# CUBE
# Project 1: The statisfaction of parsing CSVs

There are a number of datafiles with COVID numbers and income data by zipcode. Your first job is to read the README file in the COVID repo and check out all the csv (comma separated values) files. Try opening them in vim (or some text editor) and read over them. The first line gives the various fields, and each subsequent line is a entry of data. This is an extremely common format for data in csv files.

## 1 The goal: making sense of COVID numbers

There are many different goals in this project and you are welcome (nay, encouraged) to come up with your own. Let us first state various goals, and then discuss I/O formatting.

1. Given a country, a start date, and an end date, generate a plot with the number of COVID cases with respect to those dates. So the $x$-axis is a date, and the $y$-axis is the number of COVID cases.
2. Generate a scatter plot of the rate (percentage) of COVID cases by mean family income for every zipcode in Santa Clara county.
3. For your zipcode, plot the fraction of vaccinated population by day.
4. Generate a scatter plot of vaccinations by mean family income for every zipcode in California.
5. Given any county in California, output the month with the largest number of COVID cases (in that month).
6. Given any county in California, output the month with the largest number of COVID deaths (in that month).

There are multiple way of actualizing these (different) goals through programming. One option is to write a separate program for each task. For each task, the input can be provided either through an input file or as command line arguments. This is the best way to start.

But a better option, at least in my opinion, is to have *single program* that performs all these tasks. These tasks can be specified in an input file, line by line. Or they can be specified as console input, where the program asks the user what she wants. For example, the line `lc Alameda` could lead to the program computing the largest number of cases in Alameda county. The line `pic` could plot income by cases for Santa Clara. The line `pcc United_States 03-01-2021 04-01-2021` could plot cases for the country United States within those dates.

## 2 So much to do, so little time

Yes, this may seem daunting. There are so many tasks; how could you possibly figure all of them out? The trick is in realizing that once you figure out how to *one* of those tasks, the others become easy. All of them involve parsing a data file, storing some of the data, and either plotting it or searching within it.

So start by picking a *single* task that looks the easiest (don't do the one you find most interesting, start with the easiest). Find a task that only involves reading a single data file (like the country analysis). Or try to pick a task involving the smallest files (like the Santa Clara zip codes).

Here are some suggestions on tacking a single task.

1. Start by either hardcoding the inputs, or inputting them in the easiest way for you. If that is by command line arguments, fine. If you prefer typing in console input, sure. *Whatever is easiest.*

2. Instead of plotting, first, just try to print the relevant data out into a file. Inspect the file to make sure you have the right output.

3. Next, try to generate the plots.

4. Now get the inputs either through a file or the command line.

5. Finally, work on making your plot look nice. Try to put a title, label the axes, put a legend if you wish, etc.

Try to do two tasks independently, before you write a single program that tackles multiple tasks. The way to manage multiple tasks is to write separate functions that perform each tasks, and then writing a *wrapper code* that calls the tasks. When you compile/call the code from the command line, you call the wrapper code, which will parse the inputs and call the appropriate function.

## 3 We all need advice

• Start simple, then build complex. Pick a single task, break it down in smaller subtasks, write code for each, and then build the final solution.

• In writing code, wastage is ok! Do not expect the write the final solution (even for one task) is a single go. You may initially write code where you (say) hardcode inputs, and then finally get rid of that restriction. It is common to end up with a lot of "extra" code beyond your solution.

• For parsing in Python, the `split` function is your friend. In C++, parsing is quite a hassle. But you can use stringstreams, or the function `strtok`. The latter is actually not recommended any more, but it is easy to code up.

- To plot, use the Python library *matplotlib.pyplot*. There are many tutorials online about matplotlib. Indeed, one of the great things about Python is the existence of so many convenient libraries.
- Write code within functions, so you break up multiple tasks into independent blocks.
- Please comment your code! Or else, you will find it hard to debug.
- Think about how you would validate or test your code.

## 4 Reading your libraries

Once you figure out some portion of the above tasks, repeat them. But using the Python library `pandas`. This is a popular library used to parse csv files and other structured datasets. It is the standard library for many data science applications. The way to learn pandas is to do some Googling, experimenting with some commands, reading the documentation, rinse and repeat.

The fundamental object is a *dataframe*, which is a fancy word for a table of data. You can read a csv file into a dataframe, and then perform various operations on it. For example, you can select a subset of the dataframe, or sort it, or group it by some attribute/column. You can also plot it, and pandas will use matplotlib to do so. The way to learning is by experimentation and trial and error. So you have to keep at it, and do a mix of reading and coding.

Eventually, you can put all of this into a Python Notebook, which is a convenient way to write code and document it. You can easily swap out the input csv files, and rerun the code. You can also add comments and explanations in the notebook.