### **A lightweight Ingestion Pipeline:**

<https://excalidraw.com/#room=da74e970da014535b11e,28SMnCZl4KqwIEGHIe6qsQ>

### 

### **SOME DESCRIPTIONS:**

### **1. Understand your data and metadata to be onboarded**

Before starting the mapping process, it is crucial to understand the structure of your metadata and the semantic meaning of each column.

You need to extract and curate the metadata from the dedicated databases at source. The output of this step is metadata sources, cleaned, wrangled and ready to go through the transformation pipeline.

Also, in this step you decide how each piece of data relates to RDF concepts like classes, properties, and entities.

Each csv or json file is describing a dataset, or a resource, or an image or a sample.

Each row in the csv can be mapped to the target properties and target class in the core metadata schema.

### **2. Understand the ontology (dcat)**

An ontology defines the vocabulary (classes, properties, etc.) used to describe your data in RDF.

In our case we use dcat v3 for transformation purposes

We use dcat-ap for evaluation purposes. --> this is a constraint model, and you use it for understanding which fields are mandatory and other constraints.

This step is vital for ensuring interoperability and making your data understandable and reusable by others.

### **3. Define URIs for each row**

Determine what each row in your CSV represents.

The URI acts as a unique identifier for resources in the RDF world.

### **4. Map Columns to Properties**

Each column in the CSV usually corresponds to a property of your primary resources. Map each column to an RDF property defined in your ontology. For instance, a column named "title" might map to a property such as dcat:title in the dcat:Resource class.

### **5. Convert Values**

Transform the values in each cell into RDF literals or resources, depending on their nature. For literal values (e.g., names, descriptions), you can directly use the cell's content. For values that represent relationships or references to other entities, you will need to create or use existing URIs. (linking to controlled vocabularies)

### **6. Use a Mapping Language or Tool**

Several languages and tools can automate the mapping process from CSV to RDF, such as:

* **RML (RDF Mapping Language):** An extension of R2RML for mapping various file formats, including CSV, to RDF.
* **Tarql (Transforming ARbitrary Queries into Linked data):** A command-line tool for mapping CSV to RDF using SPARQL-like templates.
* **OpenRefine:** A powerful tool for working with messy data, including features for converting data to RDF.

### **7. Create RDF Triples**

Using the mappings you have defined, generate RDF triples for each row in your CSV. Each triple consists of a subject (the resource URI), a predicate (the property URI), and an object (the value or another resource URI).

### **8. Validate and Refine**

After converting your data, validate the RDF output to ensure it accurately represents your original CSV data and adheres to the ontology's structure. You may need to refine your mappings or data to correct any issues.

**Use Pyshacl script for validating your metadatagraph.**

**It is a very simple script with two input**

* **Metadatagraph.ttl**
* **Rule.shape.ttl**

**And the output is a report with all the violation and severity messages.**

### **9. Share and Publish your validated metadata graph as FDP**

Once your RDF data is ready, consider how you will share or publish it to make it accessible to your community for instance FDP. This might involve hosting it on a SPARQL endpoint, within a triplestore, or through other data publishing platforms.

**Example:**

Project – PRISM

Input example from the Prisma:

A screenshot of a computer

Description automatically generated

So the metadata set that needs to be curated and wrangled is something like this:

Then the mapping file (informal yet) looks like this:

The metadata graph in ttl looks like this:

Then this metadata graph in ttl needs to be validated against the rules defined in shacl: