

STAT 243 Problem Set 1

Treves Li

2024-09-12

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.

```
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] \div number of columns [given] \div 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")
```

30.777461 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 $15.5 \times 100000 \times 20 = 31$ million characters.

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

```
import os
import pandas as pd

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")
```

14.777452 MB
16.000573 MB

The answer above shows that CSV **would** have saved disk space compared to pickle file with sufficient rounding 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).

```
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 0.20029258728027344 seconds
read_pickle took 0.0037603378295898438 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 0.0218355655670166 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 0.04572463035583496 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.