P3: Wrangle OpenStreetMap Data

Final Report

Andri Rizhakov

December 01, 2015

Supporting Course: Data Wrangling with MongoDB

Data Analyst Nanodegree

Udacity

1. Revision History

Rev. 0. 12/01/2015. Initial submission.

2. References

1) https://wiki.openstreetmap.org/wiki/OSM_XML
2) https://wiki.openstreetmap.org/wiki/Key:amenity
3) https://wiki.openstreetmap.org/wiki/Main_Page
4) https://docs.python.org/2/library/re.html
5) Code to create a smaller sample of .osm file used, as recommended in Instructor Notes. Final version of the code used is in the Submission Contents.
6) https://www.youtube.com/watch?v=E2PkF-XHj-o
7) http://stackoverflow.com/questions/30953611/mongodb-error-validating-settings-only-one-positional-argument-is-allowed
8) https://docs.mongodb.org/manual/tutorial/manage-mongodb-processes/
9) OpenStreetMap Sample Project

3. Submission Contents

Please navigate to ".../udacity_P3/Submission/" to find the below files.

File	**Description of contents**
DAN_P3_writeup_151201_Rev0.pdf	A pdf document containing answers to the rubric questions.
All .py files in the folder	Python code for Lesson 6 quizzes, as well as additional code used in auditing and cleaning dataset for the final project.
Seattle.txt	A text file containing a link to the map position wrangled in project, a short description of the area, and a reason for choice.
sample_seattle_washington.osm	An .osm file containing a sample part of the map region used (around 1 - 10 MB in size)
References.txt	A text file containing a list of Web sites, books, forums, blog posts, github repositories, etc. that referred to or used in

	this submission (Add N/A if you did not use such resources).

4. Background and Methodology

This project is connected to the Udacity's Data Wrangling with MongoDB course. The steps required for completion per Project Description and methodology used is discussed in detail in Appendix C of this document.

The high-level requirements of this Project are to clean an OpenStreetMap dataset of a designated region with established data wrangling techniques anytime there is poor data. The data quality audit assesses for validity, accuracy, completeness, consistency, and uniformity. If any cleaning of data is needed from the data quality audit step, a data cleaning blueprint and plan is formulated and then executed to achieve the data quality level of interest. The cleaning blueprint

- first audited the data (e.g., finding problems with the data, such as those in Section 5),
- then, developed a cleaning plan to wrangle the data (e.g., finding steps to correct each problem to the desired level of data quality),
- then, programming each step of the cleaning plan,
- then, executing the plan, and
- finally revising the cleaning plan if desired data quality is not achieved.

The input OpenStreetMap dataset was chosen for the Seattle, Washington area:

This was chosen due to personal interest in the area (i.e., interest in potential immigration) after a weekend that was recently spent there. For these purposes, quality of life considerations are chosen for exploration with the goal of getting an idea of the types of amenities and entertainment available in the area (massive assumption exists in that an adequate work opportunity is found for employment).

5. Problems Encountered in the Map

While auditing the OpenStreetMap dataset and assessing the measures of data quality (validity, accuracy, completeness, consistency, and uniformity), some issues were observed. In summary, they are:

1) Street type inconsistency
2) Zip code errors
3) Amenity type inconsistency
4) Lack of country entries
5) Need for truncated dataset due to size

These issues are discussed further in this section.

5.1. Street type inconsistency

Following the Lesson 6 walkthrough and code, there were many issues with the street naming and conventions that appeared in the data set. This affected the validity, consistency, and uniformity of data.

Manual investigation (specifically via Notepad++ text editor) and automated investigation via printing out in DWM_Project_L6_4_Improving Street Names.py to figure out issues with street names. For example,

NE 95th St. ===> Northeast 95th Street

The issue was resolved by adapting ‘update street name’ function from Lesson 6. Augmenting the Lesson 6 problem set, my version added “Street”, if no road type was given.

To check that the ‘update street name’ function was properly implemented into the .osm to .json conversion code, the following pipeline was used to query the database:

```
pipeline = [{"$match":{"address.street":{"$exists":1}, "address.street":"Northeast 95th
             Street"}}
            ]
```

With the following result (modified for readability):

```
{u'_id': ObjectId('564f87cfcafb1e2916cf8b2f'),
 u'address': {u'city': u'Seattle',
              u'housenumber': u'2400',
              u'postcode': u'98115',
              u'state': u'WA',
              u'street': u'Northeast 95th Street'},
 ....
 u'type': u'way'}
```

Note that ‘address.street’ has the correct syntax, as illustrated earlier.

5.2. Zip code errors

Manual investigation (specifically via Notepad++ text editor) and automated investigation via MongoDB queries to figure out issues with zip codes. Preliminarily, it was understood that there could be many issues that affected the validity, accuracy, consistency, and uniformity of data.

To pull the zip codes of the data entries out of the MongoDB database, the following pipeline was used to query the database:

```
pipeline = [{"$match":{"address.postcode":{"$exists":1}}},
            {"$group":{"_id":"$address.postcode", "count":{"$sum":1}}},
            {"$sort":{"count":-1}}
           ]
```

With the following result (modified for readability):

```
[{u'_id': u'98034', u'count': 11475},
 {u'_id': u'98033', u'count': 9679},
 {u'_id': u'98115', u'count': 9072},
 {u'_id': u'98103', u'count': 8418},
 {u'_id': u'98118', u'count': 7162},
 {u'_id': u'98117', u'count': 6760},
 {u'_id': u'98125', u'count': 6056},
 {u'_id': u'98105', u'count': 4970},
 {u'_id': u'98108', u'count': 4642},
 {u'_id': u'98144', u'count': 4631},
```

Note that the top entries have a valid US zip code, around the Seattle area (verified manually by entering zip code into Google Maps). There were outliers to the standard US zip code format, however, and the most occurring or otherwise different outliers are presented below:

```
[...{u'_id': u'V9B1L8', u'count': 23},
 {u'_id': u'Lacey, WA 98503', u'count': 7},
{u'_id': u'Olympia, 98502', u'count': 7},
 {u'_id': u'V9B 1R6', u'count': 6},
 {u'_id': u'186631629:186700775:186700777', u'count': 1},
…]
```

It was later realized that the zip codes that started with “V9” were Canadian zip codes of the Victoria area. Thusly, this was not an issue in terms of data accuracy or consistency, though there were issues with some other US zip codes (e.g., including city names present in zip field, possible node values in zip field, etc.). These issues are deemed as outliers, and effort to correct is higher than the possible reward because only ~10 of ~11000 datasets, or ~0.1%, is

affected. Thusly, no resolution was needed for this purpose, and the database was sufficient for this verification check.

5.3. Amenity type inconsistency

After cross-referencing with .osm wiki documentation page, the amenities that are in the data set do not exist in the documentation page. Examples are “martial_arts” and “Martial Arts Studio”; not only are neither in the wiki documentation page, but they are redundant when compared to each other. Data consistency is compromised when data doesn’t match a documented standard.

5.4. Lack of country entries

Tipped off from the ‘zip code’ investigation above (specifically, with the “V9” prefix of zipcodes), some addresses of the data entries are in Canada and there is no country provided. For the same Canadian city, some data entries have a ‘country’ field populated, while others do not have the field at all. An example MongoDB query for Victoria, Canada:

```
pipeline = [{"$match":{"address.city":{"$exists":1}, "address.city":"Victoria"}}
            ]
```

Yields (results truncated for readibility):

```
{u'_id': ObjectId('564f87c7cafb1e2916cccb87'),
 u'address': {u'city': u'Victoria',
              u'country': u'CA',
              u'housenumber': u'3475',
              u'postcode': u'V8X 1G8',
              u'street': u'Quadra Street'},
 ...
 u'type': u'way'},
{u'_id': ObjectId('564f87cccafb1e2916ce9b54'),
 u'address': {u'city': u'Victoria',
              u'housenumber': u'25',
              u'postcode': u'V8Z 1S4',
              u'street': u'Crease Avenue'},
 ....
 u'wheelchair': u'limited'},
```

The effect of this oversight may be important. For example, can be dangerous if one looks up venue name carelessly, and has to go to Canada passport-less.

5.5. Need for truncated dataset due to size

Initial attempt in .osm to .json conversion was made with the original dataset (.osm file ~1.4GB). The conversion via DWM_Project_L6_5_Preparing for Database.py was not successful due to memory errors. The reduction via DWM_Project_smallerFile.py helped the conversion, but taking every 10 data entries placed too granular of a filter on the data: this made observations

from this data incomplete. Another solution was needed to improve the completeness of the data set.

The best solution for this was to obtain 64-bit Python and rerun the .osm to .json conversion. Again, the full file was not converted due to memory errors, but DWM_Project_L6_5_Preparing for Database.py was used again to take every 2 data entries from the original dataset. This was the highest resolution possible to run with current given computing resources; the memory usage during the conversion step was ~22 GB!

With respect to measures of data quality, this truncation step mainly affected the accuracy, completeness, and consistency measures. Because an incomplete dataset is used (due to the necessary truncation), it is not possible to know the issues that may remain in the unused data entries.

6. Data Overview

This section contains some basic data overview of the MongoDB database and collections that were created. Some queries and results are presented for further depiction of the data set.

The code used for the high-level statistics of the database, and its output below the code are displayed below.

Database statistics:

```
pprint.pprint(db.command("dbstats"))   ### db stats

{u'avgObjSize': 305.1741995126776,
 u'collections': 5,
 u'dataFileVersion': {u'major': 4, u'minor': 22},
 u'dataSize': 1581221968.0,
 u'db': u'PROJECT',
 u'extentFreeList': {u'num': 0, u'totalSize': 0},
 u'fileSize': 4226809856.0,
 u'indexSize': 168147616.0,
 u'indexes': 3,
 u'nsSizeMB': 16,
 u'numExtents': 47,
 u'objects': 5181375,
 u'ok': 1.0,
 u'storageSize': 2066792448.0}
```

Collection statistics:

```
pprint.pprint(db.command("collstats", "date151030"))   ### coll stats

{u'avgObjSize': 308,
```

```
 u'capped': False,
 u'count': 740195,
 u'indexDetails': {},
 u'indexSizes': {u'_id_': 24029264},
 u'lastExtentSize': 68579328.0,
 u'nindexes': 1,
 u'ns': u'PROJECT.date151030',
 u'numExtents': 13,
 u'ok': 1.0,
 u'paddingFactor': 1.0,
 u'paddingFactorNote': u'paddingFactor is unused and unmaintained in 3.0. It remains
hard coded to 1.0 for compatibility only.',
 u'size': 228063184,
 u'storageSize': 243314688,
 u'totalIndexSize': 24029264,
 u'userFlags': 1}
```


Numbers of original documents:

```
print "db.date151120.find().count() ::: ", db.date151120.find().count()

db.date151120.find().count() :::  3700975
```

Number of nodes:

```
print "db.char.find({type:node}).count() ::: ", db.date151120.find({"type":"node"}).count()

db.char.find({type:node}).count() :::  3377333
```

Number of ways:

```
print "db.char.find({type:way}).count() ::: ", db.date151120.find({"type":"way"}).count()

db.char.find({type:way}).count() :::  323567
```

Number of distinct users:

```
print "len(db.date151120.distinct(created.user)) ::: ",
        len(db.date151120.distinct("created.user"))

len(db.date151120.distinct(created.user)) :::  2341
```

The creator of most documents (result shows the creator name and the total count of their documents):

```
pipeline = [{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
            {"$sort":{"count":-1}},
            {"$limit":1}
            ]

[{u'_id': u'Glassman', u'count': 612862}]
```

The number of users that have created only 1 document:

```
pipeline = [{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
            {"$group":{"_id":"$count", "num_users":{"$sum":1}}},
            {"$sort":{"_id":1}},
            {"$limit":1}
            ]

[{u'_id': 1, u'num_users': 509}]
```

7. Additional Ideas

7.1. Standardizing naming conventions

There appears to be an issue with standardizing naming conventions of key and value pairs in the data set. The amenity “martial arts studio” is an example (Section 5.3), and others include phone numbers (e.g., “+1” at front of numbers or not) and websites (e.g., “www” at front of webpages or not”. Perhaps these are minor errors, but for perfecting data accuracy, validity, consistency, and uniformity to an established “gold standard” would make analysis much easier by not performing so much data wrangling on each data set by different analysts every time.

Perhaps as a front-end check, the submission page should run a check versus the documentation, and then give it a label of “sufficient” (i.e., does conform to a ‘gold standard’ that is described in the documentation) or ”not sufficient” (i.e., does not conform to a ‘gold standard’ that is described in the documentation). Thus, if the data that was submitted with the amenity “martial arts studio”, the data would be given a label of ”not sufficient”. It is possible that the data would be read in and stored (to keep with the open source idea and free access to anyone that wants to edit and contribute), but there would exist a “sufficient only”, “not sufficient only”, or “mixed” option at the download page. If “sufficient only” is chosen, the data set that would be downloaded is one that is comprised of “sufficient” data points. Conversely, if “not sufficient only” is chosen, the data set that would be downloaded is one that is comprised of “not sufficient” data points; this can be useful for person that want to data wrangle the data points that are not conforming to the documentation’s gold-standard. Finally, if “mixed” is chosen, the data set that would be downloaded is one that is comprised of “sufficient” and “not sufficient” data points, for those needs that require as comprehensive information as possible, though potentially not consistent. Note that the redundancy of “sufficient only”, “not sufficient only”, or “mixed” options is identified (i.e., “mixed” is redundant because it consists of the other two data

sets), but this is still made available, assuming there are no constraints on data storage or execution speeds.

7.2. Further data exploration with MongoDB queries

To meet the goal of getting an idea of the types of amenities and entertainment available in the area for quality of life considerations, the following queries were run.

First, to get an idea of the distribution of all amenities in the area:

```
pipeline = [{"$match":{"amenity":{"$exists":1}}},
            {"$group":{"_id":"$amenity", "count":{"$sum":1}}},
            {"$sort":{"count":-1}}#,
            #{"$limit":10}
            ]

[{u'_id': u'parking', u'count': 3775},
 {u'_id': u'bicycle_parking', u'count': 1520},
 {u'_id': u'school', u'count': 1346},
 {u'_id': u'restaurant', u'count': 1290},
 {u'_id': u'bench', u'count': 790},
 {u'_id': u'place_of_worship', u'count': 744},
 {u'_id': u'fast_food', u'count': 588},
 {u'_id': u'cafe', u'count': 532},
 {u'_id': u'fuel', u'count': 502},
 {u'_id': u'toilets', u'count': 378},
 {u'_id': u'waste_basket', u'count': 370},
 {u'_id': u'bank', u'count': 353},
 {u'_id': u'bar', u'count': 134},
…
]
```

Next, since one particular physical activity is yoga, the following query was used to get an idea of yoga studios in the area:

```
pipeline = [{"$match":{"amenity":{"$exists":1},"amenity":"yoga"}}
            ]

[{u'_id': ObjectId('564f87c6cafb1e2916cc8337'),
  u'address': {u'city': u'Seattle',
               u'housenumber': u'2238',
               u'postcode': u'98107',
               u'street': u'Northwest Market Street'},
  u'amenity': u'yoga',
  u'building': u'yes',
...
  u'source': u'King County GIS',
```

```
 u'type': u'way',
 u'website': u'www.shaktivinyasa.com'}]
```

Next, because my girlfriend is a major foodie, a breadth of restaurants is a must:

```
pipeline = [{"$match":{"amenity":{"$exists":1},"amenity":"restaurant"}},
            {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
            {"$sort":{"count":-1}},
            {"$limit":20}
            ]
```

```
[{u'_id': None, u'count': 436},
 {u'_id': u'mexican', u'count': 106},
 {u'_id': u'pizza', u'count': 91},
 {u'_id': u'american', u'count': 88},
 {u'_id': u'thai', u'count': 52},
 {u'_id': u'italian', u'count': 50},
 {u'_id': u'japanese', u'count': 49},
 {u'_id': u'chinese', u'count': 49},
 {u'_id': u'asian', u'count': 44},
 {u'_id': u'burger', u'count': 34},
 {u'_id': u'sandwich', u'count': 33},
 {u'_id': u'vietnamese', u'count': 30},
 {u'_id': u'seafood', u'count': 25},
 {u'_id': u'indian', u'count': 20},
 {u'_id': u'regional', u'count': 17},
 {u'_id': u'sushi', u'count': 14},
 {u'_id': u'greek', u'count': 13},
 {u'_id': u'steak_house', u'count': 10},
 {u'_id': u'mediterranean', u'count': 9},
 {u'_id': u'international', u'count': 8}]
```

It appears that at a high-level, there exists plenty of interesting entertainment. Certainly enough to continue to deeper exploration of the area for possible relocation.

8. Conclusion

In order to obtain some preliminary knowledge about interesting entertainment and amenities in the Seattle, Washington area, a successful conversion of raw .osm data was required into .json (the format for importing into MongoDB). This successful conversion also needed to correct any data quality issues because this could lead to less meaningful results during the final MongoDB query. An assessment of quality of data was made for validity, accuracy, completeness, consistency, and uniformity via the data cleaning blueprint, in regards to meeting the purpose of understanding the interesting entertainment and amenities in the Seattle, Washington.

8.1. Validity

The validity of the data gauges how the data conforms to a schema. Although there were some issues in the consistency of the usage of some amenity fields, the validity was appropriate for the purposes of understanding the interesting entertainment and amenities in the area because appropriate insight was gained from the final, post-audit data set to assist with understanding.

8.2. Accuracy

The accuracy of the data gauges how the data conforms to a gold standard. Although this is the most difficult measure of quality to meet because a gold standard for the data set did not exist, the accuracy was appropriate for the purposes of understanding the interesting entertainment and amenities in the area because appropriate insight was gained from the final, post-audit data set to assist with understanding. Spot checking by Googling some field values, such as zip codes, allowed some matching with real-world, true results. It is impossible to say if all of the data points in the data set are accurate (match to the real-world truth), but operating under the assumption that it is sufficient poses to risk for the high-level purpose at hand.

8.3. Completeness

The completeness of the data gauges if all records exist. Although there was an issue in truncating the data to enable processing due to memory constraints as well as other missing data points, the completeness was appropriate for the purposes of understanding the interesting entertainment and amenities in the area because appropriate insight was gained from the final, post-audit data set to assist with understanding.

8.4. Consistency

The consistency of the data gauges how well data points match other data points. Although there were issues with street type consistencies and are insignificant issues with zip code and country inconsistencies, the completeness was appropriate for the purposes of understanding the interesting entertainment and amenities in the area because appropriate insight was gained from the final, post-audit data set to assist with understanding.

8.5. Uniformity

The uniformity of the data gauges if same units are used. Although a fix was needed to ensure that missing street types were appended a “street” to the field value, the uniformity was appropriate for the purposes of understanding the interesting entertainment and amenities in the area because appropriate insight was gained from the final, post-audit data set to assist with understanding.

Overall, the conversion process, including the data cleaning blueprint execution, and import was successful, allowing for successful execution of MongoDB queries to obtain high-level understanding of the data set and thusly the geographical area of interest.

Appendix A. Code summary and Calculations

Calculations to previous lessons and problem sets were performed in the files found attached to the report submission via “Background” folder (“.../udacity_P3/Background/”).

Appendix B. Reviewer Comments and Disposition

Reviewer Comment	Disposition in latest version
N/A	N/A

Appendix C. Detailed Methodology

Step One - Finish Lesson 6

Requirement: Make sure all Lesson 6 programming exercises are solved correctly.

Analysis: Confirmed all Lesson 6 programming exercises are solved correctly.

Resolution: Step complete.

Step Two - Review the Rubric and Sample Project

Requirement: The Project Rubric will be used to evaluate your project. It will need to Meet Specifications for all the criteria listed. The Sample Project is an example of what your final report could look like.

Analysis: Project Rubric and Sample Project Report reviewed, and objectives understood.

Resolution: Step complete.

Step Three - Choose Your Map Area

Requirement: Choose any area of the world from https://www.openstreetmap.org , and download a XML OSM dataset. The dataset should be at least 50MB in size (uncompressed). We recommend using one of following methods of downloading a dataset:

- Download a preselected metro area from Map Zen (Note that data obtained from Map Zen is compressed and will usually expand to sizes that meet project requirements.)
- Use the Overpass API to download a custom square area. Explanation of the syntax can found in the wiki. In general you will want to use the following query:(node(minimum_latitude, minimum_longitude, maximum_latitude, maximum_longitude);<;);out meta; e.g.(node(51.249,7.148,51.251,7.152);<;);out meta; the meta option is included so the elements contain timestamp and user information. You can use the Open Street Map Export Tool to find the coordinates of your bounding box. Note: You will not be able to use the Export Tool to actually download the data, the area required for this project is too large.

Analysis: Project area chosen. Data downloaded from Map Zen ["Seattle, Washington" region; Web address: https://s3.amazonaws.com/metro-extracts.mapzen.com/seattle_washington.osm.bz2].

Resolution: Step complete.

Step Four - Process your Dataset

Requirement: Thoroughly audit and clean your dataset, converting it from XML to JSON format. It is recommended that you start with the Lesson 6 exercises and modify them to suit your chosen data set. As you unravel the data, **take note of problems** encountered along the way as well as issues with the dataset. You are going to need these when you write your project report. Finally, import the clean JSON file into a MongoDB database and run some queries against it.

Analysis: Lesson 6 exercises modified to suit chosen data set. Problems encountered along the way as well as issues with the dataset are noted for future use (See below). Finally, imported the clean JSON file into a MongoDB database and ran queries against it.

running MDB locally:
1. need to run the database command, so starts database/server [, after navigating to where mongod.exe is]:
   mongod.exe --dbpath "d:\test\mongo db data"
2. then can execute pymongo
3. if needed, can start “mongo” for the MDB server

-file too big to parse. -> need dir about how to compress, make smaller,
-used suggested script to abbreviate file for processing on local machine [memory limit]
    - investigation resulted in need to get 64-bit python version via Continuum’s Anaconda package to make the conversion from .osm to .json.
-repeated L6 ex with truncated .osm file. All ex worked all the way until .json export.
    - file mods made to the L6 files to improve parsing, such as “Street” addition to street address if no road type specified.
    - L6_1: returns ‘tags’ dict of .osm file
    - L6_2: returns ‘tag_types’ dict of .osm file
    - L6_3: returns ‘users’ dict of .osm file
    - L6_4: audits and improves street name.
        - if no road type, “Street” is added. Assumption: because this is a street field, the entry, if blank, was meant to contain a “Street” extension.
        - process: used pretty print to get all street types, then added to ‘mapping’ list for any mods.
    - L6_5: prepare for database. Added ‘update_street_name’ procedure from L6_4 to clean street names on import as .json.

-.json file made
-lots of trouble with import of .json file
    but! “C:\Program Files\MongoDB\Server\3.0\bin>mongoimport --db PROJECT -c 151030 --file sample_seattle_washington_15103014.json” worked. imported. {ref: http://stackoverflow.com/questions/30953611/mongodb-error-validating-settings-only-one-positional-argument-is-allowed }
    ====

```
C:\Program Files\MongoDB\Server\3.0\bin>mongoimport --db PROJECT -c
151030 --file sample_seattle_washington_15103014.json
2015-10-30T17:57:19.547-0400    connected to: localhost
2015-10-30T17:57:19.661-0400    [........................] PROJECT.151030       990.0
KB/164.7
MB (0.6%)
2015-10-30T17:57:22.660-0400    [###.....................] PROJECT.151030       25.9
MB/164.7 MB (15.7%)
2015-10-30T17:57:25.660-0400    [#######.................] PROJECT.151030
49.5 MB/164.7 MB (30.0%)
2015-10-30T17:57:28.663-0400    [###########.............] PROJECT.151030
71.4 MB/164.7 MB (43.3%)
2015-10-30T17:57:31.660-0400    [###############.........] PROJECT.151030
96.9 MB/164.7 MB (58.8%)
2015-10-30T17:57:34.660-0400    [###################.....] PROJECT.151030
122.5 MB/164.7
MB (74.4%)
2015-10-30T17:57:37.661-0400    [########################..]
PROJECT.151030       152.5 MB/164.7
MB (92.6%)
2015-10-30T17:57:38.806-0400    imported 740195 documents
=====
```

- mongodb querying:
    1) problems with “mongoimport --db PROJECT -c 151030 --file sample_seattle_washington_15103014.json”, needed to rename a collection ‘date151030’.
    2) query [“db.date151030.find()”] worked.
    3) reloaded “...every2.json”, for better data into MDB. This is approximately half of the entries, since the ‘makeFile’ code read every 2 lines of the original .osm file.
    4) proceed with queries similar to those in Sample Final Report:
        a) stats() command, etc worked (see .py file for pymongo code)
        b) All pipeline commands work

Resolution: Step complete.

Step Five - Document your Work

Requirement: Create a document (pdf, html) that directly addresses the following sections from the Project Rubric:

- Problems encountered in your map
- Overview of the Data
- Other ideas about the datasets

Try to include snippets of code and problematic tags (see Sample Project) and visualizations in your report if they are applicable.

Analysis: pdf document created with the required sections. Snippets of code and visualizations used as needed.

Resolution: Step complete.