Data Analysis – Modeling and Parsing

15-110 – Wednesday 04/08

Learning Goals

- Read and write data from **files**

- Interpret data according to different **protocols:** plaintext, CSV, and JSON

- **Reformat** data to add, remove, or reinterpret pre-existing data

Unit Overview

New Unit: CS as a Tool

Our next unit focuses on how computer science can be used to benefit other domains.

We'll investigate three different applications of computer science: **data analysis**, **simulation**, and **machine learning**.

These three applications share a core idea in common: all three **organize data** to **help people answer questions**.

Schedule for Unit 5

The schedule for this unit will be **staggered**.

The first week (Wed 4/8-Mon 4/13) will focus on how the applications **organize data**.

The second week (Wed 4/15-Mon 4/20) will focus on how the applications **find answers**.

Each of these weeks will end with a **short written assignment** that covers the main learning goals of the week. These assignments are part of Check6-1 and Check6-2.

Hw6 is a Guided Project

Hw6 is organized differently from the past assignments. In this homework, you will spend three weeks **building a code project** that uses computer science in some domain.

This project will be **heavily guided**, with lots of algorithmic instruction in the writeup. It will also have two check-ins at Check6-1 and Check6-2 before the full project is due in Hw6.

Most importantly – you get to choose which project you complete!

Hw6 Project Options

Each of the five projects implements one of the three applications we'll teach in class.

Battleship is focused on building a game. It uses **simulation**.

Circuit Simulator is focused on implementing circuits. It uses **simulation**.

Language Modeling is focused on identifying patterns in text. It uses **machine learning**.

Protein Sequencing is focused on analyzing DNA data. It uses **data analysis**.

Tweet Analytics is focused on analyzing political Twitter data. It uses **data analysis**.

Hw6 Schedule

Here are the important deadlines for Hw6:

Sunday 4/12 noon – Fill out this form to select which project you plan to do: https://forms.gle/NyBQJgbFpnzAz8Bz9

Wednesday 4/15 noon – Check6-1 is due (Hw6 check-in, and written assignment)

Wednesday 4/22 noon – Check6-2 is due (Hw6 check-in, and written assignment)

Wednesday 4/29 noon – Hw6 is due (full project, including work from both check-ins)

Data Analysis

Data Analysis Gains Insights on Data

Data Analysis is the process of using computational or statistical methods to **gain insight** about data.

Data Analysis is used widely by many organizations to answer questions in many different domains. It plays a role everything from advertising and fraud detection, to airplane routing and political campaigns.

Data Analysis is also used widely in **logistics**, to determine how many people and how much stock is needed, and where they should go.

Data Analysis and COVID-19

Data Analysis is also being used in the current pandemic to find answers to important questions, which may help save lives.

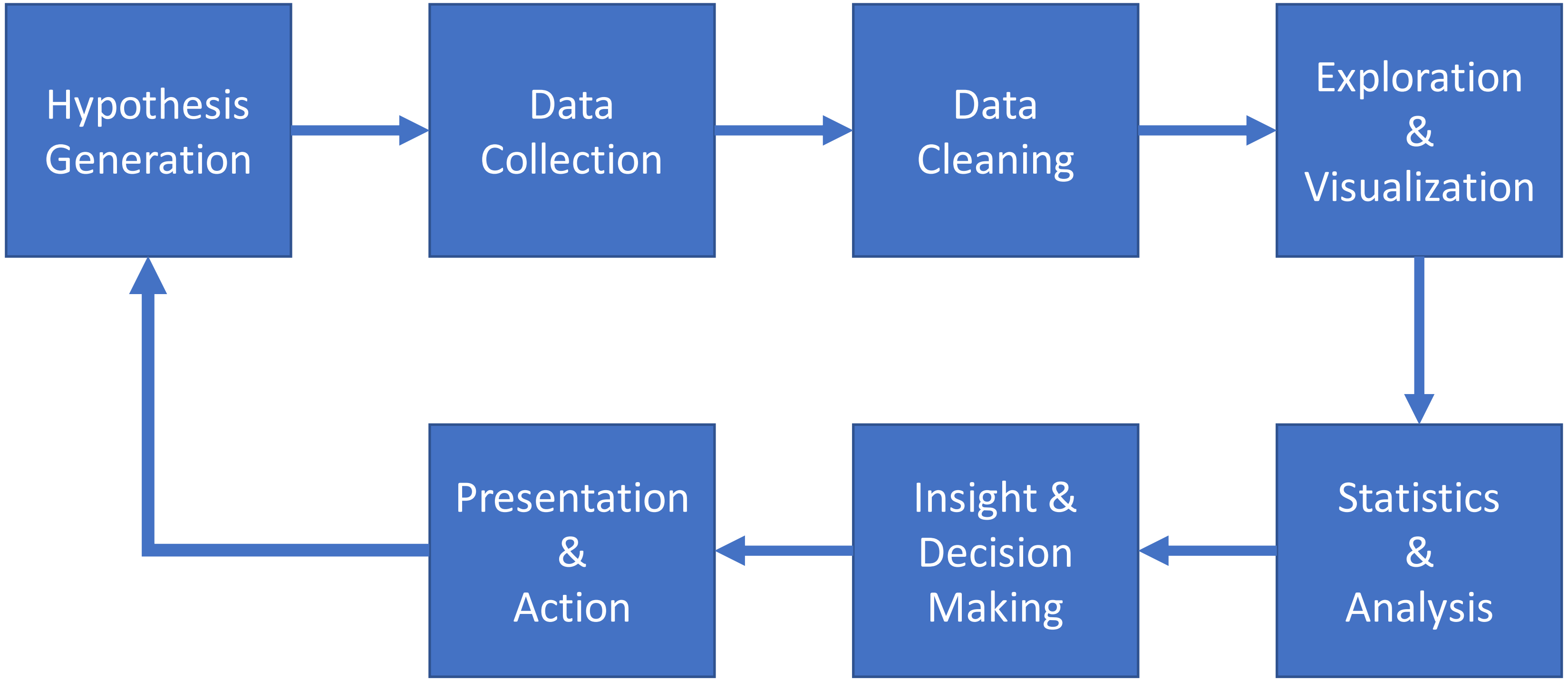
Data Analysis can determine how common different symptoms are, and which symptoms occur together. It can be used to estimate the infection rate of the disease, as well as the fatality rate.

Example: https://ourworldindata.org/coronavirus

Data Analysis Process

The full process of data analysis involves multiple steps to acquire data, prepare it, analyze it, and make decisions based on the results.

We'll focus mainly on three steps: Data Cleaning, Exploration & Visualization, and Statistics & Analysis



Data is Complicated

Before diving into data analysis, we have to ask a general question. What does data **look** like?

Data varies greatly based on the context; every problem is unique.

Example: let's collect our own data! Fill out the following short survey:

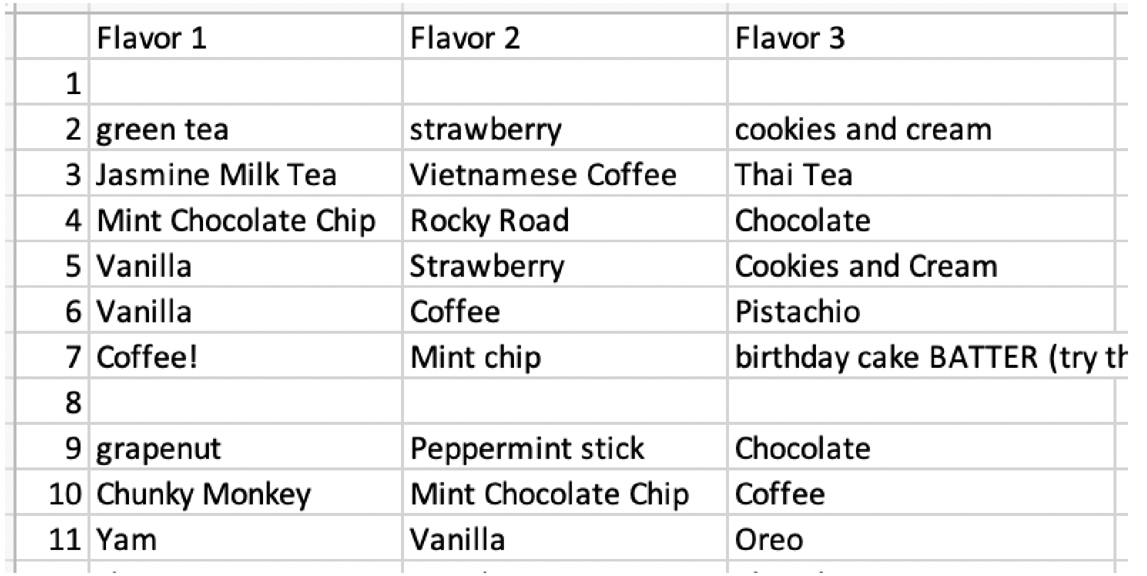
bit.ly/110-ice-cream

Data is Messy

Let's look at the results of our ice cream data.

Most likely, there are some **irregularities** in the data. Some flavors are capitalized; others aren't. Some flavors might have typos. Some people who don't like ice cream might have put 'n/a', or 'none', or 'I'm lactose intolerant'. And some flavors might have multiple names – 'green tea' vs. 'matcha'.

Data Cleaning is the process of taking raw data and smoothing out all these differences. It can be partially automated (all flavors are automatically made lowercase), but usually requires some level of human intervention.



	Flavor 1	Flavor 2	Flavor 3
1			
2	green tea	strawberry	cookies and cream
3	Jasmine Milk Tea	Vietnamese Coffee	Thai Tea
4	Mint Chocolate Chip	Rocky Road	Chocolate
5	Vanilla	Strawberry	Cookies and Cream
6	Vanilla	Coffee	Pistachio
7	Coffee!	Mint chip	birthday cake BATTER (try th
8			
9	grapenut	Peppermint stick	Chocolate
10	Chunky Monkey	Mint Chocolate Chip	Coffee
11	Yam	Vanilla	Oreo

Data Formats

Reading Data From Files

Once data has been cleaned, we need to access that data in a Python program. That means we need to **read data from a file**.

Recall that all the files on your computer are organized in **directories**, or **folders**. The file structure in your computer is a **tree** – directories are the inner nodes (recursively nested), and files are the leaves.

When you're working with files, always make sure you know which folder your file is located in. A sequence of folders from the top-level of the computer to a specific file is called a **filepath**.

Opening Files in Python

To interact with a file in Python, we'll need to access its contents. We can do this by using the built-in function `open(filepath)`. This will create a **File object**, which we can read from or write to.

```
f = open("sample.txt")
```

`open()` can either take a full filepath or a **relative path** from the location of the python file. It's usually easier to put the file you want to read/write in the same directory as the python file, so you can simply refer to the filename directly.

Reading and Writing from Files

When we open a file, we need to specify whether we plan to **read from** or **write to** the file. This will change the **mode** we use to open the file.

```
f = open("sample.txt", "r") # read mode
line = f.readline() # reads a single line of the file as a string
text = f.read() # reads the whole file as a string

f = open("sample2.txt", "w") # write mode
f.write(text) # writes a string to the file
```

Only one instance of a file can be kept open at a time, so you should always **close** a file once you're done with it.

```
f.close()
```

Be Careful When Programming With Files!

WARNING: when you write to files in Python, backups are not preserved. If you overwrite a file, the previous contents are gone forever. **Be careful when writing to files**.

WARNING: if you have multiple Python files open in Pyzo and you try to open a file from a relative path, Pyzo might get confused. To be safe, when working with files, only have one file open in Pyzo at a time. And make sure to 'Run File as Script' when working with files.

Data has Many Different Formats

Once you've read data from a file, you need to determine what the **structure** of that data is. That will inform how you store the data in Python.

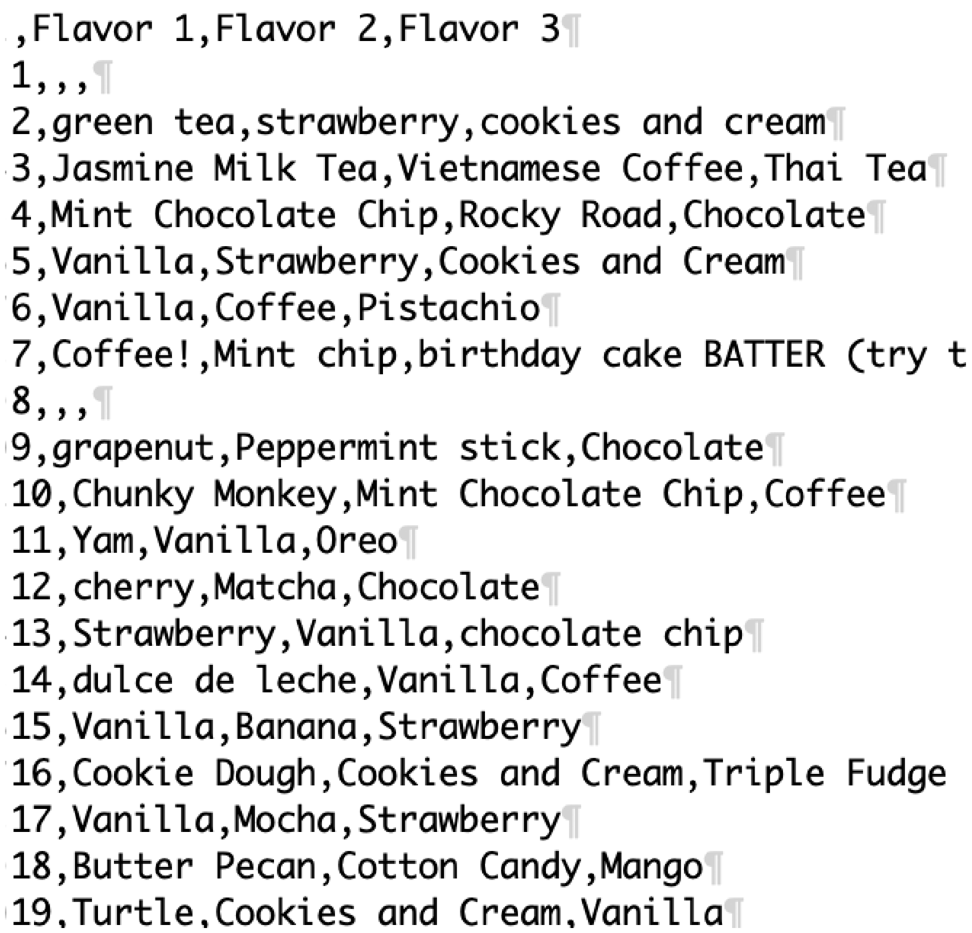
We'll discuss three formats here: CSV, JSON, and plaintext. Many other formats exist; we'll introduce a few in a later lecture.

CSV Files are Like Spreadsheets

First, **Comma-Separated Values (CSV)** files store data in two dimensions. They're effectively spreadsheets.

The data we collected on ice cream was downloaded as a CSV. If we open it in a plain text editor, you can see that values are separated by **commas**.

These files don't always have to use commas as separators, but they do need a **delimiter** to separate values (maybe spaces or tabs).



```
,Flavor 1,Flavor 2,Flavor 3¶
1,,,¶
2,green tea,strawberry,cookies and cream¶
3,Jasmine Milk Tea,Vietnamese Coffee,Thai Tea¶
4,Mint Chocolate Chip,Rocky Road,Chocolate¶
5,Vanilla,Strawberry,Cookies and Cream¶
6,Vanilla,Coffee,Pistachio¶
7,Coffee!,Mint chip,birthday cake BATTER (try t
8,,,¶
9,grapenut,Peppermint stick,Chocolate¶
10,Chunky Monkey,Mint Chocolate Chip,Coffee¶
11,Yam,Vanilla,Oreo¶
12,cherry,Matcha,Chocolate¶
13,Strawberry,Vanilla,chocolate chip¶
14,dulce de leche,Vanilla,Coffee¶
15,Vanilla,Banana,Strawberry¶
16,Cookie Dough,Cookies and Cream,Triple Fudge
17,Vanilla,Mocha,Strawberry¶
18,Butter Pecan,Cotton Candy,Mango¶
19,Turtle,Cookies and Cream,Vanilla¶
```

Reading CSV Data into Python

We could open a CSV file as plaintext and parse the file as we read it. Or we could use the **csv library** to make reading the file easier.

This library creates a **Reader** object out of a File object. When each line is read from a Reader object, the line is automatically parsed into a 1D list, by separating the values based on the delimiter.

We can pass optional values into the `csv.reader` call to set the delimiter.

```
import csv

f = open("icecream.csv", "r")
reader = csv.reader(f)

data = [ ]
for row in reader:
    data.append(row)

print(data)

f.close()
```

Writing CSV Data to a File

What if we've processed data in a 2D list, and want to save it as a CSV file?

Create a CSV **Writer** object based on a file. You can use it to write one row at a time using `writer.writerow(row)`.

Again, the delimiter can be set to values other than a comma by updating the optional parameters.

```
import csv

data = [[ "chocolate", "mint chocolate",
           "peppermint" ],
        [ "vanilla", "matcha", "coffee" ],
        [ "strawberry", "mango", "cherry" ]]

f = open("results.csv", "w", newline="")
writer = csv.writer(f)

for row in data:
    writer.writerow(row)

f.close()
```

JSON Files are Like Trees

Second, **JavaScript Object Notation (JSON)** files store data that is **nested**, like trees. They are commonly used to store information that is organized in some structured way.

```
{
  "vanilla" : 10,
  "chocolate" : {
                  "chocolate" : 15,
                  "chocolate chip" : 7,
                  "mint chocolate chip" : 5
                },
  "other" : [ "strawberry", "matcha", "coffee" ]
}
```

JSON files can store data types including Booleans, numbers, strings, lists, dictionaries, and any combination of the above.

Reading JSON Files into Python

The easiest way to read a JSON file into Python is to use the **JSON library**.

This time, we'll use `json.load(file)` or `json.loads(string)`. These functions read a piece of data that matches the type of the outermost data in the text (usually a list or dictionary).

In our example from the last slide, it would be a dictionary mapping strings to integers, dictionaries, and lists.

```
import json
f = open("icecream.json", "r")
j = json.load(f)
print(j)
f.close()

j = json.loads("""{
  "vanilla" : 10,
  "chocolate" : {
                 "chocolate" : 15,
                 "chocolate chip" : 7,
                 "mint chocolate chip" : 5
               },
  "other" : [ "strawberry", "matcha", "coffee" ]
}""")
print(j)
```

Writing JSON Data to a File

What if we want to store JSON data in a file for later use?

Again, use the JSON library. The `json.dump(value, file)` method will take a JSON-compatible value and write it to a file in JSON format.

We can also use `json.dumps(value)` to convert a value to a JSON-friendly string, then write that string to a file.

```
import json

d = { "vanilla" : 10,
      "chocolate" : 27,
      "other" : [ "strawberry", "matcha", "coffee" ]
    }

f = open("results.json", "w")
json.dump(d, f)
f.close()

f = open("results2.json", "w")
s = json.dumps(d)
f.write(s)
f.close()
```

Reading Plaintext Data

Finally, a lot of the data we work with might not fit nicely into either a CSV or JSON format. If we can read this data in a simple text editor, we call this **plaintext data**.

To work with plaintext, you need to identify what kinds of **patterns** exist in the data, and use that information to structure it. The patterns you identify may depend on which question you're trying to answer.

Working with Data

Questions to Ask

When parsing data in a plaintext file, start by identifying the pattern; then ask yourself a few questions about that pattern.

- Does the pattern occur across lines, or some other delimiter?
- Where is the information in a single line/section?
- What comes before or after the information you want?

Tools to Use

Once you've identified where the information is located, use **string methods** to separate out the information you need.

String slicing (`s[start:end:step]`) can be used to remove parts of the data that are unnecessary.

String splitting (`s.split(".")`) can be used to break up data that is separated by a known delimiter.

String finding (`s.find(":")`) can be used to find the location of the beginning or end of a section. That can be combined with slicing or splitting to isolate the needed data.

Example: Parsing a Chat Log

`chat.txt` is a dataset based on a chat log from a previous class. (All student names have been modified to preserve student privacy).

How could we get the names of everyone who participated in the chat? What's the **pattern**?

`"From  "` occurs before each name, and `" :"` occurs afterwards. Find those indices and split based on them.

A few lines don't match the pattern; account for those too.

```
f = open("chat.txt", "r")
text = f.read()
f.close()
people = [ ]
for line in text.split("\n"):
    start = line.find("From  ") \
            + len("From  ")
    line = line[start:]
    end = line.find(" :")
    line = line[:end]
    if "(Privately)" in line:
        end = line.find("  to")
        line = line[:end]
    people.append(line)
print(people)
```

Updating, Adding, and Removing Values

Once we've parsed our data into an appropriate format, we may need to change the structure to achieve the analysis we want. Let's assume that we're working with a 2D list produced from the ice cream data.

To **update** a value, access the appropriate column in each row, and change it. For example, you might want to convert a string to a different type via type-casting.

To **remove** a value, pop an element of each row based on the column that needs to be removed. To **add** a value, append or insert a new value into each row, potentially based on the pre-existing values.

Make sure to update the **header** according to a separate rule!

```
# Assume data is a 2D list parsed from the file
header = data[0]
header.pop(0) # remove the ID
header.append("# chocolate")
for row in range(1, len(data)):
    data[row].pop(0) # remove the ID
    chocCount = 0 # count number of chocolate
    for col in range(len(data[row])):
        # Make all flavors lowercase
        data[row][col] = data[row][col].lower()
        if "chocolate" in data[row][col]:
            chocCount += 1
    # track chocolate count
    data[row].append(chocCount)
print(data)
```

Learning Goals

- Read and write data from **files**

- Interpret data according to different **protocols:** plaintext, CSV, and JSON

- **Reformat** data to add, remove, or reinterpret pre-existing data