Assignment #5 - Data from the Web, Combining Data

👀 Click on this link: https://classroom.github.com/a/bNhiysGD (https://classroom.github.com/a/bNhiysGD) to accept this assignment in GitHub classroom. This will create your homework repository. Clone your new repository.

In this homework, you'll:

1. "Screen scrape" data from a web site
   - Using `requests` and `BeautifulSoup` to download and parse html
   - Use `merge` to join DataFrames by a key
2. Work with JSON / Use an API
   - Use the `json` module with `requests`

Part 1 - Combine Course Info with Requirements

```
           ///////|
          /////// |
         ///////  |
        |~~~~~|   |
        |=====|   |
        |  D  |   |
        |  A  |   |
        |  T  |   |
        |  A  |   |
        |  B  |   |
        |  A  |   |
        |  E  |   |
        |     |   '
    jgs |=====| /
        '_____'`
```

ASCII art source (http://www.oocities.org/spunk1111/school.htm), with modifications

Prep

In this part of the assignment, you'll work on parsing html with a library and regular expressions to extract course information data. Additionally, you'll use the `merge` function to put together data from two different DataFrames based on key.

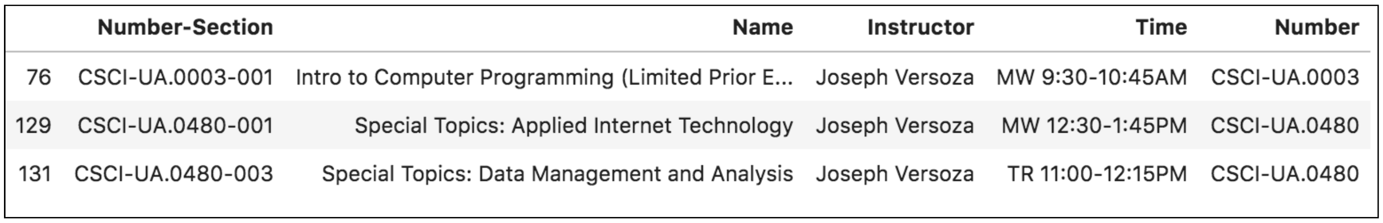
1. Download the course schedule for this semester (https://cs.nyu.edu/dynamic/courses/schedule/) by right-clicking and choosing "Save As". Save this in the root of your project repository; give it a short, but descriptive file name.
2. Download the course catalog (https://cs.nyu.edu/dynamic/courses/catalog/) by right-clicking and choosing "Save As". Save this in the root of your project repository; give it a short, but descriptive file name.
3. Make sure to install any modules necessary for working with html
4. Open up the empty notebook, `courses.ipynb` to work on this part of the assignment

Instructions

1. Read the course schedule into a DataFrame

- the frame should have the following columns:
  - Number-Section: the course number and section number
    - Use your discretion to deal with *issues* encountered here (for example, some issues may include courses listed with two different course numbers!), but document what you've done and give some rationale for your methodology

        - ⚠️ **you may encounter an invisible space character (a zero width space) in the course number depending on how you extract the text (view the markup in Chrome's web inspector tools or try printing it out in Python)**
        - the easiest way to deal with this is to replace it with emtpy string (assuming `s` contains the zero width space): `s.replace('\u200b', '')`
    - Name: the name of the course
    - Instructor: the name of the professor
        - Again, there may be issues here, such as multiple instructors; use your discretion, but describe what you've done and why
    - Time: the day(s) and time(s) the course meets
    - (once you read in the data, you'll add a couple of rows)
- this is what a portion of the `DataFrame` may look like (note, the courses are from a past semester, though)



	Number-Section	Name	Instructor	Time	Number
76	CSCI-UA.0003-001	Intro to Computer Programming (Limited Prior E...	Joseph Versoza	MW 9:30-10:45AM	CSCI-UA.0003
129	CSCI-UA.0480-001	Special Topics: Applied Internet Technology	Joseph Versoza	MW 12:30-1:45PM	CSCI-UA.0480
131	CSCI-UA.0480-003	Special Topics: Data Management and Analysis	Joseph Versoza	TR 11:00-12:15PM	CSCI-UA.0480

- once you've read in your data, break apart the `Number-Section` column into two separate columns: `Number` and `Section`
    - `Number` is something *like* `CSCI-UA.0480`
    - `Section` is something *like* `001`
    - try to **use regular expressions with groups to do this**
        - the `str` accessor method to use is `extract`
        - check out the end of the regex slides (../slides/python/regex.html) or the official pandas docs (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.str.extract.html)
- show:
    - `info` to show the data types and counts
    - the first 5 rows
    - the last 5 rows
    - a random sampling of 5 rows

General Workflow and Hints

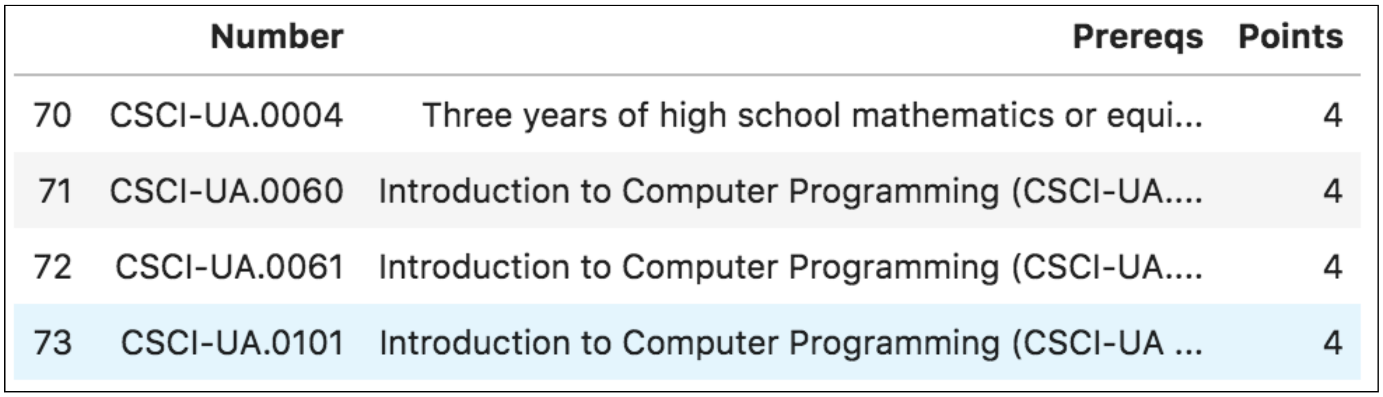
- read in the html file
- parse the html
- pick out /extract the data
- it's very useful to use your browser's web inspector tools (right-click on element and inspect to see html and parent, sibling, and child elements
- ⚠️ see this guide on using web inspector tools for chrome (https://developers.google.com/web/tools/chrome-devtools/dom/)
- note that in the parsing library we cover in the slides...
- `select`, `select_one`, `find`, etc. can be called on an element to find elements nested within it
    - for example, if `my_div` is the result of calling `select_one`
    - ... and `my_div` is `<div><p class="foo">one</p><p>two</p></div>`
    - ...you can select elements within `my_div` with `my_div.select('p')`
- if you just want the first nested element
    - you can dot (`.`) the parent element
    - ... and use the name of the nested element next
    - for example if, `my_h1` is `<h1><a>foo</a> bar</h1>`
    - `my_h1.a` can be used to access the nested `a`
- if a collection of elements is returned, you can index into it with `[]` (essentially, a list)
- `getText()`, `.text` or `.get_text()` returns all of the text within an element (even nested elements!)
- pay attention to patterns in data (what *makes* a course number, what element is a course number usually in?)
- also helpful to print out elements themselves (without using `.text`) to see what element was actually selected
- some example markup and parsing code:

```
<section class='container'>
  <div class='row'>
    <p>Course Name: <a href="cs123.html">CS-123<a/></p>
    <p>Alice Ahn</p>
  </div>
  <div class='row'>'
    <p>Course Name: <a href="cs456.html">CS-456</a></p>
    <p>Bob Bernstein</p>
  </div>
</container>
```

```
# get every element in the section element with class container
# that has the class attribute, row
rows = rom.select('section.container .row')
for row in rows:
    # looping over this selection gives us each div element

    # within each div element, find the paragraphs
    paragraphs = row.select('p')

    # show ALL text in first paragraph in current div
    print(paragraphs[0].text) # on 1st iteration: Coure Name: CS-123

    # dotting an element with a tag name retrieves the
    # first nested element with that tag name
    print(paragraphs[0].a.text) # on 1st iteration: CS-123

    print(paragraphs[1].text) # on 1st iteration: Alice Ahn
```

2. Read the course catalog into a DataFrame

- the frame should have the following columns:
  - Number: the course number
  - Prereqs: a text description of the prerequisites
  - Points: the number of credits
- here's example of a `DataFrame` with some course catalog rows (again, the data is from another semester)



	Number	Prereqs	Points
70	CSCI-UA.0004	Three years of high school mathematics or equi...	4
71	CSCI-UA.0060	Introduction to Computer Programming (CSCI-UA....	4
72	CSCI-UA.0061	Introduction to Computer Programming (CSCI-UA....	4
73	CSCI-UA.0101	Introduction to Computer Programming (CSCI-UA ...	4

- show:
  - `info` to show the data types and counts
  - the first 5 rows
  - the last 5 rows
  - a random sampling of 5 rows
- use a similar parsing strategy as above to read in this DataFrame

3. Put together both DataFrames

- create a new DataFrame by....

- finding a way to show all scheduled classes in the semester along with their points and prereqs
- only show the following columns, in this order:
  - Number: course number
  - Name: course name
  - Instructor: professor's name
  - Time: meeting time
  - Prereqs: course prerequisites
  - Points: number of credits
- hints:
  - use `pd.merge` to do this (../slides/python/pandas-join-combine.html)
  - `how=left` will keep all rows in the first `DataFrame`

4. Conclusion

- did you spot any anomalies, discrepancies, or unexpected data or relationships between data?
- if so, in a markdown cell, describe any problem(s) you saw
- additionally, describe how you might fix them (or *if you already fixed them*!)
- lastly, based on the resulting `DataFrame`, describe the behavior of `how=left` on these particular `DataFrames`
- if you need to see all rows, use `pd.set_option('display.max_rows', 200)`

Part 2 - Using an API

Overview

In this part of the assignment, you'll request data from a server in json format, parse it, and load it into a `DataFrame`. Using this `DataFrame` you'll use aggregations to produce a report.

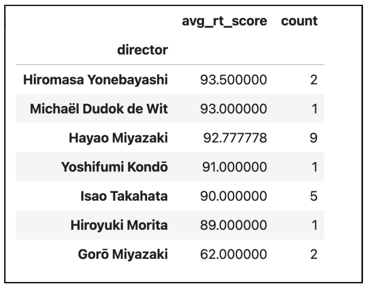
The data set is composed of films from the Japanese animation film studio, Studio Ghibli (https://en.wikipedia.org/wiki/Studio_Ghibli).

It is being served from a mirror of the data on `linserv1.cims.nyu.edu`. Note, however, that the original data is from https://ghibliapi.herokuapp.com/ (https://ghibliapi.herokuapp.com/), which is under an MIT License (https://github.com/janaipakos/ghibliapi/blob/master/LICENSE). This is mirrored so that we do not overwhelm the original data source with requests.

Instructions

The goal of the assignment is to create a report showing director's names, the number of Ghibli films that the directors was involved in, and the average rotten tomatoes score of the Studio Ghibli films made by that director.

The expected output is shown below:



	avg_rt_score	count
director		
Hiromasa Yonebayashi	93.500000	2
Michaël Dudok de Wit	93.000000	1
Hayao Miyazaki	92.777778	9
Yoshifumi Kondō	91.000000	1
Isao Takahata	90.000000	5
Hiroyuki Morita	89.000000	1
Gorō Miyazaki	62.000000	2

1. Retrieve the data, and examine it.

- In `films.ipynb', programmatically retrieve one page of json from this URL: http://linserv1.cims.nyu.edu:10000/films?_page=1 (http://linserv1.cims.nyu.edu:10000/films?_page=1)
- You can use `requests` to do this
  - you can use the `json` module to manually parse the response content

- **Or** .... use a feature of the `requests` module that allows immediate parsing of a json response by calling the `json()` method
- r = requests.get('some.url')
- d = r.json() # parses json into dictionary!

- Examine the keys and values of the dictionary
- In a markdown cell, write out what keys you may be interested in to create the report specified above
- Try incrementing the last number in the url where page is `1` ... do you get different results?
- In a markdown cell, describe what happens when you modify the url

2. Load the data into a `DataFrame`

1. Make a request to http://linserv1.cims.nyu.edu:10000/films?_page=1 (http://linserv1.cims.nyu.edu:10000/films?_page=1) again, but this time, load the result into a `DataFrame`
2. Continue collecting additional data and adding to the `DataFrame` until there is no more data to retrieve

3. Report

Create a report that shows:

- the directors' names as the index (Note that the `index.name` can be set to get what appears to be a title for the index (../slides/python/pandas-basics.html#62))
- the average rotten tomatoes score (review aggregator website)
- the number of films directed
- `concat` and `groupby` may be helpful