"Creating an Interactive Statistical Scouting Dashboard with R: A Step-by-Step Guide"

Learning new skills through the 2024 League of Ireland Premier Division Season

Lorcán Mason

Table of contents

Introduction

More Resources Available My Website

This tutorial will focus on creating player profiles for the 2024 League of Ireland Premier Division season. The data used in this tutorial is from the 2024 season and is more information on the league is available on the League of Ireland website. The data used in this tutorial is for ***illustrative purposes only***, there are people far smarter than myself who can do this much better - I just document what I learn!

In this article, we'll create an interactive sports dashboard using R and Shiny. If you've ever been curious about how to transform raw data into something that's not only informative but also genuinely engaging, this is for you. I've kept it quite basic as I'm still learning myself, but I hope it's helpful to you!

Our aim is to walk through every aspect of the process, from the initial steps of setting up the environment to crafting dynamic visualizations. The code will be explained in a clear and detailed manner, discussing the choices made along the way, and exploring various ways to apply these techniques to a range of different scenarios.

This guide is designed with the absolute beginner (myself!) in mind, ***so prior coding or data analysis experience is not required***. We will take things slow, providing clear explanations for each step. By the end of this post, you'll have a practical understanding of how to build an interactive dashboard and the skills to adapt it to various use cases.

Our example will focus on visualizing player statistics from the 2024 League of Ireland Season. However, the methods learned here can be used for any type of data, such as tracking performance testing data, creating visualizations for GPS outputs, or even compiling detailed player scouting reports.

Hope, this helps!

Setting Up Your Environment

Before we begin any form of coding, it's essential to ensure that we have the necessary tools installed on our machine. This will provide a solid foundation for the work that follows.

1. **Install R:**

- If you haven't already, the first step is to download and install R, which is the programming language at the heart of our dashboard. It's completely free and can be downloaded from the official website: https://www.r-project.org/. Please ensure that you select the correct installer for your operating system (Windows, macOS, or Linux).

2. **Install RStudio (Optional but Recommended):**

   - While R is essential, we highly recommend using RStudio, which is an Integrated Development Environment (IDE) that significantly simplifies the R coding process. It enhances readability, offers helpful tips, and generally makes life easier. You can download the free RStudio Desktop version from their website: https://www.rstudio.com/.

3. **Install Required Packages:**

   - Once you've successfully installed R and RStudio (if you chose to), please open RStudio. Locate the console panel (typically in the lower left), which is where we will type and execute R commands. Please type and execute each command, one at a time as follows:

```
install.packages("shiny")
install.packages("tidyverse")
install.packages("plotly")
install.packages("DT")
```

   - `shiny`: This package provides the core functionality needed to transform your R code into an interactive web application. For further details, please visit the official Shiny website.
   - `tidyverse`: This is a suite of R packages designed to facilitate data manipulation and visualization. It provides functions like `read.csv`, `rename`, `mutate` from the `dplyr` package, and charting functions from `ggplot2` and all under a cohesive framework. You can further explore the capabilities at the Tidyverse website.
   - `plotly`: This package is used to create interactive graphs with zoom, hover, and other interactive features. Visit Plotly's website to see a full set of its capabilities.
   - `DT`: This package creates interactive, sortable data tables within your dashboard. You can find the full documentation of the package at DT documentation.

4. **Create a New R Script:**

- Now that all the tools and packages are ready to go, let's begin by creating a new R script in RStudio. Navigate to "File," then "New File," and then select "R Script." You can then save this file with an appropriate name, such as `loi_dashboard.R`.

The R Code Explained Step-by-Step

Now that we have completed the setup phase, we can now examine the R script and its individual components in more detail.

1. Loading Libraries

```
# Load Libraries
library(shiny)
library(tidyverse)
library(plotly)
library(DT)
```

- **Explanation:** This initial section is where we tell R which packages we intend to use by calling the `library()` function on the packages we previously installed.
  - `library(shiny)`: By using this line, we load the `shiny` package which allows us to use all the UI controls and functionality to make our app interactive. You can explore the full details on the Shiny website.
  - `library(tidyverse)`: This command loads the `tidyverse` package, which is essential for a wide range of tasks such as data cleaning, transformation, and visualization. Further exploration of the `tidyverse` can be found here.
  - `library(plotly)`: The command loads the `plotly` library, which transforms static `ggplot2` charts into interactive visualizations. You can learn more at the Plotly website.
  - `library(DT)`: Lastly, we load `DT` which provides the tools to create interactive HTML data tables within our Shiny app. More info on the DT documentation.

2. Loading and Cleaning the Data

```
# Load the Data
player_data <- read.csv("all_player_loi.csv")

# Data Cleaning
player_data <- player_data %>%
  rename(
    "Player Name" = "Player",
    "Minutes Played" = "Minutes.Played",
    "Accurate Long Balls per 90" = "Accurate.Long.Balls.per.90",
```

```
    "Accurate Pass per 90" = "Accurate.Pass.per.90",
    "Big Chances Missed" = "Big.Chances.Missed",
    "Big Chances Created" = "Big.Chances.Created",
    "Blocks per 90" = "Blocks.per.90",
    "Chances Created" = "Chances.Created",
    "Clean Sheets" = "Clean.Sheets",
    "Clearances per 90" = "Clearances.per.90",
    "Player Rating" = "Player.Rating",
    "Fouls per 90" = "Fouls.per.90",
    "Goals and Assists" = "Goals.and.Assists",
    "Goals Conceded per 90" = "Goals.Conceded.per.90",
    "Goals per 90" = "Goals.per.90",
    "Interceptions per 90" = "Interceptions.per.90",
    "Penalties Conceded" = "Penalties.Conceded",
    "Penalties Won" = "Penalties.Won",
    "Possession Won Att 3rd per 90" = "Possession.Won.Att.3rd.per.90",
    "Red Cards" = "Red.Cards",
    "Save Percentage" = "Save.Percentage",
    "Saves per 90" = "Saves.per.90",
    "Shots on Target per 90" = "Shots.on.Target.per.90",
    "Shots per 90" = "Shots.per.90",
    "Successful Dribbles per 90" = "Successful.Dribbles.per.90",
    "Successful Tackles per 90" = "Successful.Tackles.per.90",
    "Yellow Cards" = "Yellow.Cards",
    "Fouls Committed per 90" = "Fouls.per.90",
    "Possession Won Attacking 3rd per 90" = "Possession.Won.Att.3rd.per.90"
  )

# Calculate Goal Contributions
player_data <- player_data %>%
  mutate(goal_contributions = `Goals` + `Assists`)


teams <- sort(unique(player_data$Team))
```

- **Explanation:**
  - `player_data <- read.csv("all_player_loi.csv")`: This line is where the data is brought into R. The `read.csv()` function is called from the `utils` package. It imports our data from a CSV file named `all_player_loi.csv`, which needs to be in the same directory as your R script. You can read the full documentation on the `read.csv` function in R here.

- `player_data <- player_data %>% rename(...)`: This is the first of a few lines where we clean up and organise the loaded data. The `rename()` function, which comes with the `dplyr` package, is used here to make column names more readable. For example, the original column name "Player" is renamed to "Player Name". More information on the `rename` function can be found here.
- `player_data <- player_data %>% mutate(goal_contributions = Goals + Assists)`: Here, we generate a new column that combines data from two others, using the `mutate` function from `dplyr`. We create the `goal_contributions` column by adding the numerical values from both the `Goals` and `Assists` columns. You can learn more about the `mutate` function here.
- `teams <- sort(unique(player_data$Team))`: This line creates a sorted list of all unique teams within the data. The `unique` function extracts all the unique values from the `Team` column, which are then sorted by `sort`.

3. Creating UI Elements and Server Logic Functions

```
# Function to create a filtered bar chart and table UI
create_stats_ui <- function(id, variable_name, label) {
  ns <- NS(id)
  tagList(
    fluidRow(
      column(12,
             h4(label),
             selectInput(ns("team_select"), "Select Team", choices = c("All
              ↪  Teams", teams), selected = "All Teams"),
             plotlyOutput(ns("bar_chart")),
             dataTableOutput(ns("player_table"))
      )
    )
  )
}


# Function to create the server logic
create_stats_server <- function(id, variable_name, is_goalkeeping = FALSE) {
  moduleServer(
    id,
    function(input, output, session) {

      filtered_data <- reactive({
        if(input$team_select == "All Teams"){
```

```
        player_data
      } else {
        player_data %>%
          filter(Team == input$team_select)
      }
    })


    output$bar_chart <- renderPlotly({

       if (is_goalkeeping) {
            top_players <- filtered_data() %>%
             filter(`Clean Sheets` > 0 | `Goals Conceded per 90` > 0 |
              ↪  `Save Percentage` > 0 | `Saves per 90` >0) %>%
            arrange(desc(!!sym(variable_name))) %>%
              head(10)
         } else {
            top_players <- filtered_data() %>%
                arrange(desc(!!sym(variable_name))) %>%
                 head(20)
          }

      avg_value <- filtered_data() %>%
        summarize(avg = mean(!!sym(variable_name), na.rm = TRUE)) %>%
        pull(avg)

      plot <- ggplot(top_players, aes(x = reorder(`Player Name`,
↪  !!sym(variable_name)), y = !!sym(variable_name), fill = Team,text=`Player
↪  Name`)) +
        geom_bar(stat = "identity") +
        geom_hline(aes(yintercept = avg_value, text = paste("Average:",
         ↪  round(avg_value, 2))), color = "red", linetype="dashed") +
        coord_flip() +
         geom_label(aes(y = avg_value, label = paste("Average:",
          ↪  round(avg_value,2))),
                    hjust = 0, vjust = -0.5, fill = "white", color =
                     ↪  "black") +
        labs(y = NULL) +
        theme_minimal()

      ggplotly(plot, tooltip = c("y", "text"))
```

```
      })

      output$player_table <- renderDataTable({
          if(is_goalkeeping){
                 top_players <- filtered_data() %>%
                 filter(`Clean Sheets` > 0 | `Goals Conceded per 90` > 0 |
                  ↪  `Save Percentage` > 0 | `Saves per 90` >0) %>%
                  arrange(desc(!!sym(variable_name))) %>%
                 select(`Player Name`, Team, !!sym(variable_name))
          } else {
                top_players <- filtered_data() %>%
                  arrange(desc(!!sym(variable_name))) %>%
                  select(`Player Name`, Team, !!sym(variable_name))
          }

       datatable(top_players, options = list(order = list(3, 'desc'),
                                             pageLength = 5,
                                             lengthMenu = c(5,10, 15)
       ))
      })

    }
  )
}
```

- **Explanation:**
  - `create_stats_ui`: This function encapsulates a UI block which contains a header, team selection dropdown, an area to display the bar chart and a data table.
    - `ns <- NS(id)`: This line creates a namespace, and this is used to ensure unique IDs for each element created inside a module to avoid namespace clashes by using the `NS` function. You can learn more about the `NS` function and modularity here.
    - `selectInput(ns("team_select"), "Select Team", choices = c("All Teams", teams), selected = "All Teams")`: This generates a dropdown menu for selecting a team. The `selectInput` function can be explored further in the Shiny UI documentation here.
    - `plotlyOutput(ns("bar_chart"))`: This line creates a placeholder where the bar chart, made with `plotly`, will be displayed. The function can be seen in the documentation here.
    - `dataTableOutput(ns("player_table"))`: This is a placeholder for the interactive table made using the `DT` package. You can see further info on

`dataTableOutput` here.

- `create_stats_server`: This function contains all the server side logic for each UI block. This function also contains:
  - `moduleServer(id, function(input, output, session) { ... })`: Using the `moduleServer` function makes the logic defined here reusable in other parts of the application. Details on this function can be found here.
  - `filtered_data <- reactive({...})`: This creates a *reactive expression* which means it automatically updates when the input changes. It uses the `reactive` function which takes the current selected team in the `selectInput` and filters the dataset to only include the selected team, or all teams if the "All Teams" option is selected.
  - `output$bar_chart <- renderPlotly({...})`: Inside this, we are defining how to create and render an interactive bar chart using the `renderPlotly` function which uses `ggplot2` and `plotly`. Firstly the top 20 players are selected for the given variable, and if on the goalkeeping tab, the top 10 goalkeepers with data are selected. An average is calculated for all values in the given column and displayed as a horizontal line with the text of the average. The full set of `ggplot2` components are described here. And, the documentation for `plotly` is here.
  - `output$player_table <- renderDataTable({...})`: Here we are constructing an interactive table, rendered using the `datatable` function from the `DT` package. Players are sorted by a given metric and the users can set the number of rows to display on each page, as specified in the `options` parameter. Details of `datatable` can be found here.

4. Defining the UI

```
# Define UI
ui <- fluidPage(
  titlePanel(
    HTML(
      '<div style="display: flex; align-items: center;">
          <img
 ↪  src="https://www.sligorovers.com/wp-content/uploads/2018/03/221009_SSE-LOI-MENS-PREM_LOG
 ↪  style="height: 80px; margin-right: 10px;">
          <span style="font-size: 24px; font-weight: bold;">League of Ireland
 ↪  Player Statistics</span>
        </div>'
    )
  ),
```

```
  tabsetPanel(
    tabPanel("Attacking",
             create_stats_ui("attacking_goals", "Goals", "Goals"),
             create_stats_ui("attacking_goals_per_90", "Goals per 90", "Goals
              ↪  per 90"),
             create_stats_ui("attacking_goals_and_assists", "Goals and
              ↪  Assists", "Goals and Assists"),
             create_stats_ui("attacking_big_chances_missed", "Big Chances
              ↪  Missed", "Big Chances Missed"),
             create_stats_ui("attacking_shots_on_target_per_90", "Shots on
              ↪  Target per 90", "Shots on Target per 90"),
             create_stats_ui("attacking_shots_per_90", "Shots per 90", "Shots
              ↪  per 90")
    ),
    tabPanel("Creativity",
             create_stats_ui("creativity_assists", "Assists", "Assists"),
             create_stats_ui("creativity_accurate_long_balls_per_90",
              ↪  "Accurate Long Balls per 90", "Accurate Long Balls per 90"),
             create_stats_ui("creativity_accurate_passes_per_90", "Accurate
              ↪  Pass per 90", "Accurate Pass per 90"),
             create_stats_ui("creativity_big_chances_created", "Big Chances
              ↪  Created", "Big Chances Created"),
             create_stats_ui("creativity_chances_created", "Chances Created",
              ↪  "Chances Created"),
             create_stats_ui("creativity_successful_dribbles_per_90",
              ↪  "Successful Dribbles per 90", "Successful Dribbles per 90"),
             create_stats_ui("creativity_penalties_won", "Penalties Won",
              ↪  "Penalties Won")
    ),
    tabPanel("Defending",
             create_stats_ui("defending_interceptions_per_90", "Interceptions
              ↪  per 90", "Interceptions per 90"),
             create_stats_ui("defending_successful_tackles_per_90",
              ↪  "Successful Tackles per 90", "Successful Tackles per 90"),
             create_stats_ui("defending_blocks_per_90", "Blocks per 90",
              ↪  "Blocks per 90"),
             create_stats_ui("defending_clearances_per_90", "Clearances per
              ↪  90", "Clearances per 90"),
             create_stats_ui("defending_possession_won_attacking_3rd_per_90",
              ↪  "Possession Won Attacking 3rd per 90", "Possession Won
              ↪  Attacking 3rd per 90")
    ),
```

```
    tabPanel("Discipline",
             create_stats_ui("discipline_yellow_cards", "Yellow Cards",
              ↪  "Yellow Cards"),
             create_stats_ui("discipline_red_cards", "Red Cards", "Red
              ↪  Cards"),
             create_stats_ui("discipline_penalties_conceded", "Penalties
              ↪  Conceded", "Penalties Conceded"),
             create_stats_ui("discipline_fouls_committed_per_90", "Fouls
              ↪  Committed per 90", "Fouls Committed per 90")
    ),
        tabPanel("Goalkeeping",
             create_stats_ui("goalkeeping_clean_sheets", "Clean Sheets",
              ↪  "Clean Sheets"),
             create_stats_ui("goalkeeping_goals_conceded_per_90", "Goals
              ↪  Conceded per 90", "Goals Conceded per 90"),
             create_stats_ui("goalkeeping_save_percentage", "Save
              ↪  Percentage", "Save Percentage"),
               create_stats_ui("goalkeeping_saves_per_90", "Saves per 90",
                ↪  "Saves per 90")
    )
  )
)
```

- **Explanation:**
  - `ui <- fluidPage(...)`: This sets up the main user interface using the `fluidPage` function from shiny, which is designed to be responsive and work well on different screen sizes. You can see more info on the `fluidPage` function here.
  - `titlePanel(...)`: This adds a title at the top of the page, using the `titlePanel` function. We have used `HTML()` within this function to embed a logo along with a text title for the dashboard. More information on how to use this to create more customized titles can be found here.
- `tabsetPanel(...)`: This sets up the tab based navigation, enabling users to interact with the data across different categories using the `tabsetPanel` function. Details on this function can be found here.
  - Inside the `tabsetPanel` we can define multiple `tabPanel` elements, and the `create_stats_ui` is called multiple times to build the required elements for each tab. The `tabPanel` function is described in the documentation here.

5. Defining the Server Logic

```
# Define server
server <- function(input, output, session) {

  # Server logic for each tab
  create_stats_server("attacking_goals", "Goals")
  create_stats_server("attacking_goals_per_90", "Goals per 90")
  create_stats_server("attacking_goals_and_assists", "Goals and Assists")
  create_stats_server("attacking_big_chances_missed", "Big Chances Missed")
  create_stats_server("attacking_shots_on_target_per_90", "Shots on Target
   ↪  per 90")
  create_stats_server("attacking_shots_per_90", "Shots per 90")

  create_stats_server("creativity_assists", "Assists")
  create_stats_server("creativity_accurate_long_balls_per_90", "Accurate Long
   ↪  Balls per 90")
  create_stats_server("creativity_accurate_passes_per_90", "Accurate Pass per
   ↪  90")
  create_stats_server("creativity_big_chances_created", "Big Chances
   ↪  Created")
  create_stats_server("creativity_chances_created", "Chances Created")
  create_stats_server("creativity_successful_dribbles_per_90", "Successful
   ↪  Dribbles per 90")
  create_stats_server("creativity_penalties_won", "Penalties Won")

  create_stats_server("defending_interceptions_per_90", "Interceptions per
   ↪  90")
  create_stats_server("defending_successful_tackles_per_90", "Successful
   ↪  Tackles per 90")
  create_stats_server("defending_blocks_per_90", "Blocks per 90")
  create_stats_server("defending_clearances_per_90", "Clearances per 90")
  create_stats_server("defending_possession_won_attacking_3rd_per_90",
   ↪  "Possession Won Attacking 3rd per 90")

  create_stats_server("discipline_yellow_cards", "Yellow Cards")
  create_stats_server("discipline_red_cards", "Red Cards")
  create_stats_server("discipline_penalties_conceded", "Penalties Conceded")
  create_stats_server("discipline_fouls_committed_per_90", "Fouls Committed
   ↪  per 90")
```

```
  create_stats_server("goalkeeping_clean_sheets", "Clean Sheets",
   ↪  is_goalkeeping = TRUE)
  create_stats_server("goalkeeping_goals_conceded_per_90", "Goals Conceded
   ↪  per 90", is_goalkeeping = TRUE)
  create_stats_server("goalkeeping_save_percentage", "Save Percentage",
   ↪  is_goalkeeping = TRUE)
  create_stats_server("goalkeeping_saves_per_90", "Saves per 90",
   ↪  is_goalkeeping = TRUE)
}
```

- **Explanation:**
  - `server <- function(input, output, session) { ... }`: This defines the server-side logic of the application. We are now linking up the UI elements with their corresponding functionality, by calling the `create_stats_server` for each metric to make each graph and table in the UI dynamic.
  - The `create_stats_server` calls are made in this section, with the goalkeeping tab getting the parameter `is_goalkeeping = TRUE` passed to ensure only goalkeepers are shown on the visualizations.

6. Running the Shiny App

```
# Run the app
shinyApp(ui, server)
```

- **Explanation:** This line is like the final step - when everything comes together and the application is launched. It calls the `shinyApp(ui, server)` function, which takes the UI and server definitions and launches the Shiny application in your browser. More details on how to use the `shinyApp` function is available here.

Alternative Use Cases

Now, let's explore how you can adapt this dashboard to different contexts:

1. Athlete Monitoring

- **Data:** Instead of soccer stats, you might have data from wearable sensors like heart rate monitors, GPS trackers, and accelerometers.

- **Metrics:** Track metrics such as:
  - Distance covered
  - Top speed
  - Heart rate variability
  - Acceleration and deceleration
  - Player load
- **Visualization:**
  - Time-series plots showing trends over time.
    * Time Series Data with ggplot2: Using the `geom_line` is a great way to plot time series data in `ggplot2`, and it helps us visualize how data changes over time.
  - Heatmaps showing the intensity of activity on a field.
    * Heatmaps with ggplot2: Heatmaps are great at visualising the intensity of activity over a given area.
  - Individual player profiles with personalized metrics.

2. GPS Dashboard

- **Data:** GPS data providing location, speed, and distance.
- **Metrics:**
  - Distance covered in various zones of a field.
  - Speed in different directions.
  - Heatmaps showing coverage and density of player movement.
- **Visualization:**
  - Geographical maps to visualize player movement during a match.
    * Mapping with R: In this guide, learn how to use R for plotting mapping data.
  - Animation of player positions over time.
    * Animation with gganimate: The `gganimate` package allows you to animate your ggplot2 charts.
  - Individual speed profiles.

3. Player Scouting Reports

- **Data:** Data for a large database of players with multiple attributes.
- **Metrics:** Combine performance and physical metrics to understand the potential of a player

- **Visualization:**
  - Radar charts to visualize the attributes of a player compared to other players in a particular league.
    - Radar Charts in R: If you want to learn more about radar charts, this R graph gallery page details how to create this using R.
  - Box plots to visualize the distribution of different stats for a group of players
    - Box Plots in R: Box plots are a great way to show the distribution of data.
  - Player comparison charts where the attributes of 2 different players can be compared side by side.
  - Include video clips or game footage, if accessible.

Adapting the Code

To adapt the current code, you can make the following changes:

1. **Data Loading:** The `read.csv()` call is specifically used to load CSV data. Depending on the format of your data, this line may need modification. There are many different methods available for importing different file types in R.
2. **Data Cleaning:** You'll likely need to adjust the `rename()` and `mutate()` functions to match the column names and transformations required by your data.
3. **Visualization:** The use of `ggplot2` means that we can create a range of different charts based on our needs. You might want to research the correct `geom_` layer to be used to visualize your data depending on what you want to show in the chart.
4. **UI elements:** The Shiny package contains a range of different UI elements that can be used to create interactive controls for your app. Please explore the documentation here.
5. **Server logic:** Reactive expressions are at the core of dynamic content within Shiny apps, and they make it easier to create interactive apps. Further information on reactive expressions can be found here.

Conclusion: Have Fun

That's that! You've successfully taken your first steps into building interactive data visualizations with R and Shiny. Now you should have a good grasp of the main R code blocks, the Shiny package and its functionality, how to visualise your data using the `ggplot2` package and how to make it all interactive using `plotly` and `DT`. I hope you've enjoyed this tutorial and that it has provided some value to you. Remember, the best way to get better is by building more dashboards, and you can use the code that we have created as a starting point.

I've no doubt that this can be done more efficiently than I have done it but this is what I have learned so far. I hope this helps in some way and if you are a more experienced R user, I would

love to hear your feedback on how I can improve this process - feel free to reach out to me on Twitter or LinkedIn. To learn from the experts, please check out TidyX by Patrick Ward and Ellis Hughes on Youtube. Mladen Jovanović also has fantastic courses on his website Complementary Training.

Stay Curious!