**Efficient, Automated, and Reproducible Data Workflows in R Using “drake”**

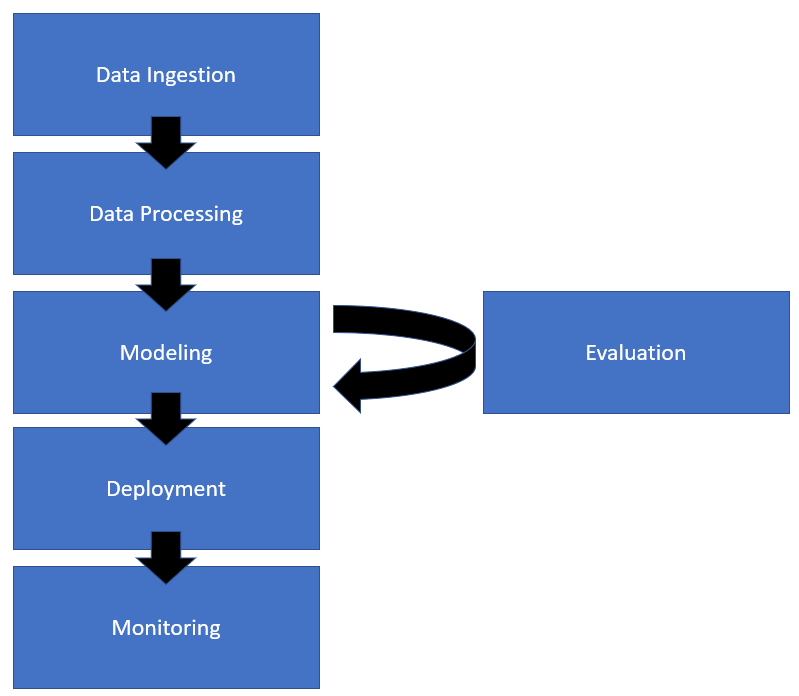
A duck floating in a body of water

Description automatically generated

**Why Use Drake?**

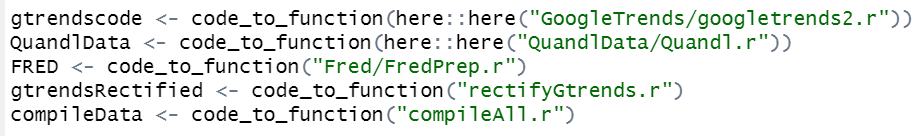
Modern analytic projects involve many steps and each individual step can have its own intricacies. When unorganized, trying to make these steps cooperate to achieve the desired result can be mystifying to all but the original writer of the code. With proper annotation an experienced outsider may be able to piece together the various scripts to achieve the result at the cost of time. One solution to make multi-step analyses in R reproducible, efficient, and automated is to use the drake package.

Below, I illustrate my take on the many steps of an analytical project, assuming the question and data necessary to answer that question have been determined. This workflow is more suited to a predictive project than an inferential project, where another step would be added for communicating inferential findings. Even with my simplifications we can still see there are many high-level steps, each of which can have sub-steps. In my example, I will focus only on data ingestion and pre-processing, which does indeed have multiple sub-steps. The zipped folder of the work can be found [here](https://github.com/jamesyoung93/DrakeWorkflow/tree/master).



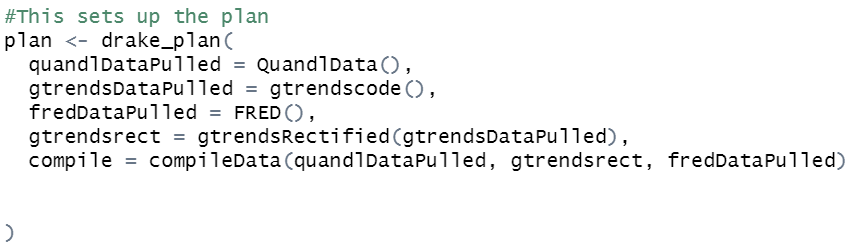
**Setting Up The Code**

Drake uses existing or user-defined functions organized in sequential steps to complete a workflow. However, if you have been using scripts prior to using drake you may wonder how to turn whole scripts to functions. Luckily, drake has a fix for this, code\_to\_function(). Below is the code I used to convert my scripts to functions. These scripts pull data in through API’s to Google, QUANDL (alternative data service), and FRED (Federal Reserve Economic Data). Then I have two extra scripts, one to fix a shortcoming in the Google search trends API, and another to compile all of the separate sources of data into a single csv file and fill in missing observations with the most recent observation.



**Organizing the plan**

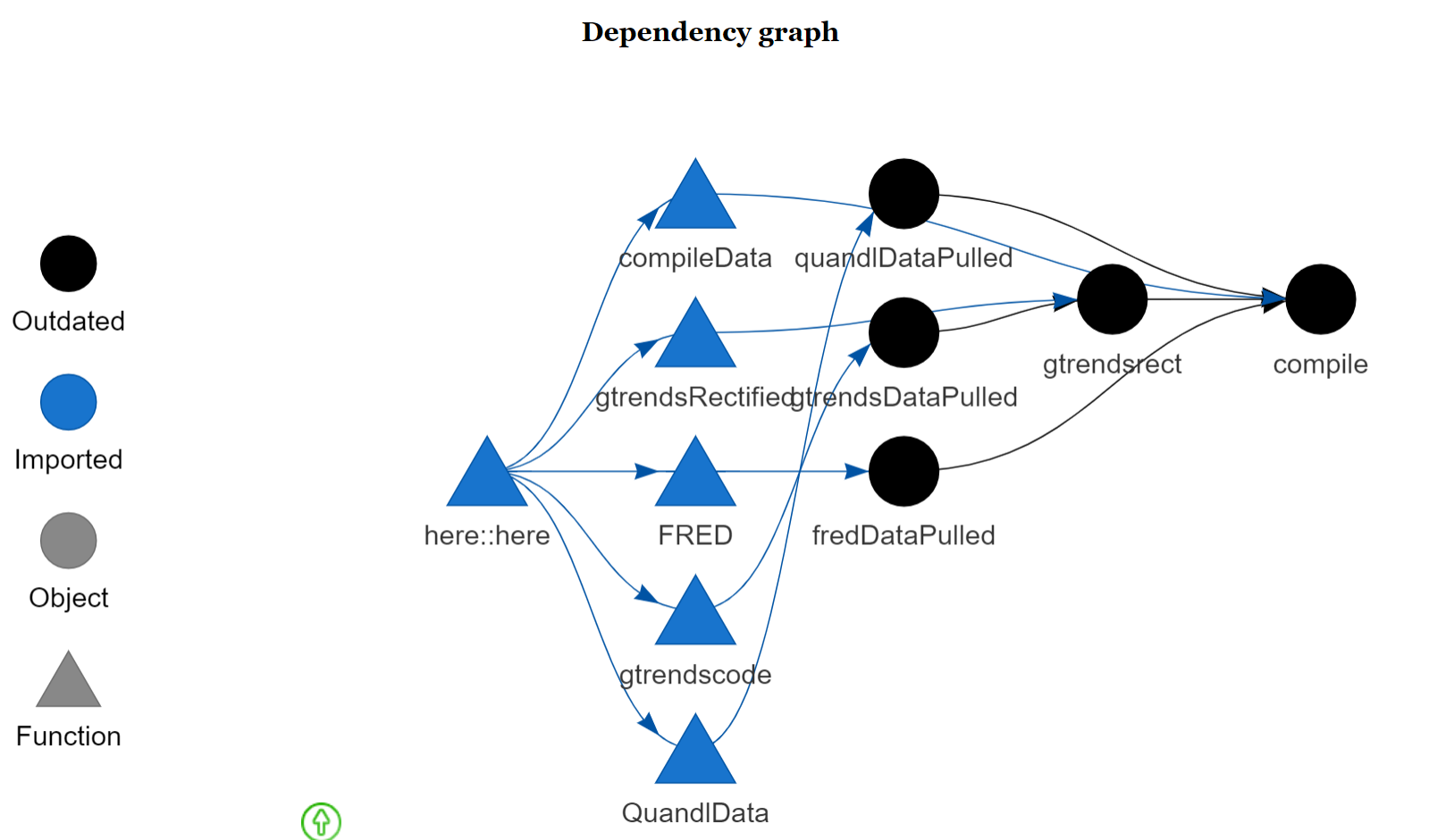
After the appropriate functions are defined, the next step is organizing them in a plan for drake to execute.



**Viewing the plan**

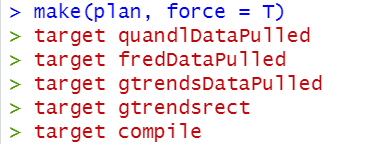
Using the command shown below we can see the workflow and its dependencies. Since we have not executed the plan yet, the steps are referred to as outdated. After executing the plan, steps that have successfully run will turn green and if a step fails it will turn red. One of the benefits of drake is that it will not repeat successfully run steps. So, if you run a workflow and many time-consuming steps run before an error occurs, you can fix the error and the workflow will start from point of the error.





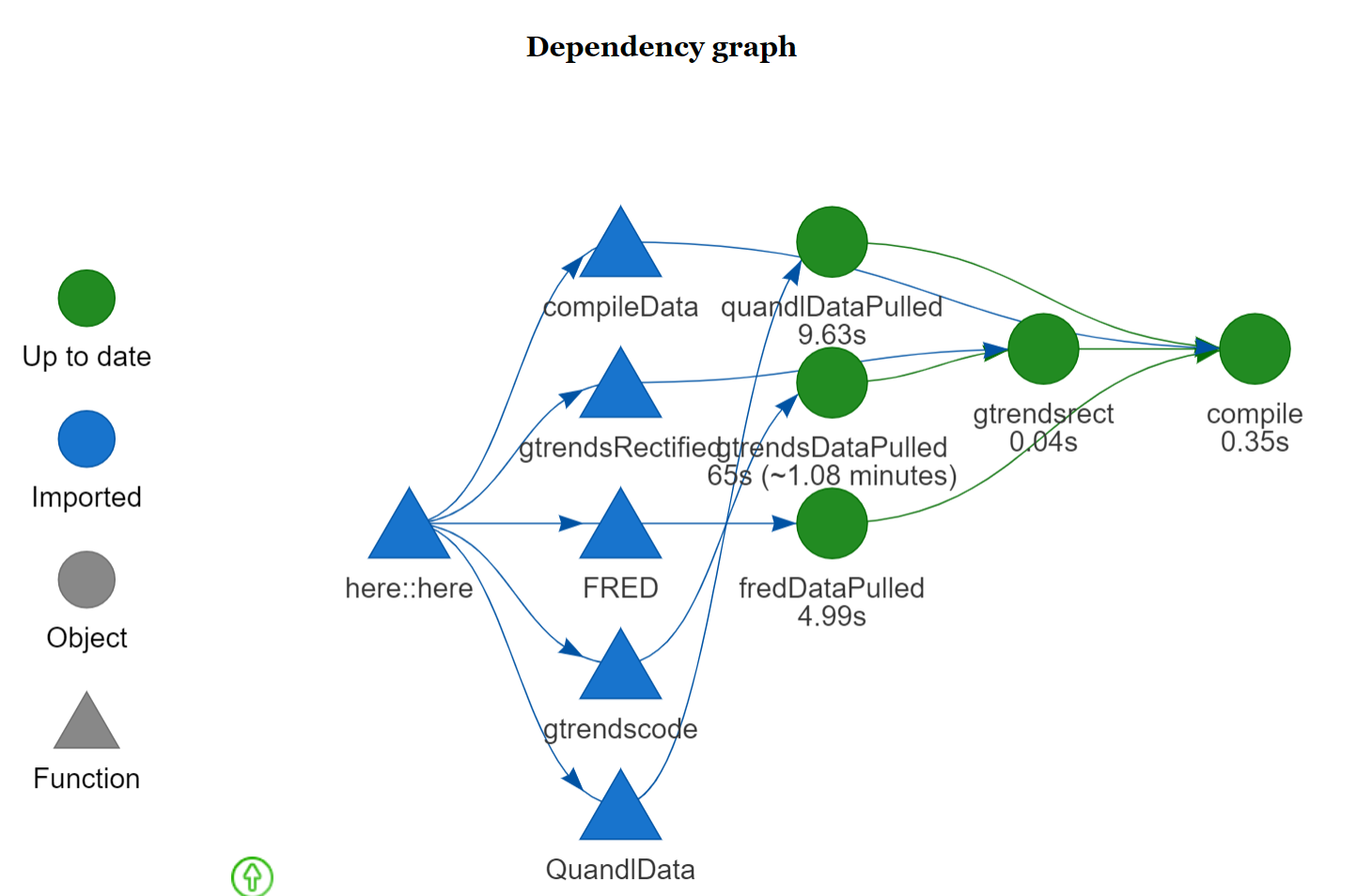
**Execute the plan**

We execute the plan with the make() function.



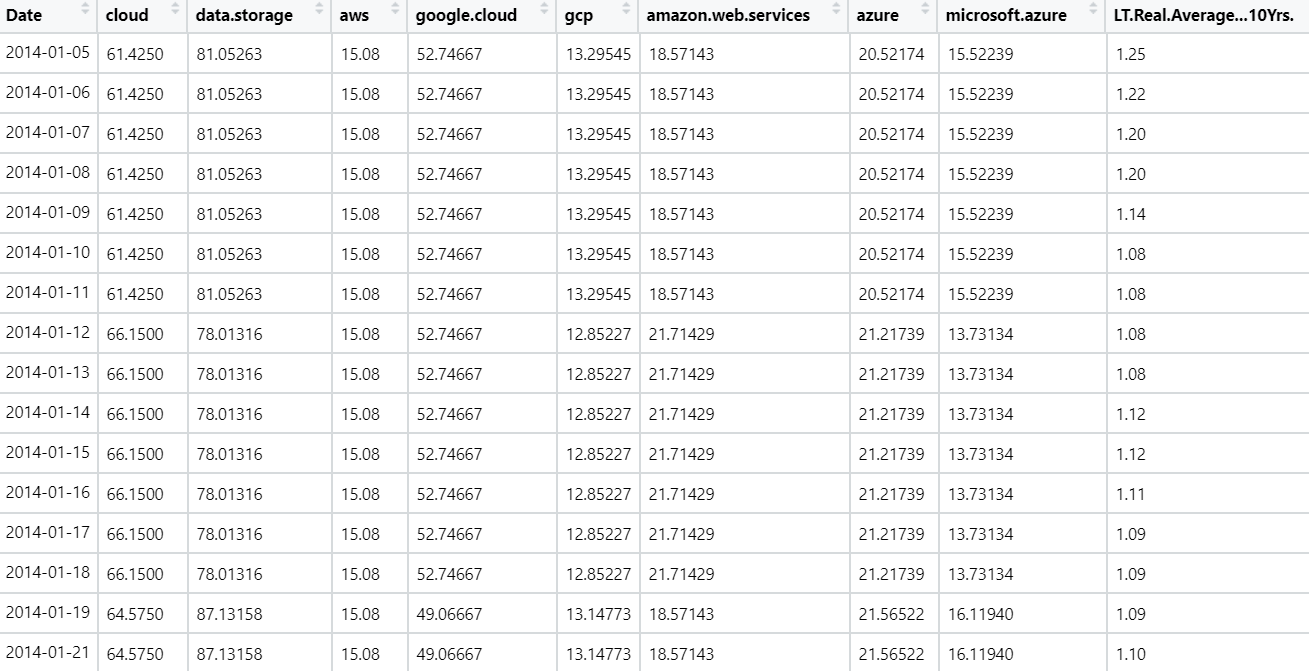
**Check the steps**

After executing the make() command, we can see that the various data ingestion and processing steps all ran successfully and are colored green (up to date).



**The Result**

Below we can see a snippet of the final csv file after ingestion and pre-processing. Again, this workflow is just an example, but this same approach could be extended to further steps in an analytic workflow.



**Closing Remarks**

When reproducibility and efficiency are key for your analytical projects, workflow tools such as drake can be the solution. Drake is freely available in the R programming language. Some alternatives to drake include the targets package, also in R, or Python flows such as Snakemake. We have covered the automation steps by making and executing the plan, the efficiency of not rerunning steps, and reproducibility in results. For more information on drake I recommend visiting the official [GitHub](https://github.com/ropensci/drake#:~:text=For%20projects%20in%20R%2C%20the,ability%20to%20trust%20your%20research.) page.