



SUL ROSS
The **FRONTIER** University of Texas

Rshiny Apps and Executables

- Presenter: Aaron Majek, MS in Sociology
(aaron.Majek@sulross.edu)
- TAIR Conference 2024

Presentation Objectives

- Understand basics of Reactive Programming
- Review process of creating an app in Rshiny
- Go over several methods of launching an Rshiny app
- Detail common pitfalls faced when making an Rshiny app



Presentation Outlines

- **Part 1:** Rstudio and Rshiny
- **Part 2:** Creating a Shiny app
- **Part 3:** Implementing a Shiny app
- **Part 4:** Final thoughts and Considerations

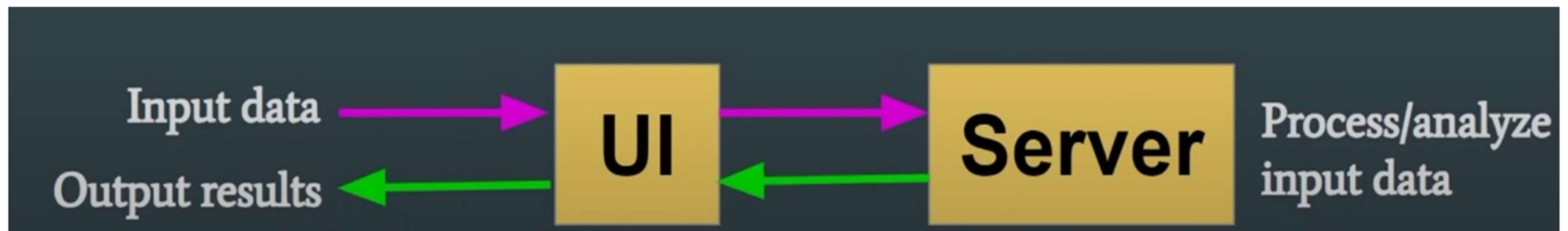
Part 1: What is Rstudio?

- Rstudio is an Integrated development environment (IDE) providing users with an environment for writing and executing code.
- It is available with a GNU General public license meaning it is open source and available to users at no cost.
- Rstudio contains a library of ‘packages’ – sets of code and documentation which may be accessed from a centralized depository.
- As of Feb. 2023, there are at least 19,254 of such packages available for public use.



What is Rshiny

- There are two key components in every shiny app:
 - 1) the user interface (UI) which defines how the application looks
 - 2) the server functions which tell the app how to work.
 - 3) shinyApp function (which fuses the UI with the server components)
- The UI is the frontend (similar to the HTML on a website) that accepts user inputs
- The server is the backend where inputs values are processed and may be used to execute data manipulations for display on the UI.



- <https://www.youtube.com/watch?v=9uFQEck30kA> – img source

Reactive Programming

- Reactive programming is a type of programming whereby updates to inputs result in an update to a given output. This means when an input changes, Shiny will automatically update an output based on a selection from an input.
- Reactivity allows for automatic updates in outputs to correspond with changes in inputs. (IE: an update to one variables results in an update to another).
- `observe()` is a function in Rshiny that facilitates reactivity. It allows the chaining of

- Rshiny has a gallery of publicly accessible Rshiny apps:
- <https://shiny.posit.co/r/gallery/>
- **Live Examples:**
- Simple - <https://shiny.posit.co/r/gallery/start-simple/faithful/>
- Complex - <https://shiny.posit.co/r/gallery/interactive-visualizations/bus-dashboard/>



- **Part 2: Creating a Shiny App**

Building The shiny app

- Who?
 - Who does the application serve? Who are the users?
- What?
 - What does the application do? What are its benefits?
- When?
 - Is there a deadline involved with launching the app?
- Where?
 - Where are you getting the data used in creating your application.
- Why?
 - Why does this process need to be done as a Shiny app? Why not an internal tool?



SUL ROSS
The **FRONTIER** University of Texas

Building The shiny app

- **Who:** The audience for the app is internal stakeholders. app was created as a conversation starter within our university. The aim is to make an app that is simultaneously useful and which may serve as an example of what can be made with the assistance of AI (EG: Chatgpt).
- **What:** The app our IR Office created is a geocoding tool that performs two functions 1) it calculates distance from the university and 2) determines if student address is rural or urban.
- **When:** There are no deadlines involved in launching the app; however, we had to ensure that its creation did not interfere with standard reporting (IE: IPEDS, LBB, etc).
- **Where:** Data on Rural / Urban codes are sourced from US census SHP Files. Additional info on university addresses are sourced from IPEDS datasets. API calls are executed using Census Bulk Geocoding. Distance from the university is calculated using OSRM API calls as well.
- **Why:** The app is multifunctional. Distance from the university is a key predictor of retention. Bringing that data in our student files may assist in retention modeling and predictive analytics. Also, our office frequently calculates out



Building The shiny app – Packages used

- TidyVerse: Contains sets of R packages that are standard to most Rstudio functions.
- Tigris: Load Census TIGER/Line Shapefiles. Connects with US census API facilitating upload of shp files to your Rstudio IDE.
- Tidycensus: Connects with US census API to allow upload of census data files to your Rstudio IDE (IE: American Community Survey).
- Sf: supports a standardized way of encoding spatial vector data. Allows for conversion and manipulation of shp files.
- Tidygeocoder: allows for a the stepwise execution of queries made to numerous geocoding API services at once.
- Osmr: a package the that interfaces between R and the OSRM API hosted by OpenStreetMap. These api calls are used for calculating distance by roads between different points.
- Shiny: Assists in building web applications in Rstudio programming code.

TidyGeocode API

- The TidyGeocode Package is a tool used in Rstudio to geocode student addresses and extract lat and long coordinates.
- To the right are a set of geocoding APIs that can be queried using the Tidygeocode package.
- Tidygeocode has a 'cascade' feature that allows users to assign prioritization to certain API calls over others in an iterative sequence
- For this analysis, we are only using the one free API calls: the census batch query api.

Details

The API documentation for each service is linked to below:

- [Nominatim](#)
- [US Census](#)
- [ArcGIS](#)
- [Geocodio](#)
- [Location IQ](#)
- [Google](#)
- [OpenCage](#)
- [Mapbox](#)
- [HERE](#)
- [TomTom](#)
- [MapQuest](#)
- [Bing](#)



SF Package

- The SF package supports the standardization of spatial vector data by representing geographic information as a dataframe.
- It interfaces with the GEOS C/C++ library used in Geographic information systems (GIS) software allowing transformations on projected geographic points.
- Additionally, it interfaces with PROJ, a coordinate transformation library that allows for performing conversions between cartographic projections.
- The combination of these two interfacing capacities (with GEOS and PROJ) facilitates the transformation of otherwise uni-dimensional data frames into multi-dimensional datafiles across spatial interfaces.
- Together with Tidygeocode and other packages – you can merge census data into your student records.

The Functions

- **Function 1: geocode_chunkify()**
 - This is the function that geocodes uploaded addresses. Because the US Census Bulk Geocoder has a limit to how many rows of data can be processed at once, it is necessary to use a 'chunkify' sequence that breaks the data frame into segments. The geocoding API calls are then iteratively run on each segment before the results are returned back as a whole complete data frame.

```
#FUNCTION - 1 - ADDRESS GEOCODER*.
geocode_chunkify <- function(df, chunk_size) {

  #breaking the geocoder into chunks allows for larger file sizes while the census api
  chunk_count <- ceiling(nrow(df) / chunk_size)
  results_list <- list()

  for (i in 1:chunk_count) {
    start_index <- (i - 1) * chunk_size + 1
    end_index <- min(i * chunk_size, nrow(df))
    df_chunk <- df[start_index:end_index, ]

    # Geocode using the census method
    df_chunk <- df_chunk %>%
      tidygeocoder::geocode(
        method = 'osm',
        street = ADDRESS,
        city = CITY,
        state = STATE,
        postalcode = ZIP,
        lat = "latitude",
        lon = "longitude"
      )

    # Filter out rows with missing coordinates
    df_chunk <- df_chunk[complete.cases(df_chunk[, c("latitude", "longitude")]), ]

    # Convert the dataframe to sf object
    df_chunk <- st_as_sf(df_chunk, coords = c("longitude", "latitude"), crs = 4326)

    results_list[[i]] <- df_chunk
  }
}
```

The Functions

- **Function 2: add_new_geometry()**
 - This function is used for calculating the geographic (AKA Crows) distance from a point given another point. The first step in the process is to ensure that the comparison geometry and the sf element in the uploaded dataframe share the same coordinate system (CRS). It then compares the two points against each other to calculate the distance between them. The outputs are in meters by default so we adjusted the results by dividing by 1609.344 to convert it to miles.
 - **NOTE: Because OSRM has limits on API usage, we added a chunkify sequence to break it up into sections, similar to census API limits.**

```
add_new_geometry_osrm <- function(df, selected_geometry, chunk_size) {  
  
  # Determine the number of chunks  
  chunk_count <- ceiling(nrow(df) / chunk_size)  
  results_list <- list()  
  
  for (i in 1:chunk_count) {  
    start_index <- (i - 1) * chunk_size + 1  
    end_index <- min(i * chunk_size, nrow(df))  
    df_chunk <- df[start_index:end_index, ]  
  
    selected_geometry_sf <- st_as_sf(data.frame(geometry = selected_geometry()), crs = st_crs(df_chunk))  
  
    # calculate distances between each row's geometry and the new geometry using osrm  
    distances <- osrmTable(  
      df_chunk$geometry, |  
      selected_geometry_sf$geometry,  
      measure = 'duration'  
    )$durations  
  
    # Add the calculated distances as a new column "Driving_Distance"  
    df_chunk <- df_chunk %>%  
      mutate(Driving_Distance = distances)  
  
    results_list[[i]] <- df_chunk  
  }  
}
```

The Functions

- **Function 3: add_new_geometry_osrm()**
 - Similar to prior add_new_geometry function, except this one uses an api call to OSRM. OSRM provides a quick method of getting distance by roads. Similar to the prior method, the results are returned in the original dataframe as a new column titled "Driving_Distance".

```
#FUNCTION - 3 - DRIVING DISTANCE CALCULATOR
#This uses OSRM api to calculate distance in driving from a location*.
add_new_geometry_osrm <- function(df, selected_geometry) {

  selected_geometry_sf <- st_as_sf(data.frame(geometry = selected_geometry()), crs = st_crs(df))

  # calculate distances between each row's geometry and the new geometry using osrm
  distances <- osrmTable(
    df$geometry,
    selected_geometry_sf$geometry,
    measure = 'duration'
  )$durations

  # Add the calculated distances as a new column "Bird_distance_osrm"
  df <- df %>%
    mutate(Driving_Distance = distances)

  return(df)
}
```


The Functions

- **Function 4: validate_file()**
 - This is added for security. By placing this function at the top of the geocoding sequence, we can block all api calls from being run if the validation check is not satisfied. Here, the code is setting a criteria such that all values in the ID column must meet the criteria of being 3 alphabetical values followed by 7 numeric values. The intention here is to prevent the upload the sensitive information to the cloud or any server out of network.

```
#FUNCTION 4| - FILE VALIDATOR
# Function to validate the uploaded file and prevent execution if upload doesn't meet criteria.
validate_file <- function(df) {
  # Check if the dataframe contains the column "ID_COLUMN"
  if (!"ID_COLUMN" %in% colnames(df)) {
    return(FALSE)
  }

  # Check if all values in the "ID_COLUMN" meet the criteria
  if (!all(nchar(trimws(as.character(df$ID_COLUMN))) == 10) ||
    any(!grep1("^[:alpha:]{3}[:digit:]{7}$", gsub("\\s", "", as.character(df$ID_COLUMN)))) {
    return(FALSE)
  }

  return(TRUE)
}
```



Data Cleaning and Data Validation Checks

- **Cleaning IPEDS files**
 - Hd2021 files were used. The dataset contains information on institutional characteristics. Data was first filtered on the variable HDEGOFR1, removing all institutions that do not grant a degree.
 - “LONGITUD” and “LATITUDE” variables were converted into geometry sf coordinate points. (the hd2021 files already had the institutional coordinates in the dataframe).
- **Cleaning TIGRIS files**
 - US Census files were used. SHP files were broken out then recompiled by FIP codes and then merged into their regional equivalents using the IPED Standards (IE: “Southwest” region contains data on FIP codes related to AZ, NM, OK, and TX).
 - This was done as Rshiny struggles to handle large batches of information. Breaking the files into sections allows for easier uploads and faster manipulations.
- **Validation Checks**
 - Validation checks detected possible issues when using multi-year comparisons between 2023 and 2013 Rural-Urban codes in the state of CT.

App Design

- Before making a multi-purpose shiny app, test the apps in a dev environment.
- Recommended best practice is as follows:
 - Make separate sets of raw R code
 - After verifying that the code works, convert each set of raw R code into a working Shiny app
 - Lastly, after testing that each shiny app is working, bring them together into a singular Shiny app UI.

(Live Demonstration of App code)



- **Part 3: Implementing a Shiny app**

Method 1: Shinyapps.io using Rconnect

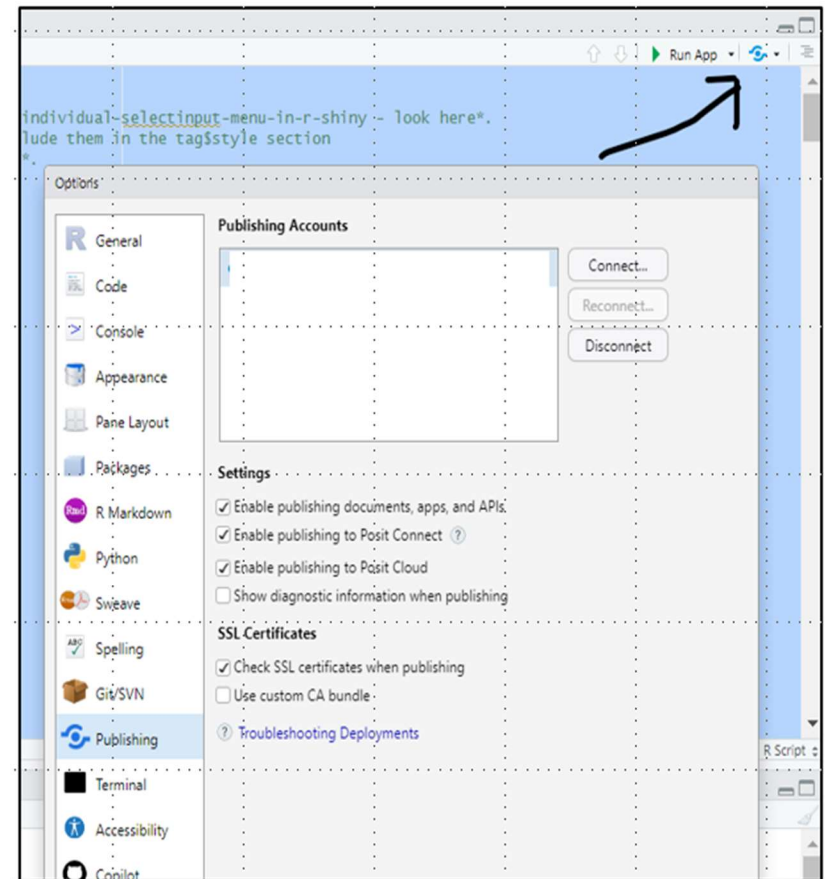
- Shinyapps.io is a platform as a service (PaaS) that hosts shiny app. You can deploy apps to this website using an R dev environment and the R package: rconnect
- Step 1: install Rconnect in Rstudio environment
- Step 2: create an <https://www.shinyapps.io/> account
- Step 3: create a domain name and account with your shinyapps.io profile.
- Step 4: use the setAccountInfo() function in the rconnect package to tie your Shiny apps to your shinyapps.io account. Example is below:
 - `rconnect::setAccountInfo(name="<ACCOUNT>", token="<TOKEN>", secret="<SECRET>")`
- Step 5: lastly, deploy the app using deployApp() function from rconnect

Method 1: Shinyapps.io using Rconnect

There is an alternative method to launching in Shinyapps.io

Step 1: follow all prior steps up to step 4 in the prior PowerPoint slide.

Step 2: once you have the rconnect set up, you will have a new Blue circular button titled 'Publishing' that should be accessible in your Rstudio environment. Click it and then click 'Connect'.



Method 1: Shinyapps.io using Rconnect

When executed, the contents of a given folder directory will be bundled together and deployed in an online application hosted by the shinyapps.io servers.

```
— Preparing for deployment —
i Bundling 12 files: 'app.R', 'FAR_WEST_RURAL_URB_CODES.csv', 'functions.R', 'GREAT_LAKES_RURAL_URB_
'NEW_ENGLAND_RURAL_URB_CODES.csv', 'OTHER_JURISDICTIONS_RURAL_URB_CODES.csv', 'PLAINS_RURAL_URB_COD
RAL_URB_CODES.csv', and 'SOUTHWEST_RURAL_URB_CODES.csv'
i Capturing R dependencies with renv:
✓ Found 131 dependencies
✓ Created 50,835,343b bundle
i Uploading bundle...
✓ uploaded bundle with id 8428680
— Deploying to server —
waiting for task: 1399364866
  building: Processing bundle: 8428680
  building: Parsing manifest
  building: Building image: 10224320
  building: Installing system dependencies
  building: Fetching packages
```



Method 1: Shinyapps.io Live Example

(Live Demonstration of Launching in Shinyapps.io)



Method 1: Shinyapp.io Final Thoughts

Benefits:

- You can deploy five (5) apps with 25 active hours for free. You can also use a paid account for access to user authentications.
- No hardware installation.
- Data in the cloud is encrypted at rest.
- Shinyapps.io runs within AWS

Potential pitfalls:

- the service itself is not audited against any security frameworks.
- Shinyapps.io apps are not HIPAA-compliant.
- Requires data uploads to the cloud.

Additional info:

- Other information: <https://posit.co/about/posit-and-the-gdpr-what-you-need-to-know/>

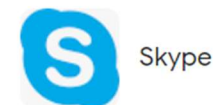
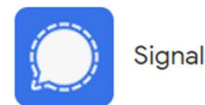
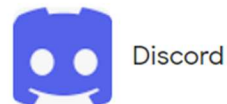
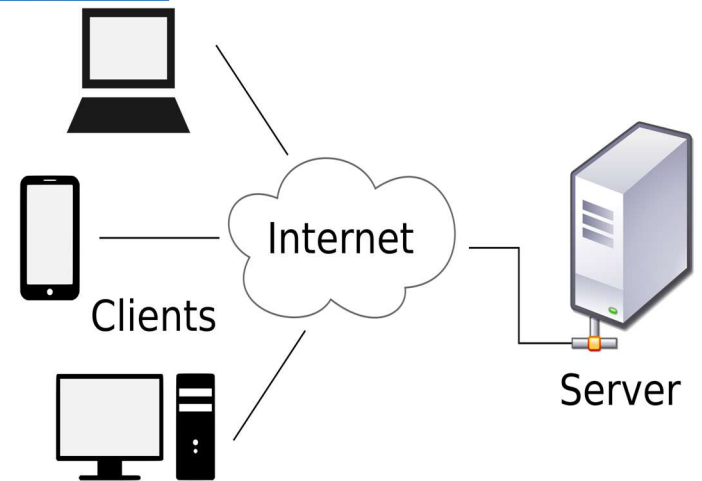


Method 2: Electron Framework (Proton)

- This method was developed by COLUMBUS COLLABORATORY based in OH.
- The organization was founded by several large companies in the region with the goal of sharing resources for rapid innovation. The company ceased operations in 2022.
- Company published in open source a rapid pace method of creating, executing, and delivering applications to stakeholders in compliance without risk to data security. This is what we call the Electron / Proton method of implementation.
- The method they created utilizes Electron and couples it with Rshiny.

Method 2: Electron Framework

- “Electron embeds Chromium and Node.js to enable web developers to create desktop applications” - <https://www.electronjs.org/>
- This allows users to make ‘thick client’ applications. These are applications with a client-server architecture. (IE: you make apps that are hosted on a browser and which may rely on HTML, CSS, and JavaScript for its UI – the code is rendered back via the Chromium Browser engine).
- Below are some examples of apps using a similar framework:





Method 2: Electron Framework

The coupling of Chromium and Nodejs with the Electron framework allows users to launch an instance of a browser window that is capable of hosting an application.

The Electron Method works by taking R and packaging it into a directory that is compatible with the electron framework (using Rportable) for conversion into a think-client application. There are several key elements that are needed for this to work:

- Nodejs - <https://nodejs.org/en>
- R Portable - <https://sourceforge.net/projects/rportable/>
- Electron - <https://www.electronjs.org>

Method 2: How Electron works with Shiny

- Thick client apps - provide the functionality of a full web app independently of a remote web server. (IE: your computer hosts both the app and the code locally on the device). This differs from a standard webpage where you go to a site and the changes you make as you navigate it are communicated back to the server.
- Node.js – the main process that launches the shiny app. (server side process within Electron Framework). Nodejs launches the app and redirects the ‘renderer’ (Chromium) to the URL where the shiny app is running (your local home IP address / local port).
- Rportable – A version of Rstudio that functions as a standalone application. It allows you to use R and rstudio without needing to install them locally. Within the electron framework,

Method 2: Electron Framework

- Step 1: download and install R Portable, Nodejs, and electron.
- Step 2: download or acquire the basic application framework provided by the github repo located here: <https://github.com/COVAIL/electron-quick-start>. Follow the guide below:

To Use

To clone and run this repository you'll need [Git](#) and [Node.js](#) (which comes with [npm](#)) installed on your computer.
From your command line:

```
# Clone this repository
git clone https://github.com/ColumbusCollaboratory/electron-quick-start
# Install Electron Packager (if first time)
npm install electron-packager -g
# Go into the repository
cd electron-quick-start
# Install dependencies
npm install
# Run the app
npm start
# Build the Executable/App
cd electron-quick-start
npm run package-win
OR
npm run package-mac
```

Note: If you're using Linux Bash for Windows, [see this guide](#) or use `node` from the command prompt.

Method 2: Electron Framework

Step 3: follow the sequence of instructions provided in the repo

- Step 3a - perform npm install to install electron to the file directory.
- Step 3b – go to the application page. And go to the Rwin.exe in either the Mac or Windows folders.
- Step 3c – R.exe in your computer terminal and begin procedure of installing packages needed for your Shiny app.
- Step 3d – if needed, update names of key files to be ‘app.R’ and ‘functions.R’.
- Step 3e – close out and then click npm run package-win or run package – mac from your terminal.

Step 4: Enjoy your executable application!

- Step 4a – you can compress and zip the file and send it to your identified users.



Method 2: Electron Live Example

(Live Demonstration of Columbus Collaboratory Method)



Method 2: Electron Framework Final Thoughts

- **Benefits of Electron Framework:**
- You can create a fully functional web-app that is not tied to a remote web server (IE:You run both the user interface (UI) AND you run it locally at the same time.
- In terms of security, this means your app will default to the same internet connection as well as security standards that are in use on your local device.
- Scalable - allows users to effectively download an instance of the application to their local device and run the application from there rather than a cloud based hosting application. In contrast to other methods, since the server and ui are both local to your device, there is no installation involved with Rstudio. This allows you to make apps quickly.



- **Part 4: Final Considerations**



Roadmap for GeocodIR

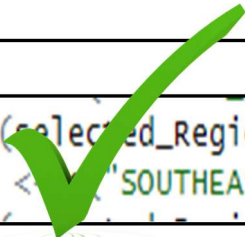
- Make data files and code available through Github or another open source depository.
- Fix issue with CT state data files.
- Modify Geocoding procedure so that null values are retained with an imputed value instead of being dropped. (IE: If a set of coordinates is not found in the census geocoder, the student's address is removed from the dataframe; we will add a procedure that identifies an approximate address based on the Zip code or other elements).
- Improve UI.
- Add button that allows for switching on and off different geocoding approaches (EG: make a selector for Crows Distance / Driving Distance).
- Add progress bar
- Add additional functionalities.

Namespace

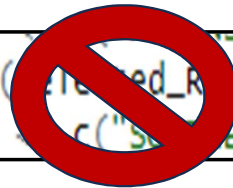
R is a case-sensitive programming language. When creating and calling functions in Rstudio, ensure that all variables and items are coded in the same name. If an item in your Shiny UI `selectInput` function() is listed as 'Southeast', ensure the same spelling is used in your code on the server side.

```
div(  
  selectInput("selected_region", "Select Region",  
    selectize = TRUE, choices = c("New England", "Mid East", "Great Lakes",  
      "Plains", "Southeast", "Southwest",  
      "Rocky Mountains", "Far West", "Other Jurisdictions"),  
    selected = NULL, multiple = FALSE), style="font-size: 24px; padding: 10px; color: black;"),  
div(  
  
```

```
} else if ((selected_region) == "Southeast") {  
  RURAL_URB <- "SOUTHEAST_RURAL_URB_CODES.csv")  
}
```



```
} else if ((selected_region) == "SOUTHEAST") {  
  RURAL_URB <- c("SOUTHEAST_RURAL_URB_CODES.csv")  
}
```



Data Types

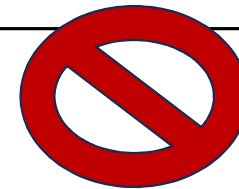
R supports a variety of Data types beyond the scope of numeric, string, and date types. Rshiny uses a data type called 'Reactive Values'. The procedure for assigning these values is different from most data types and is more akin to calling a function().

* Here you create a variable and assign it a value as a Null but reactiveVal() data type. You then may create a separate value that can be assigned the value that is being held in the reactiveVal() variable.

```
selected_geometry <- reactiveval()  
inst_name <- reactiveval()
```

```
inst_name_val <- IPEDS_ADDR$INSTNM[IPEDS_ADDR$INSTNM == selected_inst]  
inst_name(inst_name_val)
```


```
x <- "Houston, TX"
```




Directory Inputs

When running a set of code local on your desktop, you may use directory pathing that is local to your device. If you are converting the raw code into a shiny app, you must remove the local directory pathing in favor of one that is more general.

```
# Load the geocoding functions  
#Step1 execution button*.  
source("functions.R")
```



```
# Load the geocoding functions  
#Step1 execution button*.  
source("C:/Users/admin/Desktop/GeocodingFolder/functions.R")
```



Common Errors when Making Shiny apps

API calls

There may be some service restrictions to making API calls on a locally hosted application vs one that is hosted on the web. Be sure to check the terms of service agreements associated with a given service provider before incorporating their API calls into an executable app or an app that is hosted online.

```
# Geocode using the census method
df_chunk <- df_chunk %>%
  tidygeocoder::geocode(
    method = 'census',
    street = ADDRESS,
    city = CITY,
    state = STATE,
    postalcode = ZIP,
    lat = "latitude",
    lon = "longitude"
  )
```



```
Query completed in: 1.2 seconds
Passing 49 addresses to the US Census batch geocoder
Query completed in: 0.7 seconds
Warning: Error in : OSRM API request failed [414]
3: runApp
2: print.shiny.appobj
1: <Anonymous>
```





Common Errors when Making Shiny apps

Library Installations

There are two (2) problematic rstudio packages that may cause issues if you have both Rstudio Desktop and Rportable installed simultaneously on the same device. Both packages are common dependencies used in many core Rstudio packages.

- Lifecycle() – provides warnings to Rstudio users on the deprecation and status of library components.
- Rlang() - toolbox for working with base functions.
- When installing these into Rportable, use this installation instead:
install.packages("rlang", type = "binary")



Rstudio Packages for Executable Apps

Packages that can help create and launch apps:

- `install.packages("rhino")` <https://cran.r-project.org/web/packages/rhino/index.html>
- `install.packages("golem")` <https://cran.r-project.org/web/packages/golem/index.html>
- `remotes::install_github("ColumbusCollaboratory/photon")` (NOTE: This is not hosted on CRAN)

Additional thought:

* Previously, Shiny apps implementations could be done via Shiny Server Pro, Rstudio Connect, and others; however, these different subscriptions were phased out in favor of Posit Connect. <https://shiny.posit.co/r/articles/share/deployment-web/>



Benefits in Adding Rshiny apps in IR offices

Open Source Apps

Rstudio is an open source software that is free to use. Using Rstudio at its basic service level will not affect the overall cost of performing core IR / IE functions.

Professional Development

Rshiny functions are basically compilations of HTML, Javascript, CSS, and Rstudio code. Users may also install packages compatible with Rshiny that allow them to create and chain their own HTML, JS, and CSS code. This allows IR professionals to create applications while gaining exposure to new programming and markdown languages.

Functionality

With over 19,000 libraries available in the CRAN repository, there is hardly a limit to what can be accomplished with Rshiny applications.

Final Thoughts

Previously it took a team of four (4) Graduate level Computer Scientists to make applications. With AI assistance you can do the same output in a much smaller window of time.