
  value.toDate().datePart("time")/1000
  ```

- Epoch/Unix time string → ISO8602 string

  We have use the following Jython script:

  ```
  import time; epochlong = int(float(value));
  datetimestamp = time.strftime('%Y-%m-%dT%H:%M:%SZ',
  time.localtime(epochlong)); return datetimestamp
  ```

3.9.2 [VI.b] Change letter case

This is an important feature that allows a future textual analysis without worrying about case-sensitivity.

OpenRefine has this feature. The required steps are (see Figure 3.9.2):

1. Choose the column and click on its dropdown menu arrow.
2. Choose "Edit cells" then "Common transforms" and click on one of the following options: "To titlecase" (it capitalizes the first letter of each word), "To uppercase", "To lowercase".
3. The operation will be executed, then a message in the top-center will appear confirming it.

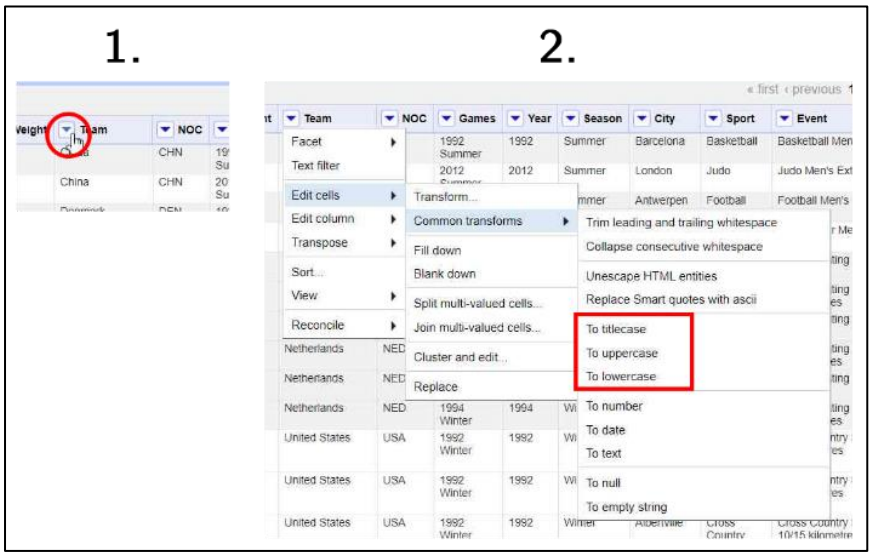


Figure 3.9.2: Steps to Change Letter Case

3.9.3 [VI.c] Change number format

Applying the appropriate number format in a dataset produces an organized and coherent view. This means that we want to store numeric values formatted accordingly with the quantity they represent, and also accordingly with a defined set of rules.

Some examples are: percentage values, to use a defined decimal separator, to format negative numbers between parenthesis, to use scientific notation. Figure 3.9.3 provides a visualization of these examples.

OpenRefine automatically decides how to visualize numeric values; therefore, there is not a feature that allows that operation.

Original Value	*What we want*	*Formatted*
0.123	explicit percentage	12.3%
1234,5	use "." as decimal separator	1234.5
-123423	negative numbers between parenthesis	(-123423)
12345	scientific notation	1,2345E4

Figure 3.9.3: Formatting Numeric Values - Examples

3.9.4 [VI.d] Deduplicate data

One of the most important steps in data cleaning is to remove duplicate data (in detail, duplicate columns or duplicate rows).

In OpenRefine we can perform both.

Regarding the delete operation for columns see section 3.6.2. Knowing that, we just have to define how to check whether a column is duplicated. To do so we have to:

1. Choose a column and click on its dropdown menu arrow.
2. Choose "Facet" and click on "Custom text facet...".
3. A pop-up window will appear. In the expression field insert the following GREL expression, replacing *ColA* and *ColB* with the names of the two column that you need to check for duplication. Then click on "Ok" to apply.

```
(cells['ColA'].value == cells['ColB'].value)
```

4. A facet tab will appear on the left. A duplicated column will produce only the choice "true" with a number of occurrences equal to the number of rows.

Instead, the delete operation for duplicated rows consists of two cases:

A. Each row has a unique identifier; therefore, a duplication in its column means that that the entire row is duplicated.
    1. Identify the column click on its dropdown menu arrow.
    2. Choose "Edit column" then click on "Move column to beginning".
    3. Apply an appropriate sorting rule (remember to apply the reorder permanently) to group duplicated values.

4. Click on its dropdown menu arrow. Choose “Edit rows” then click on “Blank down”.
5. Click on its dropdown menu arrow. Choose “Facet”, then “Customized facets”, then click on “Facet by blank”.
6. The facet tab will appear on the left: select “true”.
7. Click on the dropdown menu arrow of the column “All”. Choose “Edit rows” then click on “Remove all matching rows”.
8. Remove the facet to finish.

B. We have to check whether the entire row is duplicated.

1. Generate a new column based on the following GREL expression. It will generate a key for each row by concatenating all its cells, separated by a unique separator (chose a character that is not used, an example could be the unit separator coded as U+241F).

```
forEach(row.columnNames, cn, cells[cn].value).join("␟")
```

2. We use the generated column to execute the previously described “point A”.

It should be noted that we can modify “point B” to concatenate even a reduced set of columns if needed.

I used the “athlete_events.csv” dataset to:

- Extract a list of the athletes. We can identify each athlete with the “ID” value, in order to provide an example for row deduplication described in point A (see Figure 3.9.4).
- Verify whether the column “Games” contains the combined information of the columns “Year” and “Season”. As shown in Figure 3.9.5 we have that the facet produces only the “true” value, confirming our hypothesis; therefore, we can decide to delete the column that we do not want. We provide this example as a useful extension of the concept of column deduplication.

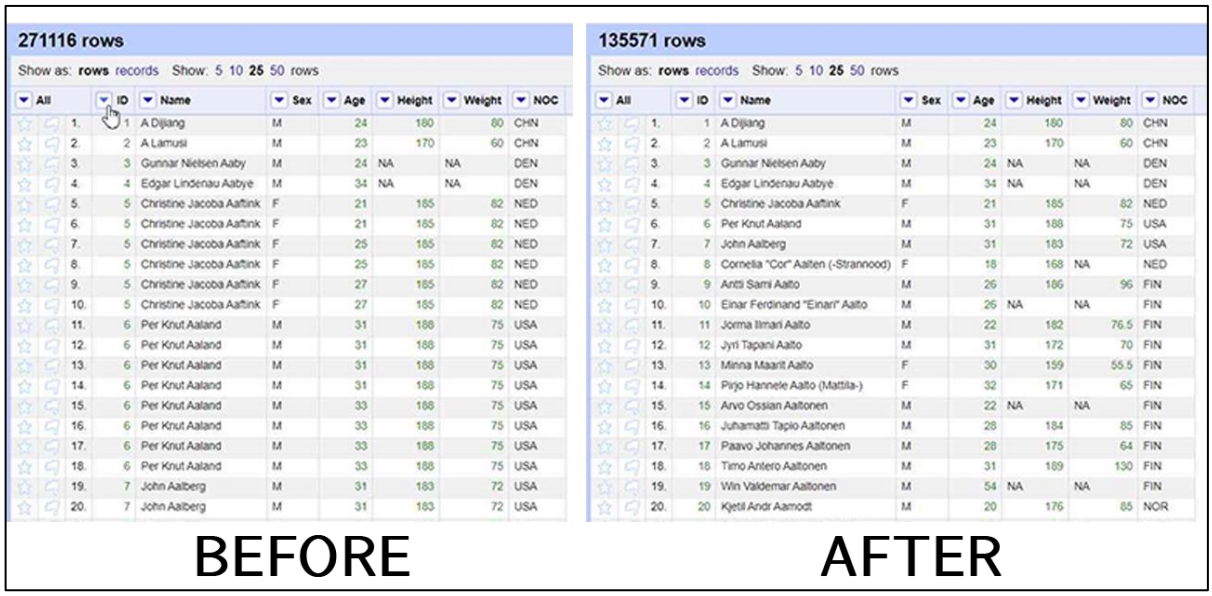


Figure 3.9.4: Row Deduplication - Before & After on a Dataset

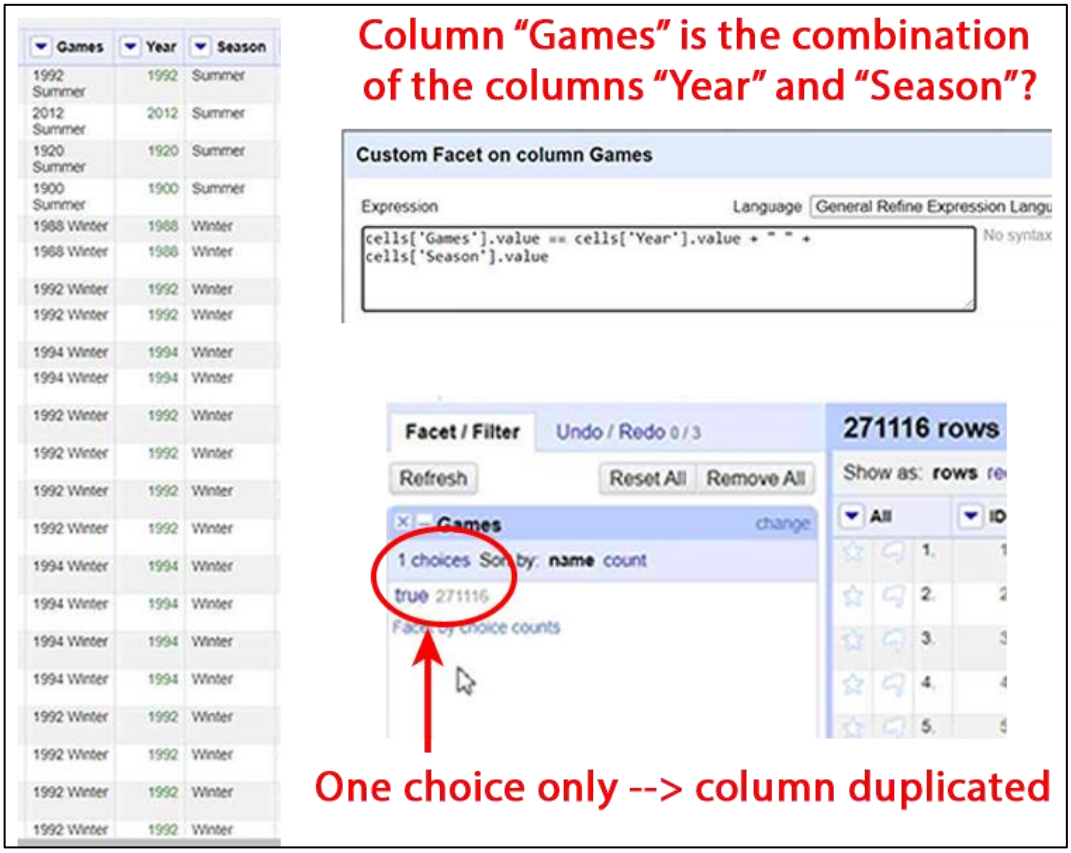


Figure 3.9.5: Example of Checking Column Duplication

3.9.5 [VI.e] Delete by pattern

In order to perform a faster cleaning, it could be useful to delete all the values that match a defined pattern. In OpenRefine this task can be performed by filtering using the pattern (see section 3.8.4) and then deleting all the filtered rows.

Instead, if we want to delete values in single cells, we can use the "replace" functionality (see section 3.9.6) using an empty string as replacement.

3.9.6 [VI.f] Edit & replace cell data

The faster way to improve/clean the dataset is through automated operations, but in some cases, it could be useful to edit single cells to correct specific errors. Another useful feature to have is a find/replace function, in order to find and correct the same error.

OpenRefine allows these operations.

Individual cells can be edited by hovering the mouse over that cell. A blue "edit" label will appear, click it to edit. A pop-up window will appear with a bigger text field. In this window you can edit the text (and also change the data type of the cell) and when you are done click on "Apply" (see Figure 3.9.6).

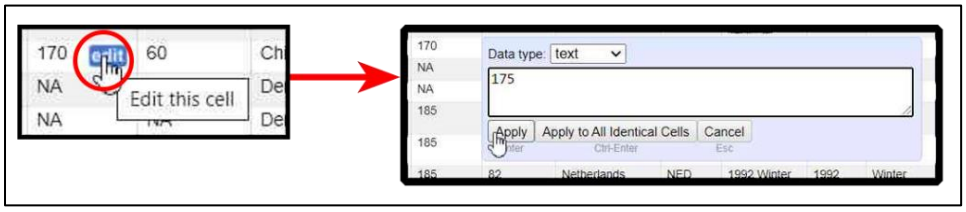


Figure 3.9.6: Edit Single Cells

A first form of find/replace operation can be executed from the edit cell pop-up window, selecting "Apply to all identical cells". Instead, the proper and more powerful find/replace function can be found following the steps (see Figure 3.9.7):

1. Choose the column and click on its dropdown menu arrow.
2. Choose "Edit cells" and click on "Replace".
3. A pop-up window will appear. The string to search and the string to replace must be input. Other options can be selected: case-sensitivity, whole words selection only, enable regular expressions. Click on "Ok" to apply it.

4. The operation will be executed, then a message in the top-center will appear confirming the replacement operation, it indicates how many cells have been modified.

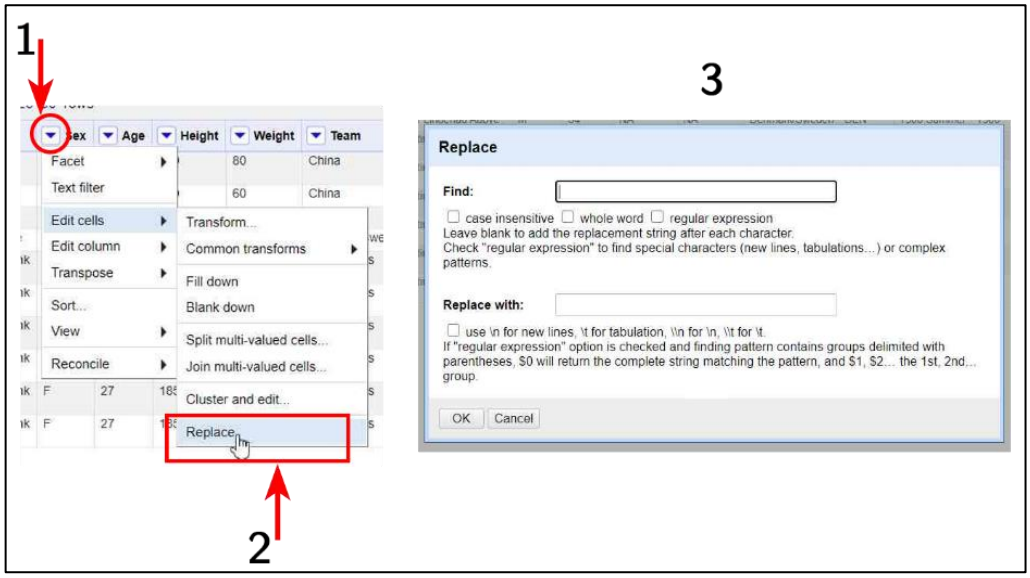


Figure 3.9.7: Steps to Find/Replace Data

3.9.7 [VI.g] Fill empty cells

This function consists of being able to populate blank cells with values from the cell above. In OpenRefine this operation is done by clicking the dropdown menu arrow, choosing "Edit cells" and the clicking on "Fill down".

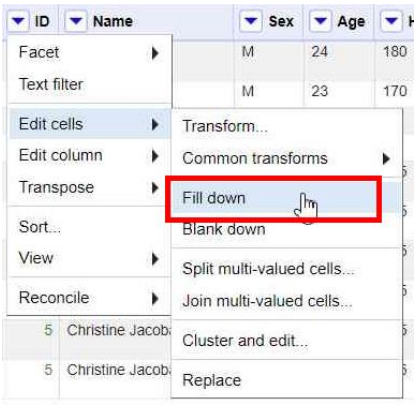


Figure 3.9.8: Fill Down

3.9.8 [VI.h] Remove extra whitespace

Whitespaces could be wrongly added to a string due to various factors (such as typing errors or encoding issues); therefore, to have a clean string it is necessary to remove these extra whitespaces.

OpenRefine has this feature. The required steps are (see Figure 3.9.9):

1. Choose the column with extra whitespace and click on its dropdown menu arrow.
2. Choose "Edit cells" then "Common transforms" and click on one of the following options depending on the case:
   - "Trim leading and trailing whitespace": removes all the whitespaces that are at the start and at the end of a string. This option is also provided in the importing phase for some formats.
   - "Collapse consecutive whitespaces": substitutes multiple consecutive whitespaces with just one space.
3. The operation will be executed, then a message in the top-center will appear confirming it.

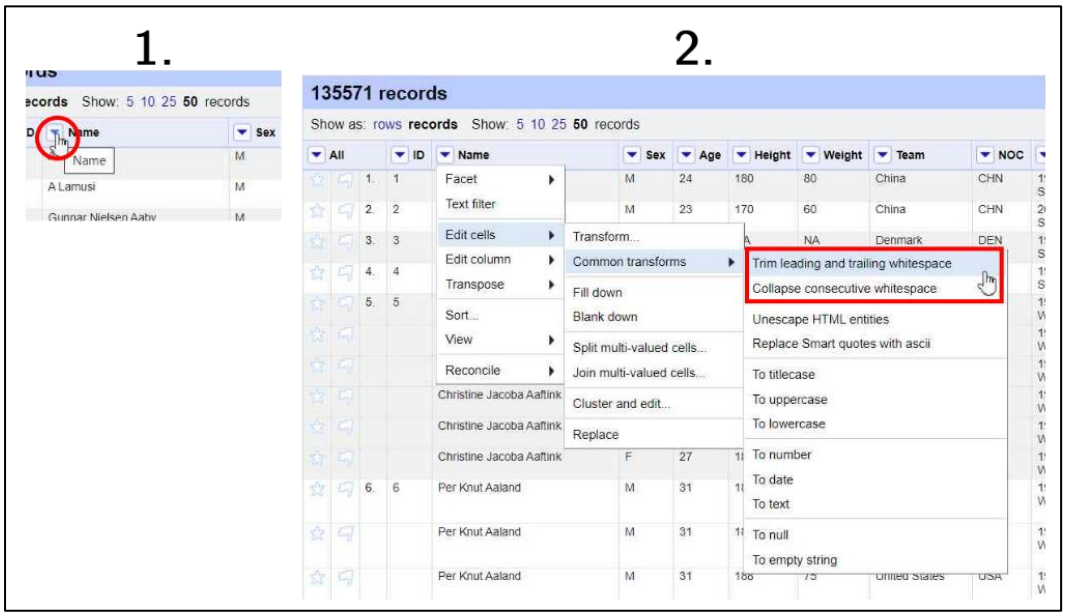


Figure 3.9.9: Steps to Remove Extra Whitespaces

3.9.9 [VI.i] Remove diacritics

This operation consists in removing the accent from letters and substituting them with the basic form. This operation is fundamental in order to produce a more usable/compatible dataset, because it reduces future encoding/decoding issues. An example of this operation is: "Boris Villazón" → "Boris Villazon".

OpenRefine does not directly have this feature.

Nevertheless, consulting the user manual we find how to extend Jython functionalities with modules/libraries. Following these instructions, we can use the "unidecode" module that does exactly what we need [16].

3.9.10 [VI.j] Standardize strings by pattern

This feature consists in being able to homogenize the representation pattern of strings that use different formats for elements of the same type (i.e., phone numbers, dates).

In OpenRefine we can accomplish this result similarly to what has been described for changing the date format (section 3.9.1). We use regular expressions to extract the values from the string and then recombine them in a custom format.

3.9.11 [VI.k] Standardize values in clusters

This is an important operation in order to fix inconsistencies in data that may represent the same thing but that it is stored in different ways: it is similar but not an exact match, it is an alternative representation of the same thing. These inconsistencies are produced by misspellings, input errors, and non-standardized value formatting. If we are working with data that comes from public surveys or if we are aggregating data from different sources, then these types of inconsistencies are frequent.

An example of values that needs standardization is the following: "PeAr, apple, pear, aplpe, pare". Once that this set has been standardized the result may be similar to "pear, apple, pear, apple, pear".

OpenRefine has this feature, it is called "Clustering" and it is one of the most powerful features of this software. When applied to a column, it works at the syntactic level (character composition of the cell) using what is called "fuzzy matching" to determine if cells look similar enough to be possible matches. Once applied it provides a list of the detected clusters and it lets you choose what to do next (ignore, merge, inspect, change standardized value). Clustering operation provides a selected number of clustering methods to choose from depending on the situation. The two types of available methods are: key collision and nearest neighbor. Key collision is based on the

idea of creating a key that represents only the most valuable part of the original string, then it creates groups strings that have the same key ("key collision" indeed). These methods are fast, but also tend to be too strict or too lax. Instead, nearest neighbor methods are based on the idea of distance between two strings, then providing a distance threshold we can group together strings which distance is less than that value. Then each method defines distance at his own way. Nearest neighbor methods are much slower than key collision but provide much more refined clusters.

Further investigating clustering goes beyond the scope of this thesis; therefore, we recommend that you consult the literature and the software manual to better understand the differences between the different methods available, in order to find the most suitable for each purpose.

3.10 Results

The following table summarizes the results of the evaluation described in previous pages. I marked for each preparator whether OpenRefine can execute it or not. Furthermore, I inserted the results provided by the research "Data Preparation: A Survey of Commercial Tools" [1] in order to provide a comparison between OpenRefine and other data preparation tools.

Categories	ID	Preparators	Data Preparation Tools							
			OpenRefine	Altair	Paxata	SAP	SAS	Tableau	Talend	Trifacta
Data Discovery	I.a	Locate missing values (nulls)	✔	✔	✔	✔	✔	✔	✔	✔
	I.b	Locate outliers	✔		✔		✔			✔
	I.c	Search by pattern	✔	✔	✔	✔	✔	✔	✔	✔
	I.d	Sort data	✔	✔	✔	✔	✔	✔	✔	✔
Data Validation	II.a	Compare values	✔	✔	✔	✔		✔	✔	✔
	II.b	Check data range	✔	✔	✔	✔		✔	✔	✔
	II.c	Check permitted characters	✔							✔
	II.d	Check column uniqueness	✔	✔	✔	✔		✔	✔	✔
	II.e	Find type-mismatched data	X		✔	✔		✔	✔	✔
	II.f	Find data-mismatched datatypes	X		✔				✔	✔
Data Structuring	III.a	Change column data type	X	✔	✔	✔	✔	✔	✔	✔
	III.b	Delete column	✔	✔	✔	✔	✔	✔	✔	✔
	III.c	Detect & change encoding	✔						✔	✔
	III.d	Pivot / unpivot	✔	✔	✔	✔		✔		✔
	III.e	Rename column	✔	✔	✔	✔	✔	✔	✔	✔
	III.f	Split column	✔	✔	✔	✔	✔	✔	✔	✔
	III.g	Transform by example	X						✔	✔
Data Enrichment	IV.a	Assign semantic data type	X				✔	✔	✔	
	IV.b	Calculate column using expressions	✔	✔	✔	✔	✔	✔	✔	✔
	IV.c	Discover & merge external data	✔	✔	✔	✔			✔	✔
	IV.d	Duplicate column	✔	✔	✔	✔		✔	✔	✔
	IV.e	Generate primary key column	✔			✔				✔
	IV.f	Join & union	✔	✔	✔	✔	✔	✔	✔	✔
	IV.g	Merge columns	✔	✔		✔		✔	✔	✔
	IV.h	Normalize numeric values	X	✔	✔	✔	✔	✔	✔	✔
Data Filtering	V.a	Delete/keep filtered rows	✔	✔	✔	✔	✔	✔	✔	✔
	V.b	Delete empty and invalid rows	✔	✔	✔	✔	✔	✔	✔	✔
	V.c	Extract value parts	✔	✔			✔		✔	✔
	V.d	Filter with regular expressions	✔							✔
Data Cleaning	VI.a	Change date & time format	✔	✔	✔	✔	✔	✔	✔	✔
	VI.b	Change letter case	✔	✔	✔	✔	✔	✔	✔	✔
	VI.c	Change number format	X	✔	✔	✔	✔	✔	✔	✔
	VI.d	Deduplicate data	✔	✔	✔	✔	✔		✔	✔
	VI.e	Delete by pattern	✔	✔	✔		✔	✔	✔	✔
	VI.f	Edit & replace cell data	✔	✔	✔	✔	✔	✔	✔	✔
	VI.g	Fill empty cells	✔	✔	✔				✔	✔
	VI.h	Remove extra whitespace	✔	✔	✔	✔	✔	✔	✔	✔
	VI.i	Remove diacritics	✔			✔				
	VI.j	Standardize strings by pattern	✔		✔	✔	✔	✔	✔	✔
	VI.k	Standardize values in clusters	✔		✔	✔	✔	✔	✔	✔

Figure 3.10.1: Preparator List - Results and Comparison

To better visualize these results, I produced the following graph. For each data preparation tool, it displays the number of preparators available for each of the six categories.

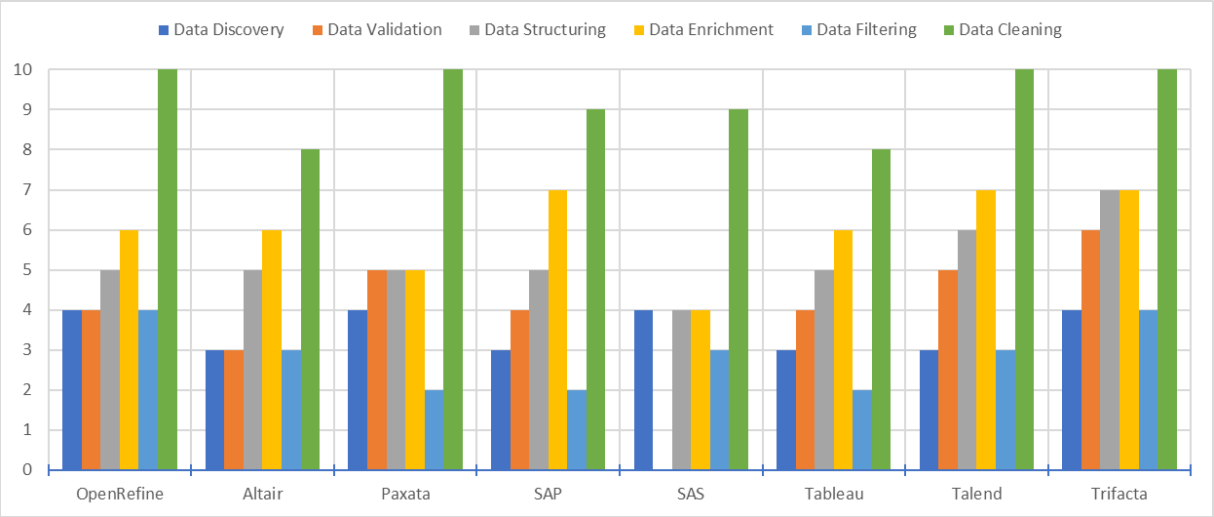


Figure 3.10.2: Preparators - Comparison Graph

It is evident that OpenRefine has what it takes to be used in a data preparation process, holding the comparison with other software dedicated to the same task.

Furthermore, some other consideration must be reported to have a complete evaluation of the software. These considerations are mainly related to the user experience and improvements that can be made:

- It lacks a wide range of data types and the possibility to assign a data type to a column. These two features would enable a better manipulation of the data because we could assign a meaning/context to each column, and maybe even having a set of related operations.
- It offers the opportunity to "star" or "flag" individual rows, in order to be able to conduct a future dedicate inspection or manipulation.
- The most common operations are easily accessible and intuitive to use; however, in some cases useful and relatively common features are not directly implemented but require following an alternative sequence of operations to be realized; since these sequences are mostly standardized,

it would be useful to provide a direct and explicit command to execute such operations.

- Using expressions, we extend the operations that we can perform. For some of them adding shortcuts should be considered. Another useful addition would be implementing an autocompletion mechanism in order to suggest/remind the parameters of the functions.
- The navigation system of the dataset is a bit poor; we can only visualize a defined number of rows and switch only to the previous or next page. Anyway, it seems that in the next release this problem will be fixed [17].
- The decision of referring to the functions that perform the pivot and unpivot operations without using their explicit names makes a bit challenging to find them; this can be considered, since these are common and standard ways to denote these features.
- A transpose operation could be implemented.
- A graphical restyling could be necessary to make it more appealing and modern; in fact, this aspect has remained virtually unchanged since its first release.
- At present, the documentation is a bit scattered between the GitHub page of the software and the dedicated webpage. Therefore, if you need detailed and rapid information you may not find everything in one place. In order to improve the product usability, it could be useful to reorganize the documentation, put it all in one place and enrich it with more examples.

Conclusion

This thesis allows us to have a more complete vision of the software OpenRefine and of its functionalities as a data preparation tool.

The final judgment is composed of the evaluation of the data preparation features and the overall evaluation of the software. As reported in the previous chapter, we have found that the software has checked almost all the features that are required in a data preparation tool, being on par with other data preparation tools. All the basic functionalities are easy to learn and to use, but more experience and coding basics are needed to find solutions that realize the more complex functionalities (in fact, some functionalities do not have a direct implementation, so it is required to perform a series of operations in order to execute them). It supports a lot of input/output formats and a lot of extensions to expand its functionalities.

It must be considered that running locally we can perform data preparation regardless of having an internet connection; furthermore, there is no need to worry about the privacy of the information contained in our datasets.

This thesis also highlighted some critical points, reported in a detailed way in section "3.10 Results" (e.g., it is not possible to set a type for a column, the navigation system is a bit poor, there is no autocompletion mechanism for the expressions, some useful functionalities are not directly implemented). In fact, they do not constitute a limiting factor in the use of the software; however, it is possible to present these suggestions to the community in order to improve the quality of the software.

As a matter of fact, the community puts a lot of effort into improving and fixing the software despite it is a free product. This last bit of information is an important factor for the selection of a software to be used for data preparation by a university, a company, a researcher, etc. If OpenRefine meets our needs for free, why should we pay for another software?

In conclusion, considering the characteristics and the supported functionalities of the analyzed software, net of the relatively marginal highlighted critical points, we can affirm that OpenRefine is a valid free and open-source tool to perform data preparation.

References

[1] M. Hameed and F. Naumann, "Data Preparation: A Survey of Commercial Tools," *ACM SIGMOD Rec.*, vol. 49, no. 3, pp. 18–29, Dec. 2020.

[2] "End User Data Preparation Market Study 2018," 2018. [Online]. Available: https://www.trifacta.com/wp-content/uploads/2018/02/End-User-Data-Preparation-Market-Study-2018.pdf

[3] G. Press, "Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says," *Forbes*, 23-Mar-2016. [Online]. Available: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/

[4] F. Castanedo, "Data Preparation in the Big Data Era," pp. 5, 6, 8, 9 ,10, Aug. 2015.

[5] H. Wickham, "Tidy Data," *J. Stat. Softw.*, vol. 59, no. 1, pp. 1–23, Sep. 2014.

[6] "OpenRefine - GitHub," *GitHub*. [Online]. Available: https://github.com/OpenRefine

[7] "OpenRefine - Website." [Online]. Available: https://openrefine.org/

[8] "OpenRefine - User Manual." [Online]. Available: https://docs.openrefine.org//

[9] "2020 survey results · OpenRefine." [Online]. Available: https://openrefine.org/blog/2020/02/20/2020-survey-results.html

[10] "Kaggle - 120 years of Olympic history: athletes and results." [Online]. Available: https://kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results

[11] "IMDb – data about movies." [Online]. Available: ftp://ftp.fu-berlin.de/pub/misc/movies/database/frozendata/

[12] "UK government web archive - Research..." [Online]. Available: https://webarchive.nationalarchives.gov.uk/+/http://www.bis.gov.uk/assets/biscore/further-education-skills/docs/n/11-708-data-nlss-2009.csv

[13] C. Cunningham, C. Galindolegaria, and G. Graefe, "PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS," in *Proceedings 2004 VLDB Conference*, Elsevier, 2004, pp. 998–1009 [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/B9780120884698500875

[14] Z. Jin, M. R. Anderson, M. Cafarella, and H. V. Jagadish, "Foofah: Transforming Data By Example," in *Proceedings of the 2017 ACM International Conference on Management of Data*, Chicago Illinois USA, 2017, pp. 683–698 [Online]. Available: https://dl.acm.org/doi/10.1145/3035918.3064034

[15] "VIB-Bits Extension." [Online]. Available: https://www.bits.vib.be/index.php/software-overview/openrefine

[16] "Remove Diacritics," *GitHub*. [Online]. Available: https://github.com/OpenRefine/OpenRefine/wiki/Extending-Jython-with-pypi-modules

[17]"OpenRefine - Changes for 3.5," *GitHub*. [Online]. Available: https://github.com/OpenRefine/OpenRefine/wiki/Changes-for-3.5
[18]M. Loukides, "What is data science?," *O'Reilly Media*, 02-Jun-2010. [Online]. Available: https://www.oreilly.com/radar/what-is-data-science/
[19]"What is Data Preparation? (+ How to Make It Easier) - Talend," *Talend Real-Time Open Source Data Integration Software*. [Online]. Available: https://www.talend.com/resources/what-is-data-preparation/
[20]D. Friedland, "A Fresh Look at Data Preparation," *IRI*, 07-Sep-2016. [Online]. Available: https://www.iri.com/blog/business-intelligence/a-fresh-look-at-data-preparation/
[21]J. M. Hellerstein, J. Heer, and S. Kandel, "Self-Service Data Preparation: Research to Practice," pp. 23, 25, 26.
[22]"OpenRefine - Documentation." [Online]. Available: https://openrefine.org/documentation.html
[23] *Wikipedia, the free encyclopedia*. [Online]. Available: https://en.wikipedia.org/
[24]"Trifacta Wrangler Documentation," *Trifacta Documentation*. [Online]. Available: https://docs.trifacta.com/display/SS/Documentation
[25]"Altair Monarch Documentation." [Online]. Available: https://docs.datawatch.com/monarch/
[26]"Talend Data Preparation Help Center." [Online]. Available: https://help.talend.com/
[27]"Clustering in Depth in OpenRefine," *GitHub*. [Online]. Available: https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth

Appendix A

Environment

- Computational characteristics of the machine used for the execution of tests:

  CPU: Intel(R) Core(TM) i7-8565U @ 1.80GHz

  RAM: 16.0 GB

  Operating System: Windows 10 Home 64 bit

  Browser: Microsoft Edge (Version 89.0.774.57)

- OpenRefine:

  Release 3.4.1 (released on September 24, 2020)

  Windows kit with embedded Java