Introducción a R con datos del CIS

R es un lenguaje de programación orientado a la estadística lanzado por Robert Gentleman y Ross Ihaka en 1994. La principal ventaja que tiene frente a otros paquetes estadísticos como SPSS, SAS o STATA es que es **gratuito y de código abierto**. También está integrado con múltiples plataformas y se puede trabajar con él en dispositivos con sistemas operativos distintos. R se orienta a la **reproducibilidad** al trabajar con scripts de código (salvo que usemos **Rcommander**, una buena opción para aprender con una interfaz de menús). Esto permite que un script se pueda ejecutar por varios usuarios llegando al mismo resultado. Pero la gran ventaja de R está en su **comunidad**, que desarrolla paquetes, ofrece ayuda para cualquier problema y organiza eventos para difundir las posibilidades de este lenguaje de programación.

Además de estas ventajas generales, **R para las ciencias sociales** permite automatizar procesos que pueden resultar laboriosos de realizarse manualmente, organizar y tener un registro replicable de todos los procedimientos realizados durante una investigación y acceder a múltiples paquetes. Estos paquetes van desde el análisis estadístico tradicional hasta otras aplicaciones más innovadoras como la descarga de tweets o el *web-scrapping*.

Cuando buscamos cursos para aprender R, a menudo nos topamos con que los ejemplos siempre utilizan datasets de coches o vuelos. El objetivo de esta serie de posts es acercar R a profesionales y estudiantes del ámbito de las ciencias sociales. Por ello, utilizaremos datos provenientes de la encuesta postelectoral de las generales de 2019 o la serie temporal de ubicación ideológica del Partido Popular desde que el Centro de Investigaciones Sociológicas lo empezase a preguntar en enero de 1989.

Este proyecto no deja de ser una breve introducción, por lo que no aprenderás todas las funcionalidades de R. Es, más bien, una pequeña guía para orientar el aprendizaje, que deberá ser completada con otros cursos, búsquedas en Google y Stackoverflow y con la propia práctica. Recomiendo enormemente que descargues un conjunto de datos de tu interés y vayas replicando lo que voy haciendo aquí.

Si partes de cero, Internet está lleno de recursos para aprender lo básico. Algunas de mis recomendaciones son:

- El canal de Youtube de R para muy principiantes.
- El curso de Estadística para las ciencias sociales con R de David Sulmont.
- El manual de Emmanuel Paradis.
- El paquete swirl.
- Cualquier curso introductorio que encuentres con una búsqueda en Google, aunque recuerda que para aprender lo básico de R no es necesario gastar mucho dinero.

También recomiendo el libro de Jesús Bouso El paquete estadístico R, de la colección Cuadernos Metodológicos del CIS.

No obstante, si no tienes el tiempo necesario para hacer un curso de introducción (aunque es casi imprescindible), el mínimo para seguir esta serie se resume en que:

- El *software* más común para programar con R es RStudio. Este programa contiene, entre otros y en una interfaz visual amigable, la consola de R y una ventana para escribir *scripts* (que son una especie de documentos donde se escribe el código que después se ejecutará).
- El símbolo <- crea un objeto y le asigna un valor, que puede ir desde un simple número hasta todo un conjunto de datos (el típico fichero de microdatos en .sav del CIS) o una sucesión de comandos que crea un gráfico.
- Los comandos ejecutan acciones determinadas por lo que se escribe entre los paréntesis que la acompañan.
- Si escribes código y no pones <-, no lo asignarás a un objeto y no se guardará dentro del entorno de cara a las siguientes operaciones.

- El símbolo %>% se llama **pipe**, proviene del paquete **magrittR** y sirve para concatenar código de cara a facilitar su programación y lectura.

Estos mínimos pueden sonar abstractos si nunca has programado y extremadamente fáciles si ya tienes una base de R. Si estás en el primer caso, conforme la serie se vaya desarrollando podrás verlo de forma más práctica. Si estás en el segundo, espero tratar algún contenido de tu interés que resulte útil. En el próximo post comenzaremos con algo de visualización con el famoso paquete **ggplot2**.

En este post vamos a comenzar a explorar las posibilidades que ofrece R. En el proceso de análisis de datos, la visualización es uno de los últimos pasos a llevar a cabo. Sin embargo, aquí vamos a comenzar a visualizar inmediatamente después de la importación de los microdatos. Esta decisión se debe a que un principiante verá de forma más clara los efectos de los cambios de código sobre un gráfico.

Importación de microdatos

En primer lugar importamos los microdatos de la encuesta postelectoral de las elecciones generales de 2019 (estudio número 3248), descargados en formato .sav del banco de datos del CIS. Utilizaremos la librería **haven**, que ya viene integrada en RStudio.

```
library(haven)
dataset <- read_sav("C:/Users/jlrsd/OneDrive/Trabajando con datos del CIS en R/data/3248.sav")
```

Haven importa los valores con sus códigos (1,2,3,4) y etiquetas. Para tener otro dataset que muestre solo las etiquetas (Mucho, bastante, poco, nada) tenemos que aplicarle la función as_factor.

```
datasetconetiquetas <- as_factor(dataset)
```

Para importar únicamente los códigos utilizaríamos **val_labels** del paquete **labelled**.

```
#install.packages("labelled")
library(labelled)
val_labels(dataset) <- NULL
```

Tidyverse y visualización de datos con ggplot2

Una vez importados los datos vamos a conocer las posibilidades de visualización de R a través de **ggplot2**. En muchos cursos de R se comienza por la visualización sin respetar el orden lineal del proceso de investigación: extracción de información, procesamiento, análisis y visualización. El motivo de empezar visualizando es que el usuario principiante puede ver de forma más tangible el resultado de lo que está modificando en su código.

Este paquete forma parte del **Tidyverse**, una colección de paquetes que amplían las funcionalidades de R para ciencia de datos. Su creador es Hadley Wickham, todo un referente en la comunidad de R.

Los paquetes más importantes de Tidyverse son:

- **ggplot2** para visualización.
- **dplyr** para manipulación de datos.
- **tidyr** para manipulación de dataframes.
- **readr** para importar datos.
- **purrr** para programación con funciones

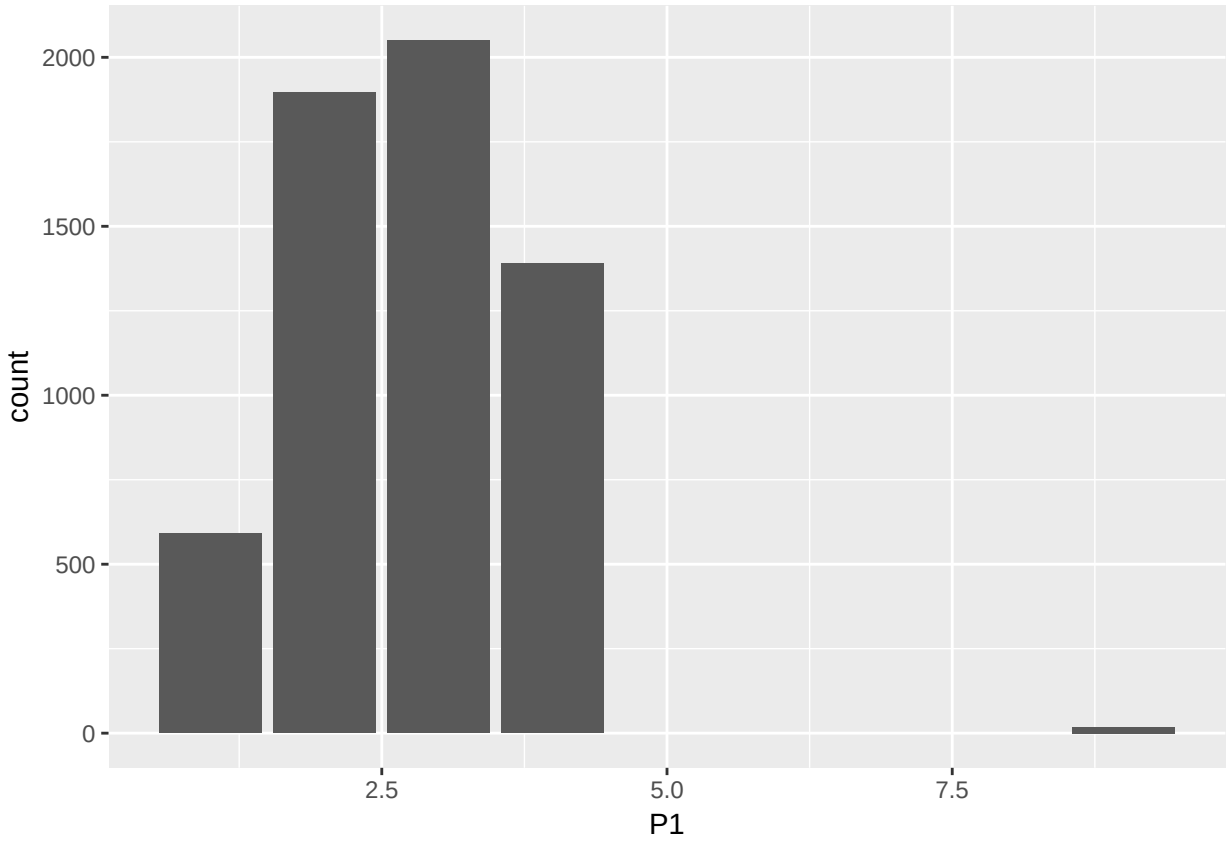
- **tibble** para trabajar con estructuras de datos alternativas a los dataframes.
- **stringr** para trabajar con variables de tipo cadena.
- **forcats** para trabajar con variables de tipo factor.

En primer lugar instalamos y activamos el paquete que aglutina la colección Tidyverse. También iniciamos el paquete scales, que se utiliza para personalizar las escalas de los gráficos. Si es la primera vez que vas a utilizar estos paquetes tendrías que quitar la almohadilla (#) antes de *install.packages*. La almohadilla se utiliza para que esa parte de código no se ejecute, por lo que es útil cuando queremos hacer comentarios o marcar un proceso ya realizado, como en este caso.

```
#install.packages("tidyverse")
library(tidyverse)
#install.packages("scales")
library(scales)
```

En primero lugar, vamos a realizar un gráfico de barras muy básico para ver el grado de interés por la política (P1) de la muestra.

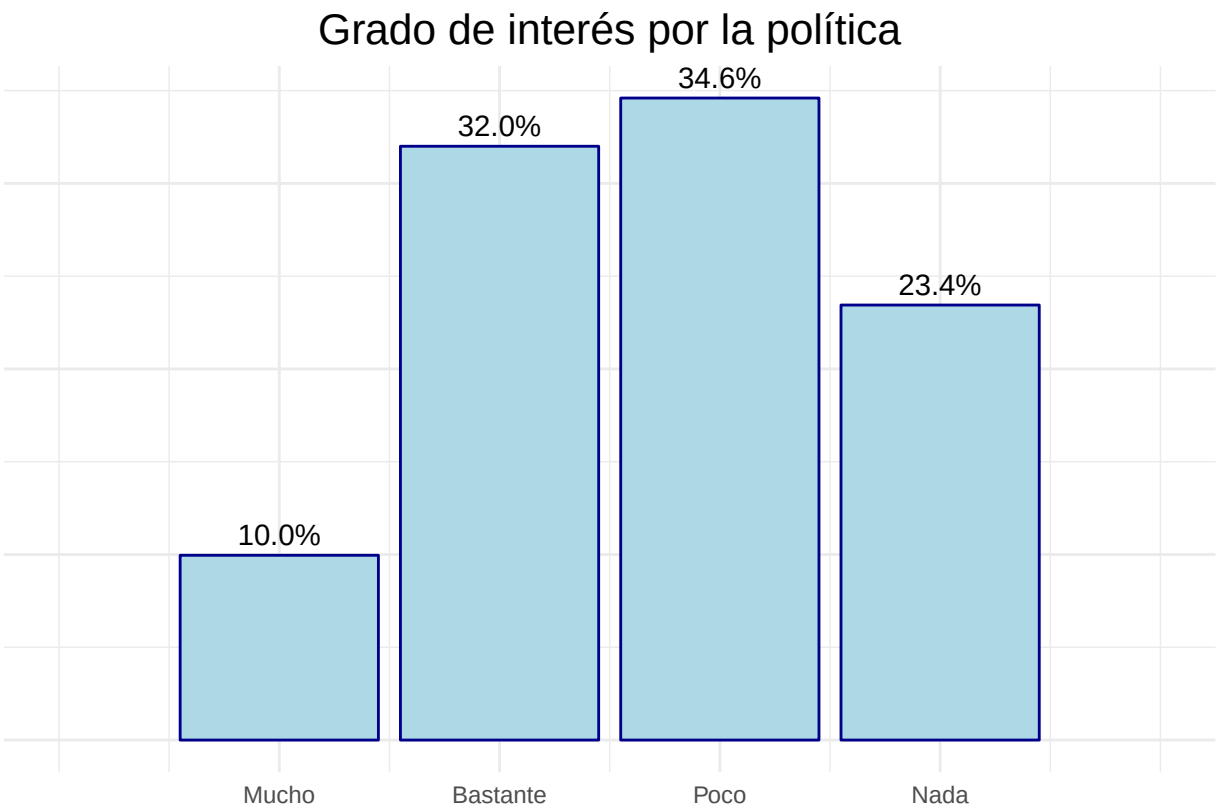
```
ggplot(data = dataset) +
  geom_bar(mapping = aes(x = P1))
```



De manera similar a Photoshop, ggplot2 trabaja por capas. Por tanto, al gráfico anterior le podemos añadir cuantas capas queramos de cara a mejorar la visualización de aquello que queremos comunicar.

Incluyendo algunas líneas al gráfico anterior lo podemos presentar mucho más mejorado

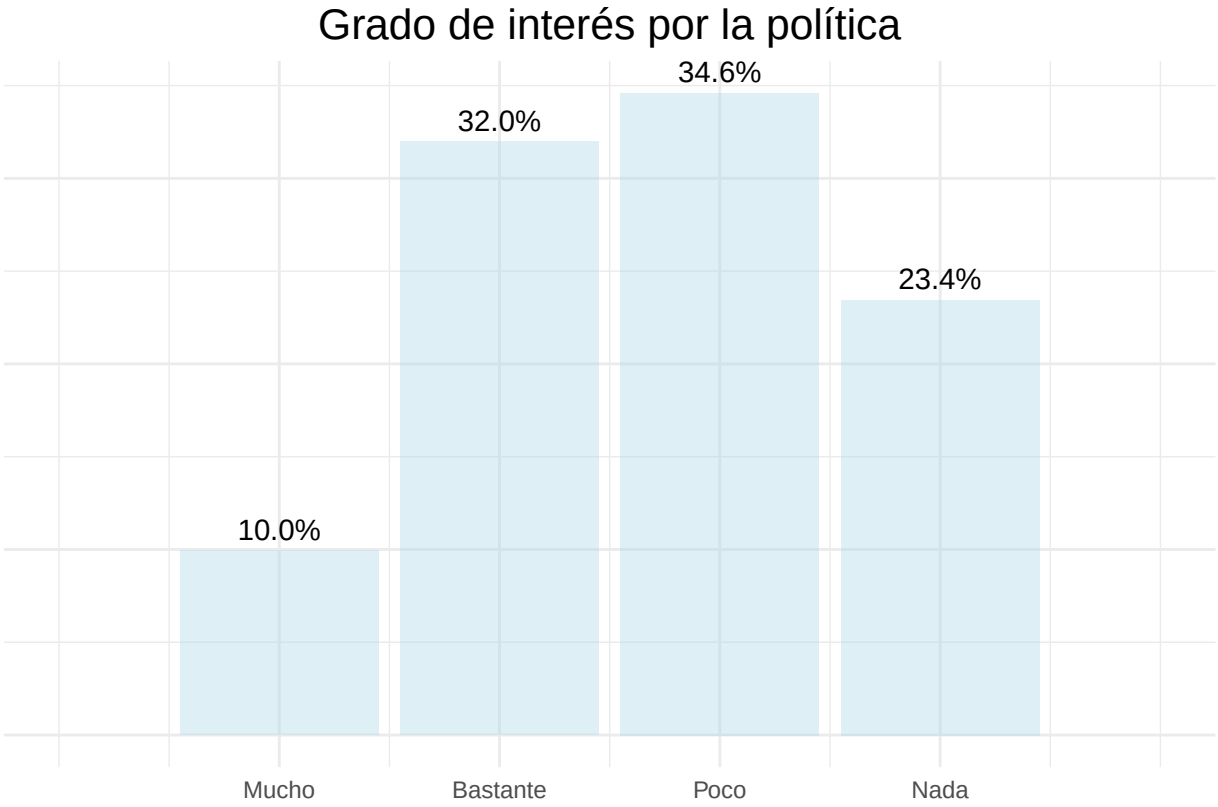
```
ggplot(data = dataset, aes(x = P1)) +
  geom_bar(aes(y = ..prop.., group = 1), fill = "lightblue", color = "darkblue") +
  labs(title = "Grado de interés por la política", caption = "Elaboración propia. Fuente: Encuesta poste
  theme_minimal() +
  theme(plot.title = element_text(size = 16, hjust = 0.5), axis.title.x = element_blank(), axis.title.y
  scale_x_continuous(limits = c(0, 5), breaks = c(1,2,3,4), labels = c("Mucho", "Bastante", "Poco", "Na
  scale_y_continuous(labels = percent_format()) +
  geom_text(aes(label = percent(..prop..), y= ..prop..), stat= "count", vjust = -.5)
```



Elaboración propia. Fuente: Encuesta postelectoral del CIS (2019)

Al trabajar por capas, por ejemplo podemos dejar al fondo el gráfico de barras y poner uno de línea encima. Las posibilidades con ggplot2 son casi infinitas y no hay más que informarse, practicar y empaparse de lo que los mejores comparten, como este *cookbook* del equipo de datos de la BBC.

```
ggplot(data = dataset, aes(x = P1)) +
  geom_bar(aes(y = ..prop.., group = 1), fill = "lightblue", alpha = 0.4) +
  labs(title = "Grado de interés por la política", caption = "Elaboración propia. Fuente: Encuesta poste
  theme_minimal() +
  theme(plot.title = element_text(size = 16, hjust = 0.5), axis.title.x = element_blank(), axis.title.y
  scale_x_continuous(limits = c(0, 5), breaks = c(1,2,3,4), labels = c("Mucho", "Bastante", "Poco", "Na
  scale_y_continuous(labels = percent_format()) +
  geom_text(aes(label = percent(..prop..), y= ..prop..), stat= "count", vjust = -.5)
```



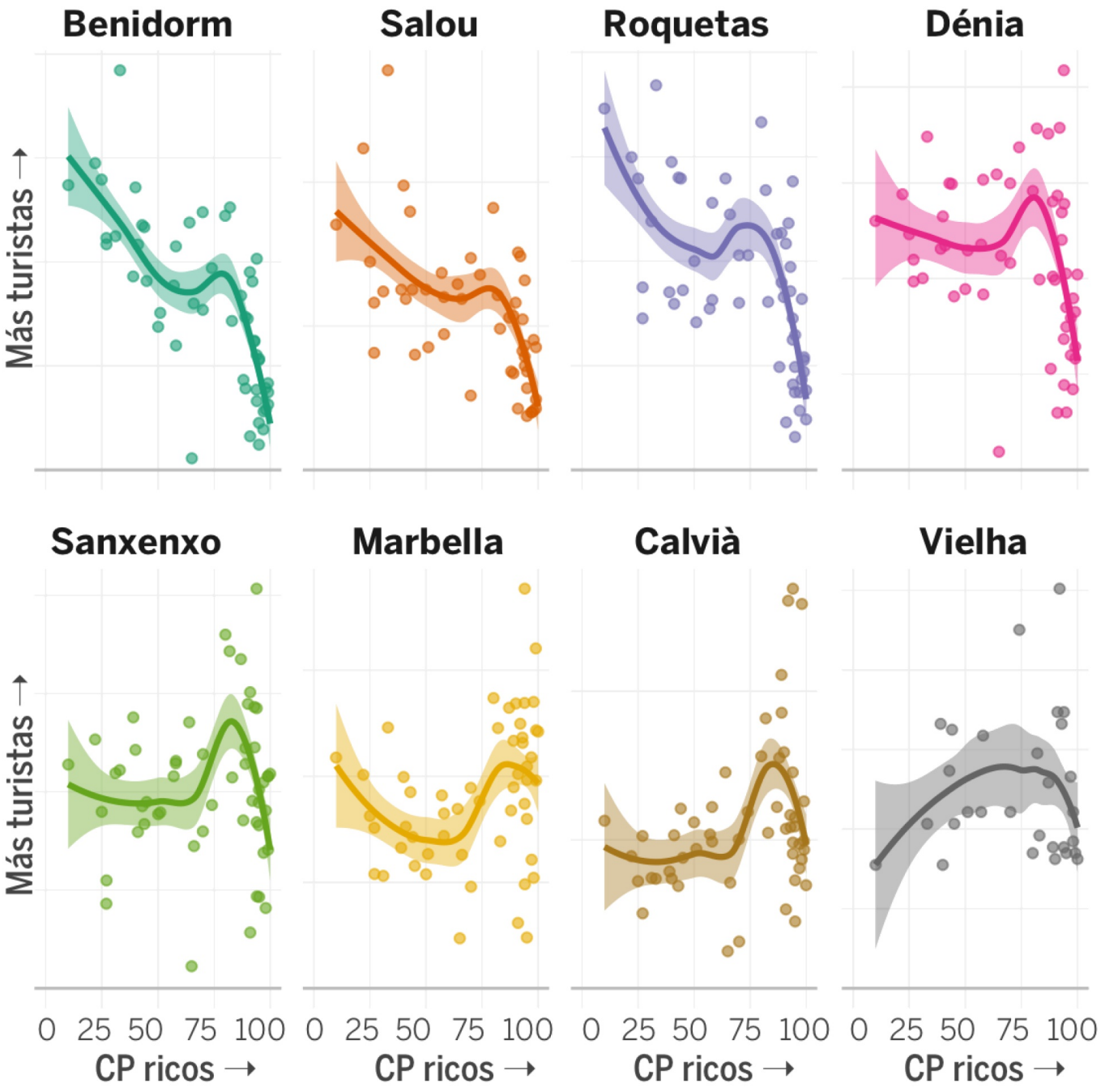
Elaboración propia. Fuente: Encuesta postelectoral del CIS (2019)

Algunos ejemplos de lo que se puede llegar a hacer

Con un poco de práctica y habilidad se pueden crear gráficos de calidad como este de Kiko Llaneras.

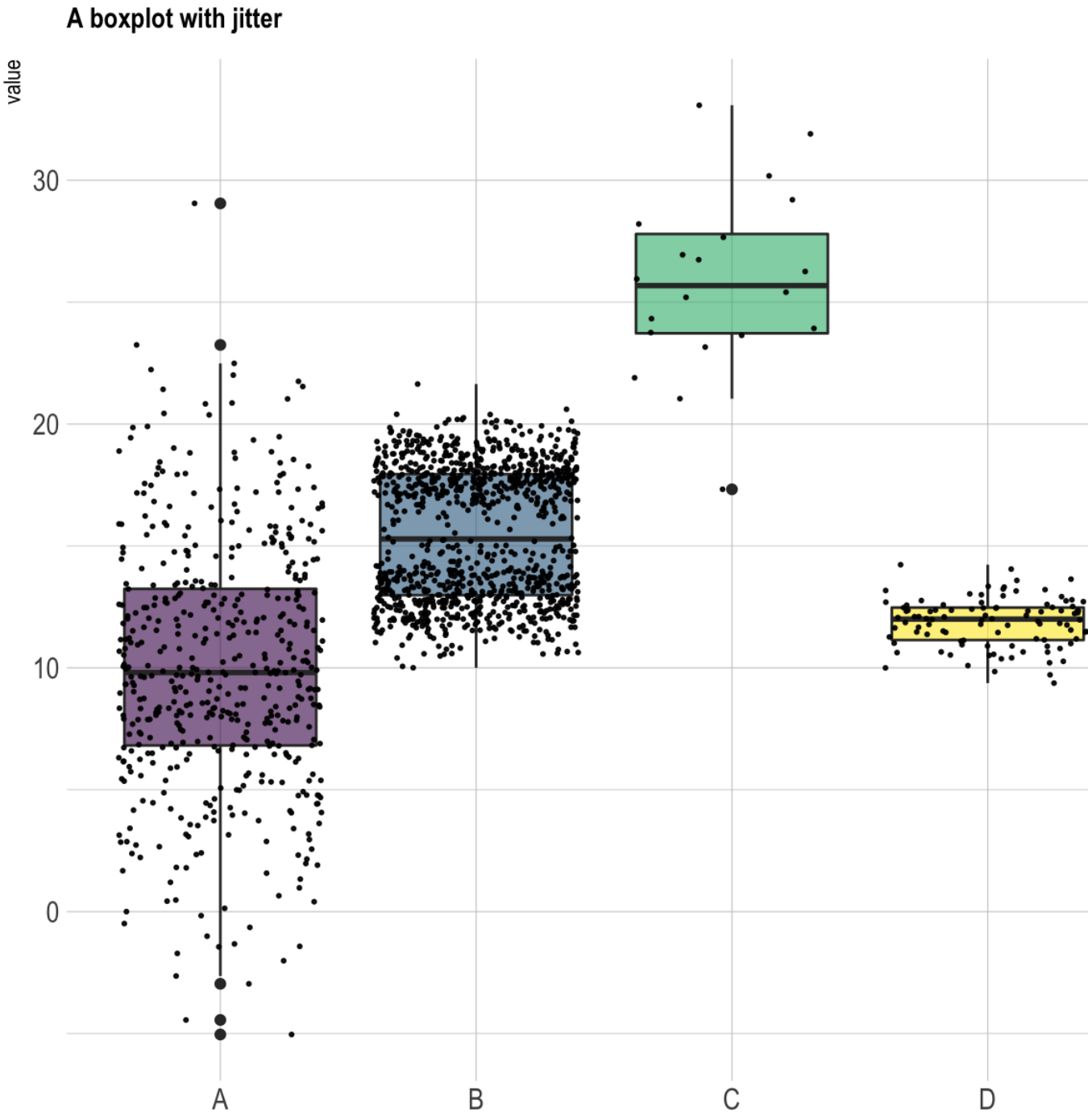
Desde Madrid

Cada punto son turistas del código postal (CP) al destino.
La altura son viajeros (%) y el eje x la renta del CP (centil)



Fuente: Ministerio de Interior / EL PAÍS

O este boxplot de The R Graph Gallery, una web útil para obtener plantillas de gráficos y reutilizarlas para nuestras propias visualizaciones.

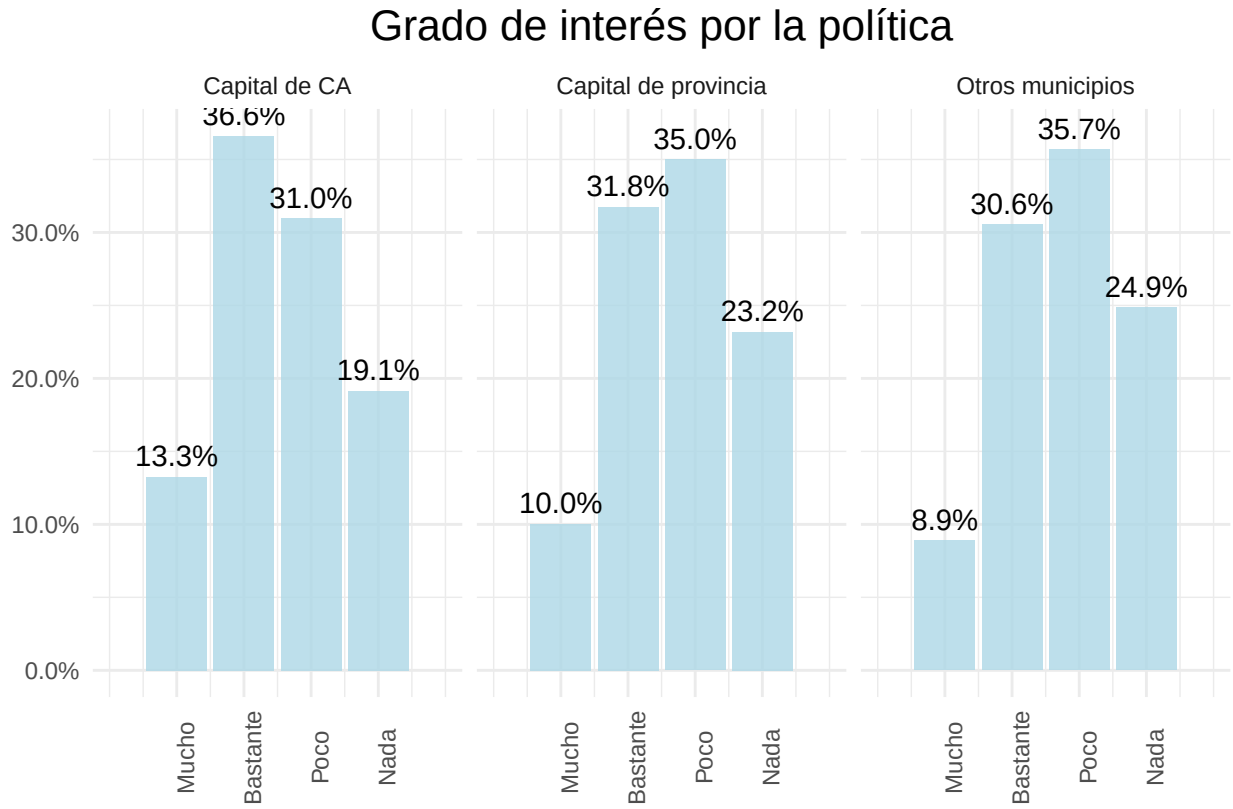


Facets

Otra funcionalidad interesante de ggplot2 son los **facets**, que permiten presentar un mismo gráfico varias veces en función de otra variable. En este caso, mostramos el interés en la política para capitales de comunidad autónoma, capitales de provincia y otros municipios.

```
labels <- c(
                    `1` = "Capital de CA",
                    `2` = "Capital de provincia",
                    `3` = "Otros municipios"
                    )

ggplot(data = dataset, aes(x = P1)) +
  geom_bar(aes(y = ..prop..), fill = "lightblue", alpha = 0.8) +
  facet_wrap(~CAPITAL, labeller = as_labeller(labels)) +
  labs(title = "Grado de interés por la política", caption = "Elaboración propia. Fuente: Encuesta poste
  theme_minimal() +
  theme(plot.title = element_text(size = 16, hjust = 0.5), axis.title.x = element_blank(), axis.title.y
  scale_x_continuous(limits = c(0, 5), breaks = c(1,2,3,4), labels = c("Mucho", "Bastante", "Poco", "Na
  scale_y_continuous(labels = percent_format()) +
  geom_text(aes(label = percent(..prop..), y= ..prop..), stat= "count", vjust = -.5)
```



Series temporales

La postelectoral del CIS proporciona datos en un momento temporal dado. Sin embargo, las series temporales de esta institución permiten analizar una misma variable longitudinalmente. Para mostrar otros tipos de gráficos con ggplot2 vamos a cargar la serie temporal de percepción de la ubicación ideológica del Partido Popular.

```
library(readxl)
Ubicacion_PP <- read_excel("C:/Users/jlrsd/OneDrive/Trabajando con datos del CIS en R/data/UbicacionPP.x
```

Para simplificar el gráfico, primero calculamos nuevas variables para agregar los posicionamientos de 1 a 4 como izquierda, 5 y 6 como centro y de 7 a 10 como derecha. Después, redondeamos a un decimal. También se define la variable Año como fecha.

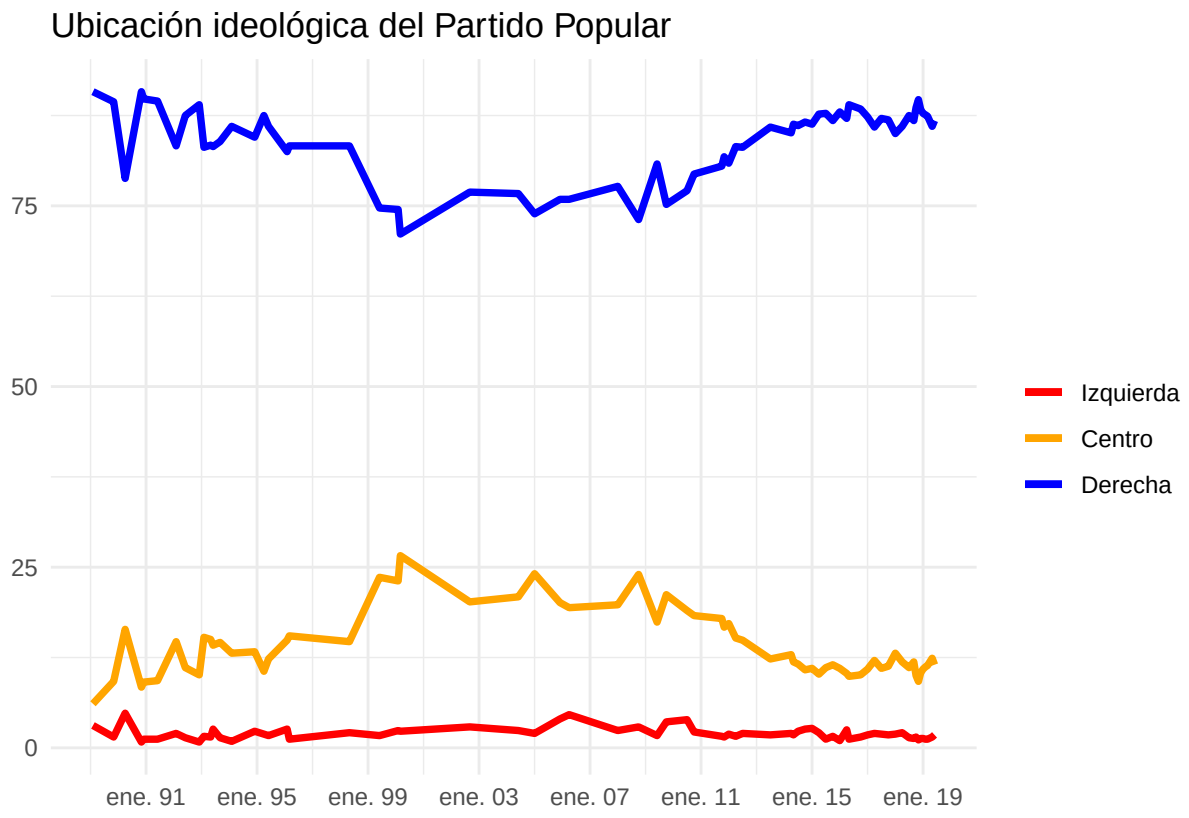
```
Ubicacion_PP <- Ubicacion_PP %>%
  mutate(izquierda = (a + b)/(a+b+c+d+e)*100) %>%
  mutate(centro = c/(a+b+c+d+e)*100) %>%
  mutate(derecha = (d + e)/(a+b+c+d+e)*100)

Ubicacion_PP$izquierda <- round(Ubicacion_PP$izquierda, digits = 1)
Ubicacion_PP$centro <- round(Ubicacion_PP$centro, digits = 1)
Ubicacion_PP$derecha <- round(Ubicacion_PP$derecha, digits = 1)

Ubicacion_PP$Año <- as.Date(Ubicacion_PP$Año)
```

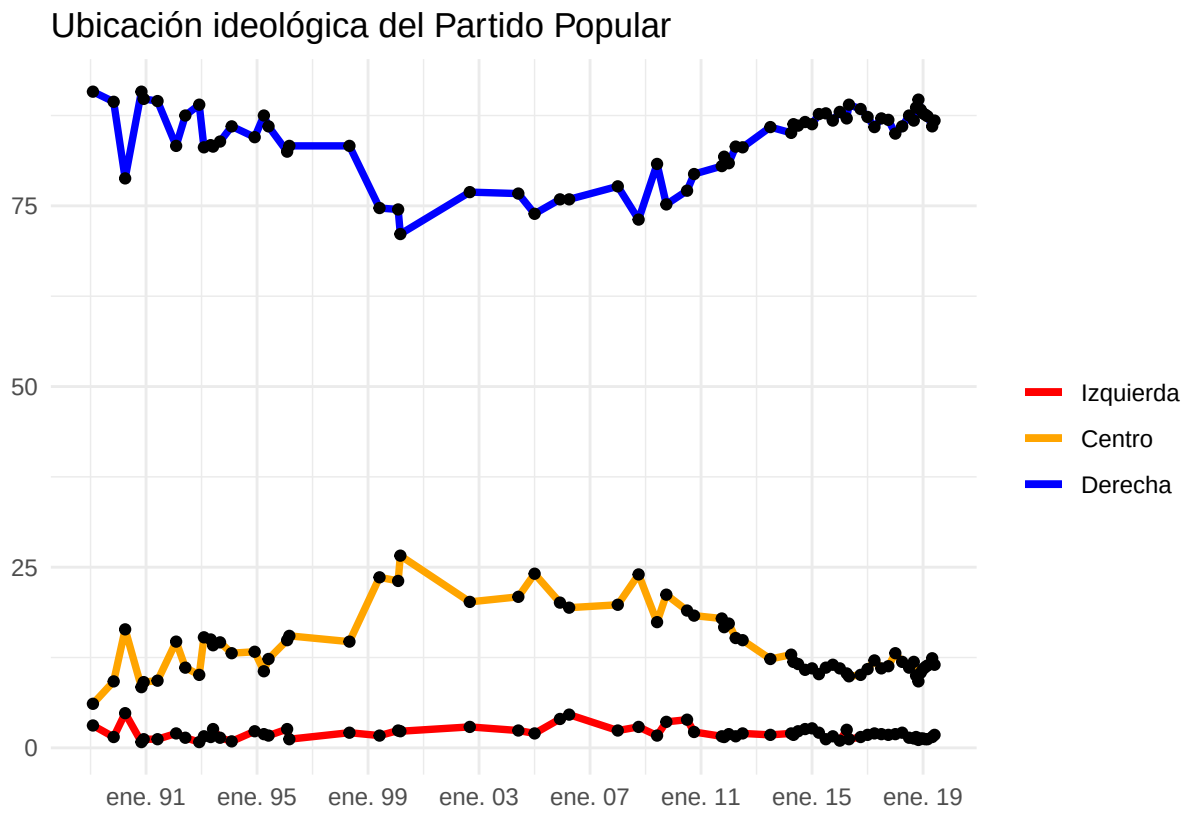
Con ggplot se visualiza la serie temporal con la geometría **geom_line**.

```
Ubicacion_PP %>%
  ggplot() +
    geom_line(mapping = aes(x = Año, y = izquierda, color = "Izquierda"), size = 1.3) +
    geom_line(mapping = aes(x = Año, y = centro, color = "Centro"), size = 1.3) +
    geom_line(mapping = aes(x = Año, y = derecha, color = "Derecha"), size = 1.3) +
    scale_colour_manual("",
                        breaks = c("Izquierda", "Centro", "Derecha"),
                        values = c("orange", "blue", "red")) +
    scale_x_date("", breaks = "4 year", date_labels = "%b %y") +
    scale_y_continuous("") +
    ggtitle("Ubicación ideológica del Partido Popular")+
    theme_minimal()
```



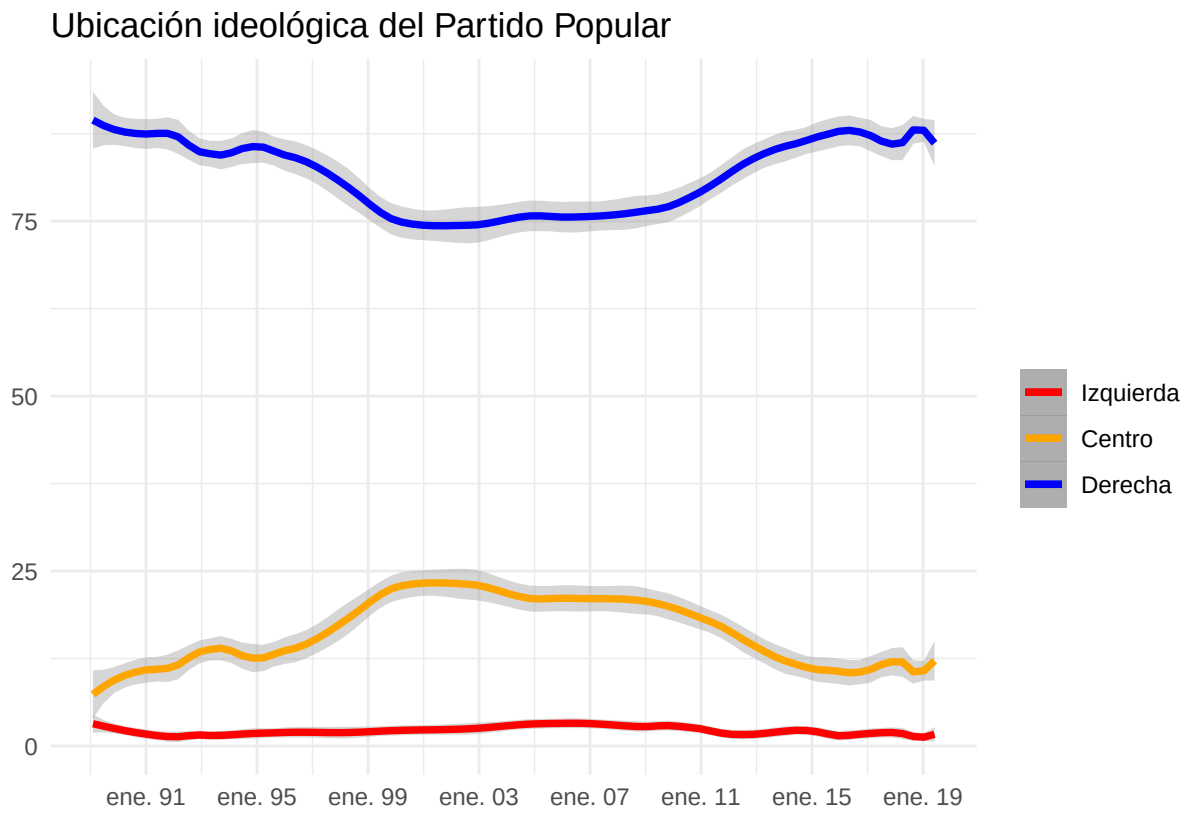
Gracias a la gramática por capas de ggplot se pueden superponer gráficos de puntos sobre un gráfico de línea. Este ejemplo es algo trivial, pero se abren posibilidades muy interesantes.

```
Ubicacion_PP %>%
  ggplot() +
    geom_line(mapping = aes(x = Año, y = izquierda, color = "Izquierda"), size = 1.3) +
    geom_line(mapping = aes(x = Año, y = centro, color = "Centro"), size = 1.3) +
    geom_line(mapping = aes(x = Año, y = derecha, color = "Derecha"), size = 1.3) +
    scale_colour_manual("",
                        breaks = c("Izquierda", "Centro", "Derecha"),
                        values = c("orange", "blue", "red")) +
    scale_x_date("", breaks = "4 year", date_labels = "%b %y") +
    scale_y_continuous("") +
    ggtitle("Ubicación ideológica del Partido Popular")+
    theme_minimal() +
    geom_point(mapping = aes(x= Año, y = izquierda)) +
    geom_point(mapping = aes(x= Año, y = centro)) +
    geom_point(mapping = aes(x= Año, y = derecha))
```



Existe otra geometría, denominada **geom_smooth**, que permite suavizar la serie y añadir un intervalo de confianza para mostrar incertidumbre.

```
Ubicacion_PP %>%
  ggplot() +
    geom_smooth(mapping = aes(x = Año, y = izquierda, color = "Izquierda"), size = 1.3, span = 0.2) +
    geom_smooth(mapping = aes(x = Año, y = centro, color = "Centro"), size = 1.3, span = 0.2) +
    geom_smooth(mapping = aes(x = Año, y = derecha, color = "Derecha"), size = 1.3, span = 0.2) +
    scale_colour_manual("",
                        breaks = c("Izquierda", "Centro", "Derecha"),
                        values = c("orange", "blue", "red")) +
    scale_x_date("", breaks = "4 year", date_labels = "%b %y") +
    scale_y_continuous("") +
    ggtitle("Ubicación ideológica del Partido Popular")+
    theme_minimal()
```



Ggplot2 es una herramienta muy interesante, pero por si sola no es suficiente para cubrir todas las necesidades del analista de datos. Tidyverse tiene paquetes útiles para otras fases del proceso de análisis de datos y te recomiendo encarecidamente que investigues sobre ellos. En esta serie, con introducir *ggplot2* y *dplyr* será suficiente. En el próximo post mostraremos cómo realizar un análisis exploratorio de los datos mediante métricas cuantitativas.

En este post vamos a realizar un análisis exploratorio de los datos (tablas univariadas y bivariadas) y algunas operaciones estadísticas básicas (intervalos de confianza y prueba t de Student). Lo que voy a desarrollar no es más que una breve introducción, ya que las posibilidades de R a nivel estadístico son muy potentes.

Tablas univariadas

En primer lugar vamos a obtener una tabla de frecuencias sencilla de la P1 (grado de interés por la política) con el comando **table**.

```
table(datasetconetiquetas$P1)
```

```
##
##    Mucho Bastante     Poco     Nada     N.C.
##      590     1896     2050     1389       18
```

Ahora con frecuencias relativas, añadiendo primero **prop.table**.

```
prop.table(table(datasetconetiquetas$P1))
```

```
##
##       Mucho    Bastante        Poco        Nada        N.C.
## 0.099276460 0.319030793 0.344943631 0.233720343 0.003028773
```

Si queremos expresar la tabla de frecuencias relativas en porcentajes hay que multiplicar lo anterior por 100.

```
prop.table(table(datasetconetiquetas$P1))*100
```

```
##
##      Mucho   Bastante       Poco       Nada       N.C.
##  9.9276460 31.9030793 34.4943631 23.3720343  0.3028773
```

Para redondear la tabla anterior, primero habría que guardarla en un objeto al que llamamos *tabla* y después aplicar la función **round**.

```
tabla <- prop.table(table(datasetconetiquetas$P1))*100
round(tabla, digits = 1)
```

```
##
##    Mucho Bastante     Poco     Nada     N.C.
##      9.9     31.9     34.5     23.4      0.3
```

Ahora vamos a recodificar entre interesados y no interesados en política, filtrando (con la función filter de dplyr, que veremos después) los no sabe y no contesta. Después recodificamos 1 y 2 como interesados y 3 y 4 como no interesados en política. En la tabla se ve la proporción de cada uno de estos grupos sobre el total de la muestra.

```
subset <- datasetconetiquetas %>%
              filter(P1 != "N.C.", P2 != "N.S.", P2 != "N.C.")
```

```
library(car)

subset$P1 <- as.numeric(subset$P1)
subset$P1.r <- recode(subset$P1, "1=1; 2=1; 3=2; 4=2")

subset$P1.r <- as.factor(subset$P1.r)
levels(subset$P1.r) <- c("Interesados", "No interesados")

prop.table(table(subset$P1.r))
```

```
##
##    Interesados No interesados
##      0.4197698      0.5802302
```

También se pueden calcular estadísticos como la media o la mediana con la función **summarise**, de Tidyverse, que veremos en mayor profundidad en el próximo post. Para no tener en cuenta los casos perdidos, antes de calcular la media y la mediana filtramos para quedarnos solo con los casos donde la P1 sea menor que 9.

```r
dataset %>%
  filter(P1 < 9) %>%
  summarise(media = mean(P1), mediana = median(P1))
```

```
## # A tibble: 1 x 2
##   media mediana
##   <dbl>   <dbl>
## 1  2.72       3
```

Tablas bivariadas

Ya hemos visto cómo explorar una sola variable. Ahora mostraré cómo hacer cruces entre dos variables. Para hacer una tabla de contingencia con frecuencias absolutas de la P1 (interés por la política) y P2 (interés por la campaña electoral) volvemos a usar **table**, aunque esta vez incluyendo el par de variables en cuestión.

```r
table(datasetconetiquetas$P1, datasetconetiquetas$P2)
```

```
##
##            Con mucho interés Con bastante interés Con poco interés
##   Mucho                  342                  184               52
##   Bastante               323                 1217              305
##   Poco                    60                  445             1264
##   Nada                    15                   80              391
##   N.C.                     2                    4                6
##
##            Con ningún interés N.S. N.C.
##   Mucho                    11    1    0
##   Bastante                 46    1    4
##   Poco                    275    3    3
##   Nada                    898    4    1
##   N.C.                      1    0    5
```

Para mostrar ahora las frecuencias relativas sin los que no contestan reaprovecharemos el dataset *subset* creado en el apartado anterior. Al igual que para las frecuencias relativas de una variable, el comando a utilizar es **prop.table** y **round** si queremos redondear, especificando el número de dígitos a los que hacerlo.

```r
round(prop.table(table(subset$P1, subset$P2)) * 100, digits = 2)
```

```
##
##     Con mucho interés Con bastante interés Con poco interés
##   1              5.79                3.11             0.88
##   2              5.47               20.60             5.16
##   3              1.02                7.53            21.39
##   4              0.25                1.35             6.62
##
##     Con ningún interés  N.S.  N.C.
##   1               0.19  0.00  0.00
##   2               0.78  0.00  0.00
##   3               4.65  0.00  0.00
##   4              15.20  0.00  0.00
```

Con la librería descr y su comando **crosstab** podemos hacer tablas de contingencia similares a las de SPSS de forma muy sencilla. En este caso calcularemos el % sobre el total. Cambiando el parámetro prop.t por prop.r y prop.c se mostraría la proporción por fila y por columna, respectivamente. Con el paquete **xtable** se le pueden dar diferentes formatos a las tablas para mejorar la visualización.

```
#install.packages("descr")
library(descr)

levels(subset$P2) <- c("Mucho", "Bastante", "Poco", "Ninguno", "N.S.", "N.C.")
crosstab(subset$P1, subset$P2, prop.t = TRUE, plot = FALSE)
```

```
##    Cell Contents
## |-------------------------|
## |                   Count |
## |           Total Percent |
## |-------------------------|
##
## ============================================================
##              subset$P2
## subset$P1    Mucho   Bastante    Poco   Ninguno   Total
## ------------------------------------------------------------
## 1             342        184      52        11     589
##              5.8%       3.1%    0.9%      0.2%
## ------------------------------------------------------------
## 2             323       1217     305        46    1891
##              5.5%      20.6%    5.2%      0.8%
## ------------------------------------------------------------
## 3              60        445    1264       275    2044
##              1.0%       7.5%   21.4%      4.7%
## ------------------------------------------------------------
## 4              15         80     391       898    1384
##              0.3%       1.4%    6.6%     15.2%
## ------------------------------------------------------------
## Total         740       1926    2012      1230    5908
## ============================================================
```

Intervalos de confianza y prueba t de Student

Las técnicas de investigación social cuantitativas tienen problemas a la hora de acceder a datos de la totalidad de la población. El *big data* está paliando parcialmente esta situación, porque las compañías pueden acceder a los datos que generan sus dispositivos conectados al Internet de las cosas. Por ejemplo, Polar probablemente tendrá una gran base con datos de relojes deportivos con GPS. Sin embargo, la encuesta no tiene esta facilidad para acceder al total de la población. Es aquí donde entra en juego el muestreo, que en términos ideales consiste en acceder a un conjunto de entrevistados representativo del total de la población. De los estadísticos de muestras representativas se pueden inferir parámetros poblacionales.

Una operación estadística de inferencia básica es la estimación de intervalos de confianza dadas una media, una desviación típica y un nivel de confianza. El parámetro poblacional es desconocido, pero con el intervalo de confianza nos aproximamos probabilísticamente a su valor asumiendo que la muestra es representativa. Con el paquete **rmisc** podemos calcular un intervalo de confianza con un nivel de confianza del 95% a partir de la autoubicación ideológica de la muestra, por ejemplo. Antes de calcular el intervalo hemos filtrado para quedarnos solo con los casos que no son perdidos.

```r
#install.packages("Rmisc")
library(Rmisc)

subset <- dataset %>%
            filter(dataset$P32 < 97)

CI(subset$P32, ci = 0.95)
```

```
##    upper     mean    lower
## 4.578137 4.522615 4.467093
```

Asumiendo que la muestra es representativa de la población, la autoubicación ideológica de la población estará entre 4,46 y 4,57 con un 95% de probabilidad.

Otra de las bases de la estadística inferencial es el contraste de hipótesis, que se puede hacer mediante el test t de Student. Si supiésemos que el parámetro poblacional de la autoubicación ideológica es 5, podríamos hacer la prueba y ver si existen diferencias estadísticamente significativas con el valor obtenido en la muestra.

```r
t.test(subset$P32, mu = 5)
```

```
##
##  One Sample t-test
##
## data:  subset$P32
## t = -16.856, df = 5062, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 5
## 95 percent confidence interval:
##  4.467093 4.578137
## sample estimates:
## mean of x
##  4.522615
```

En este caso observamos que el p valor es muy bajo, por lo que existen diferencias estadísticamente significativas entre el valor obtenido en la muestra y el hipotético parámetro poblacional de 5 con un nivel de confianza del 95%. Por tanto, la muestra, bajo el supuesto de que es representativa, no estaría tomada de una población donde la media de autoubicación ideológica fuese de 5 con una alta probabilidad.

En este post he sintetizado lo más básico que se puede hacer a nivel estadístico con R, pero cabe señalar una vez más que sus posibilidades son mucho mayores. Con una búsqueda en *Google* o en *Stackoverflow* se puede obtener mucha información acerca de cómo aplicar técnicas más sofisticadas como las de clasificación o las de predicción. También hay webs recomendables para ello, como r-statistics.co o el Github de Joaquín Amat.

Dplyr es un paquete del Tidyverse orientado a la manipulación de conjuntos de datos. Las transformaciones que se realizan con este paquete se pueden hacer también con R base, pero la gramática de Dplyr es más eficiente y fácil de recordar. Sus funciones con unos **verbos clave**:

- **filter** para filtrar observaciones.
- **arrange** para ordenar observaciones.
- **select** para seleccionar variables.
- **mutate** para crear variables nuevas.
- **summarise** para resumir valores de variables.
- **group_by** para agrupar por una variable y después operar, visualizar, etc.

Filter

Filter ya lo hemos utilizado previamente. Como hemos comentado, sirve para filtrar casos u observaciones. En el caso de la encuesta del CIS, cada caso es un entrevistado. Por ejemplo, si quisiéramos quedarnos solo con los que valoran a Pedro Sánchez con un 5 o más sobre 10.

```
dataset %>%
  filter(P3406 >=5)
```

De esta forma hemos creado un nuevo dataset llamado sanchezaprobado, que contiene solo a los encuestados que en la P3406 han dado un 5 o más. En este **chunk** se utiliza el símbolo %>%. Se trata de una **pipe**, que es un operador que sirve para concatenar operaciones. En el código se observa que hemos puesto el dataset en la primera línea de código, de modo que a partir de ahí R interpreta que todas las operaciones siguientes se hacen sobre ese dataset.

Los **operadores lógicos** de filter son:

- Mayor que (>).
- Menor que (<).
- Mayor o igual que (>=).
- Menor o igual que (<=).
- Igual que (==).
- Distinto de (!=).

Con el uso de | (o) o & (y) se pueden concatenar varias condiciones en la función **filter**, como se muestra en el siguiente fragmento de código.

```
dataset %>%
  filter(P3406 >=5 | P3404 >= 5)
```

Arrange

Para ordenar filas utilizaremos el comando **arrange**. En este caso queremos ordenar de menor a mayor por satisfacción con el funcionamiento de la democracia.

```
dataset %>%
  arrange(P3)
```

Por el contrario, para ordenar de mayor a menor hay que usar el comando **desc()**.

```
dataset %>%
  arrange(desc(P3))
```

Si ordenamos de forma descendente, primero nos saldrán los 98 de "N.S.". Gracias a las pipes podemos combinar la función *filter* con *arrange* y eliminar los no sabe para después ordenar. Además tendremos un código más limpio y legible.

```
dataset %>%
  filter(P3 != 98, P3 != 99) %>%
  arrange(desc(P3))
```

También se pueden encadenar varios *arranges*. En el siguiente caso ordenaremos primero de menor a mayor la P3 y después de mayor a menor por P4. En caso de que la P3 sea igual, las filas se ordenarán de mayor a menor por la P4.

```
dataset %>%
  filter(P3 != 98, P3 != 99, P4 != 8, P4 != 9) %>%
  arrange(P3, desc(P4))
```

Select

Hasta ahora hemos trabajado con las 309 variables del dataset completo y tarda más en computar, por lo que la programación es menos eficiente. La función **select** nos va a permitir quedarnos solo con las variables de nuestro interés.

Si solo quisiéramos obtener la variable CUES, que es el identificador del cuestionario, tendríamos que ejecutar lo siguiente:

```
dataset %>%
  select(CUES) %>%
  head()
```

```
## # A tibble: 6 x 1
##    CUES
##   <dbl>
## 1   519
## 2   520
## 3   521
## 4   522
## 5   523
## 6   524
```

Si queremos quedarnos solo con las variables relativas al desarrollo de la entrevista, que vienen ordenadas al final, no es necesario escribirlas todas en el paréntesis del comando select. Bastaría con poner la primera y la última.

```
dataset %>%
  select(P6001:P6102) %>%
  head()
```

```
## # A tibble: 6 x 7
##   P6001 P6002 P6003 P6004 P6005 P6101 P6102
##   <dbl> <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1     1    NA    NA    NA    NA -     -
## 2    NA    NA    NA    NA     1 -     -
## 3    NA    NA    NA    NA    NA -     -
## 4    NA    NA    NA     1    NA -     -
## 5    NA    NA    NA    NA     1 51    53
## 6    NA    NA    NA     1    NA -     -
```

Y para elegir todas las variables menos las relativas al desarrollo de la entrevista bastaría con poner un símbolo de -.

```
dataset %>%
  select(-(P6001:P6102)) %>%
  head()
```

```
## # A tibble: 6 x 302
##    ESTU  CUES  CCAA  PROV   MUN TAMUNI CAPITAL DISTR SECCION ENTREV    P0
##   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl> <dbl>   <dbl>  <dbl> <dbl>
## 1  3248   519    11     6    15      5       2     0       0      0     1
## 2  3248   520    11     6    15      5       2     0       0      0     1
## 3  3248   521    11     6    15      5       2     0       0      0     1
## 4  3248   522    11     6    15      5       2     0       0      0     1
## 5  3248   523    11     6    15      5       2     0       0      0     1
## 6  3248   524    11     6    15      5       2     0       0      0     1
## # ... with 291 more variables: P1 <dbl>, P2 <dbl>, P3 <dbl>, P4 <dbl>,
## #   P5 <dbl>, P6 <dbl>, P7 <dbl>, P8 <dbl>, P901 <dbl>, P902 <dbl>,
## #   P903 <dbl>, P904 <dbl>, P905 <dbl>, P906 <dbl>, P907 <dbl>,
## #   P908 <dbl>, P909 <dbl>, P910 <dbl>, P911 <dbl>, P912 <dbl>, P10 <dbl>,
## #   P10A01 <dbl>, P10A02 <dbl>, P10A03 <dbl>, P10A04 <dbl>, P10A05 <dbl>,
## #   P10A06 <dbl>, P10A07 <dbl>, P10A08 <dbl>, P10A09 <dbl>, P10A10 <dbl>,
## #   P10A11 <dbl>, P10A12 <dbl>, P10A13 <dbl>, P10A14 <dbl>, P11 <dbl>,
## #   P11A <dbl>, P11B <dbl>, P12 <dbl>, P1301 <dbl>, P1302 <dbl>,
## #   P1303 <dbl>, P1304 <dbl>, P13A <dbl>, P13B <dbl>, P13C <dbl>,
## #   P13D01 <dbl>, P13D02 <dbl>, P13D03 <dbl>, P13D04 <dbl>, P13D05 <dbl>,
## #   P13D06 <dbl>, P13D07 <dbl>, P13E01 <dbl>, P13E02 <dbl>, P13E03 <dbl>,
## #   P13E04 <dbl>, P13E05 <dbl>, P13E06 <dbl>, P13E07 <dbl>, P14 <dbl>,
## #   P14A <dbl>, P14B <dbl>, P14C <dbl>, P14D <dbl>, P1501 <dbl>,
## #   P1502 <dbl>, P1503 <dbl>, P1504 <dbl>, P1505 <dbl>, P16 <dbl>,
## #   P16A01 <dbl>, P16A02 <dbl>, P16A03 <dbl>, P16A04 <dbl>, P16A05 <dbl>,
## #   P16A06 <dbl>, P16A07 <dbl>, P16A08 <dbl>, P16A09 <dbl>, P16A10 <dbl>,
## #   P16A11 <dbl>, P16A12 <dbl>, P16A13 <dbl>, P16A14 <dbl>, P16A15 <dbl>,
## #   P16A16 <dbl>, P16A17 <dbl>, P17 <dbl>, P17A <dbl>, P17B01 <dbl>,
## #   P17B02 <dbl>, P17B03 <dbl>, P17B04 <dbl>, P17B05 <dbl>, P17B06 <dbl>,
## #   P17B07 <dbl>, P17B08 <dbl>, P17B09 <dbl>, P17B10 <dbl>, ...
```

Otra posibilidad es quedarnos solo con las variables que empiezan por P. Para seleccionar las que acaban bajo el criterio que establezcamos el comando a usar sería **ends_with**. A los comandos ya mencionados se les suman otros como **contains**, **matches** o **num_range**.

```
dataset %>%
  select(starts_with("P")) %>%
  head()
```

```
## # A tibble: 6 x 272
##    PROV    P0    P1    P2    P3    P4    P5    P6    P7    P8  P901  P902
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     6     1     1     1     1     1     4     4     3     1     2     1
## 2     6     1     2     2     5     1     3     3     3     1     2     2
## 3     6     1     1     1     5     1     4     4     4     2     2     2
## 4     6     1     2     2     8     1     2     3     2     1     2     1
## 5     6     1     3     2     5     1     3     3     4     1     2     2
## 6     6     1     1     2     1     1     4     4     5     2     2     2
## # ... with 260 more variables: P903 <dbl>, P904 <dbl>, P905 <dbl>,
## #   P906 <dbl>, P907 <dbl>, P908 <dbl>, P909 <dbl>, P910 <dbl>,
```

```
## #   P911 <dbl>, P912 <dbl>, P10 <dbl>, P10A01 <dbl>, P10A02 <dbl>,
## #   P10A03 <dbl>, P10A04 <dbl>, P10A05 <dbl>, P10A06 <dbl>, P10A07 <dbl>,
## #   P10A08 <dbl>, P10A09 <dbl>, P10A10 <dbl>, P10A11 <dbl>, P10A12 <dbl>,
## #   P10A13 <dbl>, P10A14 <dbl>, P11 <dbl>, P11A <dbl>, P11B <dbl>,
## #   P12 <dbl>, P1301 <dbl>, P1302 <dbl>, P1303 <dbl>, P1304 <dbl>,
## #   P13A <dbl>, P13B <dbl>, P13C <dbl>, P13D01 <dbl>, P13D02 <dbl>,
## #   P13D03 <dbl>, P13D04 <dbl>, P13D05 <dbl>, P13D06 <dbl>, P13D07 <dbl>,
## #   P13E01 <dbl>, P13E02 <dbl>, P13E03 <dbl>, P13E04 <dbl>, P13E05 <dbl>,
## #   P13E06 <dbl>, P13E07 <dbl>, P14 <dbl>, P14A <dbl>, P14B <dbl>,
## #   P14C <dbl>, P14D <dbl>, P1501 <dbl>, P1502 <dbl>, P1503 <dbl>,
## #   P1504 <dbl>, P1505 <dbl>, P16 <dbl>, P16A01 <dbl>, P16A02 <dbl>,
## #   P16A03 <dbl>, P16A04 <dbl>, P16A05 <dbl>, P16A06 <dbl>, P16A07 <dbl>,
## #   P16A08 <dbl>, P16A09 <dbl>, P16A10 <dbl>, P16A11 <dbl>, P16A12 <dbl>,
## #   P16A13 <dbl>, P16A14 <dbl>, P16A15 <dbl>, P16A16 <dbl>, P16A17 <dbl>,
## #   P17 <dbl>, P17A <dbl>, P17B01 <dbl>, P17B02 <dbl>, P17B03 <dbl>,
## #   P17B04 <dbl>, P17B05 <dbl>, P17B06 <dbl>, P17B07 <dbl>, P17B08 <dbl>,
## #   P17B09 <dbl>, P17B10 <dbl>, P17B11 <dbl>, P17B12 <dbl>, P17B13 <dbl>,
## #   P17B14 <dbl>, P17B15 <dbl>, P17B16 <dbl>, P17B17 <dbl>, P17C01 <dbl>,
## #   P17C02 <dbl>, P17C03 <dbl>, ...
```

Una función muy similar a *select* es **rename**. Como su propio nombre indica, permite cambiar el nombre de las variables para hacerlas más manejables.

```
dataset %>%
  dplyr::rename(interespolitica = P1, interescampaña = P2, satisfacciondemocracia = P3)
```

Los dos últimos comandos se pueden combinar, de modo que se puede seleccionar y renombrar a la vez.

```
dataset %>%
  select(interespolitica = P1, interescampaña = P2, satisfacciondemocracia = P3)
```

```
## # A tibble: 5,943 x 3
##    interespolitica interescampaña satisfacciondemocracia
##              <dbl>          <dbl>                  <dbl>
##  1               1              1                      1
##  2               2              2                      5
##  3               1              1                      5
##  4               2              2                      8
##  5               3              2                      5
##  6               1              2                      1
##  7               4              3                      5
##  8               3              4                      2
##  9               3              3                      4
## 10               2              2                      9
## # ... with 5,933 more rows
```

Mutate

El comando **mutate** sirve para calcular nuevas variables. Este comando resultaría útil si quisiéramos, por ejemplo, calcular la distancia percibida entre PP y PSOE por los encuestados. Antes de operar con *mutate* hemos enviado a perdidos todos los 98 y 99 de las variables implicadas en los próximos ejemplos. También

incorporamos *select* con esas variables implicadas para agilizar la computación y facilitar la visualización de los resultados de usar *mutate*.

```
dataset$P3301[dataset$P3301 == 98] <- NA
dataset$P3301[dataset$P3301 == 99] <- NA
dataset$P3302[dataset$P3302 == 98] <- NA
dataset$P3302[dataset$P3302 == 99] <- NA
dataset$P3302[dataset$P3303 == 98] <- NA
dataset$P3302[dataset$P3303 == 99] <- NA


dataset %>%
  select(P3301, P3302) %>%
  mutate(distpppsoe = P3302 - P3301)
```

```
## # A tibble: 5,943 x 3
##    P3301 P3302 distpppsoe
##    <dbl> <dbl>      <dbl>
##  1     5     7          2
##  2     3     8          5
##  3     4     6          2
##  4     2     9          7
##  5     3     8          5
##  6     2     9          7
##  7     3     8          5
##  8     4     8          4
##  9     4     7          3
## 10     2     8          6
## # ... with 5,933 more rows
```

También se puede añadir a lo anterior el cálculo de una segunda y una tercera variable, como la distancia percibida entre PSOE y Ciudadanos y entre PP y Ciudadanos. Con estas variables se puede operar dentro de la misma secuencia de código que las crea, sin necesidad de empezar una nueva.

```
dataset %>%
  select(P3301, P3302, P3303) %>%
  mutate(distpppsoe = P3302 - P3301, distpsoecs = P3301 - P3303, distppcs = P3302 - P3303)
```

```
## # A tibble: 5,943 x 6
##    P3301 P3302 P3303 distpppsoe distpsoecs distppcs
##    <dbl> <dbl> <dbl>      <dbl>      <dbl>    <dbl>
##  1     5     7     7          2         -2        0
##  2     3     8     7          5         -4        1
##  3     4     6     5          2         -1        1
##  4     2     9     7          7         -5        2
##  5     3     8     7          5         -4        1
##  6     2     9     7          7         -5        2
##  7     3     8     6          5         -3        2
##  8     4     8     6          4         -2        2
##  9     4     7     6          3         -2        1
## 10     2     8     6          6         -4        2
## # ... with 5,933 more rows
```

Estas nuevas variables creadas también pueden operar con estadísticos como la media. Para calcular la media es importante excluir los valores NA, ya que en caso de no hacerlo mediante el comando **na.rm**, la media no podrá ser calculada. Si en un cálculo hay un NA, el resultado será NA independientemente de la media de los demás valores. En el siguiente fragmento de código calculamos la distancia entre la ubicación ideológica del PSOE percibida por el entrevistado y la autoubicación media del PSOE percibida por la muestra.

```
dataset %>%
  select(P3301) %>%
  mutate(distpsoemedia = P3301 - mean(P3301, na.rm = TRUE))
```

```
## # A tibble: 5,943 x 2
##    P3301 distpsoemedia
##    <dbl>         <dbl>
##  1     5       1.00
##  2     3      -0.997
##  3     4       0.00341
##  4     2      -2.00
##  5     3      -0.997
##  6     2      -2.00
##  7     3      -0.997
##  8     4       0.00341
##  9     4       0.00341
## 10     2      -2.00
## # ... with 5,933 more rows
```

Mutate añade las variables calculadas al conjunto de datos existente conservando todas las variables. Sin embargo, existe una variación de este comando, llamada **transmute**, que crea un conjunto de datos que solo contiene las variables de nueva creación.

```
dataset %>%
  transmute(distpppsoe = P3302 - P3301, distpsoecs = P3301 - P3303, distppcs = P3302 - P3303)
```

```
## # A tibble: 5,943 x 3
##    distpppsoe distpsoecs distppcs
##         <dbl>      <dbl>    <dbl>
##  1          2         -2        0
##  2          5         -4        1
##  3          2         -1        1
##  4          7         -5        2
##  5          5         -4        1
##  6          7         -5        2
##  7          5         -3        2
##  8          4         -2        2
##  9          3         -2        1
## 10          6         -4        2
## # ... with 5,933 more rows
```

Summarise

El último comando de **dplyr** que vamos a ver es **summarise**, que resume variables del dataframe en un solo estadístico. Permite, por ejemplo, calcular la media de autoubicación ideológica de los encuestados.

```
dataset %>%
  summarise(mean(P32))
```

```
##   mean(P32)
## 1   18.4279
```

Sin embargo, el dato que hemos obtenido con el código anterior está falseado por los 98 y 99 correspondientes al no sabe y al no contesta. Para paliar este defecto hay que recodificarlos con **na_if** de *mutate*. Este comando es especialmente útil cuando tratamos con datos de encuesta, donde es frecuente tener que lidiar con valores perdidos. Con el siguiente código enviamos a perdidos (NA, de *not available*) los 98 y 99 y repetimos el cálculo anterior para obtener la media real de la muestra excluyendo los perdidos

```
dataset %>%
  select(P32) %>%
  mutate(P32 = na_if(P32, 98)) %>%
  mutate(P32 = na_if(P32, 99)) %>%
  summarise(mediaubicacion = mean(P32, na.rm = TRUE))
```

```
##   mediaubicacion
## 1       4.522615
```

Por último, con **group_by** podemos crear categorías en función de una variable, por ejemplo la religión, y tener en cuenta esos grupos para calcular estadísticos, como por ejemplo la autoubicación ideológica media de cada grupo religioso. Previamente hemos definido las etiquetas de cada código con **levels**. Este comando permite segmentar la muestra y acceder a estadísticos desagregados.

```
dataset$P48 <- as_factor(dataset$P48)
levels(dataset$P48) <- c("Católico practicante", "Católico no practicante", "Creyente de otra religión"

dataset %>%
  select(P32, P48) %>%
  mutate(P32 = na_if(P32, 98)) %>%
  mutate(P32 = na_if(P32, 99)) %>%
  mutate(P48 = na_if(P48, 9)) %>%
  group_by(P48) %>%
  dplyr::summarise(mediaubicacion = mean(P32, na.rm = TRUE))
```

```
## # A tibble: 7 x 2
##   P48                        mediaubicacion
##   <fct>                               <dbl>
## 1 Católico practicante                 5.77
## 2 Católico no practicante              4.62
## 3 Creyente de otra religión            4.56
## 4 Agnóstico                            3.49
## 5 Indiferente, no creyente             3.84
## 6 Ateo                                 3.06
## 7 N.C.                                 3.62
```

Para cerrar esta serie de posts en los que estamos comenzando a programar en R con datos del CIS vamos a hablar de las funciones y de las dos aportaciones más comunes de la comunidad: los paquetes y el código abierto.

Las **funciones** permiten condensar un procedimiento en una sola función. Es la lógica que siguen los comandos que hemos ido viendo durante toda la serie. Si queremos crear una función que cruce dos variables llamadas x e y, usaremos el comando **function**.

```
tablacruzada <- function(x,y){table(x, y)}
```

La función llamada *tablacruzada* cogerá los valores de x e y y devolverá una tabla cruzada de esas dos variables. Antes de aplicarla hay que dar un valor a x y otro a y. En este caso x será la interés en la política e y el interés en la campaña electoral.

```
x <- dataset$P1
y <- dataset$P2
```

Por último aplicamos la función, que nos devuelve la tabla cruzada de P1 y P2. La programación de este tipo de funciones es especialmente útil cuando necesitamos repetir varias veces un mismo procedimiento y queremos tener un código limpio y legible.

```
tablacruzada(x,y)
```

```
##    y
## x      1    2    3    4    8    9
##   1  342  184   52   11    1    0
##   2  323 1217  305   46    1    4
##   3   60  445 1264  275    3    3
##   4   15   80  391  898    4    1
##   9    2    4    6    1    0    5
```

Los **paquetes** son conjuntos de funciones que facilitan la programación en R. Son una especie de complemento que amplía las funcionalidades del R básico. Al tener una comunidad tan amplia, este lenguaje de programación cuenta con paquetes muy diversos que podemos encontrar en su repositorio oficial llamado **CRAN** o en los Github de los propios desarrolladores (aunque hay que tener precaución y saber qué estamos instalando para prevenir problemas). Dos paquetes no oficiales interesantes para científicos sociales son **CisUtils**, de Gonzalo Rivero para tratar datos del CIS, y **Elecciones**, de Héctor Meleiro para descargar resultados electorales.

La instalación de paquetes, como ya hemos ido viendo durante la serie, se hace mediante el comando **install.packages** si está en el CRAN y **devtools::install_github** si está alojado en Github. Para instalar Elecciones haremos lo siguiente:

```
devtools::install_github("hmeleiro/elecciones")
```

La activación de paquetes ya instalados se hace con **library**.

```
library(elecciones)
```

Una vez instalada y activada la librería, no hay más que ir a su página web y buscar la documentación, en la que aparecerá una descripción de las funcionalidades que añade y ejemplos de su uso. Con Elecciones, por ejemplo, podemos descargar los resultados de las elecciones generales de 1979 a nivel de municipio como se comenta en su documentación.

```
generales.1979 <- municipios(tipoeleccion = "generales",
                             yr = 1979, mes = "03")
```

Otros usuarios de R comparten directamente el **código** de su análisis para que pueda ser leído y replicado por las personas interesadas. Habitualmente este código se acompaña de comentarios que documentan el proceso y facilitan su interpretación. Ejemplos de esto son el código compartido por Ariane Aumaitre donde enseña a animar gráficos de ggplot2 o el de Daiana Emili donde analiza el guion de La Casa de Papel.