LUCIANO GABBANELLI

**FINAL PROJECT:**

**(QUASI-)BIG DATA**

Línea horizontal

DATE: /10/2022

Introduction

In this capstone project, you will apply all the content you have been learning during the first Python module.

You will go over how to create a Jupyter Notebook from scratch, handle strings, use lists and dictionaries, create functions, lambda functions, higher-order functions, and functions within functions, among many other methods.

The main goal of this final project is to understand a [data table](https://en.wikipedia.org/wiki/Table_(database)) with a large amount of information. You will deal with many [web servers](https://en.wikipedia.org/wiki/Web_server) as table rows and their features as columns. These features are the names of the web server, their [URL](https://en.wikipedia.org/wiki/URL) and a short description of what they are for.

We introduce tables because they are a key ingredient in [relational databases](https://en.wikipedia.org/wiki/Relational_database). Tables are a basic way of storing data and allow easy access to information.

The final task of this project is to generate a report of the common features found in the table (valid entries, repeated features, etc.). For this, you will have a guide to follow.

What are the main objectives in this project?

* Create a Jupyter Notebook from scratch called “***9.2-Final\_project.ipynb***”.
* Read a .txt with the table and write .txt files as outputs of your code.
* Review various methods learned during the first module to automate the exploration of information to help us understand the database.
* Learn new methods to write powerful algorithms.
* Write a report that describes in detail the structure of the database.

1. General analysis

The following steps will help you to approach the content of the database:

Step 0) Browse the database. Familiarize yourself with it[[1]](#footnote-0). Read the .txt file and understand its overall structure. As you can see when exploring, what is represented by rows in a table corresponds to each valid line of the file, and columns are represented by the values separated by commas within each line. In this way any structure with table format can be easily represented as a text file.

Count the number of total lines, blank lines[[2]](#footnote-1) and the number of comments[[3]](#footnote-2). Also predict how many valid entries will be left after cleanup. All this will be part of the final report.

Verify that everything is correct and set up for starting the exploration. Here you will plan the first steps for a major cleaning of the information.

1. Create at least one function (you can subdivide the content of this function into different functions that call each other, if you prefer) called read\_and\_clean(File, characters\_for\_comments) to read the .txt file with the database and clean it.

Parameters: when called, the function arguments must be the directory of the .txt file, together with the characters that indicate the presence of comments. These characters can be as many as the user wants[[4]](#footnote-3) and are useful to skip comments. The function must also work when no characters are introduced.

Return: besides reading and cleaning the initial file, your function should return all valid entries of the database grouped all together in a manageable python object; in whatever kind of type you consider <3

Report how many valid entries remain. Did it match the prediction made in the previous section?

1. Define three auxiliary functions that will be used later as statements for other functions. It is good practice to invent small and simple code to verify that the function does what you want. It is not mandatory to create all functions right now; if you are not sure how to proceed, skip on to the next item and come back when you need them when creating the next functions.

Functions are defined with the following purpose:

* + 1. A features\_picker\_cleaner(entries, feature, leftover\_chars) function that will act as a features selector.

Parameters: this function must receive the entries already cleaned by the function defined in the previous section, an integer (to choose one of the three values of each entry: name, url or description -if any-)[[5]](#footnote-4), and a variable number of string characters (which will clean each individual feature of leftover characters; for example the name “burrrd.” which corresponds to one of the entries should be replaced by “burrrd” when running this function). The function must also work when no characters are entered.

Return: a single feature, cleaned of undesired characters and ready to operate with or to be reported.

The function must contain at least one higher-order function and one lambda function inside it.

* + 1. A counter(value\_to\_search, where\_to\_look\_for) function to count occurrences.

Parameters: this function receives a value to be searched and an object in where to look for.

Return: the output must be an integer with the number of times the value\_to\_search appears in where\_to\_look\_for.

Use at least a higher-order function and a lambda function inside the function.

* + 1. Create a function called values\_and\_frequencies(where\_to\_look\_for, min\_num\_repetitions) to produce a dictionary with (a mapping between) the values and their respective frequencies of occurrence.

Parameters: in addition to an object to search for values, the function must be free to choose a threshold of minimum repetitions.

Return: a dictionary where its keys are the counted values, and its values, their respective frequencies of occurrence.

Since the minimum value to start counting is a free parameter of the function, the latter is not only useful to get an idea of the frequency of all values, but it can also work as a duplicate counter. Thus, by setting two repeats as the minimum count parameter, the function provides a dictionary of repeat values, filtering out only entries that repeat more than twice, and reporting how many times each occurs in total.

Specifications: here you will call a function from another function. To produce the counts for each of the entries, you will need to use the counter() function previously created. One would be tempted to just write where\_to\_look\_for.counts(value\_to\_search), but since we want to practice higher-order functions, you will need to implement your counter(value\_to\_search, where\_to\_look\_for) function, which performs the same task but is a user-created function containing higher-order functions and lambda functions.

1. Create a function called filter\_names() that filters out the second half of the entries’ names, after ordering them alphabetically. Here you will use one of the auxiliary functions defined above: the one that extracts and cleans only the names (first feature) from the entries (which contains three features, remember?).

Parameters: the number and type you consider.

Output: the function should create a text file called "patients\_names.txt" with the filtered names. If you want, the function can print some information about feature names when it is executed.

At this point you can follow these instructions: open the text file and scroll to the bottom. Text readers usually point to the line number you are standing on with your cursor. Check that what your text reader shows as the last line matches the length of half of the valid entries that you should have calculated before. If the number of valid names is odd, for instance 1719, then keep the floor division, i.e. 895.

1. Function to extract the webpage [domains](https://en.wikipedia.org/wiki/Domain_name)[[6]](#footnote-5). Create a function called domain\_extraction() that extracts all non-repeating domains. As you did for the previous function, fetch the URLs with the features\_picker\_cleaner() function. Remember that you can use sets to preserve unique values. Try typing

url\_list = read\_transform\_return(list\_of\_lines, 1);

(1 or the number corresponding to URLs that you have chosen) and then convert this list to sets with

url\_set = set(url\_list).

My code returns that out of 1790 initial entries, only 1730 are not repeated. You can explore repeated entries with your values\_and\_frequencies() function by setting min\_num\_repetitions to 2.

Parameters: the number and type you consider.

Return: a list with unique domains and report its structure in the final report.

1. Function to extract shared characteristics in the description. You will now use all the auxiliary functions in the same code. Create a function called characteristics\_organization() that takes the description feature first, and organizes common descriptions into a dictionary. Return this dictionary: the keys will be each description and the values, the number of web servers that share this characteristic. Use n/a for ambiguous or empty information.

To conclude the project, write a report documenting the structure of the database.

2. Pill organization

It is important to document your own work in each pill or project you do. In this way you can easily after a time read your own resume of what you did in each project.

The project is mainly based in a Jupyter notebook called "***9.2-Final\_project.ipynb***" that you must create yourself. This notebook reads and creates files located in a folder called “Files”, which must be placed in the same directory as the notebook itself.

Use the **markdown** cells as a guide to understand what your code is actually doing on a global level; its functionality. Use sections for each function. Be organized!

**Comments** in code cells (remember that code comments start with a hashtag, i.e. #) should have information about how the algorithms work internally. If a programmer opens your code, he must understand how the program flows by reading your comments.

3. Requirements

* You will have to create the notebook file where the cleanup and exploration will take place.
* All code must be contained in a single notebook.
* It is essential that you add comments on the most important elements during the development of the project. Markdown cells or comments in code cells.
* All code, including comments, must be written in English.
* All files that were used in the creation of the project as well as its solution must be attached.
* Delete files that are not used or are not necessary to evaluate the project.

4. Deliverables

To evaluate the pill you will need to present a folder with:

* A Jupyter notebook called: “***9.2-Final\_project.ipynb***”.
* A PDF version for the project documentation.
* A file with a report of the structure of the database.
* A text file with a record of incidents that were detected during project execution.
* Explain what you have learned during this project.
* Conclusions you have obtained from the analyses carried out when necessary.
* What problems have you encountered when developing this project?

1. The print() function will help you a lot to have control over your code. [↑](#footnote-ref-0)
2. You will have to deal with blank lines because that may complicate database manipulation. [↑](#footnote-ref-1)
3. Comments are often recognized because they start with special characters. Although they contain useful information for those who are dealing with the database, these entries should be avoided when you want to start operating with their content. Be careful! Not all comments begin with the Python # character for comments in this language. [↑](#footnote-ref-2)
4. Different databases will likely have different choices for indicating the presence of comments. It is better if your code can be adapted to different files and not just the one you are dealing with. [↑](#footnote-ref-3)
5. Beware of Python! It uses zero-based indexing. [↑](#footnote-ref-4)
6. The domain is the part of the URL that comes after the protocol (for example, http) and after the host-specific label of the hostname (for example, www). [↑](#footnote-ref-5)