Gráficos en R

Instituto Nacional de Estadística (INE)

David Pérez Ros

18/10/24

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.

- **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.

[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)
```

```
# 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
```

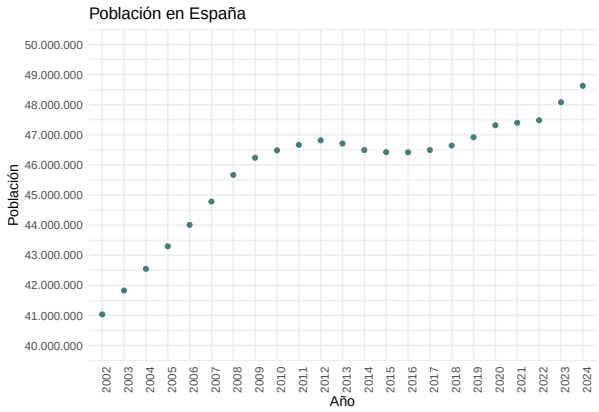


Figure 2.1: Evolución de la población a lo largo de los años.

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:

```
# 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

```
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:

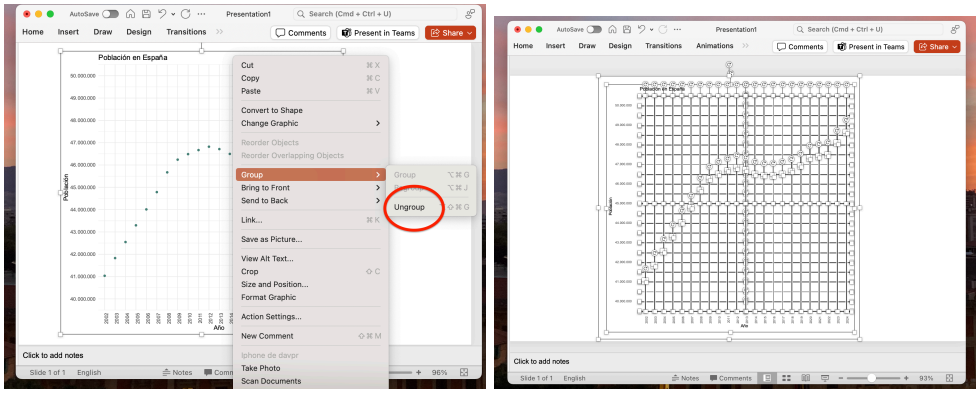


Figure 2.2: Pegado del gráfico como imagen vectorial

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

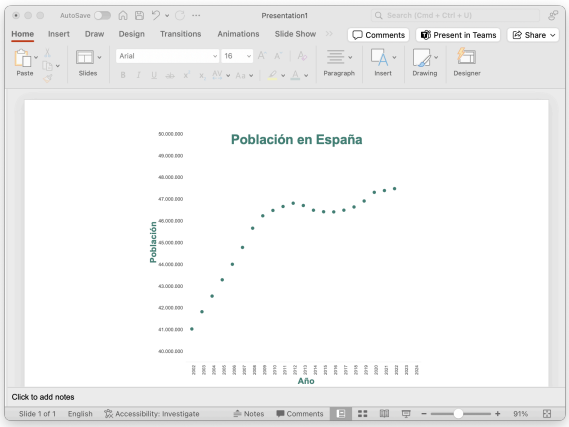


Figure 2.3: Modificación del gráfico en Power Point

Paquetes principales

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) ")
```

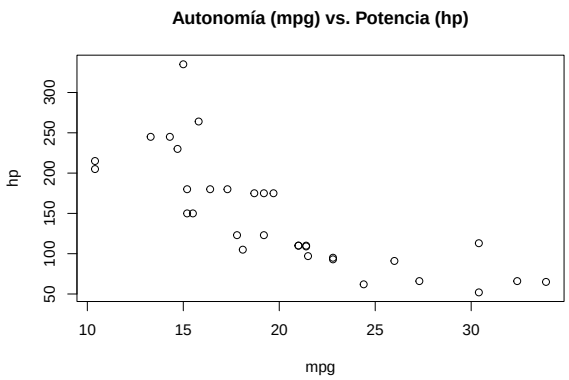


Figure 3.1: Gráfico de dispersión de potencia y auonomía de los automóviles.

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)))
)
```

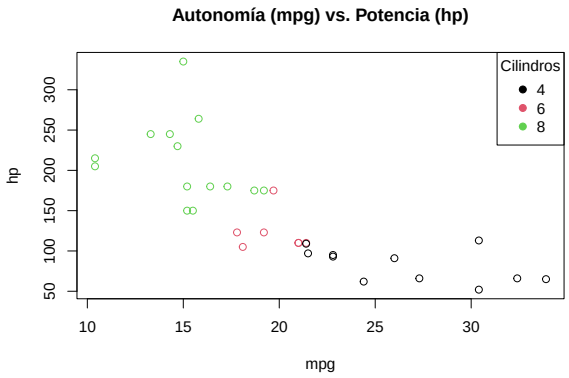


Figure 3.2: Gráfico de dispersión condicionado por el número de cilindros.

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`

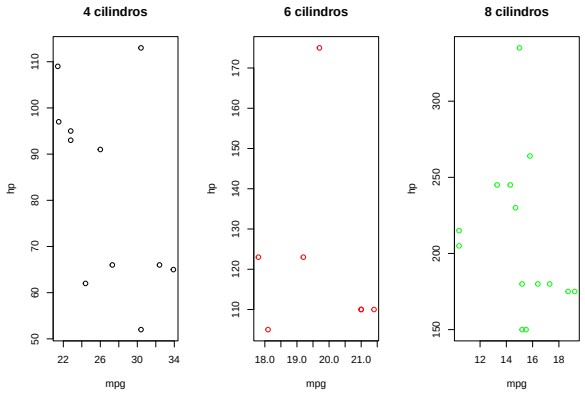


Figure 3.3: Gráfico de dispersión condicionado por el número de cilindros (separados).

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")
```

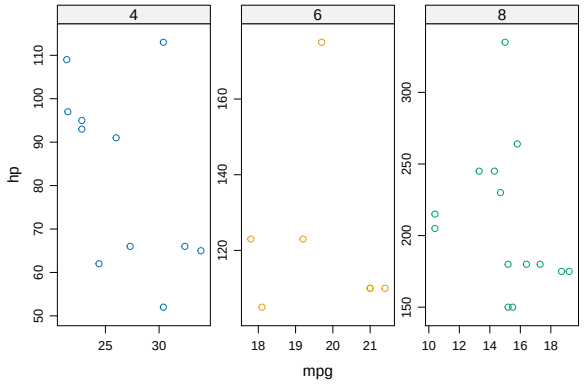


Figure 3.4: Gráfico de dispersión con lattice condicionado por el número de cilindros.

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

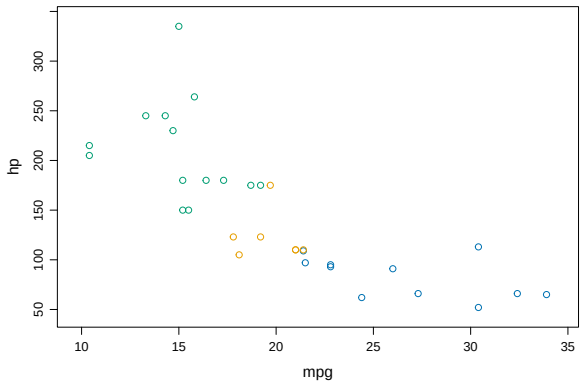


Figure 3.5: Gráfico de dispersión con lattice condicionado por el número de cilindros.

  - **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)
```

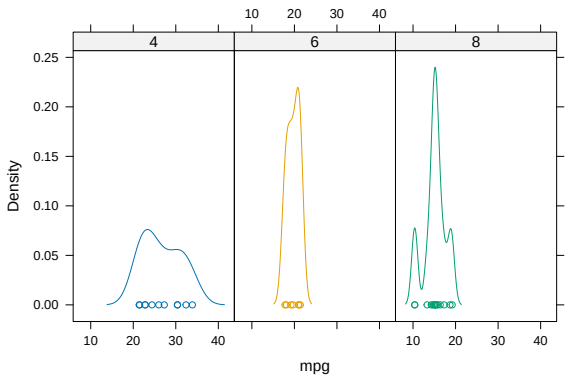


Figure 3.6: Densidad de la autonomía condicionada al número de cilindros (en gráficos distintos).

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

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

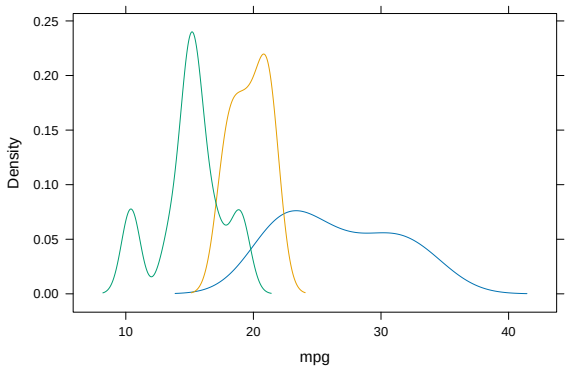


Figure 3.7: Densidad de la autonomía condicionada al número de cilindros.

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* .

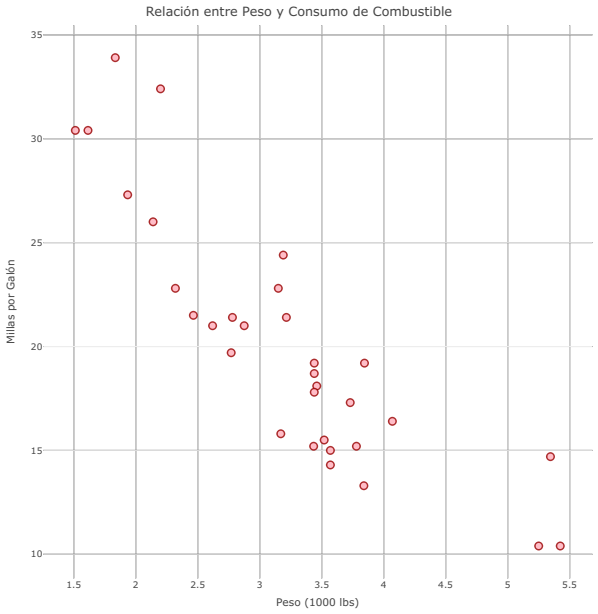
4.3 Ejemplo de Uso

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("dplyr")
if (!require("webshot")) install.packages("webshot")
# Cargar paquete
library(dplyr)
library(webshot)

# 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 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.

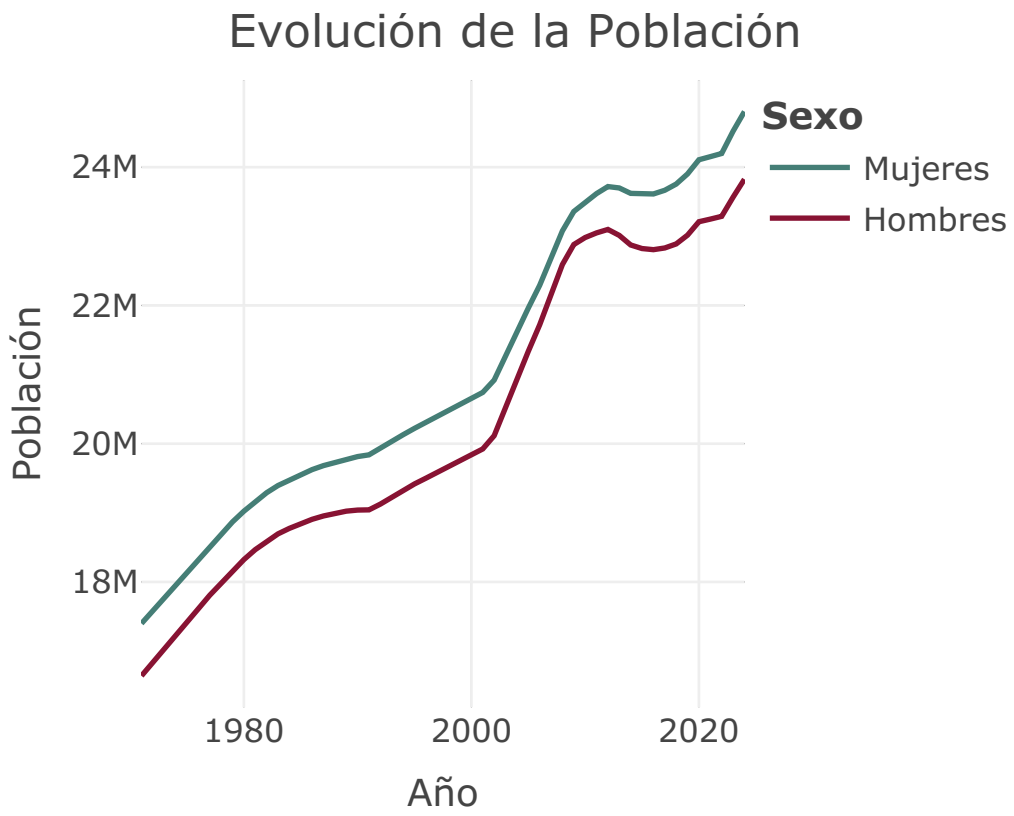
Características Clave de Plotly para Series Temporales

- **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.
- **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)).

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

library("ineapir")
# 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"
  )

# Mostrar el gráfico
fig
```



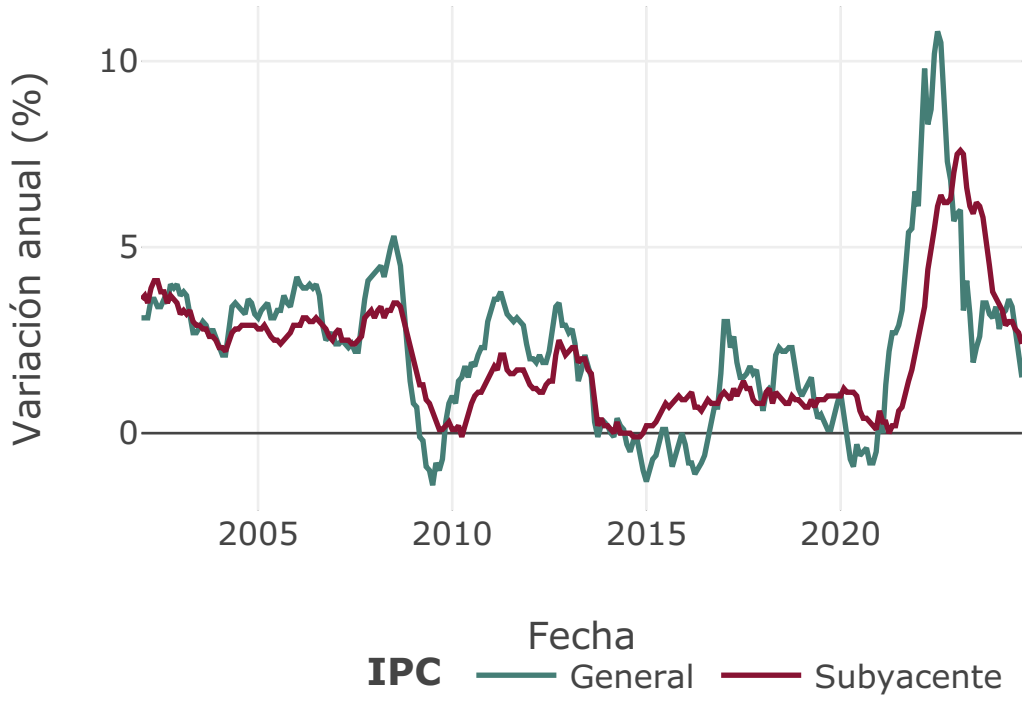
Ahora vamos a mostrar la evolución del IPC y el IPC Subyacente a lo largo de los años. Para ello, lo primero buscamos en la web del INE en que tabla se encuentran dichos datos, siendo esta la "50902". A continuación, con `get_metadata_table_varval(50902)` vemos que variables y valores hay que tomar para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

```
# get_metadata_table_varval(50902)
# get_metadata_variables()

# Filter cpi overall index
# Todos los approaches 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")



# Request data of cpi
# 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
)




# get_metadata_table_varval(50907)
# get_metadata_variables()

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)

fig
```



4.3.3 Grafico de barras

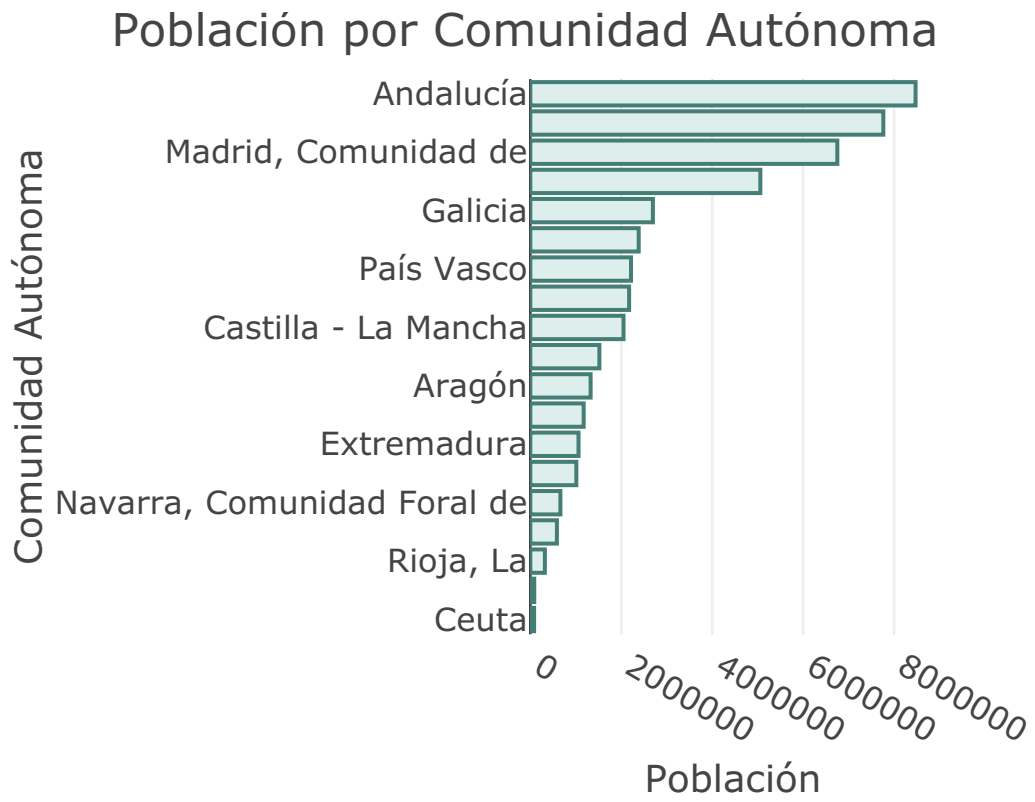
Los gráficos de barras en Plotly permiten comparar fácilmente valores entre diferentes categorías. Pueden ser verticales u horizontales, y se pueden personalizar con colores específicos para cada barra. También soportan la agrupación de barras (bar charts) y la apilación (stacked bar charts). La orientación, 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.

```
# get_metadata_table_varval(2915)
# 34(Tamaño municipios)=20275(Total)
# nlast=1 nos devuelve el último año sólo

filter <- list("34" = "20275")
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
  )
```

```
# Mostrar el gráfico
fig
```



4.3.4 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 [1].

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

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

```
##        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)
```

```r
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")
```

```r
#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)
```

```r
# 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)
```

```r
# 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)
  )
fig
```

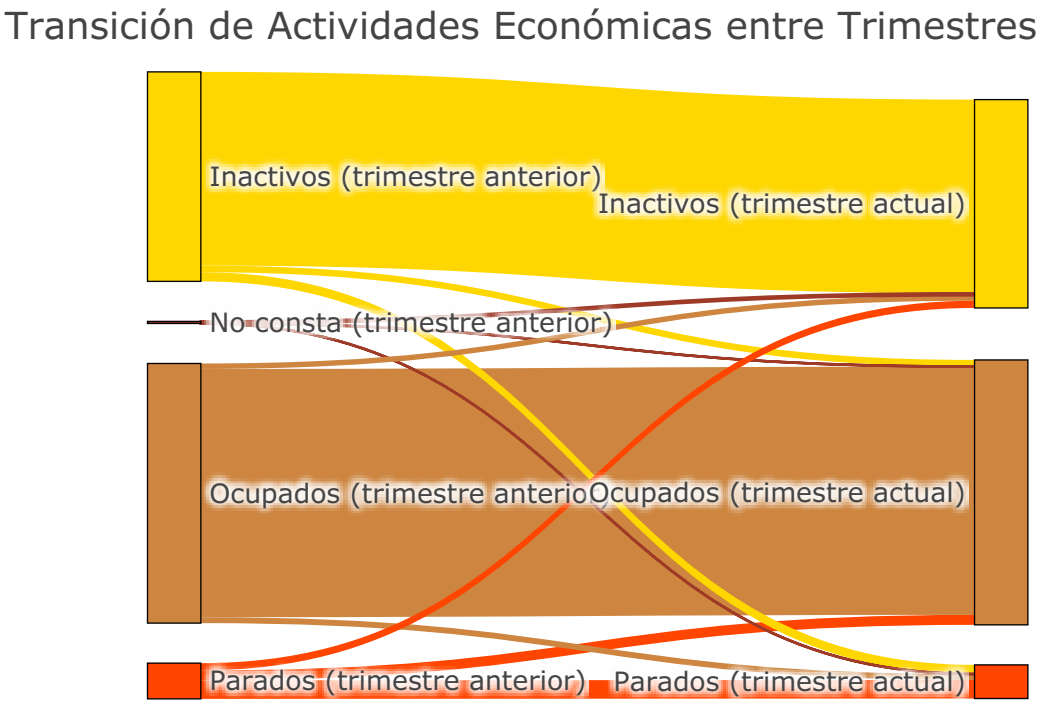


Figure 4.1: Flujos de personas en el mercado laboral en España 2024 T2

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/

Capítulo 5

sdf

dfasxzcvghbjklñxvzcvb

bvnm hvjbk

```
2+2
```

```
## [1] 4
```

Capítulo 6

jk

fdas

Appendix A

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.

A.1 General

A.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.

A.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.

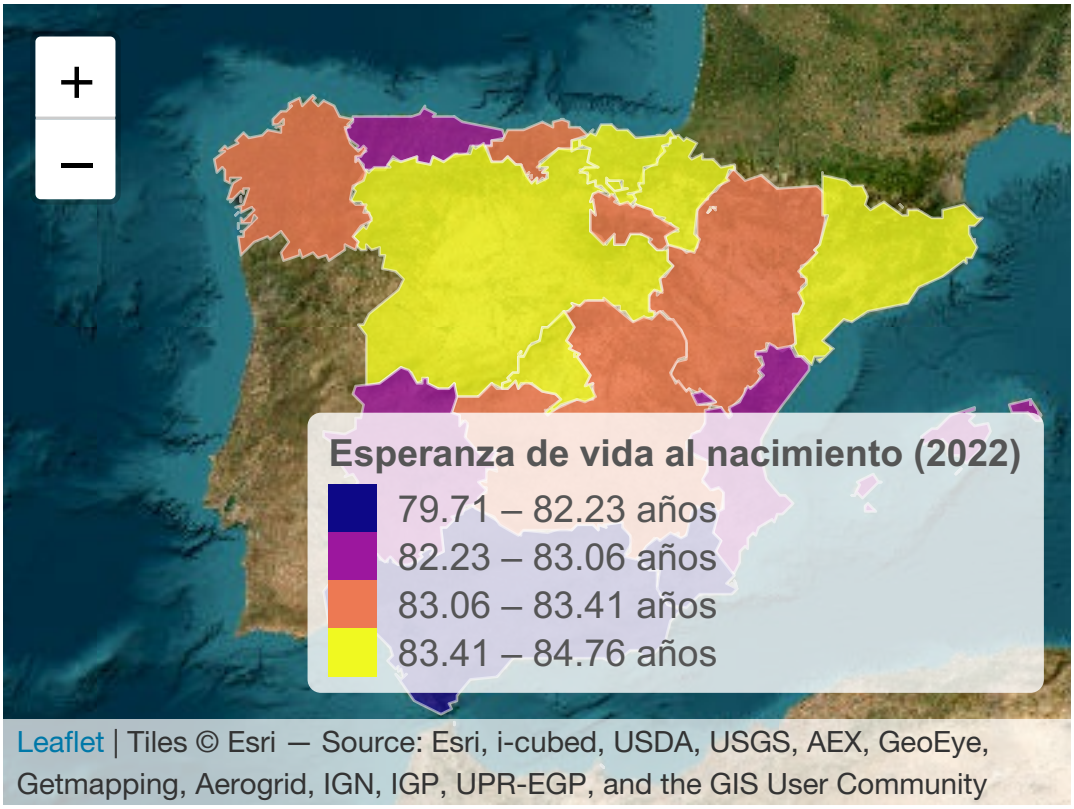
2. **Marcadores**:
   - **Marcadores Simples**: Puntos que indican ubicaciones específicas en el mapa.
   - **Marcadores Agrupados**: Agrupan varios marcadores en una sola vista para evitar el desorden visual (clustering).
3. **Polígonos y Líneas**:
   - **Polígonos**: Representan áreas geográficas específicas (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.

A.2 Mapas

```r
# Verificar si está instalado
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)

# Get the boundaries of the autonomous communities
ccaa <- read_sf("https://www.ine.es/wstempus/geojs/ES/CONTORNOS/70")

# Filter for life expectancy
filter = list(ccaa = "", sexo = "total", values = c("0 años", "esperanza de vida"))

# Table: Mortality tables by year, autonomous communities and cities, sex, age and functions.
# 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)
```

```r
# Select a set on columns from data
esp <- subset(esp, select = c("Comunidades.y.Ciudades.Autónomas.Id", "T3_Periodo",
                              "Anyo","Valor"))

# Join boundaries information with data information
ccaa <- merge(ccaa, esp, by.x = "id_region",
              by.y = "Comunidades.y.Ciudades.Autónomas.Id" )

# Create the palette of the legend
pal <- colorBin("plasma", domain = NULL, bins = c(quantile(ccaa$Valor)))

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


+
−
Esperanza de vida al nacimiento (2022)
79.71 – 82.23 años
82.23 – 83.06 años
83.06 – 83.41 años
83.41 – 84.76 años
Leaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

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/