This week you have no actual SQL problem, per say, but instead are just focused on setting up the initial steps of your data generation project. I'll walk you through the individual steps below, but they closely mimic what we've done already either in homework on in class demonstrations. I've included a *lot* of text here, but it is all just hopefully supporting and walking you through the same setup I showed in class.

As per usual, I am creating a starting repository for you, which is also the repository you could use with Railway.app to build your Docker container. In order for this to work, I'll need to be approving your repositories as they are created (as you accept the assignment). If, when you go to add your repo, you don't see it, check with me!

Assignment link: https://classroom.github.com/a/s9GRKuCL

1. To begin, you are going to need to write an R script which can scrape the results from a website or API of your choice. Before you can worry about writing the script though, you need to come up with an idea of what you might want to scrape! I'm trying to leave this as open as possible, so that you can pursue something of particular interest to yourself. The only real restriction is that you should plan to scrape data that is changing on at least a *mostly* daily basis. If it is something changing even more often, then you could get more data. But to get something interesting you want whatever you are scraping to at least change perhaps 5 days a week. Some ideas might include:

   - Weather data (OpenWeather has a free API)
   - Traffic data (TomTom has a freemium version that gives you *plenty* of requests per day)
   - News data (Perigon has a free version with up to 500 requests per month)
   - Currency rates
   - Sports club information
   - Event data (Ticketmaster has an API)
   - Stock information
   - Many games have an API that you can use to get information about player stats, etc.
   - The National Park Service has a free API
   - There are many APIs for tracking Covid-19 information
   - Apple's iTunes has a freely available API that can search its listings
   - There is a pretty comprehensive Space-X API on their rockets and spacecraft
   - Airline flight data

   The above are mostly APIs. Scraping websites directly may still work, but many websites that are frequently updated (and thus might be interesting to scrape) likely rely on a back-end server and probably use Javascript or similar to serve up the latest results. But if you find something interesting, pursue it!

   Then you can write your script to access the data you need. I've given you a starting template in the starting repository here. Initially I'd just store that scraped data into a data frame and print it out to ensure the data is being scraped and parsed properly. Make sure you can acquire the data you want before continuing!

2. The second part of this endeavor entails getting the information that you are scraping saved into a database. To make this as flexible as possible, we are going to set up several environment variables that you can use within your script to connect to your personal database. This method is better than hard-coding in the values directly, as not only will it keep things like passwords out of your scripts, but the environmental variables will automatically update when uploaded to the cloud, making it seamless to transition the script to writing to a different database.

   Where to set environment variables is somewhat nuanced across operating systems. If you were launching all of your programs through a shell environment, like BASH or ZSH, then the place to define environment variables would be in a file called `.bash_profile` or `.zsh_profile`, where you'd write something like

   ```
   export VARIABLE_NAME=VALUE
   ```

   This works great for programs launched from a terminal, as the contents of these files are run whenever you open a new terminal. However, if you launch programs through a different graphical method, these environmental variables never get picked up. This would be a problem for many, as they launch RStudio through a graphical launcher. Instead, you can define environment variables specifically to be used within RStudio by defining them in a special `.Renviron` file. This file should exist at

   - `/Users/<your-user-name>/.Renviron` on MacOS, or
   - `C:\Users\<your-user-name>\Documents\.Renviron` in Windows

   Note that in either case, there is *no file extension*. I might suggest just creating this file within RStudio directly. Within this file you should type a list of all the environment variables you need to define, followed by their value. Set up the following environment variables to be able to connect to your local database, substituting in your specific values where necessary:

   ```
   PGHOST="localhost"
   PGPORT=5432
   PGDATABASE="analysis"
   PGUSER="postgres"
   PGPASSWORD=""
   ```

   Once you have created this file, if you reopen RStudio and run something that looks like `Sys.getenv('PGHOST')`, then you should see `"localhost"` printed to the screen, so that you know it is working.

   At this point you can use `DBI` and `RPostgres` to make a connection object to your local database (use your environment variables by using `Sys.getenv`!). I'd suggest using `RPostgres` instead of the `RPostgreSQL` I showed in class owing to issues with the latter on Windows systems. It is otherwise a mostly drop-in replacement. Add to your scraping script the code necessary to write the data you have scraped into your local database (set up a table for this, or include the code to create the table if it doesn't exist in your script). Ensure that running your script correctly scrapes and dumps information into your local database.

   Once you have this working as a single run, wrap the scraping part of your script in an infinite loop with a `Sys.sleep(duration)` command somewhere within. At the very least, you should have your duration such that your scraping program is run once per day. (Remember that the duration is given in seconds!)

3. You now need to set up the Dockerfile (starting template included in the repo) so that it could correctly build your image. This should just be a question of creating a working directory, adding any extra necessary R libraries, copying over your R script, and then setting up the command to automatically run your script. This will likely follow very closely the example that was done in class! Once you have the Dockerfile complete, *build it on your local system* to ensure the build completes successfully. It is easier to see and troubleshoot on your local system. Note that if everything goes well and the container builds, if you launch a container with that image it should start running, but it will likely fail to add anything to your local database because we haven't set up any environment variables here for it to understand what database to connect to. That is ok though, because once we deploy it that will be fixed. Once you are sure your image builds and at least launches without issue, make sure to upload your updated script and Dockerfile back to GitHub, as you'll need those for the next steps.

   Proceed to Railway.app and use your GitHub account to sign up. Initially this will grant you with $5 of credit and 500 free hours of use per month. You will always get $5 of free credit each month, even after you upgrade your account and enter a credit card number. 500 free hours will not get you through an entire month, but for what we are doing, $5 of free credit *should*. So the system should be effectively free to you, even if you'll want to upgrade your account with a credit card at some point this month so that the 500 hour limit doesn't stop your deployment.

   Following the tutorial we went through in class, create a new project, and add a PostgreSQL database to the project. Once this is done, ensure you can connect to it from your local client, and add whatever starting table(s) you might have relied on in your scraping script. Then click to add a new component and select *GitHub Repo*. Assuming you signed in with your GitHub account, I've already approved our class organization, and hopefully have already individually approved your repo. If all went well, you should see the repo for this assignment available for you to use. If you don't, let me know! But otherwise just select that repository, and, as soon as that goes through, you should see the new component pop up in Railway, which will then say "Deploying". It might take it a handful of minutes to build and deploy the image initially, so be patient! But once it gets a green checkmark on it, check into your database to make sure the first element has been added to your table.

   Once you are sure information is being added to your table(s) in your remote database, you are done! *Make sure you check whenever the next database write would be*, to see that things are still being added!