Práctico 11 Manejo de datos I (Loops y funciones en R)

Introducción a la Línea de Comandos para Análisis Bioinformáticos

March 22, 2005

dia lunes

Data wrangling: ¿Cómo manejamos nuestros datos?

- *Data wrangling*: es el proceso mediante el cual modificamos datos iniciales con el fin de analizarlos.
- *"In our experience, the tasks of exploratory data mining and data cleaning constitute 80% of the effort that determines 80% of the value of the ultimate data mining results. (. . .)"*. **Dasu & Johnson.** *Exploratory Data Mining and Data Cleaning* (2003).
- Incluye la edición, el filtrado, la obtención de nuevos y valores y más.
- En términos prácticos, en R existen librerías especializadas para esto.

Obtenemos datos... ¿y luego?

- Cada vez hay más datos... la cuestión es cómo se procesan esos datos.
- Esto se divide en dos cuestiones: conceptual (*qué analizar*) y técnica (*cómo hacer el análisis*), las cuales van de la mano.
- Una cosa es saber si tiene sentido eliminar *outliers*... la otra es cómo eliminarlos.
- Vamos a dar apenas un vistazo en cómo hacer este análisis en R.

A modo de ejemplo

- Aca va un pequeño pipe inventado en el que van pasando cosas: se loopea, se usan funciones de librerías y diseñadas por uno, se filtran, agregan y hacen cosas con columnas, y finalmente se guarda la tabla y se plotea...
- La idea es separar un poco el gráfico en dos partes: algo más como partes de loops y funciones, y otra más de uso de tablas, aclarando que igual fue algo medio artificial. No tienen que ser dos bloques: simplemente marcar que en ese proceso se van mezclando cosas según el caso, y que vamos a intentar ver algunas de esas cosas.

Estructura de las clases

- Teórico/práctico.
- Práctico 11: repaso de loops y armado de funciones en R
- Práctico 12: manejo de datos con paquetes de la librería **tidyverse**.

Funciones: una parte central de R

- R tiene funciones que vienen incorporadas por defecto
- Utilizando librerías obtenemos nuevas funciones (como las de **seqinr**, por ejemplo)
- Nosotros podemos hacer nuestras propias funciones

Funciones: una parte central de R

```
ls
```

```
## function (name, pos = -1L, envir = as.environment(pos),
##     pattern, sorted = TRUE)
## {
##     if (!missing(name)) {
##         pos <- tryCatch(name, error = function(e) e)
##         if (inherits(pos, "error")) {
##             name <- substitute(name)
##             if (!is.character(name))
##                 name <- deparse(name)
##             warning(gettextf("%s converted to character
##                 sQuote(name)), domain = NA)
##             pos <- name
##         }
##     }
##     all.names <- .Internal(ls(envir, all.names, sorted))
##     if (!missing(pattern)) {
```

¿Por qué hacer una función?

-

Componentes de una funcion

En la parte practica meter funciones del stringr, glue, magrittr...

- Entonces
- En el teorico
- Tiene que aparecer algo de ellas . Tambien podria ir purrr::map, para ver otra cuestion sobre como utilizar funciones

¿Cómo hacer una función?

```
mi_funcion = function(argumento_1, argumento_2, ...){
  # en este bloque suceden operaciones con argumento_1
  ...
  # en este bloque suceden operaciones con argumento_2
  ...
  # se devuelve algo como resultado de aplicar
  # la funcion a los argumentos
  return(una_variable_nueva)
}
```

Un ejemplo un poco más refinado

```r
# se arma una función arbitraria que evalúa si un número es

evalua_par = function(numero){
  # si la variable numero no es de clase numeric devuelvo
  if(!is.numeric(numero)){
    return('Ingrese un numero')
  }

  else {
    # evaluo si numero es par, considerando el resto de di
    resto = numero %% 2

    if(resto == 0){
      return(TRUE)
    }

    else {
      return(FALSE)
```

Llamamos a programas externos con funciones definidas por nosotros

- En esta seccion muestro como podriamos llamar a FastTree desde R, para usarlo directo en un pipe.

Dia martes

Slide con muchos tipos de graficos

Idea de lo que lleva hacer un grafico para hacerlo bien

Que hay detras de hacer estos graficos en R? Muchas librerias, de las cuales un monton son del universo tidyverse.

Los que no son de universo tidyverse igual tienen que ser retocadas generalmente, so... its good to learn tidyverse.

Cuales son las operaciones que generalmente se hacen con los datos?

- Cargar datos *crudos*/Guardar datos finales y tablas de interés.
- Filtrar datos (con criterio).
- Unir datos que vienen de diferentes fuentes, referentes a un mismo conjunto estudiado.
- Hacer modificaciones: crear *tags*, correcciones ortográficas, filas y columnas de tablas, etc. . .
- Generar nuevos datos: obtener promedios, medianas, aplicar funciones de librerías.
- Dejar anotado y reportar lo hecho.

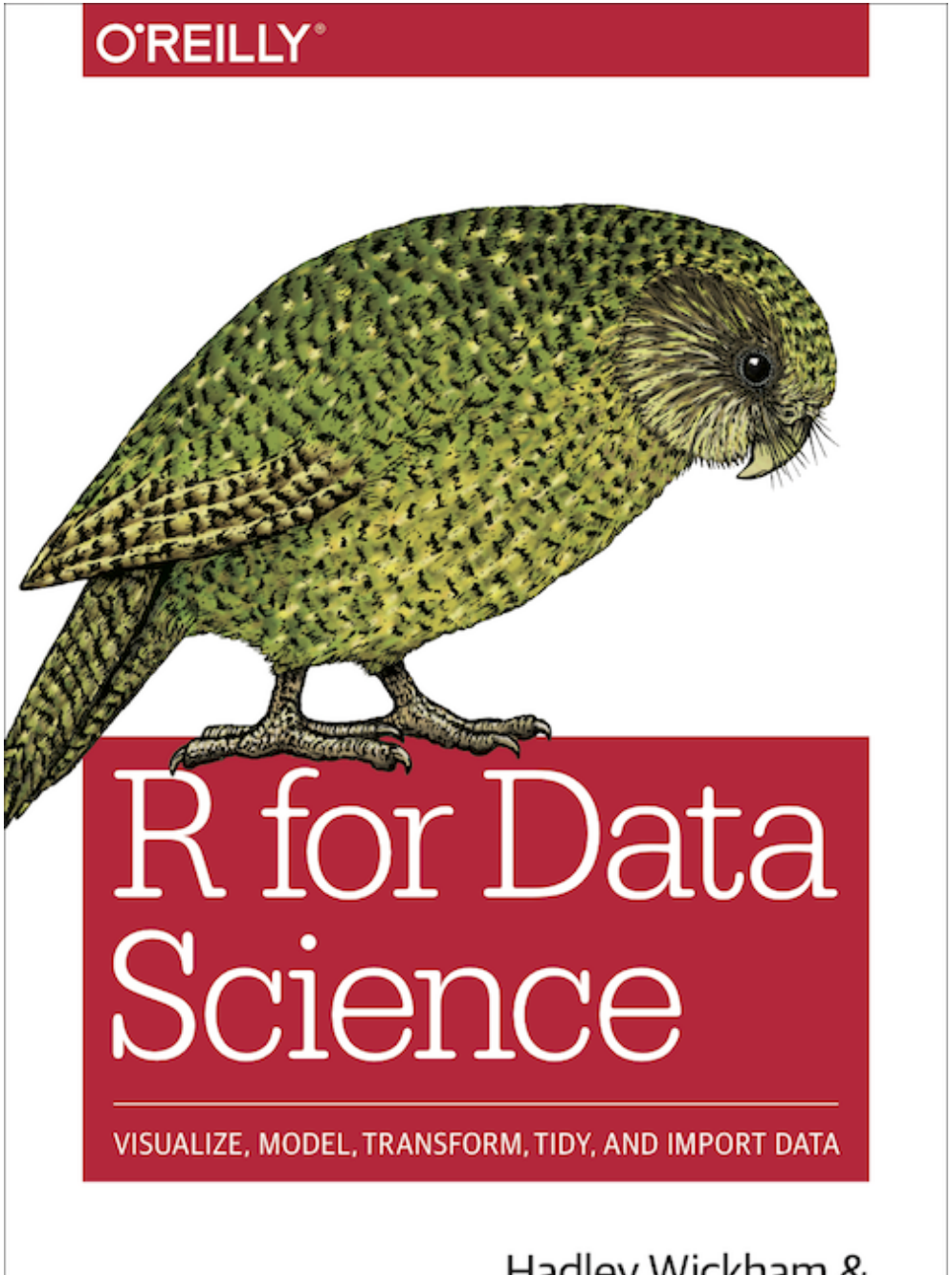
Una pequeña ilustración: es posible que querramos trabajar con un GFF

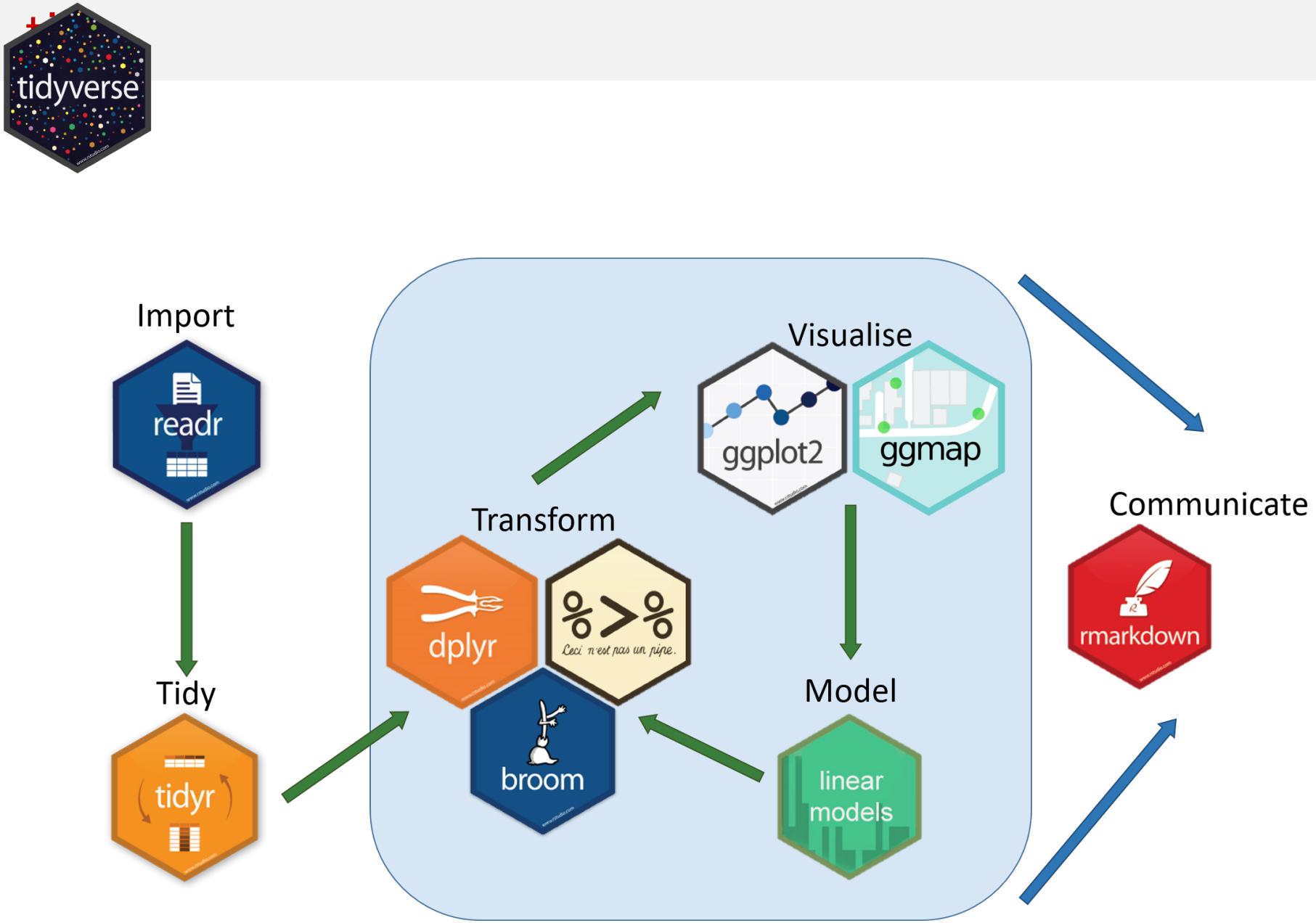
- Cargo un GFF
- Filtro para quedarme con transcriptos codificantes
- De ahí capaz quiero hacer una tabla con datos como largo de transcripto y contenido GC
- Entonces tengo que generarme una tabla que tenga las secuencias, y luego unir ambas
- Subsetear por transcripto , obtener el largo, y calcular GC
- Capaz me armo un tag que diga si son 'low GC', 'medium GC' o 'high GC'.
- Después puedo llegar a querer plotear.

Y así con todos los datos que nos vamos cruzando... nos vamos a ir encontrando cosas que vamos a tener que hacer.

Ahi recien paso a hablar del paquete

libro que vamos a usar




tidyverse
Import
readr
Tidy
tidyr
Transform
dplyr
%>%
Ceci n'est pas un pipe.
broom
Visualise
ggplot2
ggmap
Model
linear models
Communicate
rmarkdown

- Tibbles are data frames, but they tweak some older behaviours to make life a little easier.
- Almost all of the functions that you'll use in this book produce tibbles, as tibbles are one of the unifying features of the tidyverse
- It's possible for a tibble to have column names that are not valid R variable names: backtick!
- Another way to create a tibble is with tribble(), short for transposed tibble. tribble() is customised for data entry in code: column headings are defined by formulas (i.e. they start with ~), and entries are separated by commas.


```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1            5.1         3.5          1.4         0.2
## 2            4.9         3.0          1.4         0.2
## 3            4.7         3.2          1.3         0.2
## 4            4.6         3.1          1.5         0.2
## 5            5.0         3.6          1.4         0.2
## 6            5.4         3.9          1.7         0.4
## 7            4.6         3.4          1.4         0.3
## 8            5.0         3.4          1.5         0.2
## 9            4.4         2.9          1.4         0.2
## 10           4.9         3.1          1.5         0.1
## 11           5.4         3.7          1.5         0.2
## 12           4.8         3.4          1.6         0.2
## 13           4.8         3.0          1.4         0.1
## 14           4.3         3.0          1.1         0.1
```

```r
library(tibble)

as_tibble(iris)
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Spe
##           <dbl>       <dbl>        <dbl>       <dbl> <fc
##  1          5.1         3.5          1.4         0.2 set
##  2          4.9         3            1.4         0.2 set
##  3          4.7         3.2          1.3         0.2 set
##  4          4.6         3.1          1.5         0.2 set
##  5          5           3.6          1.4         0.2 set
##  6          5.4         3.9          1.7         0.4 set
##  7          4.6         3.4          1.4         0.3 set
##  8          5           3.4          1.5         0.2 set
##  9          4.4         2.9          1.4         0.2 set
```

- read_csv() reads comma delimited files, read_tsv() reads tab delimited files, and read_delim() reads in files with any delimiter.
- read_fwf() reads fixed width files. You can specify fields either by their widths with fwf_widths() or their position with fwf_positions(). read_table() reads a common variation of fixed width files where columns are separated by white space.
- **These functions all have similar syntax: once you've mastered one, you can use the others with ease**
- Sometimes there are a few lines of metadata at the top of the file. You can use skip = n to skip the first n lines; or use comment = "#" to drop all lines that start with (e.g.) #.
- The data might not have column names. You can use col_names = FALSE to tell read_csv() not to treat the first

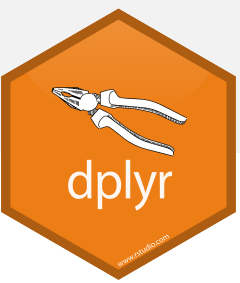
magrittr

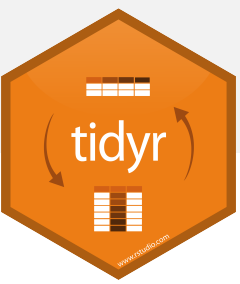
ttr

- Es el *pipe* de R.
- El uso es exactamente igual al '|' de Bash.
- Un único detalle: se utiliza **.** para hacer referencia a resultados intermedios en un pipe.

```
# hacer operaciones con una columna sin magrittr.

# Forma 1
Sepal.Width = iris$Sepal.Width
Sepal.Width.Median = median(Sepal.Width)

# Forma 2
Sepal.Width.Median =  median(iris$Sepal.Width)
```

```
# con magrittr
library(magrittr)
```


dplyr


tidyr
www.rstudio.com

tidyr

- You can represent the same underlying data in multiple ways.
  $\rightarrow$ ta bueno para mostrar diferentes tipos de datos **y decir por que esto es importante**.

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Spe
##           <dbl>       <dbl>        <dbl>       <dbl> <fc
##  1          5.1         3.5          1.4         0.2 set
##  2          4.9         3            1.4         0.2 set
##  3          4.7         3.2          1.3         0.2 set
##  4          4.6         3.1          1.5         0.2 set
##  5          5           3.6          1.4         0.2 set
##  6          5.4         3.9          1.7         0.4 set
##  7          4.6         3.4          1.4         0.3 set
##  8          5           3.4          1.5         0.2 set
##  9          4.4         2.9          1.4         0.2 set
## 10          4.9         3.1          1.5         0.1 set
## # ... with 140 more rows
```

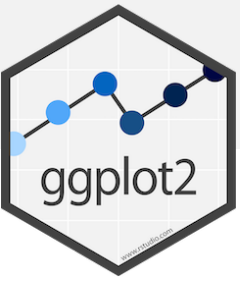
```
## # A tibble: 600 x 3
##    Species Variable     value
##    <fct>   <chr>        <dbl>
##  1 setosa  Sepal.Length   5.1
##  2 setosa  Sepal.Width    3.5
##  3 setosa  Petal.Length   1.4
##  4 setosa  Petal.Width    0.2
##  5 setosa  Sepal.Length   4.9
##  6 setosa  Sepal.Width    3
##  7 setosa  Petal.Length   1.4
##  8 setosa  Petal.Width    0.2
##  9 setosa  Sepal.Length   4.7
## 10 setosa  Sepal.Width    3.2
## # ... with 590 more rows
```

tidyr

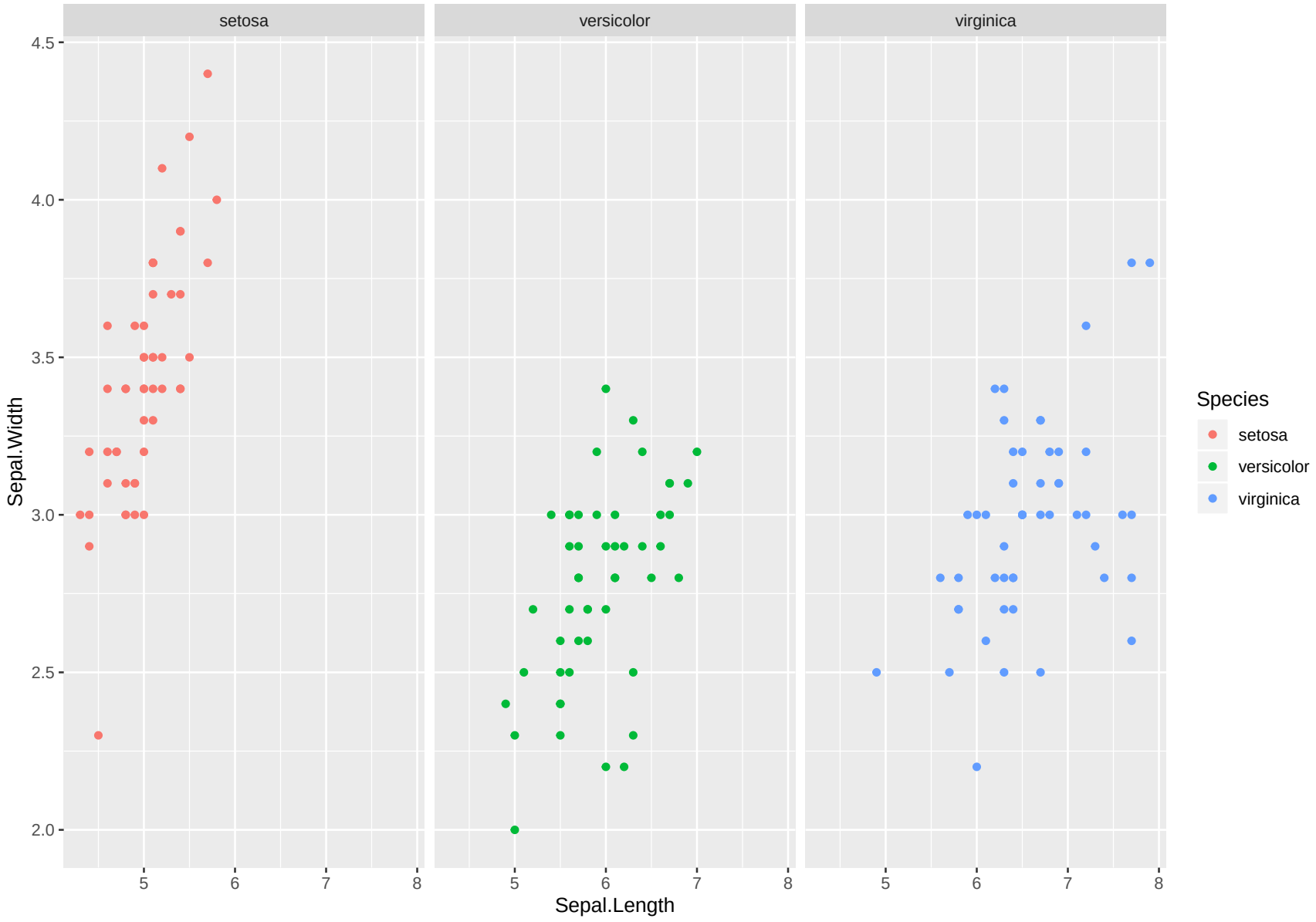
- There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

- [**most real world data is untidy**] This means for most real analyses, you'll need to do some tidying. The first step is always to figure out what the variables and observations are.
- The second step is to resolve one of two common problems:
    1. One variable might be spread across multiple columns.
    2. One observation might be scattered across multiple rows.
- Typically a dataset will only suffer from one of these problems; it'll only suffer from both if you're really unlucky! To fix these problems, **you'll need the two most important functions in tidyr**: `pivot_longer()` and `pivot_wider()`.
- A common problem is a dataset where some of the column


ggplot2

Visualización de datos



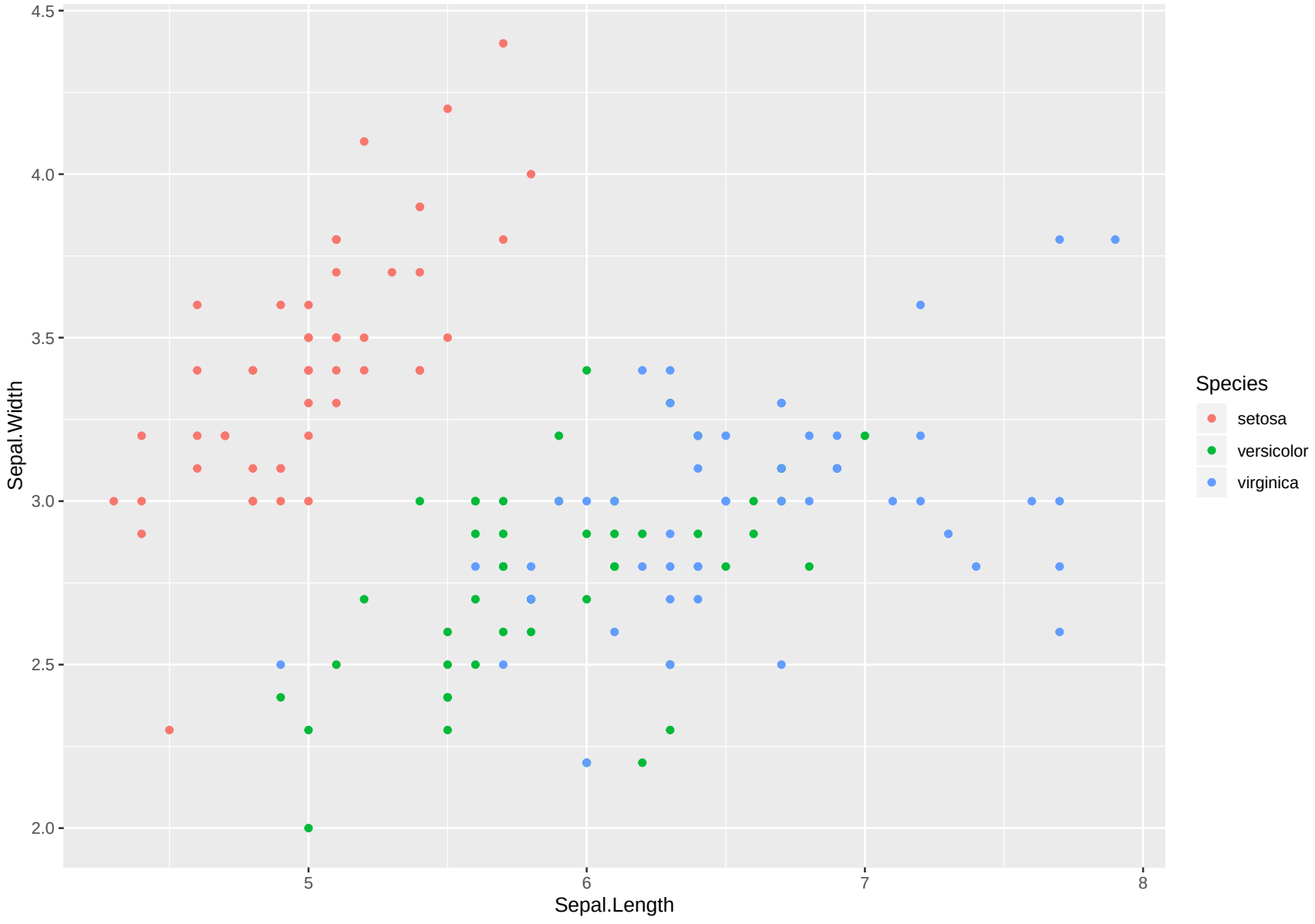
Una ventaja: reproduclbilidad

```
library(tidyverse)

# se utiliza el set de datos iris
iris %>%
  # se grafica Sepal.Length vs Sepal.Width,
  # coloreando segun Species
  ggplot(data = .,
         aes(x = Sepal.Length,
             y = Sepal.Width,
             color = Species,
             fill = Species)) +
  # se grafica utilizando puntos
  geom_point() +
  # se separa el set de datos segun Species
  facet_wrap(~Species)
```

Otra ventaja: fácil modificación

```
library(tidyverse)

# se utiliza el set de datos iris
iris %>%
  # se grafica Sepal.Length vs Sepal.Width,
  # coloreando segun Species
  ggplot(data = .,
         aes(x = Sepal.Length,
             y = Sepal.Width,
             color = Species,
             fill = Species)) +
  # se grafica utilizando puntos
  geom_point() +
#  # se separa el set de datos segun Species
#  facet_wrap(~Species)
```

Otra ventaja: fácil modificación



¿Donde encuentro info sobre estos paquetes?

- Cheatsheets
- Vignettes

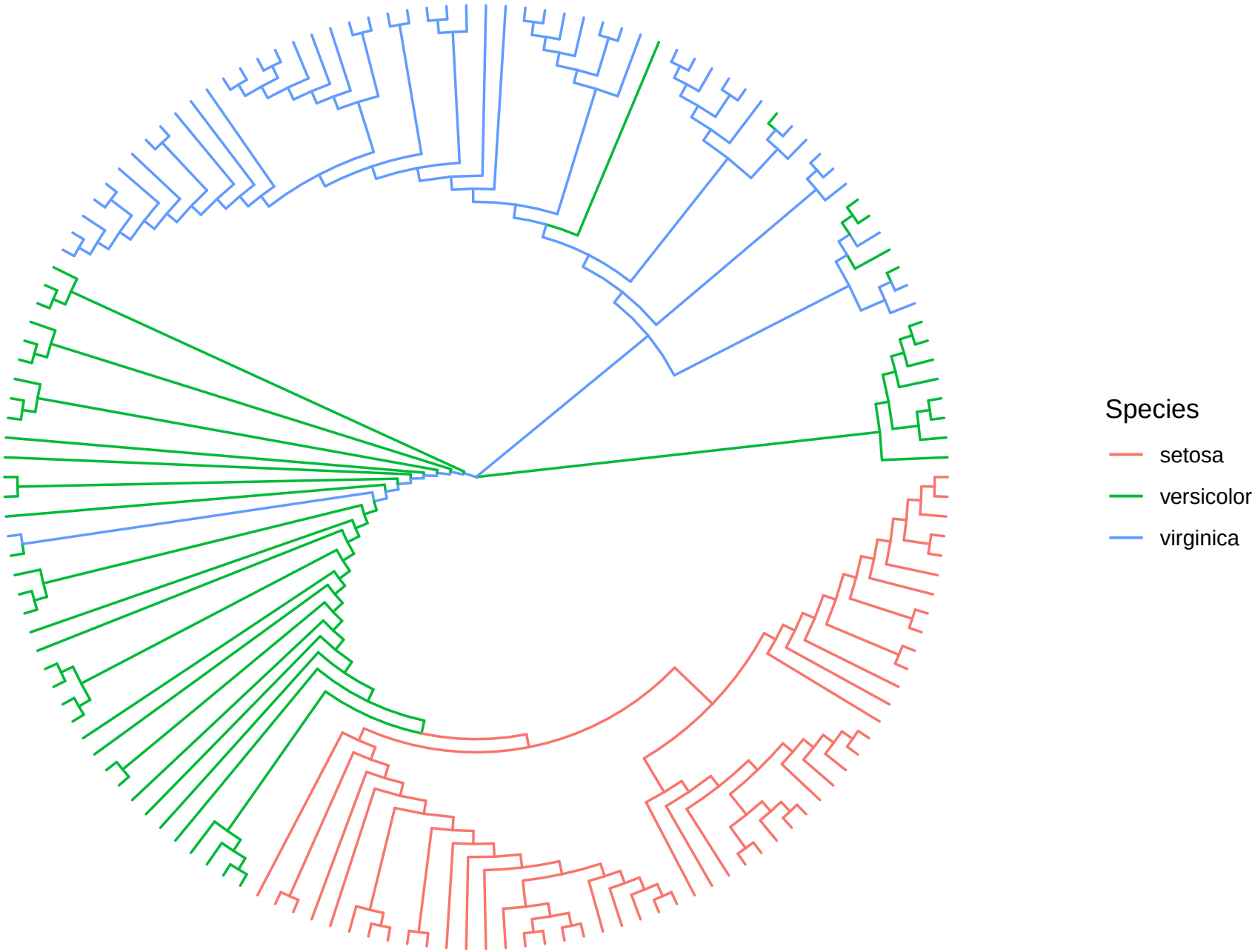
Otros ejemplos

Estadística: análisis multivariado

Clustering: librería pheatmap

GGally

Filogenética: librería ggtree



Genómica: BioCircos/rcirclize y gggnomics, ggbio

Biología estructural:

Espectrometria de masas

Ecologia