STAT 243 Problem Set 1

Treves Li

2024-09-12

Collaboration Statement

I did not collaborate with anyone.

Question 1

> Please read these lecture notes about how computers work, used in a class on statistical computing at CMU. Briefly (a few sentences) describe the difference between disk and memory based on that reference and/or other resources you find.

Memory refers to RAM, which has the benefit of being quickly accessed although at the cost of volatility. It needs a constant power supply; therefore storage is temporary.

Disk refers to larger storage, but is slower to access. It can retain data with no power supply, and is therefore used for long-term storage.

Their strengths and downsides seem to be a function of physical constraints. In short, memory is fast access, disk is slower access.

Question 2

This problem uses the ideas and tools in Unit 2, Sections 1-3 to explore approaches to reading and writing data from files and to consider file sizes in ASCII plain text vs. binary formats in light of the fact that numbers are (generally) stored as 8 bytes per number in binary formats.

Question 2a

Generate a numpy array (named x) of random numbers from a standard normal distribution with 20 columns and as many rows as needed so that the data take up about 16 MB (megabytes) in size. As part of your answer, show the arithmetic (formatted using LaTeX math syntax) you did to determine the number of rows.

The strategy here is to look for a function that could generate random numbers from a standard normal distribution. I then need to know the size of numbers in binary format (I spent too long googling this only to find it was already provided: 8 bytes.) Then it is just a matter of dividing the size cap by the number of columns and bytes in each value.

However, I'm not sure whether I should be converting MB to bytes by multiplying 1,000,000 or 1,048,576 ...

```python
import numpy as np
from sys import getsizeof

size_cap = 16 * 1000000 # Convert from 16 MB to bytes
element_size = 8 # 8 bytes in float64 outputted by np.random.normal

num_cols = 20   # per question requirements
num_rows = size_cap // num_cols // element_size

x = np.random.normal(loc=0, scale=1.0, size=(num_rows, num_cols))
print(f"The array has {num_rows} rows and {num_cols} columns.")
print("The size of x is:", getsizeof(x) / 1000000, "MB.")
```

```
The array has 100000 rows and 20 columns.
The size of x is: 16.000128 MB.
```

To determine the number of rows:

number of rows = data size limit [in bytes] ÷ number of columns [given] ÷ float64 element size [in bytes]

$$\text{number of rows} = (16 \times 1,000,000) \div 20 \div 8$$

Question 2b

> Explain the sizes of the two files created below. In discussing the CSV text file, how many characters do you expect to be in the file (i.e., you should be able to estimate this reasonably accurately from first principles without using wc or any explicit program that counts characters). Hint: what do we know about numbers drawn from a standard normal distribution?

```
import os
import pandas as pd
x = x.round(decimals = 12)

pd.DataFrame(x).to_csv('x.csv', header = False, index = False)
print(f"{str(os.path.getsize('x.csv')/1e6)} MB")

pd.DataFrame(x).to_pickle('x.pkl', compression = None)
print(f"{str(os.path.getsize('x.pkl')/1e6)} MB")
```

```
30.776744 MB
16.000573 MB
```

> Suppose we had rounded each number to four decimal places. Would using CSV have saved disk space relative to the pickle file?

The data exported as a pickle file is smaller than the CSV because pickle is a binary format, and binary formats can be more efficient than text formats (due to the way binaries store, structure, and allow access to data).

To estimate the size of the CSV text file, we previously specified a mean of 0 and a standard deviation of 1 (i.e., `loc=0` and `scale=1.0` in `np.random.normal`). In a normal distribution, most numbers will thus fall within three standard deviations, i.e., from -3 to 3.

With 12 decimal places and a decimal point, a typical value will thus have 14.5 characters (the number is not round since we need to account for negative signs). Add another character to each value to account for the comma delimiter, bringing us to 15.5. We know from Question 2a that there are 100000 rows and 20 columns, so 15.5 x 100000 x 20 = 31 million characters.

To see if rounding each number would make a difference, we re-run the code:

```
x_4dec = x.round(decimals = 4)

pd.DataFrame(x_4dec).to_csv('x_4dec.csv', header = False, index = False)
print(f"{str(os.path.getsize('x_4dec.csv')/1e6)} MB")
```

```
pd.DataFrame(x_4dec).to_pickle('x_4dec.pkl', compression = None)
print(f"{str(os.path.getsize('x_4dec.pkl')/1e6)} MB")
```

```
14.776787 MB
16.000573 MB
```

The answer above shows that CSV **would** have saved disk space compared to pickle file with sufficient rouding of values.

Question 2c

> Now consider saving out the numbers one number per row in a CSV file. Given we no longer have to save all the commas, why is the file size unchanged?

We've now introduced a "hidden" new line character, which takes the place of the commas that were removed. So there is no net difference in file size.

Question 2d

> Read the CSV file into Python using `pandas.read_csv`. Compare the speed of reading the CSV to reading the pickle file with `pandas.read_pickle`. Note that in some cases you might find that the first time you read a file is slower; if so this has to do with the operating system caching the file in memory (we'll discuss this further in Unit 7 when we talk about databases).

Use the `time` library to compare how long it takes to run each process.

```python
import time

# pd.read_csv
t0 = time.time()
pd.read_csv('x.csv')
print(f"read_csv took {time.time() - t0} seconds")

# read_pickle
t0 = time.time()
pd.read_pickle('x.pkl')
print(f"read_pickle took {time.time() - t0} seconds")
```

```
read_csv took 0.22459840774536133 seconds
read_pickle took 0.003863811492919922 seconds
```

Question 2e

> Finally, in the next parts of the question, we'll consider reading the CSV file in chunks as discussed in Unit 2. First, time how long it takes to read the first 10,000 rows in a single chunk using `nrows`.

```
t0 = time.time()
pd.read_csv('x.csv', nrows=10000)
print(f"Reading first 10,000 rows took {time.time() - t0} seconds")
```

```
Reading first 10,000 rows took 0.04174399375915527 seconds
```

Question 2f

> Now experiment with the `skiprows` to see if you can read in a large chunk of data from the middle of the file as quickly as the same size chunk from the start of the file. What does this indicate regarding whether Pandas/Python has to read in all the data up to the point where the chunk in the middle starts or can skip over it in some fashion? Is there any savings relative to reading all the initial rows and the chunk in the middle all at once?

```
t0 = time.time()
pd.read_csv('x.csv', skiprows=45000, nrows=10000)
print(f"Reading a middle chunk of 10,000 rows took {time.time() - t0} seconds")
```

```
Reading a middle chunk of 10,000 rows took 0.14955568313598633 seconds
```

The experiment shows reading the middle chunk took nearly twice as long as reading the initial chunk. This indicates that Pandas/Python probably still has to do some processing of lines before it reaches the middle chunk. This would also mean that the savings of using `skiprows` relative to reading all the initial rows and then the middle chunk all at once would be marginal.

Question 3

> Please read the Code syntax and style section of Unit 4 on good programming/project practices and incorporate what you've learned from that reading into your solution for Problem 4. (You can skip the section on Assertions and Testing, as we'll cover that in Lab.) In particular, lint your code (e.g., using `ruff` or another tool of your choice as discussed in the Unit 4. (This is most straightforward for code in a .py file. If your code is directly in the qmd file, you probably need to copy-paste it into another file to lint it, unfortunately. If anyone figures out a good way to directly lint the code chunks, please post on Ed!)
>
> As your response to this question, very briefly (a few sentences) note what you did in your code for Problem 4 that reflects what you read. Please also note anything in what you read in Unit 4 that you disagree with, if you have a different stylistic perspective.

I used docstrings to explain what my function does, including descriptions of inputs and outputs. Similarly, I put comments explaining each main coding step/chunk.

I used defensive coding to flag any errors, especially when retrieving the html.

I used spaces (and did not use them) where appropriate.

I broke up code into several lines, both to be better able to explain each argument and to ensure that the code didn't bleed over the page when rendered in Quarto.

Finally, i used `ruff` to lint my code, and ensure it follows recommended styling and formatting.

Question 4

We'll experiment with webscraping and manipulating HTML by getting publication information off of Google Scholar. Note that Google Scholar does not have an API, so we are forced to deconstruct the queries that are produced when we point and click on the website.

Your functions may be short enough that it's ok to put a function directly within a code chunk in your qmd file. Or you might choose to put one or more functions into a .py file and use `inspect.getsource()` to show us the code. For this problem in which the focus of the work is the function and how it works, it will generally be best to show the function code as part of the problem solution rather than in an appendix.

Question 4a

Go to Google Scholar and enter the name (including first name to help with disambiguation) for a researcher whose work interests you. (If you want to do the one that will match the problem set solutions, you can use "Michael Jordan", who is a well-known statistician and machine learning researcher here at Berkeley.) If you've entered the name of a researcher that Google Scholar recognizes as having a Google Scholar profile, you should see that the first item shown in the results page is a "User profile". Now, based on the information returned, show the HTML element containing the Google Scholar ID and determine the Google Scholar ID for the researcher. Ideally (see the extra credit part (e)) we would automate that process and write a Python function that returns the ID, but see Question 5 for why that seemingly would violate Google Scholar's terms of use.

I used my web browser's (Firefox) web dev tools.

```
<a href="/citations?user=yxUduqMAAAAJ&amp;hl=en&amp;oi=ao"><b>Michael </b>I. <b>Jordan</b></a>
```

The Google Scholar ID for Michael I. Jordan is thus "yxUduqMAAAAJ".

Question 4b

Create a function that constructs the http GET request (and submits that request) to get the citations for the scholar, taking the ID as the input argument and returning the HTML as a Python object.

IMPORTANT: While running the query in an automated fashion is seemingly allowed (see Problem 5), Google may return "429" errors because it detects automated usage. Here are some things to do in that case:

1. You can try to download the HTML file via the UNIX `curl` command, which you can run within Python as `subprocess.run(["curl", "-L", request_string], capture_output=True)`. If necessary for part (c), you can use `curl` or `wget` from the command line to separately download the file and then read that file into Python.
2. When developing your code, once you have the code in this part of the problem working to download the HTML, use the downloaded HTML to develop the remainder of your code for part (c) and don't keep re-downloading the HTML as you work on the remainder of the code.

For now, you can assume the user will provide a valid ID and that Google Scholar returns a result for the specified person. We'll deal with making the code more robust in PS2.

I had to google usage of BeautifulSoup. I ran into errors trying to open the html as utf-8. The strategy here is to use Python to emulate a Unix `wget` command, then use bs to parse the result.

```python
import subprocess
from bs4 import BeautifulSoup as bs


def get_scholar_citations(scholar_id):
    """Constructs an http GET request
    (and submits that request) to get a
    scholar's citations from Google Scholar

    Args:
      scholar_id (str): Google scholar ID

    Returns:
      citations_html (python object): The scholar's citations
    """
```

```
# Define the url
url = (
    "https://scholar.google.com/citations?user="
    + str(scholar_id)
    + "&hl=en&oi=ao"
)

# Define the saved html; give it a filename
saved_html = "scholar.html"

# Download the html via UNIX to avoid 429 errors
subprocess.run(
    ["wget", "-qO", saved_html, url],  # save to scholar.html
    check=True,  # raise errors if any
)

# Open and read the html using BeautifulSoup
with open(saved_html, "r", encoding="ISO-8859-1") as file:
    citations_html = bs(file, "html.parser")

return citations_html
```

Question 4c

> Now write a function that processes the HTML to return a Pandas (or Polars) data frame with the citation information (article title, authors, journal information, year of publication, and number of citations as five columns of information) for the researcher. Try your function on a second researcher to provide more confidence that your function is working properly. (We'll add unit tests in PS2). Given the comments in (b), ideally your function will work either if given (i) the name of a file that you've already downloaded or (ii) the HTML content produced by your function in (b).
>
> Hint: a possibly useful argument for `find_all` is to request element(s) with certain attributes, e.g., `html.find("p", attrs = {'class': 'songtext'})` for finding a `p` element whose class is `songtext`.

Again, use bs to parse the html. Then find the right tags to correlate to the required dataframe headers. Extracting `authors` and `journal` was tricky since these shared similar tags.

```python
def get_scholar_info(filename):
    """Process html to get scholar information

    Args:
      html (object): html with scholar's citations

    Returns:
      df (Pandas dataframe): The scholar's information
    """

    # Open and read the html using BeautifulSoup
    with open(filename, "r", encoding="ISO-8859-1") as file:
        citations_html = bs(file, "html.parser")

    # Define df to store info with defined column labels
    df = pd.DataFrame(columns=["title", "authors", "journal", "year", "num_cite"])

    # Identify the scholar
    scholar = citations_html.find("div", attrs={"id": "gsc_prf_in"}).text

    # Create a list of citations
    citations = citations_html.find_all("tr", attrs={"class": "gsc_a_tr"})

    print(f"Found {len(citations)} entries for {scholar}.")

    # For each entry, extract relevant data
    # Note that authors and journal share html tags
    # so need to call them by their element
    for entry in citations:
        title = entry.find("a", class_="gsc_a_at").text
        authors = entry.find_all("div", class_="gs_gray")[0].text
        journal = entry.find_all("div", class_="gs_gray")[1].text
        year = entry.find("span", class_="gsc_a_h").text
        num_cite = entry.find("a", class_="gsc_a_ac").text

        # Append all data to a new row in the df
        new_row = pd.DataFrame(
            [
                {
                    "title": title,
                    "authors": authors,
                    "journal": journal,
                    "year": year,
                    "num_cite": num_cite,
                }
            ]
        )
        df = pd.concat([df, new_row], ignore_index=True)

    return df


pd.set_option("display.max_colwidth", 20)
display(get_scholar_info("scholar.html"))
display(get_scholar_info("scholar2.html"))
```

```
Found 20 entries for Michael I. Jordan.
```

	title	authors	journal	year	num_cite
0	Latent dirichlet...	DM Blei, AY Ng, ...	Journal of machi...	2003	54531
1	On spectral clus...	A Ng, M Jordan, ...	Advances in neur...	2001	12450
2	Machine learning...	MI Jordan, TM Mi...	Science 349 (624...	2015	9458
3	Trust Region Pol...	J Schulman	arXiv preprint a...	2015	8494
4	Adaptive mixture...	RA Jacobs, MI Jo...	Neural computati...	1991	6114
5	Learning transfe...	M Long, Y Cao, J...	International co...	2015	5953
6	Graphical models...	MJ Wainwright, M...	Foundations and ...	2008	5535
7	An introduction ...	MI Jordan, Z Gha...	Machine learning...	1999	5438

	title	authors	journal	year	num_cite
8	Sharing clusters...	Y Teh, M Jordan,...	Advances in neur...	2004	5427
9	An internal mode...	DM Wolpert, Z Gh...	Science 269 (523...	1995	4246
10	Hierarchical mix...	MI Jordan, RA Ja...	Neural computati...	1994	4197
11	Distance metric ...	E Xing, M Jordan...	Advances in neur...	2002	4035
12	High-dimensional...	J Schulman, P Mo...	arXiv preprint a...	2015	3834
13	On discriminativ...	A Ng, MI Jordan	Advances in Neur...	2002	3670
14	An introduction ...	C Andrieu, N De ...	Machine learning...	2003	3600
15	Optimal feedback...	E Todorov, MI Jo...	Nature neuroscie...	2002	3567
16	Learning the ker...	GRG Lanckriet, N...	Journal of Machi...	2004	3141
17	Kalman filtering...	B Sinopoli, L Sc...	IEEE transaction...	2004	2989
18	Deep transfer le...	M Long, H Zhu, J...	International co...	2017	2878
19	Theoretically pr...	H Zhang, Y Yu, J...	International co...	2019	2641

```
Found 20 entries for Kenichi Soga.
```

	title	authors	journal	year	num_cite
0	Fundamentals of ...	JK Mitchell, K Soga	John Wiley & Son...	2005	12921
1	Physical propert...	WF Waite, JC San...	Reviews of geoph...	2009	1108
2	Biogeochemical p...	JT Dejong, K Sog...	Bio-and chemo-me...	2014	946
3	Energy pile test...	PJ Bourne-Webb, ...	Géotechnique 59 ...	2009	761
4	Factors affectin...	A Al Qabany, K S...	Journal of Geote...	2012	749
5	Effect of chemic...	AA Qabany, K Soga	Bio-and chemo-me...	2014	698
6	Trends in large-...	K Soga, E Alonso...	Géotechnique 66 ...	2016	531
7	Thermo-mechanica...	BL Amatya, K Sog...	Géotechnique 62 ...	2012	516
8	Particle shape c...	ET Bowman, K Sog...	Geotechnique 51 ...	2001	467
9	Estimating the e...	TE Vorster, A Kl...	Journal of Geote...	2005	426
10	Coupling of soil...	S Bandara, K Soga	Computers and ge...	2015	404
11	Soil—pipe intera...	A Klar, TEB Vors...	Géotechnique 55 ...	2005	317
12	Critical state s...	S Uchida, K Soga...	Journal of geoph...	2012	316
13	A review of NAPL...	K Soga, JWE Page...	Journal of Hazar...	2004	307
14	The applicabilit...	NJ Jiang, K Soga	Géotechnique 67 ...	2017	263
15	Material point m...	K Abe, K Soga, S...	Journal of Geote...	2014	257
16	Soil engineering...	JT DeJong, K Sog...	Journal of the R...	2011	245
17	Lateral and upwa...	S Yimsiri, K Sog...	Journal of geote...	2004	239
18	DEM analysis of ...	S Yimsiri, K Soga	Géotechnique 60 ...	2010	237
19	Microbially indu...	NJ Jiang, K Soga...	Journal of Geote...	2017	228

Question 4d

Create a requirements file (based on either pip or Conda) that has the necessary information (in particular Python package versions) to reproduce the environment in which you ran your code. Include this file in your GitHub repository directory for this problem set.

```
$ pip freeze > requirements.txt
```

Question 4e

> (Extra credit) If you'd like extra practice, write a Python function that will return the Google Scholar ID when the function is provided an html file as its argument. The file would be the file that is returned by searching for a researcher name as the input at scholar.google.com. As discussed in (a), your function should not query Google Scholar using the `requests` package, but rather should manipulate an HTML file that you download after manually querying Google Scholar yourself.

The strategy is to extract the data embedded in a tag for the scholar. Instead of accessing the text of a tag, I need to access the href itself. I do this by saving the tag as a string, and using the `re` library to find the characters that come after `user=`.

```python
import re


def get_scholar_id(filename):
    """Process html to get scholar ID

    Args:
      html (object): html with scholar's citations

    Returns:
      scholar_id (string): The scholar's ID
    """

    # Open and read the html using BeautifulSoup
    with open(filename, "r", encoding="ISO-8859-1") as file:
        query_html = bs(file, "html.parser")

    # Get scholar ID, which is stored in an h4 tag
    h4 = query_html.find_all("h4", class_="gs_rt2")

    # The scholar ID is always after the string "user="
    search_term = re.compile(r"user=([^\&]+)")

    # Look for the search term in the href
    result = search_term.search(str(h4))

    scholar_id = result.group(1)

    return scholar_id
```

```
print("The Google Scholar ID for the given html file is", get_scholar_id("query.html"))
```

```
The Google Scholar ID for the given html file is yxUduqMAAAAJ
```

Question 4f

> (Extra credit) If you'd like extra practice, fix your function so that you get all of the results for a researcher and not just the first 20. E.g., for Michael Jordan there are several hundred.

I used Stack Overflow to find the keywords that would enable me to cycle through several pages of citations. It seemed like Google Schoalr would only show me max 100 citations at a time. The code within the `while` loop was structured with help from ChatGPT.

```python
def get_all_citations(scholar_id):
    """Constructs an http GET request
    (and submits that request) to get ALL
    scholar's citations from Google Scholar

    Args:
      scholar_id (str): Google scholar ID

    Returns:
      citations_mega (python object): All the scholar's citations
    """

    # We need to increment "cstart" in the search term
    # until the page returns the string "No citations found."
    cstart_inc = 0

    # The while loop was with assistance from ChatGPT
    # Finding "cstart" and "pagesize" keywords was from Stack Overflow
    while True:
        # Define the url
        url = (
            "https://scholar.google.com/citations?user="
            + str(scholar_id)
            + "&hl=en&oi=ao&cstart="
            + str(cstart_inc)
            + "&pagesize=100"  # 100 seems to be max allowable page size
        )

        # Define the saved html; give it a filename
        saved_html = f"scholar_mega_cstart{cstart_inc}.html"

        # Download the page
        subprocess.run(["wget", "-O", saved_html, url], check=True)
        # Check if the file contains any results
        with open(saved_html, "r", encoding="ISO-8859-1") as file:
            content = file.read()

            # # Look for the message that indicates no more results
            if "There are no articles in this profile." in content:
                print("No more results found. Ending download.")

                # Delete the last downloaded file
                subprocess.run(["rm", saved_html])

                break

        cstart_inc += 100

    # Combine all downloaded webpages into 1
    subprocess.run("cat scholar_mega_cstart* > scholar_mega_combined.html", shell=True)

    # Delete all the intermediate htmls
    subprocess.run("rm scholar_mega_cstart*", shell=True)

    return


# We can then use this code with the code from Question 4c
# e.g. get_scholar_info("scholar_mega_combined.html'")
```

Question 5

> Look at the `robots.txt` for Google Scholar (scholar.google.com) and the references in Unit 2 on the ethics of webscraping. Does it seem like it's ok to scrape data from Google Scholar? Hopefully this will make clear why our scraping in Problem 4 did not include programmatically obtaining the Google Scholar ID.

Seems like it's not okay to scrape data from Google Scholar. Specifically, `robots.txt` disallows use of the following for scraping bots:

```
Disallow: /search
Disallow: /index.html
Disallow: /scholar
Disallow: /citations?
Disallow: /citations?*cstart=
Disallow: /citations?user=*%40
Disallow: /citations?user=*@
```

These rules relate somewhat to some of the exercises in the problem set, and that's why we didn't automate some of the processes that we otherwise could have.