Class Exercise #2

The guardians of data mining

Dr. Andrea De Angelis
Spring Term 2024

Submit by <u>Friday 3th May at 19:00</u>

Introduction

In this class exercise, we are going to learn how to get data from an open API. You will get minimal guidance, in order to let you develop your independence. The exercise is a realistic scraping case: adjust your goals to your capacity and don't feel stressed or overwhelmed if you have difficulties. You are allowed to fail the class exercise and this will not be reflected in the grade, as far as I see evidence of your effort on GitHub.

Learning goals

This CE aims at experiencing some real-life scraping scenarios providing opportunities to self-learn more advanced coding and API-tapping. Moreover, you will continue familiarizing with GitHub.

API: The Guardian API

The

The API you will use is "The Guardian Open Platform": https://open-platform.theguardian.com. This is an award-winning API that opens to everyone 2 million pieces from the Guardian newspaper since 1999, including text articles, audio files and videos. The API has a free non-commercial usage requiring a registration. Check out the website for all the documentation

Tasks

1) Register to the Guardian API and get the API key. Find a way to use the API with this API key **without ever pushing the key to GitHub**.
2) **Create a corpus of articles** of choice following your instinct and interests. For instance, you may scrape all the articles about Brexit, all the articles about Switzerland, or all the articles about COVID-19, inflation… You may also work by sections, looking at cultural articles… There are endless possibilities, but start with a very specific search to avoid getting too many articles.
3) Once you created your corpus, make a basic application of choice. This could be a plot or a statistical model. For instance, you may plot the count of some keywords of interest, or simply track the number of articles over a topic over time. You can also do something a bit more advanced applying text analytics methods (a good starting point for this are quanteda tutorials: https://tutorials.quanteda.io) or calling the Open AI word embeddings of free Hugging Face embeddings.

The expected list of deliverables includes:

- A **GitHub based project** using a fully-reproducible workflow, including:
    - A short introduction to the project (2/3 lines).
    - A consistent folder structure explained on GitHub in the `readme.md`. E.g.: `brexit_articles, src,figs`…
    - Scripts with meaningful names (e.g.: `00_setup.R, 01_get_brexit.R, 02_brexit_sentiment.R`)
    - Reader-friendly code: meaningful variable names (e.g., `party_size` instead of `v21`, foldable sections and comments).
    - All activities should occur on GitHub (i.e. pushing, pulling, opening pull requests, opening issues, discussing issues with @mentions…).
    - Everyone should at least: 1. Open one issue with a proposal; 2. Create a branch to implement a new feature of the code (many commits and push); 3. Create a pull request to merge the branch into the main branch. The more the better (i.e., better making small branches for small sub-tasks).

- The GitHub repo must be public and placed inside the course organization (here, not in your personal GitHub space).

- A **short report (max 250 words**, about one page or more to fit plots**)** with the research question, hypothesis and main tests, written using RMarkdown and stored in `/output/report.Rmd`. The report must be compiled to HTML (the .html file should also be there).
- A few short **R scripts** (e.g., one to import the raw data; one to preprocess the data and save it to the dedicated folder; one to create one figure, one with the statistical test). The R Markdown document can of course use the code from these scripts.
- Report the **summary of individual contributions** on the grades' file at this link (use the tab "Class exercise #1"). Indicate the following:
  - Link to the repo and team name;
  - number of individual commits;
  - number of issues opened;
  - number of pull requests opened, accepted and merged.

Notes

- You can find these values by clicking to "Insights" and "contributors" [example link]. At [this link] you can find an **example CE** from a past edition: click on "Code" and "Download as zip" to store it on your laptop (please use the folder "output" and not "docs" to place the report as in this example).

Important: each team member should place the **data and the API key** in their local folder but should not push it to the repo. In fact, this may infringe the conditions for data release. Add the file to the **`.gitignore`** file as described here.

Deadline

Deliverables are expected **by Friday 3rd of May before 19:00**. You don't have to send me an email, I will just clone your GitHub repository.

Resources and tips

The API homepage is here: https://open-platform.theguardian.com and its full documentation is here: https://open-platform.theguardian.com/documentation/. If you feel very insecure, you can start using a pre-made wrapper functions since there are pre-made R packages to tap into the Guardian API. One is the guardianapi package (link to GitHub repo). Another possibility is the guardian package (not available on CRAN). However, you should try to do the extra mile, and I encourage you to write a custom-made wrapper function containing the HTTP request following the indications in the API documentation (you can do this!).

One tip is that you can collaborate supporting each other opening class-issues at out GitHib Q&A repository: https://github.com/UniLuFS2020-ReplicationSeminar/general-Q-A/issues. Supporting each other is always allowed, but sharing direct coding solutions is not because that would kill the learning process.

Another tip is to work developing a group "work theme" (e.g., conduct a sentiment analysis) and then applying this theme to a personal application (e.g., one member applies this to Brexit articles checking if the sentiment of the articles has evolved over time, member two applies it to AI-related articles…). This way, you can still help each other but avoid big merge conflicts since you work on separate scripts.

If you decide to write a custom wrapper function, you can refer to chapter 19 of the R4DS book, and to this httr documentation. Also don't forget my tips from CE#1:

- ***Tip #1:*** embrace **imperfection** as that thing that allows you to get things… done!
- ***Tip #2***: **collaborate** also with the other teams!
- ***Tip #3:*** good code… just works. Don't waste time trying to make you code more efficient or elegant (although this is something to keep in mind as you progress).

Assessment

The exercise is evaluated considering the individual GitHub activity as well as the report as a whole. As an individual learner, you should **make (at least) one contribution** to the group project, e.g.: analyzing a subset of articles, etc...

To this end, you must adopt the standard GitHub flow:

- Making many `git commit / git push / git pull` cycles.
- Open an issue describing a new feature you are going to implement (e.g., a new plot)
- Open a new branch connected to the issue.
- Let others check and contribute to your new feature opening a **pull request**.
- Provide feedback on a pull requests opened by your colleagues.
- Incorporate the new code into the project **merging** the pull request.

Feel free to experiment: we are just here to learn!

Code of Conduct

Remember to "be nice", intended in its widest possible sense. Declinations of this principle demand you to:

- Use welcoming and inclusive language.
- Be respectful of different viewpoints and experiences.
- Gracefully accept constructive criticism.
- Focus on what is best for the community.
- Show courtesy and respect towards other community members.

Disagreement can naturally arise during collaborative work and its management is part of job of being a good data science professional (and human being). If you believe that someone is violating this code of conduct, I ask you to report the violation to me to allow me to take appropriate action. No form of harassment will be tolerated.