water

MDPI

Technical Note

Reproducibility Starts at the Source: R, Python, and Julia Packages for Retrieving USGS Hydrologic Data

Timothy O. Hodson [1,*,†], Laura A. DeCicco [2,†], Jayaram A. Hariharan [3], Lee F. Stanish [3], Scott Black [4] and Jeffery S. Horsburgh [5]

1 U.S. Geological Survey Central Midwest Water Science Center, Urbana, IL 61801, USA
2 U.S. Geological Survey Upper Midwest Water Science Center, Madison, WI 53726, USA; ldecicco@usgs.gov
3 U.S. Geological Survey Water Mission Area, Reston, VA 20192, USA; lstanish@usgs.gov (L.F.S.)
4 Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI), Arlington, MA 02476, USA; sblack@cuahsi.org
5 Civil and Environmental Enginnering, Utah State University, Logan, UT 84322, USA; jeff.horsburgh@usu.edu
* Correspondence: thodson@usgs.gov
† These authors contributed equally to this work.

Abstract: Much of modern science takes place in a computational environment, and, increasingly, that environment is programmed using R, Python, or Julia. Furthermore, most scientific data now live on the cloud, so the first step in many workflows is to query a cloud database and load the response into a computational environment for further analysis. Thus, tools that facilitate programmatic data retrieval represent a critical component in reproducible scientific workflows. Earth science is no different in this regard. To fulfill that basic need, we developed R, Python, and Julia packages providing programmatic access to the U.S. Geological Survey's National Water Information System database and the multi-agency Water Quality Portal. Together, these packages create a common interface for retrieving hydrologic data in the Jupyter ecosystem, which is widely used in water research, operations, and teaching. Source code, documentation, and tutorials for the packages are available on GitHub. Users can go there to learn, raise issues, or contribute improvements within a single platform, which helps foster better engagement and collaboration between data providers and their users.

Keywords: packaged workflows; water data; reproducibility; open science; open data; open source; R; Python; Julia; Jupyter; USGS

Citation: Hodson, T.O.; DeCicco, L.A.; Hariharan, J.A.; Stanish, L.F.; Black, S.; Horsburgh, J.S. Reproducibility Starts at the Source: R, Python, and Julia Packages for Retrieving USGS Hydrologic Data. *Water* **2023**, *15*, 4236. https://doi.org/10.3390/w15244236

Academic Editors: Daniel P. Ames, Gustavious Paul Williams, Huidae Cho and Xiaohui Qiao

Received: 7 November 2023
Revised: 29 November 2023
Accepted: 6 December 2023
Published: 9 December 2023

1. Introduction

Open data initiatives have pushed most scientific data to the cloud to ease accessibility such that a typical scientific workflow begins by querying a cloud database and loading the response into the computational environment for further analysis. In that paradigm, data are accessed using either some kind of graphical user interface (GUI) or by writing code to retrieve data via an application programming interface (API). Non-programmers find GUIs more intuitive, but their manual nature creates barriers to reproducibility and scalability because it can be difficult to record the exact sequence of steps within a GUI, and GUIs often change. In contrast, APIs are typically versioned, which means that code written to programmatically access an API can be executed repeatably, shared, tracked in version control, and run through automated tests, all of which are tenets of computational reproducibility and open science.

The U.S. Geological Survey (USGS) operates the largest water-monitoring network in the United States, whose data are widely used for research, as well as operationally for modeling, flood forecasting, water resources management investigations, etc. Thus, there is a great benefit to science and society in having standardized and reusable packages for

programmatically accessing USGS data, ensuring that the first step in many water science workflows—loading USGS data from the cloud—is reproducible.

There have been several efforts to improve programmatic data access of USGS data. For example, the Consortium of Universities for the Advancement of Hydrologic Science, Inc. (CUAHSI) Hydrologic Information System (HIS) project developed a web service interface that proxied the USGS National Water Information System (NWIS) database [1] (i.e., it translated web service requests to the NWIS, retrieved and parsed the resulting data, and translated the results into Water Markup Language (WaterML) format before returning the data to the user [2]). Multiple software tools were then built to use the CUAHSI HIS proxy web services to retrieve data, including HydroR for accessing data via the HIS HydroDesktop software [3,4], the waterML R package for retrieving data from HIS web services [5], and the ulmo (https://github.com/ulmo-dev/ulmo (accessed on 5 December 2023)) Python package for retrieving data from the HIS and several other sources. Many others have written custom code to automate retrieval of data from NWIS using different languages and approaches, prompting the USGS to issue documentation and recommendations for automating data retrieval from NWIS (https://waterdata.usgs.gov/nwis?automated_retrieval_info (accessed on 5 December 2023)).

Since these earlier efforts, the USGS has enhanced its web service interfaces, including adopting the model of the CUAHSI HIS and providing the option for encoding data returned from USGS web services in WaterML format. Along with these efforts to enhance their interface, the USGS and collaborators have worked to develop and support tools for accessing data via USGS web services that not only standardize data access functions and responses where possible, but also use USGS web services directly without proxy and expand access to additional datasets not available via prior tools. To that end, we developed R, Python, and Julia "dataretrieval" packages that provide programmatic access to data from any streamflow gauge, water quality monitoring station, or groundwater well, as well as other datasets available via the NWIS and the multi-agency Water Quality Portal [6]. R, Python, and Julia are open-source languages with large communities of scientific users and developers. They have become the lingua franca—the common language—of the open science movement. Notably, all three can run within Jupyter Notebooks, a web-based interactive computing platform that scientists increasingly use to explore data and communicate their findings [7], create and share reproducible workflows [8], and access data in the cloud [9]. By wrapping web service API calls in a common interface, the dataretrieval packages simplify and standardize data access across three of the standard programming languages used in water science. This simple abstraction allows users to focus on their particular science or coding problem rather than remembering details about the underlying web services, while also making their workflows simpler, more reproducible, and easier to maintain.

2. Sharing Scientific Knowledge as Reproducible Workflows

Given that this paper presents relatively simple utilities for retrieving data, we reflect on their role within the broader scientific enterprise. Fundamentally, these utilities facilitate the development of reusable packages and reproducible workflows. There is growing awareness of a reproducibility crisis in science, e.g., [10] by one estimate, 95 percent of recent hydrology and water resources publications cannot be reproduced [11]. In response, many within the scientific community are advocating for greater transparency and reproducibility of research results. Journals increasingly require submissions to be accompanied by data, code, and other research artifacts that enable the reproduction of the analyses and results. However, the original code and data are insufficient to ensure reproducibility; one also needs the original computational environment, or at least the means to recreate it.

A *package* is an archive of software along with metadata intended to make the software more easily shared and reused by others [12]. It is essentially a set of software tools that may be reused to accomplish different computational tasks, either by expanding the functionality of other packages or by performing a particular task, such as in a *workflow*.

A *workflow* is a sequence of steps that produces a particular result. A recipe for baking bread is a workflow, but in this context, we mean workflows that run in a computational environment, known as computational workflows. Often, workflows that begin as notebooks or scripts go on to be developed into packages that more formally organize and codify a set of functionality along with scientific knowledge for reuse by others. Just as in open-source software development, packages are fundamental organizational units within open science, where researchers contribute expertise to help develop packages, and then use and combine those packages to create flexible and reproducible workflows. In this regard, one might consider the development and availability of scientific code packages to be a revolution in scientific philosophy (metascience). Recent advances in machine learning, data science, and many other domains have been accelerated through the availability of open-source packages [13,14].

The principal purpose of a package is reusability: If one researcher writes a package to accomplish *X*, then another researcher can use that package to accomplish *X* without having to write the code themselves. There is also an expectation that packages evolve as code and knowledge are contributed over time. A workflow is, in essence, another type of package but its purpose and lifecycle differ. Once published, a workflow is intended as a static archive; its principal purpose is to ensure reproducibility. To achieve that, workflows adopt many of the same tools and practices used in software packaging, such as dependency resolvers to reproduce a particular computation environment, version control, automated testing, and open web-based publication platforms, etc.

A *packaged workflow* combines both concepts by using computer science tools and practices in a manner that allows it to easily migrate from one computational environment to another. Such workflows are becoming an increasingly important component of scientific communication. An example is HydroShare [15,16], which is an online repository that supports the sharing and publication of packaged workflows. Using HydroShare, a researcher can upload a Jupyter Notebook containing their workflow and then share it publicly or permanently publish it with a citable digital object identifier (DOI). Anyone can then rerun the notebook using HydroShare's linked JupyterHub environment.

GitHub increasingly serves a similar role as a general research platform, and is a perfect example of how scientific research is adopting and adapting to software development practices. On GitHub, researchers can develop and publish their packages and workflows, use automated tests to ensure they work as intended, provide pre-configured computational environments (called Codespaces), and engage in collaborative research and development.

The importance of the packages described in this paper, as well as others like `waterData` [17], `waterML` [5], `HyRiver` [18], and `hydroloom` [19], is that they facilitate programmatic data access, which is a key component in creating open and reproducible scientific workflows that are distributed on platforms like HydroShare and GitHub.

3. Design and Functionality

The data retrieval packages were designed to operate within the context of the services-oriented architecture provided by the USGS' NWIS, which provides a set of "water services" that enable automated retrieval of USGS data encoded in extensible markup language (XML) and other data encodings (Table 1). These services generally correspond with data types or products produced by the USGS (e.g., an Instantaneous Values Service for retrieving current streamflow and other real-time data for USGS monitoring sites, a Groundwater Levels Service for retrieving historical, manually recorded groundwater levels from monitoring sites, etc.). These services are REST-friendly, meaning they are accessible via URLs and can be called from any programming language or environment. However, understanding the specific URL syntax and query parameters to make requests and writing code to interpret the results can be challenging. The data retrieval packages provide convenience functions within each programming language that abstract the construction of the required URLs, handle the interpretation and parsing of the returned data format, and generally make working with the USGS' web services easier (Figure 1).

Additionally, using similar function names, argument names, and responses (where possible) across the different packages makes it easier for programmers to work across languages or to choose the language that is most appropriate for their current application. Helping to abstract data access enables users to focus on their particular coding task or science question rather than learning and remembering the details of the underlying web services. For example, consider retrieving streamflow values for the most recent day for a single USGS streamflow gage in Python.

```
site_code = "10109000"  # USGS site code
parameter = "00060"  # USGS code for discharge
start_date = date.today().isoformat()
end_date = date.today().isoformat()
```

To make such a request, a programmer needs to remember that the Python requests library retrieves data using a URL, how to formulate the URL required to retrieve the data from NWIS using service-specific query parameters, the data structure of the response, and how to parse that into a Pandas dataframe.

```
import requests
import pandas as pd
from datetime import date

# Build a URL to retrieve the unit discharge values
url = (f"https://waterservices.usgs.gov/nwis/iv/?format=json&sites="
       f"{site_code}&startDT={start_date}&endDT="
       f"{end_date}&parameterCd={parameter}")
response = requests.get(url)

# Extract discharge data from the response
discharge_data = response.json()

# Create a Pandas dataframe from the JSON data
df = pd.DataFrame(discharge_data["value"]["timeSeries"][0]
                  ["values"][0]["value"])

# Convert dateTime column to datetime index and value column to numeric
df["dateTime"] = pd.to_datetime(df["dateTime"])
df.set_index("dateTime", inplace=True)
df["value"] = pd.to_numeric(df["value"])
```

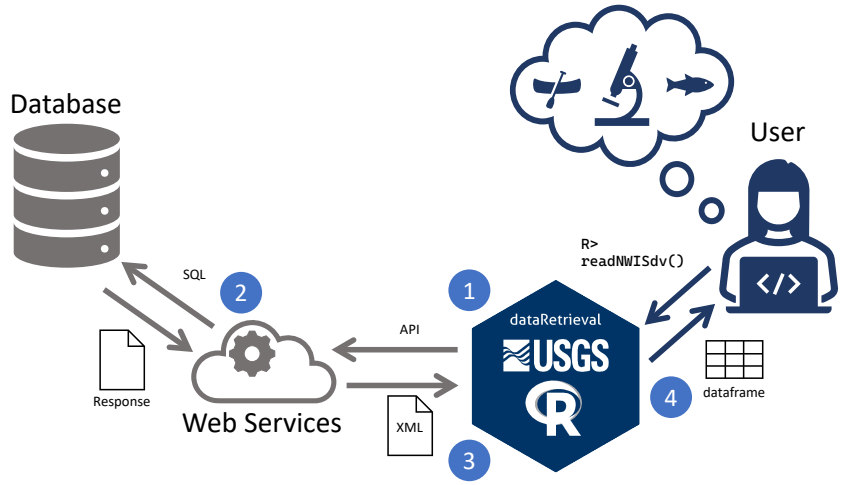

Figure 1. Information flow for data retrieval packages. A data consumer uses a data retrieval package to make a request for data to the appropriate USGS web service (1). The service handles the request by querying the underlying NWIS database to retrieve the requested data (2), and then encoding the results in XML, text, or JSON format for return to the user (3). The data retrieval package then parses the result (4) and loads it into a performant data structure for analysis (e.g., a dataframe object in R or Pandas for Python).

In contrast, the dataretrieval package provides a significant simplification to the workflow. Primarily, the programmer must remember that the `get_iv` function retrieves instantaneous values data from NWIS, but in practice the function name, as well as its functionality, can be found within the Python interpreter via built-in functions (e.g., `dir()` and `help()`).

```
from dataretrieval import nwis

df, metadata = nwis.get_iv(sites=site_code,
                           start=start_date,
                           end=end_date,
                           parameterCd=parameter)
```

Even experienced users who are familiar with the web services may write custom code to manage data retrieval. We still encourage that practice, but it is also our justification for organizing data retrieval code into community- or organization-developed packages. A brief overview of the three USGS data retrieval packages' functionality is given here, but for more detailed information, refer to the package documentation. The core web services supported by the USGS data retrieval packages are listed in Table 1, though these may change over time.

Table 1. Core web services supported by the USGS data retrieval packages. Several services are not yet supported (NS) but may be added in future versions.

Web Service	Description	Function Names (R; Python; Julia)
Daily Values [1] (https://waterservices.usgs.gov/docs/dv-service/daily-values-service-details/)	Daily data available at USGS water sites including mean, median, maximum, minimum, and/or other derived values	readNWISdv; nwis.get_dv; readNWISdv
Groundwater Levels (https://waterservices.usgs.gov/docs/groundwater-levels/groundwater-levels-details/)	Historical manually-recorded groundwater levels from hydrologic sites served by the USGS	readNWISgwl [2]; nwis.get_gwlevels; NS
Instantaneous Values (https://waterservices.usgs.gov/docs/instantaneous-values/instantaneous-values-details/)	Current streamflow, gage height, and hundreds of other real-time data	readNWISunit; nwis.get_iv; readNWISiv
Measurements (https://waterdata.usgs.gov/nwis/measurements/)	Manual measurements of streamflow and gage height. These measurements are used to supplement and (or) verify the accuracy of the automatically recorded observations, as well as to compute streamflow based on gage height	readNWISmeas; nwis.get_discharge; NS
Parameter Codes (https://help.waterdata.usgs.gov/codes-and-parameters)	Lookup 5-digit codes used to identify measurement types	readNWISpCode; nwis.get_pmcodes; readNWISpCode
Peaks (https://nwis.waterdata.usgs.gov/usa/nwis/peak)	Annual maximum instantaneous peak streamflow and gauge height	readNWISpeak; nwis.get_peaks; NS
Ratings (https://waterdata.usgs.gov/nwisweb/get_ratings/)	Stage-discharge rating tables for USGS streamgages	readNWISrating; nwis.get_ratings; NS
Site https://waterservices.usgs.gov/docs/site-service/site-service-details/	Site information including location coordinates	readNWISsite; nwis.get_info; readNWISsite
Statistics (https://waterservices.usgs.gov/docs/statistics/statistics-details/)	Daily, monthly or annual statistics for sites	readNWISstat; nwis.get_stats; NS
Water Use (https://waterdata.usgs.gov/nwis/water_use)	Water use data collected by local, State, and Federal agencies as well as academic and private organizations	NS; nwis.get_water_use; NS
Water Quality (https://www.waterqualitydata.us/)	Publicly available water-quality data from USGS, the Environmental Protection Agency, and over 400 state, federal, tribal, and local agencies	readWQPdata [3]; wpq.get_results [2]; readWQPdata [2]

Notes: [1] All URLs accessed on 5 December 2023. [2] The R package also retrieves groundwater data from the National Groundwater Monitoring Network. [3] The Water Quality Service has additional functions for different types of queries; refer to the documentation for details.

3.1. Usage Examples

This section provides brief usage demonstrations in each of the three languages: R, Python, and Julia. A typical analysis workflow would be more complicated, but these highlight some common use cases. On one hand, they are trivial—simple enough to be

understood by non-programmers—but also illustrate the value of abstraction. In each case, several high-level packages are used together: each abstracting away some of the complexity to yield a simple workflow. For many more examples and tutorials see the links to the package documentation in the Data Availability Section.

R

Of the three packages, the R version, `dataRetrieval` was developed first and has been downloaded over 195,000 times (as of November 2023; [20]). Along with simplifying workflows, its functionality has become integral in other packages like `EGRET` (Exploration and Graphics for RivEr Trends), which provides utilities for the analysis of long-term changes in water quality and streamflow [21]. Several `EGRET` functions use `dataRetrieval` to retrieve data, and then preprocess the output into an analysis-ready format. A typical `EGRET` workflow retrieves data, calibrates a model, and displays long-term trend calculations. Here we use it to retrieve orthophosphate data from an USGS monitoring location (01631000), then model and plot the orthophosphate load through time (Figure 2). Using `dataRetrieval`, both `EGRET` and the workflow are simpler and, therefore, easier to understand, use, and maintain.

```
library(EGRET)
site <- "01631000"
parameter <- "00660" # USGS code for orthophosphate
Sample <- readNWISSample(site, parameter)
Daily <- readNWISDaily(site,
                       startDate = min(Sample$Date))
INFO <- readNWISInfo(site, parameter,
                     interactive = FALSE)
eList <- mergeReport(INFO, Daily, Sample)
eList <- modelEstimation(eList, verbose = FALSE)
plotConcHist(eList, printTitle=FALSE)
```

3.2. Python

Jupyter's interactivity is often cited as being key for rapid prototyping and exploratory data analysis [7]. Here, we demonstrate a common usage pattern for data exploration, which uses `dataretrieval` to query sites matching a particular criteria, then displays the results in an interactive webmap using `hvplot` (Figure 3).

```
from dataretrieval import nwis
import geopandas as gpd
import hvplot.pandas

parameter = "00665" # USGS code for total phosphorus
df, meta = nwis.what_sites(stateCd="IL", parameterCd=parameter)
geometry = gpd.points_from_xy(df.dec_long_va, df.dec_lat_va)
gdf = gpd.GeoDataFrame(df, geometry=geometry)

gdf.hvplot.points(geo=True, hover_cols=["site_no", "station_nm"],
                  tiles=True, width=300, size=3)
```

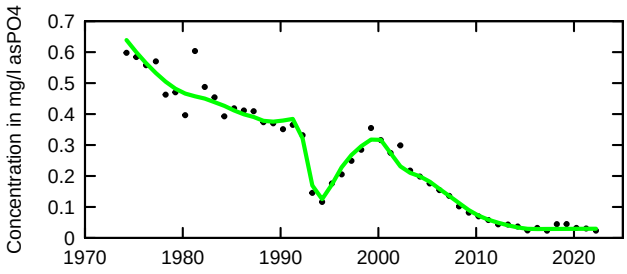

Figure 2. `EGRET` generated time series of flow-normalized concentration of orthophosphate (PO4) in miligrams per liter (mg/L) for the South Fork Shenanndoah River at Front Royal, Virginia. Dots depict the annual mean concentration.

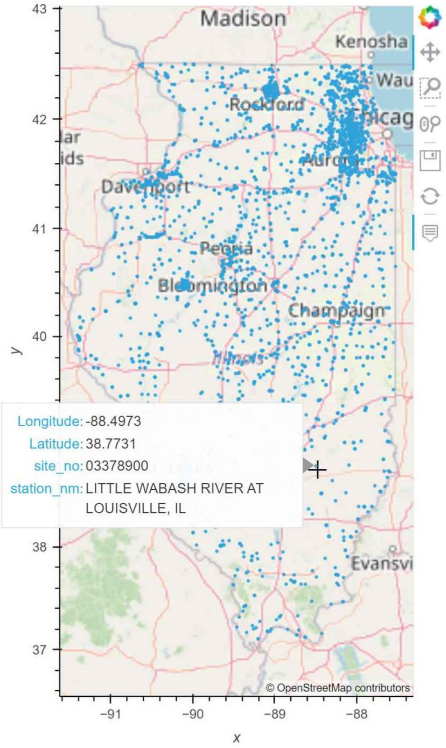

Figure 3. Interactive web map displaying locations in Illinois with phosphorus samples.

3.3. Julia

As the youngest programming language and data retrieval package, our Julia demonstration is more introductory. We use `DataRetrieval.jl` to retrieve annual groundwater levels from a single site in Delaware, then compute summary statistics on an annual basis using the `Statistics` package [22] and format the output for publication using `Latexify` [23] (Table 2).

```
using DataRetrieval, Dates, Statistics, DataFrames, Latexify

site = "393617075380403"
parameter = "72019" # USGS code for depth to water level
df, response = readNWISdv(site,
                          parameter,
                          startDate="1776-07-04",
                          endDate="2022-12-31",
                          format="json");
df.datetime = Dates.DateTime.(df.datetime, "yyyy-mm-ddTHH:MM:SS.SSS");
df.year = Dates.year.(df.datetime);
df2 = combine(groupby(df, :year),
              parameter => minimum => :Minimum,
              parameter => maximum => :Maximum,
              parameter => mean => :Mean;

latexify(df2, env=:table) |> print
```

Table 2. Annual (calendar year) summary statistics for groundwater levels (depth to water level in feet below land surface) at U.S. Geological Survey site 393617075380403 in Delaware.

Year	Minimum	Maximum	Mean
2012	−0.27	−0.0	−0.11
2015	−1.2	−0.03	−0.26
2016	−0.6	0.01	−0.2
2020	−0.6	0.12	−0.22
2021	−0.38	0.1	−0.16
2022	−0.44	0.15	−0.045

4. Usage Scenarios

As basic utilities, the data retrieval packages support a range of usage scenarios. For scientific research and publishing, they automate hydrologic data retrieval in workflows such that data access can be encoded in scripts or notebooks that can be shared, re-run, and built upon by other researchers. By reducing the time and effort required for retrieving and loading data into analysis-ready structures like "data frames", they also lower barriers for novice users and enable experienced users to dedicate more energy to research.

The packages serve a similar role for water resource professionals who depend on USGS data for operational purposes including flood forecasting and warning, operating dams and hydraulic control structures, bridge design, allocating irrigation water, planning for energy development, assessing water quality and pollution, and more. While our focus has been on the reproducibility of scientific work, practitioners also need transparent, reliable, and reproducible modeling and analysis workflows.

In the classroom, instructors use these packages to teach data science and hydroinformatics concepts, which are becoming increasingly important skills as scientific and engineering work becomes more data-intensive. A growing part of hydrologic science is shifting from collecting data for testing or supporting existing conceptual models toward analyses based on models derived from observational data [24]. A recent survey of hydroinformatics or water-data-science instructors found that most incorporate basic programming, data formatting and wrangling, visualization and plotting, and other data science topics into their courses [25]. Nearly all of them used Python or R in their course materials, and several used one of the data retrieval packages. Feedback from that survey was used in developing the Hydroinformatics and Water Data Science module on HydroLearn, which also uses the Python `dataretrieval` [26].

5. Conclusions

Software packages that facilitate programmatic data retrieval are a critical component in open and reproducible scientific workflows. However, scientific data providers typically have limited resources and serve large user communities. In this situation, the best way to serve a community can be to leverage it: By adopting open-source practices, data providers can better engage and organize their users to collaboratively develop software that serves the community as a whole.

As a prime example, that open-source model enabled development of the R, Python, and Julia dataretrieval packages. These three programming languages are used extensively in scientific computing and data science, and are the core languages supported by Jupyter Notebooks, a web-based computing platform widely used to teach programming and develop scientific workflows. By wrapping web service API calls in a common interface, these packages simplify data access, allowing users to focus on their particular science or coding problem rather than remembering details about the underlying web services. Although this is a relatively simple abstraction, by serving many users it provides substantial benefit by making workflows simpler and more reproducible. The paper gives an overview of the USGS dataretrieval packages along with some simple examples of their usage. To learn more or to contribute, refer to the GitHub repositories linked in the Data Availability Section.

Author Contributions: Conceptualization, T.O.H., L.A.D. and J.S.H.; software, T.O.H., L.A.D., J.A.H. and S.B.; writing—original draft, T.O.H., L.A.D., J.A.H. and J.S.H.; project administration, L.F.S.; writing—review and editing, L.F.S. All authors have read and agreed to the published version of the manuscript.

Funding: This material is partially based upon work supported by the National Science Foundation (NSF) under award 1931297. Any opinions, findings, conclusions, or recommendations expressed in this material do not necessarily reflect the views of the NSF.

Data Availability Statement: Source code, documentation, and tutorials for each package are available from their respective GitHub repositories: R (https://github.com/DOI-USGS/dataRetrieval), Python (https://github.com/DOI-USGS/dataretrieval-python), and Julia (https://github.com/DOI-USGS/dataretrieval.jl). Users are encouraged to raise issues and contribute improvements to the packages via GitHub. For easy installation, packaged versions are also available on CRAN (for R), PyPI and conda-forge (for Python), and `Pkg` (Julia's built-in package manager). Please cite this paper when discussing the software in an abstract sense or other ideas from the paper. When using the software, we recommend citing the specific version and its associated software release. For example, the R, Python, and Julia versions used in the paper are available as software releases [20,27,28], respectively. The Python example was run in Jupyter on Windows Subsystem for Linux 2 (WSL2) with an Intel processor. The supplemental environment.yml (https://raw.githubusercontent.com/DOI-USGS/dataretrieval-python/paper-env/demos/webmap/environment.yml (accessed on 5 December 2023)) contains the package metadata to reproduce our Python computational environment. As with all the examples, different package managers, operating systems, and hardware may yield different results. If you are unable to reproduce the examples, please raise an issue on the relevant GitHub repository.

Acknowledgments: This work was conducted as part of the USGS Integrated Water Availability Assessments (IWAAs) Program, which examines the spatial and temporal distribution of water quantity and quality in both surface and groundwater, as related to human and ecosystem needs and as affected by human and natural influences. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

Conflicts of Interest: Authors T.O.H., L.A.D., J.A.H., and L.F.S. were employed by the U.S. Geological Survey. Author S.B. is employed by CUAHSI. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

USGS	United States Geological Survey
WQP	Water Quality Portal
GUI	Graphical User Interface
API	Application Programming Interface
CRAN	Comprehensive R Archive Network
PyPI	Python Package Index

References

1. U.S. Geological Survey. National Water Information System Data Available on the World Wide Web (USGS Water Data for the Nation). 2023. Available online: https://waterdata.usgs.gov/nwis (accessed on 5 December 2023).
2. Goodall, J.L.; Horsburgh, J.S.; Whiteaker, T.L.; Maidment, D.R.; Zaslavsky, I. A first approach to web services for the National Water Information System. *Environ. Model. Softw.* **2008**, *23*, 404–411. [CrossRef]
3. Horsburgh, J.S.; Reeder, S.L. Data visualization and analysis within a Hydrologic Information System: Integrating with the R statistical computing environment. *Environ. Model. Softw.* **2014**, *52*, 51–61. [CrossRef]
4. Ames, D.P.; Horsburgh, J.S.; Cao, Y.; Kadlec, J.; Whiteaker, T.; Valentine, D. HydroDesktop: Web services-based software for hydrologic data discovery, download, visualization, and analysis. *Environ. Model. Softw.* **2012**, *37*, 146–156. [CrossRef]
5. Kadlec, J.; StClair, B.; Ames, D.P.; Gill, R.A. WaterML R package for managing ecological experiment data on a CUAHSI HydroServer. *Ecol. Inform.* **2015**, *28*, 19–28. [CrossRef]
6. National Water Quality Monitoring Council. Water Quality Portal. 2023. Available online: https://www.waterqualitydata.us/ (accessed on 5 December 2023).
7. Granger, B.E.; Pérez, F. Jupyter: Thinking and Storytelling With Code and Data. *Comput. Sci. Eng.* **2021**, *23*, 7–14. [CrossRef]

8. Beg, M.; Taka, J.; Kluyver, T.; Konovalov, A.; Ragan-Kelley, M.; Thiéry, N.M.; Fangohr, H. Using Jupyter for Reproducible Scientific Workflows. *Comput. Sci. Eng.* **2021**, *23*, 36–46. [CrossRef]
9. Abernathey, R.P.; Augspurger, T.; Banihirwe, A.; Blackmon-Luca, C.C.; Crone, T.J.; Gentemann, C.L.; Hamman, J.J.; Henderson, N.; Lepore, C.; McCaie, T.A.; et al. Cloud-Native Repositories for Big Scientific Data. *Comput. Sci. Eng.* **2021**, *23*, 26–35. [CrossRef]
10. Baker, M. 1500 scientists lift the lid on reproducibility. *Nature* **2016**, *533*, 452–454. [CrossRef] [PubMed]
11. Stagge, J.H.; Rosenberg, D.E.; Abdallah, A.M.; Akbar, H.; Attallah, N.A.; James, R. Assessing data availability and research reproducibility in hydrology and water resources. *Sci. Data* **2019**, *6*. [CrossRef] [PubMed]
12. Hillard, D. *Publishing Python Packages*; Manning: Shelter Island, NY, USA, 2023; p. 248.
13. Nguyen, G.; Dlugolinsky, S.; Bobák, M.; Tran, V.; García, Á.L.; Heredia, I.; Malík, P.; Hluchý, L. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: A survey. *Artif. Intell. Rev.* **2019**, *52*, 77–124. [CrossRef]
14. Langenkamp, M.; Yue, D.N. How Open Source Machine Learning Software Shapes AI. In Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society, Oxford, UK, 19–21 May 2021. [CrossRef]
15. Tarboton, D.G.; Horsburgh, J.S.; Idaszak, R.; Heard, J.; Ames, D.; Goodall, J.L.; Band, L.; Merwade, V.; Couch, A.; Arrigo, J.; et al. HydroShare: Advancing Collaboration through Hydrologic Data and Model Sharing. In Proceedings of the 7th International Congress on Environmental Modelling and Software, San Diego, CA, USA, 15–19 June 2014. [CrossRef]
16. Horsburgh, J.S.; Morsy, M.M.; Castronova, A.M.; Goodall, J.L.; Gan, T.; Yi, H.; Stealey, M.J.; Tarboton, D.G. HydroShare: Sharing Diverse Environmental Data Types and Models as Social Objects with Application to the Hydrology Domain. *JAWRA J. Am. Water Resour. Assoc.* **2016**, *52*, 873–889. [CrossRef]
17. Ryberg, K.R.; Vecchia, A.V. *waterData—An R Package for Retrieval, Analysis, and Anomaly Calculation of Daily Hydrologic Time Series Data*, Version 1.0; U.S. Geological Survey: Reston, VA, USA, 2012. Available online: https://pubs.usgs.gov/of/2012/1168 (accessed on 5 December 2023).
18. Chegini, T.; Li, H.Y.; Leung, L.R. HyRiver: Hydroclimate Data Retriever. *J. Open Source Softw.* **2021**, *6*, 3175. [CrossRef]
19. Blodgett, D. Hydroloom: Utilities to Weave Hydrologic Fabrics, Version 1.0. Available online: https://CRAN.R-project.org/package=hydroloom (accessed on 5 December 2023).
20. De Cicco, L.A.; Lorenz, D.; Hirsch, R.M.; Watkins, W.; Johnson, M. *DataRetrieval: R Packages for Discovering and Retrieving Water Data Available from U.S. Federal Hydrologic Web Services*; U.S. Geological Survey: Reston, VA, USA, 2023. [CrossRef]
21. Hirsch, R.M.; Moyer, D.L.; Archfield, S.A. Weighted Regressions on Time, Discharge, and Season (WRTDS), with an Application to Chesapeake Bay River Inputs. *JAWRA J. Am. Water Resour. Assoc.* **2010**, *46*, 857–880. [CrossRef] [PubMed]
22. JuliaStats Contributors. Statistics.jl: The Statistics Stdlib That Ships with Julia. 2023. Available online: https://juliastats.org/Statistics.jl/dev/ (accessed on 5 December 2023).
23. Korsbo, N.; Other Contributors. Latexify.jl: Functions for Producing LaTeXFormatted Strings from Julia Objects. 2023. Available online: https://korsbo.github.io/Latexify.jl/stable/ (accessed on 5 December 2023).
24. Chen, Y.; Han, D. Big data and hydroinformatics. *J. Hydroinform.* **2016**, *18*, 599–614. [CrossRef]
25. Jones, A.S.; Horsburgh, J.S.; Pacheco, C.J.B.; Flint, C.G.; Lane, B.A. Advancing Hydroinformatics and Water Data Science Instruction: Community Perspectives and Online Learning Resources. *Front. Water* **2022**, *4*, 901393. [CrossRef]
26. Jones, A.S.; Horsburgh, J.S.; Pacheco, C.J.B. Hydroinformatics and Water Data Science. 2022. Available online: https://edx.hydrolearn.org/courses/course-v1:USU+CEE6110+2022/about (accessed on 5 December 2023).
27. Hodson, T.O.; Hariharan, J.A.; Black, S.; Horsburgh, J.S. *Dataretrieval (Python): A Python Package for Discovering and Retrieving Water Data Available from U.S. Federal Hydrologic Web Services*; U.S. Geological Survey: Reston, VA, USA, 2023. [CrossRef]
28. Hariharan, J.A. *DataRetrieval.jl—Julia Package for Obtaining USGS Water Data Directly from Web Services*; U.S. Geological Survey: Reston, VA, USA, 2023. [CrossRef]