Importing Data in R Directly From the Internet Using APIs: Exploring Various Packages

Mohammed Tariq Azeez A K

MBA Business Analytics - Matriculation No. 400983409, Hochschule Fresenius University of Applied Sciences

Author Note

Correspondence concerning this article should be addressed to Mohammed Tariq Azeez A K, Email: tariq.azeez.2024@gmail.com

Abstract

This handout complements the presentation on importing data in R directly from the internet using web APIs. It introduces the basic ideas behind HTTP-based APIs, shows how typical response formats such as JSON can be parsed in R, and outlines a reproducible workflow using httr2, jsonlite, and tidyverse. A worked example with the GitHub REST API illustrates the full pipeline from request construction to data wrangling and interpretation. The handout concludes with common pitfalls, best practices, and a short template exercise for further practice.

Keywords: R, APIs, data import, httr2, jsonlite, tidyverse

Importing Data in R Directly From the Internet Using APIs: Exploring Various Packages

1 Introduction

Modern business analytics rarely relies only on static CSV or Excel files that are downloaded once and then analysed in isolation. Instead, many real-world projects require **up-to-date information** from online platforms, financial services, or open data portals. Web APIs (Application Programming Interfaces) are the standard way to access these data sources programmatically.

In the course *Data Analysis for Decision-Making (WS 2025/26)*, this topic focuses on how to use R to import data directly from the internet via web APIs. The goal of this handout is to provide a compact but complete overview that supports the presentation:

- What web APIs are and why they matter.
- How JSON responses can be turned into usable R objects.
- Which R packages are most useful for API workflows.
- A small, realistic example using the GitHub API.
- Challenges and best practices when working with APIs.
- A short template for an exercise you can adapt to other APIs.

2 Web APIs and Response Formats

2.1 What is a web API?

A **web API** is an interface that allows one piece of software (for example, an R script) to request data from another system over the internet. Most modern APIs are **HTTP-based** and use standard methods such as:

- `GET` – retrieve data (read-only),
- `POST` – send or create data,
- `PUT` / `PATCH` / `DELETE` – update or delete data

APIs expose **endpoints**, which are specific URLs that provide a certain kind of information. Examples include:

- `https://api.github.com/repos/{owner}/{repo}`,
- `https://api.exchangerate.host/latest`,
- `https://api.coindesk.com/v1/bpi/currentprice.json`.

Each endpoint has documented parameters (for example, which currency to use or how many results to return) and a clear specification of what the response will look like.

2.2 JSON as the dominant format

Most modern APIs respond with data in **JSON (JavaScript Object Notation)**. Conceptually, JSON is a nested structure of:

- objects (key–value pairs),
- arrays (ordered lists),
- primitive values (numbers, strings, logicals, or null).

From R's perspective, JSON is naturally represented as nested lists. A simplified example response might look like:

```
{
  "time": "2025-11-19T07:00:00Z",
  "value": 123.45,
  "meta": {
    "source": "example-api",
    "units": "index points"
  }
}
```

In an R workflow, we typically:

1. Send a request and receive the raw HTTP response.
2. Extract the body as text or as JSON.

3. Parse the JSON into R objects using **jsonlite**.
4. Convert the nested lists into a tidy tibble or data frame.

Other formats such as XML or CSV still appear in some APIs, but JSON has become the default in many modern services.

3 R Packages and Basic Workflow

3.1 Core packages

For this topic, a small set of R packages covers most needs:

- **httr2** – a modern HTTP client for building, sending, and inspecting requests and responses.
- **jsonlite** – a robust JSON parser that converts JSON into lists or data frames.
- **dplyr**, **tidyr**, **purrr** – tidyverse tools for wrangling and transforming nested data structures.
- **ggplot2** – for visualising the results of the analysis.

These packages can be combined into a simple but powerful workflow.

3.2 A generic API workflow in R

The basic pattern for working with an HTTP API in R looks like this:

The code itself is usually short. The main intellectual work lies in understanding the **structure** of the JSON and designing a tidy representation that is suitable for analysis.

4 Worked Example: GitHub Repository Metadata

To connect the presentation and this handout, we use the same simple example: querying the GitHub REST API for information about a public repository. GitHub exposes rich metadata about repositories, commits, issues, and more.

4.1 Goal of the example

> Retrieve basic metadata about the `tidyverse/ggplot2` repository and summarise it in a small tibble that can be used in further analysis.

4.1.1 Step 1 – Inspect the endpoint

The GitHub API documentation tells us that repository metadata is available at:

```
https://api.github.com/repos/{owner}/{repo}
```

For the repository `tidyverse/ggplot2`, the URL becomes:

```
https://api.github.com/repos/tidyverse/ggplot2
```

4.1.2 Step 2 – Request and parse in R

4.1.3 Step 3 – Light transformation

The tibble can be enriched with a derived variable and then simplified:

Even though this is a small example, it demonstrates the complete chain from endpoint to tidy tibble, which is the core skill required for more complex APIs.

5 Challenges and Best Practices

Working with APIs is powerful but also comes with recurring challenges. The most important ones for students in this course are summarised below.

5.1 Authentication and rate limits

Many APIs require **API keys** or OAuth tokens. Even public APIs often impose **rate limits**, which restrict how many requests can be made per minute or hour. Best practice is to:

- read the documentation carefully,
- store keys in environment variables (for example, via `.Renviron`),
- avoid hard-coding secrets in scripts or sharing them on GitHub.

5.2 Nested and evolving JSON structures

Deeply nested JSON can be hard to convert into a tidy format. It is helpful to:

- start by inspecting the raw structure with `str()` or `glimpse()`,

- use `purrr::map_*()` and `tidyr::unnest_*()` when lists of lists appear,
- keep wrangling code modular and well commented.

APIs sometimes change their structure over time, so code should be written in a way that makes such assumptions explicit and easy to update.

5.3 Error handling and reproducibility

Requests can fail due to network problems, invalid parameters, or server-side issues. The `httr2` package provides helpers such as `resp_status()` and `resp_status_desc()` to inspect responses and react accordingly.

For reproducibility, it is good practice to:

- log which endpoints and parameters were used,
- cache important responses (for example, as `.rds` files),
- document the date on which the data were retrieved.

6 Conclusion

Importing data in R directly from the internet using APIs is a key skill for data-driven decision-making. Once the basic ideas of HTTP requests and JSON parsing are understood, the combination of **httr2**, **jsonlite**, and the tidyverse allows analysts to build workflows that are:

- automated rather than manual,
- transparent and reproducible,
- flexible enough to incorporate different APIs and domains.

The small GitHub example in this handout is only a starting point. The same principles extend to more advanced use cases such as authenticated endpoints, scheduled data collection, interactive dashboards, or integration with Shiny applications.

7 Affidavit

I hereby affirm that this submitted paper was authored unaided and solely by me. Additionally, no other sources than those in the reference list were used. Parts of this paper, including tables and figures, that have been taken either verbatim or analogously from other works have in each case been properly cited with regard to their origin and authorship. This paper either in parts or in its entirety, be it in the same or similar form, has not been submitted to any other examination board and has not been published.

I acknowledge that the university may use plagiarism detection software to check my work. I agree to cooperate with any investigation of suspected plagiarism and to provide any additional information or evidence requested by the university.

Name: Mohammed Tariq Azeez A K

Place: Cologne ____________________

Date: 19 November 2025 ____________