**SI 206 – Data-Oriented Programming I | Winter 2017**

# Final Project Instructions

---

## Please read this entire document carefully!

## Table of Contents



## Overview

**You have 3 options for your final project:**

**(1)** BeautifulSoup & National Parks

**(2)** Twitter & NYTimes

**(3)** Propose your own

In total, your final project is worth 3000 points. Its workload will be somewhat like 1.5 "projects" in general, but the process of working on this project will be broken up into a few chunks, following the process of completing a project that we've discussed in class.

You will have 2 "assignment checkpoints" to submit before the final project. The first one is a homework assignment, separate from the final project points. See below for more detail.

You must pick either Option 1, Option 2, or Option 3 - Propose your own, which may be a slight alteration on one of the options 1 or 2, or totally different! If you propose your own project, you must answer a few more questions in the first final project assignment to propose the project, which we will either accept, accept with revisions, or say no to.

You will still get points for the first project planning assignment even if you do not end up using the work you do in your final product -- that's part of the process!

*You may change your project at any time up* ***until the second part of the final project assignment (the Complete Data Access + Database Loading assignment) is due***, but we strongly suggest you make a decision and stick to it. That way you'll have more time to address and work out problems.

It's great to be excited about the project, and we hope you will be! You should think about putting your own twist on whatever option you do that will make you excited about it, if that is something you want to do. But we'd rather you get the learning objectives from this and feel okay about the process and learn from it, than see you pick an idea you're excited about in theory that you will find *extremely* stressful or not be able to complete.

**Below, you will find:**

- Details about collaboration & academic honesty on the final project
- A timeline for project deadlines, detailing what is due at what time for the final project
- Details on each of the three project options (1), (2), and (3)
- Details on overall project requirements

All template files you will need for this project will be found on Canvas, in Files > Projects > Final Project. (Not all of them are uploaded as of 4/5/17, but will be soon)

# Please read this entire document carefully! (Seriously.)

---

## Project requirements

In any option you pick, your project must include the following:

- **Getting data from 2 sources (like Option 2) or from 1 complex source (like Option 1), at least one of which must be an Internet source, accessed via an API or with BeautifulSoup.** If you choose your own source, you must justify its complexity such that your project will have equivalent challenge and cover the same learning objectives as options (1) and (2) do.
- **Caching all data you get from the internet, whether from an API or from BeautifulSoup.** You must set up a caching pattern, and cache all data from your project in a file called `206_final_project_cache.json`.
- **Defining at least one class in Python with at least 3 instance variables and at least 2 methods in addition to the constructor.** At least one instance of this class must be created, and each of the additional methods you define must be invoked and used at least once. You must create methods for a reason -- things that have effect upon your program. (See specifics about options below)
- **Loading data into a database file with at least 3 tables, which have relationships to one another.** In other words, it must be possible to make many different INNER JOIN queries among these database tables, reasonably.
- **You must make at least 3 queries to your database, resulting in information that would have been more difficult to combine without use of the database**, **and use the results of the queries to process your data. At least one of those queries should be a JOIN query (of any level of complexity).** In other words, it would not count to get data, accumulate a list of, say, book titles, load the titles into a Books table in your database, and then make a query to get all of the book titles from the database. You didn't need a query for that! But if you were to get all the book titles by authors that had been nominated for >3 awards, based on a separate Authors table that had a relationship to the Books table -- that would count.
- **You must process the data you gather and store and extract from the database in at least four of the following ways**:
    - Set / dictionary comprehensions, and/or list comprehensions
    - Using new containers from the **collections** library
    - Using iteration methods from the **itertools** library
    - Accumulation in dictionaries and processing of the data (e.g. counts, lists associated with keys… like **umsi_titles**, but of course something different)

    - Using generator expressions and/or generator functions (recall HW6)
    - Sorting with a key parameter
    - Using the builtins: **map** or **filter** (which each return iterators) in order to filter a sequence or transform a sequence of data
    - Using regular expressions
- **At least 2 test methods pertaining to each function and to each class method you define.** Basically, you must have a test suite for your code! It should cover edge cases, just like we've focused on and/or looked at all semester. You should write your tests first, and, following that, use the iterative process of tests and code that we've discussed in class… Writing tests initially part of one of these upcoming assignments! See more below.
- **You must ultimately provide a complete README file in a clear, readable outline structure, which should be a .PDF or a .txt file.** Like Project 1, you must provide a complete set of documentation for your code. It should not be essay-format (we're a little more strict now that you have more experience writing and using documentation), but rather documentation outline format. We'll show some examples! It must contain all of the information listed in the README REQUIREMENTS.

Options (1) and (2) have suggested opportunity for completing these requirements in one way or another.

---

# Collaboration & Academic Honesty

You may help one another, as always, but **this is *not* a group project or a pair project**.

We will be "diffing" all of the code as part of grading, which means **we will be able to see if your code was copied and pasted from someone else's or copied and pasted with different variable names. That is not acceptable and neither of you will earn points. You must write all your own code and you must be able to explain it.** If you think you are likely to be tempted by this happening because of the way you work with classmates/friends, I suggest that you choose a different project option from your friend, and then help each other with debugging, with ideas, with various things that you each figure out as you work on your projects!

Ask questions! Talk through problems! Debug together! Walk each other through the code you write! Listen to others (staff or classmates) talk about approaches to your problems and what code you should think about! Share problems you're having in person and on Piazza, showing your code and your error messages, and suggest hypotheses about why you're getting semantic errors! TEST, TEST, TEST, & PRINT, PRINT, PRINT. Take it slow, step by step, follow the steps you lay out for yourself and for others too! Print debugging is your friend. A really good friend.

All of these are excellent strategies to make your work easier and more productive. **But make your own original plans within the boundaries of the project option you've chosen, and adjust from those based on others' recommendations, and do not turn in code for your project that anyone else typed, or that you typed simply by copying code over someone's shoulder.**

**You may use external libraries and provided code from any class session or office hours (and in some cases that will be very useful to you!), but you must cite it.** If you use code from class/someone else, put a comment beside that code noting that it is from class/someone else. If you borrow code from elsewhere, cite in your readme in which lines of code you are using someone else's work.

**You cannot fulfill project requirements with someone else's code or from previous HW that we provided for you, or from code in lecture.** You may use it, but you must cite it in a comment that it is not originally by you (we know the code that's been provided… ) and you cannot fulfill requirements by copying it into your project. But you can (and should) use code from section, lecture, past HW... as starting points: "if that worked, how can we change it to make something different happen…", etc.

---

# Timeline

## **Complete Project Plan + Tests** (Due Sunday, April 9 11:59 PM)

**This assignment is worth 500 points and it is NOT included in the 3000 points for the final project. It counts as a HW assignment.** *This assignment may be submitted late for a deduction of 100 points per day late -- if you submit late, you get feedback late!*

The **project plan and tests** assignment consists of a couple pieces:

- **Pick your project option.**
- If you choose Option 3 - Propose Your Own, answer the questions pertaining to proposing your own project in `206_finalproject_plan.txt`
- If you choose Option 1 or 2, **answer the other questions in the `206_finalproject_plan.txt` file**.
- **Write any import statements you believe you'll need** for your project code at the beginning of the `206_project_plan.py` file.
- **Write at least 8 test methods for your project at the end of the `206_project_plan.py` file.** It's OK if you need to change them later! Think, though

-- say for option 1, "I have to write a function that returns HTML string data, what will I call it?" or "I have to write a function that returns a list of HTML strings, what will I call it?" and then, "What test methods should I write to check that that function works? Then, you'll work backward from these. **You should answer the questions in** `206_finalproject_plan.txt` **first, because that will help you think about what exactly your program needs to have, and then you can write tests that will ensure those things are true if they pass!**

Overall, what you submit here will be much like your HW you get from us -- import statements at the top, comments with instructions, sometimes pseudocode/provided code, and then, tests!

You should make sure that the code runs, though probably you'll get ERRORs for all the tests you've written, because you don't have any code yet! We'll get feedback to you ASAP when you submit this file, so you can respond to those comments and begin work on the next part.

**Notes and Suggestions:**

- If you do begin trying out sample code for the project as well as writing out the comments, you can submit it in an additional file if you like. (We won't grade it in detail, because we don't have time to -- but if you'd like to supply it, that's fine!) This is *not required,* but it's always nice to get ahead if you find you have the time.
- Keep in mind that what is required here is **good pseudo code and tests** -- you want a good plan that will help you build a project. You want it to be like an outline with unit tests that we'd give you for a project or HW, but it's also great to get a little more detailed, even if that's after you submit initially -- give yourself more help!
- You will have time during section this week to work on this as well as a discussion assignment related to it.
- **You are not going to be graded on the *correctness* of your ideas, but rather, whether you completed everything asked of you in this assignment.** Being careful and thoughtful about your work on this project plan assignment will be beneficial to you in the not-so-long run, though!
- **THIS IS THE ONLY TIME AT WHICH YOU CAN PROPOSE YOUR OWN PROJECT.** If you wish to, **you must submit answers to these questions by April 9th to get approval from the instructional team**. If you have major concerns, email the team, but exceptions will be rare. You need time to propose your own project, it's not easy!

**You should submit to the Final Project Plan Canvas assignment on April 9th:**

- **Your** `206_project_plan.py` **file, edited**
- You should also submit `206_finalproject_plan.txt` if you are doing Option 3

**(One of the days of lecture in this intervening week, April 9 - 16, will be work time, like office hours, for your project.)**

# **Complete Data Access + Database Loading assignment** (Due Tuesday, April 18, 11:59 PM -- last day of classes) - **600 points**

These 600 points count *within* the final project 3000 points. The completed final project, due on the 25th, will be 2400 points.

**We will have an extra set of office hours during study days at some point during this period of time.**

- Based on your project plan ideas and the feedback you receive on them,
- Write instructions for yourself in comments in the `206_data_access.py` file. You should use your project plan, and the feedback you receive on it, as a basis, and make instructions for yourself, just like we do for you on problem sets.
- Include the tests you wrote for HW9 in this file, but make any alterations to them you need to, based on decisions that you've made during this project plan process.

- *(We just realized this was missing from the document 4/18, but providing for your reference.) Submitting this late is a 50 point deduction the first day with no promises on late feedback. The deduction will double by day late. If you have a major concern, you should already have contacted Jackie!*

Your file at this point should look a lot like a project file WE would give you, except that you will have written it for yourself!

Now...

- Write code in your file `206_data_access.py` that fulfills the following parts of the instructions you've written for yourself:
    - Define a function to get data from each of your sources. (For Option 1 and Option 2, this will be either 1 or 2 functions, depending upon how you have decided to organize your code.) Each function you write should cache data, as you've learned this semester.
- Write code that invokes each function you've just written to get data and store it in a variable. (Consider: what did your instructions tell you to do? What tests have you written? Consider e.g. the tests we wrote for you for **umich_tweets** in project 3, etc…)
- Write code to set up your database file and database tables. (This should accord with the instructions you wrote yourself!)
- Write at least enough code to load data into one of your database tables with a sensible primary key. (Should also accord with the instructions you wrote yourself)
    - However, feel free to continue working / especially if it makes sense to write this code all at once, write code to load data into all of your database tables.

- Include comments in your code file (or in a separate file if you prefer) about anything you are unsure about and point us to them with a comment on your submission, so we can try to help you asynchronously!

**SUBMIT:**

- `206_data_access.py`
- A .json file that holds your cached data

## **Suggested Milestone: Complete first complete draft of documentation + fully complete test suite** (by Friday, April 21)

## **Complete and submit full Final Project, with documentation.** (Due 11:59 PM Tuesday April 25) *This is a hard deadline.*

*This is the last day the registrar allows us to have submissions for this course. If you need to discuss an extension due to a major emergency, or wish to discuss the possibility of an Incomplete in the course, please contact Jackie about this by Thursday, April 20.*

**SUBMIT before 11:59 PM Tuesday April 25** inside a .zip file so the names do not change**:**

- **A README file containing all documentation necessary for us to run and understand your code, in a clear and readable outline format.**
    - This README file should be either a .PDF file or a .TXT file.
    - This README file should explain each element of your code: each function, each class and its methods/what the constructor accepts as input, and what any additional mechanics/processing in your code do.
    - It should also include examples of how to run your code, an explanation of what the result / output of your code running ought to be, and an explanation of all dependencies / any modules that must be installed for it to run. This includes any that we have installed in the process of the course OR any new ones you have used.
    - See below for more detail on this readme file.
- **A screenshot of your code running.** (Show us whatever you want here that will make evident that your code has run for you successfully)
- **A screenshot of your git commits for the project. Y***ou should have at least 5 total git commits for the final project, not 3.* Though we encourage even more! **These 5 git commits (or more) will count for 300 points in total**, like 2 of your earlier "git commit" assignments.

- **The .py code file that contains your code, to run**. (Tell us what it is called in your README!)
    - This should be commented clearly with what each bit of your code does (this is the work that is due on April18th, edited for accuracy by the time you turn in your complete project.
    - It should include all requirements (see project requirements in this document), including at least 1 test method testing each function and each class method.
    - If you have more than one .py file that must be included, of course include each file
- **The .db/.sqlite database file that you have created** to store your database tables/which your project depends on.
- **The .json file that contains your cache data.**
    - If you chose to create more than 1 .json file cache, include all necessary, but we only expect/you only need one!

---

# Documentation Details

- **Your README documentation should be in an outline format**
    - Think: How can you structure this, make it look nice?

**The following are pieces of information you need to include:**

- What option did you pick (1, 2, or 3)
- What does it do? 1-3 sentences
    - Why does it exist / what can you use it for
    - With (any input?), what does it output when run
    - Does it create a database?
- How do you run it?
    - 1 line description / example of how to run the correct file.
- What are its dependencies? You should list these with bulletpoints.
    - Any modules to install with `pip` ?
    - Any particular files you have to have? (e.g. your own `twitter_info.py` file with certain specifications?
- What files are included? Another bulletpoint list.
    - What are their names
    - 1 full sentence or less, what does each do (a sub bullet point or in parens after the file name)
- Each function
    - name

    - input
        - required
        - optional
    - return value
    - behavior if applicable (if you need to explain that the function does something that is not evident from its return value, e.g. that it caches data)
- Each class
    - name
    - what does one instance represent
    - required constructor input
    - each method
        - name
        - any input besides `self`
        - behavior (does it change/add a new instance var? for example)
        - return value if any, or specify `None` if None
- Database creation
    - What tables does the database have
    - For each table
        - What does each row represent
        - What attributes in each row
- Data manipulation code
    - What else does the code do?
    - How is it useful?
    - What will it show you?
    - What should a user expect?
    - Note that each of these things should be 1-2 sentences at *most*. Just a clear idea: what's going on here?
- Why did you choose to do this project?
    - Because this is a class and we want to know so we have context for your work.
- If you chose option 3
    - Make sure you explain your end result particularly clearly
- **Specific things to note for SI 206 -- ok to put this in a section -- copy this right into a .txt file and fill in the correct line numbers**
    - Line(s) on which each of your data gathering functions begin(s):
    - Line(s) on which your class definition(s) begin(s):
    - Line(s) where your database is created in the program:
    - Line(s) of code that load data into your database:
    - Line(s) of code (approx) where your data processing code occurs — where in the file can we see all the processing techniques you used?
    - Line(s) of code that generate the output.
    - OK to be approximate here — ok if it ends up off by 1 or 2. Make it easy for us to find!
- Anything else brief and clear we need to know?

- *Show us a screenshot / image of your project running. This can be included in a PDF file or included as a separate file*
- *Remember that you must also submit proof of at least 5 git commits for work on the final project, in a screenshot (worth 300 git commits points)*

---

# **Option 1** - BeautifulSoup and National Parks

*NOTE: We'll review BeautifulSoup in class on the week of April 11*

- Get data from the National Parks website using BeautifulSoup, which has a lot of hierarchy of links inside it. The root page is: https://www.nps.gov/index.htm
    - Write a function to get and cache HTML data about each national park/monument from every state.
    - I recommend storing the data in a big list, so you can return a list of HTML strings which each represent a park/site/monument from a function. Each HTML string should contain all the data you need, like what state(s) the park is in, the park's name, etc… so you could pass each HTML string straight into a class constructor!
    - Write a function to get and cache HTML data from each of the articles on the front page of the main NPS website. An example of an article can be found at https://home.nps.gov/ever/wilderness.htm -- which you get to by clicking on this displayed link on the front page of NPS.gov, for example:

**The Woman Who Saved the Everglades ›**

Marjory Stoneman Douglas victory "The Everglades: River of Grass", published in 1947 -- the year Everglades National Park was established.

    - I recommend storing these HTML strings, each of which represents 1 article, in a big list, so you have one big list of HTML strings that represent NPS articles, and returning that lis from your function.

*(Remember that it is possible for the site to be updated and changed! New park data, new articles... So if I were you, I'd write both of those functions to get and cache all the HTML you want, similar to what you did for Project 2, and make sure they work properly, before you start playing around with the data, and then, if you want new data, you can delete your cache file or rename it.)*

- Define a class `NationalPark` which accepts an HTML-formatted string as input, and uses BeautifulSoup data parsing to access the data you want and store it in instance variables, etc.

- This class should have:
    - At least 3 instance variables
    - At least 2 methods besides a constructor
    - **Data you might consider either storing as an instance variable or using a method to compute:**
        - the street address or text location (e.g. "AL,AR,GA,IL,KY,MO,NC,OK,TN" or "Tuskegee, AL", etc, depending...) of the visitor's center of each park/monument,
        - the state/list of states in which the national park/monument/site is in,
        - the contact phone number for the park/monument,
        - the link to plan your visit to the park,

        - the description of the park/monument,
        - information about the language/characters/words used in that description, information about alerts pertaining to that park/monument… Explore the website, see what's interesting that falls within these parameters! Decide what input you want your NationalPark class to have, what you want its instance variables to represent, and what you want its methods to do.

- Define a class `Article` that accepts a big HTML string representing one article on the NPS.gov front page. E.g. the one at https://home.nps.gov/ever/wilderness.htm, the one at https://www.nps.gov/subjects/worldwari/index.htm ...
    - This class should have at least 2 instance variables. Instance variables that might be useful for this class could be: **title, text, description**
    - You could also define methods for this class if you think they will be useful for your plan! (You do not have to.)

- **Write code to create a list of instances of the NationalPark class.**
    - This might be a good place to use a comprehension structure… and go through the data you saved about the national parks from your first function! You'll be able to use this list of instances when you load data into your database.

- **Write code to create a list of instances of the Article class.**
    - The second function described above will be pretty useful here…
    - You'll be able to use these lists of instances when you load data into your database tables!

- **Use requests / BeautifulSoup on this page:** https://www.currentresults.com/Weather/US/average-annual-state-temperatures.php **to gather each state's average temperature (in Fahrenheit OR Celsius as you prefer)** and store them in a dictionary with key:value pairs that represent states as keys and average-temps as their associated values

- **Create a database file with at least 3 tables.** Suggestions:
    - A **Parks** table (for parks and monuments) containing information specific to each park like name, a reference to the States table, a description, and/or anything else you find interesting that is specific to each park/monument/site...
    - A **States** table with info about each state, like name, abbrv, avg temp…
        - (**note** that if a park is in multiple states and has a location like "AL,AR,GA,IL,KY,MO,NC,OK,TN", you have to decide how to handle this relationship. This is the sort of thing you describe your choice about in your readme! "I decided to use just the first state listed in the list of states for any park in multiple states for the relationship to the states table…")
    - An **Articles** table with info about each article from the front page of the NPS website, e.g. the article title and the text of the article (HINT: the text of the article

is a string, and you can always find out if smaller strings are inside larger strings… say, if you wanted to find out which parks were mentioned in an article…)
        - **NOTE:** You should explore the data you get about Articles and use that to decide what queries to make and how to process your data. Look through it! What's cool about it?
        - If none of the articles mention parks by name, maybe it would be more interesting to determine the most common names/proper nouns in all the articles than it would be to find out what the titles of articles are where the articles mention your favorite parks...

    - **Each table in your database must have a primary key** (you can decide what each table's primary key is -- whether you use a piece of unique information or use a default integer primary key ID) and therefore should not be able to include duplicate rows.

- **Load data into your database tables!**
    - Your list of NationalPark instances, list of Article instances, and the dictionary with states and their average temperatures you created earlier in your program are likely to be very useful here…
    - Consider: you could have a method in each class, NationalPark and Article, that generates a tuple of the information you'll want to load into one row in a database table..!

- **Use some of the tools/skills you've learned to make queries and process your data to produce some output.**
    - Write a summary text file of the following data, clearly represented in a text file. OK to replace one of these with another interesting result if you prefer it!
        - The 5 states/territories that have the largest numbers of national parks(/monuments/sites)
        - The most common word (besides "the", "a", "who", "that", or other "stopwords" you want to take out) in all of the articles (this might be a good place to use one of the collections containers or dictionary accumulation, or the filter method…)
        - Pick 3-10 parks/monuments/sites you particularly like, and find out, using queries/data processing, whether any articles in your Articles table mention them. Once you find a few that have corresponding articles, you can complete the next step:
        - Write to your file each of those parks/monuments/sites and what the titles and URLs of the articles that mention them are, so someone looking at the file would be able to copy or click on the URL if that title sounded interesting to them, knowing that the article mentioned e.g. the Everglades, or Yosemite, etc.

        - And/or any other piece of interesting data processing you'd like to do and share about this data! Be creative!

---

# **Option 2** - API Mashup: Twitter & OMDB

- **Write function(s) to get and cache data from Twitter:**
    - A function to get and cache data based on a search term
    - A function to get and cache data about a Twitter user

- **Write function(s) to get and cache data from the OMDB API** with a movie title search (Test it out!) http://www.omdbapi.com

- Required: **define a class Movie**.
    - The constructor should accept a dictionary that represents a movie.
    - It should have at least 3 instance variables and at least 2 methods besides the constructor (You might want one of them to be a __str__ method…)
    - Data you'll definitely need to save about each movie in the class, either by saving the data directly in instance variables or defining methods that will compute these things using the instance variable values:
        - Its title
        - Its director
        - Its IMDB rating
        - A list of its actors (check out the data and consider how to get a list of strings that represent actor names!)
        - The number of languages in the movie
        - Anything else you're interested in...

- **Optional but recommended:** create a class or classes to handle the Twitter data and make it easier for you. E.g. a class Tweet, and/or a class TwitterUser… Check out the data you want in your database tables to make plans for those if you choose to. Otherwise, you'll have to find some other way of handling this data...

- **Pick at least 3 movie title search terms for OMDB. Put those strings in a list.** (Depending upon your interests, and what results you want to find, it might be nice to have more than 3! But probably a good idea to test with 3, and then try to write code with more.)

- **Make a request to OMDB on each of those 3 search terms, using your function**, accumulate all the dictionaries you get from doing this, each representing one movie, into a list.

- Using that list of dictionaries, **create a list of instances of class Movie**.

- **Make invocations to your Twitter functions.**
    - Use your Twitter search function to search for (your choice): either each of the directors of those three movies, or one star actor in each of those three movies, or each of the titles of those three movies.
        - NOTE: It may be useful to have a class Tweet for this reason, because you can make a list of instances of the Tweet class each time you get data, and then concatenate (squish) the lists together…
        - Save either a list of instances of class Tweet, or a list of Tweet dictionaries, or a list of Tweet info tuples, in a variable to use later.
    - Then, **use your function to access data about a Twitter use**r to **get information about each of the Users** in the "neighborhood", as it's called in social network studies -- **every user who posted any of the tweets you retrieved and every user who is mentioned in them.**
        - It may be useful to have a class TwitterUser for this reason, as you can then create lists of user instances instead of lists of dictionaries! But you can manage the data in dictionaries as well, if you like.
        - Save either a resulting list of instances of your class TwitterUser or a list of User-representative dictionaries in a variable.
        - Make sure your code accesses the whole neighborhood -- every tweet poster and

- **Create a database file with 3 tables:**
    - A **Tweets** table
    - A **Users** table (for Twitter users)
    - A **Movies** table

    - **Your Tweets table should hold in each row:**
        - Tweet text
        - Tweet ID (primary key)
        - The user who posted the tweet (represented by a reference to the users table)
        - The movie search this tweet came from (represented by a reference to the movies table)
        - Number favorites
        - Number retweets
    - **Your Users table should hold in each row:**
        - User ID (primary key)

        - User screen name
        - Number of favorites that user has ever made
        - Any other information you want to deal with or find interesting…
    - **Your Movies table should hold in each row:**
        - ID (primary key) (NOTE title is dangerous for a primary key, 2 movies could have the same title!)
        - Title of the movie
        - Director of the movie
        - Number of languages the movie has
        - IMDB rating of the movie
        - The top billed (first in the list) actor in the movie
        - Any other info you find interesting…

- **Load data into your database!**
    - Your lists of instances/dictionaries/etc will be useful here… Think through the process step by step, much like code you saw in class, and you'll be just fine.

- **Process the data and create an output file!**
    - **Make queries to the database to grab intersections of data, and then use at least four of the processing mechanisms in the project requirements to find out something interesting or cool or weird about it.**
    - You could focus on language: most common characters, words, etc… You could focus on names, or on twitter stats (favs, rts, number of mentions, or hashtags…) whatever you want.
    - **Make sure that each processing technique you use has a *reason* for using it to get your result, and you're not just using a technique and then doing nothing with the result.**
        - E.g. a set comprehension is good when you want a sequence with no duplicates, a dictionary comprehension is good if there's an easy relationship between a key-value pair based on individual elements of another sequence
        - Collecting information together in a new structure is a good use of Collections containers
        - Dictionary accumulation and sorting are useful to find commonalities as well, or to associate things with one another,
            - e.g. if the values of keys that represent twitter screennames were lists or sets of unique words they'd used in the tweets in your data set, and you wanted to find out how long each of those lists was…

    - **And, finally, write that data to a text file -- a sort of "summary stats" page with a clear title, e.g**. **<List your 3 movie titles> + Twitter summary, <current date>** or something, and have each result of your data processing clearly written to the file neatly below that, on several different lines.

---

# **Option 3** - Propose your own project

**As long as you:**

- Will fulfill all of the listed project requirements (see above)
- Use at least one internet source and at least 2 complex sources overall
    - (A second source that is a simple CSV is not acceptable, but a second source that is a complex CSV e.g. many columns of stock information, or sports stats information, or the full text of a book from Project Gutenberg, or some other form of complex data you can download e.g. from an open data website or government website, is OK)

**You can propose anything you want in your project plan, due April 9th, answering the questions. We will then either:**

- Approve it
- Approve it with concerns/edits to address
- Not approve it. If we do not approve it, you can discuss your concerns so it can be sufficiently altered, or you can choose Option 1 or 2.
    - You will still get points for completing the project plan, but this may be harder -- so if you really want to propose your own project, you should put thought into whether you have a plan forming about how to achieve all of the project requirements! We can help with that -- talk to us!