GROUP 15

Francisco de Borjan Sánchez González
Alberto Aragón Calvo
Ester Álvarez Garcia
Joan Fernando Moncayo Pozo
ChengJian Zhou

HANDS-ON 1:

Dataset selection & application requirements

What we did

We selected the **Smart City → Environmental Monitoring (Air Quality in Madrid)** domain.

We located the **open CSV dataset** "Calidad del aire Madrid (daily measurements)" from the **Madrid City Council** on the open data **portal: datos.madrid.es**.

We verified the **open license** and the **minimum documentation**.

Example monthly CSV (Sept 2025):

https://datos.madrid.es/FWProjects/egob/Catalogo/MedioAmbiente/CalidadAire/CalidadAireDiaria/Ficheros/2025/202509_calidad_aire.csv

Representative sample used: csv/madrid_air_quality_sample.csv

We analyzed **requirement compliance (R1–R6)**.

We defined the **target application** and its **functional/non-functional requirements**.

We included **screen mockups** and the **navigation structure** in applicationRequirements.html.

What could be improved

Add more sources (e.g., meteorology) and compare stations.

Evaluate temporal coverage bias and data quality (null values, codes).

What we learned

- Practical criteria for choosing linkable datasets (locations, times, stations).

Differences between mandatory requirements (R1–R4) and optional ones (R5–R6).

How to turn a dataset into a product idea with concrete screens.

HANDS-ON 2:

Methodological guidelines for the generation of Linked Data

1. Analysis of the dataset

Linkability: all three include location (coordinates/address), enabling links to neighborhoods/districts or external gazetteers.

Temporal coverage: air quality provides long-term time series.

Air Quality – Daily data since 2001 (Ayto. de Madrid).

URL: Calidad del aire. Datos diarios desde 2001

Schema (CSV yearly): station_code, date, pollutant, value, unit.

BiciMAD – Stations (Ayto./EMT).

URL: Estaciones de Bicimad (CSV)

Schema: station_id, name, address, lat, lon, capacity, status.

Sports facilities – Municipal basic facilities (Ayto. de Madrid).

URL: Instalaciones Deportivas Básicas Municipales

Schema: facility_id, name, address, lat, lon, type.

2. Licensing

Portal license: Condiciones de uso – Datos Abiertos Ayuntamiento de Madrid (compatible con CC BY 4.0-style attribution).

Each dataset page lists this license explicitly.

Implications: attribution required; redistribution and derivative works allowed; commercial use allowed.

3. Resource Naming Strategy

- Base namespace:

  https://example.org/group15/

Collections & resources:

  - Ontology terms:

    https://example.org/group15/def
  - Air quality stations:

    https://example.org/group15/station/air/{stationId}
  - BiciMAD stations:

    https://example.org/group15/station/bike/{stationId}
  - Sports facilities:

    https://example.org/group15/facility/{facilityId}

- Measurements (daily):
  - https://example.org/group15/measurement/{stationId}--{yyyymmdd}--{pollutant}

- **Persistence policy**: URIs are stable; if resources move we will preserve identifiers and use HTTP redirections. Content negotiation will provide HTML (humans) and RDF (machines).

4. Ontology design

We provide a lightweight ontology reusing standard vocabularies:

- Reuse: schema:address, geo:lat, geo:long.

Core classes:

- ex:Station,
- ex:AirQualityStation,
- ex:BicycleStation,
- ex:SportsFacility,
- ex:Measurement.

Object properties:

- ex:hasMeasurement.

Datatype properties:

- ex:stationId,
- ex:observedValue,
- ex:unit,
- ex:observedProperty,
- ex:dateTime.

HANDS-ON 3:

Cleaning and preparation

1. Objective y Scope

- **Objective of the assignment**
  - Import the datasets into OpenRefine.
  - Analyze and fix errors (whitespace, casing, date formats, etc.).
  - Transform columns to facilitate subsequent RDF generation.
- **Scope**
  - The work covers three samples: *air_quality*, *bike_availability*, *traffic_incidents*.
  - All operations are documented in .json files for reproducibility.

2. Workflow in OpenRefine

- Imported CSVs into OpenRefine.
- Trimmed spaces, fixed capitalization and normalized date formats (ISO 8601).
- Coerced numeric fields (pollutant values, coordinates, counts).
- Exported cleaned datasets and operation histories.

3. Common operations applied

Columna	station_name	municipality	datetime
Action:	Trim leading/trailing spaces	Normalize capitalization to Title Case	Convert to **ISO 8601 UTC**
GREL:	*value.trim()*	*value.toTitlecase()*	*value.toDate().toString('yyyy-MM-dd'T'HH:mm:ss'Z'')*
Motivation:	Avoid different URIs/joins caused by invisible spaces.	Standardize values for future reconciliations (e.g., Wikidata/GeoNames) and consistent filtering.	Align timestamps to a standard format for RML/RDFlib/SPARQL and temporal comparisons.

- **Note:** The same three transformations were applied to **air_quality**, **bikes**, and **traffic** to ensure consistency across sources.

HANDS-ON 4:

Transformation with RML/YARRRML

And

Validation with SPARQL

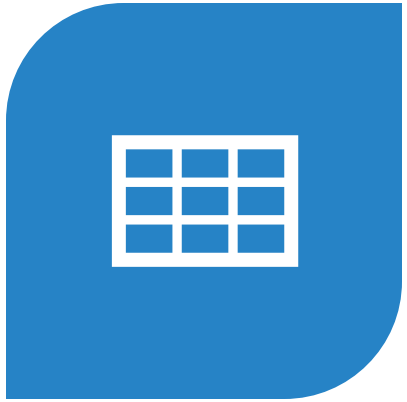
Assignment Brief

Define RML mappings and transform data into RDF.

Materials: templates on GitHub; examples for RMLMapper and Morph-KGC.

Deliverables in /HandsOn/Group15/: mappings/*.rml ,*rdf/*.ttl, rdf/queries.sparql, selfAssessmentHandsOn4.md..

DATASET AND VOCABULARIES



DATASET: GTFS (GENERAL TRANSIT FEED SPEC).
USED: STOPS, ROUTES Y STOP_TIMES.

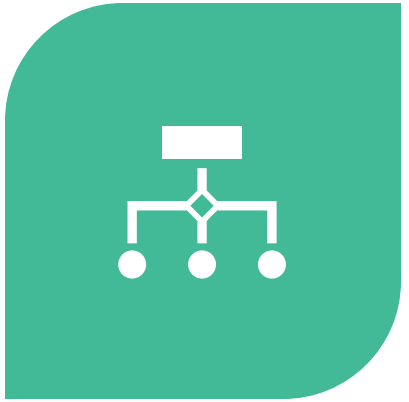
NAMESPACES:

EX:,

GTFS:,

WGS:,

SCHEMA:.



URIS:

EX:STOP/{STOP_ID},

EX:ROUTE/{ROUTE_ID},

EX:STOPTIME/{TRIP_ID}-{STOP_ID}-{ARRIVAL_TIME}

Mapeo YARRRML → RML:

stops, routes and stop_times

```
- stops:
  sources: ["csv/stops-updated.csv~csv"]
  s: ex:stop/$(stop_id)
  po:
    - [ a, gtfs:Stop ]
    - [ schema:name, $(stop_name) ]
    - [ wgs:lat, $(stop_lat) ]
    - [ wgs:long, $(stop_lon) ]

- routes:
  sources: ["csv/routes-updated.csv~csv"]
  s: ex:route/$(route_id)
  po:
    - [ a, gtfs:Route ]
    - [ gtfs:shortName, $(route_short_name) ]
    - [ gtfs:longName, $(route_long_name) ]

- stoptimes:
  sources: [" csv/stop_times-updated.csv~csv "]
  s: ex:stoptime/$(trip_id)-$(stop_id)-$(arrival_time)
  po:
    - [ a, gtfs:StopTime ]
    - p: gtfs:stop
      o:
        mapping: stops
        condition:
          function: equal
          parameters:
            - [ str1, $(stop_id) ]
            - [ str2, $(stop_id)]
    - p: gtfs:route
      o:
        mapping: routes
        condition:
          function: equal
          parameters:
            - [ str1, $(route_id) ]
            - [ str2, $(route_id) ]
```

Mapeo YARRRML → RML: routes

```
routes:
  sources: ["csv/routes-updated.csv~csv"]
  s: ex:route/$(route_id)
  po:
    - [a, gtfs:Route]
    - [gtfs:shortName, $(route_short_name)]
    - [gtfs:longName, $(route_long_name)]
stoptimes:
  sources: ["csv/stop_times-updated.csv~csv"]
  s: ex:stoptime/$(trip_id)-$(stop_id)-$(arrival_time)
  po:
    - [a, gtfs:StopTime]
    - p: gtfs:stop
      o:
        mapping: stops
        condition:
          function: equal
          parameters:
            - [str1, $(stop_id)]
            - [str2, $(stop_id)]
    - p: gtfs:route
      o:
        mapping: routes
        condition:
          function: equal
          parameters:
            - [str1, $(route_id)]
            - [str2, $(route_id)]
```

Mapeo YARRRML → RML: stop_times (joins)

```
- stoptimes:
    sources: ["csv/stop_times-updated.csv~csv"]
    s: ex:stoptime/$(trip_id)-$(stop_id)-
$(arrival_time)
    po:
      - [a, gtfs:StopTime]
      - p: gtfs:stop
        o:
          mapping: stops
          condition:
            function: equal
            parameters:
              - [str1, $(stop_id)]
              - [str2, $(stop_id)]
      - p: gtfs:route
        o:
          mapping: routes
          condition:
            function: equal
            parameters:
              - [str1, $(route_id)]
              - [str2, $(route_id)]
```

Execution with

RMLMapper

- # Asumiendo rmlmapper-all.jar disponible localmente
  java -jar rmlmapper-*-all.jar -m mappings/gtfs-madrid.rml.ttl -o rdf/gtfs-sample.ttl

Morph-KGC (YARRRML)

- # 1) Install
  # pip install morph-kgc

  # 2) Ejecutar usando la config YAML
  python -m morph_kgc -c mappings/gtfs-madrid.yml -o rdf/gtfs-sample.ttl

SPARQL verification queries

```
PREFIX ex:
<http://group15.linked.opendata.es/transport/>
PREFIX gtfs: <http://vocab.gtfs.org/terms#>
PREFIX wgs:
<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX schema: <http://schema.org/>

# Q1: ¿Cuántas paradas y rutas se han generado?
SELECT (COUNT(DISTINCT ?s) AS ?stops)
(COUNT(DISTINCT ?r) AS ?routes) WHERE {
  { ?s a gtfs:Stop } UNION { ?r a gtfs:Route }
}

# Q2: Muestra 5 paradas con nombre y
coordenadas
SELECT ?stop ?name ?lat ?long WHERE {
  ?stop a gtfs:Stop ;
        schema:name ?name ;
        wgs:lat ?lat ;
        wgs:long ?long .
} LIMIT 5
```

SPARQL verification queries

- PREFIX ex: <http://group15.linked.opendata.es/transport/>
PREFIX gtfs: <http://vocab.gtfs.org/terms#>
PREFIX schema: <http://schema.org/>

```
# Q3: Une stop_times con su stop y su route
SELECT ?st ?stopName ?route ?shortName WHERE {
  ?st a gtfs:StopTime ;
      gtfs:stop ?stop ;
      gtfs:route ?route .
  ?stop schema:name ?stopName .
  OPTIONAL { ?route gtfs:shortName ?shortName }
} LIMIT 10

# Q4: Consistencia mínima: todo StopTime debe tener stop y route
ASK {
  ?st a gtfs:StopTime .
  FILTER NOT EXISTS { ?st gtfs:stop ?_s }
  FILTER NOT EXISTS { ?st gtfs:route ?_r }
}
# Esperado: false
```

Best practices and improvements

Tipar literales:

xsd:decimal para wgs:lat/long;

xsd:time para arrival_time si se materializa.

Normalizar URIs y evitar espacios/acentos en plantillas.

Extender mapeos a trips, calendar_dates, shapes si aplica.

Añadir validación con SHACL (shape de Stop, Route, StopTime).

Project structure

- mappings/gtfs-madrid.rml.ttl (mapeos RML)
- mappings/gtfs-madrid.yml (YARRRML opcional)
- rdf/gtfs-sample.ttl (RDF Turtle generado)
- rdf/queries.sparql (consultas de verificación)
- selfAssessmentHandsOn4.md (autoevaluación)
- README.md (instrucciones de ejecución)

Conclusion

RML/YARRRML allows mapping CSV to RDF in a declarative way.

Vocabularies (GTFS, WGS84, Schema.org) and stable URIs are reused.

SPARQL queries are used to verify the quality and consistency of the knowledge graph (KG).

HANDS-ON 5:

RDF Data Linking:

GTFS →

WIKIDATA/

DBPEDIA

1) Goal & Scope

Exercise objective: link instances of our classes to equivalent resources in external datasets (Wikidata/DBpedia), update mappings, and validate links with SPARQL.

Base dataset: GTFS (stops, routes, stop_times).

Strategy: enrich the CSV with linking columns (external URIs), reflect that in RML, and regenerate RDF.

Verification: SPARQL smoke tests to count and check links.

2) Linkable Classes & Target Datasets

Linkable classes:

- Stops → city/administrative area.
- (Future optional) Routes → operator (Wikidata)
- Trips → transport mode (Wikidata/DBpedia).

Chosen external datasets:

- Wikidata (Madrid item: Q2807).
- DBpedia (resource: dbpedia.org/resource/Madrid).

Design criterion:

- start with deterministic links (same municipality) and later move to fine-grained reconciliation (neighborhoods, districts, homonymous stops).

3) OpenRefine Enrichment — Key Operations

Add column city_label with constant value “Madrid” to prepare linking.

From that label, derive:

city_wikidata: http://www.wikidata.org/entity/Q2807

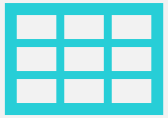
city_dbpedia: http://dbpedia.org/resource/Madrid

Result:

“*-with-links.csv” files with 2 new URI columns per row (+ label) and the OpenRefine operations JSON for reproducibility (“*-with-links.json”).

OpenRefine — Implementation Notes

The flow documents expressions to derive URIs from city_label and records the steps in stops-with-links.json.



For future iterations, add a reconciliation service (e.g., Wikidata Reconciliation Service) when a single deterministic city link is insufficient.

4) RML Mapping Update

Update the mapping to:

Map columns city_wikidata and city_dbpedia as objects of a linking predicate (e.g., owl:sameAs, schema:sameAs, dcterms:spatial, or schema:containedInPlace, depending on your vocabulary).

Keep the original mappings (IDs, lat/lon, stop name, etc.).

Illustrative pattern (inside Stop Triples Map):

```
# Dentro del Triples Map de Stop
rr:predicateObjectMap [
rr:predicate owl:sameAs ;
rr:objectMap [ rml:reference "city_wikidata" ]
] ;
rr:predicateObjectMap [
rr:predicate owl:sameAs ;
rr:objectMap [ rml:reference "city_dbpedia" ]
] .
```

5) RDF Generation

01

Re-run the transformation with the chosen engine (RMLMapper/Morph-KGC) to produce RDF containing external links.

02

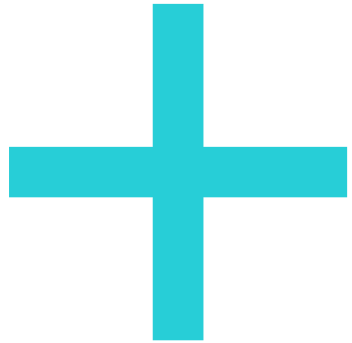
Requested syntax: N-Triples for the transformation; publish in Turtle (*-with-links.ttl) for readability (deliverable under /rdf/).

03

Outcome: a graph where stops now point to Madrid resources in Wikidata and DBpedia.

6) SPARQL Verification — Principle

Use predicate-agnostic queries so you are not tied to owl:sameAs vs schema:sameAs.



Goal: match any predicate whose object is the external URI.

SPARQL — Counts

DBpedia

- -- Stops linked to DBpedia (Madrid)

```
SELECT (COUNT(DISTINCT ?s) AS
?stopsLinkedDBpedia)
WHERE {
  ?s ?p <http://dbpedia.org/resource/Madrid> .
}
```

Wikidata

- -- Stops linked to Wikidata (Madrid)

```
SELECT (COUNT(DISTINCT ?s) AS
?stopsLinkedWikidata)
WHERE {
  ?s ?p <http://www.wikidata.org/entity/Q2807> .
}
```

SPARQL

- **-- Same stop linked to both Wikidata and DBpedia**

```
SELECT (COUNT(DISTINCT ?s) AS ?stopsLinkedBoth)
WHERE {
  ?s ?p1 <http://www.wikidata.org/entity/Q2807> ;
     ?p2 <http://dbpedia.org/resource/Madrid> .
}
```

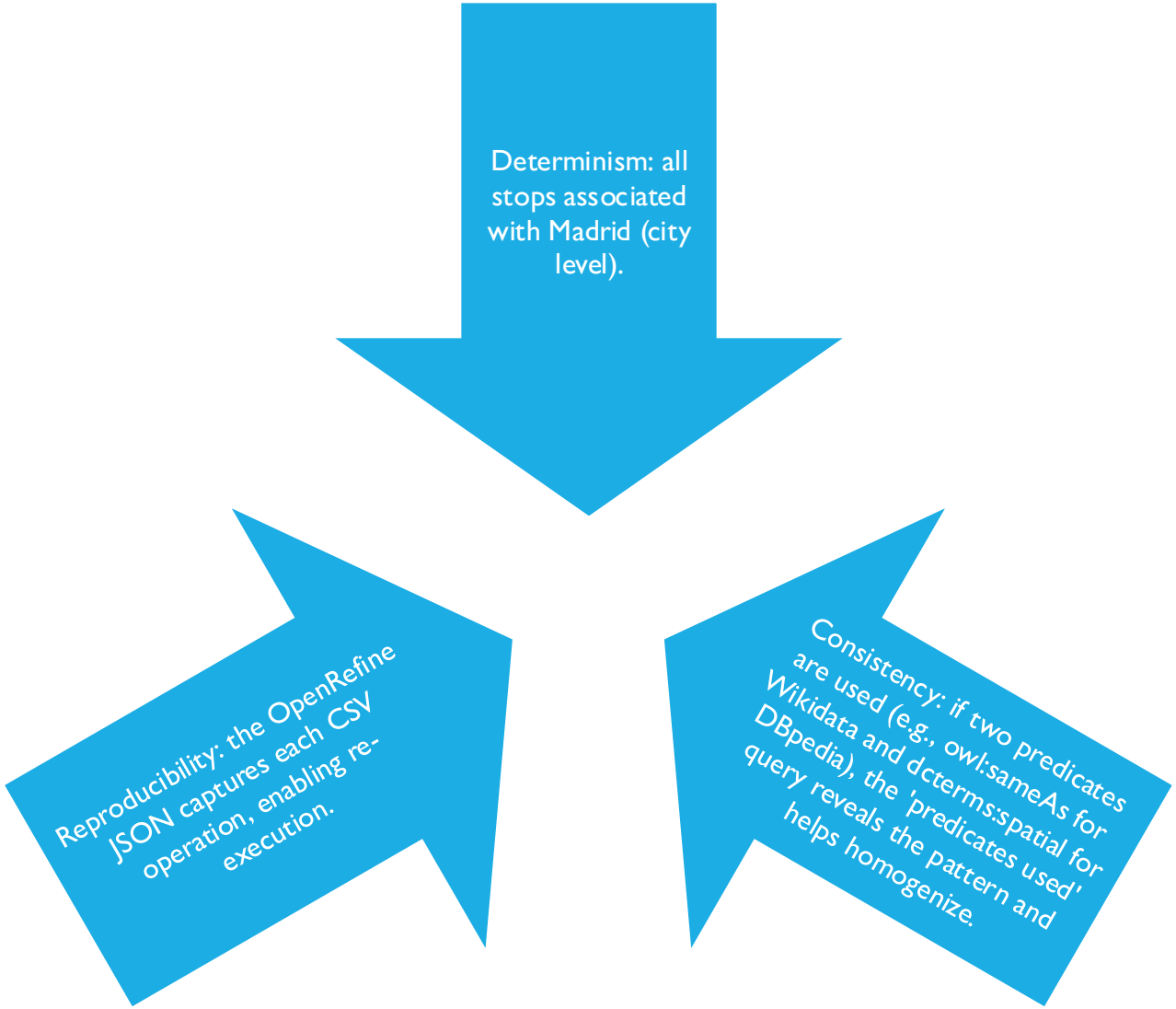
- **-- Predicates actually used for linking (mapping telemetry)**

```
SELECT ?p (COUNT(*) AS ?num)
WHERE {
  ?s ?p <http://www.wikidata.org/entity/Q2807> .
}
GROUP BY ?p
ORDER BY DESC(?num)
```

- **-- Smoke test — sample of created links**

```
SELECT ?stop ?p
WHERE {
  ?stop ?p <http://dbpedia.org/resource/Madrid> .
}
LIMIT 20
```

7) Expected Results & Quality Control



8) Improvements (Next Iteration)

- Refine reconciliation to neighborhoods/districts (Wikidata, GeoNames, OSM) for richer spatial queries.
- Normalize predicates (choose one for equivalence and another for 'located in').
- Validation with SHACL: shapes that ensure every stop links to at least one city_* URI.

9) Deliverables — Where to Find Things

/openrefine/ → stops-with-links.json (applied operations).

/csv/ → stops-with-links.csv, routes-with-links.csv, stop_times-with-links.csv (enriched datasets).

/mappings/ → *-with-links.rml (updated mappings).

/rdf/ → *-with-links.ttl (linked RDF).

/rdf/ → queries-with-links.sparql (verification queries).

10) Lessons Learned

Start with deterministic links to reduce ambiguity and accelerate delivery.

Document the pipeline (OpenRefine JSON + RML) to make auditing and maintenance easier.

Verification queries should be predicate-agnostic to tolerate vocabulary changes.

11) Appendix A — OpenRefine Operations Extract

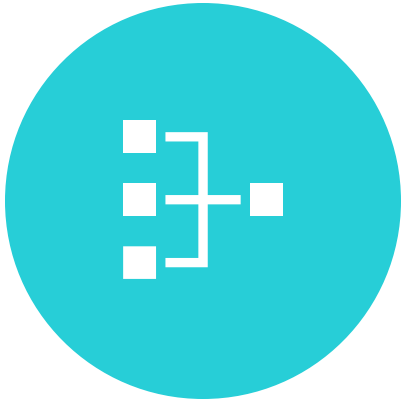
city_label (constant 'Madrid') → linking base.

city_wikidata = http://www.wikidata.org/entity/Q2807

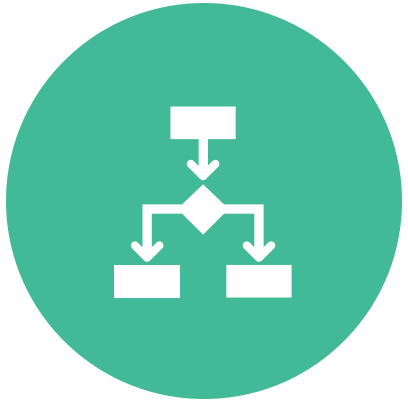
city_dbpedia = http://dbpedia.org/resource/Madrid

12) Appendix B — Suggested Vocabularies

EQUIVALENCE: OWL:SAMEAS / SCHEMA:SAMEAS.



SPATIAL RELATION: DCTERMS:SPATIAL, SCHEMA:CONTAINEDINPLACE.



GTFS (IF USED): HTTP://VOCAB.GTFS.ORG/TERMS#STOP TO TYPE STOPS (OPTIONAL PER YOUR MODEL).