# MCIS6273 Data Mining (Prof. Maull) / Fall 2018 / HW0

**This assignment is worth up to 20 POINTS to your grade total if you complete it on time.**

| Points Possible | Due Date | Time Commitment (estimated) |
|---|---|---|
| 20 | Sunday, Sep 09 @ Midnight | *up to* 20 hours |

- **GRADING:** Grading will be aligned with the completeness of the objectives.
- **INDEPENDENT WORK:** Copying, cheating, plagiarism and academic dishonesty *are not tolerated* by Univerisity or course policy. Please see the syllabus for the full departmental and University statement on the academic code of honor.

## OBJECTIVES

- Familiarize yourself with the JupyterLab environment
- Familiarize yourself with Github and basic git
- Explore JupyterHub Linux console integrating what you learned in the prior parts of this homework
- Explore Python for basic text mining
- Explore Python for data munging and analysis, with an introduction to JSON and Pandas

## WHAT TO TURN IN

You are being encouraged to turn the assignment in using the provided Jupyter Notebook. To do so, make a directory in your Lab environment called `homework/hw0`. Put all of your files in that directory. Then zip that directory, rename it with your name as the first part of the filename (e.g. `maull_hw0_files.zip`), then download it to your local machine, then upload the `.zip` to Blackboard.

If you do not know how to do this, please ask, or visit one of the many tutorials out there on the basics of using zip in Linux.

## ASSIGNMENT TASKS

### (5%) Familiarize yourself with the JupyterLab environment

Jupyter (https://jupyter.org) is the core platform we will be using in this course and is the platform of choice for data scientists around the world. We have a JupyterLab setup for this course so that we can operate in a cloud-hosted environment, free from some of the resource contraints of running Jupyter on your local machine (though you are free to set it up and seek my advice if you desire to do that).

The Jupyter envitonment we have setup can be found here https://js-16-117.jetstream-cloud.org/. *You must contact me directly (and immediately) for your login credentials.*

We will be using the Anaconda (https://anaconda.com) distribution of Python for our course.

As you will soon find out Notebooks are an incredibly effective way to mix code with narrative and you can create cells that are entirely code or entirely Markdown. Markdown (MD or md) is a highly readable text format that allows for easy documentation of text files, while allowing for HTML-based rendering of the text in a way that is style-independent.

We will be using Markdown frequently in this course, and you will learn that there are many different "flavors" or Markdown. We will only be using the basic flavor, but you will benefit from exploring the "Github flavored" Markdown, though you will not be responsible for using it in this course – on the "basic" flavor.

§ **THERE IS NOTHING TO TURN IN FOR THIS PART.** Play with and become familiar with the basic functions of the Lab environment at https://js-16-117.jetstream-cloud.org/.

§ **PLEASE *TURN IN A MARKDOWN DOCUMENT* WITH 3 SENTENCES/FRAGMENTS THAT ANSWER THE FOLLOWING QUESTION:**

- **What do you wish to accomplish this semester in Data Mining?**

Read the documentation for basic Markdown here. Turn in the text `.md` file *not* the processed `.html`. In whatever you turn in, you must show the use of *ALL* the following:

- headings (one level is fine),
- bullets,
- bold and italics

Again, the content of your document needs to address the question above. **You will lose points** if you do not do that.

### (5%) Familiarize yourself with Github and basic git

Github (https://github.com) is the *de facto* platform for open source software in the world based on the very popular git (https://git-scm.org) version control system. Git has a sophisticated set of tools for version control based on the concept of local repositories for fast commits and remote repositories only when collaboration and remote synchronization is necessary. Github enhances git by providing tools and online hosting of public and private repositories to encourage and promote sharing and collaboration. Github hosts some of the world's most widely used open source software.

**If you are already familiar with git and Github, then this part will be very easy!**

§ **CREATE A GITHUB.COM ACCOUNT AND RECORD THE URL TO YOUR ACCOUNT.** It is OK if you already have an account, record the account URL either way. Use your `.edu` email address – you will be able to get free private repositories for as long as you are a student with that email address. Enjoy this perk while you can – private repositories start at around $7/month when you are no longer using the `.edu` address. Public repositories are always free.

§ **CREATE A PUBLIC GITHUB REPO NAME `"MyDataSci2018"` AND PLACE A README.MD FILE IN IT.**
Create your first file called `README.md` at the top level of the repository. You can put whatever text you like in the file (If you like, use something like lorem ipsum to generate random sentences to place in the file.). Please include the link to **your** Github repository that now includes the minimal `README.md`. You don't have to have anything elaborate in that file or the repo.

### (0%) Explore JupyterHub Linux console integrating what you learned in the prior parts of this homework

The Linux console in JupyterLab is a great way to perform command-line tasks and is an essential tool for basic scripting that is part of a data scientist's toolkit. Open a console in the lab environment and familiarize yourself with your files and basic commands using git as indicated below.

1. In a new JupyterLab command line console, run the `git clone` command to clone the new repository you created in the prior part. You will want to read the documentation on this command (try here https://www.git-scm.com/docs/git-clone to get a good start).
2. Within the same console, modify your `README.md` file, check it in and push it back to your repository, using `git push`. Read the documentation about `git push`.

3. The commands `wget` and `curl` are useful for grabbing data and files from remote resources off the web. Read the documentation on each of these commands by typing `man wget` or `man curl` in the terminal. Make sure you pipe the output to a file or use the proper flags to do so.

**§ THERE IS NOTHING TO TURN IN FOR THIS PART.**

### (45%) Explore Python for basic text mining

Python is one of the most important languages in contemporary data science right now. Its strengths are in readability and clarity of computational expressiveness. There are a number of data science libraries and modules that we will be using throughout the course to achieve many important outcomes with Python.

You will be tasked with writing two basic programs in Python using Jupyter Lab. For starters, here are a few recommended resources you are encouraged to use though you are free to use whatever resources you find useful:

- Python website (https://python.org)
- Hitchhikers Guide to Python by Kenneth Reitz and Tanya Schlusser (https://docs.python-guide.org)
- Think Python by Allen Downey (http://greenteapress.com/wp/think-python/)

For the first part of the assignment, you will be tasked with a "warm up" tapping into the text processing capabilities of Python. Please turn in the code and answers according to the instructions.

**§ WRITE A PROGRAM IN YOUR JUPYTER NOTEBOOK TO KEEP TRACK OF THE FREQUENCY OF THE TOP 30 WORDS THAT END IN "*ng*" FOR THE FOLLOWING 3 BOOKS:**

(*The top 30 should be sorted by descending frequency – most frequent to least.*)

- The Republic by Plato http://www.gutenberg.org/cache/epub/1497/pg1497.txt
- Don Quixote by Miguel de Cervantes Saavedra http://www.gutenberg.org/cache/epub/996/pg996.txt
- The Strange Case Of Dr. Jekyll And Mr. Hyde by Robert Louis Stevenson http://www.gutenberg.org/files/43/43-0.txt

For this part we will be using the open and freely available Gutenberg.org, which contains a large number of books in the public domain available for reading (or whatever purpose). We will just grab the text documents we would like to analyze and process them, though you are welcome to hang out on Gutenberg and do some light evening reading whenever you have free time.

To speed things up, you can store the documents locally (grabbing them through `wget` or `curl`) or you can use the boilerplate code provided below that uses the `requests` library and loads the document files each time over the web. If you store them locally, you are still resposible for writing the code for reading and processing the documents. Either way, it is up to you how you'd like to craft your solution.

If you find it useful, you are welcome to read the documentation for and use the `Counter` object that is part of the Collections module. While `Counter` may be useful, it is not required and there is a solution without it that is equally elegant. NOTE: you should **normalize the text** so that all words are lowercase and free of punctuation (commas, periods, dashes, semicolons, etc.).

**Do not use any other text processing libraries beyond those provided within Python directly (the good news is that you won't have to, but ask me if you have any doubts).** You also do not need to do any stripping of text from the preamble, introductory remarks, etc. from the documents. Process them as they are.

You will also find that `split()` is useful in accomplishing tasks for this part, and the `requests` library is useful for grabbing the text directly from Gutenberg.org as shown in the demo code below, though you are free to use whatever HTTP library (e.g. `urllib`) you like. Finally, you can consider normalizing text with the `re` regular expressions library. For example, you can remove all `,` (comma) from a word with

`re.sub(r',','','thus,')`. To extend that to all punctuation, read the documentation on character classes and punctuation https://docs.python.org/2/library/re.html#regular-expression-syntax.

Your notebook should include the solution code and answers structured in a table like below:

| document_1 | word count ending in *ng* |
|---|---|
| ng_ending_word_1 | `ng_ending_word1_count` |
| ng_ending_word_2 | `ng_ending_word2_count` |
| ... | ... |
| ng_ending_word_30 | `ng_ending_word30_count` |

| document_2 | word count ending in *ng* |
|---|---|
| ng_ending_word_1 | `ng_ending_word1_count` |
| ng_ending_word_2 | `ng_ending_word2_count` |
| ... | ... |
| ng_ending_word_30 | `ng_ending_word30_count` |

| document_3 | word count ending in *ng* |
|---|---|
| ng_ending_word_1 | `ng_ending_word1_count` |
| ng_ending_word_2 | `ng_ending_word2_count` |
| ... | ... |
| ng_ending_word_30 | `ng_ending_word30_count` |

> *here is code using* `requests` *to read in a text file from gutenberg.org and store it in a variable called* `data`.

```
import requests

url = "http://www.gutenberg.org/files/31475/31475-0.txt"
r = requests.get(url)

if r.status_code == 200:
   data = r.content

   # YOUR CODE TO PROCESS THIS DOCUMENT

else:
   print("[warn] GET request did not return HTTP/200 (HTTP/{} returned instead".format(r.status_code))
```

**(45%) Explore Python for data munging and analysis, with an introduction to JSON and Pandas**

Python's strengths shine when tasked with data munging and analysis. In this section we will be working with a dataset from the City of Baltimore Parks division which maintains open, public datasets for all the public parks in the Baltimore area. For the curious such datasets on public parks are plentiful and vary in quality and detail.

**Please familiarize yourself with the data here** :

https://data.baltimorecity.gov/resource/4qnj-2zbc.json

As you will notice the data is a JSON file. If you are unfamiliar with JSON, please take a look at this resource: the JSON specification.

As you look at the data, you will notice that there are some key data missing, which will be important for answering our data-related questions later. Let's enumerate what those issues are:

1. the original JSON file contains the name and address of the park, but **there is no Geolocation data indicating the latitude and longitude of the park**.
2. **the original JSON file has no ZIP code for the park** – the address is complete, but without a ZIP code we will not be able to answer some critical questions later.

**You will need to complete the following tasks which involve data extraction and manipulation in Python as well as answering specific questions about the data.**

**§ WRITE THE CODE IN YOUR NOTEBOOK TO ENRICH THE JSON SO IT INCLUDES OUR MISSING DATA.** *You will need to perform the following steps*:

1. **load the parks JSON file into memory** using `requests` or by loading it directly from the file system,
2. **iterate over the parks addresses and lookup the ZIP, latitude and longitude** using the census API,
3. **update the in-memory JSON parks data with the ZIP, latitude and longitude**,
4. **store the new JSON file locally** and name it `baltimore_parks_enriched.json`.

We indicated above that the latitude and longitude of the parks were missing, but that there is an address. Lucky for us, there are a host of free online APIs that allow us to fill in the missing pieces. The first API we will use is called the Geocoder API provided by the US Census, which allows us to query the API with a street address and get the ZIP, latitude and longitude of that address (among other things).

> *Here is a code snippet you might like to study and use to craft your solution:*

```
# a sample address
addr = "301 32nd St"

CENSUS_GEO_ADDR_ENDPOINT = "https://geocoding.geo.census.gov/geocoder/locations/address"
r = requests \
      .get("{}?street={}&city=baltimore&state=MD&benchmark=9&format=json" \
      .format(CENSUS_GEO_ADDR_ENDPOINT, addr)

if r.status_code == 200:
  data = r.json()

  # grab the first matching result, we're pretty confident in the matching for this part
  data['result']['addressMatches'][0]

  # YOUR CODE TO EXTRACT/INSERT THE MATCHING DATA (e.g. ZIP, LAT, LON)
else:
   print("[warn] GET request did not return HTTP/200 (HTTP/{} returned instead".format(r.status_code))
```

Notice, we stored the output as a JSON object (`data = r.json()`). You will need to familiarize yourself with the `json` library and the `json` object. Specifically, how JSON is transformed as a Python dictionary allowing easy manipulation. Look at an example usage of the `json` library: https://docs.python-guide.org/scenarios/json/. There are also many other examples you can explore.

**§ USE PANDAS TO CONVERT THE JSON OBJECT TO A DATAFRAME AND ANSWER THE FOLLOWING QUESTIONS:**

1. Which zipcode has the largest *number* of parks?
2. Which zipcode has the largest *acreage* of parks?

3. How many acres of parks are in zipcode 21229?
4. What is the average acreage of parks in zipcode 21215?

To answer these questions, you'll need to dive further into Pandas, which is the standard tool in the Python data science stack for loading, manipulating, transforming, analyzing and preparing data as input to other tools such as Numpy (http://www.numpy.org/), SciKitLearn (http://scikit-learn.org/stable/index.html), NLTK (http://www.nltk.org/) and others.

For this assignment, you will only need to learn how to load and select data using Pandas.

**LOADING DATA** The core data structure in Pandas is the `DataFrame`. You will need to visit the Pandas documentation (http://pandas.pydata.org/pandas-docs) to learn more about the library, but to help you along with a hint, read the documentation on the `pandas.read_json()` method.

**SELECTING DATA** The tutorial here on indexing and selecting should be of great use in understanding how to index and select subsets of the data to answer the questions.

**EXAMPLE CODE**

> *Here is example code that should give you clues about the structure of your code for this part.*

```
import pandas as pd

df = pandas.read_json('your_json_file.json')

# code for question 1 ... and so on
```

**§ BUILD A "DISTANCE TABLE" (WITH A PANDAS DATAFRAME) OF PARK DISTANCES USING THE HAVERSINE FORMULA.** *Your solution should:*

1. create and calculate a distance table from each park to one another, so that **for a given park** you can **compute the distances to all other parks**. Your solution can just involve the `(lat,lon)` for each park as a tuple. The diagonal of your table will, of course, be zero (the distance from the park to itself is 0). The row and column labels of the table need only be the index of the park in the prior exercise.

To use the Haversine formula to compute the distance (in miles) between lat/lon park pairs – study its distance calculation given by:

$$H_d\big((\varphi_1,\lambda_1),(\varphi_2,\lambda_2)\big) = 2r\arcsin\left(\sqrt{(\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)+\cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right)$$

Where $\varphi$ represent the longitude and $\lambda$ the latitude for two points $p_1 = (\varphi_1, \lambda_1)$ and $p_2 = (\varphi_2, \lambda_2)$, where $p_1$ and $p_2$ are the latitude and longitude of two parks.

You can write the Python for the Haversine function on your own or you can integrate the one I have coded here. Just remember $r = 3961$ and that you will need to convert the degrees input (lat/lon) to radians, using the `math.radians` function. Your Haversine function should take two pairs of lat/lon coordinates and return their distance in miles.

To create the table, you will simply load the park data from the previous section, iterate over all park pairs and compute the Haversine distance and appropriately update the table by index. **HINT:** Simply implement this as nested loop, though be patient with the execution time since there are 275 parks (~75K distance pairs) and the loop $O(n^2)$.

The final DataFame (table) will look like an $n \times m$ matrix so that the $n$th row and $m$th column represent the distance from park $n$ to park $m$ using the Haversine distance.

**HINT:** Make sure you do not delete or alter the table from the previous answer as you will need it to make reference to indices of the park names to correctly answer the questions for this part.

Here is what your DataFrame will start to look like:

| | 0 | 1 | 2 | 3 | 4 | 5 | ... |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0.939578 | 3.37182 | 1.78964 | 1.80147 | 3.40832 | ⋯ |
| 1 | 0.939578 | 0 | 4.01058 | 2.18962 | 1.36067 | 2.72742 | |
| 2 | 3.37182 | 4.01058 | 0 | 4.73579 | 3.52097 | 4.77002 | ⋮ |
| 3 | 1.78964 | 2.18962 | 4.73579 | 0 | 3.46797 | 4.91533 | |
| 4 | 1.80147 | 1.36067 | 3.52097 | 3.46797 | 0 | 1.66318 | |
| 5 | 3.40832 | 2.72742 | 4.77002 | 4.91533 | 1.66318 | 0 | ⋯ |
| ... | | | | ... | | | ... |

**§ NOW THAT YOU HAVE ALL THE DISTANCES BETWEEN PARKS PLEASE ANSWER THE FOLLOWING QUESTIONS:**

1. What are the 5 closest parks to 32nd Street Park?
2. What is the park with the *highest* average distance from its closest 12 parks?
3. What is the park with the *lowest* average distance from its closest 12 parks?