Escuela de métodos para
la actualización docente

MANUAL DE MÉTODOS Y HERRAMIENTAS PARA EL ANÁLISIS DE INFORMACIÓN USANDO EL LENGUAJE R

Volumen 1

Coordinadores:
Francisco Mora Ardila
Mario Martínez Salgado
Ana Yesica Martínez Villalba

ESCUELA
NACIONAL
DE ESTUDIOS
SUPERIORES
UNIDAD MORELIA

MANUAL DE MÉTODOS Y HERRAMIENTAS PARA EL ANÁLISIS DE INFORMACIÓN USANDO EL LENGUAJE "R"

Coordinadores
Francisco Mora Ardila
Mario Martínez Salgado
Ana Yesica Martínez Villalba

Autores
Francisco Mora Ardila, Mario Martínez Salgado,
Ana Yesica Martínez Villalba, Wesley Dáttilo,
Rodrigo Tapia McClung y Ayari Pasquier Merino

Escuela de Métodos
para la actualización docente

Proyecto PAPIME 207917 "Escuela de Métodos: una estrategia para la actualización docente en torno a métodos y herramientas de análisis de información"

ESCUELA NACIONAL DE ESTUDIOS SUPERIORES
UNIDAD MORELIA

iies
INSTITUTO DE INVESTIGACIONES EN ECOSISTEMAS Y SUSTENTABILIDAD
UNAM

UDIR

Morelia, Michoacán

MANUAL DE MÉTODOS Y HERRAMIENTAS PARA EL ANÁLISIS DE INFORMACIÓN USANDO EL LENGUAJE “R”

Índice

Sobre la Escuela de Métodos 6
Sobre este Manual 8
Antes de empezar 11
PARTE I. INTRODUCCIÓN AL MANEJO DE INFORMACIÓN USANDO R 13
Capítulo 1. Introducción al lenguaje R 14
1.1. Preliminares 14
1.1.1. R y RStudio 14
1.1.2. Comentarios en el código 17
1.1.3. Ejecutar una instrucción 17
1.1.4. La Consola 17
1.1.5. Directorio de trabajo 17
1.1.6. Ayuda 18
1.2. Operadores 19
1.3. Objetos 19
1.3.1. Numéricos 19
1.3.2. Caracteres 20
1.3.3. Vector numérico 20
1.3.4. Vector de caracteres 21
1.3.5. Matrices 21
1.3.6. Factor 22
1.3.7 Data frame 23
1.4. Funciones 25
1.5. Paquetes y librerías 26
1.6. Importar datos 30
1.7. Explorar una base de datos 31
1.7.1. Resumen de datos 33
1.8. Selección de información 35
1.8.1. Ejercicio 37
1.8.2. Selección de variables y/o casos 38
1.8.3. Ejercicio 40
1.9. Pegado de bases de datos 40
1.9.1. Ejercicio 42
1.10. Gráficos 43
1.10.1. Edición de gráficos 47
1.10.2. Varias gráficas en una imagen 55
1.10.3. Guardar imágenes 56
Capítulo 2. Gestión de datos con R 57
Introducción 57
2.1. ¿Cómo vaciar mis datos en una tabla? 58
2.2. Criterios básicos para crear una tabla ordenada 59
2.3. Manejo de tablas: modificación del contenido y resúmenes 61
2.3.1. Instalación de paquetes a usar 62

2.3.2. Lectura de datos 62
2.3.3. Selección de registros empleando *filter* 64
2.3.4. Ordenación de registros con `arrange` 70
2.3.5. Selección de atributos (columnas) empleando select 71
2.3.6. Generación de tablas agrupadas usando *group_by* 74
2.3.7. Adición de nuevas columnas con mutate 76
2.3.8. Resúmenes de tablas con summarise 79
2.4. Manejo de tablas: depurado y ajuste de tablas 83
2.4.1. Apilado de datos empleando gather 85
2.4.2. Extensión de tablas empleando spread 87
2.5. Pegado de tablas con _join 89
2.6. Ejercicio final 94
2.7. Consideraciones finales 95
2.8. Respuestas a los ejercicios 95
PARTE II. MÉTODOS DE ANÁLISIS 99
Capítulo 3. Introducción al Análisis de Redes de Interacciones Ecológicas entre Especies 100
Introducción 100
3.1. Análisis de redes 103
3.1.1. Análisis de redes bipartitas 104
3.2. La red de polinización *Safariland* 105
3.4. Graficando redes bipartitas 106
3.5. Índices para el análisis de redes bipartitas 114
3.5.1. Índices de la red 115
3.5.3. Anidamiento y modularidad 119
3.6. Modelos nulos en el análisis de redes 122
3.7. Más sobre el análisis de redes ecológicas en R 126
Capítulo 4. Análisis de dependencia y autocorrelación espacial usando R . 128
Introducción 128
4.1. Análisis de Patrones de Puntos 130
4.1.1. Umbral de distancia y matriz de adyacencia 132
4.1.2. Hacer una *gráfica de Moran* 134
4.1.3. Prueba estadística 138
4.1.4. Estadística local 142
4.1.5. Mapa de cúmulos 142
4.1.6. Mapa de significancia 146
4.2. Vecindad y dependencia espacial para polígonos 151
4.2.1. Exploración visual de los datos 153
4.2.2. Un mapa de desviaciones estándar - GISTools 154
4.2.3. Varios mapas de desviaciones estándar 155
4.2.4. Mapas de desviaciones estándar - ggplot2 158
4.2.5. Adyacencia y estadística espacial 165
4.2.6. Matriz de adyacencia 166

4.2.7. Autocorrelación espacial global y local 168
4.2.8. Diagrama de dispersión de Moran 170
4.2.9. Mapa de cúmulos 171
4.2.10. Pseudo-significancia y permutaciones 175
Capítulo 5. Ciencias Sociales: Análisis de contenido y uso de RQDA 181
Introducción 181
5.1. El análisis de contenido en las ciencias sociales 185
5.1.1. Análisis de contenido convencional 187
5.1.2. Análisis de contenido dirigido 189
5.1.3. Análisis de contenido sumatorio 190
5.1.4. El papel de la teoría en el análisis de contenido 191
5.1.5. Fuentes y organización de material 192
5.2. Análisis de contenido con RQDA 195
5.2.1. Estructura y herramientas generales 197
5.3. Codificación y categorías 208
5.3.1. Estrategias de codificación 208
5.3.2. Construcción de categorías y tipos de relaciones 212
5.4. RQDA: categorías, casos y atributos 213
5.4.1. Categorías de archivos 213
5.4.2. Categorías de códigos 215
5.4.3. Uso de “casos” y “atributos” 217
5.4.4. Esquemas de recuperación de información 221
Sobre los autores 225

Sobre la Escuela de Métodos

El avance de las ciencias ambientales y de las ciencias sociales durante las últimas dos décadas está claramente ligado a un cambio en la concepción de sus objetos o sistemas de estudio. Hemos pasado de visiones simplificadas y lineales de los sistemas ecológicos, sociales y socio-ecológicos a concepciones de estos como sistemas complejos, compuestos de múltiples actores, cuyas relaciones no son necesariamente lineales o unidireccionales. Junto a esta evolución conceptual se ha presentado un cambio en la obtención y procesamiento de la información asociada a dichos sistemas de estudio. En general, la disponibilidad de datos e información se ha incrementado exponencialmente, al tiempo que contamos con herramientas analíticas cada vez más sofisticadas y con una mayor capacidad de procesamiento de información.

El desarrollo acelerado de los métodos y herramientas analíticas conlleva necesariamente a un cuestionamiento sobre la capacidad que tienen las universidades, como máximos centros de generación de conocimiento científico, de asimilar dichos cambios y ajustar su quehacer. Por una parte, el desarrollo de la investigación científica de frontera requiere la asimilación de dichos avances durante el proceso de producción de conocimiento y, por otro lado, la formación de estudiantes que se inician en la construcción del conocimiento científico requiere que la comunidad docente esté ampliamente capacitada y actualizada. Sin embargo, se ha demostrado la existencia de un desfase temporal entre el desarrollo de los métodos y herramientas empleados para la generación del conocimiento científico y su oferta en los programas de formación de nivel superior. En otras palabras, los estudiantes de este nivel educativo aprenden métodos y herramientas desactualizados,

que tienen un alcance limitado para el análisis de aquellos objetos de estudio que se pretende aborden en el futuro.

La experiencia docente de los coordinadores de este manual, así como la de los profesores de las licenciaturas y posgrados ofertados en la UNAM Unidad Morelia, nos indica que existe, en general, un conocimiento deficiente sobre métodos y herramientas analíticas acordes a los objetos de estudio. Por citar un ejemplo, en el caso de la licenciatura de Ciencias Ambientales de la ENES Morelia, los sistemas ambientales son abordados desde el discurso como sistemas complejos, pero el currículo no incluye la capacitación en torno a métodos y herramientas *ad hoc*. Aunque este tipo de limitaciones han sido previamente identificadas y abordadas, se requiere de un esfuerzo continuo de actualización, tanto de los profesores como de los estudiantes que se encuentran desarrollando sus investigaciones de grado y posgrado.

Como una estrategia para ayudar a solucionar dicho rezago, durante el año 2017 se implementó el proyecto denominado "Escuela de Métodos". La Escuela propuso el desarrollo de una serie de talleres de capacitación sobre métodos y herramientas de análisis de datos usando el lenguaje de programación R, abarcando áreas de interés general para la comunidad académica del campus Morelia de la UNAM. Se impartieron un total de seis talleres, a los que asistieron 105 personas. Adicionalmente, se generaron video-cápsulas que sirven como presentación de la Escuela y como invitación al abordaje de algunos de los métodos y análisis de los talleres. Dado el alto nivel de satisfacción de los asistentes, tanto con los contenidos como con el desempeño de los instructores, así como la solicitud de

talleres adicionales sobre temáticas específicas, la Escuela de Métodos sigue trabajando hoy día.

Queremos agradecer a la Dirección General de Asuntos de Personal Académico (DGAPA) de la UNAM por el apoyo económico recibido a través del proyecto PAPIME 207917: "Escuela de Métodos: una estrategia para la actualización docente en torno a métodos y herramientas de análisis de información". Este proyecto sin lugar a dudas ha sido la semilla necesaria para echar a andar la idea de una Escuela de Métodos de carácter permanente. Agradecemos también a la Escuela Nacional de Estudios Superiores Unidad Morelia (ENES), al Instituto de Investigaciones en Ecosistemas y Sustentabilidad (IIES) y la Unidad de Investigación sobre Representaciones Culturales y Sociales (UDIR) por todo el apoyo institucional brindado.

Sobre este Manual

El presente manual constituye uno de los principales logros de la Escuela de Métodos 2017. Sus cinco capítulos presentan por escrito los contenidos abordados en cada uno de los talleres ofrecidos en la emisión 2017 de la Escuela. El objetivo principal es presentar algunos métodos y herramientas de análisis de datos de utilidad en los contextos de las ciencias sociales y ambientales. En todos los casos se usan paquetes del lenguaje de programación R, que es el *software* libre para análisis estadístico más utilizado en el ámbito de la investigación científica.

Cada uno de los capítulos de este manual aborda un método o herramienta específica de análisis de datos. En primer lugar se hace una pequeña introducción al tema, seguida de la presentación de al menos

un ejemplo o caso de estudio sobre el cual se desarrolla el método y se muestra paso a paso el uso de herramientas específicas. La primera parte de este manual contiene dos capítulos introductorios al manejo de información usando el lenguaje de programación R. El primer capítulo presenta una introducción al uso de R, dirigida a personas que no han tenido ningún contacto con este *software*. Como cualquier otra "lengua", R puede ser al principio un poco difícil de abordar, ya que es poco intuitivo; sin embargo, si el análisis de datos constituirá una parte importante del quehacer del lector, no cabe la menor duda de que el costo de su aprendizaje será compensado rápidamente por las posibilidades de análisis de información que ofrece.

En el segundo capítulo se presenta una introducción al manejo de tablas y bases de datos en R. Este capítulo es de utilidad particular para personas acostumbradas a manejar tablas de datos en programas como Excel®, pero que por el volumen de la información o la complejidad de las operaciones que deben realizar, necesitan herramientas más potentes y flexibles para el desarrollo de sus actividades.

En la segunda parte del manual se abordan métodos de análisis específicos. En el tercer capítulo se presenta una introducción al análisis de redes ecológicas, que le permitirá al lector conocer los conceptos y las funciones para realizar análisis de redes de interacción bipartita, es decir, aquellas en las que las especies pueden ser agregadas en dos grupos. Si bien este capítulo está dirigido al análisis de redes ecológicas, los métodos asociados pueden ser también empleados para analizar otro tipo de redes, por ejemplo, las sociales. En el cuarto capítulo se presentan varias herramientas para el análisis de datos espaciales, en particular se abordan los conceptos y métodos relacionados con el análisis de dependencia y asociación espacial que

permitirán a geógrafos, ecólogos o expertos en áreas afines, describir patrones espaciales de distribución de un fenómeno. Por último, en el quinto capítulo se presenta el uso del paquete de análisis cualitativo RQDA, mismo que es de gran utilidad para aquellas personas cuyo ámbito de estudio son las ciencias sociales o las ciencias políticas y que están interesadas en hacer análisis de contenido o análisis del discurso. En este apartado el lector encontrará tanto la fundamentación teórica del análisis de contenido como su desarrollo práctico usando el paquete RQDA.

Como puede notarse, este manual será de utilidad tanto para las personas que no estén familiarizadas, pero que les interesa aprender a usar el lenguaje R, como para aquellas que ya conocen el lenguaje, pero que quieren perfeccionar su uso e implementar paquetes específicos. En cualquiera de los casos, este manual pretende constituir una experiencia que invite al lector a seguir profundizando en el manejo del lenguaje R mediante la consulta de recursos más avanzados. Esperamos que por el hecho de estar escrito en español y ser de acceso libre y gratuito, se convierta en una herramienta de consulta valiosa para académicos y estudiantes de escuelas y universidades latinoamericanas.

Finalmente, quisiéramos agradecer a las personas y grupos que fueron importantes para la consecución de este manual. En primer lugar, queremos agradecer a todos los invitados como instructores de los talleres de la Escuela de Métodos 2017. Los contenidos que nos facilitaron constituyeron el insumo básico para la generación de este manual. Agradecemos a la ENES unidad Morelia por la facilitación de las instalaciones para el desarrollo de los talleres, así como a sus técnicos de cómputo por su constante apoyo en la adecuación de las aulas y los

equipos para el desarrollo los talleres. Esperamos que este manual sea tan su agrado, tanto como fue el producirlo para nosotros. Siéntanse libres de reproducirlo y difundirlo entre sus comunidades.

Francisco Mora Ardila, “Pacho”
Mario Martínez Salgado, “Jefecito”
Ana Yésica Martínez Villalba, “Yesi”
Daniela López, “Jefecita”
José Luis Arroyo, “Iñárritu”

Morelia, Michoacán, a 12 de marzo de 2018

Antes de empezar

Todos los ejercicios y ejemplos presentados están diseñados para ser ejecutados con R. Para realizar los ejercicios de los capítulos 1, 2, 4 y 5 es necesario que tengas disponibles algunos archivos. Estos documentos están disponibles en la página de Github de la Escuela de Métodos: github.com/metodosmorelia, o bien puedes descargarlos directamente con el vínculo: https://github.com/metodosmorelia/Manual_vol._1_Datos/archive/master.zip

Bájalos en una sola carpeta, misma que debes establecer como tu carpeta de trabajo (ver capítulo 1, sección 1.1.5) para el desarrollo de los cuatro capítulos mencionados.

PARTE I. INTRODUCCIÓN AL MANEJO DE INFORMACIÓN USANDO R

Capítulo 1. Introducción al lenguaje R[1]

Mario Martínez Salgado
Unidad de Investigación sobre Representaciones Culturales y Sociales
Universidad Nacional Autónoma de México

Este capítulo tiene como propósito proveer una introducción a las capacidades del lenguaje R. El objetivo principal, entonces, es construir un punto de partida para el aprendizaje autodidacta del lenguaje. Además de presentar los elementos básicos, se mostrarán algunos de los recursos que existen en la Red para el manejo de bases de datos y el procesamiento de información.

1.1. Preliminares

1.1.1. R y RStudio

R es un lenguaje de alto nivel y un entorno para el análisis de información. Es un *software* de código abierto que está disponible para Linux, Mac y Windows. R es una alternativa sobresaliente a las costosas licencias de la mayoría de los paquetes estadísticos.

En R los usuarios colaboran con el desarrollo del proyecto bajo la supervisión del CRAN (*Comprehensive R Archive Network*). Las aportaciones son incluidas como librerías descargables de la Red. Aunque no tiene un servicio de soporte, R tiene un sistema diverso y bien organizado de retroalimentación entre los usuarios y los desarrolladores.

[1] Los datos necesarios para la realización de algunos de los ejercicios de este capítulo se encuentran en el vínculo: https://github.com/metodosmorelia/Manual_vol._1_Datos/archive/master.zip

Para instalar R se debe ir a la página del proyecto:

1. Ir a la página del proyecto: cran.r-project.org
2. Elegir el sistema operativo correspondiente. Por ejemplo, **Windows**.
3. Pulsar la opción **base** (o *install R for the first time*)
4. Seleccionar la versión de R (Download R 3.6.0 for Windows) y guardar el archivo de instalación en la computadora (R-3.6.0-win.exe). Es recomendable visitar regularmente el CRAN para tener la versión más reciente.
5. Pulsar el archivo ejecutable y seguir el proceso de instalación habitual.

En el escritorio aparecerán 2 íconos de R: **R x64 3.6.0** y **R i386 3.6.0**. Al pulsar el primero aparece la siguiente ventana:

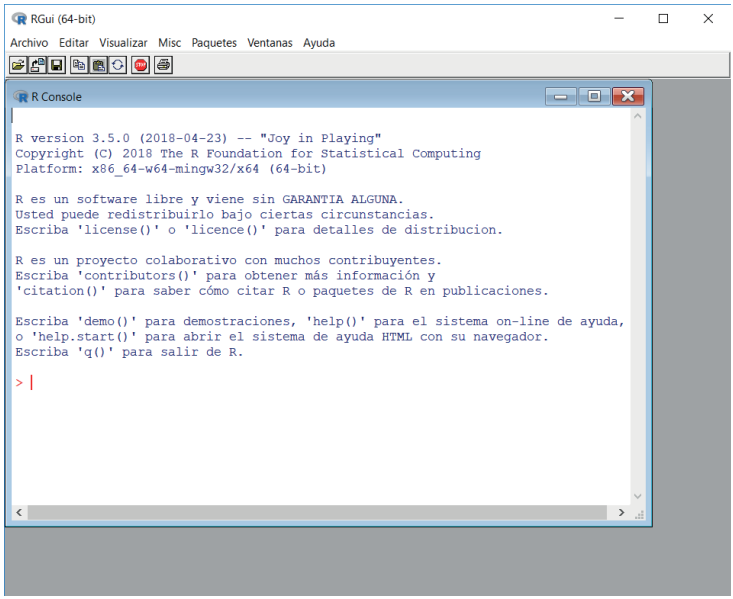


Figura 1. Pantalla principal de R.

R y muchos de los desarrollos (paquetes) se financian con recursos provenientes de donaciones, subvenciones y otras aportaciones. Por esta razón se invita a los usuarios a que le den el crédito cuando lo hayan utilizado. Se sugiere citar de la siguiente manera:

R Core Team (2019). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

Una interfaz ampliamente utilizada que dota a R de un entorno más amigable es RStudio, esto significa que primero debemos instalar R y después RStudio. Los pasos para instalarlo son los siguientes:

1. Ir a la página de RStudio: rstudio.com
2. Seleccionar *Download Rstudio.*
3. Elegir la opción gratuita (*FREE*).
4. Descargar el instalador que corresponda a nuestro sistema operativo. Por ejemplo, en **Windows**: *RStudio-1.2.1335 - Windows Vista/7/8/10.*
5. Pulsar el archivo ejecutable y seguir el proceso de instalación habitual.

La pantalla de RStudio se compone de cuatro paneles (Figura 2). Arriba a la izquierda es el lugar donde escribiremos nuestro código (*script*), abajo aparece la consola, arriba a la derecha se sitúa el *workspace* y debajo aparecerán los gráficos, la ayuda y otros recursos.

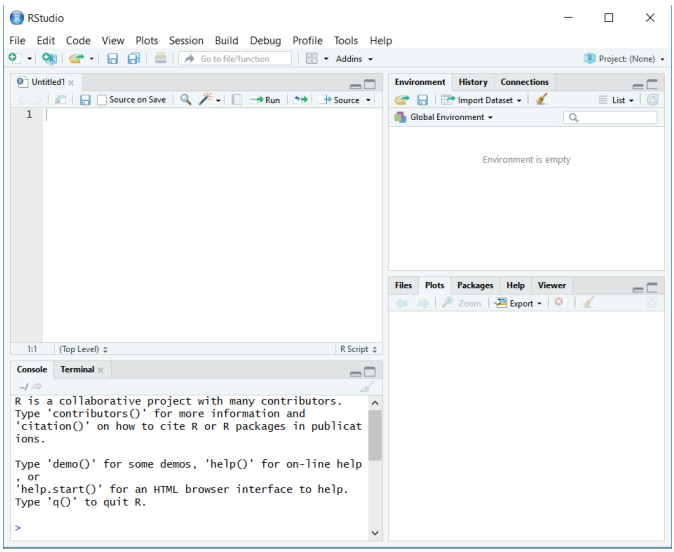


Figura 2. Pantalla principal de RStudio.

1.1.2. Comentarios en el código

Durante el trabajo de programación es común querer escribir alguna nota o recordatorio. Para incluir este tipo de anotaciones debemos anteponer el signo numeral # a la frase que queremos sea tomada como comentario.

```
# Ejemplo
```

1.1.3. Ejecutar una instrucción

Para "correr" una instrucción se debe ubicar el cursor al inicio de la línea de comando o seleccionar un conjunto de éstas y oprimir simultáneamente las teclas *Ctrl* y *Enter*.

Notar que en la consola (panel de abajo) se despliega el resultado de la instrucción.

1.1.4. La Consola

En la consola el signo > al final se llama *prompt* y significa que R está listo para ejecutar la siguiente tarea. En cambio, un signo + indica que no se ha terminado la tarea; esto es, que a la instrucción le falta "algo". Esto ocurre, la mayoría de las veces, cuando no se cierra un paréntesis o un corchete. Lo cual se corrige "cerrando" el paréntesis y ejecutándolo como una instrucción o, si no tenemos idea en dónde está la falla, podemos cancelar la tarea si nos ubicamos en la consola y oprimimos la tecla *Esc*.

1.1.5. Directorio de trabajo

Por defecto la carpeta *Documentos* es la carpeta de trabajo. R guardará en esta carpeta todo aquello que salvemos (gráficos, bases de datos, tablas resumen, etc.), también "jalará" de forma directa la información que indiquemos.

Para cambiar de carpeta (ver Figura 3):

1. En el menú principal pulsar la opción ***Session***
2. Desplegar las opciones de ***Set Working Directory***
3. Elegimos la opción ***Choose Directory...***
4. Seleccionar la carpeta de interés, por ejemplo: el escritorio (*Desktop*)

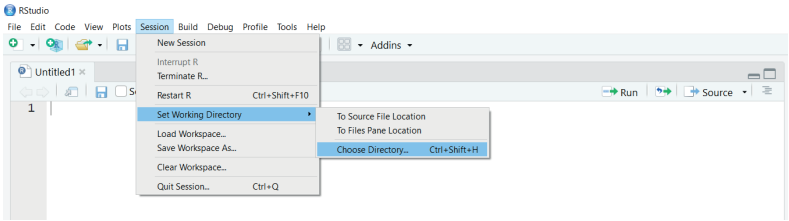


Figura 3. Elegir la carpeta de trabajo

En la Consola, justo debajo de donde dice "Console", aparecerá la ruta de la nueva carpeta de trabajo; y no sólo eso, en la consola misma se muestra el comando que hizo posible esta acción: *setwd("C:/Users/marius/Desktop")*. Esto significa que podemos cambiar la carpeta de trabajo directamente en el *script* si modificamos la ruta que aparece entrecomillada en *setwd()*.

1.1.6. Ayuda

En la Red hay un sinfín de sitios con ayudas, ejemplos y otros recursos. Lo más sencillo es hacer una consulta en Google. También:

- Los sitios stats.idre.ucla.edu y statmethods.net son muy útiles, sobre todo cuando se está empezando a utilizar R.
- La comunidad de la página de Facebook R project en español es muy activa y muchas veces atiende en tiempo récord prácticamente cualquier inquietud, no importa que tan básica sean las dudas de los usuarios.
- En la página de de RStudio están disponibles algunos acordeones para ser descargados de manera gratuita.
- El sitio r-bloggers.com es muy útil para conocer los nuevos desarrollos.

1.2. Operadores

Hay distintos tipos de operadores. Los hay aritméticos (+, -, *, / y ^), relacionales (>, >=, <, <=, == y !=), lógicos (& y |) y otros que realizan tareas específicas, como el operador : que crea una secuencia de valores. Por ejemplo:

```
3/7 ; 5^-3
 [1] 0.4285714
 [1] 0.008
-3:7
 [1] -3 -2 -1 0 1 2 3 4 5 6 7
7 < 9
 [1] TRUE
```

Notar que el operador; permite realizar dos tareas en una misma línea de comando.

1.3. Objetos

El trabajo en R se realiza mediante objetos. Hay objetos simples y otros más elaborados. Un objeto en R puede ser una tabla con datos, una base de datos, una variable o un valor. Con el operador <- o = se asigna un valor a un objeto. Los objetos aparecen en el *workspace* (panel superior de la derecha).

1.3.1. Numéricos

```
x <- 2
y = 9
```

Si "corremos" únicamente el nombre del objeto, su contenido aparecerá en la consola.

```
x
 [1] 2
```

Operaciones con objetos

Podemos hacer operaciones con los objetos o, incluso, podemos crear nuevos objetos.

```
x + y
 [1] 11
x + 5
 [1] 7
z <- y^2
z
 [1] 81
```

1.3.2. Caracteres

Una palabra o una frase también puede ser un objeto, para ello sólo debemos entrecomillarla, por ejemplo:

```
curso <- "Introducción al lenguaje R"
curso
 [1] "Introducción al lenguaje R"
```

1.3.3. Vector numérico

Un vector numérico como objeto es una cadena de valores numéricos. Una cadena de valores está delimitada por `c()` y cada valor debe estar separado por una coma.

```
cm <- c(167, 172, 153, 164, 182, 147, 171, 155, 163, 170)
```

Así como con los objetos numéricos, también podemos realizar operaciones o crear nuevos objetos a partir de los vectores numéricos. Por ejemplo, a partir del objeto `cm`, creamos el objeto `mts`.

```
mts <- cm/100
mts
 [1] 1.67 1.72 1.53 1.64 1.82 1.47 1.71 1.55 1.63 1.70
```

1.3.4. Vector de caracteres

Un conjunto de palabras puede dar lugar a un vector de caracteres siempre que se delimiten éstas por `c()` y se separen por comas.

```
colores <- c("Azul", "Rojo", "Amarillo")
```

1.3.5. Matrices

Con la función `matrix()` podemos construir una matriz. Por ejemplo, si tomamos como insumo el vector `cm`:

```
matriz_h <- matrix(cm, nrow = 5, ncol = 2, byrow = TRUE)
matriz_h
 [,1] [,2]
 [1,] 167 172
 [2,] 153 164
 [3,] 182 147
```

```
[4,] 171 155
[5,] 163 170
```

La función `matrix()`, como otras, permite la utilización de argumentos para utilizar ciertas opciones. En este contexto `nrow` y `ncol` sirven para indicarle a R cuál es la dimensión de la matriz. En el ejemplo la dimensión de `matriz_h` es 5x2.

El argumento `byrow`, por su parte, sirve para indicar la forma en que los elementos del vector `cm` se despliegan en la matriz, puede ser en horizontal (TRUE) o vertical (FALSE).

```
matriz_v <- matrix(cm, nrow=5, ncol=2, byrow=FALSE)
matriz_v
 [,1] [,2]
[1,] 167 147
[2,] 172 171
[3,] 153 155
[4,] 164 163
[5,] 182 170
```

1.3.6. Factor

Un factor es un objeto que almacena el valor de una variable nominal. Por ejemplo:

```
sexo <- factor(c("Mujer", "Hombre", "Mujer", "Mujer",
"Hombre"))
```

¿Cuál es la diferencia entre un factor y un vector de caracteres? Para dar respuesta a esta pregunta usamos la función `summary()`.

```
summary(sexo)
 Hombre Mujer
 2 3
summary(colores)
 Length Class Mode
 3 character character
```

En el caso del factor `sexo` el resultado que obtenemos es una distribución del contenido. Mientras que el resumen que obtenemos del vector de caracteres `colores` es información sobre los atributos del objeto.

1.3.7 Data frame

Un *data frame* es un arreglo rectangular de datos. Se parece a una matriz, aunque se distingue de esta porque es más general. En un *data frame* las columnas pueden tener diferentes clases de objetos (numéricos, factores, etc.), en una matriz no. Por ejemplo:

```
datos <- data.frame(c(0,1), c("Negro", "Verde"), c(TRUE,FALSE))
datos
 c.0..1. c..Negro....Verde.. c.TRUE..FALSE.
 1 0 Negro TRUE
 2 1 Verde FALSE
```

La función `data.frame` pega el contenido de los vectores `(0,1)`, `("Negro", "Verde")` y `(TRUE, FALSE)` para crear el objeto `datos`. Notar que cada uno de los vectores tiene dos entradas, si uno tiene más (o menos), R no podrá realizar el pegado.

```
datos <- data.frame(c(0,1), c("Negro", "Verde"), c(TRUE,FALSE))
datos
 c.0..1. c..Negro....Verde.. c.TRUE..FALSE.
 1 0 Negro TRUE
 2 1 Verde FALSE
```

R nombra por defecto las columnas con las mismas cadenas de valores. Para renombrar las variables podemos usar la función `names()`.

```
names(datos) <- c("var1", "var2", "var3")
datos
 var1 var2 var3
 1 0 Negro TRUE
 2 1 Verde FALSE
```

Borrar objetos del workspace

Con la función `rm()` podemos borrar objetos del *workspace*. Si queremos borrar los objetos `x` y `y`, por ejemplo:

```
rm(x, y)
```

Para borrar todos los objetos "corremos" el comando `rm(list = ls())`

1.4. Funciones

Las funciones tienen nombre, argumentos y entregan un resultado (valor, gráfico, archivo, entre otros). Con las funciones `mean()` y `var()`, por ejemplo, obtenemos la media y la varianza de un conjunto de valores:

```
mean(c(1.51, 1.52, 1.53, 1.53, 1.54, 1.55, 1.57, 1.58, 1.59, 1.60))
 [1] 1.552
var(c(1.51, 1.52, 1.53, 1.53, 1.54, 1.55, 1.57, 1.58, 1.59, 1.60))
 [1] 0.0009733333
```

o de un objeto:

```
mean(mts)
 [1] 1.644
var(mts)
 [1] 0.01080444
```

Si en el objeto o el conjunto de valores hay información faltante, entonces es posible que la función no entregue el resultado esperado. Por ejemplo, supongamos que a 10 individuos le pedimos su estatura y uno no la reportó: NA (*Not Available*); al calcular la media el resultado será NA.

```
estatura <- c(1.67, NA, 1.73, 1.74, 1.72, 1.67, 1.71,
1.85, 1.63, 1.70)
```

```
mean(estatura)
```

```
 [1] NA
```

En este caso debemos agregar el argumento `na.rm = TRUE`, que quiere decir *remove missing values*, para calcular la media de los que sí dieron su estatura.

```
mean(estatura, na.rm = TRUE)
```

```
 [1] 1.713333
```

1.5. Paquetes y librerías

Los paquetes contienen un conjunto de funciones útiles para diversos fines: manejo de datos, elaboración de gráficos, programación, etc.

En la Red existen un sin número de paquetes y están disponibles al público de manera gratuita. En r-bloggers.com y en otros *blogs* dedicados a R es posible enterarse de los nuevos desarrollos.

Para utilizar estos recursos se debe, primero, descargar e instalar el paquete de interés, esta tarea la realizamos utilizando la función `install.packages()`; y segundo, cargar el paquete a la sesión de trabajo, esto lo realizamos con la función `library()`.

El paquete `pyramid`, por ejemplo, contiene la función `pyramid()`. Esta función permite hacer de manera sencilla una pirámide de población.

```
install.packages("pyramid")
library(pyramid)
```

El paquete incluye un conjunto de datos (*data frame*) llamado `GunmaPop2005`. Al aplicar la función `pyramid()` a este objeto el resultado es la siguiente pirámide de población.

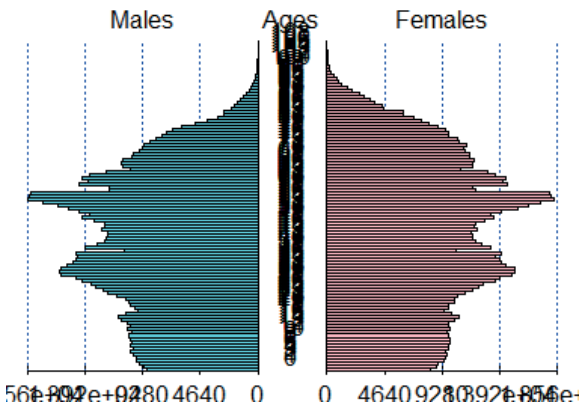


Figura 4. Pirámide de edad

La edición de este gráfico la dejamos para más adelante. En la última sección del capítulo revisaremos algunas capacidades útiles de R para realizar este trabajo.

La función `head()` aplicada al objeto `GunmaPop2005` nos muestra los primeros registros de dicho objeto.

```
Males Females Ages
1 8872 8323 0
```

```
2 9144 8750 1
3 9528 8964 2
4 9812 9359 3
5 9817 9559 4
6 10049 9605 5
```

Para usar la función `pyramid()` sobre otros datos (objeto) debemos asegurarnos, primero, que éstos tienen la misma estructura.

Para ejemplificar lo anterior utilizamos la población rural de Michoacán en 2015. Esto es, primero generamos los objetos `Hombres`, `Mujeres` y `Edad`, y después los integramos utilizando la función `data.frame()` para crear el objeto `mich15`.

```
Hombres <- c(227088, 221051, 222669, 208826, 200237,
164498, 150676,
 144043, 135905, 108809, 102534, 83350, 68458, 170317)
Mujeres <- c(218558, 218376, 219155, 215099, 214932,
188959, 170648,
 161999, 151192, 127456, 118636, 96193, 76053, 195867)
Edad <-c("0-4", "5-9", "10-14", "15-19", "20-24",
"25-29", "30-34",


 "35-39", "40-44", "45-49", "50-54", "55-59", "60-
64", "65 y +")
mich15 <- data.frame(Hombres, Mujeres, Edad)
```

El resultado es:

```
Hombres Mujeres Edad
1 227088 218558 0-4
2 221051 218376 5-9
3 222669 219155 10-14
4 208826 215099 15-19
5 200237 214932 20-24
6 164498 188959 25-29
```

Por último aplicamos la función `pyramid()` sobre el objeto `mich15`

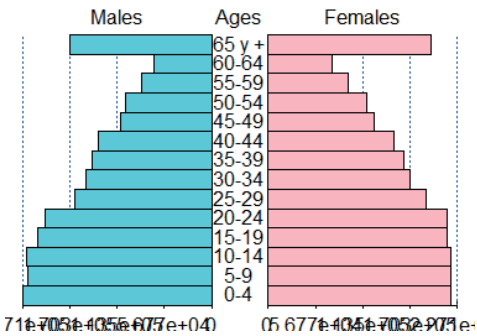


Figura 5. Pirámide de edad. Población rural de Michoacán, 2015.

Más adelante regresaremos a este gráfico para editarlo.

1.6. Importar datos

En la práctica es común encontrar o tener la información almacenada en varios formatos. Los formatos más comunes son: DBF, CSV, DTA, SAV y DAT. El INEGI, por ejemplo, pone a disposición de los usuarios los microdatos en varios formatos. Los microdatos de la Encuesta Intercensal de 2015 están disponibles en CSV, DTA, SAS y SAV.

Para varios procesadores de datos esto es un problema, pues sólo trabajan con cierto formato. En R esto no representa ningún inconveniente. R puede cargar cualquier base de datos, no importa el formato, sólo se necesita la librería `foreign`.

```
install.packages("foreign")
library(foreign)
```

En esta librería se encuentra la función `read.dta()`, por ejemplo, la cual nos permite cargar datos en formato DTA

```
enut <- read.dta("C:/Users/marius/Desktop/DATOS/CURSOS/ENUT.dta")
```

Si los datos están en la carpeta de trabajo, basta con poner el nombre del archivo. En el ejemplo anterior sería read.dta("*ENUT.dta*"), en otro caso se debe especificar la ruta y el archivo.

Para saber cuál es la carpeta de trabajo usamos la función *getwd*().

```
getwd()
```

Para cambiarla usamos, como se vio en el apartado 1.1.5, la función `setwd()` con la ruta deseada.

Por otro lado, R tiene su propio formato de almacenado de datos: `RData`. Para guardar un objeto en dicho formato usamos la función `save()`. Por ejemplo:

```
save(mich15, file = "Michoacán 2015.RData")
```

Para cargar este tipo de archivos utilizamos la función `load()`.

```
load("Michoacán 2015.RData")
```

1.7. Explorar una base de datos

Cuántos registros y variables tiene la ENUT (Encuesta Nacional sobre Uso del Tiempo de 2014).

```
dim(enut)
 [1] 40430 34
```

¿Qué variables tiene?

```
names(enut)
 [1] "control" "viv_sel" "hogar" "id_hog" "n_ren"
 [6] "paren" "sexo" "edad" "asiste_esc" "indigena"
 [11] "niv" "gra" "escoacum" "p5_1" "p5_2"
  [16] "p6_1_1_1" "p6_1_1_2" "p6_1_1_3" "p6_1_1_4"
"p6_4_3_1"
```

```
 [21] "p6_4_3_2" "p6_4_3_3" "p6_4_3_4" "p6_4_3_5"
"p6_5_2_1"
 [26] "p6_5_2_2" "p6_5_2_3" "p6_5_2_4" "p6_5_2_5"
"p7_2_1"
 [31] "p7_2_2" "p7_2_3" "p7_2_4" "p7_3"
```

Para acceder a las variables debemos poner el nombre del objeto, seguido del signo $ y del nombre de la variable. Por ejemplo, la siguiente instrucción despliega los primeros valores de la variable *edad*.

```
head(enut$edad)
 [1] 28 28 51 49 23 21
```

También podemos obtener la edad media de los entrevistados:

```
mean(enut$edad)
 [1] 39.69735
```

O saber cuál es la edad mínima y la máxima:

```
range(enut$edad)
 [1] 15 99
```

NOTA: Los últimos códigos de la variable 'edad' refieren a:

97. "97 y más años"
98. "No sabe en personas de 12 y más años"
99. "No sabe en personas menores de 12 años"

1.7.1. Resumen de datos

Con la función `table()` podemos obtener la distribución de frecuencias de los datos. Por ejemplo, la distribución de los individuos según nivel de felicidad (*p7_3*) la obtenemos mediante:

```
table(enut$p7_3)
 1  2  3  4  5
 166 1088 6221 21064 10335
# 1 - Nada; 2 - Poco; 3 - Más o menos; 4 - Feliz; y
5 - Muy feliz
```

Para obtener la distribución de los individuos por nivel de felicidad incluyendo los valores perdidos (NA) debemos agregar el argumento `useNA = "always"`.

```
table(enut$p7_3, useNA = "always")
 1  2  3  4  5 <NA>
 166 1088 6221 21064 10335 1556
```

La siguiente instrucción da como resultado una tabla de contingencia que nos indica el nivel felicidad (*p7_3*) por sexo:

```
table(enut$p7_3, enut$sexo)
 1  2
 1 59 107
 2 433 655
 3 2883 3338
```

```
4 10096 10968
5 4619 5716
```

Hasta ahora sólo hemos obtenido distribuciones absolutas. Para obtener la frecuencia relativa de la felicidad (*p7_3*) por sexo usamos una doble función: primero `table()` y luego `prop.table()`:

```
prop.table(table(enut$p7_3, enut$sexo), margin = 1)
  1 2
1 0.3554217 0.6445783
2 0.3979779 0.6020221
3 0.4634303 0.5365697
4 0.4793012 0.5206988
5 0.4469279 0.5530721
```

El argumento `margin = 1` hace que las proporciones resultantes sumen 1 por renglón (prop. hombres + prop. mujeres = 1). El argumento `margin = 2` hará que la suma de cada columna sume 1.

```
prop.table(table(enut$p7_3, enut$sexo), margin = 2)
  1 2
1 0.003261470 0.005148191
2 0.023935876 0.031514627
3 0.159369818 0.160604311
4 0.558098397 0.527713626
5 0.255334439 0.275019246
```

Otra manera de resumir un conjunto de datos es con la función `aggregate()`. Esta función nos permite, por ejemplo, calcular el promedio de una variable de acuerdo con cierto criterio. Al correr la siguiente instrucción obtenemos la edad promedio por nivel de felicidad (*p7_3*).

```
aggregate(enut$edad, by = list(enut$p7_3), FUN = mean, na.rm = TRUE)
```

```
 Group.1 x
1 1 51.15663
2 2 45.84651
3 3 43.38531
4 4 39.59775
5 5 36.20977
```

1.8. Selección de información

Hasta ahora hemos utilizado las funciones para operar sobre los vectores como un todo. Sin embargo, en ocasiones sólo se requiere aplicar las funciones a determinados elementos de un vector. Esto se consigue con los corchetes [ ], en tanto que las funciones por paréntesis ( ). En los siguientes ejemplos vemos la diferencia entre uno y otro recurso:

¿Qué resultado obtienen si ejecutan las siguientes instrucciones?

```
mean(enut$edad)
```

```
 [1] 39.69735
```

```
mean(enut$edad < 97)
```

```
 [1] 0.9968835
```

El uso de [ ] nos permite obtener la edad promedio de las personas de menos de "97 años". Es tanto como agregar una restricción al conjunto de observaciones (no me interesan todas las personas, sólo las que tienen menos de "97 años").

```
mean(enut$edad[enut$edad < 97])
 [1] 39.51538
```

También los puedo utilizar para construir un vector con las edades de las mujeres:

```
edad.m <- enut$edad[enut$sexo == 2]
```

O un vector con las edades de las primeras 10 mujeres:

```
edad.m10 <- edad.m[1:10]
```

También los puedo usar para mostrar un elemento de un objeto. Por ejemplo, el 4to elemento del objeto `edad.m10`:

```
edad.m10[4]
 [1] 46
```

Igualmente, este recurso es útil para cambiar los valores de cierta variable. Por ejemplo, la siguiente instrucción convierte los códigos 97, 98 y 99 de la variable *edad* en valores perdidos (NA).

```
enut$edad[enut$edad >= 97 ] <- NA
```

Por otro lado, los corchetes permiten operar sobre los renglones y/o las columnas de un objeto. Esto se logra agregando una coma: `[ , ]`.

Cuando una entrada aparece vacía entonces la operación se realizará sobre todo el renglón o toda la columna. Por ejemplo, si escribimos `[ , 4]`, la operación se realizará sobre todos los renglones de la columna 4. En cambio, si escribimos `[2 , ]` tendremos todas las columnas del renglón 2.

Para ejemplificar lo anterior veamos las siguientes situaciones:

- El valor de una celda específica: registro 3, 7a variable (sexo).

```
enut[3, 7]
 [1] 1
```

- Valores de ciertas variables del 10mo individuo.

```
enut[10, 5:7]
 n_ren paren sexo
 10 2 2 2
```

- El sexo de las personas 19, 113, 217.

```
enut[c(19, 113, 217), "sexo"]
 [1] 2 2 1
```

1.8.1. Ejercicio

a. Obtener los años de escolaridad (*escoacum*) promedio según el nivel de felicidad (*p7_3*).
b. Entre hombres y mujeres mayores de 30 años ¿quién diría que está más satisfecho con la vida (*p7_2_1*)?

```
# R.a
aggregate(enut$escoacum, by = list(enut$p7_3), FUN =
mean, na.rm = TRUE)
# R.b
aggregate(enut$p7_2_1[enut$edad > 30],
 by = list(enut$sexo[enut$edad > 30]), FUN = mean,
na.rm = TRUE)
```

1.8.2. Selección de variables y/o casos

Existen varias formas para seleccionar un conjunto de variables y/o casos de un objeto. Los siguientes ejemplos exploran algunas de estas variantes.

- El objeto `temp` es un extracto de la ENUT. Contiene todos los casos, pero únicamente de las primeras 5 variables.

```
temp <- enut[ , c(1:5)]
head(temp)
 control viv_sel hogar id_hog n_ren
 1 100001 1 1 1011 1
 2 100001 1 1 1011 2
 3 100001 2 1 1021 1
 4 100001 2 1 1021 2
 5 100001 2 1 1021 3
 6 100001 2 1 1021 4
```

- También podemos utilizar el nombre de las variables:

```
temp2 <- enut[ , c(“control”, “viv_sel”, “hogar”,
“id_hog”, “n_ren”)]
```

```
head(temp2)
```

```
 control viv_sel hogar id_hog n_ren
1 100001 1 1 1011 1
2 100001 1 1 1011 2
3 100001 2 1 1021 1
4 100001 2 1 1021 2
5 100001 2 1 1021 3
6 100001 2 1 1021 4
```

- El objeto `temp3` reúne toda la información de los hombres.

```
temp3 <- enut[enut$sexo == 1, ]
```

- El objeto `temp4` tiene la información de las primeras 5 variables, pero únicamente de los menores de 30 años.

```
temp4 <- enut[enut$edad < 30, c(1:5)]
head(temp4)
```

```
 control viv_sel hogar id_hog n_ren
1 100001 1 1 1011 1
2 100001 1 1 1011 2
5 100001 2 1 1021 3
6 100001 2 1 1021 4
8 100001 3 1 1031 2
11 100001 4 1 1041 3
```

- El objeto `temp5` es una base con cierta información de las mujeres menores de 20 años.

```
temp5 <- enut[enut$sexo == 2 & enut$edad < 20,
c("edad", "paren")]
head(temp5)
 edad paren
 19 18 3
 38 16 4
 64 15 3
 67 16 3
 77 17 3
 84 17 3
```

1.8.3. Ejercicio

a. Crear un objeto apartir de los datos de la ENUT con la información de los jóvenes (15 a 24 años) que no estudian (*asiste_esc*) ni trabajan (*p5_1*)

```
# R.a
nini <- enut[enut$edad < 25 & enut$asiste_esc == 2 &
enut$p5_1 == 2, ]
```

1.9. Pegado de bases de datos

Existen varias formas de pegar dos bases de datos. La más segura es con la función `merge()`. Para esto es necesario tener una variable identificador (o un conjunto de ellas). Esta variable permite articular las bases de interés. En la ENUT las variables identificador son *control*, *viv_sel*, *hogar*, *id_hog* y *n_ren*.

Para mostrar algunas de las posibilidades de la función `merge()` generamos los objetos `base1` y `base2`

```
base1 <- enut[c(1:4) , c(1:5, 8)]
base2 <- enut[c(3:6) , c(1:5, 7)]
```

La primera tiene las variables identificador y la variable *edad* de los primeros 4 registros de la ENUT. La segunda también tiene las variables identificador y la variable *sexo* de los individuos 3 al 6.

```
head(base1)
 control viv_sel hogar id_hog n_ren edad
 1 100001 1 1 1011 1 28
 2 100001 1 1 1011 2 28
 3 100001 2 1 1021 1 51
 4 100001 2 1 1021 2 49
head(base2)
 control viv_sel hogar id_hog n_ren sexo
 3 100001 2 1 1021 1 1
 4 100001 2 1 1021 2 2
 5 100001 2 1 1021 3 1
 6 100001 2 1 1021 4 1
```

Al usar la función `merge()` primero se deben señalar las bases que se van a pegar. Las variables indicador se incorporan en la función con el argumento `by`. Con opción `all = TRUE` se mantienen todos los casos; aquellos registros con información faltante se completan con NA.

```
base3 <- merge(base1, base2, by = c(1:5), all = TRUE)
head(base3)
 control viv_sel hogar id_hog n_ren edad sexo
 1 100001 1 1 1011 1 28 NA
 2 100001 1 1 1011 2 28 NA
 3 100001 2 1 1021 1 51 1
 4 100001 2 1 1021 2 49 2
 5 100001 2 1 1021 3 NA 1
 6 100001 2 1 1021 4 NA 1
```

Con `all = FALSE` se excluyen aquellos registros que no tienen las mismas variables identificador.

```
base4 <- merge(base1, base2, by = c("control", "viv_
sel", "hogar",
 "id_hog", "n_ren"), all = FALSE)
head(base4)
 control viv_sel hogar id_hog n_ren edad sexo
 1 100001 2 1 1021 1 51 1
 2 100001 2 1 1021 2 49 2
```

1.9.1. Ejercicio

Considerando las bases `Morelia2015 per.dta` y `Morelia2015 viv.dta`, realizar:

a. Crear un objeto con la información de los hijos mayores de 20 años.
b. Calcular la escolaridad promedio de las personas (*escoacum*), según el material del piso de la vivienda (*pisos*)?

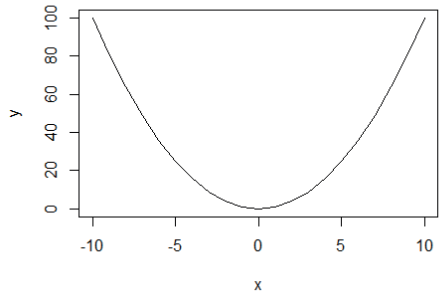
```
# R.a
morelia.per <- read.dta("Morelia2015 per.dta")
morelia.viv <- read.dta("Morelia2015 viv.dta")
hijos20 <- morelia.per[morelia.per$edad > 20, ]

#R.b
morelia <- merge(morelia.per, morelia.viv, by = c("id_viv"), all = FALSE)
aggregate(morelia$escoacum[morelia$escoacum != 99],
  by = list(morelia$pisos[morelia$escoacum != 99]), FUN = mean, na.rm = TRUE)
```

1.10. Gráficos

La función `plot()` permite graficar varios objetos en R. Con esta función podemos gráficar, por ejemplo, la parábola .

```
# Insumos
x <- seq(-10,10)
y <- x^2
# Gráfica
plot(x, y, type = "l")
```

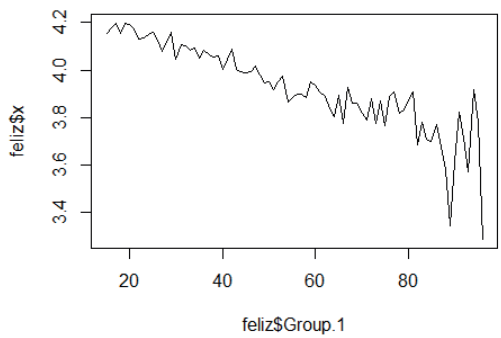


La opción "l" en el argumento `type` produce una gráfica de línea. Hay otras opciones para este argumento (b, h, s).

Con esta función y los datos de la ENUT podemos graficar el nivel de felicidad según la edad.

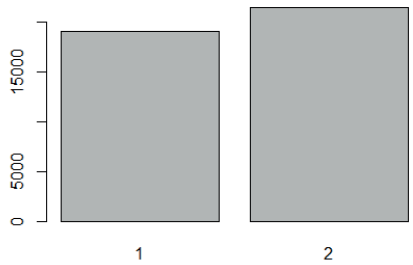
```
# Insumo
feliz <- aggregate(enut$p7_3, by = list(enut$edad),
FUN = mean,
 na.rm = TRUE)
# Estructura del objeto
head(feliz)
 Group.1 x
 1 15 4.156604
 2 16 4.177551
 3 17 4.195568
 4 18 4.156504
 5 19 4.198083
```

```
 6 20 4.194146
# Gráfica
plot(feliz$Group.1, feliz$x, type = “l”)
```



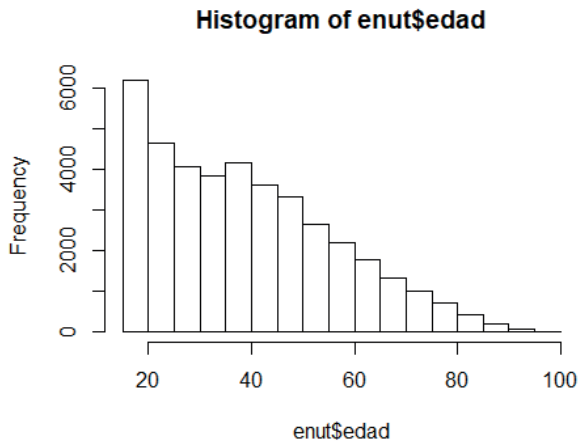
Con la función `barplot()` podemos hacer una gráfica de barras.

```
# Insumo
distsex <- table(enut$sexo)
# Gráfico
barplot(distsex)
```



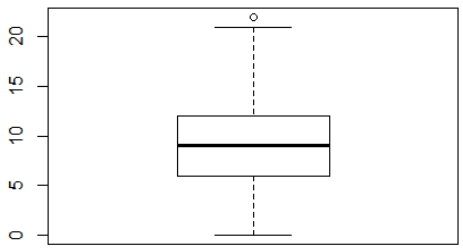
Y con la función `hist()` podemos realizar un histograma.

```
hist(enut$edad)
```



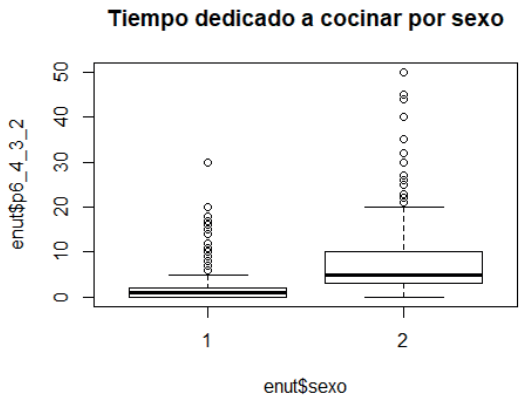
Una gráfica de caja la podemos hacer con la función `boxplot()`.

```
boxplot(enut$escoacum)
```



Con el operador ~ podemos tomar una variable como criterio de división.

```
boxplot(enut$p6_4_3_2 ~ enut$sexo,
 main = “Tiempo dedicado a cocinar por sexo”)
```

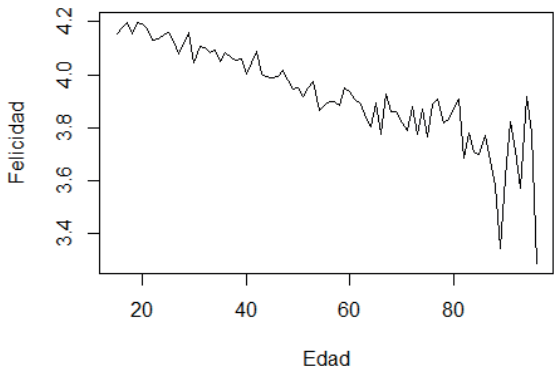


1.10.1. Edición de gráficos

Varios elementos de los gráficos anteriores pueden ser modificados mediante el uso de algunos argumentos. Los argumentos `xlab` y `ylab`, por ejemplo, permiten modificar las leyendas de los ejes.

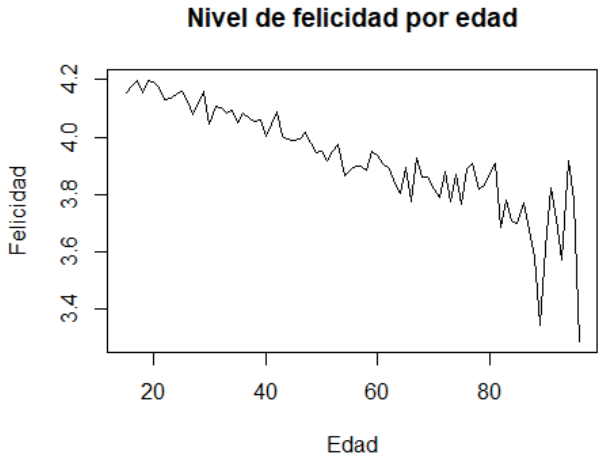
```
   plot(feliz$Group.1, feliz$x, type = “l”, xlab =
“Edad”,

    ylab = “Felicidad”)
```



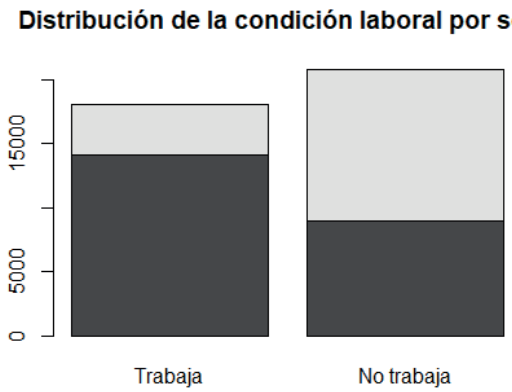
Con el argumento `main`, por otro lado, podemos agregar un título.

```
plot(feliz$Group.1, feliz$x, type = “l”, xlab =
“Edad”,
 ylab = “Felicidad”, main = “Nivel de felicidad por
edad”)
```



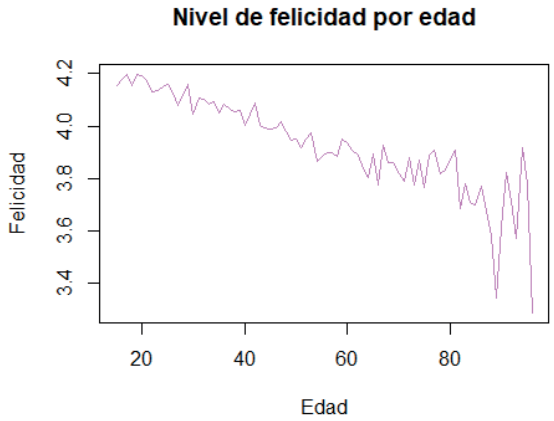
El texto de debajo de categorías en una gráfica de barras.

```
barplot(table(enut$p5_1, enut$sexo),
 main = "Distribución de la condición laboral por sexo",
 names.arg=c("Trabaja", "No trabaja"))
```



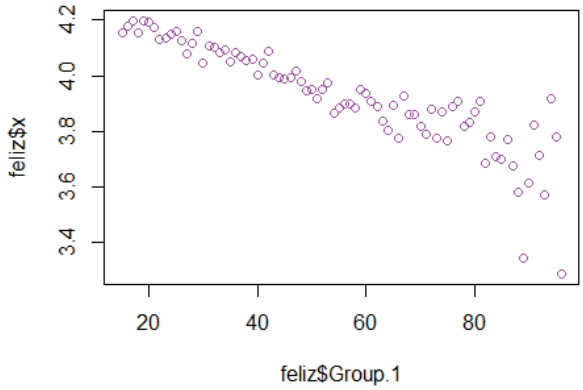
También podemos cambiar el color de la línea con el argumento `col`.

```
plot(feliz$Group.1, feliz$x, type = "l", xlab = "Edad",
 ylab = "Felicidad", main = "Nivel de felicidad por edad",
 col = "violet")
```



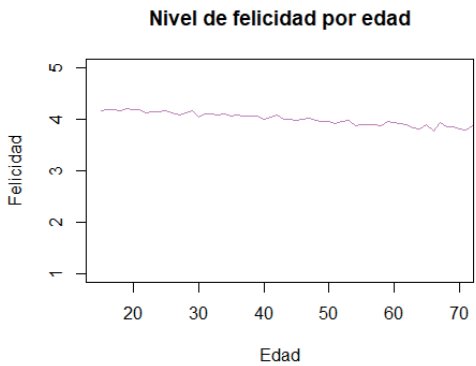
La función `colors()` muestra otras opciones de colores. también la Rueda Cromática de Adobe es un excelente recurso para obtener prácticamente cualquier color, basta con usar el signo # más el código HEX del color deseado. Ejemplo:

```
plot(feliz$Group.1, feliz$x, col = “#FF00FB”)
```



Otro elemento que podemos modificar es la escala de los ejes. Esto se logra con los argumentos `xlim` y `ylim`.

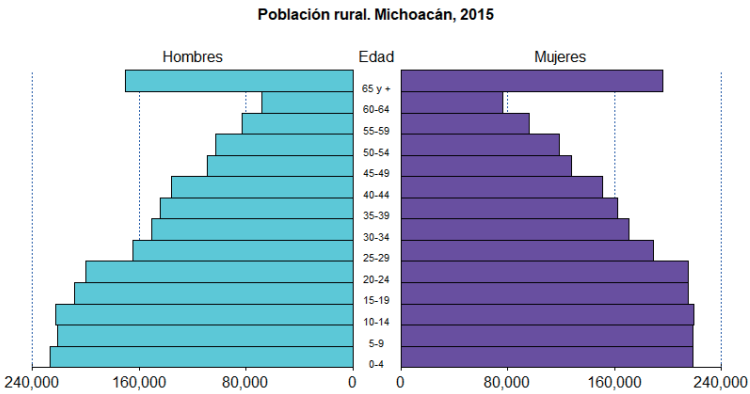
```
plot(feliz$Group.1, feliz$x, type = "l", xlab = "Edad",
 ylab = "Felicidad", main = "Nivel de felicidad por edad",
 col = "violet", xlim = c(15, 70), ylim = c(1, 5))
```



Cada función tiene una serie de argumentos susceptibles de ser modificados que permiten editar el gráfico en cuestión. En la función `pyramyd()`, por ejemplo, al modificar algunos argumentos podemos obtener la siguiente pirámide de población:

```
pyramid(mich15, Llab = "Hombres", Rlab = "Mujeres",
Clab = "Edad",
 main = "Población rural. Michoacán, 2015", cex.main = 1,
  Cstep = 1, Cgap = 0.15, Csize = 0.7, Laxis = (0:3*80000),
```

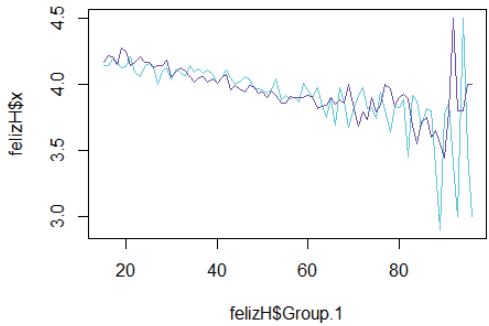
```
 AxisFM = “fg”, AxisBM = “,”, Lcol = “cyan”, Rcol =
“purple”)
```



Agregar elementos

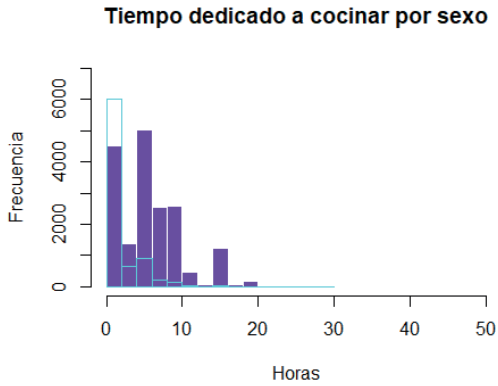
Con la función `lines()` podemos agregar una línea a un gráfico ya existente. Por ejemplo: Nivel de felicidad por sexo y edad

```
# Insumo
felizH <- aggregate(enut$p7_3[enut$sexo == 1],
 by = list(enut$edad[enut$sexo == 1]),
 FUN = mean, na.rm = TRUE)
felizM <- aggregate(enut$p7_3[enut$sexo == 2],
 by = list(enut$edad[enut$sexo == 2]),
 FUN = mean, na.rm = TRUE)
# Gráfica
plot(felizH$Group.1, felizH$x, type=”l”, col=”cyan”)
lines(felizM$Group.1,      felizM$x,      type=”l”,
col=”purple”)
```



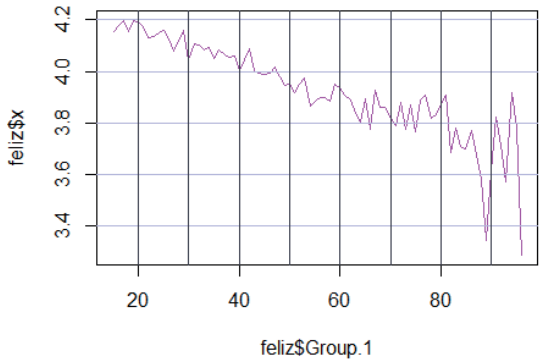
Algo similar a lo anterior se puede hacer para los histogramas. Con el argumento add podemos hacer dicha tarea. Ejemplo: Tiempo dedicado a cocinar por sexo.

```
hist(enut$p6_4_3_2[enut$sexo == 2], ylab = “Frecuencia”, xlab = “Horas”,
 main = “Tiempo dedicado a cocinar por sexo”, breaks = 20,
 ylim = c(0, 7000), col = “purple”, border = FALSE)
hist(enut$p6_4_3_2[enut$sexo == 1], border = “cyan”, add=TRUE)
```



también podemos agregar líneas auxiliares con la función `abline()`.

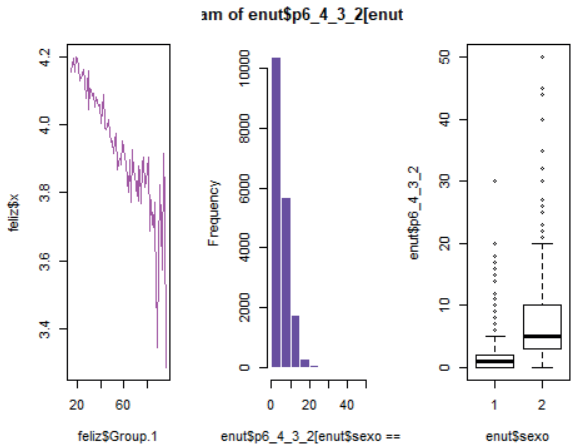
```
plot(feliz$Group.1, feliz$x, type = “l”, col = “#FF00FB”)
abline(h = seq(3, 5, by = 0.2), col = “#C0C0E8”)
abline(v = seq(10, 90, by = 10), col = “#41414F”)
```



1.10.2. Varias gráficas en una imagen

En ocasiones es necesario poner dos o más gráficas en una imagen. En el siguiente ejemplo, con ayuda de la función `par()`, insertaremos un gráfico de línea, un histograma y una gráfica de caja en una imagen:

```
par(mfrow = c(1, 3))
  plot(feliz$Group.1, feliz$x, type = "l", col = "#FF00FB")
 hist(enut$p6_4_3_2[enut$sexo == 2], col = "purple", border = FALSE)
 boxplot(enut$p6_4_3_2 ~ enut$sexo)
```




```
par(mfrow = c(1, 1))
```

Con el argumento `mfrow` generamos los espacios necesarios para las tres gráficas. Si en lugar de `c(1, 3)` ponemos `c(3, 1)` entonces los gráficos se despliegan en vertical.

Al cerrar el código anterior con la instrucción `par(mfrow = c(1, 1))` lo que hacemos es regresar el espacio de dibujo a su estado original.

1.10.3. Guardar imágenes

Las funciones `png()`y `dev.off()` nos permiten guardar una gráfica o figura.

```
png(“C:/Users/marius/Desktop/Figura 1.png”)
 par(mfrow = c(2, 1))
  plot(feliz$Group.1, feliz$x, type = “s”, col = “#72FF0C”)
 hist(enut$edad, col = “#FF8508”, border = FALSE)
 par(mfrow = c(1, 1))
dev.off()
 png
 2
```

Capítulo 2. Gestión de datos con R

Francisco Mora
Instituto de Investigaciones en Ecosistemas y Sustentabilidad
Universidad Nacional Autónoma de México

Introducción

Muchas de las actividades que se desarrollan en nuestra sociedad implican la colecta de datos. Ejemplos simples de ello son la realización de encuestas de satisfacción de los clientes de un almacén, la medición de flujos de autos por las calles de una ciudad, o el estudio de los tipos, tamaños y abundancias de los árboles en un área de extracción forestal. En todos estos casos se toman datos sobre los objetos observados (personas, calles, árboles), los cuales son almacenados en un dispositivo particular (un formato de papel, una grabación de video o un archivo electrónico). Lograr el objetivo para el cual se recabaron los datos (mejorar el servicio prestado, establecer la necesidad de un semáforo, definir los volúmenes de extracción de madera), requiere del procesamiento de los mismos para la generación de información. La planificación del proceso de análisis de datos facilita el desarrollo del mismo, pues permite entre otras cosas, diseñar formatos para la toma de datos que faciliten tanto su colecta como el vaciado posterior, establecer cuántas y cuáles tablas de datos deben ser generadas para vaciar los datos, definir cómo transformar y relacionar dichas tablas y, finalmente, generar resúmenes gráficos o tabulares, así como modelos predictivos. A todo este proceso que va desde la adquisición de los datos hasta la producción de resúmenes y modelos lo podemos llamar **gestión de datos**.

El objetivo de este capítulo es presentar los elementos básicos relacionados con la gestión de datos en R. En particular, se presentan una serie de criterios fundamentales para la creación de tablas de datos ordenadas (*tidy* en inglés), así como un conjunto de funciones que permiten transformar cualquier tabla en una tabla ordenada, es decir, lista para la generación de resúmenes, gráficos o modelos. Estas funciones se encuentran contenidas en **tidyverse**, un conjunto de librerías de R especializadas en la gestión de datos. El capítulo se desarrolla a través de un caso de estudio sobre crecimiento de árboles. Si bien los datos son ficticios, el ejercicio está construido sobre los errores más comunes que los biólogos, ecólogos y profesionales en general cometemos durante el manejo de los datos. El objetivo del ejercicio es que el lector aplique los criterios fundamentales para la creación de tablas de datos ordenadas empleando las funciones básicas de **tidyverse**. El capítulo está construido en gran medida sobre la información contenida en el libro **"R for Data Science"**, escrito por Garrett Grolemund y Hadley Wickham, el cual está disponible gratuitamente en inglés[1] y en español.[2]

2.1. ¿Cómo vaciar mis datos en una tabla?

En el archivo Actividad datos arboles.docx (pica en el nombre del archivo para descargar una carpeta zip que lo contiene. También puedes consultar la dirección electrónica del Github[3] en la introducción del manual) encontrarás una copia de algunas secciones de una libreta de campo, en las que se reportan datos de un muestreo permanente (longitudinal) de árboles en dos localidades (Las Cruces y Buenavista) en dos años consecutivos (2016 y 2017). El muestreo consistió en medir

[1] https://r4ds.had.co.nz/
[2] https://es.r4ds.hadley.nz/?fbclid=IwAR0SBNGOg0gv4Mf_PDervPn2p20h7drgeP2Yc3rrWJkpJzJDvuA_d8-m4rg
[3] https://github.com/metodosmorelia/Manual_vol._1_Datos/archive/master.zip

todos los años, exactamente en la misma ubicación (un cuadro de 10 x 10m), el diámetro a la altura del pecho (DAP, en cm) y la altura (Alt, en m) para todos los troncos cuyo DAP > 5.0 cm. El objetivo del estudio es estimar cuánto crecieron los árboles a través del tiempo. Con base en el documento anterior, desarrolla el siguiente ejercicio:

a) Contesta las siguientes preguntas. Sé lo más detallado posible en tus respuestas.
   - ¿Cuántas o cuáles serían las tablas de datos donde vaciarías esta información? ¿Cuál sería su estructura, es decir, cuáles serían sus columnas y sus filas?
   - ¿Qué *software* emplearías para elaborarlas?
   - ¿Qué cálculos se tendrían que hacer?

b) Ahora, establece una ruta crítica, es decir, enumera y describe de la manera más detallada posible el procedimiento para lograr el objetivo. Describe qué transformaciones y cálculos tendrías que hacer en tu(s) tabla(s), en que *software* lo harías y qué funciones específicas emplearías.

Guarda tus respuestas, pues realizarás el mismo ejercicio al final de este capítulo con el objetivo de poner en evidencia los alcances y limitaciones de tu estrategia de gestión de datos actual. Dicha reflexión deberá llevarte a generar tu propia estrategia de gestión de datos, adecuada a tu situación de trabajo particular.

2.2. Criterios básicos para crear una tabla ordenada

Existen una amplia gama de opciones sobre cómo estructurar una tabla de datos, es decir, sobre cómo deben ir acomodados los datos en un arreglo de filas y columnas. En el ejercicio anterior, es posible que hayas propuesto hacer tablas independientes para los dos sitios o para los dos años, o alternativamente hacer una sola tabla con todos los datos. En este último caso, es posible que hayas pensado que las mediciones de cada año van en diferentes columnas, de tal

forma que toda la información de un mismo árbol va en una sola fila. Puesto que los árboles pueden tener más de un tronco y por ende más de una medida de diámetro, pudiste haber pensado en la creación de tantas columnas de diámetro como fueran necesarias, o alternativamente almacenar todos los datos de diámetro una sola celda. Alternativamente, pudiste haber planteado que cada fila corresponde a un tronco, con tantas filas por árbol como troncos tiene el mismo. Finalmente, ¿qué harías cuando un dato no aparece reportado, por ejemplo una altura? Si bien cada una de las alternativas previas puede servir para propósitos específicos de procesamiento de los datos, existen una serie de criterios generales sobre cómo estructurar una **tabla de datos ordenada**. A continuación se presentan los más básicos.

1. **Cada columna de la tabla es un atributo y cada atributo está en una única columna**. Siguiendo este criterio, todos los datos del atributo diámetro deben ir en una única columna, así sean de años, especies y sitios e individuos diferentes. Asimismo, no es correcto incluir datos de más de un atributo en la misma columna, por ejemplo, incluir diámetros y alturas.
2. **Cada registro de la tabla corresponde a una unidad mínima de observación y cada unidad mínima de observación debe ocupar solo un registro**. Para cumplir este criterio hay que identificar claramente cuál es la **unidad mínima de observación**, es decir, el nivel al cual se está tomando la información más detallada y sobre el cual se construirá la tabla. En nuestro ejemplo, hay diferentes tipos de objetos: los sitios, las especies, los individuos o incluso los troncos. Sin embargo, la información de diámetro y altura está tomada al nivel del tronco, por lo que la tabla debe construirse empleando los troncos como registros. A la unidad mínima de observación también se le denomina **unidad observacional**.
3. **Cada celda debe contener un único dato**. Siguiendo este criterio no sería adecuado, por ejemplo, guardar varios datos de diámetro en una misma celda de la tabla con el fin de que la tabla estuviera construida al nivel de los individuos.

4. **Cada tipo de unidad observacional origina una tabla**. Si el contenido de nuestros datos también hace referencia a características que no son propias de la unidad observacional sino de una unidad jerárquica superior, dicha información debe ir en otra tabla. Por ejemplo, para calcular la biomasa de un árbol comúnmente se usa el dato de densidad de la madera. Sin embargo, la densidad de la madera es un atributo de la especie (o al menos así se asume comúnmente), no de cada tronco, por lo que dicha información no debería ser almacenada en la tabla de datos que estamos construyendo con información de los troncos. Se requeriría una tabla adicional que contenga información referente a las especies, donde cada especie fuera un registro y hubiera una columna de densidad de madera. Veremos más adelante cómo es posible ligar la información contenida en tablas diferentes durante el procesamiento de los datos.

Ahora que conocemos los criterios básicos para la elaboración de una tabla de datos ordenada, abordaremos una serie de funciones disponibles en la librería **tidyverse**, que nos permiten manejar tablas de datos, ya sea con el fin de convertirlas en tablas que siguen los lineamientos de una tabla ordenada, de juntarlas o de generar resúmenes de información. En la siguiente sección abordaremos el uso de funciones que permiten modificar el contenido de una tabla o generar resúmenes de estas. En la sección final abordaremos un par de funciones que permiten transformar la estructura de una tabla, es decir, modificar la disposición de la información ya existente en relación con el número de filas y columnas con el fin de cumplir los lineamientos de una tabla ordenada.

2.3. Manejo de tablas: modificación del contenido y resúmenes

En esta sección nos enfocaremos en la manipulación básica de tablas en R para modificar su contenido mediante la adición o exclusión de atributos o registros. También abordaremos la generación de

resúmenes. Para todo esto emplearemos seis funciones de la librería *dplyr*:

- *filter*: para obtener un subconjunto de filas (registros) de la tabla.
- *arrange*: para reordenar las filas de la tabla.
- *select*: para extraer un subconjunto de atributos.
- *mutate*: para generar nuevos atributos a partir de los ya existentes.
- *summarise*: para generar estadísticas de resumen de la tabla.
- *group_by*: para hacer operaciones por grupos de datos.

Las primeras cinco funciones generan nuevas tablas de datos a partir de una tabla de datos de partida y siguiendo los criterios especificados a través de los argumentos de dichas funciones. Si bien es posible realizar todas estas operaciones con funciones incluidas en otras librerías de R, o incluso usando notación matricial, es mucho más práctico y rápido el uso de las funciones incluidas en *dplyr*.

2.3.1. Instalación de paquetes a usar

Primero instalaremos *tidyverse* y luego cargaremos tres de sus librerías, *dplyr* , *tidyr* y *stringr*

```
install.packages("tidyverse")
library(dplyr)
library(tidyr)
library(stringr)
library(readr)
```

2.3.2. Lectura de datos

Para desarrollar los ejercicios subsecuentes emplearemos otro conjunto de datos sobre plantas. En este caso se trata de una que

contiene datos de plantas leñosas medidas en tres sitios de bosque tropical seco para dos años diferentes, 2004 y 2010. El objetivo del estudio es monitorear el cambio en la biomasa y en el número de especies de estos sitios a través del tiempo. Los datos están almacenados en un archivo de formato delimitado por comas: datos ecologicos.csv (pica en el nombre del archivo para descargar una carpeta zip que lo contiene. También puedes consultar la dirección electrónica del Github en la sección "Antes de empezar" de este manual). Extrae el archivo de la carpeta zip que descargaste a tu carpeta de trabajo, luego importa la tabla de datos a R y guardarla en un objeto llamado *dveg*. Finalmente visualiza toda la tabla empleando la función *View*.

```
dveg <- read.csv("datos ecologicos.csv", header=T,
sep=",")
View(dveg)
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 1 APOPAN 4.3 6
## 2 T 5 AESAMO 3.4 NA
## 3 T 6 CROSUB 1.7 NA
## 4 T 6 CROSUB 1.9 NA
## 5 T 6 CROSUB 2.3 NA
## 6 T 7 RANTET 2.8 NA
## 7 T 7 RANTET 4.5 NA
```

Cada fila en la tabla corresponde a un tallo cuyo diámetro a la altura del pecho (DAP) fue al menos 1 cm. La tabla incluye información del árbol (*individuo*) al que pertenence cada tallo, el sitio en el que se encuentra dicho árbol (*Sitio*), la especie a la que pertenece escrita

en forma de clave (*clave*) y los DAP para los dos años de muestreo (*DAP2004* y *DAP2010*).

2.3.3. Selección de registros empleando *filter*

Supongamos que de nuestra tabla de datos queremos filtrar los registros (filas) correspondientes a la especie *SPOPUR*, o los registros de los individuos identificados con el número *48*. Para ello podemos emplear la función *filter*:

```
filter(dveg, Clave == "SPOPUR")
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 19 SPOPUR 2.9 6.3
## 2 G 10 SPOPUR 13.6 15.8
## 3 G 48 SPOPUR 3.9 14.2
## 4 G 48 SPOPUR 10.0 4.0
filter(dveg, Indiv == 48)
## Sitio Indiv Clave DAP2004 DAP2010
## 1 H 48 ACHGRA 3.0 2.5
## 2 H 48 ACHGRA NA 4.7
## 3 G 48 SPOPUR 3.9 14.2
## 4 G 48 SPOPUR 10.0 4.0
```

En ambos casos, el primer argumento de la función corresponde a la tabla de datos que queremos filtrar (*dveg*), mientras que los siguientes argumentos definen los valores de los atributos que usaremos para filtrar la información. El resultado siempre es una nueva tabla de datos constituida sólo por los registros que cumplen los criterios de interés. Nótese cómo cuando el atributo es de tipo caracter, el valor específico se coloca entre comillas, mientras que cuando es numérico no.

Existen varios *"operadores de comparación"* que se pueden usar para filtrar:

- == : igual que
- != : diferente a
- > : mayor que
- >= : mayor o igual que
- < : menor que
- <= : menor o igual que

Por ejemplo, si queremos obtener una tabla que contenga solo los tallos que tenían en el año 2004 un diámetro mayor a 10 cm escribimos:

```
filter(dveg, DAP2004 >= 20)
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 29 CHLMAN 22.1 22.0
## 2 T 63 CAEERI 24.3 23.9
## 3 T 63 CAEERI 24.5 24.1
## 4 H 33 HELPAL 24.3 NA
## 5 G 15 MIMARE 20.7 24.4
## 6 G 15 MIMARE 29.6 34.0
```

En otras ocasiones queremos seleccionar registros cuyo contenido coincida con una serie de caracteres. Por ejemplo, si quisiéramos filtrar todos los registros que en la clave de la especie tuvieran la sílaba *MEX*, podríamos utilizar la función `str_detect()`:

```
filter(dveg, str_detect(Clave, "MEX"))
## Sitio Indiv Clave DAP2004 DAP2010
```

```
## 1 H 3 HYBMEX 1.9 1.7
## 2 H 3 HYBMEX 2.6 2.0
## 3 H 3 HYBMEX NA 1.2
## 4 H 4 HYBMEX 2.3 2.2
## 5 H 8 HYBMEX 1.2 1.4
## 6 H 8 HYBMEX 2.0 1.4
## 7 H 8 HYBMEX NA 2.4
```

Otras veces queremos seleccionar aquellos registros que comienzan o terminan con cierta cadena de caracteres. Por ejemplo, para buscar todos los registros cuya *Clave* comienza por *CRO* o termina en *CON*:

```
filter(dveg, str_detect(Clave, “^CRO”))
filter(dveg, str_detect(Clave, “CON$”))
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 6 CROSUB 1.70 NA
## 2 T 6 CROSUB 1.90 NA
## 3 T 6 CROSUB 2.30 NA
## 4 T 11 CROSUB 1.91 NA
## 5 T 11 CROSUB 2.40 NA
## 6 T 27 CROSEP 4.20 NA
## 7 T 31 CROROX 1.40 NA
## Sitio Indiv Clave DAP2004 DAP2010
## 1 H 29 LONCON 3.6 4.2
## 2 H 29 LONCON 5.6 5.7
## 3 G 2 PIPCON 3.6 4.1
```

```
## 4 G 2 PIPCON 3.7 5.5
## 5 G 2 PIPCON 3.8 3.3
## 6 G 2 PIPCON 5.1 4.3
## 7 G 51 PIPCON 5.8 6.6
```

Existen muchas más funciones para evaluar coincidencias con una cadena de caracteres como parte de la librería `stringr`. Puedes descargar una ayuda para aprender más sobre cadenas de caracteres en el siguiente "link".[4]

Es posible emplear más de un criterio de selección de datos, por ejemplo:

```
filter(dveg, Indiv == 48, Clave == "SPOPUR")
## Sitio Indiv Clave DAP2004 DAP2010
## 1 G 48 SPOPUR 3.9 14.2
## 2 G 48 SPOPUR 10.0 4.0
```

En este caso, los registros seleccionados deben cumplir ambos criterios: número del individuo y especie. Es posible utilizar operadores booleanos para generar un espectro más amplio de combinaciones:

- &: "y"
- |: "o"
- !: "no"

Por ejemplo, para seleccionar los tallos de individuos cuyo número es *48* o que pertenecen a la especie *SPOPUR*:

[4] https://github.com/rstudio/cheatsheets/raw/master/strings.pdf

```
filter(dveg, Indiv == 48 | Clave == "SPOPUR")
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 19 SPOPUR 2.9 6.3
## 2 H 48 ACHGRA 3.0 2.5
## 3 H 48 ACHGRA NA 4.7
## 4 G 10 SPOPUR 13.6 15.8
## 5 G 48 SPOPUR 3.9 14.2
## 6 G 48 SPOPUR 10.0 4.0
```

Es posible extraer los registros que cumplen al menos uno de varios criterios respecto al mismo atributo:

```
filter(dveg, Indiv == 48 | Indiv == 49)
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 49 EXOCAR 9.3 12.0
## 2 H 48 ACHGRA 3.0 2.5
## 3 H 48 ACHGRA NA 4.7
## 4 H 49 THOPAU 2.7 4.8
## 5 H 49 THOPAU NA 1.0
## 6 H 49 THOPAU NA 1.7
## 7 H 49 THOPAU NA 2.0
## 8 G 48 SPOPUR 3.9 14.2
## 9 G 48 SPOPUR 10.0 4.0
filter(dveg, Indiv %in% c(48,49))
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 49 EXOCAR 9.3 12.0
```

```
## 2 H 48 ACHGRA 3.0 2.5
## 3 H 48 ACHGRA NA 4.7
## 4 H 49 THOPAU 2.7 4.8
## 5 H 49 THOPAU NA 1.0
## 6 H 49 THOPAU NA 1.7
## 7 H 49 THOPAU NA 2.0
## 8 G 48 SPOPUR 3.9 14.2
## 9 G 48 SPOPUR 10.0 4.0
```

Ambas expresiones producen el mismo resultado. En el segundo caso se usó el operador `%in%`, que permite identificar aquellos objetos de *Indiv* cuyo valor coincide con alguno de los valores en el vector c(48,49).

Ejercicios sobre `filter`

Usando la función *filter*, responde a las siguientes preguntas:

a) ¿Cuántos registros en el sitio T cuyos tallos en el 2004 tuvieron más de 5cm de diámetro hay en la tabla?
b) ¿Para cuántos registros el diámetro se incrementó entre el 2004 y el 2010?
c) Obtén el conjunto de los registros del sitio T que pertenecen a especies que no se encuentran en el sitio H.
d) ¿Cómo filtrar los registros de la especie *LONMUT* cuyo diámetro en el 2010 se encontraba entre 5.0 y 9.9 cm?

Podrás encontrar algunas propuestas de respuesta al final del capítulo.

2.3.4. Ordenación de registros con `arrange`

La función `arrange` sirve para modificar el orden de los registros. Por ejemplo, para ordenar la tabla por número de especie:

```
arrange(dveg, Clave)
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 35 ACHGRA 4.2 NA
## 2 T 64 ACHGRA 1.2 NA
## 3 T 64 ACHGRA 1.3 1.2
## 4 T 64 ACHGRA 1.3 NA
## 5 T 64 ACHGRA 1.4 NA
## 6 T 64 ACHGRA 2.8 NA
## 7 H 28 ACHGRA 2.3 3.0
```

Y para ordenar primero por especie y luego desempatar por diámetro del 2004:

```
arrange(dveg, Clave, DAP2004)
## Sitio Indiv Clave DAP2004 DAP2010
## 1 T 64 ACHGRA 1.2 NA
## 2 T 64 ACHGRA 1.3 1.2
## 3 T 64 ACHGRA 1.3 NA
## 4 T 64 ACHGRA 1.4 NA
## 5 H 39 ACHGRA 1.4 1.0
## 6 H 39 ACHGRA 1.7 1.2
## 7 H 28 ACHGRA 2.3 3.0
```

La función `arrange` ordena de manera ascendente. Se puede ordenar de manera descendente usando la función `desc` sobre la columna respecto a la cual se está ordenando:

```
arrange(dveg, desc(DAP2004))
## Sitio Indiv Clave DAP2004 DAP2010
## 1 G 15 MIMARE 29.6 34.0
## 2 T 63 CAEERI 24.5 24.1
## 3 T 63 CAEERI 24.3 23.9
## 4 H 33 HELPAL 24.3 NA
## 5 T 29 CHLMAN 22.1 22.0
## 6 G 15 MIMARE 20.7 24.4
## 7 H 19 CAEERI 19.5 20.3
```

2.3.5. Selección de atributos (columnas) empleando select

La función `select` puede emplearse para generar una tabla cuyas columnas sean un subconjunto de la tabla original. Por ejemplo, si no importara la información del sitio o del individuo se podría obtener una nueva tabla seleccionando las columnas de interés, así:

```
select(dveg, Clave, DAP2004, DAP2010)
## Clave DAP2004 DAP2010
## 1 APOPAN 4.3 6
## 2 AESAMO 3.4 NA
## 3 CROSUB 1.7 NA
## 4 CROSUB 1.9 NA
## 5 CROSUB 2.3 NA
```

```
## 6 RANTET 2.8 NA
## 7 RANTET 4.5 NA
```

Es posible utilizar algunos atajos para seleccionar múltiples columnas al mismo tiempo sin tener que escribir todos los nombres. Por ejemplo, podemos volver a seleccionar los atributos de interés utilizando el siguiente llamado de la función `select`:

```
select(dveg, Clave:DAP2010)
```

Los dos puntos (`:`) permiten seleccionar todas las columnas que se encuentran entre las columnas *Especie* y *FV*. Por el contrario, el siguiente llamado permite excluir dicha información:

```
select(dveg, -(Clave:DAP2010))
```

El signo menos (`-`) significa en este caso "quitar" las columnas especificadas.

Es posible combinar el uso de la función `select()` con otras funciones especializadas en la búsqueda de secuencias de caracteres dentro de los nombres de los atributos:

- `starts_with`("abc"): busca columnas cuyo nombre comienza con la secuencia "abc"
- `ends_with`("xyz"): busca atributos cuyo nombre termina con la secuencia "xyz"
- `contains`("ijk"): busca atributos cuyo nombre contiene la secuencia "ijk"
- `num_range`("x", 1:3): busca atributos cuyo nombre sea x1, x2 o x3
- `everything`(): busca todos los atributos.

Por ejemplo, para seleccionar simultáneamente las dos columnas de diámetro se puede usar:

```
select(dveg, starts_with("DAP"))
select(dveg, num_range("DAP", 2000:2020))
## DAP2004 DAP2010
## 1 4.3 6
## 2 3.4 NA
## 3 1.7 NA
## 4 1.9 NA
## 5 2.3 NA
## 6 2.8 NA
## 7 4.5 NA
## DAP2004 DAP2010
## 1 4.3 6
## 2 3.4 NA
## 3 1.7 NA
## 4 1.9 NA
## 5 2.3 NA
## 6 2.8 NA
## 7 4.5 NA
```

Finalmente, es posible seleccionar una columna específica para modificar su nombre utilizando la función `rename()`:

```
rename(dveg, sp = Clave)
## Sitio Indiv sp DAP2004 DAP2010
```

```
## 1 T 1 APOPAN 4.3 6
## 2 T 5 AESAMO 3.4 NA
## 3 T 6 CROSUB 1.7 NA
## 4 T 6 CROSUB 1.9 NA
## 5 T 6 CROSUB 2.3 NA
## 6 T 7 RANTET 2.8 NA
## 7 T 7 RANTET 4.5 NA
```

Ejercicios sobre `select()`

a) ¿Qué pasa cuando se incluye más de una vez el nombre de una misma columna durante el uso de la función *select*?
b) ¿Cómo podrías usar *select* para reordenar los atributos de la tabla *dveg* de tal forma que la primera columna que aparezca sea la de la clave de la especie?

Las respuestas las puedes encontrar al final del capítulo.

2.3.6. Generación de tablas agrupadas usando *group_by*

En algunas ocasiones deseamos que los procedimientos que hacemos sobre una tabla no se apliquen de manera uniforme para toda la tabla, sino que se apliquen por subconjuntos de la misma. Por ejemplo, es posible que queramos calcular una nueva variable con base en un promedio que cambia de grupo en grupo. En otros casos lo que queremos es obtener resúmenes, tales como sumas o promedios, pero no generales sino para diferentes subconjuntos de la tabla. La función `group_by` modifica una tabla de datos, agrupando internamente los datos con base en las columnas empleadas para hacer la agrupación, de tal forma que cualquier operación o cálculo que se haga posteriormente

sobre dicha tabla se hará por grupo. La modificación es "invisible", pues no hay ningún cambio aparente en la tabla:

```
dveg_agrup <- group_by(dveg, Sitio)
View(dveg_agrup)
## # A tibble: 7 x 5
## # Groups: Sitio [1]
## Sitio Indiv Clave DAP2004 DAP2010
## <fct> <fct> <fct> <dbl> <dbl>
## 1 T 1 APOPAN 4.3 6
## 2 T 5 AESAMO 3.4 NA
## 3 T 6 CROSUB 1.7 NA
## 4 T 6 CROSUB 1.9 NA
## 5 T 6 CROSUB 2.3 NA
## 6 T 7 RANTET 2.8 NA
## 7 T 7 RANTET 4.5 NA
```

En este caso creamos una nueva tabla llamada *dveg_agrup*, que contiene los mismos datos que la tabla *dveg*, pero agrupados por categorías de la variable *Sitio*. El cambio es evidente cuando preguntamos qué clase de objeto es *dveg_agrup*:

```
class(dveg_agrup)
## [1] "grouped_df" "tbl_df" "tbl" "data.frame"
```

La nueva tabla es un objeto de R de tipo "grouped_df" (*grouped data frame*). El potencial de la función `group_by` es evidente cuando se usa conjuntamente con funciones como `mutate`y `summarise`,

como se verá a continuación. Finalmente, nótese que existe también la función `ungroup()`, para desagrupar una tabla agrupada.

2.3.7. Adición de nuevas columnas con mutate

Una práctica muy común durante el trabajo con tablas de datos es la inclusión de nuevas columnas, por ejemplo, cuando hacemos cálculos con base en las columnas ya existentes, o cuando queremos fusionar dos columnas de texto. Para ello podemos usar las funciones `mutate()` o `transmute()`, dependiendo de si lo que queremos es adicionar la columna creada a la tabla existente o de si queremos generar una nueva tabla que incluya solo las nueva(s) columna(s). Por ejemplo, si queremos generar una nueva columna en la tabla *dveg* que se llame *SI* y que fusione la información de las columnas "Sitio" e "Indiv":

```
mutate(dveg, SI=paste(Sitio,Indiv))
transmute(dveg, SI=paste(Sitio,Indiv))
## Sitio Indiv Clave DAP2004 DAP2010 SI
## 1 T 1 APOPAN 4.3 6 T 1
## 2 T 5 AESAMO 3.4 NA T 5
## 3 T 6 CROSUB 1.7 NA T 6
## 4 T 6 CROSUB 1.9 NA T 6
## 5 T 6 CROSUB 2.3 NA T 6
## 6 T 7 RANTET 2.8 NA T 7
## 7 T 7 RANTET 4.5 NA T 7
## SI
## 1 T 1
## 2 T 5
## 3 T 6
```

```
## 4 T 6
## 5 T 6
## 6 T 7
## 7 T 7
```

`mutate()` agregó la columna *SI* al final de la tabla *dveg*, mientras que `transmute()` generó una nueva tabla cuya única columna es *SI*. En el uso de ambas funciones es posible generar más de una variable a la vez, como lo muestra el siguiente ejemplo:

```
transmute(dveg,
 SI = paste(Sitio,Indiv),
 crecim = DAP2010-DAP2004)
## SI crecim
## 1 T 1 1.7
## 2 T 5 NA
## 3 T 6 NA
## 4 T 6 NA
## 5 T 6 NA
## 6 T 7 NA
## 7 T 7 NA
```

En este caso se generó una nueva tabla con dos columnas con un solo llamado a la función `transmute`.

Algunas veces es necesario definir atributos que requieren cálculos un poco más complejos. Por ejemplo, uno podría querer saber, para

cada individuo, qué porcentaje de su área basal es aportado por cada tallo o tronco. El **área basal** es una medida indirecta de la cantidad de madera (técnicamente, de biomasa) que contiene un tallo. La suma de todas las áreas basales de los troncos de un individuo da una idea de la cantidad total de madera contenida en el mismo. Para lograr incluir una nueva columna que contenga la información del aporte relativo de cada tallo al área basal del individuo al que pertenece, requerimos primero calcular el área basal de cada tallo y luego ver qué porcentaje representa ese valor especto al área basal total del individuo. Veamos cómo podemos hacer eso de manera muy sencilla combinando las capacidades de `group_by` y `mutate`:

```
dveg_agrup <- group_by(dveg, Sitio, Indiv,)
mutate(dveg_agrup,
 AB2004 = pi*(DAP2004^2)/4,
 Pc_AB2004 = 100*AB2004/sum(AB2004))
## # A tibble: 7 x 7
## # Groups: Sitio, Indiv [4]
## Sitio Indiv Clave DAP2004 DAP2010 AB2004 Pc_
AB2004
## <fct> <fct> <fct> <dbl> <dbl> <dbl> <dbl>
## 1 T 1 APOPAN 4.3 6 14.5 100.
## 2 T 5 AESAMO 3.4 NA 9.08 100
## 3 T 6 CROSUB 1.7 NA 2.27 24.5
## 4 T 6 CROSUB 1.9 NA 2.84 30.6
## 5 T 6 CROSUB 2.3 NA 4.15 44.9
## 6 T 7 RANTET 2.8 NA 6.16 27.9
## 7 T 7 RANTET 4.5 NA 15.9 72.1
```

El procedimiento está dividido en dos pasos. En el primero, generamos una versión de la tabla de datos `dveg` pero agrupada por las variables *Sitio* e *Indiv*. esto quiere decir que en esta tabla cada árbol constituye un grupo diferente. Cualquier operación que se realice sobre esta tabla, se hará de manera independiente para cada uno de los árboles. En el segundo paso añadimos dos columnas a la tabla de datos. La primera corresponde al área basal de cada tallo, utilizando para ello la fórmula del área de un círculo. La segunda columna corresponde al porcentaje que dicha área basal aporta al área basal total del individuo. Puesto que la tabla está agrupada por árbol, cuando se calcula la suma del área basal con la función `sum`, esa suma solo incluye a las áreas basales del mismo árbol. Para comprobar que el resultado es correcto, revisa la tabla generada y suma los valores de *Pc_AB2004* para un mismo árbol. Para todos los árboles, esa suma es 100.

Ejercicios sobre `mutate()`

a) Genera una nueva tabla que contenga solo los registros correspondientes a tallos que fueron medidos en ambos años y que además mostraron un crecimiento positivo durante ese periodo.
b) Genera una tabla que contenga los registros de los 3 diámetros más grandes en el año 2004 para cada sitio (pista: prueba las funciones *rank* y *desc*).

2.3.8. Resúmenes de tablas con summarise

Un de los procedimientos más comunes cuando estamos trabajando con tablas es la elaboración de resúmenes que nos permitan obtener estadísticas de las variables y por ende generar información con base en los datos contenidos en la tabla. Con `dplyr` es posible resumir tablas empleando la función `summarise`. Por ejemplo, podemos calcular los diámetros promedio para ambos años, o calcular el número de especies diferentes que aparecen en la tabla:

```
summarise(dveg,
 prom2004 = mean(DAP2004, na.rm = TRUE),
 prom2010 = mean(DAP2010, na.rm = TRUE),
 Num_sp = n_distinct(Clave))
## prom2004 prom2010 Num_sp
## 1 3.683652 3.781063 58
```

Sin embargo, el verdadero potencial de `summarise()` aparece cuando se combina con la función `group_by`. Por ejemplo, si aplicamos `summarise()`a los datos agregados por especie:

```
summarise(group_by(dveg, Sitio),
 DAP2004 = mean(DAP2004, na.rm = TRUE),
 prom2010 = mean(DAP2010, na.rm = TRUE),
 Num_sp = n_distinct(Clave))
## # A tibble: 3 x 4
## Sitio DAP2004 prom2010 Num_sp
## <fct> <dbl> <dbl> <int>
## 1 G 3.75 4.47 23
## 2 H 3.96 3.31 26
## 3 T 3.42 3.72 29
```

Nótese cómo ahora el resumen de las variables se calcula por Sitio, esto debido a que la tabla de datos fue previamente agregada usando dicha variable. Por supuesto, es posible agregar por más de una variable:

```
summarise(group_by(dveg, Sitio, Clave),
 DAP2004 = mean(DAP2004, na.rm = TRUE),
 prom2010 = mean(DAP2010, na.rm = TRUE))
## # A tibble: 78 x 4
## # Groups: Sitio [3]
## Sitio Clave DAP2004 prom2010
## <fct> <fct> <dbl> <dbl>
## 1 G APOPAN 3.8 4.6
## 2 G BUNPAL 5.1 3.6
## 3 G CAEERI 2.6 2.9
## 4 G "CAEERI " 1.9 NaN
## 5 G CASNIT 2.17 2.5
## 6 G CORALL 8.18 10.4
## 7 G CRONIf 1.8 1.5
## 8 G CRONIV 1.72 2.03
## 9 G CROROX 1.2 1.6
## 10 G DIOAEQ 1.53 1.4
## # ... with 68 more rows
```

En cuyo caso el resumen se hace para todas las combinaciones posibles de los niveles de las variables usadas para agrupar, en este caso *Sitio* y *Clave*. El resultado es presentado siguiendo el orden en el que se incluyeron las variables agrupadas.

En los comandos previos usamos el argumento `na.rm = TRUE` dentro de la función `mean`. El objetivo de dicho argumento es excluir del cálculo a las celdas que no contienen información (*NA*), puesto que su inclusión siempre conlleva a que el resultado sea un *NA*. Este argumento

puede ser incluido en todas las funciones para el cálculo de estadísticas. También es posible excluir previamente los *NA* empleando un filtro y la función `is.na`:

```
summarise(group_by(filter(dveg,!is.na(DAP2004)),
Sitio, Clave), DAP2004 = mean(DAP2004))
## # A tibble: 68 x 3
## # Groups: Sitio [3]
## Sitio Clave DAP2004
## <fct> <fct> <dbl>
## 1 G APOPAN 3.8
## 2 G BUNPAL 5.1
## 3 G CAEERI 2.6
## 4 G "CAEERI " 1.9
## 5 G CASNIT 2.17
## 6 G CORALL 8.18
## 7 G CRONIf 1.8
## 8 G CRONIV 1.72
## 9 G CROROX 1.2
## 10 G DIOAEQ 1.53
## # ... with 58 more rows
```

Existen múltiples funciones que son muy útiles para resumir tablas:

- Aritméticas
  - `sum`: suma
  - `cumsum`: suma acumulativa
  - `prod`: producto
- Medidas de tendencia central
  - `mean`: media

  - `median`: mediana
- Medidas de dispersión
  - `sd`: desviación estándar
  - `var`: varianza
  - `IQR`: rango inter-cuartil
- Medidas de ranking
  - `min`: mínimo
  - `max`: máximo
  - `quantile`: cuantiles
- Medidas de posición
  - `first`: ubica el primer registro
  - `nth(, i)`: ubica el *i*ésimo registro
  - -`last()`: ubica el último registro
- Medidas de conteo
  - `n()`: cuenta el número de registros
  - `n_distinct()`: cuenta el número de registros diferentes (niveles)
  - `counts()`: genera un resumen únicamente de conteos

Ejercicios sobre `summarise`

a) Encuentra los árboles que tienen al menos 5 tallos con medida de DAP para el año 2010.
b) Cuenta el número de registros que tienen un diámetro mayor o igual a 5cm en 2004.

Encontrarás algunas opciones de respuesta al final del capítulo.

2.4. Manejo de tablas: depurado y ajuste de tablas

Los procedimientos abordados en la sección anterior constituyen herramientas que nos permiten modificar el contenido de las tablas (cuántos y qué datos hay) o generar nuevas tablas que resumen la tabla original. Sin embargo, hasta ahora hemos supuesto que el formato de la tabla de datos es el adecuado y/o que la tabla de datos no contiene errores. Pero en la práctica cotidiana nuestras tablas tienen formatos que no se ajustan al análisis que se pretende hacer, o al menos lo hace

más complicado. Adicionalmente, muchas de las tablas con las que trabajamos poseen errores, algunos de los cuales son detectables, mientras que otros no.

Al proceso de ajustar una tabla de datos eliminando la mayor cantidad de errores posibles y re-arreglándola en un formato idóneo para el análisis posterior, se le conoce como **depurado y ajuste de tablas** (data wrangling en inglés). Desde la perspectiva de Hadley Wickham (que es la perspectiva de `tidyverse`), el proceso de depurar y ajustar tablas resulta en un conjunto de datos ordenados (*tidy data* en inglés). *Tidy* es entonces un formato particular para la organización de los datos (puedes encontrar una descripción más detallada sobre este formato en el siguiente link).[5] Modificar una tabla cruda en una tabla ordenada implica asumir los principios básicos que se describieron en la sección 2.2, los cuales resumimos a continuación.

Existen múltiples formatos (literalmente) de guardar el mismo conjunto de datos. El formato *tidy* propone las siguientes reglas:

- Cada columna es una variable.
- Cada observación es una fila.
- Cada valor debe ir en su propia celda.

Una serie de pasos básicos para convertir una tabla de datos cruda en una ordenada es:

1. Identificar en la tabla cruda cuáles son las variables y cuáles son las observaciones.
2. Evaluar si en la tabla cruda se presenta uno de los dos problemas descritos a continuación: –Una misma variable está ubicada en diferentes columnas. –Una misma observación está ubicada en múltiples filas.

[5] http://vita.had.co.nz/papers/tidy-data.html

Para solucionar estos problemas podemos hacer uso de dos funciones en *tidyverse*: *gather* y *spread*

2.4.1. Apilado de datos empleando gather

Un problema muy común en tablas crudas es que múltiples columnas contienen una misma variable, de tal forma que cada columna muestra los valores de dicha variable para cierto valor de otra variable. En nuestro caso de estudio, las columnas *DAP2004* y *DAP2010* contienen información de la misma variable (DAP), pero cada una para diferentes valores de otra variable (año). Como consecuencia, cada registro tiene observaciones de DAP en columnas diferentes, violando el primer criterio de una tabla *ordenada*. En estos casos lo ideal es colocar la información correspondiente a la variable DAP en una sola columna y tener otra columna que identifique el valor de la segunda variable, es decir, el año. Es posible realizar esto de una manera sencilla usando la función *gather()*, que apila la información de dos o más variables en una sola columna:

```
dveg.flaca <- gather(data = dveg, value = "DAP", key
= "fecha", DAP2004, DAP2010)
head(dveg.flaca)
## Sitio Indiv Clave fecha DAP
## 1 T 1 APOPAN DAP2004 4.3
## 2 T 5 AESAMO DAP2004 3.4
## 3 T 6 CROSUB DAP2004 1.7
## 4 T 6 CROSUB DAP2004 1.9
## 5 T 6 CROSUB DAP2004 2.3
## 6 T 7 RANTET DAP2004 2.8
```

La función *gather()* requiere cuatro argumentos básicos. El primero es *data*, que define la tabla que se quiere transformar. El segundo se llama *value* y define el nombre de la variable donde se guardarán los datos que antes se encontraban en columnas separadas y que ahora se apilarán en una sola columna. El tercero es *key*, que define el nombre que tomará la nueva columna donde se almacenerá la información relativa al criterio que separaba las columnas que ahora serán apiladas. El último argumento es una lista de variables que se quiere apilar, las cuales pueden ser llamadas directamente por su nombre. El resultado del llamado a la función es una tabla que ya no contiene las variables que se *apilaron* y que contiene a cambio las nuevas variables definidas por *key* y *value*, así como todas las columnas restantes.

En el caso particular de nuestro ejemplo, las variables apiladas fueron *DAP2004 DAP2010*, las cuales ya no aparecen en la tabla generada. A cambio, aparece una variable llamada *DAP* donde se apilaron los datos de diámetro, así como una variable año, que indica la columna de la que proviene cada dato de *DAP*. Nótese que los valores que toma esta variable son los nombres de las columnas originales. Adicionalmente se mantienen las columnas restantes: *Sitio*, *Indiv* y *Clave*. Finalmente, es muy posible que quisiéramos que el año apareciera como una variable numérica y no como un caracter. Podemos usar la función *mutate()* para transformarla:

```
dveg.flaca <- mutate(dveg.flaca, fecha = parse_
integer(str_extract(fecha,"\\d{1,4}")))

head(dveg.flaca)
## Sitio Indiv Clave fecha DAP
## 1 T 1 APOPAN 2004 4.3
```

```
## 2 T 5 AESAMO 2004 3.4
## 3 T 6 CROSUB 2004 1.7
## 4 T 6 CROSUB 2004 1.9
## 5 T 6 CROSUB 2004 2.3
## 6 T 7 RANTET 2004 2.8
```

La tabla resultante es una tabla en formato ordenado: cada columna es una variable y cada registro contiene información de la unidad observacional, el tallo.

2.4.2. Extensión de tablas empleando spread

La función `spread` realiza el procedimiento contrario, genera una tabla *extendida* a partir de una columna `key` de la que se originan los nombres de las nuevas columnas, y una columna `value` cuyos valores serán reubicados en las nuevas columnas. En biología y ecología, este formato es muy común para la realización de ciertos análisis. Por ejemplo, para el análisis de comunidades biológicas comúnmente se parte de una matriz de sitios x especies, esto es, una tabla en la que cada registro corresponde a un sitio y cada columna a una especie. Los datos que se reportan en las celdas de la tabla son las abundancias, biomasa u otra medida que hable de la presencia de las especies. Otras tablas que comúmente usan este formato son las matrices de especies x genes.

Veamos entonces cómo generar una matriz de sitios x especies en nuestro estudio de caso. La matriz incluirá datos sobre la abundancia de cada especie en cada sitio. Para ello, necesitamos calcular primero la abundancia de cada especie en cada sitio. Lo haremos sin diferenciar los años.

```
dveg.flaca.agru <- group_by(dveg.flaca, Sitio, Clave)
abund <- summarise(dveg.flaca.agru,
 n = n_distinct(Indiv))
View(abund)
## # A tibble: 10 x 3
## # Groups: Sitio [1]
## Sitio Clave n
## <fct> <fct> <int>
## 1 G APOPAN 1
## 2 G BUNPAL 2
## 3 G CAEERI 1
## 4 G "CAEERI " 1
## 5 G CASNIT 2
## 6 G CORALL 4
## 7 G CRONIf 1
## 8 G CRONIV 9
## 9 G CROROX 1
## 10 G DIOAEQ 2
```

La función `n_distinct` está contando el número de etiquetas de *Indiv* distintas. Puesto que la tabla está agrupada por *Sitio* y *Clave*, el resultado de dicho conteo es el número de individuos por especie y sitio. Con esta tabla ya podemos proceder a obtener la matriz de sitios x especies empelando la función `spread`:

```
sitxesp <- spread(abund, key = Clave, value = n,
fill = 0)
```

```
View(sitxesp)
## # A tibble: 3 x 6
## # Groups: Sitio [3]
## Sitio ACHGRA AESAMO aff.MALsp.1 AMPADS APOPAN
## <fct> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 G 0 0 0 0 1
## 2 H 4 0 1 0 9
## 3 T 2 1 0 2 16
```

El resultado es una tabla en la que cada sitio aparece en un registro diferente y cada especie en una columna diferente. Los datos contenidos son las abundancias de las especies en cada sitio. El argumento `fill` rellena las casillas vacías con ceros.

Ejercicios sobre `gather` *y* `spread`

a) Intenta revertir el apilado de la tabla *dveg* empleando la función `spread`. ¿Qué ocurre? ¿Cómo podrías solucionar esta situación? (Ayuda: considera incluir un indicador de cada tallo desde el principio).

2.5. Pegado de tablas con `_join`

Como mencionamos desde el principio de este capítulo, un criterio importante de construcción de tablas ordenadas es que cada tipo de unidad observacional origina una tabla. En el caso de nuestro ejercicio, puede haber datos adicionales sobre los sitios, como la ubicación, las coordenadas, el tipo de suelo, la inclinación o la orientación de la ladera. También puede haber datos adicionales sobre las especies: el nombre científico completo, la familia taxonómica a la que pertenece, la forma de vida, su altura o diámetro máximo, etc. Es importante que

esta información vaya en tablas aparte, pues ello reduce el nivel de error en los registros (asociados por ejemplo a errores de dedo al pasar los datos), reduce el espacio requerido para almacenar la información (critero muy importante cuando se tienen millones de datos) y facilita el manejo de la información.

Sin embargo, cuando uno quiere hacer uso de datos contenidos en más de una tabla es necesario juntar la información de las mismas. En el contexto del manejo de datos, hacer un ***join*** significa literalmente *juntar o ligar tablas*. Ese es el papel de una serie de funciones contenidas en `dplyr` y genéricamente llamadas `join`. Estas funciones toman la información contenida en un par de tablas y la juntan siguiendo criterios particulares. El punto fundamental a entender para ligar tablas es que se requiere identificar en el par de tablas al menos una variable que servirá como criterio para pegar los datos, es decir, para saber qué registros de una tabla se deben empatar con los registros de la otra tabla. A esa(s) variable(s) se le(s) denomina *llave*(s) (*key*).

Veamos con un par de ejemplos sencillos el uso de las funciones `join`. Para ello, requerimos cargar una tercera tabla llamada Datos taxonomia.csv (pica en el nombre del archivo para descargar una carpeta zip que lo contiene. También puedes consultar la dirección electrónica del Github en la sección "Antes de empezar" de este manual). Esta tabla contiene datos taxonómicos de las especies: a cada *Clave* viene asociado el nombre de la *Familia*, el *género*, la *especie* y el *autor* del nombre.

```
taxon <- read.csv("Datos taxonomia.csv", header=T,
sep=",")
View(taxon)
```

```
## Family Genus Specific.epithet
## 1 Leguminosae Acacia farnesiana
## 2 Leguminosae Acacia macracantha
## 3 Leguminosae Acaciella angustissima
## 4 Achatocarpaceae Achatocarpus gracilis
## 5 Leguminosae Aeschynomene amorphoides
## 6 Leguminosae Albizia occidentalis
## 7 Leguminosae Albizia tomentosa
## species.author Clave
## 1 (L.) Willd. ACAFAR
## 2 Humb. & Bonpl. ex Willd. ACAMAC
## 3 (Mill.) Britton & Rose ACAANG
## 4 H. Walter ACHGRA
## 5 (S. Watson) Rose ex B.L. Rob. AESAMO
## 6 Brandegee ALBOCC
## 7 (Micheli) Standl. ALBTOM
```

Si quisiéramos adicionar los datos de la tabla de taxonomía a la tabla de datos de tallos podríamos usar la función `left_join`:

```
final1 <- left_join(dveg, taxon, key = "Clave")
View(final1)
## Sitio Indiv Clave DAP2004 DAP2010 Family Genus
## 1 T 1 APOPAN 4.3 6 Leguminosae Apoplanesia
## 2 T 5 AESAMO 3.4 NA Leguminosae Aeschynomene
## 3 T 6 CROSUB 1.7 NA Euphorbiaceae Croton
```

```
## 4 T 6 CROSUB 1.9 NA Euphorbiaceae Croton
## 5 T 6 CROSUB 2.3 NA Euphorbiaceae Croton
## 6 T 7 RANTET 2.8 NA <NA> <NA>
## 7 T 7 RANTET 4.5 NA <NA> <NA>
## Specific.epithet species.author
## 1 paniculata C. Presl
## 2 amorphoides (S. Watson) Rose ex B.L. Rob.
## 3 suberosus Kunth
## 4 suberosus Kunth
## 5 suberosus Kunth
## 6 <NA> <NA>
## 7 <NA> <NA>
```

Revisa con cuidado la tabla resultante. ¿Qué observas? La nueva tabla contiene la información de la tabla *dveg*, pero la complementó con los datos taxonómicos. ¿Cómo supo qué datos taxonómicos agregarle a cada tallo? Para ello empleo la *llave*, que en este caso era la variable *Clave*. Dependiendo del valor de *Clave* que tenía cada tallo en la tabla *dveg*, colocó la información taxonómica correspondiente presente en la tabla *taxon*. Sin embargo, no todos los tallos tienen datos taxonómicos. Esto se debe a que no existe su valor de *Clave* en la tabla de taxonomía, por lo que no les pudo asignar datos taxonómicos. Nota como la tabla final tiene el mismo número de filas que la tabla de tallos: se conservaron en la tabla final todos los tallos, independientemente de que se les pudiera asignar o no información taxonómica.

Veamos qué pasa si cambiamos el orden de las tablas en el llamado a la función:

```
final2 <- left_join(taxon, dveg, key = "Clave")
View(final2)
## Family Genus Specific.epithet species.author
## 1 Leguminosae Acacia farnesiana (L.) Willd.
## 2 Leguminosae Acacia macracantha Humb. & Bonpl. ex Willd.
## 3 Leguminosae Acaciella angustissima (Mill.) Britton & Rose
## 4 Achatocarpaceae Achatocarpus gracilis H. Walter
## 5 Achatocarpaceae Achatocarpus gracilis H. Walter
## 6 Achatocarpaceae Achatocarpus gracilis H. Walter
## 7 Achatocarpaceae Achatocarpus gracilis H. Walter
## Clave Sitio Indiv DAP2004 DAP2010
## 1 ACAFAR <NA> <NA> NA NA
## 2 ACAMAC <NA> <NA> NA NA
## 3 ACAANG <NA> <NA> NA NA
## 4 ACHGRA T 35 4.2 NA
## 5 ACHGRA T 64 1.2 NA
## 6 ACHGRA T 64 1.3 1.2
## 7 ACHGRA T 64 1.3 NA
```

Hay tres diferencias importantes respecto al resultado contenido en *final1*. En *final2* aparecen primero las columnas de la tabla *taxon* y luego se adicionan las de la tabla *dveg*. Esto se debe a que en el llamado de la función `left_join` el primer argumento corresponde a la tabla de la izquierda (*left*), la cual es complementada con los datos contenidos en la segunda tabla. Entonces a cada fila de la tabla *taxon*

se le complementó con la información contenida en *dveg*. Puesto que a cada especie en *taxon* le correspondía más de un tallo de *dveg*, la información de taxonomía se repitió tantas veces como fue necesario para albergar todos los tallos de *dveg*. La segunda diferencia importante es que la tabla *final2* tiene más registros que la tabla *dveg*. Esto se debe a que en la tabla de taxonomía hay más especies de las que hay en la tabla de tallos. Sin importar que algunas especies no fueron realmente muestreadas, `left_join` las incluye en la tabla final. La tercera diferencia es que los tallos contenidos en *dveg* a los cuales no se les pudo asignar su información taxonómica (porque dicha *Clave* no existía en *taxon*) fueron excluidos de *final2*. Este comportamiento de la función debe ser tenido en cuenta para asegurar que toda la información de interés está incluida en la tabla final.

En resumen, en la función `left_join` la primera tabla (o tabla de la izquierda) es la que manda y la de la derecha es la que complementa. Esto quiere decir que la tabla resultante tendrá todos los registros de la tabla de la izquierda, a los cuales se les adiciona sus datos correspondientes contenidos en la tabla de la derecha. La función contraria `right_join` procede de manera exactamente contraria: la tabla de la derecha (segundo argumento) es la que manda y sobre ella se adicionan los datos contenidos en la primera tabla (primer argumento).

2.6. Ejercicio final

Al principio del capítulo te presentamos un ejercicio donde te cuestionábamos acerca de cómo vaciar una serie de datos de campo en una tabla (sección 2.1.). Intenta contestar nuevamente las preguntas de dicha actividad, aprovechando los elementos conceptuales y prácticos abordados en este capítulo. Compara tus nuevas respuestas

con las que guardaste previamente y con base en ello evalúa tu grado de evolución con relación al manejo de tablas.

2.7. Consideraciones finales

Esperamos que este capítulo induzca una reflexión crítica acerca de aspectos relacionados con el manejo de datos, tales como el diseño de los formatos de toma de datos, el de las tablas para vaciar la información, y el de las tablas que se requieren para proceder con el análisis de datos. No existe una solución única al manejo de datos, pues las características específicas de cada contexto determinarán las estructuras de las tablas, su transformación y su fusión. Sin embargo, el seguimiento de los criterios para el manejo de tablas de datos presentados en este capítulo y su implementación a través de las funciones y procedimientos básicos ilustrados con seguridad contribuirá a hacer el proceso de manejo de datos más expedito y sencillo.

2.8. Respuestas a los ejercicios

Ejercicios sobre `filter`

```
# a:
nrow(filter(dveg, Sitio == “T”, DAP2004 > 5))

# b:
nrow(filter(dveg, DAP2004 > DAP 2010))

# c:
sitioH <- droplevels(filter(dveg, Sitio == “H”))
```

```
especiesH <- levels(sitioH$Clave)
TsinHs <- filter(dveg, Sitio == "T", !(Clave %in% especiesH))


# d:
filter(dveg, str_detect(Clave, "LONMUT"), DAP2004 > 5, DAP2004 <= 9.9)
```

Ejercicios sobre `select`

```
# a.
La función solo la reconoce la primera vez que fue llamada la columna.

#b.
select(dveg, Clave, everything())
```

Ejercicios sobre `mutate`

```
# a.
dveg %>%
 drop_na(DAP2004,DAP2010) %>%
 mutate(crecim = DAP2010-DAP2004) %>%
 filter(crecim > 0)

Nótese en esta solución el uso de dos comandos adicionales: `drop_na`, que es una forma especial de `filter` para excluir de la tabla a todos los registros `NA` que aparecen en columnas específicas, y el **operador de tubería** (**pipe**), que permite
```

escribir código de manera secuencial (cuando el objeto generado en una línea de código es sobre el que operará la siguiente línea).

```
# b.
dveg %>%
group_by(Sitio) %>%
mutate(ranking = rank(desc(DAP2004))) %>%
filter(ranking <= 3)
```

Ejercicios sobre `group_by` *y* `summarise`

```
# a.
dveg %>%
 drop_na(DAP2010) %>%
 group_by(Sitio, Indiv) %>%
 summarise(n_tallos = n()) %>%
 filter(n_tallos >= 5)
```

```
# b.
dveg %>%
 filter(DAP2004 >= 5) %>%
 summarise(tallos_mas5 = n())
```

Ejercicios sobre `gather` *y* `spread`

```
# a.
spread(dveg.flaca, key = fecha, value = DAP)
```

El procedimiento no funciona porque en la tabla *dveg.flaca* no es posible identificar a qué tallo corresponden los registros de un mismo individuo, de tal forma que al querer asignar en columnas por año los valores de DAP, no sabe qué valor de 2004 empatar con qué valor del 2010. Esto nos permite identificar un problema en la tabla: los tallos no tienen un identificador único, solo los individuos. Es posible completar el ejercicio si desde el incio se agrega una columna que sirva como identificador de cada tallo:

```
dveg <- dveg %>%
 mutate(tallo = seq(1:n())) %>%
 select(Sitio:Clave, tallo, everything())
```

Ahora sí es posible proceder a generar la tabla en versión larga y posteriormente devolverla a la versión ancha:

```
dveg.flaca <- gather(data = dveg, value = “DAP”, key = “fecha”, DAP2004, DAP2010)
spread(dveg.flaca, key = fecha, value = DAP)
```

PARTE II. MÉTODOS DE ANÁLISIS

Capítulo 3. Introducción al Análisis de Redes de Interacciones Ecológicas entre Especies

Francisco Mora
Instituto de Investigaciones en Ecosistemas y Sustentabilidad
Universidad Nacional Autónoma de México

Wesley Dattilo
Red de Ecotecnología
Instituto de Ecología A.C.

Introducción

La palabra **Red** se ha convertido en un término de uso común, con seguridad asociado al avance de las tecnologías de transmisión de la información. Muchos hemos participado en lo que se denomina hoy en día como *redes sociales*, plataformas virtuales a través de las cuales compartimos información y nos relacionamos con otras personas. La mayoría de nosotros tenemos teléfonos celulares que usamos para comunicarnos con otros a través de una red de telefonía. O simplemente, muchos hemos accedido a la *red* (internet) para buscar información, hacer trámites o entretenernos con algo. Sin embargo, el concepto de red no es exclusivo a cuestiones tecnológicas. Por el contrario, muchos de los fenómenos que ocurren a nuestro alrededor funcionan o pueden ser entendidos como redes: la red de carreteras, la red de tráfico aéreo, la red de transmisión eléctrica, las relaciones sociales, las redes de apoyo ante desastres naturales… Podríamos afirmar sin lugar a equivocarnos que existen redes en cualquier lugar alrededor nosotros.

El estudio de los fenómenos biológicos no ha sido ajeno a esta visión: las redes neuronales, las redes de transmisión de enfermedades, las redes de regulación génica o las redes tróficas, son tan solo algunos ejemplos de cómo el concepto de red ha permeado en las últimas décadas el análisis de los sistemas biológicos (Luke 2015).[1] En ecología en particular el ejemplo más común (y tal vez más antiguo) es el estudio de las redes tróficas: ¿quién se come a quién? El análisis de redes tróficas permite entender los flujos de materia y energía entre diferentes especies. Más aún, dependiendo de su rol en la red, las especies han sido agrupadas bajo diferentes nombres que definen el "nivel trófico"" o "gremio" al que pertenecen: productores primarios, productores secundarios, predadores, polinizadores, herbívoros, etc.

Si todos estos fenómenos tecnológicos, sociales y biológicos siempre han existido, ¿a qué se debe el auge en su conceptualización y estudio desde la perspectiva del análisis de redes? Existen principalmente dos razones básicas. La primera tiene que ver con el potencial que tiene la perspectiva de redes para el análisis de sistemas complejos. Es decir, aquellos sistemas que están compuestos por varias partes interconectadas donde surgen propiedades emergentes que no pueden explicarse a partir de las propiedades de los elementos aislados, como por ejemplo las relaciones entre las personas, las rutas de los autobuses, redes metabólicas, o las conexiones sinápticas de las neuronas, entre otros.

La segunda razón la constituye el incremento en la capacidad de procesamiento de información durante las últimas décadas. Los análisis que anteriormente tardaban días, ahora pueden ser fácilmente procesados en algunas pocas horas. Existen en la actualidad múltiples opciones de *software* especializado para el análisis de redes en general

[1] Luke, D. A. 2015. A User's Guide to Network Analysis in R. Springer.

y ecológicas en particular (Antoniazzi et al. 2018).[2] Además, una porción importante de estas herramientas son de acceso libre, y muy posiblemente mucho más amigables para un auditorio no especializado en programación, lo que ha ampliado de manera significativa el público y por ende la aplicación del análisis de redes.

Dada la preponderancia que ha tomado la perspectiva de redes para el análisis de procesos en general y ecológicos en particular, es necesario reconocer los conceptos básicos asociados al análisis de redes, así como tener la capacidad de manejar las herramientas para su análisis. El objetivo del presente capítulo es introducir al lector en los conceptos y métodos más básicos relacionados con el análisis de redes aplicado a sistemas ecológicos a través del uso de una herramienta de libre acceso como lo es el lenguaje R. Una vez abordados los contenidos básicos de esta guía, se sugiere al lector interesado que profundice tanto en la teoría en torno al análisis de redes ecológicas (Proulx et al. 2005,[3] Silk et al. 2017a,[4] Dáttilo y Rico-Gray 2018),[5] como en métodos y herramientas avanzados (Luke 2015, Mello et al. 2016,[6] Silk et al. 2017b).[7]

En el presente capítulo presentaremos una serie de herramientas útiles para el análisis de redes ecológicas. La presentación de dichas

[2] Antoniazzi, R., W. Dáttilo, y V. Rico-Gray. 2018. A Useful Guide of Main Indices and Software Used for Ecological Networks Studies BT - Ecological Networks in the Tropics: An Integrative Overview of Species Interactions from Some of the Most Species-Rich Habitats on Earth. Páginas 185–196 *en* W. Dáttilo y V. Rico-Gray, eds. Springer International Publishing, Cham.

[3] Proulx, S. R., D. E. L. Promislow, y P. C. Phillips. 2005. Network thinking in ecology and evolution. Trends in Ecology and Evolution 20:345-353.

[4] Silk, M. J., D. P. Croft, R. J. Delahay, D. J. Hodgson, M. Boots, N. Weber, y R. A. McDonald. 2017a. Using Social Network Measures in Wildlife Disease Ecology, Epidemiology, and Management. BioScience 67:245-257.

[5] Dáttilo, W., y V. Rico-Gray. 2018. Ecological Networks in the Tropics.

[6] Mello, M. A. R. de, R. de L. Muylaert, R. B. P. Pinheiro, y G. M. F. Ferreira. 2016. Guia para Análise de Redes Ecológicas. Páginas 1-111. Edição dos autores, Belo Horizonte.

[7] Silk, M. J., D. P. Croft, R. J. Delahay, D. J. Hodgson, N. Weber, M. Boots, and R. A. McDonald. 2017b. The application of statistical network models in disease research. Methods in Ecology and Evolution 8:1026-1041.

herramientas se hará a través del análisis de datos derivados de una red de interacción entre plantas y polinizadores, que incluye datos sobre el número de visitas de las diferentes especies de polinizadores a las diferentes especies de plantas.

A través de este ejercicio el lector aprenderá a generar gráficas para representar redes, a calcular los descriptores básicos de la estructura de una red y de sus elementos componentes, y finalmente a desarrollar pruebas de hipótesis sencillas respecto a dichos descriptores que permitan establecer si dichas propiedades de la red son resultado del azar. Esperamos que al final del capítulo el lector haya logrado obtener una perspectiva general de las herramientas disponibles para el análisis de redes.

3.1. Análisis de redes

Pero ¿qué es una red? En su definición más simple, una **red** es una colección de objetos que tienen la capacidad de interactuar y que como resultado de dichas interacciones forman un sistema. El **análisis de redes** consiste en la aplicación de un conjunto de métodos, provenientes principalmente de la teoría de grafos en matemáticas, que permiten 1) visualizar redes, 2) describir las características tanto de la estructura general de la red como de sus elementos componentes, y 3) desarrollar modelos matemáticos y estadísticos de dichas estructuras y su dinámica (Luke 2015).

Las redes pueden ser representadas empleando objetos conocidos como **grafos**: representaciones de las relaciones entre objetos (esas representaciones no necesariamente son *gráficas*, como veremos más adelante). En un grafo, los objetos de una red son definidos como **vértices** o **nodos** y las relaciones entre ellos se indican como pares

de nodos llamados **aristas**. El grafo puede contener información acerca de la **direccionalidad** de la relación (por ejemplo, quién es el "predador" y quién la "presa"), así como del **peso** de la misma (por ejemplo, la frecuencia de visitas del polinizador para cada par de relaciones polinizador-planta) (Zweig 2016).[8]

3.1.1. Análisis de redes bipartitas

Muchos de los análisis de redes ecológicas desarrollados hasta ahora tienen que ver con las relaciones que se presentan entre los elementos (especies) de dos grupos o niveles tróficos diferentes. Por ejemplo, es posible analizar cómo interactúan especies de predadores con especies de presas, polinizadores o dispersores de semillas con plantas, u hongos micorrícicos con sus hospederos. A este tipo de redes se les llama **redes bipartitas**. En una red bipartita es común considerar una jerarquía entre los dos grupos de especies que la conforman, definiéndose un nivel superior y uno inferior. Comúnmente el nivel hace referencia al nivel trófico. Por ejemplo, en una red de polinización las plantas son asignadas al nivel inferior, mientras que los polinizadores son asignados al nivel superior. Esta diferencia es relevante al momento de interpretar los resultados del análisis, como veremos más adelante.

Para introducir el análisis de redes bipartitas primero haremos una breve presentación de la red que usaremos para el desarrollo de los ejemplos, continuaremos con una introducción a la graficación de redes bipartitas, posteriormente ilustraremos el cálculo de descriptores de redes útiles para el análisis de redes ecológicas, y finalizaremos con el desarrollo de pruebas de hipótesis simples que evalúan si la estructura

[8] Zweig, K. A. 2016. Network Analysis Literacy. A Practical Approach to the Analysis of Networks. Springer, Wien.

observada de una red puede ser producto de un proceso meramente aleatorio.

3.2. La red de polinización ***Safariland***

Dentro del lenguaje R, el paquete `bipartite` incluye datos de una red de polinización en un bosque de "Coihue" (*Nothofagus dombeyi*) en Argentina, publicada originalmente por Vázquez y Simberloff (2003).[9] La red aparece representada a manera de matriz (llamémosla A), en la que las filas corresponden a las especies de plantas (m = 9 especies) y las columnas los polinizadores (n = 27 especies). El valor en la celda A_{ij} de la matriz indica al número de veces que la especie de planta en la fila *i* fue visitada por el polinizador en la columna *j*. Esta asignación a filas y columnas no es aleatoria: las especies correspondientes al nivel inferior deben ir siempre como filas, mientras que las del nivel superior deben aparecer como columnas.

```
library(bipartite)
data(Safariland)
View(Safariland)
```

Cuadro 1. Extracto de la matriz de la red de polinización Safariland. Las filas corresponden a especies de plantas y las columnas a polinizadores. Los valores al interior de la tabla indican los eventos de polinización efectiva

	Policana albopilosa	Bombus dahlbomii	Ruizantheda mutabilis	Trichophthalma amoena	Syrphus octomaculatus
Aristotelia chilensis	673	0	110	0	0
Alstroemeria aurea	0	154	0	0	5

Continúa

[9] Vázquez, D. P., y D. Simberloff. 2003. Changes in interaction biodiversity induced by an introduced ungulate. Ecology Letters 6:1077-1083.

	Policana albopilosa	Bombus dahlbomii	Ruizantheda mutabilis	Trichophthalma amoena	Syrphus octomaculatus
Schinus patagonicus	0	0	0	0	0
Berberis darwinii	0	67	0	0	5
Rosa eglanteria	0	0	6	0	4

3.4. Graficando redes bipartitas

La representación gráfica es tal vez la forma más intuitiva de visualizar una red y por ende la presentación de resultados del análisis de redes ecológicas comúnmente incluye una figura que representa la red analizada. Existen varios paquetes de R que permiten hacer gráficas de redes, así como páginas de ayuda. En este documento exploraremos algunas de las posibilidades del paquete `igraph` (Dormann et al. 2008).[10]

```
library(igraph)
```

Para hacer una representación gráfica de la red, es necesario convertir la matriz *Safariland* en un objeto de tipo `igraph` usando la función `graph.incidence`. Esta función permite codificar una matriz que representa una red bipartita como un grafo de tipo `igraph`. Al nuevo grafo lo llamaremos *red*. Un argumento importante de la función `graph.incidence` es *weighted*, el cual establece si en la generación del grafo se deben o no tener en cuenta las frecuencias de interacción que aparecen en la matriz. Si *weighted* toma el valor `NULL`, el grafo creado corresponderá al de una matriz de presencia/ausencia de interacciones.

[10] Dormann, C. F., B. Gruber, y J. Fründ. 2008. Introducing the bipartite Package: Analysing Ecological Networks. R News 8:8-11.

```
# Sin incluir la frecuencia de las interacciones:
red <- graph.incidence(Safariland, weighted=NULL)
# Incluyendo la frecuencia de las interacciones:
red <- graph.incidence(Safariland, weighted=TRUE)

red
```

Puesto que el objeto *red* es un grafo, está compuesto por vértices y aristas, los cuales pueden ser obtenidos usando las funciones V y E, respectivamente. Tanto vértices como aristas tienen atributos asociados. Por ahora, los vértices solo tienen asignados los atributos *type* (valor lógico que indica si la especie pertenece al nivel superior, *i.e.*, si es polinizador) y *name* (nombre de la especie). Por su parte, las aristas, que se encuentran definidas como pares de especies, contienen por ahora información del número de interacciones en su atributo *weight*.

```
V(red)
V(red)$type
V(red)$name
E(red)
E(red)$weight
```

Ahora sí podemos utilizar la función `plot` para hacer una representación gráfica sencilla de *red*. Podemos también incluir argumentos adicionales para modificar el formato de la figura, como por ejemplo que excluya los nombres de los vértices o que cambie su posición relativa, entre otros. Para mayor detalle sobre argumentos adicionales de

formato de la gráfica consultar las ayudas `?plot.igraph` y `?igraph.plotting`.

```
plot(red) # Figura 1A
plot(red,    vertex.label=NA,    layout=layout_in_
circle(red)) # Figura 1B
plot(red,    vertex.label=NA,    layout=layout_on_
sphere(red)) # Figura 1C
```

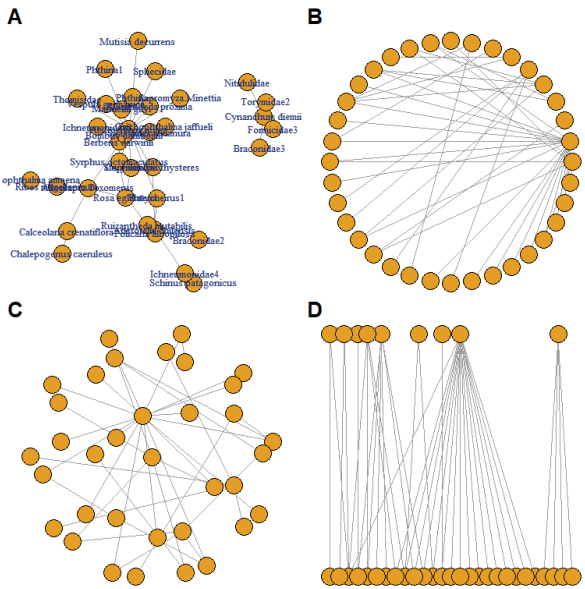


Figura 1. Diagramas de la red Safariland empleando el paquete `igraph`. A, diagrama predeterminado que incluye los nombres de los vértices (especies). B-D, diagramas con diseños modificados en forma de círculo (B), esférico (C) y bipartido (D)

Estas representaciones aún no expresan una de las características fundamentales de una red bipartita: que la red está dividida en dos partes o grupos de especies (de allí el nombre de la librería). Una forma simple de evidenciar el carácter bipartito de la red es la de utilizar el

diseño `layout_as_bipartite`, que grafica elementos de diferentes grupos en posiciones diferentes.

```
plot(red, vertex.label=NA, layout=layout_as_
bipartite(red)) # Figura 1D
```

Alternativamente, es posible modificar los atributos de los vértices, tales como tamaño, color, o nombres (entre otras), con el fin de resaltar características particulares de las especies: abundancia, si son especies exóticas, el grupo taxonómico particular al que pertenecen, etc. Los atributos de los vértices pueden ser modificados empleando la función V.

```
# Asignación del color del ícono según el grupo:
V(red)$color <- adjustcolor( c("darkgreen",
"orange"), alpha=.15)[V(red)$type+1]

plot(red, layout=layout_as_bipartite(red)) # Figura
2A

# Asignación del color del ícono según el grupo:
V(red)$color <- adjustcolor( c("darkgreen",
"orange"), alpha=.15)[V(red)$type+1]
# Asignación del color del borde del ícono (quitar
el borde):
V(red)$frame.color <- NA
# Asignación del tipo de ícono según el grupo:
V(red)$shape <- c("square","circle")[V(red)$type+1]
# Definición del tamaño del ícono de cada vértice:
V(red)$size <- 10
```

```
# Asignación de una etiqueta a cada vértice a partir de los nombres:
V(red)$label <-substr(V(red)$name, 1,3)
# Modificación del tamaño de la etiqueta proporcional al logaritmo del grado del vértice:
V(red)$label.cex = 2.5*log10(degree(red))
# Asignación del color de la etiqueta según el grupo:
V(red)$label.color <- c(“orange”, “darkgreen”)[V(red)$type+1]

plot(red) # Figura 2B
plot(red, layout=layout_in_circle(red)) # Figura 2C
```

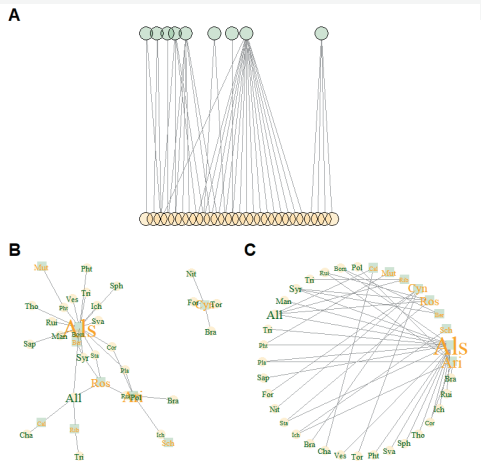


Figura 2. Diagramas de la red Safariland con atributos de los vértices modificados. A, diagrama bipartido. B, diagrama predeterminado. C, diagrama circular. En los tres casos el color de los vértices indica el grupo al que pertenece la especie (verde para plantas, amarillo para polinizadores).

Nótese cómo en las Figura 2A y 2B, el tamaño de las etiquetas de los vértices está en función del *grado* de cada vértice, es decir del número

de relaciones que éste tiene (ver más abajo la definición precisa de grado). De igual forma es posible modificar los atributos de las aristas, tales como grosor o tipo, entre otras, empleando la función E para que éstas representen atributos como la intensidad o frecuencia de la interacción.

```
# Ancho de la arista proporcional a la frecuencia de la interacción:
E(red)$width <- log(E(red)$weight)

plot(red) # Figura 3A
plot(red,  layout=layout_in_circle(red))  #  Figura 3B
plot(red, layout=layout_as_bipartite(red), vertex.color=NA) # Figura 3C
```

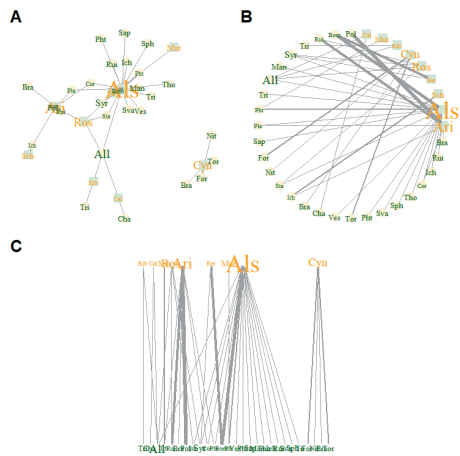


Figura 3. Diagramas de la red *Safariland* con atributos de los vértices y las aristas modificados. A, diagrama predeterminado. B, diagrama circular. C, diagrama bipartido. En ambos casos el grosor de las aristas es proporcional a la frecuencia de la interacción (en escala logarítmica).

Otra forma de representar gráficamente una red bipartita es a través de un mapa de calor, es decir, una matriz cuyas celdas indican con colores la intensidad o frecuencia de las interacciones entre las especies. Los mapas de calor van comúnmente acompañados de dendrogramas, que permiten indicar la similitud entre los objetos al interior de cada grupo. La similitud está definida por la similitud en sus interacciones, y el criterio con el que se evalúa dicha similitud puede ser modificado de acuerdo a las características particulares de los datos. Para elaborar los mapas de calor utilizaremos el paquete `gplots`.

```
library(gplots)
```

Puesto que el número de interacciones efectivas presentes en la matriz *Safariland* varía entre 0 y 673, siendo la mayoría < 5, es conveniente primero transformar estos valores a escala logarítmica para obtener una mejor ilustración de los mismos. Adicionalmente, el nombre de cada vértice se acortará para mejorar su despliegue en el gráfico.

```
# Transformación de las interacciones a escala logarítmica:
Safariscaled <- log1p(Safariland)
# Reducción del tamaño de los nombres de las plantas:
rownames(Safariscaled)                          <-
substr(rownames(Safariscaled), 1,3)
# Reducción del tamaño de los nombres de los polinizadores:
colnames(Safariscaled)                          <-
substr(colnames(Safariscaled), 1,3)
```

Ahora sí procedemos a elaborar el mapa de calor. Para ello primero definimos una escala de colores usando la función `colorRampPalette`. Posteriormente usamos la función `heatmap.2` para elaborar el mapa de calor del objeto *Safariscaled.*

```
my_palette <- colorRampPalette(c("lavenderblush", "red"))(n = 25)
heatmap.2(Safariscaled, density.info="none", trace="none", margins =c(5,5), col=my_palette, key = TRUE, key.title = "Interac", cexRow = 1, cexCol = 1, xlab= "Polinizadores", ylab="Plantas", dendrogram="both")
```

Finalmente, existen también funciones como `tkplot` o `visNetwork` (en los paquetes `igraph` y `visNetwork`, respectivamente) que permiten interactuar con la gráfica de redes y modificarla de acuerdo a las necesidades específicas. La representación gráfica de redes es en sí misma un extenso mundo con múltiples posibilidades dentro del lenguaje R, por lo que invitamos al lector a que acuda a las ayudas de los paquetes y a las páginas de internet sugeridas para profundizar en el tema.

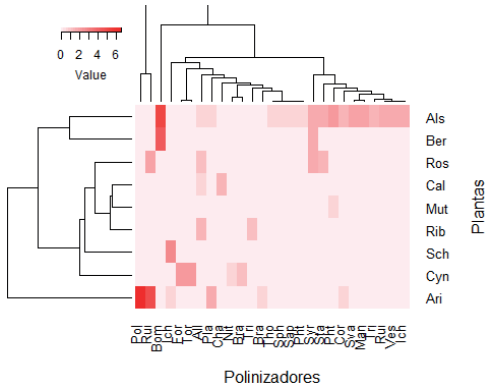


Figura 4. Mapa de calor de la red *Safariland*. La intensidad del color representa la frecuencia de la interacción (en escala logarítmica).

3.5. Índices para el análisis de redes bipartitas

Existen múltiples índices o **descriptores** para caracterizar las redes ecológicas. Dormann et al. (2009)[11] y Antoniazzi et al. (2018) sintetizan los descriptores más comúnmente usados en ecología, mientras que de Mello et al. (2016) proveen una exhaustiva guía para el análisis de redes ecológicas en general. Dada esta diversidad, es importante no perder de vista que la selección de los índices debe estar guiada por una pregunta de investigación, o de lo contrario se corre el riesgo de perderse en el cálculo de una plétora de índices cuyo significado ecológico puede ser bastante difuso (Dehling 2018).[12] De igual forma existen varias herramientas para el cálculo de los descriptores. En el contexto particular de R, existen varios paquetes especialmente diseñados para el análisis de redes ecológicas, tales como `foodweb` (Perdomo et al. n.d.),[13] `bipartite` (Dormann et al. 2008), o `igraph` (Csárdi y Nepusz 2006).[14] En este manual ilustraremos el uso del paquete `bipartite`, y presentaremos la interpretación básica de algunos de los descriptores, invitando al lector a revisar las referencias bibliográficas para ampliar su conocimiento al respecto.

Los índices de una red pueden dividirse en dos grandes grupos según el objeto específico respecto al cual se calculan: descriptores de la red y descriptores de los vértices. Los descriptores de la red hacen referencia a la estructura de la red: el número de relaciones totales,

[11] Dormann, C. F., J. Frund, N. Bluthgen, Y B. Gruber. 2009. Indices, Graphs and Null Models: Analyzing Bipartite Ecological Networks. The Open Ecology Journal 2:7-24.

[12] Dehling, D. M. 2018. The Structure of Ecological Networks BT - Ecological Networks in the Tropics: An Integrative Overview of Species Interactions from Some of the Most Species-Rich Habitats on Earth. Páginas 29-42 *en* W. Dáttilo y V. Rico-Gray, editors. Springer International Publishing, Cham.

[13] Perdomo, G., R. Thompson, y P. Sunnucks. (n.d.). Foodweb: An open-source program for the visualisation and analysis of compilations of complex food webs. Environmental Modelling and Software.

[14] Csárdi, G., y T. Nepusz. 2006. The igraph software package for complex network research. InterJournal Complex Systems 1695:1-9.

el grado de anidamiento de la red, la diversidad o distribución de las interacciones entre las especies, entre otras. Estos descriptores ayudan a entender los procesos ecológicos que definen la organización de la red, así como la robustez de las redes ante perturbaciones (Dehling 2018). Por el contrario, los descriptores de los vértices proveen estadísticas que resumen las características de los nodos, tales como el número promedio de interacciones por vértice, el número promedio de vértices con los que se relacionan, entre otros. Estos descriptores describen el papel de las especies en la red, así como su importancia tanto para sus compañeros de interacción como para la cohesión de la red en su conjunto (Dehling 2018).

En la librería `bipartite` existen dos funciones básicas para calcular los descriptores, `networklevel` y `specieslevel`, que como sus nombres indican, permiten calcular descriptores de la red y de los vértices (especies), respectivamente.

3.5.1. Índices de la red

Ilustremos el uso de `networklevel` para el cálculo de descriptores de la red Safariland:

```
networklevel(Safariland)
```

Cuadro 2. Índices de la red Safariland calculados empleando la función `networklevel`

No.	Indice	Valor	No.	Indice	Valor
1	connectance	0.160	2	web asymmetry	0.5
3	links per species	1.083	4	number of compartments	2
5	compartment diversity	1.496	6	cluster coefficient	0.111

Continúa

No.	Indice	Valor	No.	Indice	Valor
7	nestedness	19.69	8	NODF	24.55
9	weighted nestedness	0.400	10	weighted NODF	11.61
11	interaction strength asymmetry	0.391	12	specialisation asymmetry	-0.16
13	linkage density	1.603	14	weighted connectance	0.045
15	Fisher alpha	7.834	16	Shannon diversity	1.582
17	interaction evenness	0.288	18	Alatalo interaction evenness	0.41
19	H2	0.854	20	number.of.species.HL	27
21	number. of.species.LL	9.000	22	mean.number.of.shared. partners.HL	0.473
23	mean.number. of.shared. partners.LL	0.444	24	cluster.coefficient.HL	0.153
25	cluster.coefficient. LL	0.282	26	weighted.cluster. coefficient.HL	0.381
27	weighted.cluster. coefficient.LL	0.061	28	niche.overlap.HL	0.328
29	niche.overlap.LL	0.048	30	togetherness.HL	0.161
31	togetherness.LL	0.035	32	C.score.HL	0.562
33	C.score.LL	0.770	34	V.ratio.HL	0.581
35	V.ratio.LL	6.493	36	discrepancy.HL	21
37	discrepancy.LL	16.00	38	extinction.slope.HL	2.59
39	extinction.slope. LL	1.377	40	robustness.HL	0.717
41	robustness.LL	0.582	42	functional. complementarity.HL	1684
43	functional. complementarity. LL	1627	44	partner.diversity.HL	0.177

Continúa

No.	Indice	Valor	No.	Indice	Valor
45	partner.diversity.LL	0.609	46	generality.HL	1.248
47	vulnerability.LL	1.957			

El llamado de la función `networklevel` genera el despliegue de un vector de 47 descriptores de la red. Para conocer el significado de cada una de ellas, haga uso de la ayuda de la función: `?network.level` o consulte Dormann et al. (2009) y Antoniazzi et al. (2018). Los primeros 19 índices hacen referencia a la estructura de la red en general. A partir del índice 20, los cálculos se hacen para cada uno de los dos niveles de la red bipartita: nivel superior (HL) y nivel inferior (LL). La función tomó automáticamente a las filas de la matriz *Safariland* como las especies del nivel inferior y a las columnas como las del nivel superior. Es por ello que, como se indicó previamente, es necesario tomar en cuenta este comportamiento de la función para construir la matriz de interacciones.

Es posible especificar el cálculo de un número reducido de descriptores empleando el argumento *index* para definir un vector con los nombres de los descriptores deseadas:

```
networklevel(Safariland,      index=c("connectance",
"links per species"))
## connectance links per species
## 0.1604938 1.0833333
```

En el caso de la red *Safariland*, la conectancia, que indica la proporción de relaciones observadas respecto al total de relaciones posibles (*i.e.*, número de vértices existentes dividido entre el número

de vértices posibles), toma un valor de 0.160. Por su parte, el número promedio de relaciones por especie (*i.e.*, número de vértices existentes dividido entre el número de especies en la red) es de 1.083, indicando que, en promedio, cada especie tiende a relacionarse con una única especie.

3.5.2. Índices de los vértices

El llamado de la función `specieslevel` genera el cálculo de 20 descriptores diferentes para cada especie dentro de cada nivel jerárquico (resultados no mostrados).

```
specieslevel(Safariland)
```

Al igual que con la función `networklevel`, es posible utilizar el argumento *index* para escoger algun índice particular, así como el argumento *level* para establecer el nivel jerárquico respecto al cual se quiere hacer el cálculo. Por ejemplo, para calcular el grado de cada especie de planta:

```
specieslevel(Safariland, index=c(“degree”),
level=”lower”)
## degree
## Aristotelia chilensis 6
## Alstroemeria aurea 17
## Schinus patagonicus 1
## Berberis darwinii 2
## Rosa eglanteria 4
## Cynanchum diemii 4
## Ribes magellanicum 2
## Mutisia decurrens 1
## Calceolaria crenatiflora 2
```

El resultado indica que la especie de planta que tiene más relaciones con diferentes polinizadores es *Alstroemeria aurea*, que se relaciona con 17 polinizadores diferentes. A este tipo de especies se les denomina **de alto grado** o **hubs**. Por el contrario, especies como *Mutisia decurrens* y *Schinus patagonicus* solo tuvieron interacción con una especie de polinizador. A este tipo de especies se les denomina **de bajo grado**. Dicho resultado es evidente en la representación gráfica de la red (**Figura 3**).

3.5.3. Anidamiento y modularidad

Dos atributos fundamentales de las redes ecológicas, que emerge de muchos tipos de redes de interacción mutualista (e.g. polinización y dispersión de semillas), son el anidamiento y la modularidad. El primero se refiere al nivel de sobrelapamiento en las relaciones de especies de alto y bajo grado. En una red bipartita perfectamente anidada, las interacciones de una especie de bajo grado siempre son un subconjunto de las interacciones de las especies de grado mayor (**Figura 5A**). El anidamiento es un fenómeno común en las redes ecológicas, con consecuencias importantes para su funcionamiento. Una de ellas es la asimetría en las interacciones: las especies generalistas de un nivel trófico comúnmente interactúan tanto con generalistas como con especialistas del otro nivel trófico, mientras que las especies especialistas tienden a interactuar solo con especies generalistas. Otra consecuencia importante es que confiere estabilidad a la red, puesto que implica una mayor resiliencia ante la extinción de especies de alto grado (Dehling 2018).

Por su parte, la modularidad se refiere a la tendencia de diferentes subconjuntos de especies a interactuar más entre sí que con las otras especies de la red, de tal forma que la red puede ser dividida en compartimentos o **módulos** (**Figura 5B**). Un ejemplo de ello los constituye

los síndromes de polinización: ciertas especies de plantas tienen algunos atributos morfológicos, químicos o fenológicos que permiten o limitan la visita solamente a un determinado grupo de polinizadores. La mayoría de las redes ecológicas modulares presentan especies que interactúan solo dentro del módulo al que pertenecen, así como otras que sirven como enlaces entre módulos. La modularidad de las redes también confiere a la red cierto grado de robustez ante disturbios, pues la perturbación no se propaga tan fácilmente dentro de la red debido a la presencia de módulos pobremente conectados (Dehling 2018).

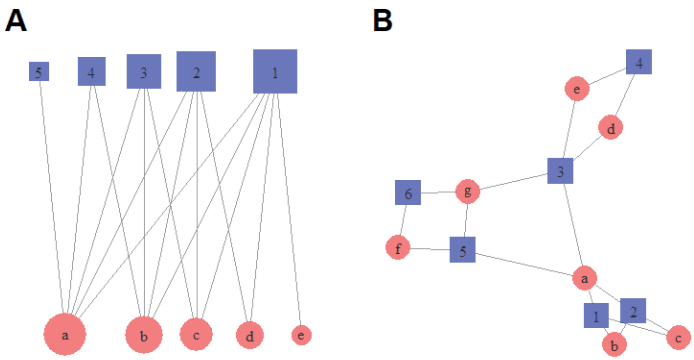


Figura 5. Anidamiento y modularidad. A, Red perfectamente anidada; nótese cómo las relaciones de una especie de grado menor en el nivel trófico inferior constituyen siempre un subconjunto de las relaciones de la especie de grado inmediatamente superior (el grado está indicado por el tamaño del ícono). B, Red modular, donde las especies se relacionan formando tres subgrupos, unidos entre sí por especies que actúan como puentes entre los módulos.

Para calcular el grado de anidamiento de una red podemos usar la función `nested`, mientras que para la modularidad podemos emplear la función `computeModules`. Es posible visualizar los módulos en una representación matricial de la red empleando la función `plotModuleWeb`, así como conocer la composición específica de cada módulo usando la función `printoutModuleInformation` (resultado no mostrado).

```
anidam <- nested(Safariland, method = “NODF2”)
anidam

modul <- computeModules(Safariland)

# Visuzalización matricial de los módulos, ver
**Figura 6**
plotModuleWeb(modul, labsize=0.5, weighted=FALSE)
# Obtención de la composición de cada módulo
printoutModuleInformation(modul)
## NODF2
## 24.5478
```

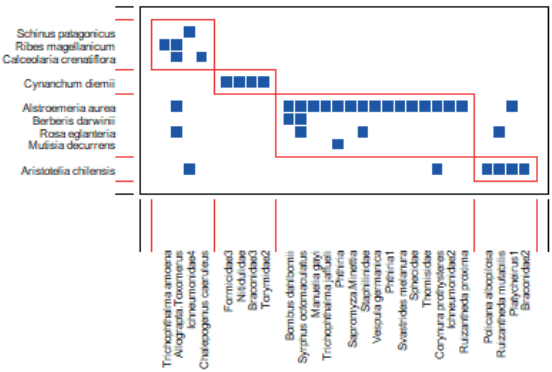


Figura 6. *Módulos de la red Safariland.* En esta representación matricial, cada módulo se encuentra definido por un rectángulo rojo, y las especies que lo constituyen son aquellas correspondientes a las filas y columnas incluidas en el mismo.

Es importante resaltar que existen varios métodos para calcular el anidamiento de una red, los cuales pueden ser definidos con el argumento *method* en la función `nested`. Invitamos al lector a revisar la ayuda de la función con el fin de que identifique el método más

acorde a su situación de estudio. También es importante tener en cuenta que los módulos identificados por la función `computeModules` pueden cambiar cada vez que se use la función. La función adicional `metaComputeModules` permite obtener el resultado más modular entre múltiples corridas de `computeModules`. En el caso del resultado ilustado en la **Figura 6**, se identificaron cuatro módulos en la red *Safariland* (**Figura 6**).

3.6. Modelos nulos en el análisis de redes

Una de las preguntas fundamentales en ecología es si lo que uno observa en la naturaleza es resultado de la acción de un proceso o mecanismo ecológico, o si existe la posibilidad de que sea resultado simplemente de un proceso aleatorio. En el caso de las redes de interacciones, puede haber procesos ecológicos que definen el número y la identidad de las especies que interactúan. Por ejemplo, puede existir selección de formas o tamaños de flores por los polinizadores, de calidad de las hojas por los herbívoros, o de hospederos que proveen más recursos por parte de las micorrizas, que pudiera hacer que algunas especies interactúen más frecuentemente con otras, o que se formen subgrupos en la red debido a procesos de especialización en la interacción. Por el contrario, se podría pensar que las interacciones están distribuidas aleatoriamente entre las especies, obedeciendo solamente a la abundancia relativa de las mismas, de tal forma que especies más abundantes tienen más interacciones.

La pregunta que surge entonces es ¿cómo saber si el patrón que observamos en una red es resultado o no del azar? Para ello podemos hacer uso de los modelos nulos (Gotelli y Graves 1996),[15] *i.e.*, proce-

[15] Gotelli, N. J., y G. R. Graves. 1996. Null models in ecology. Smithsonian Institution Press, Washington, D.C.

dimientos que con base en la aleatorización de los datos originales de la red generan nuevas redes "aleatorias", que tienen el mismo número de especies e interacciones que la red original, pero re-organizadas en estructuras derivadas completamente al azar. De estas redes aleatorias se derivan patrones que se esperaría observar en ausencia de un mecanismo ecológico estructurador. Posteriormente, los datos originales pueden ser contrastados contra estos patrones aleatorios para establecer qué tan probable es que la red presente un arreglo aleatorio o que, por el contrario, haya evidencia de procesos no aleatorios operando en su estructuración.

Para ilustrar la utilidad de los modelos nulos en el análisis de redes vamos a plantear la siguiente pregunta respecto a la red de polinización *Safariland*: ¿las especies tanto de polinizadores como de plantas presentan cierta preferencia por algunas plantas y polinizadores, respectivamente? Podemos plantear la pregunta de otra forma. La especie de polinizador *Bombus dahlbomii* presenta un total de 221 interacciones efectivas. Si esta especie no tuviera preferencia por ninguna planta, es de esperarse que sus interacciones se distribuyeran de manera aleatoria entre las nueve especies de plantas. Sin embargo, esta especie concentra sus interacciones con dos especies de plantas, *Alstroemeria aurea* y *Berberis darwinii*. Si suponemos por un momento que el número de interacciones es resultado únicamente del azar, ¿qué tan probable es observar que *Bombus dahlbomii* solo interactúe con dos especies? Si extendemos la misma pregunta a todas las especies de la red, ¿no deberíamos esperar observar un mayor número de aristas en la red (*i.e.*, mayor conectancia) si las interacciones se rigieran por el azar?

Para probar si la conectancia de la red obedece a un proceso aleatorio, lo primero que debemos hacer es generar un conjunto de redes aleatorias con base en los datos reales. Para ello usaremos la función `nullmodel`:

```
Safari.aleat <- nullmodel(Safariland, N=1000)
```

El parámetro *N* establece el número de redes aleatorias que se generan. El objeto *Safari.aleat* es una lista que contiene las mil matrices correspondientes a las mil redes aleatorias. El **Cuadro 2** presenta un extracto de una de dichas matrices, en la que se observa que las interacciones de *Bombus dahlbomii* (y las de todas las especies de polinizadores y plantas) han sido re-distribuidas.

Cuadro 2. Extracto de una de las matrices aleatorias generadas a partir de la red de polinización *Safariland.* Compárese con la matriz Safariland original presentada en el **Cuadro 1**.

	Policana albopilosa	Bombus dahlbomii	Ruizantheda mutabilis	Trichophthalma amoena	Syrphus octomaculatus
Aristotelia chilensis	472	150	78	2	11
Alstroemeria aurea	122	46	20	0	1
Schinus patagonicus	9	5	1	0	0
Berberis darwinii	41	14	12	0	2
Rosa eglanteria	9	2	1	0	0

Ahora podemos proceder a calcular la conectancia para este conjunto de redes aleatorias, para lo cual aplicaremos la función `networklevel` a todas las redes usando la función `sapply`. Posteriormente podemos comparar la conectancia real de la red *Safariland* con las conectancias de las redes aleatorias, calculando la probabilidad de observar la conectancia real bajo el supuesto de que la red está estructurada de manera aleatoria.

```
conect.aleat <- sapply(Safari.aleat, networklevel,
index="connectance")
conect.real <- networklevel(Safariland,
index="connectance")
prob.real <- sum(conect.aleat>conect.real) /
length(conect.aleat)
ifelse(prob.real > 0.5, 1-prob.real, prob.real)
## [1] 0
```

¡La probabilidad de que la conectancia de la red *Safariland* sea resultado de un proceso aleatorio es cero! (**Figura 7A**). Podemos inferir entonces que las especies de esta red se relacionan de manera dirigida o selectiva: las interacciones efectivas de un polinizador particular se dan con un subconjunto no aleatorio de especies de plantas, y viceversa.

Podemos hacer el mismo ejercicio de prueba de hipótesis, pero ahora para el anidamiento:

```
anidam.aleat <- sapply(Safari.aleat, nested, method
= "NODF2")
prob.real <- sum(anidam.aleat>anidam) /
length(anidam.aleat)
```

```
ifelse(prob.real > 0.5, 1-prob.real, prob.real)
## [1] 0
```

De igual forma, la probabilidad de que el anidamiento observado en la red *Safariland* sea resultado de un proceso aleatorio es en la práctica nula (**Figura 7B**). Lo interesante de este resultado es que, contrario a lo esperado para una red ecológica, el anidamiento es inferior a lo esperado por el azar.

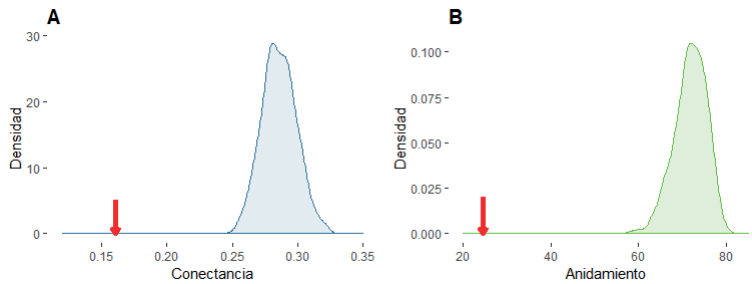


Figura 7. Distribución de frecuencias de la conectancia (A) y del anidamiento (B) de mil redes aleatorias derivadas de la red *Safariland*. Nótese cómo los valores de conectancia y anidamiento reales de la red Safariland (flechas rojas) son claramente inferiores a las de las redes aleatorias.

Finalmente, es importante resaltar que existen varios métodos diferentes para generar los modelos nulos, cada uno de los cuales permite probar hipótesis específicas. Más aún, es posible generar modelos nulos *ad hoc* para la pregunta particular de investigación (Gotelli y Graves 1996). Por ello, invitamos al lector a que revise la ayuda de la función `nullmodel`.

3.7. Más sobre el análisis de redes ecológicas en R

En el presente capítulo hemos abordado una muy breve introducción a las herramientas disponibles para el análisis de redes empleando el

lenguaje R. Sin embargo, su universo de aplicación es mucho más complejo e interesante. Las redes ecológicas no están restringidas ados grupos. Una planta puede por ejemplo estar relacionada con herbívoros, polinizadores, dispersores de semillas, hormigas mutualistas, hongos micorrízicos, plantas o líquenes epífitos, patógenos, etc. Más aún, dichas relaciones pueden cambiar a través del espacio, del tiempo y del estado de desarrollo de los seres vivos. El análisis de redes constituye una herramienta para dar respuesta a diversas preguntas en el contexto de esta "maraña" de relaciones: ¿cómo afectan en su conjunto estas relaciones al desempeño de una población en particular?, ¿cuáles son los mecanismos que ayudan a estructurar a las comunidades biológicas?, ¿cuál es el rol de la biodiversidad en el mantenimiento y funcionamiento de las redes mismas?, o ¿cómo fluye la materia y energía a través de los niveles tróficos de un ecosistema?

Para el lector interesado en profundizar en la implementación del análisis de redes ecológicas, sugerimos que revise a Antoniazzi et al. (2018), que ofrece un resumen de las principales funciones y librerías disponibles en R, a Luke (2015) que presenta un buen resumen sobre el análisis de redes en R no restringido a situaciones de estudio ecológicas, o finalmente a Mello et al. (2016) para una perspectiva más amplia en torno a conceptos y software especializado.

Capítulo 4. Análisis de dependencia y autocorrelación espacial usando R[1]

Rodrigo Tapia-McClung
Centro de Investigación en Ciencias de Información Geoespacial, A.C.

Introducción

En el estudio de patrones y procesos que ocurren en algún lugar en el espacio o en el territorio, la autocorrelación espacial es un concepto importante. De manera general, se puede decir que extiende las ideas y los conceptos de la estadística tradicional para tomar en cuenta las relaciones que ocurren en una región de estudio. La autocorrelación es una medida de qué tan parecidos son los datos con los que estamos trabajando. Es de particular interés medir qué tanto se parecen las observaciones cercanas. Sin embargo, en el espacio la cercanía se puede medir de varias maneras. Si pensamos en ubicaciones sobre un terreno plano, podemos medir la distancia entre ellas en dos dimensiones, pero si pensamos en un terreno más rugoso, la distancia entre estas ubicaciones debe incluir una tercera dimensión. Para poder medir la autocorrelación espacial, es esencial estudiar la configuración de los datos en el espacio en el que se encuentran.

Un estadístico que se usa comúnmente para medir la autocorrelación espacial global es la *I* de Moran. Se puede escribir como sigue:

$$I = \frac{n}{\sum_i \sum_{j \neq i} w_{ij}} \frac{\sum_i \sum_{j \neq i} w_{ij} (x_i - \overline{x})(x_j - \overline{x})}{\sum_i (x_i - \overline{x})^2}, \qquad (1)$$

[1] Los datos necesarios para la realización de algunos de los ejercicios de este capítulo se encuentran en el vínculo: https://github.com/metodosmorelia/Manual_vol._1_Datos/archive/master.zip o los puedes descargar de este repositorio.

donde *x* son las observaciones y los representan los elementos de una matriz de pesos (renglón *i*, columna *j*). La matriz de pesos es una forma sintética de codificar las relaciones de cercanía entre los diferentes elementos que se consideran en el estudio de la autocorrelación espacial. Si vemos detenidamente la ecuación (1), es como una covarianza ($(\sum_i \sum_{j \neq i} w_{ij} (x_i - \overline{x})(x_j - \overline{x})$ ponderada por una varianza de la muestra ($\frac{1}{n} \sum_i (x_i - \overline{x})^2$).

Cuando en un mapa se agrupan los valores parecidos de las observaciones, decimos que hay autocorrelación espacial positiva (ocurre que y mientras más cercana sea a , mayor es la autocorrelación). En el caso contrario, cuando en un mapa se agrupan valores diferentes de las observaciones, decimos que la autocorrelación es negativa (ocurre que y mientras más cercana sea a , mayor es la autocorrelación). Un valor cercano a 0 indica que la autocorrelación espacial no es significativa. Otra manera de interpretar el valor de la *I* de Moran es que sugiere que los datos parecidos se acumulan y implica que los datos están dispersos.

Como la estadística tradicional depende del hecho de que las observaciones sean independientes entre sí, en el estudio de procesos y patrones se vuelve relevante saber si hay autocorrelación espacial ya que, de haberla, se viola este principio fundamental. Sin embargo, al analizar las relaciones y la dependencia espacial de los datos se puede entender un poco más acerca del proceso que da origen al patrón observado.

La ecuación (1) nos da una medida de autocorrelación espacial para todo el conjunto de datos. Es decir, es una medida global. Anselin

propuso los indicadores locales de asociación espacial (LISA[2] por sus siglas en inglés), una manera de medir las contribuciones locales de cada observación a la autocorrelación global. En particular, propuso una versión local de la *I* de Moran que se puede escribir como:

$$I_i = \frac{(x_i - \overline{x}) \sum_{j \neq i} w_{ij} \left(x_j - \overline{x}\right)}{\frac{1}{n} \sum_{j \neq i} (x_j - \overline{x})^2} \qquad (2)$$

El objetivo de este capítulo es analizar la autocorrelación espacial global y local de dos conjuntos de datos espaciales con características diferentes: unos son puntos y otros polígonos. Aunque R no es un sistema de información geográfica, la comunidad ha escrito librerías específicas que permiten extender el uso del lenguaje de programación para analizar y visualizar datos geográficos. En este capítulo vamos a interactuar con datos espaciales y hacer mapas en los que se muestren algunas conclusiones a las que se puede llegar después del estudio de las relaciones de dependencia y autocorrelación espacial.

4.1. Análisis de Patrones de Puntos

Queremos estudiar el comportamiento espacial de un conjunto de puntos. En particular, explorar si presentan autocorrelación espacial global y local. Para ello, es importante tener presente las ecuaciones (1) y (2) y recordar que la matriz de pesos () es fundamental para expresar las relaciones de cercanía entre los diferentes elementos que se están estudiando. Hay que construir esta matriz.

Los datos espaciales que usaremos están en formato shapefile (SHP). Un SHP es una colección de varios archivos que permiten codificar geometrías (puntos, líneas o polígonos) de datos espaciales

[2] https://doi.org/10.1111/j.1538-4632.1995.tb00338.x

que tienen atributos y que pueden tener una proyección geográfica. Como mínimo, un SHP requiere tres archivos con extensiones: `shp` (la geometría), `shx` (tabla de índices de las geometrías), `dbf` (la tabla de atributos). Adicionalmente, se puede tener un archivo con extensión `prj` en el cual se define el sistema de coordenadas y la proyección geográfica de los datos.

Requisitos: tener los SHPs en alguna carpeta y cambiar a ese directorio de trabajo. Por ejemplo:

```
# Cambiar directorio de trabajo:
setwd("C:/Descargas/R/practica1")
```

Asegurarse de tener instaladas las librerías que vamos a usar:

```
# Lista de librerías:
list.of.packages <- c("rgdal", "spdep")
# Ver qué no está instalado:
new.packages <- list.of.packages[!(list.of.packages %in%
 installed.packages()[, "Package"])]
# Si falta algo, instalarlo:
if (length(new.packages)) install.packages(new.packages)
```

Usaremos `rgdal` para abrir dos shapefiles en R (`mpb00.shp` y `BOUNDARY.SHP`)

```
library(rgdal)
mpb <- readOGR(".", "mpb00")
frontera <- readOGR(".", "BOUNDARY")
```

Ahora cargamos la librería `spdep`:

```
library(spdep)
```

4.1.1. Umbral de distancia y matriz de adyacencia

Queremos encontrar el umbral de distancia que garantice que **cada** punto tenga **al menos** un vecino. Para ello, primero encontramos el primer vecino más cercano a cada punto (con `knearneigh`), luego calculamos la distancia (de cada punto a ese vecino más cercano) y por último encontramos la distancia máxima (de todas esas distancias). Uso `k1` como referencia a los *k Nearest Neighbors* (KNN) con `k=1`:

```
# Calcular los vecinos más cercanos y hacer una lista:
k1 <- knn2nb(knearneigh(mpb, k = 1, longlat = TRUE))
```

Ahora encontramos la distancia entre cada punto y su vecino más cercano (con `nbdists`), hacemos un vector a partir de esa lista de distancias de los vecinos más cercanos (con `unlist`) y encontramos su máximo (con `max`).

```
# Calcular distancias de los vecinos más cercanos,
hacer un vector y encontrar el máximo:
distancia <- max(unlist(nbdists(k1, mpb)))
 [1] 6927.891
```

Con este umbral de distancia garantizamos que **TODOS** los puntos tendrán por lo menos 1 vecino. Ahora encontramos los vecinos de cada punto a esa distancia.

```
# Encontrar vecinos:
vecinos <- dnearneigh(mpb, 0, distancia)
```

Si quieres, puedes ver algo de información de esta lista de vecinos con:

```
summary(vecinos)
```

Ahora definimos una lista de pesos para los puntos y sus vecinos. Hay de diferentes tipos: binarias (0, 1), estandarizadas por renglón (cada renglón suma 1), estandarizadas por renglón y columna, etc. Claramente, la elección de cómo se construyan los pesos afectará la estadística. Recordar que la definición de la *I* usa .

Vamos a usar el caso típico, que es una matriz binaria estandarizada por renglones, caso `W`.

```
# Lista de pesos:
mpb.lw <- nb2listw(vecinos) # style = "W" estandarizada por renglón
#mpb.lw <- nb2listw(vecinos, style = "B") # pesos binarios
#mpb.lw <- nb2listw(vecinos, style = "C") # estandarizada por renglón globalmente
```

Si quieres ver información acerca de la lista de pesos puedes escribir:

```
mpb.lw
```

4.1.2. Hacer una *gráfica de Moran*

En una gráfica de dispersión de Moran, graficamos nuestros valores estandarizados contra una variable de retraso espacial, la cual es un promedio de las unidades vecinas (contiguas). Queremos ver una relación en términos de desviaciones estándar alrededor de la media, de modo que transformamos los valores de en una variable normalizada estandarizada que podemos calcular con `scale()`. Necesitamos un par de variables nuevas:

```
# Estandarizar valores de Z y salvarlos como una nueva columna:
mpb$zScaled <- scale(mpb$Z)
# Calcular la variable de retraso y guardarla:
mpb$lagZ <- lag.listw(mpb.lw, mpb$Z)
```

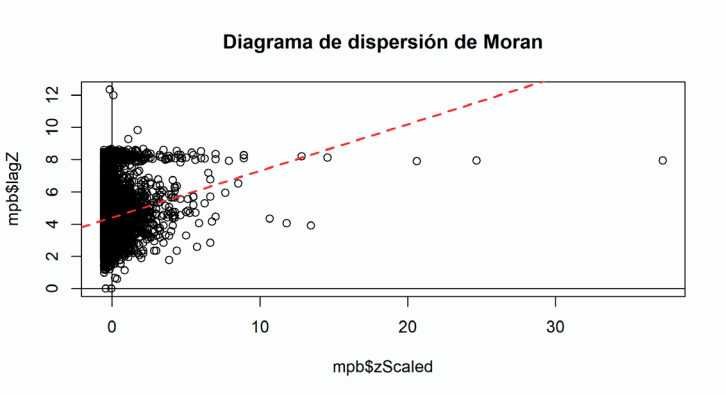
Si quieres, puedes explorar estas variables:

```
summary(mpb$zScaled)
summary(mpb$lagZ)
```

Y hacemos un diagrama de dispersión:

```
# Diagrama de dispersión:
plot(mpb$zScaled, mpb$lagZ)
```

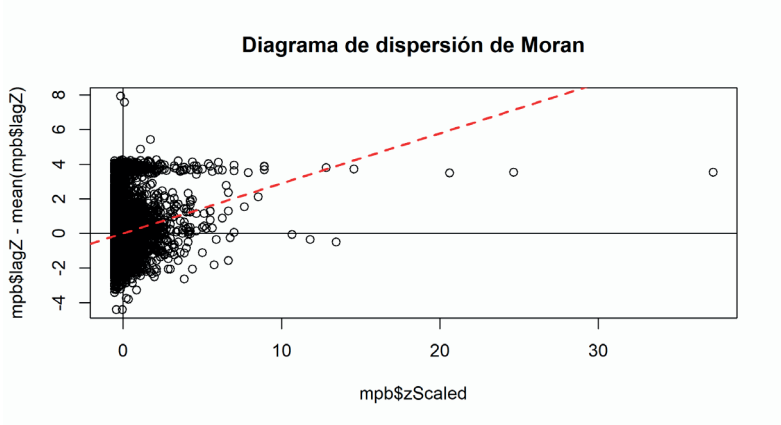
```
# Ejes que pasan por el origen:
abline(h = 0, v = 0)
# Agregar recta de ajuste lineal entre las dos variables:
abline(lm(mpb$lagZ ~ mpb$zScaled), lty = 2, lwd = 2, col = "red")
title("Diagrama de dispersión de Moran")
```



Algo no está bien porque la recta de ajuste no pasa por el origen. Hay que hacer un ligero cambio en las variables que queremos graficar. Con las variables estandarizadas, las unidades en la gráfica corresponderán a desviaciones estándar. La variable Z ya está normalizada, solo tenemos que modificar la variable de retraso espacial. Para esto, a `lagZ` le restamos su promedio.

```
# Con variables estandarizadas:
plot(mpb$zScaled, mpb$lagZ - mean(mpb$lagZ))
# o bien, plot(mpb$zScaled, scale(mpb$lagZ))
abline(h = 0, v = 0)
```

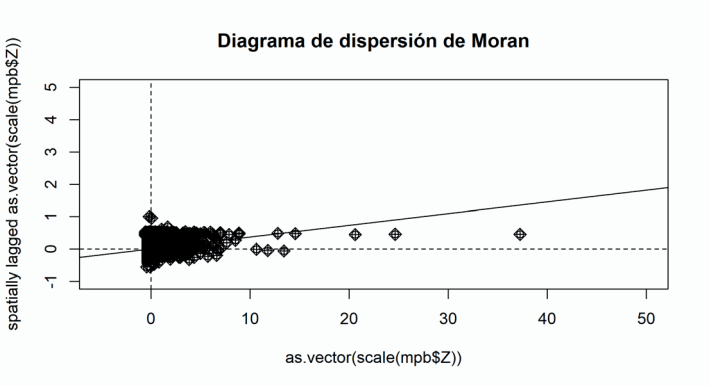
```
abline(lm(mpb$lagZ - mean(mpb$lagZ) ~ mpb$zScaled),
lty = 2, lwd = 2,
 col = "red")
title("Diagrama de dispersión de Moran")
```



Ahora sí, los ejes están centrados en el origen y la recta de ajuste pasa por `(0, 0)`. Hay una función más práctica que hace esto por nosotros:

```
# Moran plot
#moran.plot(mpb$Z, mpb.lw)
#moran.plot(mpb$Z, mpb.lw, xlim = c(-300, 300), ylim
= c(-20, 20))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw)
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim =
c(-45, 45))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim =
c(-45, 45), ylim = c(-5, 5))
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim =
c(-45, 45), ylim = c(-45, 45))
```

```
#moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim =
c(-300, 300), ylim = c(-45, 45))
moran.plot(as.vector(scale(mpb$Z)), mpb.lw, xlim =
c(-5, 50),
 ylim = c(-1, 5), labels = F)
title("Diagrama de dispersión de Moran")
```

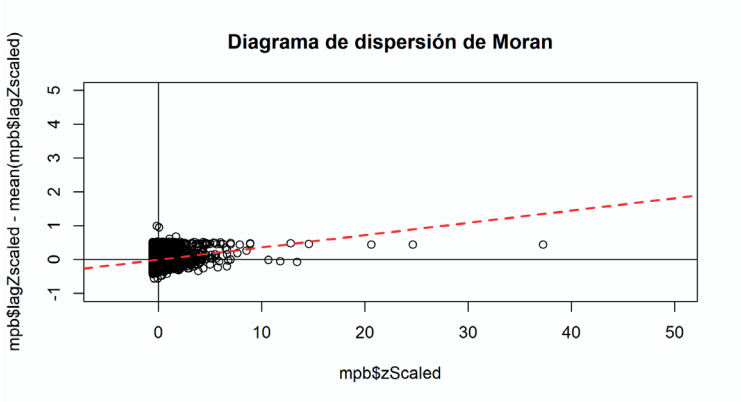


La pendiente del diagrama de dispersión parece ser positiva, lo que indicaría que hay un cierto nivel de autocorrelación espacial positiva a nivel **global**.

Una diferencia importante es que los límites de nuestra gráfica están muy diferentes de los que se obtienen con `moran.plot`. Se ven parecidas, pero no iguales. Para lograr esto, nos damos cuenta de que la variable de retraso espacial en el eje horizontal también debe ser una variable estandarizada y normalizada.

```
# Variables de retraso espacial estandarizadas
mpb$lagZscaled <- lag.listw(mpb.lw, scale(mpb$Z)) #
moran.plot hace esto internamente
plot(mpb$zScaled,          mpb$lagZscaled          -
```

```
mean(mpb$lagZscaled),
 xlim = c(-5, 50), ylim = c(-1, 5))
abline(h = 0, v = 0)
abline(lm(mpb$lagZscaled - mean(mpb$lagZscaled) ~
mpb$zScaled),
 lty = 2, lwd = 2, col = “red”)
title(“Diagrama de dispersión de Moran”)
```



4.1.3. Prueba estadística

Para una prueba estadística sobre la presunción de autocorrelación en un patrón de puntos podemos hacer lo siguiente:

```
# Prueba estadística:
moran.test(mpb$Z, mpb.lw)
 Moran I test under randomization
 data: mpb$Z
 weights: mpb.lw
 Moran I statistic standard deviate = 25.915, p-value
< 2.2e-16
 alternative hypothesis: greater
```

```
sample estimates:
Moran I statistic Expectation Variance
3.639684e-02 -1.190476e-04 1.985404e-06
```

Como el valor de la estadística es mayor que el valor esperado y la varianza es muy pequeña, podemos pensar que, en efecto, hay autocorrelación espacial positiva, es decir, que el patrón de puntos que observamos tiende a *acumularse* en ciertas zonas y que no estamos ante un proceso espacial completamente aleatorio. Pero, ¿cómo saber si el valor de la *I* de Moran es *suficientemente* grande, o no, para que sugiera que un modelo de un proceso con autocorrelación sea una mejor alternativa a uno en el que los valores que tenemos sean observaciones independientes?

Hay dos partes que tenemos que analizar: i) el valor de *I* en una escala absoluta y ii) qué tan probable es que el valor observado de *I* se obtenga si las observaciones fueran independientes.

Para i), podemos usar la siguiente función y obtener el rango de valores de *I*. **NOTA:** esta ejecución puede tardar **mucho** tiempo.

```
# Rango de valores de I:
moran.range <- function(lw){
 wmat <- listw2mat(lw)
 return(range(eigen((wmat+t(wmat))/2)$values))
}
moran.range(mpb.lw)
 [1] -1.000000 1.037054
```

Como el valor que se obtiene está entre `-1.000` y `1.037`, podemos pensar que, en efecto, obtuvimos un valor que indica autocorrelación espacial global positiva, pues es de 0.036, mayor que la media (0.018).

Para ii), podemos hacer una prueba de inferencia estadística clásica, usar la hipótesis nula de que no haya autocorrelación espacial y calcular la probabilidad de obtener un valor de la *I* de Moran igual o mayor a la observada. Sin entrar en mayores detalles, bajo ciertos supuestos se puede mostrar que *I* se puede aproximar por una distribución normal. Podemos hacer una prueba de inferencia estadística con una distribución normal al usar el argumento `randomisation = FALSE` como sigue:

```
# Inferencia bajo normalidad:
moran.test(mpb$Z, mpb.lw, randomisation = FALSE)
 Moran I test under normality
 data: mpb$Z
 weights: mpb.lw
 Moran I statistic standard deviate = 25.419, p-value
< 2.2e-16
 alternative hypothesis: greater
 sample estimates:
 Moran I statistic Expectation Variance
 3.639684e-02 -1.190476e-04 2.063765e-06
```

Volvemos a obtener valores parecidos y tenemos evidencia para rechazar la hipótesis nula de que los valores de Z están distribuidos de manera normal.

Si seguimos sin estar convencidos, podemos emplear una estrategia más: *simulaciones Monte Carlo*. En este enfoque se hace un cierto número de permutaciones sobre los datos, se les asignan a los puntos y se calcula la *I* de Moran para cada permutación. La *I verdadera* se calcula a partir de los datos y ubicaciones originales. Si la hipótesis nula es cierta, la probabilidad de obtener el patrón observado es igual a cualquier otra permutación de los valores de Z en los demás puntos. Si *m* es el número de simulaciones que obtienen una *I* de Moran mayor a la observada y *M* es el número total de simulaciones, la probabilidad de obtener la *I* de Moran observada o un valor mayor es:

Podemos hacer este cálculo como sigue. El tercer argumento es el número de simulaciones. **Atención:** dependiendo el valor del tercer argumento, puede tardar **mucho** en terminar de correr las permutaciones…

```
# Simulaciones Monte Carlo:
moran.mc(mpb$Z, mpb.lw, 9999)
```

```
 Monte-Carlo simulation of Moran I
 data: mpb$Z
 weights: mpb.lw
 number of simulations + 1: 10000
 statistic = 0.036397, observed rank = 10000, p-value
= 1e-04
 alternative hypothesis: greater
```

Observa que volvemos a obtener evidencia en favor de rechazar la hipótesis nula.

4.1.4. Estadística local

Recuerda que al inicio definimos la versión local de la *I* de Moran, . La podemos calcular como sigue:

```
# Calcular la I de Moran local:
lmoran <- localmoran(mpb$Z, mpb.lw)
```

Si quieres explorar el resultado, puedes usar:

```
summary(lmoran)
```

4.1.5. Mapa de cúmulos

A partir de la gráfica de dispersión de Moran, vamos a codificar los valores de cada cuadrante en un *tipo de cúmulo*: aquellos de valores altos rodeados de valores altos, los de valores bajos rodeados de bajos, los de valores bajos rodeados de altos y los de valores altos rodeados de bajos.

Primero definimos una variable en la cual vamos a almacenar esta definición de los cuadrantes. Haremos que este nuevo vector sea de tipo numérico y su tamaño sea igual al número de elementos que hay en el cálculo de la *I* de Moran local (igual al número de puntos en el patrón).

```
# Definir vector de cuadrante:
cuadrante <- vector(mode = “numeric”, length =
nrow(lmoran))
```

También vamos a definir un nivel de significancia estadística con la cual vamos a trabajar. Empezamos con un valor *relajado* en vez de uno *estricto*.

```
# Definir significancia:
significancia <- 0.05
```

Como lo hicimos anteriormente, definimos dos variables estandarizadas:

```
# Centrar la variable de interés alrededor de su media:
centerZ <- scale(mpb$Z) #Es lo mismo que (mpb$Z-mean(mpb$Z))/sd(mpb$Z)
# Centrar el valor de la I de Moran local alrededor de su media:
centerLag <- mpb$lagZscaled
```

Ahora catalogamos los valores de acuerdo con el cuadrante en el que se encuentren en la gráfica de Moran:

```
# Definición de cúmulos:
cuadrante[centerZ > 0 & centerLag > 0] <- 1 # Altos-Altos
cuadrante[centerZ < 0 & centerLag < 0] <- 2 # Bajos-Bajos
cuadrante[centerZ < 0 & centerLag > 0] <- 3 # Bajos-Altos
cuadrante[centerZ > 0 & centerLag < 0] <- 4 # Altos-Bajos
```

```
cuadrante[lmoran[,5] > significancia] <- 0 # No
significativos
```

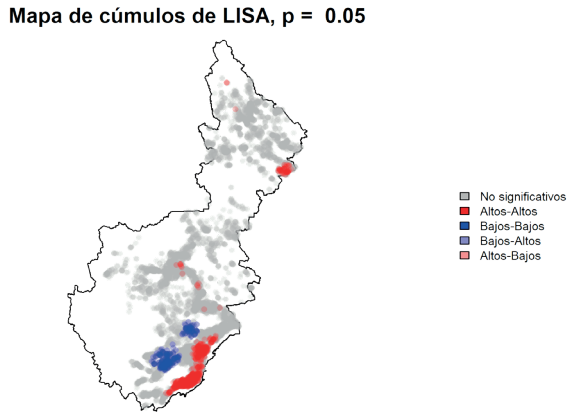
Definimos los colores que vamos a usar para el mapa de cúmulos:

```
# Colores para los cúmulos:
cColors <- c(rgb(0.74, 0.74, 0.74), rgb(1, 0, 0),
rgb(0, 0, 1),
 rgb(0.58, 0.58, 1), rgb(1, 0.58, 0.58))
# gris (no significativos), rojo (altos-altos), azul
(bajos-bajos), azul
# claro (bajos-altos) y rojo claro (altos-bajos)
```

Hacemos el primer mapa:

```
# Definir márgenes para ocupar todo el espacio:
par(mar = c(0, 0, 1, 0))
# Primer mapa:
plot(frontera)
# Pintar no significativos:
plot(mpb[cuadrante == 0, ], col = rgb(0.74, 0.74,
0.74, alpha = 0.2),
 add = T, pch = 16, cex = 0.75)
# Pintar altos-altos:
plot(mpb[cuadrante == 1, ], col = rgb(1, 0, 0, alpha
= 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar bajos-bajos:
```

```
plot(mpb[cuadrante == 2, ], col = rgb(0, 0, 1, alpha
= 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar bajos-altos:
plot(mpb[cuadrante == 3, ], col = rgb(0.58, 0.58, 1,
alpha = 0.75), add = T,
 pch = 16, cex = 0.75)
# Pintar altos-bajos:
plot(mpb[cuadrante == 4, ], col = rgb(1, 0.58, 0.58,
alpha = 0.75), add = T,
 pch = 16, cex = 0.75)
legend("right", fill = cColors, bty = "n", y.intersp
= 1, x.intersp = 1,
  legend = c("No significativos", "Altos-Altos ",
"Bajos-Bajos",
 "Bajos-Altos", "Altos-Bajos"), cex = 0.7,)
title(paste("Mapa de cúmulos de LISA, p = ",
significancia))
```



4.1.6. Mapa de significancia

Ahora vemos cómo están distribuidas las distintas significancias en nuestros datos. Primero definimos un vector para guardar los valores:

```
# Definir vector de significancias:
pValues <- vector(mode = “numeric”, length = nrow(lmoran))
```

Y obtenemos cada nivel de significancia estadística a partir de los datos:

```
# Definir niveles de significancia:
pValues[(lmoran[, 5] > 0.05)] <- 0 # No significativos
pValues[(lmoran[, 5] <= 0.05)] <- 4
pValues[(lmoran[, 5] <= 0.01)] <- 3
pValues[(lmoran[, 5] <= 0.001)] <- 2
pValues[(lmoran[, 5] <= 0.0001)] <- 1
```

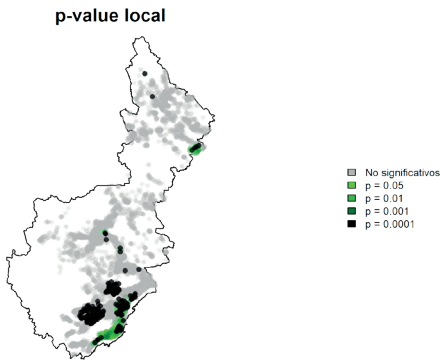
Definimos los colores para este mapa:

```
# Colores para las significancias:
colorsPValue <- c(rgb(0.74, 0.74, 0.74), rgb(0.22, 0.98, 0.3),
 rgb(0, 0.75, 0), rgb(0, 0.44, 0), rgb(0, 0, 0))
# gris, verde claro, verde medio, verde oscuro, negro
```

Hacemos el segundo mapa:

```
# Segundo mapa. Definir márgenes para ocupar todo el
espacio:
par(mar = c(0, 0, 1, 0))
plot(frontera)
# Pintar no significativos:
plot(mpb[pValues == 0, ], col = rgb(0.74, 0.74, 0.74,
alpha = 0.2),
 add = T, pch = 16, cex = 0.75)
# Pintar 0.05:
plot(mpb[pValues == 1, ], col = rgb(0.22, 0.98, 0.3,
alpha = 0.2),
 add = T, pch = 16, cex = 0.75)
# Pintar 0.01:
plot(mpb[pValues == 2, ], col = rgb(0, 0.75, 0, alpha
= 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar 0.001:
plot(mpb[pValues == 3, ], col = rgb(0, 0.44, 0, alpha
= 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar 0.0001:
plot(mpb[pValues == 4, ], col = rgb(0, 0, 0, alpha =
0.75), add = T,
 pch = 16, cex = 0.75)
legend("right", fill = colorsPValue, bty = "n",
y.intersp = 1, x.intersp = 1,
 legend = c("No significativos", "p = 0.05", "p =
0.01", "p = 0.001",
```

```
 “p = 0.0001”), , cex = 0.7)
title(“p-value local”)
```



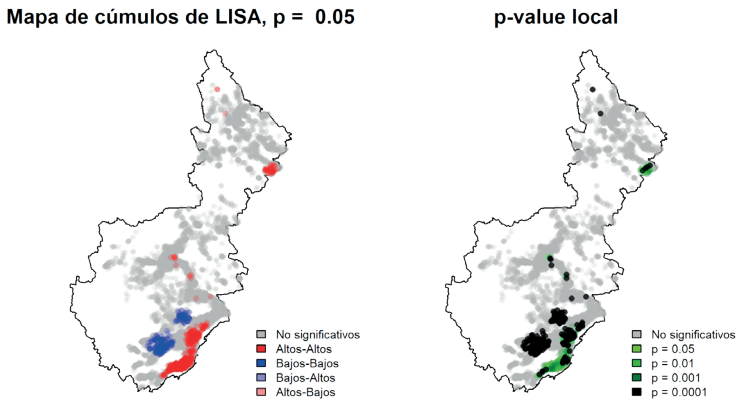
Los dos mapas juntos:

```
# Definir que tendremos un gráfica de 1 renglón y dos
columnas:
par(mfrow = c(1, 2))
# Sin margen arriba, un poco de margen entre cada
plot y definir tipo de
# región cuadrada:
op <- par(oma = c(0, 0, 0, 0), mar = c(0, 0, 2, 0),
pty = “s”)
# Primer mapa:
plot(frontera)
# Pintar no significativos:
plot(mpb[cuadrante == 0, ], col = rgb(0.74, 0.74,
0.74, alpha = 0.2),
 add = T, pch = 16, cex = 0.75)
```

```
# Pintar altos-altos:
plot(mpb[cuadrante == 1, ], col = rgb(1, 0, 0,
alpha = 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar bajos-bajos:
plot(mpb[cuadrante == 2, ], col = rgb(0, 0, 1,
alpha = 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar bajos-altos:
plot(mpb[cuadrante == 3, ], col = rgb(0.58, 0.58,
1, alpha = 0.75),
 add = T, pch = 16, cex = 0.75)
# Pintar altos-bajos:
plot(mpb[cuadrante == 4, ], col = rgb(1, 0.58,
0.58, alpha = 0.75),
 add = T, pch = 16, cex = 0.75)
legend("bottomright", fill = cColors, y.intersp = 1,
x.intersp = 1,
  legend = c("No significativos", "Altos-Altos",
"Bajos-Bajos",
 "Bajos-Altos", "Altos-Bajos"), bty = "n", cex =
0.7)
title(paste("Mapa de cúmulos de LISA, p = ",
significancia))


# Segundo mapa:
plot(frontera)
```

```
# Pintar no significativos:
plot(mpb[pValues == 0, ], col = rgb(0.74, 0.74,
0.74, alpha = 0.2),
 add = T, pch = 16, cex = 0.75)
# Pintar 0.05:
plot(mpb[pValues == 1, ], col = rgb(0.22, 0.98,
0.3, alpha = 0.2),
 add = T, pch = 16, cex = 0.75)
# Pintar 0.01:
plot(mpb[pValues == 2, ], col = rgb(0, 0.75, 0,
alpha = 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar 0.001:
plot(mpb[pValues == 3, ], col = rgb(0, 0.44, 0, alpha
= 0.2), add = T,
 pch = 16, cex = 0.75)
# Pintar 0.0001:
plot(mpb[pValues == 4, ], col = rgb(0, 0, 0, alpha =
0.75), add = T,
 pch = 16, cex = 0.75)
legend("bottomright", fill = colorsPValue, y.intersp
= 1, x.intersp = 1,
 legend = c("No significativos", "p = 0.05", "p =
0.01", "p = 0.001",
 "p = 0.0001"), bty = "n", cex = 0.7)
title("p-value local")
```




```
# Regresar a una sola gráfica en toda la página:
par(mfrow = c(1,1))
```

4.2. Vecindad y dependencia espacial para polígonos

En esta segunda parte nos interesa estudiar algo parecido a la parte anterior, pero ahora con un conjunto de polígonos: explorar si presentan autocorrelación espacial global y local.

Requisitos: descargar los SHPs en alguna carpeta y cambiar a ese directorio de trabajo. Por ejemplo:

```
# Cambiar directorio de trabajo:
setwd("C:/Descargas/R/practica2")
```

Para esta segunda parte requerimos unas cuantas librerías más. De modo que hay que asegurarse de tenerlas instaladas:

```
# Lista de librerías:
list.of.packages <- c("rgdal", "sp", "GISTools",
"RColorBrewer", "ggplot2",
```

```
 "reshape2", "grid", "gridExtra", "spdep")
# Ver qué no está instalado:
new.packages <- list.of.packages[!(list.of.packages
%in%
 installed.packages()[, "Package"])]
# Si falta algo, instalarlo:
if (length(new.packages)) install.packages(new.
packages)
```

Volvemos a usar la librería `rgdal` para abrir un shapefile en R (`estados_sorted.shp`)

```
library(rgdal)
edos <- readOGR(".","estados_sorted",stringsAsFactors
= FALSE,
 GDAL1_integer64_policy = T)
```

Ahora nos interesa leer un archivo de tipo CSV que tiene información acerca de homicidios dolosos vinculados con la delincuencia organizada por año y por estado:

```
# La primera columna es de texto y las demás de
enteros:
homicidios <- read.csv("homicidios.csv", colClasses
= c(rep("character", 1),
 rep("integer", 7)))
```

Si quieres, puedes ver las columnas de este archivo con:

```
colnames(homicidios)
```

Observa que en nuestro CSV los nombres de las columnas son los años. R les asigna una letra al inicio, de modo que se llaman `A2006`, `A2007`, etc.

Cargamos la librería `sp`:

```
library(sp)
```

Ahora usamos la función `merge` para unir las tablas de atributos por medio de una llave única. En este caso, la clave del estado en cuestión, `CVE_ENT`.

```
# Unir los datos de homicidios a los estados:
edos <- merge(edos, homicidios, by.x = "CVE_ENT")
```

Si ejecutas `edos@data` verás que cada estado tiene la nueva información que le acabamos de unir. Puedes ver las nuevas columnas en la tabla de los estados con:

```
names(edos@data)
```

4.2.1. Exploración visual de los datos

Antes de entrar de lleno en el análisis de la autocorrelación y la dependencia espacial para el caso de polígonos, es interesante primero mapear las variables que nos interesan. Tenemos 7 años de datos de homicidios y solo toma algo de tiempo hacer que la computadora haga todo el procesamiento de los datos por nosotros. Pero vale la pena ser un poco sensatos en lo que lo que vamos a pedir.

Podemos hacer un mapa de desviaciones estándar que nos muestre la variación espacial alrededor del promedio de homicidios por año por estado.

Vamos a ver dos maneras de hacer estos mapas, cada una con sus pros y contras. Primero vamos a trabajar con la librería `GISTools` y después con `ggplot2`.

4.2.2. Un mapa de desviaciones estándar - `GISTools`

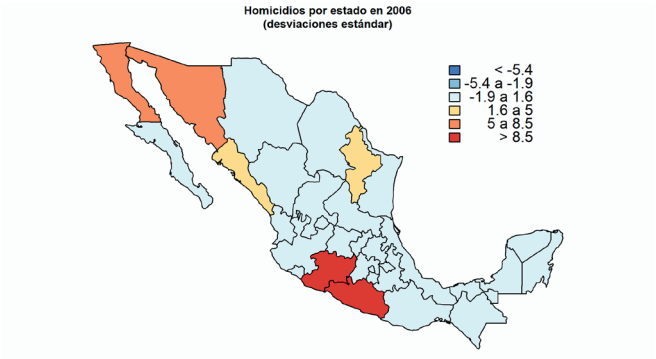
Cargamos la librería y otra más para usar colores:

```
library(GISTools)
library(RColorBrewer)
```

Definimos nuestro mapa temático:

```
# Definir márgenes para ocupar todo el espacio:
par(mar = c(0, 0, 1.5, 0))
# Definir un esquema para colorear el mapa de acuerdo
a desviaciones
# estándar:
shades <- auto.shading(edos$A2006, cutter = sdCuts,
n = 6,
 cols = rev(brewer.pal(6, "RdYlBu")))
# Definimos el mapa temático:
choropleth(edos, edos$A2006, shades)
# Agregar una leyenda:
```

```
choro.legend(-95, 32, shades, under = "<", over = ">", between = "a",
 box.lty = "blank", x.intersp = 0.5, y.intersp = 0.75)
# Agregar título:
title(main = "Homicidios por estado en 2006\n(desviaciones estándar)",
 cex.main = 0.75)
```



Este mapa muestra que, para 2006, la mayoría de los estados tienen valores por debajo del promedio y hay uno con valores entre y 1 y 2 SD, y dos con valores por encima de 2 SD. Pero, ¿cómo se compara esto año con año? Vamos a hacer mapas de este estilo para cada año y compararlos.

4.2.3. Varios mapas de desviaciones estándar

Podríamos repetir las mismas instrucciones de arriba tantas veces como mapas queramos producir, pero eso es ineficiente. Mejor vamos a hacer una función:

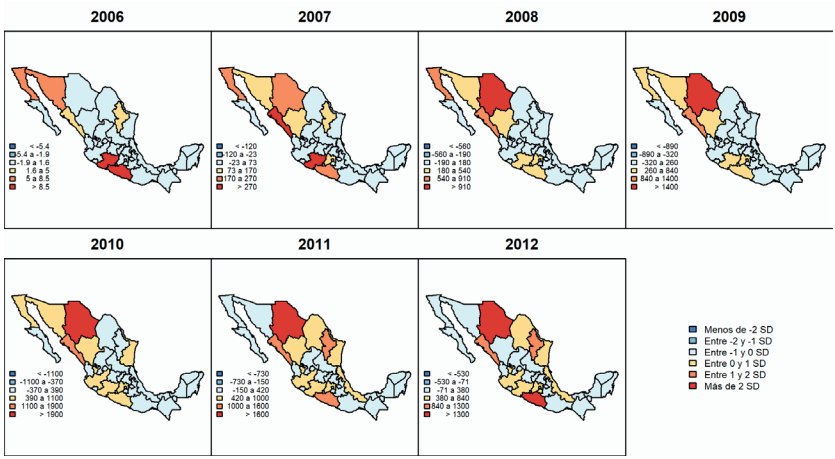
```
# Definir una función que haga un mapa como el anterior:
makeChoro <- function(var, title) {
```

```
 shades <- auto.shading(var, cutter = sdCuts, n = 6,
 cols = rev(brewer.pal(6, "RdYlBu")))
 choropleth(edos, var, shades)
 choro.legend(-118.5, 22.5, shades, under = "<", over
= ">",
 between = "a", x.intersp = -2.5, y.intersp = 0.9,
 box.lty = "blank", cex = 0.6)
 title(title)
 box(col = "black")
}
```

Ahora modificamos los parámetros del *plot* para poder acomodar bien: sin margen arriba, hacer un *grid* (un acomodo) de 2 x 4 y un poco de margen entre cada *plot* y hacemos un mapa para cada año y agregamos una leyenda:

```
op <- par(oma = c(0, 0, 0, 0), mfrow = c(2, 4), mar
= c(0, 0, 2, 0))
p2006 <- makeChoro(edos$A2006, "2006")
p2007 <- makeChoro(edos$A2007, "2007")
p2008 <- makeChoro(edos$A2008, "2008")
p2009 <- makeChoro(edos$A2009, "2009")
p2010 <- makeChoro(edos$A2010, "2010")
p2011 <- makeChoro(edos$A2011, "2011")
p2012 <- makeChoro(edos$A2012, "2012")
plot.new() # Hacerle creer a R que hacemos un nuevo
plot para avanzar en el grid
```

```
legend("center", fill = shades$cols, y.intersp = 1,
x.intersp = 1,
 ncol = 1, bty = "n", cex = 0.75,
 legend = c("Menos de -2 SD", "Entre -2 y -1 SD",
"Entre -1 y 0 SD",
 "Entre 0 y 1 SD", "Entre 1 y 2 SD", "Más de 2 SD"))
```



A partir de estos mapas, podemos ver que hay algunos estados de la República que tienen un número de homicidios por arriba del promedio anual en distintos años (por ejemplo, Chihuahua). Sin embargo, es un poco difícil comparar tanto los valores reales del número de homicidios como de las desviaciones estándar para distintos años. No obstante, esto nos da una indicación de que en Chihuahua algo pasa. Pero esto lo retomaremos más adelante.

Por lo pronto, vamos a hacer otros mapas de desviaciones estándar con otra librería.

4.2.4. Mapas de desviaciones estándar - ggplot2

Primero vamos a definir una función para asignarle una etiqueta a cada estado para ver en qué intervalo de desviaciones estándar se encuentra en cada año:

```
sdClass <- function(var){
 mean <- mean(var)
 sd <- sd(var)
  sd.vec  <-  vector(mode  =  “character”,length  =
length(var))

 sd.vec[var <= mean-2*sd] <- ‘<-2’ # var <= -2 SD
 sd.vec[var > mean-2*sd & var < mean-sd] <- ‘-2-1’ #
-2 SD < var <= -1 SD
 sd.vec[var > mean-sd & var <= mean] <- ‘-1-0’ # -1
SD < var <= 0 SD
 sd.vec[var > mean & var <= mean+sd] <- ‘0-1’ # 0 SD
< var <= 1 SD
 sd.vec[var > mean+sd & var < mean+2*sd] <- ‘1-2’ #
1 SD < var < 2 SD
 sd.vec[var >= mean+2*sd] <- ‘>2’ # var => 2 SD

 # Lo hacemos un factor para que, aunque no haya
elementos
 # en el intervalo de -2SD, exista y lo podamos usar:
 sd.vec <- factor(sd.vec, levels = c(‘<-2’, ‘-2-1’,
‘-1-0’, ‘0-1’,
 ‘1-2’, ‘>2’))
 return(sd.vec)
}
```

Para no confundirnos con los datos que trabajamos anteriormente, vamos a volver a leer los archivos y asignarlos a nuevas variables:

```
edos.gg <- readOGR(“.”, “estados_sorted”,
stringsAsFactors = FALSE,
 GDAL1_integer64_policy = T)
homicidios.gg <- read.csv(“homicidios.csv”,
 colClasses = c(rep(“character”, 1),
 rep(“integer”, 7)))
```

Ahora creamos un nuevo `data.frame` para esta nueva clasificación. Para ello, le copiamos los valores de las claves de los estados y ejecutamos la función que acabamos de definir para cada año:

```
# Definir data frame y copiar datos de clave de entidad:
sd.homicidios <- as.data.frame(homicidios.gg[,1])
# Renombrar la columna del nuevo data frame:
colnames(sd.homicidios) = c(‘CVE_ENT’)
# Ejecutar la función y clasificar cada año:
sd.homicidios$’2006’ <- sdClass(homicidios.gg$A2006)
sd.homicidios$’2007’ <- sdClass(homicidios.gg$A2007)
sd.homicidios$’2008’ <- sdClass(homicidios.gg$A2008)
sd.homicidios$’2009’ <- sdClass(homicidios.gg$A2009)
sd.homicidios$’2010’ <- sdClass(homicidios.gg$A2010)
sd.homicidios$’2011’ <- sdClass(homicidios.gg$A2011)
sd.homicidios$’2012’ <- sdClass(homicidios.gg$A2012)
```

Cargamos la librería:

```
library(ggplot2)
```

Y le unimos la clasificación que acabamos de hacer a los polígonos de los estados:

```
# Agregar clases de SD a los estados:
edos.gg <- merge(edos.gg, sd.homicidios, by.x = "CVE_ENT")
```

Ahora queremos cambiarles el orden a los datos para poder tener una lista *hacia abajo* en vez de *hacia la derecha*. Para esto, usamos la función `melt` de la librería `reshape2`:

```
# Cargar reshape2:
library(reshape2)
# Hacer el melt de las clases de SD:
sd.homicidios.melt <- melt(sd.homicidios, id = c("CVE_ENT"))
```

Toma un momento para ver la diferencia entre `sd.homicidios` y `sd.homicidios.melt`.

Lo malo de usar `ggplot` es que **necesita** usar un *data frame* de R. Si te fijas, nuestros estados son un `SpatialPolygonsDataFrame`. Hay que hacer algo al respecto. Usamos `fortify` para crear un *data frame* que contenga la geometría de los estados. Esto hace que perdamos muchos atributos, pero se los volvemos a pegar.

```
# Hacer un data frame de R:
edos_geom.gg <- fortify(edos.gg, region = "CVE_ENT")
# Volverle a pegar los datos que ya tenía:
edos_geom.gg <- merge(edos_geom.gg, edos.gg@data,
by.x = "id",
 by.y = "CVE_ENT")
```

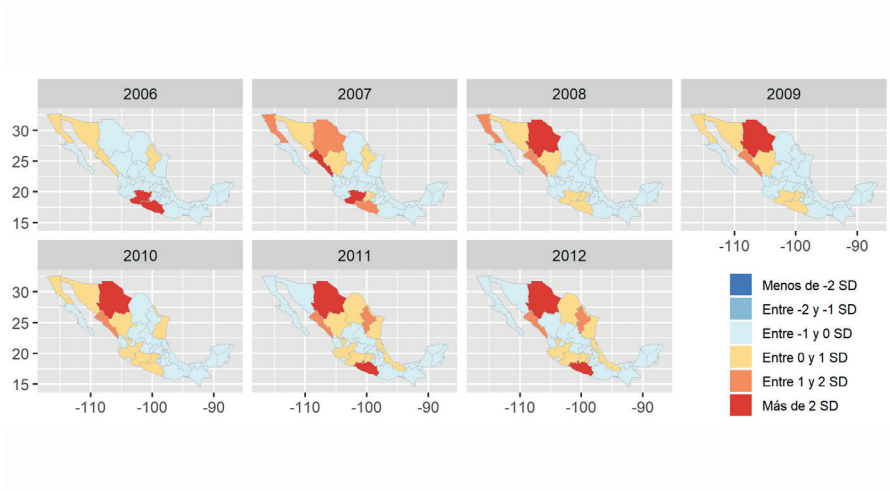
Esto provoca que cada estado se separe en segmentos y que tenga sus atributos. Ahora le pegamos los datos de la clasificación de los intervalos de desviaciones estándar:

```
# Pegarle los datos de las clasificaciones de SD a
cada segmento:
plot.data.gg <- merge(edos_geom.gg, sd.homicidios.
melt, by.x = "id",
 by.y = "CVE_ENT")
```

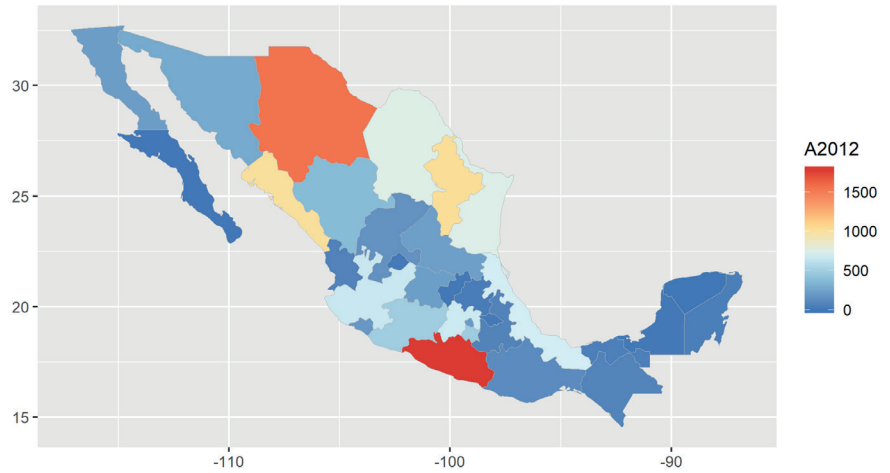
Todo este procedimiento parece inútil, pero es necesario para poder usar una función poderosa de `ggplot` que nos permita generar todos los mapas con una sola instrucción:

```
# Volvemos a hacer que la clasificacón de las SD sea
un factor, porque se
# perdió al hacer el melt:
plot.data.gg$value <- factor(plot.data.gg$value,
 levels = c('<-2', '-2-1', '-1-0', '0-1',
 '1-2', '>2'))
```

```
# Definir el objeto de ggplot:
ggplot(data = plot.data.gg,
 aes(x = long, y = lat, fill = value, group = group)) +
# Agregar la geometría de los polígonos, colorear los
bordes y aspecto 1:1
 geom_polygon() + geom_path(colour = “grey”, lwd =
0.1) + coord_equal() +
# Hacer el facet wrap con 4 columnas:
 facet_wrap(~variable, ncol = 4) +
# Agregar colores, leyenda y quitar nombres de los
ejes:
 scale_fill_brewer(palette = “RdYlBu”, direction = -1,
name = “”,
  breaks = c(‘<-2’, ‘-2-1’, ‘-1-0’, ‘0-1’, ‘1-2’,
‘>2’),
 labels = c(‘Menos de -2 SD’, ‘Entre -2 y -1 SD’,
 “Entre -1 y 0 SD”, “Entre 0 y 1 SD”,
 “Entre 1 y 2 SD”, “Más de 2 SD”),
 drop = FALSE, guide = “legend”) +
 labs(x = NULL, y = NULL) +
# Modificar en donde aparece la leyenda:
 theme(legend.position = c(0.89, 0.13), legend.key.
size = unit(5, “mm”),
 legend.text = element_text(size = 8),
 legend.margin = margin(t = -1, unit = ‘cm’))
```

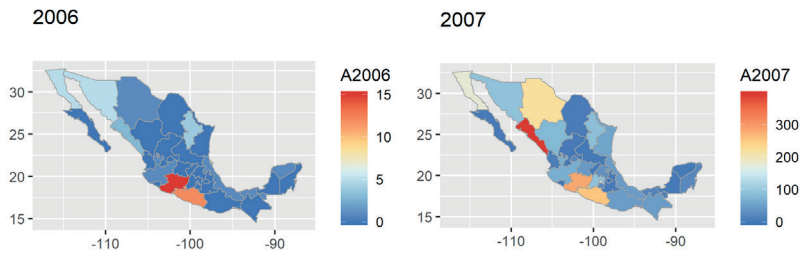


Claro, puedes hacer un solo mapa de desviaciones estándar con `ggplot`. Pero eso se queda de tarea.



Si te parece muy complicado trabajar los datos para hacer un *facet wrap*, hay otra forma de hacer los mapas en un *grid* con `ggplot`.

```
# Definir plot para un año:
p2006 <- ggplot(edos_geom.gg2, aes(x = long, y = lat,
group = group)) +
 geom_polygon(aes(x = long, y = lat, group = group,
fill = A2006),
 color = “dark grey”, size = 0.3) + coord_equal() +
   scale_fill_gradientn(colors   =   rev(brewer.pal(6,
‘RdYlBu’))) +
 theme(legend.position = “right”) +
 labs(x = NULL, y = NULL, title = “2006”, subtitle =
“”, caption = “”)
# Definir plot para otro año:
p2007 <- ggplot(edos_geom.gg2, aes(x = long, y = lat,
group = group)) +
 geom_polygon(aes(x = long, y = lat, group = group,
fill = A2007),
 color = “dark grey”, size = 0.3) + coord_equal() +
   scale_fill_gradientn(colors   =   rev(brewer.pal(6,
‘RdYlBu’))) +
 theme(legend.position = “right”) +
 labs(x = NULL, y = NULL, title = “2007”, subtitle =
“”, caption = “”)
# Cargar un par de librerías:
library(grid)
library(gridExtra)
# Hacer el grid:
grid.arrange(p2006, p2007, ncol = 2)
```



Un inconveniente es que se repiten las leyendas y `ggplot` no tiene una función específica para hacer que todas las gráficas compartan la misma leyenda. Se puede hacer, pero es un poco complicado. Sin embargo, podrías hacer un *facet wrap* para histogramas de cada año que muestren la distribución de los homicidios por estado.

Pero por ahora, regresemos al objetivo que teníamos antes de analizar la información de manera visual: identificar si hay autocorrelación espacial.

4.2.5. Adyacencia y estadística espacial

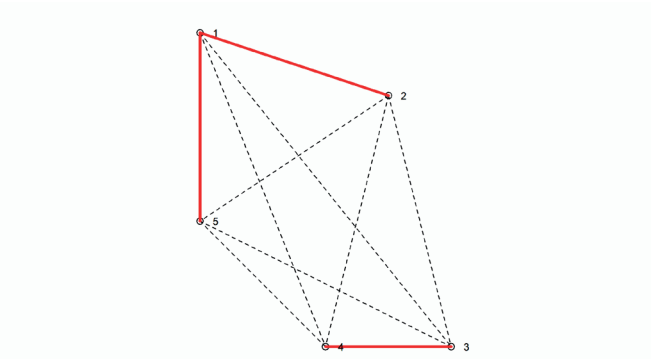
Vamos a definir la adyacencia de polígonos. Para ello, primero veremos un ejemplo sencillo al usar puntos y después extenderemos el concepto a polígonos. Posteriormente seguiremos con el análisis de la estadística espacial de los homicidios en los estados.

Cargamos la librería `spdep`:

```
library(spdep)
```

4.2.6. Matriz de adyacencia

Una matriz de adyacencia es una manera de expresar si dos entidades son contiguas o no. La manera más fácil de codificar esto es con una matriz de entradas binarias: 0 o 1. Muchas veces estas matrices son simétricas (¡pero no siempre!) pues si A es vecino de B, B suele ser vecino de A. Por ejemplo, considera el siguiente arreglo de puntos:



Los puntos 1 y 2 son vecinos, al igual que el 3 y el 4 y el 5 y el 1. En forma de matriz, esto lo podemos expresar como:

$w_{ij} \rightarrow$

	1	2	3	4	5
1	0	**1**	0	0	**1**
2	**1**	0	0	0	0
3	0	0	0	**1**	0
4	0	0	**1**	0	0
5	**1**	0	0	0	0

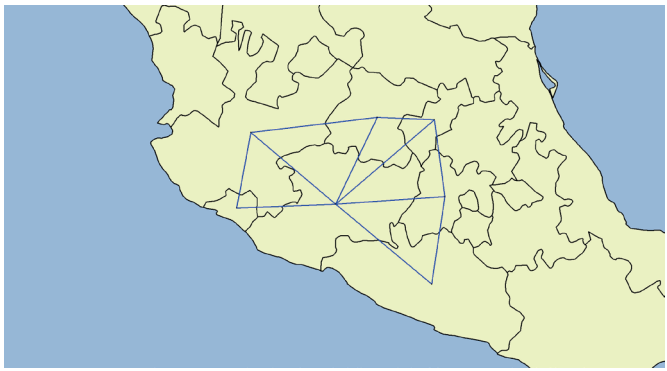
Los s y **1**s son los **pesos**, que en este caso son binarios e indican si hay adyacencia o no. Esta matriz se puede transformar a una que se llama *estandarizada por renglones*. Esto significa sumar los pesos

y dividirlos entre el total de la columna. Nuestra matriz anterior estandarizada por renglón se ve así:

$w_{ij} \rightarrow$

	1	2	3	4	5
1	0	**0.5**	0	0	**0.5**
2	**1**	0	0	0	0
3	0	0	0	**1**	0
4	0	0	**1**	0	0
5	**1**	0	0	0	0

Si definimos la adyacencia con base a los vecinos de orden mayor, las matrices pueden no ser simétricas. Por ahora, nos vamos a concentrar en adyacencia de primer orden de polígonos. Podemos pensar varias formas en que los polígonos podrían ser vecinos, pero las más comunes son: adyacencia tipo **reina** y tipo **torre**. Por ejemplo, consideremos solo los vecinos de Michoacán:



Aquí podemos ver que **todos** los polígonos que están *pegados* a Michoacán están catalogados como sus vecinos. Ahora tenemos que hacer lo mismo, pero para los 32 estados de la República. Para definir

los vecinos de contigüidad para los estados como una lista, utilizamos el siguiente código:

```
# Construir la lista de vecinos:
edos.nbq <- poly2nb(edos, queen = T) # TRUE: tipo reina
# Convertir la lista de vecinos en una lista de pesos estandarizados por renglón:
edos.nbq.w <- nb2listw(edos.nbq)
```

Si quieres ver información acerca de la lista de pesos, puedes usar:

```
summary.nb(edos.nbq)
```

4.2.7. Autocorrelación espacial global y local

Queremos ver si hay alguna indicación de que las observaciones se acumulan en cierta región, o si estamos ante la presencia de un proceso aleatorio. Podemos calcular la *I* de Moran para algún año en particular, digamos 2012.

```
# Definimos una variable para no repetir todo el tiempo:
var <- edos$A2012
moran.test(var, edos.nbq.w)

 Moran I test under randomisation
 data: var
 weights: edos.nbq.w
```

```
 Moran I statistic standard deviate = 1.4612, p-value
= 0.07199
 alternative hypothesis: greater
 sample estimates:
 Moran I statistic Expectation Variance
 0.14045922 -0.03225806 0.01397264
 Moran I test under randomization


 data: var
 weights: edos.nbq.w


 Moran I statistic standard deviate = 1.5111, p-value
= 0.06538
 alternative hypothesis: greater
 sample estimates:
 Moran I statistic Expectation Variance
 0.14808294 -0.03225806 0.01424226
```

Como tiene un valor de `0.14` bastante más alto que el valor esperado, es un indicador de que estamos ante la presencia de autocorrelación espacial global. Ahora calculamos la *I* de Moran local:

```
# Calcular la I de Moran local
lmoran <- localmoran(var, edos.nbq.w)
```

Puedes ver información acerca de la *I* local con `summary(lmoran)`.

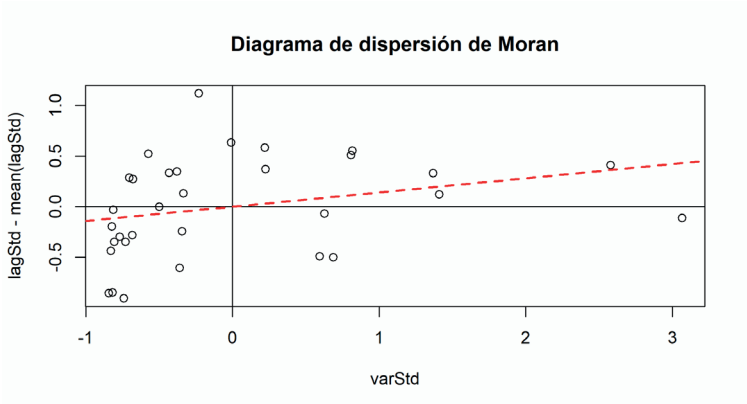
4.2.8. Diagrama de dispersión de Moran

Lo primero que hacemos es estandarizar las variables:

```
lagStd <- lag.listw(edos.nbq.w, scale(var)) # usamos
la variable estandarizada
varStd <- (var - mean(var))/sd(var)
# Es lo mismo que as.vector(scale(var))
```

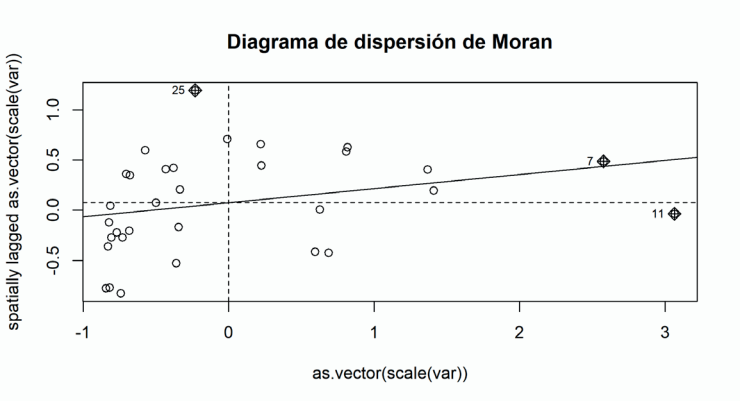
Posteriormente hacemos la gráfica:

```
plot(varStd,lagStd-mean(lagStd))
# Ejes que pasan por el origen:
abline(h = 0, v = 0)
# Recta de ajuste lineal entre las dos variables:
#abline(lm(lagStd ~ varStd), lty = 2, lwd = 2, col =
“red”)
abline(lm(lagStd - mean(lagStd) ~ varStd), lty = 2,
lwd = 2, col = “red”)
title(“Diagrama de dispersión de Moran”)
```



O más rápido, usamos la función `moran.plot()`

```
# Diagrama de dispersión
moran.plot(as.vector(scale(var)), edos.nbq.w)
title("Diagrama de dispersión de Moran")
```



Como resultado, vemos que la pendiente es positiva, pues coincide con el valor de la *I* global.

4.2.9. Mapa de cúmulos

Definimos una significancia para nuestros mapas:

```
# Definir significancia:
significancia <- 0.05
```

y un vector para almacenar el cuadrante en el que se encuentra cada punto en el diagrama de Moran:

```
# Definir vector de cuadrante del plot de Moran:
cuadrante <- vector(mode = "numeric",length =
length(var))
```

En el diagrama de dispersión de Moran, el eje x es la variable varStd y el eje y es lagStd. Así que encontramos en cuál cuadrante está cada punto:

```
# Definición de cúmulos:
cuadrante[varStd > 0 & lagStd > 0] <- 1 # Altos-Altos
cuadrante[varStd < 0 & lagStd < 0] <- 2 # Bajos-Bajos
cuadrante[varStd < 0 & lagStd > 0] <- 3 # Bajos-Altos
cuadrante[varStd > 0 & lagStd < 0] <- 4 # Altos-Bajos
cuadrante[lmoran[,5] > significancia] <- 0 # No
significativos
```

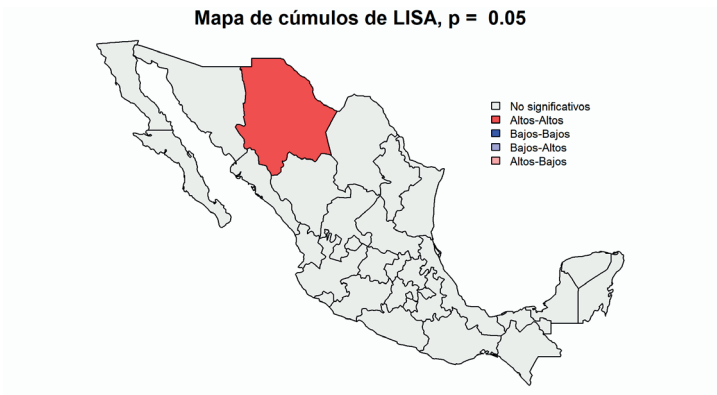
Y definimos los colores que vamos a usar:

```
# Colores para los cúmulos:
cColors <- c(rgb(0.74, 0.74, 0.74, alpha = 0.2),
rgb(1, 0, 0, alpha = 0.75),
 rgb(0, 0, 1, alpha = 0.75), rgb(0.58, 0.58, 1, alpha
= 0.75),
 rgb(1, 0.58, 0.58, alpha = 0.75))
# gris (no significativos), rojo (altos-altos), azul
(bajos-bajos), azul
# claro (bajos-altos) y rojo claro (altos-bajos)
```

El mapa de cúmulos:

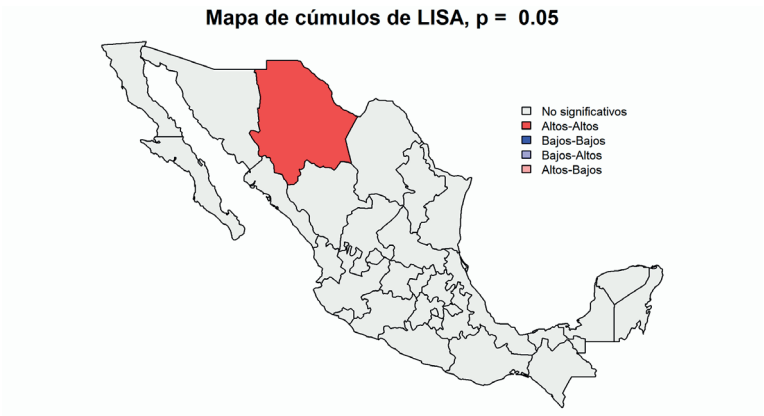
```
# Definir márgenes para ocupar todo el espacio:
par(mar = c(0, 0, 1, 0))
# Primer mapa. Pintar no significativos:
plot(edos[cuadrante == 0, ], col = cColors[1], pch =
16, cex = 0.75)
```

```
# Pintar altos-altos:
plot(edos[cuadrante == 1, ], col = cColors[2], add = T, pch = 16, cex = 0.75)
# Pintar bajos-bajos:
plot(edos[cuadrante == 2, ], col = cColors[3], add = T, pch = 16, cex = 0.75)
# Pintar bajos-altos:
plot(edos[cuadrante == 3, ], col = cColors[4], add = T, pch = 16, cex = 0.75)
# Pintar altos-bajos:
plot(edos[cuadrante == 4, ], col = cColors[5], add = T, pch = 16, cex = 0.75)
legend(-95, 30, fill = cColors, y.intersp = 1, x.intersp = 1,
 legend = c("No significativos", "Altos-Altos", "Bajos-Bajos",
 "Bajos-Altos", "Altos-Bajos"), bty = "n", cex = 0.7)
title(paste("Mapa de cúmulos de LISA, p = ", significancia))
```



O de una forma más rápida, pintamos todos con una sola instrucción:

```
# Definir márgenes para ocupar todo el espacio:
par(mar = c(0, 0, 1, 0))
intervalos <- c(0, 1, 2, 3, 4)
plot(edos, col = cColors[findInterval(cuadrante, intervalos)])
legend(-95, 30, fill = cColors, y.intersp = 1, x.intersp = 1,
 legend = c(“No significativos”, “Altos-Altos”, “Bajos-Bajos”,
 “Bajos-Altos”, “Altos-Bajos”), , bty = “n”, cex = 0.7)
title(paste(“Mapa de cúmulos de LISA, p = “, significancia))
```



4.2.10. Pseudo-significancia y permutaciones

Veamos qué tan confiable es el resultado de la *I* local que obtuvimos. Para esto, podemos hacer permutaciones de las cuentas observadas en cada estado para ver qué tan diferente es de una distribución normal, pero manteniendo las ubicaciones espaciales. Definimos un número de permutaciones a realizar:

```
# Definir el número de simulaciones:
sims <- 999
```

Definimos una matriz y un vector para guardar los resultados de las simulaciones.

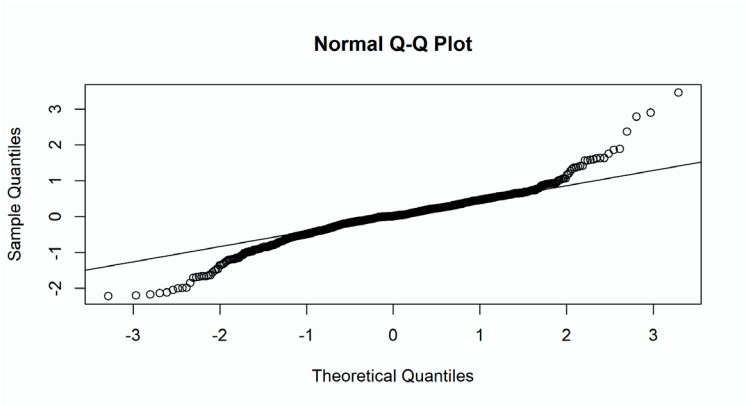
```
# Definir una matriz y un vector para guardar dos resultados. La I local simulada:
sim.I <- matrix(0, sims, 32)
# Los p-values simulados:
sim.p <- matrix(0, 1, 32)
```

Y encontramos los valores de la *I* local para cada simulación:

```
# Encontrar la I de Moran local para cada simulación:
for(i in 1:sims){
 sim.I[i,] <- localmoran(sample(var), edos.nbq.w)[,1]
}
```

Podemos hacer una gráfica QQ para ver si nuestras simulaciones se parecen, o no, a una distribución normal:

```
# QQ plots para los polígonos:
qqnorm(sim.I[,1])
qqline(sim.I[,1])
```



Queremos comparar simulaciones de los datos con el valor observado y contar cuántas son mayores o menores que el valor local observado. La pseudo-significancia la podemos definir como:

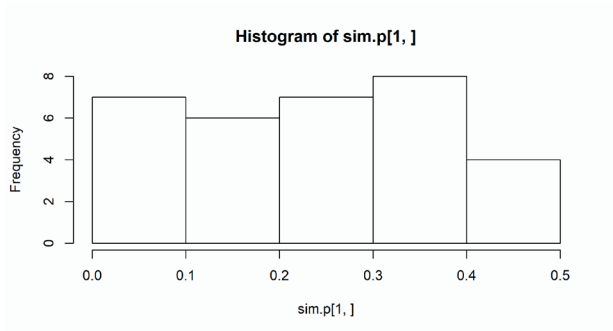
$$p = \frac{M+1}{R+1}$$

donde M es el número de veces que la estadística simulada es mayor (menor) o igual que la observada y es el número de simulaciones.

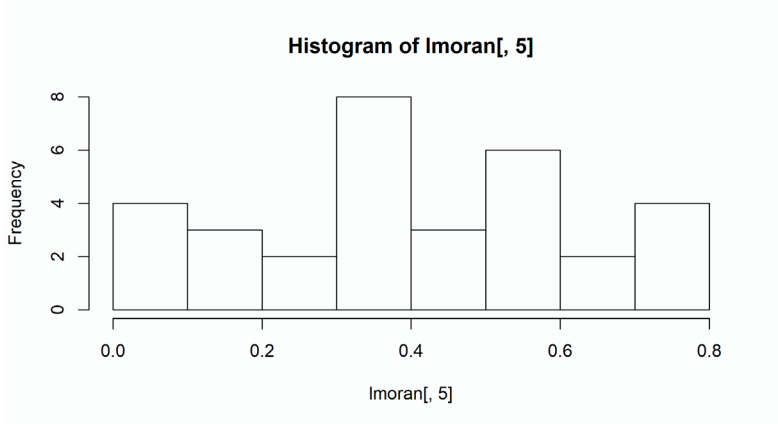
```
# Calcular el valor p definido arriba:
for(i in 1:32){
        ifelse(lmoran[i,1]>0,        larger        <-
sim.I[,i]>=lmoran[i,1],
 larger <- sim.I[,i]<=lmoran[i,1])
 sim.p[i] <- (sum(larger == T)+1)/(sims+1)
}
```

Con esto, podemos comparar la distribución de las significancias para la *I* de Moran observada y las simuladas:

```
# Comparar los histogramas de los p-values:
hist(sim.p[1,])
```




```
hist(lmoran[,5])
```



Y por último, podemos hacer un mapa de cúmulos tomando en cuenta estas pseudo-significancias. Como antes, definimos un vector para los cuadrantes de los valores simulados:

```
# Definir vector de cuadrante del plot de Moran de las
simulaciones:
```

```
cuadrante.sim <- vector(mode = “numeric”, length =
length(var))
```

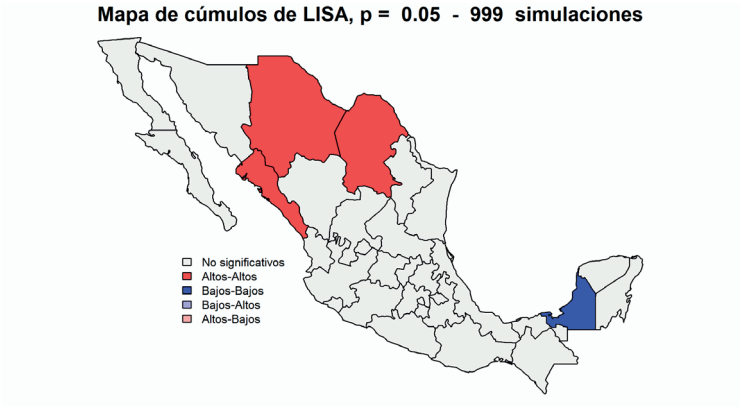
Encontramos en cuál cuadrante está cada punto:

```
# Definición de cúmulos:
cuadrante.sim[varStd > 0 & lagStd > 0] <- 1 # Altos-
Altos
cuadrante.sim[varStd < 0 & lagStd < 0] <- 2 # Bajos-
Bajos
cuadrante.sim[varStd < 0 & lagStd > 0] <- 3 # Bajos-
Altos
cuadrante.sim[varStd > 0 & lagStd < 0] <- 4 # Altos-
Bajos
cuadrante.sim[sim.p > significancia] <- 0 # No
significativos
```

Y hacemos el mapa de las simulaciones:

```
# Definir márgenes para ocupar todo el espacio:
par(mar = c(0, 0, 1, 0))
plot(edos, col = cColors[findInterval(cuadrante.sim,
intervalos)])
legend(-113, 22, fill = cColors, , y.intersp = 1,
x.intersp = 1,
 legend = c(“No significativos”, “Altos-Altos”, “Bajos-
Bajos”,
 “Bajos-Altos”, “Altos-Bajos”), , bty = “n”, cex =
0.7)
```

```
title(paste("Mapa de cúmulos de LISA, p = ", significancia, " - ", sims, " simulaciones"))
```

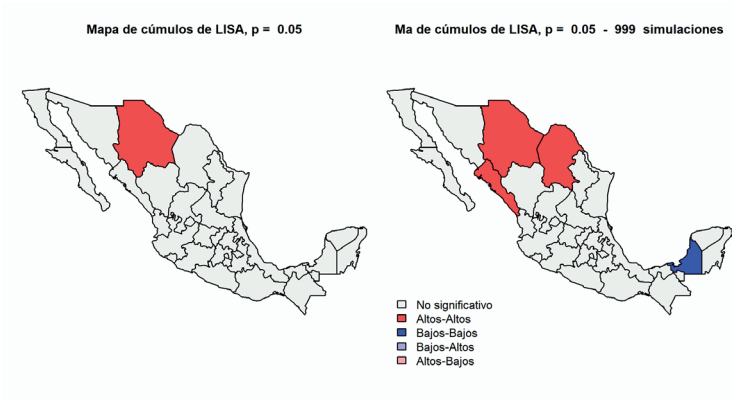


Para comparar los dos mapas de cúmulos, los podemos poner juntos:

```
# Definir que tendremos una gráfica de 1 renglón y dos columnas:
par(mfrow = c(1, 2))
# Sin margen arriba, un poco de margen entre cada plot y definir tipo de
# región cuadrada:
op <- par(oma = c(0, 0, 0, 0), mar = c(0, 1, 0, 0), pty = "s")

# Primer mapa:
plot(edos, col = cColors[findInterval(cuadrante, intervalos)],
 xaxs = "i", yaxs = "i")
title(paste("Mapa de cúmulos de LISA, p = ", significancia), cex.main = 0.75)
```

```
# Segundo mapa:
plot(edos, col = cColors[findInterval(cuadrante.sim,
intervalos)],
 xaxs = "i", yaxs = "i")
legend("bottomleft", fill = cColors, y.intersp = 1,
x.intersp = 1,
 legend = c("No significativo", "Altos-Altos", "Bajos-
Bajos",
 "Bajos-Altos", "Altos-Bajos"), bty = "n", cex = 0.7)
title(paste("Ma de cúmulos de LISA, p = ",
significancia, " - ",
 sims, " simulaciones"), cex.main = 0.75)
```




```
# Regresar a una sola gráfica en toda la página
par(mfrow = c(1,1))
```

Capítulo 5. Ciencias Sociales: Análisis de contenido y uso de RQDA

Ana Yésica Martínez Villalba
Escuela Nacional de Estudios Superiores
Universidad Nacional Autónoma de México, unidad Morelia

Ayari Pasquier Merino
Secretaría de Desarrollo Institucional
Universidad Nacional Autónoma de México

Introducción

El análisis cualitativo ha adquirido creciente importancia en los últimos años y actualmente es utilizado como estrategia de investigación en una gran diversidad de disciplinas. Este tipo de análisis puede ser desarrollado a través de distintos enfoques, como la etnografía, la teoría fundamentada, la fenomenología, entre otros.

El análisis cualitativo utiliza diversas herramientas, entre éstas una de las más conocidas es el análisis de contenido. De manera general, se puede decir que esta estrategia de análisis tiene como objetivo interpretar la información presente en un conjunto de documentos a partir de un proceso sistemático de clasificación, buscando comprender las relaciones entre las unidades temáticas identificadas que resultan significativas para la investigación a través de un ejercicio de progresiva abstracción.

Este capítulo presenta las características centrales del análisis de contenido e introduce al lector en el uso de RQDA, un paquete de R especialmente diseñado para el análisis cualitativo de documentos de texto que, al estar asociado a la plataforma R, permite hacer análisis estadístico de la codificación usando las funciones de R.[1]

Si bien es cierto que el análisis de contenido ha sido utilizado sobre todo en el marco de las ciencias sociales, su uso se ha extendido a otras disciplinas y resulta una herramienta útil para estudiantes e investigadores de cualquier rama del conocimiento interesados en el análisis cualitativo, siendo una herramienta particularmente útil para el análisis de textos como entrevistas, artículos periodísticos, discursos públicos, etcétera.

Este tipo de análisis suele ser utilizado para precisar conceptos que no pueden ser definidos de manera precisa *a priori* y que no pueden ser medidos a partir de instrumentos preexistentes –como lo haría una encuesta–. En general, se trata de una estrategia óptima de análisis para aquellas investigaciones que buscan conocer las definiciones dadas a un fenómeno por los actores involucrados, describir cómo opera un fenómeno social o desarrollar conceptos teóricos.

Los datos cualitativos no tienen una estructura o significado intrínseco y el investigador debe crear una estructura para organizar los datos, esta estructura está basada en las preguntas de la investigación y se desarrolla a partir de distintas etapas de análisis, articulando progresivamente un esquema que permita convertir los datos en una explicación.

[1] Huang, R. (2016). *What is RQDA and what are its features?* Recuperaado de http://rqda.r-forge.r-project.org/

El análisis de contenido suele ser considerado como un método flexible para analizar datos de texto, sin embargo, se trata de una herramienta compleja que es necesario utilizar con rigor y sistematicidad para poder garantizar la validez de sus resultados.

Existen distintas estrategias para llevar a cabo un análisis cualitativo de contenido, que varían según los enfoques analíticos, los intereses de investigación y el tipo de preguntas planteadas. En cualquier caso, es importante subrayar que cualquier investigación parte de categorías preconcebidas vinculadas con la formación disciplinar, la teoría desde donde se plantean las preguntas de investigación, las cosmovisiones personales o incluso los prejuicios sociales. Estos factores funcionan como "filtros" y pueden incidir en la definición de las preguntas, la selección primaria del material, las unidades de análisis, el análisis mismo y su interpretación, introduciendo sesgos en la investigación.

Para llevar a cabo un análisis sistemático del material empírico es necesario tener en cuenta un conjunto de estrategias que permiten mejorar la validez de los estudios cualitativos, estas incluyen:

1. Discutir los enfoques teóricos desde donde se articularon el problema y planteamiento de la investigación.
2. Hacer explícitos los criterios utilizados para el desarrollo de las preguntas, la recolección de los datos y la definición de las primeras categorías de análisis.
3. Describir con precisión las condiciones en que fue desarrollada la investigación.
4. Explicar por qué es significativo el material recopilado y cuáles podrían ser sus particularidades.
5. Realizar observaciones prolongadas y tener un buen conocimiento del contexto donde se trabaja.
6. Trabajar con muestreos intencionales definidos con cuidado y buscar llegar a un punto de "saturación" en la recopilación de los datos.

7. Utilizar estrategias de triangulación en el análisis.
8. Contar con la validación de los actores locales y expertos en la temática analizada.

También resultan centrales la contextualización de los datos y la discusión de la literatura sobre el tema específico del que se esté hablando, dando cuenta de los casos que sostienen o contradicen las relaciones teóricas propuestas y los resultados de la investigación.

Para desarrollar un análisis se pueden utilizar diferentes estrategias, las cuales se sitúan en un continuo entre un análisis de contenido dirigido y un análisis basado en categorías emergentes. El análisis de contenido dirigido se utiliza comúnmente para validar, desarrollar o ilustrar un esquema teórico–conceptual previamente establecido. Este tipo de investigaciones se basan en trabajos previos que guían la definición de las preguntas de investigación, la identificación de variables y sus posibles relaciones, lo cual facilita el esquema primario de codificación. Por su parte, en el caso de las categorías emergentes, lo que interesa es dejar que los datos "hablen" por sí mismos, sin necesariamente "aprisionarlos" en categorías teóricas establecidas *a priori*.

En las investigaciones cualitativas, la información suele ser recopilada a través de entrevistas semiestructuradas diseñadas a partir de las variables centrales planteadas por la teoría. En el otro extremo encontramos investigaciones basadas en entrevistas abiertas y el análisis se lleva a cabo a partir de distintas etapas de lectura del material en las que se busca identificar los temas más relevantes para los informantes y los significados que les atribuyen. La mayor parte de las investigaciones se sitúa en un punto intermedio entre estos

dos extremos y combina ambas estrategias, basando la codificación primaria en el planteamiento de la investigación y dejando espacio abierto para la identificación de temas emergentes a lo largo del proceso de análisis. En cualquiera de los casos es fundamental llevar a cabo la codificación de manera sistemática, misma que va siendo redefinida y precisada a lo largo del proceso de análisis.

El presente capítulo tiene como objetivo describir algunas estrategias que pueden ser utilizadas para establecer esquemas de codificación para el análisis de contenido. Asimismo, pretende ilustrar el uso del paquete RQDA como una herramienta de apoyo en el proceso de análisis cualitativo de datos.

Para lograr lo anterior, este capítulo incluye apartados teóricos y apartados prácticos. En los primeros se hace una breve descripción de ciertas estrategias de categorización, codificación y análisis de textos. Por otro lado, en los apartados prácticos se muestra cómo funciona el paquete RQDA usando un ejemplo de un proyecto en el que se pretende hacer análisis de los discursos ambientales de la ONU entre 1972 y 2012. Cada apartado práctico contiene instrucciones paso a paso para acceder al ejemplo y para el manejo básico de RQDA que esperamos sean de utilidad para quienes consulten el presente manual.

5.1. El análisis de contenido en las ciencias sociales

En las ciencias sociales la investigación cualitativa se ha desarrollado desde enfoques como la etnografía, la teoría fundamentada, la fenomenología o la investigación histórica. En las últimas décadas se observa un interés cada vez mayor por hacer análisis de contenido en estas y otras disciplinas, siendo identificado como una herramienta útil para examinar datos en formato de texto y materiales audiovisuales.

Cabe señalar que este capítulo se enfoca en el análisis de contenido de textos, que es una de las principales aplicaciones del paquete de análisis cualitativo RQDA.

El análisis de contenido es una herramienta que nos ayuda en el proceso de organización e interpretación de materiales empíricos en distintos formatos y puede ser definida como un proceso sistemático que nos permite:

- Clasificar fragmentos de información en categorías significativas para la investigación.
- Reconocer las relaciones entre las categorías identificadas.

Dicho de otra forma, el análisis de contenido nos permite ordenar la información empírica en categorías de manera sistemática y convertir una gran cantidad de datos en un número limitado de proposiciones encaminadas a describir y explicar nuestro objeto de estudio. Sin embargo, el proceso de análisis cualitativo es complejo y muchas veces puede resultar ambiguo, pues se suelen utilizar conceptos que no están definidos *a priori* de manera precisa y muchas veces no pueden ser medidos a partir de instrumentos preexistentes.

Este tipo de análisis puede ser de gran utilidad, pues permite explicitar los conceptos clave de la investigación cualitativa. Generalmente, el análisis de contenido está guiado por una estructura que el investigador propone a partir de sus preguntas de investigación y que utiliza para organizar los datos obtenidos en el proceso de pesquisa y convertirlos en una explicación plausible del problema que indaga.

Existen distintas estrategias y técnicas de análisis según los enfoques analíticos, los intereses de la investigación y el tipo de

preguntas planteadas. A grandes rasgos, para organizar los datos se pueden seguir las siguientes estrategias:

1. Juntar los fragmentos de información vinculados con un tema, buscar las relaciones entre estos fragmentos para formar unidades más amplias y examinar los vínculos entre éstas para llegar a proposiciones más generales.
2. Basarse en un modelo predefinido.

Las estrategias utilizadas varían principalmente según si los esquemas de codificación son de tipo teórico o empírico. Hsieh y Shannon[2] distinguen tres tipos de análisis:

a) **Convencional**: las categorías de codificación se derivan directamente de los textos analizados.
b) **Dirigido**: se establecen las categorías primarias en función de la teoría.
c) **Sumatorio**: se cuentan y comparan palabras clave, para después interpretar el contexto de su uso.

5.1.1. Análisis de contenido convencional

En este tipo de análisis los datos son recolectados principalmente a través de entrevistas abiertas, buscando maximizar la diversidad de la información y evitando utilizar categorías preestablecidas, pues lo que busca este tipo de análisis es que las categorías "emerjan" de los datos. En estos casos, el análisis está enfocado en mostrar la complejidad de un fenómeno y es útil para hacer una descripción detallada de éste.

[2] Hsieh, H.-F., y Shannon, S. E. (2005). Three Approaches to Qualitative Content Analysis. *Qualitative Health Research*, *15*(9), 1277–1288. https://doi.org/10.1177/1049732305276687

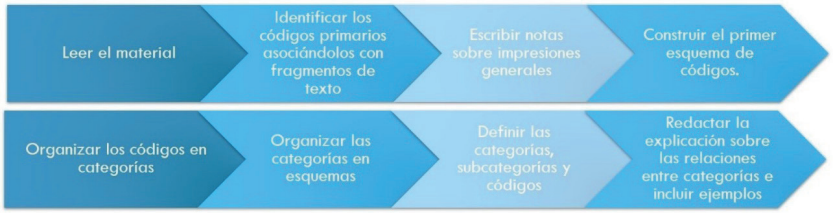


Figura 1. Fases del análisis de contenido convencional (Hsieh y Shannon, 2005).

Hsieh y Shannon llaman a este proceso inducción analítica pues busca generar categorías a partir de datos empíricos a través de un proceso de abstracción de los elementos esenciales observados en un caso concreto. Este proceso parte del supuesto de que la información se genera "directamente" de los informantes, sin imponer categorías preconcebidas.

Esta perspectiva de análisis es cercana a la "teoría fundamentada" y a la "fenomenología" y se utiliza sobre todo para el desarrollo de conceptos. Su validez se fundamenta en la observación prolongada del fenómeno que se estudia y en un alto grado de involucramiento con un tema y con el contexto en el cual se desarrolla la investigación. También se usan estrategias de triangulación de información y validación de los resultados con los actores que participaron en la investigación.

La validez de los resultados de este tipo de análisis depende en gran medida del carácter lógico y sistemático de los criterios utilizados para tomar cada una de las decisiones del proceso de investigación. Para mejorar las condiciones de validez del análisis se recomienda:

1. Discutir los enfoques teóricos de partida y hacer explícitos los criterios utilizados para el desarrollo de preguntas, la recolección de datos y la definición de las primeras categorías de análisis.
2. Organizar el análisis en diversas etapas.

5.1.2. Análisis de contenido dirigido

El análisis de contenido dirigido tiene como principal objetivo validar, desarrollar o ilustrar un esquema teórico-conceptual. Este tipo de análisis está guiado por la teoría e investigaciones previas, mismas que sirven como marco para la definición de preguntas de investigación, de las variables y de las relaciones entre ellas. En este caso, la definición del esquema primario de categorías y códigos para el análisis de textos está también guiada por el marco teórico-conceptual.

El proceso de este tipo de análisis es más estructurado. Los datos son recolectados principalmente a través de entrevistas semiestructuradas diseñadas a partir de las categorías propias de la teoría en la cual el trabajo de investigación se fundamenta. Sin embargo, también hay espacio para "categorías emergentes", las cuales ayudan en la identificación de elementos que respaldan o contradicen a la teoría.

Figura 2. Fases del análisis de contenido dirigido (Hsieh & Shannon, 2005).

Debido a que es poco probable que se obtengan datos que puedan ser comparados usando pruebas estadísticas, cuando resulta significativo, en el análisis de contenido se hacen comparaciones de frecuencia de códigos, lo cual da cuenta de la incidencia de ciertos códigos o se destaca el porcentaje de los códigos-testimonios que respaldan una hipótesis en comparación con otra. Los principales riesgos del análisis de contenido dirigido es que se pueden priorizar los postulados de la teoría sin considerar aspectos contextuales relevantes para el fenómeno de estudio.

5.1.3. Análisis de contenido sumatorio

Este tipo de análisis se utiliza para analizar el uso contextual de ciertas palabras o expresiones. El objetivo es comprender el significado subyacente de una palabra o expresión.

Figura 3. Análisis de contenido sumatorio (Hsieh & Shannon, 2005).

Aspectos comunes

Todos los enfoques comparten un proceso analítico similar. El éxito de un análisis de contenido depende en gran medida del proceso de codificación, esto es, organizar un gran número de fragmentos de texto en un número limitado de categorías. Para llevar a cabo la codificación de manera sistemática es necesario desarrollar un esquema de codificación que establezca las reglas del análisis de los datos, aunque los esquemas de codificación son redefinidos y precisados a lo largo de la

investigación, este es un paso fundamental para garantizar la validez del análisis.

Figura 4. Fases comunes en el análisis de contenido (Hsieh & Shannon, 2005).

5.1.4. El papel de la teoría en el análisis de contenido

A decir de Weber la investigación no "revela" la relación "real" entre las cosas, sino las relaciones conceptuales entre las ideas y los conceptos. El lugar y la función de los conceptos en la investigación social empírica no siempre son explícitos, pero es indudable que las categorías de pensamiento determinan la observación y que las observaciones cualitativas adquieren sentido cuando los datos son ordenados de manera reflexiva y en permanente diálogo con los conceptos teóricos de los que parte el investigador.

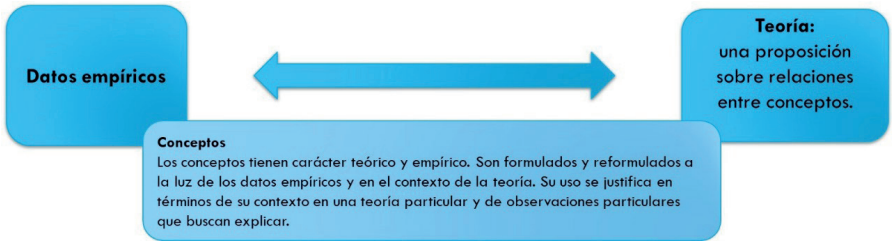


Figura 5. Conceptos como mediadores entre los datos empíricos y la teoría. Elaboración propia.

Los conceptos nos permiten identificar relaciones entre los fenómenos que observamos y definir categorías de análisis. Por su parte, las teorías son las que proveen los patrones dentro de los cuales

los datos cobran sentido. En términos generales, las observaciones están cargadas de teoría, pues nos acercamos a la realidad con supuestos sobre el significado social de los gestos y los espacios que son observados.

Tomando en cuenta lo anterior, en el análisis de contenido se parte de la idea de que los conceptos no son desarrollados a partir de las observaciones, pero tampoco deben ser impuestos *a priori*. Los conceptos siempre deben de estar justificados en términos del contexto teorético del que parten, tanto como de las características particulares de la realidad que se busca explicar.

5.1.5. Fuentes y organización de material

El análisis de contenido es una herramienta que puede ser utilizada para analizar distintos tipos de material, este capítulo se enfoca en el análisis de contenido de textos, que es uno de los formatos utilizados con mayor frecuencia. Los textos incorporados al análisis pueden provenir de distintas fuentes, por ejemplo, narrativas producto de entrevistas, respuestas a preguntas abiertas en cuestionarios, conversaciones hechas en grupos focales, narración de las observaciones del investigador, notas de prensa, artículos científicos y un largo etcétera.

Una de las herramientas más comunes para obtener textos que luego serán analizados son las entrevistas cualitativas, las cuales pueden ser definidas como conversaciones controladas por el investigador para que sus informantes se expresen de manera dialógica. Hay varios tipos de entrevistas: estructurada, semiestructurada, enfocada y no estructurada. La diferencia radica en qué tan dirigidas y cerradas son las preguntas, siendo la estructurada la más rígida y la no estructurada la más flexible.

Figura 6. Tipos de entrevista (Chávez Méndez et al., 2013).[3]

Como puede leerse en la figura 6, la entrevista estructurada se caracteriza por una lista de preguntas bien definidas que poseen un conjunto limitado de respuestas, mientras que en la entrevista no estructurada el investigador se limita a provocar la narración abierta del entrevistado, pues el interés está puesto sobre todo en su propia experiencia de vida. Un ejemplo de este tipo de entrevista sería una pregunta general como: *cuéntenos su experiencia en el movimiento social X*. Por otro lado, las entrevistas enfocadas se interesan más en el tema que en el sujeto entrevistado, una pregunta como: *Me gustaría saber su punto de vista sobre X…* es una pregunta enfocada (Chávez Méndez et al., 2013).

La preparación de una buena entrevista tiene al menos tres momentos: la pre-producción, la producción y la post-producción (Chávez Méndez et al., 2013). En la primera fase debemos asegurarnos de tener el guión de preguntas, de haber contactado al informante, asegurarnos de su disponibilidad para dar la entrevista y haber obtenido su consentimiento para grabarla.

[3] Chávez Méndez, M. G., Covarruvias, K., y Uribe, A. (eds.). (2013). *Metodología de investigación en ciencias sociales. Aplicaciones prácticas*. Colima, México: Universidad de Colima.

La fase de la producción es el desarrollo de la entrevista. Una estrategia común es comenzar de temas más generales a preguntas más específicas, procurando que éstas sean del interés del entrevistado y no sólo del investigador. Durante una entrevista el investigador debe estar pendiente de los gestos del entrevistado y otras reacciones que tenga ante los cuestionamientos. Además, hay que mantenerse atentos a ambigüedades o contradicciones en las afirmaciones del entrevistado para, sutilmente, pedirle que nos aclare los puntos. Es de suma importancia que el investigador evite el uso de un lenguaje técnico y sea demasiado directivo en las preguntas.

Finalmente, durante la post-producción, el investigador hace anotaciones en su diario de campo sobre puntos importantes percibidos en las entrevistas y, lo más pronto posible, transcribe los audios obtenidos. Es ideal que sea el investigador quien transcriba las entrevistas, pues el análisis de contenido puede comenzar desde ese momento.

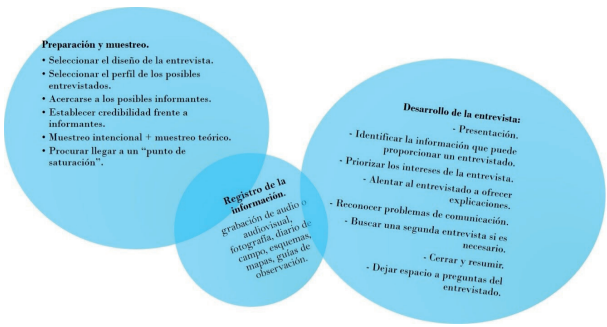


Figura 7: Aspectos importantes a tomar en cuenta en una entrevista (Chávez Méndez et al., 2013).

Otras buenas prácticas en el desarrollo y tratamiento de las entrevistas son:

1. Priorizar los intereses de la entrevista.
2. Alentar al entrevistado a ofrecer explicaciones sobre su comportamiento, integrar los hechos.
3. Reconocer problemas de comunicación con los entrevistados.
4. Completar la entrevista (mejor varias entrevistas cortas, según el tiempo con el que cuenta el entrevistado).
5. Cerrar y resumir.
6. Agradecer y dejar espacio para preguntas del entrevistado.
7. Hacer respaldos de todos los archivos.
8. Convertir todas las notas y entrevistas en archivos de texto.
9. Incluir la fecha de creación en todos los archivos.
10. Identificar los archivos según el tipo de datos que contienen: entrevistas, cuestionarios, notas de campo, tipos de participante, tipos de organización, temáticas generales, etcétera.
11. Organizar y catalogar los documentos en carpetas.
12. Crear un índice de los contenidos de los datos.
13. Revisar las preguntas de investigación a la luz de los datos obtenidos.
14. Identificar lagunas (información faltante).
15. Regresar a campo para recoger datos adicionales.

5.2. Análisis de contenido con RQDA

RQDA es un paquete de R para Análisis Cualitativo de Datos y es utilizado para analizar colecciones de texto plano, es decir, en este paquete no pueden cargarse archivos pdf, de audio o fotografías. Su función principal es la de sistematizar el análisis de contenido pero, al estar asociado a R, permite hacer también análisis cuantitativos.[4] Este paquete fue desarrollado por Huang Ronggui quien describe las funciones principales del paquete y las instrucciones de instalación en la página:

http://rqda.r-forge.r-project.org/

Para poder cargar el paquete RQDA necesitas haber bajado en tu computadora la plataforma de R y RStudio y algunos otros

[4] Si no tienes experiencia haciendo análisis cuantitativos con R, te sugerimos que consultes el Capítulo 1. Introducción al lenguaje R, en este mismo manual.

complementos indicados en la liga de arriba. Es muy importante que bajes la versión 3.4.1 de R, pues RQDA no funciona con versiones posteriores, baja dicha versión de la siguiente página:

https://cran.r-project.org/bin/windows/base/old/

Una vez que tienes R y RStudio, en la consola de RStudio debes copiar el siguiente comando:

```
install.packages("RQDA",dependencies=c("Depends",
"Imports"))
```

Con RQDA podrás:

1. Codificar cada uno de los textos.
2. Ordenar los códigos y los archivos en categorías.
3. Aplicar atributos al archivo, lo que es útil para el análisis de contenido.
4. Establecer casos.
5. Graficar la relación entre códigos.
6. Hacer búsquedas por código, categorías de código, categorías de archivo o casos.
7. Guardar tu proyecto en una base de datos SQLite.
8. Hacer memos (notas) sobre códigos, categorías de códigos, categorías de archivos y sobre el proyecto en general que podemos usar para describir, lo cuál es sumamente útil para recuperar la ruta metodológica de la investigación.
9. Eliminar temporalmente archivos y códigos.
10. Renombrar archivos, códigos, categorías de códigos y casos, entre otros.

Para facilitar el uso de RQDA puedes trabajar sobre un proyecto de ejemplo al que puedes acceder en: github.com/metodosmorelia, o bien puedes descargarlos directamente con el vínculo:

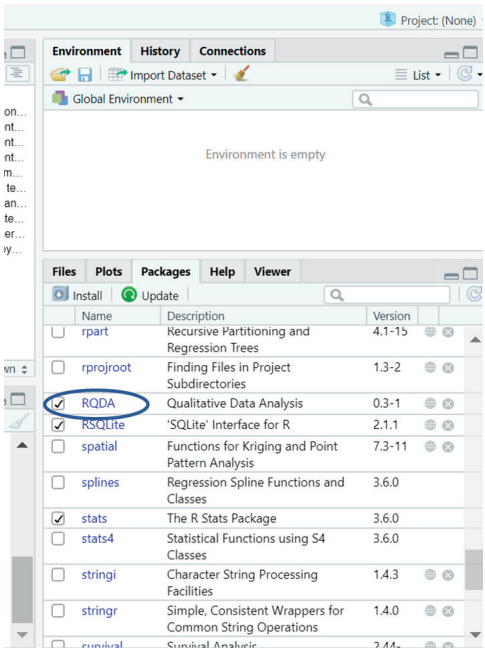
https://github.com/metodosmorelia/Manual_vol._1_Datos/archive/master.zip.

Se trata de un proyecto a través del cuál se hace un breve análisis del discurso de las declaraciones ambientales del ONU desde 1972 hasta 2012. A lo largo de los apartados práctivos haremos referencia a este proyecto con la finalidad de que te queden más claras las funciones del paquete. Además, se incluyen capturas de pantalla para indicar el uso de las funciones de RQDA.

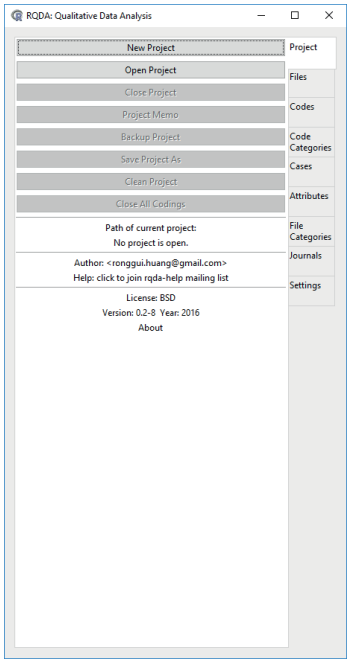
5.2.1. Estructura y herramientas generales

Crear un Proyecto

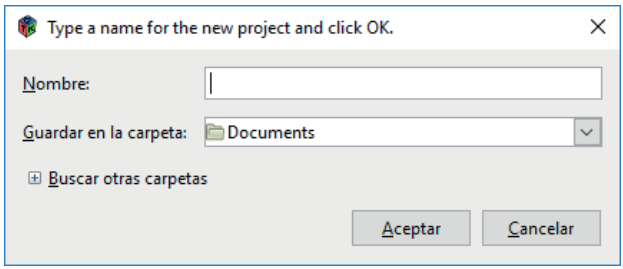
Una vez hayas instalado RQDA desde la consola de RStudio, deberás buscar en la ventana inferior izquierda, en la pestaña *Packages*, la casilla de RQDA y seleccionarla, como se muestra en la siguiente ilustración:



Al abrir RQDA desde tu computadora verás esta ventana:



Por defecto, la ventana se abre justo en la primera pestaña (Project) pues es ahí donde puedes crear un nuevo proyecto (*New Project*) o abrir uno ya existente (*Open Project*). Al pulsar en el botón *New Project* verás una ventana de diálogo que te pide nombrar al proyecto y elegir la carpeta donde quedará ubicado.



Al pulsar *Aceptar* RQDA creará un archivo con extensión *.rqda* y en ese único archivo se guardará toda la información almacenada en tu

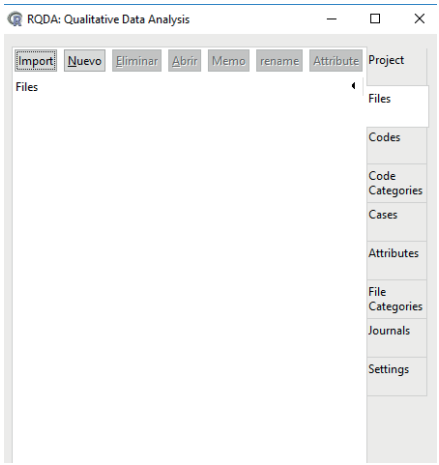
proyecto: archivos, lista de códigos, categorías, memos, etc.. Un archivo *.*rqda* te dará la oportunidad de migrar la información generada en este paquete a otros paquetes de R.

Es importante mencionar que siempre que trabajes con RQDA todos los cambios se escribirán automáticamente en el archivo *.*rqda*, por lo cual **NO** es necesario guardar tu proyecto manualmente, excepto que quieras crear un respaldo de una versión particular de tu trabajo, para lo cual debes pulsar el botón *Backup Projec*. Este respaldo aparecerá en la carpeta que has elegido para guardar tu proyecto, pero con el nombre *demo.rqda* seguido de la hora y fecha del respaldo.

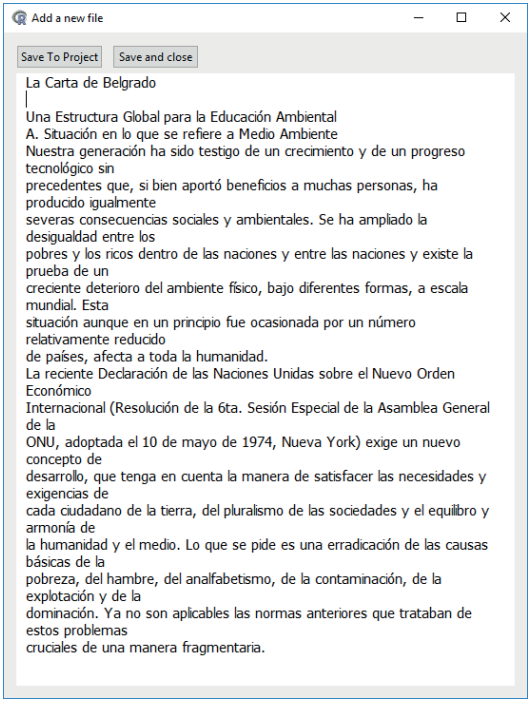
Cargar archivos

Una vez hayas creado tu proyecto debes cargar los archivos de texto que vas a analizar, estos pueden ser transcripciones de entrevistas, artículos periodísticos, declaraciones, etc. Es importante que todos los documentos que hagan parte de tu proyecto sean archivos de texto (de preferencia del tipo *.txt), ya que este programa no acepta archivos en formato PDF, imágenes o audio.

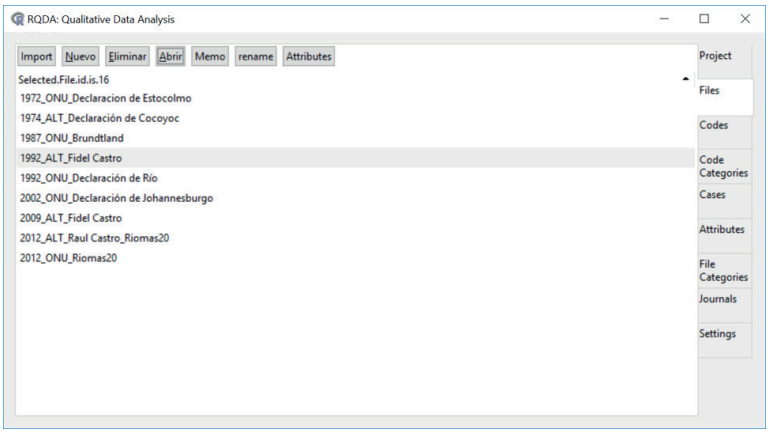
Para cargar archivos en RQDA puedes hacerlo importándolos de alguna de tus carpetas o generándolos directamente en RQDA. En nuestra experiencia es mejor generarlos en RQDA, esto significa que debes copiar los textos de su fuente original y pegarlos en la ventana diseñada para ello. Lo primero que tienes que hacer es colocarte en la pestaña *Files* y pulsar el botón *Nuevo*.



Al pulsar *Nuevo* se abrirá otra ventana y es ahí donde debes pegar el texto que vas a analizar. Después deberás pulsar *Save and close* para que tu archivo quede guardado.



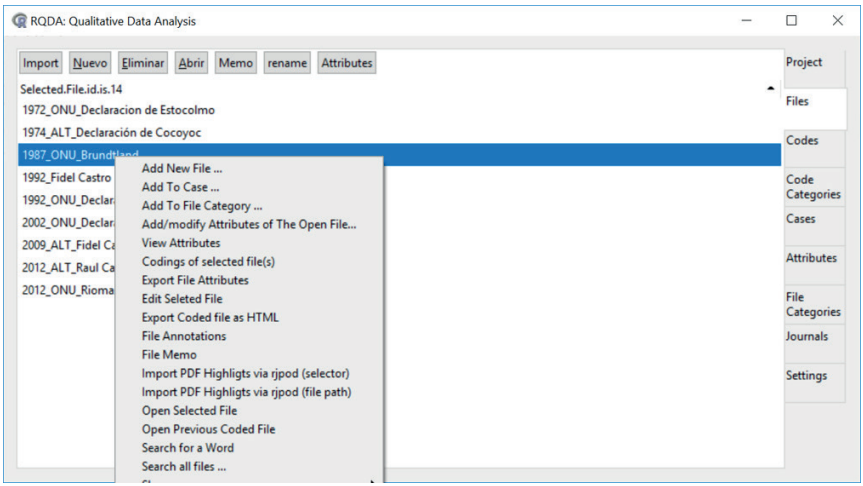
Una vez hayas cargado todos tus archivos, la lista de estos debe aparecer en la pestaña de *Files*. Es importante que reflexiones sobre el nombre que darás a tus archivos de tal manera que los nombres tengan cierta coherencia con tus objetivos de análisis, tus preguntas de investigación e, incluso, tu marco teórico-conceptual. Por ejemplo, puedes tener archivadas entrevistas separándolas por grupo de edad o sexo e indicar esto en el nombre de tus archivos. En el proyecto de ejemplo[5] verás que es el año de publicación del documento el criterio principal para organizar los archivos, esto debido a que en este proyecto se pretende hacer un análisis de la transformación de los discursos ambientales a lo largo de las décadas que van de 1972 a 2012.



Puedes borrar archivos o renombrarlos. Para eliminar un archivo selecciónalo y pulsa el botón *Eliminar*, asegúrate de que estás colocado en la pestaña *Files*. Para cambiar el nombre del archivo selecciónalo y pulsa el botón *rename*. Para abrir un archivo haz doble clic sobre él. También puedes agregar el archivo seleccionado a una categoría de

5 Para acceder al proyecto de ejemplo: a) baja el archivo **.rqda* de: https://github.com/metodosmorelia/Manual_vol._1_Datos/archive/master.zip
b) guárdalo en la carpeta de tu preferencia, c) abre RQDA desde RStudio y d) elige *Open Project* y selecciona el archivo **.rqda* que bajaste en el paso a).

archivo, ver sus atributos, ver los códigos del archivo, etc. Explora el menú que aparece cuando das clic derecho sobre el nombre de los archivos para conocer las posibilidades.



Si tienes una gran cantidad de archivos y necesitas hacer una búsqueda, además de explorar el menú dando clic derecho, también puedes usar la consola de RStudio y explorar tu proyecto usando algunas funciones. Si escribes en la consola `??SearchFiles`, aparecerá una lista de los archivos que componen tu proyecto. Si quieres calcular la cantidad total de archivos en un proyecto, utiliza la función `getFileIds()`, para obtener la cantidad total de archivos sin codificar utiliza `getFileIds(type=”uncoded”)` y para calcular la cantidad de archivos codificados escribe en la consola de RStudio `getFileIds(type=”coded”)`.

Abre el proyecto de ejemplo y usa estas funciones en la cosola para que observes el tipo de la información que se genera.

Códigos

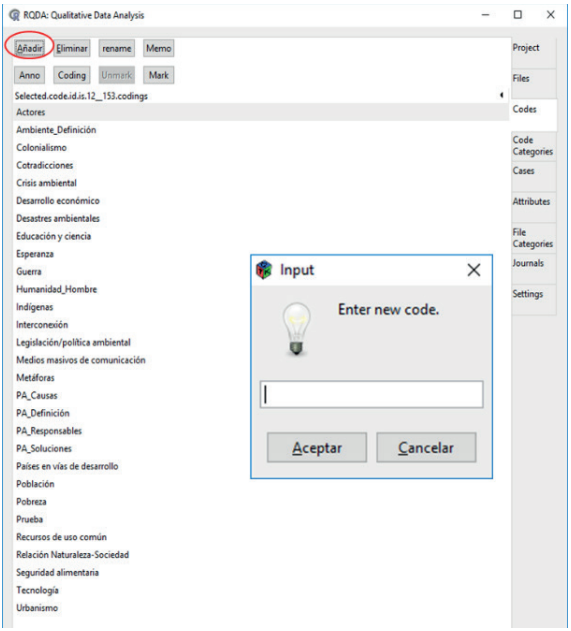
El objetivo principal de los programas de análisis cualitativo es permitirnos codificar los textos (imágenes, audios, etcétera) que queremos analizar, es decir, poner etiquetas a fragmentos de texto relacionados con conceptos relevantes de nuestra investigación. A estas etiquetas las llamamos códigos (*Codes*).

El proceso de codificación es mucho más amplio que el simple uso de una herramienta como el RQDA. El marco conceptual y el enfoque teórico de la investigación, así como la selección de los textos que serán analizados, son la base sobre la cual se elaboran los códigos con los que trabajará el *software*. Para llegar a una lista de códigos debemos hacer una lectura previa de los documentos y reflexionar sobre los patrones y significados que "emergen" del texto y su relación con nuestras preguntas de investigación.

RQDA nos ayudará a marcar aquellos fragmentos que serán codificados, contados y organizados. Algunos consejos para hacer una lista de códigos son:

1. Identificar elementos que se repiten con frecuencia en el texto y aparecen como patrones o tendencias.
2. Observar si elementos que suponíamos debían aparecer, están ausentes
3. Tomar en cuenta los elementos que los informantes identifican como algo significativo.

Para codificar un texto en RQDA es necesario crear los códigos en la pestaña *Codes* y presionar el botón *Añadir*, esta orden hará que aparezca una ventana de diálogo donde puedes escribir el nombre del código. Puedes crear tantos códigos como sea necesario.

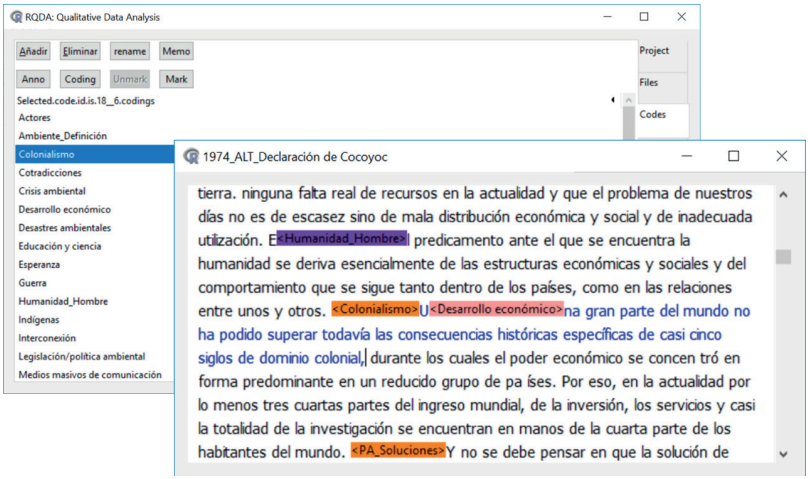


Posteriormente aplicaremos estos códigos para trabajar con los textos que forman parte de nuestro proyecto. Para iniciar con esta tarea debemos abrir uno de los archivos, leer cuidadosamente y aplicar uno o más códigos a las partes del texto que nos parecen significativas. Para codificar debemos:

- Abrir uno de los archivos de nuestro proyecto.
- Leer cuidadosamente y seleccionar los fragmentos del texto que son significativos y que están relacionados con nuestras preguntas de investigación.
- Colocarse en la pestaña *Codes* y dar clic en el código que queremos aplicar al fragmento seleccionado.
- Dar clic en el botón *Mark*.

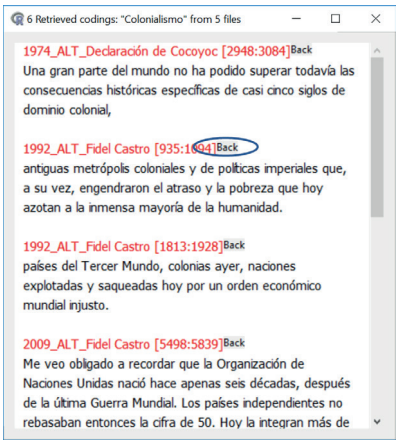
Después de hacer el procedimiento, verás que el segmento seleccionado se resalta en color azul y una etiqueta de codificación ha sido insertada al principio del segmento. Un mismo fragmento puede ser

codificado con más de un código y también puede ser desmarcado, para hacer esto último debes dar clic en la etiqueta de codificación y después hacer clic en el botón “Desmarcar”. Ahora, el resaltado se elimina, lo que significa que ha eliminado con éxito esa codificación. La siguiente captura de pantalla muestra parte del procedimiento descrito.



Si deseas ver qué información está asociada a un código, debes hacer doble clic en un código y se abrirá una ventana con todos los fragmentos de todos los archivos que han sido marcados con ese código, indicando en rojo el nombre del archivo donde se localiza el fragmento codificado.

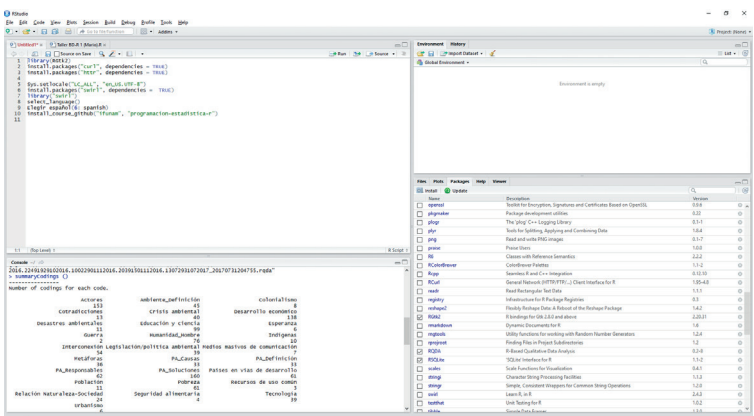
A este proceso le llamamos “recuperación de codificación” y en la ventana que se abre puedes volver al archivo original de donde proviene el texto codificado dando clic en la palabra *Back*. Volver al archivo te permite ver el contexto de la cita y, en su caso, recodificar o desmarcar la cita.



Para obtener una estadística descriptiva de la codificación, puedes trabajar sobre la consola de RStudio. Por ejemplo, la función `summaryCodings()` te dará un resumen de:

- Número de codificaciones para cada código.
- Promedio de palabras asignadas con cada código
- Número de archivos asociados con cada código”.

La siguiente captura de pantalla te muestra un resultado similar al que obtendrías si aplicaras la función `summaryCodings()` al proyecto de ejemplo sobre los disucursos ambientales.



Para obtener una tabla que después puedas exportar a una hoja de cálculo[6] y analizar la frecuencia de cada código, debes usar la función `getCodingTable ()`.

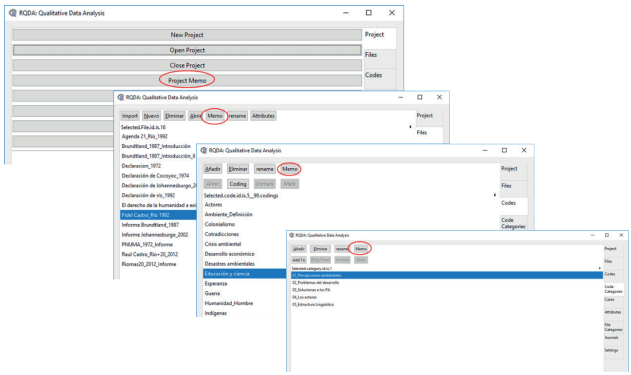
```
12:1   (Top Level) ÷                                                          R Script
Console ~/
                    Urbanismo
                            2
> getCodingTable ()
     rowid cid fid                     codename                filename index1 index2
1        1   1  13          Ambiente_Definición        Declaracion_1972    415    433
2        2   8  13  Relación Naturaleza-Sociedad       Declaracion_1972    443    559
3        3   8  13  Relación Naturaleza-Sociedad       Declaracion_1972    566    610
4        4   8  13  Relación Naturaleza-Sociedad       Declaracion_1972    674    771
5        7   4  13          Desarrollo económico       Declaracion_1972   1047   1204
6        8   7  13                PA_Soluciones        Declaracion_1972   1047   1289
7        9   4  13          Desarrollo económico       Declaracion_1972   1294   1642
8       10   8  13  Relación Naturaleza-Sociedad       Declaracion_1972   1436   1641
9       12   9  13              Humanidad_Hombre       Declaracion_1972    373    434
10      13   9  13              Humanidad_Hombre       Declaracion_1972   1294   1424
11      14   6  13                    PA_Causas        Declaracion_1972   1436   1754
12      15   9  13              Humanidad_Hombre       Declaracion_1972   1774   1870
13      16   2  13               PA_Definición         Declaracion_1972   1774   2224
14      20   4  13          Desarrollo económico       Declaracion_1972   2231   2723
15      21   6  13                    PA_Causas        Declaracion_1972   2231   2338
16      22   4  13          Desarrollo económico       Declaracion_1972   2741   2856
17      23   6  13                    PA_Causas        Declaracion_1972   2856   2999
18      24   6  13                    PA_Causas        Declaracion_1972   3002   3112
19      25   8  13  Relación Naturaleza-Sociedad       Declaracion_1972   3212   3456
20      26   7  13                PA_Soluciones        Declaracion_1972   3457   3615
21      27   6  13                    PA_Causas        Declaracion_1972   3802   3946
```

Memos

Escribir tus reflexiones durante la lectura y la codificación es una parte fundamental en el proceso de Análisis Cualitativo de Datos, y una tarea particularmente importante en proyectos colectivos. Además, escribir memos será un insumo especialmente útil a la hora de hacer la narrativa sobre tu proceder metodológico. RQDA te da la posibilidad de escribir varios tipos de notas o memorandums:

- Memo del proyecto: aquí puedes explicar el objetivo y estructura de tu proyecto, o cualquier otra idea relacionada con tu investigación en general.
- Memo de archivo: son notas sobre cada uno de los archivos que conforman tu proyecto.
- Memo de códigos: puede incluirse una definición de cada uno de los códigos que se usan en el proyecto.
- Memo de categorías de códigos: describe los criterios que fueron utilizados para agrupar los códigos en categorías.

[6] Consulta el capítulo 1 de este manual para saber cómo obtener una hoja de cálculo a partir de funciones en R.



Para cada uno de los memos deberás colocarte en la pestaña correspondiente: Proyecto (*Project*), Archivo (*File*) o señalar alguno de los códigos o categoría de códigos y dar clic en el botón *Memo*, al hacer esto se abrirá una caja de texto donde podrás registrar tus notas. Al terminar de escribir tu texto debes presionar el botón *Save Memo*.

5.3. Codificación y categorías

5.3.1. Estrategias de codificación

La codificación es una fase del proceso de análisis cualitativo que tiene como objetivo identificar los elementos básicos que caracterizan el contenido textual estudiado. La codificación nos permite identificar patrones y estructuras en el comportamiento de nuestro objeto de estudio.

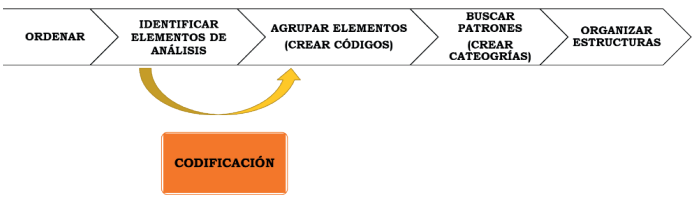


Figura 8. Lugar de la codificación en el proceso de análisis (LeCompte, 2000).[7]

[7] LeCompte, M. D. (2000). Analyzing Qualitative Data. *Theory into Practice*, *39*(3), 146–154.

Margaret D. LeCompte hace una analogía entre analizar datos y armar un rompecabezas, proceso en el cual: 1) identificamos y agrupamos piezas con características similares, 2) vamos armando partes del rompecabezas, y 3) juntamos las partes agrupadas para armar el rompecabezas en su totalidad (LeCompte, 2000).

La tarea de codificar comprende fases similares y, aunque en el caso del análisis de datos no tenemos un modelo pre-establecido de cómo debe verse la imagen final del rompecabezas, es justamente el proceso de análisis el que nos permite organizar los datos que hemos obtenido en el proceso investigativo. Por ello, es importante que a lo largo del análisis tengamos presentes los objetivos, el marco teórico, las preguntas y las hipótesis de nuestra investigación.

El proceso de codificación se compone de al menos 5 fases

- **Ordenar**: tiene que ver con establecer criterios para ordenar los archivos (fecha, tipo, etc.), crear carpetas según el tipo de datos, participantes, organizaciones o temas; catalogar y etiquetar el material, entre otras.
- **Identificar "fragmentos" (primera lectura)**: una vez seleccionados y organizados los materiales de análisis, se leen los textos y se identifican los fragmentos que son susceptibles de ser codificados para el análisis.
- **Crear categorías estables (códigos)**: después de la primera lectura de los materiales debemos organizar los "fragmentos" en grupos similares, es decir, debemos establecer los códigos y las categorías que serán la base del análisis de contenido. No hay reglas fijas para la codificación o la categorización, éstas dependen de la investigación (sus objetivos, preguntas, marco teórico, etcétera).
- **Crear patrones**: se trata de integrar taxonomías de manera significativa, buscando posibles relaciones para construir explicaciones coherentes o descripciones del objeto de estudio. Para ello se utilizan algunas características observadas en los elementos

analizados, como frecuencia, omisión, similitud, secuencia, etcétera.

- **Armar estructuras**: una vez que se han identificado patrones, estos son integrados en una estructura, es decir, se hacen grupos de patrones relacionados o vinculados entre sí y cuya totalidad da una explicación coherente del objeto de estudio y ofrece interpretaciones significativas para el campo de estudio y la teoría.

Podemos identificar distintos tipos de códigos. Algunos autores llaman sustantivos a aquellos códigos que están directamente relacionados con los datos empíricos y códigos teóricos a los que se establecen a partir de los conceptos de la literatura usada en la investigación. También se habla de códigos *In vivo*, esto es, los que provienen del lenguaje de los informantes.

Autores como Saldaña (2009)[8] han establecido algunas estrategias y tipos de codificación. En las tablas que se incluyen a continuación se muestran algunas de las estrategias más comunes, que el autor distingue como codificación gramatical y codificación elemental.

	TIPOS DE CÓDIGO	DESCRIPCIÓN	Uso	Análisis
COD. GRAMATICAL	De atributos	Información descriptiva sobre las características de la población de estudio.	Proveer información sobre los actores y su contexto para el análisis y la interpretación.	Los atributos más relevantes pueden seleccionarse como dimensiones transversales de análisis.
	De magnitud	Símbolos o palabras que sugieren intensidad, frecuencia, dirección, presencia o evaluación.	Agregar información a los códigos.	Desarrollo de metodologías mixtas.
	Simultanea	Uso de dos o más códigos en un fragmento de texto.	Cuando un fragmento de texto se vincula con distintos temas.	Según los tipos de codificación utilizados.

Tabla 1. Codificación gramatical (Saldaña, 2009).

El autor considera una codificación como "elemental" cuando se establecen códigos a partir de las preguntas de investigación, de la

[8] Saldaña, J. (2009). *The Coding Manual for Qualitative Researchers* (p. 240). https://doi.org/10.1017/CBO9781107415324.004

descripción de temas, del lenguaje de los informantes, de las acciones que reportan los informantes, de los elementos que “emerjan” de los datos (ver Tabla 2).

	TIPOS DE CÓDIGO	DESCRIPCIÓN	Uso	Análisis
COD. ELEMENTAL	Estructural	Vinculo de segmentos del texto con las preguntas de investigación y/o la estructura de entrevistas.	Diseños de investigación organizados a través de variables y/o uso de entrevistas semiestructuradas.	Se identifican fragmentos de texto donde se analiza el contenido a través técnicas de codificación secundaria. Pueden utilizarse análisis de frecuencias.
	Descriptiva	Identificación de temas (no síntesis del contenido).	Introducción a casos de estudio o fragmentos temáticos, etnografías.	Hacer un “índice” de la información
	“In vivo”	Referencia a una palabra o expresión de los actores locales.	Usar el leguaje de los actores locales, analizar conceptos locales, etnografía.	Recuperar la palabra de los actores, dar cuenta de su significado y relación con el estudio.
	De procesos	Hacen referencia a acciones.	Análisis de acciones e interacciones.	Construir secuencias, diagramas de flujo y líneas de tiempo
	Inicial o abierta	Identificación de unidades de texto y análisis comparativo, dejando abierta la posibilidad de categorías emergentes.	Organizar el material y encontrar “categorías emergentes”.	Identificar líneas de análisis que profundizar.

Tabla 2. Codificación elemental (Saldaña, 2009).

También pueden utilizarse métodos participativos para establecer los criterios de codificación, por ejemplo, discutir en un taller comunitario (o grupo focal) cuáles son los efectos de la migración en distintos ámbitos, y caracterización en buenos, malos y neutros respecto a aspectos específicos. Esto puede ayudar a hacer más robustas y significativas las categorías utilizadas en las investigaciones.

Hemos presentado algunas estrategias de codificación, sin embargo, hay una buena variedad de ellas, pues éstas dependen de la orientación del investigador, de los objetivos de su investigación (describir, ilustrar, comprender, establecer relaciones causales, etcétera), del tipo de fenómeno que estudia y de las preguntas que guían su pesquisa.

5.3.2. Construcción de categorías y tipos de relaciones

Es importante describir de manera clara y sistemática los criterios utilizados para codificar y, después, construir un esquema de clasificación. En el contexto del análisis de dominios culturales, Spradley (1980)[9] propone un conjunto de relaciones posibles para establecer categorías de análisis (ver Cuadro 1).

Cuadro 1. Dominios culturales (Spradley, 1980)
X es un tipo de y
X es un lugar en y
X es parte de y
X es resultado de y
X es causa de y
X es una razón de y
X es un lugar para hacer y
X es utilizado para y
X es un paso o etapa de y
X es una característica de y

Por ejemplo, en una investigación relacionada con alimentación, se establecieron categorías según ciertas relaciones, por ejemplo:

Relación	Categoría
X es un tipo de y	Alimentos industriales incorporados en la dieta cotidiana
X en un lugar en y	Lugares de siembra en la cuenca
X es parte de y	Alimentos que componen una comida
X es resultado de y	Impactos de la urbanización en las prácticas alimentarias
X es causa de y	Factores vinculados con la inseguridad alimentaria
X es una razón de y	Factores vinculados con la vida urbana que afectan la alimentación

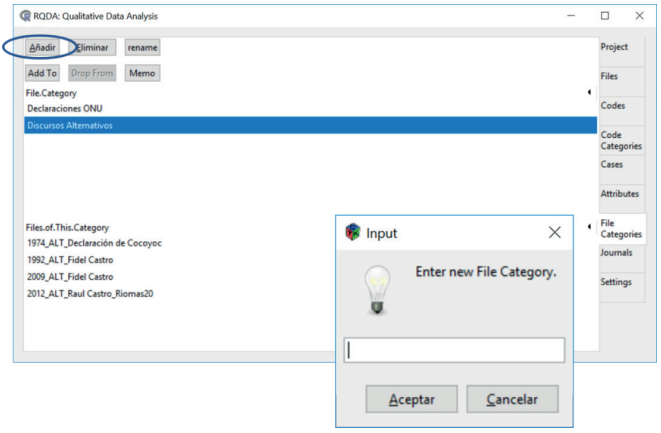
[9] Spradley, J. (1980). *Hacer análisis de dominio El Análisis Etnográfico Patrones Culturales y Situaciones Sociales*. 1–13.

X es un lugar para hacer y	Lugares en donde se abastecen de comida las mujeres
X es utilizado para y	Insumos utilizados en las actividades agropecuarias
X es un paso o etapa en y	Decisiones vinculadas con las prácticas alimentarias
X es una característica de y	Nutrientes presentes en distintos tipos de alimentos

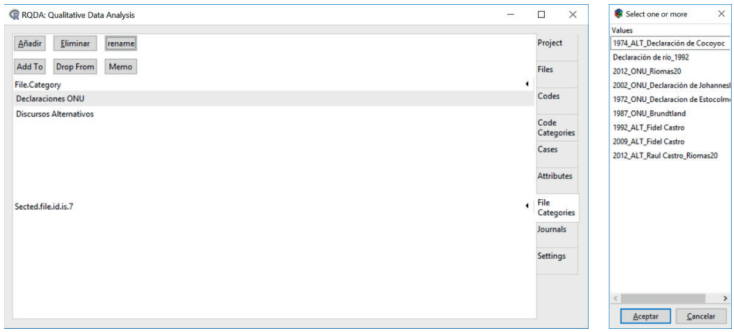
5.4. RQDA: categorías, casos y atributos

5.4.1. Categorías de archivos

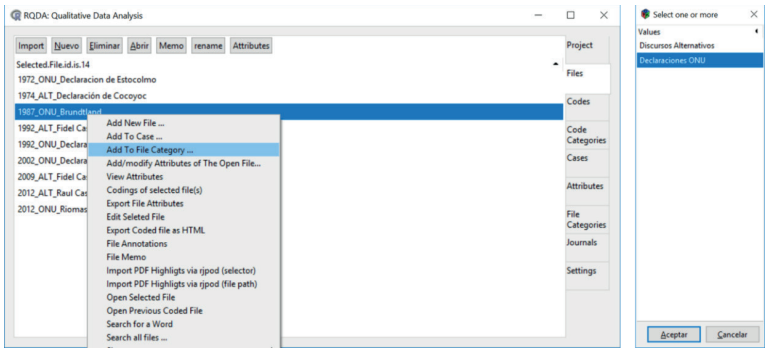
RQDA nos permite categorizar los archivos que subimos al *sofware*. Esta categorización responderá a nuestras unidades de análisis, a las preguntas de investigación, etc. En el ejemplo sobre análisis de los discursos ambientales, se categorizaron los archivos según el tipo y procedencia del discurso: si es una declaración o un informe, si lo elaboró un organismo internacional o un organismo local. Tal como se hizo para crear archivos y códigos, para incluir una categoría de archivo, debes colocarte en la pestaña de *File Categories* y hacer clic en el botón *Añadir*, eso generará una pequeña ventana donde colocaremos el nombre de la categoría y daremos *Aceptar*.



Después de crear las categorías, deberás asociar cada uno de los archivos del proyecto a las categorías. Para hacerlo colócate en la *Pestaña Categorías*, da clic en la categoría a la que agregarás archivos, presiona el botón *Add to* y aparecerá un cuadro con todos los archivos que tienes en tu proyecto, selecciona aquellos que prefieras y da clic en *Aceptar*. Observa que cada vez que seleccionas una categoría, en la parte inferior de la ventana se enumeran los archivos que están asociados a ella.



Si deseas eliminar archivos de una categoría, primero selecciona una categoría y luego señala el archivo que quieres eliminar y oprime el botón *Drop From*. La otra forma de agregar archivos a una categoría es ir a la pestaña *Files*, seleccionar los archivos que deseas agregar, dar clic derecho y seleccionar la opción *Add to File Category*, al hacerlo te aparecerá un menú donde están todas las categorías que has agregado a tu proyecto, selecciona una de ellas y da clic en *Aceptar*. En el archivo de ejemplo, no hay archivos asociados a la categoría “Declaraciones ONU”, asocia los archivos que crees corresponden a esta categoría, utiliza cualquiera de las rutas que hemos descrito.

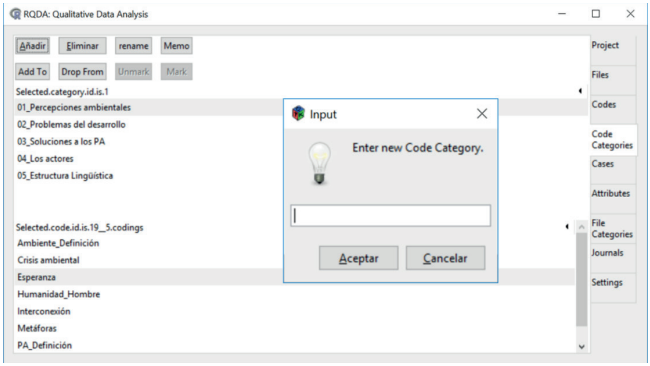


Puedes cambiar el nombre de la categoría del archivo o eliminarlo. La eliminación de la categoría del archivo no afectará a los archivos originales. Si quieres eliminar cualquiera de los archivos originales, debes ir a la pestaña *Files* para eliminarlos.

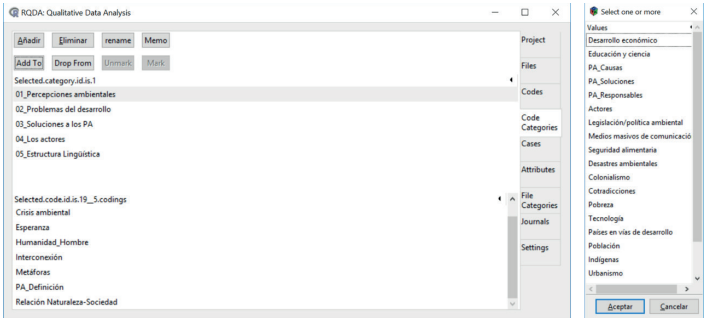
5.4.2. Categorías de códigos

Como hemos dicho antes, la codificación tiene como objetivo identificar los elementos básicos que caracterizan al fenómeno estudiado. Este conjunto de elementos traducido en códigos nos permite identificar patrones y estructuras en el comportamiento de nuestro objeto de estudio. RQDA nos permite crear *Categorías de códigos*, esto es, organizar los códigos en grupos que tienen una relación lógica entre ellos y en coherencia con nuestras preguntas de investigación e hipótesis de trabajo.

Generar las categorías de códigos, es muy similar a la forma en la generamos las categorías de archivos. Primero, agrega cada una de las categorías que agruparán tus códigos, para ello, colócate en la pestaña *Code Categories* y haz clic en el botón *Añadir*.

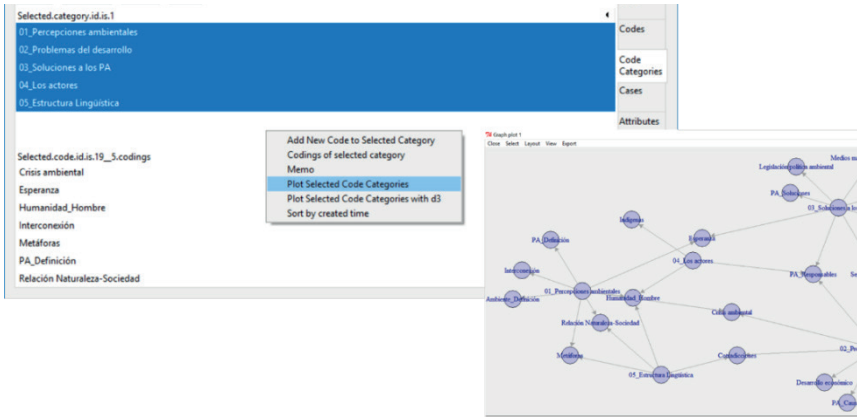


Una vez que has generado todas tus categorías, deberás asociar los códigos correspondientes, puedes asociar un código a varias categorías si así lo crees necesario. Para ello, señala una de las categorías y da clic en el botón *Add to*, al hacerlo se abrirá una ventana con todos los códigos existentes en tu proyecto y deberás seleccionar aquellos que quieres asociar a tu categoría. Nota que, en la parte inferior de la ventana principal, aparecen los códigos asociados a la categoría sombreada.



Si quieres eliminar una categoría, selecciónala y da clic en *Eliminar*. Si lo que quieres es desligar un código de una categoría, señala la categoría, después señala en la parte inferior de la ventana el código que deseas desligar y oprime el botón *Drop From*. Usa el archivo de

ejemplo para practicar cómo vincular y desvincular códigos, intenta identificar la lógica para construir las cinco categorías de código que se observan en el proyecto.



RQDA proporciona una herramienta básica para identificar las relaciones entre códigos y categorías de códigos, lo cual expresa en forma de red.[10] Esta posibilidad puede ser de mucha ayuda para iniciar una reflexión acerca de las relaciones entre códigos. Para obtener la red, colócate en la pestaña de *Code Categories*, selecciona aquellas categorías que quieres modelar (usa la tecla *control* en tu teclado para seleccionarlas de manera salteada y *shift* para seleccionarlas de manera continua), colócate fuera de la zona sombreada y da clic derecho. Selecciona la opción *Plot Selected Code Categories*. Se generará una ventana con la figura de red, que puedes editar para su mejor comprensión.

5.4.3. Uso de "casos" y "atributos"

Establecer casos y atributos en nuestro proyecto es parte fundamental del análisis y es una tarea que debe ser coherente con los métodos y

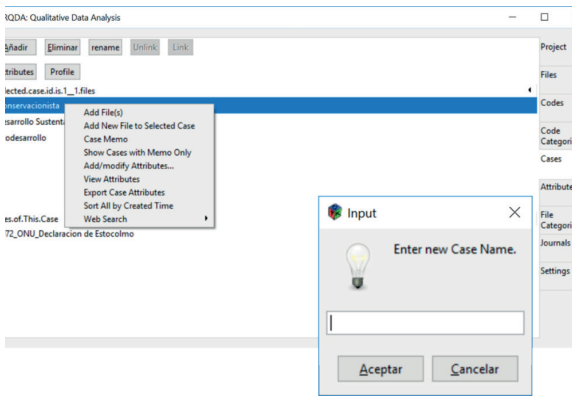
[10] En el capítulo 3 se explican algunas estrategias de análisis de redes, consultalo para saber más acerca de las posibilidades de este tipo de análisis.

teorías que enmarcan nuestra investigación. El principal objetivo de esta herramienta es permitir hacer comparaciones entre unidades de análisis (las cuales en RQDA deberían estar relacionadas con los archivos y con los casos) a partir de ciertos criterios o variables establecidos (atributos).

En el proyecto de ejemplo se establecieron como casos, las tendencias discursivas o perspectivas que, según la literatura consultada, resultaron ser las más significativas, nos referimos a:

a) Conservacionismo.
b) Ecodesarrollo
c) Desarrollo sustentable.

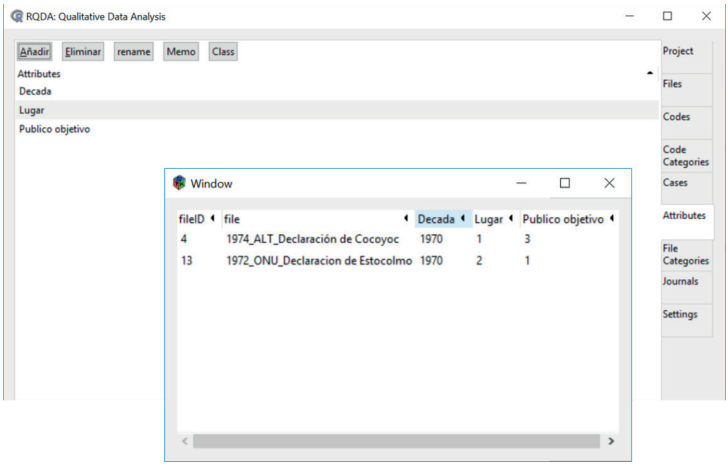
En RQDA los casos son un conjunto de archivos o de fragmentos dentro de un archivo. Para crear casos, usa el botón *Añadir* en la pestaña *Cases*. Una vez creados los casos, puedes asignar archivos a cada caso, para hacerlo selecciona un archivo en la pestaña *Files*, haz clic con el botón derecho y en el menú emergente selecciona la opción *agregar a un caso*.



En el caso de los atributos, éstos pueden ser entendidos como variables, en RQDA puedes asignar atributos a archivos y a casos, la

elección depende del diseño de la investigación. El desarrollador de RQDA ha pensado en los atributos de archivos como un proceso de recopilación de datos, mientras que los atributos de un caso son más de tipo analítico (Huang, 2018).[11]

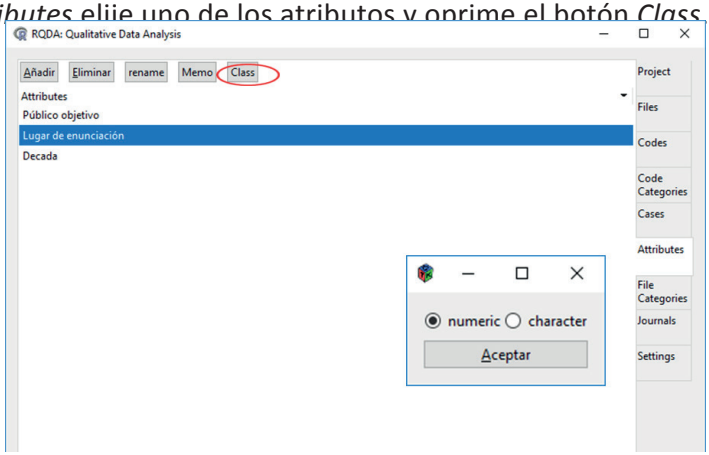
Por ejemplo, en la investigación sobre discursos ambientales que hemos mencionado antes, los atributos de los archivos y de los casos corresponden a algunos de los criterios fundamentales de la metodología de análisis crítico del discurso (ACD), donde el contexto histórico, la procedencia del que enuncia el discurso y el público al que se dirige deben ser tomados en cuenta. Por ello, se eligieron como algunos de los atributos la fecha en la que se publicó el discurso, el lugar desde donde se enuncia en términos de país central o país periférico, y el público al que se dirige cada discurso.



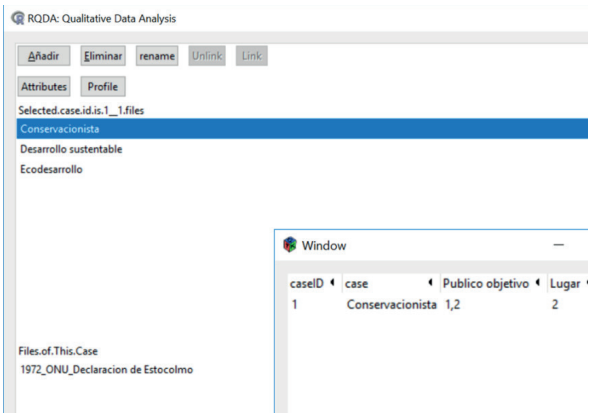
Con la misma lógica como la que se ha usado para crear categorías o casos, usa el botón *Añadir* en la pestaña de *Attributes* para generar una lista de atributos. En esta ocasión, además de agregar el nombre del atributo, debes establecer cómo será medido este atributo, si de

[11] Huang, R. (2018). *Package RQDA*. https://doi.org/10.4135/9781446288719

manera numérica o partir de caracteres, para hacer esto, en la pestaña de *Attributes* elije uno de los atributos y oprime el botón *Class*.



Tanto para los casos como para los atributos, usa la opción de memo, para registrar bajo qué criterios los has establecido. Si quieres ver qué atributos has asignado a cada caso, ve a la pestaña de *Cases*, sombrea uno de los casos y da clic derecho, en el menú emergente elije la opción *View Attributes*, esto desplegará una ventana donde podrás observar todos los atributos.



Para eliminar atributos, señala el atributo y da clic en el botón *Eliminar*. Esta operación eliminará todos los atributos de caso y de archivo.

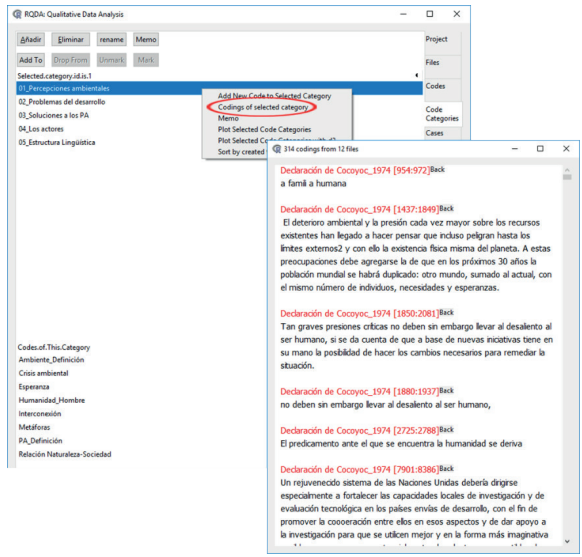
Abre el proyecto de ejemplo y observa cómo fueron asignados los atributos, abre los memos de cada uno de ellos, para identificar si son numéricos o de caracteres, así como la lógica bajo la cual fueron establecidos.

5.4.4. Esquemas de recuperación de información

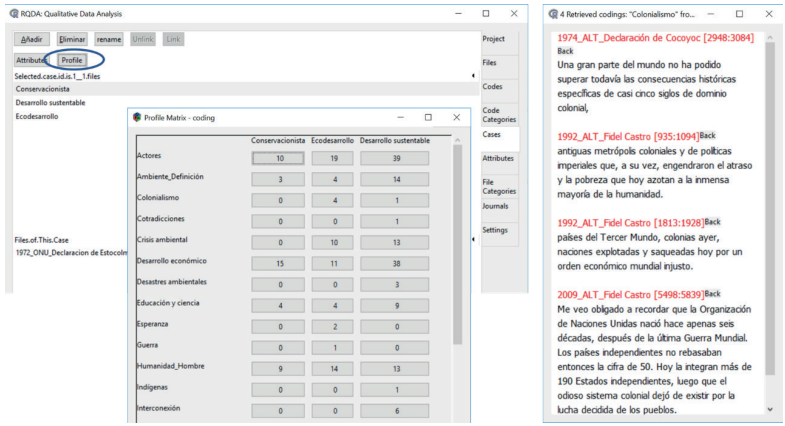
Codificar, asignar categorías de archivos y de códigos, identificar casos e identificar atributos son todas herramientas que nos permitirán recuperar información precisa de los archivos que conforman nuestro proyecto. Subir archivos, codificarlos y clasificarlos; establecer casos y asignar atributos nos permite crear un esquema base para analizar e interpretar nuestros textos. Podemos hacer la recuperación de información en varios niveles:

1. **Por código:** da clic en uno de los códigos y se abrirá una ventana que muestra todos los fragmentos codificados, indicando a qué archivo pertenece y dando la posibilidad de volver al documento original (a través del botón *Back*) si es que queremos analizar el contexto que rodea al fragmento seleccionado.

2. **Por categoría de código:** señala la categoría de código y da clic derecho, elije la opción *Codings of Selected Category*.

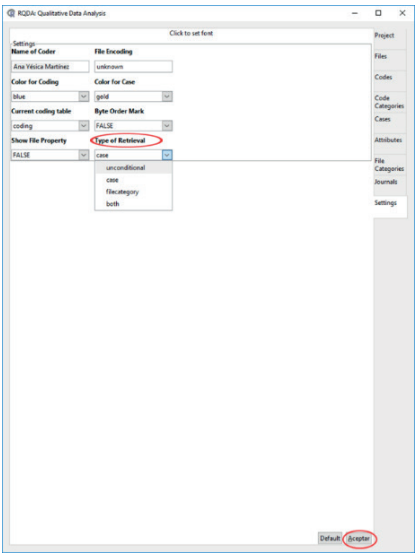


3. **Por casos:** En la pestaña de casos hay una opción muy interesante para llamar información, la opción de *Profile* que, al darle clic, arroja una matriz donde indica cuál es la incidencia de cada uno de los códigos con respecto a cada uno de los casos. Así, podemos identificar que códigos son más comunes en qué casos y, al dar clic en cada una de las celdas, nos abrirá una ventana donde despliega cada una de las citas asociadas a los códigos.



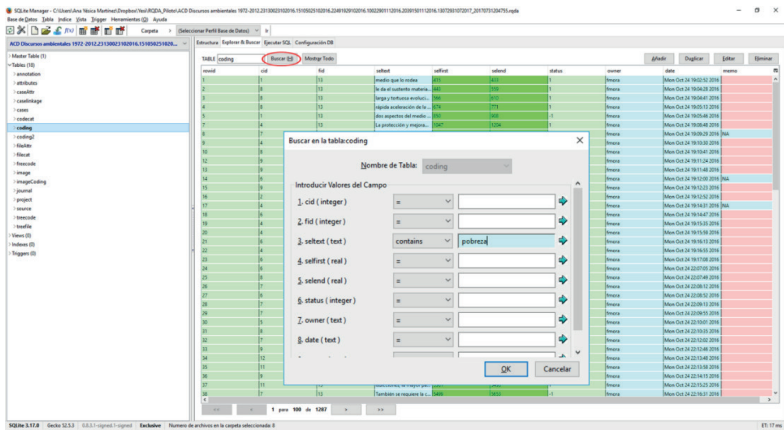
4. **Por caso o por categorías de códigos:** en la pestaña de *Settings* hay una opción que nos permite llamar información diferenciado

casos y categorías de archivo. Debes ubicarte en *Type of Retrieval* y seleccionar la opción que prefieras. Asegúrate de dar clic en el botón *Aceptar* (esquina inferior derecha) para que tu selección quede confirmada. En la pestaña *Settings* puedes también colocar el nombre del codificador (*Name of Coder*), lo cual es útil si hay más de un codificador en el proyecto. También existe la posibilidad de cambiar el color del subrayado al codificar o al asignar casos.



5. **Visualización de la base de datos:** el contenido de tu proyecto en RQDA está en una base de datos que puedes recuperar a través de un administrador de base de datos como el complemento de Firefox MSQLite Manager. Una vez hayas descargado este complemento, abre el archivo de tu proyecto de RQDA.

En esta base de datos encontrarás todos los fragmentos codificados, los memos que has escrito, los códigos, casos y categorías. Hay varias razones para que quieres exportar tu proyecto a una base de datos, por ejemplo; tal vez quieras encontrar todos los fragmentos codificados que contengan cierta palabra, o hacer alguna otra búsqueda más compleja que no puedes hacer desde la consola de RQDA.



Otro motivo importante para tener la base de datos de tu proyecto es extraer datos que te permitan ejecutar funciones directamente en R y hacer algunos análisis de tipo cualitativo y generar figuras como nubes de palabras, dendogramas, histogramas, entre otros. Puedes consultar la documentación en la página de RQDA (http://rqda.r-forge.r-project.org/) o consultar a los usuarios a través de la página: https://www.r-bloggers.com/qualitative-text-analysis-in-r-with-rqda/.

Finalmente, te recordamos que si usamos RQDA debemos citarlo, pues esto no sólo da reconocimiento al desarrollador, sino que lo anima a él y a los usuarios a mejorar esta herramienta de acceso libre. Para citar el *software* ejecuta la siguiente función en R `citation("RQDA")`.

Sobre los autores

Ana Yesica Martínez Villalba. Es licenciada y maestra en Estudios Latinoamericanos por la UNAM y actualmente es candidata a doctora en Ciencias y Humanidades para el Desarrollo Interdisciplinario por la Universidad Autónoma de Coahuila. Es técnico académico en la Licenciatura en Ciencias Ambientales de la Escuela Nacional de Estudios Superiores unidad Morelia, UNAM. Su labor está relacionada principalmente con la elaboración de materiales educativos para procesos de enseñanza-aprendizaje en ciencias ambientales, así como con el desarrollo y fortalecimiento de métodos de análisis cualitativos y análisis mixtos. Tiene más de 10 años de experiencia docente en asignaturas relacionadas con teoría social y conflictos socio-ambientales.

Ayari Pasquier Merino. Es doctora en Ciencias Sociales por El Colegio de México, obtuvo una maestría en Antropología por la Universidad de Bologna (Italia) y una licenciatura en Comunicación social por la UAM-Xochimilco. Actualmente colabora en la Secretaría de Desarrollo Institucionalde la UNAM, donde forma parte de la recién creada Coordinación Universitaria para la Sustentabilidad. Es docente del Posgrado en Ciencias de la Sostenibilidad, donde ha impartido los cursos: Participación social, Desarrollo y Sustentabilidad, Cultura Alimentaria, Salud y Sustentabilidad; y Herramientas de análisis cualitativo en las ciencias sociales. Sus líneas de investigación son: seguridad alimentaria, sistemas alimentarios sustentables, participación social y universidades sustentables, temas sobre los que ha publicado en diversas revistas científicas. También se ha desempeñado como consultora de diversos organismosnacionales e internacionales, como Conabio, FAO y OXFAM.

Francisco Mora Ardila. Es Biólogo por la Universidad Nacional de Colombia y Doctor en Ciencias Biológicas por la UNAM. En la actualidad se desempeña como Técnico Académico a cargo de la Unidad de Apoyo en Estadística, Modelación y Manejo de Datos del Instituto de Investigaciones en Ecosistemas y Sustentabilidad de la UNAM. Está dedicado a la aplicación y el desarrollo de modelos estadísticos para el análisis de sistemas ecológicos, así como a la enseñanza de métodos y herramientas para el análisis de datos. Su principal objeto de estudio son los bosques tropicales que se encuentran en recuperación luego de haber sido modificados por actividades humanas.

Mario Martínez Salgado. Investigador en la Unidad de Investigación sobre Representaciones Culturales y Sociales de la UNAM. Es doctor en estudios de población y maestro en demografía, ambos por El Colegio de México, y actuario por la Facultad de Ciencias de la UNAM. Su investigación actual se centra en los temas de familia y curso de vida; uso del tiempo y trabajo no remunerado; y métodos cuantitativos de investigación social. Ha impartido cursos sobre distintos tópicos de estadística y dinámica poblacional a nivel licenciatura y posgrado en la UNAM, El Colegio de México y la Universidad Anáhuac del Norte.

Rodrigo Tapia McClung. Investigador en el Centro de Investigación en Ciencias de Información Geoespacial, A.C (CentroGeo). Estudió física en la Facultad de Ciencias de la UNAM y obtuvo una maestría en Ciencias Ambientales en la Wilfrid Laurier University en Waterloo, Ontario, Canadá. Es candidato a Doctor en el posgrado en CentroGeo. Su investigación actual se centra en temas de visualización y analítica geovisual con el fin de comunicar información de manera dinámica y oportuna de datos geoespaciales, datos de geografía de voluntarios, demografía, transporte, migración, etc. Ha impartido cursos en las

licenciaturas de Biología y Física en Facultad de Ciencias de la UNAM y en la especialidad y maestrías de CentroGeo.

Wesley Dáttilo. Biólogo por la Universidad Estadual do Norte Fluminense (Brasil), maestro en Ecología y Conservación de la Biodiversidad por la Universidad Federal de Mato Grosso (Brasil) y doctor en Neuroetología por la Universidad Veracruzana (México). Por su desempeño recibió en 2015 el premio por la mejor tesis de doctorado de la Universidad Veracruzana. Desde 2015 trabaja como investigador en el Instituto de Ecología A.C. (Inecol). Junto con sus estudiantes y colaboradores, el Dr. Dáttilo combina el modelaje matemático, la teoría de grafos, los sistemas de información geográfica y extensos inventarios de campo para comprender la dinámica ecológica y evolutiva de las interacciones entre los organismos. Ha publicado al rededor de 50 artículos en revistas indizadas, algunos de ellos en importantes revistas como: *Science*, *Proceedings of the Royal Society B*, *Ecology*, *Biological Conservation*, *Oikos* y *Biology Letters*.

Catalogación en la publicación UNAM. Dirección General de Bibliotecas
Nombres: Mora, Francisco, 1980- , autor, editor. | Martínez Salgado, Mario, autor, editor. | Martínez Villalba, Ana Yesica, autor, editor. | Dáttilo, Wesley, autor. | Tapia McClung, Rodrigo, autor. | Pasquier Merino, Ayari, autor.
Título: Manual de métodos y herramientas para el análisis de información usando el Lenguaje "R" / coordinadores Francisco Mora Ardila, Mario Martínez Salgado, Ana Yesica Martínez Villalba ; autores Francisco Mora Ardila, Mario Martínez Salgado, Ana Yesica Martínez Villalba, Wesley Dáttilo, Rodrigo Tapia McClung y Ayari Pasquier Merino.
Descripción: Primera edición. | Morelia, Michoacán : Universidad Nacional Autónoma de México, Escuela Nacional de Estudios Superiores, Unidad Morelia, 2020.
Identificadores: LIBRUNAM |
Temas: Investigación – Metodología. | Investigación – Métodos estadísticos. | R (Lenguaje de programación para computadoras).
Clasificación: LCC QA276.45.R3.M35 2020 | DDC 519.502855133—dc23

Manual de métodos y herramientas para el análisis de información usando el lenguaje R se publica gracias al apoyo de Proyecto PAPIME 207917 "Escuela de Métodos: una estrategia para la actualización docente en torno a métodos y herramientas de análisis de información".

Primera edición: enero, 2021.

Ciudad Universitaria, Alcaldía de Coyoacán, C.P. 04510, Ciudad de México.

Escuela Nacional de Estudios Superiores, Unidad Morelia
Antigua Carretera a Pátzcuaro 8701, Col. Ex Hacienda de San José de la Huerta, C.P. 58190, Morelia, Michoacán.

La presente publicación contó con dictámenes de expertos externos de acuerdo con las normas editoriales de la ENES Unidad Morelia, UNAM.

El cuidado de la edición estuvo a cargo de Cecilia López Ridaura y Juan Benito Artigas Albarelli. El diseño de interiores y la portada estuvieron a cargo de Gospa, S. A. de C. V.

Hecho en México.

En la actualidad hemos pasado de visiones simplificadas y lineales de los sistemas socio-ecológicos a conceptualizaciones complejas, compuestas de múltiples actores, cuyas relaciones no son necesariamente lineales o unidireccionales. Junto a esta evolución conceptual se ha presentado un cambio en la obtención y procesamiento de la información asociada a dichos sistemas de estudio. En general, la disponibilidad de datos e información se ha incrementado exponencialmente, al tiempo que contamos con herramientas analíticas cada vez más sofisticadas y con una mayor capacidad de procesamiento de información.

Durante el año 2017 se implementó en la Escuela Nacional de Estudios Superiores Unidad Morelia de la UNAM el proyecto denominado "Escuela de Métodos" que propuso el desarrollo de una serie de talleres de capacitación sobre métodos y herramientas de análisis de datos usando el lenguaje de programación R. El presente manual constituye uno de los principales logros de la Escuela de Métodos. Sus cinco capítulos presentan contenidos abordados en cada uno de los talleres ofrecidos en la emisión 2017 de la Escuela. Su objetivo principal es presentar algunos métodos y herramientas de análisis de datos de utilidad en los contextos de las ciencias sociales y ambientales. En todos los casos se usan paquetes del lenguaje de programación R, que es el *software* libre para análisis estadístico más utilizado en el ámbito de la investigación científica en la actualidad.

Autores:

Francisco Mora Ardila • Mario Martínez Salgado
Ana Yesica Martínez Villalba • Wesley Dáttilo
Rodrigo Tapia McClung • Ayari Pasquier Merino