Islas Canarias
Del 15 al 19 de noviembre de 2021

Junta de Andalucía
Consejería de Transformación Económica, Industria, Conocimiento y Universidades
Instituto de Estadística y Cartografía de Andalucía

Tratamiento de grandes volúmenes de datos con tidyverse. Ejemplos a partir de la MCVL

Isabel Padilla Sánchez
Instituto de Estadística y Cartografía de Andalucía
isabel.padilla@juntadeandalucia.es

Joaquín Planelles Romero
Instituto de Estadística y Cartografía de Andalucía
joaquin.planelles@juntadeandalucia.es

1.-Introducción

En la última década la demanda y la oferta de datos se han expandido enormemente. Un reflejo de este proceso lo encontramos en el desarrollo de diversos paquetes informáticos, tanto de naturaleza libre como comercial, que facilitan el tratamiento de grandes volúmenes de datos. En este artículo nos vamos a centrar en algunas librerías de R específicas para estos fines.

Al trabajar con muchos datos, debemos resolver varias dificultades. Por un lado, el almacenamiento. Se necesitan máquinas potentes que almacenen la información. Otro aspecto importante a tener en cuenta es la eficiencia en el tratamiento. Al trabajar con grandes volúmenes de datos los tiempos de respuesta son mayores, lo que dificulta el trabajo. Esto pone de manifiesto la necesidad de disponer de programas eficientes para el tratamiento de la información y de la capacitación adecuada para poder usarlos.

R es uno de los lenguajes de programación más extendidos a nivel mundial. Además cuenta con una extensa comunidad de desarrolladores, que han ido creando librerías para problemas concretos (en el repositorio oficial de CRAN hay actualmente más de 18 mil librerías disponibles). En esta ponencia vamos a poner un ejemplo de uso de un conjunto de paquetes de R que resuelven parte de los problemas que surgen en el tratamiento de datos, utilizando para ello una sintaxis a la vez sencilla de entender y eficiente, el entorno ‘tidyverse’.

No obstante, R presenta ciertas limitaciones al trabajar con grandes volúmenes de datos, ya que consume recursos de la memoria volátil del ordenador. Existen diversas estrategias para resolver estas limitaciones. En esta ponencia se propondrá una alternativa posible, mediante el uso de la librería dbplyr, perteneciente también al entorno tidyverse. Esta librería funciona como un traductor a SQL. De este modo, es posible alojar la información en una base de datos externa y conectarse a la misma desde R.

2.-Objetivos

En esta ponencia se pretende dar respuesta a tres cuestiones. En primer lugar, se hace una propuesta respecto al tipo de informe técnico que debe acompañar al trabajo con fuentes de información no estructuradas, a la que denominamos 'relato metodológico'. En segundo lugar, se hablará del ecosistema de librerías de R llamado tidyverse, haciendo un especial énfasis en cómo la librería dbplyr, perteneciente a ese ecosistema, puede ayudar a resolver los problemas de memoria volátil típicos de R. Por último, se mostrará esta filosofía de trabajo mediante algunos ejemplos extraídos de la explotación que realizamos en el IECA de la Muestra Continua de Vidas Laborales. Se trata de un conjunto voluminoso de datos almacenados en una base de datos mediante tablas relacionales. Se mostrará cómo acceder a esos datos desde R, con una sintaxis a la vez sencilla y computacionalmente eficiente.

3.-Metodología

En el análisis de datos y particularmente si se trabaja con fuentes de información no-estructuradas (como puedan ser datos sensorizados procedentes de móviles, imágenes satélite o páginas web) o con fuentes semi-estructuradas (como los registros administrativos), te enfrentas a dos dificultades distintas: (1) Qué hacer. Se trata de establecer los objetivos de tu análisis y la metodología que vas a seguir. Esta dificultad tiene un carácter cognitivo y es propia del dominio al que pertenecen los datos. Por ejemplo, al ámbito del medio ambiente, el mercado de trabajo, la salud, etcétera. (2) Hacerlo. Esta segunda dificultad tiene un carácter computacional.

Por supuesto, una vez has pensado qué hacer y lo has implementado, también habrás adquirido un conocimiento adicional sobre el ámbito en estudio y sobre los datos que estás explotando. Te planteas nuevas preguntas y, poco a poco, a través de un proceso iterativo que te lleva de la dificultad (1) a la (2) y de vuelta a la (1) lo que empezó siendo un análisis exploratorio/rudimentario pasa a ser un auténtico análisis de datos.

En este tipo de contextos rara vez dispones de una metodología estandarizada que te permita explotar tu fuente de información, sino que es el propio proceso iterativo de aproximación a los datos el que acaba generando conocimiento. Es decir, hay un valor en el proceso en sí mismo, que pensamos debe quedar reflejado en la documentación que acompaña a la producción estadística, pues puede ser muy útil para otros actores. A esto le vamos a denominar 'relato metodológico' y es lo que nos proponemos hacer con esta ponencia.

3.1 tidyverse

Se denomina 'tidyverse' a un conjunto de librerías de R concebidas para Ciencia de Datos. En estas librerías están programadas una serie de funciones que facilitan el trabajo en las distintas fases del ciclo del dato: recogida de la información, tratamiento y difusión de resultados. Con ellas se puede hacer web-scrapping, tratamiento de textos, gráficos interactivos, informes automatizados… Se trata, además, de unas librerías de uso muy extendido, lo que facilita encontrar documentación para resolver cualquier problema concreto.

Las librerías que componen tidyverse comparten un diseño y una gramática comunes, lo que facilita la inter-operabilidad entre las distintas librerías/fases del ciclo del dato. Abundando en la relación dialéctica entre las dificultades (1) y (2) que mencionábamos

antes, la filosofía de tidyverse es simplificar al máximo la parte computacional, para poder dedicar el tiempo a lo que de verdad importa: pensar qué hacer. Tomemos el ejemplo de la librería dplyr, la librería de tidyverse pensada para manipular/tratar los datos. Esta librería dispone de 6 verbos/funciones principales (select, filter, arrange, mutate, group_by y summarise). Con estas 6 piezas, que conectamos entre si mediante tuberías[1] puedes realizar casi cualquier tratamiento de datos que imagines. Has simplificado el problema a construir tu lego (script) a partir de las 6 piezas de las que dispones. El código que te queda es sencillo y legible. También es interoperable con otras librerías de tidyverse, concebidas para otras fases del ciclo del dato: modelización, visualización, etc.

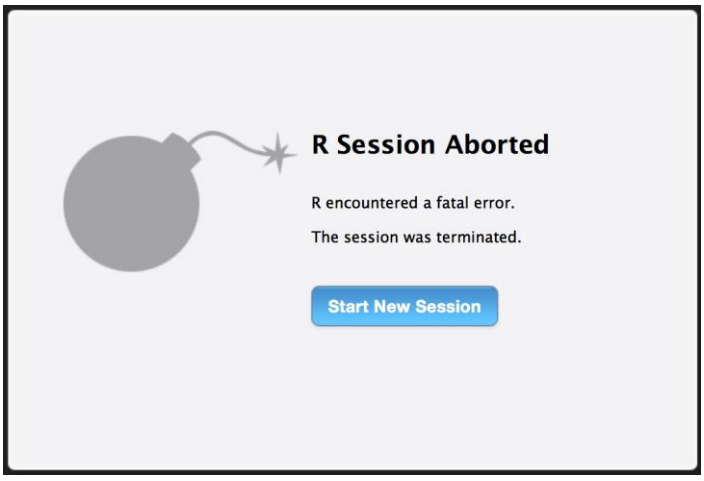
3.2 Problemas de memoria

Si trabajas con gran cantidad de datos desde R, no te serán ajenos mensajes como el siguiente en el que el programa te advierte de que el tamaño del set de datos le supera. El problema es que R tiene limitaciones al trabajar con grandes volúmenes de datos, ya que consume recursos de la memoria volátil (RAM) del ordenador.

```
> arquivo_atual_validado<-inner_join(arquivo_atual,arquivo_antigo_fi
ltrado,by="codigo_estado")
Error: cannot allocate vector of size 1.4 Gb
>
```

Incluso, cuando insistes en pedirle más de lo que puede ofrecerte, te puede llegar a aparecer un mensaje muy gráfico que te invita a salir y a reiniciar la sesión de R:

Figura 1: Sesión de R saturada.



Una situación no tan extrema como las anteriores, pero mucho más frecuente, se da cuando el volumen de datos no llega a colapsar tu ordenador, pero sí afecta a su rendimiento. ¿Qué hacer? En esta sección propondremos distintas estrategias de acción, aunque pondremos el énfasis en la conexión a bases de datos y en el uso de la librería

[1]Las tuberías son un operador que sirve para realizar varias operaciones de forma secuencial sin recurrir a paréntesis anidados o a realizar una operación en cada línea de código (con lo que eso supone en términos de extensión del script y de crear/sobrescribir multitud de objetos). Se representan mediante el símbolo %>%.

Así, lo que normalmente representaríamos como y=f(g(x)) con tuberías se representaría y = x %>% g() %>% f(), mucho más sencillo de interpretar cuando hablamos de consultas complejas realizadas contra una base de datos.

Además, al utilizar tuberías el output de un proceso es el input del siguiente, lo que simplifica la sintaxis, al no tener que nombrar cada vez el objeto que contiene los datos que vas a usar.

dbplyr. Como veremos, se trata de una solución elegante y muy potente, que te permite trabajar desde R con cientos de millones de registros, si es necesario. Pero primero asegúrate de que es la solución ideal para ti.

La solución más adecuada para cada caso dependerá de un amplio conjunto de factores: equipos informáticos, conocimientos y lo más importante, tiempo (tiempo de aprendizaje, tiempo de procesamiento y tiempo de elaboración de los scripts). Queremos subrayar que la mejor solución no será siempre la solución que computacionalmente hubiera sido más eficiente (de hecho, rara vez lo es). Quizás la mejor solución para tu problema no requerirá conectarte a una base de datos externa sino que te bastará con optimizar el script, o con ampliar tu memoria RAM. Dicho de otro modo, con la computación también opera una ley de rendimientos marginales decrecientes. Y hay costes de oportunidad.

He aquí algunas de las opciones a tu alcance para esquivar los problemas de memoria:

Opción 1. Empecemos por la mencionada extensión de la memoria RAM del equipo. Actualmente (año 2021) en los ordenadores personales disponibles en el mercado encontrarás opciones entre los 8 GB de un equipo básico hasta los 32 GB de un equipo premium/gamer. También hay ordenadores con mayor capacidad, pensados para trabajo profesional.

No obstante, es muy importante tener en cuenta una cosa. Frecuentemente lo que limita no es tanto la memoria RAM del equipo, sino la cantidad de esa memoria RAM que el equipo reserva a la sesión de R. Cuando trabajas en tu equipo y abres distintos programas, java asigna una determinada cantidad de memoria a cada uno. De ese modo, asignará a la sesión de R una pequeña parte de la memoria total, frecuentemente 256/512 MB. Esto puedes cambiarlo en cada sesión de trabajo. El comando para hacerlo, es el siguiente: options(java.parameters = "-Xmx4g") donde el final ‘4g’ está indicando que quieres reservar 4 gigas. Puedes indicar la cantidad que te sea más útil, teniendo en cuenta la capacidad de tu ordenador. Para que esto tenga el efecto deseado hay que indicarlo al inicio de la sesión de R, antes de cargar las librerías. Por ello, cuando nosotros creamos un script en el que hay un gran volumen de datos implicados, ubicamos el comando como el primer elemento del script,

Figura 2: Parámetro de Java modificado desde R.

```
options(java.parameters = "-Xmx4g")

#Cargamos las librerías necesarias
library(tidyverse)
library(readxl)

...
```

Opción 2. Esta será válida en determinados contextos, consiste en particionar esos datos que te resultan demasiado grandes, y enfrentarte a cada bloque (chunk) de información por separado. Si no puedes procesar de una vez los datos de toda España, pero sí puedes procesar los datos de cada provincia, quizás te resulta posible comenzar realizando una partición de tus ficheros y trabajar con cada sub-fichero por separado. De hecho, también

hay paquetes de R pensados específicamente para implementar este tipo de soluciones, como los paquetes LaF y chunked.

Esta idea no sirve únicamente para leer ficheros planos, también vale para otros grandes conjuntos de datos a los que accedemos a través de una API, o para comunicarse de forma rudimentaria con una base de datos.

Opción 3. Conectarse a bases de datos relacionales, dejando que sea el motor de la propia base de datos el que soporte la mayor parte del trabajo. En nuestro trabajo como productores de información estadística es muy frecuente recibir la información en un texto plano que unas veces te interesará leer directamente desde R, pero en otras ocasiones te interesará empezar por almacenar en una base de datos relacional. Los motivos que típicamente te llevan a ello son la seguridad (control de accesos, diferenciación de roles) y el tamaño (porque tu conjunto de datos sea más grande que la memoria RAM de tu ordenador, o lo suficientemente grande como para comprometer el rendimiento). De todo esto hablaremos con más detenimiento en el apartado 3.3.

Opción 4. Para determinados conjuntos de datos particularmente grandes las bases de datos relacionales tampoco resuelven el problema. O necesitarías unos servidores especiales, que quedan fuera de tu alcance. También se está produciendo en los últimos años un cierto cambio de paradigma en el almacenamiento de datos, pasando de almacenes estructurados (bases de datos relacionales) a no-estructurados (data lakes).

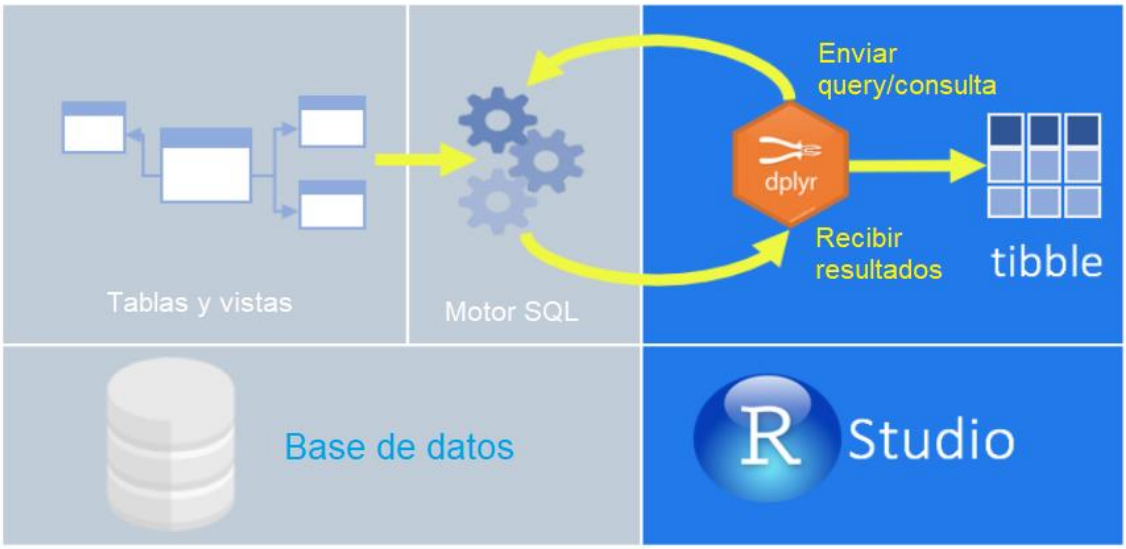
En estos casos, necesitas otro tipo de soluciones: clústers de ordenadores, programación en paralelo, etc. Ejemplos de esto son hadoop, apache spark y google BigQuery. Para ellos hay programas de R que te ofrecen soluciones similares a las que vamos a proponer para las bases de datos: primero, establecer la conexión a esos clústers y segundo, utilizar un programa que traduce tu código R/tidyverse al código que entienden estos clústers, externalizando en ellos el trabajo. Este es el caso de las librerías sparklyr y bigrquery.

3.3 Conexión a bases de datos y dbplyr

Como hemos comentado, una de las estrategias a seguir cuando tenemos volúmenes de datos grandes es alojar la información en una base de datos relacional, que lo que entenderá no es R, sino lenguaje SQL.

¿Qué hacer en estos casos? ¿Debes renunciar a los sencillos scripts que obtienes con tidyverse? No es necesario, porque se puede establecer una conexión a la base de datos directamente desde R. Y ahora viene lo mejor: una vez conectado puedes usar la librería dbplyr, que transforma el script de R en un script optimizado de SQL. Esta transformación se realiza, a no ser que se indique otra cosa, de un modo que es ciego para el usuario, que solo visualiza su script de R. De este modo, te puedes quedar con lo mejor de los dos mundos: la sencillez e interoperabilidad de tu código tidyverse y externalizar las operaciones más pesadas para que las realice directamente el motor de tu base de datos. Así liberarás la memoria RAM del ordenador y se gana en tiempos/velocidad de procesamiento.

Figura 3: Interacción entre Rstudio y la base de datos.



Algunos de los sistemas gestores de bases de datos (SGBD) de uso más extendido en la actualidad son PostgreSQL, MySQL, SQLite y ORACLE. Según el SGBD que utilicemos, tendremos un driver específico. Generalmente el driver es proporcionado por la empresa propietaria del gestor y tiene una labor importante de cara al usuario, pues gracias a ellos las migraciones de un SGBD a otro tan solo requieren cambiar el driver, ya que éste coloca una capa intermedia para abstraer la conectividad.

Para conectar R a tu base de datos, será necesaria la instalación previa del paquete DBI (acrónimo de DataBase Interface). Además, en el caso de usar software comercial necesitarás controladores OBDC (o si tu aplicación utiliza java JDBC).

Finalmente, una vez se tiene establecida la conexión de acuerdo a las características técnicas que se tengan, ya se puede trabajar desde la consola de R para extraer la información que interese de la base de datos sin necesidad de utilizar lenguaje SQL.

Llegados a este punto tenemos dos formas de trabajar. (1) trayendo la información de la base de datos y trabajar con esa información desde R. No obstante, esto sería ineficiente, ya que la consola quedaría inundada de datos. (2) comunicarse con la base de datos en su idioma (SQL) pidiéndole que ejecute el trabajo pesado y traer a R solo el resultado de ese proceso. Esto se puede llevar a cabo de dos maneras:

- Mediante la función dbSendQuery() de la librería DBI. Puedes escribir trocitos de código en SQL insertados en medio de tu código R y lanzarlo a la base de datos. He aquí un ejemplo de esta forma de trabajar:

Figura 4: Ejemplo dbSendQuery().

```
AFI<- dbSendQuery(con, paste0("CREATE TABLE AFI AS
                          SELECT pk_id,IDPF,GRUPCOT,
                          CASE
                            WHEN  GRUPCOT not in ('01','02','03','04','05','06','07','08','09','10','11','12') THEN '00'
                            ELSE GRUPCOT
                            END AS GRUPCOT_R
                          FROM MCVL_AFILIACION
                          WHERE LOTE=",ANOREF," AND substr(FALTA, 1, 4)<", ANOREF+1," AND substr(FBAJA, 1, 4)>",ANOREF-1))

AFI<- tbl(con, "AFI")
```

- Mediante la librería dbplyr escribiendo el código íntegramente en R. La librería se encarga de hacer la traducción a SQL y comunicarse con la base de datos. La primera versión de la librería dbplyr se lanzó a finales del año 2017 y actualmente es uno de los campos en los que más está evolucionando el ecosistema tidyverse[2].

Figura 5: Ejemplo librería dbplyr.

```
AFI<- tbl(con, "MCVL_AFILIACION")%>%
                  filter(LOTE==ANOREF) %>%
                  filter(substr(FALTA, 1, 4)< ANOREF+1 & substr(FBAJA, 1, 4)>ANOREF-1) %>%
                  mutate(GRUPCOT_R=if_else(!GRUPCOT%in%c('01','02','03','04','05','06','07','08','09','10','11','12'),'00',GRUPCOT))%>%
                  select(PK_ID,IDPF,GRUPCOT,GRUPCOT_R)
```

Este segundo modo de lanzar las consultas a la base de datos presenta algunas ventajas. (1) Generas tu código íntegramente con sintaxis dplyr, enlazando una acción con la siguiente mediante tuberías. De este modo, obtienes un código compacto y fácil de leer. `dbplyr´se encargará de traducir esto a una consulta de SQL. (2) dbplyr traduce a distintos dialectos de SQL, así que no tendrás que preocuparte por el dialecto concreto que entiende tu base de datos. (3) no te hace falta conocer SQL. O no hace falta ser un experto en SQL para programar consultas complejas, te bastará con los verbos/funciones básicos de dplyr y las tuberías.

4.-Resultados

En este apartado se mostrará un ejemplo de uso de tidyverse con un gran conjunto de datos, la MCVL.

4.1 Descripción de la MCVL

Los datos de la Muestra Continua de Vidas Laborales proceden de una selección aleatoria de los afiliados y/o pensionistas de la Seguridad Social en el año de referencia. Para ellos, la Seguridad Social incorpora información relativa a cada episodio de cotización y de cobro de prestaciones. A estos datos se les añaden otros procedentes del Padrón Continuo Municipal (INE) y de la Agencia Tributaria.

El proceso de selección da lugar a una muestra de 1,2 millones de personas para España de las que unas 200 mil son residentes en Andalucía, con la particularidad de que para cada una de esas personas se incorpora todo su historial de relaciones con la Seguridad Social, no sólo las del año de la edición[3].

El modelo de datos que recibimos consta de un conjunto de tablas que se pueden cruzar entre sí mediante identificadores comunes. Para hacerse una idea de la magnitud de datos de la que estamos hablando, la tabla más grande de la MCVL (afiliaciones) contiene unos 27 millones de registros en cada edición anual y 37 dimensiones/variables por registro. En total, unos mil millones de datos. Solo en esta tabla, solo en esta edición.

---

[2]En la web https://db.rstudio.com/r-packages/odbc/ se puede ampliar información acerca de la conexión y de la librería dbplyr.

[3]Para una descripción detallada de las características de la MCVL y de los distintos tipos de datos que se pueden producir con ella, remitimos a la ponencia “La MCVL como valor añadido a las estadísticas laborales. Explotación retrospectiva, transversal y longitudinal”, presentada en las JECAS de 2018.

Si cargamos todos los datos de las distintas tablas y ediciones (desde 2006), estamos hablando de unos 30 mil millones de datos. Y aquí viene, sin darse uno mucha cuenta, una de las decisiones más importantes de tu análisis de datos. ¿Cómo vas a almacenarlos? Nosotros hemos almacenado los datos de la MCVL en una base de datos relacional de ORACLE. Y lo más importante, hemos respetado la estructura de datos que recibimos de la Seguridad Social, con la salvedad de que incluimos en cada tabla un nuevo campo que representa el lote/año de la edición. Optar por mantener la estructura de datos tal cual llega tiene ventajas e inconvenientes.

Empecemos por los inconvenientes. Al actuar así no estás optimizando el espacio que ocupa tu proyecto. Tu base de datos va a pesar más de lo que pesaría si le dedicaras tiempo a optimizar el almacenamiento, pues gran parte de la información que recibes con cada lote anual de la MCVL es redundante con la información que habías recibido en lotes anteriores. Incluso puedes pensar que dentro del mismo lote puedes idear una estructura de tablas más comprimida que la que recibes.

No obstante, mantener la estructura de datos tal cual llega ahorra tiempo en otras fases del ciclo del dato. Ya de partida no le tienes que dedicar tiempo a idear cómo debes estructurar tus tablas. En segundo lugar, los procesos de carga serán más sencillos y rápidos, pues la nueva información que introduces es independiente de la que ya tuvieras cargada anteriormente. En tercer lugar, tu modelo de datos se va a adaptar con naturalidad a los cambios que se produzcan en la estructura de datos que recibes. Finalmente, te será más fácil extraer/explotar la información que tienes almacenada, incluso para finalidades distintas a las que habías imaginado inicialmente.

En definitiva, las dos opciones (alterar la estructura del modelo de datos o mantenerlo tal cual) tienen costes y ventajas. La decisión ‘ideal’ dependerá del proyecto en concreto en el que estés trabajando y de los recursos que estén a tu alcance. Como marco general, pensamos que es una buena política simplificar la parte computacional del problema (cómo hacer las cosas) y concentrar los recursos en la parte cognitiva (qué hacer).

4.2 Ejemplos de uso

En este apartado se mostrará un ejemplo de la filosofía de trabajo empleada en la explotación de los datos de la MCVL del IECA mediante scripts de R. En el código no se observa diferencia alguna, pues todo está escrito en lenguaje dplyr pero una parte del mismo es código SQL traducido por dbplyr y la otra, dplyr en sí.

De cara al futuro, nos gustaría ubicar los scripts completos en un espacio compartido como github, de modo que podamos compartir, para facilitar un espacio de intercambio con otros usuarios/analistas.

La primera parte del código es fundamental ya que aquí se cargan las librerías necesarias en el script y se establece la conexión con las base de datos. Los datos de la MCVL se encuentran alojados es ORACLE. Por ello, las librerías de R necesarias son tidyverse (esta librería carga los ocho paquetes que forman el núcleo del ecosistema del mismo nombre, tidyverse), DBI (librería necesaria para establecer la conexión con ORACLE), RJDBC (permite ejecutar operaciones sobre bases de datos desde Java) y por último el paquete dbplyr que tiene implementadas la instrucciones para traducir el código tidyverse a SQL.

Finalmente, para el caso particular de la conexión a Oracle es necesario añadir unas últimas líneas de código, que son las que aparecen al final de la Figura 6 y que facilitan que dbplyr pueda traducir todas la sentencias. Sería de esperar que en futuras versiones de dbplyr ya no será necesario incluir estas sentencias, y que estos aspectos más técnicos relacionados con la conectividad los gestione la propia librería dbplyr. No obstante, para trabajar con Oracle y a día de hoy, siguen siendo necesarias.

Figura 6: Carga de librerías y conexión a la Base de Datos.

```
#Cargamos las librerías
library(tidyverse)
library(RJDBC)
library(DBI)
library(dbplyr)

#Establecemos la conexión con ORACLE:
driver <- JDBC("oracle.jdbc.OracleDriver", classPath="./ojdbc6.jar")
Err<-try(con<- dbConnect(driver, "jdbc:oracle:thin:@miservidor01:1521:MIBD", "BASEDATOS", "CLAVE"),silent=FALSE)

#La conexión JDBC actual apunta a la traducción de Oracle dentro dbplyr:
sql_translation.JDBCConnection <- dbplyr:::sql_translation.Oracle
sql_select.JDBCConnection <- dbplyr:::sql_query_select.Oracle
sql_subquery.JDBCConnection <- dbplyr:::sql_query_wrap.Oracle
```

Con esto, ya tendríamos todo lo necesario para trabajar contra la base de datos y extraer la información deseada. A continuación vamos a proponer un ejemplo, extraído de las propias tablas en las que tenemos alojadas las ediciones 2006-2019 de la MCVL. El código R se muestra en la Figura 7. Nótese que en este ejemplo estamos realizando una secuencia de acciones, cada una de las cuales implica grandes volúmenes de datos.

1. En primer lugar, se utilizan dos de las tablas de la MCVL. La mayor de ellas, MCVL_AFILIACIONES, tiene más de 300 millones de registros (14.000 millones de datos), mientras que la 'pequeña', MCVL_PERSONAS, tiene 17 millones de registros.

2. En ambas tablas se filtran los registros para el último lote disponible (2019) y en el caso particular de la de afiliación, se seleccionan los episodios activos en el año de referencia y se recodifica la variable grupo de cotización (GRUPCOT).

3. Se cruzan las dos tablas entre sí, a partir del identificador de la persona, para añadir la variable SEXO al fichero "afi obtenido".

4. Se agrupa por las variables sexo y grupo de cotización, obteniendo un estadístico resumen (conteo de casos).

5. Se cargan los datos en la consola de R mediante el comando collect().

6. Finalmente, ya en R, se hacen nuevas transformaciones: se recodifica la variable SEXO, se pasa la a columnas y se ordena, algo así como si estuviéramos trabajando con tablas dinámicas.

Figura 7. Código dbplyr/dplyr con los datos de la MCVL

```
afi <- tbl(con, "MCVL_AFILIACION") %>%
       filter(LOTE==ANOREF) %>%
       filter(substr(FALTA, 1, 4)< ANOREF+1 & substr(FBAJA, 1, 4)>ANOREF-1) %>%
       mutate(GRUPCOT_R = if_else(!GRUPCOT %in% c('01','02','03','04','05','06','07','08','09','10','11','12'),'00',GRUPCOT)) %>%
       select(PK_ID,IDPF,GRUPCOT,GRUPCOT_R)

personas <- tbl(con, "MCVL_PERSONAS") %>%
            filter(LOTE==ANOREF) %>%
            select(IDPF, SEXO)

afi_ampliado <- inner_join(afi,personas,by='IDPF')

episodios <- afi_ampliado %>% group_by(GRUPCOT_R,SEXO) %>% summarise(n=n()) %>% collect()

episodios <- episodios %>% mutate(SEXO=recode(SEXO,'1'='Hombres','2'='Mujeres')) %>%
                           pivot_wider(names_from=SEXO, values_from=n) %>%
                           rename(Grupo_cot=GRUPCOT_R) %>% arrange(Grupo_cot)
```

El resultado de todo este proceso se puede visualizar en la Figura 9. El tiempo total que hemos tardado en ejecutar esta consulta es de 37 segundos (Figura 8). Además, si haces una consulta más compleja o incluso un script de explotación completo los tiempos no son mucho mayores a esos 37 segundos.

Figura 8. Tiempo de procesamiento del código de la Figura 7

```
   user  system elapsed
   0.34    0.10   36.94
```

Figura 9. Resultados obtenidos de la consulta de la Figura 7

Grupo_cot	Hombres	Mujeres
00	108680	87263
01	52352	53752
02	34714	73966
03	52936	32323
04	29094	26889
05	63061	88492
06	40873	49331
07	95383	176666
08	231919	96204
09	151914	118170
10	319119	268650
11	844	459

4.-Conclusión

En esta ponencia se ha sugerido que en el análisis de datos hay dos componentes. El primero tiene un carácter cognitivo (qué hacer), mientras que el segundo tiene un carácter computacional (cómo hacerlo). La relación entre ellos es dinámica: te aproximas a los datos, realizas una primera explotación, adquieres un mejor conocimiento de tus datos y te planteas nuevas preguntas. En el trabajo con fuentes de información no-estructuradas o semi-estructuradas este proceso supone gran parte del valor final y debería tener una mayor presencia en la documentación que acompaña a la producción estadística.

En segundo lugar, se ha hablado de cómo trabajar con grandes conjuntos de datos con una sintaxis que sea a la vez sencilla y computacionalmente eficiente. Es decir, teniendo en cuenta los dos componentes del análisis de datos. Para ello, se ha propuesto utilizar un conjunto de librerías de R de uso muy extendido, denominado tidyverse. Para resolver las limitaciones de R con la memoria volátil (RAM) se ha propuesto el uso de la librería dbplyr, perteneciente también al entorno tidyverse. Esta librería funciona como un traductor a SQL. De este modo, es posible alojar la información en una base de datos externa y conectarse a la misma desde R.

5.-Bibliografía

Grolemund, Garrett & Wickham, Hadley (2014). A cognitive interpretation of data analysis. International Journal of Statistics. Vol 82(2). pp 184-204

Padilla, Isabel & Planelles, Joaquín (2018). La MCVL como valor añadido a las estadísticas laborales. Explotación transversal, retrospectiva y longitudinal. XX JECAS

Wickham, Hadley & Grolemund, Garrett (2014). R for data science. O'Reilly Media.