Building a U.S. Census Data Explorer

A Shiny Dashboard for ACS, Decennial Census, and LEHD Data

Pukar Bhandari

2024-05-11

Table of contents

## 1 Introduction

As a transportation planner, accessing and visualizing demographic and employment data is crucial for informed decision-making. The U.S. Census Bureau provides a wealth of information through various datasets, but navigating multiple APIs and data formats can be challenging. To streamline this process, I developed an interactive R Shiny dashboard that provides unified access to three major Census data sources:

* **American Community Survey (ACS)** - Detailed demographic and socioeconomic estimates
* **Decennial Census** - Complete population counts every 10 years
* **Longitudinal Employer-Household Dynamics (LEHD)** - Employment and commuting patterns

This post walks through the development process, key technical decisions, and lessons learned while building this data exploration tool.

## 2 Project Overview

The dashboard serves as a one-stop interface for Census data retrieval, featuring:

* Interactive geographic selection (state/county level)
* Variable selection through searchable tables
* Real-time data visualization on interactive maps
* Export capabilities for both tabular and geospatial data
* Responsive design optimized for different screen sizes

**🔗 Repository**: You can find the complete source code and documentation at: <https://github.com/ar-puuk/uscensus-dashboard/>

## 3 Technical Architecture

### Core Dependencies

The application leverages several specialized R packages for Census data access and geospatial processing:

# Census data access
library(tidycensus) # ACS and Decennial Census API
library(lehdr) # LEHD Origin-Destination data
library(tigris) # Geographic boundaries

# Geospatial processing
library(sf) # Simple features for spatial data
library(leaflet) # Interactive mapping

# Shiny ecosystem
library(shiny)
library(shinydashboard)
library(shinyWidgets)
library(DT) # Interactive data tables

### Application Structure

The app follows a modular design pattern with separate UI and server components for each data source. This approach enhances maintainability and allows for independent feature development.

# Modular UI components
acsUI <- fluidPage(...) # American Community Survey interface
censusUI <- fluidPage(...) # Decennial Census interface
lehdUI <- fluidPage(...) # LEHD interface

# Unified navigation
ui <- navbarPage(...)

## 4 Data Source Integration

### American Community Survey (ACS)

The ACS module provides access to detailed demographic estimates from 2009-2022. Key implementation features:

**Dynamic Variable Loading**: Variables are loaded based on the selected year and geographic level, ensuring users only see relevant options.

variables\_acs <- reactive({
 if (!is.null(input$year\_acs)) {
 variables <- load\_variables(year = as.numeric(input$year\_acs),
 dataset = "acs5", cache = TRUE)
 # Filter variables based on geographic level
 if (input$level\_acs == "tract") {
 variables <- variables |>
 filter(geography %in% c("tract", "block group"))
 }
 return(variables)
 }
})

**Geographic Hierarchy**: The interface maintains proper geographic relationships, with county options updating based on state selection.

### Decennial Census

The Decennial Census module focuses on the 2000, 2010, and 2020 complete counts, using the PL 94-171 dataset optimized for redistricting data.

**Streamlined Variable Selection**: Since Decennial Census has fewer variables than ACS, the interface emphasizes ease of use while maintaining the same interaction patterns.

### Longitudinal Employer-Household Dynamics (LEHD)

The LEHD module provides employment statistics and origin-destination flows, crucial for transportation planning applications.

**Version Management**: Different LODES versions cover different time periods, requiring dynamic year filtering:

observe({
 version <- if (!is.null(input$version\_lehd)) input$version\_lehd else "default"

 options <- if (version == "LODES5") {
 2002:2009
 } else if (version == "LODES7") {
 2002:2019
 } else {
 2002:2021
 }

 updateSelectInput(session, "year\_lehd", choices = options, selected = max(options))
})

## 5 Interactive Mapping

### Base Map Configuration

The application uses Leaflet for interactive mapping, with a carefully chosen base layer that balances aesthetics and functionality:

default\_map <- leaflet(options = leafletOptions(crs = leafletCRS())) |>
 addProviderTiles("CartoDB.Voyager") |>
 addPolygons(data = sf\_states, color = "#222222", weight = 1, fillOpacity = 0.15)

**CartoDB.Voyager** was selected for its clean design and good contrast with overlay data, though the code structure allows for easy switching between provider tiles.

### Dynamic Geographic Focus

Maps automatically adjust to show selected geographic areas, providing contextual awareness:

output$map\_acs <- renderLeaflet({
 if (!is.null(input$state\_acs) && input$state\_acs != "") {
 selected\_state\_acs <- sf\_states[sf\_states$NAME == input$state\_acs, ]
 selected\_bbox\_acs <- sf::st\_bbox(selected\_state\_acs)

 default\_map |>
 addPolygons(data = selected\_state\_acs, color = "#222222", weight = 4) |>
 fitBounds(selected\_bbox\_acs[[1]], selected\_bbox\_acs[[2]],
 selected\_bbox\_acs[[3]], selected\_bbox\_acs[[4]])
 }
})

## 6 Data Export Functionality

### Flexible Format Support

Users can export data in multiple formats depending on their needs:

* **CSV files** for tabular analysis
* **Shapefiles (zipped)** for GIS applications

output$download\_acs <- downloadHandler(
 filename = function() {
 if (input$geometry\_acs) {
 paste0("acs\_data\_", input$state\_acs, "\_", input$year\_acs, ".zip")
 } else {
 paste0("acs\_data\_", input$state\_acs, "\_", input$year\_acs, ".csv")
 }
 },
 content = function(file) {
 if (input$geometry\_acs) {
 write\_sf\_zip(data\_acs(), file, overwrite = TRUE)
 } else {
 readr::write\_csv(data\_acs() |> st\_drop\_geometry(), file)
 }
 }
)

### Custom Shapefile Export

Since R’s sf package doesn’t directly export zipped shapefiles, I implemented a custom function to handle the complete shapefile format:

write\_sf\_zip <- function(obj, zipfile, overwrite = FALSE) {
 # Create temporary directory for shapefile components
 tmp <- tempfile()
 dir.create(tmp)
 on.exit(unlink(tmp, recursive = TRUE, force = TRUE))

 # Write shapefile and zip all components
 sf::write\_sf(obj, file.path(tmp, shp\_name), delete\_layer = TRUE)
 withr::with\_dir(tmp, zip(tmp\_zip, list.files()))

 file.copy(file.path(tmp, tmp\_zip), zipfile, overwrite = overwrite)
}

## 7 User Experience Considerations

### Variable Selection Interface

One of the biggest UX challenges was making Census variable selection intuitive. The solution uses modal dialogs with searchable data tables:

var\_modal\_acs <- modalDialog(
 title = h4("Select Variable(s) from the List"),
 DTOutput("var\_table\_acs"),
 size = "l",
 easyClose = TRUE,
 footer = actionButton("selectVarButton\_acs", "Select Variable(s)")
)

This approach allows users to browse thousands of variables efficiently while maintaining a clean main interface.

### Responsive Geographic Selection

The cascading geographic selection (State → County → Geographic Level) follows familiar patterns while enforcing data availability constraints.

### Progress Feedback

API calls can take several seconds, so the interface provides clear feedback through action buttons and conditional panels that appear after data is loaded.

## 8 Performance Optimizations

### Caching Strategy

options(tigris\_use\_cache = TRUE) # Geographic boundaries
load\_variables(..., cache = TRUE) # Variable metadata

Caching is enabled for static data like geographic boundaries and variable definitions, significantly reducing load times for repeat users.

### Data Processing Efficiency

For ACS data, margin of error columns are automatically removed to focus on estimates:

result\_acs <- result\_acs |>
 select(-matches("M$")) |> # Remove margin columns
 rename\_all(~ sub("E$", "", .)) # Clean estimate column names

## 9 Deployment Considerations

### API Key Management

The application requires Census API keys but handles them securely through user input rather than hardcoding. This approach ensures:

* No sensitive credentials in source code
* Users maintain control over their API usage
* Easy deployment across different environments

### Error Handling

Robust error handling prevents crashes when API calls fail or users make invalid selections:

req(input$api\_key\_acs, input$year\_acs, input$state\_acs, input$county\_acs, input$level\_acs)

The req() function ensures all required inputs are available before processing.

## 10 Future Enhancements

Several features are planned for future releases:

* **Data Visualization**: Built-in charts and maps with Census data overlays
* **Comparison Tools**: Side-by-side analysis across years or geographies
* **Custom Geography**: Support for user-uploaded boundary files
* **Batch Processing**: Multiple state/year combinations in single requests
* **API Integration**: Direct connection to external GIS platforms

## 11 Conclusion

This Census data explorer demonstrates the power of R Shiny for creating specialized data access tools. By combining multiple Census APIs into a single interface, it significantly reduces the technical barrier for accessing demographic and employment data.

The modular architecture and attention to user experience make it a valuable tool for researchers, planners, and analysts who regularly work with Census data. The open-source approach ensures continued development and community contributions.

For transportation planners specifically, having easy access to demographic characteristics, employment patterns, and commuting flows in a single application streamlines the data gathering phase of project development, allowing more time for analysis and decision-making.

*Want to contribute or suggest improvements? Visit the project repository at:* <https://github.com/ar-puuk/uscensus-dashboard>