**README**

The current code is tested on Windows platform (Windows 10 Pro) with PyCharm Community 2020.2 and Python 3.8.

Steps to run offline pipeline:

1. Populate the data folder with the following files. The original source of these files are enclosed in the round brackets:
   * Thesaurus.owl (<https://evs.nci.nih.gov/ftp1/NCI_Thesaurus/>)
   * MRREL.RRF (<https://evs.nci.nih.gov/metval>)
   * nci\_code\_cui\_map\_XXXXXX.dat (<https://evs.nci.nih.gov/metval>)
   * cellosaurus.txt (<https://ftp.expasy.org/databases/cellosaurus/>)

The current versions of these files used in ArcheGEO\_offline can be found in:

* cellosaurus.txt (<https://www.mediafire.com/file/v21zem5h0o3om9c/cellosaurus.txt/file>)
* MRREL.RRF (<https://www.mediafire.com/file/so8pqal1mz19cjv/MRREL.RRF/file>)
* nci\_code\_cui\_map\_202008.dat (<https://www.mediafire.com/file/vo6i93rxrxiqhy5/nci_code_cui_map_202008.dat/file>)
* Thesaurus.owl (<https://www.mediafire.com/file/w3tvmiax72mmbsg/Thesaurus.owl/file>)

1. Fill up the relevant credentials inside ../configuration/configuration\_credentials.txt
   * PostgreSQL database user, password, port, database
   * UMLS\_apikey (Refer to <https://documentation.uts.nlm.nih.gov/rest/authentication.html>)
2. Run powershell and key in the following command: *pip install –r requirements.txt*
3. Run offline pipeline using following command in powershell: *python offline.py*

The offline pipeline consists of several portions:

* [NCIThesaurus processor](#NCIThesaurus_Processor) (ncit folder)
* [NCIMetathesaurus processor](#NCIMetathesaurus_Processor) (ncim folder)
* [Cellosaurus\_processor](#Cellosaurus_Processor) (cellosaurus folder)
* [GDS scrapy crawler](#GDS_Scrapy_Crawler) (scrapgds folder)
* [GDS processor](#GDS_Processor) (gds folder)

**Portion 1: NCIThesaurus Processor**

The NCIThesaurus processor is used to extract data from NCIThesaurus (<https://evs.nci.nih.gov/ftp1/NCI_Thesaurus/>). The Thesaurus contains terminology and concepts of biology. There are several releases in a year. The latest version is stored in the root directory (<https://evs.nci.nih.gov/ftp1/NCI_Thesaurus/>) as Thesaurus.OWL.zip and Thesaurus.FLAT.zip files. Other recent versions are also found in the root directory as Thesaurus\_YY.vvv.OWL.zip and Thesaurus\_YY.vvv.FLAT.zip files. Note that YY implies the year (e.g., for year 2021, YY=21) and vvv implies the version number (e.g., 06e and 07d). Older versions can be found in the archive directory (<https://evs.nci.nih.gov/ftp1/NCI_Thesaurus/archive/>). Refer to the ReadMe.txt in the directory to see the structure of the Thesaurus.txt file which is tab-delimited.

Currently on DESKTOP-NVRTB63, the downloaded NCIThesaurus files from <https://evs.nci.nih.gov/ftp1/NCI_Thesaurus/> are stored in C:\Users\hechua\Desktop\NCIThesaurus.

Note that entries in the NCIThesaurus references entries in NCI Metathesaurus. The NCIThesaurus (we use Thesaurus.OWL for extraction) contains concepts of taxonomy, anatomy and disease/abnormality concepts. The NCIThesaurus file contains various tags:

* <NHC0>: NCI Thesaurus ID
* <P106>: Category
* <P108>: Preferred name
* <P310>Retired\_Concept</P310>: Refers to a retired concept. Note that we ignore retired concepts
* <P207>: NCI Metathesaurus ID
* <P331>: NCBI Taxon ID
* <P90>: Synonym

Markers for each NCIThesaurus entry:

if '<owl:Class rdf:about=\"http://ncicb.nci.nih.gov/xml/owl/EVS/Thesaurus.owl#C' in thisline:

Anatomy concepts (extract categories below):

* Extract categories tagged with relevant <P106>

<<P106>Body Part, Organ, or Organ Component</P106>' in thisline  
or '<P106>Body System</P106>' in thisline  
or '<P106>Anatomical Structure</P106>' in thisline  
or '<P106>Body Space or Junction</P106>' in thisline  
or '<P106>Tissue</P106>' in thisline  
or '<P106>Body Location or Region</P106>' in thisline  
or '<P106>Embryonic Structure</P106>' in thisline  
or '<P106>Spatial Concept</P106>'

Taxonomy concepts:

* Extract entries with tagged with NCBI Taxon ID <P331>
* Extract synonyms of the taxonomy <P90>

Disease/abnormality concepts:

* Extract entries with tagged with relevant <P106>

if '<P106>Finding</P106>' in thisline or '<P106>Disease or Syndrome</P106>' in thisline\  
 or '<P106>Sign or Symptom</P106>' in thisline or '<P106>Neoplastic Process</P106>' in thisline\  
 or '<P106>Congenital Abnormality</P106>' in thisline or '<P106>Cell or Molecular Dysfunction</P106>' in thisline\  
 or '<P106>Mental or Behavioral Dysfunction</P106>' in thisline or '<P106>Pathologic Function</P106>' in thisline\  
 or '<P106>Injury or Poisoning</P106>' in thisline or '<P106>Anatomical Abnormality</P106>' in thisline\  
 or '<P106>Physiologic Function</P106>' in thisline or '<P106>Organ or Tissue Function</P106>' in thisline\  
 or '<P106>Acquired Abnormality</P106>' in thisline or '<P106>Environmental Effect of Humans</P106>' in thisline:

* Extract synonyms of the taxonomy <P90>

The main file (main.py) does several things:

1. check\_postgres\_ncithesaurus\_exists(): Check and create ***hogwarts*** database in postgreSQL if it does not exists
2. check\_postgres\_tables\_exist(): Check and create required tables in hogwarts database if they do not exist
3. extract\_thesaurus\_P106\_category(): extracts all unique P106 categories inside the OWL file into a file called thesaurus\_unique\_P106\_category.txt. This is for us identify manually via inspection of the OWL file which categories are relevant for anatomy and for disease/abnormality. For thesaurus\_summary, the manually determined P106 categories are saved in thesaurus\_P106\_category\_extractedfor\_thesaurus\_summary.txt. We expect this to be done only once to set up the structure of the code and would unlikely use it again. Hence, has been commented out.
4. extract\_thesaurus\_summary(): this extracts all NCIThesaurus entries relevant to disease/abnormality. It will output 2 files in the same directory containing the OWL file that was used as input: thesaurus\_summary.txt and thesaurus\_error.txt. The first file contains the necessary data from the OWL file needed to populate the table in ncithesaurus database. For successful population of these tables, some columns in the tables are set as NOT NULL, in which case, the code checks if these fields are found during the extraction. If they are not found, those entries that are missing required fields are saved into thesaurus\_error.txt for further examination (if required).
5. extract\_ thesaurus\_summary\_anatomy(): similar to (4), but for anatomy. The 2 files are thesaurus\_summary\_anatomy.txt and thesaurus\_error\_anatomy.txt.
6. extract\_ thesaurus\_summary\_taxonomy(): similar to (4), but for taxonomy. The 2 files are thesaurus\_summary\_taxonomy.txt and thesaurus\_error\_ taxonomy.txt.
7. populate\_thesaurus\_taxonomy(): use thesaurus\_summary\_taxonomy.txt to populate relevant tables in ncithesaurus database.
8. populate\_thesaurus\_anatomy(): use thesaurus\_summary\_anatomy.txt to populate relevant tables in hogwarts database.
9. populate\_thesaurus(): use thesaurus\_summary.txt to populate relevant tables in hogwarts database.

If there’s a need to refresh the hogwarts database (NCI Thesaurus portion) in postgreSQL, we can either:

* delete the associated tables and rerun 1,2,4-9 OR
* drop cascade all tables and rerun 4-9.
* Remember to change the OWL file to the one you require.

**­­­­­­­­­­­­­­­­­­­­­­­­­­Portion 2: NCIMetathesaurus Processor**

The NCIMetathesaurus processor is used to extract data from NCIMetathesaurus (<https://evs.nci.nih.gov/metval>). For ArcheGEO, we extract the relationships (parent, child, others – anatomy) from Metathesaurus.RRF>META>MRREL.RRF file. In MRREL.RRF file, the parent, child and others relationship we are interested in are tagged as:

* PAR
* CHD
* RO (“Is\_Associated\_Anatomic\_Site\_Of” and “Is\_Primary\_Anatomic\_Site\_Of”)

Reference from nci\_code\_cui\_map\_XXXXXX.dat (in the same META folder containing MRREL.RRF) is required to connect the relationship from NCIMetathesaurus back to NCIThesaurus.

E.g.:

[NCI Thesaurus entry] Breast Carcinoma (NCIt id: C4872, NCIm id: C0678222)

[MRREL.RRF entry] C0678222 | A14276351 | AUI | PAR | C0006142 | A5929841 | AUI | | R114637265 | | MDR | MDR | | | N | |

[nci\_code\_cui\_map\_202008.dct entry] C9335 | C0006142 | Malignant Breast Neoplasm | Malignant neoplasm of breast |

\*\* C9335 is the NCIt ID mapped to C0006142

**IMPORTANT: The MRREL.RRF and nci\_code\_cui\_map\_XXXXXX.dat files are very large and cannot be uploaded to Github. If you wish to run this code, remember to download these two files from** [**https://evs.nci.nih.gov/metval**](https://evs.nci.nih.gov/metval) **and then copy them into the data folder.** Current tested version of nci\_code\_cui\_map\_XXXXXX.dat is nci\_code\_cui\_map\_202008.dat

Folders:

1. **data folder:** should contain MRREL.RRF and nci\_code\_cui\_map\_XXXXXX.dat files
2. **configuration folder:** contains configuration\_database.txt (you can modify your postgres credentials here) and configuration\_file.txt (if you decide to store MRREL.RRF and nci\_code\_cui\_map\_XXXXXX.dat files elsewhere, you can specify it here. Currently code uses a relative path directory assuming that these files are stored in a subdirectory within NCIMetahesaurus folder)
3. **generatedFile folder:** for containing 3 generated files (anatomy\_file.txt, child\_file.txt and parent\_file.txt)

Code:

1. main.py:
   * obtain the relationship and nci2cui map file locations
   * obtain PostgreSQL credentials
   * call ncim\_ncicuimap\_processor [process\_nciCuiMap function] to perform processing of the nci\_code\_cui\_map\_XXXXXX.dat file
   * call ncim\_relationship\_processor [process\_relationship function] to perform processing of the MRREL.RRF file
2. ncim\_ncicuimap\_processor.py:
   * extract data (ncit\_id, ncim\_id and preferred\_name) from nci\_code\_cui\_map\_XXXXXX.dat file and store in PostgreSQL table (public.nciterm)
3. ncim\_relationship\_processor.py:
   * extract data (anatomy, child and parent relationships) from MRREL.RRF file to 3 generated files (anatomy\_file.txt, child\_file.txt and parent\_file.txt)
   * create postgreSQL tables (public.nciterm\_anatomy, public.nciterm\_parent and public.nciterm\_child)
   * get data from anatomy\_file.txt and store in PostgreSQL table (public.nciterm\_anatomy)
   * get data from parent\_file.txt and store in PostgreSQL table (public.nciterm\_parent)
   * get data from child\_file.txt and store in PostgreSQL table (public.nciterm\_child)

If there’s a need to refresh these tables in postgres:

* update the MRREL.RRF and nci\_code\_cui\_map\_XXXXXX.dat files to the one you require.
* delete the public.nciterm\_anatomy, public.nciterm\_parent and public.nciterm\_child
* run NCIMetathesaurus project

**­­­­­­­­­­­­­­­­­­­­­­­­­­****Portion 3: Cellosaurus Processor**

The cellosaurus processor is used to extract the accession number (AC), synonyms (SY), taxonomy (OX) and disease (DI) of the cellosaurus data file (<https://ftp.expasy.org/databases/cellosaurus/>, we use the .txt format file), save them into a file (cellosaurus\_details.tx) and then it uses this file to populate tables in postgres DB. Note that we are only interested in entries where OX references NCBI\_TaxID and DI reference NCIt IDs.

The main file (main.py) does several things:

1. preprocess\_Cellosaurus(): Preprocesses the original cellosaurus.txt file to only extract entries containing DI that references NCIt IDs that are existing in the NCI Thesaurus in postgres DB. Note that in this preprocessing step, the OX field is also filtered for references that points to NCBI\_TaxID. The preprocessed data is saved into cellosaurus\_data.txt
2. extract\_details(): The details from cellosaurus\_data.txt are then extracted and saved into 3 tables (cellosaurus, cellosaurus\_synonym, cellosaurus\_taxonomy)

**­­­­­­­­­­­­­­­­­­­­­­­­­­Portion 4: GDS Scrapy Crawler**

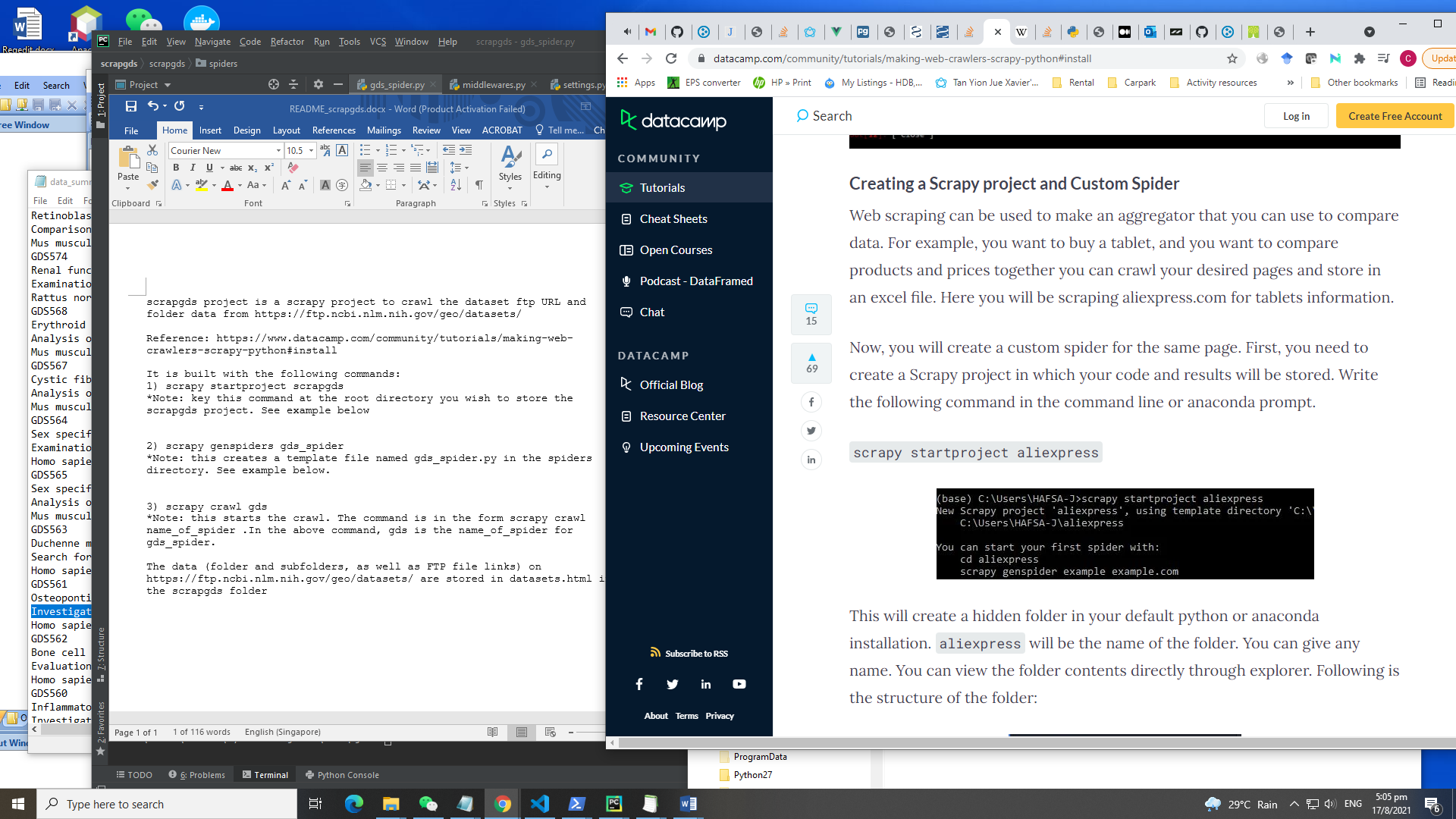
The GDS Scrapy Crawler (Scrapgds) uses scrapy to crawl the dataset ftp URL and folder data from https://ftp.ncbi.nlm.nih.gov/geo/datasets/

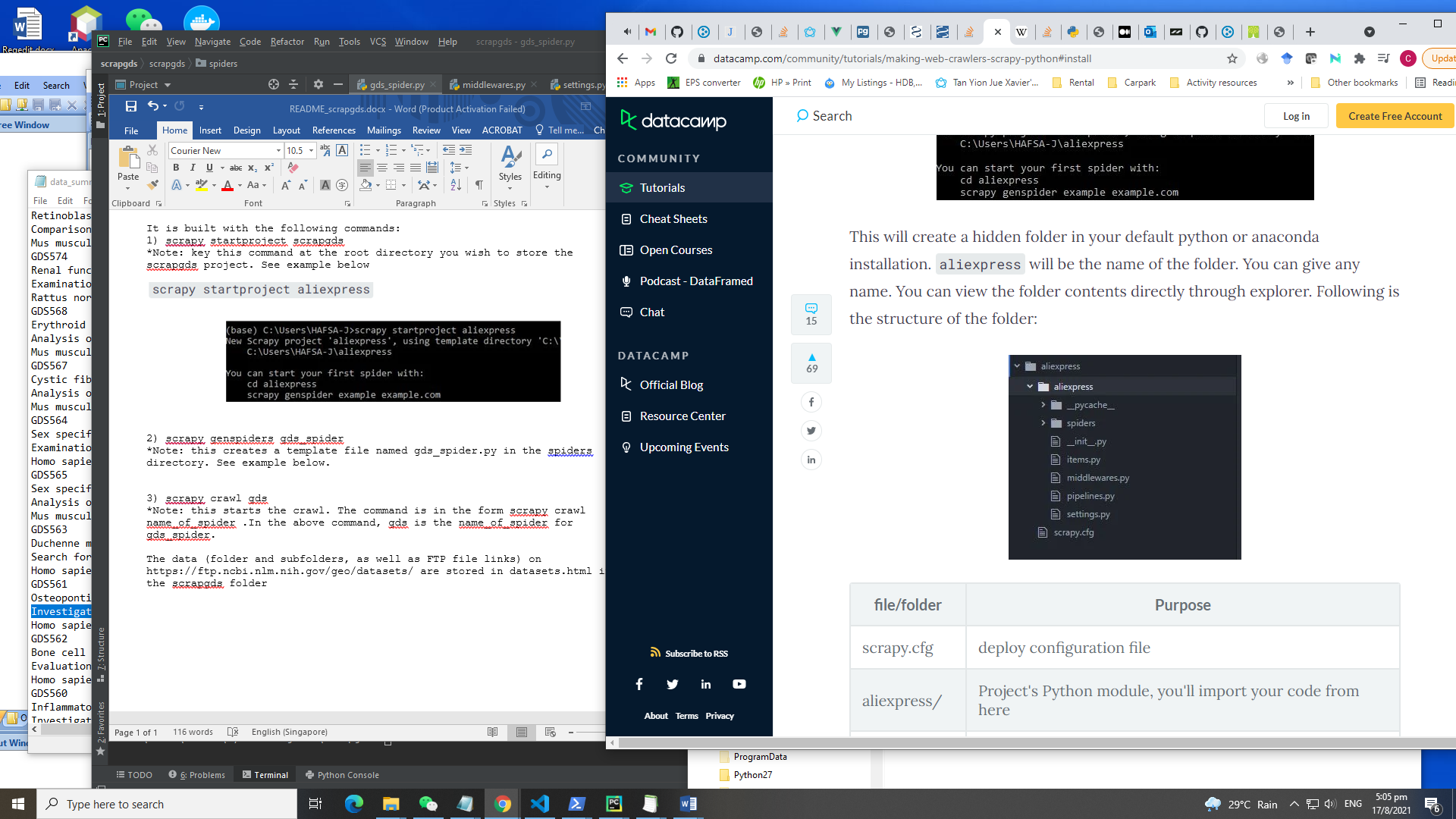
Reference: https://www.datacamp.com/community/tutorials/making-web-crawlers-scrapy-python#install

It is created as a separate PyCharms project initially with the following commands:

1) scrapy startproject scrapgds

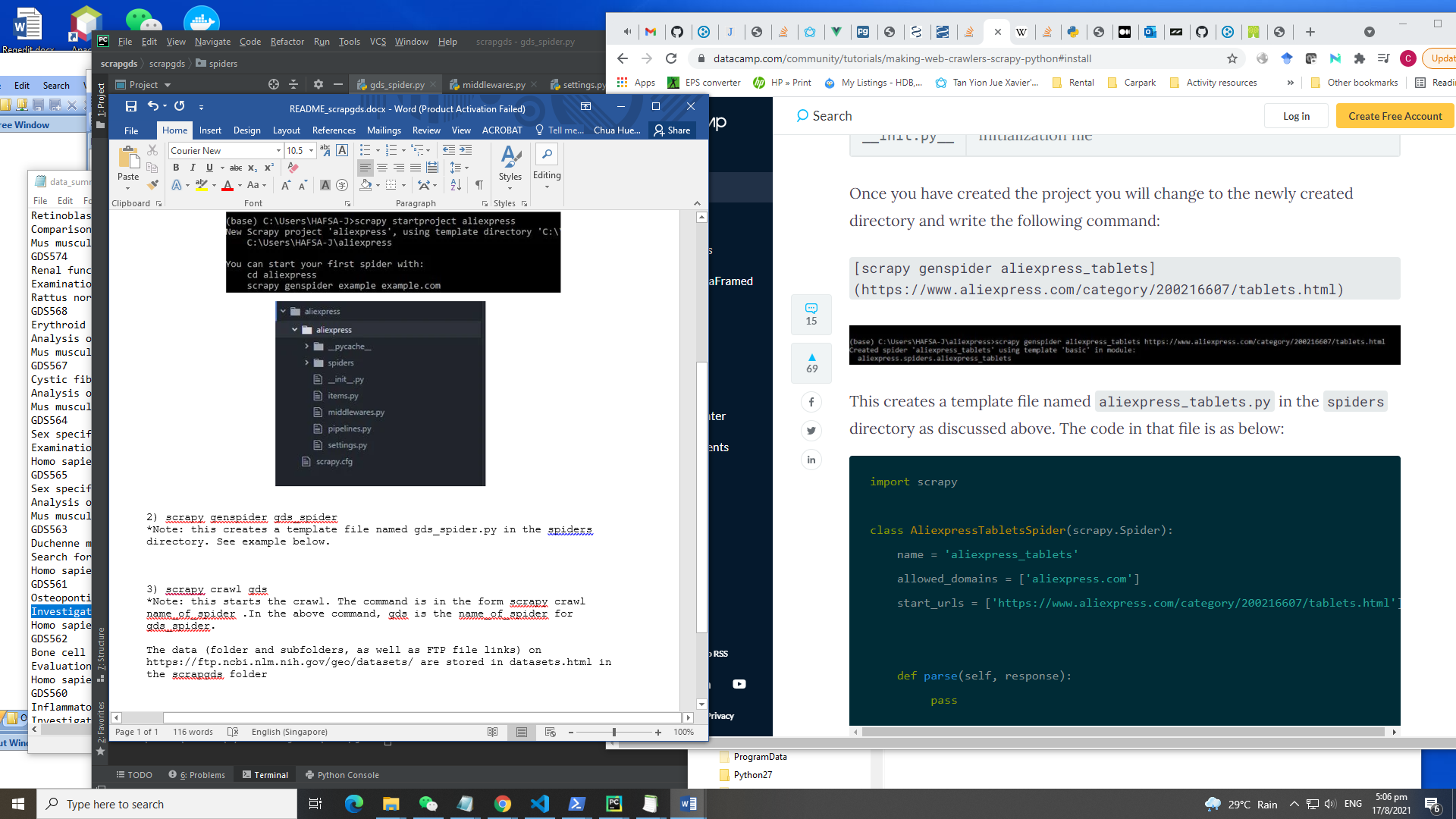
\*Note: key this command at the root directory you wish to store the scrapgds project. See example below





2) scrapy genspider gds\_spider

\*Note: this creates a template file named gds\_spider.py in the spiders directory. See example below.



3) scrapy crawl gds

\*Note: this starts the crawl. The command is in the form scrapy crawl name\_of\_spider .In the above command, gds is the name\_of\_spider for gds\_spider.

The data (folder and subfolders, as well as FTP file links) on https://ftp.ncbi.nlm.nih.gov/geo/datasets/ are stored in datasets.html in the scrapgds folder

\*\* Subsequently, the scrapgds folder is integrated into the offline pipeline.

**­­­­­­­­­­­­­­­­­­­­­­­­­­Portion 5: GDS Processor**

The GDS processor is used to download and decompress GDS FTP soft files, and also to perform NLP to retrieve disease and cellline information from the title and description fields in the soft files. The current version of scispacy (v0.4.0) leverages spacy (v3.0.0 onwards) and seem to give poorer NER results (<https://github.com/explosion/spaCy/issues/8138>, <https://github.com/allenai/scispacy/issues/342>). Hence, we reverted to older version of scispacy(v0.3.0, <https://github.com/allenai/scispacy/releases/tag/v0.3.0>.) and downgraded spacy accordingly (v2.3.7 is installed when we do pip install scispacy==0.3.0). Note that NER models of scispacy follow closely to the version of scispacy. For scispacy(v0.3.0), the links are:

* en\_ner\_bc5cdr\_md: <https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.3.0/en_ner_bc5cdr_md-0.3.0.tar.gz>
* en\_ner\_jnlpba \_md: <https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.3.0/en_ner_jnlpba_md-0.3.0.tar.gz>
* en\_core\_sci\_lg: <https://s3-us-west-2.amazonaws.com/ai2-s2-scispacy/releases/v0.3.0/en_core_sci_lg-0.3.0.tar.gz>

The version 0.3.0 model files are currently downloaded to scispacyVersion030 folder.

The downloaded and decompressed FTP files are found in *ftp* and *extractedFtp* folders, respectively.

The main file (main.py) does several things:

1. generate\_gdslist\_ftplist(): generate the ftplist of the GDS records
2. download\_ftplist(): download all ftp files above
3. decompress\_gzfiles(): decompress the downloaded ftp files
4. extract\_data\_gzfiles(): extract data from the decompressed ftp files
5. perform NLP using scispacy to extract disease terms in title and description fields
   1. functions: run\_scispacy\_disease(), multiprocess\_utr\_rest\_disease(), union\_title\_description\_table\_disease()
   2. extract UMLS (https://www.nlm.nih.gov/research/umls/index.html) and NCIT ids related to disease terms if they are available. Refer to <https://github.com/HHS/uts-rest-api/blob/master/samples/python/walk-hierarchy.py> and <https://github.com/hannesUlrich/uts-rest-api/blob/master/samples/python/walk-hierarchy.py> for examples of using the UMLS Terminology Services REST API. Note that UMLS requires a user apikey for using the REST API. This key has to be added to configuration\_credentials.txt (look for UMLS\_apikey tag). In cases where disease terms are found, but no related UMLS are available, then we save the disease name. This allows the code to check for synonyms of disease to try to map to existing NCI terms in the database if it’s available.
   3. The above data are stored in the PostgreSQL DB (Hogwarts) as different categories of tables: (1) mapping of gds to umls (gds\_description\_mapto\_umls, gds\_title\_mapto\_umls, gds\_titledescription\_mapto\_umls), (2) mapping of umls to ncit (umls\_mapto\_ncit) and (3) mapping of gds to disease name without umls records (gds\_description\_mapto\_disease, gds\_title\_mapto\_disease, gds\_titledescription\_mapto\_disease). Note that we split the title and the description source so that the code can assign different logic accordingly. For clarity, gds\_titledescription\_mapto\_umls is a union of the 2 tables gds\_description\_mapto\_umls and gds\_title\_mapto\_umls.
6. perform NLP using scispacy to extract cellline terms in title and description fields
   1. functions: run\_scispacy\_cellline(), save\_cellline\_to\_db(), union\_title\_description\_table\_cellline()
   2. The cell line term extracted tend to be different from the synonyms found in Cellosaurus. Synonyms in Cellosaurus tend to be short (i.e., a word such as MCF-7 that represents the cell line) where the cell line term can be more verbose (e.g., MCF-7 cell line). To handle this, we split the cell line term into words and try to match each word to existing synonyms in Cellosaurus. If a match is found, the corresponding Cellosaurus accession number and disease NCI Thesaurus ID is retrieved and an entry is created for that particular GDS record in the tables (i.e., gds\_title\_mapto\_cellosaurus, gds\_description\_mapto\_cellosaurus, gds\_titledescription\_mapto\_cellosaurus). Note that we split the title and the description source so that the code can assign different logic accordingly. For clarity, gds\_titledescription\_mapto\_ cellosaurus is a union of the 2 tables gds\_description\_mapto\_ cellosaurus and gds\_title\_mapto\_ cellosaurus.

Basic idea to decide what disease terms are related to a particular GDS record:

* Disease terms extraction (NER model = en\_ner\_bc5cdr\_md, ent.label\_ = DISEASE) using scispacy
* Cell line terms extraction (NER model = en\_ner\_jnlpba\_md, ent.label\_ = CELL\_LINE) using scispacy

Note that we do this for title field of GDS record first and then if result is unverified, we proceed to apply the same logic using the description field of GDS record.



If there’s a need to refresh the gds related tables database in postgres, we will need to drop cascade all the tables listed in points 5 & 6 above and then rerun the GDS processor.