



S-109A Introduction to Data Science

Homework 1

Harvard University

Summer 2018

Instructors: Pavlos Protopapas and Kevin Rader

Main Theme: Data Collection - Web Scraping - Data Parsing

Learning Objectives

In this homework, your goal is to learn how to acquire, parse, clean, and analyze data. Initially you read the data from a file, then you scrape them directly from a website. You look for specific pieces of information by parsing the data, you clean the data to prepare them for analysis, and finally, you answer some questions.

Instructions

- To submit your assignment follow the instructions given in Canvas.
- The deliverables in Canvas are: a) This python notebook with your code and answers, b) a .pdf version of this notebook, c) The BibTex file you created. d) The JSON file you created.
- Exercise **responsible scraping**. Web servers can become slow or unresponsive if they receive too many requests from the same source in a short amount of time. Use a delay of 10 seconds between requests in your code. This helps not to get blocked by the target website. Run the webpage fetching part of the homework only once and do not re-run after you have saved the results in the JSON file (details below).
- Web scraping requests can take several minutes. This is another reason why you should not wait until the last minute to do this homework.

```
In [466]: # Import the necessary libraries
%matplotlib inline
import numpy as np
import scipy as sp
import matplotlib as mpl
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import pandas as pd
import time
pd.set_option('display.width', 500)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repr_html', True)
```

Part A [50 pts]: Help a professor convert his publications to bibTex

Overview

In Part 1 your goal is to parse the HTML page of a Professor containing some of his publications, and answer some questions. This page is provided to you in the file `data/publist_super_clean.html`. There are 44 publications in descending order from No. 244 to No. 200.

You are to use python's **regular expressions**, a powerful way of parsing text. You may **not** use any parsing tool such as Beautiful Soup yet. In doing so you will get more familiar with three of the common file formats for storing and transferring data, which are:

- CSV, a text-based file format used for storing tabular data that are separated by some delimiter, usually comma or space.
- HTML/XML, the stuff the web is made of.
- JavaScript Object Notation(JSON), a text-based open standard designed for transmitting structured data over the web.

Question 1: Parsing using Regular Expressions

1.1 Write a function called `get_pubs` that takes a `.html` filename as an input and returns a string containing the HTML page in this file (see definition below). Call this function using `data/publist_super_clean.html` as input and name the returned string `prof_pubs`.

1.2 Calculate how many times the author named 'C.M. Friend' appears in the list of publications.

1.3 Find all unique journals and copy them in a variable named `journals`.

1.4 Create a list named `pub_authors` whose elements are strings containing the authors' names for each paper.

Hints

- Look for patterns in the HTML tags that reveal where each piece of information such as the title of the paper, the names of the authors, the journal name, is stored. For example, you might notice that the journal name(s) is contained between the <I> HTML tag.
- Each publication has multiple authors.
- C.M. Friend also shows up as Cynthia M. Friend in the file. Count just C. M. Friend .
- There is a comma at the end of the string of authors. You can choose to keep it in the string or remove it and put it back when you write the string as a BibTex entry.
- You want to remove duplicates from the list of journals.

Resources

- **Regular expressions:** a) [\(https://docs.python.org/3.3/library/re.html\), b\) \[\\(https://regexone.com\\), and c\\) \\[\\\(https://docs.python.org/3/howto/regex.html\\\).\\]\\(https://docs.python.org/3/howto/regex.html\\)\]\(https://regexone.com\)](https://docs.python.org/3.3/library/re.html)
- **HTML:** if you are not familiar with HTML see <https://www.w3schools.com/html/> (<https://www.w3schools.com/html/>) or one of the many tutorials on the internet.
- **Document Object Model (DOM):** for more on this programming interface for HTML and XML documents see https://www.w3schools.com/js/js_htmldom.asp (https://www.w3schools.com/js/js_htmldom.asp).

1.1

In [467]: # Import the regular expressions Library
`import re`

In [468]: # Use this file
`pub_filename = 'data/publist_super_clean.html'`

In [469]: # Definition of get_pubs
`def get_pubs(filename: str) -> str:`

 `'''Open the file using the filename.`

 `Args:`
 `filename: A string name of the file.`

 `Returns:`
 `A string containing the HTML page ready to be parsed.`
 `...`
 `with open(filename, 'r') as f:`
 `return f.read()`

`prof_pubs = get_pubs(pub_filename)`

In [470]: # Check your code

```
print(prof_pubs)
```

```
<I>Phys. Rev. Lett. </I> <B>102</B>, 149703 (2009).  
<BR>  
</LI>  
</OL>
```

```
<OL START=205>  
<LI>  
<A HREF="Papers/PhysRevE_79_030901RC_2009.pdf" target="paper205">  
"Quantized biopolymer translocation through nanopores: Departure from simple scaling"  
</A>  
<BR> S. Melchionna, M. Bernaschi, M. Fyta, E. Kaxiras and S. Succi,  
<I>Phys. Rev. E - Rap. Comm. </I> <B>79</B>, 030901 (2009).  
<BR>  
</LI>  
</OL>
```

```
<OL START=204>  
<LI>  
<A HREF="Papers/PhvsRevLett 102 095504 2009.pdf" target="paper204">
```

You should see an HTML page

```

<LI>
<A HREF="Papers/2011/PhysRevB_84_125411_2011.pdf" target="paper244">
&quot;Approaching the intrinsic band gap in suspended high-mobility graph
ene nanoribbons&quot;</A>
<BR>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang
Zhang, Mark Ming-Cheng Cheng,
<I>PHYSICAL REVIEW B </I> <b>84</b>, 125411 (2011)
<BR>
</LI>
</OL>

<OL START=243>
<LI>
<A HREF="Papers/2011/PhysRevB_84_035325_2011.pdf" target="paper243">
&quot;Effect of symmetry breaking on the optical absorption of semiconduc
tor nanoparticles&quot;</A>
<BR>JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
<I>PHYSICAL REVIEW B </I> <b>84</b>, 035325 (2011)
<BR>
</LI>
</OL>

<OL START=242>
<LI>
<A HREF="Papers/2011/PhysRevB_83_054204_2011.pdf" target="paper242">
&quot;Influence of CH2 content and network defects on the elastic prop
erties of organosilicate glasses&quot;</A>
<BR>Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras,
<I>PHYSICAL REVIEW B </I> <b>83</b>, 054204 (2011)
<BR>
</LI>
</OL>

```

1.2

```
In [471]: # Find C.M. Friend
friend_list = re.findall(r"C.M. Friend", prof_pubs)

# Print answer
print("C.M. Friend appears {} times in the publications.".format(len(friend_list))
```

C.M. Friend appears 5 times in the publications.

1.3

```
In [472]: # Utilized a set data structure to find all unique journals
journals= set(re.findall(r"\s+<I>(.*)</I>", prof_pubs))
```

```
In [473]: # Check your code: print journals
journals
```

```
Out[473]: {'2010 ACM/IEEE International Conference for High Performance ',
'ACSNano. ',
'Acta Mater. ',
'Catal. Sci. Technol. ',
'Chem. Eur. J. ',
'Comp. Phys. Comm. ',
'Concurrency Computat.: Pract. Exper. ',
'Energy & Environmental Sci. ',
'Int. J. Cardiovasc. Imaging ',
'J. Chem. Phys. ',
'J. Chem. Theory Comput. ',
'J. Phys. Chem. B ',
'J. Phys. Chem. C ',
'J. Phys. Chem. Lett. ',
'J. Stat. Mech: Th. and Exper. ',
'Langmuir ',
'Molec. Phys. ',
'Nano Lett. ',
'NanoLett. ',
'New J. Phys. ',
'New Journal of Physics ',
'PHYSICAL REVIEW B ',
'Phil. Trans. R. Soc. A ',
'Phys. Rev. B ',
'Phys. Rev. E - Rap. Comm. ',
'Phys. Rev. Lett. ',
'Sci. Model. Simul. ',
'Sol. St. Comm. ',
'Top. Catal. '}
```

Your output should look like this (remember, no duplicates):

```
'ACSNano.',  
'Ab initio',  
'Ab-initio',  
'Acta Mater.',  
'Acta Materialia',  
'Appl. Phys. Lett.',  
'Applied Surface Science',  
'Biophysical J.',  
'Biosensing Using Nanomaterials',  
  
...  
  
'Solid State Physics',  
'Superlattices and Microstructures',  
'Surf. Sci.',  
'Surf. Sci. Lett.',  
'Surface Science',  
'Surface Review and Letters',  
'Surface Sci. Lett.',  
'Surface Science Lett.',  
'Thin Solid Films',  
'Top. Catal.',  
'Z'}
```

1.4

```
In [474]: # All authors are found between the <BR> and <I> tags  
pub_authors = set(re.findall(r"<BR>\s*\W*(.*?)\n<I>", prof_pubs))
```

```
In [475]: # Check your code: print the list of strings containing the author(s)' names
for item in pub_authors:
    print (item)
```

Kejie Zhao, Wei L. Wang, John Gregoire, Matt Pharr, Zhigang Suo,
Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mar
k Ming-Cheng Cheng,
H.P. Chen, R.K. Kalia, E. Kaxiras, G. Lu, A. Nakano, K. Nomura,
E.M. Kotsalis, J.H. Walther, E. Kaxiras and P. Koumoutsakos,
W.L. Wang and E. Kaxiras,
Jun Ren, Sheng Meng, Yi-Lin Wang, Xu-Cun Ma, Qi-Kun Xue, Efthimios Kaxiras,
S. Meng and E. Kaxiras,
H. Chen, W.G. Zhu, E. Kaxiras, and Z.Y. Zhang,
M. Fyta, S. Melchionna, M. Bernaschi, E. Kaxiras and S. Succi,
A. Gali, E. Janzen, P. Deak, G. Kresse and E. Kaxiras,
S.K.R.S. Sankaranarayanan, E. Kaxiras and S. Ramanathan,
J R Maze, A Gali, E Togan, Y Chu, A Trifonov,
H. Li, J.M. Knaup, E. Kaxiras and J.J. Vlassak,
T.A. Baker, B.J. Xu, X.Y. Liu, E. Kaxiras and C.M. Friend,
Simone Melchionna, Efthimios Kaxiras, Massimo Bernaschi and Sauro Succi,
S.K.R.S. Sankaranarayanan, E. Kaxiras, S. Ramanathan,
Bingjun Xu, Jan Haubrich, Thomas A. Baker, Efthimios Kaxiras, and Cynthia M. Fr
iend,
T.A. Baker, C.M. Friend and E. Kaxiras,
T.A. Baker, C.M. Friend and E. Kaxiras,
F.J. Rybicki, S. Melchionna, D. Mitsouras, A.U. Coskun, A.G. Whitmore, E. Kaxir
as, S. Succi, P.H. Stone and C.L. Feldman,
T.A. Baker, E. Kaxiras and C.M. Friend,
JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
E. Manousakis, J. Ren, S. Meng and E. Kaxiras,
C.L. Chang, S.K.R.S. Sankaranarayanan, D. Ruzmetov, M.H. Engelhard, E. Kaxiras
and S. Ramanathan,
Masataka Katono, Takeru Bessho, Sheng Meng, Robin Humphry-Baker, Guido Rothenbe
rger,
Sheng Meng, Efthimios Kaxiras, Md. K. Nazeeruddin, and Michael Gratzel,
Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam Gali,
C.E. Lekka, J. Ren, S. Meng and E. Kaxiras,
Jan Haubrich, Efthimios Kaxiras, and Cynthia M. Friend,
A. Peters, S. Melchionna, E. Kaxiras, J. Latt, J. Sircar, S. Succi,
Youdong Mao, Wei L. Wang, Dongguang Wei, Efthimios Kaxiras, and Joseph G. Sodro
ski,
A. Gali and E. Kaxiras,
S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F.J. Rybicki, D. Mitsouras,
A.U. Coskun and C.L. Feldman,
W.L. Wang, O.V. Yazyev, S. Meng and E. Kaxiras,
S. Melchionna, M. Bernaschi, M. Fyta, E. Kaxiras and S. Succi,
E. Kaxiras and S. Succi,
J. Ren, E. Kaxiras and S. Meng,
Thomas D. Kuhne, Tod A. Pascal, Efthimios Kaxiras, and Yousung Jung,
Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras,
Thomas A. Baker, Bingjun Xu, Stephen C. Jensen, Cynthia M. Friend and Efthimios
Kaxiras,
L.A. Agapito, N. Kioussis and E. Kaxiras,
M. Bernaschi, M. Fatica, S. Melchionna, S. Succi and E. Kaxiras,
M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, E. Kaxiras

Your output should look like this (a line for each paper's author(s) string, with or without the comma)

S. Meng and E. Kaxiras,

G. Lu and E. Kaxiras,

E. Kaxiras and S. Yip,

...

Simone Melchionna, Efthimios Kaxiras, Massimo Bernaschi and Sauro Succi,

J R Maze, A Gali, E Togan, Y Chu, A Trifonov,

E Kaxiras, and M D Lukin,

Question 2: Parsing and Converting to bibTex using Beautiful Soup

A lot of the bibliographic and publication information is displayed in various websites in a not-so-structured HTML files. Some publishers prefer to store and transmit this information in a .bibTex file which has the following format:

```
@article { _number_
    author = John Doyle
    title = Interaction between atoms
    URL = Papers/PhysRevB_81_085406_2010.pdf
    journal = Phys. Rev. B
    volume = 81
}

@article
{    author = Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussi
s, Yiyang Zhang, Mark Ming-Cheng Cheng
    title = "Approaching the intrinsic band gap in suspended high-mobili
ty graphene nanoribbons"
    URL = Papers/2011/PhysRevB_84_125411_2011.pdf
    journal = PHYSICAL REVIEW B
    volume = 84
}
```

About the [bibTex format](http://www.bibtex.org) (<http://www.bibtex.org>).

In Question 2 you are given an .html file containing a list of papers scraped from the author's website and you are to write the information into .bibTex format. We used regular expressions for parsing HTML in the previous question but just regular expressions are hard to use in parsing real-life websites. A useful tool is [BeautifulSoup] (<http://www.crummy.com/software/BeautifulSoup/>) (<http://www.crummy.com/software/BeautifulSoup/>) (BS). You will parse the same file, this time using BS, which makes parsing HTML a lot easier.

2.1 Write a function called `make_soup` that accepts a filename for an HTML file and returns a BS object.

2.2 Write a function that reads in the BS object, parses it, converts it into the .bibTex format using python string manipulation and regular expressions, and writes the data into `publist.bib`. You will need to create that file in your folder.

HINT

- Inspect the HTML code for tags that indicate information chunks such as `title` of the paper. You had already done this in Part 1 when you figured out how to get the name of the journal from the HTML code. The `find_all` method of BeautifulSoup might be useful.
- Question 2.2 is better handled if you break the code into functions, each performing a small task such as finding the author(s) for each paper.
- Make sure you catch exceptions when needed.
- Regular expressions are a great tool for string manipulation.

Resources

- [BeautifulSoup Tutorial](https://www.dataquest.io/blog/web-scraping-tutorial-python/) (<https://www.dataquest.io/blog/web-scraping-tutorial-python/>).
- More about the [BibTex format](http://www.bibtex.org) (<http://www.bibtex.org>).

```
In [476]: # Import the necessary libraries
from bs4 import BeautifulSoup
```

2.1

```
In [477]: # Definition of make_soup
def make_soup(filename: str) -> BeautifulSoup:
    '''Open the file and convert into a BS object.

    Args:
        filename: A string name of the file.

    Returns:
        A BS object containing the HTML page.
    ...
    return BeautifulSoup(filename, 'html.parser')

soup = make_soup(prof_pubs)
```

In [478]: # Check your code: print the BeautifulSoup object, you should see an HTML page
print (soup)

```
<a href="Papers/PhysRevLett_102_149703_2009.pdf" target="paper206">  
"Comment on '<i>Ab initio</i> Electronic and Optical Properties of the N-V-Ce  
nter in Diamond'"  
</a>  
<br/> A. Gali and E. Kaxiras,  
<i>Phys. Rev. Lett. </i> <b>102</b>, 149703 (2009).  
<br/>  
</li>  
</ol>  
<ol start="205">  
<li>  
<a href="Papers/PhysRevE_79_030901RC_2009.pdf" target="paper205">  
"Quantized biopolymer translocation through nanopores: Departure from simple  
scaling"  
</a>  
<br/> S. Melchionna, M. Bernaschi, M. Fyta, E. Kaxiras and S. Succi,  
<i>Phys. Rev. E - Rap. Comm. </i> <b>79</b>, 030901 (2009).  
<br/>  
</li>  
</ol>
```

Your output should look like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">

<title>Kaxiras E journal publications</title>
<head>
<meta content="text/html; charset=utf-8" http-equiv="Content-Type"/>
<link href="../styles/style_pubs.css" rel="stylesheet" type="text/css"/>
<meta content="" name="description"/>
<meta content="Kaxiras E, Multiscale Methods, Computational Materials" name="keywords"/>
</head>
<body>
<ol start="244">
<li>
<a href="Papers/2011/PhysRevB_84_125411_2011.pdf" target="paper244">
"Approaching the intrinsic band gap in suspended high-mobility graphene nanoribbons"</a>
<br/>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mark Ming-Cheng Cheng,
<i>PHYSICAL REVIEW B </i> <b>84</b>, 125411 (2011)
<br/>
</li>
</ol>
<ol start="243">
<li>
<a href="Papers/2011/PhysRevB_84_035325_2011.pdf" target="paper243">
"Effect of symmetry breaking on the optical absorption of semiconductor nanoparticles"</a>
<br/>JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
<i>PHYSICAL REVIEW B </i> <b>84</b>, 035325 (2011)
<br/>
</li>
</ol>

...

```

2.2

```
In [479]: def get_authors (soup2: BeautifulSoup) -> str:  
  
    """Retrieves a string containing the authors' names using the article's soup object.  
  
    Args:  
        soup2: A BeautifulSoup object pertaining to a single article.  
  
    Returns:  
        A string containing the names of the authors.  
    """  
  
    # Breaks the BeautifulSoup object into its children  
    soup_children = list(soup2.children)  
  
    # Returns a stripped version of the fifth child, which contains the list of authors  
    return soup_children[4].rstrip().replace("\n", "").rstrip(",")  
  
  
def get_title (soup3: BeautifulSoup) -> str:  
  
    """Retrieves a string containing the title using the article's soup object.  
  
    Args:  
        soup3: A BeautifulSoup object pertaining to a single article.  
  
    Returns:  
        A string containing the article's title.  
    """  
  
    # Finds the first instance of the <a> tag  
    soup_a = soup3.find("a")  
  
    # Returns the title in a stripped string  
    return soup_a.text.rstrip().replace("\n", " ")  
  
  
def get_url (soup4: BeautifulSoup) -> str:  
  
    """Retrieves a string containing the url using the article's soup object.  
  
    Args:  
        soup4: A BeautifulSoup object pertaining to a single article.  
  
    Returns:  
        A string containing the article's url.  
    """  
  
    # Finds the first instance of the <a> tag  
    soup_a = soup4.find("a")  
  
    # Returns the url  
    return soup_a.get("href")  
  
  
def get_journal (soup5: BeautifulSoup) -> str:  
  
    """Retrieves a string containing the journal name using the article's soup object.  
  
    Args:
```

```

soup5: A BeautifulSoup object pertaining to a single article.

>Returns:
    A string containing the journal name.
"""
# Finds the first instance of the <I> tag
soup_i = soup5.find("i")

# Returns the journal in a string
return soup_i.text


def get_volume (soup6: BeautifulSoup) -> str:

    """Retrieves a string containing the volume number using the article's soup object.

Args:
    soup6: A BeautifulSoup object pertaining to a single article.

>Returns:
    A string containing the volume.
"""
# Finds the first instance of the <b> tag
soup_b = soup6.find("b")

# If no volume is found returns an empty string. Else returns the text of b as a string.
try:
    return soup_b.text
except AttributeError:
    return ""


def get_articles (soup1 : BeautifulSoup) -> None:

    """Writes the information for all journal articles to 'publist.bib' in .bibTeX format.

Args:
    soup1: A BeautifulSoup object containing information for all journal articles.

>Returns:
    Returns None (a file is written too)
"""

with open("publist.bib", "w") as pub:
    # Creates a set of all unique articles' html code
    soup_set = set(soup1.find_all("li"))

    # Iterate through each article's html
    for k in soup_set:
        # Create string of info in .bibTeX format
        info = """@article{"""
        {{ author = {0}
        title = {1}
        URL = {2}
        journal = {3}
        volume = {4}
}}\n\n""".format(get_authors(k), get_title(k), get_url(k), get_journal(k), get_volume(k))
        # Write info to file

```

```
    pub.write(info)

    # Call get_articles
    get_articles(soup)
```

In [480]: *# check your code: print the BibTex file*

```
f = open('publist.bib', 'r')
print(f.read())

URL = Papers/PhysRevB_79_235202_2009.pdf
journal = Phys. Rev. B
volume = 79
}

@article:
{   author = Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam
Gali
    title = "Direct correlation of crystal structure and optical properties
in wurtzite/zinc-blende GaAs nanowire heterostructures"
    URL = Papers/2011/PhysRevB_83_045303_2011.pdf
    journal = PHYSICAL REVIEW B
    volume = 83
}

@article:
{   author = W.L. Wang, O.V. Yazyev, S. Meng and E. Kaxiras
    title = "Topological Frustration in Graphene Nanoflakes: Magnetic Order
and Spin Logic Devices"
```

Your output should look like this

```

@article
{   author = Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussi
s, Yiyang Zhang, Mark Ming-Cheng Cheng
    title = "Approaching the intrinsic band gap in suspended high-mobili
ty graphene nanoribbons"
    URL = Papers/2011/PhysRevB_84_125411_2011.pdf
    journal = PHYSICAL REVIEW B
    volume = 84
}

...
@article
{   author = E. Kaxiras and S. Succi
    title = "Multiscale simulations of complex systems: computation meet
s reality"
    URL = Papers/SciModSim_15_59_2008.pdf
    journal = Sci. Model. Simul.
    volume = 15
}
@article
{   author = E. Manousakis, J. Ren, S. Meng and E. Kaxiras
    title = "Effective Hamiltonian for FeAs-based superconductors"
    URL = Papers/PhysRevB_78_205112_2008.pdf
    journal = Phys. Rev. B
    volume = 78
}

```

Part B [50 pts]: Follow the stars in IMDb's list of "The Top 100 Stars for 2017"

Overview

In Part 3 your goal is to extract information from IMDb's Top 100 Stars for 2017 (<https://www.imdb.com/list/ls025814950/>) and perform some analysis on each star in the list. In particular we are interested to know: a) how many performers made their first movie at 17? b) how many performers started as child actors? c) who is the most prolific actress or actor in IMDb's list of the Top 100 Stars for 2017? . These questions are addressed in more details in the Questions below.

When data is **not** given to us in a file, we need to fetch them using one of the following ways:

- download a file from a source URL
- query a database

- query a web API
- scrape data from the web page

Question 1: Web Scraping Using Beautiful Soup

1.1 Download the webpage of the "Top 100 Stars for 2017" (<https://www.imdb.com/list/ls025814950/>) (<https://www.imdb.com/list/ls025814950/>) into a `requests` object and name it `my_page`. Explain what the following attributes are:

- `my_page.text`,
- `my_page.status_code`,
- `my_page.content`.

1.2 Create a Beautiful Soup object named `star_soup` giving `my_page` as input.

1.3 Write a function called `parse_stars` that accepts `star_soup` as its input and generates a list of dictionaries named `starlist` (see definition below). One of the fields of this dictionary is the `url` of each star's individual page, which you need to scrape and save the contents in the `page` field. Note that there is a ton of information about each star on these webpages.

1.4 Write a function called `create_star_table` to extract information about each star (see function definition for the exact information to extract). **Only extract information from the first box on each star's page. If the first box is acting, consider only acting credits and the star's acting debut, if the first box is Directing, consider only directing credits and directorial debut.**

1.5 Now that you have scraped all the info you need, it's a good practice to save the last data structure you created to disk. That way if you need to re-run from here, you don't need to redo all these requests and parsing. Save this information to a JSON file and **submit** this JSON file in Canvas with your notebook.

1.6 Import the contents of the teaching staff's JSON file (`data/staff_starinfo.json`) into a pandas dataframe. Check the types of variables in each column and clean these variables if needed. Add a new column to your dataframe with the age of each actor when they made first movie (name this column `age_at_first_movie`).

1.7 You are now ready to answer the following intriguing questions:

- How many performers made their first movie at 17?
- How many performers started as child actors? Define child actor as a person less than 12 years old.
- Who is the most prolific actress or actor in IMDb's list of the Top 100 Stars for 2017?

1.8 Make a plot of the number of credits versus the name of actor/actress.

Hints

- Create a variable that groups actors/actresses by the age of their first movie. Use pandas' `.groupby` to divide the dataframe into groups of performers that for example started performing as children (`age < 12`). The grouped variable is a `GroupBy` pandas object and this object has all of the information needed to then apply some operation to each of the groups.

- When cleaning the data make sure the variables with which you are performing calculations are in numerical format.
- The column with the year has some values that are double, e.g. '2000-2001' and the column with age has some empty cells. You need to deal with these before performing calculations on the data!
- You should include both movies and TV shows.

Resources

- The requests library makes working with HTTP requests powerful and easy. For more on the requests library see <http://docs.python-requests.org/> (<http://docs.python-requests.org/>)

In [481]: `import requests`

1.1

In [482]: `# Download https://www.imdb.com>List/Ls025814950/ to my_page
my_page = requests.get("https://www.imdb.com/list/ls025814950/")

my_page.text
print("'my_page.text' is the literal character for character text found in the HTI

my_page.status_code
print("""\n'my_page.status_code' is issued in terms of the success of the
HTTPRequest made to the server, indicating that the client
request was successfully received and the information obtained """)

my_page.content
print("""\n'my_page.content' is the actual HTML source code on the
page at the current moment, including injected content scripts.""")`

'my_page.text' is the literal character for character text found in the HTML file at the link.

'my_page.status_code' is issued in terms of the success of the HTTPRequest made to the server, indicating that the client request was successfully received and the information obtained

'my_page.content' is the actual HTML source code on the page at the current moment, including injected content scripts.

Your answers here

1.2

```
In [483]: # Using previously defined function make_soup to create the BeautifulSoup object
star_soup = make_soup(my_page.text)
```

```
In [484]: # Check your code - you should see an HTML page
print (star_soup.prettify()[:])
```

```
<a href="/name/nm6748436/?ref_=nm_ls_hd" >
    
</a>
</div>
<div class="lister-item-content">
    <h3 class="lister-item-header">
        <span class="lister-item-index unbold text-primary">
            34.
        </span>
        <a href="/name/nm6748436?ref_=nm_ls_hd">
            Dafne Keen
        </a>
    </h3>
    <p class="text-muted text-small">
        Actress
        <span class="ghost">
            |
        </span>
    </p>
```

1.3

Function

`parse_stars`

Input

`star_soup`: the soup object with the scraped page

Returns

a list of dictionaries; each dictionary corresponds to a star profile and has the following data:

`name`: the name of the actor/actress as it appears at the top

`gender`: 0 or 1: translate the word 'actress' into 1 and 'actor' into '0'

`url`: the url of the link under their name that leads to a page with details

`page`: the string containing the soup of the text in their individual info page (from url)

Example:

```
{'name': Tom Hardy,  
 'gender': 0,  
 'url': https://www.imdb.com/name/nm0362766/?ref_=nm_ls_hd,  
 'page': BS object with 'html' text acquired by scraping the 'url' page'  
}
```

```
In [485]: def determine_gender(soup_sub: BeautifulSoup) -> int:
    """Accepts a BeautifulSoup object pertaining to a moviestar and returns their gender.

    Args:
        soup_sub: The BeautifulSoup object containing the movie star's information.

    Returns:
        An integer 0 (male) or 1 (female).
    """
    # Pulls out text for the movie star's bio
    star_par = soup_sub.find("p").text
    male = re.compile(r"\s*Actor")
    female = re.compile(r"\s*Actress")
    # Check for gender match by 'Actor' or 'Actress' appearance in bio
    if(female.match(star_par)):
        return 1
    elif(male.match(star_par)):
        return 0
    else:
        # Print exception statement
        rank = soup_sub.find("span", {"class": "lister-item-index unbold text-primary"})
        name = re.findall(r"\s*(.*\n)", soup_sub.find("a").text)[0]
        print("EXCEPTION: Actor or Actress not found for # {0}, {1}.".format(rank, name))
        print("Fall back on second criteria: personal pronouns.")

        # Declare regex patterns for gender pronouns
        male_regex = re.compile(r"\n*.*([H|h]is)|([H|h]e)|([H|h]im).*")
        female_regex = re.compile(r"\n*.*([H|h]ers)|([S|s]he)|([H|h]er).*")
        neutral_regex = re.compile(r"\n*.*([T|t]hey)|([T|t]hem)|([T|t]heirs).*")

        # Get the celebrity's bio
        star_par_bio = soup_sub.find_all("p")[1].text.replace("\n", " ")

        # Check for gender match
        if(female_regex.match(star_par_bio)):
            return 1
        elif(male_regex.match(star_par_bio)):
            return 0
        elif(neutral_regex.match(star_par_bio)):
            print("Nonbinary gender")
            raise AttributeError
        else:
            print("Gender not found")
            raise AttributeError

    def parse_stars(str_soup: BeautifulSoup) -> list:
        """Accepts a BeautifulSoup object as input and generates a list of dictionaries containing information about each star.

        Args:
            str_soup: A BeautifulSoup object that includes the html code

        Returns:
            Returns a list of dictionaries containing information about each star.
        """
        # Find the stars' data in the HTML
        star_data = str_soup.find_all("div", {"class": "lister-item-content"})
```

```

# Initialize list of stars
list_of_stars = []
for k in star_data:
    star_dict = {}
    # Name
    star_dict["name"] = re.findall(r"\s*(.*\n*", k.find("a").text)[0]
    # Gender
    star_dict["gender"] = determine_gender(k)
    # URL
    star_dict["url"] = ("https://www.imdb.com{0}".format(k.find("a").get("href"))
    # Get page
    star_page = requests.get(star_dict["url"])
    star_dict["page"] = make_soup(star_page.text)
    # Append the star's data to the list
    list_of_stars.append(star_dict)

return list_of_stars

# Create star list
starlist = parse_stars(star_soup)

```

EXCEPTION: Actor or Actress not found for # 66, Christopher Nolan.
Fell back on second criteria: personal pronouns.

In [486]:

```

# this list is large because of the html code into the `page` field
# to get a better picture, print only the first element
starlist[0]

<script>
    if (typeof uex == 'function') {
        uex("ld", "NameFilmographyWidget", {wb: 1});
    }
</script>

<script>
    if (typeof uet == 'function') {
        uet("bb", "NameVideoStrip", {wb: 1});
    }
</script>
<script>
    if ('csm' in window) {
        csm.measure('csm_NameVideoStrip_started');
    }
</script>
<div class="article">
<h2>Related Videos</h2>
<div class="mediastrip_big">
<span class="video_slate">

```

Your output should look like this:

```
{
  'name': 'Gal Gadot',
  'gender': 1,
  'url': 'https://www.imdb.com/name/nm2933757?ref_=nm_ls_hd',
  'page':
    <!DOCTYPE html>

    <html xmlns:fb="http://www.facebook.com/2008/fbml" xmlns:og="http://ogp.me/ns#">
      <head>
        <meta charset="utf-8"/>
        <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
        <meta content="app-id=342792525, app-argument=imdb:///name/nm2933757?src=mdot" name="apple-itunes-app"/>
        <script type="text/javascript">var IMDbTimer={starttime: new Date().getTime(),pt:'java'};</script>
        <script>
          if (typeof uet == 'function') {
            uet("bb", "LoadTitle", {wb: 1});
          }
        </script>
        <script>(function(t){ (t.events = t.events || {})[ "csm_head_pre_title" ] = new Date().getTime(); })(IMDbTimer);</script>
      ...
    
```

1.4

Function

`create_star_table`

Input

`the_starlist`

Returns

a list of dictionaries; each dictionary corresponds to a star profile and has the following data:

- `star_name`: the name of the actor/actress as it appears at the top
- `gender`: 0 or 1 (1 for 'actress' and 0 for 'actor')
- `year_born` : year they were born

`first_movie`: title of their first movie or TV show
`year_first_movie`: the year they made their first movie or TV show
`credits`: number of movies or TV shows they have made in their career.

Example:

```
{'star_name': Tom Hardy,  
 'gender': 0,  
 'year_born': 1997,  
 'first_movie' : 'Batman',  
 'year_first_movie' : 2017,  
 'credits' : 24}
```

In [487]:

```
def create_star_table(starlist: list) -> list:
    """Extracts information for each star from the star list of dictionaries and
    information about each star.

    Args:
        starlist: A list of dictionaries holding information about each star

    Returns:
        Returns a list of dictionaries with new data for each star to be used in
    """
    # Initialize the list of star dictionaries
    star_dict_list = []

    # Iterate through each star's dictionary
    for k in starlist:

        # Initialize the star's dictionary
        k_dict = {}

        # Store movie star's name and gender
        k_dict["name"] = k["name"]
        k_dict["gender"] = k["gender"]

        # Get the movie html data
        k_movies = k["page"].find("div", {"class": "filmo-category-section"})

        # Name of first movie
        movie_1 = k_movies.find_all("div", {"class": "filmo-row"})[-1]
        k_dict["first_movie"] = re.findall(r"\n.*\n*(.*)\n", movie_1.text)[0]

        # Year of first movie
        year = re.findall(r"\n*\xa0(.*)\n*", movie_1.text)[0]
        k_dict["year_first_movie"] = year

        # Birth year
        regex_year = r"<time datetime=\"(\d\d\d\d)\""
        try:
            k_time = re.findall(regex_year, str(k["page"].find("time")))[0]
            k_dict["year_born"] = k_time
        except IndexError:
            print(k_dict["name"] + " has no listed birth year.")
            k_dict["year_born"] = ""

        # Number of credits
        regex_credit = r"[\n|.]*(\d+).*"
        k_films = k["page"].find("div", {"class": "head"}).text
        k_dict["credits"] = re.findall(regex_credit, k_films)[0]

        # Add the movie star's dictionary to the list
        star_dict_list.append(k_dict)

    return star_dict_list
```

```
In [488]: star_table = []
star_table = create_star_table(starlist)
```

Dafne Keen has no listed birth year.
Christian Navarro has no listed birth year.

```
In [489]: # check your code
star_table
```

```
'year_first_movie': '1997',
'year_born': '1984',
'credits': '53'},
{'name': 'Taron Egerton',
'gender': 0,
'first_movie': 'The Last of the Haussmans',
'year_first_movie': '2012',
'year_born': '1989',
'credits': '16'},
{'name': 'Anya Taylor-Joy',
'gender': 1,
'first_movie': 'Vampire Academy',
'year_first_movie': '2014',
'year_born': '1996',
'credits': '17'},
{'name': 'Elizabeth Debicki',
'gender': 1,
'first_movie': 'A Few Best Men',
'year_first_movie': '2011',
'year_born': '1990',
```

Your output should look like this:

```
[{'name': 'Gal Gadot',
'gender': 1,
'year_born': '1985',
'first_movie': 'Bubot',
'year_first_movie': '2007',
'credits': '25'},
{'name': 'Tom Hardy',
'gender': 0,
'year_born': '1977',
'first_movie': 'Tommaso',
'year_first_movie': '2001',
'credits': '55'},
```

...

1.5

```
In [490]: import json

with open('moviestar_data.json', 'w') as output:
    json.dump(star_table, output)
```

1.6

```
In [491]: stardata = pd.read_json("data/staff_starinfo.json")

# Prints data types for each column utilizing bold style
print("{0}Pandas type:{1}\n{2}".format("\033[1m", "\033[0m", stardata.dtypes))

# Initialize age_at_first_movie to be a cleaned version of year_first_movie (to pi
stardata["age_at_first_movie"] = stardata["year_first_movie"].apply(lambda x: int(x))

# Correct age_at_first_movie by finding difference of age_at_first_movie and year_
stardata["age_at_first_movie"] = stardata["age_at_first_movie"] - stardata["year_
```

Pandas type:

credits	int64
first_movie	object
gender	int64
name	object
year_born	int64
year_first_movie	object
dtype:	object

1.7

```
In [492]: # Prints the number of performers who debuted at age 17
age = 17
movie_teens = len(stardata[stardata["age_at_first_movie"] == age])
print("{0} performers made their first movie at {1}".format(movie_teens, age))

# Prints the number of performers who started as a child actor
child_age = 12
movie_children = len(stardata[stardata["age_at_first_movie"] <= child_age])
print("{0} performers started their careers as child actors".format(movie_children))

# Prints the most prolific movie star
most_prolific = stardata["credits"].idxmax()
name_most = stardata.at[most_prolific, "name"]
print("{0} is the most prolific movie celebrity".format(name_most))
```

8 performers made their first movie at 17
 21 performers started their careers as child actors
 Sean Young is the most prolific movie celebrity

Your output should look like this:

8 performers made their first movie at 17

1.8

```
In [493]: # Sort and Reindex the list of movie stars by credits
stardata_new = stardata.sort_values(by="credits").reset_index()

# Group the data by male and female
x_female = stardata_new.groupby("gender").get_group(1)
x_male = stardata_new.groupby("gender").get_group(0)

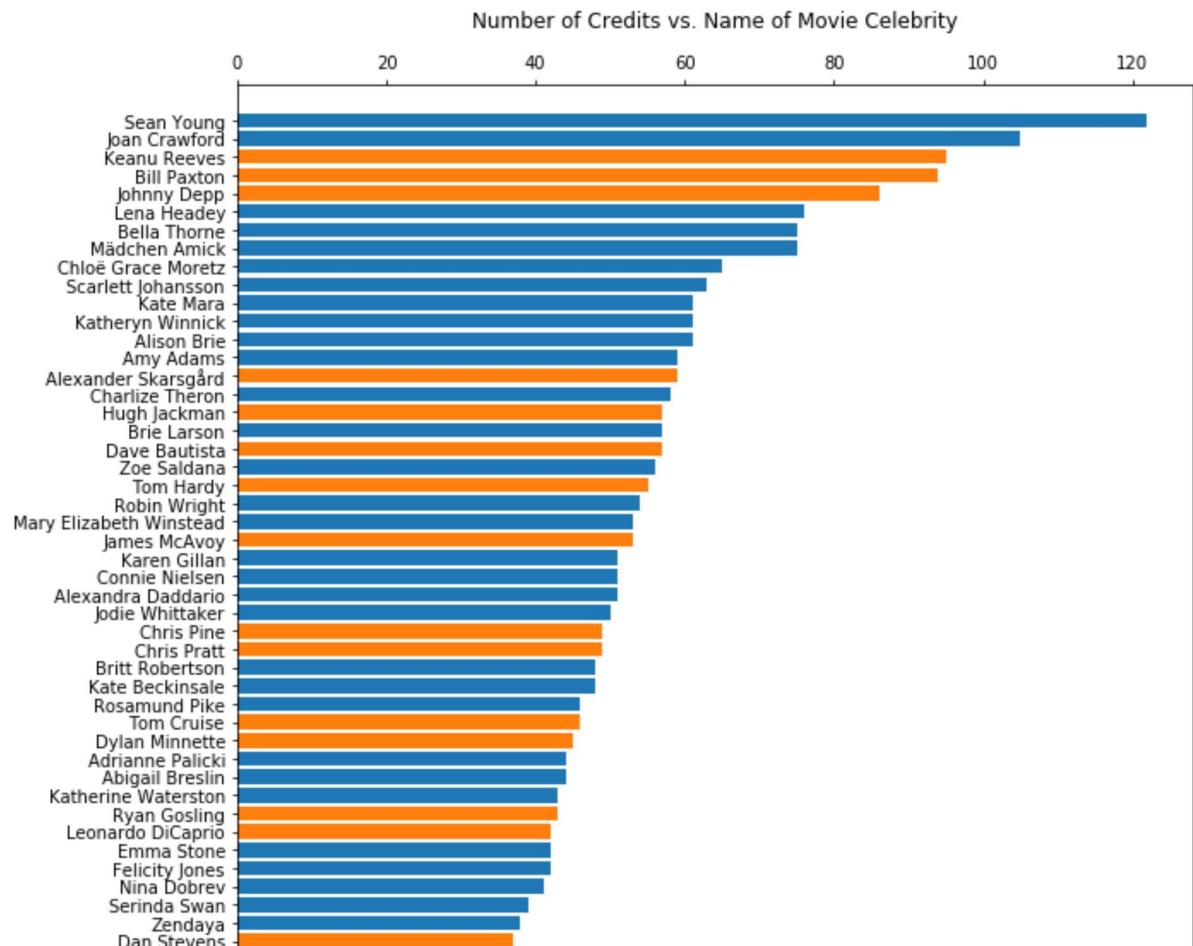
# Create horizontal bar chart with male and female data
plt.figure(figsize=(10,20))
plt.barh(x_female.index.values, x_female["credits"].values, label="female")
plt.barh(x_male.index.values, x_male["credits"].values, label="male")

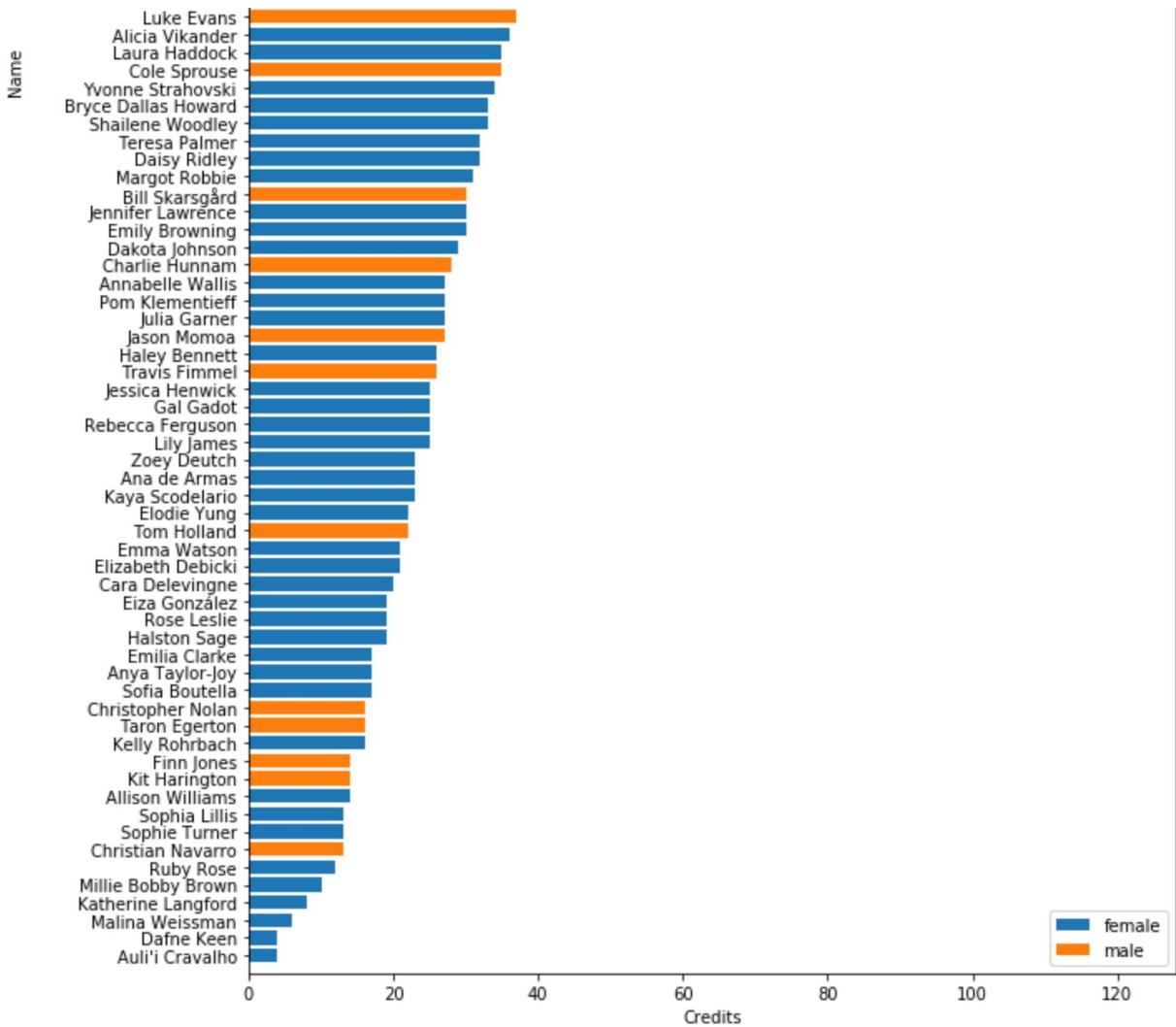
# Title
plt.title("Number of Credits vs. Name of Movie Celebrity", y=1.025)

# x-axis
plt.xlabel("Credits")
plt.tick_params(labeltop=True, top=True)

# y-axis
plt.ylabel("Name")
plt.yticks(stardata_new.index.values, stardata_new[ "name" ].values)
plt.ylim(-1,len(stardata_new)+1)
plt.tick_params(labeltop=True, top=True)

# Display chart and legend
plt.legend(loc="lower right")
plt.show()
```





Your answer here

```
In [494]: from IPython.core.display import HTML
def css_styling(): styles = open("styles/cs109.css", "r").read(); return HTML(styles)
css_styling()
```

Out[494]:

In []: