Aplicaciones y Desarrollos en R

DESARROLLO DE UNA APLICACIÓN INTERACTIVA USANDO *SHINY*

MÁSTER DE FORMACIÓN PERMANENTE EN *BIG DATA* DEPORTIVO

Xavier Rosinach Capell

Universidad Europea - Escuela Real Madrid

La siguiente actividad se basa en el desarrollo de una aplicación interactiva en R utilizando *Shiny*, cuyo objetivo es visualizar y analizar datos de fútbol. En primer lugar, se presenta una breve introducción sobre qué es *Shiny* y cuáles son sus principales aplicaciones en el ámbito del análisis deportivo, especialmente en el fútbol. A continuación, se describe el funcionamiento de la aplicación, detallando su estructura y codificación. Finalmente, se analizan los principales desafíos técnicos identificados durante el uso de *Shiny* y las soluciones adoptadas para abordarlos.

1 Desarrollo de aplicaciones web usando *Shiny*

La librería de R *Shiny* es un *framework* que permite crear aplicaciones web interactivas de forma sencilla. Está especialmente diseñada para analistas y científicos de datos, ya que elimina la necesidad de contar con conocimientos avanzados en desarrollo web.

En *Shiny* se distinguen dos componentes principales:

- **UI (*User Interface*)**: la UI define la interfaz visual de la aplicación, es decir, cómo se presenta y cómo los usuarios interactúan con ella. Este componente organiza los distintos elementos visuales, como cajas de texto, gráficos, tablas o botones.

```
ui <- page_fluid(
  p("Esto es una app para saludar"),
  textInput("nombre", "Escribe tu nombre:"),
  textOutput("saludo"))
```

Ejemplo de código 1: Ejemplo básico de UI.

- ***Server* (*Backend*)**: el *server* contiene la lógica de la aplicación. Es el encargado de responder a las interacciones del usuario en la UI y de procesar los datos de *input* y generar los correspondientes *output* que se muestran en la aplicación.

```
server <- function(input, output) {
    output$saludo <- renderText({
        paste("Hola,", input$nombre)})}
```

Ejemplo de código 2: Ejemplo básico de *server*.

De este modo, la estructura básica de una aplicación desarrollada con *Shiny* se compone de la definición de la UI, la lógica del *server* y la conexión entre ambos mediante la función *shinyApp(ui, server)*.

1.1 Aplicaciones de *Shiny* en el análisis deportivo (enfocado al fútbol)

En el ámbito del análisis deportivo, *Shiny* se ha consolidado como una herramienta clave para la exploración, visualización y comunicación de datos. Su principal valor reside en la capacidad de transformar análisis complejos en aplicaciones interactivas accesibles tanto para analistas como para cuerpos técnicos, directivos o jugadores, sin necesidad de que estos tengan conocimientos técnicos en programación.

Una de las aplicaciones más habituales de *Shiny* en fútbol es la creación de *dashboards* interactivos para el análisis del rendimiento. Estas aplicaciones permiten visualizar estadísticas de equipos y jugadores, y filtrarlas dinámicamente por temporada, competición, jornada, rival o jugador concreto. De este modo, el analista puede explorar patrones de rendimiento de forma rápida, adaptando el análisis a distintas cuestiones tácticas o estratégicas.

Shiny también resulta especialmente útil en el ámbito del *scouting* y la detección de talento. A través de aplicaciones interactivas, los analistas pueden filtrar bases de datos de jugadores por posición, edad, liga o métricas de rendimiento, y generar comparativas visuales entre distintos perfiles. Este enfoque permite identificar jugadores que encajan en un determinado modelo de juego o que destacan en variables específicas, agilizando notablemente los procesos de búsqueda y evaluación.

En conjunto, el uso de *Shiny* en el análisis futbolístico aporta una clara ventaja: permite conectar el trabajo técnico del analista con los usuarios finales de la información, ofreciendo soluciones visuales, dinámicas y adaptables que mejoran la interpretación de los datos y apoyan la toma de decisiones en el entorno profesional del fútbol.

2 Estructura de la aplicación

El producto final es una aplicación interactiva que permite visualizar y analizar datos de fútbol. La aplicación integra las siguientes funcionalidades principales:

- **Carga de datos**: es posible cargar los datos a partir de un archivo CSV (con la primera fila como *header*, valores separados por ;' y codificación UTF-8'). Los datos deben contener información sobre equipos o jugadores de fútbol. En la carpeta `example-data` se incluyen los archivos `players.csv` y `teams.csv` como ejemplos.
- **Interfaz interactiva para la visualización de los datos**: los datos cargados pueden visualizarse en formato de tabla y permiten operaciones de filtrado, ordenación y exportación a formato CSV.
- **Creación de gráficos**: a partir de la tabla cargada, ordenada y filtrada, el usuario puede generar distintos tipos de gráficos para representar la información contenida en los datos. Estos gráficos pueden personalizarse mediante títulos, colores, leyendas y escalas.
- **Pestaña de información**: la aplicación incorpora un pequeño *pop-up* en el que se explica brevemente su funcionamiento.

A partir de estas características, se describe a continuación el diseño e implementación de la aplicación, así como los puntos clave que han permitido que la codificación funcione correctamente.

2.1 Codificación de la aplicación

La codificación principal de la aplicación se encuentra en el archivo `app.R`, donde se define su estructura general. Para mejorar la claridad, la modularidad y el mantenimiento del código, las distintas funcionalidades se han dividido en varios archivos independientes, cada uno responsable de una parte concreta de la aplicación. Estos archivos son los siguientes:

2.1.1 Carga de datos

El proceso de carga de datos se implementa en el archivo `data_loader.R`. Este módulo gestiona tanto la carga como la validación del conjunto de datos. El flujo de trabajo se basa en la entrada de un archivo CSV mediante *fileInput()*, su validación y un botón para confirmar el uso de los datos una vez cargados. Consulta la Figura 1 para observar la página de carga de datos.

1. Se permite la carga de un archivo CSV y se comprueba que cumple los requisitos establecidos: formato CSV, primera fila como *header*, valores separados por ';' y codificación UTF-8'. Estas restricciones se imponen para minimizar posibles errores posteriores en el tratamiento de los datos.

2. Si la carga es correcta, se verifica que el archivo no sea nulo y que contenga al menos una columna.
3. Una vez validado el archivo, se procede a su lectura. Aunque en condiciones normales no deberían producirse errores en este paso, se captura y muestra un mensaje de error en caso de que ocurra alguno.
4. Si los datos se leen correctamente, se muestra un mensaje informativo y se devuelven en formato *data frame* para su uso en los módulos posteriores.
5. Una vez validados los datos, si el usuario decide usarlos, va a entrar a la página de visualización de datos. Se puede observar su estética en la Figura 2.

2.1.2 Visualización de los datos en una tabla

La funcionalidad encargada de mostrar los datos cargados se implementa en `data_view.R`. Este módulo permite al usuario visualizar el conjunto de datos en forma de tabla, aplicar operaciones de ordenación y filtrado, exportar los datos resultantes y reiniciar la carga para utilizar un nuevo archivo CSV. En la Figura 3 se puede observar lo que el usuario va a ver.

1. Se utiliza como *input* el *data frame* correspondiente al CSV cargado previamente en `data_loader.R`.
2. Se implementa un desplegable que permite ordenar los datos. El usuario puede seleccionar una de las columnas del *data frame* y definir si la ordenación debe realizarse de forma ascendente o descendente. Cuando se selecciona una opción válida, se aplica el orden correspondiente. Se puede observar su forma en Figura 4.
3. Adicionalmente, se incorpora un sistema de filtros por columnas. Para las columnas numéricas, se habilitan dos campos de entrada (valor mínimo y valor máximo), inicializados con los valores actuales del conjunto de datos. Para las columnas no numéricas, se incluye un filtro de texto del tipo “la columna contiene”. Todos los filtros seleccionados se aplican de forma conjunta. Podemos fijarnos en Figura 5 para ver cómo se puede usar.
4. Cuando el usuario realiza modificaciones sobre los datos, se activa un botón que permite confirmar el uso del conjunto de datos modificado para la generación de gráficos.
5. El usuario dispone también de la opción de guardar el conjunto de datos resultante en formato CSV mediante *downloadHandler()*.
6. Finalmente, se ofrece la posibilidad de cambiar el archivo CSV inicial. Esta acción reinicia la aplicación y elimina todos los cambios realizados, mostrando previamente un mensaje de advertencia.
7. Como *output*, se devuelve el *data frame* original, un indicador booleano que refleja si se han realizado y aplicado cambios, y el *data frame* final con las transformaciones aplicadas.

2.1.3 Creación de gráficos

La parte central de la aplicación es la encargada de la creación de gráficos, cuyo objetivo es facilitar la identificación de métricas y tendencias relevantes en jugadores o equipos. Esta funcionalidad se implementa en el archivo `charts_view.R`. El módulo utiliza el *data frame* previamente filtrado u ordenado y permite generar cuatro tipos de gráficos: barras, líneas, dispersión y gráfico circular. Asimismo, ofrece diversas opciones de personalización. En la Figura 6 podemos observar lo que el usuario va a poder ver.

1. Se utiliza como *input* el *data frame* inicial o modificado, según el estado del módulo anterior. Previamente, se identifican las columnas numéricas y no numéricas, ya que su tipo condiciona las variables que pueden asignarse a los distintos ejes del gráfico.

2. Se presenta un desplegable que permite seleccionar uno de los cuatro tipos de gráficos disponibles. Una vez seleccionado, el usuario debe definir las variables correspondientes a cada eje o componente del gráfico:
   - **Gráfico de barras**: permite seleccionar una variable categórica para el eje X, una medida de agregación (suma, media o conteo) y una variable para el eje Y. Ejemplo de *output* en Figura 8.
   - **Gráfico de líneas**: permite seleccionar cualquier variable para el eje X y una variable numérica para el eje Y. Ejemplo de *output* en Figura 9.
   - **Gráfico de dispersión**: tanto el eje X como el eje Y deben corresponder a variables numéricas. Ejemplo de *output* en Figura 10.
   - **Gráfico circular**: similar al gráfico de barras, aunque en este caso no se definen ejes, sino una variable categórica y un valor asociado. Ejemplo de *output* en Figura 11.
3. Una vez seleccionados el tipo de gráfico y las variables correspondientes, el gráfico se genera utilizando la librería *ggplot2*.
4. A continuación, el usuario puede personalizar el gráfico antes de su exportación:
   - **Personalización de títulos**: se permite modificar el título del gráfico y los nombres de los ejes. En el caso del gráfico circular, únicamente se define el título principal.
   - **Personalización de colores**: el gráfico puede personalizarse mediante el uso de paletas de colores predefinidas (obtenidas con *grDevices::hcl.colors(n, pal)*, que generan paletas de *n* colores) o mediante la introducción manual de colores en formato hexadecimal (por ejemplo, #4C8DFF), separados por comas.
   - **Escalas**: el usuario puede limitar el rango de los ejes mediante la definición de valores mínimos y máximos.
5. Tras aplicar los cambios, el gráfico se actualiza automáticamente y puede exportarse en formato PNG.
6. Este módulo no devuelve ningún *output* adicional, ya que corresponde a la etapa final de la aplicación.

2.1.4 Otros componentes

Además de los módulos descritos, se ha desarrollado el archivo `instructions.R`, que contiene el contenido mostrado en la sección de instrucciones de la aplicación. Podemos ver las instrucciones en Figura 7. Asimismo, el estilo visual y la estética general de la aplicación web se definen en `style.R`, donde se especifican aspectos como el tipo y tamaño de letra de títulos y párrafos, los colores, el fondo de la página y el diseño de los botones.

3 Desafíos y soluciones en la codificación

El desarrollo de aplicaciones con *Shiny* presenta varios retos técnicos, especialmente cuando se busca construir una herramienta robusta y potente. Durante la codificación de la aplicación, se identificaron los siguientes desafíos y se adoptaron las siguientes estrategias.

3.1 Rendimiento y escalabilidad con grandes volúmenes de datos

Uno de los principales problemas en aplicaciones de análisis es el rendimiento al trabajar con tablas grandes. En *Shiny*, la renderización de tablas completas y la re-ejecución de cálculos reactivos en cada cambio de *input* puede producir latencias perceptibles y una experiencia de usuario pobre. En esta aplicación, se evitó mostrar el *data frame* completo y se implementó:

- **Paginación manual**: Se muestra un subconjunto de filas mediante un sistema de páginas y flechas, reduciendo la carga de renderizado al solo fijarnos en la página actual.

- **Confirmación explícita del *data frame* modificado**: El botón “Usar datos” permite que el *data frame* para gráficos se actualice únicamente cuando el usuario confirma los cambios, evitando recalcular gráficos continuamente mientras se ajustan filtros.

3.2 Gestión de reactividad y control del flujo de la aplicación

Un reto habitual en *Shiny* es controlar cuándo deben ejecutarse cálculos reactivos. Si no se gestiona correctamente, la aplicación puede recalcular y renderizar componentes con demasiada frecuencia, generando errores intermitentes - por ejemplo, intentar graficar cuando todavía no existen columnas seleccionadas. En la codificación, se usaron esta técnicas:

- **Validación de la existencia de los datos**: Para garantizar que los datos existen y que el usuario ha seleccionado los *input*s mínimos antes de renderizar tabla o gráficos.
- **Separación de estados del *data frame***: se distingue entre *data frame* cargado y *data frame* “confirmado” por el usuario.
- **Introducción de un paso de confirmación**: Usamos el botón de “Usar estos datos” para evitar una actualización constante de los gráficos.

3.3 Validación de datos

Al permitir la carga de archivos externos, es frecuente que el usuario utilice CSVs con separadores, codificaciones o cabeceras inconsistentes. Esto puede romper la lectura de datos o producir errores posteriores en filtros/gráficos.

El módulo `data_loader.R` restringe explícitamente el formato esperado (CSV con separador “;”, cabecera y codificación UTF-8) y captura errores con *tryCatch()*, mostrando los errores (si hay) claros en la interfaz. Además, se añade un botón de confirmación para evitar que el resto de la aplicación se habilite sin una carga correcta.

3.4 Visualización flexible y personalización de gráficos

Otro reto consiste en permitir gráficos flexibles sin forzar al usuario a comprender detalles técnicos como el tipo de variable, escalas o configuraciones estéticas. Por ejemplo, no todos los gráficos aceptan ejes numéricos en ambos lados (gráfico de barras con X categórica) y una mala configuración puede provocar errores.

Se construyeron selectores que diferencian variables numéricas y categóricas, limitando la elección según el tipo de gráfico. Los gráficos se implementan con *ggplot*, ya que es un estándar en R para visualización y permite personalización modular (títulos, colores, leyendas y escalas). Además, se exige una selección mínima (X, Y, y medida) antes de mostrar cualquier gráfico.

4 Anexo

4.1 Imágenes

4.1.1 Capturas de la aplicación web

Figura 1: Carga de datos en formato CSV.

Figura 2: Página principal de la aplicación web.

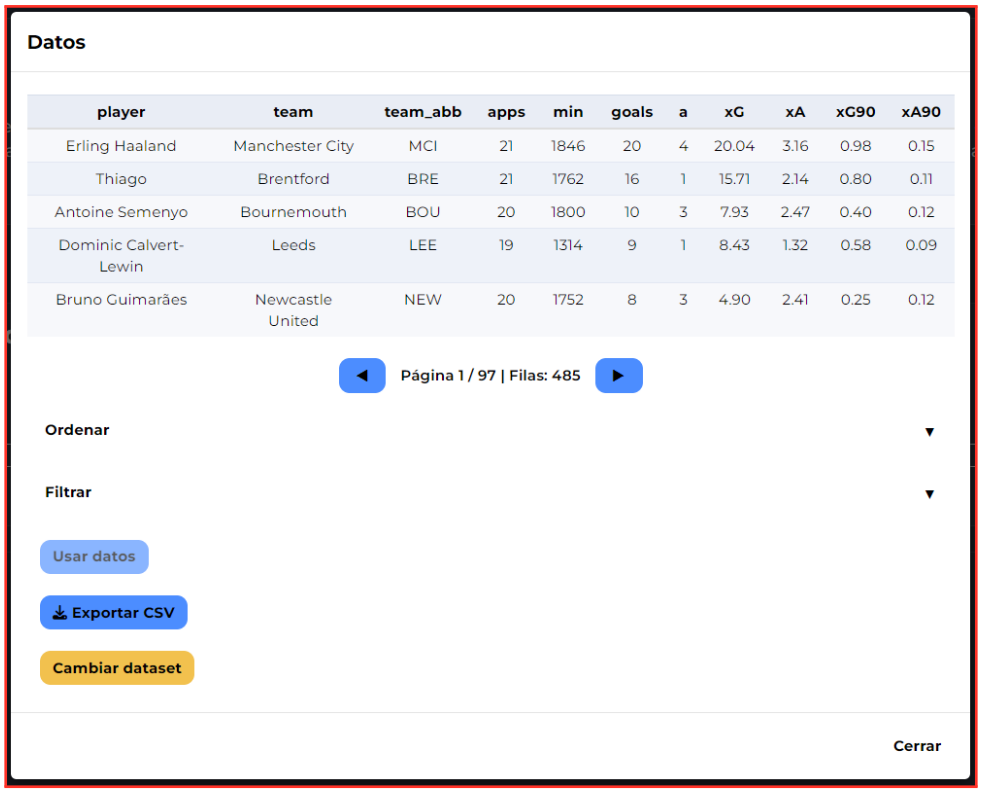

Figura 3: Visualización de datos mediante un *data frame*.

Figura 4: Desplegable para ordenar los datos.

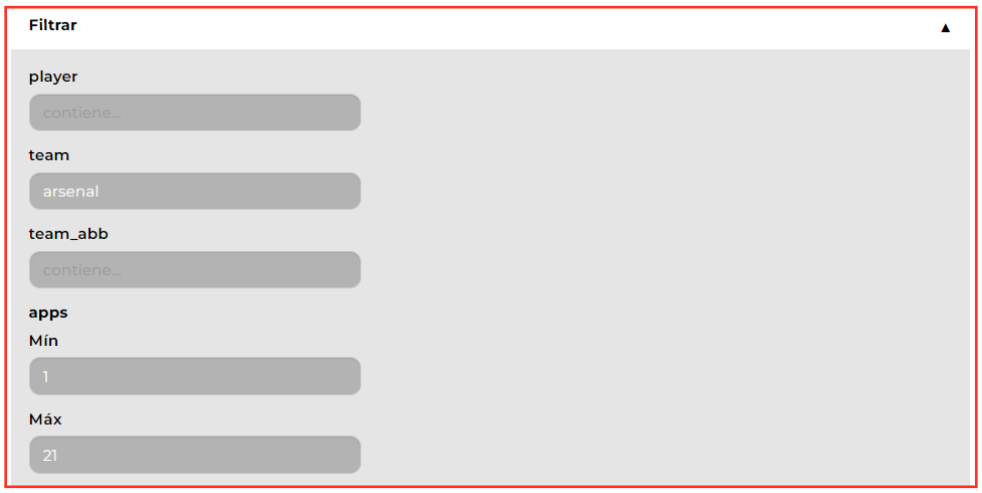

Figura 5: Desplegable para filtrar los datos.

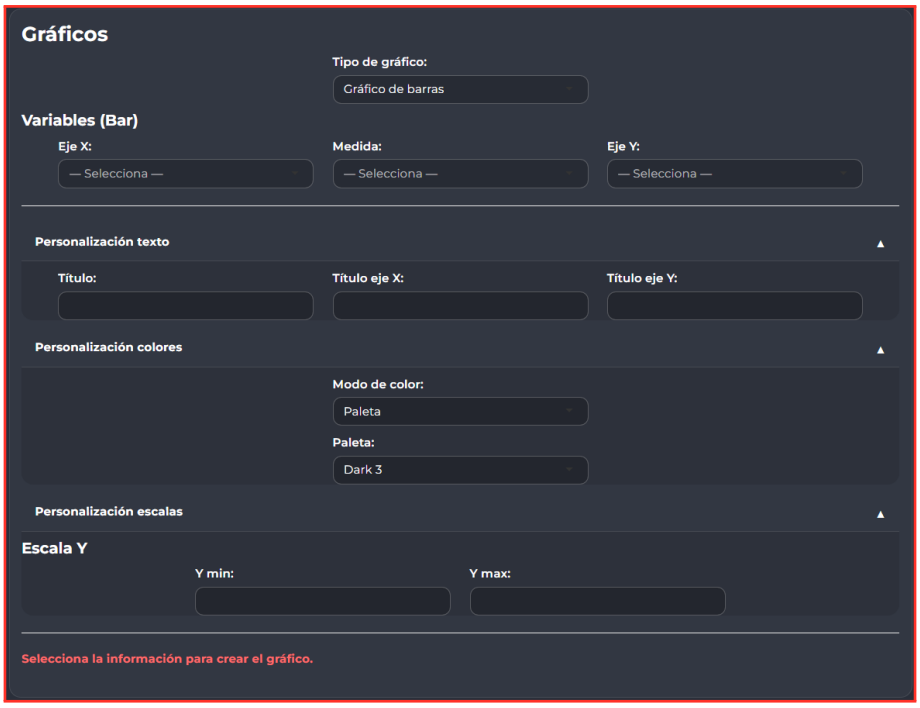

Figura 6: Opciones para la creación de los distintos gráficos.

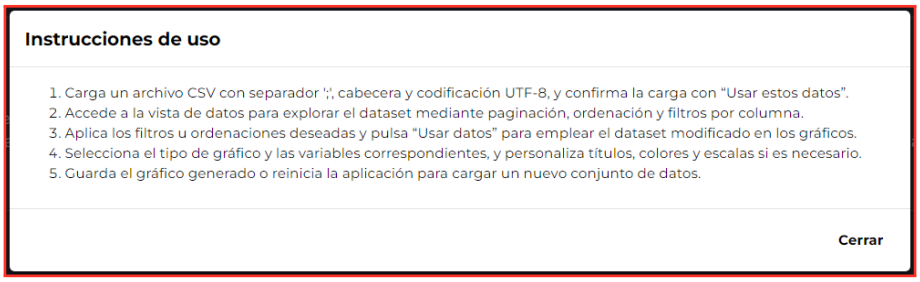

Figura 7: Página de instrucciones.

4.2 Ejemplos de creación de gráficos

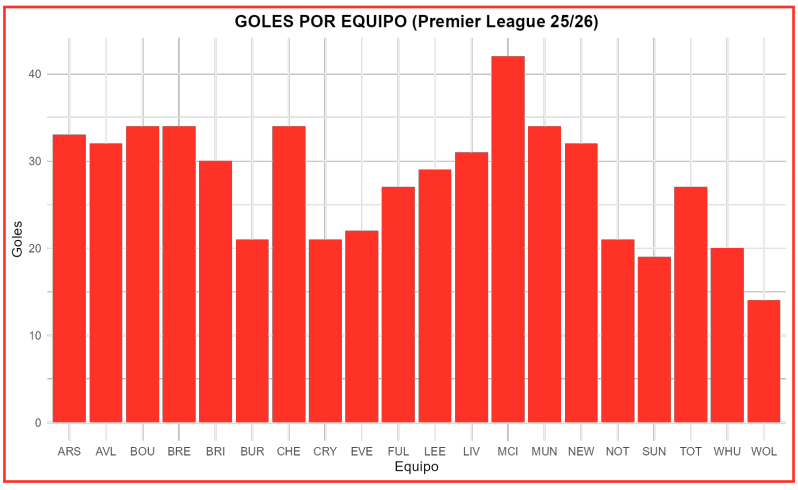

Figura 8: Ejemplo de gráfico de barras (Goles por equipo).

Figura 9: Ejemplo de gráfico de líneas (Evolución de goles).

Figura 10: Ejemplo de gráfico de dispersión (Goles vs. xG).

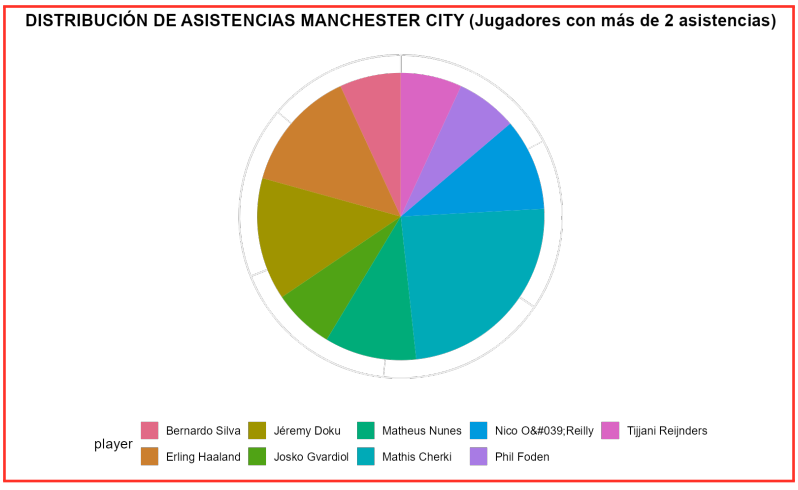

Figura 11: Ejemplo de gráfico circular (distribución de asistentes).

4.3 Bibliografía

- Introducción a *Shiny* en R. https://bastianolea.rbind.io/
- Tutorial *Shiny*. https://bookdown.org/shiny_tuto
- *Shiny* avanzado — Datapalooza UC 2023. https://jkunst.com/datapalooza
- Creación de una aplicación *Shiny*. https://programminghistorian.org/es
- *Shiny* for *Python*: render.table. https://shiny.posit.co
- Introducción a *ggplot2*: https://rpubs.com/ggplot2-introduction
- Repositorio GitHub con toda la codificación, conjuntos de datos de ejemplos, e imagenes: https://github.com/xavierrosinach/Actividad-Shiny.git