APEC 8221 - Assignment 2: Automation & API Data Acquisition

Due Saturday, September 27, 2025 at 8:00 AM

Table of contents

1 Assignment Overview

1.1 Technical Requirements

- Use the provided `assignment2-template.qmd` Quarto template
- Save your file as `assignment2-[your-x500-id].qmd`
- All file paths must be relative (R Project structure)
- Code must be well-commented with clear section headers
- Document your function design choices and debugging process
- Quarto document must knit to PDF without errors
- Include your GitHub repository URL in your submission
- Git history should show logical progression with meaningful commit messages

1.2 Required Packages

```r
# Load packages
library(tidyverse)
library(httr2)
library(jsonlite)
```

1.3 Grading Rubric

Criteria	Needs Improvement	Satisfactory	Excellent	Points
Correctness & Accuracy	Code fails to run or produces incorrect results	Code runs with largely correct results, minor errors possible	Code is correct, efficient, and performs all tasks accurately	30 pts
Code Clarity & Style	Poor formatting, lacks comments, unclear logic	Readable code with adequate comments	Clean, well-commented code following style guidelines	4 pts
Reproducibility & Quarto	Document won't knit or uses absolute paths	Document knits successfully with relative paths	Professional document with clear methodology	3 pts

Criteria	Needs Improvement	Satisfactory	Excellent	Points
Version Control	Single commit or no meaningful history	Several commits with generic messages	Clear development story with descriptive commits	3 pts

2 Tasks

2.1 Build Your API Function

The Federal Reserve Economic Data (FRED) API provides access to thousands of economic time series. For this assignment, you'll focus on state unemployment rates.

Key FRED Unemployment Series Pattern:

- National: `UNRATE`
- State unemployment rates: `{STATE_CODE}UR`
- Examples: `CAUR` (California), `TXUR` (Texas), `NYUR` (New York)

2.1.1 API Setup and Testing

2.1.1.1 Obtain and Store Your FRED API Key

1. Go to https://fred.stlouisfed.org/docs/api/api_key.html
2. Request a free API key
3. Copy the API key number and save it in your R environment for future reference with `usethis::edit_r_environ()`. Name the key `FRED_API_KEY = "your api key"`.

```r
# Secure API key storage - load from .Renviron file
fred_api_key <- Sys.getenv("FRED_API_KEY")
```

2.1.1.2 Make a Simple Request

```r
# Test with a simple API call to get national unemployment rate
base_url <- "https://api.stlouisfed.org/fred/series/observations"
test_url <- paste0(base_url, "?", "series_id=UNRATE",
                   "&api_key=", fred_api_key, "&file_type=json")

# Make the request
response <- request(test_url) %>% req_perform()

# Check if it worked
resp_status(response)  # Should be 200

# Look at the JSON structure
raw_data <- response %>% resp_body_json()
str(raw_data, max.level = 2)
```

2.1.2 Write Your Data Acquisition Function

Task:

Write a function called `get_fred_unemployment()` that meets the following requirements:

1. **Input Validation:** Check that `state_code` is a valid 2-letter abbreviation
2. **API Request:** Use `httr2` to make the API call
3. **Error Handling:** Check for HTTP status codes and handle API failures gracefully
4. **JSON Parsing:** Extract the observations from the nested JSON structure
5. **Data Cleaning:** Return a clean data frame with:
    - `date`: Properly formatted date column
    - `unemployment_rate`: Numeric unemployment rate
    - `state`: The state code for identification

```r
get_fred_unemployment <- function(state_code, api_key) {
  # Your code here
  # Should return a clean data frame with columns: date, unemployment_rate, state
}
```

2.2 Automate Data Collection

2.2.1 Define Your State List

Create a vector of 8-10 state codes representing different regions of the US:

```r
# Select states from different regions for interesting comparisons
# For example:
target_states <- c("CA", "TX", "NY", "FL", "OH", "WA", "CO", "GA")
```

2.2.2 Implement Rate Limiting and Include Messaging

1. The FRED API has rate limits. Modify your function to include a `Sys.sleep()` between calls.
2. It's helpful to know the state of your API calls when implementing multiple API calls. Include informative messages within your function.

2.2.3 Compare Four Iteration Methods

Demonstrate four different approaches to collect data for all your target states:

1. **Method 1: `for` Loop**

   ```r
   # Method 1: for loop approach
   unemployment_data_loop <- data.frame()

   for (state in target_states) {
     # Your code here
     cat("Downloading data for", state, "...\n")
     # Add to unemployment_data_loop
   }
   ```

2. **Method 2: `lapply`**

   ```r
   # Method 2: lapply approach
   unemployment_list <- lapply(target_states, function(state) {
     # Your code here
   })

   # Combine into single data frame
   unemployment_data_apply <- # Your code to combine results
   ```

3. **Method 3: `purrr::map`**

   ```r
   # Method 3: Modern tidyverse approach with purrr
   unemployment_data_purrr <- target_states %>%
     # Your code here
     map()
     # Your code to combine into a single data frame
   ```

4. **Method 4: Manual calls (for comparison)**

```
# Method 4: What it would look like without automation
ca_data <- get_fred_unemployment_safe("CA", fred_api_key)
tx_data <- get_fred_unemployment_safe("TX", fred_api_key)
# ... (show how tedious this would be)
```

5. **Analysis:** Write 2-3 sentences comparing these approaches. Which is most readable? Most efficient? Why is automation valuable here?

6. **Scaling Analysis:** How would you modify your approach for analyzing unemployment data for all 50 states instead of just 8? What challenges would arise and how would each iteration method handle increased scale?

2.3 Data Analysis and Visualization

2.3.1 Regional Unemployment Patterns

Using your collected data, analyze unemployment patterns:

Basic Summary Statistics:

1. Calculate average unemployment rate by state (2019-2024)
2. Identify which state had the highest peak unemployment
3. Find which state shows the most volatility (highest standard deviation)

Requirements:

- Use `dplyr` operations we've learned (`group_by`, `summarise`, `arrange`)
- Create a well-formatted summary table
- Include brief interpretation of patterns

2.3.2 Time Series Visualization

Create a professional line plot showing unemployment trends:

Specifications:

- X-axis: Date (months/years)
- Y-axis: Unemployment rate (%)
- One line per state, colored by state
- Focus on 2019-2024 period to show COVID impact and recovery
- Add vertical line at March 2020 (COVID start): `geom_vline(xintercept = as.Date("2020-03-01"))`
- Professional labels, title, and theme

2.3.3 State Comparison Analysis

Create a bar chart comparing **average unemployment rate by state** for two periods:

1. **Pre-COVID**: January 2019 - February 2020
2. **Post-COVID**: January 2023 - December 2024

Requirements:

- Use `filter()` to create the two time periods
- Calculate average unemployment for each state in each period
- Create side-by-side bars using `position = "dodge"`
- Color bars by time period
- Order states by one of the periods (your choice)
- Professional formatting

2.4 Extension and Critical Thinking

2.4.1 API Pagination Challenge

Real-world APIs often return large datasets that are split across multiple "pages" to avoid overwhelming servers and clients. The FRED API supports pagination using `limit` and `offset` parameters.

Your Challenge: Extend your unemployment function to handle pagination and retrieve complete historical data.

1. **Explore API Metadata**

```
# Make a request with a small limit to see pagination in action
response <- request("https://api.stlouisfed.org/fred/series/observations") %>%
  req_url_query(
    series_id = "CAUR",  # California unemployment
    api_key = fred_api_key,
    file_type = "json",
    limit = 100  # Only get 100 observations
  ) %>%
  req_perform()

full_data <- response %>% resp_body_json()

# Explore the metadata
str(full_data, max.level = 1)
```

What pagination info is available?

A. How many total observations are available?
B. How many observations were retrieved in the first call?

2. **Build a Complete Data Function:** Write a new function `get_fred_unemployment_complete()` that:

   A. **Gets metadata first** to determine total number of observations
   B. **Calculates required pages** based on a reasonable page size (e.g., 1000 observations)
   C. **Loops through pages** using `offset` parameter to get all data
   D. **Combines results** into a single, complete dataset
   E. **Includes progress tracking** so users know what's happening

```r
get_fred_unemployment_complete <- function(state_code, api_key, page_size = 1000) {

  # Step 1: Get metadata to find total observations
  # Your code here

  # Step 2: Calculate number of pages needed
  # Your code here

  # Step 3: Loop through pages and collect data
  # Your code here - use offset parameter

  # Step 4: Combine and return complete dataset
  # Your code here
}
```

Requirements:

- Handle cases where total observations < page_size (no pagination needed)
- Include informative progress messages: "Getting page 1 of 3…"
- Test with both small and large datasets
- Compare results with your original function on recent data

Analysis Questions:

1. For California unemployment data, how many total observations are available?
2. If you used a page size of 500, how many API calls would be required?
3. What are the trade-offs between larger and smaller page sizes?
4. When would you use complete historical data vs. recent data only?

2.4.2 Choose Your API Adventure

Choose **one** of the following APIs to practice your skills with a different data provider. Each has different documentation styles and response formats—this will test your ability to read API documentation and adapt your functions.

Option A: World Bank API – International Development Data

- **Base URL:** `https://api.worldbank.org/v2/country/{country}/indicator/{indicator}`
- **Example:** GDP per capita for Brazil: `https://api.worldbank.org/v2/country/BR/indicator/NY.GDP`
- **Task:** Write a function to get an economic/social indicator for 5-6 different countries
- **Analysis:** Compare development patterns across countries or regions

Option B: Bureau of Labor Statistics API – Employment & Wage Data

- **Free Registration:** bls.gov/developers/
- **Base URL:** `https://api.bls.gov/publicAPI/v2/timeseries/data/`
- **Example Series:** Average hourly earnings (`CES0500000003`), Employment level (`CES0000000001`)
- **Task:** Write a function to get employment/wage data for different industries or regions
- **Analysis:** Examine wage growth or employment trends across sectors

Option C: Alpha Vantage API – Financial Markets Data

- **Free API Key:** alphavantage.co/support/#api-key
- **Base URL:** `https://www.alphavantage.co/query`
- **Example:** Daily stock prices: `function=TIME_SERIES_DAILY&symbol=MSFT&apikey=demo`
- **Task:** Write a function to get stock prices for 4-5 major companies
- **Analysis:** Compare stock volatility, returns, or sector performance

Option D: US Census American Community Survey API – Demographics & Social Data

- **Base URL:** `https://api.census.gov/data/{year}/acs/acs1`
- **Example:** Median income by state: `https://api.census.gov/data/2021/acs/acs1?get=NAME,B19013`
- **Task:** Write a function to get demographic data (income, education, health insurance) for multiple states
- **Analysis:** Examine geographic patterns in social outcomes

Option E: OpenWeatherMap API – Climate & Environmental Data

- **Free API Key:** openweathermap.org/api
- **Base URL:** `https://api.openweathermap.org/data/2.5/weather`
- **Example:** Current weather: `https://api.openweathermap.org/data/2.5/weather?q=Minneapolis&a`
- **Task:** Write a function to get weather data (temperature, humidity, air quality) for multiple cities

- **Analysis:** Compare climate patterns across cities or examine seasonal variations

Requirements for Any Option:

1. **Read the Documentation:** Spend time understanding the API structure and parameters
2. **Write a Function:** Create a function to query your chosen API
3. **Collect Multiple Data Points:** Use iteration to get data for multiple entities (countries, states, companies, etc.)
4. **Create a Visualization:** Make one professional plot showing patterns in your data
5. **Document Challenges:** Write 2-3 sentences about what was different/challenging compared to the FRED API

2.4.3 API Landscape Survey

To build confidence working with diverse APIs, make **simple exploratory calls** to these three different APIs. You don't need to write full functions—just single requests to understand how API structures can vary.

```
# API 1: NASA Astronomy Picture of the Day (nested JSON, government API)
nasa_response <- request("https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY") %>%
  req_perform()
nasa_data <- nasa_response %>% resp_body_json()
str(nasa_data)

# API 2: JSONPlaceholder (flat JSON, RESTful design)
posts_response <- request("https://jsonplaceholder.typicode.com/posts/1") %>%
  req_perform()
posts_data <- posts_response %>% resp_body_json()
str(posts_data)

# API 3: Cat Facts (simple response, no authentication)
facts_response <- request("https://catfact.ninja/fact") %>%
  req_perform()
facts_data <- facts_response %>% resp_body_json()
str(facts_data)
```

Analysis Questions:

1. How do these response structures differ from FRED?
2. Which APIs require authentication and which don't?
3. What do you notice about government vs. commercial vs. fun APIs?

4. After seeing 5+ different APIs, what patterns emerge in API design?

❗ Important

Academic Integrity Reminder
This assignment requires you to write custom functions and make API calls. While you may collaborate on understanding the concepts, your function implementations and analysis should be your own work. Properly cite any online resources, AI assistance, or peer collaboration.