Week 7 Progress Report - Group 2

In this week we were given task to take the objects that have “instance of” relationship, and try to connect the relationship between them. We decided that the class that we will take will be :

- City
- Taxon
- Food
- Physical Objects
- Brands

The first step to associate the relationship is to gather the objects of the class. We can do it by using query in the *query.wikidata.org* website. In this example, I will query the objects from the “City” Class.

```
SELECT ?s ?p ?o
WHERE{
  ?s wdt:P31 wd:Q515 .
  ?s ?p ?o.
  FILTER(CONTAINS(STR(?o),"wikidata.org/entity"))
 }

```

The variable P31 is the “instance of” relationship, and the variable Q515 is the code of the class City. From this query we can download the json file that consist of all of the query in JSON format.

After acquiring the json format of the queries, we need to change the format into a NTriple format. This step is needed so that it can be readable by the tools name apache-fuseki-jenna which will be explained later.

In [11]:
```
f = open(r'D:\UniversityStuff\FourthTerm\AIDS\GA\City\query_city.nt','w')
fr = open(r'D:\UniversityStuff\FourthTerm\AIDS\GA\City\query_city.txt','w')
```

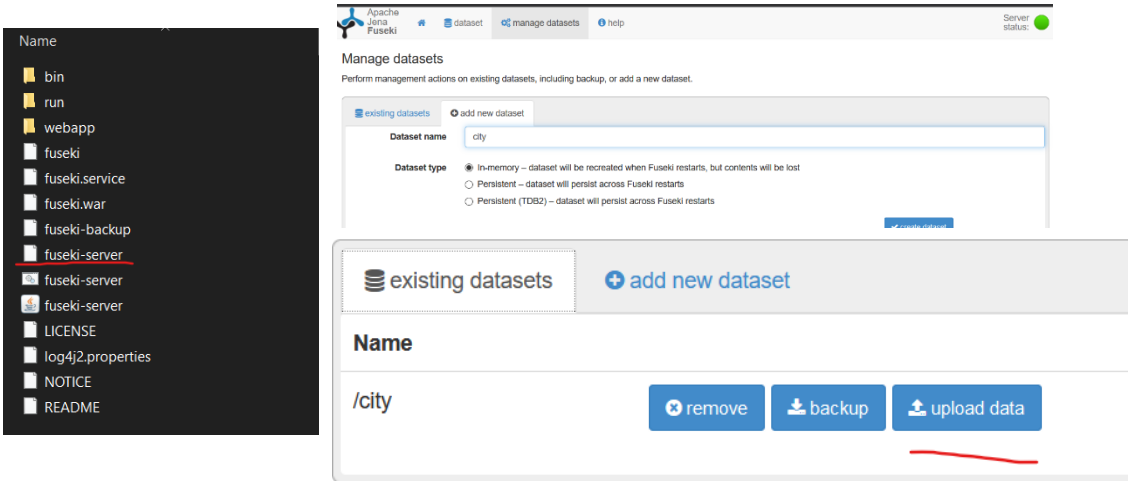
In [14]:
```
for i in tqdm(range(len(data))):
    s = data[i]['s']
    p = data[i]['p']
    o = data[i]['o']
    propRaw = p.split('/')
    prop = propRaw[-1]
    if prop in extProp:
        continue
    if 'http://www.wikidata.org/entity/' not in o:
        o = "\"\"@en ."
        triple = '<'+ s +'> <' + p + '> ' + o
        f.write(triple + "\n")
        fr.write(triple + "\n")
        continue
    triple = '<'+ s +'> <' + p + '> <' + o + '> .'
    f.write(triple + "\n")
    fr.write(triple + "\n")
```

```
100%|██████████| 330227/330227 [00:26<00:00, 12678.94it/s]
```

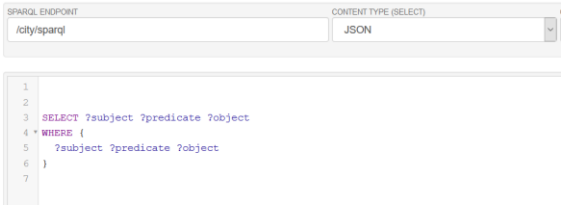
This step will create 2 new file which are query_city.nt and query_city.txt. The first file is needed for the next step while the second is optional where its purpose only to check the output.

Next step we need to download tools called apache-jena-fuseki. From my understanding, it is some kind of SPARQL query tools where we can run it on local. Therefore it is faster to run it on the actual wikidata query website.

Once we download it, we can run it locally from the system terminal and open localhost:3030. Inside, we need to create a new dataset and upload the NTriple file.



This will give us a tool where we can query our NTriple data locally.




```
import numpy as np
import pandas as pd
from SPARQLWrapper import SPARQLWrapper, JSON
from apyori import apriori
from mlxtend.preprocessing import TransactionEncoder
from tqdm import tqdm
```

```
queryString = "SELECT * WHERE { ?s ?p ?o. }"
sparql = SPARQLWrapper("http://localhost:3030/city/query")
sparql.setQuery(queryString)

try :
    ret = sparql.query()
    # ret is a stream with the results in XML, see <http://www.w3.org/TR/rdf-sparql-XMLres/>
except :
    pass
```

We can retrieve the information of the relationship between objects by grouping their properties. We can do this by using the association rules by MLXtend. But first, we need to process the data so that it can be readable using the panda library.

In [13]:
```
sparql.setQuery("""
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
SELECT ?s (COUNT(?prop) AS ?total) {

  SELECT DISTINCT ?s ?prop
    WHERE {
    ?s wdt:P31 wd:Q515 .
    ?s ?prop ?value .
}

} GROUP BY ?s
ORDER BY DESC(?total)
""")
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
```

In [14]:
```
res = []
for results in results["results"]["bindings"]:
    print('%s: %s' % (results["s"]["value"], results["total"]["value"]))
    entity = str(results["s"]["value"]).split('/')
    res.append(entity[-1])
```

In [15]:
```
db = []

for i in range(len(res)):
    query_string = """
    PREFIX wd: <http://www.wikidata.org/entity/>
    SELECT DISTINCT ?s ?prop {
    VALUES ?s {wd:""" + res[i] + """}
    ?s ?prop ?value .
    }
    """

    sparql.setQuery(query_string)
    sparql.setReturnFormat(JSON)
    results_entity = sparql.query().convert()
    propLabel = []
    for results in results_entity["results"]["bindings"]:
#         print('%s: %s' % (results["country"]["value"], results["propLabel"]["value"]))
        propLabel.append(results["prop"]["value"])
#     print('---------------------------')
    db.append(propLabel)
```

In [16]:
```
te = TransactionEncoder()
te_ary = te.fit(db).transform(db)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

That will give the result in this kind of table:

Out[16]:

	http://www.wikidata.org/prop/P10	http://www.wikidata.org/prop/P1001	http://www.wikidata.org/prop/P1012	http://www.wikidata.org/prop/P1019	http://www.wikidata.org/prop/P1028	http://www.wikidata.org/prop/P103	http://www.wikidata.org/prop/P
0	False	False	False	False	False	False	F
1	False	False	False	False	False	False	F
2	False	False	False	False	False	False	F
3	False	False	False	False	False	False	F
4	False	False	False	False	False	False	F
...	...	...	...	...	...	...	
8919	False	False	False	False	False	False	F
8920	False	False	False	False	False	False	F
8921	False	False	False	False	False	False	F
8922	False	False	False	False	False	False	F
8923	False	False	False	False	False	False	F

We can see that the properties are still not defined yet. Therefore, we need to find the properties by accessing the wikidata website.

In [17]:
```
wikidata = SPARQLWrapper("https://query.wikidata.org/sparql")
```

In [18]:
```
propList = df.columns.tolist()
for i in range(len(propList)):
    propList[i]=propList[i].split('/')[-1]
```

In [19]:
```
propLabel = []

for i in tqdm(range(len(propList))):
    query_string = """
    SELECT DISTINCT ?propLabel {
      VALUES ?p {wdt:""" + propList[i] + """}
      SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
      ?prop wikibase:directClaim ?p .
    }
    """

    wikidata.setQuery(query_string)
    wikidata.setReturnFormat(JSON)
    results_prop = wikidata.query().convert()
    for results in results_prop["results"]["bindings"]:
#         print('%s: %s' % (results["country"]["value"], results["propLabel"]["value"]))
        propLabel.append(results["propLabel"]["value"])
#     print('----------------------------')
```

```
100%|██████████| 491/491 [06:58<00:00,  1.17it/s]
```

Which will give the table:

In [20]:
```
df.columns = propLabel
df
```

Out[20]:

	video	applies to jurisdiction	including	web feed URL	donated by	native language	product or material produced	occupation	Human Development Index	population	...	foods traditionally associated	economy of topic	commissioned by	category for the view from the item
0	False	False	False	False	False	False	False	False	True	True	...	False	True	False	False
1	False	False	False	False	False	False	False	False	True	True	...	False	True	False	True
2	False	False	False	False	False	False	False	False	False	True	...	False	True	False	False
3	False	False	False	False	False	False	False	False	False	True	...	False	True	False	False
4	False	False	False	False	False	False	False	False	False	True	...	False	True	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8919	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
8920	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
8921	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
8922	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False
8923	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False

8924 rows × 491 columns

From here we can implement the association rules by MLXtend to automate the relationship between each property.

In [21]:
```
from mlxtend.frequent_patterns import association_rules, fpmax, fpgrowth
```

In [29]:
```
frequent_itemsets = fpgrowth(df, min_support=0.5, use_colnames=True)
```

In [30]:
```
frequent_itemsets
```

Out[30]:

	support	itemsets
0	1.000000	(instance of)
1	0.999888	(instance of)
2	0.946997	(country)
3	0.946773	(country)
4	0.857575	(coordinate location)
...	...	...
634	0.785970	(located in the administrative territorial ent...
635	0.785970	(located in the administrative territorial ent...
636	0.785970	(located in the administrative territorial ent...
637	0.785970	(located in the administrative territorial ent...
638	0.785970	(located in the administrative territorial ent...

639 rows × 2 columns

In [31]:
```
res = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
res
```

Out[31]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(instance of)	(country)	0.999888	0.946773	0.946661	0.946767	0.999994	-0.000006	0.999888
1	(country)	(instance of)	0.946773	0.999888	0.946661	0.999882	0.999994	-0.000006	0.946773
2	(instance of)	(coordinate location)	0.999888	0.857575	0.857463	0.857559	0.999981	-0.000016	0.999888
3	(coordinate location)	(instance of)	0.857575	0.999888	0.857463	0.999869	0.999981	-0.000016	0.857575
4	(country)	(coordinate location)	0.946773	0.857575	0.852308	0.900225	1.049733	0.040380	1.427459
...	...	...	...	...	...	...	...	...	...
485	(country, coordinate location)	(located in the administrative territorial ent...	0.852308	0.843792	0.785970	0.922167	1.092884	0.066799	2.006956
486	(located in the administrative territorial ent...	(instance of, country, coordinate location)	0.843792	0.852196	0.785970	0.931474	1.093028	0.066894	2.156902
487	(instance of)	(located in the administrative territorial ent...	0.999888	0.785970	0.785970	0.786059	1.000112	0.000088	1.000412
488	(country)	(located in the administrative territorial ent...	0.946773	0.786755	0.785970	0.830157	1.055167	0.041092	1.255546
489	(coordinate location)	(located in the administrative territorial ent...	0.857575	0.839422	0.785970	0.916503	1.091827	0.066103	1.923169

490 rows × 9 columns