EPA OntoSearcher: CSV to RDF Conversion

NIOSH Dataset

This document uses EPA's OntoSearcher application to convert multiple CSVs, derived from an Excel workbook of nanomaterial research data provided by NIOSH, into **Resource Description Framework (RDF)**. RDF is a data format which uses unique web addresses, called **Internationalized Resource Identifiers (IRIs)**, to identify pieces of unique information. Associating data with these unique identifiers and publishing that data in RDF format allows for any data regarding the same entity (that shares an IRI) to be interoperable.

EPA OntoSearcher is a prototype application developed at the **Dr. Holly Mortensen** lab at **EPA ORD CPHEA** to expedite the conversion of relational data into RDF. The application provides functions for importing CSV data, importing ontology and RDF data, search algorithm functions to compile a dictionary of IRI's for csv terms, and functions that build RDF from csv data and term-IRI associations.

This document will showcase all of this functionality, as well as how to query RDF data using SPARQL, the RDF query language.

Why are we here?

to answer three questions

- How should we **format our relational data** to make interoperability easier?
- What is **RDF/OWL** and **what utility does it have** for my needs?
- **How can I convert my data** into RDF/OWL (without breaking a sweat)?

In [1]:
```
# import EPA OntoSearcher modules, and other packages
from onto import ontoliter, ontocontext
from csv_importer import load_data
from find import matcher
from onto_api import bioportal_search, unpack_superclass
from onto_api import bioportal_sample, dict_samp, hl_summary
from rdf_print import table_from_file, term_editor, term_lookup
from rdf_print import basic_rdf, relational_rdf_loader
from rdf_print import prlmenode, node_one, node_two, multi_editor
from rdflib import Graph, URIRef, Literal
import pandas as pd
import json
import matplotlib.pyplot as plt
```

Before starting, it is important to note that the NIOSH dataset was modified. The major reason for this is that this application is meant to be performed on CSV files- not on excel workbooks, especially those with multiple sheets. To prepare the NIOSH dataset for processing in this application, the sheets of the original workbook where converted into seperate CSV files.

In [2]:
```
# CSV import
# load in NIOSH csvs
niosh_material_csv = load_data("C:\\Users/wslaught//Documents//csv//niosh_csv//material.csv")
niosh_experiment_csv = load_data("C:\\Users/wslaught//Documents//csv//niosh_csv//experiment.csv")
niosh_assay_csv = load_data("C:\\Users/wslaught//Documents//csv//niosh_csv//assay.csv")

# CSV to dataframe
# load NIOSH csvs into pandas dataframe
niosh_dfs = table_from_file("C:\\Users/wslaught//Documents//csv//niosh_csv//")
```

```
RETURNING 3 OBJECTS:
[0] pandas dataframe of CSV
[1] tail of file path

it is recommended you use the tail of file path as table name if reasonable
RETURNING 3 OBJECTS:
[0] pandas dataframe of CSV
[1] tail of file path

it is recommended you use the tail of file path as table name if reasonable
RETURNING 3 OBJECTS:
[0] pandas dataframe of CSV
[1] tail of file path

it is recommended you use the tail of file path as table name if reasonable
```

We start by loading in a handful of **ontologies** relevant to our area of research- eNanoMapper, Nano Particle Ontology, and a few others. Then we load in our target CSV, in this case these CSVs of data from a sample **NIOSH** nanomaterial research dataset. The OntoSearcher application has many functions that help with finding and importing desired ontologies, but we are loading them from a local json file here.

It is important to note that the csv import functionality, the function **load_data()** below, actually performs a series of text parsing processes to create a list of unique terms from any input csv. It is this list of terms which will be passed to the matching algorithms later to find correct IRIs for all entities in the data.

In [3]:
```
# OWL import
# eNM, NPO, NCIT, ECAO, OBI, SCTO
with open('C:\\Users\\wslaught\\Documents\\db_to_rdf\\demo\\full_onto.json', 'r', encoding='utf-8') as f:
    onto = json.load(f)
```

Now that we have imported target ontologies as well as our csv data, we run a matching function **matcher()** on our CSV terms and our target ontologies. This function applies binary search algorithms to the terms we loaded in from our CSV, and associates those terms with matching Internationalized Resource Identifiers (IRIs) from our ontologies.

In [ ]:
```
# run matcher
materials_match, materials_unmatch = matcher(onto, niosh_material_csv, context=True)
experiment_match, experiment_unmatch = matcher(onto, niosh_experiment_csv, context=True)
assay_match, assay_unmatch = matcher(onto, niosh_assay_csv, context=True)
```

Next we use curation functions from OntoSearcher to manually assert the term-IRI associations for column names- this is extremely important because columns, just like in relational data, define the relationship between a row entry and its data. Correct associations are one of the utmost importance for columns.

In [5]:
```
materials_termedits = [
                       # material columns
                       ('Material_Name', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C93410'),
                       ('Material_Chem_Formula', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C48808'),
                       ('Surface_Charge_mV', 'http://purl.bioontology.org/ontology/npo#NPO_1812'),
                       ('Zeta_Potential_mV', 'http://purl.enanomapper.org/onto/ENM_8000111'),
                       ('Length_nm', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C25334'),
                       ('PP_Diameter_nm', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C25265'),
                      ]

experiment_termedits = [
                        # experiment columns
                        ('ROE', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C83121'),
                        ('Exp_Dose_Mean', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C25488'),
                        ('Exp_Dose_Unit', 'http://purl.obolibrary.org/obo/UO_0000307'),
                       ]

assay_termedits = [
                   # assay columns
                   ('Assay_Organ_System', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C12919'),
                   ('Assay_Endpoint', 'http://www.bioassayontology.org/bao#BAO_0000410'),
                   ('Assay_AssayName', 'http://purl.obolibrary.org/obo/OBI_0000070'),
                   ('Exp_Dose', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C25488'),
                   ('Assay_Mean', 'http://purl.bioontology.org/ontology/npo#NPO_1800'),
                   ('Assay_Data_Unit', 'http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C48553')
                  ]
```

In [6]:
```
# GRAPH CREATION
# create an RDF Graph triplestore
niohGraph = Graph()
```

In [7]:
```
matchfiles = [
    (materials_match[0], materials_unmatch, materials_termedits),
    (experiment_match[0], experiment_unmatch, experiment_termedits),
    (assay_match[0], assay_unmatch, assay_termedits)
]
```

In [ ]:
```
# apply edits, then create graph!
for matched, unmatched, edits in matchfiles:
    # apply edits
    multi_editor(unmatched, matched, edits)
    # add to graph
    basic_rdf(matched, niohGraph)
    relational_rdf_loader(niosh_dfs, matched, niohGraph)
```

We have successfully loaded our RDF and curated some columns of interest- below we will use SPARQL to answer a question with the RDF data we have just created.

We know this is nanomaterial data, transcribed from experiments with different assays testing material toxicological outcomes on animals. We have a loaded table about the material, one about the experiment, and one about the assay.

As an example, we will query data to address the question: **What is the dose-response relationship for the studied nanomaterials and *mus musculus* respiratory outcomes?**

In [9]:
```
niohQuery = niohGraph.query("""
        SELECT DISTINCT  ?nanomaterial ?particle_length_nm ?RoE ?assay ?endpoint ?dose ?result_mean ?res

                # MATERIAL LENGTH
                ?mat_nodes <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C93410> ?nanomaterial ;
                           <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C25334> ?particle_length_nm

                # EXPERIMENTAL EXPOSURE
                ?exp_nodes <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C93410> ?nanomaterial ;
                           <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C83121> ?RoE .

                # ASSAY and RESULTS
                ?nodes <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C93410> ?nanomaterial ;
                       <http://purl.obolibrary.org/obo/OBI_0000070> ?assay ;
                       <http://www.bioassayontology.org/bao#BAO_0000410> ?assay_endpoint ;
                       <http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C25488> ?dose ;
                       <http://purl.bioontology.org/ontology/npo#NPO_1800> ?result_mean .


                ?assay_endpoint rdfs:label ?endpoint .


        }""")


nioh_data = pd.DataFrame(niohQuery.bindings)
nioh_data.columns = nioh_data.columns.str.strip()

# shift column 'Nanomaterial' to first position
first_column = nioh_data.pop('nanomaterial')
nioh_data.insert(0, 'nanomaterial', first_column)
```

In [11]:
```
nioh_data.head(10)
```

Out[11]:

	nanomaterial	RoE	assay	dose	endpoint	particle_length_nm	result_mean
0	mwcnt-24ps	aspiration (pa/oa/other)	bal pmn (%)	40.0	inflammation	5000	11.91
1	mwcnt - ar10	aspiration (pa/oa/other)	bal pmn (%)	2.5	inflammation	1000	2.03
2	mwcnt - ar10	aspiration (pa/oa/other)	bal pmn count	40.0	inflammation	1000	46.667
3	mwcnt - ar10	aspiration (pa/oa/other)	cytokine level	10.0	inflammation	1000	6.9484
4	mwcnt-24t	aspiration (pa/oa/other)	cytokine level	2.5	inflammation	5000	151.42
5	mwcnt-24ps	aspiration (pa/oa/other)	bal pmn (count)	10.0	inflammation	5000	60.952
6	mwcnt-24t	aspiration (pa/oa/other)	bal pmn (%)	2.5	inflammation	5000	0.66
7	mwcnt - ar10	aspiration (pa/oa/other)	bal pmn (%)	10.0	inflammation	1000	2.74
8	mwcnt-24t	aspiration (pa/oa/other)	cytokine level	40.0	inflammation	5000	292.92
9	mwcnt - ar10	aspiration (pa/oa/other)	cytokine level	40.0	inflammation	1000	183.96

In a single query, we have accurately combined the nanomaterial research across the sheets of the excel workbook, using the name of the nanomaterial. It is easy to imagine performing this operation using a csv of data from an extant dataset, though of course, it also shows the utility of RDF/SPARQL for exploring and combining internal data as well.

Let's use the data we gathered to explore our study question.

In [12]:
```
niosh_data['dose'] = pd.to_numeric(niosh_data['dose'])
niosh_data['result_mean'] = pd.to_numeric(niosh_data['result_mean'])
niosh_data['assay'] = niosh_data['assay'] .apply(str)
```

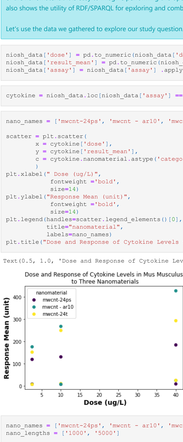
In [13]:
```
cytokine = niosh_data.loc[niosh_data['assay'] == 'cytokine level']
```

In [17]:
```
nano_names = ['mwcnt-24ps', 'mwcnt - ar10', 'mwcnt-24t']

scatter = plt.scatter(
        x = cytokine['dose'],
        y = cytokine['result_mean'],
        c = cytokine.nanomaterial.astype('category').cat.codes

plt.xlabel(" Dose (ug/L)",
           fontweight ='bold',
           size=14)
plt.ylabel("Response Mean (unit)",
           fontweight ='bold',
           size=14)
plt.legend(handles=scatter.legend_elements()[0],
           title="nanomaterial",
           labels=nano_names)
plt.title("Dose and Response of Cytokine Levels in Mus Musculus\n to Three Nanomaterials")
```

Out[17]:
```
Text(0.5, 1.0, "Dose and Response of Cytokine Levels in Mus Musculus\n to Three Nanomaterials")
```



In [19]:
```
nano_names = ['mwcnt-24ps', 'mwcnt - ar10', 'mwcnt-24t']
nano_lengths = ['1000', '5000']

scatter = plt.scatter(
        x = cytokine['dose'],
        y = cytokine['result_mean'],
        c = cytokine.particle_length_nm.astype('category').cat.codes

plt.xlabel(" Dose (ug/L)",
           fontweight ='bold',
           size=14)
plt.ylabel("Response Mean (unit)",
           fontweight ='bold',
           size=14)
plt.legend(handles=scatter.legend_elements()[0],
           title="nanomaterial length (nm)",
           labels=nano_lengths)
plt.title("Dose and Response of Cytokine Levels in Mus Musculus\n by Length of Nanomaterials")
```

Out[19]:
```
Text(0.5, 1.0, "Dose and Response of Cytokine Levels in Mus Musculus\n by Length of Nanomaterials")
```