Two Way Analytics with R Shiny and Pokemon

[](https://medium.com/@mpkrass?source=post_page-----e9eae225fd46----------------------)

[Marshall Krassenstein](https://medium.com/@mpkrass?source=post_page-----e9eae225fd46----------------------)

[Jul 2](https://medium.com/swlh/two-way-analytics-with-r-shiny-and-pokemon-e9eae225fd46?source=post_page-----e9eae225fd46----------------------) · 12 min read

A picture containing cake, table, small, sitting

Description automatically generated

Photo by [Janis Fasel](https://unsplash.com/@janisfasel?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

**Background:**

When I first started working in Data Analytics, I learned to build Tableau dashboards populated by SQL queries on our data warehouse systems. SQL → Tableau was my ‘tech stack’ so to speak. For its purpose, that system is great: simple, refreshable, fast. Yet anyone who works with Tableau/SQL for a while can easily list the limitations in a SQL, Tableau pipeline. For example, one of the biggest limitations for me is that a Tableau dashboard cannot *ingest* any data input from the end-user. Never fear, using R Shiny we can build our own web applications to solve this problem with none of the full stack engineering knowledge required! This tutorial will show you how to create an R Shiny Application with a Sqlite backend in order to both ingest and report on data.

**Prerequisites**

Following this tutorial will require you to know R at a bare minimum. It would also help if you know Shiny although if you don’t know Shiny, reading through how easy it is to stand up a data driven application might inspire you to learn more. SQL experience would be good, but not required. With that, let’s get started!

**The Database**

The first thing we’re going to need is a database to store the information we’re presenting and taking in. For this tutorial, I’m going to use SQLite because it’s lightweight and easy to set up. In fact, Macs come with SQLite installed. If you have Windows, you can download SQLite from their official site in the link below.

**[SQLite Home Page](https://www.sqlite.org/index.html" \t "_blank)**

[Small. Fast. Reliable.Choose any three. SQLite is a C-language library that implements a small, fast, self-contained…](https://www.sqlite.org/index.html" \t "_blank)

[www.sqlite.org](https://www.sqlite.org/index.html" \t "_blank)

To see what we’re doing in the backend, I downloaded the DB Browser for SQLite tool. It is a database user interface that makes managing a database so simple that you hardly need to know SQL to build your dashboard.

**[DB Browser for SQLite](https://sqlitebrowser.org/" \t "_blank)**

[DB Browser for SQLite (DB4S) is a high quality, visual, open source tool to create, design, and edit database files…](https://sqlitebrowser.org/" \t "_blank)

[sqlitebrowser.org](https://sqlitebrowser.org/" \t "_blank)

Now we need to create the actual database that we’ll write to, which we can do by opening up the DB Browser for SQLite and creating a new database which it will save as a .db file. Finally we’ll need some data. Since Pokemon occupied a huge portion of my upbringing, I decided to make a basic Pokemon analytics app. I downloaded the ‘Complete Pokemon Dataset’ from Kaggle <https://www.kaggle.com/rounakbanik/pokemon>.

We’ll want to make a new shiny project and a script to read the csv into R and then ingest it into SQLite. To do this, I’m going to write a quick R script called *ingest\_pokemon.*I probably could have also just read the CSV into the database from DB Browser but we’re making a Shiny app so we may as well get our feet wet.

# Load Packages  
library(dplyr)  
library(DBI)  
library(RSQLite)# Read CSV  
poke\_data <- read.csv('pokemon.csv', stringsAsFactors = F)# Connect to Database  
con <- dbConnect(RSQLite::SQLite(), 'pokemon\_db.db')#Write to Database  
dbWriteTable(con, 'pokemon\_data', poke\_data, overwrite=T)#Read from database to prove everything is there  
str(dbReadTable(con, 'pokemon\_data'))

Now if we look at the data we can see that we have 41 columns of information across 801 different Pokemon. Inspecting the generation shows that there are Pokemon until the 7th generation, which I think was Sun and Moon. We can also look at the data in our database by going to ‘browse data’ on the pokemon\_data table as I show below:

A screenshot of a computer

Description automatically generated

**Building the Application**

Amazing! Now we have a dataset, so we need to do something with it. I’m going to build a dashboard with RShiny, one of RStudio’s most popular tools. If you build dashboards and you’ve used R, but never R Shiny before, I strongly suggest you give it a try. There are a ton of tutorials on the RStudio website for you to check out:

**[Shiny](https://shiny.rstudio.com/" \t "_blank)**

[Shiny comes with a variety of built in input widgets.](https://shiny.rstudio.com/" \t "_blank)

[shiny.rstudio.com](https://shiny.rstudio.com/" \t "_blank)

Alright, if you knew Shiny before, great! If not, I’m just going to assume you went through a lot of learning and maybe built a web app or two and continue on to building out this app.

I will structure my application as an app.R file to keep the server and UI functions in the same place. After making the base app structure which you can easily do in Rstudio by going to file → new file → Shiny Web App, I’ll start by giving the user the ability to choose a generation, and a Pokemon from that generation.

library(shiny)  
library(DBI)  
library(RSQLite)# Read in data from the SQLite Database  
setwd("~/Desktop/Shiny Project/pokeshiny")  
con <- dbConnect(RSQLite::SQLite(), 'pokemon\_db.db')  
poke\_data <- dbReadTable(con, 'pokemon\_data')# UI Function  
ui <- fluidPage(  
 # Application title  
 titlePanel("Pokemon Selector"),  
 # Choose a pokemon here  
 sidebarLayout(  
 sidebarPanel(  
 selectInput('poke\_gen',  
 'Generation',  
 choices=poke\_data$generation,  
 selected=1),  
 uiOutput('pokemon\_ui'),  
 ),  
   
 # Main Stuff  
 mainPanel(  
 h5('Output will go here')  
 )  
 )  
)# Server Function  
server <- function(input, output) {#Choose pokemon based on generation selected   
 output$pokemon\_ui <- renderUI({  
 choices <-poke\_data[which(poke\_data$generation==input$poke\_gen),'name']  
 selectInput("pokemon\_name",  
 "Choose a Pokemon:",  
 choices = choices)  
 })  
# Filter dataset for selected pokemon   
selected\_pokemon <- reactive({  
 poke\_selection <- poke\_data[which(poke\_data$name == input$pokemon\_name),]  
 })  
   
}# Run the application   
shinyApp(ui = ui, server = server)

The above code allows a user choose a generation from the dataset, and then filters the list of pokemon in the dataset to that generation. At this point I have a selection sidebar that looks like this:

A screenshot of a cell phone

Description automatically generated

Select a generation and Pokemon

Now I’ll add in some code to the server function that creates a butterfly chart comparing the selected Pokemon’s base stats to the average of the dataset. I created this in ggplot and then turned it into an interactive plotly chart by using the ggplotly function. I didn’t include the UI output portion here for brevity, but I’ll make sure to post a link to the full code at the end of this article.

output$bar\_comp <- renderPlotly({  
 req(input$pokemon\_name)  
 df <- selected\_pokemon() %>%  
 select(attack, defense, sp\_attack, sp\_defense, hp, speed) %>%  
 gather("Stat", "Value") %>%  
 mutate(side = 'Pokemon')  
 df\_avg <- poke\_data %>%  
 select(attack, defense, sp\_attack, sp\_defense, hp, speed) %>%  
 summarise\_all(list(mean)) %>%  
 summarise\_all(list(round)) %>%  
 gather("Stat", "Value") %>%  
 mutate(side = 'Average') %>%  
 mutate(Value = -Value)  
   
   
 df\_full <- rbind(df,df\_avg)l <- list(  
 font = list(  
 family = "sans-serif",  
 size = 12,  
 color = "#000"),  
 x = -.001, y = 0, orientation = 'h')  
   
 plot <- df\_full %>%   
 ggplot(aes(x = Stat, y = Value, group = side, fill = side,  
 text = paste0(ifelse(side=='Average', 'Group Average', input$pokemon\_name),  
 '<br>', Stat, ': ', abs(Value)  
 ))) +   
 geom\_bar(stat = "identity", width = 0.75) +  
 coord\_flip() +#Make horizontal instead of vertical  
 scale\_x\_discrete(limits = df$Stat) +  
 scale\_y\_continuous(breaks = seq(-300, 300, 50),  
 labels = abs(seq(-300, 300, 50))) +  
 labs(x = "", y = "") +  
 ggtitle(paste0("Comparison of ", input$pokemon\_name, " Stats Against Average")) +  
 theme(legend.position = "bottom",  
 legend.title = element\_blank(),  
 plot.title = element\_text(hjust = 0.5),  
 panel.background = element\_rect(fill = "white")) +   
 scale\_fill\_manual(values=c('#2a75bb','#ffcb05'),  
 name="",  
 breaks=c(input$pokemon\_name, "Average"),  
 labels=c(input$pokemon\_name, "Average"))  
   
 ggplotly(plot, tooltip = c("text")) %>%  
 layout(legend = l)  
 })

Ok so with that, we have a dynamic butterfly chart that changes for any Pokemon we select. I also added a little bit of code to display the abilities of the selected Pokemon:

­

A screenshot of a cell phone

Description automatically generated

We now have a chart with some basic filtering and tooltips. So far, we’ve done nothing outside what is easily possible in typical BI visualization tools. To make our app different, we can start by doing some basic database operations. For one, I have a lot of Pokemon I like, they are near and dear to my heart from many hours of childhood gameboy games. It would be interesting if I could build in some functionality for users to submit their favorite Pokemon.

Submitting data will not be challenging. First we’ll add a submit button to the UI in the side bar and a paragraph tag as a message for users.

# Include in the UI side panel  
p("If this is one of your favorite Pokemon, press submit below"),  
actionButton('submit', "Submit Favorite")

Then we’ll add an event handler for clicking the action button to the server function.

# Include in server. Writes to poke\_survey table after the user presses submit  
observeEvent(input$submit, {  
 submit\_date <- as.character(Sys.time())  
 df <- data.frame(generation = input$poke\_gen,   
 pokemon\_name = input$pokemon\_name,  
 submit\_date = submit\_date)# Append data to existing table if exists  
dbWriteTable(con, 'poke\_survey', df, append=T)  
 })

I’ll submit a couple of names, and then go back to the DB Browser to look at the results.

A screenshot of a social media post

Description automatically generated

DB Browser results from clicking the submit button

Boom, in just a few lines of code, we have user submitted data. If you can do that on Tableau outside of using ***a lot***of custom javascript, feel free to reach out to me and let me know. And as we can see in the image above, I also capture the exact date of the submission, which suggests we can write in any data at our disposal when a user clicks submit.

Unfortunately, as I wrote that last paragraph and admired the picture of our new submission table, I realized that I like Sudowoodo, but I don’t know if it’s one of my favorite Pokemon. In addition, it would have been nice to confirm with the user before saving the data. Finally, after 7+ generations of Pokemon, I’m not sure I even remember what they all look like anymore.

Let’s deal with my memory problems first. It turns out someone made another post on Kaggle including images of all the Pokemon (<https://www.kaggle.com/vishalsubbiah/pokemon-images-and-types>). Besides the exciting notion that bigger nerds than me exist, this dataset also allows an image of the selected Pokemon pop up in the application with surprisingly little code.

# Include in the UI side panel  
imageOutput("pokemon\_image", inline = T)# Server piece  
output$pokemon\_image <- renderImage({  
 req(input$pokemon\_name)  
 # filename is ./images/`pokemon\_name`.png  
 # Won't work with the .jpg images which I should fix at some point  
 filename <- normalizePath(file.path('./images',  
 paste(tolower(input$pokemon\_name), '.png', sep='')))  
   
 # Return a list containing the filename  
 list(src = filename)  
 }, deleteFile = FALSE)

And now we have images pop up when we select a Pokemon as in the example below:

A screenshot of a cell phone

Description automatically generated

I actually do love Ducklett because it feels like it spawned in a moment when the creators had a mental lapse, and said ‘screw it, let’s just make a duck Pokemon and call it ‘Ducklett’. Like, it’s just a blue-ish duck. I could go outside my house and find a closely related animal right now.

Anyway, before I reveal to the world how much I love Ducklett, I want to give myself a chance to confirm I love it. Below, I use Dean Atali’s fabulous ‘shinyalerts’ package to make a confirmation dialog popup.

# Submit button pop up  
observeEvent(input$submit, {  
 results <- shinyalert(  
 title = "Submit Pokemon?",  
 text = sprintf("Click Confirm to submit %s as one of your favorites",input$pokemon\_name),  
 closeOnEsc = TRUE,   
 closeOnClickOutside = TRUE,  
 html = TRUE,  
 type = "warning",  
 showConfirmButton = TRUE,  
 showCancelButton = TRUE,  
 confirmButtonText = "Submit",  
 confirmButtonCol = "#539BBD",  
 cancelButtonText = "Cancel",  
 inputId = 'submission\_alert',  
 callbackR = function(value) {if(input$submission\_alert){shinyalert("Congratulations! You submitted something")}}  
 )  
 })  
   
 observeEvent(input$submission\_alert, {  
 if (input$submission\_alert) {  
 submit\_date <- as.character(Sys.time())  
 df <- data.frame(generation = input$poke\_gen,   
 pokemon\_name = input$pokemon\_name,  
 submit\_date = submit\_date)  
 dbWriteTable(con, 'poke\_survey', df, append=T)  
 }  
 })

A screenshot of a cell phone

Description automatically generated

Shinyalert confirm submission modal

Beautiful. At this point we have an application that shows information and ingests information. But we can’t actually see what was submitted. To deal with that, I’m going to create a new tab that will display some summaries of the information. First, I need to make a navigation bar. And while I’m doing that, I may as well format my dashboard by adding an icon, making use of the ‘shinythemes’ package, and adding a border around my plot. This will change the UI function in a lot of places so I have the complete UI code below:

# UI Function  
ui <- navbarPage(selected = "Selector",  
 title=div(tags$img(src="poke\_ball.png", height =50),  
 style="margin-top: -25px; padding:10px"),  
 #theme = "journal",  
   
 windowTitle="Clerb is Crey",  
 tabPanel("Selector",   
 fluidPage(  
 theme=shinythemes::shinytheme('flatly'),  
 tags$head(  
 tags$link(rel = "icon", type = "image/png", href = "pika\_logo.png")  
 ),  
 useShinyalert(),  
 tags$head(  
 tags$style("  
 #abilities{  
 display:inline  
 }"),  
 ),# Application title  
 fluidRow(  
 column(width=4,  
 titlePanel("Pokemon Selector"),  
 div(p('Choose a Pokemon and compare their base stats against   
 the average in the dataset'),style='width:100px; display:inline'))  
 ),  
 # Sidebar with a slider input for number of bins   
 sidebarLayout(  
 sidebarPanel(  
 selectInput('poke\_gen',  
 'Generation',  
 choices=unique(poke\_data$generation),  
 selected=1),  
   
 uiOutput('pokemon\_ui'),  
   
 p("If this is one of your favorite Pokemon, press submit below"),  
 div(actionButton('submit', "Submit Favorite"),  
 imageOutput("pokemon\_image", inline = T), style='margin-top:-10px')  
 ),# Show a plot of the generated distribution  
 mainPanel(  
 div(plotlyOutput('bar\_comp'),style="background: margin-top:-85x; border-style: groove; padding-right:10px"),  
 div(  
 h5("Abilities:", style="display:inline"),  
 textOutput('abilities')  
 ),  
 )  
 )  
)),  
 tabPanel("Survey Display",  
 fluidPage(  
 fluidRow(  
 column(width=6,  
 h3('Top 10 Pokemon By Votes'),  
 plotlyOutput('survey\_by\_pokemon'),  
 ),  
 column(width=6,  
 h3('Count of Favorites by Generation'),  
 plotlyOutput('survey\_by\_generation'),  
 )  
 ),  
 fluidRow(  
 h3("Survey Results"),  
 DT::dataTableOutput('survey\_overall')  
 )  
 )  
 ),  
 tabPanel("Analysis",  
 fluidPage(  
 h3("Pokemon Clustering Will go here")  
 )))

A screenshot of a cell phone

Description automatically generated

So fine..

Now I’m going to submit a few entries and show the results on the Survey Display tab. I built out two graphs to summarize the counts by Pokemon submitted and Pokemon generation in the code below.

#Update Survey Data  
 survey\_data <- reactive({  
 input$submit  
 con <- dbConnect(RSQLite::SQLite(), 'pokemon\_db.db')  
 poke\_survey <- dbReadTable(con, 'poke\_survey')  
 dbDisconnect(con)  
 poke\_survey  
 })  
   
 output$survey\_overall <- DT::renderDataTable({  
 survey\_data()  
 }, rownames = FALSE,  
 options = dt\_config)  
   
 output$survey\_by\_pokemon <- renderPlotly({  
   
 data <- survey\_data() %>%  
 group\_by(pokemon\_name) %>%  
 summarise(Count = n()) %>%  
 select(Count, Pokemon = pokemon\_name) %>%  
 arrange(desc(Count), desc(Pokemon))  
 data <- data[1:10,]  
 print(data)  
 order <- rev(unique(data$Pokemon))  
   
 g <- ggplot(data, aes(x=Pokemon, y=Count, fill = Count,  
 text = paste0('Pokemon:', Pokemon,  
 '<br>Votes: ', Count))) +  
 geom\_col(color='black') +   
 scale\_x\_discrete(limits = order) +   
 coord\_flip() +  
 xlab("") + ylab("") +  
 theme(legend.position = "none",  
 panel.background = element\_rect(colour = 'white'),  
 plot.background = element\_rect(fill = "transparent",colour = NA))  
   
 ggplotly(g, tooltip = c("text"))  
   
 })  
   
 output$survey\_by\_generation <- renderPlotly({  
 data <- survey\_data() %>%  
 group\_by(generation) %>%  
 summarise(Count= n()) %>%  
 select(Count, Generation = generation)  
   
 g <- ggplot(data, aes(x=Generation, y=Count, fill=Count,  
 text = paste0('Generation:', Generation,  
 '<br>Votes: ', Count))) +  
 geom\_col(color = 'black') +  
 ylab("") +  
 theme(legend.position = "none",  
 panel.background = element\_rect(colour = 'white'),  
 plot.background = element\_rect(fill = "transparent",colour = NA))  
 ggplotly(g, tooltip = c("text"))  
   
 })

If I make a few more submissions, we’ll see the results appear on the dashboard in real time.

A screenshot of a social media post

Description automatically generated

Ducklett is the greatest Pokemon but the first 150 are the greatest generation

**Parting Words**

That’s it! If you’ve stuck with me up to this point, you can see that building web applications that ingest data is easy with Shiny. Since SQLite has file-locking built in, no user’s updates will overwrite those of another user even if they’re made at the exact same time. I haven’t shown how to modify or delete entries yet, but with the infrastructure set up that we have, doing so will be a quick fix. You can view the application at the link below to see the contents of this tutorial as well as some new features I’ve been working on. You can grab the full script on my Github repo here: <https://github.com/mpkrass7/shiny_pokemon>

**[PokemonShinyApp](https://marshallp.shinyapps.io/pokeshiny/" \t "_blank)**

[marshallp.shinyapps.io](https://marshallp.shinyapps.io/pokeshiny/" \t "_blank)

One last thought: If you’re at all worried you can’t build a web application like this because you haven’t done it before, I really encourage you to give it a try. Everyone starts somewhere, and once you get the hang of it, you’ll find building your application to be approachable and actually kind of fun.

This is my first article on Medium. If this article was useful or you have questions expanding on this topic, feel free to leave a comment and I’ll do my best to help. If you like Pokemon, also feel free to say so! I learned to spell ‘flamethrower’ before I learned how to spell ‘chair’ because of Pokemon Blue. Best of luck!

**Edit**

After noticing that submissions would not persist on shinyapps.io I stumbled on this post from 2018 <https://community.rstudio.com/t/shinyapps-io-and-sqlite-as-persistent-local-data-storage/19361>. Apparently, while the free version of shinyapps supports SQLite databases, it does not yet support persistent storage outside of paid versions of the server such as RStudio Connect. You can learn more about storage for shiny apps below:

**[Persistent data storage in Shiny apps](https://shiny.rstudio.com/articles/persistent-data-storage.html" \t "_blank)**

[Shiny apps often need to save data, either to load it back into a different session or to simply log some information…](https://shiny.rstudio.com/articles/persistent-data-storage.html" \t "_blank)

[shiny.rstudio.com](https://shiny.rstudio.com/articles/persistent-data-storage.html" \t "_blank)

For those who need an easy way to collect information remotely, I changed the application to use a remote MySQL host, and I hope to write a tutorial on it in the coming days.