Parte III
Conceptos fundamentales

Capítulo 1
Propiedades de estructuras de datos

El término variable define un modo de acceder a información almacenada en la memoria de la PC. En R una variable hace referencia a un objeto, una de las estructuras de datos vistas en el Módulo 1. Cada estructura es de un tipo en particular y el tipo determina el modo de almacenamiento y también el modo como opera. Es importante, pues, poderconocer el tipo de una estructura de datos. Este es el tema de este capítulo.

Las estructuras de datos pueden estar constituidas por elementos de diferente tipo, por ejemplo, cadenas de caracteres o números enteros. Por otra parte, las estructuras tienen asociadas atributos que almacenan metadatos pertinentes a ellas. El modo como está conformada una estructura de datos y sus atributos son propiedades. En muchas situaciones es necesario conocer el tipo de dato con que estamos operando y también cuales atributos están disponibles. En este capítulo se ve cómo inspeccionar propiedades de datos y cómo modificarlas para adecuar el dato a determinadas operaciones.

1.1. Examinar y modificar propiedades

R ofrece modos de revelar al analista las propiedades de las estructuras con que opera. Adicionalmente, las propiedades de una estructura en ciertos casos pueden ser modificadas y la modificación puede darse automáticamente, go- bernada por R, o explícitamente, determinada por el analista. A continuación se presentan las principales propiedades de datos y los recursos para operar con ellas.

1.1.1. typeof(), mode()

Las funciones typeof() y mode() tienen similares propósitos, describir la manera como R almacena un determinado dato, y ambas devuelven cadenas de caracteres. Posibles resultados de aplicar typeof() son: logical, integer, double, complex, character (cinco tipos de datos vectoriales), list y NULL. mode() da resultados similares con la excepción de que no distingue entre integer y double, asigna ambos tipos a numeric. Tres ejemplos

> typeof(pi) devuelve double, en cambio

> mode(pi) devuelve numeric

> typeof(c('Hola', 'como','le','va?')) devuelve character

> mode(c('Hola', 'como','le','va?')) también.

Aplicado a un marco de datos, como por ejemplo mtcars_tb_tb, tanto typeof(mtcars_tb_tb) como mode(mtcars_tb_tb) devuelven list porque cada fila es una lista. En presencia de un número escrito sin separador decimal, la tendencia normal es interpretarlo como un número entero; esto es particularmente cierto para años. Para R, sin embargo, un número es real a menos que se especifique lo contrario, como muestra el ejemplo

> fecha <- 2009

> typeof(fecha) devuelve double.

En R, para asegurarse de que un número sea interpretado como entero debe agregarse el sufijo L

> fecha <- 2009L typeof(fecha) devuelve integer. El sufijo L denota un número entero.

En el uso general, mode() es menos útil que typeof().

1.1.2. length()

El número de componentes de una estructura se recupera con length().

> length(pi) devuelve 1

> length(c(2,3,4) devuelve 3

> length(c('Hola', 'como','le','va?')) devuelve 4.

Tener en cuenta, sin embargo, que length() aplicada a una matriz o array devuelve el número de columnas y no el total de elementos. Probar

> length(mtcars_tb_tb) devuelve 11

length() es también una función activa que puede alterar el largo de un vector o lista. Hacer a <- c('Hola', 'como','le','va?'), , con 4 elementos, e imponerle un largo mayor con length(a) <- 6. Si se revisa a se apreciará que ahora contiene 6 elementos, dos de los cuales son NA. Un desafío es remplazar los NA por "Bien, gracias".

Nota:
mtcars_tb_tb es la versión tibble de mtcars_tb dataframe.

1.1.3. str(), summary()

Las funciones vistas brindan información específica acerca de las estructuras de datos, en tanto que str() y summary() son más complexivas. Escribir

> str(mtcars_tb_tb) e inspeccionar la salida. Devuelve el tipo de estructura y la dimensión; cada variable va precedida por el signo $ y se da el tipo de dato que contiene y los primeros valores

```
tibble [32 × 4] (S3: tbl_df/tbl/data.frame)
$ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
$ wt  : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
```

En cambio

> summary(mtcars_tb_tb) brinda un resumen estadístico de cada variable, como se muestra parcialmente abajo.

```
mpg              cyl              disp
Min.   :10.40   Min.   :4.000   Min.   : 71.1
1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8
Median :19.20   Median :6.000   Median :196.3
Mean   :20.09   Mean   :6.188   Mean   :230.7
3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0
Max.   :33.90   Max.   :8.000   Max.   :472.0
```

Notar la presencia de estadísticos aplicables a distribuciones asimétricas, como cuartiles y rango. Esto es útil pues muchas distribuciones de datos no se ajustan a una distribución normal.

En resumen, **str()** informa sobre la estructura del objeto, indicando las clases y tipos de datos y **summary()** devuelve un conjunto de estadísticos de cada variable.

1.1.4. class() y class

Al presentar los atributos se mencionó class, que define la clase de la cual hereda el objeto inspeccionado. Existe también la función class(), que devuelve la clase de un objeto. Unos ejemplos aclararán la diferencia.

class(mtcars_tb_tb) devuelve tbl_df y data.frame, clases de las cuales hereda mtcars_tb_tb. Y aplicado a un vector numérico class(c(1,2,3,4) devuelve numeric. Veamos una aplicación de class() que ilustra diferencias entre dataframe y tibble. Se inspecciona la clase de una columna de mtcars_tb_tb, es decir, de un vector

>class(mtcars_tb_tb[,4]) devuelve tbl_df y data.frame, lo cual no debiera sorprendernos.

Pero si se inspecciona el dataframe mtcars_tb_df

class(mtcars_tb_tb[,4])

devuelve numeric, y faltan los nombres de filas. Para el tibble el vector sigue siendo un tibble mientras que para el dataframe el vector deja de ser un dataframe.

1.1.5 Funciones genéricas

El uso de class() es mayormente interno a R, y tiene raíces en la programación orientada a objetos (POO), tema que no se ve en este curso. Muy brevemente, una clase describe las propiedades de un objeto y, en consecuencia, determina cómo se ha de comportar ese objeto. Así, un objeto resulta de instanciar (término propio de la POO) una clase, es decir, activarla. A su vez, el comportamiento de un objeto está gobernado por métodos. En la sintaxis objeto y método van separados por un punto, objeto.metodo. En R el uso de POO no es fuerte pero estos conceptos sirven para explicar el uso de ciertas funciones como print(), plot() y summary().

Estas funciones se denominan genéricas en R, debido a que pueden tener

comportamientos diferentes según sea el uso que se les da. R determina el comportamiento fijándose en la clase, o mejor dicho, en el argumento de esa clase. Supongamos el objeto x con los argumentos arg1 y arg2. Cuando se llama print(x) R busca la función print.arg1() y la ejecuta si existe, de lo contrario busca print.arg2(), y la ejecuta si existe. Si ambas faltan busca print.default(), siempre presente pero que puede no ser la adecuada, en cuyo caso resultará error. Información más extensa sobre este tema hallarán en el documento R Language Definition.

1.2. Coerción

Nivel de Complejidad

El material en esta sección es apropiado para un nivel intermedio deconocimiento de R.

Al principio de este capítulo se dijo que las propiedades de datos pueden modificarse, y que la modificación puede ser implícita o explícita. La acción de modificar una estructura se denomina coerción. La coerción implícita se da cuando R detecta un problema en la asignación de una estructura e intenta resolverlo modificándola. Este caso se da si se intenta mezclar tipos de datos en un vector atómico. Crear el vector mixto

> v.mixto <- c(5, 'cinco')

Al pulsar Enter R lo toma sin queja, en aparente violación del principio de atomicidad. Pero al visualizar el contenido con

> v.mixto

R devuelve el dígito como un carácter, '5'. Para cumplir con la regla de atomicidad R fuerza, coerciona, al dato numérico a carácter, por eso aparece entre ". La coerción opera de modo que todos los elementos son forzados a adoptar el modo del elemento de mayor generalidad. En orden de menor a mayor generalidad: logical, numeric, complex y character. Este modo de coerción, llevado a cabo automáticamente por R, se denomina coerción implícita.

Ejemplo

Se desea calcular el promedio del vector lógico

> a <- c(TRUE, TRUE,FALSE,TRUE)

> mean(a) devuelve 0.75.

¿Cómo puede ser? Pues porque R coercionó los valores lógicos a números enteros, 0 1, para satisfacer las exigencias de mean().

Otro ejemplo

> x <- 1:16

> typeof(x)

> class(x)

Se convierte x a matriz

> dim(x) <- c(4, 4)

> typeof(x)

> class(x)

El tipo se mantiene integer pero la clase fue coercionada a matriz.

1.2.1. funciones as.algo

También puede darse una coerción explícita, a voluntad del analista. La familia de funciones con prefijo as.: as.integer(), as.character(), as.numeric(), ad.complex(), as.logical(), tienen este fin. Retomar el vector lógico

> a <-c(TRUE, TRUE,FALSE,TRUE) en Global Environment se inscribe como logi [1:4] TRUE TRUE FALSE TRUE. Mientras que

> f <- as.integer(a) en Global Environment se inscribe int [1:4] 1101, as.integer() convirtió los valores lógicos en los enteros que internamente los representan. Hacer

> b <- 5

> typeof(b) devuelve double.

Coercionar a entero

> c <- as.integer(b) en Global Environment la variable c se inscribe como 5L, indicador de número entero.

Crear el factor

> sexo <- factor(c('macho','hembra','hembra','macho'))

> sexo devuelve los componentes y el atributo levels. Seguidamente se lo coerciona con

> as.character(sexo) con lo que pierde el atributo levels y sólo queda un vector de caracteres. O también con

> as.numeric(sexo) que coerciona al factor a un vector de números enteros, en este caso los que usa R para registrar los componentes del factor.

Ejemplo del uso de coerción explícita. En ocasiones se desea operar matemáticamente con un factor. Se tiene el factor compuesto por años

> evento <- factor(c(1990, 1983, 1977, 1998, 1990))

tiene atributo Levels: 1977 1983 1990 1998 y se desea restar 1983 de 1990. La operación de sustracción edad[1] – edad[2] no es válida con factores y da error. La coerción a numérico devuelve los números enteros con que se almacenan los años, por lo cual no resuelve el problema. La solución es coercionar a carácter y luego a numérico

> a <- as.numeric(as.character(edad)) y

> a[1] – a[2] <- 7

1.2.2. funciones is.algo

Una familia de funciones con prefijo is. permiten determinar el tipo de elementos en un objeto. Habiendo ejecutado

> c <- as.integer(5)

> is.integer(c) devuelve TRUE.

Un caso especial de uso de is. es is.na(), que detecta elementos faltantes en una estructura. Escribir

> a <- c(T,T,T,F,NA)

> is.na(a) devuelve FALSE FALSE FALSE FALSE TRUE

Al presentar la estructura factor se dijo que R automáticamente genera variables de factor con aquellas variables categóricas representadas por cadenas de caracteres. Ahora se tienen las herramientas para demostrar esto. El marco de datos ToothGrowth, incorporado en R, resume resultados de un experimento sobre el efecto de la vitamina C en el crecimiento de dientes en conejillos de Indias. Escribir

> head(ToothGrowth) para inspeccionar la tabla.

En la tabla original, en formato ASCII, la columna supp contenía los elementos categóricos OJ y VC. Al convertir la tabla a marco de datos esa columna fue coercionada a factor con dos levels, como se aprecia con

> str(ToothGrowth).

La tabla abajo resume funciones útiles para la inspección de vectores, matrices y marcos de datos. Varias de estas funciones se han visto ya y otras se verán más adelante. Se las ha agrupado por función principal.

1.3. Inspección de conjuntos de datos

head(), **tail()** devuelven las primeras, o últimas, 6 filas de la tabla. Se puede modificar el número de filas con head(nombre, num).

1.3.1. Tamaño de una estructura

dim() devuelve un vector con número de filas y de columnas **nrow()**, **ncol()** devuelven el número de filas, número de columnas **nchar()** devuelve el número de caracteres en un vector de cadena de texto

1.3.2. Nombres de filas y columnas

Ciertas operaciones insertan nombres de filas o columnas por defecto, o dado un nombre en particular se desea cambiarlo. Se ven aquí las funciones colnames() y rownames(), del paquete base, que permiten inspeccionar, modificar y remplazar nombres de filas y columnas.

colnames(marcodedatos) devuelve los nombres de las columnas y **rownames(marcodedatos)** los de las filas.

rownames() devuelve los nombres de filas

Las funciones colnames() y rownames() también permiten asignar nombres a las columnas y filas

colnames(marco de datos) <- c("Nombre1","Nombre2")

Probar con el siguiente ejemplo

> colnames(iris)
> a<-iris
> colnames(a) <- c("SL","SW","PL","PW","Sp")
> colnames(a)

Capítulo 2

Manejo de fecha y hora

Un adecuado manejo del tiempo es esencial en el análisis de series temporales. En R, el tiempo suele contarse a partir de un inicio arbitrario en días y en segundos. Ciertas funciones operan sólo con fechas, otras sólo con tiempo, y otras con fecha y tiempo. El propósito de este capítulo es introducir recursos que ofrece R para convertir fechas y horas a formatos numéricos. Se ven las funciones POSIX() y as.Date(), integradas en R, y los paquetes externos chron, hms y lubridate.

Para empezar se muestra cómo recuperar la fecha actual

```
> fecha_hoy <- Sys.Date()
```

2.0.1. as.Date()

Las fechas comúnmente se presentan como cadenas de caracteres, por ejemplo, ’2020-03-27’. De modo similar suele presentarse el tiempo, ’22:30:23’. Tales presentaciones no se prestan para operaciones matemáticas. Otro inconve- niente con fechas es la variedad de formatos en que aparecen. 20 Abril 2020, 20/abril/2020, 20/Abril/20, Abril 20, 2020, 04-20-2020, y así otros.

La clase Date se usa para operar con fechas; no procesa el tiempo: horas-minutos-segundos. La función as.Date(), incluida en R, permite convertir la fecha en caracteres en un formato numérico. Si la fecha está ya en formato internacional: ’2020-03-27’, la conversión es inmediata; si no es así hay que acondicionarla.

Ejemplo

Crear el dato

```
> Fecha1 <- ’2020-03-27’ y verificar que tiene clase character
> class(Fecha)
```

[1] ’character’

Seguidamente convertir Fecha a la clase Date

```
> Fecha_Date <- as.Date(Fecha)
> class(Fecha_Date)
```

[1] ’Date’

Ahora se puede operar con Fecha1

```
> Fecha_Date2 <- Fecha_Date + 30
```

Crear otra fecha y hallar la diferencia entre ambas

```
> Fecha3 <- ’2021-03-27’
```

```
> Fecha_Date3 <- as.Date(Fecha3)
> Fecha_Date3 - Fecha_Date
```

devuelve 365 días

Si no estuviera en formato internacional hay que indicar cómo está ordenada la fecha empleando los códigos en la Tabla2.1.

Ejemplos

Convertir fechas en diferentes formatos

```
> as.Date('1/15/2001', format<-'%m/%d/%Y')
[1] '2001-01-15'
> as.Date('April 26,2001', format<-'%B%d,%Y')
[1] '2001-04-26'
> as.Date('22JUN01',format<-'%d%b%y')
[1] '2001-06-22'
```

2.0.2. paquete lubridate

Nivel de Complejidad

Las funciones en el paquete lubridate son fundamentales para el manejo de fechas. El resto del material en este capítulo es apropiado para un nivel intermedio de conocimiento de R.

El paquete lubridate, incluido en el superpaquete tidyverse, contiene funciones especializadas en el manejo algebraico de fechas. Aunque lubridate se instala

Codigo	Valor
%d	Dia en número
%m	Mes en números
%b	Mes en letras abreviado
%B	Mes en letras completo
%y	Año 2 dígitos
%Y	Año 4 dígitos

Tabla 2.1

con tidyverse, debe activarse por separado con library(lubridate).

Una de las funciones es ymd(), que convierte un vector escrito en el formato año mes día, a un vector de clase POSIX en formato internacional, '2020-30- 03'. En realidad ymd() es una familia de funciones que se diferencian por el orden de los componentes de la fecha: ymd(), dmy(), mdy(). ymd() supone que se usan 2 dígitos para año, mes y día. Dado el vector con fechas

```
> fechas <- c('09-01-21', '09-01-22', '09-01-23')
```

> ymd(fechas) devuelve las fechas en formato internacional y como clase Date.

Una cualidad conveniente es que ymd() no requiere separadores. Dado

> fechas <- c(’090121’, ’090122’, ’090123’)

> ymd(fechas)

devuelve idéntico resultado.

Se puede alterar el orden de los componentes en las fechas ingresadas, pero se debe alterar el orden en el comando

> fechas <- c(’210109’, ’220109’, ’230109’)

> dmy(fechas)

2.0.3. paquete chron

La función chron convierte fecha y hora a objetos chron. No considera los husos horarios. Almacena el tiempo como fracciones de días; por ello tiene menor precisión que POSIX, que lo almacena en segundos. El formato de entrada típica para fechas es: 03/27/2020, y para el tiempo es: 23:30:23. Otros arreglos requieren que se los indique empleando los códigos que da la Tabla 2.2. Antes de procesar deben separarse fecha y tiempo; para ello se usa la función strsplit().

Formatos para fechas	
Codigo	Valor
m	Mes en número
d	Mes en números
y	Mes en letras abreviado
mon	Mes en letras completo
month	Año 2 dígitos
Formatos para el tiempo	
Codigo	Valor
h	Hora
m	Minuto
s	Segundo

Tabla 2.2

Ejemplo

Dada fecha tiempo como única cadena

> FechaHora <- ’2002-06-09 12:45:40’

Separar los componentes en dos cadenas

> strsplit(FechaHora,’ ’)

devuelve Fecha y Hora en cadenas separadas: ’2002-06-09’ ’12:45:40’.

2.0.4. paquete hms()

El paquete hms está dedicado al manejo del tiempo. El tiempo suele darse en caracteres como ’H:M:S’ Hora:Minuto:Segundo.

hms() permite convertir a un formato procesable numéricamente, en segundos. La sintaxis es
hms(seconds <- NULL, minutes <- NULL, hours <- NULL, days <- NULL) Los argumentos deben ser todos de dos dígitos.

2.0.5. POSIX

Una fecha se puede transformar a número de días transcurridos desde una fecha de origen. POSIX almacena el tiempo en segundos. POSIX puede convertir objetos de la clase Date. Si se le da una fecha sin especificar hora, supone que la hora es la medianoche UTC (Universal Coordinated Time Zone). POSIX permite

Código	Valor	Código	Valor
%a	Día semana abrev.	%A	Día semana completo
%b	Mes abrev.	%B	Mes completo
%c	Fecha y hora local	%d	Fecha en número
%H	Hora (24 horas)	%I	Hora en número
%j	Día del año en número	%m	Mes en número
%M	Minuto en número	%p	Locales specific AM-PM
%S	Segundo en número	%U	Semana del año (inicia en Domingo)
%w	Semana del año (inicia en Domingo)	%W	Semana del año (inicia en Lunes)

%x	Fecha local	%X	Tiempo local
%y	Año 2 dígitos	%Y	Año 4 dígitos
%z	Desfase de GMT	%Z	Huso horario

Tabla 2.3

Hay dos funciones POSIX distinguidas por el sufijo, POSIXlt y POSIXct,, donde lt representa local time y ct calendar time. POSIXlt almacena una lista con día mes año hora minuto segundo, lo cual facilita trabajar con cada componente por separado. POSIXct almacena las fechas en segundos a partir del 1 de enero de 1970. El formato de entrada de fechas es año-mes-dia o año/mes/dia. Y para el tiempo, hora:minuto:segundo, o hora:minuto.

Ejemplos

El formato internacional para fechas tiempo se muestra en los siguientes ejemplos
Sin segundos 2020-03-27 11:25

Con segundos 2020-03-27 11:25:05

Si el formato de entrada no se adecua a estos ejemplos hay que usar los códigos que da la Tabla 2.3 para que POSIX pueda interpretar la fecha y hora correctamente.

Dada una fecha tiempo en formato no convencional, se indica cómo está ordenada empleando los códigos pertinentes

> as.POSIXct('080406 10:11', format <- '%y%m%d%H:%M')

y la devuelve en formato internacional '2008-04-06 10:11:00 -03'

Otro ejemplo con separadores

> as.POSIXct('2008-04-0610:11:01PM',format<-'%Y-%m-%d%I:%M:%S:%p')

devuelve '2008-04-06 22:11:01 -03'
Los dígitos finales indican la diferencia horaria local.

La diferencia entrePOSIXlt y POSIXct se aprecia en los ejemplos siguientes.

> hoy <- Sys.Date()

> attributes(as.POSIXct(hoy)) devuelve

```
$class
[1] ' POSIXct'  ' POSIXt
```

mientras que

> attributes(as.POSIXlt(hoy)) devuelve

```
$names
[1] ' sec'  ' min'  ' hour' ' mday' ' mon'  ' year' ' wday' ' yday'
[9] ' isdst'

$class
```

```
[1] " POSIXlt"  " POSIXt"

$tzone
[1] " UTC"
```

POSIXlt incluye una lista con diversos componentes de fecha y tiempo. Tener en cuenta que POSIX cuenta los meses a partir de 0 y los años a partir de 1900. Por ejemplo, el año 1800 es -100.

Parte IV Gestión de datos

Capítulo 3 Importar/Exportar

Dos operaciones esenciales en todo proyecto son ingresar los datos en R y exportar los resultados. Este capítulo trata de las funciones que se ocupan de estas tareas.

La interfaz gráfica de RStudio ofece un modo práctico para ingresar y exportar datos, como se vio en la Introducción. Esta opción, sin embargo, tiene aplica- ción muy limitada. En la mayor parte de los casos es necesario cargar datos programáticamente, es decir, durante la ejecución de un programa, empleando comandos en línea. En lo que sigue de este curso no se usará la interfaz de RStudio para ingresar o exportar datos. Se ven primero herramientas de importación de datos y luego las de exportación.

Nivel de Complejidad

Las funciones read.table() y read.csv() para la importación de datos, y la función save() para guardar lo hecho, son indispensables en cualquier trabajo con R. Las demás son apropiadas para un nivel intermedio de conocimiento de R, por lo que su aprendizaje puede diferirse. Aun cuando la función scan() pertenece al segundo grupo se presenta en primer lugar para poner de manifiesto que está en la base de varias otras y porque habilita el ingreso de datos por teclado.

3.0.1. scan()

La función scan() permite ingresar datos por teclado o leer de archivo.

Escribir

```
> mi_txt <- scan()
```

al presionar Enter da lugar a ingresar un dato. Ingresar sucesivamente 1 2 3 4. Al cerrar estos números habrán quedado guardados en mi_txt.

Para leer desde almacenaje en la computadora la sintaxis es
scan(file = , what = 0, n = -1, sep = ' ', skip <- 0, y otros argumentos menos usados) donde file indica el origen de los datos; what indica el tipo de datos:

numérico, lógico, carácter, el default 0 es para numérico; n da el número de registros a leer, con -1 indicando llegar al final de la tabla; sep describe el separador, que por defecto es un espacio; y skip indica el número de líneas a saltear de encabezamiento. scan() devuelve un vector o una lista. En la carpeta **Datos**>**Tablas** está el archivo de texto delimitado por comas 'Empleados.txt'. Escribir

```
> mi_txt2 <- scan(file="Empleados.txt", what=list(ID=0, Nombre='',
Apellido='', Educacion='', IngresoAnual=''), sep=',', skip=1)
```

donde se da a what una lista con los tipos de campos que se han de leer; 0 para campos numéricos y '' para campos con cadenas de caracteres. sep = describe el separador y skip = indica el número de líneas a saltear al inicio.

3.0.1. read_csv()

La función read_csv, del paquete readr, es la preferida al momento de leer archivos de texto delimitados por comas.

```
> txt <- read_csv('Empleados.txt', header = T)
```

Notar que las variables lógicas TRUEy FALSE se pueden abreviar.
Hacer class(txt) y devuelve tbl_df, es decir la lectura convirtio el archivo de texto plano en un tibble.

Una sintaxis similar es read.csv(). Esta es anterior a los tibbles y la lectura devuelve un dataframe.

La ventajas de read_csv son las siguientes

- Lee más rápido
- No modifica los tipos de dato
- Permite que las columnas sean listas
- Permite encabezamientos prohibidos para read.csv, como variables que empiecen con un número
- No generan nombres de fila

Es posible que nos topemos con archivos de texto que no están delimitados por comas, sino por espacios u otro carácter. Veamos algunos recursos

read_csv2() usa punto y coma como separador, útil en países donde la coma es separador decimal

read_tsv() usa tabulación como separador

read_delim() usa un separador arbitrario.

Ver video Transferencia de Datos.mp4.

3.0.2. Ingresar datos por teclado

Cuando el número de datos a ingresar son pocos, puede ser conveniente ingresarlos directamente por el teclado, es decir, interactuando con el usuario. Por ejemplo al llenar un cuestionario, o ingresar un dato que desencadena un proceso más largo.

Para esta tarea emplear readline() junto con prompt:

Nombre <- readline(prompt='Ingrese sunombre')

Luego puede utilizar Nombre.

El ingreso de números requiere atención. readline() implícitamente coerciona los

números a cadenas. Por ejemplo, en:

Edad <- readline(prompt='Ingrese su edad')

si ingresan 20, devuelve '20'. Para obtener un número entero deben coercionar con as.integer(Edad).

3.0.3. download.file()

La función download.file() permite descargar archivos de sitios en internet y dirigirlos a un destino explícito. La sintaxis es

> download.file(url, archivo destino, método a emplear)

Se ejemplifica el uso descargando una tabla delimitada por comas de Figshare, un repositorio de trabajos no publicados en revistas y de datos que acompañen a publicaciones en revistas. Escribir

```
> download.file(url<-'¥url{https://ndownloader.figshare.com/files/
2292169}',destfile <-
'D:/Gustavo/Cursos/R/cursoR/MiFigshare.csv')
```

El archivo descargado recibirá el nombre MiFigshare.csv. La función opera con varios métodos de descarga.

3.0.4. file.choose()

Los modos de ingreso de datos presentados implican conocer la ubicación del archivo que se desea cargar. La función file.choose() ofrece una alternativa útil.

> read.csv(file.choose())

abre una pantalla de exploración que facilita la búsqueda de un archivo en un directorio. Si la tabla tiene una primer fila de encabezamientos agregar header = TRUE, si tiene una primer columna ocupada por nombres de filas, agregar rownames<-1.

Al trabajar con tablas notarán que R agrega nombres a la scolumnas y a las filas por defecto, y se verán modos de cambiarlos.

3.0.5. sink()

Para exportar datos R ofrece varias alternativas. sink() se usa para dirigir el resultado de una operación a un destino. La sintaxis es

> sink(nombre de archivo de salida, si append = TRUE anexa los datos si FALSE (por defecto) crea el archivo. Operar con sink() implica tres pasos: apertura, ejecución y cierre.

Ejemplo

Guardar en un archivo de texto los ingresos anuales de la tabla Empleados.txt. Asignar la tabla a var

> var <- read.table('Empleados.txt', header = T, sep = ','). Los ingresos están en var$IngresoAnual. Hacer

> sink('Ingresos.txt') para abrir la operación de exportación; las operaciones que se ejecuten con posterioridad serán derivadas al archivo nuevo Ingresos.txt, en el directorio de trabajo o dirección indicada. Ejecutar

> var$IngresoAnual

que recupera el dato y cerrar la operación con un llamado vacío

> sink()

El archivo Ingresos.txt se guardó en el directorio de trabajo. Si se desea eliminarlo hacer

> unlink('Ingresos.txt')

Con scan('Ingresos.txt') debiera recuperarse el contenido, pero da error. Abrir Ingresos.txt en un editor de texto. El problema reside en que la operación guardó la salida completa, incluyendo el indicador de posición [1]. Borrarlo y scan('Ingresos.txt') operará correctamente.

3.0.6. dump()

La función dump() guarda una representación de los datos y permite exportar varios objetos al mismo tiempo. La sintaxis por defecto es

>dump(lista, file <- 'nombre salida', append <-FALSE, y otros argumentos menos usados) donde lista es un vector de caracteres. Los objetos son exportados en formato ASCII. Se recuperan con source(). Un ejemplo de sesión

> a <- 1:5

> b <- c(2,4,8) los objetos a y b se agregan en Global Environment. Seguidamente se guardan

> dump(c('a','b'), 'MiDump.R') los objetos a y b se guardan en Mi-Dump.R.

> rm(a, b) los elimino del Global Environment.

> source('MiDump.R') se recuperan y se puede operar con ellos. Con doble click en Global Environment se muestra el contenido en eleditor.

3.0.7. save(), save.image()

Varias funciones permiten guardar lo hecho en R y difieren en sus prestaciones.

La función saveRDS() permite guardar un único objeto; no inserta la ex- tensión *.Rds, aunque tampoco es absolutamente necesaria. La sintaxis es: saveRDS(objeto a guardar,"nombre.rsd") El formato RDS se carga con readRDS(). readRDS("nombre.Rds") Algo a tener en cuenta es la manera como readRDS() recupera los datos. Se tiene el vector a<-c(1,2,3,4) y se lo guarda con

> saveRDS(a,"mi_archivo.rsd")

Seguidamente recuperarlo de dos maneras, con
> readRDS("mi_archivo.Rds") y con
val <- readRDS("mi_archivo.Rds") Inspeccionar los resultados. En el primer caso se recupera el contenido del vector, [1] 1 2 3 4, pero no el vector, mientras que en el segundo se recupera el vector completo y val aparece en el Global Environment. Si intentan guardar dos o más objetos con saveRDS() dará error.

La función save() permite guardar varios objetos a la vez; emplea la extensión *.RData, pero hay que incluirla manualmente. Por ejemplo

> save(a,b,file<-"mi_archivo.Rdata")

Los archivos guardados como RData se recuperan con load(). Tener en cuenta que

con load() hay que incluir la extensión pero que una vez cargado el archivo la extensión no se usa más. Los formatos RDS y RData son de tipo binario. Por cierto el resultado de una operación puede guardarse en otros formatos, pero se corre el riesgo de perder determinados formatos, como por ejemplo, las fechas.

Al cerrar RStudio se ofrece la opción de guardar todo el entorno de trabajo, incluso lo que aparece en Global Environment. La función

> save.image(file = "mi_archivo.RData") permite hacer esto en cualquier momento, asignando automáticamente la extensión *.RData y añadiendo el ícono de R. Notar que save.image() es una variante compacta de save().

3.0.8. write.table()

La función write.table() exporta una matriz o marco de datos a la consola o a un directorio. La sintaxis por defecto es write.table(archivo a exportar, nombre archivo nuevo, append = FALSE, sep = , dec = '.', row.names = TRUE, col.names = TRUE)

> write.table(tabla.externa, 'MiTablaExterna') crea MiTablaExterna en el directorio de trabajo.

> write.csv() es similar presuponiendo sep = ','.

3.0.9. Exportar gráficos

Opera de modo similar a sink(), hay que iniciar la operación con

> pdf('MiGrafico.pdf')

Ejecutar un código gráfico cuyo resultado se desea guardar

> x <- 1:50

> plot(s, log(x))

Cerrar la operación con

> graphics.off()

3.1. El formato netCDF

Nivel de Complejidad

El material en esta sección es apropiado para un nivel intermedio de conocimiento de R.

Los archivos en formato NetCDF (network Common Data Form) se emplean para guardar datos multidimensionales, tales como temperatura, humedad, precipitación, en función de latitud y longitud, y eventualmente cota topográ- fica y tiempo. Este formato es de uso común en oceanografía, meteorología y climatología. Cada archivo incluye un encabezamiento que describe la estructura. NetCDF está mantenido por el programa Unidata en University Corporation for Atmospheric Research (UCAR).

Los archivos netCDF tienen extensión *.nc, e incluyen un encabezamiento que

describe el contenido. El encabezamiento de un archivo netCDF típico es: netCDF example
dimensions:
EW = 87 ;
SN = 61 ;
variables:
double EW(EW) ;
EW:units = "meters";
double SN(SN) ;
SN:units = "meters";
float Elevation(SN, EW) ; Elevation:units =
"meters"; Elevation:missing_value = -1.f;

El formato puede operar en distintos sistemas operativos. Los datos se almacenan en arrays cuyos componentes básicos son la dimensión, la variable y los atributos. La dimensión normalmente es latitud, longitud y tiempo, cada uno asociado con el número que indica cuantas veces se repite. Las dimensiones por lo general son acotadas, excepto el tiempo que puede ser ünlimited". Las variables representan conjuntos de datos de un mismo tipo. Cada variable representa un fenómeno (temperatura, lluvia, NDVI). Un caso especial son las variables Coordenadas almacenan lon, lat y tiempo. Los valores de una variable en un sitio determinado a lo largo del tiempo componen un array unidimensional, y si esa variable se mide en varias localidades en un tiempo fijo, da lugar a un array bidimensional. Por fin, si esa variable se mide en varias localidades en tiempos diferentes, resulta un array tridimensional. Los atributos están asociados a variables y agregan información. Por ejemplo, lluvia:units=mm/yr. Los atributos globales dan información sobre el archivo netCDF.

Los archivos netCDF son binarios. Es posible extraer partes de los datos sin descargarlos todos, lo cual ahorra espacion y tiempo de computación.

El siguiente ejemplo muestra cómo abrir y explorar un archivo NetCDF. Tomado parcialmente de: https://rpubs.com/jonesey441/NetCDF-data.

La carpeta **Datos**>**netCDF** contiene el archivo 'cru10min30_tmp.nc'. con datos de temperatura. También se incluye el archivo RMarkdown netcdf.Rmd, con el código que se describe a continuación.

Apuntar el directorio de trabajo a esa carpeta y hacer

```
> ncname <- 'cru10min30_tmp'
```

Agregar la extensión *.nc al nombre primario sin dejar espacio, sep<-".

```
> ncfname <- paste(ncname, '.nc', sep<-")
```

La función paste() concatena los elementos ncname y .nc insertando un separador, en este caso vacío.

Crear el objeto dname.

```
> dname <- 'tmp'
```

Abrir el archivo en R. La lectura de estos archivos requiere una función especial, nc_open(), que se encuentra en el paquete ncdf4. Instalar y activar el paquete.

```
> ncin <- nc_open(ncfname)
```

La función nc_open abre una conexión con el archivo pero no descarga todos los

datos, sólo los metadatos, que se pueden ver con print().

```
> print(ncin)
```

Imprimir la estructura con print(ncin). Vista parcial abajo.

File cru10min30\tmp.nc (NC\FORMAT\CLASSIC):

2 variables (excluding dimensionvariables): float
time\bounds[nv,time]
float tmp[lon,lat,time] long\name:
air\temperature units: degC
\FillValue: -99
source:E:\Projects\cru\data\cru\cl\2.0\nc\files\cru10min\tmp.nc Vista parcial de la salida
4 dimensions: lon
Size:720
lat Size:360
time Size:12
nv Size:2

El informe arriba está organizado por dimensión y luego por variable.

Un listado de los atributos

```
> attributes(ncin)$names
 [1] ' filename'    ' writable'    ' id'          ' safemode'
 [5] ' format'      ' is¥_GMT'      ' groups'      ' fqgn2Rindex
                                                  '
 [9] ' ndims'       ' natts'       ' dim'         ' unlimdimid'
[13] ' nvars'       ' var'
```

Una vez conocidos los nombres de las variables se pueden extraer los valores con ncvar_get() y asignarlas a un objeto. Extraer las coordenadas e imprimir las dimensiones

```
> lon <- ncvar_get(ncin,'lon')
> lat <- ncvar_get(ncin,'lat') El número de registros con longitud y latitud lo dan
> nlon <- dim(lon)
> nlat <- dim(lat) La dimensión del archivo
> print(c(nlon,nlat)) [1] 720 360
```

Extraer la columna con tiempos

```
> time <- ncvar_get(ncin,'time')
> time
```

Imprimir los atributos de la variable time y su dimensión

```
> tunits <- ncatt_get(ncin,'time','units')
> nt <- dim(time)
> nt
```

tunits tiene dos componentes, $hasatt, una variable lógica, y $value, que cuenta los días desde 1 enero 1900.

```
> tunits $hasatt $value [1] 'days since 1900-01-01 00:00:00.0 -0:00'
```

Extraer la variable tmp y guardarla en un array; extraer los atributos de los argumentos long_name, units y _Fillvalue en tmp (ver estructura parcial arriba), y mostrar la dimensión.

```
> tmp_array <- ncvar_get(ncin,dname)
> dlname <- ncatt_get(ncin,dname,'long_name')
> dunits <- ncatt_get(ncin,dname,'units')
> fillvalue <- ncatt_get(ncin,dname,'_FillValue')
> dim(tmp_array)
```

Extraer parámetros globales de los metadatos del archivo.

```
> title <- ncatt_get(ncin,0,'title')
> institution <- ncatt_get(ncin,0,'institution')
> datasource <- ncatt_get(ncin,0,'source')
> references <- ncatt_get(ncin,0,'references')
> history <- ncatt_get(ncin,0,'history')
> Conventions <- ncatt_get(ncin,0,'Conventions')
```

Cerrar el archivo y verificar el contenido en el directorio de trabajo.

```
> nc_close(ncin)
> ls()
```

```
 [1] ' Conventions'  ' datasource'     ' dlname'
 [4] ' dname'        ' dunits'       ' fillvalue'
 [7] ' history'      ' institution'  ' lat'
[10] ' lon'          ' ncfname'      ' ncin'
[13] ' ncname'       ' nlat'         ' nlon'
[16] ' nt'           ' references'   ' time'
[19] ' title'        ' tmp¥_array'    ' tunits'
```

Los archivos netCDF típicamente traen latitud, longitud y tiempo, más una cuarta variable, como temperatura. El tiempo suele estar registrado en formato CF (Climate Forecast), que cuenta los días desde un inicio arbitrario. Para reconvertir el tiempo CF a un formato más fácilmente legible hay que recuperar separadamente año, mes y día. Para ello se requiere instalar y activar el paquete chron.

Luego hacer

```
> tustr <- strsplit(tunits$value, ' ')
> tdstr <- strsplit(unlist(tustr)[3], '-')
> tmonth <- as.integer(unlist(tdstr)[2])
> tday <- as.integer(unlist(tdstr)[3])
> tustr <- strsplit(tunits$value, ' ')
> tdstr <- strsplit(unlist(tustr)[3], '-')
```

```
> tmonth <- as.integer(unlist(tdstr)[2])
> tday <- as.integer(unlist(tdstr)[3])
> tyear <- as.integer(unlist(tdstr)[1])
> chron(time,origin<-c(tmonth, tday, tyear))
```

El primer comando extrae el valor de la variable unidad de tiempo: 'days' 'since' '00:00:00.0'. El segundo aisla el tercer elemento: '1900-01-01', El tercero separa los subelementos usando unlist, para dar: '1900' '01' '01'. Los tres siguientes toman, respectivamente, mes, día y año. Hacer

```
> chron(time,origin<-c(tmonth, tday, tyear))
```

Por convención, cuando se trabaja con una serie temporal, en este caso 1961-1990, se emplea como tiempo el año central en la serie, 1976, y el día central de cada mes

Capítulo 4

Procesar tablas de datos

Un conjunto de datos es una colección de valores numéricos (cuantitativos) o cadenas de texto (cualitativos) que definen un atributo (e.g., temperatura, ancho, peso, calidad). Cada valor corresponde, a la vez, a una variable y a una observación, o medición. Una variable comprende todos los valores medidos en un mismo atributo, y cada observación comprende todos los valores relativos a un mismo objeto de estudio, una persona, un automóvil, etc. Por lo general, tales valores, variables y observaciones se almacenan en tablas, y estas tablas son las que el analista utiliza para sus tareas. La más breve experiencia en compilación de datos pone de manifiesto que no existe una única manera de acomodar los datos en una tabla y que, por añadidura, una tabla puede incluir celdas vacías o con errores. Estas falencias pueden impedir el correcto funcionamiento de rutinas en R. La tarea de adecuar los datos tabulados para el análisis se divide en dos fases: emprolijar y dar consistencia. En este capítulo se muestra cómo adecuar los datos para el análisis.

4.1. Filtrar datos

En ocasiones se desea operar con solamente una fracción de los datos en una tabla, o revisar solamente un subconjunto. Al presentar las diversas estructuras de datos se vio cómo esto se puede lograr trabajando con los índices de ubicación de los elementos. Aquí se presentan otros recursos con las funciones filter(), grep(), grepl() y gsub().

4.1.1. filter()

La función filter() inspecciona una tabla identificando filas o columnas para las cuales se cumple cierta condición. filter() se encuentra en el paquete dplyr, incluido en tidyverse. Aunque diseñada para trabajar con tibbles, acepta dataframes.

Ejemplo

Se desea extraer registros de mtcars_tb en los que la variable mpg supera 23 millas por galón (ver %> % en Operadores, Módulo 1)

```
> mtcars_tb%>% filter(mpg > 23)
```

```
mpg cyl disp hp drat wt qsec vs am gear carb 1
24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
```

opera correctamente pero devuelve todas las columnas. Si se desea limitar la salida se puede combinar filter() con la función select()

```
> mtcars_tb%>% filter(mpg> 23)%>% select(mpg,hp)
```

mpg hp
1 24.4 62
devuelve solamente las columnas especificadas, mpg y hp.

4.1.2. grep(), grepl()

Las funciones grep() y grepl() se encuentran en el paquete base de R. grep(value = TRUE) devuelve un vector con los índices de los elementos de x que dieron positivo en la búsqueda.
grepl(pattern<- expresión regular o cadena de caracteres) devuelve un vector lógico, con TRUE donde se detectó coincidencia.

Ejemplo grep()

Dado
> x <- 'El COVID-19 es una pandemia'
Buscar el guión
> grep('-', x)
devuelve 1 como índice porque el vector x sólo tiene un elemento, la frase completa. Si, en cambio, se separan las palabras
> x <- c('El', 'COVID-19', 'es', 'una', 'pandemia')
> grep('e', x) devuelve
los índices 3 y 5.

Ejemplo grepl()

La función grepl() devuelve un valor lógico del índice en vez de uno numérico
> grepl('e', x) [1] FALSE FALSE TRUE FALSE TRUE

4.1.3. sub(), gsub()

En tanto que filter(), grep() y grepl() identifican elementos, sub() y gsub() remplazan elementos
sub() remplaza el primero que encuentra y gsub() remplaza todos los que coinciden con el criterio. Dado el vector
x <- 'aaabbb'
> sub('a', 'c', x)
devuelve 'caabbb'
y
> gsub('a', 'c', x)
devuelve 'cccbbb'

4.2. Depuración de tablas

Es común encontrar tablas que contienen celdas vacías por falta de un dato, o en las cuales se insertan 0s o diferentes códigos para representar ausencia de

medición/observación, como por ejemplo -9999, y también que incluyen duplicación errónea de registros.

Ciertas rutinas en R ignoran valores faltantes, otras avisan de la presencia, otras interrumpen la ejecución, y aun otras se detienen a la espera de instrucciones sobre qué hacer con ellos.

Identificar valores faltantes

En R se estila llenar las celdas que carecen de un valor por ausencia de medición con NA (Not Available). En la carpeta **Datos/Tablas** se encuentra el archivo ValoresFaltantes.txt. Examinarlo en Notepad. Notar que el separador es espacio. Hay varias celdas sin datos; Cargarlo con

```
> ValFalt <- read_delim('ValoresFaltantes.txt', delim=' ')
```

Veamos qué ocurre al calcular el promedio de alumnos

```
> mean(ValFal$NoAlumnos) devuelve el mensaje de error:
argument is not numeric or logical: returning NA
```

La función mean() no acepta NAs. Queda en el analista decidir qué hacer con los valores faltantes.

Analicemos un caso típico. La función is.na() detecta la presencia de NA y NaN. Hacer

```
> is.na(ValFal)
```

Devuelve una tabla donde hay cinco TRUE. Comprobamos que corresponden a los cuatro NAs y a un espacio vacío en la columna Escuela. Conclusión, is.na() no reconoce N/A ni NaN como datos faltantes, simplemente son cadenas.

4.2.1. na.omit()

La función na.omit(ValFalt) corta por lo sano y elimina las filas que incluyen NA, reduciendo así los registros de la tabla. Pero notar que ignora otros rellenos.

4.2.2. Remplazo de NAs

Otra opción es remplazar los valores que faltan por otros. La función replace_na() viene en el paquete tidyr y permite remplazar NA por 0. Recordar que opera sobre vectores numéricos.

```
> ValFal$Categoria_num <- as.numeric(as.character(ValFal$Categoria))
Ejecuta con advertencia NAs introduced by coercion.
Ahora remplazamos NA con 0 en la columna Categoria_num
> ValFal$Categoria_num[is.na(ValFal$Categoria_num)] <- 0
```

La función replace_na() viene en el paquete tidyr, instalado con tidyverse, y permite remplazar los NAs con un valor arbitrario.

```
>  > vf3 %>% replace_na(ValFal,replace = list(Categoria = 'falta'))
```

4.2.3. Eliminación de duplicados

Crear el vector

> prov<-c('San Juan','Mendoza','Formosa','Salta','Mendoza','Mendoza')

Las siguientes dos herramientas se incluyen en el paquete base
duplicated() - identifica los elementos duplicados con TRUE
y
unique() - extrae los elementos únicos, no repetidos, de una tabla o vector.

Si deseo extraer las filas que no están duplicadas uso negación

> prov[!duplicated(prov)]

El paquete dplyr, en tidyverse, ofrece la función distinct()
distinct() elimina las filas con elementos duplicados, pero opera sobre dataframes o tibbles. Por lo tanto hay que convertir el vector prov en un marco de datos

> distinct(tibble(prov))

4.3. Alterar el orden de filas y columas

4.3.1. Trasponer una tabla

Se tiene la Tabla 4.1 en el archivo Datos/Tablas/TablaTemp.txt.

Se desea crear un gráfico de barras con estos datos. Para tal fin es conveniente que cada mes ocupe una columna en vez de una fila. La operación requerida es trasposición.

Cargar la tabla

> temp <- read_csv('TablaTemp.txt, header = TRUE)

y luego trasponerla con

> temp_trasp <- t(temp)

Mes	Temperatura
1	30
2	27
3	25
4	20
5	20
6	16
7	18
8	19
9	20
10	23
11	25
12	26

Tabla 4.1: Temperatura anual promedio

4.3.2. Agregar y quitar columnas

En ocasiones, operaciones en un marco de datos resultan en nuevas variables que deben anexarse al mismo marco de datos como una columna nueva. La función mutate(), en el paquete dplyr, tiene esa prestación.

La tabla PBI.txt, en la carpeta **Tablas**, contiene datos socio-económicos para tres países, en particular sobre empleo precario y PBI, para tres años. Se desea agregar dos columnas, una con el promedio de los porcentajes de empleo precario y otra con el promedio del PBI en los años dados.

Importar PBI.txt en R con read_csv() como pbi_tb. Hecho esto escribir

```
pbi_ampliada <- pbi_tb %>%
select(EmpPrec1990,EmpPrec2000,EmpPrec2010,PBI1990,PBI2000,PBI2010)
%>% mutate(promEP=(EmpPrec1990+EmpPrec2000+EmpPrec2010)/3,
promPBI=(PBI1990+PBI2000+PBI2010)/3)
```

El comando calcula los promedios y los almacena en las nuevas columnas con nombres promEP y promPBI.

Si se desea eliminar una columna asignarle NULL. En este ejemplo se agregan al final pues si se eliminan antes no intervienen en el cálculo.

```
pbi_reducida <- pbi_tb %>%
        select(EmpPrec1990,EmpPrec2000,EmpPrec2010,PBI1990,PBI2000,PBI2010) %>%
        mutate(promEP=(EmpPrec1990+EmpPrec2000+EmpPrec2010)/3,
        promPBI=(PBI1990+PBI2000+PBI2010)/3,EmpPrec2000=NULL,
```

EmpPrec2000=NULL,EmpPrec2010=NULL)

Al cargar marcos de datos R asigna nombres de filas y los coloca en una columna especial excluida del resto. Es posible, sin embargo, incluirla con la función rownames_to_column(). Esta función se encuentra en el paquete tibble.

Ejemplo

El marco de datos mtcars, incluido en R, tiene como nombres de filas los modelos de automóvil. Revisarlo con head(mtcars). La tarea es incorporar esos nombres al marco de datos como una columna nueva. Escribir

> mtcars_df <- mtcars
asignar los datos a df para no alterar el original y seguidamente
> rownames_to_column(mtcars_df, 'Modelo')
> mtcars_df mostrará los nombres de filas bajo la columna Modelo.

Al convertir un dataframe a tibble se pierden las etiquetas de filas. Para preservarlas hay que pasarlas a una columna del mismo dataframe con
row_names_to_column(mtcars_df, var = 'Modelo')

> mtcars_tb <- mtcars_df %>% rownames_to_column(var = 'Modelo')

En este acápite corresponde introducir dos funciones que permiten modificar columnas; se trata de unite() y de separate(), del paquete tidyr. Supongamos que una tabla de datos muestra las fechas como día, mes y año en columnas separadas y deseamos unirlas para dar una fecha. Creamos la tabla con

tabla <- tibble(dia=c(11,22,30), mes=c(2,3,5),anio=c(1990,2000,2022))

Aplico unite
tab <- tabla %>% unite(col='fecha', c('dia','mes','anio'),sep='-')
col es el nombre de la columna a crear, el vector agrupa las columnas actuales, y sep indica cual carácter usar como separador.
Tomemos el caso inverso. Deseamos separar una fecha en sus componentes
tab %>% separate(col=fecha, into=c('dia','mes','anio'),sep='-')
col indica la columna a procesar e into= las columnas destino.

4.4. Unión de tablas

Con frecuencia la información está distribuida en dos o más tablas que deben combinarse. Desarrollamos un ejemplo empleando datos del repositorio NHANES (National Health and Nutrition Examination Survey) de los EEUU.

Para acceder a los datos se requiere previamente instalar el paquete RNHANES y activarlo. Descargamos dos tablas, una con presión sanguínea diastólica y

sistólica, y otra con niveles de LDL y de triglicéridos para dos años.

> Psang <- nhanes_load_data("BPX_G", "2011-2012") %>% select(SEQN, BPXSY1, BPXDI1)

con más de 9300 observaciones, y

> colest <- nhanes_load_data("TRIGLY_G", "2011-2012") %>% select(SEQN, LBXTR, LBDLDL)

con más de 3200 observaciones. La columna SEQN sirve de enlace entre ambas tablas.

La función merge(), incluida en el paquete base, sirve para unir dos tablas. La sintaxis es
> merge(Psang, colest, by = 'SEQN')

La unión se establece comparando el contenido en sendas columnas en los dos marcos de datos. Se extraen las filas donde hay coincidencia entre ambas columnas. Las columnas pueden ser identificadas por sus nombres o índices. Hay cuatro opciones de unión: inner join (opción por defecto de merge()), full outer join, left outer join y right outer join. Inner join preserva las filas que coinciden en ambos marcos de datos; all = FALSE. Full outer join preserva todas las filas de ambos marcos de datos; all = TRUE. Left outer join incluye todas las filas del marco x y sólo las que coinciden del marco y. x = TRUE. Right outer join incluye todas las filas del marco y, y sólo las que coinciden del x (Figura 4.1).

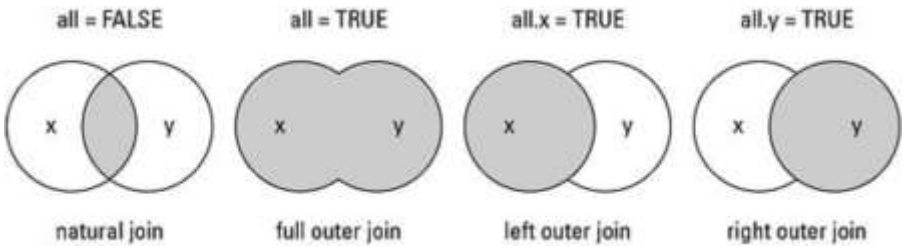

Figura 4.1: *Unión de tablas.*

Para ser fieles a tidyverse usamos funciones en el paquete dplyr.
> salida <- inner_join(data1,data2,by='SEQN')
> salida <-left_join(data1,data2,by='SEQN')
> salida <-right_join(data1,data2,by='SEQN')
> salida <-full_join(data1,data2,by='SEQN')
En cada caso comprobar las dimensiones con
dim(salida)
e imprimir sólo las primeras 10 líneas con

```
salida %>% slice(1:10)
```

4.5. Formatos de tablas

Nivel de Complejidad

Las operaciones que se ven en esta sección son apropiadas para un nivel intermedio de conocimiento de R.

Comúnmente los datos se presentan al analista en algún formato tabular y el contenido de filas y columnas puede variar considerablemente. En la Tabla 4.2 cada fila corresponde a un alumno, un sujeto, y cada columna a una prueba, que constituye la variable. Las notas, o valores, se distribuyen en varias filas y columnas. Cada fila contiene múltiples observaciones para cada sujeto. Este formato se suele denominar ancho. El programa SPSS utiliza formato ancho (Tabla 4.2).

Alumno	Prueba 1	Prueba 2	Prueba 3
Jose	6	8	7
Ana	9	6	9
Hilda	5	7	8

Tabla 4.2

Alumno	Pruebas	Notas
Jose	1	6
Jose	2	8
Jose	3	7
Ana	1	9
Ana	2	6
Ana	3	9
Hilda	1	5
Hilda	2	7
Hilda	3	8

Tabla 4.3

Otra manera de acomodar estos mismos datos es agrupar sujetos, variables y valores en sendas columnas, como en la Tabla 4.3. Este formato suele denominarse alto, o largo, debido a que ocupa menos columnas. En este formato, cada fila contiene una única observación por sujeto y las observaciones se encolumnan bajo nombres de variables. Muchas rutinas en R prefieren este arreglo. Por ejemplo, mutate, summarise y ggplot, simplemente porque R opera con vectores y en este formato cada columna es un vector. Más aun, es un vector atómico, pues las columnas contienen elementos de igual tipo. El ecosistema tidyverse prefiere el formato largo, de hecho las denomina 'prolijas' ('tidy').

Dado que ambos formatos son útiles R provee funciones para pasar de uno a otro. Para esta tarea el paquete tidyr ofrece las funciones gather() y spread(), y Pivot.longer() y pivot.wider(). gather() convierte de formato ancho a largo, y spread() revierte la acción, mientras que pivot.longer() equivale a gather() y pivot.wider() a spread(). Las cuatro funciones están incluidas en el paquete tidyr. spread() y gather() son más antiguas y están siendo remplazadas funcionalmente por las segundas.

4.5.1. gather() y spread()

Ver video Formatos.mp4.

Se crea un marco de datos con nombres, género y pesos de bebés en varios años

```
> bebes <- data.frame(nombre = c("Jose", "Ana", "Hilda", "Clota"), genero = c("M", "F", "F", "F"), a2011 = c(74.69, 74, 88.24, 88.77), a2012 = c(84.99, 87, 90, 96.45), a2013 = c(91.73, 75.74, 101.83, 105))
```

El formato ancho se reconoce en que las observaciones están distribuidas en varias columnas.

La función gather() tiene la sintaxis por defecto:

gather(data, key = "nombre", value = "valores", vector de columnas, na.rm = FALSE)

donde data es la tabla en formato ancho, key es una columna a crear nueva que incorpora los datos en las columnas indicadas en el vector de columnas, 3:5; value es otra columna nueva con los valores asignados a esas mismas variables, y na.rm elimina las celdas vacías.

Aplicar gather() al marco de datos bebes en formato ancho

```
> ancho_a_largo <- gather(bebes,key = 'Fecha',value = 'Nota', 3:5, na.rm=TRUE)
```

Devuelve

	nombre	genero	Fecha	Nota
1	Jose	M	a2011	74.69
2	Ana	F	a2011	74.00
3	Hilda	F	a2011	88.24
4	Clota	F	a2011	88.77
5	Jose	M	a2012	84.99
6	Ana	F	a2012	87.00
7	Hilda	F	a2012	90.00
8	Clota	F	a2012	96.45
9	Jose	M	a2013	91.73
10	Ana	F	a2013	75.74
11	Hilda	F	a2013	101.83
12	Clota	F	a2013	105.00

Y si se desea convertir de formato largo a formato ancho se aplica spread(), que convierte las columnas encabezadas con key y value de formato largo a columnas de formato ancho, como se ve al ejecutar el siguiente comando

```
> largo_a_ancho <- spread(ancho_a_largo,key = 'Fecha', value = 'Nota')
```

Ejemplo

En la carpeta **Datos/Tablas** está el archivo pew.csv, descargado del centro de investigación Pew. El archivo contiene la proporción de fieles a distintas religiones clasificados por el ingreso monetario.

Apuntar a la carpeta y cargarlo en R con

```
> pew <- read_csv("pew.csv")
```

Examinarlo con View(pew). Los nombres de las columnas son valores, y las observaciones están distribuídas en la tabla. Conviene reordenar con

```
> pew.largo <- gather(pew, income, freq, -religion)
```

Se crean dos columnas nuevas, income agrupa los ingresos y freq los porcentajes de afiliación; el signo - mantiene la columna religion.

4.5.2. pivot.longer() y pivot.wider()

La función pivot.longer() incrementa el número de filas de una tabla y decrece el número de columnas, haciéndola más larga, o alta. pivot.wider() opera a la inversa.

Usaremos otra tabla de modo de ampliar la experiencia con formatos. Descargar la tabla de

```
> datos_url <- "https://goo.gl/ioc2Td"
> gapminder <-read_csv(datos_url)
> head(gapminder)
```

La tabla contiene datos sobre expectativa de vida, PBI y población para varios países. Las columnas de variables continent y country albergan valores apropiados. Pero las de PBI, población y expectativa de vida tienen por encabezamiento los años. Reducir la tabla eliminando los datos de población y PBI

```
> gapminder_life <- as.data.frame(gapminder) %>% select(continent,country,starts_with("life"))
> head(gapminder_life)
```

Aplicar pivot.longer()

```
> gapminder_life %>% pivot_longer(-c(continent,country), names_to = 'year',
    values_to = "lifeExp")
```

Se obtiene una tabla en formato largo, con más de 1700 filas y cuatro columnas

```
continent country year          lifeExp
  <chr>     <chr>   <chr>           <dbl>
1 Africa    Algeria lifeExp_1952     43.1
2 Africa    Algeria lifeExp_1957     45.7
3 Africa    Algeria lifeExp_1962     48.3
4 Africa    Algeria lifeExp_1967     51.4
5 Africa    Algeria lifeExp_1972     54.5
6 Africa    Algeria lifeExp_1977     58.0
```

Aplicando pivot.wider() se revierte la conversión. Primero guardar la tabla en formato largo en un objeto

```
> gapminder_tidy<-gapminder_life%>%pivot_longer(-c(continent,country),
names_to = “year", values_to = "lifeExp")
```

Ejecutar

```
> gapminder_tidy%>% pivot_wider(names_from = year, values_from
= lifeExp)
```

Así concluye la revisión de formatos prolijos de tablas y cómo convertir entre formatos ancho y largo.

Fin del módulo 2