A Brief Introduction to R-Shiny and Dashboards

Ryan Hafen & J. Hathaway

Agenda

1. Introduction to R-Shiny and Dashboards

3. Making our first R-Shiny dashboard

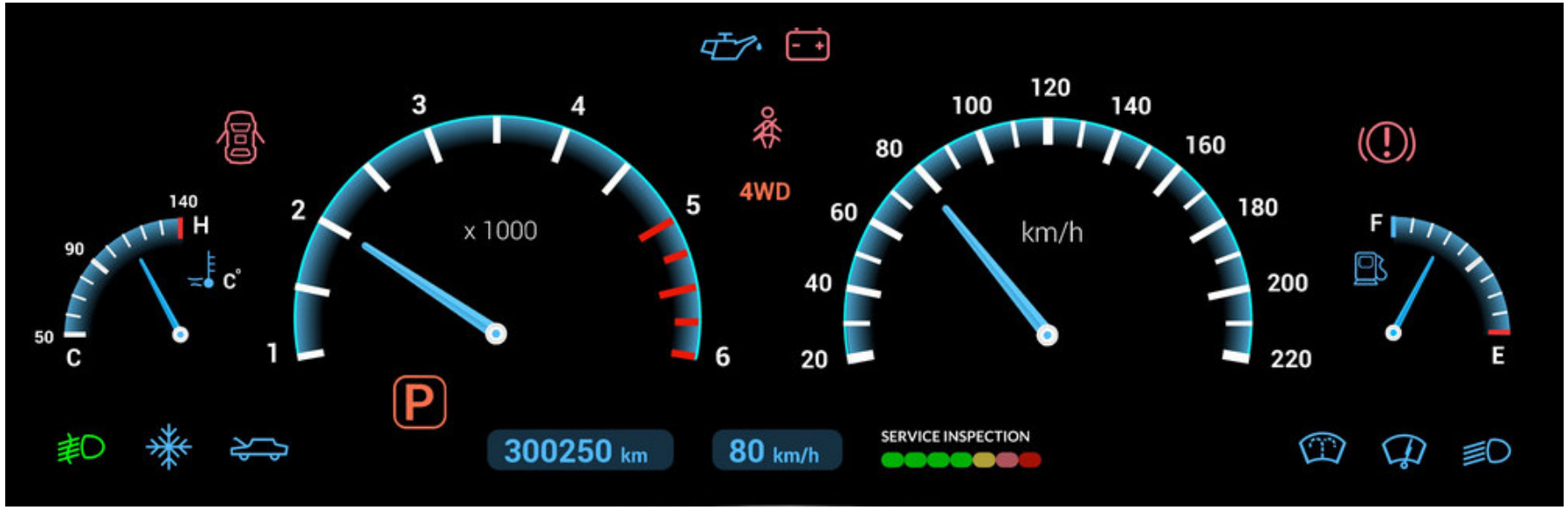
4. A few rules for dashboards

5. How do I learn more?

Introduction to R-Shiny and Dashboards

:Dashboards:

Opinionated views for decision making
that facilitate user input



What is R-Shiny?

An R package that makes it easy to build interactive web apps straight from the R language.

- RStudio Shiny website

What can I build with Shiny?

Government / Public sector

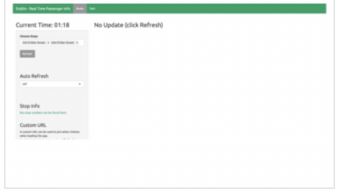
Mostly open data

Freedom of Press Index

Voronoys - Understanding voters' profile in Brazilian elections

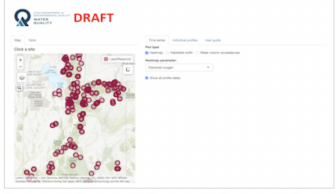
Crime Watch

Pasture Potential Tool for improving dairy farm profitability and environmental impact

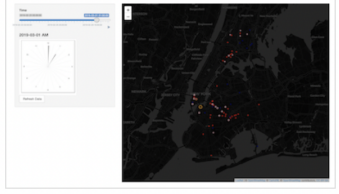


Dublin Transport Info

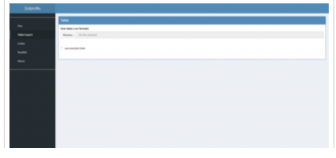
Locating Blood Banks in India

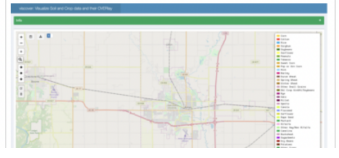


Utah Lake Water Quality Profile Dashboard



Animated NYC metro traffic





https://shiny.rstudio.com/gallery/

Why use R Shiny?

- Great for fast prototyping and fairly easy to use for someone who's not a programmer

- Good for dashboards or very simple apps used internally at your organization

- More customizable than many BI tools

- Easy to make use of R's stats/modeling/visualization packages

Why not use R Shiny?

- For non-trivial apps the codebase can become very complex and hard to follow

- Difficult to transition a shiny app to a support team, even if they know R/Shiny

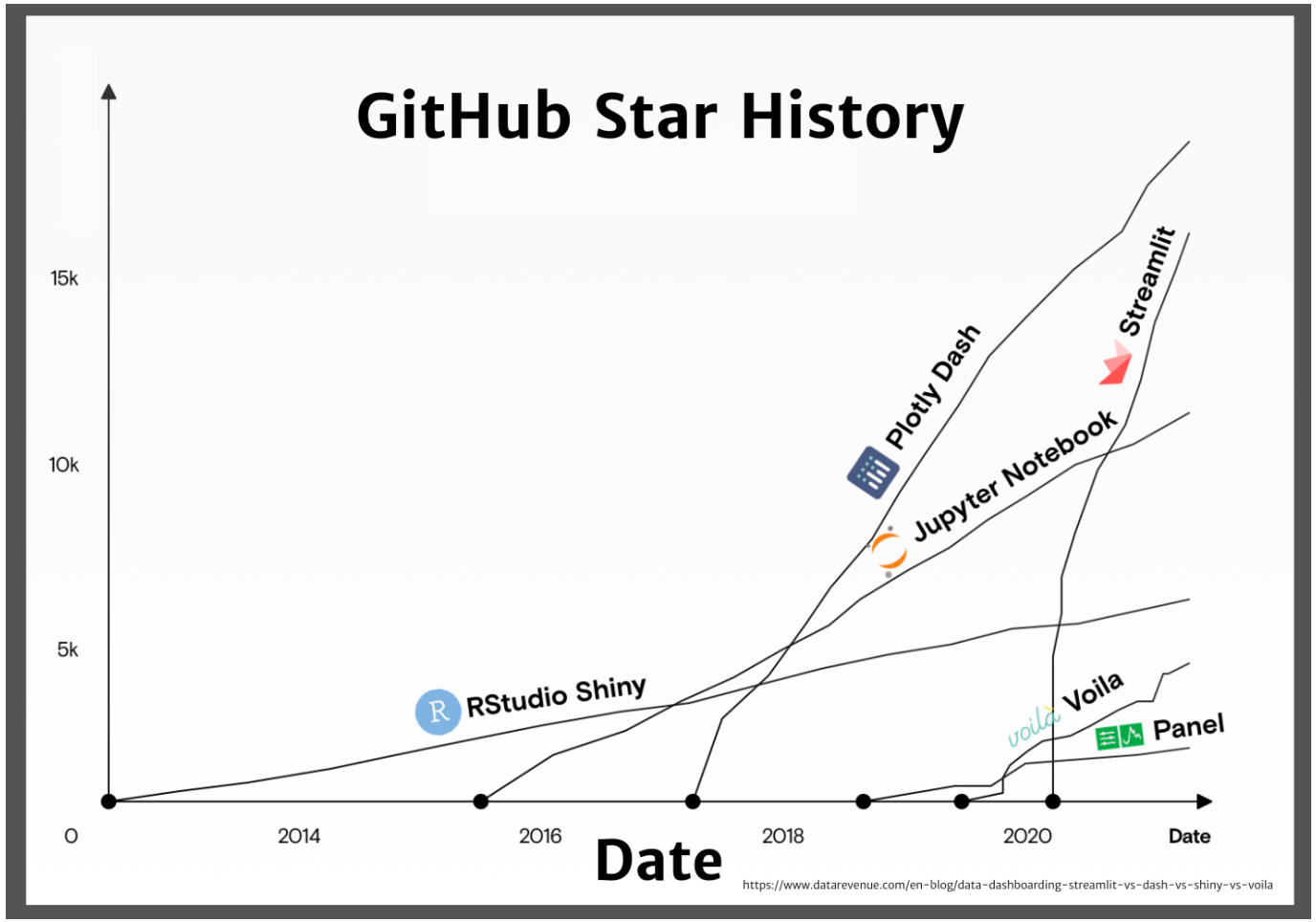
- Deployment requires a special server setup and hosting can be complicated or expensive

- In most cases, not suitable for production environments

- R is not a natural environment for specifying UI/UX

- Performance: UI interactions requiring round trips to R that could easily be handled in the web browser

What are some alternatives to R-Shiny?



Making our first R-Shiny dashboard

Thanks to shinyintro by Lisa Debruine

- We recommend her book for a smooth entrance into Shiny and ShinyDashboard
- We will introduce dashboarding with Shiny using the 'shinydashboard' package

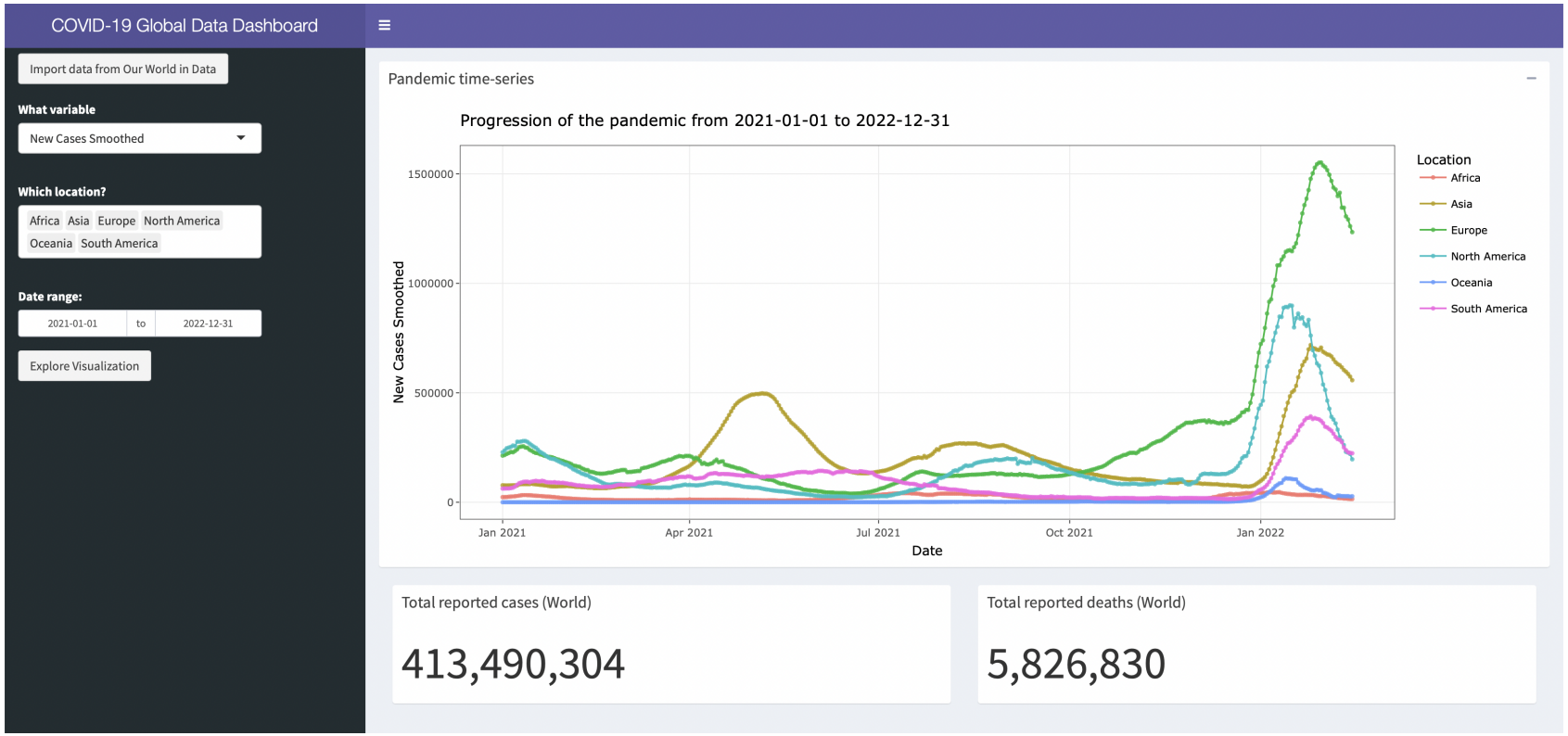
Using the `shinydashboard` package

The `shinydashboard` package is a collection of functions that make it easy to create dashboards.

You can find the code for this example at our RShinyDashboards Repository

Our first R Shiny Dashboard

- **Our Goal:** Give our user the opportunity to see reported COVID-19 cases for selected states over a specified time window using the [New York Times COVID-19 data](#).



Data

Source: [Our World in Data]

```
# A tibble: 161,909 × 67
   iso_code continent location    date       total_cases new_cases new_cases_smoothed total_deaths new_deaths
   <chr>    <chr>     <chr>       <date>           <dbl>     <dbl>              <dbl>        <dbl>      <dbl>
 1 AFG      Asia      Afghanistan 2020-02-24           5         5             NA              NA         NA
 2 AFG      Asia      Afghanistan 2020-02-25           5         0             NA              NA         NA
 3 AFG      Asia      Afghanistan 2020-02-26           5         0             NA              NA         NA
 4 AFG      Asia      Afghanistan 2020-02-27           5         0             NA              NA         NA
 5 AFG      Asia      Afghanistan 2020-02-28           5         0             NA              NA         NA
 6 AFG      Asia      Afghanistan 2020-02-29           5         0              0.714          NA         NA
 7 AFG      Asia      Afghanistan 2020-03-01           5         0              0.714          NA         NA
 8 AFG      Asia      Afghanistan 2020-03-02           5         0              0              NA         NA
 9 AFG      Asia      Afghanistan 2020-03-03           5         0              0              NA         NA
10 AFG      Asia      Afghanistan 2020-03-04           5         0              0              NA         NA
# … with 161,899 more rows, and 58 more variables: new_deaths_smoothed <dbl>, total_cases_per_million <dbl>,
#   new_cases_per_million <dbl>, new_cases_smoothed_per_million <dbl>, total_deaths_per_million <dbl>,
#   new_deaths_per_million <dbl>, new_deaths_smoothed_per_million <dbl>, reproduction_rate <dbl>,
#   icu_patients <dbl>, icu_patients_per_million <dbl>, hosp_patients <dbl>, hosp_patients_per_million <dbl>,
#   weekly_icu_admissions <dbl>, weekly_icu_admissions_per_million <dbl>, weekly_hosp_admissions <dbl>,
#   weekly_hosp_admissions_per_million <dbl>, new_tests <dbl>, total_tests <dbl>,
#   total_tests_per_thousand <dbl>, new_tests_per_thousand <dbl>, new_tests_smoothed <dbl>, …
```

Starting with our chart (part 1)

```r
# load packages and read data for chart
library(ggplot2)
library(readr)
library(dplyr)
library(snakecase)

d <- read_csv("https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/owid-covid-data.csv")

# our potential inputs
variable <- "new_deaths_smoothed_per_million"
locations <- c("Africa", "Asia", "Europe", "North America",
  "Oceania", "South America")
date_range <- as.Date(c("2020-01-01", "2022-12-31"))

ggplot(data = filter(d, location %in% locations,
  date >= date_range[1], date <= date_range[2]),
  aes_string("date", variable, color = "location")) +
  geom_point(size = 0.8, alpha = 0.8) +
  geom_line() +
  theme_bw() +
  labs(
    x = "Date",
    y = to_title_case(variable),
    color = "Location"
  ) +
  guides(color = guide_legend(
    override.aes = list(lty = NA, size = 5, shape = 15)))
```

The Shiny app basics

Every Shiny app has two key elements: `ui` and `server`. When using `shinydashboard` our `ui` is a little different.

```
header <- dashboardHeader(CODE FOR THE HEADER)
sidebar <- dashboardSidebar(CODE FOR THE INPUTS)
body <- dashboardBody(CODE FOR THE RESULTS)

ui <- dashboardPage(
  skin = 'purple',
  header,
  sidebar,
  body
)
```

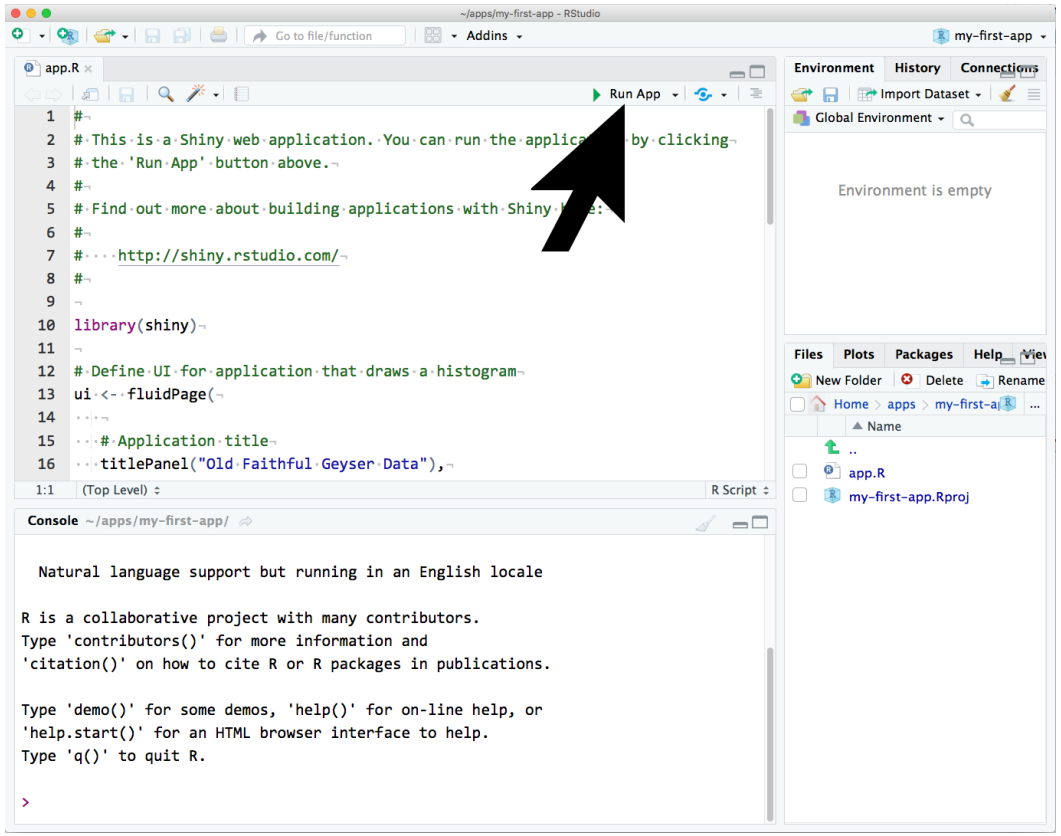
```
server <- function(input, output) {
    THE CODE THAT CREATE CHARTS, TABLES, AND TEXT FROM INPUTS
}
```

```
shinyApp(ui, server)
```

Our 'app_global.R' code for the dashboard is in the `app` folder of our repository

Using R-Studio for app development

Start R-Studio and create a new script **`app_global.R`** in a new **`project`** or **`folder`**. Once we have our script built we will click on **`Run App`**.



Building the interface: *Copy this code and paste it into* ***`app_global.R`***

```
# Setup ----
library(shiny); library(shinydashboard); library(readr); library(dplyr)
library(ggplot2); library(stringr); library(plotly); library(snakecase)
header <- dashboardHeader(title = "COVID-19 Global Data Dashboard", titleWidth = 400)
# Define UI ----
sidebar <- dashboardSidebar(width = 400,
  actionButton("load", label = "Import data from Our World in Data"),
  selectInput("variable", "What variable", "(data not loaded)",
    multiple = FALSE, selected = "(data not loaded)"),
  selectInput("location", "Which location?", "(data not loaded)",
    multiple = TRUE, selected = "(data not loaded)"),
  dateRangeInput("daterange", "Date range:",
    start = "2020-01-01", end = "2022-12-31"),
  actionButton("makeplot", label = "Explore Visualization")
)
body <- dashboardBody(
  box(title = "Pandemic time-series", solidHeader = TRUE, width = 16, collapsible = TRUE,
    plotlyOutput("timeseries", height = 500, width = "auto")),
  box(title = "Total reported cases (World)", solidHeader = TRUE, width = 6,
    div(style = "font-size:50px", textOutput("casetotal"))),
  box(title = "Total reported deaths (World)", solidHeader = TRUE, width = 6,
    div(style = "font-size:50px", textOutput("deathtotal")))
)

# Run the application ----
ui <- dashboardPage(skin = "purple", header, sidebar, body)
server <- function(input, output, session) {}
shinyApp(ui, server)
```

Server: Add body elements (Data & Totals)

Include these `reactiveVal()` assignments before the `server` function.

```
covid_data <- reactiveVal()
plot_data <- reactiveVal()
```

Now include our data import event within the `server <- function(input, output, session) { }`.

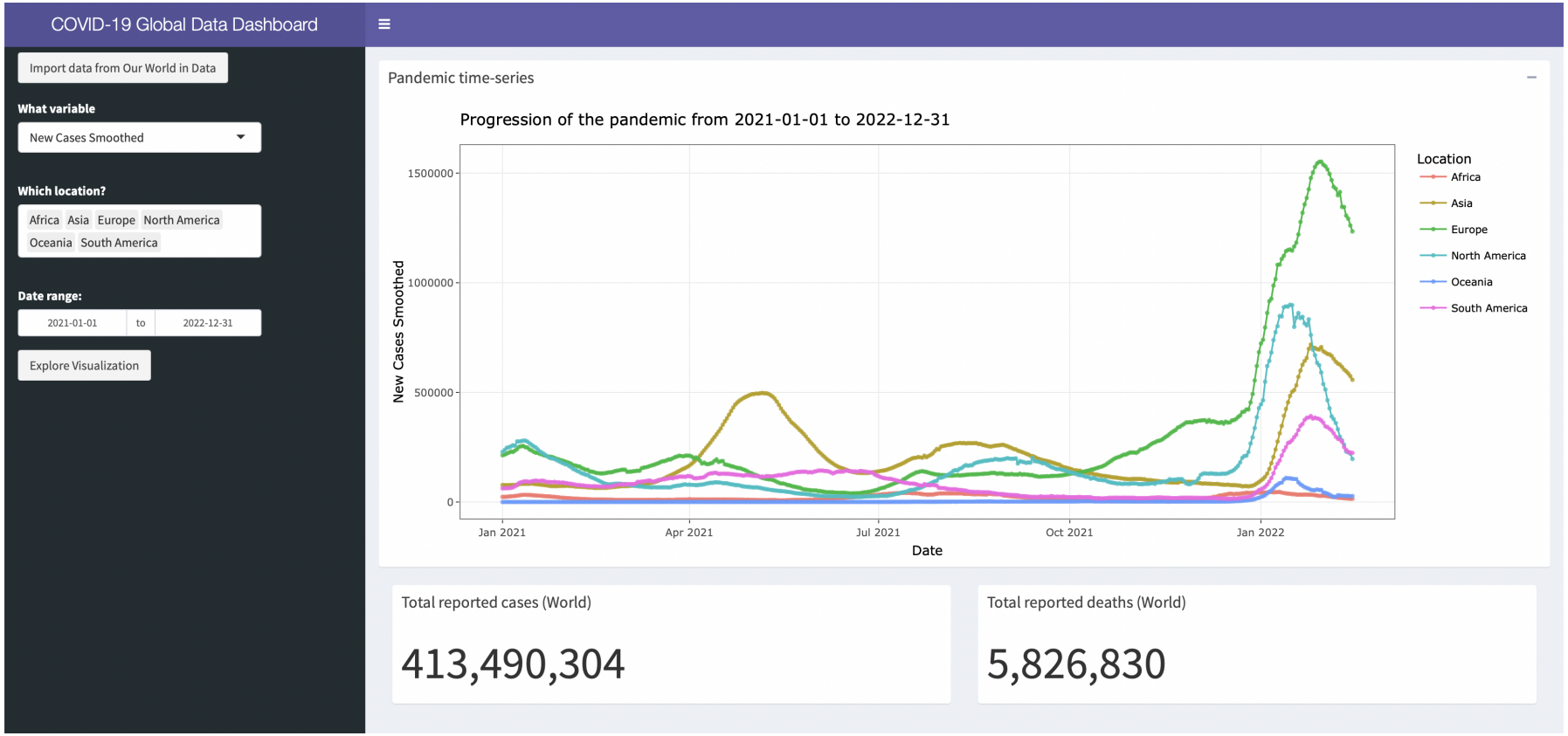
```
# Data ingestion and total number calculations. Set to work after `input$load` occurs from clicking on Import Data.
observeEvent(input$load, {
  url <- "https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/owid-covid-data.csv"
  owid_data <- readr::read_csv(url)
  covid_data(owid_data) # This reactive item now has is composed of the COVID-19 data.
  # assigning them into the output object allows them to be called by the `ui`
  output$deathtotal <- renderText(owid_data %>% filter(location == "World", date == max(date)) %>%
      pull(total_deaths) %>% sum() %>% format(big.mark = ","))

  output$casetotal <- renderText(owid_data %>% filter(location == "World", date == max(date)) %>%
      pull(total_cases) %>% sum() %>% format(big.mark = ","))

  # update inputs to reflect options found in the data
  choices <- names(owid_data); names(choices) <- to_title_case(choices)
  updateSelectInput(session, "variable", choices = choices,
    selected = "total_cases_per_million")
  updateSelectInput(session, "location", choices = unique(owid_data$location),
    selected = "World")
})
```

Server: Add body elements (Chart).

```
observeEvent(input$makeplot, {
  # notice use of `input$` to pull the user input values into the function.
  # "if" stops outputs from being created if user hasn't clicked import
  if (length(covid_data()) != 0) {
    newdat <- covid_data() %>%
      filter(location %in% input$location,
        date >= input$daterange[1],
        date <= input$daterange[2])
    # This reactive item now has is composed of the filtered data.
    plot_data(newdat)
    # notice the use of plot_data() as the data object to signal a reactive dataset.
    output$timeseries <- renderPlotly({
      ggplot(data = plot_data(),
        aes_string("date", isolate(input$variable), color = "location")) +
        geom_point(size = 0.8, alpha = 0.8) +
        geom_line() +
        theme_bw() +
        labs(
          title = paste0("Progression of the pandemic from ",
            input$daterange[1], " to ", input$daterange[2]),
          x = "Date",
          y = isolate(to_title_case(input$variable)),
          color = "Location") +
        guides(color = guide_legend(
          override.aes = list(lty = NA, size = 5, shape = 15)))
    })
  }
})
```

Our final dashboard

You can see the complete script at [app/app_global.R]



A few tips for creating dashboards

“Know the user,

they are not programmers

Define your production goals

- How often will it be used?
- How reliable does it need to be?
- What is the impact if it is innacurate?

Design with a purpose

To design without purpose is to design without a goal, without an objective or without a target. It's just design for design's sake. Not nearly as fulfilling as design with a direction. Without purpose, design is just decoration—aesthetics without true meaning.

THEIL

How do I learn more?

Online Reading Material

- Mastering Shiny
- Learn Shiny
- Get Started w/ shinydashboard
- Use shinydashboard
- Engineering Production-Grade Shiny Apps
- shinyjs
- shinydashboards
- Building Web Apps with R Shiny
- stat545 Shiny Tutorial
- R markdown: The Definitive Guide - Shiny
- YouTube: A Gentle Introduction to Creating R Shiny Webb Apps
- YouTube: Dynamic Dashboards with Shiny

Short Courses

- Building Web Applications with Shiny in R
- Shiny Fundamentals with R
- Building Data Apps with R and Shiny: Essential Training
- Creating Interactive Presentations with Shiny and R
- Interactive Visualization with R

Shinyverse of R packages

There are so many packages available. awesome-shiny has an extensive list. Use the below list to start your journey.

- shinydashboard
- shinydashboardPlus
- Shiny Themes
- shinymanager
- shiny.semantic
- shiny.react
- shiny.fluent
- shiny sense

Thanks