**Two Way Analytics with SQLite and R Shiny**

A picture containing road, blue

Description automatically generated

When I first started working in Data Analytics, I got really good at building 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 any past Tableau/SQL monkey can easily list the limitations in a SQL Tableau pipeline. One of the biggest limitations for me is that a Tableau dashboard cannot *ingest* any data input from the end-user. Never fear, there are easy solutions to this problem! This tutorial will show you how to create a Shiny Application with a Sqlite backend in order to ingest or change information as well as report on the data.

In writing this, I expect you to know R and at least a little bit of SQL. 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 educate yourself.

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 and gain more information in the link below.

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

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

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

Now we need to create a 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

Amazing! We have a dataset so now we need to do something with it. I’m now going to build adashboard 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’s a ton of tutorials right on the RStudio website:

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

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)

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 generatedUI to select a Pokemon

Now I’ll add in some code to the server function that creates a butterfly chart comparing the selected pokemon 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 generatedSelecting a Pokemon now dynamically generates a butterfly chart

Ok, so we have a chart with some basic filtering. So far, we’ve done nothing outside what is easily possible in typical BI visualization tools. Now we can start 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 effect of 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)  
 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 generatedpoke\_survey table results

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 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 that’s one of my favorites. 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. Besides the exciting suggestion that bigger nerds than me exist, by downloading this dataset, I can have an image of the selected pokemon pop up in the application with some quick 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 the image pop up as below:

A screenshot of a cell phone

Description automatically generatedAn amazing image of Ducklett

I actually do love Ducklett because it feels like a moment when the creators just had a mental lapse, and said ‘screw it, let’s just make a duck Pokemon and call it ‘Ducklett’. Like, it’s just a bluish duck. I could go outside my house and find one of those write now.

Any way, before I reveal to the world how much I love Ducklett, I want to give myself a chance to confirm I love it. I’m going to use Dean Atali’s fabulous shinyalerts package to make a confirmation dialog.

# Modal functionality  
 observeEvent(input$submit, {  
 results <- shinyalert(  
 title = "Submit Pokemon?",  
 text = "Click Confirm to Submit",  
 closeOnEsc = TRUE,   
 closeOnClickOutside = TRUE,  
 html = TRUE,  
 type = "warning",  
 showConfirmButton = TRUE,  
 showCancelButton = TRUE,  
 confirmButtonText = "Submit",  
 confirmButtonCol = "#539BBD",  
 cancelButtonText = "Cancel",  
 inputId = 'submission\_alert'  
 )  
 })  
   
 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 generatedSubmission Modal

Beautful