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

Tariq Azeez (Matriculation No. 400983409)

2025-11-19

1. What Is an API?

- **API** = Application Programming Interface
- A set of **rules and endpoints** that allow software to communicate over the internet.
- In this presentation, we focus on **HTTP web APIs**, typically:
  - `GET` requests (retrieve data)
  - `POST` requests (send data)
- Most modern APIs return data in **JSON** format (JavaScript Object Notation).
- Key concepts:
  - **Base URL** (e.g., `https://api.github.com/`)
  - **Endpoint** (e.g., `/users/{username}/repos`)
  - **Query parameters** (e.g., `?per_page=100`)
  - **Authentication** (API keys, tokens) – not used in today's simple examples.

---

2. Why This Topic Matters

- Modern business analytics often relies on **real-time or frequently updated data** from web services and platforms.
- APIs allow us to **bypass manual downloads** (CSV/Excel) and connect **directly** to the data source programmatically.
- In the R ecosystem, packages like `httr2`, `jsonlite` and `tidyverse` provide a **coherent workflow** for requesting, parsing, and analyzing API data.
- This means that using R we can automate the data for:

  - More **up-to-date information**
  - Better **reproducibility** of analyses
  - Easier **automation** of recurring reports and dashboards

- Understanding APIs is essential for **business analytics, dashboards, and decision-making**

---

3. JSON as the Standard Data Format

- Most modern APIs return data in **JSON (JavaScript Object Notation)**
- JSON = nested structure of:
  - objects (lists)
  - arrays (vectors/lists)
  - key–value pairs
- R parses JSON with `jsonlite::fromJSON()`
- Typical workflow:
  - Call API → receive JSON → parse to list → convert to tibble/data frame
- JSON preserves metadata such as:
  - timestamps
  - nested attributes (e.g., user → followers, repos, etc.)

---

4. R Workflow for Web APIs

Typical workflow in R:

1. **Construct the request**
   - Define URL, endpoint, and parameters
   - Optionally add headers or authentication
2. **Send the request** using `httr2`
3. **Parse the JSON response** into R objects with `jsonlite`
4. **Transform and analyze** using `tidyverse` (e.g., `dplyr`, `ggplot2`)

We start by loading the required packages.

```r
# Load core packages for API work
library(httr2)    # modern HTTP client for R
library(jsonlite) # JSON parsing
library(dplyr)    # data manipulation (part of tidyverse)
library(ggplot2)  # visualization (tidyverse)
```

---

5. Key R Packages Used Today

- `httr2`
  - Modern HTTP client for R
  - Fluent interface for building and sending requests
  - Built-in helpers for error handling and pagination
- `jsonlite`
  - Robust tools to parse JSON into R data frames and lists
  - `fromJSON()` simplifies nested JSON to tibbles or data frames
- `tidyverse` (e.g., `dplyr`, `ggplot2`)
  - Consistent grammar for data manipulation and plotting
  - Fits naturally with API-driven, pipeline-based workflows
- These tools together form a **pipeline**:
  - request → response → JSON → tidy data → visualization.

---

6. Using httr2: A Modern Workflow

- `httr2` focuses on:
  - clear syntax
  - robust error handling
  - support for authentication (OAuth, tokens, API keys)

Basic pattern:

1. Build a request with `request()`
2. Add parameters / headers if needed
3. Perform the request with `req_perform()`

4. Parse the response body (JSON → R)

```
library(httr2)

req <- request("https://api.github.com")
resp <- req_perform(req)

resp_status(resp)          # HTTP status code
resp_headers(resp)         # response headers
resp_body_json(resp)       # parsed JSON content
```

---

7. GitHub API (Public API)

Goal: Fetch metadata about a GitHub repository (e.g., `tidyverse/ggplot2`).

```
library(httr2)
library(jsonlite)
library(tidyverse)

# 1. Build and perform request
req <- request("https://api.github.com/repos/tidyverse/ggplot2")
resp <- req_perform(req)

# 2. Parse body as JSON
data <- resp_body_json(resp)

# 3. Convert to tibble
repo_info <- tibble(
  repo    = data$name,
  owner   = data$owner$login,
  stars     = data$stargazers_count,
  forks     = data$forks_count,
  watchers = data$watchers_count,
  issues = data$open_issues,
  created = data$created_at
)

repo_info
```

---

8. Automating API Data With the tidyverse

Typical workflow:

1. Call API and parse JSON
2. Convert nested lists to tibbles
3. Use tidyverse to clean and transform

```
clean_repo_info <- repo_info |>
  mutate(
    created = as.Date(created),
    repo_age_years = as.numeric(Sys.Date() - created) / 365
  ) |>
  select(repo, owner, stars, forks, issues, repo_age_years)

clean_repo_info
```

- `mutate()` to create new variables
- `select()` to keep only relevant columns
- Can be extended to join with other data sources (e.g., other repos, issues, etc.)

---

9. Challenges When Working With APIs

- **Rate limits:** Only a certain number of requests allowed per time period
- **Authentication:** API keys, tokens, OAuth flows
- **Unstable endpoints:** URLs may change or be deprecated
- **Response variability:** JSON structure can change between versions
- **Time zones and timestamps:** Data needs consistent time handling for time-series analysis
- **Data quality:** Missing values, inconsistent formats across sources

---

10. Best Practices for API Workflows in R

- Always read and follow the **API documentation**
- Handle errors explicitly: Check status codes (e.g., 200, 404, 429, 500)
- Use secure storage for keys: `.Renviron` or key management services
- Cache results when appropriate: Save as `.rds`, use packages like `pins`
- Log your calls: Keep track of when and how data was retrieved

- Make your scripts: **reproducible**, **parameterized**, and easy to re-run

---

11. Sample R Exercise

Task:
Use any public API → import → convert → summarize → plot.

Skeleton:

```
#| eval: false

library(httr2)
library(jsonlite)
library(dplyr)
library(ggplot2)

url <- "PUT_YOUR_ENDPOINT_HERE"

resp <- request(url) |> req_perform()

raw <- resp |> resp_body_string() |> fromJSON()

glimpse(raw)
```

---

12. Key Takeaways

- APIs allow **direct, programmatic access** to online data
- R offers strong tools:
  - `httr2`, `httr`
  - `jsonlite`
  - `tidyverse`
- Combining APIs + tidyverse enables automated workflows
- Essential for modern data analysis and decision-making

References

- Wickham, H. et al. (2023). *httr2: Perform HTTP requests and process the responses.*
- Wickham, H. (2019). *Welcome to the tidyverse.*
- GitHub API Documentation.
- R Core Team (2025). *R: A language and environment for statistical computing.*
- Course materials for Data Analysis for Decision-Making (WS 2025/26).
- OpenAI - ChatGPT, Claude.ai

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 thesis. 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: 19th Nov 2025_____