P3

Katy Waterman

2025-09-30

Part 1: Choose Your Data Collection Scenario (20 points)

Scenario:

Objective: Collect U.S. education enrollment data to analyze grade-level patterns across multiple states over time. The goal is to compare enrollment trends in different regions and identify shifts in grade distribution between 2017–2020. **Data Sources:** Urban Institute Education Data Portal **Data Types:** Year, Grade Level, State, Enrollment Counts, School Level **Geographic Scope:** Three states for comparison: California (FIPS 6), New York (FIPS 36), and Texas (FIPS 48) **Time Range:** Academic years 2017-2020

```r
knitr::opts_chunk$set(echo = TRUE, warning = FALSE, message = FALSE)
library(httr)
library(jsonlite)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
dir.create("data", showWarnings = FALSE)
dir.create("data/metadata", recursive = TRUE, showWarnings = FALSE)
dir.create("reports", showWarnings = FALSE)
```

Exercise 2.2: Your First API Call

```r
get_cat_fact <- function() {
  url <- "https://catfact.ninja/fact"
  res <- GET(url)
  if (status_code(res) == 200) {
    return(fromJSON(content(res, "text"))$fact)
  } else {
    return(NULL)
  }
}

# Collect 5 facts
cat_facts <- replicate(5, get_cat_fact())
cat_facts
```

```
## [1] "A cat’s brain is more similar to a man’s brain than that of a dog."
## [2] "Cats often overract to unexpected stimuli because of their extremely sensitive nervous system."
## [3] "Miacis, the primitive ancestor of cats, was a small, tree-living creature of the late Eocene pe
## [4] "Many Egyptians worshipped the goddess Bast, who had a woman’s body and a cat’s head."
## [5] "A cat has 230 bones in its body. A human has 206. A cat has no collarbone, so it can fit throug
```

```
# Save to JSON
write_json(list(facts = cat_facts), "demo/cat_facts.json", pretty = TRUE)
```

Exercise 2.3: API with Parameters

```
get_public_holidays <- function(country, year=2024) {
  url <- paste0("https://date.nager.at/api/v3/PublicHolidays/", year, "/", country)
  res <- GET(url)
  if (status_code(res) == 200) {
    holidays <- fromJSON(content(res, "text"))
    return(holidays[, c("date", "localName")])
  } else {
    return(NULL)
  }
}

countries <- c("US", "CA", "GB")
holiday_summary <- lapply(countries, function(c) get_public_holidays(c, 2024))
names(holiday_summary) <- countries
holiday_summary
```

```
## $US
##          date                            localName
## 1  2024-01-01                       New Year’s Day
## 2  2024-01-15          Martin Luther King, Jr. Day
## 3  2024-02-12                   Lincoln’s Birthday
## 4  2024-02-19                Washington’s Birthday
## 5  2024-03-29                          Good Friday
## 6  2024-03-29                          Good Friday
## 7  2024-05-08                           Truman Day
## 8  2024-05-27                         Memorial Day
## 9  2024-06-19 Juneteenth National Independence Day
## 10 2024-07-04                     Independence Day
## 11 2024-09-02                           Labour Day
## 12 2024-10-14                         Columbus Day
## 13 2024-10-14              Indigenous Peoples’ Day
## 14 2024-11-11                         Veterans Day
## 15 2024-11-28                     Thanksgiving Day
## 16 2024-12-25                        Christmas Day
## 
## $CA
##          date                                 localName
## 1  2024-01-01                            New Year’s Day
## 2  2024-02-19                            Louis Riel Day
## 3  2024-02-19                              Islander Day
## 4  2024-02-19                              Heritage Day
## 5  2024-02-19                                Family Day
## 6  2024-03-17                       Saint Patrick’s Day
## 7  2024-03-29                               Good Friday
## 8  2024-04-01                             Easter Monday
## 9  2024-04-23                        Saint George’s Day
## 10 2024-05-20                    National Patriots’ Day
## 11 2024-05-20                              Victoria Day
## 12 2024-06-21                   National Aboriginal Day
## 13 2024-06-24                             Discovery Day
## 14 2024-06-24                  Fête nationale du Québec
## 15 2024-07-01                                Canada Day
## 16 2024-07-12                           Orangemen’s Day
## 17 2024-08-05                             Civic Holiday
## 18 2024-08-05                      British Columbia Day
## 19 2024-08-05                              Heritage Day
## 20 2024-08-05                         New Brunswick Day
## 21 2024-08-05                                 Natal Day
## 22 2024-08-05                          Saskatchewan Day
## 23 2024-08-19                       Gold Cup Parade Day
## 24 2024-08-19                             Discovery Day
## 25 2024-09-02                                Labour Day
## 26 2024-09-30 National Day for Truth and Reconciliation
## 27 2024-10-14                              Thanksgiving
## 28 2024-11-11                             Armistice Day
## 29 2024-11-11                           Remembrance Day
## 30 2024-12-25                             Christmas Day
## 31 2024-12-26                                Boxing Day
## 
## $GB
##          date              localName
## 1  2024-01-01         New Year’s Day
## 2  2024-01-02              2 January
## 3  2024-03-18    Saint Patrick’s Day
## 4  2024-03-29            Good Friday
## 5  2024-04-01          Easter Monday
## 6  2024-05-06 Early May Bank Holiday
## 7  2024-05-27    Spring Bank Holiday
## 8  2024-07-12    Battle of the Boyne
## 9  2024-08-05    Summer Bank Holiday
## 10 2024-08-26    Summer Bank Holiday
## 11 2024-12-02     Saint Andrew’s Day
## 12 2024-12-25          Christmas Day
## 13 2024-12-26             Boxing Day
```

Reflection:

Before this project, I had very limited knowledge about APIs. Working through the exercises really helped me understand how APIs work in practice, for example how you can make a request and get structured data back instantly. Using R, I learned how to send GET requests, deal with JSON data, and write the results into files I could reuse. It felt a bit intimidating at first, but once I got the hang of the code and saw the data coming in, it was really interesting. I also learned how important it is to handle errors and be respectful of API limits so I don't accidentally overload someone's server. Overall, it was a super practical intro to working with live data, and I can definitely see myself using APIs more in this project.

Part 3: API Setup (10 points)

For this project, I used the Urban Institute Education Data API, which is fully open and does not require any authentication or API key. This made setup straightforward and accessible for first-time users like me. However, if I were using an API that did require authentication, such as the OpenWeatherMap API or NewsAPI, I would follow proper security best practices to keep my key safe.

Part 4: AI Data Collection Agent (35 points)

```r
library(httr)
library(jsonlite)
library(dplyr)
library(lubridate)

# Load config
config <- list(
    years = 2017:2020,
    grades = c(3, 8),
    states = c(6, 36, 48),
    base_delay = 1.0
)

# Initialize variables
data_store <- list()
collection_stats <- list(
    total_requests = 0,
    successful_requests = 0,
    failed_requests = 0
)

delay_multiplier <- 1.0

# Helper Functions

make_api_request <- function(year, grade, fips) {
    url <- paste0("https://educationdata.urban.org/api/v1/schools/ccd/enrollment/",
                  year, "/grade-", grade, "/?fips=", fips)
    collection_stats$total_requests <<- collection_stats$total_requests + 1

    tryCatch({
        res <- GET(url)
        if (status_code(res) == 200) {
            collection_stats$successful_requests <<- collection_stats$successful_requests + 1
            return(fromJSON(content(res, "text"))$results)
        } else {
            collection_stats$failed_requests <<- collection_stats$failed_requests + 1
            return(NULL)
        }
    }, error = function(e) {
        collection_stats$failed_requests <<- collection_stats$failed_requests + 1
        return(NULL)
    })
}

respectful_delay <- function() {
    delay <- config$base_delay * delay_multiplier * runif(1, 0.5, 1.5)
    Sys.sleep(delay)
}

assess_data_quality <- function(data) {
    complete_rows <- sum(complete.cases(data))
    round(complete_rows / nrow(data), 2)
}

adjust_strategy <- function(success_rate) {
    if (success_rate < 0.5) {
        delay_multiplier <<- delay_multiplier * 2
    } else if (success_rate > 0.9) {
        delay_multiplier <<- delay_multiplier * 0.8
    }
}

# Main Collection Loop
for (year in config$years) {
    for (grade in config$grades) {
        for (state in config$states) {
            cat("Fetching year:", year, "grade:", grade, "state (FIPS):", state, "\n")
            results <- make_api_request(year, grade, state)
            if (!is.null(results) && length(results) > 0) {
                data_store[[length(data_store) + 1]] <- results
            }
            success_rate <- collection_stats$successful_requests / collection_stats$total_requests
            if (success_rate < 0.8) {
                adjust_strategy(success_rate)
            }
            respectful_delay()
        }
    }
}
```

```
## Fetching year: 2017 grade: 3 state (FIPS): 6

## Fetching year: 2017 grade: 3 state (FIPS): 36

## Fetching year: 2017 grade: 3 state (FIPS): 48
## Fetching year: 2017 grade: 8 state (FIPS): 6

## Fetching year: 2017 grade: 8 state (FIPS): 36

## Fetching year: 2017 grade: 8 state (FIPS): 48

## Fetching year: 2018 grade: 3 state (FIPS): 6

## Fetching year: 2018 grade: 3 state (FIPS): 36

## Fetching year: 2018 grade: 3 state (FIPS): 48

## Fetching year: 2018 grade: 8 state (FIPS): 6

## Fetching year: 2018 grade: 8 state (FIPS): 36

## Fetching year: 2018 grade: 8 state (FIPS): 48

## Fetching year: 2019 grade: 3 state (FIPS): 6

## Fetching year: 2019 grade: 3 state (FIPS): 36

## Fetching year: 2019 grade: 3 state (FIPS): 48

## Fetching year: 2019 grade: 8 state (FIPS): 6

## Fetching year: 2019 grade: 8 state (FIPS): 36

## Fetching year: 2019 grade: 8 state (FIPS): 48

## Fetching year: 2020 grade: 3 state (FIPS): 6

## Fetching year: 2020 grade: 3 state (FIPS): 36

## Fetching year: 2020 grade: 3 state (FIPS): 48

## Fetching year: 2020 grade: 8 state (FIPS): 6

## Fetching year: 2020 grade: 8 state (FIPS): 36

## Fetching year: 2020 grade: 8 state (FIPS): 48
```

```
# Combine data
edu_data <- bind_rows(data_store)
dir.create("data/raw", recursive = TRUE, showWarnings = FALSE)
write_json(edu_data, "data/raw/education_data.json", pretty = TRUE)

# Metadata
generate_metadata <- function(data) {
    meta <- list(
        collection_date = as.character(Sys.time()),
        agent_version = "R-1.0",
        collector = "Katy Waterman",
        total_records = nrow(data),
        variables = names(data)
    )
    dir.create("data/metadata", recursive = TRUE, showWarnings = FALSE)
    write_json(meta, "data/metadata/metadata.json", pretty = TRUE)
    return(meta)
}

metadata <- generate_metadata(edu_data)

# Quality Report
generate_quality_report <- function(data) {
    completeness <- colMeans(!is.na(data))
    report <- list(
        summary = list(
            total_records = nrow(data),
            collection_success_rate = collection_stats$successful_requests / collection_stats$total_requ
            overall_quality_score = assess_data_quality(data)
        ),
        completeness = completeness
    )
    dir.create("reports", showWarnings = FALSE)
    write_json(report, "reports/quality_report.json", pretty = TRUE)
    return(report)
}

quality_report <- generate_quality_report(edu_data)

# Final Summary
cat("\nCollection Summary:\n")
```

```
##
## Collection Summary:
```

```
cat("Total records:", nrow(edu_data), "\n")
```

```
## Total records: 101433
```

```
cat("Success rate:", round(collection_stats$successful_requests / collection_stats$total_requests, 2),
```

```
## Success rate: 1
```

```r
cat("Quality score:", quality_report$summary$overall_quality_score, "\n")
```

```
## Quality score: 1
```

Collection Summary

- **Total Records Collected:** 101433
- **Years Covered:** 2017–2020
- **States Covered:** California, New York, Texas
- **Grades Covered:** 3 and 8
- **Success Rate:** All API calls returned valid data

Data Quality:

- Completeness across variables: high (>95%)
- No negative or missing enrollment values detected

Challenges:

- API queries for large datasets can be slow
- Requires delays to avoid overloading the server

Recommendations:

- Expand to additional states and grades
- Combine with IPEDS datasets for higher education insights