1. Transformación de los datos

Taller: calidad del aire

Minería de datos II

Curso 2019/2020

Contents

1 Introducción

El ayuntamiento de Madrid nos ha contratado para desarrollar un **modelo de la contaminación** que permita predecir el nivel de contaminación que habrá al día siguiente. Nos han enviado un conjunto de datos con mediciones diarias de la **calidad del aire de la ciudad de Madrid**. Aunque existen varias medidas que hay que tener en cuenta para medir la calidad del aire, el ayuntamiento nos ha pedido que elaboremos el modelo solamente para predecir **PM2.5**: **la concentración de partículas en suspensión de menos de** 2.5 **micras.**

Cualquier proyecto de minería de datos va a tener, al menos, **3 fases**:

- Tratamiento de datos.
- Modelización.
- Evaluación.

Empezaremos por importar los datos y hacer las transformaciones necesarias sobre ellos para poder empezar a trabajar.

2 Entorno

Cuando se trabaja en un proyecto real, es importante ser *ordenado* en los códigos desarrollados ya que solemos trabajar en equipo. Abre el proyecto de RStudio que hemos creado en clase y crea una **nueva carpeta** que se llame `taller_calidad_aire`. Dentro de esta carpeta, crea otras dos que se denominen `data` y `src`. En la primera de ellas guardaremos los datos que puedes descargar del aula virtual. En la segunda carpeta será donde iremos desarrollando los scripts (`source`, es una práctica habitual en programación).

3 Transformación de los datos

> Crea un **script R** en la carpeta `taller_calidad_aire/src` que se llame `01_transformacion.R`; será el script que desarrollaremos en este documento.

Un paquete muy importante en R para el tratamiento de datos es `tidyverse`. Recuerda que este paquete contiene, a su vez, otros paquetes. Cárgalo al principio del script (si no lo tienes instalado, deberás instalarlo primero):

```
library(tidyverse)
```

3.1 Importación

A continuación necesitamos **importar los datos** para poder empezar a trabajar con ellos:

```
calidad_aire <- read_csv("taller_calidad_aire/data/calidad_aire.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   PROVINCIA = col_double(),
##   ESTACION = col_double(),
##   MAGNITUD = col_double(),
##   ANO = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

El archivo que estamos cargando tiene extensión `csv` (*comma-separated value*) un formato de datos muy utilizado para el intercambio de información. Si abres el archivo original, verás que la primera linea contiene el nombre de las variables separadas por comas y, el resto de líneas, contienen los datos:

```
## PROVINCIA,MUNICIPIO,ESTACION,MAGNITUD,PUNTO_MUESTREO,ANO,MES,D01,V01,D02,V02,D03,V03,D04,V04,D05,V05
## 28,079,4,1,28079004_1_38,2019,01,00018,V,00020,V,00018,V,00019,V,00018,V,00018,V,00021,V,00020,V,000
## 28,079,4,1,28079004_1_38,2019,02,00013,V,00013,V,00014,V,00018,V,00019,V,00019,V,00019,V,00016,V,000
```

En el aula virtual también tienes el archivo `Interprete_ficheros_calidad_del_aire_global.pdf` con las descripción de cómo son los datos y qué estructura tienen.

3.2 Estructura de los datos

Podemos inspeccionarlos en la consola para ver cómo los ha importado R:

```
calidad_aire
```

```
## # A tibble: 5,301 x 69
##    PROVINCIA MUNICIPIO ESTACION MAGNITUD PUNTO_MUESTREO   ANO MES   D01   V01
##        <dbl> <chr>        <dbl>    <dbl> <chr>          <dbl> <chr> <chr> <chr>
##  1        28 079              4        1 28079004_1_38   2019 01    00018 V
##  2        28 079              4        1 28079004_1_38   2019 02    00013 V
##  3        28 079              4        1 28079004_1_38   2019 03    00018 V
##  4        28 079              4        1 28079004_1_38   2019 04    00003 V
##  5        28 079              4        1 28079004_1_38   2019 05    00001 V
##  6        28 079              4        1 28079004_1_38   2019 06    00001 V
##  7        28 079              4        1 28079004_1_38   2019 07    00001 V
##  8        28 079              4        1 28079004_1_38   2019 08    00008 V
##  9        28 079              4        1 28079004_1_38   2019 09    00008 V
## 10        28 079              4        1 28079004_1_38   2019 10    00010 V
## # ... with 5,291 more rows, and 60 more variables: D02 <chr>, V02 <chr>,
## #   D03 <chr>, V03 <chr>, D04 <chr>, V04 <chr>, D05 <chr>, V05 <chr>,
## #   D06 <chr>, V06 <chr>, D07 <chr>, V07 <chr>, D08 <chr>, V08 <chr>,
## #   D09 <chr>, V09 <chr>, D10 <chr>, V10 <chr>, D11 <chr>, V11 <chr>,
## #   D12 <chr>, V12 <chr>, D13 <chr>, V13 <chr>, D14 <chr>, V14 <chr>,
## #   D15 <chr>, V15 <chr>, D16 <chr>, V16 <chr>, D17 <chr>, V17 <chr>,
## #   D18 <chr>, V18 <chr>, D19 <chr>, V19 <chr>, D20 <chr>, V20 <chr>,
## #   D21 <chr>, V21 <chr>, D22 <chr>, V22 <chr>, D23 <chr>, V23 <chr>,
## #   D24 <chr>, V24 <chr>, D25 <chr>, V25 <chr>, D26 <chr>, V26 <chr>,
## #   D27 <chr>, V27 <chr>, D28 <chr>, V28 <chr>, D29 <chr>, V29 <chr>,
## #   D30 <chr>, V30 <chr>, D31 <chr>, V31 <chr>
```

El conjunto de datos contiene **69 variables** y **5301 filas**. Tenemos **dos grupos de variables**. El primero de ellos contiene la información para identificar cómo, dónde y cuándo se ha realizado una medición:

- `PROVINCIA`
- `MUNICIPIO`
- `ESTACION`
- `MAGNITUD`
- `PUNTO_MUESTREO`
- `ANO`
- `MES`

El segundo grupo de variables contiene una variable para cada día del mes `D01`, `D02`, `D03`, ..., `D31` con el valor de la magnitud medida y, asociada a cada variable, otras `V01`, `V02`, ..., `V31` que indican si la medida ha sido comprobada y es correcta (`V`) o no (`N`).

> **Esta estructura de los datos no es adecuada para realizar los análisis**. Imagina que quisiésemos obtener **la medición de la magnitud 1 de todos los lunes del año**. En este caso, la estructura de los datos hace difícil responder a esa pregunta.

3.3 Eliminación de variables irrelevantes

Antes de cambiar la estructura, vamos a hacer algunas operaciones de limpieza. Aunque no es imprescindible, vamos a convertir el nombre de las variables a minúscula con la función `tolower` para favorecer la consistencia:

```
names(calidad_aire) <- tolower(names(calidad_aire))
```

Puede que **algunas variables sean prescindibles**. Por ejemplo, a veces se da el caso de que hay **variables con un único valor** (*variables unarias*) y que, por tanto, no aportan nada al análisis. Por ejemplo, la variable `provincia` es una de ellas. Podemos comprobarlo pidiédole a R que nos diga cuántos valores distintos contiene esta variable:

```
length(unique(calidad_aire$provincia))
```

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

Obviamente, si el número de variables es elevado (en este caso son 69), no podemos hacer esta comprabación de forma manual. Podemos escribir el siguiente bucle para comprobarlo automáticamente:

```
var_unarias <- c()
for (var in names(calidad_aire)){
  if (length(unique(calidad_aire[[var]])) == 1){
    var_unarias <- c(var_unarias, var)
  }
}
```

Opcional: *aunque el bucle anterior es perfectamente válido, conforme se va ganando experiencia en programación, se busca escribir código lo más legible posible ya que esto facilitará la tarea de corregir errores. Una alternativa al código anterior podría ser:*

```
var_unarias <- sapply(calidad_aire, function(x)length(unique(x)) == 1)
var_unarias <- names(calidad_aire)[var_unarias]
```

El vector que obtenemos contiene solamente dos variables

```
var_unarias
```

```
## [1] "provincia" "municipio"
```

Por tanto, eliminamos estas dos variables del conjunto de datos

```
calidad_aire$provincia <- NULL
calidad_aire$municipio <- NULL
```

Además, la variable `punto_muestreo` no es relevante para nuestro estudio (según el pdf de la documentación, este campo es la concatenación de provincia, municipio, estación, magnitud y la técnica de muestreo, información que ya tenemos recogida en otras variables) y procedemos a eliminarla también:

```
calidad_aire$punto_muestreo <- NULL
```

3.4 Formato adecuado de variables

El siguiente paso es comprobar que las variables estén en el formato adecuado:

```
glimpse(calidad_aire)
```

```
## Observations: 5,301
## Variables: 66
## $ estacion <dbl> 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4...
## $ magnitud <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6...
## $ ano      <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019...
## $ mes      <chr> "01", "02", "03", "04", "05", "06", "07", "08", "09", "10"...
## $ d01      <chr> "00018", "00013", "00018", "00003", "00001", "00001", "000...
## $ v01      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d02      <chr> "00020", "00013", "00018", "00004", "00001", "00001", "000...
## $ v02      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d03      <chr> "00018", "00014", "00017", "00004", "00001", "00001", "000...
## $ v03      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d04      <chr> "00019", "00018", "00017", "00003", "00002", "00001", "000...
## $ v04      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d05      <chr> "00018", "00019", "00016", "00002", "00002", "00002", "000...
## $ v05      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d06      <chr> "00018", "00019", "00016", "00002", "00003", "00002", "000...
## $ v06      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d07      <chr> "00021", "00019", "00014", "00002", "00003", "00002", "000...
## $ v07      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d08      <chr> "00020", "00016", "00016", "00003", "00003", "00002", "000...
## $ v08      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d09      <chr> "00018", "00016", "00016", "00003", "00002", "00001", "000...
## $ v09      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d10      <chr> "00013", "00014", "00017", "00003", "00002", "00002", "000...
## $ v10      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d11      <chr> "00016", "00015", "00017", "00004", "00002", "00002", "000...
## $ v11      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d12      <chr> "00016", "00016", "00016", "00003", "00001", "00003", "000...
## $ v12      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d13      <chr> "00016", "00017", "00017", "00003", "00002", "00002", "000...
## $ v13      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d14      <chr> "00022", "00017", "00016", "00003", "00001", "00003", "000...
## $ v14      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d15      <chr> "00021", "00017", "00019", "00003", "00002", "00002", "000...
## $ v15      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d16      <chr> "00016", "00015", "00017", "00002", "00002", "00003", "000...
## $ v16      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d17      <chr> "00015", "00017", "00017", "00002", "00003", "00003", "000...
## $ v17      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d18      <chr> "00014", "00018", "00015", "00002", "00002", "00002", "000...
## $ v18      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d19      <chr> "00012", "00018", "00016", "00002", "00002", "00002", "000...
## $ v19      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
```

```
## $ d20      <chr> "00014", "00018", "00015", "00002", "00002", "00002", "000...
## $ v20      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d21      <chr> "00015", "00017", "00017", "00002", "00003", "00002", "000...
## $ v21      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d22      <chr> "00014", "00020", "00017", "00003", "00003", "00002", "000...
## $ v22      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d23      <chr> "00016", "00019", "00017", "00002", "00003", "00002", "000...
## $ v23      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d24      <chr> "00017", "00018", "00017", "00001", "00003", "00002", "000...
## $ v24      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d25      <chr> "00018", "00019", "00017", "00002", "00002", "00003", "000...
## $ v25      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d26      <chr> "00016", "00018", "00009", "00002", "00001", "00002", "000...
## $ v26      <chr> "V", "V", "V", "V", "V", "N", "V", "V", "V", "V", "V", "V"...
## $ d27      <chr> "00015", "00019", "00006", "00003", "00002", "00001", "000...
## $ v27      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d28      <chr> "00015", "00020", "00007", "00002", "00001", "00001", "000...
## $ v28      <chr> "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d29      <chr> "00015", "00000", "00006", "00002", "00001", "00001", "000...
## $ v29      <chr> "V", "N", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d30      <chr> "00014", "00000", "00004", "00002", "00001", "00001", "000...
## $ v30      <chr> "V", "N", "V", "V", "V", "V", "V", "V", "V", "V", "V", "V"...
## $ d31      <chr> "00014", "00000", "00004", "00000", "00002", "00000", "000...
## $ v31      <chr> "V", "N", "V", "N", "V", "N", "V", "V", "N", "V", "N", "V"...
```

La primera variable que nos encontramos en un formato no adecuado es `mes`. Esta variable aparece como carácter (`char`) mientras que lo lógico sería que fuese un número (en particular, un número entero). Podemos convertirla de la siguiente forma:

```
calidad_aire$mes <- as.integer(calidad_aire$mes)
```

Algo parecido nos sucede con las mediciones. Todas las variables `d01`, `d02`, etcétera son carácter cuando deberían ser numéricas. En este caso, convertirlas al formato adecuado es algo más complejo. Aunque hay distintas formas de hacerlo, vamos a recurrir a una función del paquete `tidyverse` que cargamos al principio:

```
calidad_aire <- mutate_at(calidad_aire, vars(starts_with("d")), as.numeric)
```

> **Si alguna de las funciones que utilizamos no entiendes bien su uso, asegúrate de acudir a la ayuda (`?mutate_at`)**

3.5 Cambio de estructura

Ahora es donde llegamos a la transformación realmente importante. No es una buena idea mantener la estructura actual de los datos. Fíjate en que no es una forma natural de representar la información, hay meses que tienen 31 días, otros 30 y febrero aún menos. Esto supondrá que no es eficiente esa codificación. Lo que nos gustaría es poder construir una variable que exprese el día del mes y una única variable con la medición correspondiente.

Para ello, primero separamos el conjunto de datos en dos, uno para las variables con el valor de la magnitud (`d01`, `d02`,...) y otro con las que verifican ese valor (`v01`, `v02`,...).

```
calidad_aire_medicion <- calidad_aire %>%
  select(
    estacion:mes,
    starts_with("d")
  )

calidad_aire_validado <- calidad_aire %>%
  select(
    estacion:mes,
    starts_with("v")
  )
```

Ahora, convertimos los datos con ayuda de la función `pivot_longer` (**nota**: *asegúrate de que entiendes qué es lo que queremos hacer y cómo nos ayuda esta función a conseguirlo*).

```
calidad_aire_medicion <- calidad_aire_medicion %>%
  pivot_longer(
    cols = d01:d31,
    names_to = "dia",
    values_to = "medicion"
  )
```

Obtenemos así la siguiente estructura del `data.frame`:

```
calidad_aire_medicion
```

```
## # A tibble: 164,331 x 6
##    estacion magnitud   ano   mes dia   medicion
##       <dbl>    <dbl> <dbl> <int> <chr>    <dbl>
##  1        4        1  2019     1 d01         18
##  2        4        1  2019     1 d02         20
##  3        4        1  2019     1 d03         18
##  4        4        1  2019     1 d04         19
##  5        4        1  2019     1 d05         18
##  6        4        1  2019     1 d06         18
##  7        4        1  2019     1 d07         21
##  8        4        1  2019     1 d08         20
##  9        4        1  2019     1 d09         18
## 10        4        1  2019     1 d10         13
## # ... with 164,321 more rows
```

Nos falta modificar la variable `dia` para que se numérica. Fíjate que todos los valores empiezan por `d`, así que no podemos utilizar directamente `as.numeric`. Primero tendremos que eliminar esa `d`:

```
calidad_aire_medicion$dia <- substr(calidad_aire_medicion$dia, start = 2, stop = 3)
```

Y después convertir a numérica que, en este caso, como el día va a ser un número entero, utilizamos `as.integer`:

```
calidad_aire_medicion$dia <- as.integer(calidad_aire_medicion$dia)
```

Repetimos el mismo procedimiento con `calidad_aire_validado`:

```
calidad_aire_validado <- calidad_aire_validado %>%
  pivot_longer(
    cols = v01:v31,
    names_to = "dia",
    values_to = "validado"
  )

calidad_aire_validado$dia <- substr(calidad_aire_validado$dia, start = 2, stop = 3)
calidad_aire_validado$dia <- as.integer(calidad_aire_validado$dia)
```

Ahora, cruzaremos la información que tenemos de la medición y su corresponiente validación:

```
calidad_aire <- calidad_aire_medicion %>%
  left_join(calidad_aire_validado)
```

```
## Joining, by = c("estacion", "magnitud", "ano", "mes", "dia")
```

3.6 Eliminación de observaciones erróneas

Aquellas mediciones que no hayan sido validadas, contienen un valor que no podemos estar seguros de su uso. Vamos a convertir a `NA` aquellas mediciones no validadas.

```
calidad_aire$medicion[calidad_aire$validado == "N"] <- NA
```

Tenemos todavía un problema que resolver: hay observaciones que no tienen sentido. Por ejemplo, tenemos una observación para el 30 de febrero:

```
filter(calidad_aire, mes == 2, dia == 30)
```

```
## # A tibble: 455 x 7
##    estacion magnitud   ano   mes   dia medicion validado
##       <dbl>    <dbl> <dbl> <int> <int>    <dbl> <chr>
##  1        4        1  2019     2    30       NA N
##  2        4        6  2019     2    30       NA N
##  3        4        7  2019     2    30       NA N
##  4        4        8  2019     2    30       NA N
##  5        4       12  2019     2    30       NA N
##  6        8        1  2019     2    30       NA N
##  7        8        6  2019     2    30       NA N
##  8        8        7  2019     2    30       NA N
##  9        8        8  2019     2    30       NA N
## 10        8        9  2019     2    30       NA N
## # ... with 445 more rows
```

> **Ejercicio**: elimina del conjunto de datos `calidad_aire` todas las observaciones erróneas. Es decir, en aquellos meses con 30 días, no debería aparecer la observación con valor `dia = 31`. También hay que eliminar las observaciones que no tiene sentido del mes de febrero (cuidado con los bisiestos).

3.7 Medición en columnas

Si te fijas, tenemos una medición asociada a cada **día**, **magnitud** y **estación de medición**. Vamos a simplificar este conjunto de datos de forma que tengamos la **medición diaria media para cada magnitud**. Esto podemos hacerlo de la siguiente forma:

```
calidad_aire <- calidad_aire %>%
  group_by(magnitud, ano, mes, dia) %>%
  summarise(medicion = mean(medicion, na.rm = TRUE)) %>%
  ungroup()
```

Para terminar la preparación de los datos, nos gustaría que cada magnitud estuviese recogida en una variable. Primero, la variable `magnitud` está codificada mediante números que no son muy informativos. En la documentación aparece con qué magnitud se corresponde cada número. Lo podemos traducir de la siguiente manera:

```
unique(calidad_aire$magnitud)
```

```
##  [1]  1  6  7  8  9 10 12 14 20 30 35 42 43 44
```

```
calidad_aire$magnitud2 <- calidad_aire$magnitud

calidad_aire$magnitud2[calidad_aire$magnitud == 1] <- "so2"
calidad_aire$magnitud2[calidad_aire$magnitud == 6] <- "co"
calidad_aire$magnitud2[calidad_aire$magnitud == 7] <- "no"
calidad_aire$magnitud2[calidad_aire$magnitud == 8] <- "no2"
calidad_aire$magnitud2[calidad_aire$magnitud == 9] <- "pm25"
calidad_aire$magnitud2[calidad_aire$magnitud == 10] <- "pm10"
calidad_aire$magnitud2[calidad_aire$magnitud == 12] <- "nox"
calidad_aire$magnitud2[calidad_aire$magnitud == 14] <- "o3"
calidad_aire$magnitud2[calidad_aire$magnitud == 20] <- "tol"
calidad_aire$magnitud2[calidad_aire$magnitud == 30] <- "ben"
calidad_aire$magnitud2[calidad_aire$magnitud == 35] <- "ebe"
calidad_aire$magnitud2[calidad_aire$magnitud == 42] <- "tch"
calidad_aire$magnitud2[calidad_aire$magnitud == 43] <- "ch4"
calidad_aire$magnitud2[calidad_aire$magnitud == 44] <- "nmhc"

calidad_aire$magnitud <- calidad_aire$magnitud2
calidad_aire$magnitud2 <- NULL
```

Opcional: una buena práctica en programación es no repetir innecesariamente el código escrito. En el trozo de código que acabamos de escribir, esto no se cumple, haciendo un código difícil de leer y de corregir si se produce un error. Una forma algo más elegante de hacerlo podría ser:

```
magnitud_diccionario <- tribble(
  ~magnitud, ~magnitud2,
   1,  "so2",
   6,   "co",
   7,   "no",
   8,  "no2",
   9, "pm25",
  10, "pm10",
```

```
  12,  "nox",
  14,   "o3",
  20,  "tol",
  30,  "ben",
  35,  "ebe",
  42,  "tch",
  43,  "ch4",
  44,  "nmhc"
  )

calidad_aire <- calidad_aire %>%
  left_join(magnitud_diccionario)

calidad_aire$magnitud <- calidad_aire$magnitud2
calidad_aire$magnitud <- NULL
```

En realidad, cada magnitud debería ser una variable distinta. Es decir, nos gustaría tener una variable para `so2`, otra para `co`, etcétera. Para ello, hay que hacer la operación *contraria* a la que ya hicimos con `pivot_longer`:

```
calidad_aire <- calidad_aire %>%
  pivot_wider(names_from = magnitud,
              values_from = medicion
              )
```

Esto nos lleva a tener un conjunto de datos listo para empezar a trabajar la modelización:

```
calidad_aire
```

```
## # A tibble: 1,064 x 17
##      ano   mes   dia   so2    co    no   no2  pm25  pm10   nox    o3   tol   ben
##    <dbl> <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1  2017     1     1   9   0.48   33.6  48    22.3  24.8  99.5 13.8   2.54  1.18
##  2  2017     1     2  10.6 0.5    32.2  53.4  19.8  25.6 103.  16.1   2.96  1.1
##  3  2017     1     3  11.9 0.67   85.4  67.2  25.8  37.8 198.   7.69  5.48  1.62
##  4  2017     1     4  11.1 0.73  103.   68.7  26.8  40.6 227.   8.29  7.44  2.02
##  5  2017     1     5  10   0.733  89.1  65.8  24.2  34.6 202.   8.43  6.56  1.86
##  6  2017     1     6  11.5 0.64   71.6  58.8  15.8  21.2 169.  15.6   5.62  1.7
##  7  2017     1     7  11.8 0.62   67.2  64    15.5  22.3 167.  15     3.6   1.28
##  8  2017     1     8  12.2 0.66   64.2  65.7  19.7  24.6 164.  10.1   4.08  1.46
##  9  2017     1     9  12.9 0.722  95    76.1  16.3  25.8 222.   8.92  4.22  1.58
## 10  2017     1    10  15.5 0.76  104.   77.5  20.5  32.3 237.   8.69  7.2   1.8
## # ... with 1,054 more rows, and 4 more variables: ebe <dbl>, tch <dbl>,
## #   ch4 <dbl>, nmhc <dbl>
```

Por último, nos será útil tener una nueva variable fecha

```
calidad_aire$fecha <- as.Date(ISOdate(year = calidad_aire$ano,
                                      month = calidad_aire$mes,
                                      day = calidad_aire$dia
                                      )
                              )
```

4 Exportación de la información

Ya dijimos que cualquier proyecto de minería de datos va a estar dividido en etapas. Estas etapas, en función de cómo sean los datos, pueden tardar tiempo en ejecutarse. Por tanto, siempre que se concluye una etapa, se guardan los datos resultantes para no tener que volver a ejecutar el proceso. Además, recuerda que **R trabaja en memoria**, es decir, si cierras RStudio y vuelves a abrirlo, habrán desaparecido los datos con los que estábamos trabajando.

Para guardar en el disco duro un `data.frame`, lo podemos hacer de la siguiente forma:

```
saveRDS(calidad_aire, file = "taller_calidad_aire/data/01_transformacion.RDS")
```

Este archivo está en un **formato que solo entiende R** ya que todavía tenemos que seguir trabajando con esos datos mediante R. Si estuviésemos en la etapa final, lo adecuado sería guardar el archivo en un formato que pudiese leer cualquier programa (por ejemplo, `.csv`).