Data Analysis – Modeling and Parsing

Learning Goals

- Read and write data from files
- Interpret data according to different protocols: plaintext, CSV, and JSON
- Reformat data to find, add, remove, or reinterpret pre-existing data

Data Analysis

Data Analysis Gains Insights on Data

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

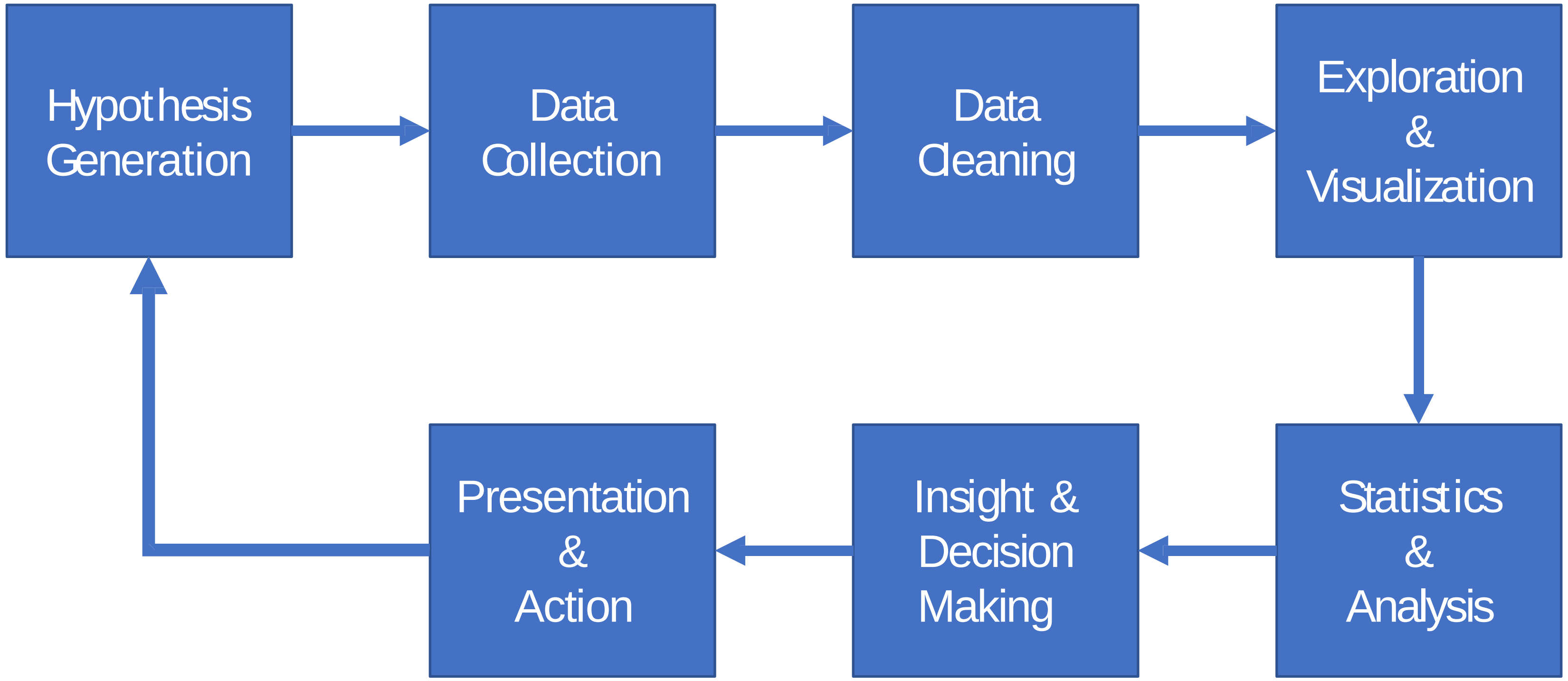
Data Analysis is used widely by many organizations to answer questions in many different domains. It plays a role in everything from advertising and fraud detection to airplane routing and political campaigns.

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 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

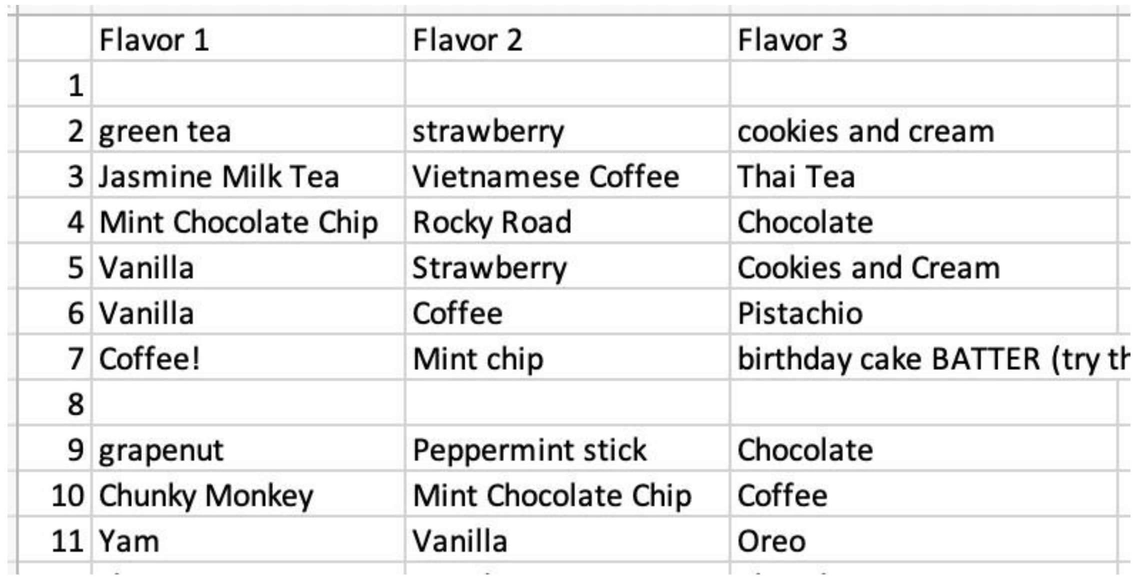


Data is Messy

Let's look at an example of 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

Reading Data from Files

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 sequence of folders 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 (relative from the location of the python file). It's usually easiest 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
lines = f.readlines() # reads the lines of a file as a list of strings
# or
text = f.read() # reads the whole file as a single 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 PyCharm and you try to open a file from a relative path, PyCharm might get confused. To be safe, when working with files, only have one file open in PyCharm at a time. And make sure to 'Run File as Script' when working with files.

Activity: Read a File

You do: Download the file test1.txt from Sakai and move it to the same folder as a python script. Try using open and read to open the file and read the contents, then print the contents.

If Python says a filename doesn't exist when you're sure that it does, go to office hours to get help; there's a few common problems that can occur.

Common file reading issues:

- make sure the file is actually in the same directory as your python script
- make sure the filename you've entered is actually the filename (including the filetype at the end!)
- make sure you're using Run File as Script (execute usually won't work)
- make sure only one file is open in Colab/Jupyter/PyCharm

Sidebar: os library for advanced files

The oslibrary lets you directly interact with your computer's operating system. You can use this library to further modify files on your computer. The following functions are especially useful:

```
os.listdir(path) # returns a list of files in the directory
os.path.exists(path) # returns True if the given path exists
os.rename(a, b) # changes file a's name to b
os.remove(path) # deletes the file.
```

Data Formats

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.

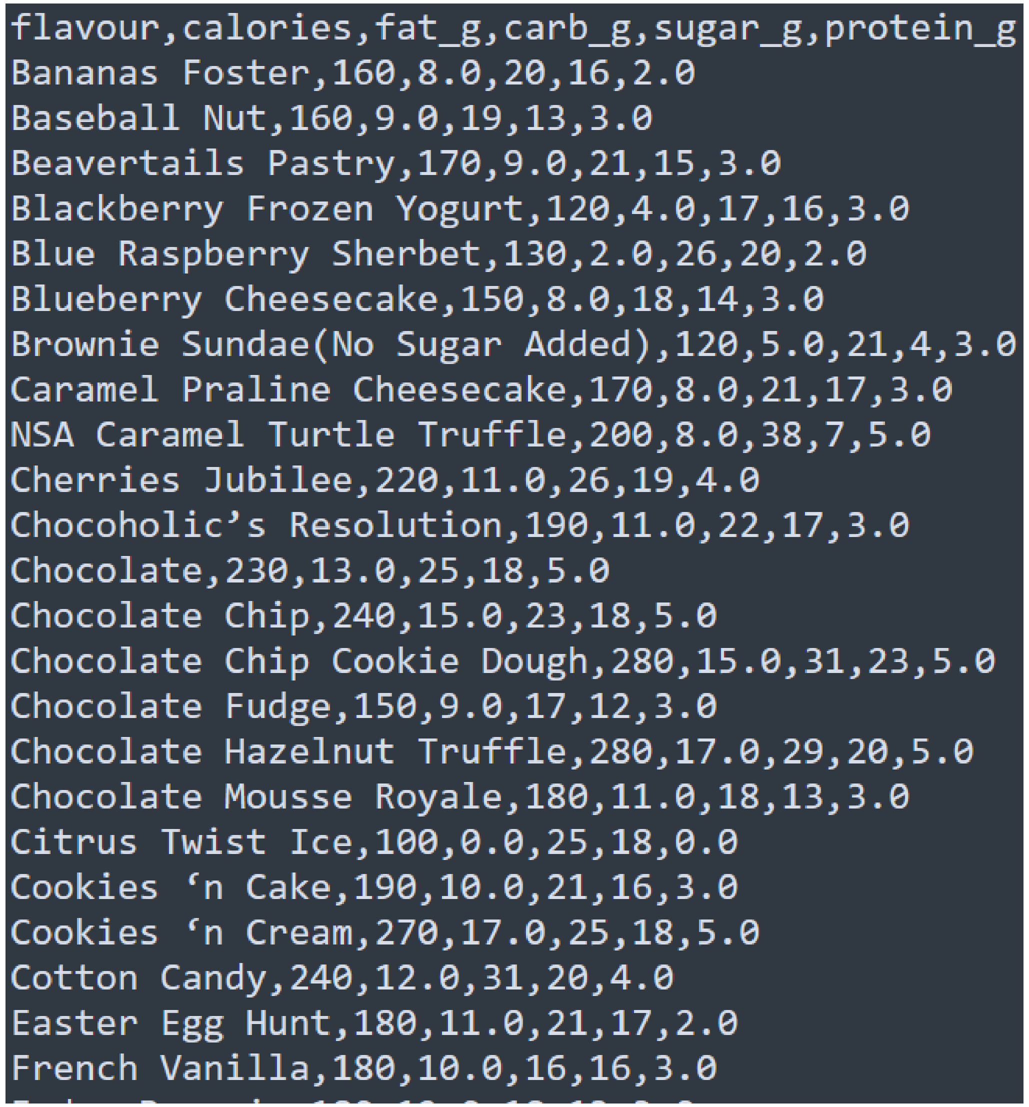
We'll discuss three formats here: CSV, JSON, and plaintext. Many other formats exist!

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).



```
flavour,calories,fat_g,carb_g,sugar_g,protein_g
Bananas Foster,160,8.0,20,16,2.0
Baseball Nut,160,9.0,19,13,3.0
Beavertails Pastry,170,9.0,21,15,3.0
Blackberry Frozen Yogurt,120,4.0,17,16,3.0
Blue Raspberry Sherbet,130,2.0,26,20,2.0
Blueberry Cheesecake,150,8.0,18,14,3.0
Brownie Sundae(No Sugar Added),120,5.0,21,4,3.0
Caramel Praline Cheesecake,170,8.0,21,17,3.0
NSA Caramel Turtle Truffle,200,8.0,38,7,5.0
Cherries Jubilee,220,11.0,26,19,4.0
Chocoholic's Resolution,190,11.0,22,17,3.0
Chocolate,230,13.0,25,18,5.0
Chocolate Chip,240,15.0,23,18,5.0
Chocolate Chip Cookie Dough,280,15.0,31,23,5.0
Chocolate Fudge,150,9.0,17,12,3.0
Chocolate Hazelnut Truffle,280,17.0,29,20,5.0
Chocolate Mousse Royale,180,11.0,18,13,3.0
Citrus Twist Ice,100,0.0,25,18,0.0
Cookies 'n Cake,190,10.0,21,16,3.0
Cookies 'n Cream,270,17.0,25,18,5.0
Cotton Candy,240,12.0,31,20,4.0
Easter Egg Hunt,180,11.0,21,17,2.0
French Vanilla,180,10.0,16,16,3.0
```

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.

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

```
{
    "vanilla" : 10,
    "chocolate" : {
                              "chocolate" : 15,  "chocolate chip" : 7,
                              "mint chocolate chip" : 5

                          },
    "other" : [ "strawberry", "matcha", "coffee" ]
}
```

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). This function reads text from a file and produces 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, the function would produce a dictionary mapping strings to integers, dictionaries, and lists.

```
import json

f = open("icecream.json", "r")
j = json.load(f)

print(j)

f.close()
```

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.

```
import json

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

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

We'll talk about this more in the next section.

Activity: Match Data Structure to Format

You do: which data format would you use to store Python data organized in a...

... tree?

... 2D list?

... string?

datetime Library

If you're working with data that includes timestamps, the datetimemodule is useful for parsing information out of the timestamp.

There are functions that let you get the day, month, hour, minute, or whatever else you might want out of a timestamp. You can also convert timestamps between different formats (like "mm/dd/yy"to "dd-mm-yyyy").

Use datetime.datetime.now()to get the timestamp at the moment the line of code runs. This is useful when you're generating log files.

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 slicing and string methods to separate out the information you need.

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

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

The find method (`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.

The strip method (`s.strip()`) can be used to remove whitespace (spaces, tabs, and newlines) from the front and back of a string. This is useful for isolating the core text of a string.