Gráficos en R

Instituto Nacional de Estadística (INE)

David Pérez Ros

27/11/24

Indice

Capítulo 1

Introducción

La visualización de datos es una herramienta fundamental en el análisis estadístico y la ciencia de datos, ya que permite interpretar y comunicar información compleja de manera efectiva. Con el creciente volumen de datos disponibles en diversas disciplinas, la capacidad de visualizar patrones y tendencias se ha vuelto esencial para la toma de decisiones informadas. En este contexto, R, un lenguaje de programación y entorno de software para análisis estadístico, ha desarrollado un ecosistema robusto y diverso de paquetes para la creación de gráficos.

Las funciones gráficas base de R ofrecían una solución sencilla y eficiente para la creación de gráficos básicos. Con el tiempo, la necesidad de visualizaciones más sofisticadas y personalizables llevó al desarrollo de paquetes adicionales que ampliaron significativamente las capacidades gráficas de R.

Hoy en día, la comunidad de R dispone de una variedad de paquetes especializados que permiten desde la creación de gráficos simples hasta la construcción de visualizaciones interactivas y altamente personalizadas. Esta interactividad mejora significativamente la capacidad de los usuarios para explorar datos de manera dinámica, ofreciendo nuevas perspectivas y facilitando la comprensión de información compleja.

En este estado del arte, se revisan los paquetes más relevantes para la visualización de datos en R, destacando sus características, ventajas y aplicaciones. Su versatilidad y su reciente incorporación de formatos como Bookdown, R Markdown, y Quarto lo hacen idóneo para realizar análisis de datos centrándose únicamente en dicho análisis y no perder tiempo en diseñar el entorno en el que se presentarán.

Este proyecto se ha realizado en Bookdown[1] con la idea de que sea fácilmente exportable a PDF o leído desde la web, además de poder evolucionar en el tiempo, incluyendo correcciones tipográficas o nuevos apartados. Se expondrán los diferentes tipos de paquetes para graficar en R, clasificándolos según su propósito: visualización exploratoria, visualización de resultados estadísticos, gráficos interactivos, gráficos para informes y publicaciones, entre otros. Esto permitirá una comprensión integral de las herramientas disponibles y su adecuada aplicación en diversos contextos de análisis de datos.

1.1 Paquetes Principales

- **ggplot2**: permite construir visualizaciones complejas mediante una sintaxis declarativa y altamente personalizable[2]. Su flexibilidad y capacidad para manejar grandes conjuntos de datos lo convierten en una herramienta esencial para analistas y científicos de datos.

- **plotly**: paquete popular que extiende la funcionalidad de ggplot2 al ofrecer gráficos interactivos. Desarrollado inicialmente como una biblioteca para Python, su integración con R ha permitido a los usuarios crear visualizaciones dinámicas y envolventes que pueden ser fácilmente compartidas en plataformas web[3].

[1] Ver (Xie, 2017) para más información sobre Bookdown.
[2] Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer.
[3] Sievert, C. (2020). *Interactive Web-Based Data Visualization with R, plotly, and shiny*. CRC Press.

- **lattice**: paquete diseñado para gráficos multivariantes, basado en el concepto de gráficos en trellis. Proporciona un enfoque sistemático para la visualización de datos multivariados mediante el uso de paneles condicionados[4]. Es particularmente útil en el análisis exploratorio de datos complejos.

- **leaflet**, **sf** y **tmap**: para la visualización de datos espaciales, los paquetes **sf** y **tmap** proporcionan herramientas especializadas. **sf** facilita la manipulación de datos geoespaciales, mientras que **tmap** permite la creación de mapas temáticos interactivos y estáticos[5]. Estos paquetes son esenciales para el análisis geoespacial y la presentación de resultados en disciplinas como la geografía y la ecología.

- **Tendencias Actuales**: Las tendencias actuales en la visualización de datos con R incluyen un aumento en la demanda de gráficos interactivos y dashboards, la integración con herramientas de presentación web, y el desarrollo de paquetes que hacen más accesible la creación de gráficos para usuarios menos técnicos[6].

1.2 Datos

En este proyecto la mayor parte de datos usados se han extraído directamente del servicio API del INE, a través del paquete `ineapir`. Para más información, (véase (Crespo, 2024)). Esto contribuye a la independencia del proyecto, sin necesidad de descargar datos de fuentes externas de manera "manual" para poder replicar los ejemplos. Del mismo modo, promociona y expone casos de uso para dicha herramienta.

Nota: Este espacio de trabajo se encuentra todavía en proceso de elaboración y por ello seguirá incluyendo información nueva de manera periódica.

[4] Sarkar, D. (2008). *Lattice: Multivariate Data Visualization with R*. Springer.
[5] Lovelace, R., Nowosad, J., & Muenchow, J. (2019). *Geocomputation with R*. CRC Press.
[6] Chang, W., Cheng, J., Allaire, J., Xie, Y., & McPherson, J. (2021). *shiny: Web Application Framework for R*. R package version 1.7.1.

Capítulo 2

Truco para infografías

En este apartado se expondrá como exportar por capas un gráfico de R a Power Point/ Photoshop. Este método puede resultar muy útil para realizar ciertas infografías, ya que en la herramienta de destino se puede modificar el formato del gráfico fácilmente.

Tomemos los datos de la población de España desde 2010 hasta 2022, para ello usaremos la API desarrollada para la obtención de datos. Para más información sobre esta, véase (Crespo, 2024).

1. **Buscar identificador de la serie**. Abriendo los datos en la web del INE, vemos que el código identificador de la serie es `ECP320` (véase cheatsheet de (Crespo, 2024)).
2. **Cargar los datos** usando función `ineapir::get_data_series()`.

```r
# Verificar si el paquete está instalado
if (!require("ineapir")) remotes::install_github("es-ine/ineapir")


# Cargar paquete
library(ineapir)

# Cargar Serie Población ECP320
a <- get_data_series("ECP320", dateStart = "2002/01/01", unnest = TRUE, tip = "AM")
datos <- data.frame(
  fecha = as.Date(a$Fecha),
  pob = a$Valor
)

# Formateamos año y mes
datos$year <- format(datos$fecha, "%Y")
datos$month <- format(datos$fecha, "%m")

# Filtramos para tener sólo población a 1 de Enero
datos <- datos[datos$month == "01", ]
```

3. Crear gráfico que represente la población a lo largo de los años.

```r
# Verificar si está instalado
if (!require("ggplot2")) install.packages("ggplot2")
library(ggplot2)
```

```r
# Function to set numbers with marks and without scientific notation
marks_no_sci <- function(x) format(x, big.mark = ".", decimal.mark = ",", scientific = FALSE)



# Crear el gráfico de puntos
grafico <- ggplot(datos, aes(x = year, y = pob)) +
  geom_point(stat = "identity", color = "#457e76") +
  labs(title = "Población en España", x = "Año", y = "Población") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(labels = marks_no_sci, limits = c(40000000, 50000000), breaks = seq(40000000, 5

grafico
```

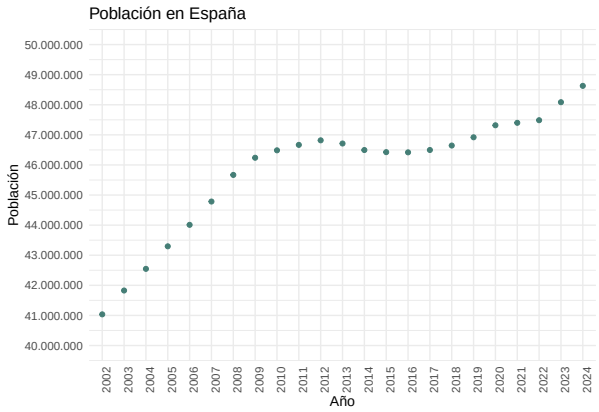


Figure 2.1: Evolución de la población a lo largo de los años.

Ahora imaginar que un equipo quiere realizar una infografía con este gráfico. Se puede guardar como `.svg` de tal manera que sea un gráfico vectorial y se pueda abrir en aplicaciones como Photoshop o Power Point.

Veamos:

```r
# Verificar si el paquete está instalado
if (!require("svglite")) install.packages("svglite")
library(svglite)
ggsave(file = "/Users/davpero/Desktop/grarfico.svg", plot = grafico)
```

Por ejemplo: Pegamos el gráfico en Power Point y clickando con el botón derecho le damos a Grupo-> Desagrupar. Lo que conseguimos es dejar la imagen como elementos vectoriales independientes, tal y como se puede ver en la Figura 2.2

```r
knitr::include_graphics(c("fig/capturas/plot1.png", "fig/capturas/plot2.png"))
```

Posteriormente, en Power Point, el equipo de infografías puede darle el formato deseado sin necesidad de tener que modificar el gráfico desde su código fuente (ggplot2). No obstante, estos cambios posibles se reducen a aspectos de formato tal como cambiar el tamaño de letra, colores, o eliminar elementos.

Veamos pues como eliminamos las rayas de fondo y cambiamos el tamaño y color de los textos:

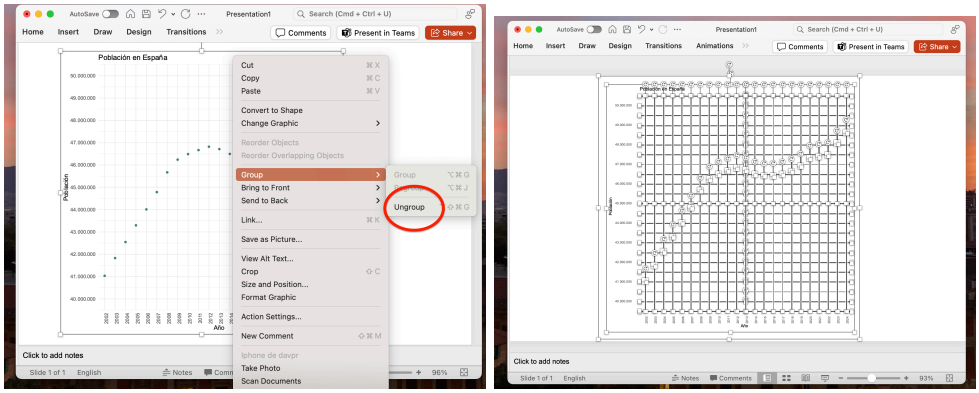


Figure 2.2: Pegado del gráfico como imagen vectorial

```
knitr::include_graphics("fig/capturas/plot3.png")
```

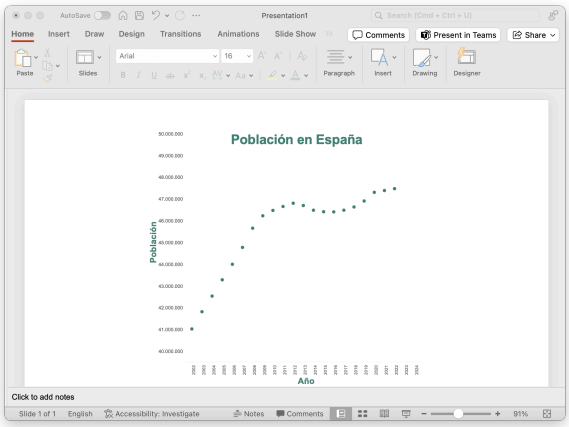


Figure 2.3: Modificación del gráfico en Power Point

Capítulo 3

lattice

El paquete **lattice** es una herramienta poderosa para la visualización de datos multivariantes en R. Desarrollado por Deepayan Sarkar, lattice está basado en el concepto de gráficos en trellis [1], que facilita la visualización de relaciones complejas entre múltiples variables mediante la creación de gráficos condicionados. Esto es especialmente útil en análisis exploratorios de datos, donde es crucial comprender las interacciones y patrones en conjuntos de datos grandes y complejos.

3.1 Características Principales

1. **Gráficos Condicionados**: lattice permite crear gráficos que muestran relaciones entre variables condicionadas a los valores de otras variables. Esto facilita la visualización de patrones en subgrupos de datos.
2. **Paneles Múltiples**: Los gráficos pueden ser divididos en paneles múltiples, cada uno mostrando una porción diferente del conjunto de datos, lo que permite una comparación visual directa entre diferentes subgrupos.
3. **Fórmulas**: Utiliza una fórmula para especificar las relaciones entre las variables que se van a graficar, proporcionando una sintaxis clara y concisa.
4. **Temas Personalizables**: lattice permite la personalización de temas gráficos, incluyendo colores, tamaños y tipos de letra, lo que facilita la creación de visualizaciones estéticamente agradables.
5. **Integración con el Modelo de Trellis**: La integración con el modelo de Trellis permite la creación de gráficos consistentes y bien organizados.

Lo primero escribimos unas líneas de código que verifiquen si el paquete está instalado, y en caso negativo lo instalen. Posteriormente lo cargamos:

```
# Verificar si el paquete está instalado
if (!require("lattice")) install.packages("lattice")

# Cargar paquete
library(lattice)
```

Como se comentaba, la principal funcionalidad de este paquete es que permite diferenciar cualquier tipo de gráfico (Diagrama de dispersión, Histograma,…) a partir de una variable categórica mostrando diferentes gráficos o superpuestos en uno mismo. Para ilustrar lo ventajoso de este gráfico se expondrá un ejemplo:

[1] Un **gráfico en trellis** es una visualización que presenta múltiples gráficos dispuestos en un conjunto de paneles, permitiendo comparar diferentes subconjuntos de datos de manera eficiente. Cada panel muestra el mismo tipo de gráfico, pero para diferentes segmentos de datos, facilitando la detección de patrones y tendencias en grupos distintos. Véase https://es.wikipedia.org/wiki/Gr%C3%A1fico_de_celos%C3%ADa

3.2 Datos de ejemplo

Vamos a usar el conjunto de datos `mtcars` que pertenece al paquete datasets el cual fue elaborado para la revista *Motor Trend US* en 1974 y que contiene el consumo de combustible y 10 aspectos relacionados con el diseño y rendimiento para 32 coches distintos. Es un dataset muy usado en el aprendizaje de R y que permite ejemplificar diversas técnicas estadísticas.

Variables del dataset mtcars:

1. **mpg**: Medida autonomía del coche. Millas recorridas por galón de combustible (miles per gallon)
2. **cyl**: Número de cilindros
3. **disp**: Desplazamiento (pulgadas cúbicas)
4. **hp**: Potencia (caballos de fuerza)
5. **drat**: Relación del eje trasero
6. **wt**: Peso del auto (miles de libras)
7. **qsec**: Tiempo de 1/4 de milla (en segundos)
8. **vs**: Tipo de motor (0 = V-shaped, 1 = Straight)
9. **am**: Tipo de transmisión (0 = Automática, 1 = Manual)
10. **gear**: Número de marchas
11. **carb**: Número de carburadores

```
data <- mtcars
data$gear <- factor(data$gear, levels = c(3, 4, 5))
data$cyl <- factor(data$cyl, levels = c(4, 6, 8))
head(data)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Suponer que se desea mostrar la relación entre `mpg` y `hp` para ver si hay relación entre la velocidad y la potencia del coche. Por ello, parece razonable realizar un gráfico tipo:

```
plot(hp ~ mpg, data = data, main = " Autonomía (mpg) vs. Potencia (hp) ")
```

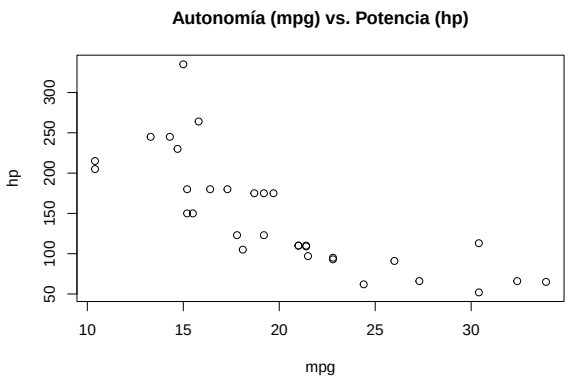


Figure 3.1: Gráfico de dispersión de potencia y auonomía de los automóviles.

Se observa que a medida que aumenta la potencia del motor (hp), disminuye el consumo de combustible medido en millas por galón (mpg), lo cual tiene sentido ya que a mayor potencia de un coche, más combustible se espera que gaste, y por tanto menor autonomía tendrá.

Ahora suponer que queremos hacer distinciones en función del número de cilindros del coche, es decir, el mismo **gráfico de dispersión condicionado por el número de cilindros**:

```
plot(hp ~ mpg, col = factor(cyl), data = data, main = " Autonomía (mpg) vs. Potencia (hp) ")
# Legend
legend("topright",
  title = "Cilindros",
  legend = levels(factor(data$cyl)),
  pch = 19,
  col = factor(levels(factor(data$cyl)))
)
```

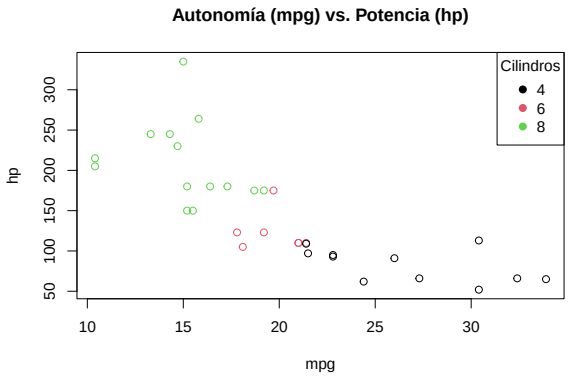


Figure 3.2: Gráfico de dispersión condicionado por el número de cilindros.

Se sigue observando una relación inversa entre potencia y autonomía de los automóviles, teniendo en cuenta que los de mayor cilindrada son los que mayor potencia y menor autonomía tienen. A medida que aumente el número de observaciones se tenderá a ver muy lleno el gráfico por lo que convendrá separar en varios gráficos dependiendo del número de cilindros.

3.3 Realización manual

Para separar estos tres gráficos, de manera "manual", se procedería:

```
# Disposición de los gráficos
par(mfrow = c(1, 3))
# 4 cilindros
plot(hp ~ mpg, data = data[data$cyl == 4, ], main = "4 cilindros", col = "black")

# 6 cilindros
plot(hp ~ mpg, data = data[data$cyl == 6, ], main = "6 cilindros", col = "red")

# 8 cilindros
plot(hp ~ mpg, data = data[data$cyl == 8, ], main = "8 cilindros", col = "green")
```

Conforme aumente el número de categorías, realizar estos gráficos será tedioso y ahí es donde entra en juego el paquete `lattice`

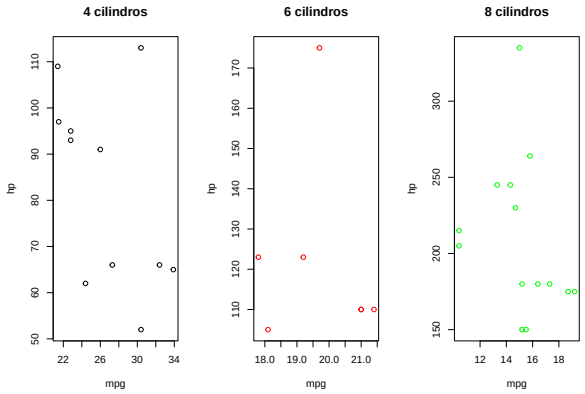


Figure 3.3: Gráfico de dispersión condicionado por el número de cilindros (separados).

3.4 Realización automática (lattice)

Véase que el siguiente gráfico realiza la misma tarea con mucho menos código:

```
# Gráfico por número de cilindros
xyplot(hp ~ mpg | cyl, group = cyl, data = data, scales = "free", aspect = "fill")
```

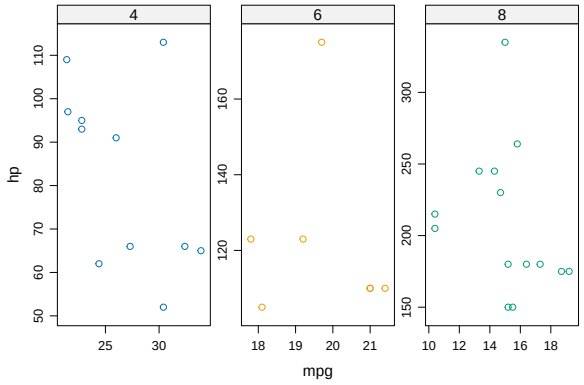


Figure 3.4: Gráfico de dispersión con lattice condicionado por el número de cilindros.

Además, se pueden combinar en un miso gráfico al igual que antes (quitando `| cyl`):

```
# Gráfico
xyplot(hp ~ mpg, group = cyl, data = data, scales = "free", aspect = "fill")
```

Es decir, veamos las forma general de esta función **Custom block ver info**

```
plot_function(y ~ x | g, group = g, data)
```

donde:

- **plot_function**: es cualquier función de graficar de `lattice`, por ejemplo
  - **xyplot**: Diagrama de dispersión
  - **density.plot**: Línea de densidad

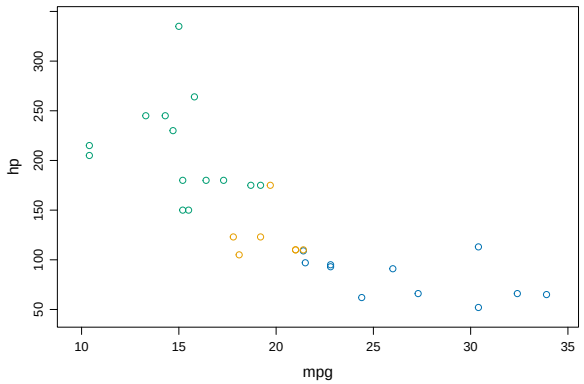


Figure 3.5: Gráfico de dispersión con lattice condicionado por el número de cilindros.

  - **histogram**: Histograma
  - **bwplot**: Diagramas de caja

- **| g** (OPCIONAL): Indica que el gráfico se va a dividir en tantos gráficos como valores tome la variable `data$g`. Es decir, nos mostrará el gráfico de `y~x` para cada grupo de la variable `g`
- **group=g**(OPCIONAL): indica que pinte los elementos dibujados agrupando por la variable `g`.
- **data**: nombre del conjunto de datos que contiene las variables `x,y,g`.

En el **siguiente ejemplo**, queremos representar la densidad de la autonomía de los vehículos (mpg) distinguiendo el número de cilindros que tienen, por ello:

- Añadir `densityplot(~mpg,data = data)` para el gráfico de densidad.
- Añadir argumento `| cyl` para realizar un gráfico por cada número de cilindros distinto.
- Añadir argumento `gorup=cyl` para colorear dependiendo del número de cilindros del vehículo.

```
# Gráficos independientes
densityplot(~ mpg | cyl, group = cyl, data = data, plot.points = TRUE)
```

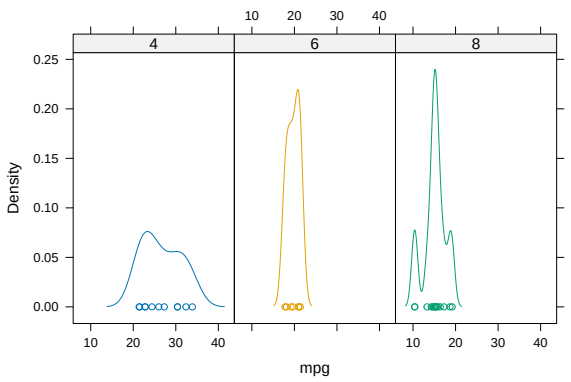


Figure 3.6: Densidad de la autonomía condicionada al número de cilindros (en gráficos distintos).

Ahora vamos a mostrarlos todos superpuestos en un mismo gráfico, eliminando el argumento `| cyl`:

```
# Superpuestos
densityplot(~mpg, group = cyl, data = data, plot.points = FALSE)
```

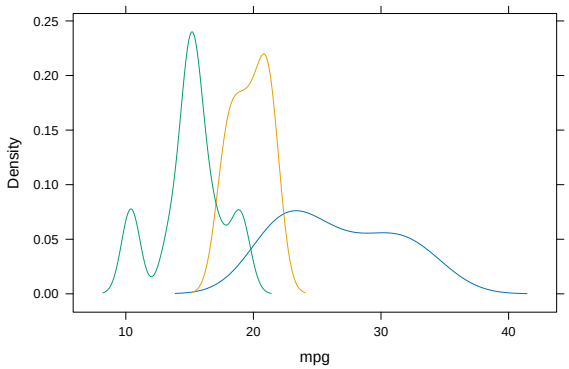


Figure 3.7: Densidad de la autonomía condicionada al número de cilindros.

3.5 Más información

Para más información acerca del paquete `lattice` y casos de uso, consultar:

- Página del paquete en CRAN:
- Sarkar, Deepayan. *Lattice: Multivariate Data Visualization with R*. Springer, 2008. Este libro proporciona una cobertura completa de las capacidades de `lattice` para la visualización de datos multivariantes en R.
- R Documentation - lattice. Página de documentación que proporciona ejemplos de uso, detalles de funciones y comentarios de la comunidad.
- Quick-R: Lattice Graphs Un tutorial práctico que cubre los conceptos básicos de los gráficos `lattice` y proporciona ejemplos de código.

Capítulo 4

plotly

El paquete plotly es una poderosa herramienta para la creación de gráficos interactivos en R. Desarrollado inicialmente como una biblioteca para Python, plotly ha sido adaptado para R, permitiendo a los usuarios beneficiarse de sus capacidades de visualización dinámica y envolvente. La interactividad de los gráficos generados con plotly facilita que se puedan integrar en sitios web, y en combinación con otros paquetes como `ineapir` (véase (Crespo, 2024)) se puede proporcionar un análisis directo e interactivo de datos extraidos en bruto del INE.

4.1 Características Principales

1. **Interactividad**: plotly destaca por sus capacidades interactivas, permitiendo a los usuarios hacer zoom, desplazar, y obtener información detallada mediante "hover" sobre los elementos del gráfico.
2. **Compatibilidad con ggplot2**: plotly puede convertir gráficos estáticos de ggplot2 en gráficos interactivos sin necesidad de reescribir el código original.
3. **Soporte para gráficos en 3D**: plotly facilita la creación de gráficos tridimensionales, como superficies y dispersión en 3D, que son difíciles de lograr con otras herramientas.
4. **Integración con R Markdown y Shiny**: los gráficos de plotly pueden integrarse fácilmente en documentos R Markdown y aplicaciones Shiny, mejorando la presentación de informes y el desarrollo de aplicaciones interactivas.
5. **Variedad de tipos de gráficos**: desde gráficos de líneas y barras hasta mapas y gráficos de superficie, plotly soporta una amplia gama de visualizaciones.

4.2 Tipos de gráficos

La base de *plotly* es la interactividad de los gráficos que permite exploraciones más profundas y detalladas. En general, se pueden realizar:

- **Mapas interactivos**: Permiten explorar datos geoespaciales por diferentes áreas.
- **Gráficos de barras**: Facilitan la comparación de categorías y pueden permitir la selección dinámica de subcategorías.
- **Series temporales**: Posibilitan filtrar por años y observar tendencias a lo largo del tiempo.
- **Histogramas interactivos**: Permiten ajustar el número de bins y explorar la distribución de los datos.
- **Box plots**: Ofrecen interactividad para seleccionar y resaltar datos atípicos o rangos específicos.
- **Gráficos de dispersión**: Permiten hacer zoom y seleccionar áreas específicas para análisis detallado.
- **Gráficos de calor**: Visualizan matrices de datos con interactividad para resaltar valores específicos.

- **Gráficos de burbujas**: Facilitan la visualización de relaciones entre múltiples variables con capacidad de ajuste de tamaño y color de las burbujas.

4.2.1 Nota

Debido a que los gráficos generados con `ploty` son interactivos, se genera un conflicto a la hora de guardar estos como pdf ya que se debe seleccionar una de las posibles vistas estáticas de dichos gráficos. La siguiente línea de código incluye el paquete `webshot` que se encarga de guardar capturas (imágenes estáticas) de dichos gráficos interactivos a la hora de imprimir la página como *.pdf* .

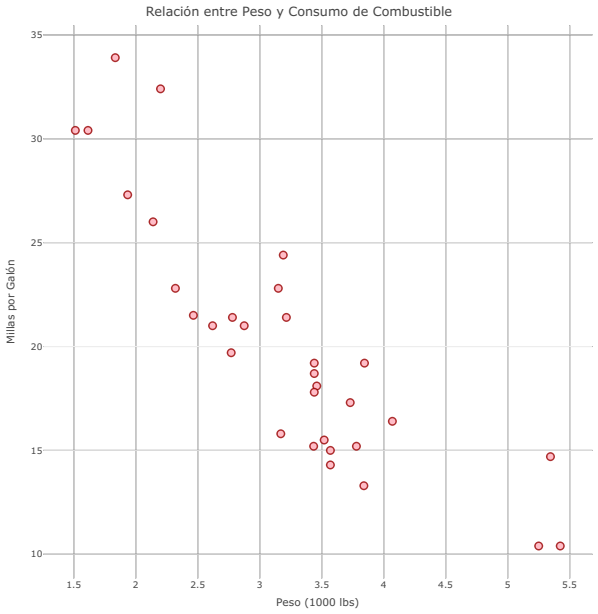
4.3 Ejemplos de Uso

4.3.1 Gráfico interactivo

A continuación se muestra un ejemplo básico de cómo crear un gráfico interactivo con plotly a partir de un conjunto de datos:

```r
# Instalar y cargar el paquete plotly
# install.packages("plotly")

# Verificar si está instalado
if (!require("plotly")) install.packages("plotly")
if (!require("dplyr")) install.packages("dplyr")
if (!require("webshot")) install.packages("webshot")
if (!require("ineapir")) install.packages("ineapir")

# Cargar paquete
library(dplyr)
library(plotly)
library(webshot)
library(ineapir)

# Datos de ejemplo
data <- mtcars
```

```r
# Crear un grfico interactivo de dispersión
p <- plot_ly(data,
  x = ~wt, y = ~mpg, width = 800, height = 800, type = "scatter", mode = "markers",
  marker = list(size = 10, color = "rgba(255, 182, 193, .9)", line = list(color = "rgba(152, 0, 0, .
) %>%
  layout(
    title = "Relación entre Peso y Consumo de Combustible",
    xaxis = list(title = "Peso (1000 lbs)"),
    yaxis = list(title = "Millas por Galón")
  )

# Mostrar el gráfico
p
```



En este ejemplo, se utiliza el conjunto de datos mtcars para crear un gráfico de dispersión interactivo que muestra la relación entre el peso del vehículo y el consumo de combustible. El argumento marker permite personalizar la apariencia de los puntos en el gráfico, mientras que layout se utiliza para añadir títulos y etiquetas a los ejes.

4.3.2 Tree Maps

En este apartado se va a exponer como crear los Tree Maps un tipo de gráfico muy útil para ver como se distribuye una cierta característica entre diferentes categorías. Por ejemplo, en este caso, como se distribuye el gasto medio por persona en España dentro de los grupos ECOICOP[1].

La ventaja de usar `plotly` es que nos permite construir TreeMaps interactivos, es decir, con distintos niveles de profundidad. En este caso en particular si apretamos en como se distribuye el gasto para el **ECOICOP 01: Agricultura, ganadería, caza y servicios relacionados con las mismas**, nos permite inspeccionar como se distribuye el gasto medio dentro de ese subnivel.

1. En la web del ine **buscamos la tabla que contiene los datos deseados**, relativos a *Gasto total, gastos medios y distribución del gasto de los hogares*, que tiene por id `25143`. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=25143&L=0)

2. Usando `ineapir` extraemos los datos usando el servicio API del INE. Ejecutando `get_metadata_table_varval(2514` vemos por que variables debemos filtrar para obtener los datos deseados.

```
get_metadata_table_varval(25143)
```

```
##        Id Fk_Variable
## 1   16473         349
## 2    8997          70
## 3    8998          70
## 4    8999          70
## 5    9000          70
## 6    9001          70
```

[1] La ECOICOP es una clasificación europea de consumo, que facilita la consulta conjunta con otras estadísticas como el IPC, y a su vez permite la comparabilidad de datos con otros países. Clasifica los bienes y servicios en 12 grandes grupos de gasto (del 01 al 12).

```
## 7    9002          70
## 8    9003          70
## 9    9004          70
## 10   9005          70
## 11   9006          70
## 12   9007          70
## 13   9008          70
## 14   9009          70
## 15   9010          70
## 16   9011          70
## 17   9012          70
## 18   9013          70
## 19   9015          70
## 20   8995          70
## 21 304092         762
## 22 304093         762
## 23 304094         762
## 24 304095         762
## 25 304096         762
## 26 304097         762
## 27 304098         762
## 28 304099         762
## 29 304100         762
## 30 304101         762
## 31 304102         762
## 32 304103         762
## 33 304104         762
## 34     72           3
## 35   8641           3
## 36   8642           3
## 37   8856          57
## 38   8857          57
## 39   8858          57
## 40   8859          57
## 41   8860          57
##                                  Nombre
## 1                       Total Nacional
## 2                            Andalucía
## 3                              Aragón
## 4                 Asturias, Principado de
## 5                         Balears, Illes
## 6                              Canarias
## 7                             Cantabria
## 8                       Castilla y León
## 9                  Castilla - La Mancha
## 10                            Cataluña
## 11                 Comunitat Valenciana
## 12                           Extremadura
## 13                               Galicia
## 14                 Madrid, Comunidad de
## 15                    Murcia, Región de
## 16          Navarra, Comunidad Foral de
## 17                          País Vasco
## 18                           Rioja, La
## 19                               Ceuta
## 20                             Melilla
## 21                      Índice general
```

```
## 22                                                    Alimentos y bebidas no alcohólicas
## 23                                                         Bebidas alcohólicas y tabaco
## 24                                                                 Vestido y calzado
## 25                                  Vivienda, agua, electricidad, gas y otros combustibles
## 26 Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar
## 27                                                                          Sanidad
## 28                                                                       Transporte
## 29                                                                  Comunicaciones
## 30                                                                  Ocio y cultura
## 31                                                                       Enseñanza
## 32                                                                Restaurantes y hoteles
## 33                                                              Otros bienes y servicios
## 34                                                                       Dato base
## 35                                                    Variación respecto al año anterior
## 36                                                        Variación respecto al año base
## 37                                                                     Gasto total
## 38                                                              Distribución porcentual
## 39                                                                Gasto medio por hogar
## 40                                                              Gasto medio por persona
## 41                                                     Gasto medio por unidad de consumo
##    Codigo
## 1      00
## 2      01
## 3      02
## 4      03
## 5      04
## 6      05
## 7      06
## 8      07
## 9      08
## 10     09
## 11     10
## 12     11
## 13     12
## 14     13
## 15     14
## 16     15
## 17     16
## 18     17
## 19     18
## 20     19
## 21     00
## 22     01
## 23     02
## 24     03
## 25     04
## 26     05
## 27     06
## 28     07
## 29     08
## 30     09
## 31     10
## 32     11
## 33     12
## 34
## 35
## 36
```

```
## 37
## 38
## 39
## 40
## 41
```

Vemos que necesitamos filtrar por:

- **Geografía**: Total Nacional.
- **ECOICOP**(2 dígitos): Todos.
- **Tipo de dato**: Dato base.
- **Gasto**: Gasto medio por persona.

Es decir, lo ponemos en formato de lista para poder pasarlo por la función `get_data_table()`como filtro.

```
# Filtro
filter_ecoicop_1 <- list(
  "349" = "16473", # Total Nacional
  "762" = "", # Todos GRUPOS de 2 dígitos del ECOICOP
  "3" = "72", # Dato Base
  "57" = "8859" # Gasto medio por persona
) # Dato base


# Obtenemos datos
# nlast=1 para devolver sólo para el último periodo
data_ecoicop_1 <- get_data_table(
  filter = filter_ecoicop_1, idTable = 25143, nlast = 1, tip = "AM", unnest = TRUE,
  metacodes = TRUE, validate = FALSE, metanames = TRUE
)
```

Como hemos dicho que queremos que en el gráfico se pueda mostrar cómo se distribuye el gasto también dentro de cada grupo de ECOICOP, necesitamos un dígito más de desagregación en la clasificación ECOICOP, es decir, 3 dígitos. Buscamos de nuevo en la web del INE en que tabla se encuentran estos datos y filtramos de nuevo por lo pertinente. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=25144&L=0)

```
# Vemos por que variables se puede fitrar
# get_metadata_table_varval(25144)
```

Vemos que necesitamos filtrar por:

- **Geografía**: Total Nacional.
- **ECOICOP**(3 dígitos): Todos.
- **Tipo de dato**: Dato base.
- **Gasto**: Gasto medio por persona.

```
filter_ecoicop_2 <- list(
  "349" = "16473", # Total Nacional
  "796" = "", # Todos GRUPOS de 2 dígitos del ECOICOP
  "3" = "72", # Dato base
  "57" = "8859" # Gasto medio por persona
)

# Obtenemos datos
# nlast=1 para devolver sólo para el último periodo
data_ecoicop_2 <- get_data_table(filter = filter_ecoicop_2, idTable = 25144, nlast = 1, tip = "AM",
```

Una vez obtenidos todos los datos, debemos juntar los datos en un data frame que contenga la siguiente estructura:

- **id**: único para cada fila
- **label**: etiqueta con cada grupo de ECOICOP
- **parent**: indica quien es el superior jerárquico de dicha fila. Por ejemplo de ECOICOP 01.1 su parent es 01. Es necesario para poder mapear bien el gráfico.
- **values**: indica el valor que toma dicha entrada.

```
# Datos para ECOICOP con 3 dígitos
data_ecoicop_2 <- data_ecoicop_2[, c("Subgrupos.de.gasto.(3.dígitos)", "Subgrupos.de.gasto.(3.dígito

# quitamos el total
data_ecoicop_2 <- data_ecoicop_2[data_ecoicop_2[, "Subgrupos.de.gasto.(3.dígitos).Codigo"] != "00",
```

```
# Función que a cada elemento de ECOICOP 3 dígitos, asocia su grupo de agregación de 2 dígitos
get_label <- function(inputs, label) {
  # Extrae los primeros dos caracteres de cada elemento del vector de entrada
  prefixes <- substr(inputs, 1, 2) # tomamos sólo 2 primeros eltos Ej: 01.1 tomamos 01
  # Usa sapply para aplicar la función a cada prefijo
  sapply(prefixes, function(prefix) {
    if (prefix == "01") {
      return(label[2])
    } else if (prefix == "02") {
      return(label[3])
    } else if (prefix == "03") {
      return(label[4])
    } else if (prefix == "04") {
      return(label[5])
    } else if (prefix == "05") {
      return(label[6])
    } else if (prefix == "06") {
      return(label[7])
    } else if (prefix == "07") {
      return(label[8])
    } else if (prefix == "08") {
      return(label[9])
    } else if (prefix == "09") {
      return(label[10])
    } else if (prefix == "10") {
      return(label[11])
    } else if (prefix == "11") {
      return(label[12])
    } else if (prefix == "12") {
      return(label[13])
    } else {
      return(NA) # Devuelve NA si el prefijo no coincide con ninguno
    }
  })
}
```

Ahora definimos los elementos `parenty` `labels` necesarios para ejecutar el TreeMap. Concretamente:

- **labels**: Define los nomrbes de cada elemento represetnado en el gráfico.
- **parent**: Contiene el grupo superior jerárquico de cada elemento. Por ejemplo GRUPO 01. ALIMENTOS Y BEBIDAS NO ALCOHÓLICAS*

```
# Etiquetas
labels <- c(data_ecoicop_1$`Grupos.de.gasto.(2.dígitos)`, data_ecoicop_2$`Subgrupos.de.gasto.(3.dígi

# Aplicamos funcion a ECOICOP con 3 dígitos y con label las de ECOICOP 2 dígitos
parent <- get_label(data_ecoicop_2$`Subgrupos.de.gasto.(3.dígitos).Codigo`, label = data_ecoicop_1$`

# parents, el del primer elemento es , es decir, el total. Los del primer nivel
# el parent es Índice Total=labels[1]
# En el segundo nivel, ya lo que corresponda
aux <- data_ecoicop_1$`Grupos.de.gasto.(2.dígitos)`
parents <- c("", rep(aux[1], length(aux) - 1), parent)
```

Ahora, se representan los datos como un TreeMap, ajustando los parámetros para que no haya problemas de visualización.

```
fig<-plot_ly(
  type = "treemap",
  labels = labels,
  parents = parents,
  textinfo = "label+value+percent entry+percent root",
  hoverinfo = "label+value+percent entry+percent root",
  values = as.integer(c(data_ecoicop_1$Valor, data_ecoicop_2$Valor)),
  maxdepth = 2,
  branchvalues = "total",
  tiling = list(squarifyratio = 2)
)
# Imporante ajustar este numero para que quede bontito
# Quitarlo para ver que pasa sin el
```

```
fig
```

4.3.3 Series Temporales

Plotly ofrece potentes herramientas para la visualización de series temporales, permitiendo a los usuarios interactuar con los datos de manera dinámica y obtener información valiosa de forma rápida y efectiva. Su capacidad para crear gráficos atractivos y funcionales lo convierte en una opción excelente para el análisis de datos temporales.

Características Clave de Plotly para Series Temporales

- **Interactividad**:
  - **Zoom y Pan**: Los usuarios pueden acercar y alejar secciones del gráfico.
  - **Hover**: Información detallada aparece al pasar el cursor sobre los puntos de datos.
  - **Selección de Rangos**: Rangos deslizantes y botones de selector para ajustar la ventana temporal visualizada.
- **Elementos Visuales**
  - **Líneas y Marcadores**: Visualización clara de tendencias y puntos específicos.
  - **Colores y Estilos Personalizables**: Facilitan la distinción entre diferentes series de datos.

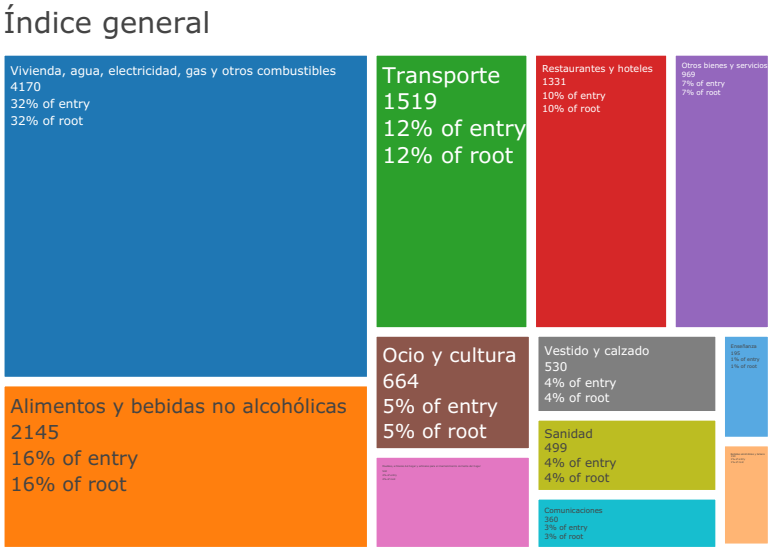


Figure 4.1: Distribución del gasto medio por persona en España para el año 2022.

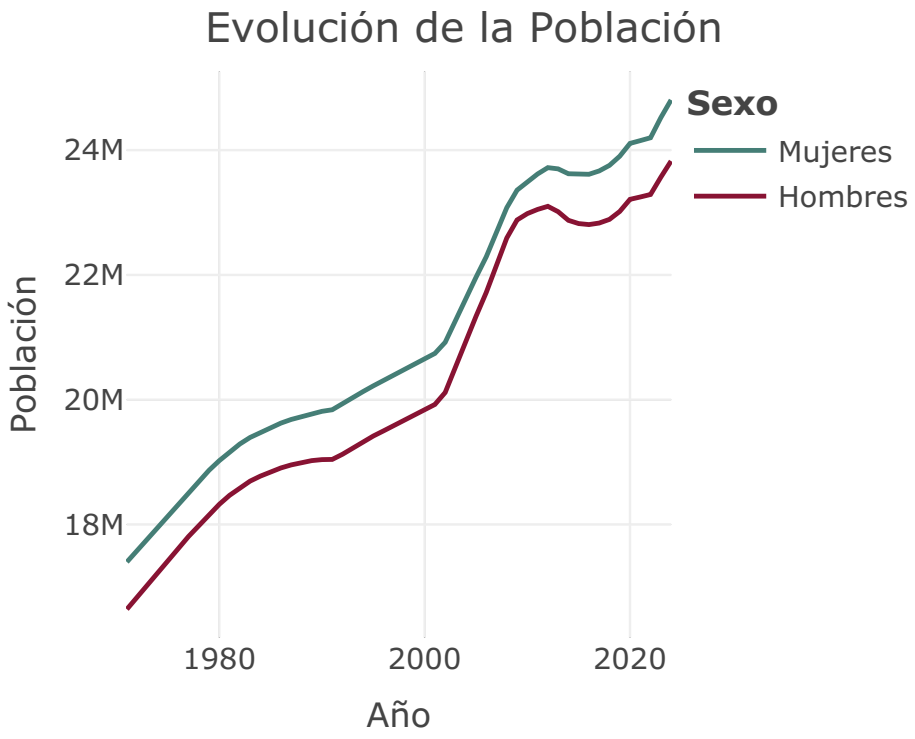
- **Funciones Avanzadas**:
  - **Rangeselector**: Botones para seleccionar periodos específicos (e.g., últimos 3 meses, último año).
  - **Rangeslider**: Barra deslizante para ajustar el rango de fechas mostrado.
  - **Anotaciones y Formateo**: Posibilidad de agregar notas y ajustar el formato de fechas.

Los datos los obtendremos usando `ineapir`, para más información (véase (Crespo, 2024)).

1. En la web del ine **buscamos la serie que contiene los datos deseados**, población de españa, que tiene por id `ECP319`(hombres) y `ECP318`(mujeres).
2. Usando `ineapir` extraemos los datos usando el servicio API del INE.

```
# Población Hombres
# Mirando en la web del INE cual es el código de la operación
# ECP319

# Hombres
a <- get_data_series(codSeries = "ECP319", tip = "AM", dateStart = "1971/01/01", unnest = TRUE)
# Mujeres
b <- get_data_series(codSeries = "ECP318", tip = "AM", dateStart = "1971/01/01", unnest = TRUE)

data <- data.frame(fecha = as.Date(a$Fecha), Year = a$Anyo, Hombres = a$Valor, Mujeres = b$Valor)


# Filtramos
data <- data %>%
  filter(format(as.Date(fecha), "%m") == "01")
```

```r
# Crear el gráfico interactivo
fig <- plot_ly(data, x = ~Year) %>%
  add_lines(y = ~Mujeres, name = "Mujeres", line = list(color = "#457e76")) %>%
  add_lines(y = ~Hombres, name = "Hombres", line = list(color = "#881333")) %>%
  layout(
    title = "Evolución de la Población",
    xaxis = list(
      title = "Año"
    ),
    yaxis = list(title = "Población"),
    legend = list(title = list(text = "<b>Sexo</b>")),
    hovermode = "closest"
  ) %>%
  config(locale = "es")

# Mostrar el gráfico
fig
```



4.3.3.1 Evolución IPC

Ahora vamos a mostrar la evolución del IPC y el IPC Subyacente a lo largo de los años. Para ello, lo primero buscamos en la web del INE en que tabla se encuentran dichos datos, siendo esta la "50902". A continuación, con `get_metadata_table_varval(50902)` vemos que variables y valores hay que tomar para filtrar IPC por Índice General y por Variación Anual.

1. En la web del ine **buscamos la tabla que contiene los datos deseados**, que tiene por id `50902`. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=50902&L=0)

2. Usando `ineapir` extraemos los datos usando el servicio API del INE. Ejecutando `get_metadata_table_varval(509`

```
get_metadata_table_varval(50902)
```

```
##        Id Fk_Variable
## 1  304092         762
## 2  304093         762
## 3  304094         762
## 4  304095         762
## 5  304096         762
## 6  304097         762
## 7  304098         762
## 8  304099         762
## 9  304100         762
## 10 304101         762
## 11 304102         762
## 12 304103         762
## 13 304104         762
## 14     83           3
## 15     84           3
## 16     74           3
## 17     87           3
##                                                                                 Nombre
## 1                                                                    Índice general
## 2                                                 Alimentos y bebidas no alcohólicas
## 3                                                      Bebidas alcohólicas y tabaco
## 4                                                           Vestido y calzado
## 5                                Vivienda, agua, electricidad, gas y otros combustibles
## 6  Muebles, artículos del hogar y artículos para el mantenimiento corriente del hogar
## 7                                                                          Sanidad
## 8                                                                      Transporte
## 9                                                                  Comunicaciones
## 10                                                                 Ocio y cultura
## 11                                                                      Enseñanza
## 12                                                           Restaurantes y hoteles
## 13                                                  Otros bienes y servicios
## 14                                                                         Índice
## 15                                                              Variación mensual
## 16                                                                Variación anual
## 17                                                 Variación en lo que va de año
##    Codigo
## 1      00
## 2      01
## 3      02
## 4      03
## 5      04
## 6      05
## 7      06
## 8      07
## 9      08
## 10     09
## 11     10
## 12     11
## 13     12
## 14      0
## 15      1
## 16      2
## 17      5
```

Vemos que necesitamos filtrar por:

- **Grupo**: Indice general.
- **Tipo dato**: Variación anual.

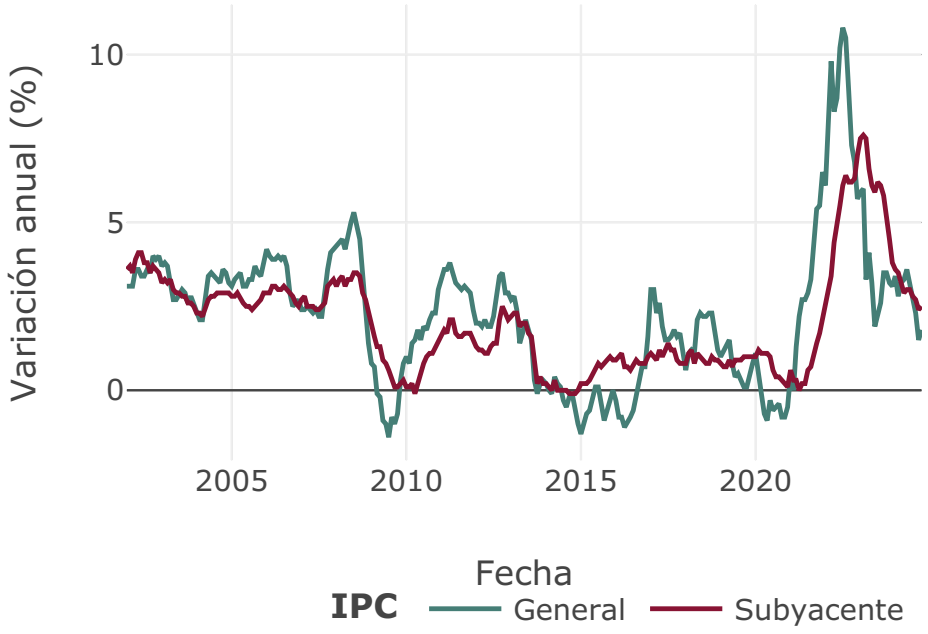
Posteriormente, se obtienen los datos, mediante el uso de dicho filtro.

```
# Todos los approaches  de filtrado son similares
# 762(grupo)= 304092(Indie general).      3(tipodato)=74(Variacion anual)
filter <- list(values = c("variación anual", "índice general"))
filter <- list(grupo = "304092", tipodato = "74")
filter <- list("762" = "304092", "3" = "74")



# IPC GENERAL
# Table url: https://www.ine.es/jaxiT3/Tabla.htm?t=50902&L=0
general <- get_data_table(
  idTable = 50902, filter = filter, unnest = TRUE,
  tip = "A", validate = FALSE
)

# IPC SUBYACENTE
# get_metadata_table_varval(50907)
filter <- list(
  "3" = "74", # Fk_variable=Id
  "269" = "12850" # Fk_variable=Id
)
# Filter core cpi
filter <- list(values = c(
  "variación anual",
  "General sin alimentos no elaborados ni productos energéticos"
))

# Request data of core cpi
# Table url: https://www.ine.es/jaxiT3/Tabla.htm?t=50907&L=0
subyacente <- get_data_table(
  idTable = 50907, filter = filter, unnest = TRUE,
  tip = "A", validate = FALSE
)

# Format Fecha column as date
general$Fecha <- as.Date(general$Fecha)
subyacente$Fecha <- as.Date(subyacente$Fecha)
```

Una vez hemos preparado los datos, los dibujamos en un gráfico interactivo:

```
# Plot cpi overall index
fig <- plot_ly(general,
  x = ~Fecha, y = ~Valor, name = "General",
  type = "scatter", mode = "lines", line = list(color = "#457e76", dash = "dashed")
)

## Plot core cpi
fig <- fig %>%
  add_trace(
```

```
    y = ~ subyacente$Valor, name = "Subyacente",
    mode = "lines", line = list(color = "#881333", dash = "dashed")
  ) %>%
  layout(
    yaxis = list(title = "Variación anual (%)"),
    legend = list(
      title = list(text = "<b> IPC </b>"),
      x = 0.25,
      y = -0.25,
      orientation = "h"
    ),
    hovermode = "x"
  ) %>%
  config(displayModeBar = FALSE) %>%
  config(locale = "es")

fig
```



4.3.4 Grafico de barras

Los gráficos de barras en Plotly permiten comparar fácilmente valores entre diferentes categorías. Pueden ser verticales u horizontales, y se pueden personalizar con colores específicos para cada barra. También soportan la agrupación de barras (bar charts) y la apilación (stacked bar charts). La orientación, orden y formato de los ejes pueden ser ajustados para mejorar la legibilidad, y se pueden añadir anotaciones y etiquetas detalladas para proporcionar contexto adicional a los datos visualizados.

1. En la web del ine **buscamos la tabla que contiene los datos deseados**, que tiene por id `2915`. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=2915&L=0)

2. Usando `ineapir` extraemos los datos usando el servicio API del INE. Ejecutando `get_metadata_table_varval(291`

```
get_metadata_table_varval(2915)
```

```
##       Id Fk_Variable                      Nombre Codigo
## 1  16473          70                       Total     00
## 2   8997          70                   Andalucía     01
## 3   8998          70                      Aragón     02
## 4   8999          70     Asturias, Principado de     03
## 5   9000          70              Balears, Illes     04
## 6   9001          70                    Canarias     05
## 7   9002          70                   Cantabria     06
## 8   9003          70              Castilla y León     07
## 9   9004          70         Castilla - La Mancha     08
## 10  9005          70                     Cataluña     09
## 11  9006          70         Comunitat Valenciana     10
## 12  9007          70                  Extremadura     11
## 13  9008          70                      Galicia     12
## 14  9009          70         Madrid, Comunidad de     13
## 15  9010          70            Murcia, Región de     14
## 16  9011          70 Navarra, Comunidad Foral de     15
## 17  9012          70                   País Vasco     16
## 18  9013          70                    Rioja, La     17
## 19  9015          70                        Ceuta     18
## 20  8995          70                      Melilla     19
## 21 20275          34                       Total      A
## 22 20267          34                Menos de 101      B
## 23 20264          34                De 101 a 500      C
## 24 20270          34               De 501 a 1.000     D
## 25 20260          34            De 1.001 a 2.000      E
## 26 20262          34            De 2.001 a 5.000      F
## 27 20274          34           De 5.001 a 10.000      G
## 28 20273          34          De 10.001 a 20.000      H
## 29 20263          34          De 20.001 a 50.000      I
## 30 20265          34         De 50.001 a 100.000      J
## 31 20266          34        De 100.001 a 500.000      K
## 32 20268          34              Más de 500.000      L
```

Vemos que necesitamos filtrar por:

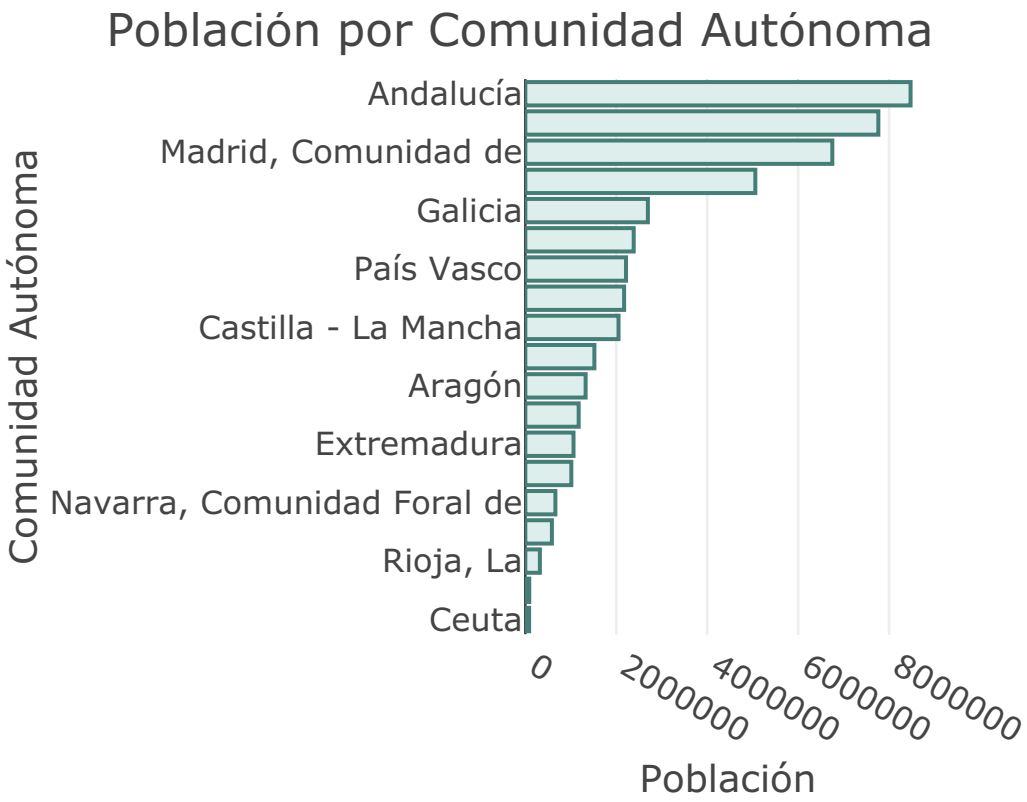
- **Tamaño de municipios**: Total.

```
filter <- list("34" = "20275") # 34(Tamaño municipios)=20275(Total)
data2 <- get_data_table(2915, filter = filter, unnest = TRUE, tip = "AM", nlast = 1, metanames = TRU
data2 <- data2[-1, ] # Quitamos primera observación. Total España
```

```
# Ordenar el dataframe por la columna 'poblacion'
data2 <- data.frame(poblacion = data2$Valor, ccaa = data2$Comunidades.y.Ciudades.Autónomas)
data2 <- data2[order(data2$poblacion), ]



# Crear el gráfico de barras horizontales
fig <- plot_ly(data2,
  x = ~poblacion, y = ~ factor(ccaa, levels = data2$ccaa), type = "bar", orientation = "h",
  marker = list(color = "#ddeeec", line = list(color = "#457e76", width = 1.5))
) %>%
```

```
  layout(
    title = "Población por Comunidad Autónoma",
    xaxis = list(
      title = "Población",
      tickformat = ".,", # Formatear los números para que se muestren con punto como separador de mi
      tickprefix = " ", # Espacio para mejorar la legibilidad
      separatethousands = TRUE # Asegurar que los miles se separen correctamente
    ),
    yaxis = list(title = "Comunidad Autónoma"),
    margin = list(l = 150) # Ajustar el margen izquierdo para evitar que las etiquetas se corten
  ) %>%
  config(locale = "es")
```

```
# Mostrar el gráfico
fig
```



4.3.5 Gráfico Sankey

Este tipo de gráfico es poco conocido pero muy útil para visualizar ciertos fenómenos sociales/demográficos. Por ejemplo: mostrar la evolución de la intención de voto, representando como se han desplazado las intenciones entre diferentes periodos; mostrar la evolución de la situación laboral de la comunidad, mostrando los flujos entre distintos estados laborales en distintos periodos.

Veamos pues este último gráfico. El objetivo es intentar reproducir la aplicación del INE Flujos de personas en el mercado laboral. En este caso, dentro de bookdown, la aplicación no se puede hacer tan dinámica como la presentada en la web del INE, ya que el formato no es el adecuado para ello [2].

1. En la web del ine **buscamos la tabla que contiene los datos deseados**, que tiene por id 13930. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=66257&L=0)

[2] Para ver una aplicación dinámica como la presente en la web del INE, ver apartado de Shiny Apps o FlexDashboards.

2. Usando `ineapir` extraemos los datos usando el servicio API del INE. Ejecutando `get_metadata_table_varval(6625`

```
# Tabla 66257. Información del mercado laboral
# Vemos sus metadatos para ver por que filtrar
get_metadata_table_varval(66257)
```

```
##        Id Fk_Variable                        Nombre Codigo
## 1     454          18                   Ambos sexos
## 2     452          18                       Hombres      1
## 3     453          18                       Mujeres      6
## 4  283949         386      Total (trimestre actual)
## 5   21332         386   Ocupados (trimestre actual)
## 6   21335         386    Parados (trimestre actual)
## 7   20279         386  Inactivos (trimestre actual)
## 8  283949         539    Total (trimestre anterior)
## 9   21332         539 Ocupados (trimestre anterior)
## 10  21335         539  Parados (trimestre anterior)
## 11  20279         539 Inactivos (trimestre anterior)
## 12 283997         539 No consta (trimestre anterior)
```

Vemos que debemos filtrar por ambos sexos:

```
filter <- list("18" = "454") # "Sexo"="Ambos sexos"

datos <- get_data_table(66257, filter = filter, metanames = TRUE, tip = "AM", metacodes = TRUE, unne

# Quitamos los datos totales
datos <- datos %>%
  filter(Relación.con.la.actividad.en.el.trimestre.actual.Id != "283949") %>%
  filter(Relación.con.la.actividad.en.trimestre.anterior.Id != "283949")
```

Para el sankey plot hay que crear un "diccionario" de etiquetas para saber como pintarlas. Véase documentación para más información.

```
# Necesario para el sankey plot
# Crear etiquetas únicas
labels <- unique(c(
  datos$Relación.con.la.actividad.en.trimestre.anterior,
  datos$Relación.con.la.actividad.en.el.trimestre.actual
))
labels <- c(labels[3], labels[1], labels[2], labels[4], labels[7], labels[5], labels[6])
# Crear un dataframe para mapear las etiquetas a índices
label_map <- data.frame(label = labels, id = seq_along(labels) - 1)
```

```
# Hacemos join con las etiquetas
filtered_data <- datos %>%
  left_join(label_map, by = c("Relación.con.la.actividad.en.trimestre.anterior" = "label")) %>%
  rename(source = id) %>%
  left_join(label_map, by = c("Relación.con.la.actividad.en.el.trimestre.actual" = "label")) %>%
  rename(target = id)
```

```
# Asignar colores a los enlaces (opcional)
link_colors <- ifelse(filtered_data$source == 0, "#FFD700", # Inactivos
  ifelse(filtered_data$source == 1, "#CD853F", # Ocupados
    ifelse(filtered_data$source == 2, "#FF4500", # Parados
```

```
      "#9e3a26" # No consta
    )
  )
)


fig <- plot_ly(
  type = "sankey",
  orientation = "h",
  node = list(
    label = labels,
    color = c("#FFD700", "#CD853F", "#FF4500", "#9e3a26", "#FFD700", "#CD853F", "#FF4500", "#9e3a26"
    pad = 15,
    thickness = 20,
    line = list(color = "black", width = 0.5)
  ),
  link = list(
    source = filtered_data$source,
    target = filtered_data$target,
    value = filtered_data$Valor,
    color = link_colors
  )
)

fig <- fig %>%
  layout(
    title = "Transición de Actividades Económicas entre Trimestres",
    font = list(size = 10)
  ) %>%
  config(locale = "es")
fig
```

4.4 Más información

Para más información sobre el paquete plotly y sus posibles funciones, buscar en la siguiente web:

https://plotly.com/r/getting-started/

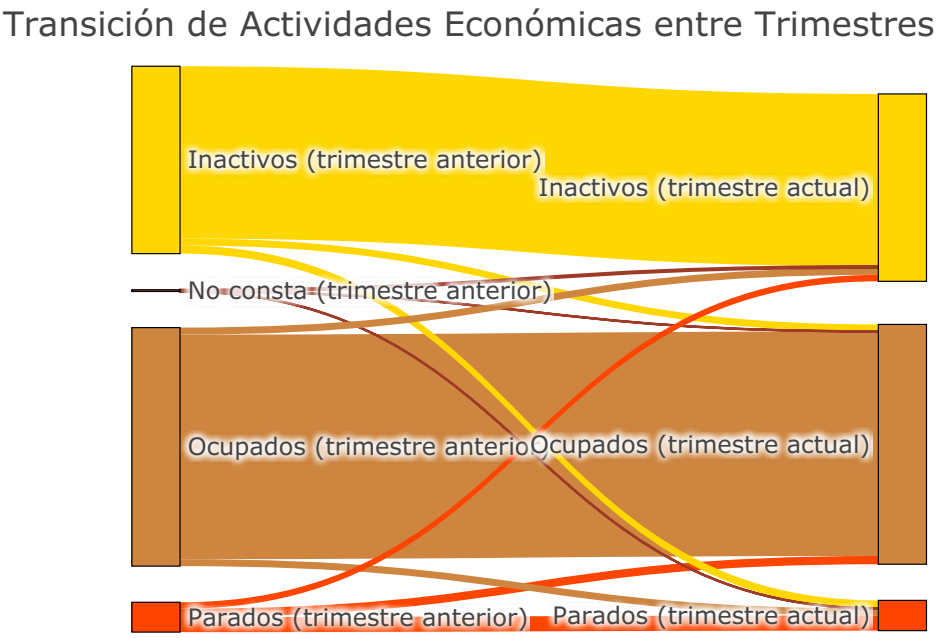


Figure 4.2: Flujos de personas en el mercado laboral en España 2024 T2

Capítulo 5

SF

fsda

Capítulo 6

Dashboards en R

Una de las principales ventajas de realizar dashboards en R es que este es de **software libre**, y que por tanto no se requiere licencias para su utilización. Además, con apoyo de librerías como *ineapir*, se puede combinar en un mismo documento el proceso que abarca desde la extracción de datos hasta la generación del dashboard, siendo a posteriori fácilmente actualizable.

En una época en la que la transparencia por parte de las instituciones se erige como pilar fundamental de sus principios, estos dashboards parecen la herramienta idónea para los constructores de estadísticas oficiales. Al ser de código abierto, se puede compartir el código fuente de los cuadros de mando, lo que conlleva ventajas significativas:

- El **proceso de elaboración** es **público**, sirviendo como base para otros trabajos y viendo que no existen manipulaciones.
- Se puede implementar servicios como **GitHub** que permiten a los usuarios hacer pull requests o abrir issues, creando una comunidad que mejore notablemente dichos recuros y que sirva como respaldo técnico.
- Permite reusar (bajo la licencia elegida) los cuadros de mando fácilmente, aportando valor añadido y que su utlización aumente.

En el siguiente cuadro se expone la comparación de distintos servicios de creación de dashboards:

Característica	R (Shiny)	Tableau	PowerBI
Facilidad de uso	Intermedio, requiere conocimientos en R y programación	Alto, interfaz intuitiva y fácil de usar	Alto, interfaz intuitiva y fácil de usar
Precio	Software libre	Licencia pagada, versiones disponibles según características	Licencia pagada, versiones disponibles según características
Proceso completo	Sí, desde la extracción de datos hasta la creación del dashboard	No, requiere herramientas adicionales para la extracción y procesamiento de datos	No, requiere herramientas adicionales para la extracción y procesamiento de datos
Publicación web	Sí, con `shinyapps.io` o servidores propios	Sí, con Tableau Public o Tableau Server	Sí, con PowerBI Service
Actualización fácil	Sí, puede automatizarse con scripts de R	Sí, pero puede requerir configuración adicional	Sí, pero puede requerir configuración adicional
Código reutilizable	Sí, scripts y funciones en R reutilizables	Limitado, se pueden reutilizar algunos componentes	Limitado, se pueden reutilizar algunos componentes

Característica	R (Shiny)	Tableau	PowerBI
Código abierto y replicable	Sí, código en R es abierto y puede ser revisado para transparencia	No, código cerrado	No, código cerrado
Capacidad de integración	Alta, puede integrarse con muchas fuentes de datos y otras herramientas de R	Alta, se puede conectar a muchas fuentes de datos pero con limitaciones en algunas integraciones	Alta, se puede conectar a muchas fuentes de datos pero con limitaciones en algunas integraciones
Interactividad	Alta, permite una interactividad compleja con `shiny` y `plotly`	Alta, permite interactividad pero con menos flexibilidad que Shiny	Alta, permite interactividad pero con menos flexibilidad que Shiny
Escalabilidad	Alta, puede manejar grandes volúmenes de datos dependiendo de la implementación	Media, puede manejar grandes volúmenes de datos pero requiere hardware adecuado	Alta, diseñado para manejar grandes volúmenes de datos con infraestructura adecuada
Soporte y comunidad	Amplia, fuerte comunidad de usuarios y mucha documentación	Amplia, fuerte comunidad de usuarios y soporte oficial	Amplia, fuerte comunidad de usuarios y soporte oficial
Curva de aprendizaje	Pronunciada, requiere aprender R y conceptos de programación	Baja, más fácil de aprender para usuarios sin experiencia técnica	Baja, más fácil de aprender para usuarios sin experiencia técnica
Seguridad	Alta, depende de la configuración del servidor	Alta, con opciones robustas en Tableau Server	Alta, con opciones robustas en PowerBI Service
Visualización	Alta, con gran flexibilidad y personalización	Alta, con una amplia gama de visualizaciones listas para usar	Alta, con una amplia gama de visualizaciones listas para usar
Colaboración	Media, se puede hacer a través de Git y otras herramientas de control de versiones	Alta, con Tableau Server y Tableau Online para colaboración en tiempo real	Alta, con PowerBI Service para colaboración en tiempo real
Mantenimiento	Requiere mantenimiento manual y gestión de dependencias	Requiere mantenimiento regular y gestión de licencias	Requiere mantenimiento regular y gestión de licencias
Despliegue	Flexible, puede desplegarse en cualquier servidor compatible	Controlado, generalmente a través de Tableau Server o Tableau Public	Controlado, generalmente a través de PowerBI Service

A continuación se expondrán las dos principales herramientas de creación de dashboards en R, que son **FlexDashboard** y **Shiny**.

6.1 Flexdashboard

Se trata de un paquete de R, flexdashboarad, cuyo propósito es proporcionar una forma simple de escribir aplicaciones interactivas en R basadas en lenguaje Markdown, lo que permite escribir el contenido usando bloques de código R y luego transformarlo en un documento HTML o PDF que incluye gráficos y widgets. Las principales características de flexdashboard son:

- **Facilidad de uso**: Permite crear dashboards sin necesidad de escribir mucho código de interfaz. Simplemente organizando el contenido en secciones y filas se genera un diseño de panel intuitivo.
- **Layouts adaptativos**: Ofrece disposiciones flexibles, como filas y columnas, para distribuir los gráficos y tablas.
- **Interactividad básica**: Se pueden agregar elementos de interacción usando bibliotecas de gráficos interactivas, como plotly o highcharter, pero el nivel de interactividad es más limitado en comparación con Shiny,que aunque no es interactivo por sí mismo, se pueden integrar elementos de Shiny para mejorar la interactividad.

6.1.1 Construcción

De aquí en adelante se usará el siguiente dashboard:

```
knitr::include_graphics("fig/capturas/flexdashboard.png")
```

Figure 6.1: Flexdashboard disponible en la siguiente web shinyapps.io/r_flexdashboard y código fuente en repositorio github.com/davidperezros/r_flexdashboard.

Este cuadro de mando contiene los siguientes **elementos fundamentales**:

- **Filtro**: Permite seleccionar una Comunidad Autónoma y un Año, para el que se muestran los elementos visuales del dashboard.
- **Cards**: Tarjetas visuales superiores que permiten resumir información de manera muy visual. Más adelante se muestra cómo se han construido.
- **TreeMap**: Gráfico que muestra el la distribución del gasto medio por persona en la comunidad y año seleccionado dentro de los grupos ECOICOP. Para ver como se construye (Sección 4.3.2).
- **Leaflet Map**: Mapa interactivo que muestra el ámbito geográfico seleccionado en el filtro.

6.1.2 Estructura proyecto

Para crear un flexdashboard se ha seguido la guía oficial, disponible en https://pkgs.rstudio.com/flexdashboard/. Concretamente tenemos el siguiente esquema,

```
r_flexdashboard_project/
   .gitignore                # Archivo de configuración para Git
   .Rhistory                 # Archivo de historial de R
   _site.yml                 # Archivo de configuración del sitio
   index.Rmd                 # Archivo principal de flexdashboard
   r_flexdashboard.Rproj     # Archivo de proyecto de RStudio
   README.md                 # Archivo de documentación README
   style.css                 # Archivo CSS (este puede estar dentro de la carpeta styles/)
```

donde:

- **.gitignore** : Contiene los archivos que se deben ignorar en git, es decir que no se desea tener un control de versiones de ellos.
- **_site.yml **: Archivo de configuración del sitio.
- **index.Rmd**: Contiene todo el cuadro de mando. Desde la lectura de datos hasta la creación de elementos visuales y su presentación.
- **r_flexdashboard.Rproj**: Ajustes globales del proyecto en R.
- **README.md**: Archivo que muestra información en la página principal del repositorio en GitHub.
- **style.css**: Archivo css de estilos, modificando los colores de las fuentes, tamaños, estructuras de elementos visuales...

6.2 Ventajas de cada servicio

Característica	Flexdashboard	Shiny
Nivel de Interactividad	Básica; depende de las bibliotecas de gráficos	Alta; interactividad avanzada y en tiempo real
Facilidad de Uso	Fácil; requiere poco código de interfaz	Moderada a compleja; más control pero más configuración
Diseño y Layout	Layout basado en RMarkdown, limitado a filas/columnas	Totalmente personalizable mediante código en R
Reactivo	Limitado; necesita Shiny para reactividad	Totalmente reactivo
Ideal para	Dashboards estáticos o semiestáticos	Aplicaciones web interactivas complejas

En general, flexdashboard es una excelente opción cuando se necesita construir dashboards rápidos y fáciles de usar sin necesidad de interactividad avanzada, mientras que Shiny es la mejor elección para aplicaciones complejas donde el usuario necesita manipular datos en tiempo real o realizar análisis detallados. Ambas herramientas pueden complementarse: un flexdashboard puede integrar componentes de Shiny para mejorar su interactividad, combinando la simplicidad de un panel con la potencia de una aplicación reactiva.

6.3 Shiny

Se trata de un paquete de R, shiny, cuyo propósito es proporcionar una forma simple de realizar aplicaciones interactivas en R. Las principales características de flexdashboard son:

- **Separación de componentes**: Una aplicación Shiny típica se divide en dos partes principales: ui (interfaz de usuario) y server (servidor).
  - **ui:** Define el diseño y la apariencia de la aplicación. Incluye los elementos de la interfaz de usuario como botones, gráficos, tablas, etc.
  - **server:** Contiene la lógica de la aplicación, es decir, el código R que procesa los datos y genera la salida que se mostrará en la interfaz de usuario.
- **Flexibilidad**: Aunque la curva de aprendizaje de la herramienta puede ser un poco lenta, ofrece una alta flexibilidad a la hora de mostrar los elementos interactivos y manejar los datos.
- **Alta reactividad**: alta capacidad para actualizar de manera automática y eficiente la interfaz de usuario en respuesta a cambios en los datos de entrada o en la interacción del usuario.
- **Comunidad web**: Shiny cuneta con una amplia comunidad web (mayoritariamente en inglés), donde se puede encontrar ejemplos y respuesta a muchos de los problemas que puedan surgir durante su creación
- **Shiny para Pyhton**: La librería de shiny también está disponible en Python, por lo que una vez se aprenda a manejar se puede usar en ambos lenguajes de programación. Aunque tienen elementos diferentes propios de cada lenguaje de programación, la base es la misma y por tanto shiny se usa de manera similar en ambos lenguajes.

6.3.1 Construcción y herramientas útiles

A la hora de empezar a crear una Shiny App (tanto en R como en Python), se recomienda seguir la guía https://shiny.posit.co, donde se explica los pasos básicos para crear una aplicación interactiva. Además existen otras webs muy útiles como pueden ser:

- **Shiny Gallery**: https://shiny.posit.co/r/gallery/. Ofrece ejemplos de código de diversas aplicaciones shiny.
- **Componentes**:https://shiny.posit.co/r/components/. Ofrece ejemplos de como incluir en la aplicación componentes que van desde sliders hasta value boxes,numeric inputs,..
- **Layouts**: Ofrece ejemplos de como implementar distintos layouts en nuestra aplicación, dependiento de nuestro objetivo. https://shiny.posit.co/r/layouts/
- **bslib**: https://rstudio.github.io/bslib/ Librería de R que dispone de una interfaz interactiva y actualizada para shiny. Contiene layouts, theming y una app interactiva en la que se permite construir value boxes con el formato y iconos/gráficas deseados. Realmente útil.

Además hay infinidad de documentación web en internet sobre como realizar un shiny, mucha de ella en libros en formato bookdown.

6.4 Publicación web

PExisten varias opciones para publicar tanto un shiny como un flexdahsboard y que un espectro amplio de potenciales consumidores tengan acceso a el. Las más comunes son:

- **PositCloud**: La cuenta básica es gratuita pero tiene un límite de apps publicadas y horas de consulta mensuales.
- **ShinyApps.io**: Es un producto de software como servicio (SaaS) alojado en la nube por Posit. Tiene planes gratuitos y de pago.

Para más información y ventajas de cada una, consultar las respectivas páginas principales de estos servicios. El flexdashboard y el shiny de estos ejemplos se han publicado con *ShinyApps.io*.

6.5 Shinylive

Leyendo por la web, existe un paquete de R llamado shinylive que permite ejecutar una aplicación shiny en la web sin necesidad de un servidor. Por lo tanto, puede ser interesante para subirlo a github y permitir crear una github pages. Es prometedor.

Capítulo 7

Leaflet

Leaflet es un paquete que permite integrar mapas interactivos en aplicaciones web y en documentos R. Utiliza la biblioteca JavaScript Leaflet, lo que proporciona una gran cantidad de funcionalidades para visualizar datos geoespaciales de forma intuitiva. Con Leaflet, puedes agregar marcadores, polígonos, capas de datos y muchos otros elementos a tus mapas. Su simplicidad y flexibilidad lo hacen ideal para visualizaciones geográficas en proyectos de análisis de datos.

7.1 General

7.1.1 Características Principales de Leaflet

1. **Interactividad**: Leaflet permite crear mapas interactivos donde los usuarios pueden acercar, alejar y hacer clic en elementos para obtener información adicional a través de ventanas emergentes (popups).
2. **Capas Múltiples**: Puedes agregar diferentes tipos de capas al mapa, como capas de imagen, capas de mosaico, y datos vectoriales, lo que permite superponer información.
3. **Marcadores Personalizables**: Leaflet ofrece la posibilidad de personalizar los marcadores (íconos, colores, tamaños) y añadir popups informativos para cada uno.
4. **Soporte para GeoJSON**: Permite la importación y visualización de datos geoespaciales en formato GeoJSON, facilitando la representación de geometrías complejas.
5. **Simplicidad de Uso**: La sintaxis es intuitiva y sencilla, lo que permite a los usuarios crear mapas rápidamente con un código mínimo.
6. **Compatibilidad con Dispositivos Móviles**: Los mapas son responsivos y funcionan bien en dispositivos móviles, garantizando una buena experiencia de usuario.
7. **Extensibilidad**: Leaflet puede ser ampliado mediante complementos (plugins) que permiten agregar nuevas funcionalidades, como gráficos de tiempo, clusters de puntos, y más.
8. **Temas Personalizables**: Se pueden aplicar diferentes estilos y temas a los mapas para adaptarlos a la visualización deseada.

7.1.2 Tipos de Gráficos en Leaflet

1. **Mapas Base**:
   - Mapas de mosaico (Tiles): Proporcionan una capa base de fondo (OpenStreetMap, Google Maps, etc.).

- Mapas de satélite y otros estilos personalizados.

2. **Marcadores**:
   - **Marcadores Simples**: Puntos que indican ubicaciones específicas en el mapa.
   - **Marcadores Agrupados**: Agrupan varios marcadores en una sola vista para evitar el desorden visual (clustering).
3. **Polígonos y Líneas**:
   - **Polígonos**: Representan áreas geográficas específicas (ejemplo: límites de comunidades autónomas).
   - **Líneas**: Representan rutas o trayectorias, como carreteras o caminos.
4. **Capas de Calor (Heatmaps)**:
   - Visualizan la densidad de puntos en un área específica, utilizando colores para representar la intensidad.
5. **Gráficos de Tiempos**:
   - Permiten visualizar datos a lo largo del tiempo, añadiendo un componente temporal a los mapas.
6. **Visualización de Datos**:
   - **Data Binding**: Vincula datos a los marcadores o polígonos para mostrar información adicional (por ejemplo, población, estadísticas).
7. **Mapas de Coropletas**:
   - Representan datos cuantitativos a través de variaciones en el color de las áreas geográficas, ideal para visualizar indicadores como la población o la densidad de un fenómeno.

7.2 Mapas

7.2.1 Ejemplo esperanza de vida

Vamos a representar la esperanza de vida por comunidades autónomas en un mapa de España. Para ello necesitamos **extraer primero los datos**, que como ya se ha comentado previamente se obtienen a partir del paquete `ineapir`.

1. En la web del ine **buscamos la tabla que contiene los datos deseados**, que tiene por id `27154`. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=13930&L=0)
2. Usando `ineapir` extraemos los datos usando el servicio API del INE. Aplicando `get_metadata_table_varval(2715`

```
get_metadata_table_varval(27154)
```

```
##        Id Fk_Variable                                  Nombre Codigo
## 1    8997          70                               Andalucía     01
## 2    8998          70                                  Aragón     02
## 3    8999          70                  Asturias, Principado de     03
## 4    9000          70                           Balears, Illes     04
## 5    9001          70                                 Canarias     05
## 6    9002          70                                Cantabria     06
```

```
## 7    9003          70                                Castilla y León     07
## 8    9004          70                           Castilla - La Mancha     08
## 9    9005          70                                       Cataluña     09
## 10   9006          70                           Comunitat Valenciana     10
## 11   9007          70                                    Extremadura     11
## 12   9008          70                                        Galicia     12
## 13   9009          70                           Madrid, Comunidad de     13
## 14   9010          70                              Murcia, Región de     14
## 15   9011          70                   Navarra, Comunidad Foral de     15
## 16   9012          70                                   País Vasco     16
## 17   9013          70                                    Rioja, La     17
## 18   9015          70                                        Ceuta     18
## 19   8995          70                                      Melilla     19
## 20    451          18                                        Total
## 21    452          18                                      Hombres      1
## 22    453          18                                      Mujeres      6
## 23  15319         355                                       0 años     Y0
## 24  15659         360                                  De 1 a 4 años   Y1T4
## 25  15664         360                                  De 5 a 9 años   Y5T9
## 26  15660         360                                 De 10 a 14 años Y10T14
## 27  15665         360                                 De 15 a 19 años Y15T19
## 28  15661         360                                 De 20 a 24 años Y20T24
## 29  15666         360                                 De 25 a 29 años Y25T29
## 30  15662         360                                 De 30 a 34 años Y30T34
## 31  15658         360                                 De 35 a 39 años Y35T39
## 32  15663         360                                 De 40 a 44 años Y40T44
## 33  15667         360                                 De 45 a 49 años Y45T49
## 34  15648         360                                 De 50 a 54 años Y50T54
## 35  15649         360                                 De 55 a 59 años Y55T59
## 36  15650         360                                 De 60 a 64 años Y60T64
## 37  15651         360                                 De 65 a 69 años Y65T69
## 38  15652         360                                 De 70 a 74 años Y70T74
## 39  15653         360                                 De 75 a 79 años Y75T79
## 40  15654         360                                 De 80 a 84 años Y80T84
## 41  15655         360                                 De 85 a 89 años Y85T89
## 42 284344         357                                   90 y más años Y-GE90
## 43  15656         360                                 De 90 a 94 años Y90T94
## 44  15657         357                                   95 y más años Y-GE95
## 45  22189         260                             Tasa de mortalidad
## 46 311510         260 Promedio de años vividos el último año de vida
## 47 311511         260                               Riesgo de muerte
## 48 311512         260                                Supervivientes
## 49 311513         260                           Defunciones teóricas
## 50 311514         260                         Población estacionaria
## 51 311515         260                                Tiempo por vivir
## 52 284290         260                              Esperanza de vida
```

Vemos que necesitamos filtrar por:

- **ccaa**: Seleccionar todas.
- **sexo**: Total.
- **Edad**: 0 años
- **Tipo de dato**: Esperanza de vida.

```
# Filtrar por esperanza de vida
# "" para indicar que tome todos los valores
```

```
filter = list(ccaa = "", sexo = "total", values = c("0 años", "esperanza de vida"))

# Tabl de mortalidad por año, ccaa, ciudadaes, sexo, edad y funciones.
# Table url: https://www.ine.es/jaxiT3/Tabla.htm?t=27154&L=0
esp <- get_data_table(idTable = 27154, filter = filter, nlast = 1, unnest = TRUE,
                      metacodes = TRUE, tip = "AM", validate = FALSE,verbose = TRUE)
```

```
## - Processing filter: 0% - Processing filter: 12%        - Processing filter: 25%        - Process
## - API URL: https://servicios.ine.es/wstempus/js/ES/DATOS_TABLA/27154?nult=1&tip=AM&ver=3&tv=70%3A
```

```
# Seleccionamos columnas de interés
esp <- subset(esp, select = c("Comunidades.y.Ciudades.Autónomas.Id", "T3_Periodo",
                              "Anyo","Valor"))
```

7.2.1.1 Contornos

Una vez obtenidos los datos que se quieren incluir en el mapa, es necesario **disponer de los contornos geográficos** sobre los cuales se quieren representar dichos datos. En este caso, los contornos de las comunidades autónomas.

El ine dispone acceso de contornos para los limites geográficos en la siguiente web:

Web `https://www.ine.es/wstempus/geojs/ES/CONTORNOS/XXXX`
sustituyendo `XXXX` por el código del ámbito geográfico necesario, siendo estos:

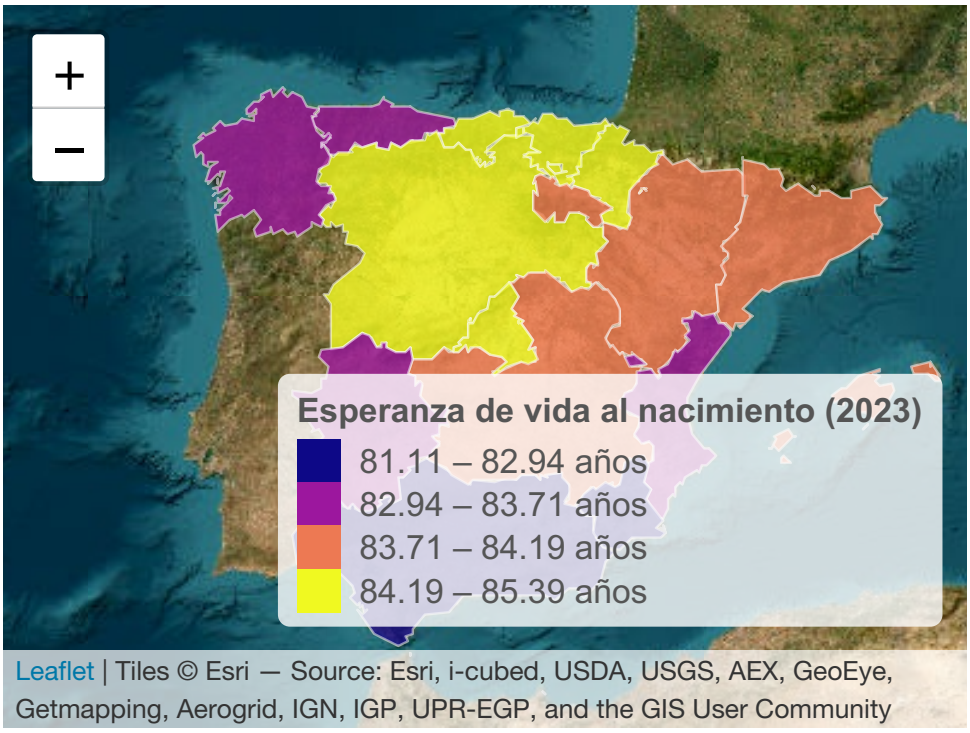
- 349: total nacional
- 70: ccaa
- 115: provincias
- 19: municipios

Para colorear el mapa, se desea que se muestren colores para intervalos de Esperanza de Vida (tomando los cuartiles), para ello será necesario usar la función `colorBin()`. Por tanto, tenemos:

```
# Verificar si los paquetes están instalados
if (!require("leaflet")) install.packages("leaflet")
if (!require("sf")) install.packages("sf")
if (!require("htmltools")) install.packages("htmltools")

# Cargar paquete
library(leaflet)
library(sf)
library(htmltools)

# Contornos de las ccaa
ccaa <- read_sf("https://www.ine.es/wstempus/geojs/ES/CONTORNOS/70")



# join de los contornos y el dataset
ccaa <- merge(ccaa, esp, by.x = "id_region",
              by.y = "Comunidades.y.Ciudades.Autónomas.Id" )

# Creamos colores para cada rango de esperanza de vida
pal <- colorBin("plasma", domain = NULL, bins = c(quantile(ccaa$Valor)))
```

```r
# Labels of the map
labels <- sprintf(
  "<strong>%s</strong><br/> Esperanza de vida al nacimiento: %.2f años ",
  ccaa$nom_region, ccaa$Val
) %>% lapply(htmltools::HTML)

# Create the map
m <- leaflet(ccaa) %>%
  addProviderTiles(providers$Esri.WorldImagery) %>%
  setView(-4, 40, zoom = 5) %>%
  addPolygons(fillOpacity = 0.8,
              fillColor = ~pal(Valor),
              weight = 1,
              color = "white",
              label = labels,
              labelOptions = labelOptions(
                style = list("font-weight" = "normal", padding = "3px 8px"),
                textsize = "15px",
                direction = "auto"
              ),
              highlightOptions = highlightOptions(fillOpacity = 1, bringToFront = TRUE,
                                                  weight = 2, color = "white")
  ) %>%
  addLegend(pal = pal, values = ~Valor, opacity = 1.0, position = "bottomright",
            labFormat = labelFormat(suffix = " años", digits = 2),
            title = sprintf("Esperanza de vida al nacimiento (%s)",
                            esp$Anyo[1]))
```

```r
m
```



7.2.2 Ejemplo salario medio

Ahora vamos a representar en un mapa el salario bruto mensual medio en cada comunidad autónoma para el año 2022. Para ello necesitamos:

1. En la web del ine **buscamos la tabla que contiene los datos deseados**, que tiene por id `13930`. (Véase https://www.ine.es/jaxiT3/Tabla.htm?t=13930&L=0)
2. Usando `ineapir` extraemos los datos usando el servicio API del INE. Ejecutando `get_metadata_table_varval(139` vemos que debemos filtrar por jornada a tiempo completo, todas ccaa, y total decil.

```
# "" para indicar que tome todos los valores
get_metadata_table_varval(13930)
```

```
##        Id Fk_Variable                      Nombre Codigo
## 1   10757         120                       Total
## 2   10758         120  Jornada a tiempo completo
## 3   10759         120   Jornada a tiempo parcial
## 4   16473         349             Total Nacional     00
## 5    8997          70                  Andalucía     01
## 6    8998          70                    Aragón     02
## 7    8999          70    Asturias, Principado de     03
## 8    9000          70             Balears, Illes     04
## 9    9001          70                   Canarias     05
## 10   9002          70                  Cantabria     06
## 11   9003          70           Castilla y León     07
## 12   9004          70       Castilla - La Mancha     08
## 13   9005          70                  Cataluña     09
## 14   9006          70       Comunitat Valenciana     10
## 15   9007          70                 Extremadura     11
## 16   9008          70                     Galicia     12
## 17   9009          70       Madrid, Comunidad de     13
## 18   9010          70          Murcia, Región de     14
## 19   9011          70 Navarra, Comunidad Foral de     15
## 20   9012          70                  País Vasco     16
## 21   9013          70                    Rioja, La     17
## 22   9015          70                        Ceuta     18
## 23   8995          70                      Melilla     19
## 24 298931         684               Total decil
## 25 298932         684                        1
## 26 298933         684                        2
## 27 298934         684                        3
## 28 298935         684                        4
## 29 298936         684                        5
## 30 298937         684                        6
## 31 298938         684                        7
## 32 298939         684                        8
## 33 298940         684                        9
## 34 298941         684                       10
```

Vemos que necesitamos filtrar por:

- **Tipo**: Jornada a tiempo completo
- **ccaa**: Seleccionar todas.
- **Decil**: Total decil

```r
# Filtrar por esperanza de vida


filter2 = list(
  "120" = "10758", # Jornada a tiempo completo
   "70" = "",      # Todas ccaa
  "684" = "298931" # Total decil
  )

# Tabl de mortalidad por año, ccaa, ciudadaes, sexo, edad y funciones.
# Table url: https://www.ine.es/jaxiT3/Tabla.htm?t=27154&L=0
esp2 <- get_data_table(idTable = 13930, filter = filter2, nlast = 1, unnest = TRUE,
                       metacodes = TRUE, tip = "AM", validate = FALSE)

# Seleccionamos columnas de interés
esp2 <- subset(esp2, select = c("Comunidades.y.Ciudades.Autonómas.Id","Anyo","Valor"))
```

7.2.2.1 Contornos

Una vez obtenidos los datos que se quieren incluir en el mapa, es necesario **disponer de los contornos geográficos** sobre los cuales se quieren representar dichos datos. En este caso, los contornos de las comunidades autónomas.

El ine dispone acceso de contornos para los limites geográficos en la siguiente web:

Web `https://www.ine.es/wstempus/geojs/ES/CONTORNOS/XXXX`
sustituyendo `XXXX` por el código del ámbito geográfico necesario, siendo estos:

- 349: total nacional
- 70: ccaa
- 115: provincias
- 19: municipios

En nuestro caso necesitamos el de las comunidades autónomas.

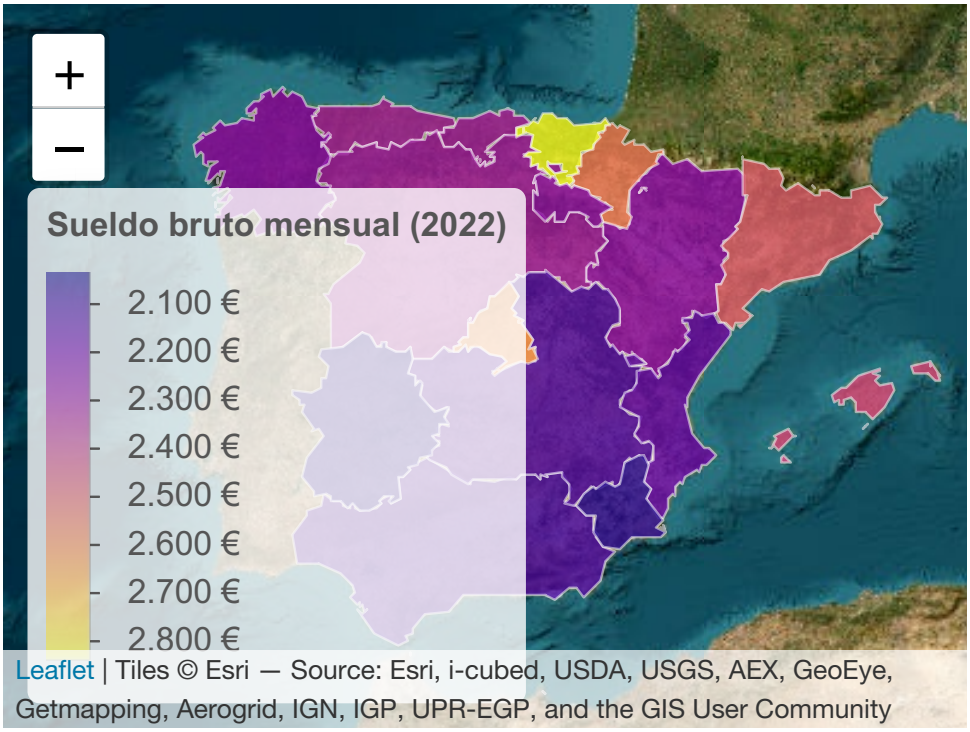
```r
# Contornos de las ccaa
ccaa2 <- read_sf("https://www.ine.es/wstempus/geojs/ES/CONTORNOS/70")
```

Para colorear el mapa, se desea que se muestren colores para intervalos de Esperanza de Vida (tomando los cuartiles), para ello será necesario usar la función `colorBin()`. Por tanto, tenemos:

```r
# join de los contornos y el dataset
ccaa2 <- merge(ccaa2, esp2, by.x = "id_region",
               by.y = "Comunidades.y.Ciudades.Autonómas.Id" )

# Creamos colores para cada rango de esperanza de vida
pal2 <- colorNumeric("plasma",domain= NULL)

# Labels of the map
labels <- sprintf(
  "<strong>%s</strong><br/> Sueldo bruto mensual %s € ",
  ccaa2$nom_region, format(ccaa2$Val, big.mark = ".", decimal.mark = ",", nsmall = 2)
) %>% lapply(htmltools::HTML)
```

```
# Create the map
m2 <- leaflet(ccaa2) %>%
  addProviderTiles(providers$Esri.WorldImagery) %>%
  setView(-4, 40, zoom = 5) %>%
  addPolygons(fillOpacity = 0.8,
              fillColor = ~pal2(Valor),
              weight = 1,
              color = "white",
              label = labels,
              labelOptions = labelOptions(
                style = list("font-weight" = "normal", padding = "3px 8px"),
                textsize = "15px",
                direction = "auto"
              ),
              highlightOptions = highlightOptions(fillOpacity = 1, bringToFront = TRUE,
                                                  weight = 2, color = "white")
              )%>%  addLegend(pal = pal2, values = ~Valor, position = "bottomleft",
            labFormat = labelFormat(suffix = " €", digits = 2,big.mark = "."),
            title = sprintf("Sueldo bruto mensual (%i)", esp2$Anyo[1]))
```

```
m2
```



Anexos

Appendix A

Ineapir

A.1 Introducción

En el ámbito de la investigación y el análisis de datos socioeconómicos, el acceso a información precisa y actualizada es fundamental. **ineapir** es paquete de R diseñado para facilitar el acceso y la manipulación de datos proporcionados por el INE. Este paquete permite a los usuarios interactuar de manera eficiente con la API del INE, simplificando la tarea de obtener datos detallados sobre diversos aspectos de la economía, demografía y sociedad española.

En los siguientes apartados se abordará:

- **Primeros pasos**: Instalación y Configuración de ineapir.
- **Obtención de Datos**: Instrucciones sobre cómo realizar consultas específicas a la API del INE y obtener datos en formatos utilizables.
- **Manipulación y Visualización de Datos**: Técnicas para limpiar, manipular y visualizar los datos obtenidos, incluyendo ejemplos prácticos. Estos se exponen a lo largo de todo el proyecto.

A.1.1 Beneficios de usar ineapir

ineapir no solo facilita el acceso a los datos del INE, sino que también mejora la eficiencia y precisión en el análisis de estos datos. Entre sus principales ventajas se encuentran:

- **Acceso Simplificado**: Permite realizar consultas complejas de manera sencilla y rápida.
- **Integración con R**: Aprovecha las capacidades de R para análisis estadísticos y visualización de datos.
- **Actualización Continua**: Facilita el acceso a los datos más recientes publicados por el INE.
- **Versatilidad**: Puede ser utilizado en diversos campos de estudio, desde la economía y la sociología hasta la demografía y las ciencias políticas.

A.1.2 Público Objetivo

Esta herramienta está dirigida a:

- Investigadores y académicos que trabajan con datos socioeconómicos.
- Profesionales del análisis de datos y la estadística.
- Estudiantes de ciencias sociales y económicas.
- Cualquier persona interesada en el análisis de datos proporcionados por el INE.

A.2 Primeros pasos

Para **instalar** el paquete **inepir** es necesario recurrir al GitHub del INE. Este se encuentra alojado en el siguiente repositorio: https://github.com/es-ine/ineapir.

Para ello, ejecutar la siguiente linea en R, junto a la carga del paquete en la sesión:

```
remotes::install_github("es-ine/ineapir")
library(ineapir)
```

Cheatsheet Existe una chaetsheet resumiendo las funcionalidades del paquete, muy útil para principiantes, https://raw.githubusercontent.com/es-ine/ineapir/main/man/figures/ineapir.pdf

A.3 Obtención de datos

Una vez se ha cargado`ineapir` en el entorno de trabajo, se va a proceder a explicar (vía ejemplos ilustrativos), como obtener los datos deseados [1]. Se explicará desde

[1] Documentación basada en la incluida en el repositorio de origen, https://github.com/es-ine/ineapir. Para casos más complejos/ menos frecuentes, consultar dichos repositorios.

Bibliography

Crespo, D. (2024). *Ineapir: Obtaining data published by the national statistics institute* [R package version 0.0.0.9000, https://es-ine.github.io/ineapir/]. https://github.com/es-ine/ineapir

Xie, Y. (2017). *Authoring books and technical documents with R markdown* (1st ed.). Chapman & Hall. https://bookdown.org/yihui/bookdown/