SIT742 Modern Data Science

Assignment 1

Student name:	Yash Ahuja
Student ID:	219608443
Date:	19-04-2021

Table of Contents

Part I - Tulip Hotel Web Logs Exploratory Data Analysis

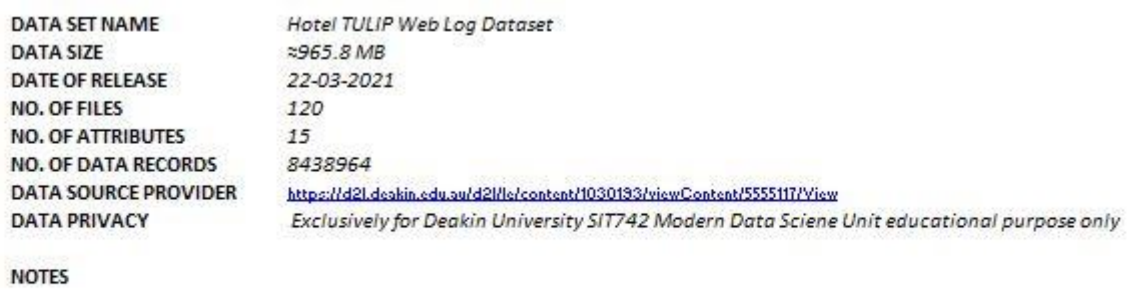
Hotel TULIP a five-star hotel located at Deakin University, and its CIO Dr Bear Guts has asked the Team-SIT742 team to analyse the weblogs files. As an employee for Hotel Tulip, working in the Information Technology Division, it is required to prepare a set of documentation for Team-SIT742 to allow them to understand the data being dealt with. Throughout this report, some source codes are to explore the weblog, which afterwards the information is presented to Dr Bear Guts in the format of a report.

1. Data ETL

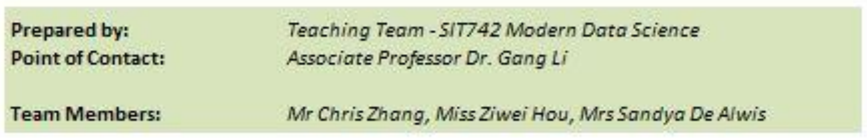
1.1. Data Loading

Fill the DataDictionary.xlsx with discovery from the result of 1.1 Data Loading from your notebook.

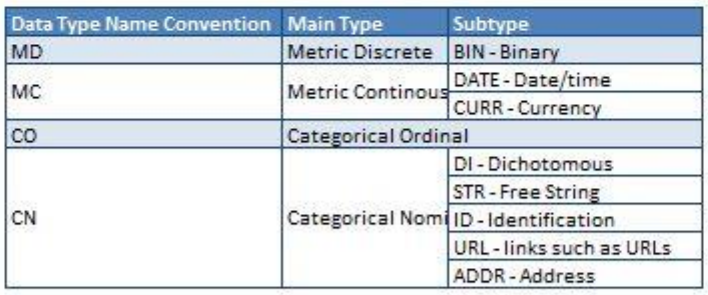
1.1.1. Dataset Description



DATA SET NAME	*Hotel TULIP Web Log Dataset*
DATA SIZE	*≈965.8 MB*
DATE OF RELEASE	*22-03-2021*
NO. OF FILES	*120*
NO. OF ATTRIBUTES	*15*
NO. OF DATA RECORDS	*8438964*
DATA SOURCE PROVIDER	https://d2l.deakin.edu.au/d2l/le/content/1030193/viewContent/5555117/View
DATA PRIVACY	*Exclusively for Deakin University SIT742 Modern Data Sciene Unit educational purpose only*
NOTES	

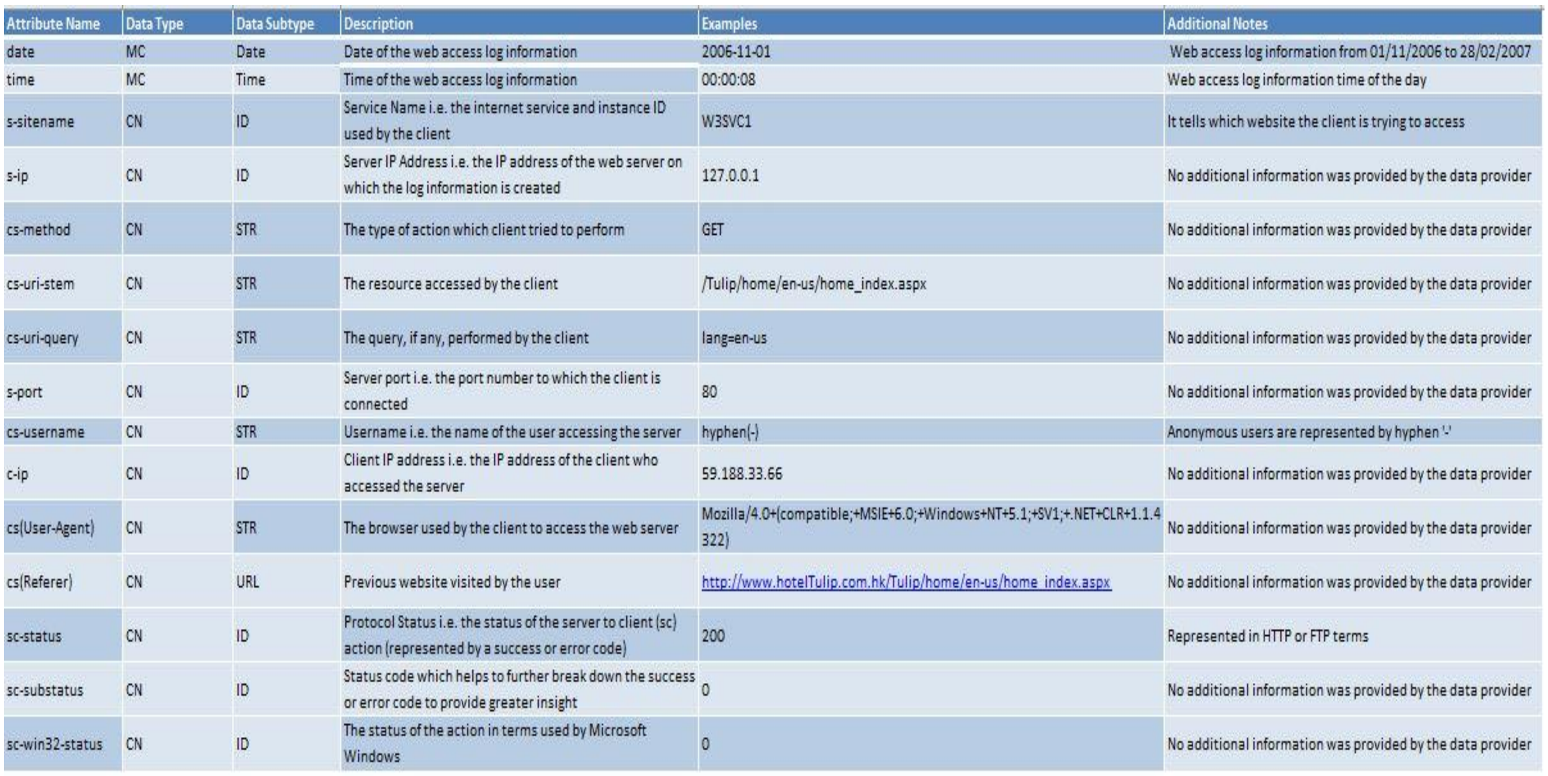


Prepared by:	*Teaching Team - SIT742 Modern Data Science*
Point of Contact:	*Associate Professor Dr. Gang Li*
Team Members:	*Mr Chris Zhang, Miss Ziwei Hou, Mrs Sandya De Alwis*



Data Type Name Convention	Main Type	Subtype
MD	Metric Discrete	BIN - Binary
MC	Metric Continous	DATE - Date/time
		CURR - Currency
CO	Categorical Ordinal	
CN	Categorical Nomi	DI - Dichotomous
		STR - Free String
		ID - Identification
		URL - links such as URLs
		ADDR - Address

1.1.2. Attribute Dictionary



Attribute Name	Data Type	Data Subtype	Description	Examples	Additional Notes
date	MC	Date	Date of the web access log information	2006-11-01	Web access log information from 01/11/2006 to 28/02/2007
time	MC	Time	Time of the web access log information	00:00:08	Web access log information time of the day
s-sitename	CN	ID	Service Name i.e. the internet service and instance ID used by the client	W3SVC1	It tells which website the client is trying to access
s-ip	CN	ID	Server IP Address i.e. the IP address of the web server on which the log information is created	127.0.0.1	No additional information was provided by the data provider
cs-method	CN	STR	The type of action which client tried to perform	GET	No additional information was provided by the data provider
cs-uri-stem	CN	STR	The resource accessed by the client	/Tulip/home/en-us/home_index.aspx	No additional information was provided by the data provider
cs-uri-query	CN	STR	The query, if any, performed by the client	lang=en-us	No additional information was provided by the data provider
s-port	CN	ID	Server port i.e. the port number to which the client is connected	80	No additional information was provided by the data provider
cs-username	CN	STR	Username i.e. the name of the user accessing the server	hyphen(-)	Anonymous users are represented by hyphen '-'
c-ip	CN	ID	Client IP address i.e. the IP address of the client who accessed the server	59.188.33.66	No additional information was provided by the data provider
cs(User-Agent)	CN	STR	The browser used by the client to access the web server	Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1;+SV1;+.NET+CLR+1.1.4322)	No additional information was provided by the data provider
cs(Referer)	CN	URL	Previous website visited by the user	http://www.hotelTulip.com.hk/Tulip/home/en-us/home_index.aspx	No additional information was provided by the data provider
sc-status	CN	ID	Protocol Status i.e. the status of the server to client (sc) action (represented by a success or error code)	200	Represented in HTTP or FTP terms
sc-substatus	CN	ID	Status code which helps to further break down the success or error code to provide greater insight	0	No additional information was provided by the data provider
sc-win32-status	CN	ID	The status of the action in terms used by Microsoft Windows	0	No additional information was provided by the data provider

1.2. Data Cleaning

Please add description of the following contents by yourself.

A. The number NAs for each column

```
date             0
time             0
s-sitename       12
s-ip             24
cs-method        24
cs-uri-stem      24
cs-uri-query     7886568
s-port           40
cs-username      8438960
c-ip             36
cs(User-Agent)   36
cs(Referer)      36
sc-status        36
sc-substatus     39
sc-win32-status  80
```

```
code-output:

s-sitename has number of NA records: 12
s-ip has number of NA records: 24
cs-method has number of NA records: 24
cs-uri-stem has number of NA records: 24
cs-uri-query has number of NA records: 7886568
s-port has number of NA records: 40
cs-username has number of NA records: 8438960
c-ip has number of NA records: 36
cs(User-Agent) has number of NA records: 36
cs(Referer) has number of NA records: 36
sc-status has number of NA records: 36
sc-substatus has number of NA records: 39
sc-win32-status has number of NA records: 80
```

B. The number of rows before removal NAs

```
8438964
```

C. The number of rows after removal NAs

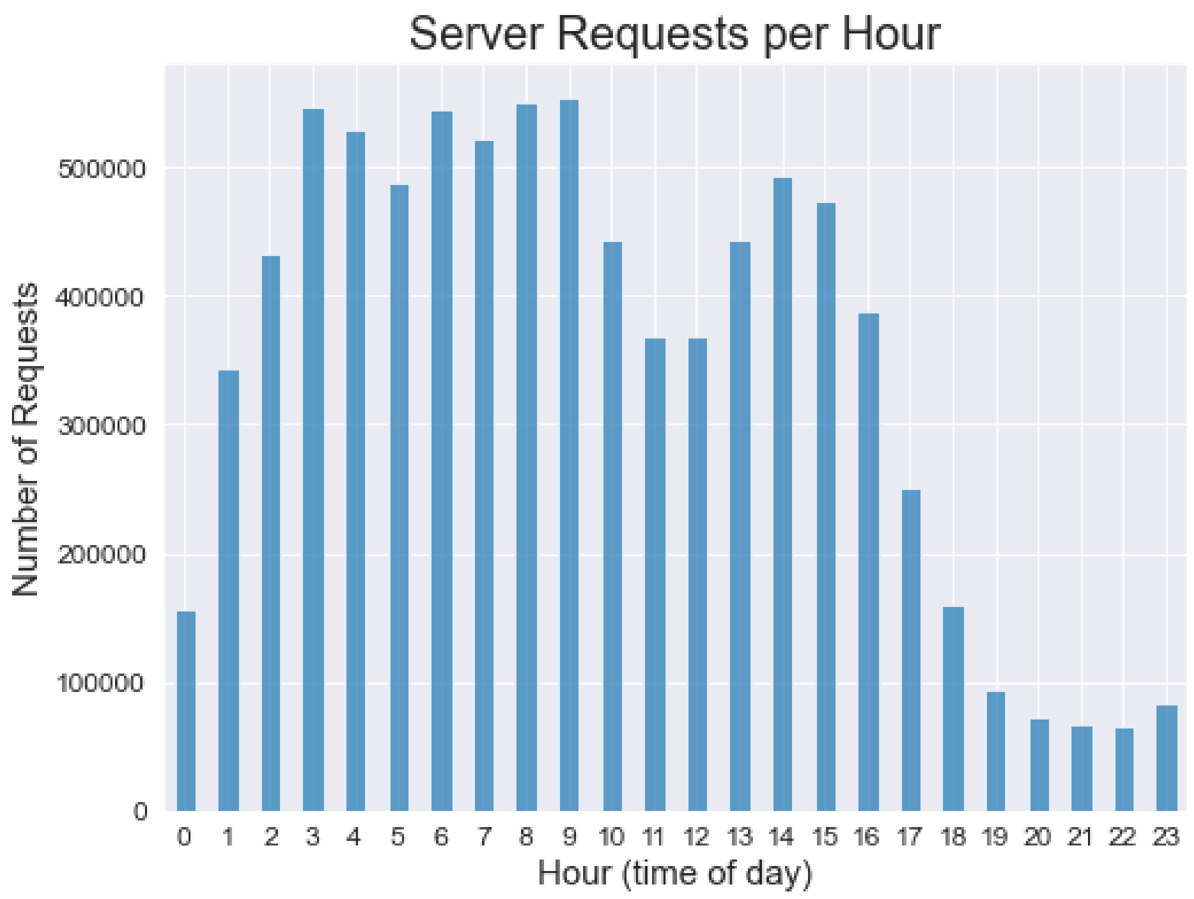
```
8438884
```

2. Data Statistics Description

2.1. Traffic Analysis

Please add description of the following contents by yourself.

A. *Please add a figure of Hourly Requests Bar Chart from your Notebook and elaborate the findings from the figure.*



We can see from the above figure that 9th hour of the day (9 – 10 AM) has the highest number of server requests whereas 22nd hour of the day (10 – 11 PM) has the lowest number of server requests. Overall, it can be said that the number of server requests are usually significantly high from 3 AM to 5 AM and 6 AM to 10 AM as compared to the other times of the day.

B. *Please add a table of filter result (hourly_request_amount >= 400000 & hourly_request_amount <= 490000)*

Hour (time of day)	*Hourly Request Amount*
2	*432315*
5	*487430*
10	*443621*
13	*442659*
15	*473376*

2.2. Server Analysis

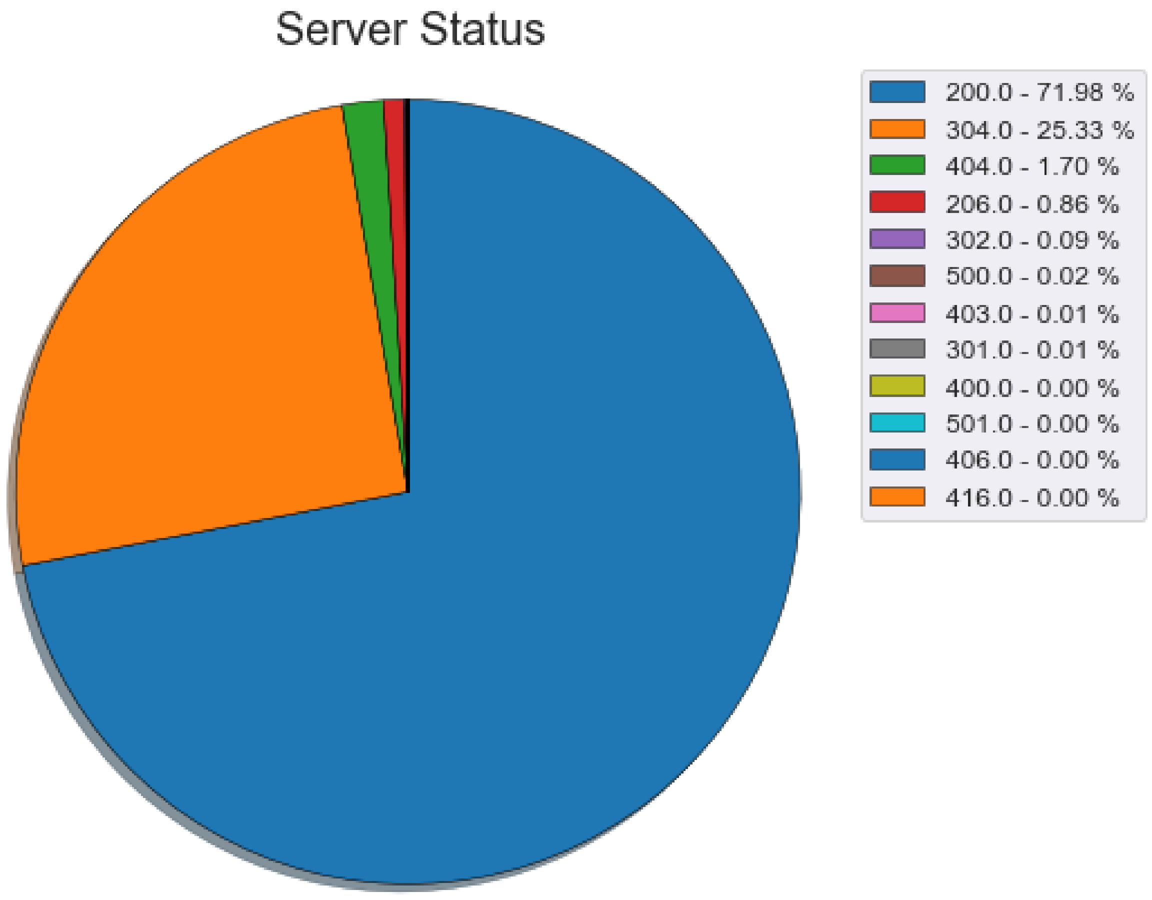
Please add description of the following contents by yourself.

A. *Please elaborate how many types of reported server status.*

Types of server status reported: 12

```
Server status type    Value Counts(Number of records)
200.0                 6074352
304.0                 2137486
404.0                 143646
206.0                 72493
302.0                 7368
500.0                 2020
403.0                 703
301.0                 436
400.0                 210
501.0                 113
406.0                 54
416.0                 3
```

B. *Please add a figure of Server Error Pie Chart from your Notebook and elaborate the findings from the figure.*



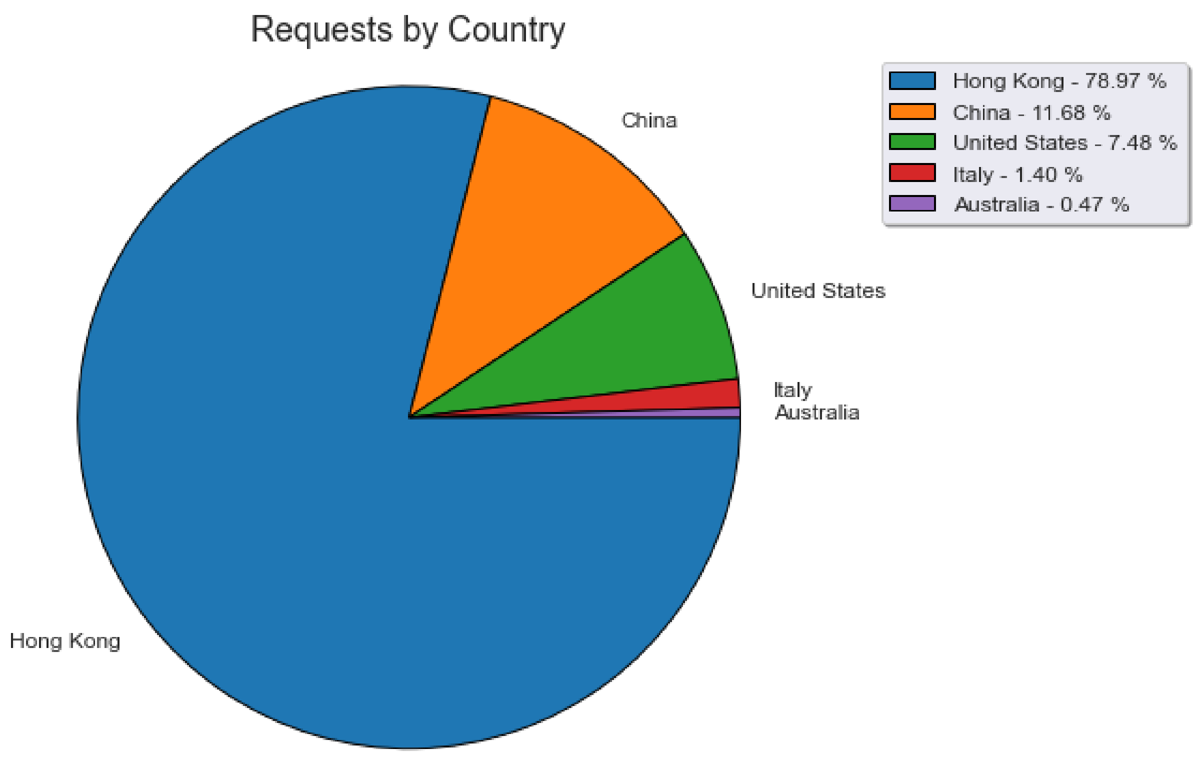
We can see from the above figure we can analyse the protocol status i.e. the status of the server to client action which is represented by a success or error code. From the figure, status code 200 has significantly high i.e. approx. 72% of the total number of server requests followed by status code 304 which has approx. 25% of the total server requests. Whereas, the status codes such as 404, 206, 302 have very less significant number of server requests and status codes such as 403, 301, 400, 501, 406, 416 have insignificant (approx. zero) server requests. Overall, it can be said that the status code 200 has most of the server requests as compared to other status codes.

2.3. Geographics Analysis

Please add description of the following contents by yourself.

A. *Please add a figure of Country distribution and list top 3 with the number of requests.*

```
1. The number of requests received in the period: 214
2. The number of countries involved in the period: 5
3. Requests by Country Figure:
```

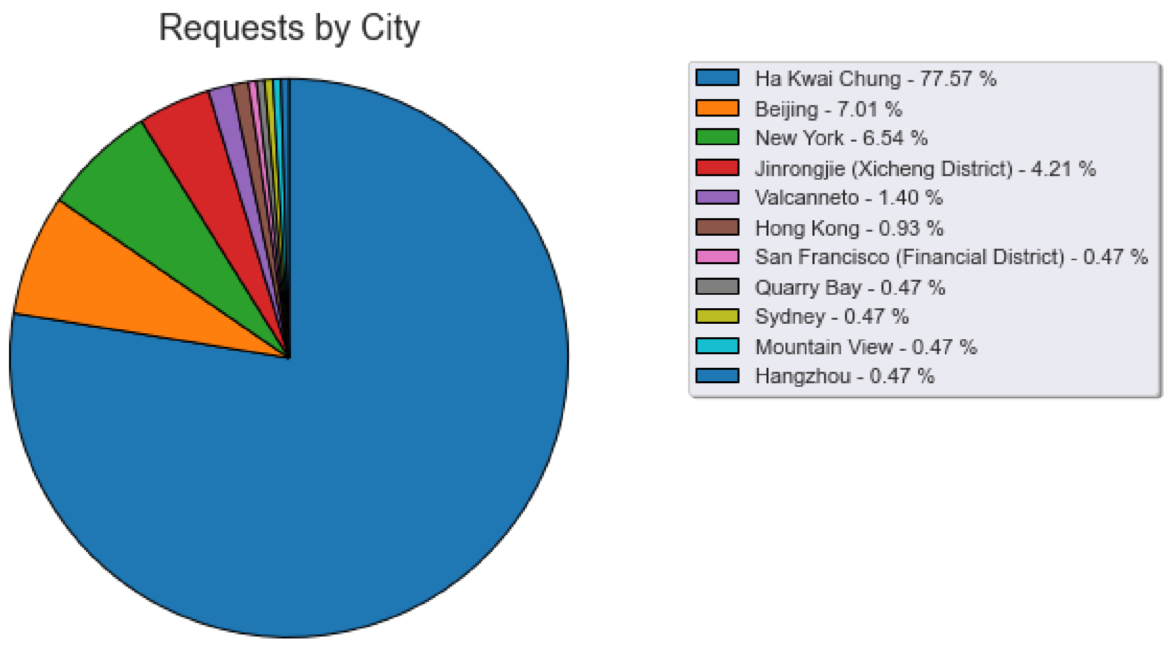


4.Top three countries with highest request numbers:

Country	Number of Requests
Hong Kong	169
China	25
United States	16

B. Please add a figure of City distribution and list top 3 with the number of requests.

1. The number of cities involved in the period: 11
2. Requests by City Figure:



3. Top three cities with highest request numbers:

City	Number of Requests
Ha Kwai Chung	166
Beijing	15
New York	14

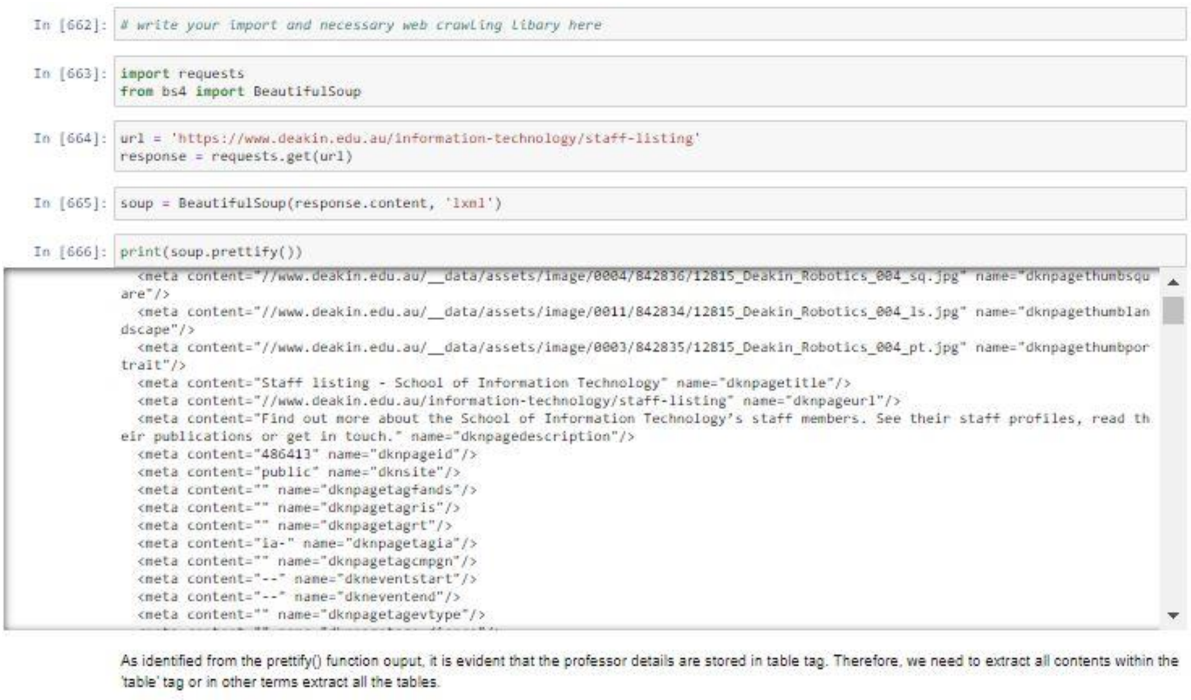
Part II - School of IT Professor Citation Information

To better introduce all the professors including the emeritus professor, the professor and also associate professor in Deakin University School of IT, faculty will need to know all the citation information on all professors. Google Scholar is a web search engine that freely indexes the metadata of articles on many authors. Majority of the professors choose to use google scholar to track their publications and research works. Therefore, the web crawling on google scholar will be able to have the citation information obtained across all the professors (who have the google scholar profile).

3. Professor List Generation

3.1. Import Web Crawling Library

Please fill this part with the screenshot of your code for import your own web crawling library.



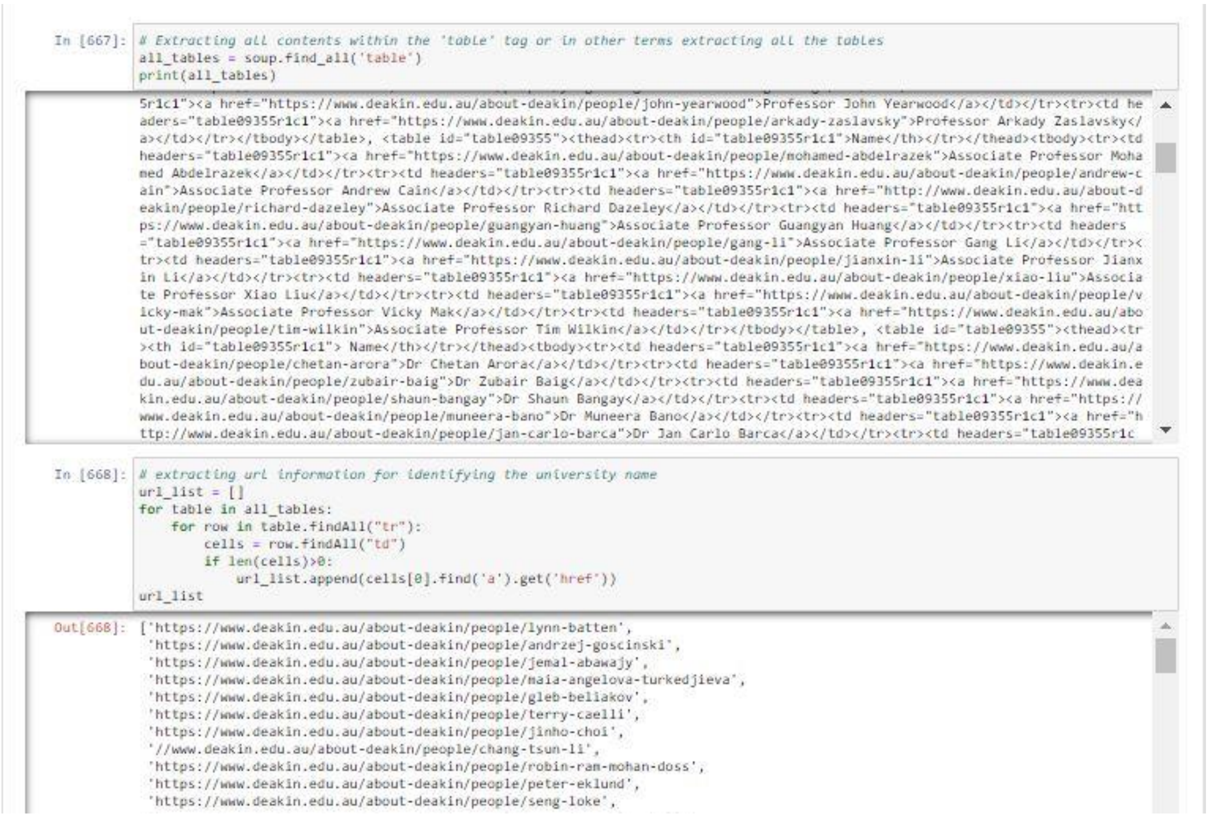
```
In [662]: # write your import and necessary web crawling libary here

In [663]: import requests
          from bs4 import BeautifulSoup

In [664]: url = 'https://www.deakin.edu.au/information-technology/staff-listing'
          response = requests.get(url)

In [665]: soup = BeautifulSoup(response.content, 'lxml')

In [666]: print(soup.prettify())
```

```
  <meta content="//www.deakin.edu.au/__data/assets/image/0004/842836/12815_Deakin_Robotics_004_sq.jpg" name="dknpagethumbsqu
are"/>
  <meta content="//www.deakin.edu.au/__data/assets/image/0011/842834/12815_Deakin_Robotics_004_ls.jpg" name="dknpagethumblan
dscape"/>
  <meta content="//www.deakin.edu.au/__data/assets/image/0003/842835/12815_Deakin_Robotics_004_pt.jpg" name="dknpagethumbpor
trait"/>
  <meta content="Staff listing - School of Information Technology" name="dknpagetitle"/>
  <meta content="//www.deakin.edu.au/information-technology/staff-listing" name="dknpageurl"/>
  <meta content="Find out more about the School of Information Technology's staff members. See their staff profiles, read th
eir publications or get in touch." name="dknpagedescription"/>
  <meta content="486413" name="dknpageid"/>
  <meta content="public" name="dknsite"/>
  <meta content="" name="dknpagetagfands"/>
  <meta content="" name="dknpagetagris"/>
  <meta content="" name="dknpagetagrt"/>
  <meta content="ia-" name="dknpagetagia"/>
  <meta content="" name="dknpagetagcmpgn"/>
  <meta content="--" name="dkneventstart"/>
  <meta content="--" name="dkneventend"/>
  <meta content="" name="dknpagetagevtype"/>
```

As identified from the prettify() function ouput, it is evident that the professor details are stored in table tag. Therefore, we need to extract all contents within the 'table' tag or in other terms extract all the tables.

3.2. Find all professors in School of IT

Please fill this part with the screenshot of your code for generating the professor name list csv. The screen shot will also include the results of the running on the code.

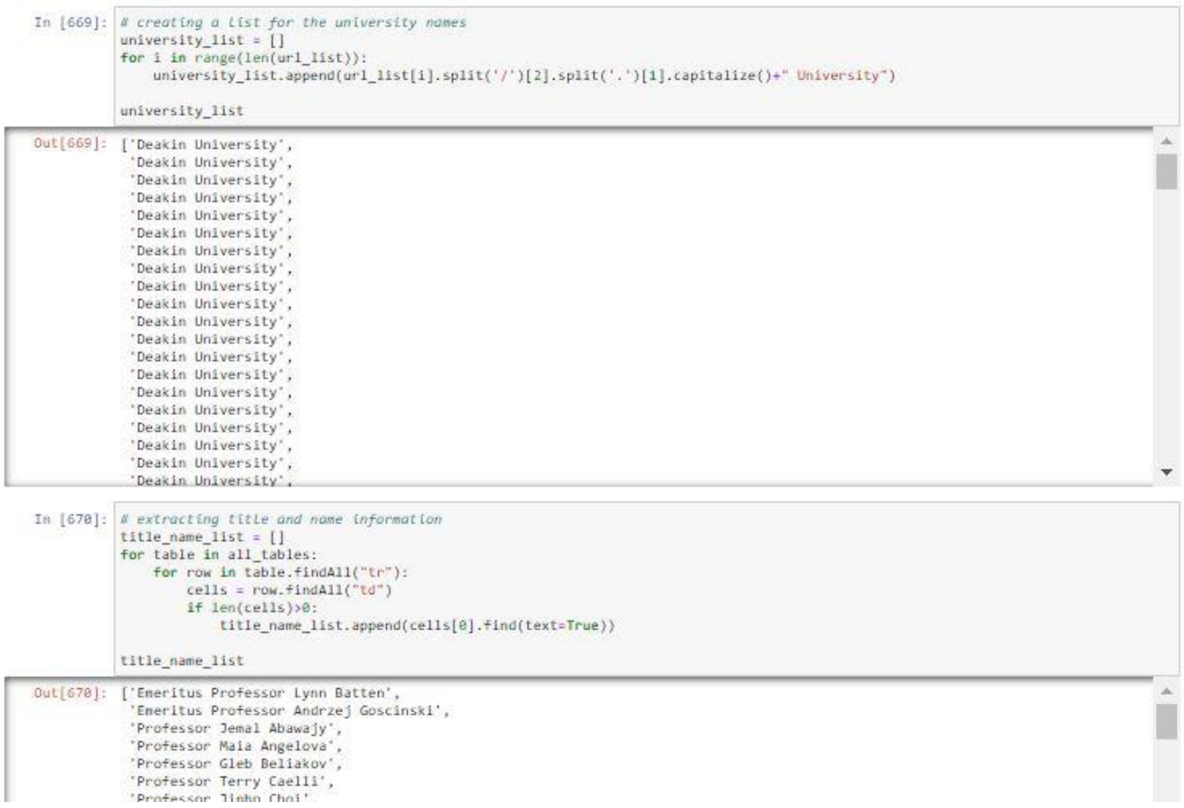


```
In [667]: # Extracting all contents within the 'table' tag or in other terms extracting all the tables
          all_tables = soup.find_all('table')
          print(all_tables)
```

```
5r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/john-yearwood">Professor John Yearwood</a></td></tr><tr><td he
aders="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/arkady-zaslavsky">Professor Arkady Zaslavsky</
a></td></tr></tbody></table>, <table id="table09355"><thead><tr><th id="table09355r1c1">Name</th></tr></thead><tbody><tr><td
headers="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/mohamed-abdelrazek">Associate Professor Moha
med Abdelrazek</a></td></tr><tr><td headers="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/andrew-c
ain">Associate Professor Andrew Cain</a></td></tr><tr><td headers="table09355r1c1"><a href="http://www.deakin.edu.au/about-d
eakin/people/richard-dazeley">Associate Professor Richard Dazeley</a></td></tr><tr><td headers="table09355r1c1"><a href="htt
ps://www.deakin.edu.au/about-deakin/people/guangyan-huang">Associate Professor Guangyan Huang</a></td></tr><tr><td headers
="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/gang-li">Associate Professor Gang Li</a></td></tr><
tr><td headers="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/jianxin-li">Associate Professor Jianx
in Li</a></td></tr><tr><td headers="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/xiao-liu">Associa
te Professor Xiao Liu</a></td></tr><tr><td headers="table09355r1c1"><a href="https://www.deakin.edu.au/about-deakin/people/v
icky-mak">Associate Professor Vicky Mak</a></td></tr><tr><td headers="table09355r1c1"><a href="https://www.deakin.edu.au/abo
ut-deakin/people/tim-wilkin">Associate Professor Tim Wilkin</a></td></tr></tbody></table>, <table id="table09355"><thead><tr
><th id="table09355r1c1"> Name</th></tr></thead><tbody><tr><td headers="table09355r1c1"><a href="https://www.deakin.edu.au/a
bout-deakin/people/chetan-arora">Dr Chetan Arora</a></td></tr><tr><td headers="table09355r1c1"><a href="https://www.deakin.e
du.au/about-deakin/people/zubair-baig">Dr Zubair Baig</a></td></tr><tr><td headers="table09355r1c1"><a href="https://www.dea
kin.edu.au/about-deakin/people/shaun-bangay">Dr Shaun Bangay</a></td></tr><tr><td headers="table09355r1c1"><a href="https://
www.deakin.edu.au/about-deakin/people/muneera-bano">Dr Muneera Bano</a></td></tr><tr><td headers="table09355r1c1"><a href="h
ttp://www.deakin.edu.au/about-deakin/people/jan-carlo-barca">Dr Jan Carlo Barca</a></td></tr><tr><td headers="table09355r1c
```

```
In [668]: # extracting url information for identifying the university name
          url_list = []
          for table in all_tables:
              for row in table.findAll("tr"):
                  cells = row.findAll("td")
                  if len(cells)>0:
                      url_list.append(cells[0].find('a').get('href'))
          url_list
```

```
Out[668]: ['https://www.deakin.edu.au/about-deakin/people/lynn-batten',
           'https://www.deakin.edu.au/about-deakin/people/andrzej-goscinski',
           'https://www.deakin.edu.au/about-deakin/people/jemal-abawajy',
           'https://www.deakin.edu.au/about-deakin/people/maia-angelova-turkedjieva',
           'https://www.deakin.edu.au/about-deakin/people/gleb-beliakov',
           'https://www.deakin.edu.au/about-deakin/people/terry-caelli',
           'https://www.deakin.edu.au/about-deakin/people/jinho-choi',
           '//www.deakin.edu.au/about-deakin/people/chang-tsun-li',
           'https://www.deakin.edu.au/about-deakin/people/robin-ram-mohan-doss',
           'https://www.deakin.edu.au/about-deakin/people/peter-eklund',
           'https://www.deakin.edu.au/about-deakin/people/seng-loke',
```

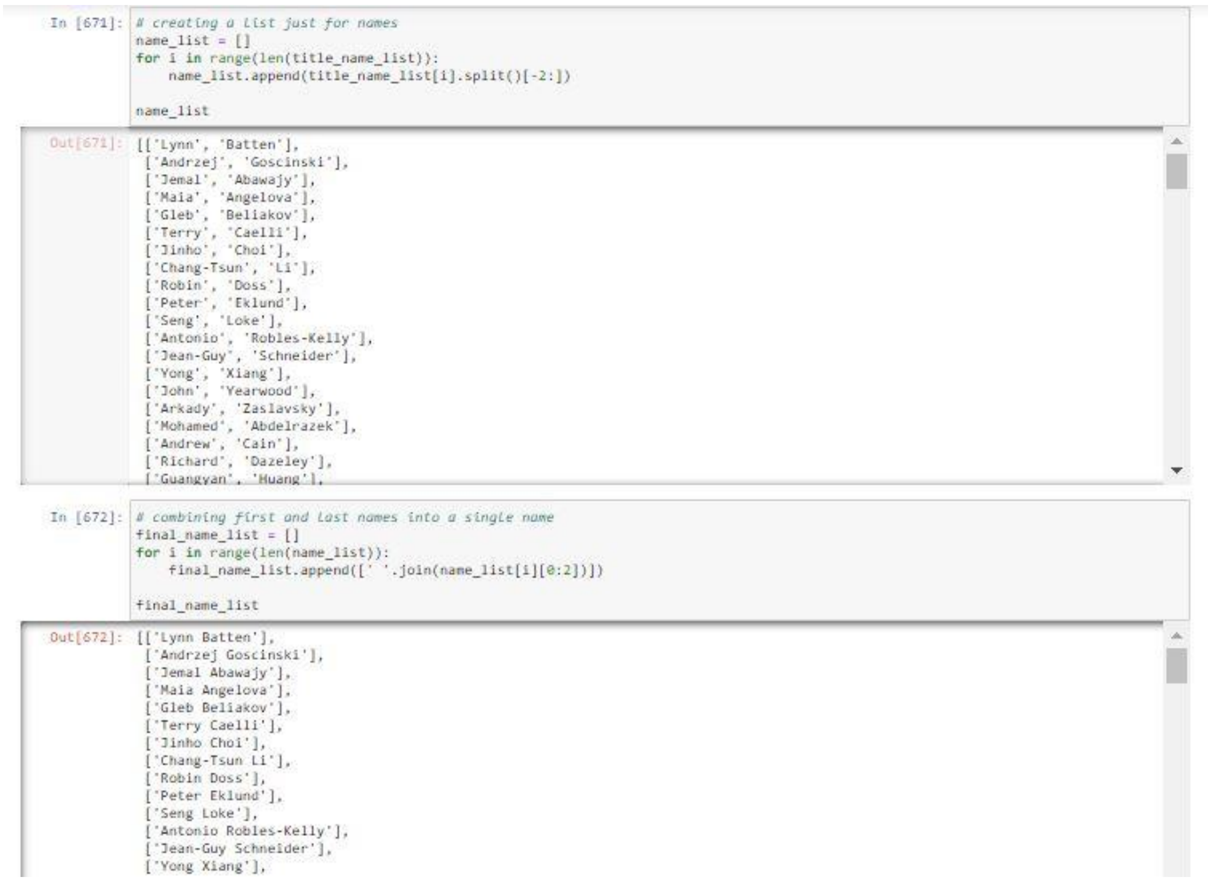


```
In [669]: # creating a list for the university names
          university_list = []
          for i in range(len(url_list)):
              university_list.append(url_list[i].split('/')[2].split('.')[1].capitalize()+" University")

          university_list
```

```
Out[669]: ['Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
           'Deakin University',
```

```
In [670]: # extracting title and name information
          title_name_list = []
          for table in all_tables:
              for row in table.findAll("tr"):
                  cells = row.findAll("td")
                  if len(cells)>0:
                      title_name_list.append(cells[0].find(text=True))

          title_name_list
```

```
Out[670]: ['Emeritus Professor Lynn Batten',
           'Emeritus Professor Andrzej Goscinski',
           'Professor Jemal Abawajy',
           'Professor Maia Angelova',
           'Professor Gleb Beliakov',
           'Professor Terry Caelli',
           'Professor Jinho Choi',
```



```
In [671]: # creating a list just for names
          name_list = []
          for i in range(len(title_name_list)):
              name_list.append(title_name_list[i].split()[-2:])

          name_list
```

```
Out[671]: [['Lynn', 'Batten'],
           ['Andrzej', 'Goscinski'],
           ['Jemal', 'Abawajy'],
           ['Maia', 'Angelova'],
           ['Gleb', 'Beliakov'],
           ['Terry', 'Caelli'],
           ['Jinho', 'Choi'],
           ['Chang-Tsun', 'Li'],
           ['Robin', 'Doss'],
           ['Peter', 'Eklund'],
           ['Seng', 'Loke'],
           ['Antonio', 'Robles-Kelly'],
           ['Jean-Guy', 'Schneider'],
           ['Yong', 'Xiang'],
           ['John', 'Yearwood'],
           ['Arkady', 'Zaslavsky'],
           ['Mohamed', 'Abdelrazek'],
           ['Andrew', 'Cain'],
           ['Richard', 'Dazeley'],
           ['Guangyan', 'Huang'],
```

```
In [672]: # combining first and last names into a single name
          final_name_list = []
          for i in range(len(name_list)):
              final_name_list.append([' '.join(name_list[i][0:2])])

          final_name_list
```

```
Out[672]: [['Lynn Batten'],
           ['Andrzej Goscinski'],
           ['Jemal Abawajy'],
           ['Maia Angelova'],
           ['Gleb Beliakov'],
           ['Terry Caelli'],
           ['Jinho Choi'],
           ['Chang-Tsun Li'],
           ['Robin Doss'],
           ['Peter Eklund'],
           ['Seng Loke'],
           ['Antonio Robles-Kelly'],
           ['Jean-Guy Schneider'],
           ['Yong Xiang'],
In [673]: # creating a list just for titles
          title_list = []
          for i in range(len(title_name_list)):
              title_list.append(title_name_list[i].split()[:-2])

          title_list
```

```
Out[673]: [['Emeritus', 'Professor'],
           ['Emeritus', 'Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Associate', 'Professor'],
           ['Associate', 'Professor'],
           ['Associate', 'Professor'],
           ['Associate', 'Professor'],
```

```
In [674]: # Combining elements of title list to assign correct titles
          final_title_list = []
          for i in range(len(title_list)):
              final_title_list.append([' '.join(title_list[i][0:3])])

          final_title_list
```

```
Out[674]: [['Emeritus Professor'],
           ['Emeritus Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
           ['Professor'],
```

It should be noted that in the final_title_list, our focus is only on Emeritus Professor, Professor and Associate Professor titles as per the requirements. But, the list contains all the other titles as well and there are some discrepancies associated with title i.e. Industry Staff, as one of their staff Abbas Kudrati has title "Professor" associated with their name. So, we need to make sure to remove this record after filtering the records for Emeritus Professor, Professor and Associate Professor titles and before saving the final professor list data to csv.

```
In [675]: df = pd.DataFrame({'Name':final_name_list, 'Title':final_title_list, 'University':university_list})
          df.head()
```

Out[675]:

	Name	Title	University
0	[Lynn Batten]	[Emeritus Professor]	Deakin University
1	[Andrzej Goscinski]	[Emeritus Professor]	Deakin University
2	[Jemal Abawajy]	[Professor]	Deakin University
3	[Maia Angelova]	[Professor]	Deakin University
4	[Gleb Beliakov]	[Professor]	Deakin University

We need to remove the [] from the cell values for Name and Title columns.

```
In [676]: columns = ['Name', 'Title']
          for name in columns:
              for i in range(df.shape[0]):
                  df[name][i] = df[name][i][0]
```

```
In [677]: df.head()
```

Out[677]:

	Name	Title	University
0	Lynn Batten	Emeritus Professor	Deakin University
1	Andrzej Goscinski	Emeritus Professor	Deakin University
2	Jemal Abawajy	Professor	Deakin University
3	Maia Angelova	Professor	Deakin University
4	Gleb Beliakov	Professor	Deakin University

Filtering out the data for only Emeritus Professor, Professor, Associate Professor titles as per the requirement.

```
In [678]: Professor_Type = ['Emeritus Professor', 'Professor', 'Associate Professor']
          df_filtered = df[df['Title'].isin(Professor_Type)]
          df_filtered
```

Out[678]:

	Name	Title	University
0	Lynn Batten	Emeritus Professor	Deakin University
1	Andrzej Goscinski	Emeritus Professor	Deakin University
2	Jemal Abawajy	Professor	Deakin University
3	Maia Angelova	Professor	Deakin University
4	Gleb Beliakov	Professor	Deakin University
5	Terry Caelli	Professor	Deakin University
6	Jinho Choi	Professor	Deakin University
7	Chang-Tsun Li	Professor	Deakin University
8	Robin Doss	Professor	Deakin University
9	Peter Eklund	Professor	Deakin University
10	Seng Loke	Professor	Deakin University
11	Antonio Robles-Kelly	Professor	Deakin University
12	Jean-Guy Schneider	Professor	Deakin University
13	Yong Xiang	Professor	Deakin University
14	John Yearwood	Professor	Deakin University
15	Arkady Zaslavsky	Professor	Deakin University
16	Mohamed Abdelrazek	Associate Professor	Deakin University
17	Andrew Cain	Associate Professor	Deakin University
18	Richard Dazeley	Associate Professor	Deakin University
19	Guangyan Huang	Associate Professor	Deakin University
20	Gang Li	Associate Professor	Deakin University
21	Jianxin Li	Associate Professor	Deakin University
22	Xiao Liu	Associate Professor	Deakin University
23	Vicky Mak	Associate Professor	Deakin University
24	Tim Wilkin	Associate Professor	Deakin University
170	Abbas Kudrati	Professor	Deakin University

Removing the record for Abbas Kudrati as the actual title for this record is Industry Staff.

```
In [679]: df_filtered.drop(index=170, axis=0, inplace=True)
```

```
In [680]: # Dimensions (rows, columns) of the final professor list dataframe
          df_filtered.shape
```

```
Out[680]: (25, 3)
```

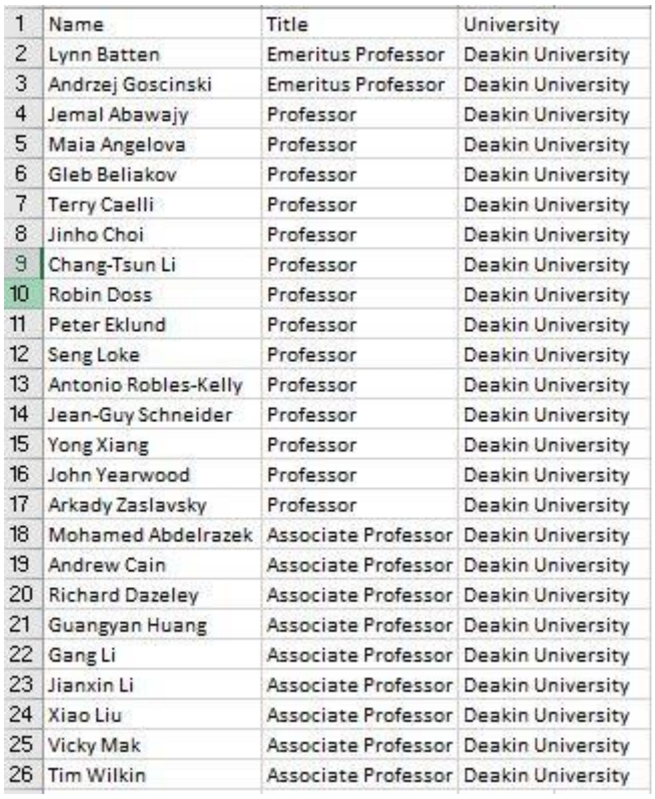
```
In [681]: # renaming the final dataframe
          df_prof = df_filtered.copy()
```

Saving the professor name list dataframe to csv format under the name Professor-name-list.

```
In [682]: df_prof.to_csv('Professor-name-list.csv', index=False)
```

3.2.1. Professor Name List CSV

Please fill this part with the screenshot of your csv.

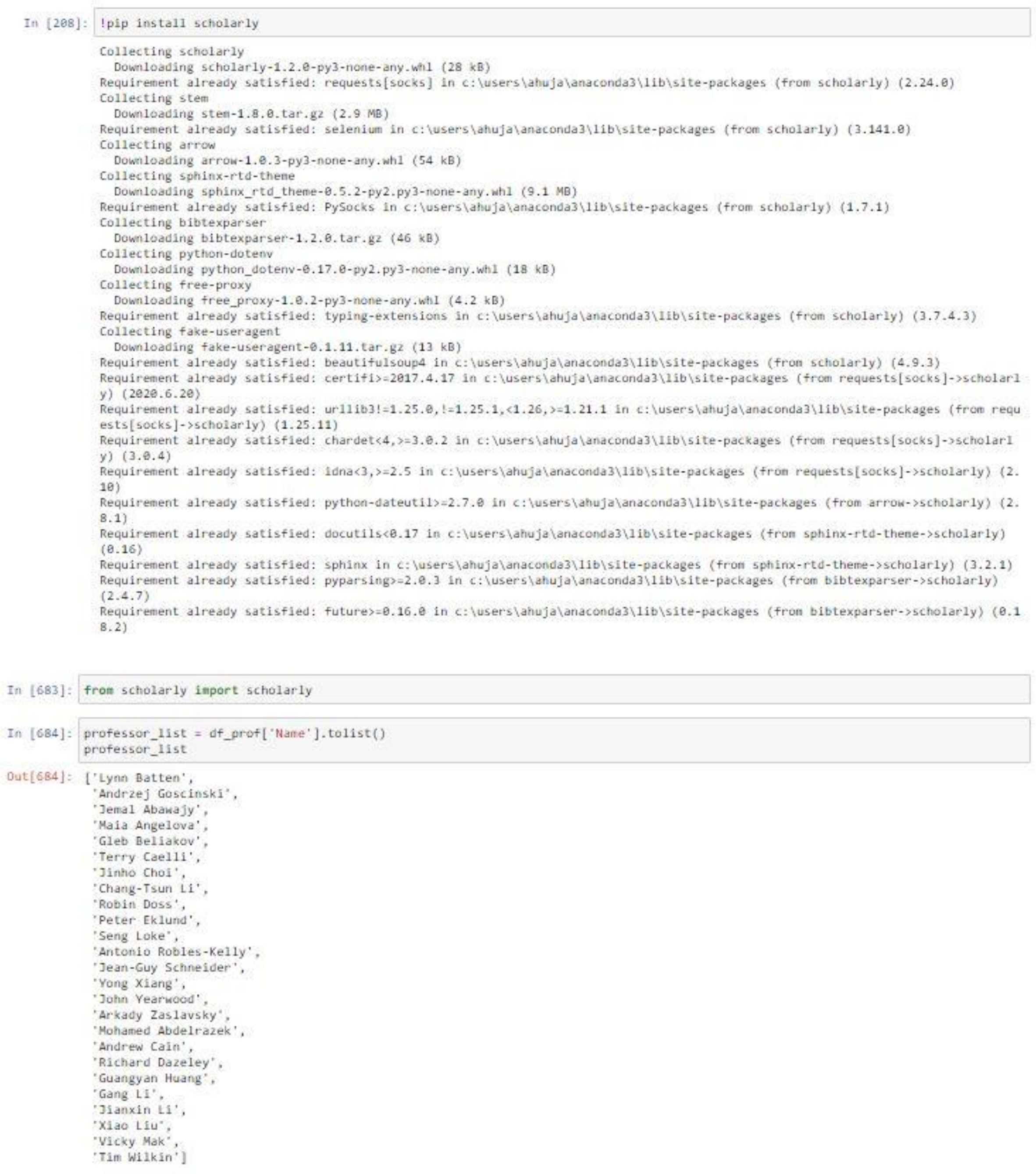


1	Name	Title	University
2	Lynn Batten	Emeritus Professor	Deakin University
3	Andrzej Goscinski	Emeritus Professor	Deakin University
4	Jemal Abawajy	Professor	Deakin University
5	Maia Angelova	Professor	Deakin University
6	Gleb Beliakov	Professor	Deakin University
7	Terry Caelli	Professor	Deakin University
8	Jinho Choi	Professor	Deakin University
9	Chang-Tsun Li	Professor	Deakin University
10	Robin Doss	Professor	Deakin University
11	Peter Eklund	Professor	Deakin University
12	Seng Loke	Professor	Deakin University
13	Antonio Robles-Kelly	Professor	Deakin University
14	Jean-Guy Schneider	Professor	Deakin University
15	Yong Xiang	Professor	Deakin University
16	John Yearwood	Professor	Deakin University
17	Arkady Zaslavsky	Professor	Deakin University
18	Mohamed Abdelrazek	Associate Professor	Deakin University
19	Andrew Cain	Associate Professor	Deakin University
20	Richard Dazeley	Associate Professor	Deakin University
21	Guangyan Huang	Associate Professor	Deakin University
22	Gang Li	Associate Professor	Deakin University
23	Jianxin Li	Associate Professor	Deakin University
24	Xiao Liu	Associate Professor	Deakin University
25	Vicky Mak	Associate Professor	Deakin University
26	Tim Wilkin	Associate Professor	Deakin University

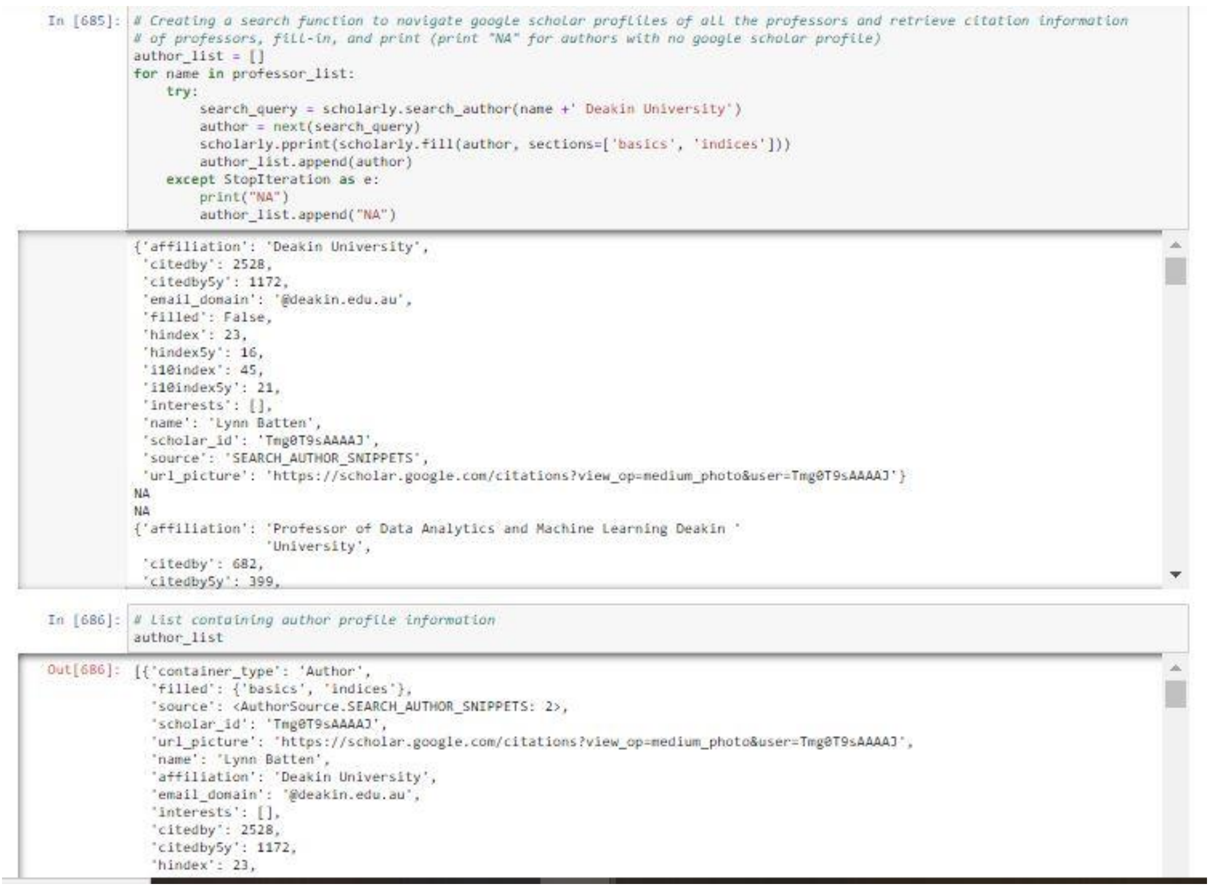
4. Professor Citation Information Generation

4.1. Search the google scholar for all

Please fill this part with the screenshot of the code for generating the professor citation information (include the actual crawling steps).



```
In [208]: !pip install scholarly

          Collecting scholarly
            Downloading scholarly-1.2.0-py3-none-any.whl (28 kB)
          Requirement already satisfied: requests[socks] in c:\users\ahuja\anaconda3\lib\site-packages (from scholarly) (2.24.0)
          Collecting stem
            Downloading stem-1.8.0.tar.gz (2.9 MB)
          Requirement already satisfied: selenium in c:\users\ahuja\anaconda3\lib\site-packages (from scholarly) (3.141.0)
          Collecting arrow
            Downloading arrow-1.0.3-py3-none-any.whl (54 kB)
          Collecting sphinx-rtd-theme
            Downloading sphinx_rtd_theme-0.5.2-py2.py3-none-any.whl (9.1 MB)
          Requirement already satisfied: PySocks in c:\users\ahuja\anaconda3\lib\site-packages (from scholarly) (1.7.1)
          Collecting bibtexparser
            Downloading bibtexparser-1.2.0.tar.gz (46 kB)
          Collecting python-dotenv
            Downloading python_dotenv-0.17.0-py2.py3-none-any.whl (18 kB)
          Collecting free-proxy
            Downloading free_proxy-1.0.2-py3-none-any.whl (4.2 kB)
          Requirement already satisfied: typing-extensions in c:\users\ahuja\anaconda3\lib\site-packages (from scholarly) (3.7.4.3)
          Collecting fake-useragent
            Downloading fake-useragent-0.1.11.tar.gz (13 kB)
          Requirement already satisfied: beautifulsoup4 in c:\users\ahuja\anaconda3\lib\site-packages (from scholarly) (4.9.3)
          Requirement already satisfied: certifi>=2017.4.17 in c:\users\ahuja\anaconda3\lib\site-packages (from requests[socks]->scholarl
          y) (2020.6.20)
          Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\ahuja\anaconda3\lib\site-packages (from requ
          ests[socks]->scholarly) (1.25.11)
          Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\ahuja\anaconda3\lib\site-packages (from requests[socks]->scholarl
          y) (3.0.4)
          Requirement already satisfied: idna<3,>=2.5 in c:\users\ahuja\anaconda3\lib\site-packages (from requests[socks]->scholarly) (2.
          10)
          Requirement already satisfied: python-dateutil>=2.7.0 in c:\users\ahuja\anaconda3\lib\site-packages (from arrow->scholarly) (2.
          8.1)
          Requirement already satisfied: docutils<0.17 in c:\users\ahuja\anaconda3\lib\site-packages (from sphinx-rtd-theme->scholarly)
          (0.16)
          Requirement already satisfied: sphinx in c:\users\ahuja\anaconda3\lib\site-packages (from sphinx-rtd-theme->scholarly) (3.2.1)
          Requirement already satisfied: pyparsing>=2.0.3 in c:\users\ahuja\anaconda3\lib\site-packages (from bibtexparser->scholarly)
          (2.4.7)
          Requirement already satisfied: future>=0.16.0 in c:\users\ahuja\anaconda3\lib\site-packages (from bibtexparser->scholarly) (0.1
          8.2)


In [683]: from scholarly import scholarly

In [684]: professor_list = df_prof['Name'].tolist()
          professor_list

Out[684]: ['Lynn Batten',
           'Andrzej Goscinski',
           'Jemal Abawajy',
           'Maia Angelova',
           'Gleb Beliakov',
           'Terry Caelli',
           'Jinho Choi',
           'Chang-Tsun Li',
           'Robin Doss',
           'Peter Eklund',
           'Seng Loke',
           'Antonio Robles-Kelly',
           'Jean-Guy Schneider',
           'Yong Xiang',
           'John Yearwood',
           'Arkady Zaslavsky',
           'Mohamed Abdelrazek',
           'Andrew Cain',
           'Richard Dazeley',
           'Guangyan Huang',
           'Gang Li',
           'Jianxin Li',
           'Xiao Liu',
           'Vicky Mak',
           'Tim Wilkin']
```



```
In [685]: # Creating a search function to navigate google scholar profiles of all the professors and retrieve citation information
          # of professors, fill-in, and print (print "NA" for authors with no google scholar profile)
          author_list = []
          for name in professor_list:
              try:
                  search_query = scholarly.search_author(name +' Deakin University')
                  author = next(search_query)
                  scholarly.pprint(scholarly.fill(author, sections=['basics', 'indices']))
                  author_list.append(author)
              except StopIteration as e:
                  print("NA")
                  author_list.append("NA")
```

```
{'affiliation': 'Deakin University',
 'citedby': 2528,
 'citedby5y': 1172,
 'email_domain': '@deakin.edu.au',
 'filled': False,
 'hindex': 23,
 'hindex5y': 16,
 'i10index': 45,
 'i10index5y': 21,
 'interests': [],
 'name': 'Lynn Batten',
 'scholar_id': 'Tmg0T9sAAAAJ',
 'source': 'SEARCH_AUTHOR_SNIPPETS',
 'url_picture': 'https://scholar.google.com/citations?view_op=medium_photo&user=Tmg0T9sAAAAJ'}
NA
NA
{'affiliation': 'Professor of Data Analytics and Machine Learning Deakin '
                'University',
 'citedby': 682,
 'citedby5y': 399,
```

```
In [686]: # List containing author profile information
          author_list
```

```
Out[686]: [{'container_type': 'Author',
            'filled': {'basics', 'indices'},
            'source': <AuthorSource.SEARCH_AUTHOR_SNIPPETS: 2>,
            'scholar_id': 'Tmg0T9sAAAAJ',
            'url_picture': 'https://scholar.google.com/citations?view_op=medium_photo&user=Tmg0T9sAAAAJ',
            'name': 'Lynn Batten',
            'affiliation': 'Deakin University',
            'email_domain': '@deakin.edu.au',
            'interests': [],
            'citedby': 2528,
            'citedby5y': 1172,
            'hindex': 23,
```

From the above output, it is evident that there are total **four** 'NA' values in the author list of profile information implicating that the relevant authors do not have google scholar profiles and the indices of these 'NA' values are **1,2,8 and 23** which implicates that in the professor name list, professor names with same indices do not have google scholar profiles.

```
In [687]: df_no_google_scholar_prof = df_prof.iloc[np.r_[1,2,8,23]]
```

```
In [688]: df_no_google_scholar_prof
```

Out[688]:

	Name	Title	University
1	Andrzej Goscinski	Emeritus Professor	Deakin University
2	Jemal Abawajy	Professor	Deakin University
8	Robin Doss	Professor	Deakin University
23	Vicky Mak	Associate Professor	Deakin University

However, while doing a google scholar search of all the above deakin professors to cross-verify the above result, we found that Vicky Mak has a google scholar profile which the scholarly crawler was not able to find. So we will scrap the google scholar profile for Vicky Mak later on by using beautifulsoup.

```
In [689]: # filtering NA values from the author google scholar profile information list
          no_profile = 'NA'
          author_list = [profile for profile in author_list if profile != no_profile]
```

```
In [690]: # filtered list for authors with google scholar profiles
          author_list
```

```
Out[690]: [{'container_type': 'Author',
            'filled': {'basics', 'indices'},
            'source': <AuthorSource.SEARCH_AUTHOR_SNIPPETS: 2>,
            'scholar_id': 'Tmg0T9sAAAAJ',
            'url_picture': 'https://scholar.google.com/citations?view_op=medium_photo&user=Tmg0T9sAAAAJ',
            'name': 'Lynn Batten',
            'affiliation': 'Deakin University',
            'email_domain': '@deakin.edu.au',
            'interests': [],
            'citedby': 2528,
            'citedby5y': 1172,
            'hindex': 23,
            'hindex5y': 16,
```

In [691]:
```python
# creating dataframe of author citation information with the above filtered author list
df = pd.DataFrame(author_list, columns=['name', 'citedby', 'citedby5y', 'hindex', 'hindex5y', 'i10index', 'i10index5y'])
# Renaming columns as per requirement
df.columns = ['Name', 'citation-all', 'citation-since2016', 'h-index-all', 'h-index-since2016', 'i10-index-all', \
              'i10-index-since2016']
df
```

Out[691]:

	Name	citation-all	citation-since2016	h-index-all	h-index-since2016	i10-index-all	i10-index-since2016
0	Lynn Batten	2528	1172	23	16	45	21
1	Maia Angelova	682	399	15	11	21	14
2	Gleb Beliakov	8173	4632	42	32	135	86
3	Terry Caelli	8665	1477	51	20	177	36
4	Jinho Choi	7822	5031	37	29	168	96
5	Chang-Tsun Li	4344	2339	33	21	104	65
6	Peter Werner Eklund	3962	778	34	14	72	23
7	Seng W. Loke	7182	2907	38	21	126	50
8	Antonio Robles-Kelly	3655	1534	24	15	62	27
9	Jean-Guy Schneider	1812	646	24	15	48	21
10	Yong Xiang	6308	4971	40	35	118	95
11	John Yearwood	3793	1998	32	23	92	45
12	Arkady Zaslavsky	14939	8339	50	33	177	80
13	Mohamed Abdelrazek	1863	1263	16	15	27	23
14	Andrew Cain	291	194	10	9	11	8
15	Richard Dazeley	1472	950	16	13	24	15
16	Guangyan Huang	1130	811	20	18	29	24
17	Gang Li	4117	2826	28	24	89	61
18	Jianxin Li	1463	1084	21	18	39	32
19	Xiao Liu	3223	1753	26	20	52	42
20	Tim Wilkin	387	310	10	10	11	10

Let us now scrap Vicky Mak's google scholar profile which the scholarly crawler was not able to find. We will scrap the google scholar profile for Vicky Mak by using beautifulsoup to get the citation information.

In [692]:
```python
# Scraping google scholar profile information for Vicky Mak
url = 'https://scholar.google.com.au/citations?hl=en&user=KAiJydoAAAAJ'
response = requests.get(url)
soup = BeautifulSoup(response.content, 'lxml')
print(soup.prettify())
```

```
e:13px;cursor:pointer;height:29px;line-height:normal;min-width:72px;padding:0 8px;color:#444;border:1px solid rgba(0,0,0,.
1);border-radius:3px;text-align:center;background-color:#f5f5f5;user-select:none;}button.gs_btn_rnd{border-radius:14px;paddi
ng:0 12px;}button.gs_btn_rnd.gs_btn_rndci{padding-left:4px;}button.gs_btn_lrge{height:41px;min-width:82px;padding:0 9px;}but
ton.gs_btn_lrge.gs_btn_rnd{border-radius:20px;padding:0 16px;}button.gs_btn_lrge.gs_btn_rnd.gs_btn_rndci{padding-left:10px;}
button.gs_btn_cir{border-radius:14.5px;min-width:29px;}button.gs_btn_lrge.gs_btn_cir{border-radius:20.5px;min-width:41px;}bu
tton.gs_btn_mini{padding:0;border:0;}.gs_el_ph button.gs_btn_mph,.gs_el_ta button.gs_btn_mta{height:41px;}button .gs_wr{posi
tion:relative;display:inline-block;width:100%;height:100%;}button .gs_wr:before{content:"";width:0;height:100%;}button .gs_w
r:before,button .gs_ico,button .gs_rdt,button .gs_lbl,button .gs_icm{display:inline-block;vertical-align:middle;}button .gs_
wr{font-size:13px;text-transform:none;}.gs_btn_lrge .gs_wr{font-size:15px;}.gs_btn_lsb .gs_wr{font-size:11px;font-weight:bol
d;}.gs_btn_lsu .gs_wr{font-size:11px;text-transform:uppercase;}.gs_btn_lrge.gs_btn_lsb .gs_wr,.gs_btn_lrge.gs_btn_lsu .gs_wr
{font-size:13px;}.gs_btn_half,.gs_el_ta .gs_btn_hta,.gs_el_ph .gs_btn_hph{min-width:36px;}.gs_btn_lrge.gs_btn_half,.gs_el_ta
.gs_btn_lrge.gs_btn_hta,.gs_el_ph .gs_btn_lrge.gs_btn_hph,.gs_el_ta .gs_btn_mta,.gs_el_ph .gs_btn_mph{min-width:41px;}.gs_bt
n_slt{border-radius:3px 0 0 3px;}.gs_btn_srt{margin-left:-1px;border-radius:0 3px 3px 0;}.gs_btn_smd{margin-left:-1px;border
-radius:0;}button:hover{z-index:2;color:#222;border-color:rgba(0,0,0,.2);background-color:#f8f8f8;}button.gs_sel{background-
color:#dcdcdc;}button:active{z-index:2;background-color:#f1f1f1;}button:focus{z-index:2;}button::-moz-focus-inner{padding:0;
border:0}button:-moz-focusring{outline:1px dotted ButtonText}.gs_pfcs button:-moz-focusring{outline:none}a.gs_in_ib{positio
n:relative;display:inline-block;line-height:16px;padding:6px 0 7px 0;user-select:none;}a.gs_btn_lrge{height:40px;padding:0;}
a.gs_in_ib .gs_lbl{display:inline-block;padding-left:21px;color:#222;}a.gs_in_ib .gs_lbl:not(:empty){padding-left:29px;}butt
on.gs_in_ib .gs_lbl:not(:empty){padding-left:4px;}a.gs_in_ib:active .gs_lbl,a.gs_in_ib .gs_lbl:active,a.gs_in_ib :active~.gs
_lbl{color:#d14836;}.gs_el_ta .gs_btn_hta .gs_lbl,.gs_el_ph .gs_btn_hph .gs_lbl,.gs_el_ta .gs_btn_mta .gs_lbl,.gs_el_ph .gs_
```

In [693]:
```python
# Name of author
title = soup.find('title').contents[0].split('Google Scholar')[0][:-2].strip().split('-')[0]
title
```

Out[693]: 'Vicky Mak'

In [694]:
```python
# Extracting all contents within the 'table' tag or in other terms extracting all the tables
all_tables = soup.find_all('table')
print(all_tables)
```

```
[<table id="gsc_rsb_st"><thead><tr><th class="gsc_rsb_sth"></th><th class="gsc_rsb_sth">All</th><th class="gsc_rsb_sth">Sinc
e 2016</th></tr></thead><tbody><tr><td class="gsc_rsb_sc1"><a class="gsc_rsb_f gs_ibl" href="javascript:void(0)" title='This
is the number of citations to all publications. The second column has the "recent" version of this metric which is the numbe
r of new citations in the last 5 years to all publications.'>Citations</a></td><td class="gsc_rsb_std">385</td><td class="gs
c_rsb_std">227</td></tr><tr><td class="gsc_rsb_sc1"><a class="gsc_rsb_f gs_ibl" href="javascript:void(0)" title='h-index is
the largest number h such that h publications have at least h citations. The second column has the "recent" version of this
metric which is the largest number h such that h publications have at least h new citations in the last 5 years.'>h-index</a
```

In [695]:
```
# Getting the right table which holds all citation data of the author
right_table = soup.find('table', id="gsc_rsb_st")
print(right_table)
```

```
<table id="gsc_rsb_st"><thead><tr><th class="gsc_rsb_sth"></th><th class="gsc_rsb_sth">All</th><th class="gsc_rsb_sth">Since 20
16</th></tr></thead><tbody><tr><td class="gsc_rsb_sc1"><a class="gsc_rsb_f gs_ibl" href="javascript:void(0)" title='This is the
number of citations to all publications. The second column has the "recent" version of this metric which is the number of new c
itations in the last 5 years to all publications.'>Citations</a></td><td class="gsc_rsb_std">385</td><td class="gsc_rsb_std">22
7</td></tr><tr><td class="gsc_rsb_sc1"><a class="gsc_rsb_f gs_ibl" href="javascript:void(0)" title='h-index is the largest numb
er h such that h publications have at least h citations. The second column has the "recent" version of this metric which is the
largest number h such that h publications have at least h new citations in the last 5 years.'>h-index</a></td><td class="gsc_rs
b_std">11</td><td class="gsc_rsb_std">8</td></tr><tr><td class="gsc_rsb_sc1"><a class="gsc_rsb_f gs_ibl" href="javascript:void
(0)" title='i10-index is the number of publications with at least 10 citations. The second column has the "recent" version of t
his metric which is the number of publications that have received at least 10 new citations in the last 5 years.'>i10-index</a>
</td><td class="gsc_rsb_std">14</td><td class="gsc_rsb_std">6</td></tr></tbody></table>
```

In [696]:
```
# Loading Vicky Mak's citation data into df
Name = []
Citations = []
All = []
Since_2016 = []

for row in right_table.findAll("tr"):
    cells = row.findAll('td')
    if len(cells) > 0 : # and len(cells) < 10:
        Citations.append(cells[0].find(text=True))
        All.append(cells[1].find(text=True))
        Since_2016.append(cells[2].find(text=True))
        Name.append(title)
        dict_vicky = dict({'Name':Name, 'Citations':Citations, 'All':All, 'Since_2016':Since_2016})

df_vicky =pd.DataFrame(dict_vicky)
```

In [697]:
```
# Vicky Mak Citation Data
df_vicky
```

Out[697]:

	Name	Citations	All	Since_2016
0	Vicky Mak	Citations	385	227
1	Vicky Mak	h-index	11	8
2	Vicky Mak	i10-index	14	6

In [698]:
```
# pivoting dataframe
df_vicky_trans = df_vicky.pivot(index='Name', columns='Citations', values=['All', 'Since_2016'])
df_vicky_trans
```

Out[698]:

	All			Since_2016		
Citations	Citations	h-index	i10-index	Citations	h-index	i10-index
Name						
Vicky Mak	385	11	14	227	8	6

In [699]:
```
df_vicky_final = df_vicky_trans.reset_index()
# Renaming columns as per requirement
df_vicky_final.columns = ['Name', 'citation-all', 'h-index-all', 'i10-index-all', 'citation-since2016', 'h-index-since2016', \
                'i10-index-since2016']
df_vicky_final
```

Out[699]:

	Name	citation-all	h-index-all	i10-index-all	citation-since2016	h-index-since2016	i10-index-since2016
0	Vicky Mak	385	11	14	227	8	6

In [700]:
```
# Previously identified Professors with no google scholar profiles
df_no_google_scholar_prof
```

Out[700]:

	Name	Title	University
1	Andrzej Goscinski	Emeritus Professor	Deakin University
2	Jemal Abawajy	Professor	Deakin University
8	Robin Doss	Professor	Deakin University
23	Vicky Mak	Associate Professor	Deakin University

In [701]:
```
# Actual Professors with no google scholar profiles
df_no_google_scholar_prof.drop(index=23, axis=0, inplace=True)
df_no_google_scholar_prof
```

Out[701]:

	Name	Title	University
1	Andrzej Goscinski	Emeritus Professor	Deakin University
2	Jemal Abawajy	Professor	Deakin University
8	Robin Doss	Professor	Deakin University

Appending the two dataframes i.e. one having all the correctly identified Deakin School of Information Technology Professors citation information except Vicky Mak and other having citation information for Associate Professor Vicky Mak.

```
In [702]: # citation dataframe
          df = df.append(df_vicky_final)
          df = df.reset_index(drop=True)
          df
```

Out[702]:

	Name	citation-all	citation-since2016	h-index-all	h-index-since2016	i10-index-all	i10-index-since2016
0	Lynn Batten	2528	1172	23	16	45	21
1	Maia Angelova	682	399	15	11	21	14
2	Gleb Beliakov	8173	4632	42	32	135	86
3	Terry Caelli	8665	1477	51	20	177	36
4	Jinho Choi	7822	5031	37	29	168	96
5	Chang-Tsun Li	4344	2339	33	21	104	65
6	Peter Werner Eklund	3962	778	34	14	72	23
7	Seng W. Loke	7182	2907	38	21	126	50
8	Antonio Robles-Kelly	3655	1534	24	15	62	27
9	Jean-Guy Schneider	1812	646	24	15	48	21
10	Yong Xiang	6308	4971	40	35	118	95
11	John Yearwood	3793	1998	32	23	92	45
12	Arkady Zaslavsky	14939	8339	50	33	177	80
13	Mohamed Abdelrazek	1863	1263	16	15	27	23
14	Andrew Cain	291	194	10	9	11	8
15	Richard Dazeley	1472	950	16	13	24	15
16	Guangyan Huang	1130	811	20	18	29	24
17	Gang Li	4117	2826	28	24	89	61
18	Jianxin Li	1463	1084	21	18	39	32
19	Xiao Liu	3223	1753	26	20	52	42
20	Tim Wilkin	387	310	10	10	11	10
21	Vicky Mak	385	227	11	8	14	6

```
In [703]: # Fixing some name incosistencies so as to merge with df_prof dataframe on name column
          df['Name'].replace('Peter Werner Eklund', 'Peter Eklund', inplace=True)
          df['Name'].replace('Seng W. Loke', 'Seng Loke', inplace=True)
```

```
In [705]: # merging df of professors with google scholar profiles with other df of profs with no google scholar profiles
          df_prof_cit = df_prof.merge(df, how='left', on='Name')
          # dropping University column as per requirement
          df_prof_cit.drop('University', axis=1, inplace=True)
          # FINAL Professor Citation Dataframe
          df_prof_cit
```

Out[705]:

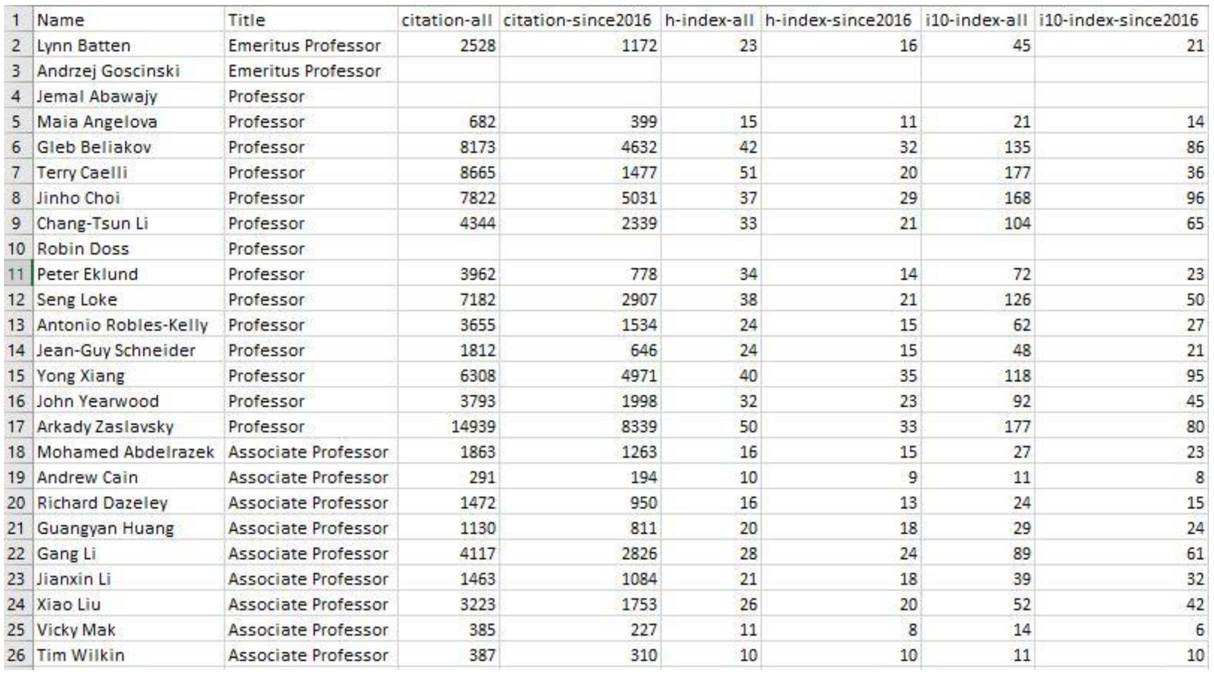
	Name	Title	citation-all	citation-since2016	h-index-all	h-index-since2016	i10-index-all	i10-index-since2016
0	Lynn Batten	Emeritus Professor	2528	1172	23	16	45	21
1	Andrzej Goscinski	Emeritus Professor	NaN	NaN	NaN	NaN	NaN	NaN
2	Jemal Abawajy	Professor	NaN	NaN	NaN	NaN	NaN	NaN
3	Maia Angelova	Professor	682	399	15	11	21	14
4	Gleb Beliakov	Professor	8173	4632	42	32	135	86
5	Terry Caelli	Professor	8665	1477	51	20	177	36
6	Jinho Choi	Professor	7822	5031	37	29	168	96
7	Chang-Tsun Li	Professor	4344	2339	33	21	104	65
8	Robin Doss	Professor	NaN	NaN	NaN	NaN	NaN	NaN
9	Peter Eklund	Professor	3962	778	34	14	72	23
10	Seng Loke	Professor	7182	2907	38	21	126	50
11	Antonio Robles-Kelly	Professor	3655	1534	24	15	62	27
12	Jean-Guy Schneider	Professor	1812	646	24	15	48	21
13	Yong Xiang	Professor	6308	4971	40	35	118	95
14	John Yearwood	Professor	3793	1998	32	23	92	45
15	Arkady Zaslavsky	Professor	14939	8339	50	33	177	80
16	Mohamed Abdelrazek	Associate Professor	1863	1263	16	15	27	23
17	Andrew Cain	Associate Professor	291	194	10	9	11	8
18	Richard Dazeley	Associate Professor	1472	950	16	13	24	15
19	Guangyan Huang	Associate Professor	1130	811	20	18	29	24
20	Gang Li	Associate Professor	4117	2826	28	24	89	61
21	Jianxin Li	Associate Professor	1463	1084	21	18	39	32
22	Xiao Liu	Associate Professor	3223	1753	26	20	52	42
23	Vicky Mak	Associate Professor	385	227	11	8	14	6
24	Tim Wilkin	Associate Professor	387	310	10	10	11	10

Saving the professor citation information dataframe to csv format under the name professor-citation-information.

```
In [706]: df_prof_cit.to_csv('Professor-citation-information.csv', index=False)
```

4.1.1. Professor Citation Information CSV

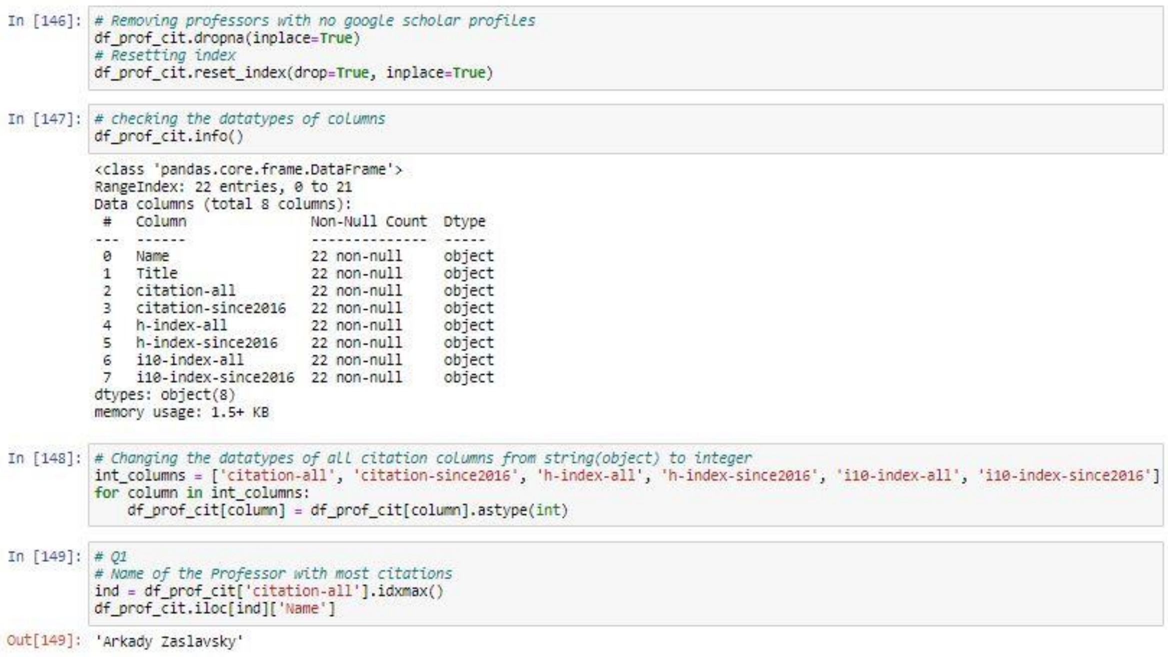
Please fill this part with the screenshot of the professor citation information CSV.



1	Name	Title	citation-all	citation-since2016	h-index-all	h-index-since2016	i10-index-all	i10-index-since2016
2	Lynn Batten	Emeritus Professor	2528	1172	23	16	45	21
3	Andrzej Goscinski	Emeritus Professor						
4	Jemal Abawajy	Professor						
5	Maia Angelova	Professor	682	399	15	11	21	14
6	Gleb Beliakov	Professor	8173	4632	42	32	135	86
7	Terry Caelli	Professor	8665	1477	51	20	177	36
8	Jinho Choi	Professor	7822	5031	37	29	168	96
9	Chang-Tsun Li	Professor	4344	2339	33	21	104	65
10	Robin Doss	Professor						
11	Peter Eklund	Professor	3962	778	34	14	72	23
12	Seng Loke	Professor	7182	2907	38	21	126	50
13	Antonio Robles-Kelly	Professor	3655	1534	24	15	62	27
14	Jean-Guy Schneider	Professor	1812	646	24	15	48	21
15	Yong Xiang	Professor	6308	4971	40	35	118	95
16	John Yearwood	Professor	3793	1998	32	23	92	45
17	Arkady Zaslavsky	Professor	14939	8339	50	33	177	80
18	Mohamed Abdelrazek	Associate Professor	1863	1263	16	15	27	23
19	Andrew Cain	Associate Professor	291	194	10	9	11	8
20	Richard Dazeley	Associate Professor	1472	950	16	13	24	15
21	Guangyan Huang	Associate Professor	1130	811	20	18	29	24
22	Gang Li	Associate Professor	4117	2826	28	24	89	61
23	Jianxin Li	Associate Professor	1463	1084	21	18	39	32
24	Xiao Liu	Associate Professor	3223	1753	26	20	52	42
25	Vicky Mak	Associate Professor	385	227	11	8	14	6
26	Tim Wilkin	Associate Professor	387	310	10	10	11	10

4.2. Find the Professor with most citations

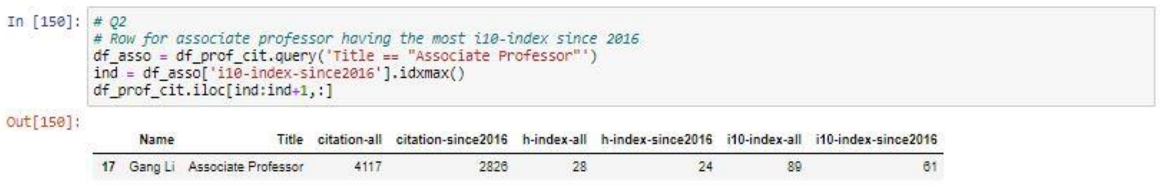
Please fill this part with the screenshot of the code (include the results of the code running).



```
In [146]: # Removing professors with no google scholar profiles
          df_prof_cit.dropna(inplace=True)
          # Resetting index
          df_prof_cit.reset_index(drop=True, inplace=True)

In [147]: # checking the datatypes of columns
          df_prof_cit.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 22 entries, 0 to 21
          Data columns (total 8 columns):
           #   Column              Non-Null Count  Dtype
          ---  ------              --------------  -----
           0   Name                22 non-null     object
           1   Title               22 non-null     object
           2   citation-all        22 non-null     object
           3   citation-since2016  22 non-null     object
           4   h-index-all         22 non-null     object
           5   h-index-since2016   22 non-null     object
           6   i10-index-all       22 non-null     object
           7   i10-index-since2016 22 non-null     object
          dtypes: object(8)
          memory usage: 1.5+ KB

In [148]: # Changing the datatypes of all citation columns from string(object) to integer
          int_columns = ['citation-all', 'citation-since2016', 'h-index-all', 'h-index-since2016', 'i10-index-all', 'i10-index-since2016']
          for column in int_columns:
              df_prof_cit[column] = df_prof_cit[column].astype(int)

In [149]: # Q1
          # Name of the Professor with most citations
          ind = df_prof_cit['citation-all'].idxmax()
          df_prof_cit.iloc[ind]['Name']

Out[149]: 'Arkady Zaslavsky'
```

4.3. Find the Associate Professor with most i10-index since 2016

Please fill this part with the screenshot of the code (include the results of the code running).

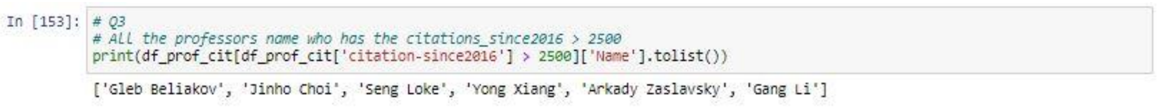


```
In [150]: # Q2
          # Row for associate professor having the most i10-index since 2016
          df_asso = df_prof_cit.query('Title == "Associate Professor"')
          ind = df_asso['i10-index-since2016'].idxmax()
          df_prof_cit.iloc[ind:ind+1,:]

Out[150]:
```

	Name	Title	citation-all	citation-since2016	h-index-all	h-index-since2016	i10-index-all	i10-index-since2016
17	Gang Li	Associate Professor	4117	2826	28	24	89	61

4.4. Find all Professors name who have the citations since 2016 > 2500

Please fill this part with the screenshot of the code (include the results of the code running).



```
In [153]: # Q3
          # All the professors name who has the citations_since2016 > 2500
          print(df_prof_cit[df_prof_cit['citation-since2016'] > 2500]['Name'].tolist())

          ['Gleb Beliakov', 'Jinho Choi', 'Seng Loke', 'Yong Xiang', 'Arkady Zaslavsky', 'Gang Li']
```