Hacking My Way to My First Shiny App

Adam Reevesman

10/3/2018

https://github.com/areevesman/MSDS610-Presentation

Introduction

This document will serve as a tutorial for recreating this shiny application. The application is a dashboard that allows users to explore Major League Baseball (MLB) data. When a user selects a team, year, and statistic they are interested in, the application renders a plot based on those values.

The tutorial will also touch on the web scraping that was necessary to gather the data.

The Idea

baseball-reference.com is a website that hosts a ridiculous amount of baseball statistics. In particular, it has these tables that record certain aspects of every regular season game for a given team and specific year.

The application will synthesize that data for every current MLB team, for all of the years they have exsited (under that name).

First Step: Gather the data

Two python scripts (in ipynb format) were created to scrape the data. They utilize the `bs4` (BeautifulSoup) module, which is explained in more detail and used for a similar purpose in this youtube tutorial. They live here.

There is one script for the tables referenced above and there is one for this table.

One csv file was saved for each table. Additional csv files were created by combining the tables for each team and adding some additional columns (discussed more later). The data files live here and are named as "teamIDs.csv", "STL2006.csv", or "STL_all_years.csv".

Building the application

According to https://shiny.rstudio.com/, "Shiny is an R package that makes it easy to build interactive web apps straight from R. You can host standalone apps on a webpage or embed them in R Markdown documents or build dashboards. You can also extend your Shiny apps with CSS themes, htmlwidgets, and JavaScript actions." Many resources and tutorials can be found here. See this gallery as well for help with implimentation of more specific features.

There is one important piece of information to know before moving forward.

Every application needs a user-interface definition and server logic. The user-interface controls how the application looks and the server logic controls what the application does.

In addition to the `ui` and `server` scripts, I have included a script to define global functions and variables.

Define Global Variables and Functions

This code gets saved into a script called `global.R`.

```r
library(shiny)
library(shinythemes)
library(RCurl)
library(plyr)
library(dplyr)
library(stringr)
library(ggplot2)
library(plotly)

### Functions to interact with github reposity and manipulate data ###

data_folder <- "https://raw.githubusercontent.com/areevesman/mlb_app_edits/master/data/"


#read in a csv file by name from data_url
#ex: get_table_by_name("STL2006.csv")
```

```r
get_csv_by_name <- function(name, data_url=data_folder){
  myCSV <- getURL(paste(data_url, name, sep=''))
  myDF <- read.csv(text=myCSV, stringsAsFactors = F) %>%
    as_tibble()
  return(myDF)
}


#add columns to the original team data from github
#year argument allows a coulmn to be added for year
add_columns <- function(original_team_data, year){

  team_data <- original_team_data

  team_data$year <- rep(year, times = nrow(original_team_data))
  team_data$games_ahead <- str_replace_all(team_data$games_behind, 'Tied', '0')
  team_data$games_ahead <- gsub(pattern = 'up', replacement = '-', x = team_data$games_ahead)
  team_data$games_ahead <- gsub(pattern = ' ', replacement = '', x = team_data$games_ahead)
  team_data$games_ahead <- -1*as.numeric(team_data$games_ahead)
  team_data$run_diff <- team_data$runs - team_data$runs_allowed
  team_data$r_so_far <- cumsum(team_data$runs)
  team_data$ra_so_far = cumsum(team_data$runs_allowed)
  team_data$rd_so_far = cumsum(team_data$run_diff)
  team_data$wins_so_far = cumsum(!grepl(x = team_data$win_or_loss, pattern='L'))
  team_data$losses_so_far = cumsum(!grepl(x = team_data$win_or_loss, pattern='W'))
  team_data$win_loss_differential = team_data$wins_so_far - team_data$losses_so_far
  team_data$record_so_far =  team_data$wins_so_far / (team_data$wins_so_far + team_data$losses_so_far)
  team_data$who_and_where = ifelse(team_data$home_or_away == "@",
                                   paste(team_data$team, "@", team_data$opponent),
                                   paste(team_data$opponent, "@", team_data$team))
  team_data$winner = ifelse(team_data$runs > team_data$runs_allowed,
                            team_data$team,
                            team_data$opponent)

  return(team_data)
}


#get one team's csv's over all years in one csv
combine_years <- function(team, start_year){
  return(get_csv_by_name(paste(team, "_all_years.csv", sep="")))
}



# get teamID data and fix minor data inconsistencies
teamIDs <- get_csv_by_name("teamIDs.csv")
teamIDs[1,] <- c("LAA", "LAA", "Los Angeles Angels of Anaheim", 2005)
teamIDs <- teamIDs[c(2:14,1,15:nrow(teamIDs)),]



### Define Global Variables ###
```

```r
#all mlb teams (as "STL")
team_choices <- teamIDs$Team_ID
#number of years team has been around
team_num_years <- 2018 - as.integer(teamIDs$First_Year)
#name the vectors with full team names
names(team_choices) <- teamIDs$Full_Team_Name
names(team_num_years) <- teamIDs$Full_Team_Name

#choices of statistics to select
select_stat_choices <- list("Record" = "record_so_far",
                            "Win-Loss Differential" = "win_loss_differential",
                            "Games Ahead/Behind in Division" = "games_ahead",
                            "Cumulative Wins" = "wins_so_far",
                            "Runs" = "runs",
                            "Runs Allowed" = "runs_allowed",
                            "Run Differential" = "run_diff",
                            "Cumulative Runs" = "r_so_far",
                            "Cumulative Runs Allowed" = "ra_so_far",
                            "Cumulative Run Differential" = "rd_so_far")
```

Defining the User-Interface and Server Logic

Adding this code to the **user-interface** definition will define the layout of the app and create a select box for team. The `uiOutput("team")` will correspond to an `output$team` object in the server logic.

```r
  fluidPage(

    #select theme
    theme = shinytheme("yeti"),

    # Application title
    titlePanel("A History of Major League Baseball's Active Franchises"),

    p("You can use this application to examine various performance metrics of Major League Baseball tea
      strong("Team,"),
      strong("Year"),
      " and the ",
      strong("Statistic") ," of interest."),

    p('Click the "Play Animation" button in the ', strong('Year'), ' section to see how performance cha

    p("Use the buttons in the top-right corner of the plot for a closer look."),

    # select layout
    verticalLayout(

      # create panel with a select box for team
      wellPanel(

        #select box for team
        selectInput("select_team",
                    label = h3("Team"),
                    choices = team_choices,
                     selected = team_choices[1]),

        #tell server to render more some ui (for slider)
        uiOutput("team")

      )
    )

  )
```

Adding the following code to the **server** logic will render make the app render the slider once a team is selected.

```
  #first year in mlb for selected team
  start_year <- reactive({
    as.numeric(teamIDs[which(teamIDs$Team_ID == input$select_team), "First_Year"][[1,1]])
  })


  #server will render some ui corresponding to "team" (slider box)
  output$team <- renderUI({

    sliderInput("slider",
                label = h3("Year"),
                min = start_year(),
                max = 2017,
                value = start_year(),
                sep = "",
                round = TRUE,
                step = 1,
                animate = animationOptions(interval = 900,
                                           playButton = "Play Animation"))
  })
```

We can finish the **user-interface** definition with a select box for the statistic and the `plotlyOutput` call to create a select box for the statistic and to tell the server to create a plot.

```
#select box for statistic
selectInput("select_stat",
            label = h3("Statistic"),
            choices = select_stat_choices,
            selected = "Runs"),

#will tell server to render plotly object
plotlyOutput("detailed_plot")
```

This code completes the **server** logic to reder a plot that is specific to the team, year, and statistic of interest.

```
  #get a team's data over all years as a tibble
  team_all_years <- reactive({
    combine_years(input$select_team)
  })

  #get limits for plot
  limit_data <- reactive({
```

```r
  stat <- input$select_stat
  #get min and max for stat over all years
  var <- team_all_years()[,c("year", stat)] %>%
    mutate_all(function(col){
      col[is.na(col)] <- "0"
      col <- as.numeric(col)
      col}) %>%
    summarise(minimum = min(as.numeric(eval(parse(text = input$select_stat))), na.rm = TRUE),
              maximum = max(as.numeric(eval(parse(text = input$select_stat))), na.rm = TRUE))


  #lower and upper bound for plot y-axis limits
  c(var[[1,"minimum"]], 1.1*var[[1,"maximum"]])

})



#for plotly
team_year_data <- reactive({
  get_csv_by_name(paste(input$select_team,
                        input$slider,
                        ".csv",
                        sep="")) %>%
    add_columns(year = input$slider)
})



#render the plotly object
output$detailed_plot <- renderPlotly({

  x <- 1:nrow(team_year_data())
  y <- team_year_data()[[input$select_stat]]
  hover <- paste(team_year_data()[["date"]], "\n",
                 team_year_data()[["who_and_where"]], "\n",
                 "Score: ", team_year_data()[["runs"]], "-", team_year_data()[["runs_allowed"]],
                 ", ", team_year_data()[["winner"]], "\n",
                 sep = '')

  team_year_data() %>%
    plot_ly(x = ~x,
            y = ~y,
            hovertext = hover,
            hoverinfo = "text",
            type = 'scatter',
            mode = 'lines+markers') %>%
    layout(xaxis = list(title = "Game Number in Season"),
           yaxis = list(title = names(select_stat_choices[select_stat_choices==input$select_stat]),
                        range = limit_data()))

})
```

Publishing the application

Shiny apps are easy to share and the free version of shinyapps.io was used for this project.

A more comprehensive outline of ways to share shiny apps is given here.