

Code Bozu Fellowship Cohort 1: Final Report  
Zinnun Malikov  
January 15, 2022

### **Fellowship Prompt Part 1.0 - Deliverable 0: Ethics of Web Scraping**

In this deliverable, each individual member was tasked with learning about the ethical and legal considerations of web scraping. Further, all CodeBozu fellows, including myself, learned how crucial it is to maintain appropriate practices. A copy of my submission is below:

#### *Legal and Ethical Considerations of Data Scraping*

Web scraping is a technique for extracting information from websites. It can be done manually but it is usually faster, more efficient, and less error-prone to automate the task. While web scraping is increasingly useful in the fields of academics, research, and journalism, this powerful tool must be used with great responsibility. After examining the multiple methods of scraping data and starting a large-scale project, it is important to consider the legal and ethical aspects of data scraping since it can potentially be used for malicious purposes.

For example, web scraping can be used to disrupt a website. A web scraper queries a website repeatedly and accesses a potentially large number of pages. For each page, a request will be sent to the webserver that is hosting the site, and the server will have to process the request and send a response back to the computer that is running the code. Each request will consume resources on the server, during which it will not be doing something else. Unfortunately, sending too many requests to a server can prevent others from accessing the website during a specific period. This is known as a Denial of Service (DoS) attack. While modern web servers include measures to ward off such illegitimate use of their resources, it is still crucial to be careful. If a web scraper exhibits similar behavior, the computer running the code can be banned from accessing a website.

Furthermore, it is necessary to understand when data scraping is illegal. This is especially true if the terms and conditions of the website being scraped specifically prohibit downloading and copying its content. However, in practice, web scraping is tolerated, provided it does not cause DoS attacks or disrupt the website in other ways. Generally, if data is available to the public and is not behind a password-protected system, it is alright to scrape it without causing disruptions. However, it is important to recall copyright protections on certain materials and remain cautious when downloading content off one website and posting it elsewhere. For the most part, however, it is permissible to use some parts of copyrighted materials. Nevertheless, it is better to be safe than sorry. Copyright and data protection laws vary in different countries. For instance, it is illegal to scrape and store personal information such as names, phone numbers, and email addresses, in Australia.

Overall, while data scraping is an essential tool for data analytics, it is important to maintain ethical and legal practices. One must stay cautious when writing code to send requests to servers and handling potentially copyrighted information. Being polite and reaching out to owners of data can additionally prevent legal consequences and wasted time. Finally, adhering to the web scraping code of conduct can help keep data analytics an engaging and secure space for everyone.

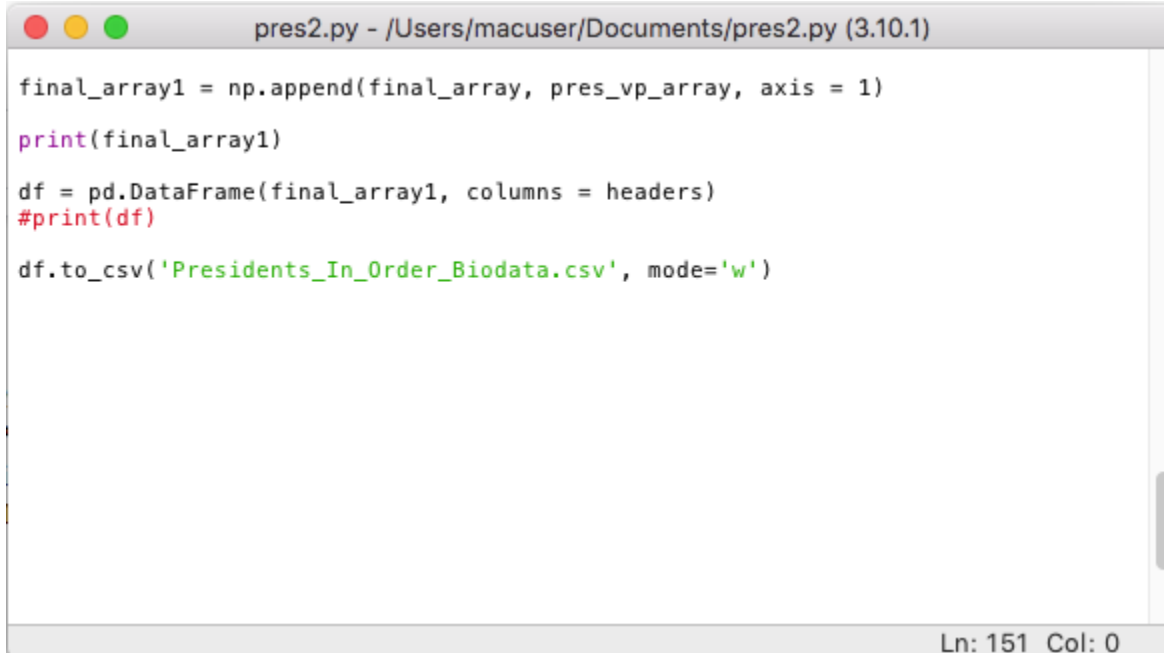
**Fellowship Prompt Part 1.1/1.2 - Deliverable 1/Deliverable 2: Gathering Biodata and Insights**

Deliverable 1 marked the introduction of Python code use for this fellowship. I was tasked with finding information about all United States Presidents through scraping Wikipedia articles. A copy of the report of my findings is attached on the next page.

### Introduction

For this assignment, insights will be taken from only the Presidents of the United States since there is not as much readily-available information about the Vice Presidents of the United States. The following details will be analyzed: state of birth, birthdate, age of inauguration, previous occupation(s), and political party. Note that the NUMPY array titled “final\_array1” will contain all the information scraped from the wikipedia pages. The list of column titles is: ['President (P)', 'State of Birth (P)', 'Term (P)', 'Birthdate (P)', 'Age of Inauguration (P)', 'Previous Occupation (P)', 'Party (P)', 'Vice President'].

Observe the code below.



```
final_array1 = np.append(final_array, pres_vp_array, axis = 1)
print(final_array1)
df = pd.DataFrame(final_array1, columns = headers)
#print(df)
df.to_csv('Presidents_In_Order_Biodata.csv', mode='w')
```

Ln: 151 Col: 0

Google Drive Link to CSV File (Deliverable 1):

[https://docs.google.com/spreadsheets/d/1AHpE-ZfjzNu0L1Q5JsEqMs7mwemhU9eCLRAPtTxr\\_A/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1AHpE-ZfjzNu0L1Q5JsEqMs7mwemhU9eCLRAPtTxr_A/edit?usp=sharing)

NUMPY Array Output:

```

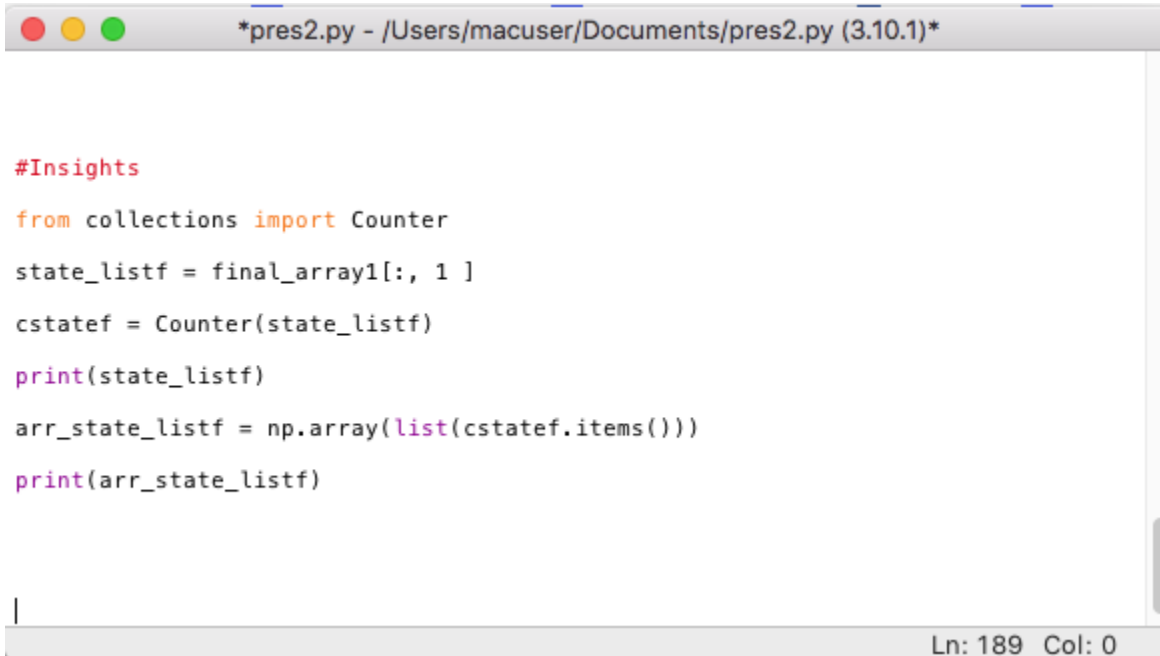
IDLE Shell 3.10.1
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec 6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/macuser/Documents/pres2.py =====
[['Washington, George' 'Virginia' '1789-1797' '2/22/1732' '57'
  'Planter, land surveyor' 'No Party Designation' 'John Adams']]
[['Adams, John' 'Massachusetts' '1797-1801' '10/30/1735' '61'
  'Lawyer, farmer' 'Federalist' 'Thomas Jefferson']]
[['Jefferson, Thomas' 'Virginia' '1801-1809' '4/13/1743' '57'
  'Planter, lawyer, land surveyor, architect' 'Democratic-Republican'
  'Aaron Burr, George Clinton']]
[['Madison, James' 'Virginia' '1809-1817' '3/16/1751' '57' 'Planter'
  'Democratic-Republican' 'George Clinton, Elbridge Gerry']]
[['Monroe, James' 'Virginia' '1817-1825' '4/28/1758' '58'
  'Planter, lawyer' 'Democratic-Republican' 'Daniel D. Tompkins']]
[['Adams, John Quincy' 'Massachusetts' '1825-1829' '7/11/1767' '57'
  'Lawyer' 'Democratic-Republican' 'John C. Calhoun']]
[['Jackson, Andrew' 'South Carolina' '1829-1837' '3/15/1767' '61'
  'Lawyer, military officer' 'Democratic'
  'John C. Calhoun, Martin Van Buren']]
[['Van Buren, Martin' 'New York' '1837-1841' '12/5/1782' '54' 'Lawyer'
  'Democratic' 'Richard M. Johnson']]
[['Harrison, William Henry' 'Virginia' '1841-1841' '2/9/1773' '68'
  'Military' 'Whig' 'John Tyler']]
[['Tyler, John' 'Virginia' '1841-1845' '3/29/1790' '51' 'Lawyer' 'Whig'
  'None']]
[['Polk, James Knox' 'North Carolina' '1845-1849' '11/2/1795' '49'
  'Lawyer, planter' 'Democratic' 'George M. Dallas']]
[['Taylor, Zachary' 'Virginia' '1849-1850' '11/24/1784' '64' 'Military'
  'Whig' 'Millard Fillmore']]
[['Fillmore, Millard' 'New York' '1850-1853' '1/7/1800' '50' 'Lawyer'
  'Whig' 'None']]
[['Pierce, Franklin' 'New Hampshire' '1853-1857' '11/23/1804' '48'
  'Lawyer' 'Democratic' 'William R. King']]
[['Buchanan, James' 'Pennsylvania' '1857-1861' '4/23/1791' '65' 'Lawyer'
  'Democratic' 'John C. Breckinridge']]
[['Lincoln, Abraham' 'Kentucky' '1861-1865' '2/12/1809' '52'
  'Lawyer, land surveyor' 'Union' 'Hannibal Hamlin, Andrew Johnson']]
[['Johnson, Andrew' 'North Carolina' '1865-1869' '12/29/1808' '56'

```

### Insights about State of Birth

The column at index 1 of the NUMPY array 'final\_array1' contains the list of the state of birth of each United States President. After indexing this column, the "Counter" class from the "collections" library is used to sort the frequency of the birth states. The code and the output, which is converted to the NUMPY array "arr\_state\_listf", are shown below.

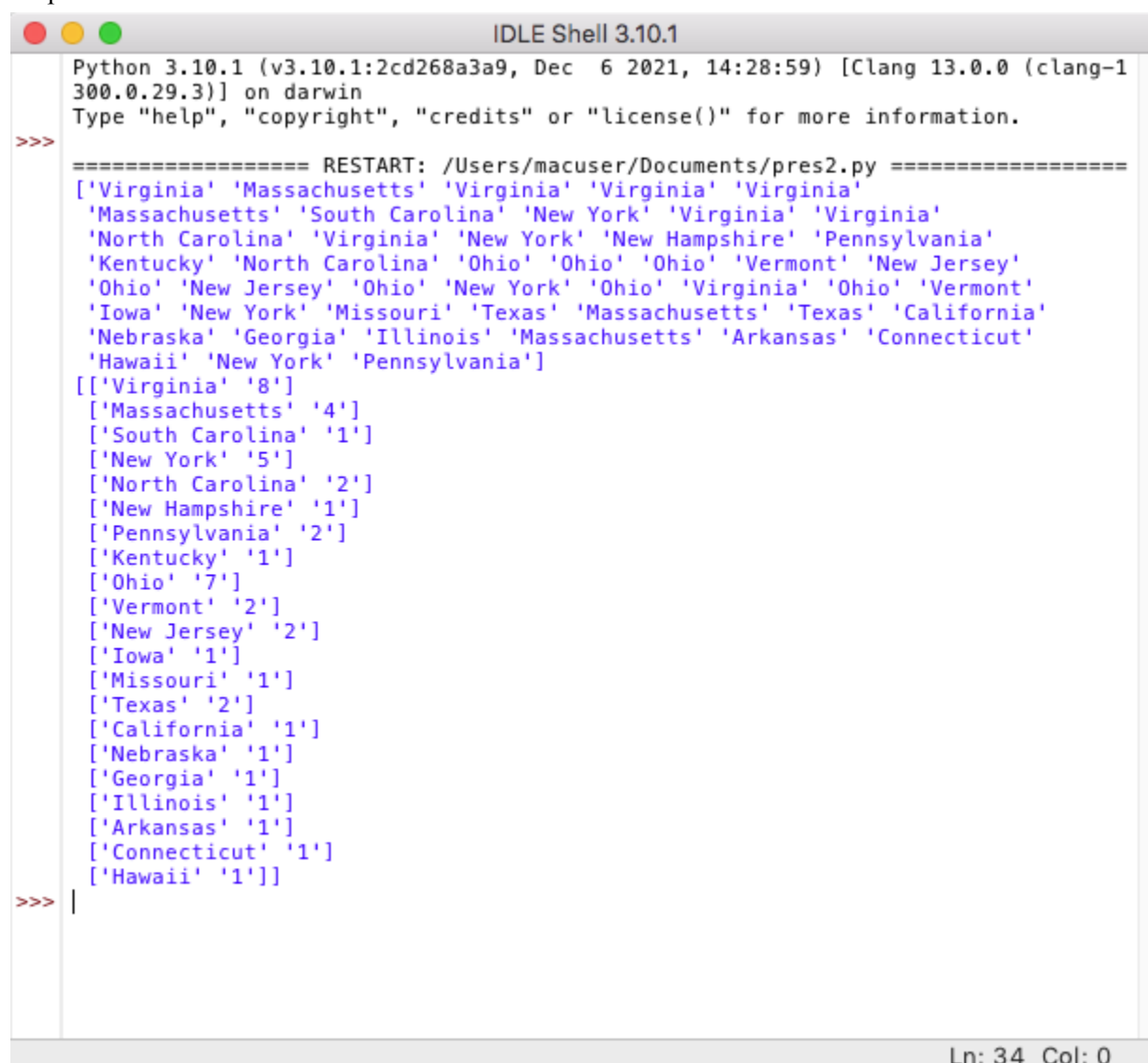
Code:



```
#Insights
from collections import Counter
state_listf = final_array1[:, 1 ]
cstatef = Counter(state_listf)
print(state_listf)
arr_state_listf = np.array(list(cstatef.items()))
print(arr_state_listf)
```

Ln: 189 Col: 0

Output:



```

Python 3.10.1 (v3.10.1:2cd268a3a9, Dec 6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/macuser/Documents/pres2.py =====
['Virginia' 'Massachusetts' 'Virginia' 'Virginia' 'Virginia'
'Massachusetts' 'South Carolina' 'New York' 'Virginia' 'Virginia'
'North Carolina' 'Virginia' 'New York' 'New Hampshire' 'Pennsylvania'
'Kentucky' 'North Carolina' 'Ohio' 'Ohio' 'Ohio' 'Vermont' 'New Jersey'
'Ohio' 'New Jersey' 'Ohio' 'New York' 'Ohio' 'Virginia' 'Ohio' 'Vermont'
'Iowa' 'New York' 'Missouri' 'Texas' 'Massachusetts' 'Texas' 'California'
'Nebraska' 'Georgia' 'Illinois' 'Massachusetts' 'Arkansas' 'Connecticut'
'Hawaii' 'New York' 'Pennsylvania']
[['Virginia' '8']
 ['Massachusetts' '4']
 ['South Carolina' '1']
 ['New York' '5']
 ['North Carolina' '2']
 ['New Hampshire' '1']
 ['Pennsylvania' '2']
 ['Kentucky' '1']
 ['Ohio' '7']
 ['Vermont' '2']
 ['New Jersey' '2']
 ['Iowa' '1']
 ['Missouri' '1']
 ['Texas' '2']
 ['California' '1']
 ['Nebraska' '1']
 ['Georgia' '1']
 ['Illinois' '1']
 ['Arkansas' '1']
 ['Connecticut' '1']
 ['Hawaii' '1']]
>>> |

```

Ln: 34 Col: 0

**Insight 1:** From this information, it can be concluded that the most frequent states of birth for United States Presidents are Virginia, Ohio, New York, and Massachusetts in descending order. Furthermore, the vast majority of these presidents are from the East coast. However, a potential bias exists; the states on the East Coast were part of the United States long before any territory was owned by the US in the West.

### Insights about Birthdate

Specifically, the month in which each president was born will be analyzed for this insight. The column at index 3 of the NUMPY array 'final\_array1' contains the list of the birthdate of each United States President in the form M(M)/DD/YYYY. Since the format is not identical (not all items have two characters for the month), it will be necessary to clean the birthdates. Afterward, the numbers 1-12 will be assigned to each month of the year.

Code:

```

pres2.py - /Users/macuser/Documents/pres2.py (3.10.1)
#Insights

from collections import Counter

bd_listf = final_array1[:, 3 ]

for z in range(46):
    item = bd_listf[z]
    item = str(item)
    abc = item[:2]
    if abc[1] == '/':
        abc = abc[0]
    elif abc[0] == '0':
        abc = abc[1]

    bd_listf[z] = abc

print(bd_listf)
bdf = Counter(bd_listf)
arr_bd_listf = np.array(list(bdf.items()))
print(arr_bd_listf)

```

Ln: 190 Col: 1

Output:

```

IDLE Shell 3.10.1
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec 6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/macuser/Documents/pres2.py =====
[['2' '10' '4' '3' '4' '7' '3' '12' '2' '3' '11' '11' '1' '11' '4' '2' '12'
  '4' '10' '11' '10' '3' '8' '3' '1' '10' '9' '12' '11' '7' '8' '1' '5'
  '10' '5' '8' '1' '7' '10' '2' '6' '8' '7' '8' '6' '11']
[['2' '4']
 ['10' '6']
 ['4' '4']
 ['3' '5']
 ['7' '4']
 ['12' '3']
 ['11' '6']
 ['1' '4']
 ['8' '5']
 ['9' '1']
 ['5' '2']
 ['6' '2']]
>>>

```



Note that the column at index 0 of the NUMPY array “arr\_bd\_listf” represents the month (‘1’ = January, ... ‘12’ = December).

**Insight 2:** *The frequencies of birth months are shown below in descending order:*

*October/November: 6*

*March/August: 5*

*January/February/April/July: 4*

*December: 3*

*May/June: 2*

*September: 1*

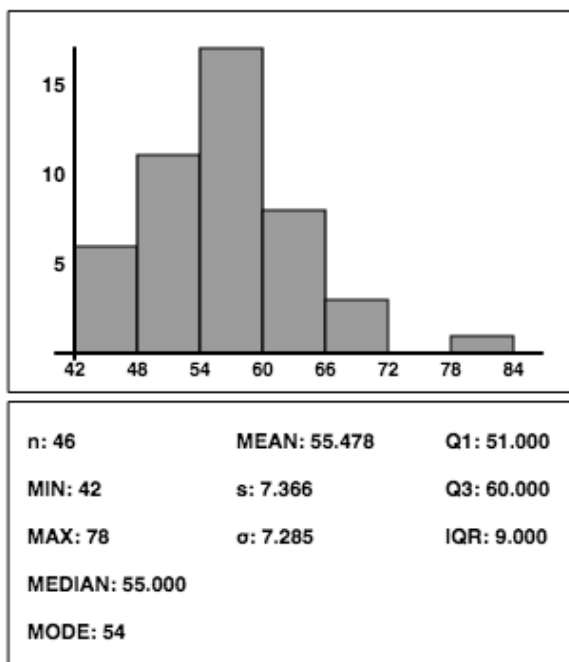
*Double check:  $2*6+2*5+4*4+3+2*2+1 = 12+10+16+3+4+1=46$*

### Insights about Age of Inauguration

Specifically, the age of inauguration of each president will be analyzed for this insight. The column at index 4 of the NUMPY array ‘final\_array1’ contains the list of the age of inauguration of each United States President. I have previously made a statistics calculator with processing.js, so it will only be necessary to extract the list of numbers. Note that Grover Cleveland served two non-consecutive terms, making him the 22<sup>nd</sup> and 24<sup>th</sup> president of the United States.

```
===== RESTART: /Users/macuser/Documents/pres2.py =====
['57' '61' '57' '57' '58' '57' '61' '54' '68' '51' '49' '64' '50' '48'
 '65' '52' '56' '46' '54' '49' '50' '47' '55' '55' '54' '42' '51' '56'
 '55' '51' '54' '51' '60' '62' '43' '55' '56' '61' '52' '69' '64' '46'
 '54' '47' '70' '78']
>>> |
```

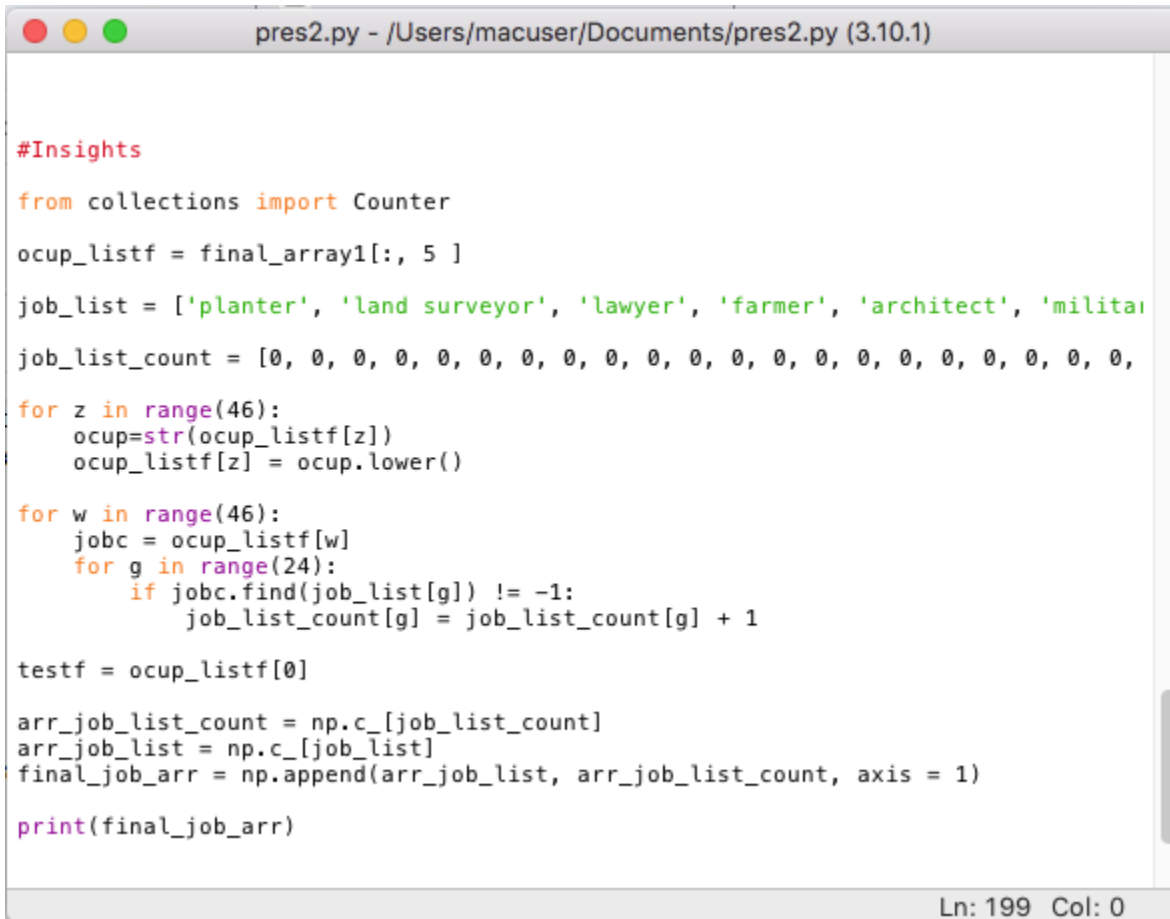
**Insight 3:** *Histogram of the Ages of Inauguration of the United States Presidents (With Statistics)*



### Insights about Previous Occupations

The column at index 5 of the NUMPY array 'final\_array1' contains the list of the previous occupation(s) of each United States President. Note that from the web scraping process, a list of 24 different occupations was created for the United States Presidents. The frequency of each job will be determined using the string method "find".

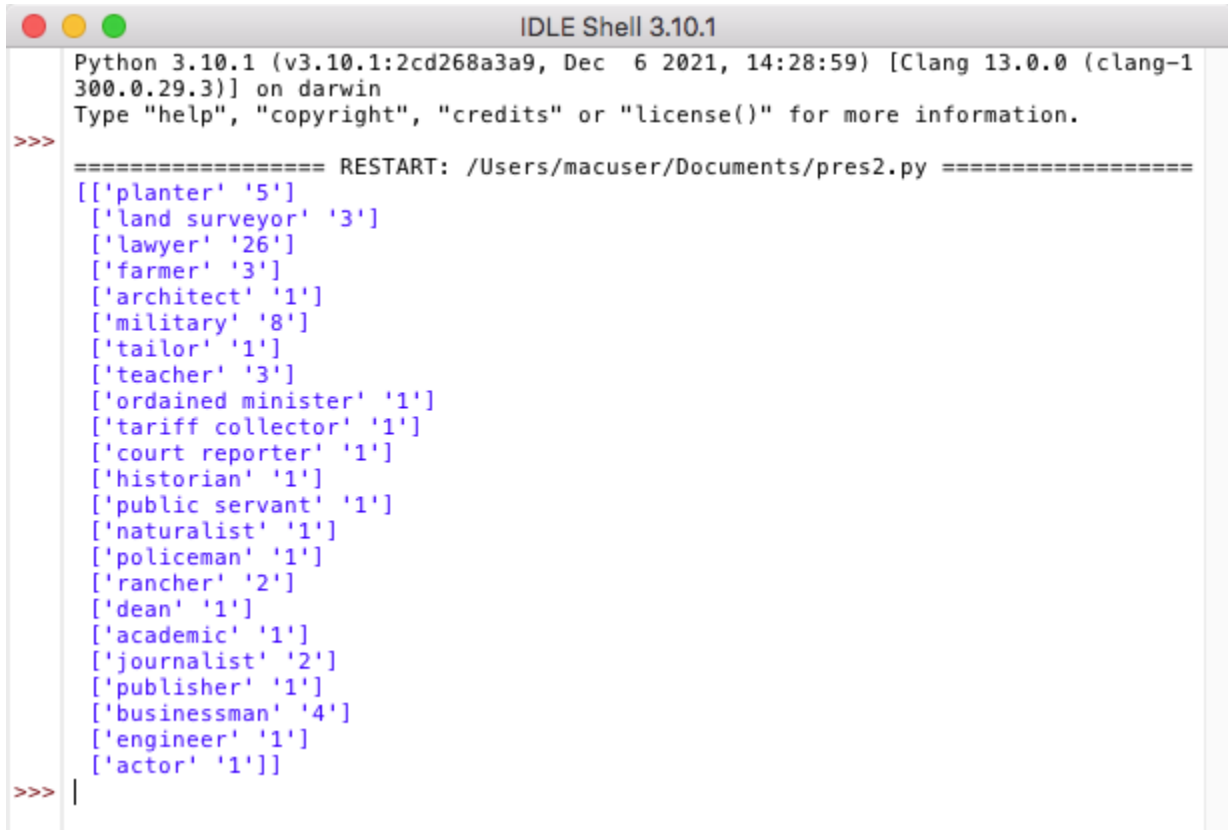
Code:



```
#Insights
from collections import Counter
ocup_listf = final_array1[:, 5 ]
job_list = ['planter', 'land surveyor', 'lawyer', 'farmer', 'architect', 'milita
job_list_count = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
for z in range(46):
    ocup=str(ocup_listf[z])
    ocup_listf[z] = ocup.lower()
for w in range(46):
    jobc = ocup_listf[w]
    for g in range(24):
        if jobc.find(job_list[g]) != -1:
            job_list_count[g] = job_list_count[g] + 1
testf = ocup_listf[0]
arr_job_list_count = np.c_[job_list_count]
arr_job_list = np.c_[job_list]
final_job_arr = np.append(arr_job_list, arr_job_list_count, axis = 1)
print(final_job_arr)
```

Ln: 199 Col: 0

Output:



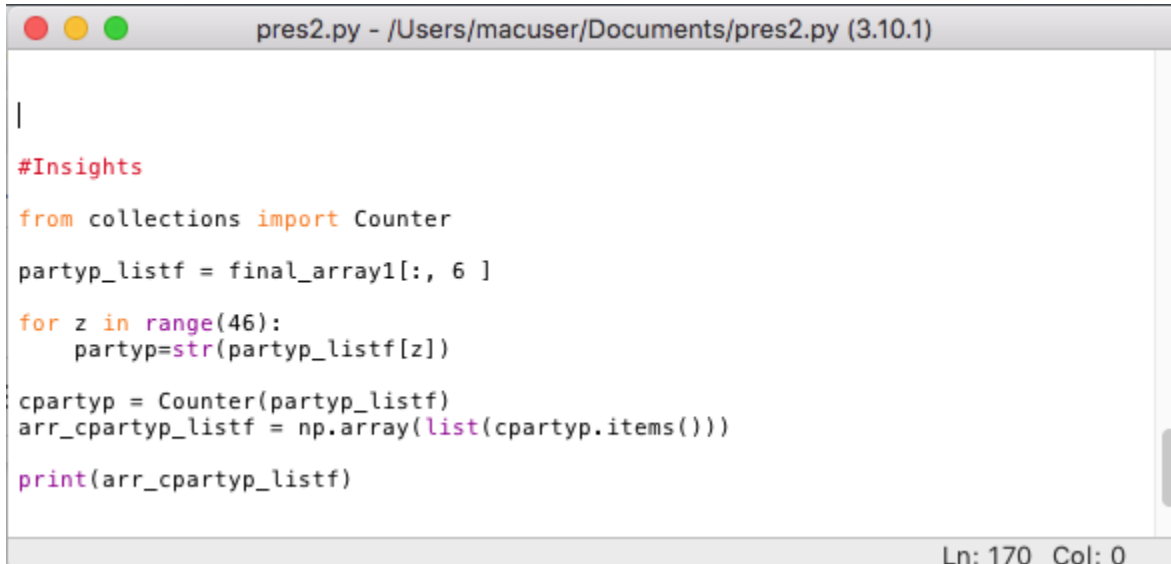
```
Python 3.10.1 (v3.10.1:2cd268a3a9, Dec 6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/macuser/Documents/pres2.py =====
[['planter' '5']
 ['land surveyor' '3']
 ['lawyer' '26']
 ['farmer' '3']
 ['architect' '1']
 ['military' '8']
 ['tailor' '1']
 ['teacher' '3']
 ['ordained minister' '1']
 ['tariff collector' '1']
 ['court reporter' '1']
 ['historian' '1']
 ['public servant' '1']
 ['naturalist' '1']
 ['policeman' '1']
 ['rancher' '2']
 ['dean' '1']
 ['academic' '1']
 ['journalist' '2']
 ['publisher' '1']
 ['businessman' '4']
 ['engineer' '1']
 ['actor' '1']]
>>> |
```

**Insight 4:** *The most frequent previous occupation United States Presidents held was being a lawyer.*

### Insights about Political Party

The column at index 6 of the NUMPY array 'final\_array1' contains the list of the political party of each United States President.

Code:

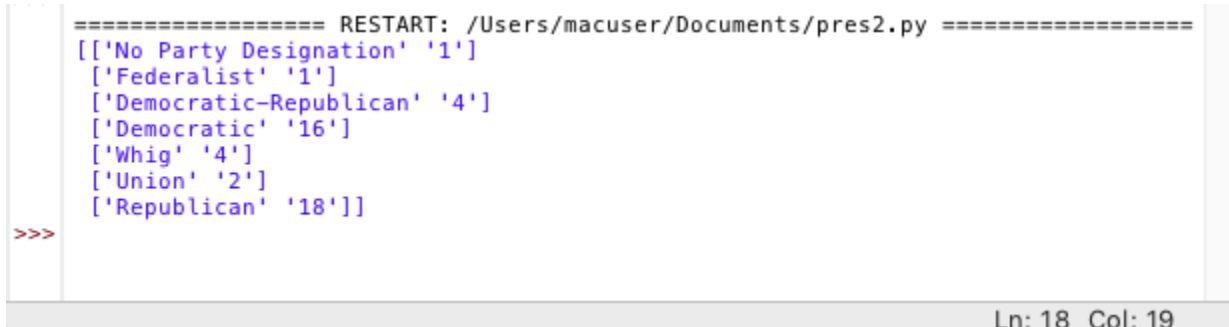


```

|
#Insights
from collections import Counter
partyp_listf = final_array1[:, 6 ]
for z in range(46):
    partyp=str(partyp_listf[z])
cpartyp = Counter(partyp_listf)
arr_cpartyp_listf = np.array(list(cpartyp.items()))
print(arr_cpartyp_listf)
Ln: 170 Col: 0

```

Output:



```

===== RESTART: /Users/macuser/Documents/pres2.py =====
[['No Party Designation' '1']
 ['Federalist' '1']
 ['Democratic-Republican' '4']
 ['Democratic' '16']
 ['Whig' '4']
 ['Union' '2']
 ['Republican' '18']]
>>>
Ln: 18 Col: 19

```

**Insight 5:** *The most frequent political parties United States Presidents were affiliated with were “Democratic” and “Republican”. Note that the first couple of presidents were affiliated with the other parties. Furthermore, George Washington remains the only United States President to not be affiliated with a political party.*

### Conclusion (Recap of Insights)

**Insight 1:** From this information, it can be concluded that the most frequent states of birth for United States Presidents are Virginia, Ohio, New York, and Massachusetts in descending order.

**Insight 2:** The frequencies of birth months are shown below in descending order:

October/November: 6

March/August: 5

January/February/April/July: 4

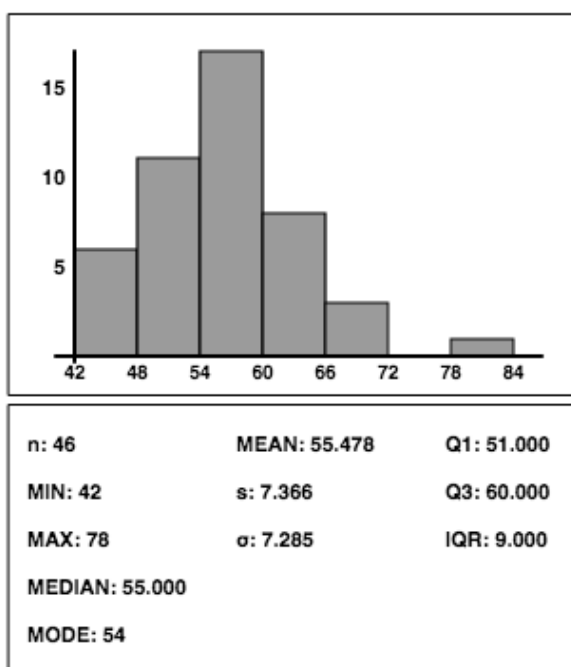
December: 3

May/June: 2

September: 1

Double check:  $2*6+2*5+4*4+3+2*2+1 = 12+10+16+3+4+1=46$

**Insight 3:** Histogram of the Ages of Inauguration of the United States Presidents (With Statistics)



**Insight 4:** The most frequent previous occupation United States Presidents held was being a lawyer.

**Insight 5:** The most frequent political parties United States Presidents were affiliated with were "Democratic" and "Republican". Note that the first couple of presidents were affiliated with the other parties. Furthermore, George Washington remains the only United States President to not be affiliated with a political party.

### **Conclusion of Fellowship Prompt Part 1**

Deliverables 0, 1, 2 served as a necessary introduction to Python and web scraping. They provided me with the fundamentals necessary to continue in the fellowship. Furthermore, I was able to use my previous project, the statistics/histogram calculator, to aid me with my report.

### **Fellowship Prompt Part 2.1 - Deliverable 3: Bozu Hops on a Plane to Politico**

Deliverable 3 tasked me with scraping a Politico article outlining 30 things Trump did during his presidency. While the scraping process originally appeared deceptively easy, some challenges quickly arose. First, one section had three “Move” headings instead of a “Move” heading, an “Impact” heading, and an “Upshot” heading. While it could have been assumed that this was a typo, I altered my code to index the faulty section and leave the “Impact” and “Upshot” blank. However, the next issue would be even more troublesome.

The text for the POLITICO website (as well as any other website) is stored in paragraph (<p>) tags, which also happen to be enclosed in special <div> tags in this case. It would make sense for each section (1 Move, 1 Impact, and 1 Upshot) to be enclosed in one div tag. However, there are inconsistencies in the website’s HTML source. Most div tags only contain one section, as usual, but a few div tags contain two sections, and one contains three sections. One section is even missing a div tag altogether. To solve this issue, I programmed three functions: `no_div_extract()`, `single_extract(div_index)`, `double_extract(div_index)`, `triple_extract()`, `move_only_extract()`. As each name suggests the purpose of each function, respectively, is to scrape the text from the missing div section, the “normal section”, the div with two sections, the div with three sections, and the section with only “Move” headings. Note that not all functions require index inputs because they only account for single cases. For more details about the code, see the GitHub link below.

After implementing this code, the following tables were created. Note that NUMPY was still used before conversion to a PANDAS data frame.

Now that the text-only table was complete, it was time to analyze the texts’ sentiments using the VADER sentiment analysis, which only took the scraped text as input. A deep copy of the text-only NUMPY array was first made. Later, a nested for loop was used to replace all the paragraph elements with positivity, neutrality, or negativity scores (total of three tables). These are attached below:

#### **Scraped Text:**

[https://docs.google.com/spreadsheets/d/1w1R54ASv6iJLaCvz0K667uAMHSCaYaFmmRa\\_qIR-BoM/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1w1R54ASv6iJLaCvz0K667uAMHSCaYaFmmRa_qIR-BoM/edit?usp=sharing)

#### **Positive Percent:**

<https://docs.google.com/spreadsheets/d/1uuFVww-KDkujeAOsNSxul6ZrhVtuy9QTTuOzEEr2deU/edit?usp=sharing>

#### **Neutral Percent:**

<https://docs.google.com/spreadsheets/d/10qFwJlbTHC8NCdxXnS3tNSG15wCBnJc8qlkXhEmVyz8/edit?usp=sharing>

#### **Negative Percent:**

<https://docs.google.com/spreadsheets/d/1eKeEo3TwNNk-13Wto1zPGs2wLiVAFKOxuR4GznfIG0o/edit?usp=sharing>

*Trump Favorability Ranking (mean of the three average positivity scores of the move, impact, and upshot columns):  $(7.397+5.762+7.341)/3 = 6.833\%$  positive*

*Trump Neutrality Ranking (mean of the three average neutrality scores of the move, impact, and upshot columns): 85.922% neutral*

*Trump Negativity Ranking (mean of the three average negativity scores of the move, impact, and upshot columns): 7.253*

*Trump Mixed Ranking (Positivity Ranking - Negativity Ranking): -0.42%*

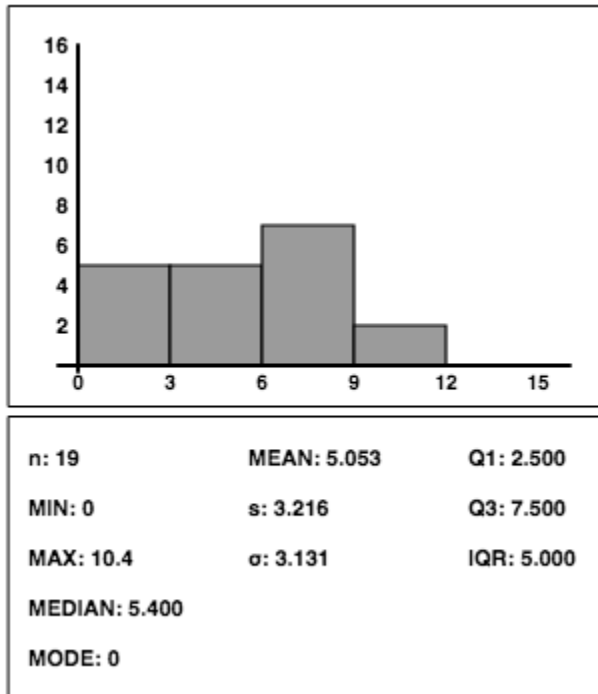
**Note:** A potential bias is introduced. The VADER sentiment analyzer cannot detect the subjects of sentences on its known and is therefore unable to tell whom a negative or positive statement is directed towards; a presence of smearing statements toward other politicians could reduce the positivity score, for instance. An attempt to resolve this issue will be highlighted further in the report.

### **Fellowship Prompt Part 2.2 - Baby Bozu's Day at BBC**

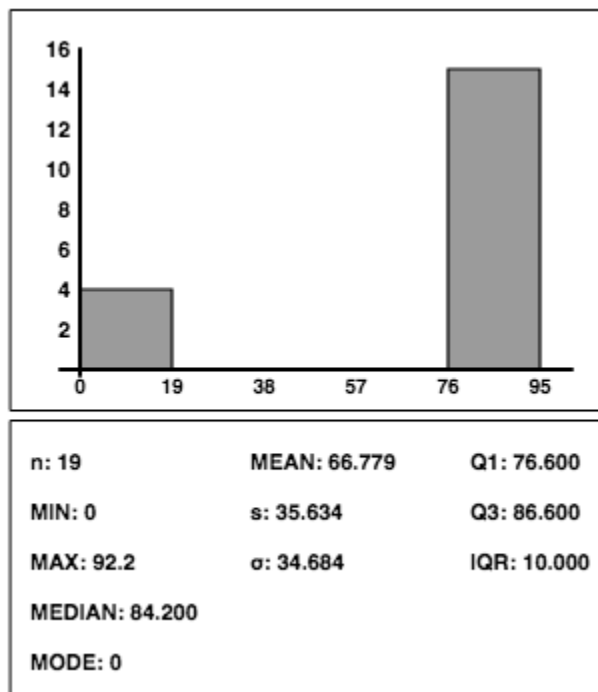
Fortunately, the BBC website did not have overlapping classes or IDs and did not have inconsistencies, so the scraping process was much more straightforward. First, a list of the links was scraped and cleaned using the `text.strip()` method of the HTML-parser. The same method was used to gather a NUMPY array of titles in the form of a column vector. Afterward, each website was individually scraped and the paragraphs were further organized into a column vector and appended to the title array. Finally, the VADER sentiment analyzer was used to create a final NUMPY array (once again in the form of a column vector) to be appended to the title + text array with `axis = 1` (to the right). Finally, this array was converted into a PANDAS data frame. Note that using NUMPY allows for easy manipulation of the data. Afterward, it is beneficial to run statistics on the positivity, neutrality, negativity, and (positivity-negativity) scores. Observe the histograms generated below:



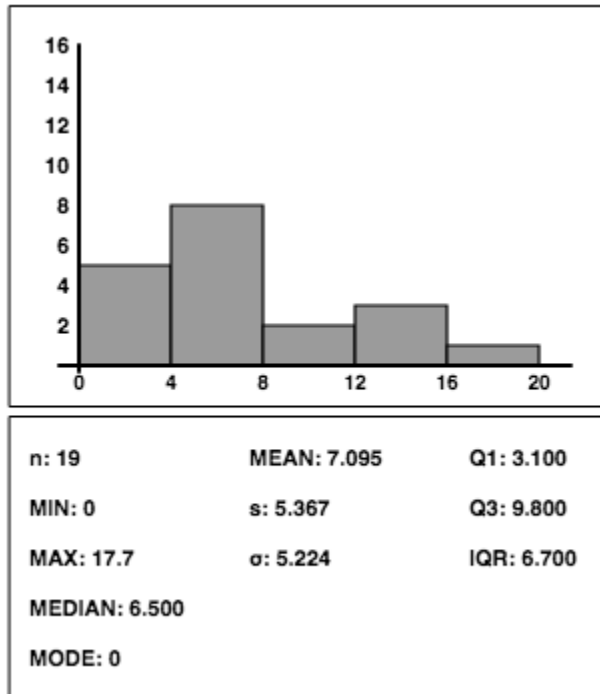
*Summary Statistics of the Positivity Percentages of 19 BBC Articles Relating to Former President Donald Trump*



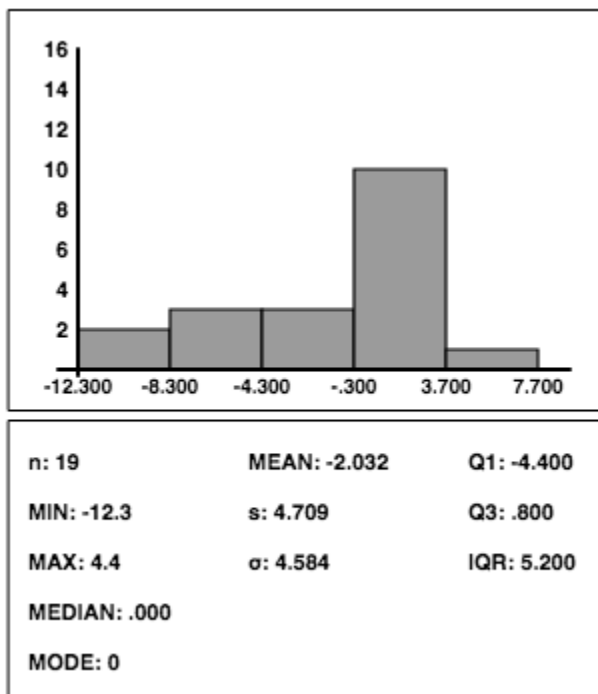
*Summary Statistics of the Neutrality Percentages of 19 BBC Articles Relating to Former President Donald Trump*



*Summary Statistics of the Negativity Percentages of 19 BBC Articles Relating to Former President Donald Trump*



*Summary Statistics of the Positivity - Negativity Percentages of 20 BBC Articles Relating to Former President Donald Trump*



*Note: These graphs were created after the attempted correction for the VADER sentiment subject bias was implemented.*

### Fellowship Prompt Part 2.3 - Bozu Finds other News Websites

In addition to the 19 BBC articles and the POLITICO passage, I scraped articles from the following news sources: AP, National Review, New York Post, and TIME magazine. Based on bias checks online, this set of sources appears to include perspectives from both sides of the political spectrum. The code to scrape these websites was nearly identical and followed the same format. First, a “master” page resulting from a search would have all the links, and the text of each passage would be scraped. Since the code was mostly copied-and-pasted at this point, I feel it is worth spending more time reviewing my attempt at solving the previously mentioned issue with the VADER sentiment analyzer.

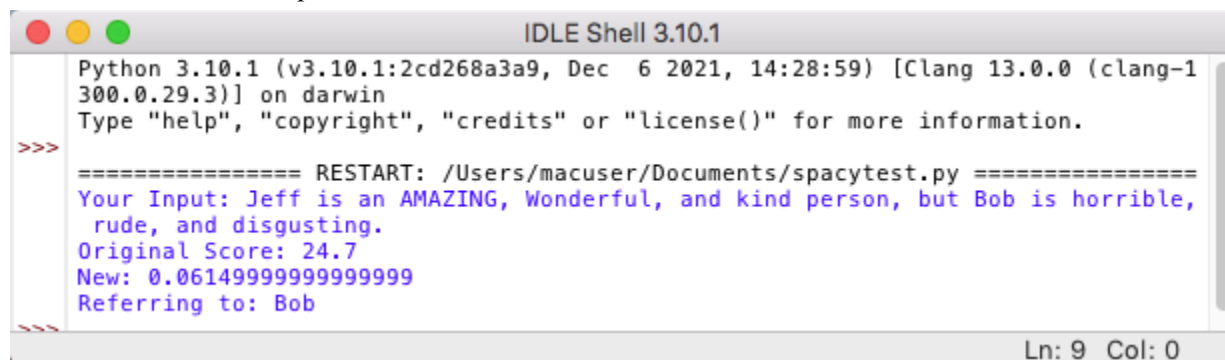
### Fellowship Prompt Part 2.4.5 -Baby Bozu Tries to Use SPACY

As previously mentioned, news sources smearing other politicians could indirectly cause the positivity score for former President Trump to fall. For instance, “right-wing” sources gave deceptively low positivity scores for Trump due to their constant smearing of “left-wing” politicians. To solve this issue, I conducted research on spaCy, an open-source software library for advanced natural language processing, and developed the following prototype (check GitHub link for code - too long to post in this doc):

1. Function with inputs “text\_input” and “opt\_subj” is defined
2. Input text is segmented into chunks (separate sentences/independent clauses) and checked.
3. Each chunk is checked, and if its subject, direct object, object of preposition, modifier, or possessive noun contains the string “opt\_subj,” the chunk is added to a “special” list. Otherwise, it remains in the original list
4. The VADER sentiment analysis is computed for both lists and a weighted average is taken. The “special” list gets a weight of at least 0.55, while the original list gets a weight of at most 0.45.

Through manual checks, I found that this prototype pushes the sentiment scores in the right direction. However, alterations to the code as well as more trials are needed to make the correction algorithm more accurate. Unfortunately, I cannot guarantee the prototype’s accuracy at the moment. However, the changes from the original scores of the passages are not unreasonable, so there is some validity.

Here is an extreme example:



```

Python 3.10.1 (v3.10.1:2cd268a3a9, Dec 6 2021, 14:28:59) [Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: /Users/macuser/Documents/spacytest.py =====
Your Input: Jeff is an AMAZING, Wonderful, and kind person, but Bob is horrible,
rude, and disgusting.
Original Score: 24.7
New: 0.06149999999999999
Referring to: Bob
  
```

It is evident that the prototype program correctly reduces the positivity score, as Bob (input for opt\_subj is “Bob”), who is being referred to, is smeared in the sentence while Jeff is praised. Unfortunately, criticisms in newspapers may be more subtle and may not even include Trump’s name, which is passed as the

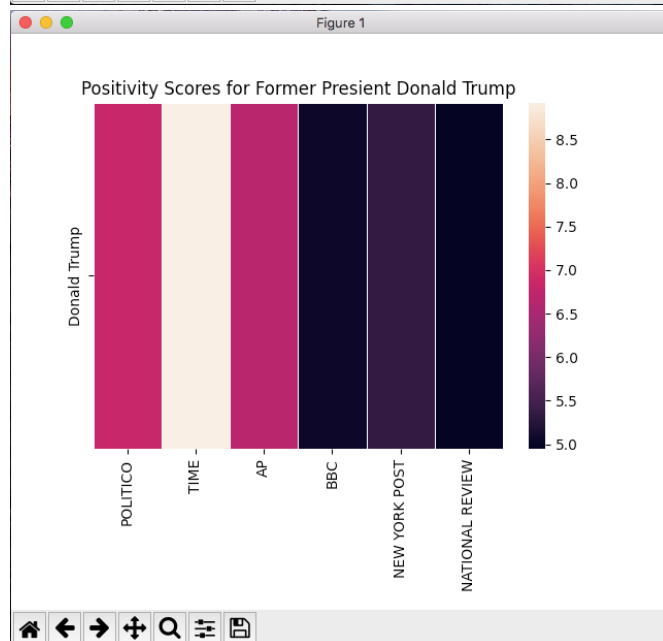
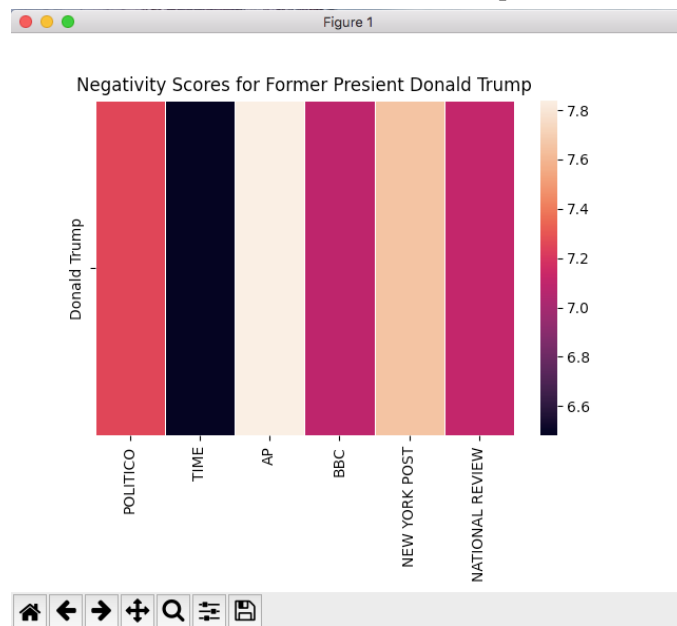
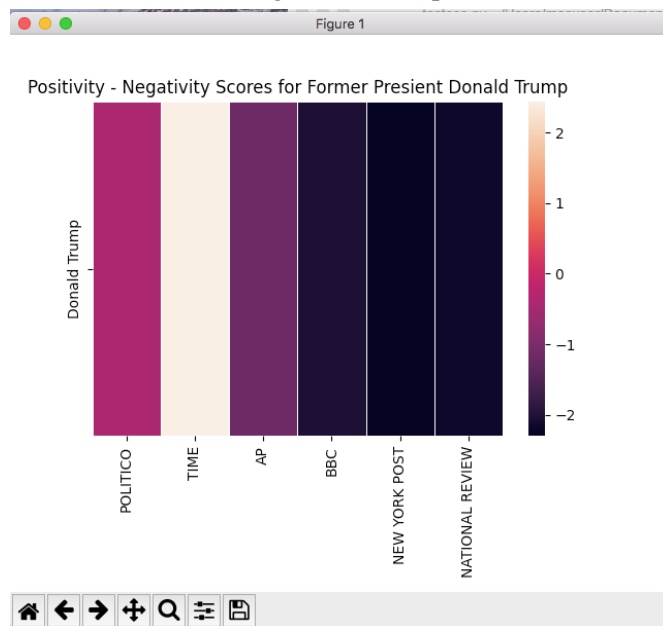
argument for “opt\_subj” in my programs. Thus, there is still a lot of room to improve upon. Nevertheless, the prototype shows promise by correctly pushing sentiment scores in the right direction for the most part. This segment of code has been implemented into almost all my programs for part 2 of the fellowship.

### Detailed Statistics for All News Sources Used:

[https://drive.google.com/drive/folders/1ehS4817u5ziC77sy9O3-6sr\\_paF3x1PH?usp=sharing](https://drive.google.com/drive/folders/1ehS4817u5ziC77sy9O3-6sr_paF3x1PH?usp=sharing)

### Fellowship Prompt Part 2.6 - Bozu the Heatmap Device

After all averages were computed, I used seaborn and Matplotlib to create these three heatmaps:



**Insights:** Based on these heatmaps, \*TIME, Politico, and AP tend to have favored President Trump the most. However, there are certain biases present, such as the people interviewed in the articles as well as the topic. Furthermore, the weighting may not have necessarily balanced the sentiment scores enough. Note that TIME magazine regarded former President Trump as “person of the year” of 2016. Either way, these\* sources can be considered mostly moderate, though some tend to sway either “Left” or “Right”

### Fellowship Prompt Part 2.7 - Next Steps

Overall, I have enjoyed the experience. This fellowship provided me with unique challenges and ultimately made me a better programmer and problem-solver. I look forward to joining the next cohort. As for the project itself, I aim to improve my spaCy code to make it more accurate by running more trials and learning more about the functionalities of this package.

### Location of all Submissions (Code + CSV):

<https://github.com/ZinnunMalikov/CodeBozu-Part-2/tree/main>

<https://www.khanacademy.org/computer-programming/statistics-calculator/5217484206456832>

### Tools Used:

1. Python language
2. bs4 - BeautifulSoup
3. requests
4. pandas
5. numpy
6. csv
7. copy
8. vaderSentiment.vaderSentiment - SentimentIntensityAnalyzer
9. spaCy
10. seaborn
11. matplotlib.pylab
12. Processing.js (Statistics Calculator)