Racket Billboard Scraper

Won Seok Chung, Tran Le, Jonathan Gomez

Abstract
While Python and R are the two most popular programming languages used to work with data, Racket also has many functionalities to perform similar tasks. We aimed to create a Racket program which can perform web scraping, a technique to extract data from web and save it as a local file. Billboard, a entertainment media company, is reputable for its music charts including Billboard Hot 100 and Billboard 200. These charts are updated weekly, based on radio play, sales, and online streaming. Our program allows users to access the Billboard Chart on Racket as a table and locally save it as a CSV file .

Background and Significance

The general pipeline of data science project has following steps: data acquisition, hypothesis & modelling, evaluation & interpretation, deployment, operations, and optimization. The first step of working with data is acquiring it, which can vary in difficulty based on its source. A common technique used is web scraping, which is extracting data from a webpage. There are many languages and frameworks designed to perform the task.

Racket, a general-purpose programming language developed from the Lisp-Scheme family, is not a popular choice for working with data. However, it is compatible with various data structures such as lists, vectors, and hash tables, meaning it has all fundamental functionalities to work with data. Accordingly, we wanted to prove that Racket also has a capability of scraping web data in an efficient manner.

As such, we created a Racket program which can scrape current Billboard charts and convert them into CSV files. We hope our program shows that Racket is a powerful language to work with data and users have an easy access to files of Billboard charts.

Methods

I. Data Description:

We obtained our data from the Billboard website's HTML documentation. Each entry of the data set includes a song title, an artist name, this week's rank, last week's rank, and the number of weeks it has been on the chart. As their names suggest, Billboard's Hot 100 has 100 rows, and Billboard 200 has 200 rows. The final table in Racket is a list of lists with each list containing six elements with rank as a number and the rest as strings.

II. Algorithm Description

The basic skeleton of our program can be described in several steps. We first accessed the HTML documentation of website using an external Racket library called html-parsing. It allowed us to parse the HTML of Billboard in Dr. Racket. Using our own procedures which returns entries with a given tag, we got rid of unnecessary lines and only left lines with relevant HTML tags. We noticed that entries under the same category had a same format, and we leveraged the fact to filter those entries based on their tags. We combined all lists for different categories as a single table, which is a list of lists. Using our own convert-table procedure, we converted the table into a CSV file, a table-structured formatted file.

III. Procedure Description:

The following list is comprised of descriptions of procedures we created and used:

1. Page-get:

The procedure takes in an URL in form of string and parses the HTML documentation of the webpage of given URL.

2. Tag?:

The procedure checks whether any of the element lists begin with tag by using the assoc procedure. It returns #t if the condition holds, and #f otherwise.

3. Take-table:
The procedure takes in a list of lists, and returns only entries that begin with the given tag. It recurs over the list, checks and takes entries that have their car equal to tag.

4. Filter-table:
When parsing a HTML documentation, it can create a complex data set with multiple level of lists. The algorithm works with table of multiple layers, reduces table to the desired level (determined by the tag) and takes entries that have the first element equal to tags. We use filter to filter out any non-list and null elements. If we haven't reached the desired level, 'reduce append' is used to merge and hence to get rid of outer lists. The procedure recurs over and over until the desired level is reached.

5. Leave-table:
The procedure works similarly to filter-table. However, it only reduces table to the desired level, without getting rid of entries that do not start with tag.

6. Convert-table:
The procedure is to write a csv file given a table and a source. Each element is separated by a comma, entry by a '(newline)'. The algorithm recurs over till the end of the table and closes the output port automatically.

7. Rid-of:
The procedure is to remove the last character of a string (using substring), which will be used to remove newline tag ("\n") in artists' name.

8. Song-title:
The procedure produces a list of song titles from the given URL. It works by extracting lists starting with the <h2> tag and take the element with the index 2.

9. Filter-artist:
The procedure produces a list of lists in which each line contains artist information. It works by checking an already filtered data set if its each list contains tags for an artist information.

10. Artist-name:
The procedure produces a list of corresponding artists' names by using filter-artist and taking the element of index 3.

11. Take-ranks:
The procedure filters for ranking numbers of every songs. It checks for lists that start with tag, and has the second element equal to '(@ (class "chart-row__value"))), then takes the element of index 2.

12. Three:
The procedure groups every three consecutive elements together. Since rankings of songs when extracted are not separated, this procedure is to make sure that rankings of the same song stay in the same list.

13. Current-rank:
Create the current ranking in an ascending order.

13. Print-table:
The procedure creates a complete Scheme-format table that has the first entry as column names and following entries as data for songs in the Billboard ranking.

14. Save-table:
The procedure creates a csv file for the cleaned data from the input URL, using the convert-table procedure.

Results and Discussions

I. Analysis

Since our project was creating a data product that is meant to be used for analysis, we did not have much to analyze, meaning there was no bias or objection. However, there was a discovery we found while we were building the program. The easiest way of working with HTML is to first discover a pattern, which usually has to do with the tags or IDs. By only leaving lines of codes with specific tags, we were able to quickly filter out an unnecessary portion.

II. Takeaways and Comments

We were genuinely surprised how Racket could perform web scraping tasks. We believe that Racket can do even more tasks. Although not every website provides an Application Programming Interface (API), web scraping based on API is almost always preferred way due to its easy extraction of already structured data. We end the project with an open question: How can Racket work with dynamically created tables or extract data using an API?

Instructions for running the program

1. Load the file in Dr. Racket
2. Edit the "save-to" above to set the folder for CSV file to be saved.
3. Run the program.
4. On Interactions, either type print-table to print the table or save-table to save the table as a CSV file in the directed folder.

References

M. M. (2014, December 18). The data science project lifecycle. Retrieved from https://www.datasciencecentral.com/m/blogpost?id=6448529:BlogPost:234092

HTML: Parsing Library. (n.d.). Retrieved from https://docs.racket-lang.org/html/index.html

Rebelsky, S. (n.d.). Functional Problem Solving. Retrieved from http://www.cs.grinnell.edu/~rebelsky/Courses/CSC151/2018S/home/