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 [1]:

```python
# 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 (https://docs.python.org/3.3/library/re.html), b) https://regexone.com (https://regexone.com), and c) https://docs.python.org/3/howto/regex.html (https://docs.python.org/3/howto/regex.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 [2]:

```
# import the regular expressions library
import re
```

In [3]:

```
# use this file
pub_filename = 'data/publist_super_clean.html'
```

In [4]:

```
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:
        publist = f.read()
    return publist
```

In [5]:

```
prof_pubs = get_pubs(pub_filename)
```

```
In [6]: print (prof_pubs)
```

```
<!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 http-equiv="Content-Type" content="text/html;charset=UTF-8">
<LINK REL="stylesheet" TYPE="text/css" HREF="../styles/style_pubs.css">
<META NAME="description" CONTENT="">
<META NAME="keywords" CONTENT="Kaxiras E, Multiscale Methods, Computation
al Materials" >
</HEAD>

<BODY>

<OL START=244>
<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>
```

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
 graphene nanoribbons&quot;</A>
<BR>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yi
yang 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 semic
onductor 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 pr
operties 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 [7]:
```
regex = r"C\.\s?M\.\s?Friend"
friend = re.findall(regex, prof_pubs)
print("'C.M. Friend' appears {0} times in the list of publications.".format(
```

```
'C.M. Friend' appears 5 times in the list of publications.
```

1.3

In [8]:

```
# this code includes the one conference with all of the journals
regex = r"\s<I>(.*)</I>"
journals = re.findall(regex, prof_pubs)
print(len(journals)) # check all 45 are captured
journals
```

```
45
```

Out[8]:

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

In [9]:

```
# remove duplicates and trailing whitespace
journals = set(journals)
journals.remove("NanoLett. ")
journals = [line.strip() for line in list(journals)]
journals.sort()
journals
```

Out[9]:

```
['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.',
 '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.']
```

In [10]:

```python
# this code is only journals (no conference included)
regex = r"\s<I>([\w\.\s\-\&:]+)</I>"
journals = re.findall(regex, prof_pubs)
print(len(journals))
journals # check 44 total are captured
```

```
44
```

Out[10]:

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

```
In [11]: # remove duplicates and trailing whitespace
         journals = set(journals)
         journals.remove("NanoLett. ")
         journals = [line.strip() for line in list(journals)]
         journals.sort()
         journals
```

```
Out[11]: ['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.',
          '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 [12]: regex = r"<BR>\s?(.*)\n\s?<I>"
         pub_authors = re.findall(regex, prof_pubs)
         len(pub_authors) # check that there are 45 to match all 45 publications
```

```
Out[12]: 45
```

In [13]:

```
# check your code: print the list of strings containing the author(s)' names
for item in pub_authors:
    print (item)
```

```
Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhan
g, Mark Ming-Cheng Cheng,
JAdam Gali, Efthimios Kaxiras, Gergely T. Zimanyi, Sheng Meng,
Jan M. Knaup, Han Li, Joost J. Vlassak, and Efthimios Kaxiras,
Martin Heiss, Sonia Conesa-Boj, Jun Ren, Hsiang-Han Tseng, Adam Gali,
Simone Melchionna, Efthimios Kaxiras, Massimo Bernaschi and Sauro Succi,
J R Maze, A Gali, E Togan, Y Chu, A Trifonov,
Kejie Zhao, Wei L. Wang, John Gregoire, Matt Pharr, Zhigang Suo,
Masataka Katono, Takeru Bessho, Sheng Meng, Robin Humphry-Baker, Guido Ro
thenberger,
Thomas D. Kuhne, Tod A. Pascal, Efthimios Kaxiras, and Yousung Jung,
Sheng Meng, Efthimios Kaxiras, Md. K. Nazeeruddin, and Michael Gratzel,
Bingjun Xu, Jan Haubrich, Thomas A. Baker, Efthimios Kaxiras, and Cynthia
M. Friend,
Jun Ren, Sheng Meng, Yi-Lin Wang, Xu-Cun Ma, Qi-Kun Xue, Efthimios Kaxira
s,
Jan Haubrich, Efthimios Kaxiras, and Cynthia M. Friend,
Thomas A. Baker, Bingjun Xu, Stephen C. Jensen, Cynthia M. Friend and Eft
himios Kaxiras,
Youdong Mao, Wei L. Wang, Dongguang Wei, Efthimios Kaxiras, and Joseph G.
Sodroski,
H. Li, J.M. Knaup, E. Kaxiras and J.J. Vlassak,
W.L. Wang and E. Kaxiras,
L.A. Agapito, N. Kioussis and E. Kaxiras,
A. Peters, S. Melchionna, E. Kaxiras, J. Latt, J. Sircar, S. Succi,
J. Ren, E. Kaxiras and S. Meng,
T.A. Baker, E. Kaxiras and C.M. Friend,
H.P. Chen, R.K. Kalia, E. Kaxiras, G. Lu, A. Nakano, K. Nomura,
S. Meng and E. Kaxiras,
C.L. Chang, S.K.R.S. Sankaranarayanan, D. Ruzmetov, M.H. Engelhard, E. Ka
xiras and S. Ramanathan,
T.A. Baker, C.M. Friend and E. Kaxiras,
S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F.J. Rybicki, D. Mitso
uras, A.U. Coskun and C.L. Feldman,
M. Bernaschi, M. Fatica, S. Melchionna, S. Succi and E. Kaxiras,
E. Manousakis, J. Ren, S. Meng and E. Kaxiras,
A. Gali, E. Janzen, P. Deak, G. Kresse and E. Kaxiras,
S.K.R.S. Sankaranarayanan, E. Kaxiras and S. Ramanathan,
M. Bernaschi, S. Melchionna, S. Succi, M. Fyta, E. Kaxiras
T.A. Baker, B.J. Xu, X.Y. Liu, E. Kaxiras and C.M. Friend,
F.J. Rybicki, S. Melchionna, D. Mitsouras, A.U. Coskun, A.G. Whitmore, E.
Kaxiras, S. Succi, P.H. Stone and C.L. Feldman,
H. Chen, W.G. Zhu, E. Kaxiras, and Z.Y. Zhang,
M. Fyta, S. Melchionna, M. Bernaschi, E. Kaxiras and S. Succi,
E.M. Kotsalis, J.H. Walther, E. Kaxiras and P. Koumoutsakos,
C.E. Lekka, J. Ren, S. Meng and E. Kaxiras,
W.L. Wang, O.V. Yazyev, S. Meng and E. Kaxiras,
A. Gali and E. Kaxiras,
S. Melchionna, M. Bernaschi, M. Fyta, E. Kaxiras and S. Succi,
S.K.R.S. Sankaranarayanan, E. Kaxiras, S. Ramanathan,
T.A. Baker, C.M. Friend and E. Kaxiras,
T.A. Baker, C.M. Friend and E. Kaxiras,
E. Kaxiras and S. Succi,
E. Manousakis, J. Ren, S. Meng and 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 Ki
oussis, Yiyang Zhang, Mark Ming-Cheng Cheng
     title = "Approaching the intrinsic band gap in suspended high-m
obility graphene nanoribbons"
     URL = Papers/2011/PhysRevB_84_125411_2011.pdf
     journal = PHYSICAL REVIEW B
     volume = 84
}
```

About the bibTex format (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/).
- More about the BibTex format (http://www.bibtex.org).

In [14]:

```
# import the necessary libraries
from bs4 import BeautifulSoup
from sys import argv
from urllib.request import urlopen
from urllib.error import HTTPError
```

2.1

In [15]:

```
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.
    '''
    with open(filename, "r") as f:
        publist = f.read()
        page = BeautifulSoup(publist, 'html.parser')
    return page
```

```
In [16]: soup = make_soup(pub_filename)
         print(soup)
```

```
<!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" na
me="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 n
anoribbons"</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)
```

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/cs
s"/>
<meta content="" name="description"/>
<meta content="Kaxiras E, Multiscale Methods, Computational Material
s" 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 graph
ene nanoribbons"</a>
<br/>Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Y
iyang 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 semiconduc
tor 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 [17]:
```
print(soup.prettify())
```

```
<!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" n
ame="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 graphe
ne nanoribbons"
   </a>
```

In [18]:
```
# return new string with new line characters and commas replaced by empty sp
rem_nl = lambda s: s.replace("\n", "").replace(",","")
# return new string with leading and trailing whitespace stripped
rem_sp = lambda s: s.strip()
# return new string with text only
text = lambda s: s.get_text()
```

In [19]:
```
def journals(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return journal/conference names
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        A list of journal/conference names
    '''
    name = [rem_sp(text(tag.next_sibling.next_sibling.next_sibling.next_sibl
    return name
```

In [20]:
```
def pub_authors(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return journal authors
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        A list of authors for each publication
    '''
    authors = [rem_sp(rem_nl(tag.next_sibling.next_sibling.next_sibling)) fo
    return authors
```

In [21]:

```
def urls(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return urls
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        A list of urls for each paper
    '''
    links = [link.get('href') for link in soup_object.find_all('a')]
    return links
```

In [22]:

```
def volume(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return volume numbers
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        A list of volume numbers for each publication
    '''
    # this pulls in the lack of conference volume as "<br/>" which is stripp
    vol = [tag.next_sibling.next_sibling for tag in soup_object.find_all('i'
    vol_text = []
    for line in vol:
        try:
            vol_text.append(line.get_text())
        except AttributeError:
            pass
    return vol_text
```

In [23]:

```
def article(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return titles
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        A list of article titles for each publication
    '''
    title = [rem_nl(line.get_text()) for line in soup_object.find_all('a')]
    return title
```

In [24]:

```
def bibtex_pub(soup_object: BeautifulSoup) -> '.bibTex file':
    '''Take BS object and return .bibTex file
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        .bibTex file of all publications and corresponding information
    '''
    with open("data/publist.bib", "w") as dataf:
        for i, j, k, l, m in zip(pub_authors(soup_object), article(soup_obje
                                 journals(soup_object), volume(soup_object))
            dataf.write("\n@article\n{{    author = {}\n     title = {}\n
                        "\n     journal = {}\n     volume = {}\n}}"
                        .format(str(i), str(j), str(k), str(l), str(m)))
```

In [25]:

```
bibtex_pub(soup)
```

In [26]:

```python
# print the BibTex file
f = open('data/publist.bib','r')
print (f.read())
```

```
@article
{    author = Ming-Wei Lin Cheng Ling Luis A. Agapito Nicholas Kioussis Y
iyang 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 = JAdam Gali Efthimios Kaxiras Gergely T. Zimanyi Sheng Meng
     title = "Effect of symmetry breaking on the optical absorption of se
miconductor nanoparticles"
     URL = Papers/2011/PhysRevB_84_035325_2011.pdf
     journal = PHYSICAL REVIEW B
     volume = 84
}
@article
```

Your output should look like this

```
@article
{    author = Ming-Wei Lin, Cheng Ling, Luis A. Agapito, Nicholas Kioussis, Yiyang Zhang, Mark Ming-Cheng Cheng
     title = "Approaching the intrinsic band gap in suspended high-mobility 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 meets 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/ (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 proliferate 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 [27]:
```
import requests
```

1.1

In [28]:
```
my_page = requests.get("https://www.imdb.com/list/ls025814950/")
```

Your answers here

In [29]:
```
my_page.text
```

Out[29]:
```
'\n\n\n\n<!DOCTYPE html>\n<html\n    xmlns:og="http://ogp.me/ns#"\n    xm
lns:fb="http://www.facebook.com/2008/fbml">\n    <head>\n        \n
   <meta charset="utf-8">\n        <meta http-equiv="X-UA-Compatible" con
tent="IE=edge">\n\n    <meta name="apple-itunes-app" content="app-id=3427
92525, app-argument=imdb:///list/ls025814950?src=mdot">\n\n\n\n        <s
cript type="text/javascript">var IMDbTimer={starttime: new Date().getTime
(),pt:\'java\'};</script>\n\n<script>\n    if (typeof uet == \'function
\') {\n      uet("bb", "LoadTitle", {wb: 1});\n    }\n</script>\n  <scrip
t>(function(t){ (t.events = t.events || {})["csm_head_pre_title"] = new D
ate().getTime(); })(IMDbTimer);</script>\n        <title>Top 100 Stars of
2017 - IMDb</title>\n  <script>(function(t){ (t.events = t.events || {})
["csm_head_post_title"] = new Date().getTime(); })(IMDbTimer);</script>\n
<script>\n    if (typeof uet == \'function\') {\n      uet("be", "LoadTit
le", {wb: 1});\n    }\n</script>\n<script>\n    if (typeof uex == \'funct
ion\') {\n      uex("ld", "LoadTitle", {wb: 1});\n    }\n</script>\n\n
     <link rel="canonical" href="https://www.imdb.com/list/ls025814950/"
/>\n        <meta property="og:url" content="http://www.imdb.com/list/ls0
25814950/" />\n\n<script>\n    if (typeof uet == \'function\') {\n      u
et("bb", "LoadIcons", {wb: 1});\n    }\n</script>\n  <script>(function(t)
```

my_page.text returns content of the response in unicode

```
In [30]: my_page.status_code
```

```
Out[30]: 200
```

my_page.status_code returns the HTTP status number, with 200 indicating the request was successful.

```
In [31]: my_page.content
```

```
Out[31]: b'\n\n\n\n<!DOCTYPE html>\n<html\n    xmlns:og="http://ogp.me/ns#"\n    x
mlns:fb="http://www.facebook.com/2008/fbml">\n    <head>\n          \n
    <meta charset="utf-8">\n        <meta http-equiv="X-UA-Compatible" co
ntent="IE=edge">\n\n    <meta name="apple-itunes-app" content="app-id=342
792525, app-argument=imdb:///list/ls025814950?src=mdot">\n\n\n\n        <
script type="text/javascript">var IMDbTimer={starttime: new Date().getTim
e(),pt:\'java\'};</script>\n\n<script>\n    if (typeof uet == \'function
\') {\n      uet("bb", "LoadTitle", {wb: 1});\n    }\n</script>\n  <scrip
t>(function(t){ (t.events = t.events || {})["csm_head_pre_title"] = new D
ate().getTime(); })(IMDbTimer);</script>\n        <title>Top 100 Stars of
2017 - IMDb</title>\n  <script>(function(t){ (t.events = t.events || {})
["csm_head_post_title"] = new Date().getTime(); })(IMDbTimer);</script>\n
<script>\n    if (typeof uet == \'function\') {\n      uet("be", "LoadTit
le", {wb: 1});\n    }\n</script>\n<script>\n    if (typeof uex == \'funct
ion\') {\n      uex("ld", "LoadTitle", {wb: 1});\n    }\n</script>\n\n
     <link rel="canonical" href="https://www.imdb.com/list/ls025814950/"
/>\n        <meta property="og:url" content="http://www.imdb.com/list/ls0
25814950/" />\n\n<script>\n    if (typeof uet == \'function\') {\n      u
et("bb", "LoadIcons", {wb: 1});\n    }\n</script>\n  <script>(function(t)
{ (t.events = t.events || {})["csm_head_pre_icon"] = new Date().getTime
```

my_page.content returns content of the response in bytes

1.2

```
In [32]: star_soup = BeautifulSoup(my_page.text, 'html.parser')
```

In [33]:

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

```
<!DOCTYPE html>
<html xmlns:fb="http://www.facebook.com/2008/fbml" xmlns:og="http://ogp.m
e/ns#">
 <head>
  <meta charset="utf-8"/>
  <meta content="IE=edge" http-equiv="X-UA-Compatible"/>
  <meta content="app-id=342792525, app-argument=imdb:///list/ls025814950?
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);
```

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 profil
    e 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 w
    ith details
        page: the string containing the soup of the text in their indivi
    dual info page (from url)

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

```
In [34]: def names(soup_object: BeautifulSoup) -> list:
             '''Take BS object and return celeb names
             Args:
                 object: BeautifulSoup object containing the HTML page
             Returns:
                 list of all celeb names
             '''
             regex = r'> (.*)\n</a>'
             names_list = re.findall(regex, str(soup_object.find_all('h3')))
             names_list = names_list[:-3] # removes text found at end of list that a
             return names_list
```

```
In [35]: def url_full(soup_object: BeautifulSoup) -> list:
             '''Take BS object and return celeb URLs
             Args:
                 object: BeautifulSoup object containing the HTML page
             Returns:
                 list of URLs for each celebrity
             '''
             regex = r'name/nm(\d*)\?ref_=nmls_hd'
             url_dig = re.findall(regex, str(soup_object.find_all('h3')))
             url_list = []
             for line in url_dig:
                 url_list.append('https://www.imdb.com/name/nm{0}/?ref_=nmls_hd'.for
             return url_list
```

```
In [36]: def celeb_page(soup_object: BeautifulSoup) -> list:
             '''Take BS object and return individual celebrity soup objects
             Args:
                 object: BeautifulSoup object containing the HTML page
             Returns:
                 list of soup objects for each celebrity HTML page
             '''
             page = [requests.get(line, timeout=10.0) for line in url_full(soup_objec
             page_text = [line.text for line in page]
             make_soup = lambda s: BeautifulSoup(s, 'html.parser')
             star_page = [make_soup(line) for line in page_text]
             return star_page
```

In [37]:

```
def gender(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return celebrity gender
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        list of each celebrity's gender (1 = female, 0=male)
    '''
    regex = r'(Actor|Actress|Writer)' # Christopher Nolan is classified as a
    actor = re.findall(regex, str(soup_object.find_all("p", class_="text-mut
    gen_temp = []
    for line in actor:
        if line == 'Actress':
            result = 1
        else:
            result = 0
        gen_temp.append(result)
    return gen_temp
```

In [38]:

```
def parse_stars(soup_object: BeautifulSoup) -> list:
    '''Take BS object and return dictionary for celebrity
    Args:
        object: BeautifulSoup object containing the HTML page
    Returns:
        list of dictionaries for each celebrity; includes name, gender, URL
    '''
    actor_names = names(soup_object)
    actor_gender = gender(soup_object)
    actor_url = url_full(soup_object)
    actor_page = celeb_page(soup_object)
    list_dict = []
    for i in range(100):
        celebs = {
            'name': actor_names[i],
            'gender': actor_gender[i],
            'url': actor_url[i],
            'page': actor_page[i]
            }
        list_dict.append(celebs)
    return list_dict
```

In [39]:

```
starlist = parse_stars(star_soup)
```

```
In [40]: # to get a better picture, print only the first element
         starlist[0]
```

```
Out[40]: {'name': 'Gal Gadot',
          'gender': 1,
          'url': 'https://www.imdb.com/name/nm2933757/?ref_=nmls_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().getT
         ime(),pt:'java'};</script>
          <script>
              if (typeof uet == 'function') {
                uet("bb", "LoadTitle", {wb: 1});
              }
```

Your output should look like this:

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

 <html xmlns:fb="http://www.facebook.com/2008/fbml" xmlns:og="htt
p://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/nm293375
7?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_tit
le"] = 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 profil
e and has the following data:

    star_name: the name of the actor/actress as it appears at the to
p
    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 sho
w
    credits: number of movies or TV shows they have made in their ca
reer.

--------
Example:

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

```
In [41]: page_data = [line['page'] for line in starlist]
```

In [42]:

```
def birth(data_list: list) -> list:
    '''Take list and return list of birth years
    Args:
        list: each index is a BeautifulSoup object containing the HTML page
    Returns:
        list of celebrity birth years
    '''
    page_list = [str(line) for line in data_list]
    regex = r'birth_year=(\d{4})'
    years = re.findall(regex, str(page_list))
    # 2 birth years are missing (only 98 were captured)
    # upon investigation, Dafne Keen and Christian Navarro are missing
    # these two birth years are available on Wikipedia
    # since there are only two, will add manually instead of scraping
    keen_year = '2005'
    navarro_year = '1991'
    years.append(navarro_year)
    years.insert(33, keen_year)
    return years
```

In [43]:

```
def film_year(data_list: list) -> tuple:
    '''Take list and return tuple of first movie and corresponding year
    Args:
        list: each index is a BeautifulSoup object containing the HTML page
    Returns:
        tuple of list of each celebrity's first movie and list of each corre
    '''
    # pull section that corresponds to table of films and years on IMDB
    film_table = [line.find_all("div", class_="filmo-category-section") for
    # keep only first table
    film_table_first = [str(line).split('</div>, <div class=', 1)[0] for lin
    year_first = []
    film_first = []
    for i in range(len(film_table_first)):
        year = re.findall(r'(\d{4})\n</span>', str(film_table_first[i]))
        year.sort() # sort all years pulled from previous line and then keep
        earliest = year[0]
        year_first.append(earliest)
        # isolate string with earliest year and respective film(s)/show(s)
        fragment = str(film_table_first[i])[re.search(earliest, str(film_tab
        film = re.findall(r'">(.*)</a>', fragment)
        # remove elements of list that contain a variant of "episodes" in el
        no_episodes = [line for line in film if not 'pisode' in line]
        # take last entry of list; based on way IMDB lists films/shows, last
        film_first.append(no_episodes[-1])
    return (film_first, year_first)
```

In [44]:

```python
def num_credits(data_list: list) -> list:
    '''Take list and return list of credits
    Args:
        list: each index is a BeautifulSoup object containing the HTML page
    Returns:
        list of credits attributed to each celebrity
    '''
    number = lambda s: re.findall(r", '(\d*)'", str(s[0]))
    credits_list = []
    for i in range(100):
        regex = r'(Actress|Actor|Writer)</a> \((\d*) credits\)\n</div>'
        # grab appropriate credits section (there are multiple) and return i
        credit_digit = number(re.findall(regex, str(data_list[i])))
        credits_list.append(credit_digit)
    # flatten list of credits
    credits_list = [line[0] for line in credits_list]
    return credits_list
```

In [45]:

```python
def create_star_table(starlist: list) -> list:
    name_data = [line['name'] for line in starlist]
    gender_data = [line['gender'] for line in starlist]
    page_data = [line['page'] for line in starlist]
    list_dict = []
    for i in range(100):
        celebs_details = {
            'star_name': name_data[i],
            'gender': gender_data[i],
            'year_born': birth(page_data)[i],
            'first_movie': film_year(page_data)[0][i],
            'year_first_movie': film_year(page_data)[1][i],
            'credits': num_credits(page_data)[i]
        }
        list_dict.append(celebs_details)
    return list_dict
```

In [46]:

```python
# RUN THIS CELL ONLY ONCE - IT WILL TAKE SOME TIME TO RUN
star_table = create_star_table(starlist)
```

```
In [47]: # check your code
         star_table
```

```
Out[47]: [{'star_name': 'Gal Gadot',
           'gender': 1,
           'year_born': '1985',
           'first_movie': 'Bubot',
           'year_first_movie': '2007',
           'credits': '25'},
          {'star_name': 'Tom Hardy',
           'gender': 0,
           'year_born': '1977',
           'first_movie': 'Tommaso',
           'year_first_movie': '2001',
           'credits': '55'},
          {'star_name': 'Emilia Clarke',
           'gender': 1,
           'year_born': '1986',
           'first_movie': 'Empty Nest',
           'year_first_movie': '2009',
           'credits': '17'},
          {'star_name': 'Alexandra Daddario',
```

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 [48]: import json
         with open('data/star_table.json', 'w') as output:
             json.dump(star_table, output)
```

1.6

```
In [49]: df_stars = pd.read_json('data/staff_starinfo.json')
```

1.6.1 - reviewing data

In [50]:

```
df_stars
```

Out[50]:

	credits	first_movie	gender	name	year_born	year_first_movie
0	25	Bubot	1	Gal Gadot	1985	2007
1	55	Tommaso	0	Tom Hardy	1977	2001
2	17	Doctors	1	Emilia Clarke	1986	2009
3	51	All My Children	1	Alexandra Daddario	1986	2002-2003
4	30	Järngänget	0	Bill Skarsgård	1990	2000
5	27	Après lui	1	Pom Klementieff	1986	2007
6	23	Una rosa de Francia	1	Ana de Armas	1988	2006
7	37	Frankenstein	0	Dan Stevens	1982	2004
8	17	Le défi	1	Sofia Boutella	1982	2002
9	8	Story of Miss Oxygen	1	Katherine Langford	1996	2015

In [51]:

```
df_stars[30:70] # view rows that aren't shown above
```

Out[51]:

	credits	first_movie	gender	name	year_born	year_first_movie
30	38	Bella Thorne & Zendaya: Watch Me	1	Zendaya	1996	2011
31	21	Harry Potter and the Sorcerer's Stone	1	Emma Watson	1990	2001
32	63	North	1	Scarlett Johansson	1984	1994
33	4	The Refugees	1	Dafne Keen	1966	2014-2015
34	16	Wilt	1	Kelly Rohrbach	1990	2012
35	19	Lola: Érase una vez	1	Eiza González	1990	2007
36	35	My Family	1	Laura Haddock	1985	2007
37	53	Touched by an Angel	1	Mary Elizabeth Winstead	1984	1997
38	16	The Last of the Haussmans	0	Taron Egerton	1989	2012

In [52]:

```
df_stars.dtypes
```

Out[52]:

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

1.6.2 - cleaning data

All birth years are captured, but the years for Dafne Keen and Christian Navarro are incorrect (likely because they aren't on their IMDB pages). Will replace with correct birth years.

In [53]:
```
df_stars.loc[33, ['year_born']] = df_stars.loc[33, ['year_born']].replace(19
df_stars.loc[99, ['year_born']] = df_stars.loc[99, ['year_born']].replace(19
```

Check that they were successfully replaced:

In [54]:
```
df_stars.loc[33]
```

Out[54]:
```
credits                          4
first_movie           The Refugees
gender                           1
name                    Dafne Keen
year_born                     2001
year_first_movie         2014-2015
Name: 33, dtype: object
```

In [55]:
```
df_stars.loc[99]
```

Out[55]:
```
credits                                     13
first_movie         Day of the Dead 2: Contagium
gender                                       0
name                          Christian Navarro
year_born                                 1991
year_first_movie                          2005
Name: 99, dtype: object
```

Need to remove ranges of years for 'year_first_movie' and keep the earliest year.

In [56]:
```
extr = df_stars['year_first_movie'].str.extract(r'^(\d{4})', expand=False)
extr.head()
```

Out[56]:
```
0    2007
1    2001
2    2009
3    2002
4    2000
Name: year_first_movie, dtype: object
```

Need to convert 'year_first_movie' from an object to a numerical column to use for calculations.

In [57]:
```
df_stars['year_first_movie'] = pd.to_numeric(extr)
df_stars # check that data is clean now
```

				Howard		
83	19	Victorious	1	Halston Sage	1993	2011
84	48	Couples	1	Kate Beckinsale	1973	1975
85	51	How Did You Get In? We Didn't See You Leave	1	Connie Nielsen	1965	1984
86	4	Moana	1	Auli'i Cravalho	2000	2016
87	75	Star Trek: The Next Generation	1	Mädchen Amick	1970	1989
88	39	Neal 'N' Nikki	1	Serinda Swan	1984	2005
89	57	OVW: Christmas Chaos	0	Dave Bautista	1969	2001
90	19	Locked Up Abroad	1	Rose Leslie	1987	2008
91	27	Dil Jo Bhi Kahey...	1	Annabelle Wallis	1984	2005
92	23	NCIS	1	Zoey Deutch	1994	2011
93	13	Another Me	1	Sophie Turner	1996	2013

In [58]:
```
df_stars['age_at_first_movie'] = df_stars['year_first_movie'] - df_stars['ye
df_stars.head()
```

Out[58]:

	credits	first_movie	gender	name	year_born	year_first_movie	age_at_first_movie
0	25	Bubot	1	Gal Gadot	1985	2007	22
1	55	Tommaso	0	Tom Hardy	1977	2001	24
2	17	Doctors	1	Emilia Clarke	1986	2009	23
3	51	All My Children	1	Alexandra Daddario	1986	2002	16
4	30	Järngänget	0	Bill Skarsgård	1990	2000	10

1.7.1 - first movie at 17

In [59]:
```
count_17 = (df_stars.age_at_first_movie == 17).sum()
print('{0} performers made their first movie at 17.'.format(count_17))
```

```
8 performers made their first movie at 17.
```

Your output should look like this:
8 performers made their first movie at 17

1.7.2 - child actor

In [60]:
```python
df_stars.groupby('age_at_first_movie').count()
```

Out[60]:

	credits	first_movie	gender	name	year_born	year_first_movie
age_at_first_movie						
-1	1	1	1	1	1	1
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	3	3	3	3	3	3
9	3	3	3	3	3	3
10	5	5	5	5	5	5

Something wrong; can't be a negative age. Look at that person's data:

In [61]:
```python
idx_age = df_stars[df_stars.age_at_first_movie == -1].index[0]
df_stars.loc[idx_age]
```

Out[61]:
```
credits                             32
first_movie             Only Yesterday
gender                               1
name                      Daisy Ridley
year_born                         1992
year_first_movie                  1991
age_at_first_movie                  -1
Name: 63, dtype: object
```

Only Yesterday was originally released in 1991, but Daisy Ridley was involved in the English release in 2016. Looking back at her IMDB page, her first debut was in *ASOS Africa* in 2012. Will replace her entry with the appropriate data.

In [62]:
```python
df_stars.loc[63, ['first_movie', 'year_first_movie']] = df_stars.loc[63, ['f
df_stars.loc[63]
```

Out[62]:
```
credits                           32
first_movie              ASOS Africa
gender                             1
name                    Daisy Ridley
year_born                       1992
year_first_movie                2012
age_at_first_movie                -1
Name: 63, dtype: object
```

Need to re-calculate ages to account for this change.

In [63]:

```
df_stars['age_at_first_movie'] = df_stars['year_first_movie'] - df_stars['ye
df_stars.loc[63]
```

Out[63]:

```
credits                           32
first_movie              ASOS Africa
gender                             1
name                    Daisy Ridley
year_born                       1992
year_first_movie                2012
age_at_first_movie                20
Name: 63, dtype: object
```

In [64]:

```
df_stars.groupby('age_at_first_movie').count()
```

Out[64]:

	credits	first_movie	gender	name	year_born	year_first_movie
age_at_first_movie						
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
5	1	1	1	1	1	1
6	1	1	1	1	1	1
7	1	1	1	1	1	1
8	3	3	3	3	3	3
9	3	3	3	3	3	3
10	5	5	5	5	5	5
11	2	2	2	2	2	2

Ages now make sense. Will make logical column to determine child actors.

In [65]:

```
df_stars['child_actor'] = df_stars['age_at_first_movie'] < 12
df_stars.head()
```

Out[65]:

	credits	first_movie	gender	name	year_born	year_first_movie	age_at_first_movie	child_acto
0	25	Bubot	1	Gal Gadot	1985	2007	22	Fals
1	55	Tommaso	0	Tom Hardy	1977	2001	24	Fals
2	17	Doctors	1	Emilia Clarke	1986	2009	23	Fals
3	51	All My Children	1	Alexandra Daddario	1986	2002	16	Fals
4	30	Järngänget	0	Bill Skarsgård	1990	2000	10	Tru

In [66]:

```
print('{0} performers started as child actors.'.format(df_stars['child_actor
```

```
19 performers started as child actors.
```

1.7.3

In [67]:

```
idx_star = df_stars[df_stars.credits == df_stars.credits.max()].index[0]
```

In [68]:

```
print('The most prolific actor/actress of 2017 is {0}.'.format(df_stars.name
```

```
The most prolific actor/actress of 2017 is Sean Young.
```

1.8

Your answer here

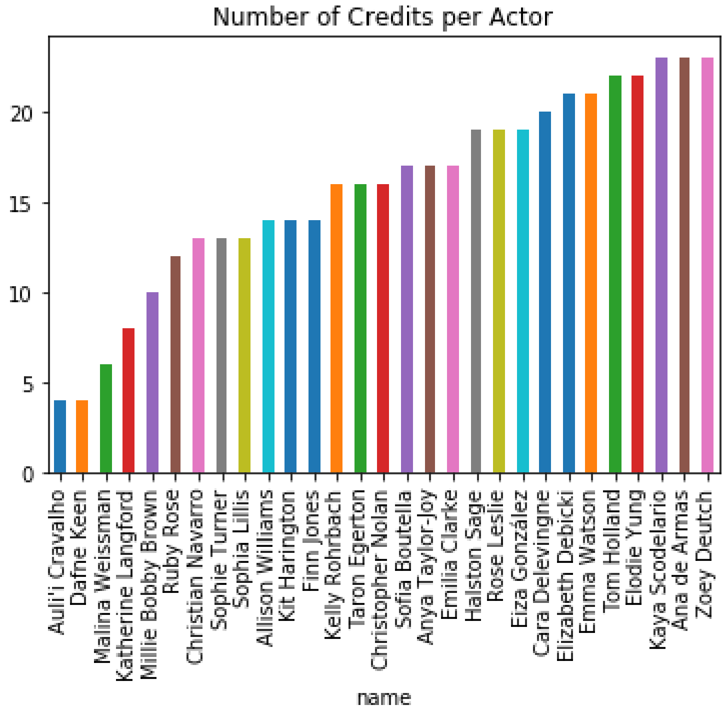
In [69]:

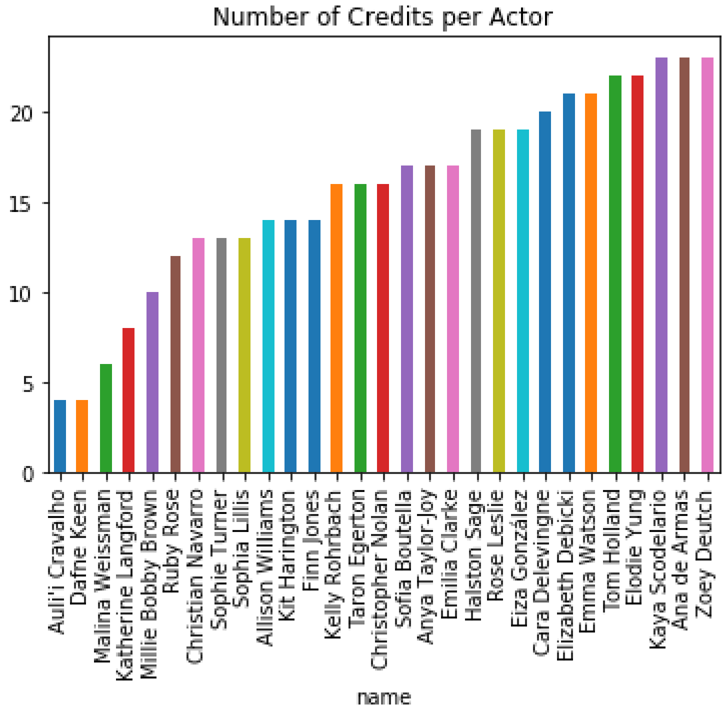
```
g = df_stars[['credits', 'name']]
g_sort = g.sort_values('credits')
```

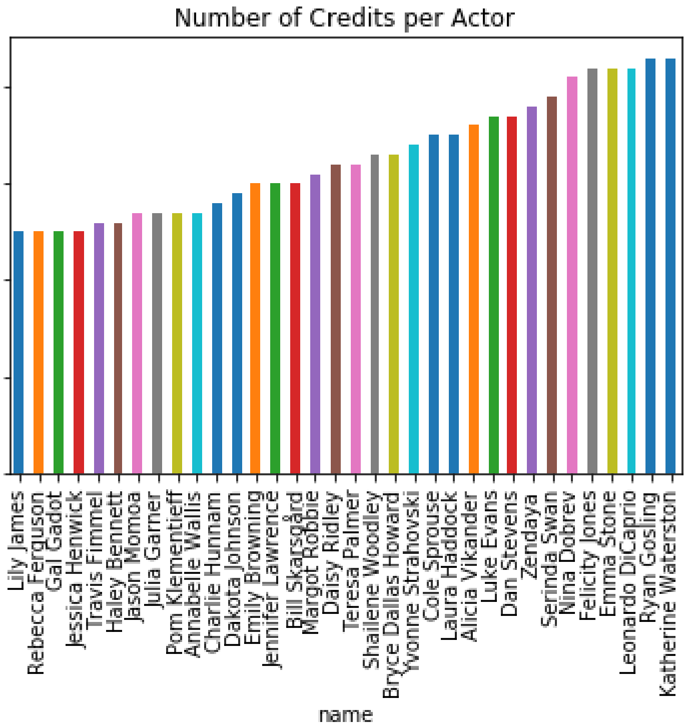
In [73]:

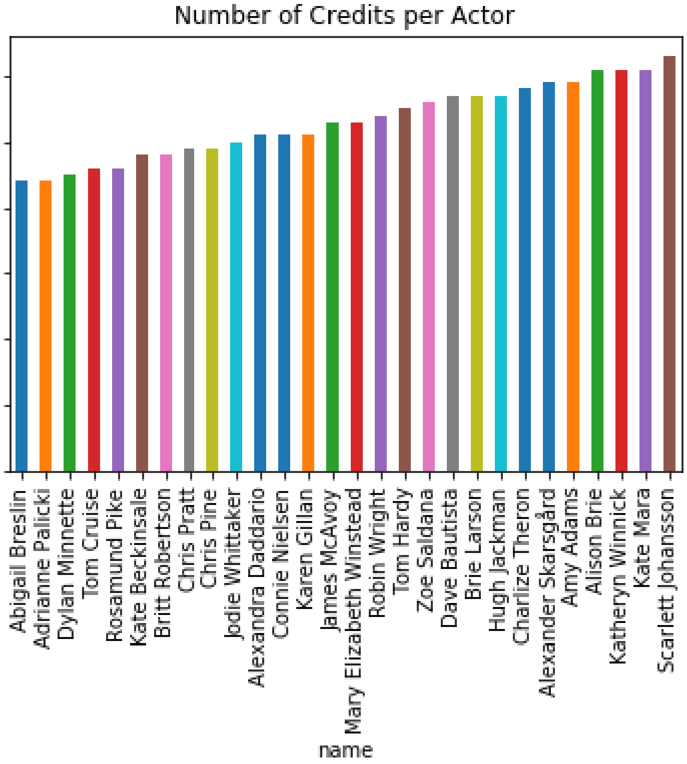
```
# group data so multiple smaller plots can be made (more readable overall),
z = g_sort.groupby(pd.cut(g_sort.credits, bins = 6, right = True))
```

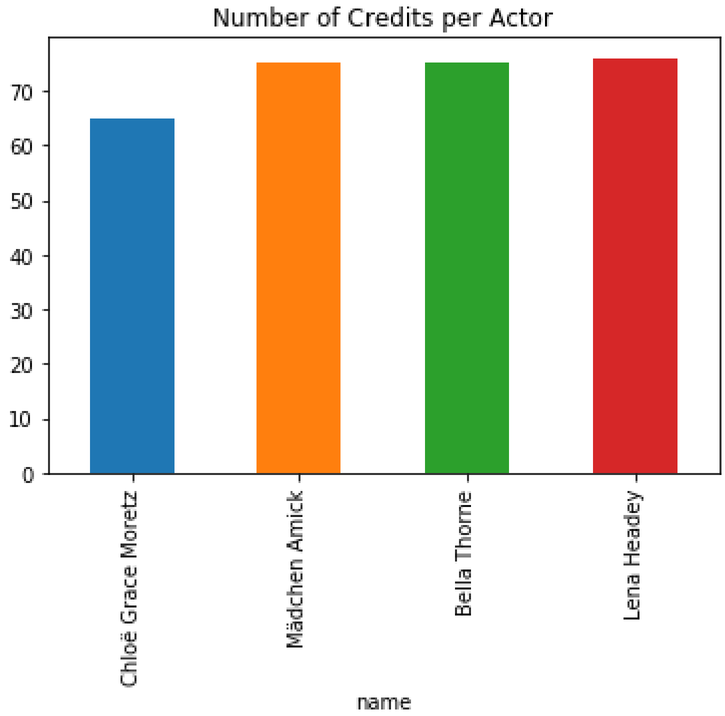
```
In [74]: z.plot.bar(x='name', y='credits', title='Number of Credits per Actor', legen
         plt.show()
         #plt.xlabel('Name of Star'); plt.ylabel('Number of Credits')
         #plt.title('Credits per Star')
```

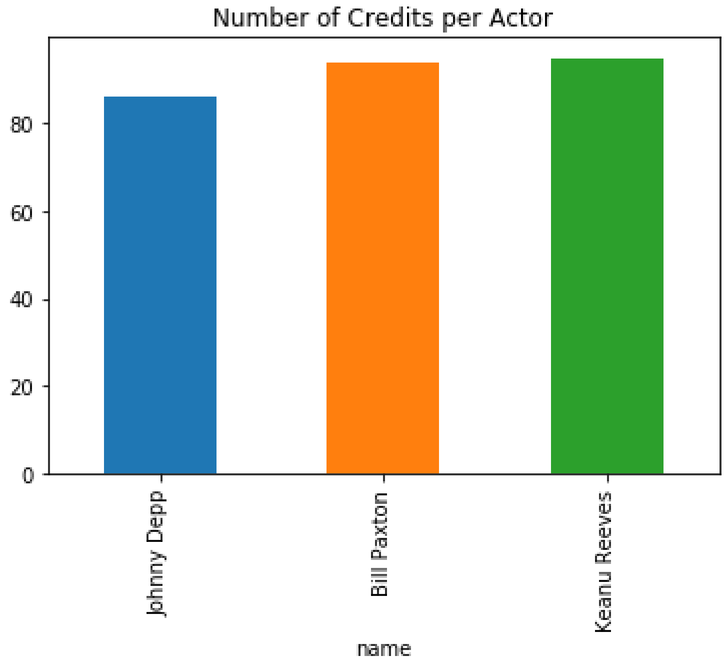


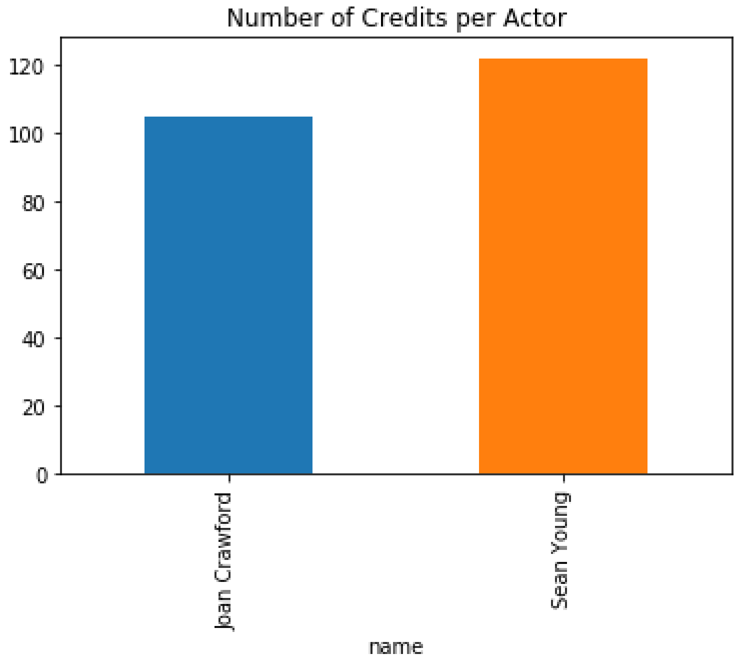



Number of Credits per Actor
0
10
20
30
40
Lily James
Rebecca Ferguson
Gal Gadot
Jessica Henwick
Travis Fimmel
Haley Bennett
Jason Momoa
Julia Garner
Pom Klementieff
Annabelle Wallis
Charlie Hunnam
Dakota Johnson
Emily Browning
Jennifer Lawrence
Bill Skarsgård
Margot Robbie
Daisy Ridley
Teresa Palmer
Shailene Woodley
Bryce Dallas Howard
Yvonne Strahovski
Cole Sprouse
Laura Haddock
Alicia Vikander
Luke Evans
Dan Stevens
Zendaya
Serinda Swan
Nina Dobrev
Felicity Jones
Emma Stone
Leonardo DiCaprio
Ryan Gosling
Katherine Waterston
name


Number of Credits per Actor
0
10
20
30
40
50
60
Abigail Breslin
Adrianne Palicki
Dylan Minnette
Tom Cruise
Rosamund Pike
Kate Beckinsale
Britt Robertson
Chris Pratt
Chris Pine
Jodie Whittaker
Alexandra Daddario
Connie Nielsen
Karen Gillan
James McAvoy
Mary Elizabeth Winstead
Robin Wright
Tom Hardy
Zoe Saldana
Dave Bautista
Brie Larson
Hugh Jackman
Charlize Theron
Alexander Skarsgård
Amy Adams
Alison Brie
Katheryn Winnick
Kate Mara
Scarlett Johansson
name


Number of Credits per Actor
70
60
50
40
30
20
10
0
Chloë Grace Moretz
Mädchen Amick
Bella Thorne
Lena Headey
name


Number of Credits per Actor
80
60
40
20
0
Johnny Depp
Bill Paxton
Keanu Reeves
name



I was trying to replicate ggplot's groupby/small multiple feature here. It somewhat worked, but I can't seem to customize the xlabel and ylabel for all of the charts (it only adds to the last one). The first grouping also appears twice, and I couldn't figure out why that kept happening. I tried numerous different methods to create these plots and customize them. Would really appreciate advice/thoughts on why the first grouping is duplicated and how to add labels to the axes of all of these charts.

```
In [72]: from IPython.core.display import HTML
         def css_styling(): styles = open("styles/cs109.css", "r").read(); return HTM
         css_styling()

Out[72]:
```