Modeling RDF Data with LA Public Safety Data

By Lauro Cabral, Christopher Vargas, and Sotheanith Sok

Datasets’ Criteria

Why we chose these datasets

Criteria 1: Related Datasets

The relationship between arrest dataset and crime dataset is complex enough to derive many conclusions and facts about crime in LA

Criteria 2: Structured Datasets

Datasets have inherent structures such as person, location, and weapon and we can theorize hierarchical structures from such datasets

Criteria 3: Large Datasets

Datasets are large enough such that a complex RDF file can be deduced.

Dataset 1: Arrest Data from 2020 to Present

URL: https://data.lacity.org/resource/amvf-fr72

Arrest Data from 2020 to Present: Labels

- Report ID
- Report Type
- Arrest Date
- Time
- Area ID
- Area Name
- Reporting District
- Age
- Sex Code
- Descent Code
- Charge Group Code
- Charge Group Description
- Arrest Type Code
- Charge
- Charge Description
- Disposition Description
- Address
- Cross Street
- LAT
- LON
- Location
- Booking Date
- Booking Time
- Booking Location
- Booking Location Code

Dataset 2: Crime Data from 2020 to Present

URL: https://data.lacity.org/resource/2nrs-mtv8

Crime Data from 2020 to Present: Labels

- DR_NO
- Date Rptd
- DATE OCC
- TIME OCC
- AREA
- AREA NAME
- Rpt Dist No
- Part 1-2
- Crm Cd
- Crm Cd Desc
- Mocodes
- Vict Age
- Vict Sex
- Vict Descent
- Premis Cd
- Premis Desc
- Weapon Used Cd
- Weapon Desc
- Status
- Status Desc
- Crm Cd 1
- Crm Cd 2
- Crm Cd 3
- Crm Cd 4
- LOCATION
- Cross Street
- LAT
- LON

Queries

Query 1

Does age affect the likelihood that a person will be involved in a crime?

Query 2

What is the safest time to travel in LA?

Query 3

Based on your gender, how likely are you to be involved with a crime in a given neighborhood?

RDF Schema

Classes

There are 11 classes:

- Report
- Person
- Location
- ArrestReport
- Charge
- Booking
- CrimeReport
- Crime
- Premise
- Weapon
- Status

Report Class

- It is an instance of RDFS: Class
- Properties:
  - “hasID” - XSD: integer
  - “hasPerson” - Person class
  - “hasID” - XSD: integer
  - “hasTime” - XSD: integer
  - “hasDate” - XSD: integer
  - “hasLocation” - Location class

Person Class

- It is an instance of RDFS: Class
- Properties:
  - “hasAge” - XSD: integer
  - “hasSex” - XSD: string
  - “hasDescendent” - XSD: string

Location Class

- It is an instance of RDFS: Class
- Properties:
  - “hasReportingDistrictNumber” - XSD: integer
  - “hasAreaID” - XSD: integer
  - “hasAreaName” - XSD: string
  - “hasAddress” - XSD: string
  - “hasCrossStreet” - XSD: string
  - “hasLatitude” - XSD: double
  - “hasLongitude” - XSD: double

ArrestReport Class

- It is a subclass of Report class
- Properties:
  - “hasDispositionDescription” - XSD: string
  - “hasReportType” - XSD: string
  - “hasArrestType” - XSD: string
  - “hasCharge” - Charge class
  - “hasBooking” - Booking class

Charge Class

- It is an instance of RDFS: Class
- Properties:
  - “hasChargeGroupCode” - XSD: integer
  - “hasChargeGroupDescription” - XSD: string
  - “hasChargeCode” - XSD: integer
  - “hasChargeDescription” - XSD: string

Booking Class

- It is an instance of RDFS: Class
- Properties:
  - “hasBookingDate” - XSD: date
  - “hasBookingTime” - XSD: time
  - “hasBookingLocation” - XSD: string
  - “hasBookingCode” - XSD: integer

CrimeReport Class

- It is a subclass of Report class
- Properties:
  - “hasDateReported” - XSD: date
  - “hasMocodes” - XSD: string
  - “hasCrime” - Crime class
  - “hasStatus” - Status class
  - “hasWeapon” - Weapon class
  - “hasPremise” - Premise class
  - “hasPart1-2 ” - XSD: integer

Crime Class

- It is an instance of RDFS: Class
- Properties:
  - “hasCrimeCommitted” - XSD: string
  - “hasCrimeCommittedDescription” - XSD: string
  - “hasCrimeCommitted1” - XSD: string
  - “hasCrimeCommitted2” - XSD: string
  - “hasCrimeCommitted3” - XSD: string
  - “hasCrimeCommitted4” - XSD: string

Status Class

- It is an instance of RDFS: Class
- Properties:
  - “hasStatusCode” - XSD: integer
  - “hasStatusDescription” - XSD: string

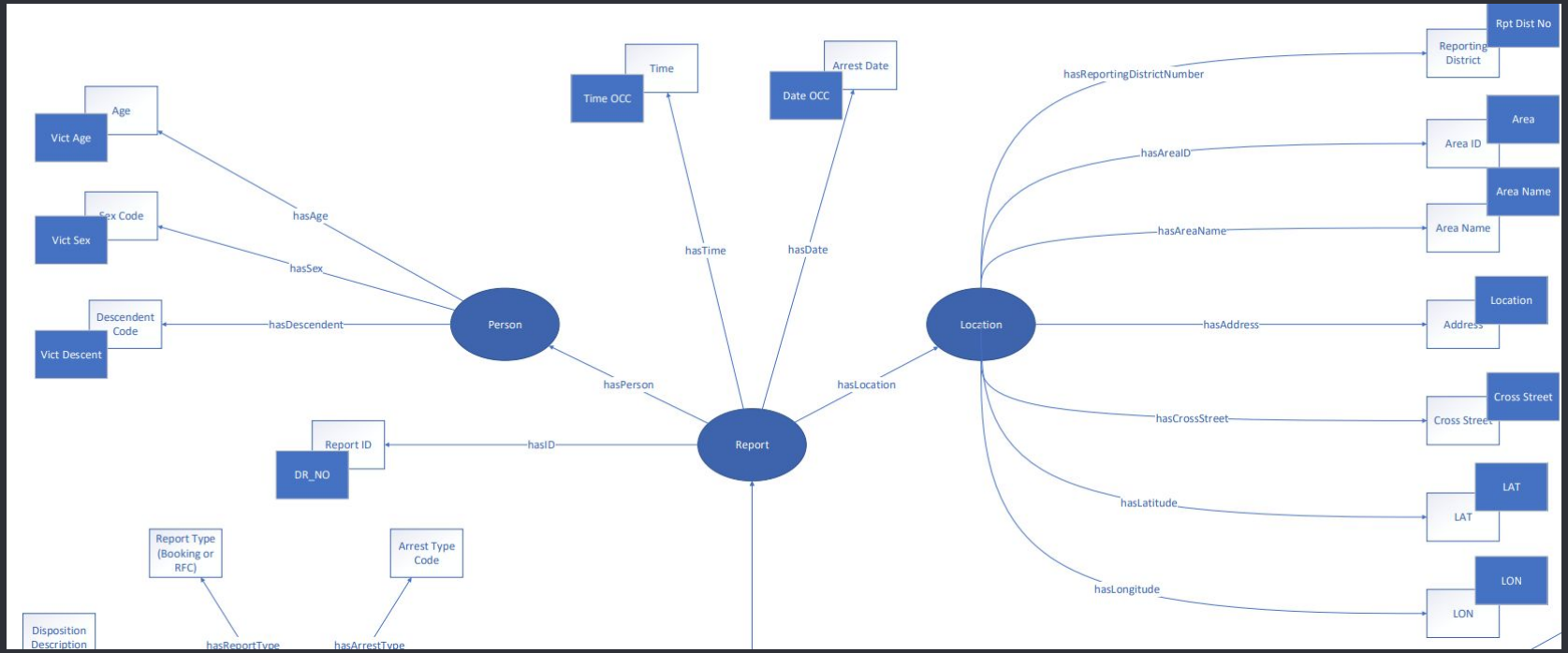
Weapon Class

- It is an instance of RDFS: Class
- Properties:
  - “hasWeaponUsedCode” - XSD: integer
  - “hasWeaponDescription” - XSD: string

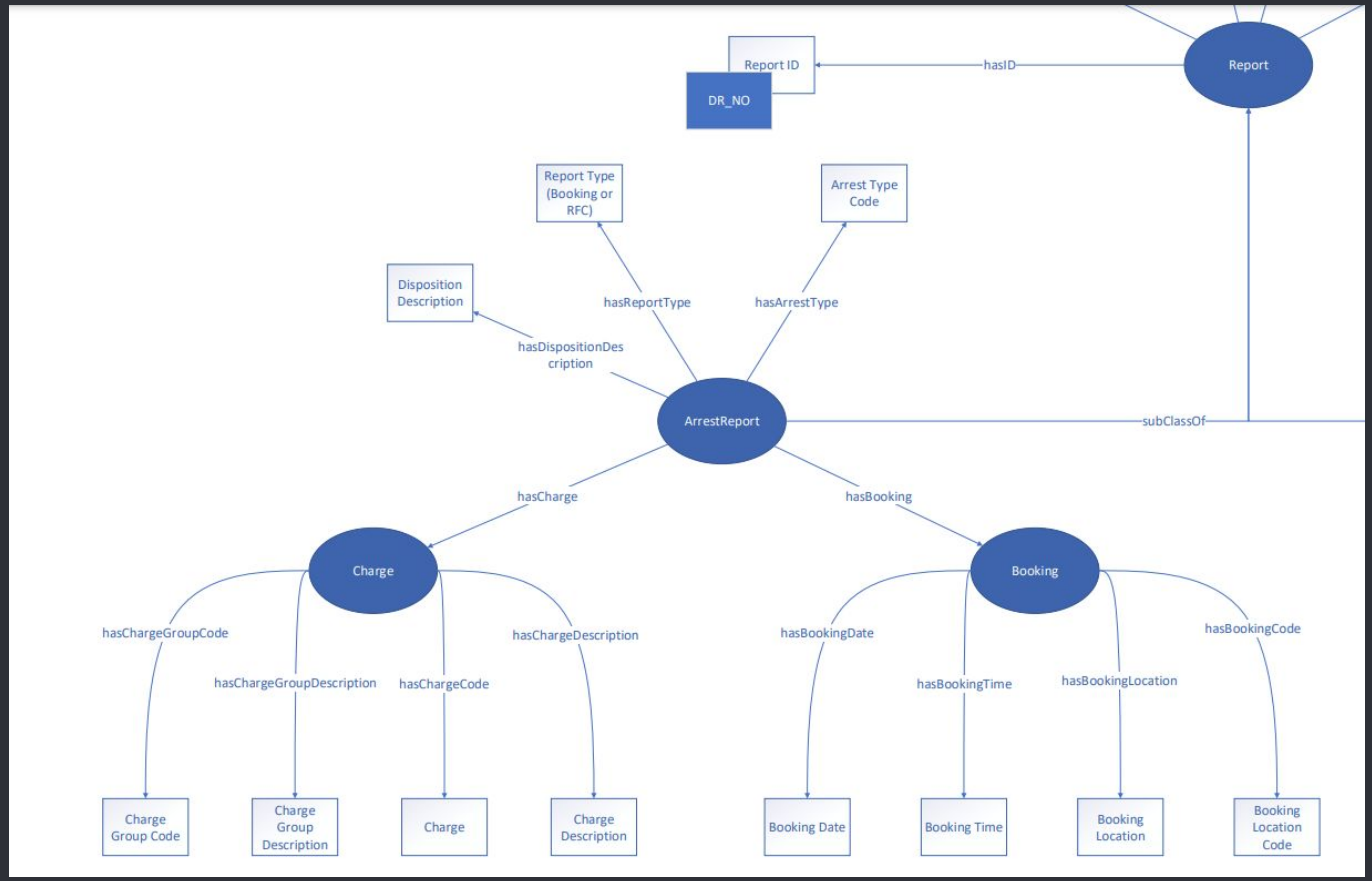
Premise Class

- It is an instance of RDFS: Class
- Properties:
  - “hasPremiseCode” - XSD: integer
  - “hasPremiseDescription” - XSD: string

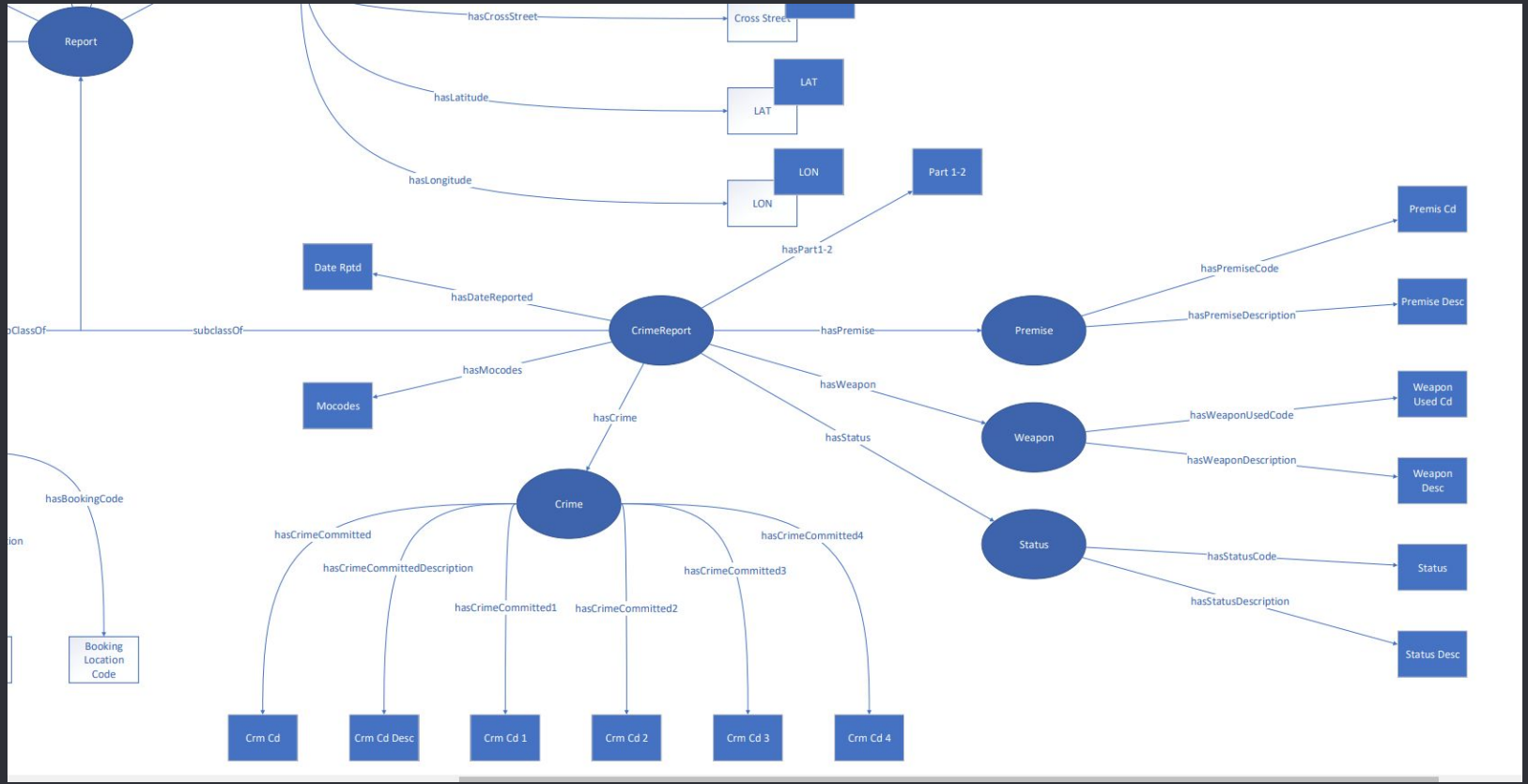
What Does It Look Like - Part 1



What Does It Look Like - Part 2



What Does It Look Like - Part 3



Source Code

Requirements

- Python
- Request
- RdfLib
- Pandas
- Auto-Py-to-Exe

Code Snippet 1: Downloading datasets

```python
def _get_dataset(self, url, max_data_count):
    """Downalod dataset and decode them as csv

    Args:
        url (string): URL to download dataset
        max_data_count (int): maximum number of data to download for a given dataset

    Returns:
        [string]: List of data formatted as CSV
    """

    #Check if url is available
    isAvailable = self._validate_url(url)

    #If url is available
    if isAvailable:

        #Check how many data are available
        available_data_count = int(requests.get(url+".json?$query=SELECT COUNT(*)").json()[0]["COUNT"])

        #Determine how many data should be download based on available_data_count and max_data_count
        nums_data_to_download = max_data_count if (max_data_count< available_data_count) else available_data_count

        #Print out the process
        print("INFO: Downloading %s data from \"%s\"..." %(nums_data_to_download, url))

        #Start downloading data and convert them to csv
        with closing(requests.get(url+".csv?$limit="+str(nums_data_to_download),stream=True)) as response:
            decoded_dataset = [line.decode('utf-8') for line in response.iter_lines()]
            dataset = csv.reader(decoded_dataset, delimiter=',')
            return list(dataset)
```

Code Snippet 2.0: Adding Arrest Reports data to the RDF graph

```python
def _add_arrest_reports_to_graph(self, arrest_reports, graph, namespace):
    """Add arrest reports dataset to the RDF graph

    Args:
        arrest_reports_dataset (string): a CSV contains arrest reports with well-formatted data
        graph (Graph): an RDF graph
        namespace (string): base namespace for all resources

    Returns:
        [Graph]: an RDF graph contains data from the arrest report dataset
    """

    #Print out the process
    print("INFO: Add arrest reports dataset to graph...")

    #Looping through everyone row of arrest reports
    for i in range(1, len(arrest_reports)):

        #Determine how many instances of Report already exist in the graph
        number_report = len(list(graph.subject_objects(predicate=namespace["hasID"])))

        #Add a new instances of ArrestReport class and fill its properties that it inherent from Report class
        graph.add((namespace["Report#" + str(number_report)], RDF.type, namespace["ArrestReport"]))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasID"], Literal(arrest_reports[i][0], datatype=XSD.integer)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasDate"], Literal(arrest_reports[i][2], datatype=XSD.date)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasTime"], Literal(arrest_reports[i][3], datatype=XSD.time)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasReporType"], Literal(arrest_reports[i][1], datatype=XSD.string)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasArrestType"], Literal(arrest_reports[i][12], datatype=XSD.string)))
        graph.add((namespace["Report#" + str(number_report)], namespace["hasDispositionDescription"], Literal(arrest_reports[i][15], datatype=XSD.string)))
```

Code Snippet 2.1: Adding Arrest Reports data to the RDF graph

```
#Add an instance of Person class
#Check if an instance of Person class already exist in the graph
people_age = set(graph.subjects(predicate = namespace["hasAge"], object=Literal(arrest_reports[i][7], datatype=XSD.integer)))
people_sex = set(graph.subjects(predicate = namespace["hasSex"], object=Literal(arrest_reports[i][8], datatype=XSD.string)))
people_decendent = set(graph.subjects(predicate = namespace["hasDescendent"], object=Literal(arrest_reports[i][9], datatype=XSD.string)))
person = list(people_age & people_sex & people_decendent)

# Return the number of instances of Person class
number_person = len(list(graph.subject_objects(predicate=namespace["hasAge"])))

#Create a new instance of Person class
if(len(person) == 0):
    # add to Person
    graph.add((namespace["Person#" + str(number_person)], RDF.type, namespace["Person"]))
    graph.add((namespace["Person#" + str(number_person)], namespace["hasAge"], Literal(arrest_reports[i][7], datatype=XSD.integer)))
    graph.add((namespace["Person#" + str(number_person)], namespace["hasSex"], Literal(arrest_reports[i][8], datatype=XSD.string)))
    graph.add((namespace["Person#" + str(number_person)], namespace["hasDescendent"], Literal(arrest_reports[i][9], datatype=XSD.string)))
    person = namespace["Person#" + str(number_person)]

#Reuse an existing instance of Person class
else:
    person = person[0]

#Add to report
graph.add((namespace["Report#" + str(number_report)], namespace["hasPerson"], person))
graph.add((namespace["Report#" + str(number_report)], namespace["hasLocation"], location))
graph.add((namespace["Report#" + str(number_report)], namespace["hasBooking"], booking))
graph.add((namespace["Report#" + str(number_report)], namespace["hasCharge"], charge))
```

Code Snippet 3: Adding Crime Reports data to the RDF graph

```
# Premise
premiseCodeList = df['PremiseCode']
premiseDescriptionList = df['PremiseDescription']

for i in range(0,len(premiseCodeList)):

    #Check if an instance of Premise class already exist in the graph
    premiseCode = set(graph.subjects(predicate = namespace["hasPremiseCode"], object=Literal(premiseCodeList[i], datatype=XSD.integer)))
    premiseDesc = set(graph.subjects(predicate = namespace["hasPremiseDescription"], object=Literal(premiseDescriptionList[i], datatype=XSD.string)))

    #Give all the triples with premise code and premise description
    premises = list(premiseDesc & premiseCode)

    #Return the number of instances of Premise class
    number_premise = len(list(graph.subject_objects(predicate=namespace["hasPremiseCode"])))

    #Create a new instance of Premise class
    if(len(premises) == 0):
        # add to Location
        graph.add((namespace["Premise#" + str(number_premise)], RDF.type, namespace["Premise"]))
        graph.add((namespace["Premise#" + str(number_premise)], namespace["hasPremiseCode"], Literal(premiseCodeList[i], datatype=XSD.integer)))
        graph.add((namespace["Premise#" + str(number_premise)], namespace["hasPremiseDescription"], Literal(premiseDescriptionList[i], datatype=XSD.string)))
        premise = namespace["Premise#" + str(number_premise)]

    #Reuse an existing instance of Premise class
    else:
        premise = premises[0]

    graph.add((namespace["Report#" + str(i + starting_report_num)], namespace["hasPremise"], premise))
```

Technical Challenges

Challenge 1: Synonym Labels of Data

Problem:

Even though both datasets come from a single source (https://data.lacity.org/), there are multiple labels used to define similar data.

Ex: Report ID (Arrest Reports) == DR_NO (Crime Reports)

Solution:

We examines all data and unify similar data under a single label.

Ex: Report ID (Arrest Reports) == DR_NO (Crime Reports) == ID (RDF Graph)

Challenge 2: Inconsistent Data Format

Problem:

In both datasets, some data do not match the format of XML Schema.

Ex:

XSD: time ⇒ hh:mm:ss

Data from dataset ⇒ “0935”

Solution:

We format all data to match XML Schema format

Ex: “0935” ⇒ 09:35:00

Challenge 3: Missing Data

Problem:

Since datasets are generated by real life events, there are a lot of missing data values.

Solution:

For completeness' sake, we will represent those missing values with black nodes