Data Visualization

Contents

Set up environment

Clean the environment, clear the workspace and console… etc.

```
#Clear the workspace
rm(list=ls())

#Clear the console
cat("\014")
```

Always need to read in our packages from their libraries, and set our working directory!

```
#Include necessary libraries
x = c('data.table',
      'rstudioapi',
      'shiny',
      'viridis',
      'sf',
      'gganimate',
      'transformr',
      'tidyverse')

# # If you don't know if you've installed the packages for some of these
# # libraries, run this:
# install.packages(x)

lapply(x, library, character.only=TRUE)
rm(x)

#Set working directory (should be universal)
setwd(
  dirname(
```

```r
    rstudioapi::callFun(
      'getActiveDocumentContext'
    )$path
  )
)
```

Initiate Shiny

UI

First, we start a shiny user-interface and define what will be in it. In this case, an image, the image will contain a gif.

```r
# Shiny UI
shiny_ui = shinyUI(fluidPage(imageOutput("covid_gif")))
```

Server

Now, we have to launch a shiny server, somewhere to host the live version of R being run to execute the code. This is really a function which will house the code we want to execute.

There's a lot of stuff going on inside of here, but, fortunately for us smart-cookies, we've already seen and created most of it! We're doing great!

You can see our data acquisition, cleaning, wrangling, merging, and visualizing all going on together below. Whew. Like drinking from a firehose! We also add in two other components we haven't really had time to get to: `data.table` as an alternative to `dplyr` and animations, a super slick way to show how things change through time in R.

```r
shiny_server = shinyServer(function(input, output) {
  # Plot animated time lapse
  output$covid_gif = renderImage({
    # Collect COVID-19 Data from Johns Hopkins University (JHU) at the
    # following site:
    jhu_url = paste("https://raw.githubusercontent.com/CSSEGISandData/",
                    "COVID-19/master/csse_covid_19_data/",
                    "csse_covid_19_time_series/",
                    "time_series_covid19_confirmed_US.csv",
                    sep = "")

    # Read in the data from the site
    us_confirmed_long_jhu = read_csv(jhu_url) %>%
      # Rename columns for more informative/pertinent reference
      dplyr::rename(state_name = "Province_State",
                    country_region = "Country_Region",
                    Long = "Long_",
                    county_name = "Admin2",
                    county_fips = "FIPS") %>%
      # Reorganize the table to "long" version for merging later, using cases by day
      pivot_longer(!c(UID,iso2,iso3,code3,county_fips,county_name,Combined_Key,
                      state_name,country_region, Lat, Long),
                   names_to = "Date",
                   values_to = "cumulative_cases") %>%
      # Adjust JHU dates back one day to reflect US time, more or less
      mutate(Date = lubridate::mdy(Date) - lubridate::days(1)) %>%
      # Select only US states, not Canada provinces or any territories
      filter(country_region == "US") %>%
      # Sort the data by state and date
      arrange(state_name, Date) %>%
      # Group the data by county
      group_by(county_fips) %>%
      # Calculate incident cases per day, rather than cumulative count
      mutate(incident_cases = c(0, diff(cumulative_cases))) %>%
      # Ungroup by county for new summarization
      ungroup() %>%
      # Select only columns of interest
      dplyr::select(-c(country_region, Lat, Long)) %>%
      # Ensure no negative incident case records, generate week date
      mutate(incident_cases = ifelse(incident_cases<0,0,incident_cases)) %>%
      # Group by week for summarized data (comment-out for days)
      group_by(wk_of_yr = as.Date(paste0(strftime(Date,
                                                  format = "%Y%U"),
                                         1),
                                  "%Y%U%u"),
               county_fips) %>%
      # Summarize data by week
      mutate(wk_incident_cases = sum(incident_cases))

    # Convert to data.table for faster processing
    us_confirmed_long_jhu_dt = data.table(us_confirmed_long_jhu)

    # Set the key of the data.table to merge
    setkey(us_confirmed_long_jhu_dt,
           county_fips)

    # Collect Census data
    # Set the TIGRIS cache for future use
    options(tigris_use_cache = TRUE)

    # # Set and store (in .Renviron) the census API Key procured from
    # # https://api.census.gov/data/key_signup.html
    # census_api_key("YOUR API KEY HERE",
    #                install = TRUE)

    # Look-up table for available variables in Standard File 1 of Decennial Data
    chk_vars = load_variables(2010,
                              "sf1",
                              cache=TRUE)

    # # Create simple features object and plot for verification of data acquistion
    # census_decennial = get_decennial(geography = "county",
    #                                  year = 2010,
    #                                  variables = "P001001",
    #                                  geometry = TRUE,
    #                                   shift_geo = TRUE) %>%
    #   mutate(per_capita = value / 100000)
    #
    # census_decennial %>%
    #   ggplot(aes(fill=per_capita)) +
    #   geom_sf() +
    #   scale_fill_viridis_c(name="Population\n(x100,000)") +
    #   theme_void()

    # Create data.table from Decennial data.
    census_decennial_dt = data.table(get_decennial(geography = "county",
                                                   year = 2010,
                                                   variables = "P001001",
                                                   geometry = TRUE,
                                                   shift_geo = TRUE) %>%
                                       mutate(per_capita = value / 100000,
                                              GEOID = as.numeric(GEOID)) %>%
                                       dplyr::rename(county_fips = "GEOID"))

    # Set key of data.table to merge
    setkey(
      census_decennial_dt,
      county_fips
    )

    # Merge data
    spatial_data = census_decennial_dt[us_confirmed_long_jhu_dt]

    # Remove any records not occuring in both data sets
    spatial_data = spatial_data[!is.na(per_capita)]

    # Calculate per capita incident case rate
    spatial_data[,
                 `:=`(
                   pc_incident_cases = incident_cases / per_capita,
                   pc_wk_incident_cases = wk_incident_cases / per_capita
                 )
    ]

    # Log data for linear comparisons between days
    spatial_data[pc_incident_cases > 0,
                 pc_incident_cases_log := log10(pc_incident_cases)]
    spatial_data[pc_incident_cases == 0,
                 pc_incident_cases_log := pc_incident_cases]
    spatial_data[pc_wk_incident_cases > 0,
                 pc_wk_incident_cases_log := log10(pc_wk_incident_cases)]
    spatial_data[pc_wk_incident_cases == 0,
                 pc_wk_incident_cases_log := pc_wk_incident_cases]

    # Condense to weekly observations
    wk_spatial_data = spatial_data[,.(county_fips,
                                      geometry,
                                      wk_of_yr,
                                        pc_wk_incident_cases_log)]

    # Convert to tibble to remove duplicates (data.table cannot handle lists),
    # convert back to data.table for ggplot efficiency
    wk_spatial_data = data.table(distinct(as_tibble(wk_spatial_data)))

    # Remove NA values
    wk_spatial_data = wk_spatial_data[!is.na(wk_of_yr)]

    # Set maximum per capita incident case for plot legend
    # upper_limit = spatial_data[,max(pc_incident_cases_log)]
    upper_limit = wk_spatial_data[,max(pc_wk_incident_cases_log)]

    # A temp file to save the output.This file will be removed later by
    # renderImage
    outfile = tempfile(fileext='.gif')

    incidentcase_percapita_plot =
      # ggplot(spatial_data,
      #        mapping = aes(fill = pc_incident_cases_log,
      #                      geometry = geometry)) +
      ggplot(wk_spatial_data,
             mapping = aes(fill = pc_wk_incident_cases_log,
                           geometry = geometry)) +
      geom_sf(color = NA) +
      scale_fill_viridis_c(name = "COVID-19\nIncident Cases\n(per 100,000)",
                           limits = c(0,upper_limit)) +
      ggtitle("Date: {frame_time}") +
      theme(
        line = element_blank(),
        rect = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank(),
        legend.text = element_text(size = 12),
        legend.title = element_text(size = 16)
      ) +
      # transition_time(Date) #+
      transition_time(wk_of_yr)

    anim_save("outfile.gif",
              animate(incidentcase_percapita_plot))

    # Return a list containing the filename
    list(src = "outfile.gif",
         contentType = 'image/gif',
         width = 800,
         height = 700)},
    deleteFile = TRUE)
})
```

Populate the shiny environment

Now we have to actually call the shiny function to stitch everything we just made together!

```
shinyApp(shiny_ui,
         shiny_server)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```