Lecture 13: Supercomputing, Computational Efficiency

LING 1340/2340: Data Science for Linguists

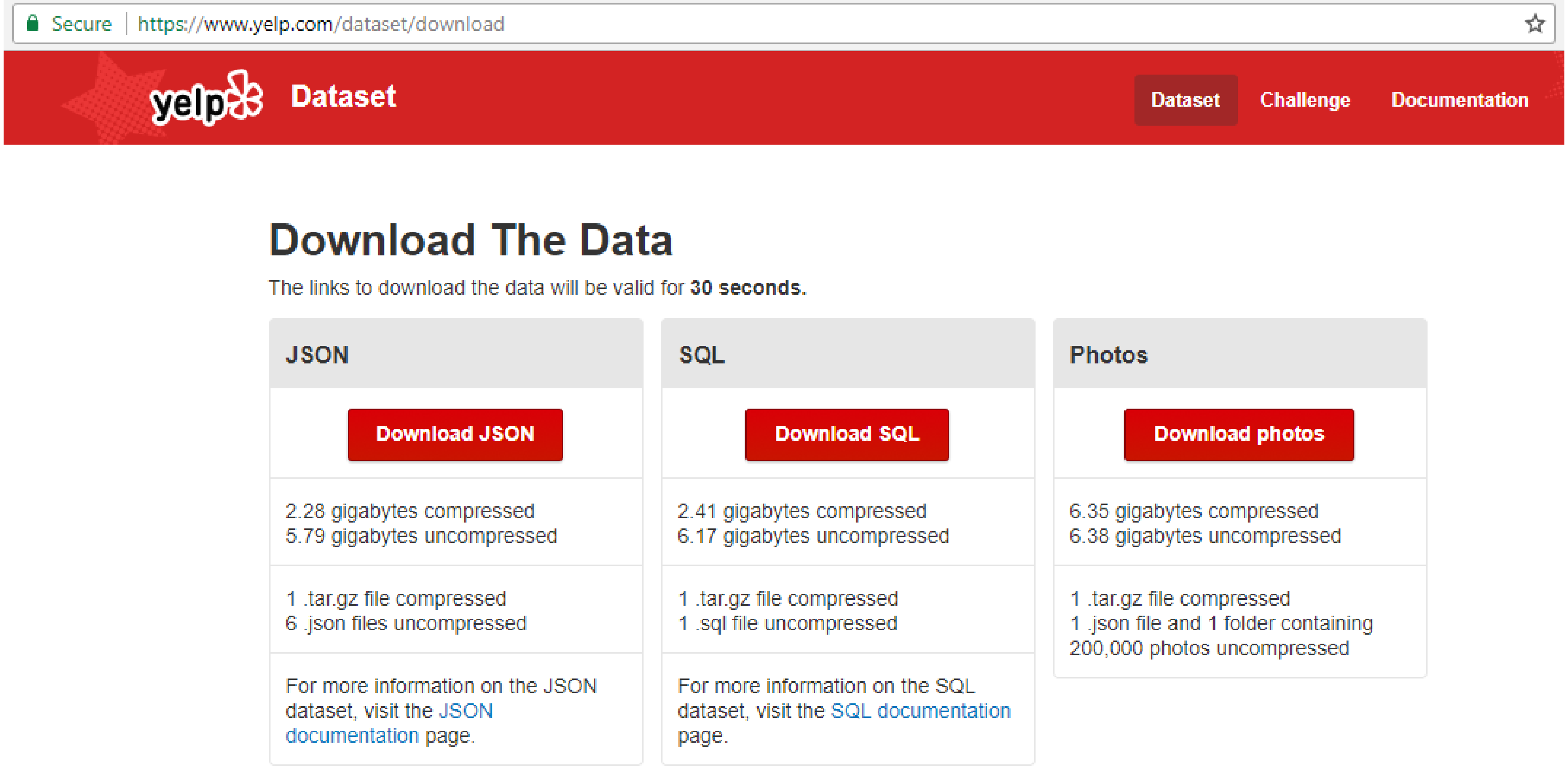
Na-Rae Han

Objectives

- Supercomputing
- Big data considerations
- Computational efficiency

The Yelp Dataset Challenge

- https://www.yelp.com/dataset/challenge

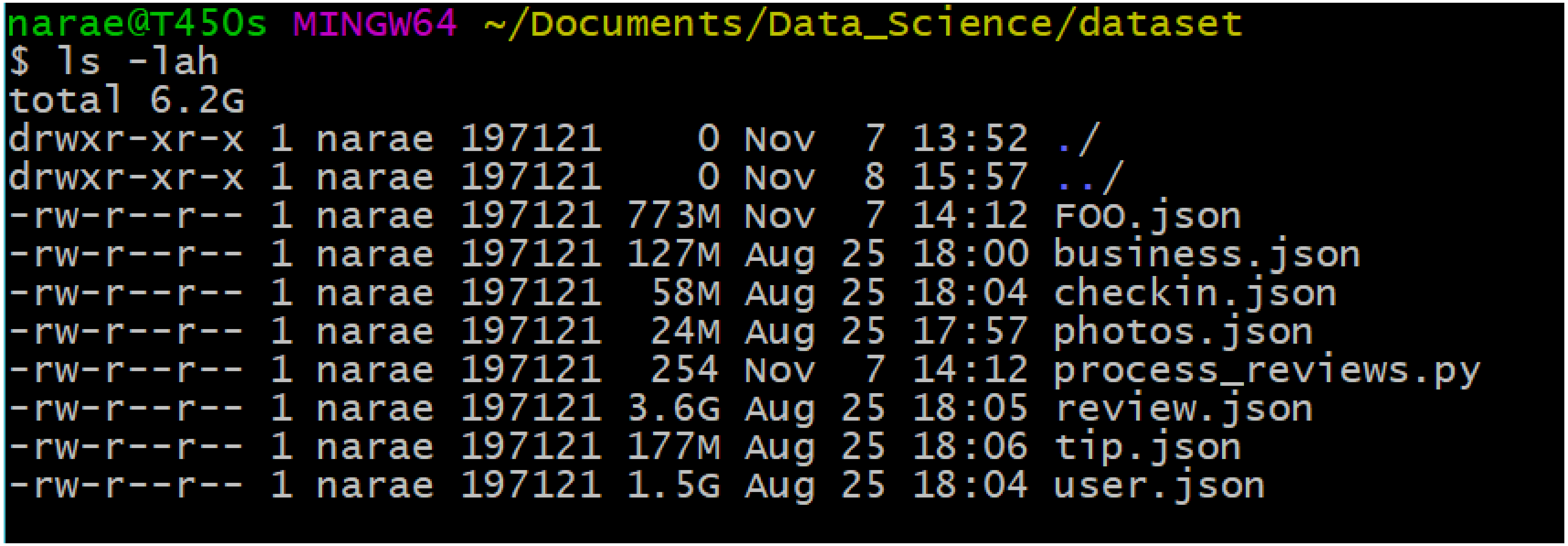


Download The Data

The links to download the data will be valid for **30 seconds.**

JSON	SQL	Photos
Download JSON	Download SQL	Download photos
2.28 gigabytes compressed 5.79 gigabytes uncompressed	2.41 gigabytes compressed 6.17 gigabytes uncompressed	6.35 gigabytes compressed 6.38 gigabytes uncompressed
1 .tar.gz file compressed 6 .json files uncompressed	1 .tar.gz file compressed 1 .sql file uncompressed	1 .tar.gz file compressed 1 .json file and 1 folder containing 200,000 photos uncompressed
For more information on the JSON dataset, visit the JSON documentation page.	For more information on the SQL dataset, visit the SQL documentation page.	

Working with big data files

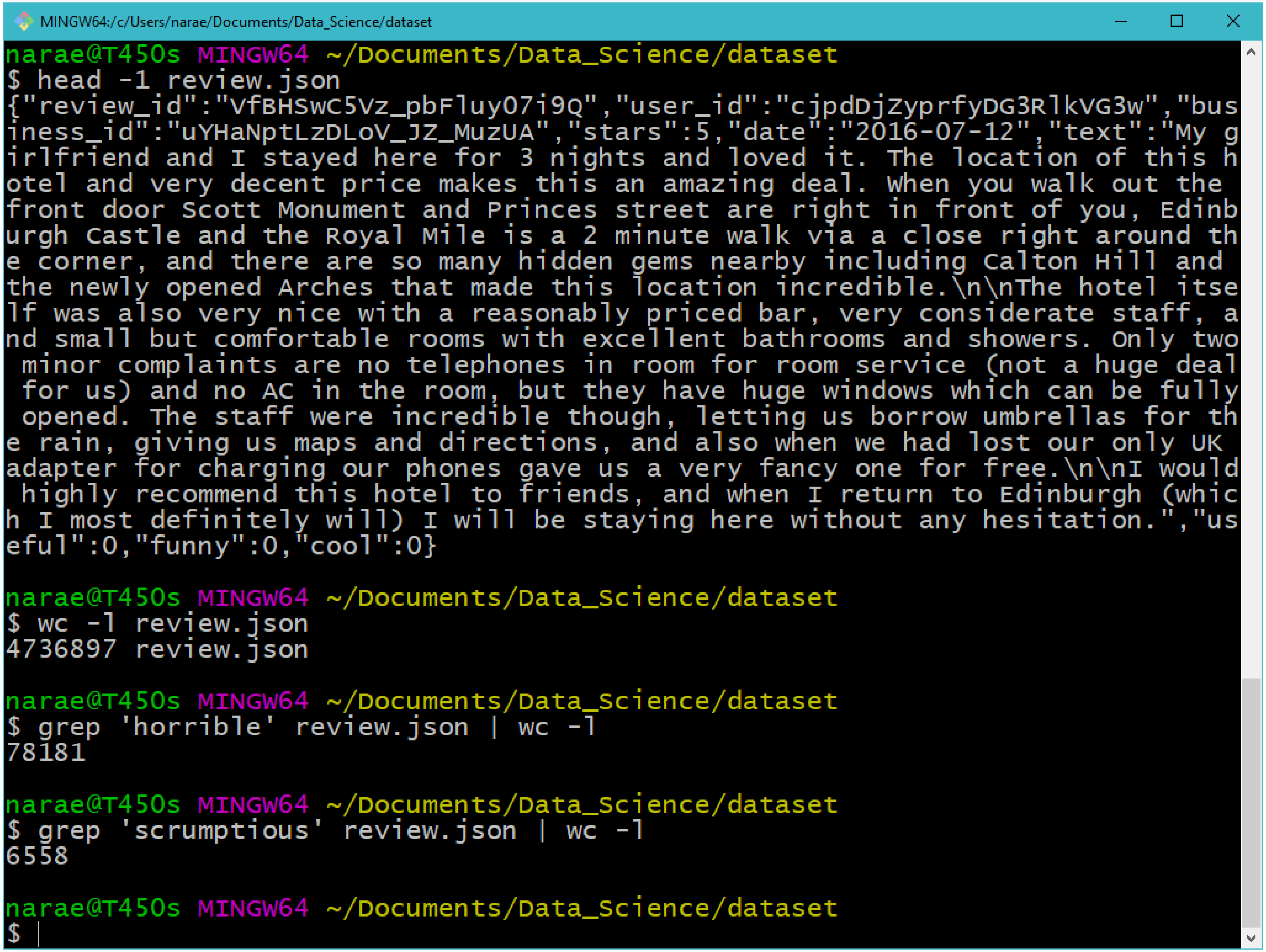


```
narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ ls -lah
total 6.2G
drwxr-xr-x 1 narae 197121    0 Nov  7 13:52 ./
drwxr-xr-x 1 narae 197121    0 Nov  8 15:57 ../
-rw-r--r-- 1 narae 197121 773M Nov  7 14:12 FOO.json
-rw-r--r-- 1 narae 197121 127M Aug 25 18:00 business.json
-rw-r--r-- 1 narae 197121  58M Aug 25 18:04 checkin.json
-rw-r--r-- 1 narae 197121  24M Aug 25 17:57 photos.json
-rw-r--r-- 1 narae 197121  254 Nov  7 14:12 process_reviews.py
-rw-r--r-- 1 narae 197121 3.6G Aug 25 18:05 review.json
-rw-r--r-- 1 narae 197121 177M Aug 25 18:06 tip.json
-rw-r--r-- 1 narae 197121 1.5G Aug 25 18:04 user.json
```

- Each file is in JSON format, and they are huge:
  - review.json is 3.6GB.
  - user.json is 1.5GB.
  - ← Too big to open in most text editors (Notepad++ couldn't.)
  - ← How to explore them?

    In command line. head/tail, grep and regular expression-based searching.

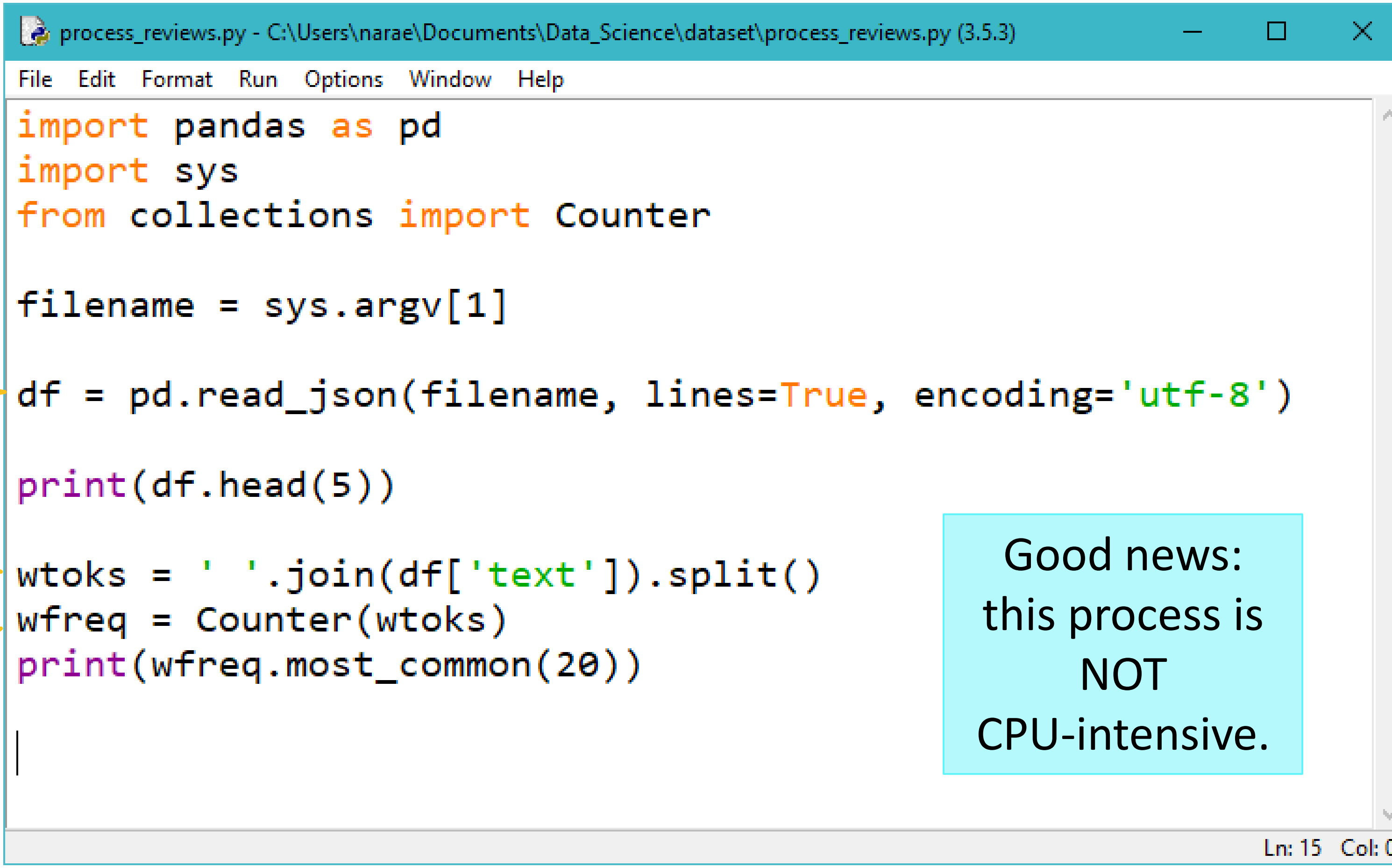
Command line exploration



```
narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ head -1 review.json
{"review_id":"VfBHSwC5Vz_pbFluy07i9Q","user_id":"cjpdDjZyprfyDG3RlkVG3w","bus
iness_id":"uYHaNptLzDLoV_JZ_MuzUA","stars":5,"date":"2016-07-12","text":"My g
irlfriend and I stayed here for 3 nights and loved it. The location of this h
otel and very decent price makes this an amazing deal. When you walk out the
front door Scott Monument and Princes street are right in front of you, Edinb
urgh Castle and the Royal Mile is a 2 minute walk via a close right around th
e corner, and there are so many hidden gems nearby including Calton Hill and
the newly opened Arches that made this location incredible.\n\nThe hotel itse
lf was also very nice with a reasonably priced bar, very considerate staff, a
nd small but comfortable rooms with excellent bathrooms and showers. Only two
 minor complaints are no telephones in room for room service (not a huge deal
 for us) and no AC in the room, but they have huge windows which can be fully
 opened. The staff were incredible though, letting us borrow umbrellas for th
e rain, giving us maps and directions, and also when we had lost our only UK
adapter for charging our phones gave us a very fancy one for free.\n\nI would
 highly recommend this hotel to friends, and when I return to Edinburgh (whic
h I most definitely will) I will be staying here without any hesitation.","us
eful":0,"funny":0,"cool":0}

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ wc -l review.json
4736897 review.json

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ grep 'horrible' review.json | wc -l
78181

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ grep 'scrumptious' review.json | wc -l
6558

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ |
```

Opening + processing big files

- How much resource does it take to process review.json file (3.6GB)?


process_reviews.py - C:\Users\narae\Documents\Data_Science\dataset\process_reviews.py (3.5.3)
File Edit Format Run Options Window Help
Ln: 15 Col: 0

```
import pandas as pd
import sys
from collections import Counter

filename = sys.argv[1]

df = pd.read_json(filename, lines=True, encoding='utf-8')

print(df.head(5))

wtoks = ' '.join(df['text']).split()
wfreq = Counter(wtoks)
print(wfreq.most_common(20))
```

There's 3.6GB

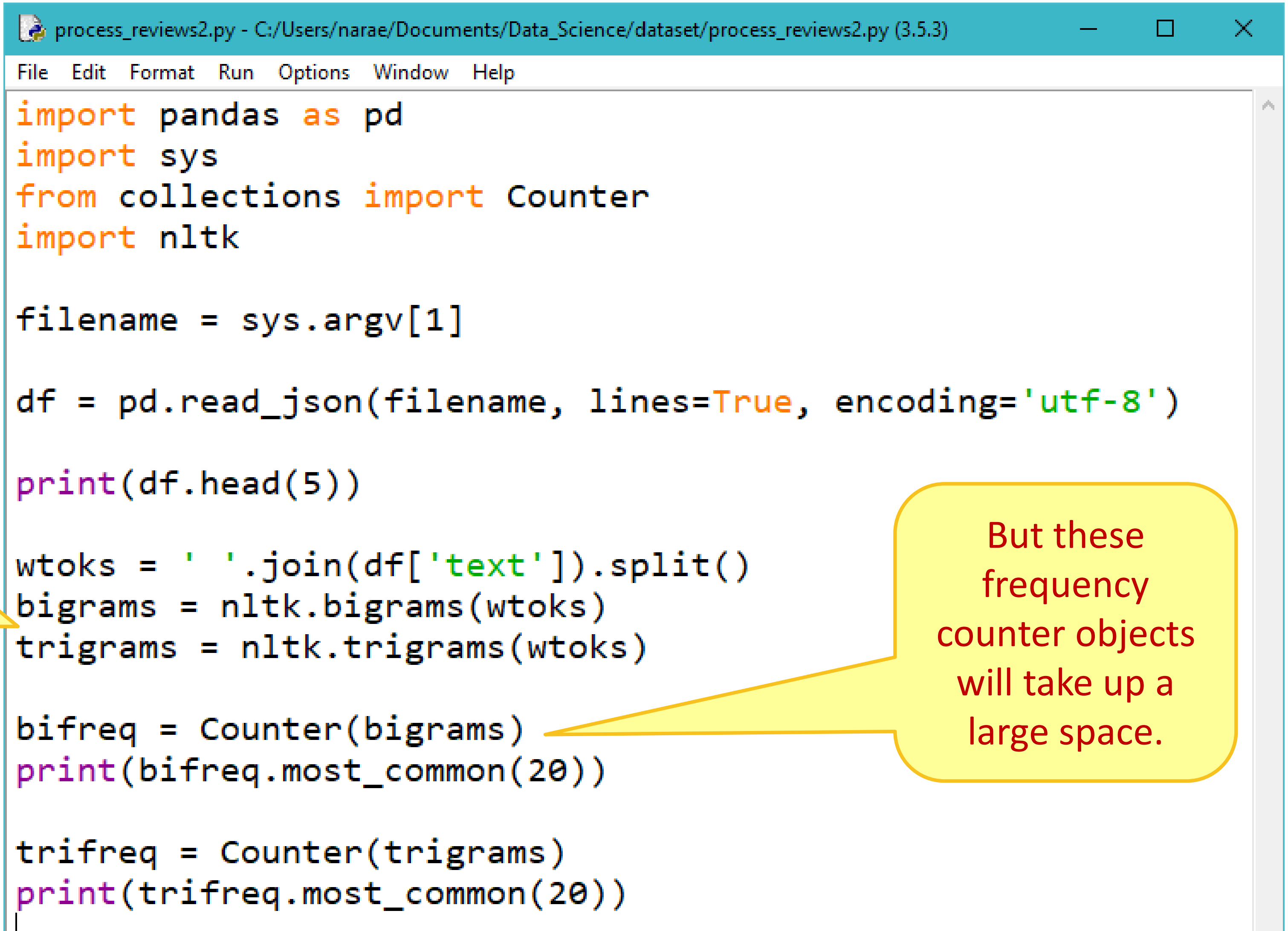
Another ~3GB

Another big object

Good news: this process is NOT CPU-intensive.

Memory consideration

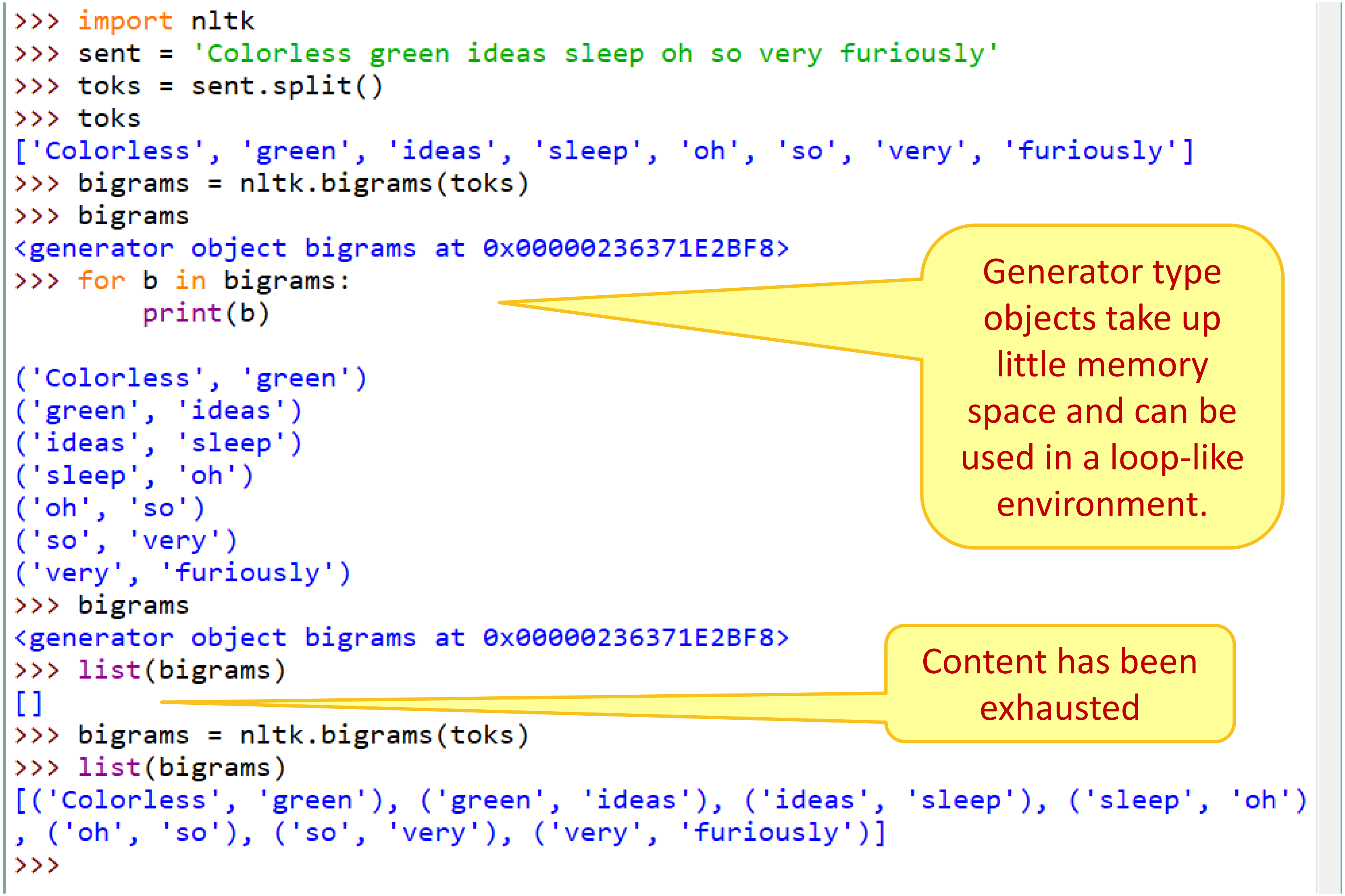
- How much space needed for bigrams? Trigrams?


process_reviews2.py - C:/Users/narae/Documents/Data_Science/dataset/process_reviews2.py (3.5.3)
File Edit Format Run Options Window Help

```
import pandas as pd
import sys
from collections import Counter
import nltk

filename = sys.argv[1]

df = pd.read_json(filename, lines=True, encoding='utf-8')

print(df.head(5))

wtoks = ' '.join(df['text']).split()
bigrams = nltk.bigrams(wtoks)
trigrams = nltk.trigrams(wtoks)

bifreq = Counter(bigrams)
print(bifreq.most_common(20))

trifreq = Counter(trigrams)
print(trifreq.most_common(20))
```

Good news! These are built as *generator* objects.

But these frequency counter objects will take up a large space.



```
>>> import nltk
>>> sent = 'Colorless green ideas sleep oh so very furiously'
>>> toks = sent.split()
>>> toks
['Colorless', 'green', 'ideas', 'sleep', 'oh', 'so', 'very', 'furiously']
>>> bigrams = nltk.bigrams(toks)
>>> bigrams
<generator object bigrams at 0x00000236371E2BF8>
>>> for b in bigrams:
        print(b)

('Colorless', 'green')
('green', 'ideas')
('ideas', 'sleep')
('sleep', 'oh')
('oh', 'so')
('so', 'very')
('very', 'furiously')
>>> bigrams
<generator object bigrams at 0x00000236371E2BF8>
>>> list(bigrams)
[]
>>> bigrams = nltk.bigrams(toks)
>>> list(bigrams)
[('Colorless', 'green'), ('green', 'ideas'), ('ideas', 'sleep'), ('sleep', 'oh')
, ('oh', 'so'), ('so', 'very'), ('very', 'furiously')]
>>>
```

Generator type objects take up little memory space and can be used in a loop-like environment.

Content has been exhausted

File opening & closing methods

```
f = open('review.json')
lines = f.readlines()
for l in lines:
    if 'horrible' in l:
        print(l)
f.close()
```

Which methods are more memory-efficient?

```
lines = open('review.json').readlines()
for l in lines :
    if 'horrible' in l:
        print(l)
```

Python will close up this file handle.

```
f = open('review.json')
for l in f:
    if 'horrible' in l:
        print(l)
f.close()
```

```
with open('review.json') as f:
    for l in f:
        if 'horrible' in l:
            print(l)
```

No need to close f.
Some folks swear by using `with`.

Handling files in chunks

```
f = open('review.json')
lines1 = f.readlines(1000000000)
lines2 = f.readlines(1000000000)
lines3 = f.readlines(1000000000)
lines4 = f.readlines()
f.close()
```

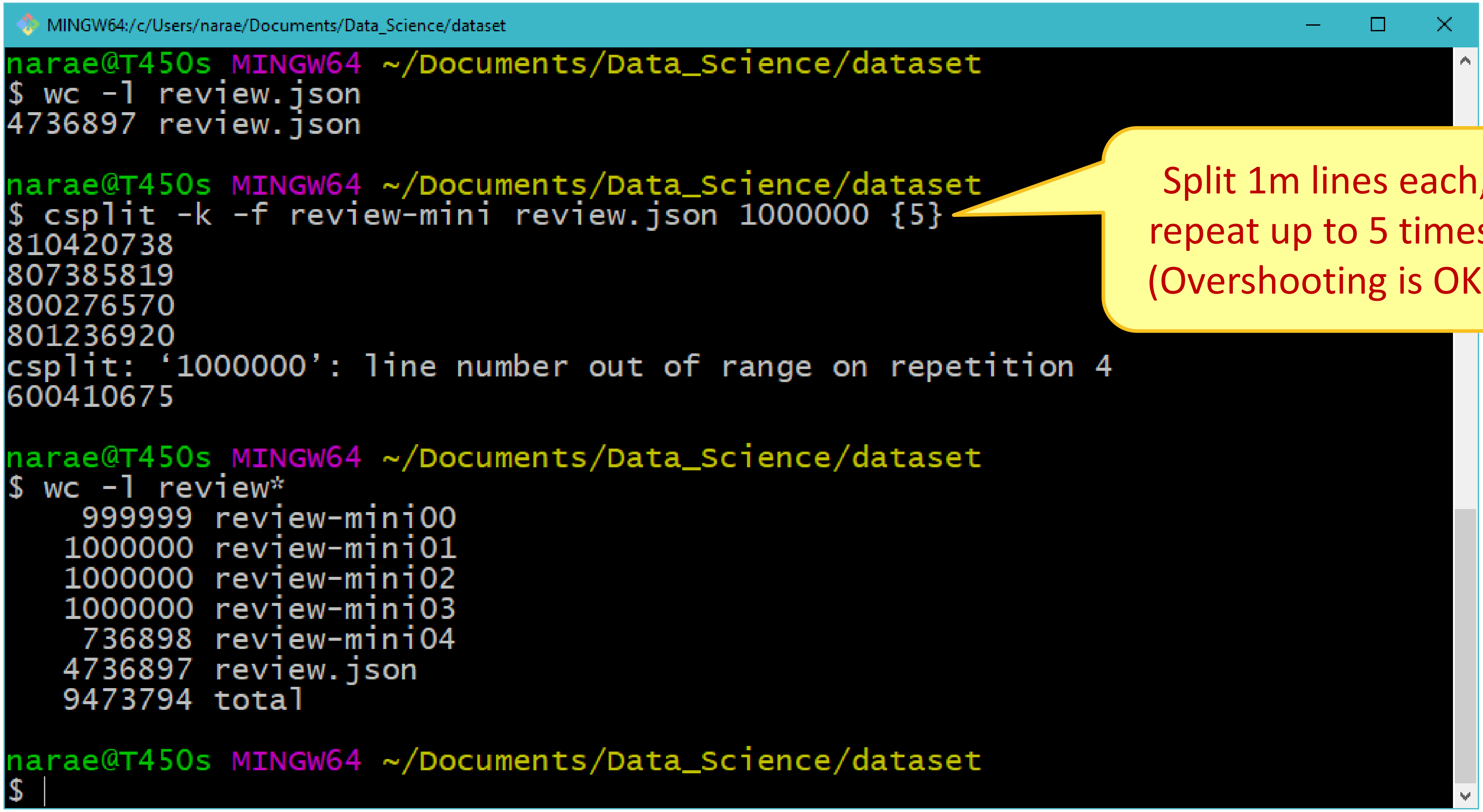
Optional # of bytes to read. (But! Not doing it through loop like this does not offer memory advantage.)

```
dfs = pd.read_json('review.json', lines=True, chunksize=10000, encoding='utf8')

wfreq = Counter()
for df in dfs:
    wtoks = ' '.join(df['text']).split()
    temp = Counter(wtoks)
    wfreq.update(temp)

print(wfreq.most_common(20))
```

`chunksize` optional parameter in pandas' `read_json` method reads in 10,000 lines at a time. Then, iterate through each small df.

Breaking up large files

- csplit splits up large files into smaller chunks with equal line counts.

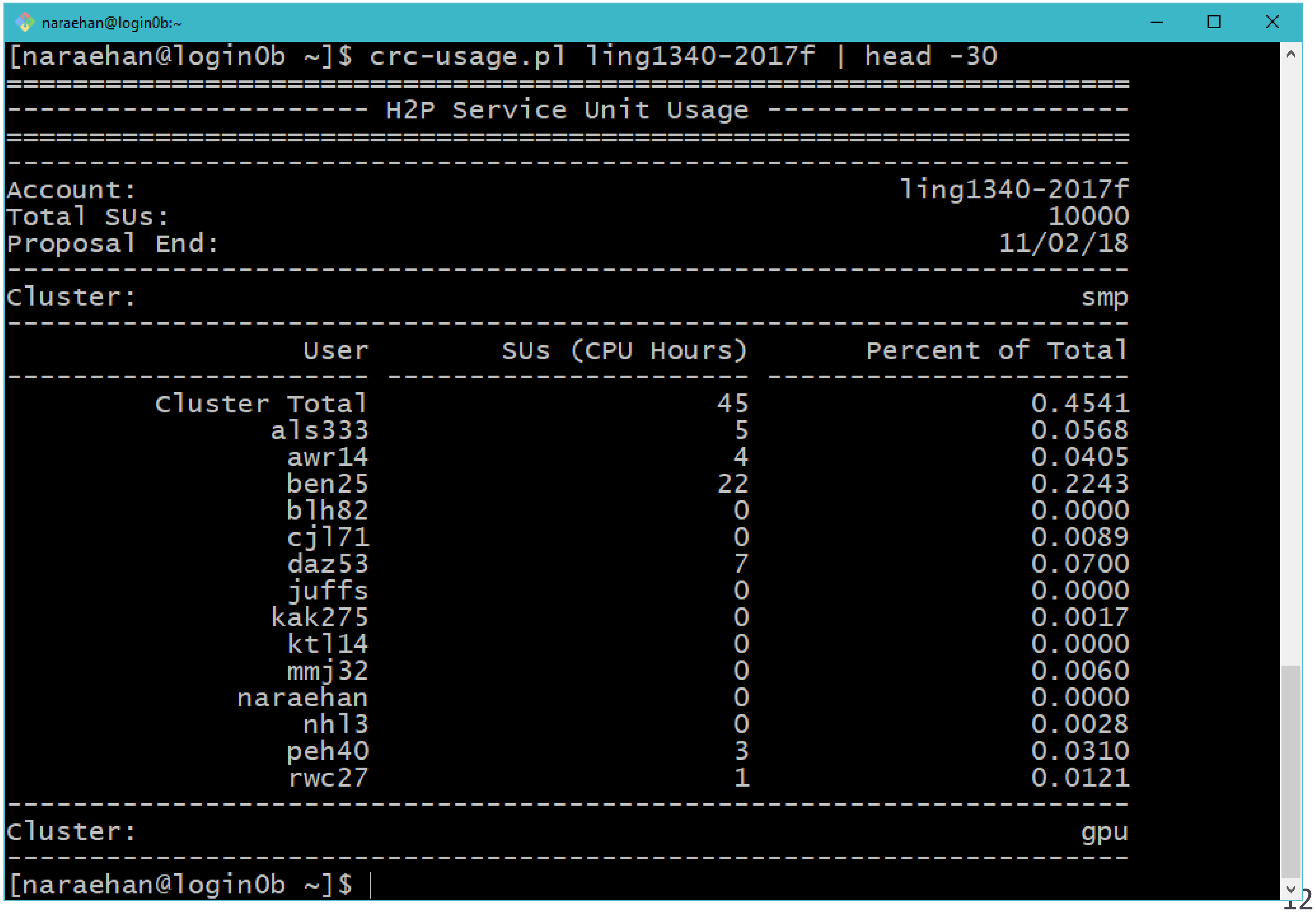


```
narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ wc -l review.json
4736897 review.json

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ csplit -k -f review-mini review.json 1000000 {5}
810420738
807385819
800276570
801236920
csplit: ‘1000000’: line number out of range on repetition 4
600410675

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ wc -l review*
    999999 review-mini00
   1000000 review-mini01
   1000000 review-mini02
   1000000 review-mini03
    736898 review-mini04
   4736897 review.json
   9473794 total

narae@T450s MINGW64 ~/Documents/Data_Science/dataset
$ 
```

Split 1m lines each, repeat up to 5 times. (Overshooting is OK.)

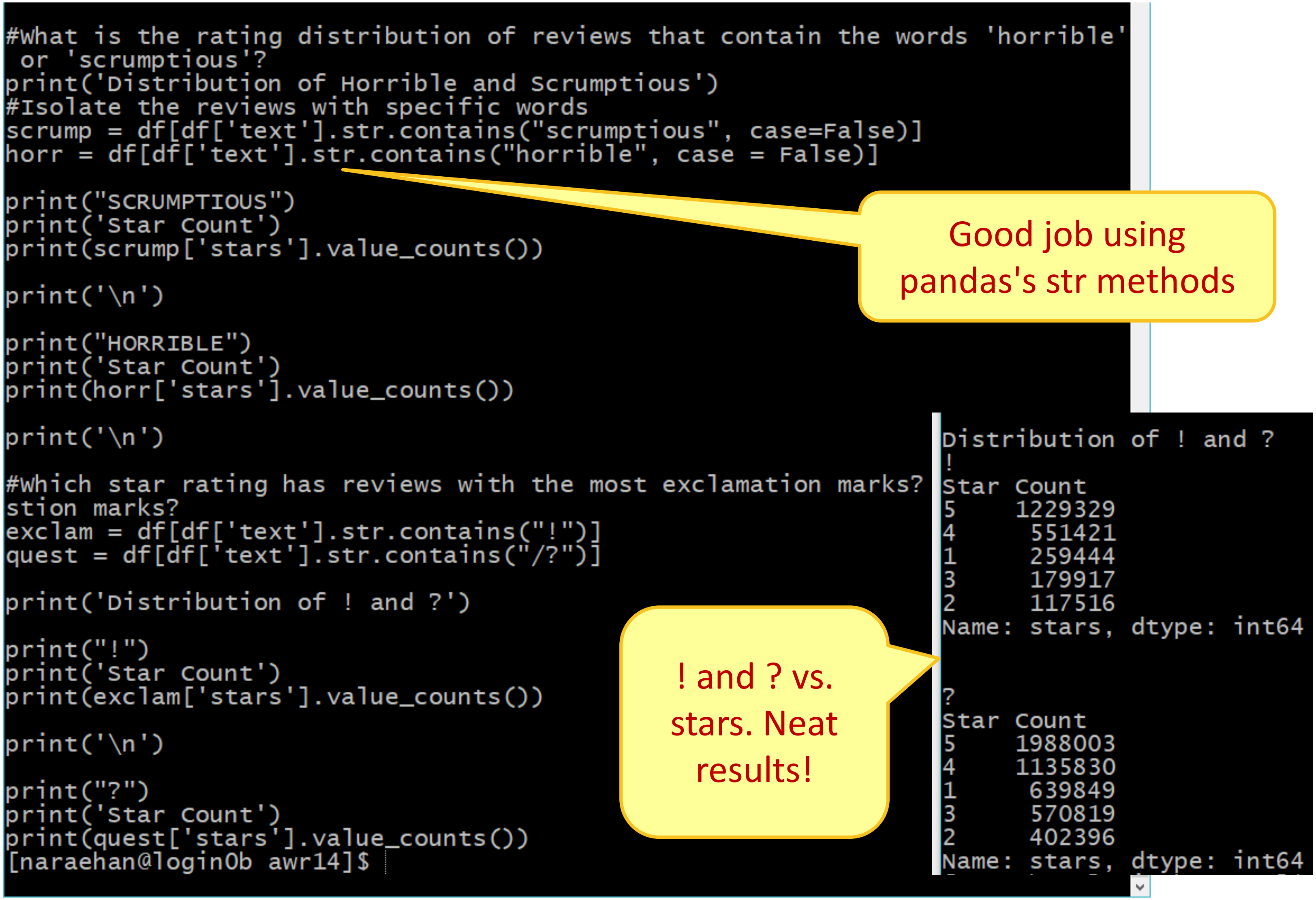
Supercomputing: what did you learn?

- All right! 45 SUs out of 10,000!



```
[naraehan@login0b ~]$ crc-usage.pl ling1340-2017f | head -30
====================================================================
---------------------- H2P Service Unit Usage ----------------------
====================================================================
--------------------------------------------------------------------
Account:                                              ling1340-2017f
Total SUs:                                                     10000
Proposal End:                                               11/02/18
--------------------------------------------------------------------
Cluster:                                                         smp
--------------------------------------------------------------------
                  User        SUs (CPU Hours)       Percent of Total
---------------------- ---------------------- ----------------------
         Cluster Total                     45                 0.4541
                als333                      5                 0.0568
                 awr14                      4                 0.0405
                 ben25                     22                 0.2243
                 blh82                      0                 0.0000
                 cjl71                      0                 0.0089
                 daz53                      7                 0.0700
                 juffs                      0                 0.0000
                kak275                      0                 0.0017
                 ktl14                      0                 0.0000
                 mmj32                      0                 0.0060
              naraehan                      0                 0.0000
                  nhl3                      0                 0.0028
                 peh40                      3                 0.0310
                 rwc27                      1                 0.0121
--------------------------------------------------------------------
Cluster:                                                         gpu
--------------------------------------------------------------------
[naraehan@login0b ~]$
```

Your code examples: Andrew



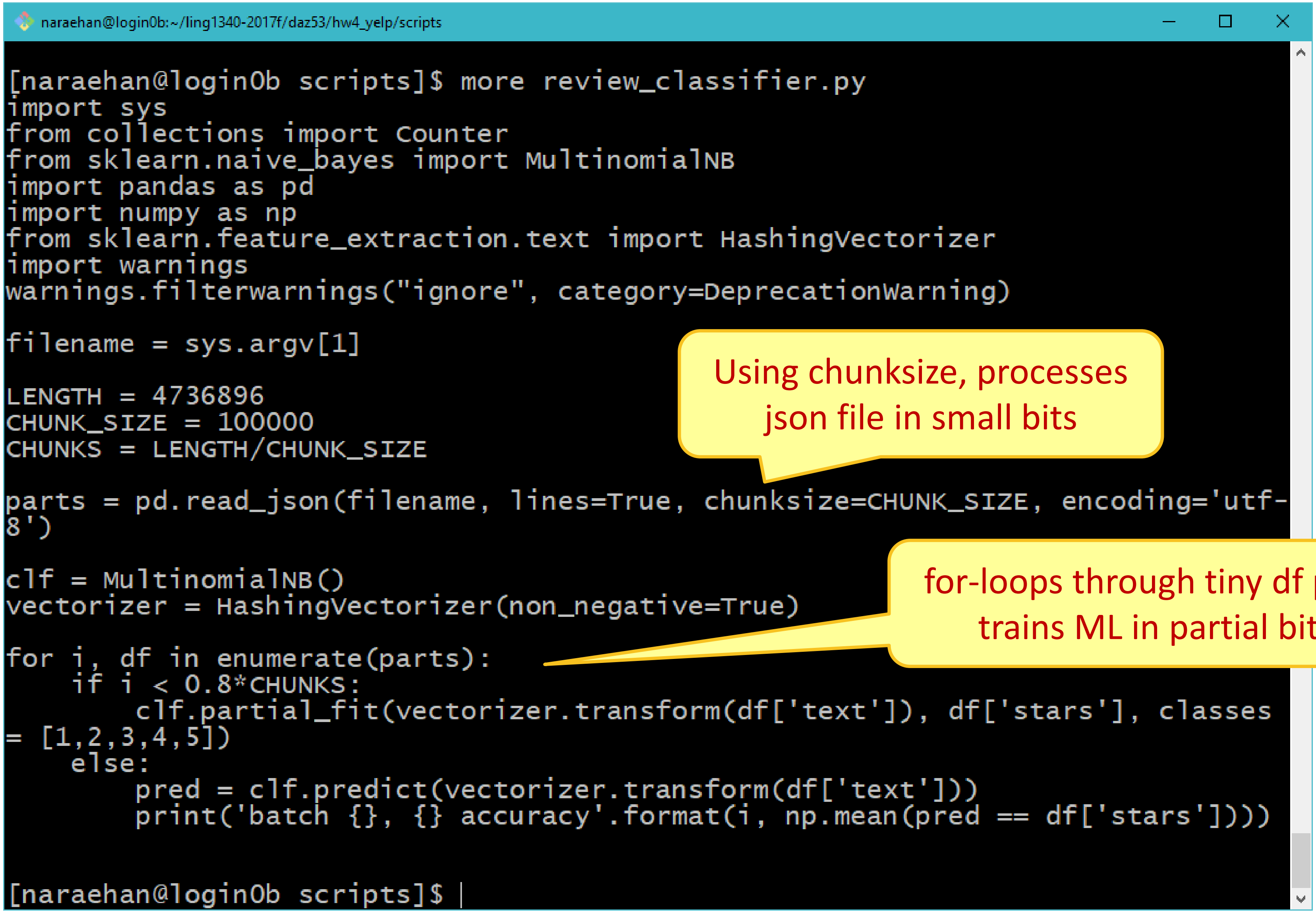
```
#What is the rating distribution of reviews that contain the words 'horrible'
 or 'scrumptious'?
print('Distribution of Horrible and Scrumptious')
#Isolate the reviews with specific words
scrump = df[df['text'].str.contains("scrumptious", case=False)]
horr = df[df['text'].str.contains("horrible", case = False)]

print("SCRUMPTIOUS")
print('Star Count')
print(scrump['stars'].value_counts())

print('\n')

print("HORRIBLE")
print('Star Count')
print(horr['stars'].value_counts())

print('\n')

#Which star rating has reviews with the most exclamation marks?
stion marks?
exclam = df[df['text'].str.contains("!")]
quest = df[df['text'].str.contains("/?")]

print('Distribution of ! and ?')

print("!")
print('Star Count')
print(exclam['stars'].value_counts())

print('\n')

print("?")
print('Star Count')
print(quest['stars'].value_counts())
[naraehan@login0b awr14]$
```

```
Distribution of ! and ?
!
Star Count
5     1229329
4      551421
1      259444
3      179917
2      117516
Name: stars, dtype: int64


?
Star Count
5     1988003
4     1135830
1      639849
3      570819
2      402396
Name: stars, dtype: int64
```

Good job using pandas's str methods

! and ? vs. stars. Neat results!

Your code examples: Dan

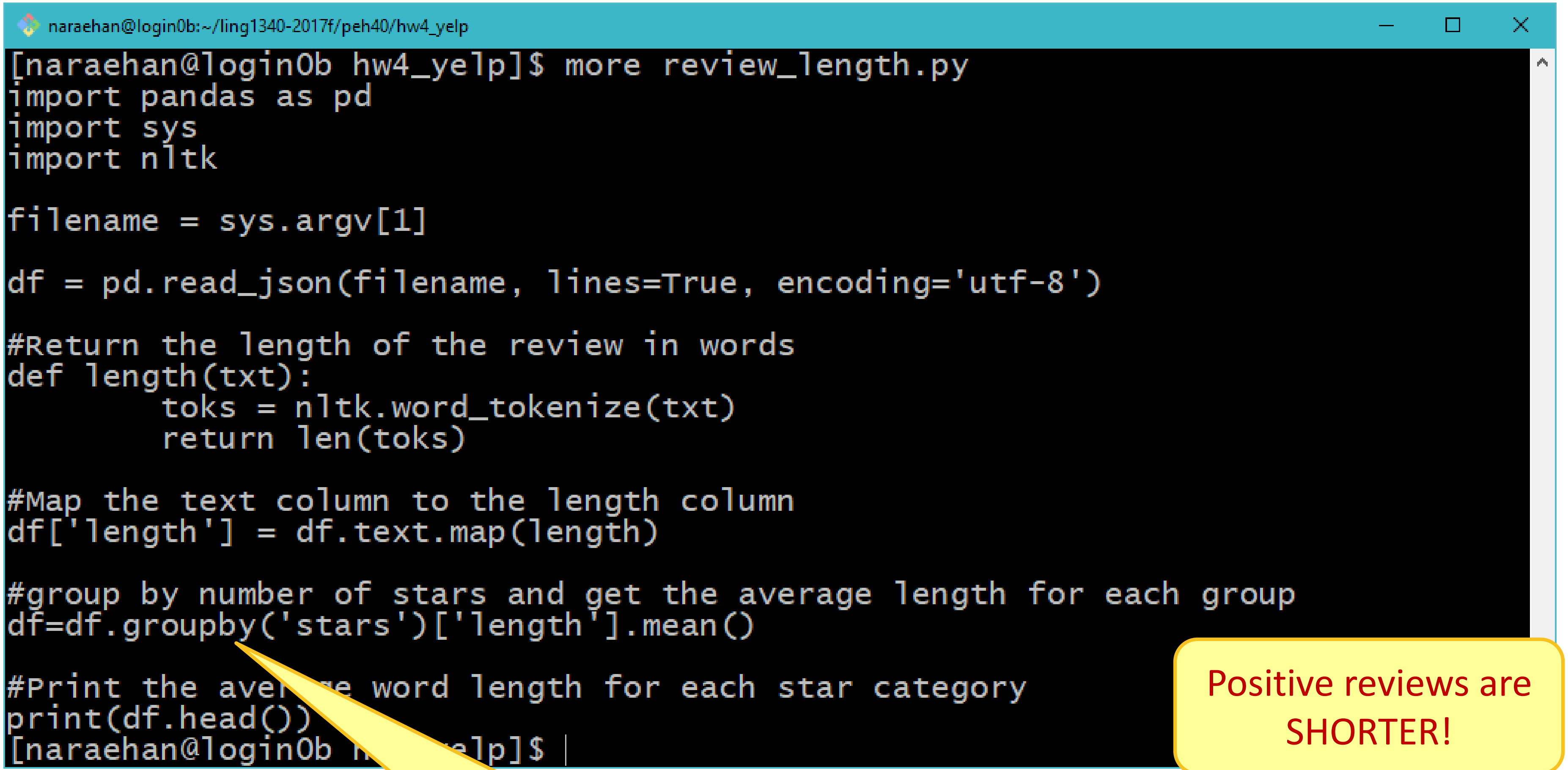


```
[naraehan@login0b scripts]$ more review_classifier.py
import sys
from collections import Counter
from sklearn.naive_bayes import MultinomialNB
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import HashingVectorizer
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

filename = sys.argv[1]

LENGTH = 4736896
CHUNK_SIZE = 100000
CHUNKS = LENGTH/CHUNK_SIZE

parts = pd.read_json(filename, lines=True, chunksize=CHUNK_SIZE, encoding='utf-
8')

clf = MultinomialNB()
vectorizer = HashingVectorizer(non_negative=True)

for i, df in enumerate(parts):
    if i < 0.8*CHUNKS:
        clf.partial_fit(vectorizer.transform(df['text']), df['stars'], classes
= [1,2,3,4,5])
    else:
        pred = clf.predict(vectorizer.transform(df['text']))
        print('batch {}, {} accuracy'.format(i, np.mean(pred == df['stars'])))


[naraehan@login0b scripts]$ |
```

Using chunksize, processes json file in small bits

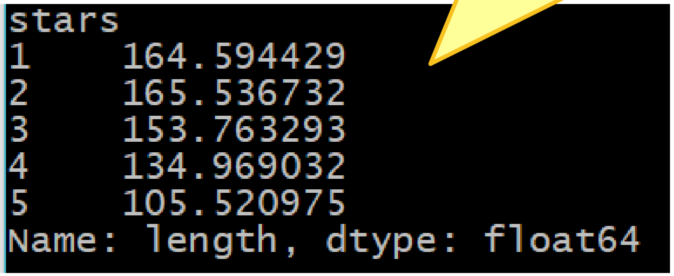
for-loops through tiny df parts, trains ML in partial bits!

Your code examples: Paige



```
[naraehan@login0b hw4_yelp]$ more review_length.py
import pandas as pd
import sys
import nltk

filename = sys.argv[1]

df = pd.read_json(filename, lines=True, encoding='utf-8')

#Return the length of the review in words
def length(txt):
        toks = nltk.word_tokenize(txt)
        return len(toks)

#Map the text column to the length column
df['length'] = df.text.map(length)

#group by number of stars and get the average length for each group
df=df.groupby('stars')['length'].mean()

#Print the aver  ge word length for each star category
print(df.head())
[naraehan@login0b h     elp]$
```

Positive reviews are SHORTER!



```
stars
1    164.594429
2    165.536732
3    153.763293
4    134.969032
5    105.520975
Name: length, dtype: float64
```

df.groupby()

is the way to go!!!

Wrapping up

- To-Do 12
  - Visit your classmates' projects.
- Work on your term project!
  - Come see me.
- Presentation schedule