# What to Read Next? Analyzing the Digital Fragmenta Historicorum Graecorum with Python and NLP

Since the onset of COVID-19 and social distancing guidelines I, like many others, have been looking for ways to assuage anxiety. I've managed to find some comfort (and edification) in the Greek and Latin classics. Now is as good a place as any in this story to offer Thomas Jefferson's evergreen, and, in some circles, famous quote:

> ***[T]o read the Latin & Greek authors in their original is a sublime luxury … I enjoy Homer in his own language infinitely beyond Pope's translation of him, & both beyond the dull narrative of the same events by Dares Phrygius, & it is an innocent enjoyment. I thank on my knees him who directed my early education for having put into my possession this rich source of delight: and I would not exchange it for any thing which I could then have acquired & have not since acquired.***

In a previous life I was a classicist. I studied the ancient novel, and wrote a thesis about Heliodorus (which I believe is the **only** thesis in Washington University's Open Scholarship library tagged with "Byzantine Literature"). When I was in college and later graduate school, I relied in no small part on digital versions of classical literature, and the tools built on top of those digital versions, like Perseus and the TLG. I am not one to romanticize the scholar hunched over a desk with Liddell and Scott, when there's an electronic version of the dictionary available from Perseus. However, that was the real limit of my experience with work in the digital humanities—until, of course, restlessness in the time of Coronavirus led me to a new discovery.

The DFHG—Digital Fragmenta Historicorum Graecorum, or Digitized Fragments of Greek Historical Writers (perhaps better *Die digitale Fragmente der griechischen historischen Schrift*) hosted by the University of Leipzig and created by Monica Berti and Gianluca Cumani, is a remarkable achievement. It's a digitized version of fragmentary Greek historical writing, mostly by authors who aren't typically taught in a classics curriculum.

Many people know the hits (everyone's heard of Thucydides), but the DFHG is the place to go for deep cuts. Maybe you're reading Gibbon and you come across an anecdote like this one:

> *The historian Priscus, whose embassy is a source of curious instruction, was accosted in the camp of Attila by a stranger, who saluted him in the Greek language, but whose dress and figure displayed the appearance of a wealthy Scythian. In the siege of Viminiacum he had lost, according to his own account, his fortune and liberty: he became the slave of Onegesius but his faithful services against the Romans and the Acatzires had gradually raised him to the rank of the native Huns, to whom he was attached by the domestic pledges of a new wife and several children. The spoils of war had restored and improved his private property; he was admitted to the table of his former lord and the apostate Greek blessed the hour of his captivity, since it had been the introduction to a happy and independent state, which he held by the honourable tenure of military service. This reflection naturally produced a dispute on the advantages and defects of the Roman government, which was severely arraigned by the apostate, and defended by Priscus in a prolix and feeble declamation.*

But wait… who the hell is Priscus? Where can I read this "source of curious instruction," or find his "prolix and feeble declamation" on the Roman system of government? The DFHG is your answer.

The DFHG also comes with a set of tools, including an API that make querying the corpus relatively easy (and it's free). The search tools are limited, but there's a wealth of information available. That brings me, finally, to the topic of this blog and a project: what can we say about the lexical complexity of the authors in the DFHG? If I were going to choose something to read, I'd want to make sure I got to spend my time **reading** and not looking words up in the dictionary. For this blog, I'm going to extract textual data from the DFHG, examine it with an NLP engine designed specifically for use with classical languages, and use the results to choose something to read. Let's see what we find.

. . .

## Getting the data

The DFHG's api relies on queries of **authors** (this makes sense because of the fragmentary nature of the corpus—we don't have specific "works" to search through, like *The Republic*, so author is the most atomic we can get). We thus need to extract a list of all the authors included in the corpus. We can then build out api calls and feed our data into an NLP engine.

This code outlines the steps I took to send a request to the DFHG's api documentation, clean up the html, and get a (clean) list of authors.

## Import modules, set params, get data

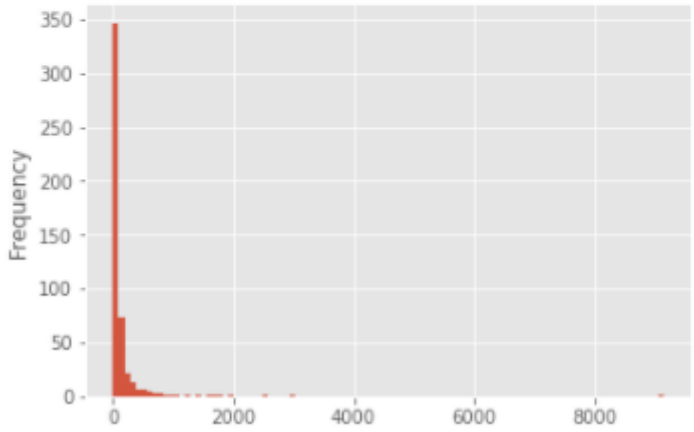
In [37]:

```
from bs4 import BeautifulSoup, NavigableSt
ring, Tag

import requests

url = 'http://www.dfhg-project.org/DFHG/ap
i.php' # api documentation URL

res = requests.get(url)

content = res.content.decode() # original
 is in bytes, convert to string
```

Let's do a quick check of the words counts from the fragments in our corpus:

```
df.total_words_in_fragment.plot('hist', bins = 100)
```

```
/Users/aleedom/anaconda3/lib/python3.7/site-packages/ip
not be called with positional arguments, only keyword a
e future. Use `Series.plot(kind='hist')` instead of `Se
  """Entry point for launching an IPython kernel.
<matplotlib.axes._subplots.AxesSubplot at 0x1a5aecf390>
```




## Latium est, non legitur

This is not a clean dataset—at least, not in the sense that we're ready to do any actual analytics yet. The problem here is the nature of the material: these are fragmentary pieces of writing, some of which survive only through quotations in other sources. What makes life even more difficult for us is disentangling the "actual" authorial content from whatever may have accrued to the text at the hands of scribes and centuries. As a more particular matter, what we have here are actually several fragments that are in *Latin*.

The DFHG does provide a Latin translation for most (if not all) of the fragments in the corpus, but when Latin comments turn up in the 'text' field of the data we scraped earlier, that suggests the influence of a commentator, not necessarily the author we want to look at.

To proceed, we need to figure out a way to separate the content we want from the commentary we don't. Luckily for us, Latin and Greek are different languages written in different alphabets. (It'd be a different question entirely if we were trying to segregate, say, Spanish and French words when they're bunched up together.)

Since Latin uses, naturally, the Latin alphabet, we can use regex to find patterns of text in the data that are made up of Latin characters. I'm going to be pretty broad here: we want to capture as much of the Latin as possible while leaving Greek. If a passage is all "original," we should let it stand. Otherwise, we should flag it. This line will pick up any row in our data that has a Latin character in the "text" field:

```
latin_index = author_df['text'].str.contains('([a-zA-Z])\)', regex = True)
```

Let's do some further cleaning to get the number of Latin words, their positions within the text, and the relative frequency of Latin/non-Laitn words in each fragment:



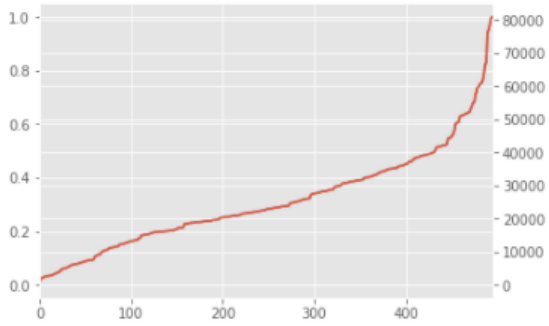
We now have counts and frequencies. At this point we have to make some decisions about how to handle these data points: we can just drop these rows and move on, or we can try and spare the texts that have a minimal amount of Latin in them.

Let's try the second route. There are options here as well: we can either drop rows entirely, or try and filter out the Latin words from each text. I'm going to try a combination of both, going through the data first and trying to drop rows that are entirely or almost entirely Latin, then scrubbing the remaining text.

First, let's look at the distribution of Latin in the corpus we have.

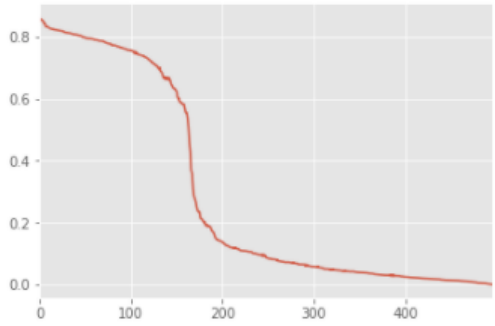
```
[517]: fig = plt.figure() # Create matplotlib figure

       ax = fig.add_subplot(111) # Create matplotlib axes
       ax2 = ax.twinx()

       df.reset_index()['cum_pct'].plot(ax = ax)
       df.reset_index()['word_sum'].plot(ax = ax2)
```

```
[517]: <matplotlib.axes._subplots.AxesSubplot at 0x1a523722e8>
```




```
[522]: df.reset_index().latin_word_freq.sort_values(ascending = False).plot()
```

```
[522]: <matplotlib.axes._subplots.AxesSubplot at 0x1a522ec208>
```




What these charts tell me is that our intuition was right—we have a few authors whose work as preserved in the DFHG is almost entirely in Latin (i.e. those fragments that fall on the left hand side of the lower figure), and many others where a few stray words have made their way into the (otherwise Greek) text.

To make a final pass at cleaning the corpus, I defined a few functions to take a list of words and return the indices of all Latin words in the list. I then filter the list of words by deleting elements at those indices.

```
for index, row in final_df.iterrows():
    indicies = row['latin_word_index']
    for index in sorted(indicies, reverse = True):
        del row['words'][index]
```

I then drop fragments with remaining Latin words.

```
final_df = df.drop(df[df['latin_word_count'] != 0])
```

## NLP Time

Now we can move on to the NLP part of this project. I'm taking my cues from some of the example notebooks posted on the CLTK Github page. I'm going to calculate a few summary metrics, including lexical density, which measures the number of "lexical" words as a percentage of total words in a passage. For example, the lexical density of the Thomas Jefferson quote at the top of this article is 48.24% (according to Analyze My Writing). We can take a first look at lexical density by **fragment** after some cleanup. We'll then look at the lexical density by author.

First, we need to regularize the texts and strip out punctuation and so-called stop words. (A special note on punctuation—Greek uses the semicolon ; to indicate a question, so we won't include that character in the punctuation list to filer.) A stop word in this case is a common word that doesn't add much to our understanding of the meaning of the text. In English these are words like "the," "is," "and," and so on. The CLTK comes with a built-in list of stop words for both Greek and Latin, so I'll use that functionality to clean up the fragments. This code will clean up the list of words we've extracted from each fragment:

```
def filter_punct(words):
    out = [word for word in words if word not in punct]
    out = [word for word in out if word not in stops]
```

```
final_df['words'].apply(clean_punct)
final_df['lemmata'].apply(clean_punct)
```

```
## Note that this function actually cleans both punctuation
and stop
## words
```

```
final_df[['author', 'clean_lemmata']] # prints the following
```

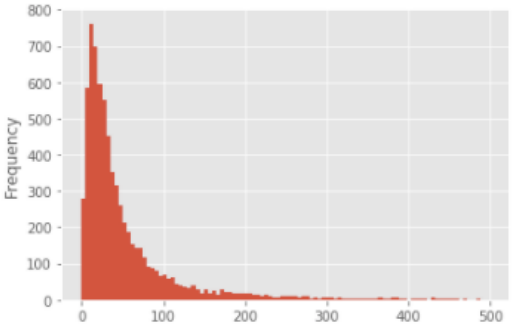
```
final_df[['author','clean_lemmata']]
```

| | author | clean_lemmata |
|---|---|---|
| **0** | ABAS | [ὁ, κανδαύλης, γυνὴ, ὅς, ἡρόδοτος, οὐ, λέγω1,... |
| **2** | ABRON VEL HABRON BATIENSIS | [βατὴ, δῆμος, ὁ, αἰγηίδος, φυλάζω, ὅθεν, εἰμί... |
| **3** | ABYDENUS | [ἐκ, ὁ, ἀβυδηνός, περὶ, ὁ, ὁ, χαλδαῖος, βασιλ... |
| **7** | ABYDENUS | [αβυδηνου, ἐντὶ, δ,', οἱ, λέγω1, τοὺς, πρότερ... |
| **13** | ABYDENUS | [μετὰ, οὗτος, τὸν, ναβουχοδονόσωρ, υἱὸς, αὐτό... |
| **...** | ... | ... |
| **8060** | ZOPYRUS BYZANTIUS | [ζώπυρος, ἐν, τέταρτος, μίλητος, κτίσις, γράφε... |
| **8061** | ZOPYRUS BYZANTIUS | [ἔρμος, ..., ζώπυρος, δὲ, ἐν, ὁ, πρεὶ, ὁ, ποτα... |
| **8062** | ZOPYRUS BYZANTIUS | [ἀφροδισιὰς, πόλις, κίλιξ, περὶ, ὅς, ἀλέξανδρο... |
| **8063** | ZOPYRUS BYZANTIUS | [δίδυμος, δ,', ἐν, ἀθήνη, ἀπὸ, ὁ, φυγή, ἔρχομ... |
| **8064** | [DEMODAMAS HALICARNASSENSIS] | [ἀνθέω, δὲ, στεφανωτικός, μιμνήσκω, ὁ, μὲν, τὰ... |

Now a quick check of the length of each fragment we have:

```
final_df[final_df['word_count'] < 500].word_count.plot(kind = 'hist', bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5f125c88>
```




And finally we can now calculate the fragment level lexical density for our corpus:

```
def lemmatize(text):
    lemmata = lemmatizer.lemmatize(text) # from cltk
    return lemmata
```

```
df['lemmata'] = df['clean_words'].apply(lemmatize)
```

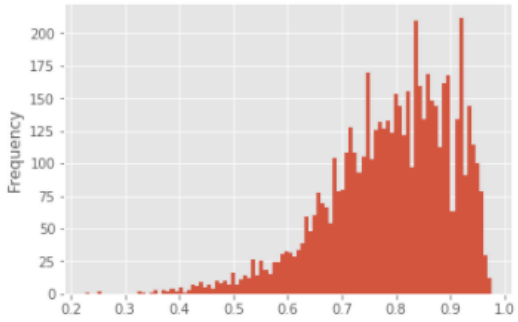
```
df['lexical_density'] = df.apply(lambda x:
len(set(x['lemmata'])) / len(x['lemmata']), 1) # python set
```

```
    return set of unique elements
```

Let's remove the fragments whose lexical density is 1 and plot to see what kind of distribution we have. (I'm omitting fragments where density is 1 because these fragments tend to be **much** shorter—the average word count for a 100% density fragment is ~10 words, while those with lower densities average ~82 words. Since these are fragments, it's likely that passages with 100% density are not representative of an author's voice)

```
df[df['lexical_density'] != 1].lexical_density.plot(kind = 'hist', bins  = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a7aa8eba8>
```




Looks like the distribution is **approximately** normal, but with lots of left skew. We can check the descriptive statistics for more:

```
df.lexical_density.describe()
```

```
count    7086.000000
mean        0.830478
std         0.131781
min         0.225822
25%         0.744681
50%         0.840909
75%         0.933333
max         1.000000
Name: lexical_density, dtype: float64
```

. . .

## Combinatio Nova

So now we have some things to say about individual fragments, but fragments are, well, fragmentary. Can we say anything about the styles of a particular author? To do so, we'll need to group our passages by author.

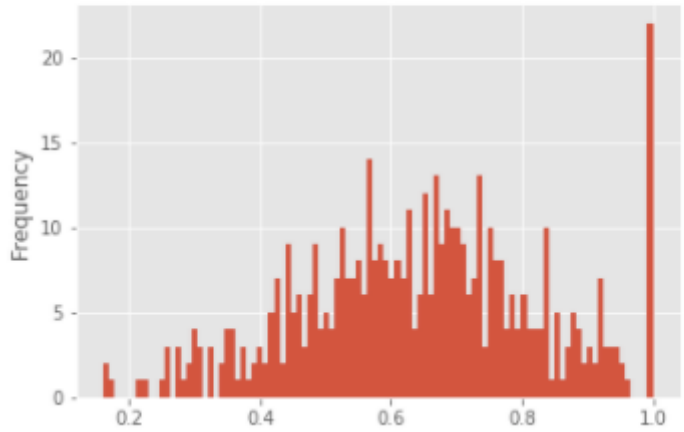
This is straightforward in Python because of the way lists behave. If we have two lists in Python, their sum is the **union** of the two sets, i.e. we'll get all elements from both lists back in one.

We can apply this to our corpus by grouping by author then summing up our words. We'll recalculate lexical density on an author by author basis, and see which authors favor particular words.

We get a different picture of lexical density if we look at things at this level, instead of at the fragment level:

```
words_df.lexical_density.plot(kind = 'hist', bins = 100)
```

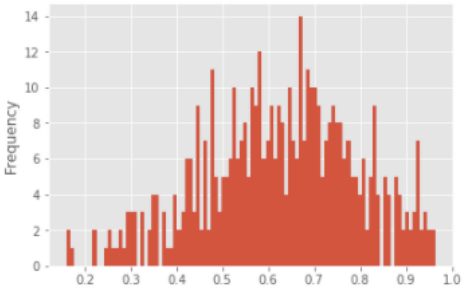
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a7d77b160>
```




And again removing authors whose density is 1:

```
words_df[words_df['lexical_density'] != 1].lexical_density.plot(kind = 'hist', bins = 100)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a7d3999b0>
```




So we now have what looks a bit more like a normal distribution of lexical densities, which is—roughly—what we'd expect. Some authors are going to be more difficult, others a little simpler in their writing. Let's take a look at the densest authors with over 100 words in the corpus:

```
words_df[(words_df['lexical_density'] >= .64) &
(words_df['word_count'] > 100)]\
.sort_values('word_count', ascending = False)

#.64 is the median density
```

```
words_df[(words_df['lexical_density'] >= .64) & (words_df['word_count'] > 100)]\
.sort_values('word_count', ascending = False)
```

| | author | clean_words | clean_lemmata | lexical_density | word_count |
|---|---|---|---|---|---|
| 388 | POSIDONIUS OLBIOPOLITA | [ὁ, δέ, τῶν, μακεδόνων, βασιλεύς, ὡς, φησι, πο... | [ὁ, δέ, ὁ, μακεδών, βασιλεύς, ὅς, φημί, πολύβι... | 0.648855 | 262 |
| 155 | CREPEREIUS CALPURNIANUS POMPEIOPOLITANUS | [οὗτος, μέν, τοιαῦτα, ἕτερος, δέ, θουκυδίδου, ... | [οὗτος, μέν, τοιοῦτος, ἕτερος, δέ, θουκυδίδης... | 0.672000 | 250 |
| 415 | SELEUCUS ALEXANDRINUS | [ὁμηρίδαι, σέλευκος, ἐν, β, περί, βίων, ἁμαρτά... | [ὁμηρίδης, σέλευκος, ἐν, β, περί, βιόω, ἁμαρτά... | 0.640693 | 231 |
| 38 | ANONYMUS CORINTHIUS | [ἄλλος, μάλα, καί, οὗτος, γελοῖος, οὐδέ, τόν, ... | [ἄλλος, μάλης, καί, οὗτος, γέλοιος, οὐδέ, τόν,... | 0.693333 | 225 |
| 157 | CRITOLAUS | [ῥωμαίων, πρὸς, πύρρον, ἠπειρώτην, πολεμούντων... | [ῥωμαίων, πρὸς, πύρρος, ἠπειρόω, πολεμέω, αἰμί... | 0.695067 | 223 |
| ... | ... | ... | ... | ... | ... |
| 314 | MENYLLUS | [σεπτίμιος, μάρκελλος, γήμας, σιλουίαν, τά, πο... | [σεπτίμιος, μάρκελλος, γαμέω, σιλουίαν, τά, πο... | 0.754717 | 106 |
| 111 | BION SOLENSIS | [ἀ, ;, :, ;, ;, καθίστων, δέ, καί, πολλοί, τού... | [ἀ, ;, :, :, ;, καθιστάω, δέ, καί, πολλοί, τού... | 0.695238 | 105 |
| 480 | ZENODOTUS TROEZENIUS | [ζηνόδοτος, δέ, τροιζήνιος, συγγραφεύς, ὁμβρικ... | [ζηνόδοτος, δέ, τροιζήνιος, συγγραφεύς, ὁμβρικ... | 0.701923 | 104 |
| 289 | LUCILLUS TARRHAEUS | [θεσσαλονίκη, πόλις, μακεδονίας, ἤτις, ἄρα, ἐκ... | [θεσσαλονίκη, πόλις, μακεδονία, ὅστις, ἄρον, κ... | 0.676471 | 102 |
| 172 | DEMAGORAS SAMIUS | [δημαγόρας, δέ, ἀπό, λιβύης, ἐλθοῦσαν, τήν, ἠλ... | [δημαγ/ορας, δέ, ἀπό, λιβύη, ἔρχομαι, τήν, ἠλέ... | 0.712871 | 101 |

71 rows × 5 columns

Conversely, if what we want to do is find someone with a sufficient sample size and a **low** lexical density (and answer the question posed at the beginning of this blog) we can execute the following:

```
words_df[words_df['word_count'] > 250]\
.sort_values('lexical_density', ascending = True)

# The median word count in our cleaned sample is 169.
# 250 gives us the 65th percentile
```

```
words_df[words_df['word_count'] > 250]\
.sort_values('lexical_density', ascending = True)
```

| | author | clean_words | clean_lemmata | lexical_density | word_count |
|---|---|---|---|---|---|
| 279 | JOANNES ANTIOCHENUS | [1, ἀπό, τῆς, ἐκθέσεως, ἰωάννου, ἀντιοχέως, τῆ... | [1, ἀπό, ὁ, ἔκθεσις, ἰωάννης, ἀντιοχέως, ὁ, πε... | 0.161421 | 47305 |
| 334 | NICOLAUS DAMASCENUS | [ἀντίπατρος, ἦν, νικολάου, τοῦ, δαμασκηνοῦ, πα... | [ἀντίπατρος, εἰμί, νικόλαος, ὁ, δαμασκηνόν, πα... | 0.164711 | 35887 |
| 304 | MENANDER PROTECTOR | [μένανδρος, προτίκτωρ, ἱστορικός, ὃς, λέγει, π... | [μένανδρος, προτίκτωρ, ἱστορικός, ὃς, λέγω1, π... | 0.170874 | 22736 |
| 80 | ARISTOTELES | [ἀπόλλων, πατρῷος, ὁ, πύθιος, ..., τόν, δέ, ἀπ... | [ἀπόλλω1, πατρῷος, ὁ, πύθιος, ..., τόν, δέ, ... | 0.218795 | 18579 |
| 393 | PRISCUS PANITES | [πρίσκος, πανίτης, σοφιστὴς, γεγονὼς, ἐπί, τῶν... | [πρίσκος, πανίτης, σοφιστὴς, γεγονὼς, ἐπί, ὁ, ... | 0.223204 | 7809 |
| ... | ... | ... | ... | ... | ... |
| 399 | PROXENUS | [ποιμὴν, νέμων, πρόβατα, ἐν, τοῖς, τῆς, δωδώνη... | [ποιμὴν, νέμω, πρόβατον, ἐν, ὁ, ὁ, δωδώνη, ἔλο... | 0.612676 | 284 |
| 64 | ARISTAGORAS MILESIUS | [ἑρμοτυμβιεῖς, μοῖρα, τῶν, μαχίμων, ἐν, αἰγύπτ... | [ἑρμοτυμβιεῖς, μοῖρα, ὁ, μάχιμος, ἐν, αἰγυπτο... | 0.613240 | 287 |
| 431 | SOSTRATUS | [σωστράτου, ἐν, δευτέρῳ, τυρρηνικῶν, αἴολος, τ... | [σώστρατος, ἐν, δεύτερος, τυρρηνικός, αἴολος, ... | 0.623188 | 345 |
| 424 | SOCRATES RHODIUS | [σωκράτης, δέ, ὁ, ῥόδιος, ἐν, τρίτῳ, ἐμφυλίου,... | [σωκράτης, δέ, ὁ, ῥόδιος, ἐν, τρίτος, ἐμφύλιος... | 0.624549 | 277 |
| 388 | POSIDONIUS OLBIOPOLITA | [ὁ, δέ, τῶν, μακεδόνων, βασιλεύς, ὣς, φησι, πο... | [ὁ, δέ, ὁ, μακεδών, βασιλεύς, ὅς, φημί, πολύβι... | 0.648855 | 262 |

Looks like Joannes Antiochenus (John of Antioch, a 7th century chronicler) is our guy, by this metric at least. Can we elaborate on this with a more sophisticated check of the difficulty, if not the density of each author?

. . .

## Ad pedem litterae

Let's try and evaluate each author's style. We'll want to get word counts, then see which authors use *hapax legomena*—unique words.

We can use Python's Counter library to get word counts. Counter returns a dictionary of words and their counts in a text. For example, `Counter(['hello', 'world'])` returns `{Counter: 'hello' : 1, 'world' : 1}` .

```
from collections import Counter

def word_counter(words):
    return Counter(words)

words_df['lemma_counts'] = words_df['clean_lemmata'].apply(word_counter)

import operator
words_df['lemma_counts'] = words_df['lemma_counts']\
.apply(lambda x: sorted(x.items(), key = operator.itemgetter(1), reverse = True))
```

```
words_df.head()
```

| | author | clean_words | clean_lemmata | lexical_density | word_count | lemma_counts |
|---|---|---|---|---|---|---|
| 0 | ABAS | [ἡ, κανδαύλου, γυνή, ἧς, ἡρόδοτος, οὐ, λέγει, ... | [ὁ, κανδαύλης, γυνή, ὅς, ἡρόδοτος, οὐ, λέγω1,... | 0.722892 | 83 | [(ὁ, 7), (ἡρόδοτος, 4), (ὄνομα, 3), (νύσιος, 3... |
| 1 | ABRON VEL HABRON BATIENSIS | [βατή, δῆμος, τῆς, αἰγηίδος, φυλῆς, ὅθεν, ἦν, ... | [βατή, δῆμος, ὁ, αἰγηίδος, φυλάζω, ὅθεν, εἰμί... | 0.937500 | 16 | [(ὁ, 2), (βατή, 1), (δῆμος, 1), (αἰγηίδος, 1)... |
| 2 | ABYDENUS | [ἐκ, τῶν, ἀβυδηνοῦ, περί, τῆς, τῶν, χαλδαίων, ... | [ἐκ, ὁ, ἀβυδηνός, περί, ὁ, ὁ, χαλδαῖος, βασιλ... | 0.576531 | 392 | [(ὁ, 23), (δέ, 15), (καί, 15), (ἐκ, 9), (, 9)... |
| 3 | ACESANDER | [εἶπον, ὅτι, καί, πελίαν, θάπτων, ἄκαστος, ὁ, ... | [εἶπον, ὅστις, καί, πελίης, θάπτω, ἄκαστος, ὁ,... | 0.519164 | 287 | [(ὁ, 22), (δέ, 18), (καί, 15), (ἐν, 8), (τήν, ... |
| 4 | ACUSILAUS | [ἡσίοδος, χάους, καί, γῆς, ἔρωτα, υἱόν, λέγει,... | [ἡσίοδος, χαόω, καί, γαῖα, ἔρως, υἱόν, λέγω1,... | 0.441973 | 1034 | [(ὁ, 96), (καί, 50), (δέ, 41), (ἀκουσίλαος, 32... |

Let's now take a closer look at the words our authors are using. We can repeat the process above, but on the corpus as a whole.

```
most_frequent_words = Counter(words_df['clean_lemmata'].sum())
most_frequent_words = sorted(most_frequent_words.items(), key = operator.itemgetter(1), reverse = True)
most_frequent_words = pd.DataFrame(most_frequent_words)
```

```
...
```

```
most_frequent_words.head(10)
```

| | word | count |
|---|---|---|
| 0 | ὁ | 40890 |
| 1 | καί | 25029 |
| 2 | δέ | 13193 |
| 3 | ἐν | 7639 |
| 5 | τήν | 7319 |
| 6 | τόν | 5975 |
| 7 | εἰμί | 5229 |
| 8 | τό | 5068 |
| 9 | αὐτός | 4838 |
| 10 | οὗτος | 4753 |

This list is basically what you'd expect to see. Greek 101 students learn all of these words in the first few weeks of class (ὁ (ho) is the definite article, for example, καί (*kai*) is "and," and so on). Note too that this data frame exposes the limitations of the CLTK (or perhaps my usage of it): in Greek, ὁ, τήν, and τὸ are all the same word, but our lemmatizer considers them to be different. Which author uses the rarest words, and which author uses the most common ones?

I'm going to approach this question by building on our word counts and developing a word frequency counter. We can then calculate an author's mean frequency of use—an ad hoc metric that should describe, along with lexical density, the lexical *difficulty* of an author's work.

I'm going to normalize each word according to occurrences per 10,000 words using our word counts data frame. We have everything we need

already: the count of each word, and the total number of words in the corpus.

There are 476,180 words in our corpus, so we need to divide each count by 47.618 to get our frequency. With frequencies in hand, we can create a weighted average frequency to gauge the lexical difficulty of an author. Lower frequencies will tend to indicate more difficult authors, and higher frequencies easier authors (I write all this with the caveat that almost nothing about Ancient Greek is "easy"). There is little (if any) intrinsic meaning to a weighted average frequency like the one we're building here, rather it's better to think of the resulting number as an **index** of difficulty, and a first step toward evaluating the composition of a passage.

It might make a bit more sense to take the inverse of the frequency per 10k column as the weight for calculating our index. Doing so would do two things: first, it would reverse the direction of our frequency weights—so now higher numbers are harder to read, which might make more intuitive sense—and second, it would assign a much higher weight to words that occur less often in our corpus, because 1/x does not scale linearly as x approaches 0. Note however, that either way you run it, this is a slow function.

What I'm going to do is define a function to walk through the word count dictionaries we have. I'm taking advantage of the fact that we've already defined our frequency data frame with each word from the corpus.

```
def weighted_word_freq(word_tuples):
    tups = word_tuples
    weight_sum = 0
    value_sum = 0
    for tup in tups:
        try:
            weight = most_frequent_words.loc[most_frequent_words['word'] == tup[0], 'inv_freq'].values[0]
            weight_sum += weight
            value_sum += weight * tup[1]
        except:
            pass
    return value_sum / weight_sum
```

```
words_df['inv_freq'] = words_df.lemma_counts.apply(weighted_word_freq)
```

```
words_df.head()
```

| | author | clean_words | clean_lemmata | lexical_density | word_count | lemma_counts | unique_lemmata | weighted_average_frequency | inv_freq |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ABAS | [ἡ, καυδαύλου, γυνή, ἧς, ἡρόδοτος, οὐ, λέγει, ... | [ὁ, καυδαύλης, γυνή, ὅς, ἡρόδοτος, οὐ, λέγω1,... | 0.722892 | 83 | [(ὁ, 7), (ἡρόδοτος, 4), (ὄνομα, 3), (υἱόσιος, 3... | [ὄνομα, ἡρόδοτος, ὡς, λέγω1, γένος, ἥν, γυνή, ... | 3.877270 | 1.039547 |
| 1 | ABRON VEL HABRON BATIENSIS | [βατή, δῆμος, τῆς, αἰγηίδος, φυλῆς, ὅθεν, ἦν, ... | [βατή, δῆμος, ὁ, αἰγηίδος, φυλάβω, ὅθεν, εἰμί,... | 0.937500 | 16 | [(ὁ, 2), (βατή, 1), (δῆμος, 1), (αἰγηίδος, 1),... | [ὁ, ἐξηγητής, καλλῖνς, αἰγηίδος, περί, ὅθεν, ἐ... | 1.543122 | 1.000009 |
| 2 | ABYDENUS | [ἐκ, τῶν, ἀβυδηνοῦ, περί, τῆς, τῶν, χαλδαίων, ... | [ἐκ, ὁ, ἀβυδηνός, περί, ὁ, ὁ, χαλδαῖος, βασιλ... | 0.576531 | 392 | [(ὁ, 23), (δέ, 15), (καί, 15), (ἐκ, 9), (, 9)... | [ἀβυδηνός, παντιβίβλιος, ἄνθρωπος, θεάω, κατακ... | 9.516595 | 1.034891 |
| 3 | ACESANDER | [εἶπον, ὅτι, καί, πελίαν, θάπτων, ἄκαστος, ὁ, ... | [εἶπον, ὅστις, καί, πελίης, θάπτω, ἄκαστος, ὁ,... | 0.519164 | 287 | [(ὁ, 22), (δέ, 18), (καί, 15), (ἐγώ, 8), (τήν, ... | [διακωμῴζω, εὔρυτος, λίβυς, ὅ, εἶπον, ἀκέσαν... | 9.956201 | 1.122428 |
| 4 | ACUSILAUS | [ἡσίοδος, χάους, καί, γῆς, ἔρωτα, υἱόν, λέγει,... | [ἡσίοδος, χαόω, καί, γαῖα, ἔρως, υἱόν, λέγω1,... | 0.441973 | 1034 | [(ὁ, 96), (καί, 50), (δέ, 41), (ἀκουσίλαος, 32... | [ὤγυγος, καταλύω, ληταῖος, πασιμάχη, μενέλαος, ... | 28.055590 | 1.091526 |

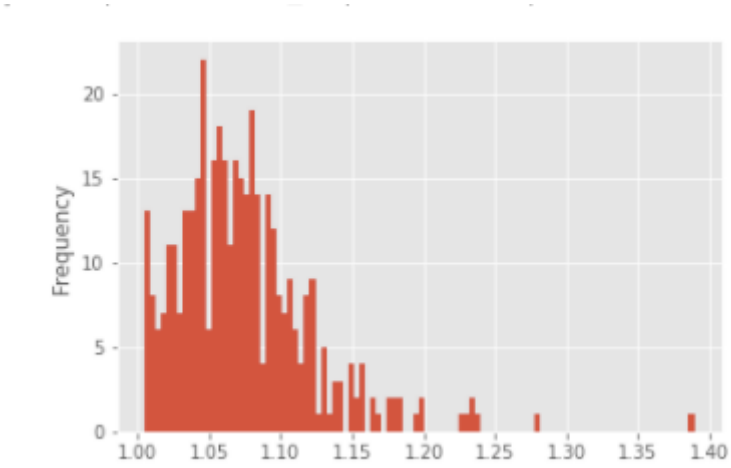
What this function does is multiply the number of occurrences of a given word in an author's body of work by that word's relative

frequency in the DFHG corpus. So, for example if we look at Abas in our words data frame, we have the following lemmata:

```
words_df.lemma_counts[0][:5]
```

```
[('ὁ', 7), ('ἡρόδοτος', 4), ('ὄνομα', 3), ('ὑάσιος', 3), ('καί', 3)]
```

What I've done is gotten the occurrences of ὁ in the Abbas (here 7) and multiplied that by the inverse frequency we calculated above. What does the resulting distribution look like?




And for comparison's sake, here are the distributions of lexical difficulties, one with the inverted frequency (bigger numbers mean more rare words) and the other with regular frequency (smaller numbers mean more rare words):

```
words_df[['inv_freq', 'weighted_average_frequency']].describe()
```

| | inv_freq | weighted_average_frequency |
|---|---|---|
| **count** | 483.000000 | 483.000000 |
| **mean** | 1.060751 | 21.431947 |
| **std** | 0.051641 | 59.307833 |
| **min** | 1.000000 | 1.000000 |
| **25%** | 1.018529 | 3.100719 |
| **50%** | 1.056069 | 6.000524 |
| **75%** | 1.088332 | 14.934546 |
| **max** | 1.389157 | 810.789588 |

Does this inform our decision on whom to read?

```
words_df[words_df['word_count'] > 250].sort_values('inv_freq')
```

| | author | clean_words | clean_lemmata | lexical_density | word_count | lemma_counts | unique_lemmata | weighted_average_frequ |
|---|---|---|---|---|---|---|---|---|
| 388 | POSIDONIUS OLBIOPOLITA | [ὁ, δέ, τῶν, μακεδόνων, βασιλεύς, ὡς, φησι, πο... | [ὁ, δέ, ὁ, μακεδών, βασιλεύς, ὅς, φημί, πολύβι... | 0.648855 | 262 | [(ὁ, 22), (καί, 9), (δέ, 8), (τόν, 8), (, 5),... | [κωλύω, ὑπέρ, δ, ὑφ, ποιέω, πόλις, μήν, θειλός... | 7.7E |
| 428 | SOSICRATES RHODIUS | [ἑτελεύτησε, δ,', ἐτῶν, ἑβδομήκοντα, ὀκτώ, ἤ,... | [τελευτάω, δ,', ἑτάζω, ἑβδομήκοντα, ὀκτώ, ἤ, ... | 0.516588 | 633 | [(ὁ, 40), (καί, 33), (δέ, 18), (, 15), (σωσικ... | [ἀρτάω, βεβοήθηκά, μόνος, κρήτης·, βοηθέω, αἰρ... | 14.94 |
| 424 | SOCRATES RHODIUS | [σωκράτης, δέ, ὁ, ῥόδιος, ἐν, τρίτῳ, ἐμφυλίου,... | [σωκράτης, δέ, ὁ, ῥόδιος, ἐν, τρίτος, ἐμφύλιος... | 0.624549 | 277 | [(ὁ, 28), (καί, 17), (, 9), (δέ, 7), (ἐν, 6),... | [ὕλη, ἐξαρτάω, ἀλουργής, θεάω, τέγος, μετά, πυ... | 10.62 |
| 399 | PROXENUS | [ποιμήν, νέμων, πρόβατα, ἐν, τοῖς, τῆς, δωδώνη... | [ποιμήν, νέμω, πρόβατον, ἐν, ὁ, ὁ, δωδώνη, ἔλο... | 0.612676 | 284 | [(ὁ, 30), (καί, 10), (δέ, 10), (οὗτος, 6), (πρ... | [γεννᾷ·, πάχνη, μετά, δ, ὑμάρου, νύκτωρ·, δειμ... | 10.37 |
| 136 | CLAUDIUS IOLAUS | [ἄκη, πόλις, φοινίκης, ..., κλαύδιος, δέ, ἰουλ... | [ἀκέω, πόλις, φοινίκη, ..., κλαύδιος, δέ, ἰουλ... | 0.595238 | 336 | [(ὁ, 28), (δέ, 10), (καί, 10), (πόλις, 6), (κα... | [ἀρκέω, ἀγαμέμνων, γόνιμος, μετά, πάντως, ἑλκό... | 9.75 |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 270 | HIPPOSTRATUS | [ἱππόστρατος, δέ, φησιν, ἐν, τῷ, περί, μίνω, α... | [ἱππόστρατος, δέ, φημί, ἐν, ὁ, περί, μίνως, αἱ... | 0.529595 | 321 | [(ὁ, 21), (δέ, 12), (καί, 12), (ἱππόστρατος, 7... | [ποίησις, θήρων, κάμικος, μετά, ἀπόλλων1, ὑπέρ,... | 8.32 |
| 334 | NICOLAUS DAMASCENUS | [ἀντίπατρος, ἦν, νικολάου, τοῦ, δαμασκηνοῦ, πα... | [ἀντίπατρος, εἰμί, νικόλαος, ὁ, δαμασκηνόν, πα... | 0.164711 | 35887 | [(ὁ, 2626), (καί, 1959), (δέ, 917), (, 704), ... | [διαπρέπω, τρίς, φλέγω, σκώπτω, τοὔνομά, ἄνομο... | 534.13 |
| 393 | PRISCUS PANITES | [πρίσκος, πανίτης, σοφιστής, γεγονώς, ἐπί, τῶν... | [πρίσκος, πανίτης, σοφιστής, γεγονώς, ἐπί, ὁ, ... | 0.223204 | 7809 | [(ὁ, 754), (καί, 396), (δέ, 194), (τήν, 169), ... | [διαπρέπω, στρατοπεδεύω, ἐπισκοπήν, διεφέροντο... | 165.59 |
| 279 | JOANNES ANTIOCHENUS | [1, ἀπό, τῆς, ἐκθέσεως, ἰωάννου, ἀντιοχέως, τῆ... | [1, ἀπό, ὁ, ἔκθεσις, ἰώαννης, ἀντιοχέως, ὁ, πε... | 0.161421 | 47305 | [(ὁ, 4611), (καί, 2627), (δέ, 1156), (τήν, 904... | [διαπρέπω, ἐπιρρεπής, ἀκρίς, ἀκρωτηριαζομένους... | 810.78 |
| 304 | MENANDER PROTECTOR | [μένανδρος, προτίκτωρ, ἱστορικός, ὅς, λέγει, π... | [μένανδρος, προτίκτωρ, ἱστορικός, ὅς, λέγω1, π... | 0.170874 | 22736 | [(ὁ, 1980), (καί, 1162), (τε, 351), (δέ, 341),... | [κωλύω, παρεγγυηθέν, ἀναίσχυντος, κώπη, βαίανο... | 405.44 |

Difficulty tends to scale with the length of an author's corpus, but density does not. That is, writers gonna write (*tenet insanabile multos scribendi cacoethes*, after all), but rare words are rare, so even long texts can only be so "dense." Turns out maybe reading Pricus' prolix and feeble declamation isn't so tough.