

Exploring GeoCODES

These pages introduce some work related to exploring the GeoCODES graph and document store.

The document store is an AWS S3 API compliant store leveraging the Minio open source project. It could leverage any such system such as AWS S3, Ceph, GCS, Wasbi, etc. The graph database is an RDF based triples accessed via SPARQL. The document store is synchronized to the graph and acts as the source of truth for the setup.

Both serve different functions and compliment each other. At this time most of these examples are using the document store but that will change. Mostly this is due to me exploring a few concepts such as:

- S3Select calls for data inspection, validation and sub-setting
- Dask based concurrent object access for updates and validation
 - SHACL
 - JSON inspection

Content in Jupyter Book

There are many ways to write content in Jupyter Book. This short section covers a few tips for how to do so.

Markdown Files

Whether you write your book's content in Jupyter Notebooks (.ipynb) or in regular markdown files (.md), you'll write in the same flavor of markdown called **MyST Markdown**.

What is MyST?

MyST stands for "Markedly Structured Text". It is a slight variation on a flavor of markdown called "CommonMark" markdown, with small syntax extensions to allow you to write **roles** and **directives** in the Sphinx ecosystem.

What are roles and directives?

Roles and directives are two of the most powerful tools in Jupyter Book. They are kind of like functions, but written in a markup language. They both serve a similar purpose, but **roles are written in one line**, whereas **directives span many lines**. They both accept different kinds of inputs, and what they do with those inputs depends on the specific role or directive that is being called.

Using a directive

At its simplest, you can insert a directive into your book's content like so:

```
```{mydirectivename}
My directive content
```
```

This will only work if a directive with name **mydirectivename** already exists (which it doesn't). There are many pre-defined directives associated with Jupyter Book. For example, to insert a note box into your content, you can use the following directive:

```
```{note}
Here is a note
```
```

This results in:

Note

Here is a note

In your built book.

For more information on writing directives, see the [MyST documentation](#).

Using a role

Roles are very similar to directives, but they are less-complex and written entirely on one line. You can insert a role into your book's content with this pattern:

```
Some content {rolename}`and here is my role's content!`
```

Again, roles will only work if `rolename` is a valid role's name. For example, the `doc` role can be used to refer to another page in your book. You can refer directly to another page by its relative path. For example, the role syntax `{doc}`intro`` will result in: [Exploring GeoCODES](#).

For more information on writing roles, see the [MyST documentation](#).

Adding a citation

You can also cite references that are stored in a `bibtex` file. For example, the following syntax:

```
{cite}`holdgraf_evidence_2014`
```

 will render like this: [\[HdHPK14\]](#).

Moreover, you can insert a bibliography into your page with this syntax: The `{bibliography}` directive must be used for all the `{cite}` roles to render properly. For example, if the references for your book are stored in `references.bib`, then the bibliography is inserted with:

```
```{bibliography}
```

Resulting in a rendered bibliography that looks like:

[\[HdHPK14\]](#)

Christopher Ramsay Holdgraf, Wendy de Heer, Brian N. Pasley, and Robert T. Knight. Evidence for Predictive Coding in Human Auditory Cortex. In *International Conference on Cognitive Neuroscience*. Brisbane, Australia, Australia, 2014. Frontiers in Neuroscience.

## Executing code in your markdown files

If you'd like to include computational content inside these markdown files, you can use MyST Markdown to define cells that will be executed when your book is built. Jupyter Book uses *jupyter* to do this.

First, add Jupyter metadata to the file. For example, to add Jupyter metadata to this markdown page, run this command:

```
jupyter-book myst init markdown.md
```

Once a markdown file has Jupyter metadata in it, you can add the following directive to run the code at build time:

```
```{code-cell}
print("Here is some code to execute")
```
```

When your book is built, the contents of any `{code-cell}` blocks will be executed with your default Jupyter kernel, and their outputs will be displayed in-line with the rest of your content.

For more information about executing computational content with Jupyter Book, see [The MyST-NB documentation](#).

## Content with notebooks

You can also create content with Jupyter Notebooks. This means that you can include code blocks and their outputs in your book.

### Markdown + notebooks

As it is markdown, you can embed images, HTML, etc into your posts!

MyST 

You can also  $\backslash\text{add\_}\{\text{math}\}\backslash$  and

$\backslash\text{math}^{\{\text{blocks}\}}\backslash$

or

$\backslash\begin\{split\}\begin\{aligned\}\mbox\{mean\}la_{\text\{tex\}}\backslash\backslash\text\{math blocks\}\end\{aligned\}\end\{split\}\backslash$

But make sure you  $\backslash\text{Escape}\backslash$  your  $\backslash\text{dollar}\backslash$  signs  $\backslash$ you want to keep!

### MyST markdown

MyST markdown works in Jupyter Notebooks as well. For more information about MyST markdown, check out [the MyST guide in Jupyter Book](#), or see [the MyST markdown documentation](#).

### Code blocks and outputs

Jupyter Book will also embed your code blocks and output in your book. For example, here's some sample Matplotlib code:

```
from matplotlib import rcParams, cycler
import matplotlib.pyplot as plt
import numpy as np
plt.ion()
```

```
Fixing random state for reproducibility
np.random.seed(19680801)

N = 10
data = [np.logspace(0, 1, 100) + np.random.randn(100) + ii for ii in range(N)]
data = np.array(data).T
cmap = plt.cm.coolwarm
rcParams['axes.prop_cycle'] = cycler(color=cmap(np.linspace(0, 1, N)))

from matplotlib.lines import Line2D
custom_lines = [Line2D([0], [0], color=cmap(0.), lw=4),
 Line2D([0], [0], color=cmap(.5), lw=4),
 Line2D([0], [0], color=cmap(1.), lw=4)]

fig, ax = plt.subplots(figsize=(10, 5))
lines = ax.plot(data)
ax.legend(custom_lines, ['Cold', 'Medium', 'Hot']);
```



There is a lot more that you can do with outputs (such as including interactive outputs) with your book. For more information about this, see [the Jupyter Book documentation](#)

## Data Unit Testing, ranking, clustering

A section on some efforts to leverage the concept of data unit testing for projects. There is also some material below related to document clustering too. Which needs to be resolved.

### About

Some material about what this page is about

- <https://github.com/capitalone/DataProfiler>
- <https://github.com/awsmlabs/deequ>
- <https://github.com/dylan-profiler/visions>
- <https://github.com/pandas-profiling/pandas-profiling>
- <https://sherlock.media.mit.edu/>
- <http://brandonrose.org/clustering>
- <https://pypi.org/project/pytextrank/>

## References

Here is a [reference to the intro](#). Here is a reference to .

# Notebooks

## About

The following notebooks are exploring both document store access but also approaches to semantic search, clustering and semantic similarity.

### Note to the reader

The following are not good notebooks to learn from.. as I am learning too. ;)

## SciKit K-means Clustering

Exploring SciKit-Learn (<https://scikit-learn.org/stable/>) for semantic search. Here I am looking at the k-means approach ([https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py)). Specifically the Mini-Batch K-Means clustering (<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html>).

There are MANY approaches ([https://scikit-learn.org/stable/auto\\_examples/index.html#cluster-examples](https://scikit-learn.org/stable/auto_examples/index.html#cluster-examples)) and it would be nice to get some guidance on what might make a good approach for building a similarity matrix of descriptive abstracts for datasets.

## Gensim

This is an exploration of Gensim as a potential to create the “node set”,  $V$ , results from a semantic search. That would be fed into a graph database and used to start the path searches and or analysis to create the desired results set for an interface.

### Note

I've not been able to get Gensim to produce the final output I expect due to a error from one of the library dependencies. I've tried clean environments and even a clean VM and not been able to get 64 bit linux to work.

This  $V_{\text{semsearch}}$  might be intersected with a  $V_{\text{spatial}}$  and or others to form a node set for the graph. This is essentially a search “preprocessor”. Another potential set might be  $V_{\text{text}}$  that uses a more classical full text index approaches.

## TXTAI

Exploring TXTAI (<https://github.com/neuml/txtai>) as yet another candidate in generating a set of nodes ( $V$ ) that could be fed into a graph as the initial node set. Essentially looking at semantic search for the initial full text index search and then moving on to a graph database (triplestore in my case) for the graph search / analysis portion.

This is the “search broker” concept I've been trying to resolve.

# EarthCube Graph Analytics Exploration

This is the start of learning a bit about leveraging graph analytics to assess the EarthCube graph and explore both the relationships but also look for methods to better search the graph for relevant connections.

## SciKit K-means Clustering

Exploring SciKit-Learn (<https://scikit-learn.org/stable/>) for semantic search. Here I am looking at the k-means approach ([https://scikit-learn.org/stable/auto\\_examples/cluster/plot\\_kmeans\\_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_assumptions.html#sphx-glr-auto-examples-cluster-plot-kmeans-assumptions-py)). Specifically the Mini-Batch K-Means clustering (<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html>).

There are MANY approaches ([https://scikit-learn.org/stable/auto\\_examples/index.html#cluster-examples](https://scikit-learn.org/stable/auto_examples/index.html#cluster-examples)) and it would be nice to get some guidance on what might make a good approach for building a similarity matrix of descriptive abstracts for datasets.

Notes:

Some search concepts are:

- Geolocation: "Gulf of Mexico", "Bay of Fundy", "Georges Bank"
- Parameter: "chlorophyll", "abundance", "dissolved organic carbon"
- Species: "calanus finmarchicus"
- Instrument: "CTD", "bongo net"
- Project: "C-DEBI"

The worry is these by themselves just become "frequency searches". What we want are search phrases that we can pull semantics from.

```
!apt-get install libproj-dev proj-data proj-bin -qq
!apt-get install libgeos-dev -qq
```

Imports and Inits

```
%%capture
!pip install -q PyLD
!pip install -q boto3
!pip install -q minio
!pip install -q rdflib==4.2.2
!pip install -q cython
!pip install -q cartopy
!pip install -q SPARQLWrapper
!pip install -q geopandas
!pip install -q contextily==1.0rc2
!pip install -q rdflib-jsonld==0.5.0
!pip install -q sklearn
```

```

import requests
import json
import rdflib
import pandas as pd
from pandas.io.json import json_normalize
import concurrent.futures
import urllib.request
import dask, boto3
from SPARQLWrapper import SPARQLWrapper, JSON
import numpy as np
import geopandas
import matplotlib.pyplot as plt
import shapely

from sklearn.cluster import MiniBatchKMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA

dbsparql = "http://dbpedia.org/sparql"
okn = "http://graph.openknowledge.network/blazegraph/namespace/samplesearth/sparql"
whoi = "https://lod.bco-dmo.org/sparql"

Pandas options
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)

```

## Gleaner Data

First lets load up some of the data Gleaner has collected. This is just simple data graph objects and not any graphs or other processed products from Gleaner.

```

Set up our S3FileSystem object
import s3fs
oss = s3fs.S3FileSystem(
 anon=True,
 client_kwargs = {"endpoint_url": "https://oss.geodex.org"}
)
oss.ls('gleaner/summoned')

A simple example of grabbing one item...
import json

jld = ""
with
oss.open('gleaner/summoned/opentopo/231f7fa996be8bd5c28b64ed42907b65cca5ee30.jsonld',
'rb') as f:
#print(f.read())
jld = f.read().decode("utf-8", "ignore").replace('\n', ' ')
json = json.loads(jld)

document = json['description']
print(document)

```

```

import json

@dask.delayed()
def read_a_file(fn):
 # or preferably open in text mode and json.load from the file
 with oss.open(fn, 'rb') as f:
 #return json.loads(f.read().replace('\n', ' '))
 return json.loads(f.read().decode("utf-8", "ignore").replace('\n', ' '))

buckets = ['gleaner/summoned/dataucaredu', 'gleaner/summoned/getiedadataorg',
#gleaner/summoned/iris', 'gleaner/summoned/opentopo', 'gleaner/summoned/ssdb',
#gleaner/summoned/wikilinkedearth', 'gleaner/summoned/wwwbco-dmoorg',
#gleaner/summoned/wwwhydroshareorg', 'gleaner/summoned/wwwunavcoorg']

buckets = ['gleaner/summoned/opentopo']

filenames = []

for d in range(len(buckets)):
 print("indexing {}".format(buckets[d]))
 f = oss.ls(buckets[d])
 filenames += f

#filenames = oss.cat('gleaner/summoned/opentopo', recursive=True)
output = [read_a_file(f) for f in filenames]
print(len(filenames))

```

```
indexing gleaner/summoned/opentopo
```

```
654
```

```

%%time

gldf = pd.DataFrame(columns=['name', 'url', "keywords", "description", "object"])

#for key in filenames:

for doc in range(len(output)):
 #for doc in range(10):
 #for key in filenames:
 #if ".jsonld" in key:
 if "/.jsonld" not in filenames[doc] :
 try:
 jld = output[doc].compute()
 except:
 print(filenames[doc])
 print("Doc has bad encoding")

 # TODO Really need to flatten and or frame this
 try:
 desc = jld["description"]
 except:
 desc = "NA"
 continue
 kws = "keywords" #jld["keywords"]
 name = jld["name"]
 url = "NA" #jld["url"]
 object = filenames[doc]

 gldf = gldf.append({'name':name, 'url':url, 'keywords':kws, 'description': desc,
'object': object}, ignore_index=True)

```

```

gleaner/summoned/opentopo/abfac50b855d576239dc12d9c9dfc28837168114.jsonld
Doc has bad encoding

```

```

gldf.info()
gldf.to_parquet('index.parquet.gzip', compression='gzip') # optional save state here
... one master parquet for Geodex?

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 654 entries, 0 to 653
Data columns (total 5 columns):
Column Non-Null Count Dtype
--- ---
0 name 654 non-null object
1 url 654 non-null object
2 keywords 654 non-null object
3 description 654 non-null object
4 object 654 non-null object
dtypes: object(5)
memory usage: 25.7+ KB

```

## Feature extraction

Let's just worry about the description section for now.

```

vec = TfidfVectorizer(stop_words="english")
vec.fit(gldf.description.values)
features = vec.transform(gldf.description.values)

```

## Kmeans clusters

Guess at the number a few times since we don't have a prior idea who many natural clusterings we might expect

```

random_state = 0

cls = MiniBatchKMeans(n_clusters=20, random_state=random_state)
cls.fit(features)

```

```

MiniBatchKMeans(n_clusters=20, random_state=0)

```

```

predict cluster labels for new dataset
cls.predict(features)

to get cluster labels for the dataset used while
training the model (used for models that does not
support prediction on new dataset).
cls.labels_

```




```
array([[18, 9, 18, 18, 18, 5, 18, 12, 18, 12, 15, 16, 0, 0, 15, 18, 15,
 1, 0, 5, 18, 12, 18, 0, 14, 8, 18, 13, 18, 12, 8, 18, 12, 18,
 16, 12, 2, 18, 0, 0, 17, 18, 6, 0, 0, 5, 2, 12, 2, 18, 12,
 15, 2, 17, 9, 11, 0, 12, 9, 1, 2, 0, 18, 2, 5, 12, 13, 15,
 17, 2, 2, 18, 18, 16, 0, 2, 18, 0, 13, 5, 18, 2, 0, 11, 18,
 0, 12, 14, 0, 0, 9, 8, 9, 11, 5, 9, 12, 17, 1, 12, 7, 0,
 9, 5, 2, 12, 13, 18, 18, 12, 2, 0, 0, 18, 18, 0, 0, 18, 18,
 8, 2, 2, 0, 18, 13, 1, 2, 9, 13, 0, 8, 18, 13, 12, 1, 2,
 2, 0, 16, 5, 0, 9, 0, 12, 0, 13, 12, 18, 18, 1, 2, 8, 13,
 1, 9, 0, 0, 2, 13, 15, 4, 0, 2, 12, 2, 18, 9, 1, 13, 5,
 0, 0, 18, 18, 0, 9, 15, 14, 18, 19, 3, 18, 15, 18, 2, 18, 0,
 8, 16, 18, 18, 0, 9, 13, 18, 17, 2, 2, 2, 18, 18, 17, 5, 11,
 18, 18, 18, 18, 13, 2, 18, 18, 18, 18, 18, 19, 0, 18, 5, 2,
 5, 2, 0, 0, 2, 9, 4, 1, 18, 0, 4, 3, 9, 16, 1, 16, 0,
 5, 13, 12, 9, 2, 11, 12, 12, 2, 18, 18, 2, 9, 12, 3, 12, 18,
 2, 18, 7, 18, 0, 3, 0, 2, 0, 5, 9, 1, 2, 16, 3, 18, 13,
 9, 14, 18, 12, 15, 11, 2, 18, 5, 9, 0, 0, 9, 5, 18, 2, 2,
 18, 10, 2, 12, 8, 12, 15, 0, 18, 2, 18, 10, 2, 9, 18, 16, 0,
 16, 8, 5, 11, 11, 2, 5, 12, 3, 18, 16, 5, 2, 0, 18, 8, 8,
 0, 15, 17, 12, 15, 2, 0, 2, 12, 18, 15, 12, 5, 18, 5, 18, 5,
 11, 8, 18, 14, 18, 15, 5, 0, 12, 18, 18, 2, 0, 9, 1, 0, 4,
 17, 17, 0, 5, 18, 2, 12, 5, 17, 2, 18, 15, 4, 12, 12, 12, 3,
 18, 2, 17, 18, 0, 5, 18, 12, 2, 15, 18, 18, 2, 5, 0, 16, 0,
 15, 0, 5, 18, 18, 12, 11, 3, 2, 18, 8, 5, 0, 1, 8, 8, 5,
 2, 0, 2, 18, 18, 12, 18, 12, 12, 4, 1, 1, 15, 8, 18, 8, 5,
 6, 18, 2, 0, 18, 17, 14, 2, 9, 18, 18, 18, 12, 0, 2, 4, 0,
 18, 0, 9, 0, 2, 3, 13, 18, 8, 4, 13, 18, 18, 0, 18, 0, 6,
 16, 13, 18, 7, 2, 17, 0, 5, 17, 2, 6, 1, 2, 18, 13, 9, 18,
 18, 12, 8, 4, 18, 13, 2, 12, 18, 16, 5, 18, 13, 0, 9, 0, 18,
 2, 13, 16, 11, 12, 3, 13, 18, 18, 5, 18, 12, 18, 2, 9, 18, 0,
 2, 18, 9, 0, 18, 19, 18, 18, 0, 19, 18, 16, 18, 1, 7, 12, 2,
 18, 9, 11, 18, 18, 2, 5, 15, 2, 4, 3, 2, 13, 0, 13, 18, 0,
 0, 0, 11, 18, 18, 9, 12, 13, 7, 0, 0, 2, 15, 5, 5, 9, 8,
 0, 13, 18, 15, 18, 12, 18, 7, 2, 0, 18, 18, 12, 4, 3, 16, 0,
 18, 12, 12, 18, 18, 5, 1, 11, 13, 9, 13, 0, 18, 18, 0, 18, 18,
 0, 18, 12, 15, 18, 18, 0, 2, 12, 0, 0, 17, 13, 12, 16, 2, 4,
 13, 12, 18, 18, 18, 18, 18, 18, 11, 12, 11, 11, 2, 12, 2, 12, 12,
 9, 18, 16, 13, 2, 2, 12, 11, 12, 15, 17, 12, 18, 16, 18, 18, 0,
 18, 18, 1, 2, 0, 5, 0, 2], dtype=int32)
```

```
reduce the features to 2D
pca = PCA(n_components=2, random_state=random_state)
reduced_features = pca.fit_transform(features.toarray())

reduce the cluster centers to 2D
reduced_cluster_centers = pca.transform(cls.cluster_centers_)
```

```
plt.scatter(reduced_features[:,0], reduced_features[:,1], c=cls.predict(features))
plt.scatter(reduced_cluster_centers[:, 0], reduced_cluster_centers[:,1], marker='x',
s=150, c='b')
```

```
<matplotlib.collections.PathCollection at 0x7f4cbaa66340>
```

\_build/jupyter\_execute/notebooks/gleaner\_clustering\_17\_1.svg

## Nearest Neighbor testing

```
from sklearn.neighbors import NearestNeighbors
knn = NearestNeighbors(n_neighbors=10, metric='cosine')
knn.fit(features)
```

```
NearestNeighbors(metric='cosine', n_neighbors=10)
```

```
knn.kneighbors(features[0:1], return_distance=False)
```

```
array([[0, 520, 395, 248, 355, 276, 176, 327, 598, 638]])
```

```
knn.kneighbors(features[0:1], return_distance=True)
```

```
(array([[0. , 0.05126516, 0.37921939, 0.4165376 , 0.6984332 ,
 0.87318168, 0.87318168, 0.87318168, 0.87318168, 0.88349159]]),
 array([[0, 520, 395, 248, 355, 276, 176, 327, 598, 638]]))
```

## Search testing

run a few test searches and then plot the first n (4) results

```
input_texts = ["New Zeland lidar data", "California housing", "new madrid seismic
zone"]
input_features = vec.transform(input_texts)

D, N = knn.kneighbors(input_features, n_neighbors=4, return_distance=True)

for input_text, distances, neighbors in zip(input_texts, D, N):
 print("Input text = ", input_text[:200], "\n")
 for dist, neighbor_idx in zip(distances, neighbors):
 print("Distance = ", dist, "Neighbor idx = ", neighbor_idx)
 print(gldf.name[neighbor_idx])
 print(gldf.description[neighbor_idx][:400])
 print("-"*200)
 print("-"*200)
 print()
```

Input text = New Zeland lidar data

Distance = 0.6759095922425897 Neighbor idx = 59

Bay of Plenty, New Zealand 2018-2019

Lidar was captured for BOPLASS Limited by AAM New Zealand between December 2018 and April 2019. The dataset was generated by AAM New Zealand and their subcontractors. The survey area includes Tauranga, Rotorua and Whakatane and the surrounding area. Data management and distribution is by Land Information New Zealand.

Distance = 0.6833093201645597 Neighbor idx = 470

Auckland South, New Zealand 2016-2017

Lidar was captured for Auckland Council by AAM New Zealand between September 2016 through to June 2017. The original dataset was generated by AAM New Zealand and their subcontractors. The survey area covers the southern Auckland suburbs and regions. Data management and distribution is by Land Information New Zealand.

Distance = 0.7069674707157929 Neighbor idx = 125

Marsden Point, Northland, New Zealand 2016

Lidar was captured for Land Information New Zealand by Aerial Surveys in November 2016. The dataset was generated by Aerial Surveys and their subcontractors. The survey area includes Marsden Point and the surrounding area. Data management and distribution is by Land Information New Zealand.

Distance = 0.7311078986510137 Neighbor idx = 134

Canterbury, New Zealand 2018-2019

Lidar was captured for Environment Canterbury Regional Council by Aerial Surveys New Zealand in March 2018 through to May 2019. The dataset was generated by Aerial Surveys New Zealand and their subcontractors. The survey area includes Amuri Plain, Ashburton, Fairlie Foothills, Pegasus, Motunau, Selwyn North, Selwyn South and Waimate. Data management and distribution is by Land Information New Zealand.

Input text = California housing

Distance = 0.7222151549783433 Neighbor idx = 129

Merced, CA: Origin and Evolution of the Mima Mounds

NCALM Seed Project. PI: Sarah Reed, University of California, Berkeley. This lidar survey was conducted on Sunday, September 17, 2006 on a 33 square kilometer polygon northeast of Merced in Merced County, California. These data were collected to investigate the origin and evolution of Mima mounds in California's Central Valley.

Distance = 0.7222151549783433 Neighbor idx = 359

Merced, CA: Origin and Evolution of the Mima Mounds

NCALM Seed Project. PI: Sarah Reed, University of California, Berkeley. This lidar survey was conducted on Sunday, September 17, 2006 on a 33 square kilometer polygon northeast of Merced in Merced County, California. These data were collected to investigate the origin and evolution of Mima mounds in California's Central Valley.

Distance = 0.7804319924949502 Neighbor idx = 314

South Fork Eel River, CA Watershed Morphology

NCALM Project. Mary Power, University of California Berkeley. The survey area consisted of a polygon located 20 kilometers east of Dos Rios, California. The survey took place over eight flights from 6/27/2004 to 6/30/2004.

Distance = 0.7804319924949502 Neighbor idx = 232

South Fork Eel River, CA Watershed Morphology

NCALM Project. Mary Power, University of California Berkeley. The survey area consisted of a polygon located 20 kilometers east of Dos Rios, California. The survey took place over eight flights from 6/27/2004 to 6/30/2004.

Input text = new madrid seismic zone

Distance = 0.6004996071077049 Neighbor idx = 286

Central U.S. ARRA Lidar, New Madrid Seismic Zone

This dataset is intended for researchers interested in active tectonics and earthquake hazards research in the New Madrid Seismic Zone. The target areas were developed and defined by USGS scientists in collaboration with colleagues working in this region. The target areas are: the Blytheville Arch, the Meeman-Shelby lineament, the Reelfoot scarp and the northern New Madrid area. The data collection

Distance = 0.6004996071077049 Neighbor idx = 400

Central U.S. ARRA Lidar, New Madrid Seismic Zone

This dataset is intended for researchers interested in active tectonics and earthquake hazards research in the New Madrid Seismic Zone. The target areas were developed and defined by USGS scientists in collaboration with colleagues working in this region. The target areas are: the Blytheville Arch, the Meeman-Shelby lineament, the Reelfoot scarp and the northern New Madrid area. The data collection

Distance = 0.8295725353504988 Neighbor idx = 487

New Madrid Seismic Zone

The New Madrid Seismic Zone (NMSZ) has been responsible for producing some of the largest intraplate earthquakes on record (Tuttle et al., 2002). Paleoseismologic studies of sand blows and the Reelfoot fault show that earthquakes occurred in the last 4000 years at intervals of approximately 400-600 years (Kelson et al., 1995; Tuttle et al., 2002; Holbrook et al., 2006). The 1811-1812 NMSZ sequence

Distance = 0.8766571303555153 Neighbor idx = 550

Durham, NH: Hyporheic Zone Extent and Exchange in a Coastal Stream

NCALM Seed Project PI: Danna Truslow, University of New Hampshire. The survey area is an irregular polygon approximately 16 km West of Portsmouth, NH and enclosing 42.5 square kilometers. The data were collected to study hyporheic zone extent and exchange in a coastal New Hampshire stream using heat as a tracer.

## Gensim

This is an exploration of Gensim as a potential to create the “node set”, V, results from a semantic search. That would be fed into a graph database and used to start the path searches and or analysis to create the desired results set for an interface.

This V\_semsearch might be intersected with a V\_spatial and or others to form a node set for the graph. This is essentially a search “preprocessor”. Another potential set might be V\_text that usses more classical full text index approaches.

## References

- <https://github.com/topics/document-similarity>
- [https://radimrehurek.com/gensim/auto\\_examples/core/run\\_core\\_concepts.html](https://radimrehurek.com/gensim/auto_examples/core/run_core_concepts.html)

```
%%capture
!pip install -q --upgrade gensim
!pip install -q dask[dataframe] --upgrade
!pip install -q s3fs
!pip install -q boto3
!pip install -q python-Levenshtein
```

```
zsh:1: no matches found: dask[dataframe]
```

ERROR: After October 2020 you may experience errors when installing or updating packages. This is because pip will change the way that it resolves dependency conflicts.

We recommend you use `--use-feature=2020-resolver` to test your packages with the new resolver before it becomes the default.

boto3 1.17.46 requires botocore<1.21.0,>=1.20.46, but you'll have botocore 1.19.52 which is incompatible.

ERROR: After October 2020 you may experience errors when installing or updating packages. This is because pip will change the way that it resolves dependency conflicts.

We recommend you use `--use-feature=2020-resolver` to test your packages with the new resolver before it becomes the default.

aiobotocore 1.2.2 requires botocore<1.19.53,>=1.19.52, but you'll have botocore 1.20.72 which is incompatible.

```
import pprint
import spacy
from spacy import displacy
import pandas as pd
import dask, boto3
import dask.dataframe as dd
```

## Gleaner Data

First lets load up some of the data Gleaner has collected. This is just simple data graph objects and not any graphs or other processed products from Gleaner.

```
Set up our S3FileSystem object
import s3fs
oss = s3fs.S3FileSystem(
 anon=True,
 client_kwargs = {"endpoint_url": "https://oss.geodex.org"}
)
```

## Further examples

```
import json

@dask.delayed()
def read_a_file(fn):
 # or preferably open in text mode and json.load from the file
 with oss.open(fn, 'rb') as f:
 #return json.loads(f.read().replace('\n', ' '))
 return json.loads(f.read().decode("utf-8", "ignore").replace('\n', ' '))

filenames = oss.ls('gleaner/summoned/opentopo')
output = [read_a_file(f) for f in filenames]
```

```

gldf = pd.DataFrame(columns=['name', 'url', "keywords", "description"])

for doc in range(len(output)):
#for doc in range(10):
 try:
 jld = output[doc].compute()
 except:
 print("Doc has bad encoding")

 # TODO Really need to flatten and or frame this

 desc = jld["description"]
 kws = jld["keywords"]
 name = jld["name"]
 url = jld["url"]
 gldf = gldf.append({'name':name, 'url':url, 'keywords':kws, 'description': desc},
ignore_index=True)

```

Doc has bad encoding

```
gldf.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 654 entries, 0 to 653
Data columns (total 4 columns):
Column Non-Null Count Dtype
--- ---
0 name 654 non-null object
1 url 654 non-null object
2 keywords 654 non-null object
3 description 654 non-null object
dtypes: object(4)
memory usage: 20.6+ KB

```

```

import re

document = "Human machine interface for lab abc computer applications"

text_corpus = [
"Human machine interface for lab abc computer applications",
"A survey of user opinion of computer system response time",
"The EPS user interface management system",
"System and human system engineering testing of EPS",
"Relation of user perceived response time to error measurement",
"The generation of random binary unordered trees",
"The intersection graph of paths in trees",
"Graph minors IV Widths of trees and well quasi ordering",
"Graph minors A survey",
]

text_corpus = []

for i in range(len(gldf)):
text_corpus += gldf.at[i, 'description']

for i in range(len(gldf)):
for i in range(10):
 d = gldf.at[i, 'description']
 # d.replace('(', '').replace(')', '').replace('\n', '')
 dp = re.sub(r'[^A-Za-z0-9]+', '', str(d))
 text_corpus.append(str(dp))

 # if not "http" in d:
 # if not "(" in d:
 # if not "<" in d:
 # text_corpus.append(str(d))

```

```

for x in range(len(text_corpus)):
print(text_corpus[x])

```

```

Create a set of frequent words
stoplist = set('for a of the and to in'.split(' '))
Lowercase each document, split it by white space and filter out stopwords
texts = [[word for word in document.lower().split() if word not in stoplist]
 for document in text_corpus]

Count word frequencies
from collections import defaultdict
frequency = defaultdict(int)
for text in texts:
 for token in text:
 frequency[token] += 1

Only keep words that appear more than once
processed_corpus = [[token for token in text if frequency[token] > 1] for text in texts]
pprint.pprint(processed_corpus)

```

```

from gensim import corpora

dictionary = corpora.Dictionary(processed_corpus)
print(dictionary)

```

```

Dictionary(4443 unique tokens: ['2010', '2016066280', 'aa', 'affonso',
'airborne']...)

```

```

pprint.pprint(dictionary.token2id)

```

```

Side demo
new_doc = "Human computer interaction"
new_vec = dictionary.doc2bow(new_doc.lower().split())
print(new_vec)

```

```

[(1304, 1)]

```

```

bow_corpus = [dictionary.doc2bow(text) for text in processed_corpus]
pprint.pprint(bow_corpus)

```

```

from gensim import models

train the model
tfidf = models.TfidfModel(bow_corpus)

transform the "system minors" string
words = "system minors".lower().split()
print(tfidf[dictionary.doc2bow(words)])

```

```

[(212, 1.0)]

```

```

from gensim import similarities

index = similarities.SparseMatrixSimilarity(tfidf[bow_corpus], num_features=12)

```

```

query_document = 'Airborne Laser Mapping'.split()
query_bow = dictionary.doc2bow(query_document)
sims = index[tfidf[query_bow]]
print(list(enumerate(sims)))

```

```

for document_number, score in sorted(enumerate(sims), key=lambda x: x[1],
reverse=True):
 print(document_number, score)

```

```

NameError Traceback (most recent call last)
<ipython-input-1-7663932f4991> in <module>()
----> 1 for document_number, score in sorted(enumerate(sims), key=lambda x: x[1],
reverse=True):
 2 print(document_number, score)

NameError: name 'sims' is not defined

```

## Gleaner & txtai

### About

Exploring TXTAI (<https://github.com/neuml/txtai>) as yet another candidate in generating a set of nodes (V) that could be fed into a graph as the initial node set. Essentially looking at semantic search for the initial full text index search and then moving on to a graph database (triplestore in my case) for the graph search / analysis portion.

This is the “search broker” concept I’ve been trying to resolve.

### References

- <https://github.com/neuml/txtai>

### Imports and Installs

```
%%capture
!pip install -q git+https://github.com/neuml/txtai
!pip install -q 'fsspec>=0.3.3'
!pip install -q s3fs
!pip install -q boto3
!pip install -q spacy
!pip install -q pyarrow
!pip install -q fastparquet
```

```
^C
ERROR: Operation cancelled by user
```

```
import pprint
import spacy
from spacy import displacy
import pandas as pd
import dask, boto3
import dask.dataframe as dd
from txtai.embeddings import Embeddings

Create embeddings model, backed by sentence-transformers & transformers
embeddings = Embeddings({"method": "transformers", "path": "sentence-transformers/bert-
base-nli-mean-tokens"})
```

### Gleaner Data

First let's load up some of the data Gleaner has collected. This is just simple data graph objects and not any graphs or other processed products from Gleaner.

```
Set up our S3FileSystem object
import s3fs
oss = s3fs.S3FileSystem(
 anon=True,
 client_kwargs = {"endpoint_url": "https://oss.geodex.org"}
)
oss.ls('gleaner/summoned')
```

```
A simple example of grabbing one item...
import json

jld = ""
with
oss.open('gleaner/summoned/opentopo/231f7fa996be8bd5c28b64ed42907b65cca5ee30.jsonld',
'rb') as f:
#print(f.read())
jld = f.read().decode("utf-8", "ignore").replace('\n', ' ')
json = json.loads(jld)

document = json['description']
print(document)
```



```
import json

@dask.delayed()
def read_a_file(fn):
 # or preferably open in text mode and json.load from the file
 with oss.open(fn, 'rb') as f:
 #return json.loads(f.read().replace('\n', ' '))
 return json.loads(f.read().decode("utf-8", "ignore").replace('\n', ' '))

buckets = ['gleaner/summoned/dataucaredu', 'gleaner/summoned/getiedadataorg',
#gleaner/summoned/iris', 'gleaner/summoned/opentopo', 'gleaner/summoned/ssdb',
#gleaner/summoned/wikilinkedearth', 'gleaner/summoned/wwwbco-dmoorg',
#gleaner/summoned/wwwhydroshareorg', 'gleaner/summoned/wwwunavcoorg']

buckets = ['gleaner/summoned/opentopo']

filenames = []

for d in range(len(buckets)):
 print("indexing {}".format(buckets[d]))
 f = oss.ls(buckets[d])
 filenames += f

#filenames = oss.cat('gleaner/summoned/opentopo', recursive=True)
output = [read_a_file(f) for f in filenames]
print(len(filenames))
```

```
indexing gleaner/summoned/opentopo
654
```

```
%%time

gldf = pd.DataFrame(columns=['name', 'url', "keywords", "description", "object"])

#for key in filenames:

for doc in range(len(output)):
#for doc in range(10):
#for key in filenames:
 #if ".jsonld" in key:
 if "/.jsonld" not in filenames[doc] :
 try:
 jld = output[doc].compute()
 except:
 print(filenames[doc])
 print("Doc has bad encoding")

 # TODO Really need to flatten and or frame this
 try:
 desc = jld["description"]
 except:
 desc = "NA"
 continue
 kws = "keywords" #jld["keywords"]
 name = jld["name"]
 url = "NA" #jld["url"]
 object = filenames[doc]

 gldf = gldf.append({'name':name, 'url':url, 'keywords':kws, 'description': desc,
'object': object}, ignore_index=True)
```

```
CPU times: user 12.3 s, sys: 830 ms, total: 13.1 s
Wall time: 59 s
```

```
gldf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 654 entries, 0 to 653
Data columns (total 5 columns):
Column Non-Null Count Dtype
--- -
0 name 654 non-null object
1 url 654 non-null object
2 keywords 654 non-null object
3 description 654 non-null object
4 object 654 non-null object
dtypes: object(5)
memory usage: 25.7+ KB
```

```
gldf.to_parquet('index.parquet.gzip', compression='gzip')
```

## Erratta

```
import re

text_corpus = []

for i in range(len(gldf)):
text_corpus += gldf.at[i, 'description']

for i in range(len(gldf)):
for i in range(10):
d = gldf.at[i, 'description']
d.replace('(', ' ').replace(')', ' ').replace('\n', ' ')
dp = re.sub(r'[^A-Za-z0-9]+', ' ', str(d))
text_corpus.append(str(dp))

if not "http" in d:
if not "(" in d:
if not "<" in d:
text_corpus.append(str(d))

for x in range(len(text_corpus)):
print(text_corpus[x])
```

```
Not needed for textai

Create a set of frequent words
stoplist = set('for a of the and to in'.split(' '))
Lowercase each document, split it by white space and filter out stopwords
texts = [[word for word in document.lower().split() if word not in stoplist]
 for document in text_corpus]

Count word frequencies
from collections import defaultdict
frequency = defaultdict(int)
for text in texts:
 for token in text:
 frequency[token] += 1

Only keep words that appear more than once
processed_corpus = [[token for token in text if frequency[token] > 1] for text in
texts]
pprint.pprint(processed_corpus)
```

## txtai section

```
import numpy as np

Create an index for the list of text_corpus
embeddings.index([(gldf.at[uid, 'object'], text, None) for uid, text in
enumerate(text_corpus)])
embeddings.save("index")
embeddings = Embeddings()
embeddings.load("index")

results = embeddings.search("lidar data ", 3)
for r in results:
 uid = r[0]
 score = r[1]
 print('score:{0} -- {1}\n\n'.format(score, uid)) #text_corpus[uid])
 #print(gldf.at[uid, 'object'])
```

```
score:0.3274398148059845 --
gleaner/summoned/opentopo/04d01beb4b6be2ea15309823124e8029a8547f82.jsonld
```

```
score:0.263794869184494 --
gleaner/summoned/opentopo/008b91b98f92c4b6110bb40ec1dae10240ec28f0.jsonld
```

```
score:0.2295398861169815 --
gleaner/summoned/opentopo/04324ac3558c70ed30fbafe4ad62637fd9d2975b.jsonld
```

## About

This is the start of learning a bit about leveraging graph analytics to assess the EarthCube graph and explore both the relationships but also look for methods to better search the graph for relevant connections.

## Thoughts

It seems we don't care about the triples with literal objects. Only the triples that represent connections between types. Could we use a CONSTRUCT call to remove the unwanted triples? Filter on only IRI to IRI.

## References

- [RDFLib](#)
- [NetworkX](#)
- [iGraph](#)
- [NetworkX link analysis](#)
- <https://faculty.math.illinois.edu/~riverag2/teaching/simcamp16/PageRankwithPython.html>
- <https://docs.dask.org/en/latest/>
- <https://examples.dask.org/bag.html>
- <https://s3fs.readthedocs.io/en/latest/>
- <https://docs.dask.org/en/latest/remote-data-services.html>

## Installs

```
!pip -q install mimesis
!pip -q install minio
!pip -q install s3fs
!pip -q install SPARQLWrapper
!pip -q install boto3
!pip -q install 'fsspec>=0.3.3'
!pip -q install rdflib
!pip -q install rdflib-jsonld
!pip -q install PyLD==2.0.2
!pip -q install networkx
```

ERROR: After October 2020 you may experience errors when installing or updating packages. This is because pip will change the way that it resolves dependency conflicts.

We recommend you use `--use-feature=2020-resolver` to test your packages with the new resolver before it becomes the default.

boto3 1.17.46 requires botocore<1.21.0,>=1.20.46, but you'll have botocore 1.19.52 which is incompatible.

ERROR: After October 2020 you may experience errors when installing or updating packages. This is because pip will change the way that it resolves dependency conflicts.

We recommend you use `--use-feature=2020-resolver` to test your packages with the new resolver before it becomes the default.

aiobotocore 1.2.2 requires botocore<1.19.53,>=1.19.52, but you'll have botocore 1.20.72 which is incompatible.

```
import sys
sys.path.append("../lib/") # path contains python_file.py

import sparqlPandas
```

## Imports

```
import dask, boto3
import dask.dataframe as dd
import pandas as pd
import json

from SPARQLWrapper import SPARQLWrapper, JSON

sweet = "http://cor.esipfed.org/sparql"
dbsparql = "http://dbpedia.org/sparql"
ufokn = "http://graph.ufokn.org/blazegraph/namespace/ufokn-dev/sparql"
```

Code inits

Helper function(s)

The following block is a SPARQL to Pandas feature. You may need to run it to load the function per standard notebook actions.

```
##@title
def get_sparql_dataframe(service, query):
 """
 Helper function to convert SPARQL results into a Pandas data frame.
 """
 sparql = SPARQLWrapper(service)
 sparql.setQuery(query)
 sparql.setReturnFormat(JSON)
 result = sparql.query()

 processed_results = json.load(result.response)
 cols = processed_results['head']['vars']

 out = []
 for row in processed_results['results']['bindings']:
 item = []
 for c in cols:
 item.append(row.get(c, {}).get('value'))
 out.append(item)

 return pd.DataFrame(out, columns=cols)
```

Set up some Pandas Dataframe options

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

Set up the connection to the object store to access the graph objects from

```
import s3fs

oss = s3fs.S3FileSystem(
 anon=True,
 key="",
 secret="",
 client_kwargs = {"endpoint_url": "https://oss.geodex.org"}
)
```

```
Simple command to list objects in the current results bucket prefix
oss.ls('gleaner/results/cdfv3')
```

```
['gleaner/results/cdfv3/bcodmo_graph.nq',
'gleaner/results/cdfv3/cchdo_graph.nq',
'gleaner/results/cdfv3/earthchem_graph.nq',
'gleaner/results/cdfv3/hydroshare_graph.nq',
'gleaner/results/cdfv3/ieda_graph.nq',
'gleaner/results/cdfv3/iris_graph.nq',
'gleaner/results/cdfv3/lipdverse_graph.nq',
'gleaner/results/cdfv3/ocd_graph.nq',
'gleaner/results/cdfv3/opentopo_graph.nq',
'gleaner/results/cdfv3/ssdb_graph.nq',
'gleaner/results/cdfv3/unavco_graph.nq']
```

Pull a graph and load

Let's pull back an example graph and load it up into an RDFLib graph so we can test out a SPARQL call on it.

```
import rdflib
import gzip

with oss.open('gleaner/results/cdfv3/opentopo_graph.nq', 'rb') as f:
 #print(f.read())
 file_content = f.read() #.decode("utf-8", "ignore").replace('\n', ' ')

print(file_content)
with gzip.open('./oceanexperts_graph.nq.gz', 'rb') as f:
file_content = f.read()
```

```
g = rdflib.Graph()
parsed = g.parse(data = file_content, format="nquads")
```

## Note

When we start to do the network analysis we don't really care about the links to literal strings. Rather, we want to see connections between various types. More specifically, types connecting to types.

Note, the isIRI filter removes the blank nodes since the rdf:type can point to both IRI and blank nodes.

```
BIND("Nca34627a4b6d4272be7e2d22bab3becd" as ?s)
```

```
prefix schema: <http://schema.org/>
SELECT DISTINCT ?s ?o
WHERE {
 ?s a schema:Dataset.
 ?s ?p ?o .
 ?o a ?type
 FILTER isIRI(?o)
}
LIMIT 1000
```

```
qres = g.query(
 """prefix schema: <https://schema.org/>
 SELECT DISTINCT ?s ?o
 WHERE {
 ?s a schema:Dataset.
 ?s ?p ?o .
 ?o a ?type
 FILTER isIRI(?o)
 }
 LIMIT 1000
 """)

qrdf = pd.DataFrame(columns=['s', 'o'])

for row in qres:
 qrdf = qrdf.append({'s': row[0], 'o': row[1]}, ignore_index=True)
print("%s : %s " % row)
```

```

KeyboardInterrupt Traceback (most recent call last)
<ipython-input-10-1138f4d110c3> in <module>
 13 qrdf = pd.DataFrame(columns=['s', 'o'])
 14
--> 15 for row in qres:
 16 qrdf.append({'s': row[0], 'o': row[1]}, ignore_index=True)
 17 print("%s : %s " % row)

~/local/lib/python3.9/site-packages/rdfLib/query.py in __iter__(self)
 256
 257 if self._genbindings:
--> 258 for b in self._genbindings:
 259 if b: # don't add a result row in case of empty binding
 {}

 260 self._bindings.append(b)

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in
evalSlice(ctx, slice)
 331 i += 1
 332 i = 0
--> 333 for x in res:
 334 i += 1
 335 if slice.length is None:

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in
evalDistinct(ctx, part)
 384
 385 done = set()
--> 386 for x in res:
 387 if x not in done:
 388 yield x

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in <genexpr>
(.0)
 393 res = evalPart(ctx, project.p)
 394
--> 395 return (row.project(project.PV) for row in res)
 396
 397

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in
evalFilter(ctx, part)
 152 def evalFilter(ctx, part):
 153 # TODO: Deal with dict returned from evalPart!
--> 154 for c in evalPart(ctx, part.p):
 155 if _ebv(part.expr, c.forget(ctx, _except=part._vars) if not
part.no_isolated_scope else c):
 156 yield c

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in
evalBGP(ctx, bgp)
 66 continue
 67
--> 68 for x in evalBGP(c, bgp[1:]):
 69 yield x
 70

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in
evalBGP(ctx, bgp)
 66 continue
 67
--> 68 for x in evalBGP(c, bgp[1:]):
 69 yield x
 70

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/evaluate.py in
evalBGP(ctx, bgp)
 43 _s = ctx[s]
 44 _p = ctx[p]
--> 45 _o = ctx[o]
 46
 47 for ss, sp, so in ctx.graph.triples((_s, _p, _o)):

~/local/lib/python3.9/site-packages/rdfLib/plugins/sparql/sparql.py in
__getitem__(self, key)
 282 _load(self.dataset, source)
 283
--> 284 def __getitem__(self, key):
 285 # in SPARQL BNodes are just labels
 286 if not type(key) in (BNode, Variable):

KeyboardInterrupt:

```

```
qrdf.head()
```

|   | s                                                                                                                                                                         | o                                                                                                                                                                         |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | <a href="https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1">https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1</a> | <a href="https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1">https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1</a> |
| 1 | <a href="https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.052018.2444.2">https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.052018.2444.2</a> | <a href="https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.052018.2444.2">https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.052018.2444.2</a> |
| 2 | <a href="https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.122016.2193.6">https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.122016.2193.6</a> | <a href="https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.122016.2193.6">https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.122016.2193.6</a> |
| 3 | <a href="https://portal.opentopography.org/raster?opentopoID=OTSDEM.082019.26912.1">https://portal.opentopography.org/raster?opentopoID=OTSDEM.082019.26912.1</a>         | <a href="https://portal.opentopography.org/raster?opentopoID=OTSDEM.082019.26912.1">https://portal.opentopography.org/raster?opentopoID=OTSDEM.082019.26912.1</a>         |
| 4 | <a href="https://portal.opentopography.org/raster?opentopoID=OTSDEM.012012.26915.1">https://portal.opentopography.org/raster?opentopoID=OTSDEM.012012.26915.1</a>         | <a href="https://portal.opentopography.org/raster?opentopoID=OTSDEM.012012.26915.1">https://portal.opentopography.org/raster?opentopoID=OTSDEM.012012.26915.1</a>         |

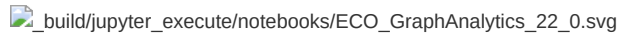
## Convert to NetworkX

Convert this to a networkx graph so we can explore some analytics calls

```
import rdflib
from rdflib.extras.external_graph_libs import rdflib_to_networkx_multidigraph
from rdflib.extras.external_graph_libs import rdflib_to_networkx_digraph
import networkx as nx
import matplotlib.pyplot as plt

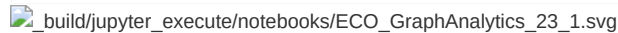
G = rdflib_to_networkx_digraph(parsed)
G=nx.from_pandas_edgelist(qrdf, source='s', target='o')
```

```
Pointless to draw for a graph this big.. just a black ball
nx.draw_circular(G, with_labels = False)
plt.show() # display
```

\_build/jupyter\_execute/notebooks/ECO\_GraphAnalytics\_22\_0.svg

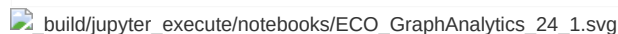
```
plt.hist([v for k,v in nx.degree(G)])
```

```
(array([0., 0., 0., 0., 0., 654., 0., 0., 0., 0.]),
 array([1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5]),
 <BarContainer object of 10 artists>)
```

\_build/jupyter\_execute/notebooks/ECO\_GraphAnalytics\_23\_1.svg

```
plt.hist(nx centrality.betweenness centrality(G).values())
```

```
(array([0., 0., 0., 0., 0., 654., 0., 0., 0., 0.]),
 array([-0.5, -0.4, -0.3, -0.2, -0.1, 0. , 0.1, 0.2, 0.3, 0.4, 0.5]),
 <BarContainer object of 10 artists>)
```

\_build/jupyter\_execute/notebooks/ECO\_GraphAnalytics\_24\_1.svg

```
nx.diameter(G) # found infinite
```

```
nx.cluster.average_clustering(G)
```

```
0.0
```

## Pagerank

Test a page rank call and see if we can load the results into Pandas and sort.

```
import pandas as pd

pr = nx.pagerank(G,alpha=0.9)

prdf = pd.DataFrame.from_dict(pr, orient='index')
prdf.sort_values(by=0,ascending=False, inplace=True,)
prdf.head(10)
```

0

```

https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1 0.001529
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.122019.4326.5 0.001529
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.022019.32637.5 0.001529
https://portal.opentopography.org/raster?opentopoID=OTSDEM.102012.26916.1 0.001529
https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.022012.26910.2 0.001529
https://portal.opentopography.org/raster?opentopoID=OTSDEM.072013.2965.1 0.001529
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.112018.32635.1 0.001529
https://portal.opentopography.org/raster?opentopoID=OTSDEM.062017.26910.1 0.001529
https://portal.opentopography.org/raster?opentopoID=OTSDEM.032013.26910.2 0.001529
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.042020.32632.1 0.001529

```

NetworkX hits

```
hits = nx.hits(G) # can also be done with a nodelist (which is interesting. provide with SPARQL call?)
```

```
hitsdf = pd.DataFrame.from_dict(hits)
hitsdf.head()
```

|   | https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1 | https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.052018.2 |
|---|-------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| 0 | 0.001529                                                                      | 0.001529                                                                 |
| 1 | 0.001529                                                                      | 0.001529                                                                 |

```
bc = nx.betweenness_centrality(G)
```

```
bcd = pd.DataFrame.from_dict(bc, orient='index')
bcd.sort_values(by=0, ascending=False, inplace=True)
bcd.head(10)
```

0

```

https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.072018.6340.1 0.0
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.122019.4326.5 0.0
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.022019.32637.5 0.0
https://portal.opentopography.org/raster?opentopoID=OTSDEM.102012.26916.1 0.0
https://portal.opentopography.org/lidarDataset?opentopoID=OTLAS.022012.26910.2 0.0
https://portal.opentopography.org/raster?opentopoID=OTSDEM.072013.2965.1 0.0
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.112018.32635.1 0.0
https://portal.opentopography.org/raster?opentopoID=OTSDEM.062017.26910.1 0.0
https://portal.opentopography.org/raster?opentopoID=OTSDEM.032013.26910.2 0.0
https://portal.opentopography.org/dataspace/dataset?opentopoID=OTDS.042020.32632.1 0.0

```

By EarthCube Office

© Copyright 2020.