corresponding to each parcel; d) then the script parses the page's HTML code to extract any attribute of interest for each parcel registered in the tax system (e.g. "chain of ownership" that does not exist in the parcel data set), this approach is referred to as "Web Scrapping" (for definition see https://en.wikipedia.org/wiki/Web_scraping); e) finally, the information scrapped from the tax system is inserted into a record in a new PostgreSQL table created in the AWS-RDS. At this point we have two tables in the AWS-RDS: the city parcels original table and the table containing the selected attributes from the original parcels table and the attributes and records scrapped from the tax system website. Figure 1 illustrates the process just described.

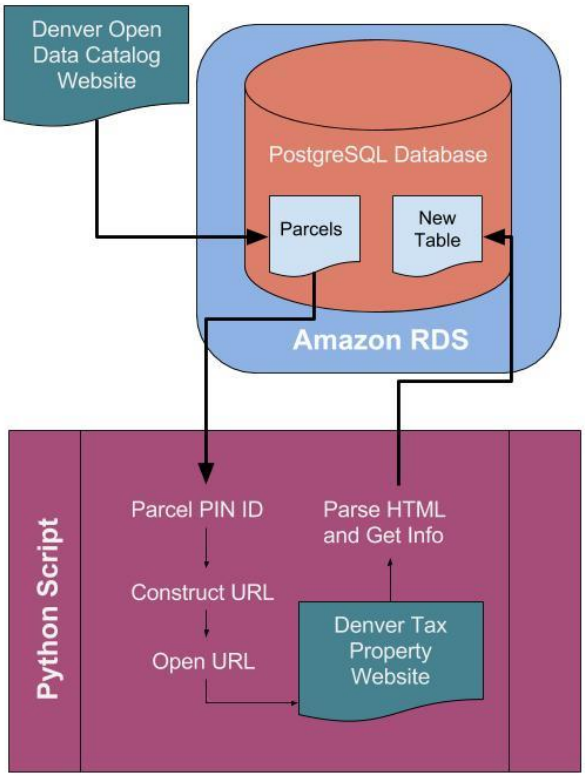

Figure 1. An overview of our methodology.

4. RESULTS

The test of matching of records between the parcels table and tax records table was successful. We tested scrapping diverse attributes from the tax system website and joining them to selected attributes and records from the city parcels table to create custom tables. Performance wise we found that the biggest constraint was how fast the tax system website could be accessed to retrieve the HTML code rendering the page corresponding to each parcel record. In our first tests, we let the Python script request the HTML of each page without delay, this resulted to be problematic as the website was unable to process the requests fast enough. After two hours of processing we cancelled the process. We found that adding as part of the script a small delay of 5 second between requests was enough to avoid this issue.

The information in the custom tables resulting from the processes described above are stored in the AWS-RDS and can be accessed by any software or GIS system that can connect to a database management system in a network. This would allow end users to easily join the new custom tables with any other information that have has a common unique identifier (such as the PIN number). For example, QGIS offers all required drivers to connect to PostgreSQL hosted locally or through a network.

Our Python script consisted of approximately 45 lines of code, most of them are simple adaptations of code found in well-known and well-documented free code libraries. The process to set-up a PostgreSQL database in the Cloud, such as in AWS as done here, has becomes much more user friendly and costs continue to decrease. Most Cloud services offer free 'tiers', our AWS RDS was in fact one of such free tiers, that allow for learning and also prototyping.

The FOSS, methods, and Cloud infrastructure here described are only one of several no-cost or low-cost options for combining information from Open Data repositories and made them broadly available through the Internet. Our focus was to find simple and reliable software and methods that are easy to learn and implement by users with basic technological skills.