Introducción a Data Science: Programación Estadística con R

por Universidad Nacional Autónoma de México

Danilo Ibáñez Rojas
19.695.921-1

Índice

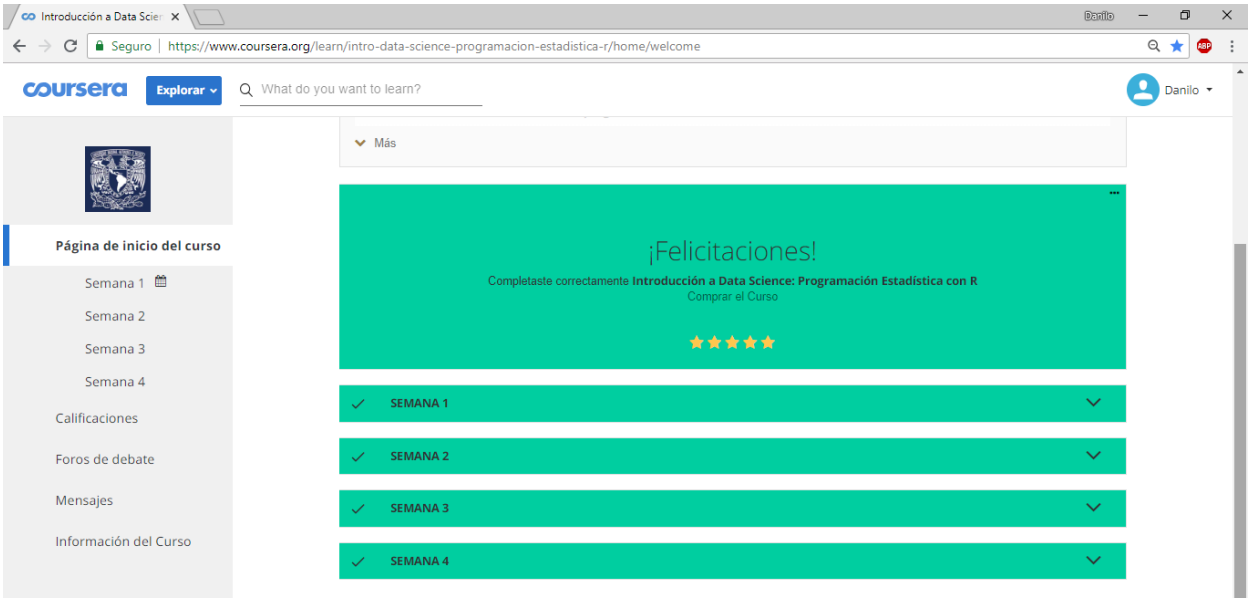
Curso terminado

Enlace del curso realizado:

https://github.com/Soumraked/Soumrak/blob/master/Coursera.pdf

Enlace acortado:

taiv.io/2D



100%

Calificación del Curso

	Vencimiento	Peso	Aprobado	Calificación
Introducción al Lenguaje				
Tareas de programación: Lección de swirl: 1. Obtener Ayuda 3h	Jun 24	2%	✔	100.00%
Tareas de programación: Lección de swirl: 2. Objetos Tipos de Datos y Operaciones Básicas 3h	Jun 24	10%	✔	100.00%
Tareas de programación: Lección de swirl: 3. Subconjuntos de Datos 3h	Jun 24	10%	✔	100.00%
Tareas de programación: Lección de swirl: 4. Leer y Escribir Datos 3h	Jun 24	10%	✔	100.00%
Utilización del Lenguaje				
Tareas de programación: Lección de swirl: 5. Funciones 3h	Jul 1	10%	✔	100.00%
Acercamiento al Sistema de Gráficos de R				
Tareas de programación: Lección de swirl: 6. Funciones apply 3h	Jul 8	15%	✔	100.00%
Tareas de programación: Lección de swirl: 7. Graficación 3h	Jul 8	15%	✔	100.00%
Tareas de programación: Lección de swirl: 8. Parámetros en el Sistema de Gráficos 3h	Jul 8	5%	✔	100.00%
Tareas de programación: Lección de swirl: 9. Colores en el Sistema de Gráficos 3h	Jul 8	5%	✔	100.00%
Tareas de programación: Lección de swirl: 8. Parámetros en el Sistema de Gráficos 3h	Jul 8	5%	✔	100.00%
Tareas de programación: Lección de swirl: 9. Colores en el Sistema de Gráficos 3h	Jul 8	5%	✔	100.00%
Expresiones Regulares, Graficación con ggplot2 y Simulación				
Tareas de programación: Lección de swirl: 12. Expresiones Regulares 3h	Jul 15	8%	✔	100.00%
Tareas de programación: Lección de swirl: 14. Simulación 3h	Jul 15	10%	✔	100.00%

Políticas de los cursos

Cómo aprobar este curso

- Aprueba la cantidad requerida de tareas calificadas (enumeradas previamente) para aprobar este curso.

Fechas de entrega

- Las fechas de entrega para cada tarea están enumeradas arriba.

Lección de swirl: 1. Obtener Ayuda

```
Selection: 1
|
|   0%

| En esta lección conocerás las principales herramientas que R tiene
para obtener ayuda.

...

  |======
|   6%
| La primera herramienta que puedes usar para obtener ayuda es help.s
tart(). En ella encontrarás un menú de
| recursos, entre los cuales se encuentran manuales, referencias y de
más material para comenzar a aprender R.

...

  |============
|  12%
| Para usar help.start() escribe en la línea de comandos help.start()
. Pruébalo ahora:

> help.start()
If nothing happens, you should open
'http://127.0.0.1:14621/doc/html/index.html' yourself

| ¡Lo estás haciendo muy bien!

  |==================
|  18%
| R incluye un sistema de ayuda que te facilita obtener información a
cerca de las funciones de los paquetes
| instalados. Para obtener información acerca de una función, por eje
mplo de la función print(), debes
| escribir ?print en la línea de comandos.

...

  |========================
|  24%
| Ahora es tu turno, introduce ?print en la línea de comandos.

> ?print

| ¡Toda esa práctica está rindiendo frutos!

  |============================
|  29%
| Como puedes observar ? te muestra en la ventana Help una breve desc
ripción de la función, de cómo usarla,
| así como sus argumentos, etcétera.

...

  |==================================
|  35%
| Asimismo, puedes usar la función help(), la cual es un equivalente
de ?. Al utilizar help(), usarás como
| argumento el nombre de la función entre comillas, por ejemplo, help
("print").
```

```
...help()

  |=========================================
|  41%
| Para buscar ayuda sobre un operador, éste tiene que encontrarse ent
re comillas inversas. Por ejemplo, si
| buscas información del operador +, deberás escribir help(`+`) o ?`+
` en la línea de comandos.

...help('+')

  |=============================================
|  47%
| Otra herramienta disponible es la función apropos(), la cual recibe
una cadena entre comillas como
| argumento y te muestra una lista de todas las funciones que conteng
an esa cadena. Inténtalo: escribe
| apropos("class") en la línea de comandos.

> apropos("class")
 [1] ".checkMFClasses"               ".classEnv"
".MFclass"
 [4] ".OldClassesList"               ".rs.getR6ClassGeneratorMethod"
".rs.getR6ClassSymbols"
 [7] ".rs.getSetRefClassSymbols"     ".rs.getSingleClass"
".rs.objectClass"
[10] ".rs.rnb.engineToCodeClass"     ".rs.rpc.get_set_class_slots"
".rs.rpc.get_set_ref_class_call"
[13] ".selectSuperClasses"           ".valueClassTest"
"all.equal.envRefClass"
[16] "assignClassDef"                "class"
"class<-"
[19] "classesToAM"                   "classLabel"
"classMetaName"
[22] "className"                     "completeClassDefinition"
"completeSubclasses"
[25] "data.class"                    "findClass"
"getAllSuperClasses"
[28] "getClass"                      "getClassDef"
"getClasses"
[31] "getClassName"                  "getClassPackage"
"getRefClass"
[34] "getSubclasses"                 "insertClassMethods"
"isClass"
[37] "isClassDef"                    "isClassUnion"
"isSealedClass"
[40] "isVirtualClass"                "isXS3Class"
"makeClassRepresentation"
[43] "makePrototypeFromClassDef"     "multipleClasses"
"namespaceImportClasses"
[46] "nclass.FD"                     "nclass.scott"
"nclass.Sturges"
[49] "newClassRepresentation"        "oldClass"
"oldClass<-"
[52] "promptClass"                   "removeClass"
"resetClass"
[55] "S3Class"                       "S3Class<-"
"sealClass"
[58] "selectSuperClasses"            "setClass"
"setClassUnion"
[61] "setOldClass"                   "setRefClass"
"showClass"
[64] "superClassDepth"               "unclass"
```

```
| Perseverancia es la respuesta.

  |======================================================
|  53%
| También puedes obtener ejemplos del uso de funciones con la función
example(). Por ejemplo, escribe
| example("read.table").

> example("read.table")

rd.tbl> ## using count.fields to handle unknown maximum number of fie
lds
rd.tbl> ## when fill = TRUE
rd.tbl> test1 <- c(1:5, "6,7", "8,9,10")

rd.tbl> tf <- tempfile()

rd.tbl> writeLines(test1, tf)

rd.tbl> read.csv(tf, fill = TRUE) # 1 column
  X1
1  2
2  3
3  4
4  5
5  6
6  7
7  8
8  9
9 10

rd.tbl> ncol <- max(count.fields(tf, sep = ","))

rd.tbl> read.csv(tf, fill = TRUE, header = FALSE,
rd.tbl+          col.names = paste0("V", seq_len(ncol)))
  V1 V2 V3
1  1 NA NA
2  2 NA NA
3  3 NA NA
4  4 NA NA
5  5 NA NA
6  6  7 NA
7  8  9 10

rd.tbl> unlink(tf)

rd.tbl> ## "Inline" data set, using text=
rd.tbl> ## Notice that leading and trailing empty lines are auto-trim
med
rd.tbl>
rd.tbl> read.table(header = TRUE, text = "
rd.tbl+ a b
rd.tbl+ 1 2
rd.tbl+ 3 4
rd.tbl+ ")
  a b
1 1 2
2 3 4

| ¡Eso es correcto!

  |======================================================
|  59%
```

```
| Con eso tendrás una idea de lo que puedes hacer con esta función.

...

  |=======================================================================
|  65%
| R te permite buscar información sobre un tema usando ??. Por ejempl
o, escribe ??regression en la línea de
| comandos.

> ??regresion

| No exactamente. Dele otra oportunidad. O escribe info() para más op
ciones.

> ??print

| No exactamente. Dele otra oportunidad. O escribe info() para más op
ciones.

> ??regression

| ¡Tu dedicación es inspiradora!

  |=======================================================================
=====                                  |  71%
| Esta herramienta es muy útil si no recuerdas el nombre de una funci
ón, ya que R te mostrará una lista de
| temas relevantes en la venta Help. Análogamente, puedes usar la fun
ción help.search("regression").

...help.search("regression")

  |=======================================================================
==========                             |  76%
| Otra manera de obtener información de ayuda sobre un paquete es usa
r la opción help para el comando
| library, con lo cual tendrás información más completa. Un ejemplo e
s library(help="stats").

...library(help="stats")

  |=======================================================================
================                       |  82%
| Algunos paquetes incluyen viñetas. Una viñeta es un documento corto
que describe cómo se usa un paquete.
| Puedes ver una viñetas usando la función vignette(). Pruébalo: escr
ibe vignette("tests") en la línea de
| comandos.

> vignette("tests")

| ¡Acertaste!

  |=======================================================================
======================                 |  88%
| Por último si deseas ver la lista de viñetas disponibles puedes hac
erlo usando el comando vignette() con
| los paréntesis vacíos.

...vignette()

  |=======================================================================
============================           |  94%
```

| Es MUY IMPORTANTE que sepas que durante todo el curso en swirl, pue
des hacer uso de las funciones help() o
| ? cuando lo desees, incluso si estas en medio de una lección.

...

  |===
==================================| 100%

Lección de swirl: 2. Objetos Tipos de Datos y Operaciones Básicas

```
Selection: 2
  |
|   0%

| En esta lección conocerás los tipos de datos que existen en e
l lenguaje R, además de las operaciones
| básicas que puedes hacer con ellos.

...

  |=
|   1%
| Cuando introduces una expresión en la línea de comandos y das
ENTER, R evalúa la expresión y muestra el
| resultado (si es que existe uno). R puede ser usado como una
calculadora, ya que realiza operaciones
| aritméticas, además de operaciones lógicas.

...

  |==
|   2%
| Pruébalo: ingresa 3 + 7 en la línea de comandos.

> 3+7
[1] 10

| ¡Buen trabajo!

  |===
|   3%
| R simplemente imprime el resultado 10 por defecto. Sin embarg
o, R es un lenguaje de programación y
| normalmente la razón por la que usas éstos es para automatiza
r algún proceso y evitar la repetición
| innecesaria.

...

  |====
|   4%
| En ese caso, tal vez quieras usar el resultado anterior en al
gún otro cálculo. Así que en lugar de volver a
| teclear la expresión cada vez que la necesites, puedes crear
una variable que guarde el resultado de ésta.

...

  |======
|   6%
| La manera de asignar un valor a una variable en R es usar el
operador de asignación, el cual es sólo un
| símbolo de menor que seguido de un signo de menos, mejor cono
cido como guion alto. El operador se ve así:
```

```
| <-

...

  |=======
|   7%
| Por ejemplo, ahora ingresa en la línea de comandos: mi_variab
le <- (180 / 6) - 15

> mi_variable <- (180 / 6) - 15

| Esa es la respuesta que estaba buscando.

  |========
|   8%
| Lo que estás haciendo en este caso es asignarle a la variable
mi_variable el valor de todo lo que se
| encuentra del lado derecho del operador de asignación, en est
e caso (180 / 6) - 15.

...(180 / 6) - 15

  |=========
|   9%
| En R también puedes asignar del lado izquierdo: (180 / 6) - 1
5 -> mi_variable

...(180 / 6) - 15 -> mi_variable

  |==========
|  10%
| Como ya te habrás dado cuenta, la asignación '<-' no muestra
ningún resultado. Antes de ver el contenido de
| la variable 'mi_variable', ¿qué crees que contenga la variabl
e ‘mi_variable’?.


1: la expresión (180 /  6) - 15
2: la dirección de memoria de la variable 'mi_variable'
3: la expresión evaluada, es decir un 15

Selection: 3

| ¡Eres bastante bueno!

  |===========
|  11%
| La variable 'mi_variable' deberá contener el número 15, debid
o a que (180 / 6) - 15 = 15. Para revisar el
| contenido de una variable, basta con escribir el nombre de és
ta en la línea de comandos y presionar ENTER.
| Inténtalo: muestra el contenido de la variable 'mi_variable':

> (180 / 6) - 15 = 15
Error in (180/6) - 15 = 15 :
  target of assignment expands to non-language object
> mi_variable
```

```
[1] 15

| ¡Tu dedicación es inspiradora!

  |============
|  12%
| Nota que el '[1]' acompaña a los valores mostrados al evaluar
las expresiones anteriores. Esto se debe a
| que en R todo número que introduces en la consola es interpre
tado como un vector.

...

  |=============
|  13%
| Un vector es una colección ordenada de números, por lo cual e
l '[1]' denota la posición del primer elemento
| mostrado en el renglón 1. En los casos anteriores sólo existe
un único elemento en el vector.

...

  |===============
|  15%
| En R puedes construir vectores más largos usando la función c
() (combine). Por ejemplo, introduce: y <-
| c(561, 1105, 1729, 2465, 2821)

> y <-
+     | c(561, 1105, 1729, 2465, 2821)
Error: unexpected '|' in:
"y <-
    |"
> y <- c(561, 1105, 1729, 2465, 2821)

| ¡Eso es trabajo bien hecho!

  |================
|  16%
| Ahora observa el contenido de la variable 'y'. Otra manera de
ver el contenido de una variable es
| imprimirlo con la función print(). Introduce print(y) en la l
ínea de comandos:

> y
[1]  561 1105 1729 2465 2821

| Estás muy cerca... ¡Puedo sentirlo! Inténtalo de nuevo. O esc
ribe info() para más opciones.

>
> print(y)
[1]  561 1105 1729 2465 2821

| ¡Es asombroso!
```

```
  |================
|  17%
| Como puedes notar, la expresión anterior resulta ser un vecto
r que contiene los primeros cinco números de
| Carmichael. Como ejemplo de un vector que abarque más de una
línea, usa el operador de secuencia para
| producir un vector con cada uno de los enteros del 1 al 100.
Introduce 1:100 en la línea de comandos.

> 1:100
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14
15  16  17  18  19  20  21  22  23  24  25  26
 [27]  27  28  29  30  31  32  33  34  35  36  37  38  39  40
41  42  43  44  45  46  47  48  49  50  51  52
 [53]  53  54  55  56  57  58  59  60  61  62  63  64  65  66
67  68  69  70  71  72  73  74  75  76  77  78
 [79]  79  80  81  82  83  84  85  86  87  88  89  90  91  92
93  94  95  96  97  98  99 100

| ¡Muy bien!

  |=================
|  18%
| El vector es el objeto más simple en R. La mayoría de las ope
raciones están basadas en vectores.

...

  |==================
|  19%
| Por ejemplo, puedes realizar operaciones sobre vectores y R a
utomáticamente empareja los elementos de los
| dos vectores. Introduce c(1.1, 2.2, 3.3, 4.4) - c(1, 1, 1, 1)
en la línea de comandos.

> c(1.1, 2.2, 3.3, 4.4) - c(1, 1, 1, 1)
[1] 0.1 1.2 2.3 3.4

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |===================
|  20%
| Nota: Si los dos vectores son de diferente tamaño, R repetirá
la secuencia más pequeña múltiples veces. Por
| ejemplo, introduce c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) - c(1, 2)
en la línea de comandos.

> c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10) - c(1, 2)
 [1] 0 0 2 2 4 4 6 6 8 8

| ¡Excelente!

  |====================
|  21%
| En R casi todo es un objeto. Para ver qué objetos tienes en u
n momento determinado, puedes usar la función
| ls(). Inténtalo ahora.
```

```
> ls()
[1] "mi_variable" "ncol"        "test1"       "tf"          "y"

| ¡Lo estás haciendo muy bien!

  |======================
|  22%
| Como sabes, existen otros tipos de objetos, como los caracter
es (character).

...(character)

  |========================
|  24%
| Las expresiones con caracteres se denotan entre comillas. Por
ejemplo, introduce "¡Hola Mundo!" en la línea
| de comandos.

> "¡Hola Mundo!"
[1] "¡Hola Mundo!"

| ¡Muy bien!

  |=========================
|  25%
| Esto es mejor conocido en R como un vector de caracteres. De
hecho, este ejemplo es un vector de longitud
| uno.

...

  |==========================
|  26%
| Ahora crea una variable llamada ‘colores’ que contenga un vec
tor con las cadenas "rojo", "azul", "verde",
| "azul", "rojo", en ese orden.

> colores <- c("rojo","azul", "verde", "azul", "rojo" )

| ¡Mantén este buen nivel!

  |===========================
|  27%
| Ahora imprime el vector ‘colores’ .

> colores
[1] "rojo"  "azul"  "verde" "azul"  "rojo"

| ¡Acertaste!

  |============================
|  28%
| En otros lenguajes como C, carácter (character) hace referenc
ia a un simple carácter, y cadena (string) se
| entiende como un conjunto de caracteres ordenados. Una cadena
de caracteres es equivalente al valor de
```

| carácter en R.

...

  |=============================
| 29%
| Además, hay objetos de tipo numérico (numeric) que se dividen en complejos (complex) y enteros (integer).
| Los últimos ya los conoces, pues has estado trabajando con ellos, además de los vectores y los caracteres.

...

  |==============================
| 30%
| Los complejos en R se representan de la siguiente manera: a+bi, donde 'a' es la parte real y 'b' la parte
| imaginaria. Pruébalo: guarda el valor de 2+1i en la variable 'complejo'.

```
> complejo <- 2+1i
```

| ¡Eres bastante bueno!

  |===============================
| 31%
| Al igual que los demás objetos de tipo numérico, lo complejos pueden hacer uso de los operadores
| aritméticos más comunes, como `+`(suma), `-` (resta, o negación en el caso unario), `/`(división), `*`
| (multiplicación) `^` (donde x^2 significa 'x elevada a la potencia 2'). Para obtener la raíz cuadrada, usa
| la función sqrt(), y para obtener el valor absoluto, la función abs().

...

  |================================
| 33%
| También hay objetos lógicos (logic) que representan los valores lógicos falso y verdadero.

...

  |=================================
| 34%
| El valor lógico falso puede ser representado por la instrucción FALSE o únicamente por la letra F
| mayúscula; de la misma manera, el valor lógico verdadero es representado por la instrucción TRUE o por la
| letra T.

...

  |==================================
| 35%

```
| Como operadores lógicos están el AND lógico: `&` y `&&` y el
OR lógico: `|` y `||`.

...

  |====================================
|  36%
| También existen operadores que devuelven valores lógicos, ést
os pueden ser de orden, como: `>` (mayor que),
| `<` (menor que), `>=` (mayor igual) y `<=` (menor igual), o d
e comparación, como: `==` (igualdad) y `!=`
| (diferencia). Por ejemplo, introduce en la línea de comandos
mi_variable == 15.

> mi_variable == 15
[1] TRUE

| ¡Eso es trabajo bien hecho!

  |====================================
|  37%
| Como puedes ver, R te devuelve el valor TRUE, pues si recuerd
as, en la variable 'mi_variable' asignaste el
| valor de la expresión (180 / 6) - 15, la cual resultaba en el
valor 15. Por lo cual, cuando le preguntas a
| R si 'mi_variable' es igual a 15, te devuelve el valor TRUE.

...

  |=====================================
|  38%
| En R existen algunos valores especiales.

...

  |======================================
|  39%
| Por ejemplo, los valores NA son usados para representar valor
es faltantes. Supón que cambias el tamaño de
| un vector a un valor más grande del previamente definido. Rec
uerda el vector 'complejo', el cual contenía
| el número complejo 2+1i; cambia la longitud de 'complejo'. In
gresa length(complejo) <- 3 en la línea de
| comandos.

> length(complejo) <- 3

| ¡Acertaste!

  |======================================
|  40%
| Ahora ve el contenido de 'complejo'.

> complejo
[1] 2+1i   NA   NA

| ¡Es asombroso!
```

```
  |=========================================
|  42%
| Los nuevos espacios tendrán el valor NA, el cual quiere decir
not available (no disponible).

...

  |==========================================
|  43%
| Si un resultado de la evaluación de alguna expresión aritméti
ca es muy grande, R regresa el valor 'Inf'
| para un valor positivo y '-Inf' para un valor negativo (infin
itos positivo y negativo, respectivamente).
| Por ejemplo, introduce 2^1024 en la línea de comandos.

> 2^1024
[1] Inf

| ¡Eres el mejor!

  |==========================================
|  44%
| Algunas veces la evaluación de alguna expresión no tendrá sen
tido. En estos casos, R regresará el valor Nan
| (not a number). Por ejemplo, divide 0 entre 0.

> 0/0
[1] NaN

| ¡Mantén este buen nivel!

  |==========================================
|  45%
| Adicionalmente, en R existe el objeto null y es representado
por el símbolo NULL.

...

  |==========================================
|  46%
| Nota que NULL no es lo mismo que NA, Inf, -Inf o Nan.

...

  |==========================================
|  47%
| Recuerda que R incluye un conjunto de clases para representar
fechas y horas. Algunas de ellas son: Date,
| POSIXct y POSTXlt.

...

  |==========================================
|  48%
| Por ejemplo, introduce fecha_primer_curso_R <- Sys.Date() en
la línea de comandos.
```

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

| ¡Acertaste!

  |=================================================
|  49%
| Ahora imprime el contenido de fecha_primer_curso_R.

> fecha_primer_curso_R
[1] "2018-06-21"

| ¡Todo ese trabajo está rindiendo frutos!

  |==================================================
|  51%
| Recuerda que R te permite llevar a cabo operaciones numéricas
y estadísticas con las fechas y horas.
| Además, R incluye funciones para manipularlas. Muchas funcion
es de graficación requieren fechas y horas.

...

  |==================================================
|  52%
| Ahora que conoces los objetos más usados en R, debes saber qu
e además de la función c(), en R existe la
| función vector() para crear vectores. Al usar la función vect
or(), debes especificar el tipo de dato que
| almacenará el vector y el tamaño.

...

  |==================================================
|  53%
| La función vector() crea un vector con los valores por defect
o del tipo especificado.

...

  |==================================================
|  54%
| Por ejemplo, escribe vector("numeric", length = 10) en la lín
ea de comandos.

> vector("numeric", length = 10)
 [1] 0 0 0 0 0 0 0 0 0 0

| ¡Excelente trabajo!

  |==================================================
|  55%
| Como puedes observar, el vector fue llenado con ceros.

...
```

```
  |=======================================================
|   56%
| Ahora introduce vector("character", length = 10) en la línea
de comandos.

> vector("character", length = 10)
 [1] "" "" "" "" "" "" "" "" "" ""

| ¡Buen trabajo!

  |=======================================================
|   57%
| Esta vez el vector fue llenado con 10 cadenas vacías.

...

  |=======================================================
|   58%
| Si le indicaras al vector que el tipo fuera "logical", ¿qué c
rees que contendría?

1: ninguna de las anteriores
2: sólo valores FALSE
3: sólo valores TRUE

Selection: 1

| Por poco era correcto, sigue intentándolo.


1: sólo valores FALSE
2: ninguna de las anteriores
3: sólo valores TRUE

Selection: 1

| Esa es la respuesta que estaba buscando.

  |=======================================================
|   60%
| Crea un vector de tipo "logical" de tamaño 10 usando la funci
ón vector().

> vector("logicalr", length = 10)
Error in vector("logicalr", length = 10) :
  vector: cannot make a vector of mode 'logicalr'.
> vector("logical", length = 10)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E

| ¡Traes una muy buena racha!

  |=======================================================
=                                       |   61%
| Nota: Ahora que conoces los objetos más usados en R, es impor
tante que sepas que los vectores sólo pueden
```

```
| contener objetos de la misma clase. Para guardar objetos de d
iferentes clases, puedes usar listas.

...

  |=======================================================
==                                                |  62%
| Las listas son una colección ordenada de objetos. Para crear
una lista, usa la función list() y especifica
| el contenido de la lista separado por comas dentro de los par
éntesis. Inténtalo: crea una lista que
| contenga un 0, la cadena "Hola" y el valor TRUE, en ese orden
.

> list(0,"Hola",TRUE)
[[1]]
[1] 0

[[2]]
[1] "Hola"

[[3]]
[1] TRUE


| ¡Tu dedicación es inspiradora!

  |=======================================================
===                                               |  63%
| Anteriormente viste que en R los vectores sólo pueden contene
r objetos de la misma clase.

...

  |=======================================================
====                                              |  64%
| Pero, ¿qué pasa si creas un vector c(T, 19, 1+3i)? Introduce
c(T, 19, 1+3i) en la línea de comandos.

> c(T, 19, 1+3i)
[1]  1+0i 19+0i  1+3i

| ¡Lo has logrado! ¡Buen trabajo!

  |=======================================================
=====                                             |  65%
| Como habrás supuesto, el número complejo 1+3i no puede ser co
nvertido a entero ni a objeto de tipo
| "logical", entonces los valores T y 19 son convertidos a los
números complejos 1+0i y 19+0i
| respectivamente. Esto no es más que la representación de esos
valores en objeto tipo "complex".

...

  |=======================================================
======                                            |  66%
```

```
| Esto se llama coerción.

...

  |=======================================================================
=======                                  |   67%
| La coerción hace que todos los objetos de un vector sean de u
na misma clase. Entonces, cuando creas un
| vector de diferentes tipos, R busca un tipo común, y los elem
entos que no son de ese tipo son convertidos.

...

  |=======================================================================
=========                                |   69%
| Otro ejemplo de coerción es cuando usas las funciones as.*().

...

  |=======================================================================
==========                               |   70%
| Inténtalo: crea un vector de longitud 5 de tipo "numeric" con
la función vector() y guardarlo en la
| variable ‘c’.

> c <- vector("numeric", length = 5)

| Esa es la respuesta que estaba buscando.

  |=======================================================================
===========                              |   71%
| Revisa el contenido de la variable ‘c’ .

> c
[1] 0 0 0 0 0

| ¡Eso es correcto!

  |=======================================================================
============                             |   72%
| Ahora usa la función as.logical() con el vector c.

> as.logical(c)
[1] FALSE FALSE FALSE FALSE FALSE

| ¡Todo ese trabajo está rindiendo frutos!

  |=======================================================================
=============                            |   73%
| Como puedes imaginar, el vector de tipo "numeric" fue explíci
tamente convertido a "logical".

...

  |=======================================================================
==============                           |   74%
```

```
| Este tipo de coerción es mejor conocida como coerción explíci
ta. Además de as.logical(), también existe
| as.numeric(), as.character(), as.integer().

...

  |=======================================================================
===============                                      |  75%
| Si usas la función class(), que te dice la clase a la que per
tenece un objeto, obtendrás que class(c) =
| "numeric." Pruébalo, ingresa class(c) en la línea de comandos
.

> class(c)
[1] "numeric"

| ¡Traes una muy buena racha!

  |=======================================================================
================                                     |  76%
| Pero si después pruebas la misma función class() enviándole c
omo argumento as.logical(c), obtendrás que es
| de tipo logical. Compruébalo:

> class(as.logical(c))
[1] "logical"

| ¡Traes una muy buena racha!

  |=======================================================================
=================                                    |  78%
| Además de los vectores y las listas, existen las matrices.

...

  |=======================================================================
==================                                   |  79%
| Una matriz es una extensión de un vector de dos dimensiones.
Las matrices son usadas para representar
| información de un solo tipo de dos dimensiones.

...

  |=======================================================================
===================                                  |  80%
| Una manera de generar una matriz es al usar la función matrix
(). Inténtalo, introduce m <-
| matrix(data=1:12,nrow=4,ncol=3) en la línea de comandos.

> m <-matrix(data=1:12,nrow=4,ncol=3)

| ¡Excelente trabajo!

  |=======================================================================
====================                                 |  81%
| Ahora imprime el contenido de 'm'.
```

```
> m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

| ¡Eso es trabajo bien hecho!

  |=======================================================================
======================                                 |  82%
| Como puedes observar, creaste una matriz con tres columnas (n
col) y cuatro renglones (nrow).

...

  |=======================================================================
======================                                 |  83%
| Recuerda que también puedes crear matrices con las funciones
cbind, rbind y as.matrix().

...

  |=======================================================================
======================                                 |  84%
| Los factores son otro tipo especial de vectores usados para r
epresentar datos categóricos, éstos pueden ser
| ordenados o sin orden.

...

  |=======================================================================
======================                                 |  85%
| Recuerda el vector 'colores' que creaste previamente y supón
que representa un conjunto de observaciones
| acerca de cuál es el color preferido de las personas.

...

  |=======================================================================
======================                                 |  87%
| Es una representación perfectamente válida, pero puede llegar
a ser ineficiente. Ahora representarás los
| colores como un factor. Introduce factor(colores) en la línea
de comandos.

> factor(colores)
[1] rojo  azul  verde azul  rojo
Levels: azul rojo verde

| ¡Tu dedicación es inspiradora!

  |=======================================================================
======================                                 |  88%
| La impresión de un factor muestra información ligeramente dif
erente a la de un vector de caracteres. En
```

| particular, puedes notar que las comillas no son mostradas y que los niveles son explícitamente impresos.

...

  |=== | 89%
| Por último, existen los dataframes, que son una manera muy útil de representar datos tabulares. Son uno de
| los tipos más importantes.

...

  |=== | 90%
| Un dataframe representa una tabla de datos. Cada columna de éste puede ser de un tipo diferente, pero cada
| fila debe tener la misma longitud.

...

  |=== | 91%
| Ahora crea uno. Introduce data.frame(llave=y, color=colores) en la línea de comandos.

> data.frame(llave=y, color=colores)
  llave color
1   561 rojo
2 1105 azul
3 1729 verde
4 2465 azul
5 2821 rojo

| Perseverancia es la respuesta.

  |=== | 92%
| ¿Recuerdas los vectores 'y' y 'colores'? Pues con ellos creaste un data frame cuya primera columna tiene
| números de Carmichael y la segunda colores.

...

  |=== | 93%
| Otra manera de crear dataframes es con las funciones read.table() y read.csv().

...

  |=== | 94%
| También puedes usar la función data.matrix() para convertir un data frame en una matriz.

...

```
  |=======================================================================
=================================    |  96%
| Antes de concluir la lección, te mostraré un par de atajos.

...

  |=======================================================================
=================================    |  97%
| Al inicio de esta lección introdujiste mi_variable <- (180 /
6) - 15 en la línea de comandos. Supón que
| cometiste un error y que querías introducir mi_variable <- (1
80 / 60) - 15, es decir, querías escribir 60,
| pero escribiste 6. Puedes reescribir la expresión o...

...

  |=======================================================================
=================================    |  98%
| En muchos entornos de programación, presionar la tecla 'flech
a hacia arriba' te mostrará comandos
| anteriores. Presiona esta tecla hasta que llegues al comando
(mi_variable <- (180 / 6) - 15), entonces
| cambia el número 6 por 60 y presiona ENTER. Si la tecla 'flec
ha hacia arriba' no funciona, sólo escribe el
| comando correcto.

> mi_variable <- (180 / 60) - 15

| ¡Buen trabajo!

  |=======================================================================
=================================== |  99%
| Por último, puedes teclear las dos primeras letras del nombre
de la variable y después presionar la tecla
| Tab (tabulador). La mayoría de los entornos de programación m
uestran una lista de las variables que has
| creado con el prefijo 'mi_'. Esta función se llama autocomple
tado y es muy útil para cuando tienes muchas
| variables en tu espacio de trabajo. Pruébalo, ingresa 'mi_' y
autocompleta. Si autocompletar no sirve en tu
| caso, sólo ingresa mi_variable en la línea de comandos).

> mi_variable
[1] -12

| ¡Lo has logrado! ¡Buen trabajo!

  |=======================================================================
=================================== | 100%
```

Lección de swirl: 3. Subconjuntos de Datos

```
Selection: 3
  |
|   0%

| En esta lección conocerás las maneras de acceder a las estruc
turas de datos en el lenguaje R.

...

  |==
|   2%
| R tiene una sintaxis especializada para acceder a las estruct
uras de datos.

...

  |====
|   4%
| Tú puedes obtener un elemento o múltiples elementos de una es
tructura de datos usando la notación de
| indexado de R.

...

  |=====
|   5%
| R provee diferentes maneras de referirse a un elemento (o con
junto de elementos) de un vector. Para probar
| estas diferentes maneras crea una variable llamada 'mi_vector
' que contenga un vector con los números
| enteros del 11 al 30. Recuerda que puedes usar el operador se
cuencia ':'.

>
> mi_vector <- 11:30

| ¡Eso es correcto!

  |=======
|   7%
| Y ahora ve su contenido.

> mi_variable
[1] -12

| No exactamente. Dele otra oportunidad. O escribe info() para
más opciones.

| Ingresa mi_vector en la línea de comandos.

> print(mi_variable)
[1] -12
```

```
| ¡Casi! Vuelve a intentar de nuevo. O escribe info() para más
opciones.

| Ingresa mi_vector en la línea de comandos.

> mi_vector
 [1] 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 3
0

| ¡Mantén este buen nivel!

  |=========
|   9%
| La manera más común de buscar un elemento en R es por medio d
e un vector numérico.

...

  |===========
|  11%
| Puedes buscar elementos por posición en un vector usando la s
iguiente notación: x[s], donde 'x' es un
| vector del cual deseas obtener elementos y 's' es un segundo
vector representando el conjunto de índices de
| elementos que te gustaría consultar.

...

  |============
|  12%
| Debes saber que en R las posiciones de los elementos de un ve
ctor comienzan en 1 y no en 0, como en
| lenguajes de programación como Java o C.

...

  |==============
|  14%
| Puedes usar un vector entero para buscar un simple elemento o
múltiples.

...

  |================
|  16%
| Por ejemplo, obten el tercer elemento de 'mi_vector'.

> mi_vector[3]
[1] 13

| ¡Eres el mejor!

  |==================
|  18%
| Ahora obten los primeros cinco elementos de 'mi_vector'.

>
```

```
> mi_vector[1]
[1] 11

| ¡Por poco lo tenía! Inténtalo de nuevo. O escribe info() para
más opciones.

| Introduce mi_vector[1:5] en la línea de comandos.

> mi_vector[1:5]
[1] 11 12 13 14 15

| ¡Eres bastante bueno!

  |==================
|  19%
| No necesariamente los índices deben ser consecutivos. Ingresa
mi_vector[c(4,6,13)] en la línea de comandos.

> mi_vector[c(4,6,13)]
[1] 14 16 23

| ¡Acertaste!

  |====================
|  21%
| Asimismo, no es necesario que los índices se encuentren orden
ados. Ingresa mi_vector[c(6,13,4)] en la línea
| de comandos.

> mi_vector[c(6,13,4)]
[1] 16 23 14

| ¡Lo estás haciendo muy bien!

  |======================
|  23%
| Como un caso especial, puedes usar la notación [[]] para refe
rirte a un solo elemento. Ingresa
| mi_vector[[3]] en la línea de comandos.

> mi_vector[[3]]
[1] 13

| Perseverancia es la respuesta.

  |========================
|  25%
| La notación [[]] funciona de la misma manera que la notación
[] en este caso.

...

  |=========================
|  26%
| También puedes usar enteros negativos para obtener un vector
que consista en todos los elementos, excepto
```

```
| los elementos especificados. Excluye los elementos 9:15, al e
specificar -9:-15.

>
> mi_vector[-9:-15]
 [1] 11 12 13 14 15 16 17 18 26 27 28 29 30

| ¡Es asombroso!

  |============================
|  28%
| Como alternativa a indexar con un vector de enteros, puedes i
ndexar a través de un vector lógico.

...

  |=============================
|  30%
| Como ejemplo crea un vector lógico de longitud 10 con valores
lógicos alternados, TRUE y FALSE
| (rep(c(TRUE,FALSE),10)), y consulta con él mi_vector[rep(c(TR
UE,FALSE),10)].

> mi_vector[rep(c(TRUE,FALSE),10)]
 [1] 11 13 15 17 19 21 23 25 27 29

| ¡Tu dedicación es inspiradora!

  |==============================
|  32%
| Como podrás notar, lo que ocurrió fue que indexaste únicament
e los elementos en las posiciones impares,
| puesto que creaste un vector con elementos TRUE en las posici
ones impares y FALSE en las pares.

...

  |===============================
|  33%
| El vector índice no necesita ser de la misma longitud que el
vector a indexar. R repetirá el vector más
| corto y regresará los valores que cacen. Ingresa mi_vector[c(
FALSE,FALSE,TRUE)] en la linea de comandos.

> mi_vector[c(FALSE,FALSE,TRUE)]
[1] 13 16 19 22 25 28

| ¡Mantén este buen nivel!

  |================================
|  35%
| Notarás que ahora indexaste los índices de los elementos múlt
iplos de 3.

...
```

```
  |=====================================
|  37%
| Es muy útil calcular un vector lógico de un mismo vector. Por
ejemplo, busca elementos más grandes que 20.
| Ingresa en la línea de comandos mi_vector > 20.

> mi_vector > 20
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[18]  TRUE  TRUE  TRUE

| ¡Mantén este buen nivel!

  |======================================
|  39%
| Y ahora indexa 'mi_vector' usando el vector previamente calcu
lado. Ingresa mi_vector[(mi_vector > 20)] en
| la línea de comandos.

> mi_vector[(mi_vector > 20)]
 [1] 21 22 23 24 25 26 27 28 29 30

| ¡Eso es correcto!

  |=======================================
|  40%
| También puedes usar esta notación para extraer partes de una
estructura de datos multidimensional.

...

  |=========================================
|  42%
| Un arreglo es un vector multidimensional. Vectores y arreglos
se almacenan de la misma manera internamente,
| pero un arreglo se muestra diferente y se accede diferente.

...

  |==========================================
|  44%
| Para crear un arreglo de dimensión 3x3x2 y de contenido los n
úmeros del 1 al 18 y guardarlo en la variable
| 'mi_arreglo', ingresa mi_arreglo <- array(c(1,2,3,4,5,6,7,8,9
,10,11,12,13,14,15,16,17,18),dim=c(3,3,2)) en
| la línea de comandos.

> mi_arreglo <- array(c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
17,18),dim=c(3,3,2))

| ¡Lo has logrado! ¡Buen trabajo!

  |===========================================
|  46%
| Ahora ve el contenido de la variable 'mi_arreglo'.

> mi_arreglo
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18


| ¡Eso es trabajo bien hecho!

  |=================================================
|  47%
| R tiene una manera muy limpia de referirse a parte de un arre
glo. Se especifican índices para cada
| dimensión, separados por comas. Ingresa mi_arreglo[1,3,2] en
la línea de comandos.

> mi_arreglo[1,3,2]
[1] 16

| ¡Mantén este buen nivel!

  |==================================================
|  49%
| Asimismo, puedes ingresar mi_arreglo[1:2,1:2,1] en la línea d
e comandos. ¡Inténtalo!

> mi_arreglo[1:2,1:2,1]
     [,1] [,2]
[1,]    1    4
[2,]    2    5

| ¡Lo estás haciendo muy bien!

  |===================================================
|  51%
| Una matriz es simplemente un arreglo bidimensional. Ahora cre
a una matriz con 3 renglones y 3 columnas con
| los números enteros del 1 al 9 y guárdala en la variable 'mi_
matriz'.

> mi_matriz <- matrix(1:9,ncol=3, nrow=3)

| ¡Traes una muy buena racha!

  |====================================================
|  53%
| Al igual que con los arreglos, para obtener todos los renglon
es o columnas de una dimensión de una matriz,
| simplemente omite los índices.
```

```
...

  |=======================================================
|  54%
| Por ejemplo, si quisiéras solo el primer renglón de 'mi_matri
z', basta con ingresar mi_matriz[1,] en la
| línea de comandos. ¡Inténtalo!

> mi_matriz[1,]
[1] 1 4 7

| ¡Tu dedicación es inspiradora!

  |=======================================================
|  56%
| ¡Ahora obtén solo la primera columna!

> mi_matriz[,1]
[1] 1 2 3

| ¡Muy bien!

  |=======================================================
|  58%
| También puedes referirte a un rango de renglones. Ingresa mi_
matriz[2:3,] en la línea de comandos.

> mi_matriz[2:3,]
     [,1] [,2] [,3]
[1,]    2    5    8
[2,]    3    6    9

| ¡Tu dedicación es inspiradora!

  |=======================================================
|  60%
| O referirte a un conjunto no contiguo de renglones. Ingresa m
i_matriz[c(1,3),] en la línea de comandos.

> mi_matriz[c(1,3),]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    3    6    9

| ¡Lo estás haciendo muy bien!

  |=======================================================
=                                                 |  61%
| En los ejemplos de arriba solo has visto estructuras de datos
basadas en un solo tipo. Recuerda que R tiene
| un tipo de datos incorporado para la mezcla de objetos de dif
erentes tipos, llamados listas.

...
```

```
  |=======================================================================
===                                                   |  63%
| Debes de saber que en R las listas son sutilmente diferentes
de las listas en muchos otros lenguajes. Las
| listas en R contienen una selección heterogénea de objetos. P
uedes nombrar cada componente en una lista.

...

  |=======================================================================
=====                                                 |  65%
| Los elementos en una lista pueden ser referidos por su ubicac
ión o por su nombre.

...

  |=======================================================================
=======                                               |  67%
| Ingresa este ejemplo de una lista con cuatro componentes nomb
rados carro <- list(color="rojo", nllantas=4,
| marca= "Renault", ncilindros=4).

> carro <- list(color="rojo", nllantas=4,marca= "Renault", ncil
indros=4)

| ¡Eso es trabajo bien hecho!

  |=======================================================================
========                                              |  68%
| Tú puedes acceder a los elementos de una lista de múltiples f
ormas. Puedes usar la misma notación que
| usaste con los vectores.

...

  |=======================================================================
==========                                            |  70%
| Y además puedes indexar un elemento por nombre usando la nota
ción $. Por ejemplo, ingresa carro$color en la
| línea de comandos.

> carro$color
[1] "rojo"

| Esa es la respuesta que estaba buscando.

  |=======================================================================
============                                          |  72%
| Además, puedes usar la notación [] para indexar un conjunto d
e elementos por nombre. Ingresa
| carro[c("ncilindros","nllantas")] en la línea de comandos.

> carro[c("ncilindros","nllantas")]
$`ncilindros`
[1] 4

$nllantas
```

```
[1] 4


| ¡Eso es trabajo bien hecho!

  |======================================================================
==============                                       |  74%
| También puedes indexar por nombre usando la notación [[]] cua
ndo seleccionas un simple elemento. Por
| ejemplo, ingresa carro[["marca"]] en la línea de comandos.

> carro[["marca"]]
[1] "Renault"

| ¡Lo estás haciendo muy bien!

  |======================================================================
===============                                      |  75%
| Hasta puedes indexar por nombre parcial usando la opción exac
t=FALSE. Ingresa carro[["mar",exact=FALSE]] en
| la línea de comandos.

> carro[["mar",exact=FALSE]]
[1] "Renault"

| ¡Eso es correcto!

  |======================================================================
=================                                    |  77%
| Ahora crea la siguiente lista: camioneta <- list(color="azul"
, nllantas=4, marca= "BMW", ncilindros=6).

> camioneta <- list(color="azul", nllantas=4, marca= "BMW", nci
lindros=6)

| ¡Mantén este buen nivel!

  |======================================================================
==================                                   |  79%
| Algunas veces una lista será una lista de listas. Ingresa coc
hera <- list(carro, camioneta).

> cochera <- list(carro, camioneta)

| ¡Excelente trabajo!

  |======================================================================
====================                                 |  81%
| Ahora ve el contenido de 'cochera'.

> cochera
[[1]]
[[1]]$`color`
[1] "rojo"

[[1]]$nllantas
[1] 4
```

```
[[1]]$marca
[1] "Renault"

[[1]]$ncilindros
[1] 4


[[2]]
[[2]]$`color`
[1] "azul"

[[2]]$nllantas
[1] 4

[[2]]$marca
[1] "BMW"

[[2]]$ncilindros
[1] 6



| ¡Eso es correcto!

  |======================================================================
======================                |  82%
| Tú puedes usar la notación [[]] para referirte a un elemento
en este tipo de estructura de datos. Para
| hacer esto usa un vector como argumento. R iterará a través d
e los elementos en el vector referenciando
| sublistas.

...

  |======================================================================
========================              |  84%
| Ingresar cochera[[c(2, 1)]] en la línea de comandos.

> cochera[[c(2, 1)]]
[1] "azul"

| ¡Lo estás haciendo muy bien!

  |======================================================================
==========================            |  86%
| Recuerda que los data frames son una lista que contiene múlti
ples vectores nombrados que tienen la misma
| longitud. A partir de este momento usarás el data frame cars
del paquete datasets. No te preocupes, este
| paquete viene cargado por defecto.

...

  |======================================================================
============================          |  88%
```

| Los datos que conforman al data frame cars son un conjunto de observaciones tomadas en la década de 1920;
| estas observaciones describen la velocidad (mph) de algunos carros y la distancia (ft) que les tomó parar.

...

  |==| 89%
| Ve el contenido del data frame cars. Ingresa cars en la línea de comandos.

```
> cars
   speed dist
1      4    2
2      4   10
3      7    4
4      7   22
5      8   16
6      9   10
7     10   18
8     10   26
9     10   34
10    11   17
11    11   28
12    12   14
13    12   20
14    12   24
15    12   28
16    13   26
17    13   34
18    13   34
19    13   46
20    14   26
21    14   36
22    14   60
23    14   80
24    15   20
25    15   26
26    15   54
27    16   32
28    16   40
29    17   32
30    17   40
31    17   50
32    18   42
33    18   56
34    18   76
35    18   84
36    19   36
37    19   46
38    19   68
39    20   32
40    20   48
41    20   52
42    20   56
43    20   64
```

```
44    22   66
45    23   54
46    24   70
47    24   92
48    24   93
49    24  120
50    25   85

| ¡Lo estás haciendo muy bien!

  |=======================================================================
================================       |  91%
| Te puedes referir a los elementos de un data frame (o a los e
lementos de una lista) por nombre usando el
| operador $. Ingresa cars$speed en la línea de comandos.

> cars$speed
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 1
4 14 14 14 15 15 15 16 16 17 17 17 18 18 18 18
[36] 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25

| ¡Toda esa práctica está rindiendo frutos!

  |=======================================================================
================================       |  93%
| Supón que deseas saber a qué velocidad iban los carros a los
que les tomó más de 100 pies (ft) frenar.

...

  |=======================================================================
================================       |  95%
| Una manera de encontrar valores específicos en un data frame
es al usar un vector de valores booleanos para
| especificar cuál o cuáles elementos regresar de la lista. La
manera de calcular el vector apropiado es así:
| cars$dist>100. ¡Inténtalo!

> cars$dist>100
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[18] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[35] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E FALSE FALSE FALSE FALSE  TRUE FALSE

| ¡Lo has logrado! ¡Buen trabajo!

  |=======================================================================
================================       |  96%
| Entonces puedes usar ese vector para referirte al elemento co
rrecto. Ingresa cars$speed[cars$dist>100] en
| la línea de comandos.

> cars$speed[cars$dist>100]
[1] 24
```

```
| ¡Todo ese trabajo está rindiendo frutos!

  |=======================================================================
=====================================  |  98%
| Ahora ya sabes cómo acceder a las estructuras de datos.

...

  |=======================================================================
=======================================| 100%
```

Lección de swirl: 4. Leer y Escribir Datos

```
Selection: 4
  |
|   0%

| En esta lección conocerás cómo cargar conjuntos de datos en R
y guardar estos conjuntos desde R.

...

  |==
|   2%
| Una de las mejores cosas acerca de R es lo fácil que es añadi
r información desde otros programas.

...

  |====
|   4%
| R puede importar conjuntos de datos desde archivos de texto,
otros softwares de estadística y hasta hojas
| de cálculo. No es necesario tener una copia local del archivo
. Tú puedes especificar la ubicación del
| archivo desde una url y R buscará el archivo en Internet.

...

  |=======
|   7%
| La mayoría de los archivos que contienen información tienen u
n formato similar. Generalmente cada línea del
| archivo representa una observación o registro, por lo que cad
a línea contiene un conjunto de diferentes
| variables asociadas con la observación.

...

  |=========
|   9%
| Algunas veces, diferentes variables son separadas por un cará
cter especial, llamado delimitador. Otra veces
| las variables son diferenciadas por su ubicación en cada líne
a.

...

  |===========
|  11%
| En esta lección trabajarás con el archivo inmigintnalpry.csv
el cual contiene la estimación de personas
| provenientes de otros países que llegan a cada uno de los est
ados de México. Si tienes suerte, el archivo
| se mostrará en algún editor; de lo contrario búscalo en el su
bdirectorio swirl_temp de tu directorio de
| trabajo y velo en una aplicación separada.
```

```
(Se ha copiado el archivo inmigintnalpry.csv a la ruta C:/Users
/Sourmak/Documents/swirl_temp/inmigintnalpry.csv ).
...

  |=============
|  13%
| Como podrás notar el primer renglón del archivo contiene los
nombres de las columnas, en este caso los
| nombres de cada una de las variables de la observación; adem
s, el archivo tiene delimitada cada variable
| de la observación por una coma.

...

  |===============
|  15%
| Para cargar este archivo a R, debes especificar que el primer
renglón contiene los nombres de las columnas
| y que el delimitador es una coma.

...

  |=================
|  17%
| Para hacer esto necesitarás especificar los argumentos header
y sep en la función read.table. Header para
| especificar que el primer renglón contiene los nombres de la
columnas (header=TRUE) y sep para especificar
| el delimitador (sep=",").

...

  |====================
|  20%
| ¡Importa el archivo inmigintnalpry.csv! Ingresa datos <- read
.table("swirl_temp/inmigintnalpry.csv",
| header=TRUE, sep=",", fileEncoding = "latin1") en la línea de
comandos.

> datos <- read.table("swirl_temp/inmigintnalpry.csv",header=TR
UE, sep=",", fileEncoding = "latin1")

| Perseverancia es la respuesta.

  |======================
|  22%
| Como podrás notar usaste el argumento fileEncoding; esto debi
do a que de no usarlo R no podría importar el
| archivo, puesto que la segunda cadena del archivo: año, no es
una cadena válida para el tipo de
| codificación que read.table usa por defecto. Para poder leer
el archivo basta con especificar el argumento
| fileEncoding. De no especificarlo R te indicará que hay un er
ror.

...
```

```
  |=======================
|  24%
| Intenta usar datos_2 <- read.table("swirl_temp/inmigintnalpry
.csv", header=TRUE, sep=","). Debido a que el
| archivo inmigintnalpry.csv contiene caracteres especiales com
o la ñ, R PUEDE MOSTRARTE UN ERROR. Si R te
| muestra el error, ingresa ok() en la línea de comandos para c
ontinuar.

> datos_2 <- read.table("swirl_temp/inmigintnalpry.csv", header
=TRUE, sep=",")

| Estás muy cerca... ¡Puedo sentirlo! Inténtalo de nuevo. O esc
ribe info() para más opciones.

| Ingresa datos2 <- read.table("swirl_temp/inmigintnalpry.csv",
header=TRUE, sep=","), después ingresa ok()
| para continuar.

> datos2 <- read.table("swirl_temp/inmigintnalpry.csv", header=
TRUE, sep=",")

| ¡Lo estás haciendo muy bien!

  |=========================
|  26%
| Este error es muy común cuando intentas leer archivos que su
contenido está en español; esto se debe a que
| usa otra codificación para poder abarcar más símbolos que no
usan otros idiomas, como en este caso la ñ.
| Para poder leer archivos que contengan ñ, basta con especific
ar el argumento fileEncoding, el cual indica
| la codificación del archivo a importar; en este caso, usarás
fileEncoding = "latin1".

...

  |==========================
|  28%
| Comúnmente las opciones más importantes son sep y header. Cas
i siempre debes saber el campo separador y si
| hay un campo header.

...

  |============================
|  30%
| Ahora ve lo que contiene 'datos'. Para hacer esto usarás la f
unción View(). Si te encuentras en Rstudio
| simplemente puedes presionar el nombre de la variable datos e
n el apartado Entorno ('Environment') y te
| mostrará su contenido. Presiona la variable datos en Rstudio
o ingresa View(datos) en la línea de comandos.

> View(datos)
```

| ¡Lo estás haciendo muy bien!

  |=================================
| 33%
| ¡Como podrás notar el archivo contiene 302060 observaciones!

...

  |==================================
| 35%
| Es importante saber que no solo existe read.table(). R además incluye un conjunto de funciones que llaman a
| read.table() con diferentes opciones por defecto para valores como sep y header, y algunos otros. En la
| mayoría de los casos encontrarás que puedes usar read.csv() para archivos separados por comas o
| read.delim() para archivos delimitados por TAB sin especificar otras opciones.

...

  |===================================
| 37%
| La mayoría de las veces deberías ser capaz de cargar archivos de texto en R con la función read.table().
| Pero algunas veces serás proveído con un archivo de texto que no pueda ser leído correctamente con esta
| función.

...

  |====================================
| 39%
| Si estás en Europa y usas comas para indicar punto decimal en los números, entonces puedes usar read.cvs2()
| y read.delim2().

...

  |=====================================
| 41%
| Una manera de agilizar la lectura de datos es usando el parámetro colClasses de la función read.table().

...

  |======================================
| 43%
| Este parámetro recibe un vector, el cual describe cada uno de los tipos por columna que va a leer. Esto
| agiliza la lectura debido a que read.table() normalmente lee toda la información y después revisa cada una
| de las columnas, y decide conforme a lo que vio de qué tipo es cada columna, y al indicar el parámetro
| colClasses le dices a la función read.table() de qué tipo son los datos que va a ver, con lo que te evitas
| el chequeo para saber el tipo de cada columna.

```
...

  |=============================================
|   46%
| Puedes averiguar la clase de las columnas de manera fácil cua
ndo tienes archivos grandes.

...

  |==============================================
|   48%
| Lo que puedes hacer es indicarle a read.table() que solo lea
los primeros 100 renglones del archivo; esto
| lo haces indicando el parámetro nrow. Cabe recordar que debes
especificar la codificación del archivo,
| debido a que usa caracteres especiales, también que el primer
renglón son los nombres de la columnas y que
| el delimitador es una coma. Ingresa inicial <- read.table("sw
irl_temp/inmigintnalpry.csv", header=TRUE,
| sep=",", fileEncoding = "latin1", nrow = 100) en la línea de
comandos.

> inicial <- read.table("swirl_temp/inmigintnalpry.csv", header
=TRUE,sep=",", fileEncoding = "latin1", nrow = 100)

| ¡Traes una muy buena racha!

  |===============================================
|   50%
| Con esto has conseguido leer las primeras 100 observaciones.

...

  |================================================
|   52%
| Después usa la función sapply mandándole como parámetros el o
bjeto inicial (el cual contiene las 100
| observaciones) y la función class(). Ingresa clases <- sapply
(inicial, class) en la línea de comandos.

> clases <- sapply(inicial, class)

| ¡Excelente trabajo!

  |=================================================
|   54%
| Con esto lo que conseguiste fue aplicar la función class() a
cada una de las columnas del objeto inicial.
| La función class() es una función que determina la clase o ti
po de un objeto. Entonces los tipos de cada
| una de las columnas fueron guardados en el objeto clases.

...

  |==================================================
|   57%
```

```
| Para ver el contenido del objeto 'clases', basta con escribir
clases en la línea de comandos.

> clases
    renglon         año         ent      id_ent      cvegeo
sexo        edad inmigintnal
  "integer"   "integer"    "factor"   "integer"   "integer"
"factor"   "integer"   "numeric"

| ¡Excelente!

  |=======================================================
|  59%
| Por último, con este vector de clases, leerás todo el archivo
usando la función read.table, pero pasándole
| el argumento colClasses. Ingresa datos <- read.table("swirl_t
emp/inmigintnalpry.csv", header=TRUE, sep=",",
| fileEncoding = "latin1", colClasses=clases) en la línea de co
mandos.

> datos <- read.table("swirl_temp/inmigintnalpry.csv", header=T
RUE, sep=",",fileEncoding = "latin1", colClasses=clases)

| ¡Eso es correcto!

  |=========================================================
=                                                 |  61%
| Como podrás notar el tiempo de lectura mejoró significativame
nte usando este truco.

...

  |===========================================================
===                                               |  63%
| Si deseas guardar objetos, la manera más simple es usando la
función save(). Por ejemplo, puedes usar el
| siguiente comando para salvar el objeto 'datos' y el objeto '
clases' en el archivo
| swirl_temp/datos_inmigrates.RData. Ingresa save(datos, clases
, file="swirl_temp/datos_inmigrates.RData") en
| la línea de comandos.

> save(datos, clases, file="swirl_temp/datos_inmigrates.RData")

| ¡Eso es correcto!

  |=============================================================
=====                                             |  65%
| La función save() escribe una representación externa de los o
bjetos especificados a un archivo señalado.
| Además, como ya te habrás dado cuenta, tú puedes guardar múlt
iples objetos en el mismo archivo, tan solo al
| listarlos en la función save().

...
```

```
  |=========================================================================
=======                                  |  67%
| Es importante notar que en R, las rutas de archivo siempre so
n especificadas con diagonales ("/"), aun
| estando en Microsoft Windows. Así que para salvar este archiv
o al directorio "C:\Documents and Settings\Mi
| Usuario\Mis Documentos\datos_inmigrates.RData, solo usarías e
l siguiente comando:
| save(datos,file="C:/Documents and Settings/Mi Usuario/Mis Doc
umentos/datos_inmigrates.RData").

...

  |=========================================================================
==========                               |  70%
| También es importante notar que el argumento file debe ser ex
plícitamente nombrado.

...

  |=========================================================================
============                             |  72%
| Ahora que has guardado los objetos 'datos' y 'clases' en un a
rchivo, puedes borrarlos. Introduce
| rm(datos,clases) en la línea de comandos.

> rm(datos,clases)

| ¡Excelente trabajo!

  |=========================================================================
==============                           |  74%
| Y si ahora usas la función ls(), la cual como recordarás mues
tra qué conjuntos de datos y funciones un
| usuario ha definido, verás que no están presentes los objetos
datos y clases. Ingresa ls() en la línea de
| comandos.

> ls()
 [1] "c"                   "camioneta"           "carro"
"cochera"
 [5] "colores"             "complejo"            "datos_2"
"datos2"
 [9] "display_swirl_file"  "fecha_primer_curso_R" "find_course
"           "inicial"
[13] "m"                   "mi_arreglo"          "mi_matriz"
"mi_variable"
[17] "mi_vector"           "ncol"                "ok"
"test1"
[21] "tf"                  "y"

| Perseverancia es la respuesta.

  |=========================================================================
================                         |  76%
| Ahora, puedes fácilmente cargar los objetos ‘datos’ y ‘clases
’ devuelta a R con la función load(). Solo
```

```
| debes especificar el nombre del archivo donde los guardaste.
Ingresa
| load("swirl_temp/datos_inmigrates.RData") en la línea de coma
ndos.

> load("swirl_temp/datos_inmigrates.RData")

| ¡Tu dedicación es inspiradora!

  |=======================================================================
==================                                    |  78%
| Y si ahora usas la función ls(), verás que están presentes lo
s objetos 'datos' y 'clases'. Ingresa ls() en
| la línea de comandos.

> ls()
 [1] "c"                    "camioneta"            "carro"
"clases"
 [5] "cochera"              "colores"              "complejo"
"datos"
 [9] "datos_2"              "datos2"               "display_swi
rl_file"   "fecha_primer_curso_R"
[13] "find_course"          "inicial"              "m"
"mi_arreglo"
[17] "mi_matriz"            "mi_variable"          "mi_vector"
"ncol"
[21] "ok"                   "test1"                "tf"
"y"

| ¡Mantén este buen nivel!

  |=======================================================================
====================                                  |  80%
| Es importante saber que los archivos guardados en R funcionar
án en todas las plataformas; es decir,
| archivos guardados en Linux funcionarán si son cargados desde
Windows o Mac OS X.

...

  |=======================================================================
======================                                |  83%
| Si deseas guardar cada uno de los objetos de tu espacio de tr
abajo (workspace), puedes hacerlo usando la
| función save.image(). De hecho, cuando salgas de la session d
e R, se te preguntará si deseas salvar tu
| actual espacio de trabajo (workspace). Si señalas que sí lo d
eseas, tu espacio de trabajo será guardado de
| la misma manera que usar esta función.

...

  |=======================================================================
========================                              |  85%
| Por último, al igual que para importar datos existe la funció
n read.table(), para exportar datos a un
| archivo de texto existe la función write.table().
```

```
...

  |=======================================================================
============================              |  87%
| Normalmente los datos a exportar son data frames y matrices.

...

  |=======================================================================
============================              |  89%
| Para exportar un objeto a un archivo basta con escribir la fu
nción write.table() y como argumento el nombre
| del objeto, además del nombre del archivo donde se guardará.
Ingresa write.table(datos,
| file="swirl_temp/datos.txt") en la línea de comandos.

> write.table(datos,
+             | file="swirl_temp/datos.txt")
Error: unexpected '|' in:
"write.table(datos,
            |"
> write.table(datos, file="swirl_temp/datos.txt")

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |=======================================================================
==============================            |  91%
| Si tienes suerte, te mostraré el archivo datos.txt en algún e
ditor; de lo contrario, búscalo en el
| subdirectorio swirl_temp de tu directorio de trabajo y velo e
n una aplicación separada.

...

  |=======================================================================
================================          |  93%
| Como podrás notar el archivo datos.txt no es igual al archivo
inmigintnalpry.csv que al inicio de esta
| lección te mostré. Una de las principales razones es que para
escribir el objeto datos no especificaste un
| delimitador (sep) y por defecto R delimitó con espacios.

...

  |=======================================================================
==================================        |  96%
| Al igual que con la función read.table(), R incluye un conjun
to de funciones que llaman a write.table() con
| diferentes opciones por defecto, como lo son write.csv() y wr
ite.csv2().

...

  |=======================================================================
====================================      |  98%
```

```
| Si deseas, puedes jugar con las funciones write.*() para logr
ar que datos.txt sea identico a
| inmigintnalpry.csv. Recuerda que para ver los parámetros de w
rite.*() puedes usar help(); por ejemplo,
| help(write.csv).

...

  |======================================================================
=======================================| 100%
```

Lección de swirl: 5. Funciones

```
Selection: 5
  |
|   0%

| En esta lección conocerás las funciones del lenguaje R.

...

  |===
|   3%
| En R las operaciones que hacen todo el trabajo son llamadas f
unciones.

...

  |======
|   6%
| Una función es un objeto en R, que puede tomar como entrada a
lgunos objetos (llamados argumentos de
| función) y puede regresar un objeto de salida.

...

  |=========
|   9%
| Las mayoría de las funciones son de la siguiente forma: f(arg
umento_1, argumento_2, ...). Donde f es el
| nombre de la función y argumento_1, argumento_2, ... son argu
mentos para la función.

...

  |============
|  12%
| Has usado alguna función anteriormente, ya que no se puede ha
cer nada interesante sin ellas. Todo el
| trabajo en R es hecho por funciones.

...

  |================
|  16%
| Una función que has estado usando a lo largo del curso es la
función c(), la cual crea un vector de los
| elementos que le sean pasados como argumentos. Por ejemplo in
troduce c(1, 03, 2016) en la línea de
| comandos.

>
> c(1, 03, 2016)
[1]    1    3 2016

| ¡Eres el mejor!
```

```
  |==================
|  19%
| La mayoría de las funciones en R regresan un valor; este valo
r puede ser calculado con base en el ambiente
| de la computadora o con base en la entrada (argumentos), como
en este caso, en donde el valor regresado es
| el vector que contiene a 1, 3 y 2016.

...

  |=====================
|  22%
| Cada inicialización de variables en R, operaciones aritmética
s, hasta repetir código en un loop, puede ser
| escrita como una función.

...

  |========================
|  25%
| Las funciones son creadas usando la función especial function
() y una vez creadas son guardadas como
| objetos de R de clase tipo function.

...

  |===========================
|  28%
| En la siguiente pregunta se te pedirá que modifiques un scrip
t. Las instrucciones de lo que debes hacer se
| encontrarán en el script. Una vez que hayas acabado de modifi
car el script, guarda tus cambios e ingresa
| submit() en la línea de comandos y así el script será evaluad
o. Si después de hacer esto la línea de
| comandos te dice que lo vuelvas a intentar y el script nuevam
ente aparece, esto se debe a que debes
| corregir tu script, siéntete libre de hacerlo, solo no olvide
s ingresar submit() cada vez que guardes tus
| cambios.

...

  |=============================
|  31%
| Generalmente el cuerpo de la función es encerrado entre llave
s {}, pero no es necesario si el cuerpo es una
| simple expresión. Por ejemplo, la expresión sucesor <- functi
on(x) x+1 es equivalente a la que se encuentra
| en el script.

> sucesor <- function(x) x+1
> sucesor <- function(x){} x+1
Error: unexpected symbol in "sucesor <- function(x){} x"
> sucesor <- function(x){x+1}
>
>
>
```

```
>
>
>
>
>
> sucesor <- function(x) {
+     #  x + 1
+ }
> sucesor <- function(x) {
+     x + 1
+ }
>
>
> {}
NULL
> test_func1()
Error in test_func1() : could not find function "test_func1"
> sucesor.R
Error: object 'sucesor.R' not found
> sucesor <- function(x) {x+1}
> sucesor.R
Error: object 'sucesor.R' not found
> sucesor.R
Error: object 'sucesor.R' not found
> test_func1()
Error in test_func1() : could not find function "test_func1"
> submit()

| Leyendo tu script...


| ¡Acertaste!

  |=================================
|  34%
| ¡Ahora que has creado tu primera función ¡pruébala! Ingresa s
ucesor(5) en la línea de comandos. Si tu
| función funciona, debería de regresar únicamente el valor 6.

>
> sucesor(5)
[1] 6

| ¡Traes una muy buena racha!

  |====================================
|  38%
| ¡Felicidades!, has escrito tu primera función.

...

  |=========================================
|  41%
| Es importante que sepas que si deseas ver el código fuente de
cualquier función, solo debes de teclear el
| nombre de la función sin argumentos ni paréntesis. Ahora ve e
l código fuente de la función que acabas de
```

```
| crear. Ingresa sucesor en la línea de comandos.

> sucesor
function(x) {
 x + 1
}
<bytecode: 0x0000000016cfb1d8>

| ¡Excelente!

  |=============================================
|  44%
| La definición de una función en R incluye los nombres de los
argumentos, como en el caso anterior que
| nombraste a 'x'.Si especificas un valor por defecto para un a
rgumento, entonces el argumento será
| considerado opcional.

...

  |=============================================
|  47%
| Ahora harás un función ligeramente más complicada, donde usar
ás argumentos por defecto. Crearás una función
| llamada diferencia_cuadrada(). Recuerda que para elevar un nú
mero a cierta potencia se usa el operador
| binario `^`. Asegúrate de guardar tus cambios antes de ingres
ar submit() en la línea de comandos.

> submit()

| Leyendo tu script...


| Eso no es precisamente lo que buscaba. Trata otra vez.

| Recuerda establecer el valor por defecto adecuado.

> submit()

| Leyendo tu script...


| ¡Sigue intentando!

| Recuerda establecer el valor por defecto adecuado.

> submit()

| Leyendo tu script...


| Por poco era correcto, sigue intentándolo.

| Recuerda establecer el valor por defecto adecuado.

> submit()
```

```
| Leyendo tu script...

| Estás muy cerca... ¡Puedo sentirlo! Inténtalo de nuevo.

| Recuerda establecer el valor por defecto adecuado.

> submit()

| Leyendo tu script...

| ¡No tan bien, pero estás aprendiendo! Intenta de nuevo.

| Recuerda establecer el valor por defecto adecuado.

> submit()

| Leyendo tu script...

| ¡Sigue intentando!

| Recuerda establecer el valor por defecto adecuado.

> submit()

| Leyendo tu script...

| ¡Mantén este buen nivel!

  |=================================================
|  50%
| Ahora prueba tu función diferencia_cuadrada(). Ingresa difere
ncia_cuadrada(3) en la línea de comandos.

> diferencia_cuadrada(3)
[1] 5

| ¡Eres bastante bueno!

  |==================================================
|  53%
| ¿Qué ha pasado? Como proveíste un solo argumento a la función
, R cazó ese argumento a 'x', debido a que 'x'
| es el primer argumento. Por lo que 'y' usó el valor por defec
to que definiste (2).

...

  |==================================================
|  56%
| Recordarás que en un llamada a función puedes sobrescribir lo
s valores por defecto. Así que ahora prueba
```

```
| diferencia_cuadrada() con dos argumentos. Ingresa diferencia_
cuadrada(10, 5) en la línea de comandos.

> diferencia_cuadrada(10, 5)
[1] 75

| ¡Eso es trabajo bien hecho!

  |=============================================================
|  59%
| En R puedes explícitamente nombrar a los argumentos. Por ejem
plo ingresa diferencia_cuadrada(y = 10, x = 5)
| en la línea de comandos.

> diferencia_cuadrada(y = 10, x = 5)
[1] -75

| ¡Mantén este buen nivel!

  |=============================================================
==                                                   |  62%
| Como podrás notar es diferente ingresar diferencia_cuadrada(1
0, 5) a diferencia_cuadrada(y = 10, x = 5).

...

  |=============================================================
======                                               |  66%
| R también caza parcialmente los argumentos; es decir, ingresa
r diferencia_cuadrada(10, y = 5) resulta en lo
| mismo que ingresar diferencia_cuadrada(x = 10, y = 5) o difer
encia_cuadrada(10, 5).

...

  |=============================================================
=========                                            |  69%
| Si no especificas un valor por defecto para un argumento, y s
i no especificas el valor de ese argumento
| cuando llamas a la función, obtendrás un error si la función
intenta usar ese argumento.

...

  |=============================================================
============                                         |  72%
| Si deseas escribir una función que acepte un número variable
de argumentos, en R puedes usar '...'; para
| hacer esto se especifica '...' en los argumentos de la funció
n.

...

  |=============================================================
===============                                      |  75%
| Ahora escribirás una función usando '...'. Cerciórate de guar
dar tus cambios en el script antes de que
```

```
| introduzcas submit().

> submit()

| Leyendo tu script...


| ¡Excelente!

  |=======================================================================
==================                                    |  78%
| Ahora prueba tu función numeros_por_vocales. Usa la función n
umeros_por_vocales pasándole como argumentos
| las cadenas que desees.

> numeros_por_vocales('hahas')
[1] "h4h4s"

| ¡Toda esa práctica está rindiendo frutos!

  |=======================================================================
====================                                  |  81%
| Muchas funciones en R pueden recibir otras funciones como arg
umentos. Por ejemplo, si deseas saber los
| argumentos de una función puedes hacer uso de las funciones a
rgs() o formals(), las cuales reciben como
| argumento el nombre de la función de la que deseas conocer lo
s argumentos.

...

  |=======================================================================
======================                                |  84%
| Ahora muestra los argumentos de la función mean(), la cual re
gresa el promedio de los elementos que recibe
| como argumentos. Usa cualquiera de la funciones antes mencion
adas.

> formals(mean)
$x


$...



| ¡Eres bastante bueno!

  |=======================================================================
==========================                            |  88%
| Es importante que sepas que la función args() es usada princi
palmente de modo interactivo para imprimir los
| argumentos de una función. Para uso en programación considera
mejor usar formals().

...
```

```
  |=======================================================================
================================         |  91%
| El concepto de pasar funciones como argumentos es muy poderos
o. Completa la función operador_binario() para
| ver cómo funciona. Recuerda guardar tus cambios en el script
antes de que introduzcas submit().

> submit
function ()
{
    invisible()
}
<bytecode: 0x000000001f3918a0>
<environment: namespace:swirl>
> submit()

| Leyendo tu script...


| ¡Sigue trabajando de esa manera y llegarás lejos!

  |=======================================================================
================================       |  94%
| Ahora prueba tu función operador_binario(). Ingresa operador_
binario(`%/%`, 7, 3) en la línea de comandos.
| Recuerda que el operador `%/%` no es más que la división ente
ra en R.

> operador_binario(`%/%`, 7, 3)
[1] 2

| ¡Toda esa práctica está rindiendo frutos!

  |=======================================================================
================================     |  97%
| Por último, recuerda que todas las funciones en R regresan un
valor. Algunas funciones en R además hacen
| otras cosas, como cambiar el estado de las variables, grafica
r, cargar o guardar archivos, o hasta acceder
| a la red.

...

  |=======================================================================
=================================| 100%
```

Lección de swirl: 6. Funciones apply

```
Selection: 6
  |
|   0%

| En esta lección aprenderás a utilizar a la familia de funcion
es *apply().

...

  |==
|   2%
| Cuando te encuentras procesando información, una operación co
mún es la de aplicar una función a un conjunto
| de elementos y regresar un nuevo conjunto de elementos.

...

  |===
|   3%
| La funciones *apply() que se encuentran en el paquete base de
R, actúan de esta manera sobre diferentes
| estructuras de datos, aplican una función con uno o varios ar
gumentos y regresan otra estructura de datos.

...

  |=====
|   5%
| La primera función que conocerás es la función apply(), la cu
al opera sobre arreglos.

...

  |=======
|   7%
| Para conocer el uso de esta función ingresa help("apply") en
la línea de comandos.

> help("apply")

| ¡Toda esa práctica está rindiendo frutos!

  |========
|   8%
| X es un arreglo (puede ser una matriz si la dimensión del arr
eglo es 2). MARGIN es una variable que define
| cómo la función es aplicada, cuando MARGIN=1 se aplica sobre
los renglones, cuando MARGIN=2 trabaja sobre
| las columnas. FUN es la función que deseas aplicar y puede se
r cualquier función de R, incluyendo funciones
| definidas por ti.

...
```

```
  |==========
|  10%
| Opcionalmente puedes especificar argumentos a FUN como argume
ntos adicionales (...).

...

  |===========
|  11%
| Para ejemplificar cómo funciona crea un arreglo bidimensional
(matriz) y guárdalo en la variable
| 'mi_matriz'. Ingresa mi_matriz <- matrix(data=1:16,nrow=4, nc
ol=4) en la línea de comandos.

> mi_matriz <- matrix(data=1:16,nrow=4, ncol=4)

| ¡Excelente trabajo!

  |=============
|  13%
| Ahora ve su contenido.

> mi_matriz
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

| ¡Toda esa práctica está rindiendo frutos!

  |===============
|  15%
| Ahora imagina que deseas saber el mínimo valor presente en ca
da columna de 'mi_matriz'. Puedes usar la
| funcion min(), la cual regresa el mínimo de los valores que r
ecibe como entrada; y así, al ingresar
| apply(X=mi_matriz, MARGIN=2, FUN=min) en la línea de comandos
obtendrás los mínimos valores por columna.
| ¡Inténtalo!

> apply(mi_matriz, 2, min)
[1]  1  5  9 13

| ¡Eres bastante bueno!

  |================
|  16%
| Es importante notar que es necesario que MARGIN=2 para que la
función min() sea aplicada sobre las
| columnas.

...

  |=================
|  18%
| Ahora haz la misma operación, solo que sobre los renglones.
```

```
> apply(mi_matriz, 1, min)
[1] 1 2 3 4

| ¡Buen trabajo!

  |====================
|  20%
| Te preguntarás qué puedes hacer si deseas aplicar una función
a una lista o un vector, ya que apply() solo
| opera sobre arreglos.

...

  |=====================
|  21%
| Pues bien, para aplicar una función a cada elemento de un vec
tor o una lista y regresar una lista, puedes
| usar la función lapply().

...

  |======================
|  23%
| Para ejemplificar esto crea una lista, la cual contenga a las
cadenas "Introducción", "a", "la",
| "Programación", "Estadística", "con", "R", en ese orden, y gu
arda la lista en la variable ‘mi_lista’.

> mi_lista <- list("Introducción", "a", "la", "Programación", "
Estadística", "con", "R")

| ¡Todo ese trabajo está rindiendo frutos!

  |=======================
|  25%
| Ahora ve el contenido de la variable 'mi_lista'.

> mi_lista
[[1]]
[1] "Introducción"

[[2]]
[1] "a"

[[3]]
[1] "la"

[[4]]
[1] "Programación"

[[5]]
[1] "Estadística"

[[6]]
[1] "con"
```

```
[[7]]
[1] "R"


| ¡Excelente trabajo!

  |=========================
|  26%
| Aunque la función lapply() es muy parecida a la función apply
(), es importante conocer la manera de usarla;
| esto lo puedes hacer conociendo sus argumentos. Ahora ve los
argumentos de la función lapply().

> formals(lapply)
$X


$FUN


$...



| ¡Sigue trabajando de esa manera y llegarás lejos!

  |==========================
|  28%
| Como podrás notar el uso de lapply() es más simple que el de
apply, puesto que no hace uso del argumento
| MARGIN.

...

  |==========================
|  30%
| Ahora ve un simple ejemplo de cómo usar lapply. Ingresa mayus
culas <- lapply(mi_lista, toupper) en la línea
| de comandos.

> mayusculas <- lapply(mi_lista, toupper)

| Esa es la respuesta que estaba buscando.

  |==========================
|  31%
| Ahora ve el contenido de 'mayusculas'.

> mayusculas
[[1]]
[1] "INTRODUCCIÓN"

[[2]]
[1] "A"

[[3]]
[1] "LA"
```

```
[[4]]
[1] "PROGRAMACIÓN"

[[5]]
[1] "ESTADÍSTICA"

[[6]]
[1] "CON"

[[7]]
[1] "R"


| ¡Traes una muy buena racha!

  |=================================
|  33%
| Como recordarás, toupper() regresa la cadena que reciba como
entrada en mayúsculas. Así que lapply
| simplemente regresa una lista que contiene el resultado de ap
licar la función toupper() a cada elemento de
| 'mi_lista'.

...

  |=================================
|  34%
| Para verificar que lapply efectivamente regresa una lista ing
resa class(mayusculas).

>
> class(mayusculas)
[1] "list"

| ¡Bien hecho!

  |==================================
|  36%
| Si ahora ingresas lapply(c("Introduccion", "a", "la", "Progra
macion", "Estadistica", "con", "R"), toupper)
| en la línea de comandos, ¿qué crees que ocurra?

1: Que te regrese una lista que contenga las cadenas "INTRODUCC
ION", "A", "LA", "PROGRAMACION", "ESTADISTICA", "CON", "R"
2: Que te regrese un vector que contenga las cadenas "INTRODUCC
ION", "A", "LA", "PROGRAMACION", "ESTADISTICA", "CON", "R"
3: Que te mande error

Selection: 1

| ¡Lo estás haciendo muy bien!

  |==================================
|  38%
| Te regresará una lista que contenga a las cadenas "INTRODUCCI
ON", "A", "LA", "PROGRAMACION", "ESTADISTICA",
```

| "CON", "R" pues si recuerdas la función lapply() opera sobre listas y vectores y SIEMPRE regresa una lista;
| es fácil que recuerdes esto si piensas que lista + apply = lapply.

...

  |=======================================
| 39%
| A partir de ahora trabajarás con el archivo ASA_estadisticasPasajeros(3).csv, el cual contiene datos
| estadísticos del año 2015 de pasajeros en servicio nacional e internacional de la Red Aeropuertos y
| Servicios Auxiliares de México. Con suerte, el archivo se mostrará en algún editor. De lo contrario,
| búscalo en el subdirectorio swirl_temp, de tu directorio de trabajo y velo en una aplicación separada.

(Se ha copiado el archivo ASA_estadisticasPasajeros(3).csv a la ruta C:/Users/Sourmak/Documents/swirl_temp/ASA_estadisticasPasajeros(3).csv).
...

  |==
| 41%
| Como podrás notar el primer renglón del archivo contiene los nombres de las columnas. Anio mes son
| representados por una cadena que contiene año y mes pegados sin espacios; Código IATA se refiere a la sigla
| utilizada por IATA(International Air Transport Association) para identificar al aeropuerto; Descripción
| contiene la ciudad donde se encuentra dicho aeropuerto; Estado, como su nombre lo indica, contiene el
| nombre del estado donde se encuentra dicho aeropuerto; Pasajeros nacionales se refiere al número de
| pasajeros mexicanos que hicieron uso del aeropuerto; y Pasajeros internacionales se refiere al número de
| pasajeros extranjeros que hicieron uso del aeropuerto.

...

  |===
| 43%
| Ahora importa el archivo ASA_estadisticasPasajeros(3).csv usando alguna función read.*() y guárdalo en la
| variable 'asa_datos'. Recuerda que se encuentra en tu directorio de trabajo, en la carpeta swirl_temp, por
| lo que la ruta del archivo es "swirl_temp/ASA_estadisticasPasajeros(3).csv"

warning messages from top-level task callback 'mini'
Warning message:
In file.copy(loc, "swirl_temp", overwrite = TRUE) :
  problem copying C:\Users\Sourmak\Documents\R\win-library\3.5\swirl\Courses\programacion-estadistica-r\Funciones_apply\ASA_estadisticasPasajeros(3).csv to swirl_temp\ASA_estadisticasPasajeros(3).csv: Permission denied

```
> asa_datos <- read.csv("swirl_temp/ASA_estadisticasPasajeros(3
).csv")

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |=========================================
|  44%
| Ahora ve lo que contiene 'asa_datos'. Para hacer esto usarás
la función View(). Si te encuentras en Rstudio
| simplemente puedes presionar el nombre de tu variable asa_dat
os en el apartado Entorno ("Environment") y se
| mostrará su contenido. Presiona la variable asa_datos en Rstu
dio o ingresa en la línea de comando:
| View(asa_datos).

> View(asa_datos)

| ¡Buen trabajo!

  |==========================================
|  46%
| Como recordarás la familia de funciones read.*() te regresan
un data frame.

...

  |==========================================
|  48%
| Los data frames no son más que una lista de vectores, así que
puedes usar la función lapply() con ellos.

...

  |==========================================
|  49%
| Es importante que cuando trabajes con datos sepas un poco más
de ellos.

...

  |==========================================
|  51%
| Si deseas saber el tipo de variables que contiene cada column
a del data frame 'asa_datos' basta con
| escribir lapply(asa_datos, class). Ingresa lapply(asa_datos,
class) en la línea de comandos.

> lapply(asa_datos, class)
$`Anio.mes`
[1] "integer"

$Codigo.IATA
[1] "factor"

$Descripcion
[1] "factor"
```

```
$Estado
[1] "factor"

$Pasajeros.nacionales
[1] "integer"

$Pasajeros.internacionales
[1] "integer"


| ¡Eso es trabajo bien hecho!

  |=======================================================
|  52%
| Como podrás notar, el data frame contiene dos tipos de datos:
enteros (Anio.mes, Pasajeros.nacionales y
| Pasajeros.internacionales) y factores (Codigo.IATA, Descripci
on y Estado).

...

  |=======================================================
|  54%
| Como recordarás los factores son una colección ordenada de el
ementos. Son usados en R para representar
| valores categóricos. Si haces un poco de memoria recordarás q
ue los valores que un factor puede tomar se
| llaman niveles ("levels").

...

  |=======================================================
|  56%
| A la hora de trabajar con factores es importante que conozcas
los niveles que los datos pueden tomar.

...

  |=======================================================
|  57%
| Si deseas conocer los niveles que las columnas del data frame
pueden tomar, los puedes conocer al imprimir
| dicha columna. Por ejemplo, ingresa asa_datos$Descripcion en
la línea de comandos para conocer las ciudades
| que contienen aeropuertos pertenecientes a la Red Aeropuertos
y Servicios Auxiliares de México.

> asa_datos$Descripcion
  [1] Ciudad Obregon    Colima            Ciudad del Carmen Cam
peche          Chetumal
  [6] Ciudad Victoria   Guaymas           Loreto            Mat
amoros         Nuevo Laredo
 [11] Nogales           Poza Rica         Puebla            Pue
rto Escondido  Tehuacan
 [16] Tepic             Tamuin            Uruapan           Ciu
dad Obregon    Colima
```

```
 [21] Ciudad del Carmen Campeche          Chetumal          Ciu
dad Victoria   Guaymas
 [26] Loreto             Matamoros        Nuevo Laredo      Nog
ales           Poza Rica
 [31] Puebla             Puerto Escondido  Tehuacan         Tep
ic             Tamuin
 [36] Uruapan            Ciudad Obregon    Colima           Ciu
dad del Carmen Campeche
 [41] Chetumal           Ciudad Victoria   Guaymas          Lor
eto            Matamoros
 [46] Nuevo Laredo       Nogales           Poza Rica        Pue
bla            Puerto Escondido
 [51] Tehuacan           Tepic             Tamuin           Uru
apan           Ciudad Obregon
 [56] Colima             Ciudad del Carmen Campeche         Che
tumal          Ciudad Victoria
 [61] Guaymas            Loreto            Matamoros        Nue
vo Laredo      Nogales
 [66] Poza Rica          Puebla            Puerto Escondido  Teh
uacan          Tepic
 [71] Tamuin             Uruapan           Ciudad Obregon    Col
ima            Ciudad del Carmen
 [76] Campeche           Chetumal          Ciudad Victoria   Gua
ymas           Loreto
 [81] Matamoros          Nuevo Laredo      Nogales           Poz
a Rica         Puebla
 [86] Puerto Escondido  Tehuacan           Tepic             Tam
uin            Uruapan
 [91] Ciudad Obregon    Colima             Ciudad del Carmen Cam
peche          Chetumal
 [96] Ciudad Victoria   Guaymas            Loreto            Mat
amoros         Nuevo Laredo
[101] Nogales           Poza Rica         Puebla            Pue
rto Escondido  Tehuacan
[106] Tepic             Tamuin            Uruapan           Ciu
dad Obregon    Colima
[111] Ciudad del Carmen Campeche          Chetumal          Ciu
dad Victoria   Guaymas
[116] Loreto            Matamoros         Nuevo Laredo      Nog
ales           Poza Rica
[121] Puebla            Puerto Escondido  Tehuacan          Tep
ic             Tamuin
[126] Uruapan           Ciudad Obregon    Colima            Ciu
dad del Carmen Campeche
[131] Chetumal          Ciudad Victoria   Guaymas           Lor
eto            Matamoros
[136] Nuevo Laredo      Nogales           Poza Rica         Pue
bla            Puerto Escondido
[141] Tehuacan          Tepic             Tamuin            Uru
apan           Ciudad Obregon
[146] Colima            Ciudad del Carmen Campeche          Che
tumal          Ciudad Victoria
[151] Guaymas           Loreto            Matamoros         Nue
vo Laredo      Nogales
[156] Poza Rica         Puebla            Puerto Escondido  Teh
uacan          Tepic
```

```
[161] Tamuin            Uruapan           Ciudad Obregon    Col
ima            Ciudad del Carmen
[166] Campeche          Chetumal          Ciudad Victoria   Gua
ymas           Loreto
[171] Matamoros         Nuevo Laredo      Nogales           Poz
a Rica         Puebla
[176] Puerto Escondido  Tehuacan          Tepic             Tam
uin            Uruapan
[181] Ciudad Obregon    Colima            Ciudad del Carmen Cam
peche          Chetumal
[186] Ciudad Victoria   Guaymas           Loreto            Mat
amoros         Nuevo Laredo
[191] Nogales           Poza Rica         Puebla            Pue
rto Escondido  Tehuacan
[196] Tepic             Tamuin            Uruapan           Ciu
dad Obregon    Colima
[201] Ciudad del Carmen Campeche          Chetumal          Ciu
dad Victoria   Guaymas
[206] Loreto            Matamoros         Nuevo Laredo      Nog
ales           Poza Rica
[211] Puebla            Puerto Escondido  Tehuacan          Tep
ic             Tamuin
[216] Uruapan
18 Levels: Campeche Chetumal Ciudad del Carmen Ciudad Obregon C
iudad Victoria Colima Guaymas ... Uruapan

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |=======================================================
|  59%
| Como podrás notar esto presentó todos los valores de la colum
na Descripcion aun estando repetidos. Al final
| se muestran todos los niveles en donde dice "Levels:" pero al
ser muchos R mostró algunos y los demás los
| omitió usando "...".

...

  |=======================================================
=                                          |  61%
| Una manera de poder ver todos los niveles aun cuando sean muc
hos es usando la función unique(). La función
| unique() elimina duplicados. Ingresa unique(asa_datos$Descrip
cion) en la línea de comandos.

>
> unique(asa_datos$Descripcion)
 [1] Ciudad Obregon    Colima            Ciudad del Carmen Camp
eche          Chetumal
 [6] Ciudad Victoria   Guaymas           Loreto            Mata
moros         Nuevo Laredo
[11] Nogales           Poza Rica         Puebla            Puer
to Escondido  Tehuacan
[16] Tepic             Tamuin            Uruapan
18 Levels: Campeche Chetumal Ciudad del Carmen Ciudad Obregon C
iudad Victoria Colima Guaymas ... Uruapan
```

```
| ¡Traes una muy buena racha!

  |=======================================================================
==                                                    |  62%
| Ahora ya conoces las ciudades con aeropuertos pertenecientes
a la red ASA.

...

  |=======================================================================
====                                                  |  64%
| Repite el proceso pero ahora para conocer los valores de la c
olumna Estado.

> unique(asa_datos$Estado)
 [1] Sonora               Colima               Campeche
Quintana Roo         Tamaulipas
 [6] Baja California  Sur Veracruz             Puebla
Oaxaca               Nayarit
[11] San Luis Potosi      Michoacan
12 Levels: Baja California  Sur Campeche Colima Michoacan Nayar
it Oaxaca Puebla ... Veracruz

| ¡Toda esa práctica está rindiendo frutos!

  |=======================================================================
======                                                |  66%
| Ahora deseas saber el número total de pasajeros nacionales qu
e viajaron en alguno de los aeropuertos
| pertenecientes a la red ASA. Para hacer esto simplemente pued
es sumar todos los elementos de la columna de
| Pasajeros.nacionales. Ingresa sum(asa_datos$Pasajeros.naciona
les) en la línea de comandos.

> sum(asa_datos$Pasajeros.nacionales)
[1] 2291419

| ¡Excelente!

  |=======================================================================
=======                                               |  67%
| La función sum() regresa la suma de todos los elementos que r
ecibe como argumentos.

...

  |=======================================================================
=========                                             |  69%
| Ahora deseas obtener el número total de pasajeros internacion
ales.

...

  |=======================================================================
==========                                            |  70%
| Podrías repetir la operación que hiciste para obtener el núme
ro total de pasajeros nacionales...
```

```
...

  |======================================================================
============                                          |  72%
| O podrías usar la función lapply() y automatizar la operación
para obtener el número total de ambos
| pasajeros.

...

  |======================================================================
==============                                        |  74%
| Primero crea un subconjunto de asas_datos, ingresa asa_pasaje
ros <- asa_datos[,c("Pasajeros.nacionales",
| "Pasajeros.internacionales")] en la línea de comandos. Con es
to obtendrás las columnas de pasajeros.

> asa_pasajeros <- asa_datos[,c("Pasajeros.nacionales", "Pasaje
ros.internacionales")]

| ¡Eso es trabajo bien hecho!

  |======================================================================
===============                                       |  75%
| Ahora ve lo que contiene 'asa_pasajeros'. Para hacer esto usa
rás la función View(). Si te encuentras en
| Rstudio simplemente puedes presionar el nombre de la variable
asa_pasajeros en el apartado Entorno
| ("Environment") y se mostrará su contenido. Presiona la varia
ble asa_pasajeros en Rstudio o ingresa
| View(asa_pasajeros) en la línea de comandos.

> View(asa_pasajeros)

| ¡Excelente!

  |======================================================================
================                                      |  77%
| Y ahora usa la función lapply() para obtener el número total
de pasajeros de ambas columna. Ingresa
| lapply(asa_pasajeros, sum) en la línea de comandos.

> lapply(asa_pasajeros, sum)
$`Pasajeros.nacionales`
[1] 2291419

$Pasajeros.internacionales
[1] 176741


| ¡Excelente!

  |======================================================================
==================                                    |  79%
| Esto te dice que se registraron un total de 2 291 419 pasajer
os nacionales y 176 741 pasajeros
```

```
| internacionales.

...

  |=======================================================================
====================                                  |  80%
| Como recordarás, las listas son útiles para guardar objetos d
e múltiples clases. Pero en este caso el
| resultado de aplicar sum() fue un entero por columna; podrías
guardar el resultado en alguna otra
| estructura.

...

  |=======================================================================
======================                                |  82%
| sapply() te permite simplificar este proceso; sapply funciona
como lapply(), pero intenta simplificar la
| salida a la estructura de datos más elemental que sea posible
. De ahí su nombre simple + apply = sapply.

...

  |=======================================================================
========================                              |  84%
| Ahora usa sapply() para obtener el número total de pasajeros
nacionales e internacionales de la misma
| manera que lo hiciste con lapply() y guarda el resultado en l
a variable 'total_pasajeros'.

> total_pasajeros <- sapply(asa_pasajeros, sum)

| ¡Toda esa práctica está rindiendo frutos!

  |=======================================================================
==========================                            |  85%
| Ahora ve el contenido de 'total_pasajeros'.

> total_pasajeros
     Pasajeros.nacionales Pasajeros.internacionales
                  2291419                    176741

| ¡Eres bastante bueno!

  |=======================================================================
============================                          |  87%
| Como notarás 'total_pasajeros' no es una lista; esto se debe
a que a sapply() simplificó el resultado a un
| vector de enteros.

...

  |=======================================================================
==============================                        |  89%
| En general, si el resultado de lapply() es una lista donde ca
da elemento es de longitud 1, sapply()
```

```
| regresará un vector. Si el resultado es una lista donde cada
elemento es un vector de la misma longitud (>
| 1), sapply() regresará una matriz. Si sapply() no puede arreg
lárselas, entonces regresará una lista, lo
| cual no sería diferente al resultado de usar lapply().

...

  |=======================================================
==============================          |  90%
| Algunas veces encontrarás que los datos proporcionados tienen
una granularidad muy fina para el tipo de
| análisis que estás realizando. Por ejemplo supón que deseas s
aber el número total de pasajeros nacionales
| por estado. Para encontrar ese número, tendrás que agrupar lo
s aeropuertos por estado e ir sumando los
| números de pasajeros nacionales. Esta tarea puede volverse un
poco conflictiva debido a que la información
| la tienes dividida por meses.

...

  |=======================================================
==============================          |  92%
| Como recordarás, la columna Estados solo puede tomar los valo
res Sonora, Colima, Campeche, Quintana Roo,
| Tamaulipas, Baja California Sur, Veracruz, Puebla, Oaxaca, Na
yarit, San Luis Potosí y Michoacán. Para ver
| cuántos registros tiene cada estado ingresa table(asa_datos$E
stado) en la línea de comandos.

> table(asa_datos$Estado)

Baja California  Sur             Campeche                Colima
Michoacan               Nayarit
                  12                   24                    12
12                   12
              Oaxaca                Puebla         Quintana Roo
San Luis Potosi                 Sonora
                  12                   24                    12
12                   36
          Tamaulipas             Veracruz
                  36                   12

| ¡Toda esa práctica está rindiendo frutos!

  |=======================================================
==============================          |  93%
| Como te podrás dar cuenta cada estado tiene un número múltipl
o de 12 registros; esto se debe a que los
| registros por aeropuerto están divididos por mes.

...

  |=======================================================
==============================          |  95%
```

```
| Pensando en que algunas veces tendrás la información con una
granularidad muy fina para el tipo de análisis
| que deseas realizar, R proporciona la función tapply(), la cu
al divide datos en grupos, basados en valor de
| alguna variable y luego aplica la función especificada a los
miembros de cada grupo.

...

  |=======================================================
=====================================   |  97%
| Ingresa tapply(asa_datos$Pasajeros.nacionales, asa_datos$Esta
do, sum) para obtener el número de pasajeros
| nacionales por estado.

> tapply(asa_datos$Pasajeros.nacionales, asa_datos$Estado, sum)
Baja California  Sur             Campeche               Colima
Michoacan              Nayarit
              12602               769864               112656
95635               113043
             Oaxaca               Puebla         Quintana Roo
San Luis Potosi               Sonora
             181706               267567               179259
1598               253788
          Tamaulipas              Veracruz
             243126                60575

| ¡Eres el mejor!

  |=======================================================
=====================================  |  98%
| Ahora obtén el promedio o media de pasajeros nacionales que v
iajaron por mes en cada aeropuerto. Recuerda
| que para calcular la media puedes usar la función mean().

> tapply(asa_datos$Pasajeros.nacionales, asa_datos$Codigo.IATA,
mean)
       CEN         CLQ         CME         CPE         CTM         C
VM         GYM         LTO         MAM         NLD
19867.4167  9388.0000 49262.8333 14892.5000 14938.2500  6124.25
00  1063.0833  1050.1667  8078.9167  6057.3333
       NOG         PAZ         PBC         PXM         TCN         T
PQ         TSL         UPN
  218.5000  5047.9167 22062.9167 15142.1667   234.3333  9420.25
00   133.1667  7969.5833

| ¡Eres el mejor!

  |=======================================================
=======================================| 100%
```

Lección de swirl: 7. Graficación

```
Selection: 7
  |
|   0%

| En esta lección conocerás el sistema base de graficación en R
.

...

  |=
|   1%
| Si estás familiarizado con Microsoft Excel, encontrarás que R
puede generar todas las gráficas con las que
| estás familiarizado: gráficas de pastel, gráficas de barras,
etc. Además, hay muchos más tipos de gráficas
| disponibles en R.

...

  |===
|   3%
| Para empezar ve las gráficas básicas que se pueden producir.
Ingresa demo(graphics) en la línea de
| comandos, y después presiona Enter para comenzar y para cambi
ar de gráfica. SI PRESENTAS ALGÚN ERROR
| ingresa ok() en la línea de comandos.

> ok()

| ¡Todo ese trabajo está rindiendo frutos!

  |====
|   4%
| Como pudiste observar, R es muy bueno a la hora de graficar.

...

  |======
|   6%
| El sistema base de gráficos de R cuenta con tres tipos básico
s de funciones: funciones de alto nivel,
| funciones de bajo nivel y funciones interactivas.

...

  |=======
|   7%
| Las funciones de alto nivel generan gráficas preestablecidas.

...

  |=========
|   9%
| Las funciones de bajo nivel añaden información a un gráfico e
xistente.
```

...

  |==========
| 10%
| Y las funciones interactivas te permiten de forma interactiva añadir información o extraer información de
| gráficos. Este curso sólo cubrirá las funciones de alto nivel y bajo nivel.

...

  |============
| 12%
| Comienza explorando las funciones de alto nivel.

...

  |=============
| 13%
| Las funciones de alto nivel están diseñadas para generar un gráfico a partir de la información pasada como
| argumentos de la función.

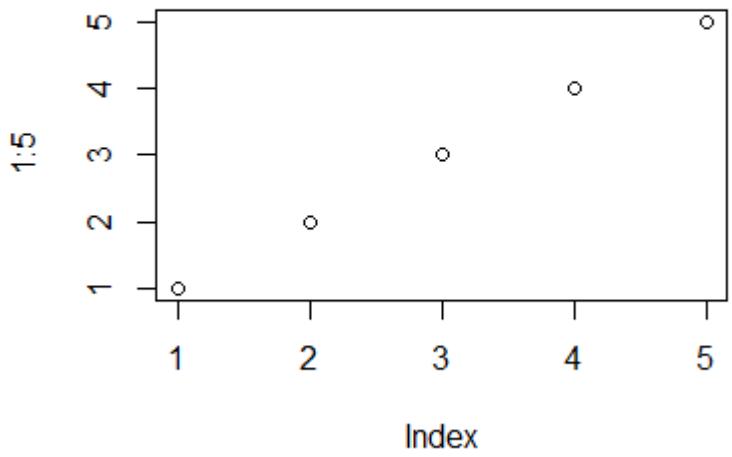
...

  |===============
| 15%
| La función plot() es una de las funciones de alto nivel más comúnmente usada.

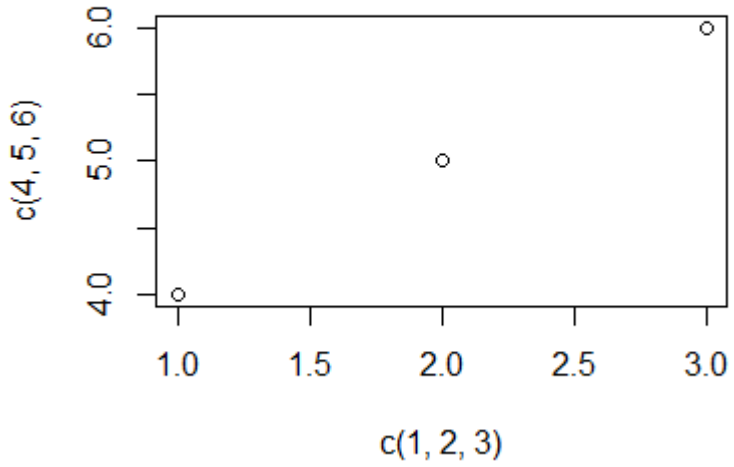
...

  |================
| 16%
| Comienza a jugar con ella. Ingresa plot(1:5) en la línea de comandos.

```
> plot(1:5)
```




```
| Perseverancia es la respuesta.

  |=================
|  18%
| Al ingresar plot(1:5) has graficado cada elemento del vector
1:5 (1, 2, 3, 4, 5) contra la posición en
| dicho vector de cada elemento; es decir, graficaste los punto
s (1, 1), (2, 2), (3, 3), (4, 4) y (5, 5).

...

  |==================
|  19%
| Es importante saber que plot() es una función genérica, por l
o que graficará dependiendo del objeto que le
| sea pasado como entrada.

...

  |====================
|  21%
| Ve qué pasa si ahora introduces plot(c(1, 2, 3), c(4, 5, 6))
en la línea de comandos.

> plot(c(1, 2, 3), c(4, 5, 6))
```




```
| ¡Eso es trabajo bien hecho!

  |======================
| 22%
| Como notarás, esta vez introdujiste dos vectores como entrada
, cada uno con tres elementos. Entonces la
| gráfica fue construida tomando un elemento del primer vector
(posición x) y un elemento del segundo vector
| (posición y) para construir cada punto, los cuales son: (1, 4
), (2, 5) y (3, 6).

...

  |========================
| 24%
| Conoce más del uso de plot(). Ingresa ?plot en la línea de co
mandos.

> ?plot

| ¡Excelente!

  |=========================
| 25%
| Como ya te habrás dado cuenta plot() recibe dos argumentos pr
incipales.

...

  |===========================
| 27%
| El primero, x, representa las coordenadas en el eje X de los
puntos en la gráfica, o alternativamente una
| única estructura para graficar, una función o cualquier objet
o de R que provea un método para graficar.
```

...

  |============================
| 28%
| El segundo, y, representa las coordenadas en el eje Y de los puntos de la gráfica. Pero este argumento es
| OPCIONAL, pues sólo es necesario si x no es una estructura apropiada.

...

  |=============================
| 30%
| Y esto explica por qué puedes graficar si le pasas uno o dos vectores como entrada. En el primer caso (1:5)
| le envías una estructura apropiada; en el segundo le envías las coordenadas de los puntos en 'x' y 'y' por
| medio de dos vectores (c(1, 2, 3) y c(4, 5, 6)).

...

  |==============================
| 31%
| Además, plot() recibe '...' como argumento, pues no en todos los casos recibirá los mismos argumentos
| adicionales; esto se debe a que plot() es una función genérica.

...

  |================================
| 33%
| Pero la mayoría de las veces acepta los siguientes argumentos: type, main, sub, xlab, ylab y asp.

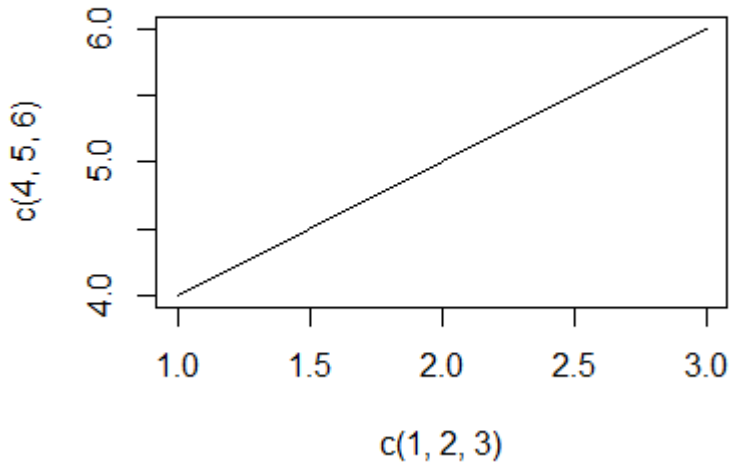
...

  |=================================
| 34%
| type sirve para especificar qué tipo de gráfica debe trazar. Los valores que puede tomar son: "p" para
| puntos, "l" para líneas, "b" para ambas (líneas y punto), "c" para la parte de líneas que se muestra usando
| "b", "o" para ambas (líneas y puntos) ‘sobrepuestas’, "h" para ‘histograma’ como líneas verticales (o ‘alta
| densidad’), "s" para escalonado, "S" para otro tipo de escalonamiento y "n" para no graficar.

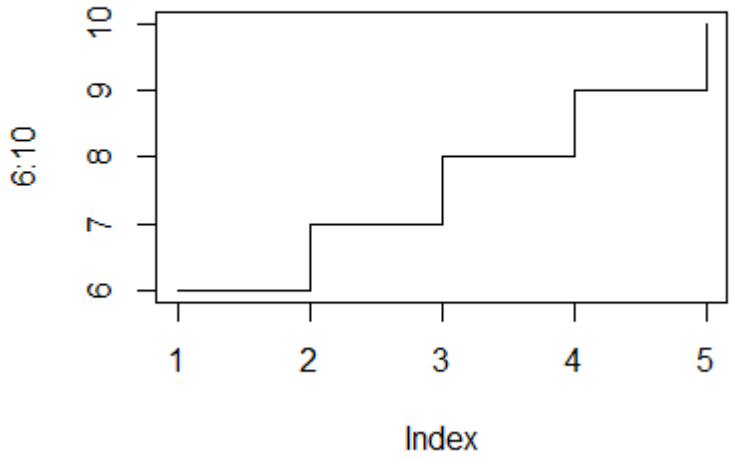
...

  |===================================
| 36%
| Como te habrás dado cuenta, cuando graficas plot(c(1, 2, 3), c(4, 5, 6)) el tipo de gráfica por defecto fue
| puntos. Ahora ingresa plot(c(1, 2, 3), c(4, 5, 6), type="l") en la línea de comandos para usar líneas.

```
>
> plot(c(1, 2, 3), c(4, 5, 6), type="l")
```



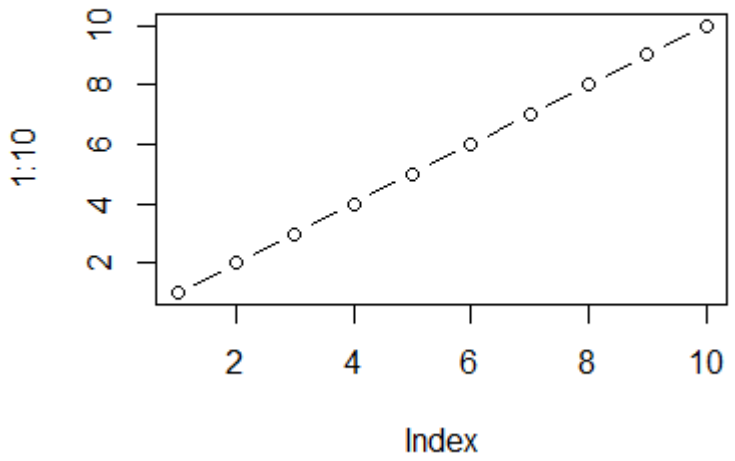

```
| ¡Eso es correcto!

  |=====================================
|  37%
| Ahora grafica el vector 6:10 de forma escalonada.

> plot(6:10, type="s")
```




```
| ¡Bien hecho!

  |=====================================
|  39%
```

| Para continuar grafica el vector 1:10 usando ambas (líneas y puntos).

```
> plot(1:10, type="b")
```

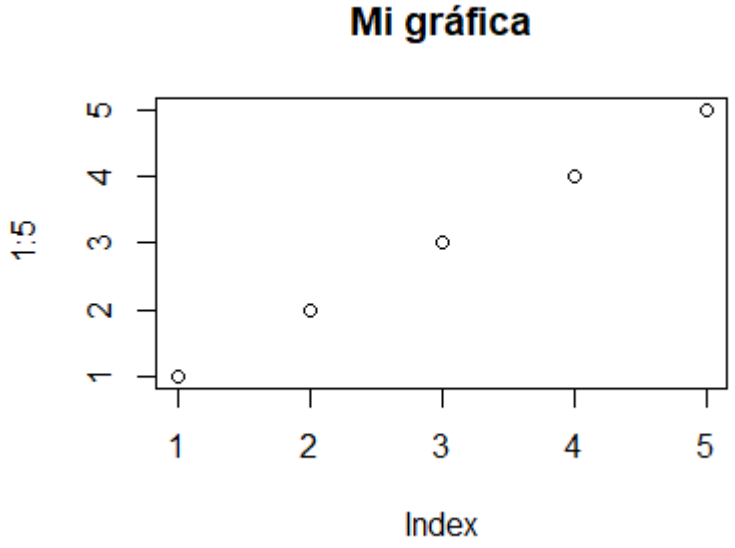


| ¡Eso es trabajo bien hecho!

```
  |=======================================
|  40%
```

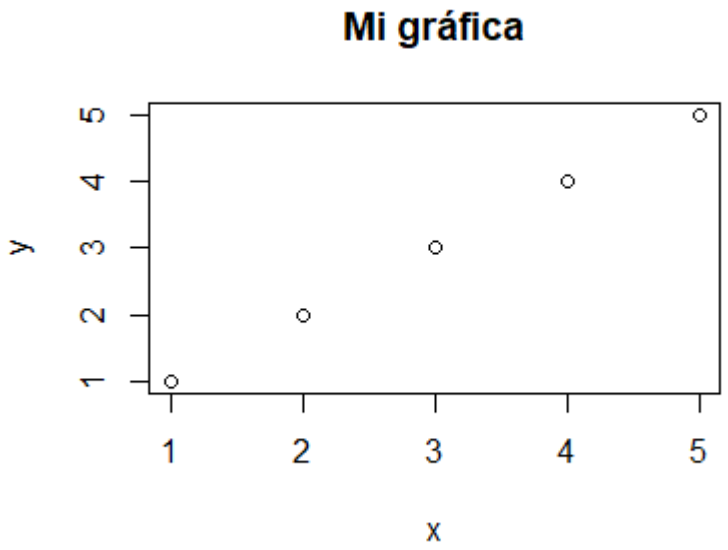
| El argumento main establece el título de la gráfica. ¡Pruébalo! Introduce plot(1:5, main="Mi gráfica") en
| la línea de comandos.

```
> plot(1:5, main="Mi gráfica")
```



| ¡Traes una muy buena racha!

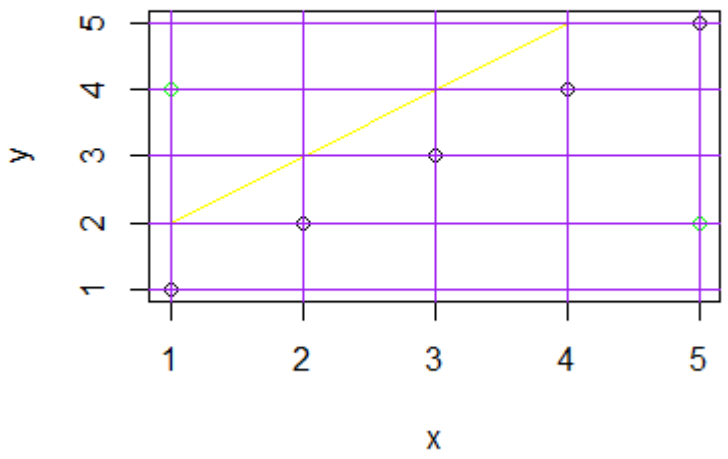
```
  |=========================================
|   42%
| Análogamente sub establece el subtítulo de la gráfica.

...

  |==========================================
|   43%
| El argumento xlab establece un título para el eje X de la grá
fica. Análogamente ylab para el eje Y.
| Establece "x" como título del eje X y "y" como título del eje
Y en la gráfica anterior.

> plot(1:5, main="Mi gráfica", xlab="x", ylab="y")
```




```
| ¡Toda esa práctica está rindiendo frutos!

  |==========================================
|   45%
| asp se refiere a la proporción x/y.

...

  |===========================================
|   46%
| Algunas veces las funciones estándar para graficar no produci
rán exactamente el tipo de gráfica que deseas.
| En estos casos las funciones para graficar de bajo nivel pued
en ser usadas para añadir información extra a
| la actual gráfica, como lo son puntos, líneas o texto.

...

  |============================================
|   48%
```

| Una de ellas es la función points(). Con la función points() tú puedes graficar puntos sobre una gráfica.
| Ingresa points(c(1, 5), c(4, 2), col="green") en la línea de comandos para graficar los puntos (1, 4) y (5,
| 2).

```
> points(c(1, 5), c(4, 2), col="green")
```



| ¡Sigue trabajando de esa manera y llegarás lejos!

```
  |=================================================
|  49%
```
| Como puedes notar, esto puede ser muy útil para añadir un con junto adicional de puntos a una gráfica
| existente. Usualmente con un color diferente o un símbolo dif erente.

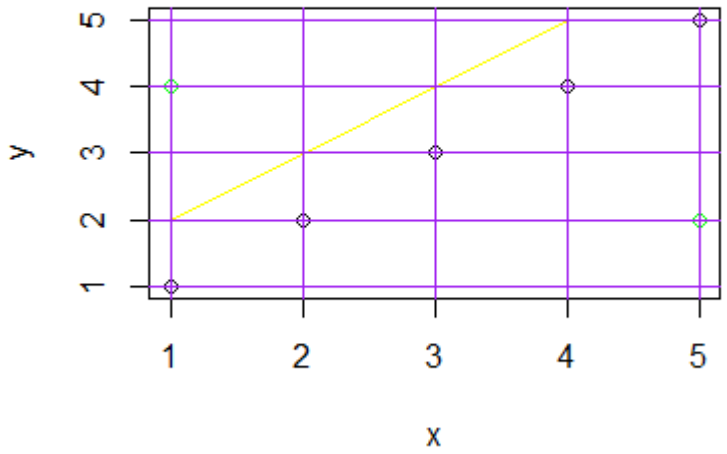
...

```
  |==================================================
|  51%
```
| La mayoría de los argumentos de la función plot() aplican par a función points(), incluyendo 'x' y
| opcionalmente 'y'. Pero los argumentos más útiles son: col pa ra especificar el color del borde para los
| puntos a graficar, bg para especificar el color de relleno de los puntos a graficar, pch para especificar
| el símbolo que se usará para graficar al punto.

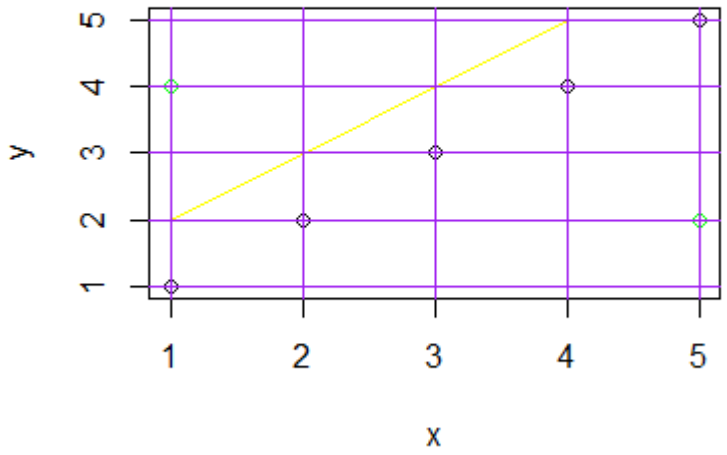
...

```
  |==================================================
|  52%
```
| Otra función muy útil es la función lines(). Ingresa lines(c(1, 4), c(2, 5), col="yellow") en la línea de
| comandos.

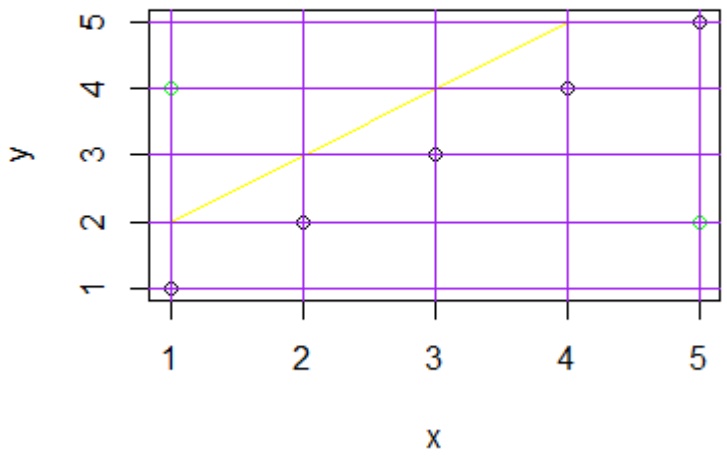
```
> lines(c(1, 4), c(2, 5), col="yellow")
```




```
| ¡Buen trabajo!

  |=======================================================
|  54%
| La función lines() grafica un conjunto de segmentos de línea
sobre una gráfica existente. Al igual que la
| función points(), muchos argumentos de plot() aplican para li
nes(). Los valores de 'x' y 'y' especifican
| las intersecciones entre los segmentos de línea.

...

  |=======================================================
|  55%
| Para trazar una sola línea a través del área de la gráfica, p
uedes utilizar la función abline(). Por lo
| general, se llama abline() para dibujar una sola línea. Por e
jemplo, ingresa abline(h=3,col="red",lty=2) en
| la línea de comandos.

> abline(h=3,col="red",lty=2)
```

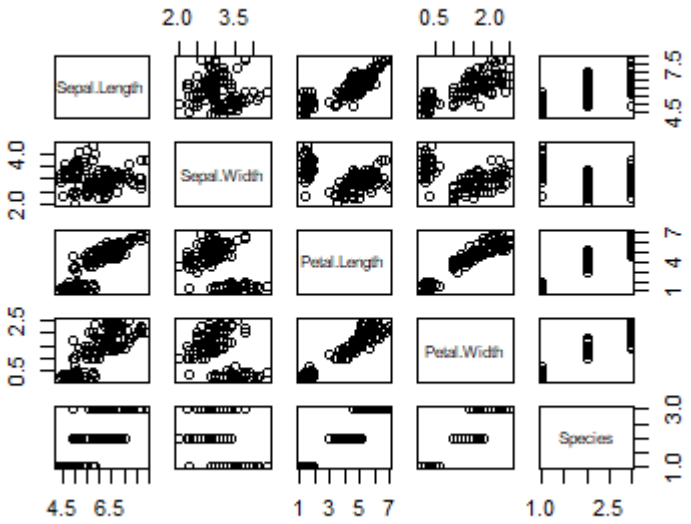



```
| Perseverancia es la respuesta.

  |=======================================================
|   57%
| Como notarás, graficaste una línea horizontal en y=3. Para gr
aficar una línea vertical en x=3, basta con
| ingresar abline(v=3,col="red",lty=2) en la línea de comandos.

...

  |=======================================================
|   58%
| También puedes especificar múltiples argumentos y abline() gr
aficará las líneas especificadas. Por ejemplo,
| ingresa abline(h=1:5,v=1:5, col="purple") en la línea de coma
ndos para graficar una cuadrícula de líneas
| entre 1 y 10.

> abline(h=1:5,v=1:5, col="purple")
```




```
| ¡Acertaste!

  |=======================================================================
|   60%
| Aunque si deseas graficar una cuadrícula en tu gráfica, es me
jor que uses la función grid().

...

  |=======================================================================
=                                                       |   61%
| Hasta ahora sólo has trabajado con datos ficticios. Para hace
r esto más interesante trabajarás con datos
| reales a partir de este momento.

...

  |=======================================================================
===                                                     |   63%
| Usarás el famoso conjunto de datos iris, el cual contiene las
medidas en centímetros de longitud y ancho de
| ambos sépalo y pétalo de tres especies de iris (setosa, versi
color y virginica) con 50 ejemplares cada una.

...

  |=======================================================================
====                                                    |   64%
| Para cargar el conjunto de datos iris, ingresa data("iris") e
n la línea de comandos.

> data("iris")

| ¡Tu dedicación es inspiradora!
```

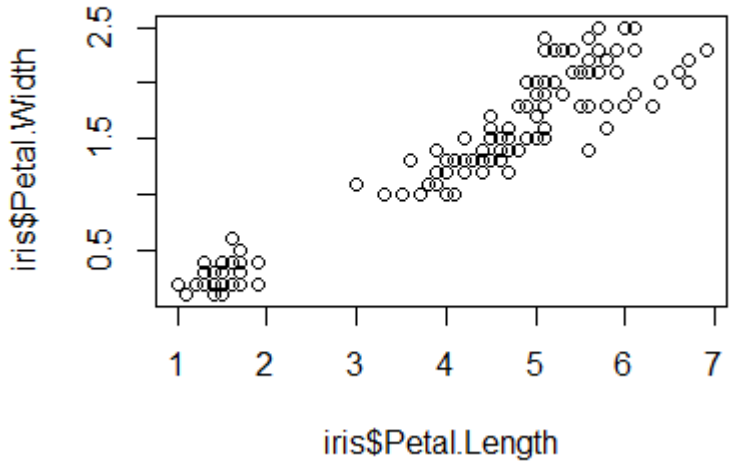
```
  |=======================================================================
======                                                  |  66%
| Como notarás, el objeto iris fue cargado. Averigua qué tipo d
e objeto es iris.

> class(iris)
[1] "data.frame"

| ¡Todo ese trabajo está rindiendo frutos!

  |=======================================================================
=======                                                 |  67%
| Como verás, iris es un data frame. Ingresa head(iris) en la l
ínea de comandos para ver las primeras seis
| líneas de contenido de iris.

> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

| ¡Acertaste!

  |=======================================================================
=========                                               |  69%
| Sepal.Length y Sepal.Width representan la longitud y ancho de
l sépalo respectivamente. Petal.Length y
| Petal.Width representan la longitud y ancho del pétalo. Speci
es representa a la especie (setosa, versicolor
| y virginica).

...

  |=======================================================================
==========                                              |  70%
| Como ya sabes, plot() es una función genérica, por lo que tam
bién puedes pedirle que te grafique un data
| frame completo. Por ejemplo, iris; ingresa plot(iris) en la l
ínea de comandos.

> plot(iris)
```



| ¡Sigue trabajando de esa manera y llegarás lejos!

  |===
============ | 72%
| O bien, puedes tomar columnas y graficarla. Ingresa plot(iris
$Petal.Length, iris$Petal.Width) en la línea
| de comandos.

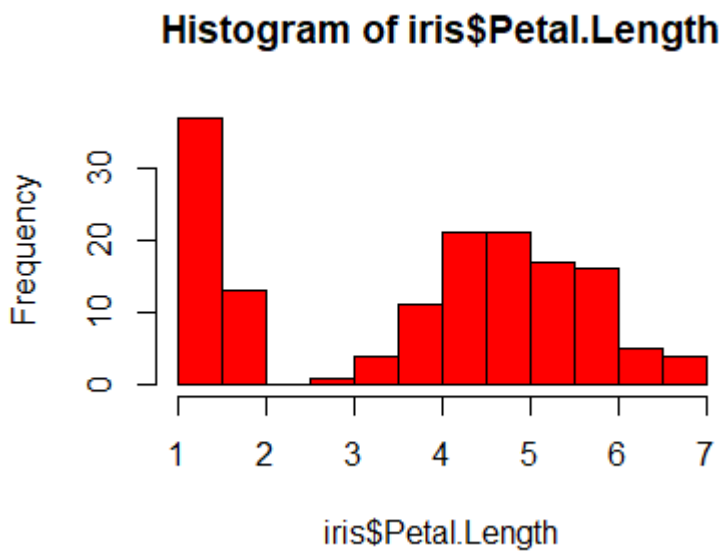
```
> plot(iris$Petal.Length, iris$Petal.Width)
```



| ¡Sigue trabajando de esa manera y llegarás lejos!

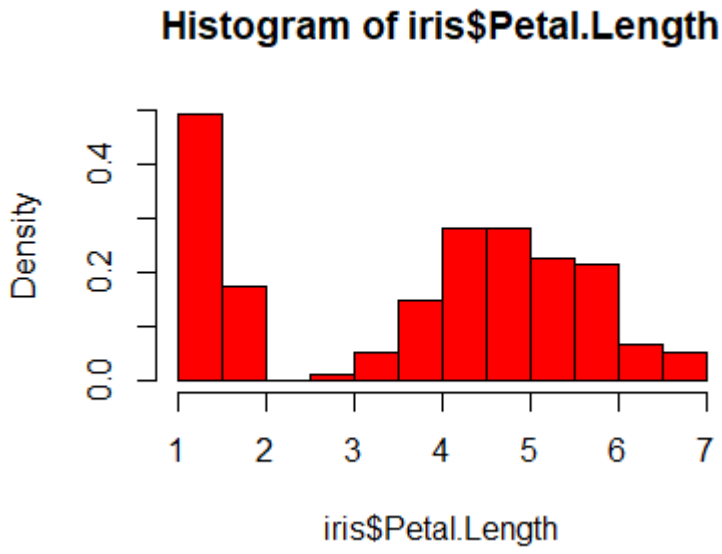
  |===
============= | 73%
| Pero no es necesario graficar pares de puntos.

```
...

  |======================================================================
===============                                      |  75%
| Un histograma es una representación visual de la distribución
de un conjunto de datos.

...

  |======================================================================
===============                                      |  76%
| Ahora usa la función hist() para graficar un histograma de la
s longitudes de los pétalos. Ingresa
| hist(iris$Petal.Length, col="red") en la línea de comandos.

>
> hist(iris$Petal.Length, col="red")
```

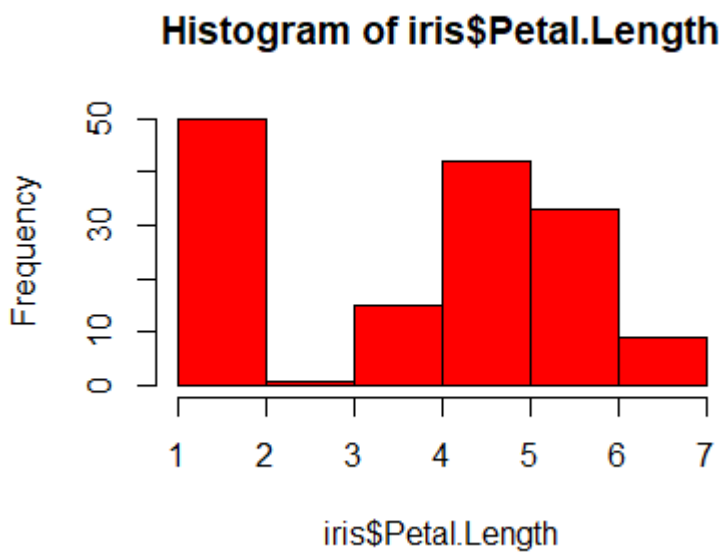



```
| ¡Traes una muy buena racha!

  |======================================================================
==================                                   |  78%
| La función hist() es una función de alto nivel y recibe un ve
ctor de valores numéricos y grafica un
| histograma. Un histograma consiste de un eje X y un eje Y, y
varias barras de diferentes tamaños. La altura
| del eje Y te muestra la frecuencia con la que aparecen los va
lores del eje X en el conjunto de datos.

...

  |======================================================================
==================                                   |  79%
| La forma de un histograma es una de sus características más i
mportantes, pues te permite ver relativamente
```

```
| donde se encuentra sitiada la mayor y menor cantidad de infor
mación. Esto te permite encontrar valores
| atípicos.

...

  |=======================================================================
=====================                                  |  81%
| En caso de que no quieras que te grafique frecuencias puedes
usar el parámetro freq = FALSE para que
| grafique probabilidades. Ingresa hist(iris$Petal.Length, col=
"red", freq=FALSE) en la línea de comandos.

> hist(iris$Petal.Length, col="red", freq=FALSE)
```




```
| ¡Bien hecho!

  |=======================================================================
=====================                                  |  82%
| También es posible cambiar el número de celdas del histograma
. Para eso usa el argumento breaks.
| Dependiendo del número de celdas especificadas el gráfico pue
de ser de una forma u otra. Ingresa
| hist(iris$Petal.Length, col="red", breaks=5) en la línea de c
omandos.

> hist(iris$Petal.Length, col="red", breaks = 5)
```



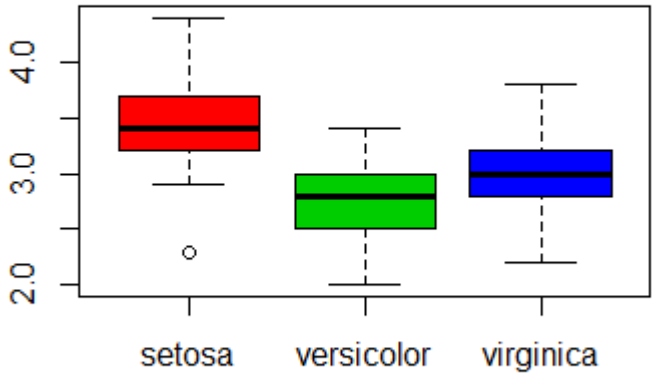

```
| ¡Eso es correcto!

  |======================================================================
=========================                    |   84%
| Además de histogramas existen las gráficas de caja.

...

  |======================================================================
=========================                    |   85%
| La función boxplots(), también de alto nivel, genera gráficas
de caja. Una gráfica de caja es una forma
| compacta para mostrar la distribución de una variable. La caj
a muestra el rango intercuartil.

...

  |======================================================================
============================                 |   87%
| Ahora ingresa ?boxplot en la línea de comandos para conocer e
l uso de la función boxplot().

> ?boxplot

| ¡Lo estás haciendo muy bien!

  |======================================================================
============================                 |   88%
| Como ya habrás notado, boxplot() además recibe un argumento '
formula', el cual generalmente es una
| expresión con una tilde (~), la cual indica la relación entre
las variables de entrada. Eso te permite dar
| como fórmula algo como Sepal.Width ~ Species para graficar la
relación entre el ancho del sépalo y la
```

| especie.

...

  |===
============================= | 90%
| Ingresa boxplot(Sepal.Width ~ Species, data=iris, col=2:4) en
la línea de comandos.

```
> boxplot(Sepal.Width ~ Species, data=iris, col=2:4)
```



| ¡Tu dedicación es inspiradora!

  |===
============================= | 91%
| boxplot() te generó por cada especie (setosa, versicolor y vi
rginica) los valores de dispersión de los
| anchos del sépalo. La gráfica te muestra que el ancho del sép
alo de la especie setosa es mucho mayor que el
| de las demás especies.

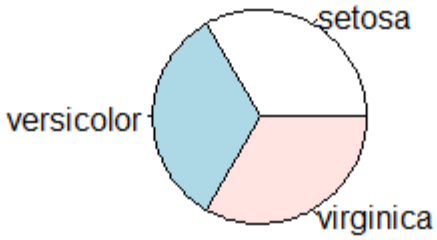
...

  |===
============================= | 93%
| boxplot() puede ser usado para crear gráficas de caja para va
riables individuales o para variables por
| grupo.

...

  |===
============================= | 94%
| Al igual que con hist() puedes usar los mismos argumentos que
usaste en plot() para añadir títulos (título,

```
| subtítulo, eje x, eje y).

...

  |=======================================================================
=================================    |  96%
| Además, existen las gráficas de pastel.

...

  |=======================================================================
=================================    |  97%
| Las graficas de pastel no son recomendadas ya que sus caracte
rísticas son algo limitadas. Los autores
| recomiendan usar gráficas de barras o de puntos en vez de grá
ficas de pastel, debido a que las personas son
| capaces de juzgar longitudes con mayor precisión que volúmene
s.

...

  |=======================================================================
=================================== |  99%
| Las gráficas de pastel son creadas con la función pie(x, labe
ls=), donde x es un vector numérico positivo
| indicando el área de las rebanadas y labels es un vector que
indica el nombre de cada rebanada. Ingresa
| pie(c(50, 50, 50), labels=levels(iris$Species)) en la línea d
e comandos.

> pie(c(50, 50, 50), label=levels(iris$Species))
```




```
| ¡Bien hecho!

  |=======================================================================
=====================================| 100%
```

Lección de swirl: 8. Parámetros en el Sistema de Gráficos

```
Selection: 8
  |
|   0%

| En esta lección verás los parámetros del sistema de gráficos
a más detalle.

...

  |===
|   3%
| Cuando creas gráficas, las opciones por defecto de R no siemp
re producirán exactamente lo que deseas. Sin
| embargo, puedes modificar casi cualquier aspecto de la gráfic
a usando los parámetros del sistema de
| gráficos.

...

  |======
|   6%
| Los parámetros del sistema de gráficos pueden establecerse de
dos maneras: de forma permanente (que afecta
| a todas las funciones de gráficos) o temporal (que afecta sól
o a la función llamada como lo viste en la
| lección Graficación).

...

  |=========
|   9%
| La función par() es usada para acceder y modificar la lista d
e parámetros de forma permanente.

...

  |============
|  12%
| Para ver más detalles introduce ?par en la línea de comandos.

> ?par

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |================
|  16%
| Las nuevas configuraciones establecidas en la función par() s
erán los valores por defecto para cualquier
| gráfica nueva hasta que la sesión sea finalizada.

...
```

```
  |==================
|  19%
| La función par() puede ser muy útil si deseas establecer los
parámetros una vez y luego graficar múltiples
| veces con ellos.

...

  |=====================
|  22%
| Puedes checar o establecer los valores de los parámetros de s
istema de gráficos con la función par().

...

  |========================
|  25%
| Para obtener una lista mostrando todos los parámetros del sis
tema de gráficos, simplemente llama a la
| función par() sin argumentos. ¡Inténtalo!

> par()
$`xlog`
[1] FALSE

$ylog
[1] FALSE

$adj
[1] 0.5

$ann
[1] TRUE

$ask
[1] FALSE

$bg
[1] "white"

$bty
[1] "o"

$cex
[1] 1

$cex.axis
[1] 1

$cex.lab
[1] 1

$cex.main
[1] 1.2

$cex.sub
[1] 1
```

```
$cin
[1] 0.15 0.20

$col
[1] "black"

$col.axis
[1] "black"

$col.lab
[1] "black"

$col.main
[1] "black"

$col.sub
[1] "black"

$cra
[1] 14.4 19.2

$crt
[1] 0

$csi
[1] 0.2

$cxy
[1] 0.2154613 0.2872818

$din
[1] 4.208333 3.343750

$err
[1] 0

$family
[1] ""

$fg
[1] "black"

$fig
[1] 0 1 0 1

$fin
[1] 4.208333 3.343750

$font
[1] 1

$font.axis
[1] 1

$font.lab
[1] 1
```

```
$font.main
[1] 2

$font.sub
[1] 1

$lab
[1] 5 5 7

$las
[1] 0

$lend
[1] "round"

$lheight
[1] 1

$ljoin
[1] "round"

$lmitre
[1] 10

$lty
[1] "solid"

$lwd
[1] 1

$mai
[1] 1.02 0.82 0.82 0.42

$mar
[1] 5.1 4.1 4.1 2.1

$mex
[1] 1

$mfcol
[1] 1 1

$mfg
[1] 1 1 1 1

$mfrow
[1] 1 1

$mgp
[1] 3 1 0

$mkh
[1] 0.001

$new
[1] FALSE
```

```
$oma
[1] 0 0 0 0

$omd
[1] 0 1 0 1

$omi
[1] 0 0 0 0

$page
[1] TRUE

$pch
[1] 1

$pin
[1] 2.968333 1.503750

$plt
[1] 0.1948515 0.9001980 0.3050467 0.7547664

$ps
[1] 12

$pty
[1] "m"

$smo
[1] 1

$srt
[1] 0

$tck
[1] NA

$tcl
[1] -0.5

$usr
[1] -2.13187  2.13187 -1.08000  1.08000

$xaxp
[1] -2  2  4

$xaxs
[1] "r"

$xaxt
[1] "s"

$xpd
[1] FALSE

$yaxp
[1] -1  1  4
```

```
$yaxs
[1] "r"

$yaxt
[1] "s"

$ylbias
[1] 0.2


| ¡Lo has logrado! ¡Buen trabajo!

  |============================
|  28%
| Puedes guardar todos los valores de par() en un objeto. Ingre
sa par_orig <- par() en la línea de comandos
| para guardar los valores en la variable par_orig.

> par_orig <- par()

| ¡Traes una muy buena racha!

  |==============================
|  31%
| Esto puede ser útil, ya que puedes volver a utilizar estos va
lores después de haberlos modificado.

...

  |================================
|  34%
| Si deseas checar el valor de un parámetro con la función par(
), usa el nombre del parámetro como argumento
| (en cadena). Por ejemplo, ingresa par("col") en la línea de c
omandos para conocer el valor por defecto del
| parámetro col (color).

> par("col")
[1] "black"

| ¡Lo has logrado! ¡Buen trabajo!

  |====================================
|  38%
| Para establecer el valor de un parámetro, basta con usar el n
ombre del parámetro como argumento de la
| función par(). Por ejemplo, puedes usar la función par() para
cambiar el valor del parámetro col. Ingresa
| par(col="blue") en la línea de comandos.

> par(col="blue")

| ¡Todo ese trabajo está rindiendo frutos!

  |======================================
|  41%
```

| Para probar que los cambios se han efectuado, en esta lección volverás a trabajar con el conjunto de datos
| iris. Carga el conjunto de datos iris.

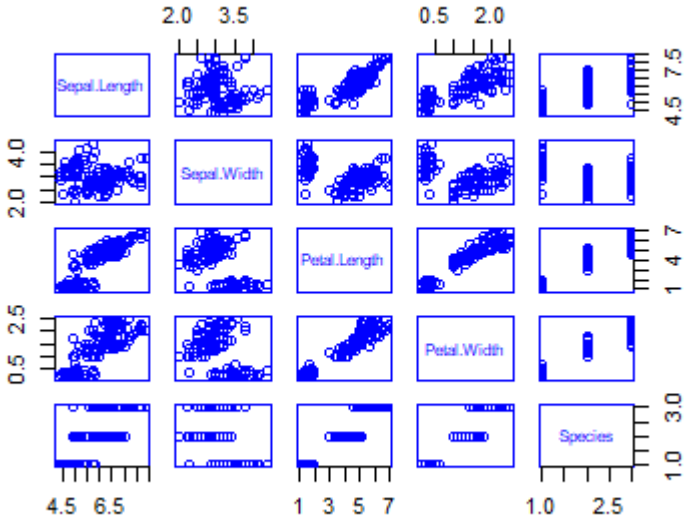
```
> data("iris")
```

| ¡Sigue trabajando de esa manera y llegarás lejos!

```
  |=========================================
|   44%
```

| Ahora prueba que el color por defecto ha sido cambiado a azul. Ingresa plot(iris) en la línea de comandos
| para verificar esto.

```
> plot(iris)
```



| ¡Eso es trabajo bien hecho!

```
  |=========================================
|   47%
```

| Y si revisas nuevamente el valor de col usando par() encontrarás que ha sido establecido a azul ("blue").
| Revisa el valor de col.

```
> par("col")
[1] "blue"
```

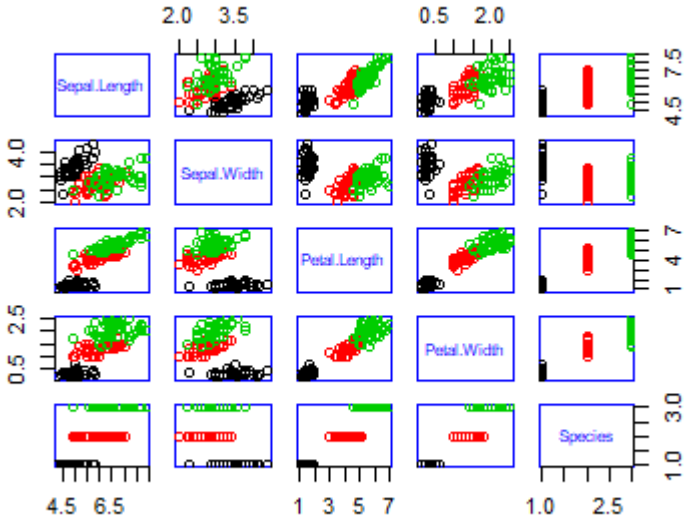
| ¡Buen trabajo!

```
  |=========================================
|   50%
```

| Cuando graficas el data frame iris usando plot(iris), los colores mostrados no dicen mucho acerca de las
| diferentes especies. Así que le puedes decir a R que grafique usando un color diferente por cada una de la

| especies usando col=iris$Species. Ingresa plot(iris, col=iris
$Species) en la línea de comandos.

```
> plot(iris, col=iris$Species)
```



| ¡Eso es trabajo bien hecho!

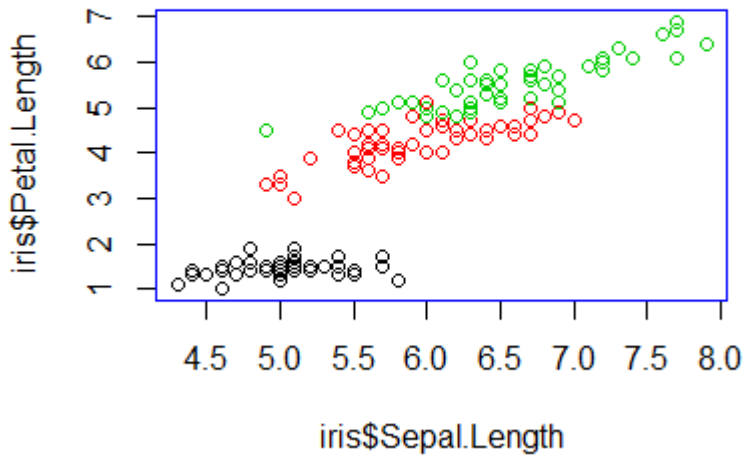
```
  |======================================================
|  53%
```
| Para continuar es importante recordar los nombres de las colu
mnas de iris. ¡Inténtalo!

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
"Species"
```

| ¡Eres el mejor!

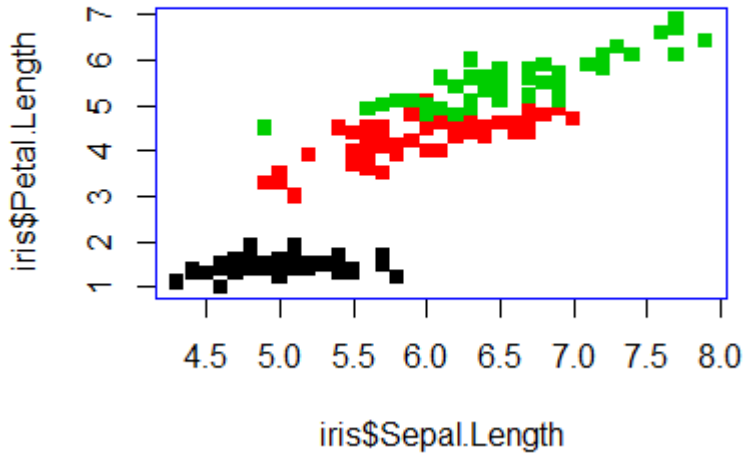
```
  |======================================================
|  56%
```
| Comienza viendo las columnas Sepal.Length y Petal.Length. Otr
a vez especifica un color por especie. Ingresa
| plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
) en la línea de comandos.

```
> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
)
```

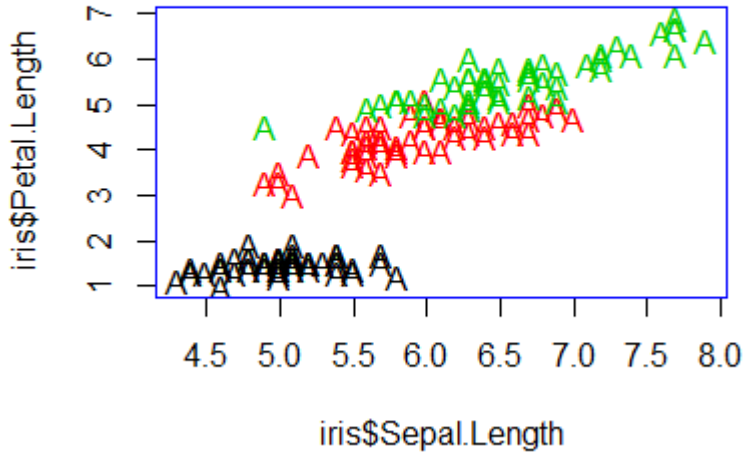



```
| ¡Lo has logrado! ¡Buen trabajo!

  |=======================================================
|   59%
| Los puntos usados para graficar son difíciles de ver. Escoge
otros diferentes.

...

  |=======================================================
==                                                      |   62%
| Un parámetro importante es el símbolo que se usa para grafica
r puntos; éstos se cambian usando el parámetro
| pch. Este parámetro puede recibir valores de dos maneras.

...

  |=======================================================
======                                                  |   66%
| Una manera es usando un código numérico, donde cada código nu
mérico corresponde a un símbolo. Pruébalo
| usando pch=15. Ingresa par(pch=15) en la línea de comandos.

> par(pch=15)

| ¡Eres el mejor!

  |=======================================================
=========                                               |   69%
| El código numérico 15 representa cuadrados. Verifica el nuevo
valor que estableciste. Ingresa
| plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
) en la línea de comandos.
```

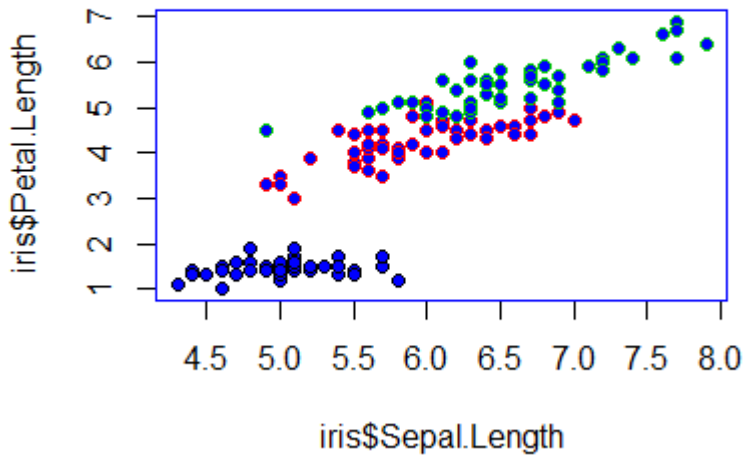
```
> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
)
```



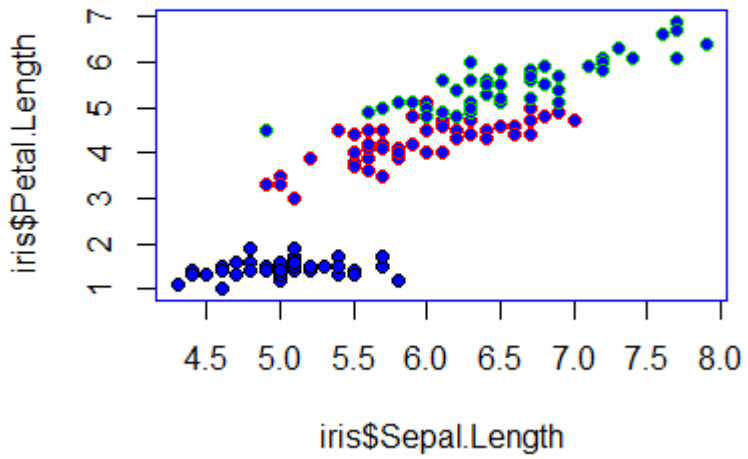

```
| ¡Eres bastante bueno!

  |======================================================================
============                                          |  72%
| La otra manera es utilizando una cadena. Ingresa plot(iris$Se
pal.Length, iris$Petal.Length, col =
| iris$Species, pch="A") en la línea de comandos.

> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
, pch="A")
```



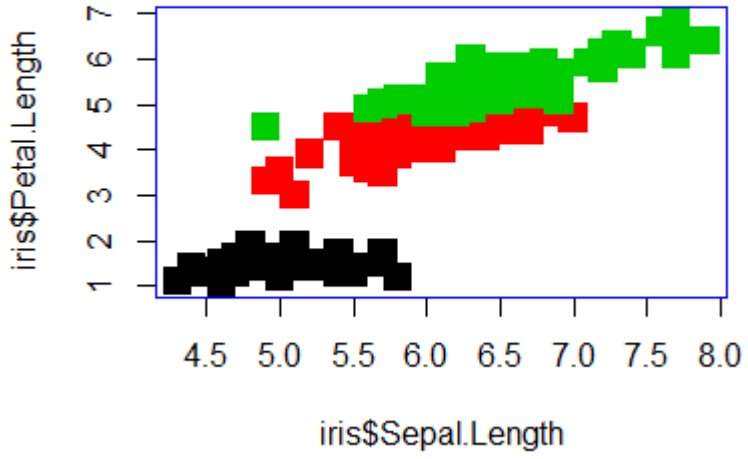
```
| ¡Lo estás haciendo muy bien!

  |=======================================================================
===============                                      |  75%
| Como te podrás imaginar, cambiaste el valor del parámetro pch
temporalmente, al haberlo modificado en la
| función de plot() y no por medio de la función par(). Si nuev
amente graficas sin especificar pch, verás que
| el símbolo graficado nuevamente es un cuadrado. ¡Verifica est
o! Ingresa plot(iris$Sepal.Length,
| iris$Petal.Length, col = iris$Species) en la línea de comando
s.

> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
)
```



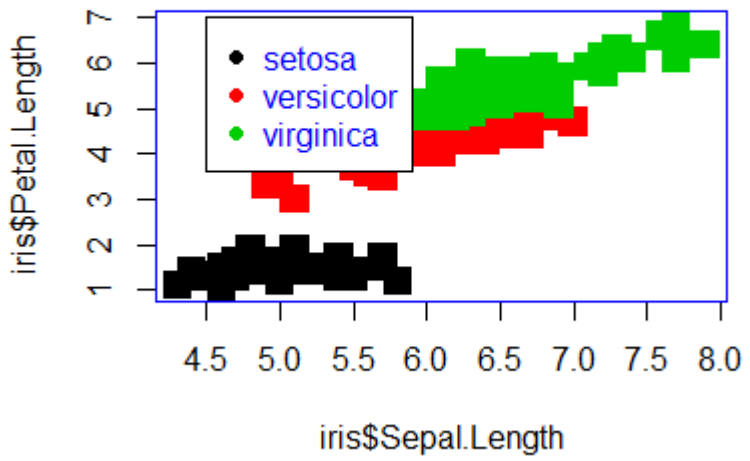

```
| ¡Mantén este buen nivel!

  |=======================================================================
==================                                   |  78%
| Los códigos numéricos que puedes usar son los números del 0 a
l 25. Para conocer los símbolos disponibles
| ingresa plot(1:26, pch=0:25) en la línea de comandos.

> plot(1:26, pch=0:25)
```




```
| ¡Excelente!

  |======================================================================
=====================                                 |  81%
| En particular a los símbolos del 21 al 25 les puedes cambiar
el color de la orilla y el de relleno. Esto se
| hace usando los parámetros col y bg. Ingresa plot(iris$Sepal.
Length, iris$Petal.Length, col = iris$Species,
| pch = 21, bg = "blue") en la línea de comandos.

> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
, pch = 21, bg = "blue")
```



```
| ¡Lo estás haciendo muy bien!

  |=======================================================================
========================                 |  84%
| También puedes cambiar el tamaño de los símbolos usando el ar
gumento cex. Ingresa plot(iris$Sepal.Length,
| iris$Petal.Length, col = iris$Species, cex = 2) en la línea d
e comandos.

> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species
, cex = 2)
```




```
| ¡Excelente trabajo!

  |=======================================================================
============================             |  88%
| Es difícil distinguir qué color pertenece a qué especie. Lo m
ejor sería hacer una leyenda. La función
| legend() puede ser usada para añadir leyendas a las gráficas.
Ingresa legend(x = 4.5, y = 7, legend =
| levels(iris$Species), col = c(1:3), pch = 16) para añadir una
leyenda.

> legend(x = 4.5, y = 7, legend = levels(iris$Species), col = c
(1:3), pch = 16)

| ¡Todo ese trabajo está rindiendo frutos!

  |=======================================================================
============================             |  91%
| Los parámetros 'x' y 'y' representan las coordenadas que será
n usadas para la leyenda; legend representa a
| las cadenas que aparecerán en la leyenda; col indica el color
de las líneas o símbolos que aparecerán en la
```

```
| leyenda; y pch indica los caracteres o símbolos que aparecerá
n en la leyenda.

...

  |=======================================================
==================================      |  94%
| Recuerda que todos los parámetros a los que modificaste su va
lor también los puedes cambiar directamente
| desde las funciones para graficar de alto nivel.

...

  |=======================================================
======================================   |  97%
| Por último, si deseas regresar a los valores de los parámetro
s por omisión que tenías en el sistema base,
| basta con llamar a la función par() pasándole como argumento
el objeto donde guardaste los valores
| originales. Regresa a los valores originales usando la funció
n par().

> par(par_orig)
Warning messages:
1: In par(par_orig) :
  el parámetro del gráfico "cin" no puede ser especificado
2: In par(par_orig) :
  el parámetro del gráfico "cra" no puede ser especificado
3: In par(par_orig) :
  el parámetro del gráfico "csi" no puede ser especificado
4: In par(par_orig) :
  el parámetro del gráfico "cxy" no puede ser especificado
5: In par(par_orig) :
  el parámetro del gráfico "din" no puede ser especificado
6: In par(par_orig) :
  el parámetro del gráfico "page" no puede ser especificado

| ¡Bien hecho!

  |=======================================================
=======================================| 100%
```

Lección de swirl: 9. Colores en el Sistema de Gráficos

```
Selection: 9
  |
|   0%

| En esta lección verás cómo se manejan los colores en el sistema
de gráficos de R.

...

  |===
|   3%
| Anteriormente viste que puedes usar el parámetro col para especi
ficar colores.

...

  |=====
|   5%
| Hasta ahora, la mayoría de las veces has usado cadenas refiriénd
ose a un simple conjunto de colores.

...

  |========
|   8%
| En R puedes especificar colores de tres maneras: como una cadena
, usando los componentes RGB (rojo, verde,
| azul) o haciendo referencia a un índice de la paleta de colores
mediante un entero.

...

  |===========
|  11%
| Para obtener la lista de los nombres de colores válidos, usa la
función colors(). Ingresa colors() en la
| línea de comandos.

> colors()
  [1] "white"                "aliceblue"            "antiquewhite"
"antiquewhite1"
  [5] "antiquewhite2"        "antiquewhite3"        "antiquewhite4
"         "aquamarine"
  [9] "aquamarine1"          "aquamarine2"          "aquamarine3"
"aquamarine4"
 [13] "azure"                "azure1"               "azure2"
"azure3"
 [17] "azure4"              "beige"                "bisque"
"bisque1"
 [21] "bisque2"             "bisque3"              "bisque4"
"black"
 [25] "blanchedalmond"      "blue"                 "blue1"
"blue2"
```

```
 [29] "blue3"                "blue4"                "blueviolet"
"brown"
 [33] "brown1"               "brown2"               "brown3"
"brown4"
 [37] "burlywood"            "burlywood1"           "burlywood2"
"burlywood3"
 [41] "burlywood4"           "cadetblue"            "cadetblue1"
"cadetblue2"
 [45] "cadetblue3"           "cadetblue4"           "chartreuse"
"chartreuse1"
 [49] "chartreuse2"          "chartreuse3"          "chartreuse4"
"chocolate"
 [53] "chocolate1"           "chocolate2"           "chocolate3"
"chocolate4"
 [57] "coral"                "coral1"               "coral2"
"coral3"
 [61] "coral4"               "cornflowerblue"       "cornsilk"
"cornsilk1"
 [65] "cornsilk2"            "cornsilk3"            "cornsilk4"
"cyan"
 [69] "cyan1"                "cyan2"                "cyan3"
"cyan4"
 [73] "darkblue"             "darkcyan"             "darkgoldenrod
"         "darkgoldenrod1"
 [77] "darkgoldenrod2"       "darkgoldenrod3"       "darkgoldenrod
4"        "darkgray"
 [81] "darkgreen"            "darkgrey"             "darkkhaki"
"darkmagenta"
 [85] "darkolivegreen"       "darkolivegreen1"      "darkolivegree
n2"       "darkolivegreen3"
 [89] "darkolivegreen4"      "darkorange"           "darkorange1"
"darkorange2"
 [93] "darkorange3"          "darkorange4"          "darkorchid"
"darkorchid1"
 [97] "darkorchid2"          "darkorchid3"          "darkorchid4"
"darkred"
[101] "darksalmon"           "darkseagreen"         "darkseagreen1
"         "darkseagreen2"
[105] "darkseagreen3"        "darkseagreen4"        "darkslateblue
"         "darkslategray"
[109] "darkslategray1"       "darkslategray2"       "darkslategray
3"        "darkslategray4"
[113] "darkslategrey"        "darkturquoise"        "darkviolet"
"deeppink"
[117] "deeppink1"            "deeppink2"            "deeppink3"
"deeppink4"
[121] "deepskyblue"          "deepskyblue1"         "deepskyblue2"
"deepskyblue3"
[125] "deepskyblue4"         "dimgray"              "dimgrey"
"dodgerblue"
[129] "dodgerblue1"          "dodgerblue2"          "dodgerblue3"
"dodgerblue4"
[133] "firebrick"            "firebrick1"           "firebrick2"
"firebrick3"
[137] "firebrick4"           "floralwhite"          "forestgreen"
"gainsboro"
```

```
[141] "ghostwhite"          "gold"                "gold1"
"gold2"
[145] "gold3"               "gold4"               "goldenrod"
"goldenrod1"
[149] "goldenrod2"          "goldenrod3"          "goldenrod4"
"gray"
[153] "gray0"               "gray1"               "gray2"
"gray3"
[157] "gray4"               "gray5"               "gray6"
"gray7"
[161] "gray8"               "gray9"               "gray10"
"gray11"
[165] "gray12"              "gray13"              "gray14"
"gray15"
[169] "gray16"              "gray17"              "gray18"
"gray19"
[173] "gray20"              "gray21"              "gray22"
"gray23"
[177] "gray24"              "gray25"              "gray26"
"gray27"
[181] "gray28"              "gray29"              "gray30"
"gray31"
[185] "gray32"              "gray33"              "gray34"
"gray35"
[189] "gray36"              "gray37"              "gray38"
"gray39"
[193] "gray40"              "gray41"              "gray42"
"gray43"
[197] "gray44"              "gray45"              "gray46"
"gray47"
[201] "gray48"              "gray49"              "gray50"
"gray51"
[205] "gray52"              "gray53"              "gray54"
"gray55"
[209] "gray56"              "gray57"              "gray58"
"gray59"
[213] "gray60"              "gray61"              "gray62"
"gray63"
[217] "gray64"              "gray65"              "gray66"
"gray67"
[221] "gray68"              "gray69"              "gray70"
"gray71"
[225] "gray72"              "gray73"              "gray74"
"gray75"
[229] "gray76"              "gray77"              "gray78"
"gray79"
[233] "gray80"              "gray81"              "gray82"
"gray83"
[237] "gray84"              "gray85"              "gray86"
"gray87"
[241] "gray88"              "gray89"              "gray90"
"gray91"
[245] "gray92"              "gray93"              "gray94"
"gray95"
[249] "gray96"              "gray97"              "gray98"
"gray99"
```

```
[253] "gray100"            "green"              "green1"
"green2"
[257] "green3"             "green4"             "greenyellow"
"grey"
[261] "grey0"              "grey1"              "grey2"
"grey3"
[265] "grey4"              "grey5"              "grey6"
"grey7"
[269] "grey8"              "grey9"              "grey10"
"grey11"
[273] "grey12"             "grey13"             "grey14"
"grey15"
[277] "grey16"             "grey17"             "grey18"
"grey19"
[281] "grey20"             "grey21"             "grey22"
"grey23"
[285] "grey24"             "grey25"             "grey26"
"grey27"
[289] "grey28"             "grey29"             "grey30"
"grey31"
[293] "grey32"             "grey33"             "grey34"
"grey35"
[297] "grey36"             "grey37"             "grey38"
"grey39"
[301] "grey40"             "grey41"             "grey42"
"grey43"
[305] "grey44"             "grey45"             "grey46"
"grey47"
[309] "grey48"             "grey49"             "grey50"
"grey51"
[313] "grey52"             "grey53"             "grey54"
"grey55"
[317] "grey56"             "grey57"             "grey58"
"grey59"
[321] "grey60"             "grey61"             "grey62"
"grey63"
[325] "grey64"             "grey65"             "grey66"
"grey67"
[329] "grey68"             "grey69"             "grey70"
"grey71"
[333] "grey72"             "grey73"             "grey74"
"grey75"
[337] "grey76"             "grey77"             "grey78"
"grey79"
[341] "grey80"             "grey81"             "grey82"
"grey83"
[345] "grey84"             "grey85"             "grey86"
"grey87"
[349] "grey88"             "grey89"             "grey90"
"grey91"
[353] "grey92"             "grey93"             "grey94"
"grey95"
[357] "grey96"             "grey97"             "grey98"
"grey99"
[361] "grey100"            "honeydew"           "honeydew1"
"honeydew2"
```

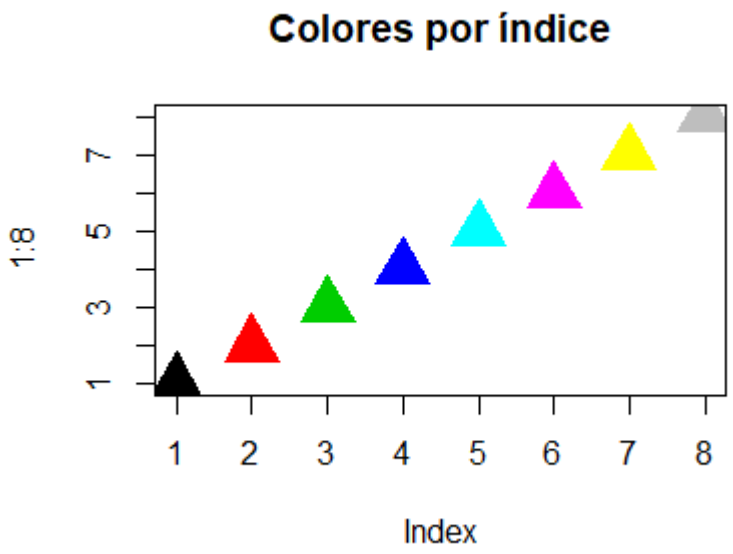
```
[365] "honeydew3"            "honeydew4"            "hotpink"
"hotpink1"
[369] "hotpink2"             "hotpink3"             "hotpink4"
"indianred"
[373] "indianred1"           "indianred2"           "indianred3"
"indianred4"
[377] "ivory"                "ivory1"               "ivory2"
"ivory3"
[381] "ivory4"               "khaki"                "khaki1"
"khaki2"
[385] "khaki3"               "khaki4"               "lavender"
"lavenderblush"
[389] "lavenderblush1"       "lavenderblush2"       "lavenderblush
3"       "lavenderblush4"
[393] "lawngreen"            "lemonchiffon"         "lemonchiffon1
"        "lemonchiffon2"
[397] "lemonchiffon3"        "lemonchiffon4"        "lightblue"
"lightblue1"
[401] "lightblue2"           "lightblue3"           "lightblue4"
"lightcoral"
[405] "lightcyan"            "lightcyan1"           "lightcyan2"
"lightcyan3"
[409] "lightcyan4"           "lightgoldenrod"       "lightgoldenro
d1"      "lightgoldenrod2"
[413] "lightgoldenrod3"      "lightgoldenrod4"      "lightgoldenro
dyellow" "lightgray"
[417] "lightgreen"           "lightgrey"            "lightpink"
"lightpink1"
[421] "lightpink2"           "lightpink3"           "lightpink4"
"lightsalmon"
[425] "lightsalmon1"         "lightsalmon2"         "lightsalmon3"
"lightsalmon4"
[429] "lightseagreen"        "lightskyblue"         "lightskyblue1
"        "lightskyblue2"
[433] "lightskyblue3"        "lightskyblue4"        "lightslateblu
e"       "lightslategray"
[437] "lightslategrey"       "lightsteelblue"       "lightsteelblu
e1"      "lightsteelblue2"
[441] "lightsteelblue3"      "lightsteelblue4"      "lightyellow"
"lightyellow1"
[445] "lightyellow2"         "lightyellow3"         "lightyellow4"
"limegreen"
[449] "linen"                "magenta"              "magenta1"
"magenta2"
[453] "magenta3"             "magenta4"             "maroon"
"maroon1"
[457] "maroon2"              "maroon3"              "maroon4"
"mediumaquamarine"
[461] "mediumblue"           "mediumorchid"         "mediumorchid1
"        "mediumorchid2"
[465] "mediumorchid3"        "mediumorchid4"        "mediumpurple"
"mediumpurple1"
[469] "mediumpurple2"        "mediumpurple3"        "mediumpurple4
"        "mediumseagreen"
[473] "mediumslateblue"      "mediumspringgreen"    "mediumturquoi
se"      "mediumvioletred"
```

```
[477] "midnightblue"        "mintcream"           "mistyrose"
"mistyrose1"
[481] "mistyrose2"          "mistyrose3"          "mistyrose4"
"moccasin"
[485] "navajowhite"         "navajowhite1"        "navajowhite2"
"navajowhite3"
[489] "navajowhite4"        "navy"                "navyblue"
"oldlace"
[493] "olivedrab"           "olivedrab1"          "olivedrab2"
"olivedrab3"
[497] "olivedrab4"          "orange"              "orange1"
"orange2"
[501] "orange3"             "orange4"             "orangered"
"orangered1"
[505] "orangered2"          "orangered3"          "orangered4"
"orchid"
[509] "orchid1"             "orchid2"             "orchid3"
"orchid4"
[513] "palegoldenrod"       "palegreen"           "palegreen1"
"palegreen2"
[517] "palegreen3"          "palegreen4"          "paleturquoise
"        "paleturquoise1"
[521] "paleturquoise2"      "paleturquoise3"      "paleturquoise
4"       "palevioletred"
[525] "palevioletred1"      "palevioletred2"      "palevioletred
3"       "palevioletred4"
[529] "papayawhip"          "peachpuff"           "peachpuff1"
"peachpuff2"
[533] "peachpuff3"          "peachpuff4"          "peru"
"pink"
[537] "pink1"               "pink2"               "pink3"
"pink4"
[541] "plum"                "plum1"               "plum2"
"plum3"
[545] "plum4"               "powderblue"          "purple"
"purple1"
[549] "purple2"             "purple3"             "purple4"
"red"
[553] "red1"                "red2"                "red3"
"red4"
[557] "rosybrown"           "rosybrown1"          "rosybrown2"
"rosybrown3"
[561] "rosybrown4"          "royalblue"           "royalblue1"
"royalblue2"
[565] "royalblue3"          "royalblue4"          "saddlebrown"
"salmon"
[569] "salmon1"             "salmon2"             "salmon3"
"salmon4"
[573] "sandybrown"          "seagreen"            "seagreen1"
"seagreen2"
[577] "seagreen3"           "seagreen4"           "seashell"
"seashell1"
[581] "seashell2"           "seashell3"           "seashell4"
"sienna"
[585] "sienna1"             "sienna2"             "sienna3"
"sienna4"
```

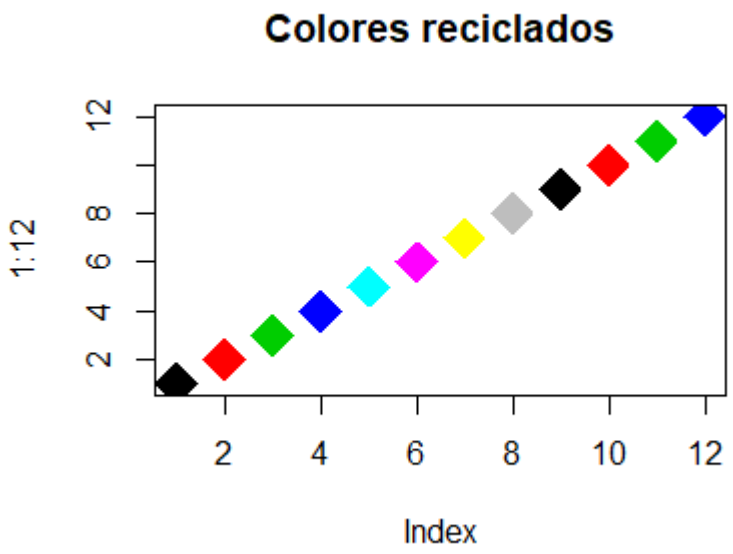
```
[589] "skyblue"             "skyblue1"            "skyblue2"
"skyblue3"
[593] "skyblue4"            "slateblue"           "slateblue1"
"slateblue2"
[597] "slateblue3"          "slateblue4"          "slategray"
"slategray1"
[601] "slategray2"          "slategray3"          "slategray4"
"slategrey"
[605] "snow"                "snow1"               "snow2"
"snow3"
[609] "snow4"               "springgreen"         "springgreen1"
"springgreen2"
[613] "springgreen3"        "springgreen4"        "steelblue"
"steelblue1"
[617] "steelblue2"          "steelblue3"          "steelblue4"
"tan"
[621] "tan1"                "tan2"                "tan3"
"tan4"
[625] "thistle"             "thistle1"            "thistle2"
"thistle3"
[629] "thistle4"            "tomato"              "tomato1"
"tomato2"
[633] "tomato3"             "tomato4"             "turquoise"
"turquoise1"
[637] "turquoise2"          "turquoise3"          "turquoise4"
"violet"
[641] "violetred"           "violetred1"          "violetred2"
"violetred3"
[645] "violetred4"          "wheat"               "wheat1"
"wheat2"
[649] "wheat3"              "wheat4"              "whitesmoke"
"yellow"
[653] "yellow1"             "yellow2"             "yellow3"
"yellow4"
[657] "yellowgreen"

| ¡Traes una muy buena racha!

  |==============
|  14%
| Notarás que al final de la lista varios colores contienen la sub
cadena "yellow". Una manera de buscar todos
| los colores que contengan la subcadena "yellow", es usando la fu
nción grep().

...

  |================
|  16%
| Ve el uso de grep() usando la función help().

>
> help("grep")

| ¡Lo estás haciendo muy bien!
```

```
  |==================
|  19%
| La función grep() toma al argumento pattern (una expresión regul
ar), lee el argumento x (un vector de
| caracteres, o un objeto que puede ser convertido por la función
as.character() a un vector de caracteres),
| y regresa las coincidencias de pattern dentro de x.

...

  |=====================
|  22%
| Para buscar todos los colores que contengan la subcadena "yellow
" ingresa grep("yellow", colors(),
| value=TRUE) en la línea de comandos.

> grep("yellow", colors(), value=TRUE)
 [1] "greenyellow"          "lightgoldenrodyellow" "lightyellow"
"lightyellow1"
 [5] "lightyellow2"         "lightyellow3"         "lightyellow4"
"yellow"
 [9] "yellow1"              "yellow2"              "yellow3"
"yellow4"
[13] "yellowgreen"

| ¡Excelente trabajo!

  |=======================
|  24%
| Ahora sabes que R contiene 13 tipos de amarillos (yellow). Notar
ás que usaste el argumento value de la
| función grep(); esto se debe a que de no haberlo usado, grep() t
e hubiera regresado las posiciones en donde
| se encontraron coincidencias, en vez de las cadenas en donde las
encontró.

...

  |===========================
|  27%
| La lista de colores válidos es muy extensa. Obten su longitud.

> length(colors())
[1] 657

| ¡Acertaste!

  |=============================
|  30%
| Entonces ahora ya sabes que R tiene una gran lista de más de 650
colores que puedes usar por nombre.

...

  |===============================
|  32%
```

```
| Para especificar un color usando componentes RGB, usa una cadena
de la forma "#RRGGBB", donde RR, GG y BB
| son valores hexadecimales que especifican la cantidad de rojo, v
erde y azul, respectivamente.

...

  |=================================
|   35%
| Si deseas hacer referencia a un índice de la paleta de colores m
ediante un entero, debes saber que existen
| ocho colores en la paleta por defecto.

...

  |=====================================
|   38%
| La función palette() es usada para ver o manipular la paleta de
colores. La paleta de colores es usada
| cuando el parámetro col es usado con un valor numérico.

...

  |=========================================
|   41%
| Ingresa palette() para conocer la paleta por defecto.

> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "y
ellow"  "gray"

| ¡Bien hecho!

  |=========================================
|   43%
| Cada color es representado por un número entero; es decir, el nú
mero 1 representa al color negro, el 2 al
| rojo… y el 8 al gris. Ingresa plot(1:8, col=1:8, main="Colores p
or índice", pch=17, cex=3) en la línea de
| comandos.

> plot(1:8, col=1:8, main="Colores por índice", pch=17, cex=3)
```

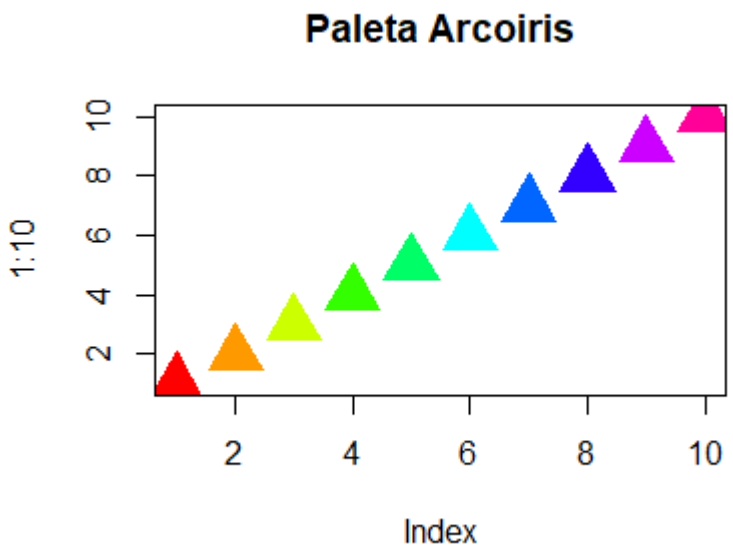



```
| ¡Excelente trabajo!

  |=============================================
|   46%
| Como recordarás, el argumento main es usado para establecer el t
ítulo de la gráfica, el argumento pch para
| establecer el símbolo con el que se graficará y el argumento cex
para indicar el tamaño del símbolo usado
| para graficar.

...

  |=============================================
|   49%
| Si usas un número más grande que ocho para graficar, los colores
serán reciclados. Compruébalo: ingresa
| plot(1:12, col=1:12, main="Colores reciclados", pch=18, cex=3) e
n la línea de comandos.

> plot(1:12, col=1:12, main="Colores reciclados", pch=18, cex=3)
```

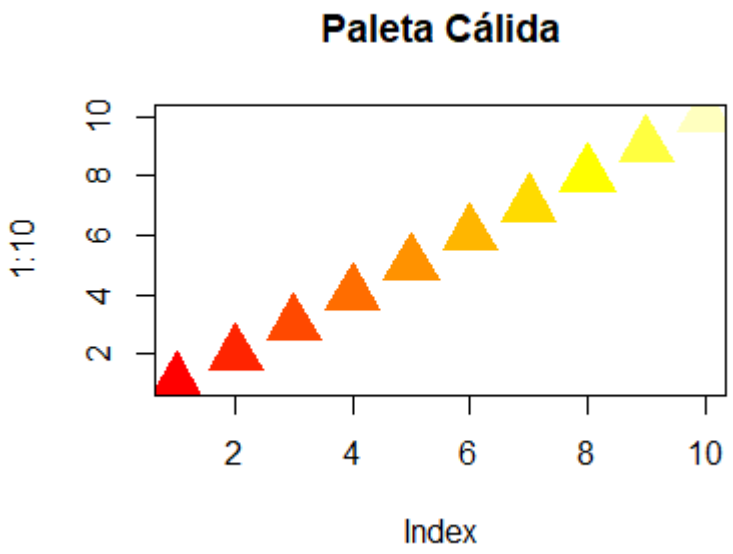



```
| ¡Buen trabajo!

  |=======================================================
|   51%
| Si requieres un número más grande de colores para graficar, nece
sitarás acceder a una paleta más grande.

...

  |=========================================================
|   54%
| Existen varias funciones incluidas en R que te regresan colores
continuos (paletas de tamaño variable), que
| le pueden dar aspectos diferentes a tus gráficas.

...

  |===========================================================
|   57%
| Algunos ejemplos de estas funciones son rainbow(), heat.colors()
, topo.colors(), y terrain.colors().

...

  |=============================================================
|   59%
| Para cambiar la paleta de colores por defecto por cualquiera de
estas paletas, lo primero que debes hacer
| es crear una paleta. Para crear una nueva paleta basta con llama
r a cualquiera de las funciones
| anteriormente mencionadas, mandando como argumento el número de
colores que deseas que contenga la paleta.
| Por ejemplo, ingresa paleta_arcoiris <- rainbow(10) en la línea
de comandos para crear una paleta con 10
| colores.
```

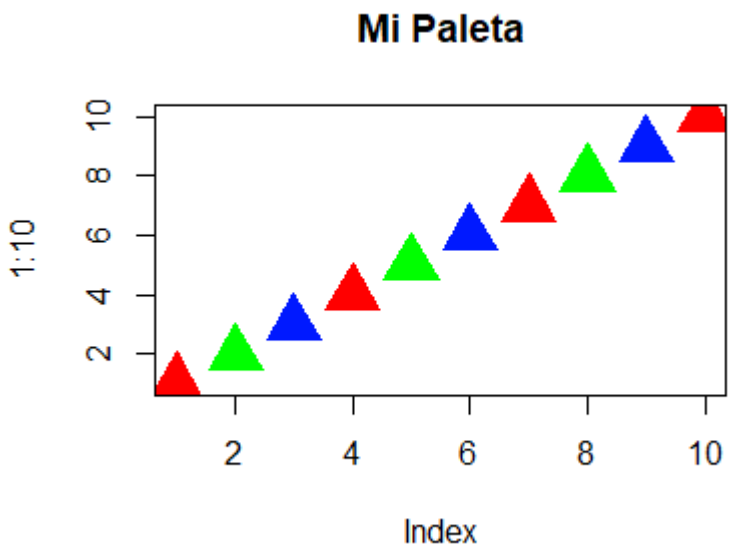
```
> paleta_arcoiris <- rainbow(10)

| ¡Toda esa práctica está rindiendo frutos!

  |======================================================================
|  62%
| Ahora ve el contenido de la paleta que acabas de crear.

> paleta_arcoiris
 [1] "#FF0000FF" "#FF9900FF" "#CCFF00FF" "#33FF00FF" "#00FF66FF" "
#00FFFFFF" "#0066FFFF" "#3300FFFF"
 [9] "#CC00FFFF" "#FF0099FF"

| ¡Lo estás haciendo muy bien!

  |======================================================================
==                                      |  65%
| Como verás, la nueva paleta que creaste usando rainbow(10) conti
ene colores que usan componentes RGB.

...

  |======================================================================
=====                                   |  68%
| Para cambiar la paleta de colores por la que creaste, ingresa pa
lette(paleta_arcoiris) en la línea de
| comandos.

> palette(paleta_arcoiris)

| ¡Acertaste!

  |======================================================================
=======                                 |  70%
| Verifica que los cambios se efectuaron en la paleta de colores.

> palette()
 [1] "red"     "#FF9900" "#CCFF00" "#33FF00" "#00FF66" "cyan"    "
#0066FF" "#3300FF" "#CC00FF" "#FF0099"

| ¡Lo estás haciendo muy bien!

  |======================================================================
==========                              |  73%
| Ahora grafica usando la nueva paleta; ingresa plot(1:10, col=1:1
0, main="Paleta Arcoiris",pch=17, cex=3) en
| la línea de comandos.

> plot(1:10, col=1:10, main="Paleta Arcoiris",pch=17, cex=3)
```



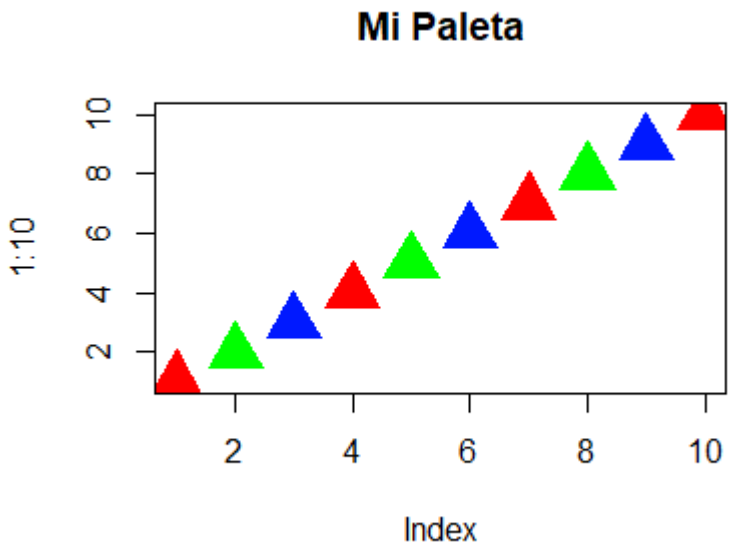

```
| ¡Muy bien!

  |======================================================================
=============                                        |  76%
| Prueba con otra paleta; ingresa paleta_calida <- heat.colors(10)
para crear una paleta con 10 colores
| cálidos.

> paleta_calida <- heat.colors(10)

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |======================================================================
===============                                      |  78%
| Ahora cambia la paleta actual por la nueva paleta creada.

> palette(paleta_calida)

| ¡Eres bastante bueno!

  |======================================================================
==================                                   |  81%
| Y ahora grafica plot(1:10, col=1:10, main="Paleta Cálida",pch=17
, cex=3) para ver los cambios.

> plot(1:10, col=1:10, main="Paleta Cálida",pch=17, cex=3)
```




```
| ¡Todo ese trabajo está rindiendo frutos!

  |======================================================================
=====================                                  |  84%
| Todas las diferentes paletas son muy bonitas y útiles de diferen
tes maneras, pero tal vez no es exactamente
| lo que estás buscando, o tal vez tú quieres tener más control so
bre los colores. Para especificar los
| colores que quieras puedes crear un vector que contenga dichos c
olores. Por ejemplo, crea el vector
| 'mi_paleta' que contenga los colores "red", "green" y "#0019FFFF
".

> mi_paleta <- c("red", "green", "#0019FFFF")

| Perseverancia es la respuesta.

  |======================================================================
======================                                 |  86%
| Y ahora puedes usar el vector directamente para graficar. Por ej
emplo, ingresa plot(1:10, col=mi_paleta,
| main="Mi Paleta",pch=17, cex=3) en la línea de comandos.

> plot(1:10, col=mi_paleta, main="Mi Paleta",pch=17, cex=3)
```



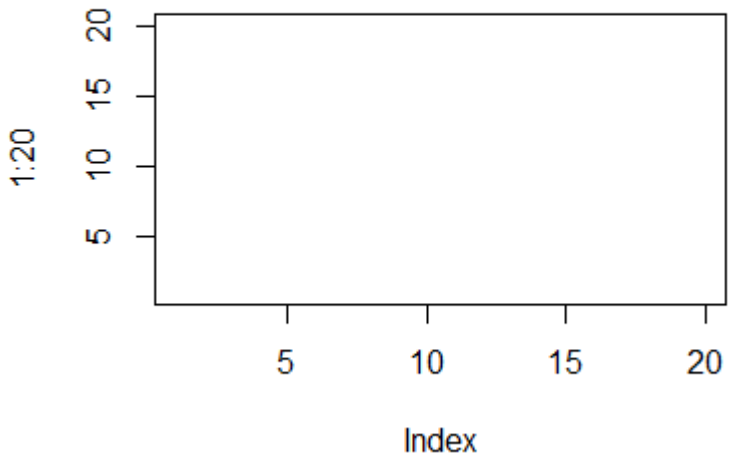

```
| ¡Traes una muy buena racha!

  |======================================================================
=========================             |  89%
| O establecerlo como tu nueva paleta. Ingresa palette(mi_paleta)
en la línea de comandos.

> palette(mi_paleta)

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |======================================================================
============================          |  92%
| Y graficar haciendo referencia por índice. Ingresa plot(1:10, co
l=1:10, main="Mi Paleta",pch=17, cex=3) en
| la línea de comandos.

> plot(1:10, col=1:10, main="Mi Paleta",pch=17, cex=3)
```




```
| ¡Eres el mejor!

  |======================================================================
================================     |  95%
| Si deseas regresar a la paleta de colores por defecto, debes de
ingresar palette("default") en la línea de
| comandos. ¡Ahora hazlo!

> palette("default")

| ¡Eres bastante bueno!

  |======================================================================
================================     |  97%
| Otras funciones que están disponibles para especificar colores s
on rgb(), hsv(), hcl() y gray(), pero su
| uso es diferente a las que usaste anteriormente. Si deseas usarl
as consulta su página de ayuda.

...

  |======================================================================
================================| 100%
```

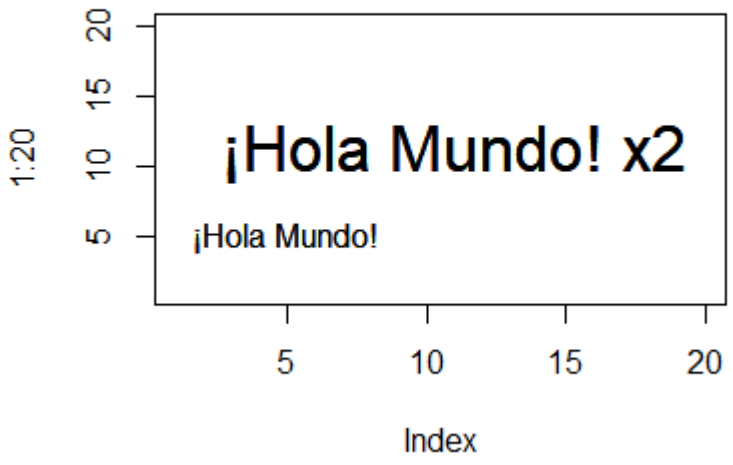
Lección de swirl: 10. Graficación con texto y notación matemática

```
Selection: 10
  |
|   0%

| En esta lección conocerás cómo utilizar notación matemática en R
.

...

  |====
|   4%
| Antes de comenzar con notación matemática es importante que sepa
s que existe la función text().

...

  |=======
|   7%
| La función text() dibuja cadenas sobre gráficas. Para probar la
función text(), crea una gráfica vacía de
| tamaño 20x20. Ingresa plot(1:20, type="n") en la línea de comand
os.

> plot(1:20, type="n")
```



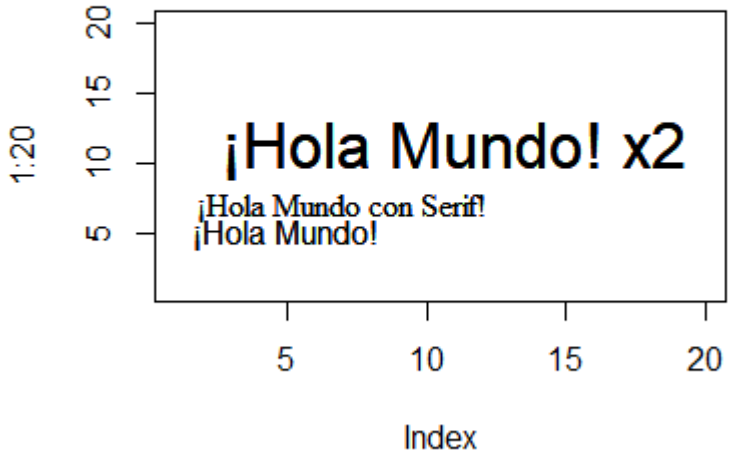

```
| ¡Eso es trabajo bien hecho!

  |===========
|  11%
| Ahora ingresa text(5, 5, "¡Hola Mundo!") en la línea de comandos
para probar la función text().
> text(5, 5, "¡Hola Mundo!")

| ¡Mantén este buen nivel!
```

```
  |==============
|  14%
| Con esto graficaste la cadena "¡Hola Mundo!" en las coordenadas
(5, 5). Para conocer más acerca de text()
| ve su página de ayuda.

> ?text

| ¡Sigue trabajando de esa manera y llegarás lejos!

  |==================
|  18%
| La función text() dibuja las cadenas dadas por el argumento labe
ls en las coordenadas x y y.

...

  |====================
|  21%
| Muchos parámetros controlan el aspecto en que los textos son mos
trados en la gráfica.

...

  |=========================
|  25%
| Si deseas cambiar el tamaño de los textos producidos por la func
ión text(), puedes usar el argumento cex de
| la función text(), el cual especifica un factor de escala por de
fecto para el texto. Pruébalo. Ingresa
| text(11, 11, "¡Hola Mundo! x2", cex=2) en la línea de comandos.

>
> text(11, 11, "¡Hola Mundo! x2", cex=2)
```

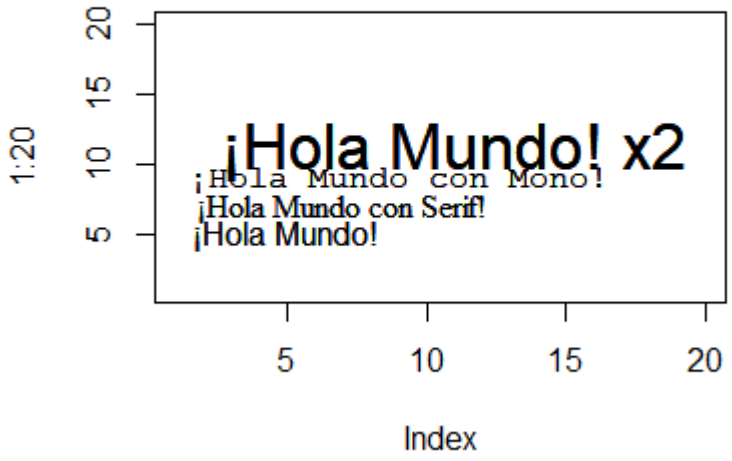



```
| ¡Bien hecho!
```

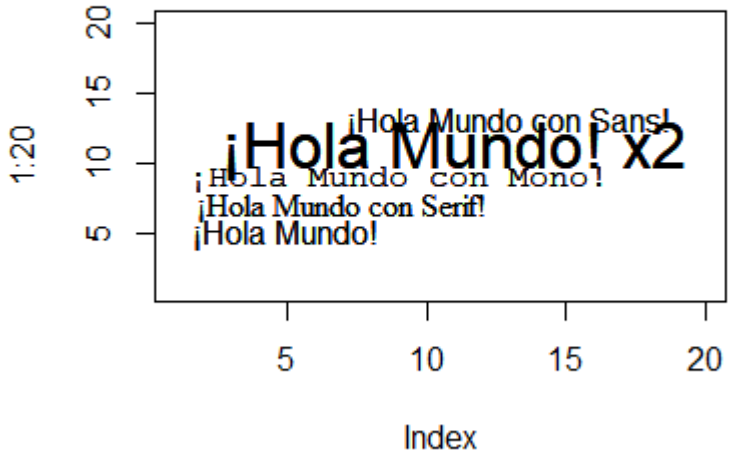
```
  |=============================
|  29%
| Otro aspecto importante son los tipos de fuentes que puedes usar
. Una manera simple para cambiar los tipos
| de fuentes es usando el parámetro family. Ingresa text(7, 7, "¡H
ola Mundo con Serif!", family="serif") en
| la línea de comandos.
> text(7, 7, "¡Hola Mundo con Serif!", family="serif")
```




```
| ¡Acertaste!

  |=============================
|  32%
| Los valores comunes usados para el argumento family son "serif",
"sans", "mono" y "symbol". Ahora prueba la
| fuente "mono". Ingresa text(9, 9, "¡Hola Mundo con Mono!", famil
y="mono") en la línea de comandos.
> text(9, 9, "¡Hola Mundo con Mono!", family="mono")
```




```
| ¡Acertaste!

  |=====================================
|   36%
| Ahora ingresa text(13, 13, "¡Hola Mundo con Sans!", family="sans
") en la línea de comandos.
> text(13, 13, "¡Hola Mundo con Sans!", family="sans")
```




```
| ¡Mantén este buen nivel!

  |=======================================
|   39%
| Notarás que la fuente por defecto es "sans".

...
```

```
  |=============================================
|  43%
| Volviendo a la notación matemática...

...

  |==============================================
|  46%
| Muchas veces necesitarás etiquetas o títulos donde tengas que ut
ilizar notación matemática; es decir, que
| tengas que hacer uso de símbolos y/o expresiones matemáticas.

...

  |===============================================
|  50%
| Ingresa demo(plotmath) para ver un ejemplo de expresiones matemá
ticas que puedes usar en R.
> demo(plotmath)


        demo(plotmath)
        ---- ~~~~~~~~

Type  <Return>        to start :

> #  Copyright (C) 2002-2016 The R Core Team
>
> require(datasets)

> require(grDevices); require(graphics)

> ## --- "math annotation" in plots :
>
> ######
> # create tables of mathematical annotation functionality
> ######
> make.table <- function(nr, nc) {
+     savepar <- par(mar=rep(0, 4), pty="s")
+     plot(c(0, nc*2 + 1), c(0, -(nr + 1)),
+          type="n", xlab="", ylab="", axes=FALSE)
+     savepar
+ }

> get.r <- function(i, nr) {
+     i %% nr + 1
+ }

> get.c <- function(i, nr) {
+     i %/% nr + 1
+ }

> draw.title.cell <- function(title, i, nr) {
+     r <- get.r(i, nr)
+     c <- get.c(i, nr)
+     text(2*c - .5, -r, title)
+     rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5))
```

```
+ }

> draw.plotmath.cell <- function(expr, i, nr, string = NULL) {
+     r <- get.r(i, nr)
+     c <- get.c(i, nr)
+     if (is.null(string)) {
+         string <- deparse(expr)
+         string <- substr(string, 12, nchar(string) - 1)
+     }
+     text((2*(c - 1) + 1), -r, string, col="grey50")
+     text((2*c), -r, expr, adj=c(.5,.5))
+     rect((2*(c - 1) + .5), -(r - .5), (2*c + .5), -(r + .5), bor
der="grey")
+ }

> nr <- 20

> nc <- 2

> oldpar <- make.table(nr, nc)
Hit <Return> to see next plot:

> i <- 0

> draw.title.cell("Arithmetic Operators", i, nr); i <- i + 1

> draw.plotmath.cell(expression(x + y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x - y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x / y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %+-% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %/% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %*% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %.% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(-x), i, nr); i <- i + 1

> draw.plotmath.cell(expression(+x), i, nr); i <- i + 1

> draw.title.cell("Sub/Superscripts", i, nr); i <- i + 1

> draw.plotmath.cell(expression(x[i]), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x^2), i, nr); i <- i + 1

> draw.title.cell("Juxtaposition", i, nr); i <- i + 1

> draw.plotmath.cell(expression(x * y), i, nr); i <- i + 1
```

```
> draw.plotmath.cell(expression(paste(x, y, z)), i, nr); i <- i +
1

> draw.title.cell("Radicals", i, nr); i <- i + 1

> draw.plotmath.cell(expression(sqrt(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(sqrt(x, y)), i, nr); i <- i + 1

> draw.title.cell("Lists", i, nr); i <- i + 1

> draw.plotmath.cell(expression(list(x, y, z)), i, nr); i <- i + 1

> draw.title.cell("Relations", i, nr); i <- i + 1

> draw.plotmath.cell(expression(x == y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x != y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x < y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x <= y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x > y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x >= y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %~~% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %=~% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %==% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %prop% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %~% y), i, nr); i <- i + 1

> draw.title.cell("Typeface", i, nr); i <- i + 1

> draw.plotmath.cell(expression(plain(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(italic(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(bold(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(bolditalic(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(underline(x)), i, nr); i <- i + 1

> # Need fewer, wider columns for ellipsis ...
> nr <- 20

> nc <- 2

> make.table(nr, nc)
Hit <Return> to see next plot:
$`mar`
```

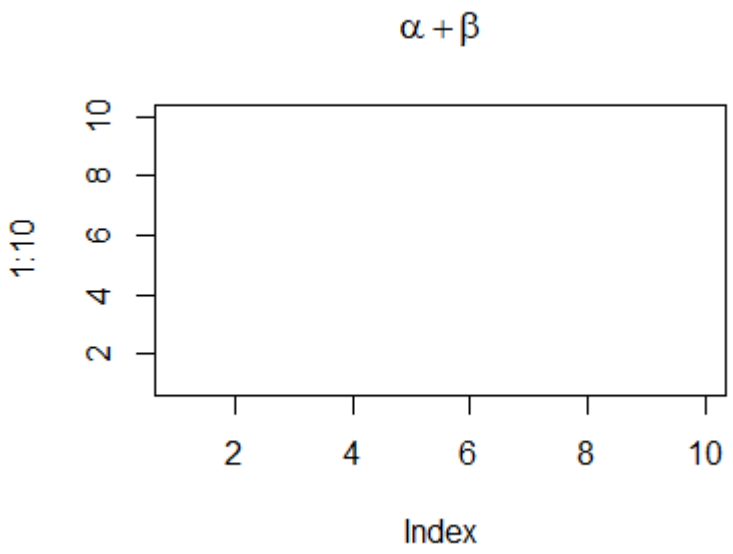
```
[1] 0 0 0 0

$pty
[1] "s"


> i <- 0

> draw.title.cell("Ellipsis", i, nr); i <- i + 1

> draw.plotmath.cell(expression(list(x[1], ..., x[n])), i, nr); i
<- i + 1

> draw.plotmath.cell(expression(x[1] + ... + x[n]), i, nr); i <- i
+ 1

> draw.plotmath.cell(expression(list(x[1], cdots, x[n])), i, nr);
i <- i + 1

> draw.plotmath.cell(expression(x[1] + ldots + x[n]), i, nr); i <-
i + 1

> draw.title.cell("Set Relations", i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %subset% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %subseteq% y), i, nr); i <- i +
1

> draw.plotmath.cell(expression(x %supset% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %supseteq% y), i, nr); i <- i +
1

> draw.plotmath.cell(expression(x %notsubset% y), i, nr); i <- i +
1

> draw.plotmath.cell(expression(x %in% y), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x %notin% y), i, nr); i <- i + 1

> draw.title.cell("Accents", i, nr); i <- i + 1

> draw.plotmath.cell(expression(hat(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(tilde(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(ring(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(bar(xy)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(widehat(xy)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(widetilde(xy)), i, nr); i <- i + 1

> draw.title.cell("Arrows", i, nr); i <- i + 1
```

```
> draw.plotmath.cell(expression(x %<->% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %->% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %<-% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %up% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %down% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %<=>% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %=>% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %<=% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %dblup% y), i, nr); i <- i + 1
> draw.plotmath.cell(expression(x %dbldown% y), i, nr); i <- i + 1
> draw.title.cell("Symbolic Names", i, nr); i <- i + 1
> draw.plotmath.cell(expression(Alpha - Omega), i, nr); i <- i + 1
> draw.plotmath.cell(expression(alpha - omega), i, nr); i <- i + 1
> draw.plotmath.cell(expression(phi1 + sigma1), i, nr); i <- i + 1
> draw.plotmath.cell(expression(Upsilon1), i, nr); i <- i + 1
> draw.plotmath.cell(expression(infinity), i, nr); i <- i + 1
> draw.plotmath.cell(expression(32 * degree), i, nr); i <- i + 1
> draw.plotmath.cell(expression(60 * minute), i, nr); i <- i + 1
> draw.plotmath.cell(expression(30 * second), i, nr); i <- i + 1
> # Need even fewer, wider columns for typeface and style ...
> nr <- 20
> nc <- 1
> make.table(nr, nc)
Hit <Return> to see next plot:
$`mar`
[1] 0 0 0 0

$pty
[1] "s"


> i <- 0
> draw.title.cell("Style", i, nr); i <- i + 1
```

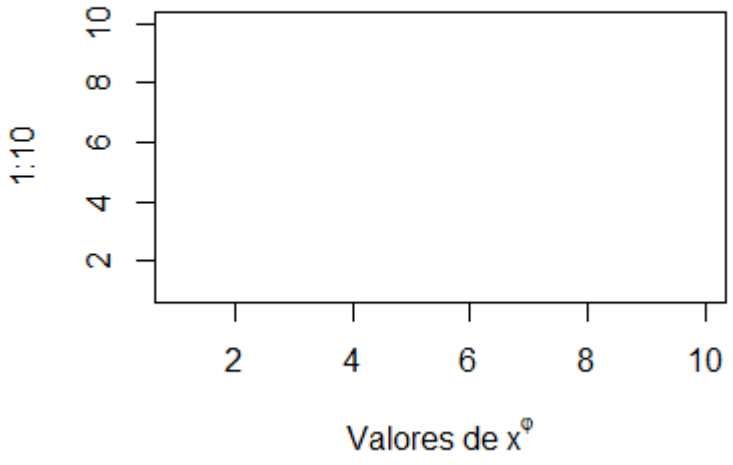
```
> draw.plotmath.cell(expression(displaystyle(x)), i, nr); i <- i +
1

> draw.plotmath.cell(expression(textstyle(x)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(scriptstyle(x)), i, nr); i <- i +
1

> draw.plotmath.cell(expression(scriptscriptstyle(x)), i, nr); i <
- i + 1

> draw.title.cell("Spacing", i, nr); i <- i + 1

> draw.plotmath.cell(expression(x ~~ y), i, nr); i <- i + 1

> # Need fewer, taller rows for fractions ...
> # cheat a bit to save pages
> par(new = TRUE)

> nr <- 10

> nc <- 1

> make.table(nr, nc)
$`mar`
[1] 0 0 0 0

$pty
[1] "s"


> i <- 4

> draw.plotmath.cell(expression(x + phantom(0) + y), i, nr); i <-
i + 1

> draw.plotmath.cell(expression(x + over(1, phantom(0))), i, nr);
i <- i + 1

> draw.title.cell("Fractions", i, nr); i <- i + 1

> draw.plotmath.cell(expression(frac(x, y)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(over(x, y)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(atop(x, y)), i, nr); i <- i + 1

> # Need fewer, taller rows and fewer, wider columns for big opera
tors ...
> nr <- 10

> nc <- 1

> make.table(nr, nc)
Hit <Return> to see next plot:
$`mar`
[1] 0 0 0 0
```

```
$pty
[1] "s"


> i <- 0

> draw.title.cell("Big Operators", i, nr); i <- i + 1

> draw.plotmath.cell(expression(sum(x[i], i=1, n)), i, nr); i <- i
+ 1

> draw.plotmath.cell(expression(prod(plain(P)(X == x), x)), i, nr)
; i <- i + 1

> draw.plotmath.cell(expression(integral(f(x) * dx, a, b)), i, nr)
; i <- i + 1

> draw.plotmath.cell(expression(union(A[i], i==1, n)), i, nr); i <
- i + 1

> draw.plotmath.cell(expression(intersect(A[i], i==1, n)), i, nr);
i <- i + 1

> draw.plotmath.cell(expression(lim(f(x), x %->% 0)), i, nr); i <-
i + 1

> draw.plotmath.cell(expression(min(g(x), x >= 0)), i, nr); i <- i
+ 1

> draw.plotmath.cell(expression(inf(S)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(sup(S)), i, nr); i <- i + 1

> nr <- 11

> make.table(nr, nc)
Hit <Return> to see next plot:
$`mar`
[1] 0 0 0 0

$pty
[1] "s"


> i <- 0

> draw.title.cell("Grouping", i, nr); i <- i + 1

> # Those involving '{ . }' have to be done "by hand"
> draw.plotmath.cell(expression({}(x , y)), i, nr, string="{}(x, y
)"); i <- i + 1

> draw.plotmath.cell(expression((x + y)*z), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x^y + z),   i, nr); i <- i + 1
```

```
> draw.plotmath.cell(expression(x^(y + z)), i, nr); i <- i + 1

> draw.plotmath.cell(expression(x^{y + z}), i, nr, string="x^{y +
z}"); i <- i + 1

> draw.plotmath.cell(expression(group("(", list(a, b), "]")), i, n
r); i <- i + 1

> draw.plotmath.cell(expression(bgroup("(", atop(x, y), ")")), i,
nr); i <- i + 1

> draw.plotmath.cell(expression(group(lceil, x, rceil)), i, nr); i
<- i + 1

> draw.plotmath.cell(expression(group(lfloor, x, rfloor)), i, nr);
i <- i + 1

> draw.plotmath.cell(expression(group("|", x, "|")), i, nr); i <-
i + 1

> par(oldpar)

Type  <Return>        to start :
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:

| ¡Excelente trabajo!

  |=======================================================
|  54%
| Las columnas de texto gris muestran las expresiones en R, y la c
olumna de texto negra muestra cómo se
| verán.

...

  |=========================================================
|  57%
| Para generar estos símbolos matemáticos, en R puedes utilizar la
función expression(). La función
| expression() recibe como argumento una expresión como las que vi
ste en color gris. Pruébalo. Ingresa
| plot(1:10, type="n", main=expression(alpha + beta)) en la línea
de comandos.

> plot(1:10, type="n", main=expression(alpha + beta))
```

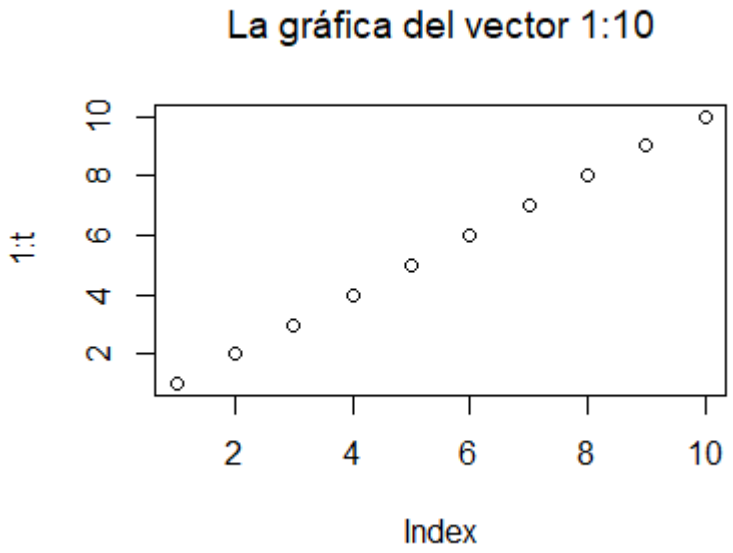



```
| ¡Sigue trabajando de esa manera y llegarás lejos!

  |======================================================================
|   61%
| Además, puedes combinar expression() con la función paste() para
poder tener texto y símbolos en una misma
| sentencia. Ingresa plot(1:10, type="n", xlab = expression(paste(
"Valores de ", x^phi1))) en la línea de
| comandos.
> plot(1:10, type="n", xlab = expression(paste("Valores de ", x^ph
i1)))
```




```
| ¡Eres el mejor!

  |======================================================================
=                                      |   64%
```

```
| Si deseas revisar la lista de expresiones disponibles, ingresa ?
plotmath en la línea de comandos.
> ?plotmath

| ¡Traes una muy buena racha!

  |=======================================================================
=====                                    |  68%
| Además, puedes utilizar la función substitute() que te permitirá
obtener el valor de una variable u objeto
| en R; esto lo podrás usar dentro de una función paste() para gen
erar una expresión.

...

  |=======================================================================
========                                 |  71%
| Para mostrar su uso, ingresa t <- 10 en la línea de comandos.

> t <- 10

| ¡Excelente!

  |=======================================================================
============                             |  75%
| Y ahora ingresa plot(1:t, main=substitute(paste("La gráfica del
vector 1:", x), list(x=t))) en la línea de
| comandos.

> plot(1:t, main=substitute(paste("La gráfica del vector 1:", x),
list(x=t)))
```




```
| ¡Eso es correcto!

  |=======================================================================
================                         |  79%
| Observa el título de la gráfica: "La gráfica del vector 1:10".
```

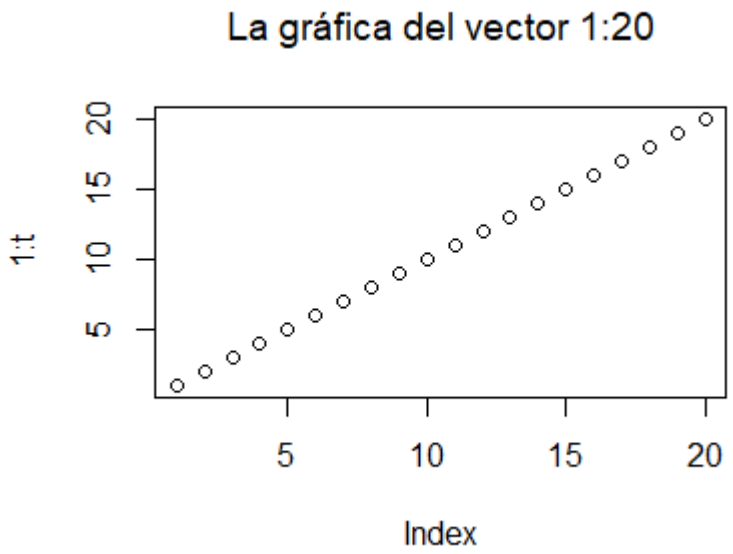
...

  |===
================== | 82%
| Si ahora cambias el valor de t, ingresa t <- 20 en la línea de c
omandos.
> t <- 20

| ¡Eso es trabajo bien hecho!

  |===
======================= | 86%
| Y nuevamente grafica plot(1:t, main=substitute(paste("La gráfica
del vector 1:", x), list(x=t))); cambiará
| el título de la gráfica. ¡Pruébalo!

> plot(1:t, main=substitute(paste("La gráfica del vector 1:", x),
list(x=t)))



| ¡Lo estás haciendo muy bien!

  |===
============================ | 89%
| Notaras que el título cambio.

...

  |===
================================ | 93%
| Esto es muy útil si quieres obtener gráficas donde alguno de los
parámetros es diferente; así puedes
| generar múltiples gráficas con distintos parámetros y las etique
tas de éstos pueden ir cambiando de acuerdo
| a los parámetros.

...

```
  |======================================================================
================================    |  96%
| Para conocer más acerca de la función substitute() ingresa ?subs
titute en la línea de comandos.
> ?substitute

| Perseverancia es la respuesta.

  |======================================================================
====================================| 100%
```

Lección de swirl: 11. Creación de Gráficas en 3D

```
Selection: 11

| Intentando cargar las dependencias de la lección...

| Esta lección requiere el paquete ‘rgl’ . ¿Quieres que lo instale
ahora?

1: Sí
2: No

Selection: 1

| Tratando de instalar el paquete ‘rgl’ ...
also installing the dependencies ‘colorspace’, ‘RColorBrewer’, ‘di
chromat’, ‘munsell’, ‘labeling’, ‘viridisLite’, ‘gtable’, ‘plyr’,
‘reshape2’, ‘scales’, ‘processx’, ‘xtable’, ‘sourcetools’, ‘ggplot
2’, ‘miniUI’, ‘webshot’, ‘htmlwidgets’, ‘shiny’, ‘crosstalk’, ‘man
ipulateWidget’

package ‘colorspace’ successfully unpacked and MD5 sums checked
package ‘RColorBrewer’ successfully unpacked and MD5 sums checked
package ‘dichromat’ successfully unpacked and MD5 sums checked
package ‘munsell’ successfully unpacked and MD5 sums checked
package ‘labeling’ successfully unpacked and MD5 sums checked
package ‘viridisLite’ successfully unpacked and MD5 sums checked
package ‘gtable’ successfully unpacked and MD5 sums checked
package ‘plyr’ successfully unpacked and MD5 sums checked
package ‘reshape2’ successfully unpacked and MD5 sums checked
package ‘scales’ successfully unpacked and MD5 sums checked
package ‘processx’ successfully unpacked and MD5 sums checked
package ‘xtable’ successfully unpacked and MD5 sums checked
package ‘sourcetools’ successfully unpacked and MD5 sums checked
package ‘ggplot2’ successfully unpacked and MD5 sums checked
package ‘miniUI’ successfully unpacked and MD5 sums checked
package ‘webshot’ successfully unpacked and MD5 sums checked
package ‘htmlwidgets’ successfully unpacked and MD5 sums checked
package ‘shiny’ successfully unpacked and MD5 sums checked
package ‘crosstalk’ successfully unpacked and MD5 sums checked
package ‘manipulateWidget’ successfully unpacked and MD5 sums chec
ked
package ‘rgl’ successfully unpacked and MD5 sums checked

| ¡El paquete ‘rgl’ se ha cargado correctamente!

  |
|   0%

| En esta lección verás cómo crear gráficas en 3D.

...

  |====
|   4%
| Este tipo de gráficas no están implementadas en el sistema de ba
se, por lo que usarás rgl.
```

```
...

  |========
|   8%
| Para ejemplificar esta práctica continuarás usando el conjunto d
e datos iris. Carga el conjunto de datos
| iris.

> data(iris)

| ¡Muy bien!

  |============
|  12%
| Usa head(iris) para ver las primeras seis líneas de contenido de
iris.

> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

| ¡Excelente trabajo!

  |================
|  17%
| Con esto recordarás que el conjunto de datos de iris contiene la
s medidas de longitud y anchura del sépalo
| y pétalo, respectivamente, por 50 flores de cada una de las tres
especies de iris. Las especies de iris
| son: setosa, versicolor y virginica.

...

  |====================
|  21%
| Ingresa x <- iris$Sepal.Length en la línea de comandos para guar
dar la columna que contiene las longitudes
| de sépalo de las muestras en la variable x.

> x <- iris$Sepal.Length

| ¡Eres bastante bueno!

  |========================
|  25%
| Repite el proceso anterior, pero guarda en la variable y, la col
umna que contiene las longitudes de pétalo.

> y <- iris$Petal.Length

| ¡Acertaste!
```

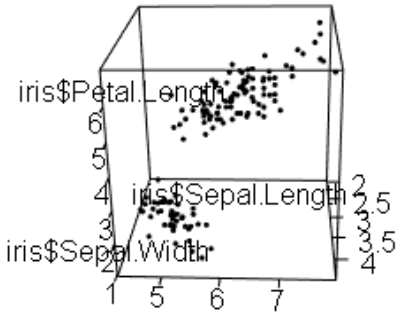
```
  |=============================
|  29%
| Nuevamente repite el proceso, pero ahora guarda en la variable z
la columna que contiene las anchuras de
| sépalo.

> z <- iris$Sepal.Width

| ¡Acertaste!

  |==================================
|  33%
| Bien; ahora puedes continuar...

...

  |=======================================
|  38%
| rgl es un paquete de gráficos 3D que produce gráficas interactiv
as en 3D en tiempo real. Permite rotar de
| forma interactiva, ampliar los gráficos, etc. Ingresa ?rgl en la
línea de comandos para conocer un poco más
| del paquete rgl.

> ?rgl

| ¡Acertaste!

  |============================================
|  42%
| rgl incluye una interfaz de nivel superior llamada r3d. Esta int
erfaz está diseñada para actuar como los
| gráficos clásicos 2D de R.

...

  |================================================
|  46%
| Para inicializar la interfaz 3D usa la función open3d(). La func
ión open3d() intenta abrir una nueva
| ventana RGL, utilizando los valores predeterminados especificado
s por el usuario. Ingresa open3d() en la
| línea de comandos.

> open3d()
wgl
  1

| ¡Es asombroso!

  |===================================================
|  50%
| Ahora puedes utilizar la función plot3d(); ésta funciona de mane
ra similar a la función plot() del sistema
| base de R. En este caso, como es en tres dimensiones, recibe una
tripleta de valores, 'x', 'y', 'z', y una
```

| vez recibida esta tripleta dibuja el punto que deseas visualizar
.

...

  |===
| 54%
| Ingresa plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.
Width) en la línea de comandos.

```
> plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width)
```



| ¡Eres el mejor!

  |===
| 58%
| Al igual que con la función plot(), la gráfica fue construida to
mando un elemento de cada vector; es decir,
| para construir cada punto plot3d() tomo un elemento del primer v
ector (posición x), un elemento del segundo
| vector (posición y) y un elemento del tercer vector (posición z)
.

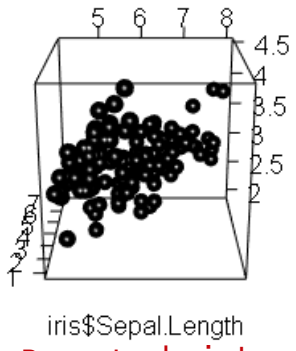
...

  |===
| 62%
| Notarás que algunos valores por defecto son el tipo de gráfica (
puntos) y el color negro.

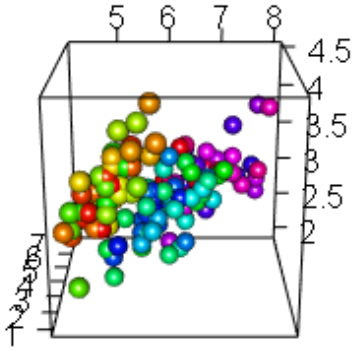
...

  |===
==== | 67%
| Al igual que con plot() puedes especificar el tipo de gráfica us
ando el parámetro type. Ingresa
| plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, t
ype="s") en la línea de comandos.

```
> plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, t
ype="s")
```



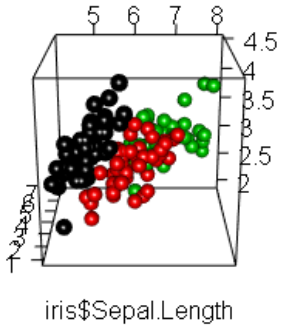

```
| ¡Buen trabajo!

  |=======================================================================
========                                  |  71%
| Los tipos soportados son 'p' para puntos, 's' para esferas, 'l'
para líneas, 'h' para segmentos de línea
| con z = 0, y 'n' para nada.

...

  |=======================================================================
============                              |  75%
| Asimismo, puedes especificar el color usado para graficar a los
elementos usando el parámetro col. Ingresa
| plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, t
ype="s", col=rainbow(150)) en la línea de
| comandos.
> plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, t
ype="s", col=rainbow(150))
```




```
| ¡Mantén este buen nivel!

  |=======================================================================
================                          |  79%
| Algo que debes saber es que si deseas graficar elemento por espe
cie no podrás enviarle al parámetro col
| iris$Species como lo habías estado haciendo con la función plot(
). Esta vez tendrás que tratar a
| iris$Species como enteros. Para hacer esto ingresa plot3d(iris$S
epal.Length, iris$Petal.Length,
```

| iris$Sepal.Width, type="s", col=as.integer(iris$Species)) en la línea de comandos.

```
> plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, type="s", col=as.integer(iris$Species))
```

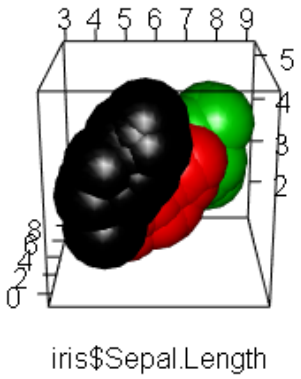


| ¡Sigue trabajando de esa manera y llegarás lejos!

```
  |=======================================================================
======================                |  83%
```

| A diferencia de la función plot() donde si querías cambiar el tamaño del elemento graficado usabas el
| parámetro cex, en plot3d() si deseas cambiar el tamaño de las esferas debes usar el parámetro radius.
| Ingresa plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, type="s",
| col=as.integer(iris$Species), radius=1.5) en la linea de comandos.

```
> plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, type="s", col=as.integer(iris$Species), radius=1.5)
```



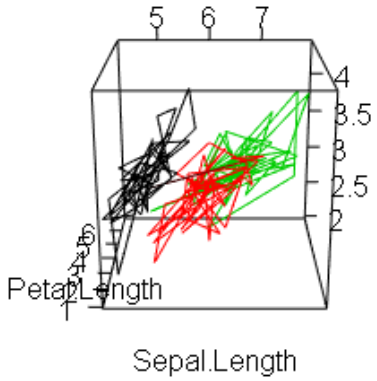
| ¡Tu dedicación es inspiradora!

```
  |=======================================================================
=========================             |  88%
```

| Si graficas líneas o puntos y deseas cambiar su tamaño, el parámetro radius no te servirá. Para cambiar el
| tamaño de las líneas plot3d() usa el parámetro lwd y size para cambiar el de los puntos.

```
...
```

```
  |=================================================================
================================        |  92%
| Al igual que con la función plot(), usando plot3d() puedes cambi
ar el título de los ejes usando xlab, ylab
| y zlab. Ingresa plot3d(iris$Sepal.Length, iris$Petal.Length, iri
s$Sepal.Width, type="l",
| col=as.integer(iris$Species), xlab ="Sepal.Length", ylab = "Peta
l.Length", zlab = "Sepal.Width") en la
| línea de comandos.

> plot3d(iris$Sepal.Length, iris$Petal.Length, iris$Sepal.Width, t
ype="l", col=as.integer(iris$Species), xlab ="Sepal.Length",  ylab
= "Petal.Length", zlab = "Sepal.Width")
```




```
| ¡Bien hecho!

  |=================================================================
================================      |  96%
| Para conocer más opciones de la función plot3d(), ve su página d
e ayuda.
> ?plot3d

| ¡Eres el mejor!

  |=================================================================
==================================| 100%
```

Lección de swirl: 12. Expresiones Regulares

```
Selection: 12
  |
|   0%

| R te provee con algunas funciones que te sirven para manipular c
adenas de caracteres.

...

  |===
|   3%
| Recuerda que una cadena contiene todo texto que se encuentre den
tro de un par de comillas. Guarda la cadena
| "¡Hola Mundo!" en la variable 'texto'.

> texto <- "¡Hola Mundo!"

| ¡Acertaste!

  |=====
|   5%
| Observa que una cadena es un vector de caracteres, pero no se pu
ede acceder directamente a sus elementos
| usando la notación de indexado([]). Ingresa texto[1] en la línea
de comandos para comprobar esto.

> texto[1]
[1] "¡Hola Mundo!"

| Esa es la respuesta que estaba buscando.

  |========
|   8%
| Y ahora ingresa texto[2] en la línea de comandos.

> texto[2]
[1] NA

| ¡Mantén este buen nivel!

  |===========
|  11%
| Esto se debe a que el acceso a los elementos de una cadena se re
aliza con la función substring(). Para
| conocer el uso de substring() abre su página de ayuda.

> ?substring

| ¡Mantén este buen nivel!

  |=============
|  13%
| Ahora que conoces su uso, usando la función substring obtén la s
ubcadena "Mundo" de la variable texto.
```

```
> substring(texto, 7, 11)
[1] "Mundo"

| ¡Tu dedicación es inspiradora!

  |================
|  16%
| Pero no sólo eso; esta función puede también utilizarse en asign
ación, lo cual tiene el efecto colateral de
| cambiar las cadenas originales. Para demostrar esto ingresa subs
tring(texto, 7, 11) <- "Swirl" en la línea
| de comandos.

> substring(texto, 7, 11) <- "Swirl"

| ¡Eso es correcto!

  |==================
|  18%
| Ahora imprime el contenido de la variable 'texto'.

> texto
[1] "¡Hola Swirl!"

| ¡Muy bien!

  |====================
|  21%
| Si deseas conocer la longitud de una cadena, no podrás hacer uso
de la función length(). En cambio deberás
| usar la función nchar(). Ingresa nchar(texto) en la línea de com
andos para conocer la longitud de texto.

> nchar(texto)
[1] 12

| ¡Todo ese trabajo está rindiendo frutos!

  |======================
|  24%
| Anteriormente usaste la función paste(), la cual concatena las c
adenas que le sean pasadas como argumento.
| Ingresa paste("¡Adiós", "Mundo!") en la línea de comandos.

> paste("¡Adiós", "Mundo!")
[1] "¡Adiós Mundo!"

| ¡Tu dedicación es inspiradora!

  |========================
|  26%
| Debes de saber que por omisión estas cadenas son separadas por u
n espacio, pero puedes cambiar el
| comportamiento usando el argumento sep. Ingresa paste("¡Adiós",
"Mundo!", sep="_") en la línea de comandos.

> paste("¡Adiós", "Mundo!", sep="_")
```

```
[1] "¡Adiós_Mundo!"

| ¡Eso es correcto!

  |============================
|  29%
| Sin embargo, a menudo es más conveniente para crear una cadena l
egible usar la función sprintf(), la cual
| tiene sintaxis del lenguaje C.

...

  |================================
|  32%
| Por ejemplo, crea la variable ‘i’ y guarda el número 9 en ella.

> i <-9

| ¡Eres el mejor!

  |==================================
|  34%
| Si quisieras conformar una cadena que te diera la información so
bre ‘i’. Puedes usar la función sprintf() y
| usar la cadena especial "%d" en cada lugar donde quieras hacer u
so de un valor numérico; en este caso ‘i’.
| Ingresa sprintf("El cuadrado de %d es %d", i, i^2) en la línea d
e comandos.

> sprintf("El cuadrado de %d es %d", i, i^2)
[1] "El cuadrado de 9 es 81"

| ¡Eso es correcto!

  |====================================
|  37%
| La función sprintf() ensambla una cadena formateada con los valo
res que vayas pidiendo. Ingresa ?sprintf en
| la línea de comandos para conocer más acerca de ella.

> ?sprintf

| ¡Muy bien!

  |======================================
|  39%
| Contrario a la función paste() tienes la función strsplit(), la
cual subdivide una cadena en cadenas más
| pequeñas, dependiendo de la cadena indicada como separación. Ing
resa strsplit("Me/gusta/programar/en/R",
| "/") en la línea de comandos.

> strsplit("Me/gusta/programar/en/R", "/")
[[1]]
[1] "Me"        "gusta"     "programar" "en"        "R"
```

| ¡Traes una muy buena racha!

  |===
| 42%
| Anteriormente usaste la función chartr() para sustituir caracteres en cadenas. Ingresa chartr("o", "a",
| "¡Hola Mundo!") en la línea de comandos.

```
> chartr("o", "a", "¡Hola Mundo!")
[1] "¡Hala Munda!"
```

| ¡Muy bien!

  |==
| 45%
| Muchas veces necesitarás reemplazar un texto con otro y chartr() funcionará perfectamente.

...

  |==
| 47%
| Pero muchas otras lo que querrás hacer será más complejo, porque puede que en vez de ser un simple texto
| querrás reemplazar todas las palabras que terminen con a.

...

  |==
| 50%
| En estos casos puedes utilizar expresiones regulares (que suelen llamarse "regex" o "regexp" de forma
| abreviada).

...

  |==
| 53%
| Las expresiones regulares o patrones no son más que una especie de comodín o un atajo para referirse a una
| gran cantidad de cadenas.

...

  |==
| 55%
| A diferencia de otros lenguajes, en los que las expresiones regulares se encierran entre algún tipo
| especial de delimitadores, en R una expresión regular se representa como una cadena de texto.

...

  |==
| 58%
| Por ejemplo, el grupo formado por las cadenas "Handel", "Händel" y "Haendel" se describe mediante el patrón

```
| "H(a|ä|ae)ndel".

...

  |=======================================================
|  61%
| Habitualmente las expresiones regulares se pasan como argumentos
de una función, que utiliza el patrón
| representado por ellas para realizar alguna tarea como búsqueda
o sustitución.

...

  |=======================================================
|  63%
| Para ejemplificar esto guarda la cadena "H(a|ä|ae)ndel" en la va
riable ‘patron’.

> patron <- "H(a|ä|ae)ndel"

| ¡Lo estás haciendo muy bien!

  |=======================================================
===                                     |  66%
| Una de las funciones que trabaja con expresiones regulares es gr
ep(). Esta función toma como argumentos
| primero un patrón y como segundo argumento un vector de cadenas
y grep() te regresa un vector numérico, el
| cual contiene los índices de las cadenas que contienen ese patró
n.

...

  |=======================================================
=====                                   |  68%
| Para probar grep() he creado un vector de cadenas y lo guarde en
la variable ‘musicos’. Revisa el contenido
| de ‘musicos’.

> musicos
[1] "Handel"    "Mendel"    "Haendel"   "HÃ¤ndel"   "Handemore" "h
andel"

| ¡Es asombroso!

  |=======================================================
========                                |  71%
| Ahora ingresa grep(patron, musicos) en la línea de comandos.

> grep(patron, musicos)
[1] 1 3

| ¡Lo estás haciendo muy bien!

  |=======================================================
===========                             |  74%
```

```
| Efectivamente grep() te regresó las posiciones de las cadenas qu
e contienen el patrón "H(a|ä|ae)ndel"; es
| decir, las posiciones de las cadenas "Handel", "Händel" , "Haend
el".

...

  |=======================================================================
=============                                |  76%
| Es importante que sepas que el patrón es sensible a mayúsculas;
esto explica por qué no encontró
| coincidencia en la cadena "handel".

...

  |=======================================================================
================                             |  79%
| Como ya te habrás imaginado el carácter "|" dentro de una expres
ión regular no representa más que un OR; es
| decir, el patrón te indica a las cadenas: "Handel" o "Händel" o
"Haendel".

...

  |=======================================================================
===================                          |  82%
| Si deseas construir un patrón que además incluya las cadenas "Me
ndel" y "handel", puedes hacerlo de la
| siguiente manera ".(a|ä|ae|e)ndel". Ingresa nuevo_patron <- ".(a
|ä|ae|e)ndel" en la línea de comandos.

> nuevo_patron <- ".(a|ä|ae|e)ndel"

| ¡Acertaste!

  |=======================================================================
======================                       |  84%
| Donde el "." indica un carácter simple, por lo que cuando lo uti
lizas dentro de la expresión regular estás
| diciendo que no te importa qué carácter está ahí, cualquier cará
cter cazará con la expresión regular.
| Ingresa grep(nuevo_patron, musicos) para comprobar esto.

> grep(nuevo_patron, musicos)
[1] 1 2 3 6

| ¡Excelente trabajo!

  |=======================================================================
========================                     |  87%
| Otra función que trabaja con expresiones regulares es regexpr(),
que al igual que grep() recibe como primer
| argumento un patrón y como segundo un vector de cadenas. A difer
encia de grep(), regexpr() regresa el
| índice en donde encuentra la primera aparición del patrón que es
tás buscando. Prueba la función; ingresa
| regexpr(patron, musicos) en la línea de comandos.
```

```
> regexpr(patron, musicos)
[1]  1 -1  1 -1 -1 -1
attr(,"match.length")
[1]  6 -1  7 -1 -1 -1

| ¡Buen trabajo!

  |=======================================================================
=========================           |  89%
| Además de regresar el índice en donde encuentra la primera apari
ción del patrón, también te regresa la
| longitud del patrón encontrado. Si no encuentra el patrón notará
s que te regresa como índice y longitud un
| -1.

...

  |=======================================================================
============================        |  92%
| Si deseas encontrar todas las posiciones donde es encontrado el
patrón y no sólo la primera, puedes usar
| gregexpr(), ya que funciona de la misma manera que regexpr(), só
lo que te regresa todos los índices donde
| encuentra el patrón. Por ejemplo, ingresa gregexpr(patron, "Geor
g Friedrich Händel, en inglés George
| Frideric Handel fue un compositor alemán.") en la línea de coman
dos.

> gregexpr(patron, "Georg Friedrich Händel, en inglés George Fride
ric Handel fue un compositor alemán.")
[[1]]
[1] 17 51
attr(,"match.length")
[1] 6 6


| ¡Eres bastante bueno!

  |=======================================================================
==============================      |  95%
| Las expresiones regulares pueden ser muy ricas e incluso muy com
plicadas. El objetivo del curso no define
| que las veas a profundidad, pero puedes ver cómo las utiliza R u
sando ?regexpr. Inténtalo.

> ?regexpr

| ¡Excelente!

  |=======================================================================
================================    |  97%
| Aquí puedes ver todas las expresiones que te sirven para utiliza
r expresiones regulares dentro del lenguaje
| con las funciones que acabas de ver.

...
```

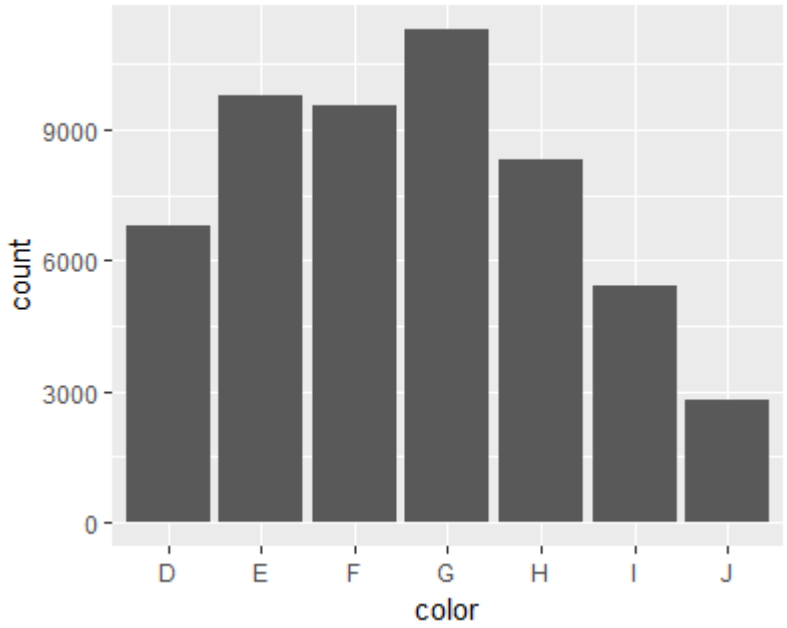
```
  |======================================================================
==================================|  100%
```

Lección de swirl: 13. Graficación con ggplot2

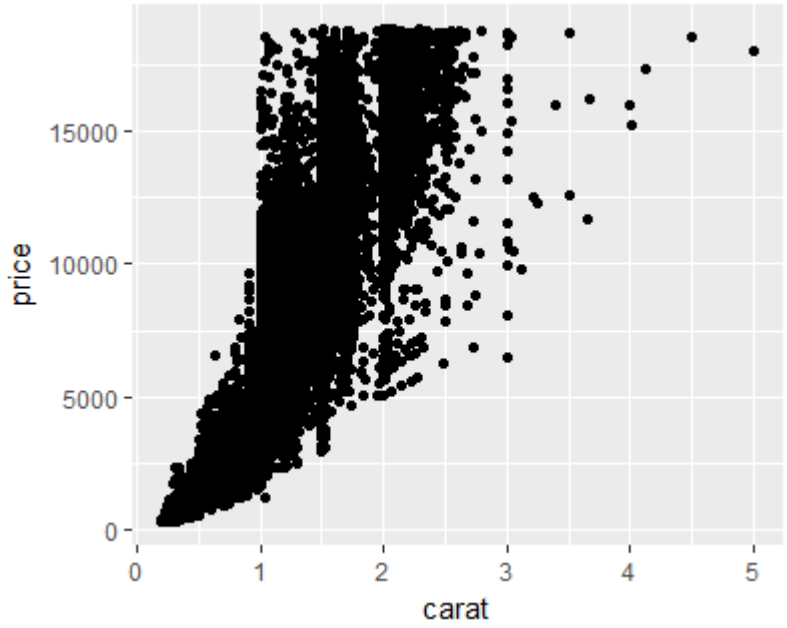
```
Selection: 13

| Intentando cargar las dependencias de la lección...

| ¡El paquete 'ggplot2' se ha cargado correctamente!

  |
|   0%

| ggplot2 es un sistema de graficación muy poderoso en R que te pr
ovee con gráficos de alto nivel.

...

  |===
|   3%
| Este modelo de gráficos hace fácil la producción de gráficos com
plejos.

...

  |======
|   6%
| No tendrás que preocuparte por las leyendas, los tipos de fuente
s y demás detalles a diferencia de usar el
| paquete base, ya que ggplot2 trae valores por omisión.

...

  |=========
|   9%
| Ya que estos valores por omisión fueron cuidadosamente escogidos
, lo que significa que la mayor parte del
| tiempo tú puedes producir gráficas de tan buena calidad que pued
en utilizarse inmediatamente en
| publicaciones.

...

  |===========
|  11%
| En esta práctica trabajarás con el conjunto de datos diamonds de
l paquete ggplot2, el cual contiene las
| medidas, información de calidad y precios de casi 54 000 diamant
es.

...

  |==============
|  14%
| Para cargar el conjunto de datos diamonds ingresa data(diamonds)
en la línea de comandos.

> data(diamonds)
```

```
| Perseverancia es la respuesta.

  |================
| 17%
| Ahora ve lo que contiene diamonds. Para hacer esto usarás la fun
ción View(). Si te encuentras en Rstudio
| simplemente puedes presionar el nombre de tu variable diamonds e
n el apartado Entorno ("Environment") y se
| mostrará su contenido. Presiona la variable diamonds en Rstudio
o ingresa en la línea de comando:
| View(diamonds).

> View(diamonds)

| ¡Buen trabajo!

  |===================
| 20%
| carat representa el peso del diamante, cut representa la calidad
del corte (Fair, Good, Very Good, Premium,
| Ideal), color representa el color del diamante, desde J (el peor
) a D (el mejor), clarity es una medida de
| qué tan claro es el diamante (I1 (el peor), SI1, SI2, VS1, VS2,
VVS1, VVS2, IF (el mejor)).

...

  |=====================
| 23%
| table indica la anchura de la parte superior del diamante con re
lación al punto más ancho, price representa
| el precio del diamante en dólares, x la longitud en milímetros,
y la anchura en milímetros, z la
| profundidad en milímetros y depth el porcentaje de profundidad t
otal, es decir, z / mean(x, y) = 2 * z / (x
| + y).

...

  |=========================
| 26%
| Dominar el paquete ggplot2 puede ser un reto, por lo que en este
curso sólo verás la función qplot() (quick
| plot), la cual es una función auxiliar, ya que esconde mucha de
su complejidad cuando crea gráficos
| estándares.

...

  |============================
| 29%
| La función qplot() puede ser usada para crear los tipos de gráfi
cas más comunes. Mientras que no expone las
| grandes capacidades de ggplot2, puede crear un gran rango de grá
ficas útiles.

...
```

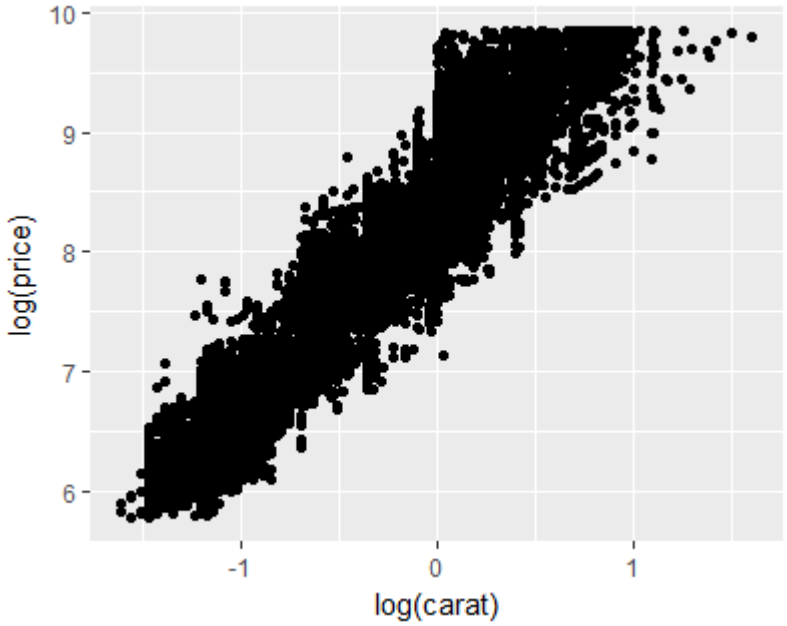
```
  |================================
|   31%
| Ingresa ?qplot para conocer más acerca de ella.

> ?qplot

| ¡Tu dedicación es inspiradora!

  |=================================
|   34%
| Puedes hacer desde histogramas. Ingresa qplot(color, data=diamon
ds) para crear un histograma de los
| colores.

> qplot(color, data=diamonds)
```



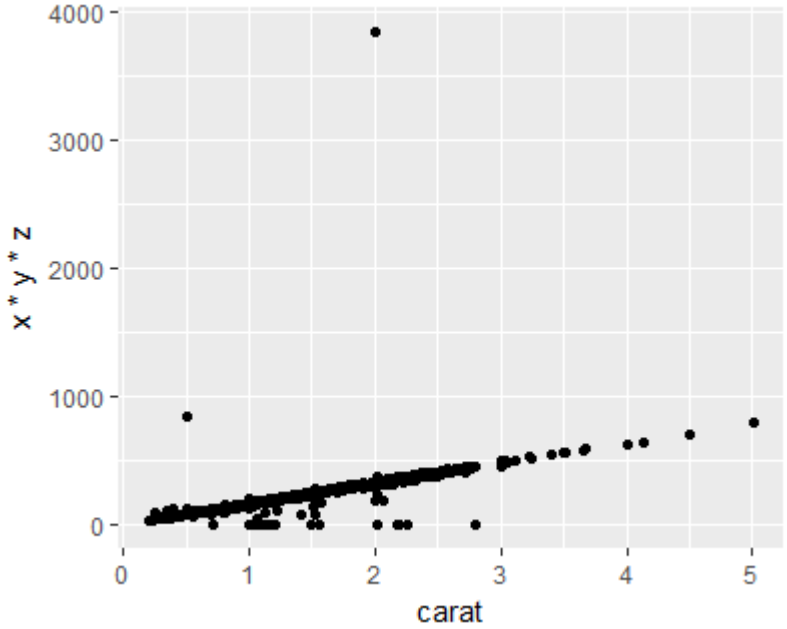

```
| ¡Tu dedicación es inspiradora!

  |=================================
|   37%
| Hasta gráficas de dispersión. Ingresa qplot(carat, price, data=d
iamonds) en la línea de comandos.
> qplot(carat, price, data=diamonds)
```




```
| ¡Mantén este buen nivel!

  |=======================================
|   40%
| Puedes ver que hay mucha saturación por la gran cantidad de dato
s que hay al usar el dataset completo.

...

  |========================================
|   43%
| Existe cierta correlación para ser exponencial y hay ciertos lug
ares donde parecen hacerse estrías. Esto
| puedes tratar de compensarlo haciendo una transformación de las
variables; es decir, en lugar de usar los
| valores lineales, podrías aplicar una transformación logarítmica
y entonces estarías graficando los valores
| logarítmicos contra los valores logarítmicos de 'x' y 'y'. Ingre
sa qplot(log(carat), log(price),
| data=diamonds) en la línea de comandos para lograr esto.
> qplot(log(carat), log(price), data=diamonds)
```

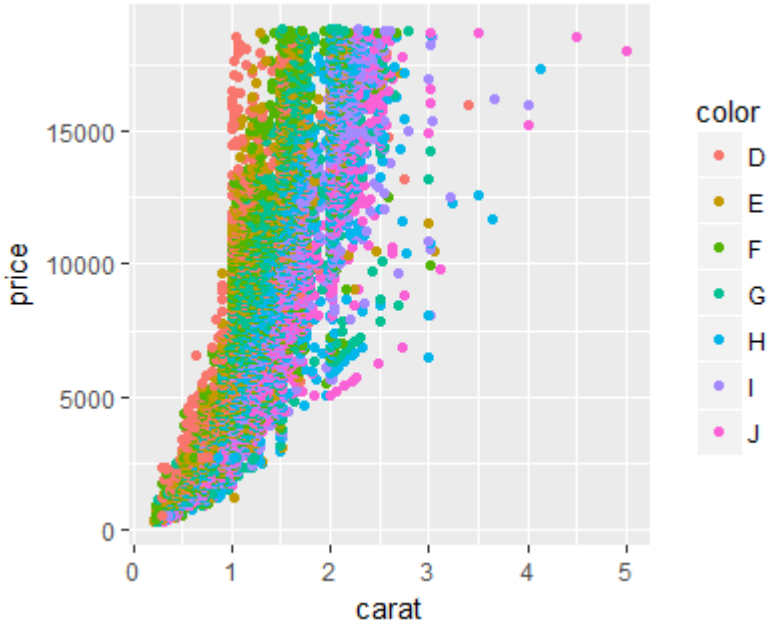



```
| ¡Eso es trabajo bien hecho!

  |=============================================
|  46%
| También puedes utilizar los argumentos como combinaciones de otr
os. Ingresa qplot(carat, x*y*z,
| data=diamonds) en la línea de comandos.
> qplot(carat, x*y*z, data=diamonds)
```

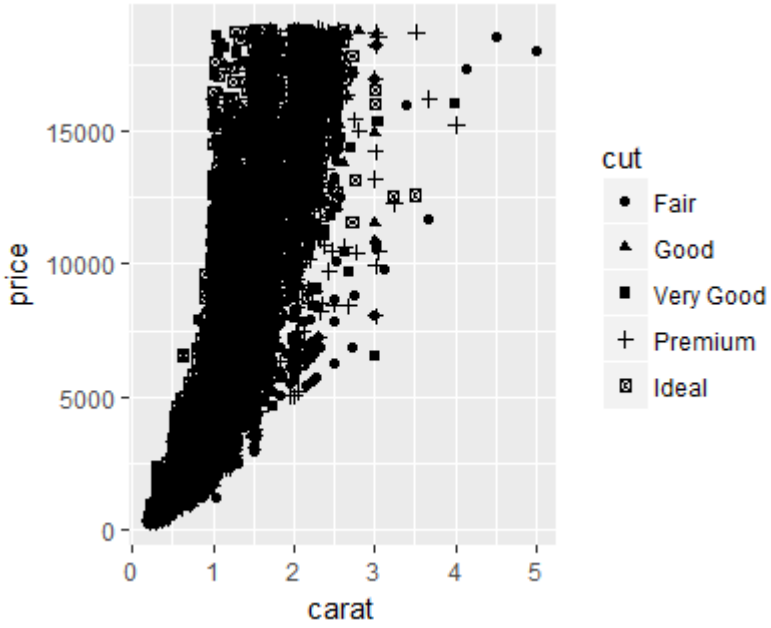



```
| ¡Mantén este buen nivel!

  |=============================================
|  49%
| En ggplot2() puedes mapear alguna categoría que tenga una variab
le a un atributo estético, en este caso el
```

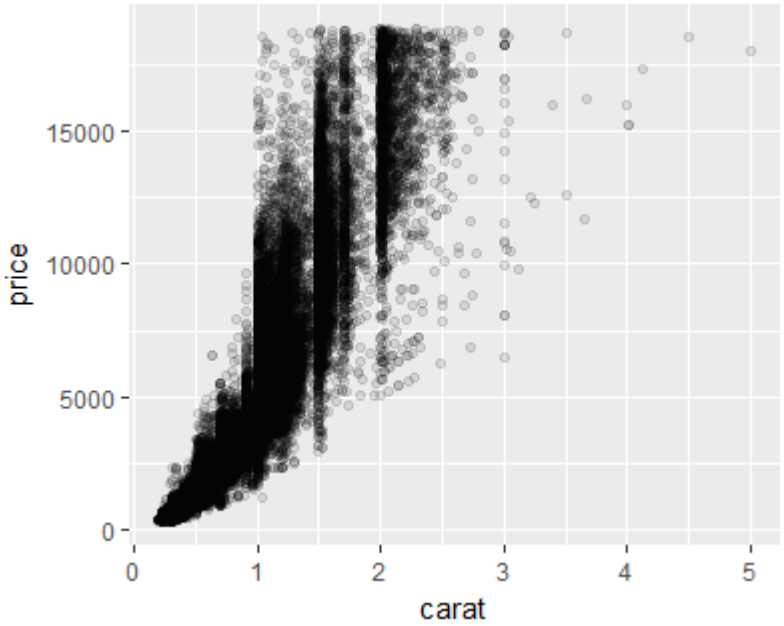
```
| atributo color de qplot(); por ejemplo, si utilizas diamonds pue
des mapear la columna color que te indica
| el color del diamante a un color distinto en la gráfica. Ingresa
qplot(carat, price, data=diamonds,
| color=color) en la línea de comandos.
> qplot(carat, price, data=diamonds, color=color)
```




```
| ¡Tu dedicación es inspiradora!

  |=======================================================
|  51%
| Además del color otro atributo estético que tiene qplot() es sha
pe; shape se refiere a la forma con la que
| va a pintar los puntos qplot(); en la gráfica por ejemplo shape
también es muy bueno para poder mapear
| características categóricas a una característica gráfica.

...

  |=======================================================
|  54%
| Puedes mapear cada uno de los cortes distintos de los diamantes
que tienes a una figura distinta. Ingresa
| qplot(carat, price, data=diamonds, shape=cut) en la línea de com
andos.
> qplot(carat, price, data=diamonds, shape=cut)
```




```
| ¡Eres el mejor!

  |=======================================================
|   57%
| El atributo alpha de qplot() indica cuántos puntos se necesitan
pintar o aparecer en un mismo lugar para
| que puedas pintar un punto completamente opaco. Ingresa qplot(ca
rat, price, data=diamonds, alpha=I(1/10))
| en la línea de comandos.
> qplot(carat, price, data=diamonds, alpha=I(1/10))
```

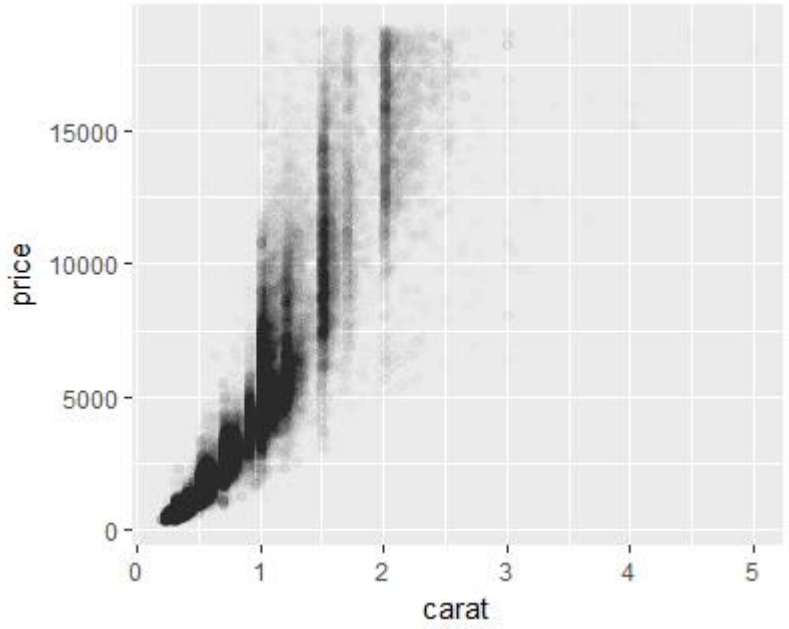



```
| ¡Lo estás haciendo muy bien!

  |=======================================================
|   60%
```

| Con esto necesitarías 10 puntos en un mismo lugar para que se pi
nte un primer punto.

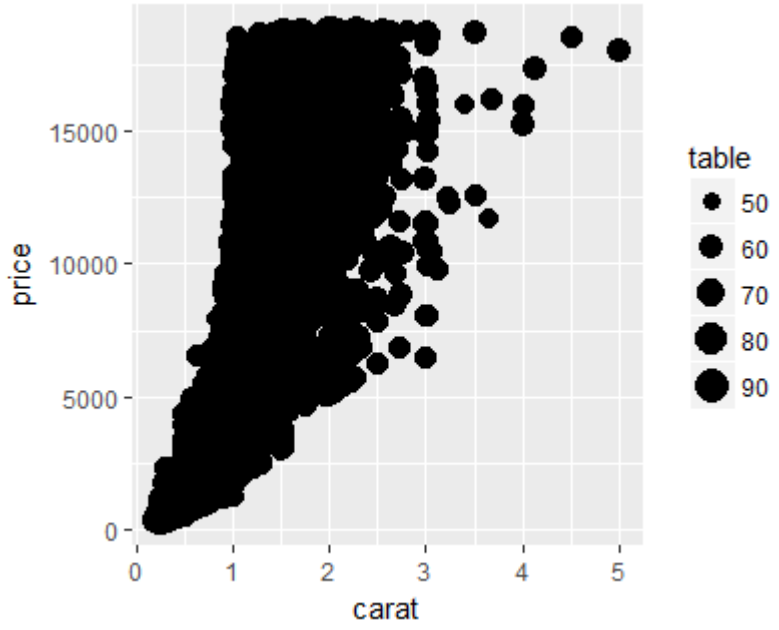
...

  |===
| 63%
| Nota cómo si cambias el alpha de un 1/10 a un 1/100 puedes encon
trar nuevos patrones que no eran claros
| cuando usabas un alpha más grande. ¡Inténtalo!
> qplot(carat, price, data=diamonds, alpha=I(1/100))

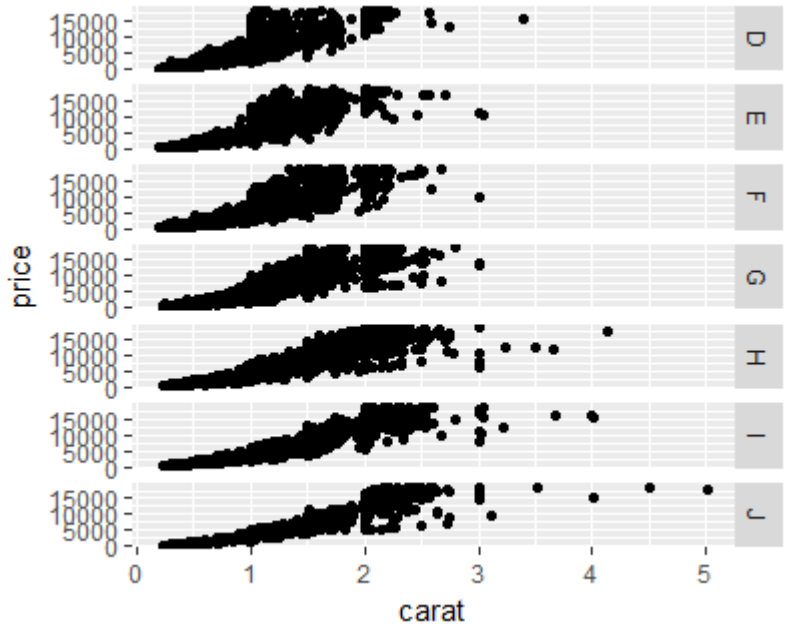


| ¡Buen trabajo!

  |===
=== | 66%
| size, es un atributo que indica el tamaño, por lo que si usas si
ze=table te pintará los puntos de diferente
| tamaño. Ingresa qplot(carat, price, data=diamonds, size=table) e
n la línea de comandos.
> qplot(carat, price, data=diamonds, size=table)




```
| ¡Tu dedicación es inspiradora!

  |======================================================================
======                                               |  69%
| Debes de saber que algunas variables se mapean mejor a ciertos a
tributos estéticos; por ejemplo, color y
| shape se mapean mejor a variables categóricas, mientras que size
se mapea mejor a variables de tipo
| continuo.

...

  |======================================================================
========                                             |  71%
| A veces la cantidad de datos puede hacer una gran diferencia y y
a viste que puede usar alpha para cambiar
| la saturación mayor o menor dependiendo del número de puntos que
tengas. Si esto no te ayuda entonces
| puedes tratar de subdividir los grupos de las variables que se f
orman en una sola variable con una técnica
| especial conocida como faceting.

...

  |======================================================================
===========                                          |  74%
| El atributo facets te permite crear una tabla donde puedes subdi
vidir en grupos. Si quisieras ver cómo se
| comporta la variable carat contra la variable price y le pides q
ue te haga el faceting por color, entonces
| por cada color de diamante te va graficando el carat contra el p
rice (peso vs precio). Ingresa qplot(carat,
| price, data=diamonds, facets = color ~ .) en la línea de comando
s.
> qplot(carat, price, data=diamonds, facets = color ~ .)
```



| ¡Mantén este buen nivel!

  |==
============== | 77%
| Entonces puedes observar cómo se va comportando para cada uno de los colores de diamante. Esto te permite
| la comparación visual inmediata.

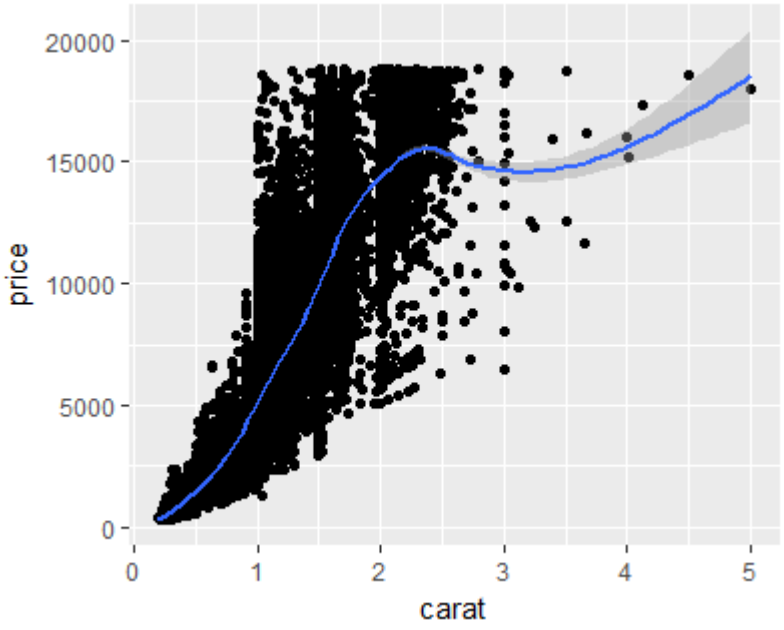
...

  |==
================= | 80%
| Por último, el atributo geom de qplot() especifica el tipo de objeto que utilizará para graficar. El valor
| por defecto es "point" (punto); esto sólo si 'x' y 'y' se encuentran especificados. Si sólo x se encuentra
| especificado el valor es "histogram" (histograma).

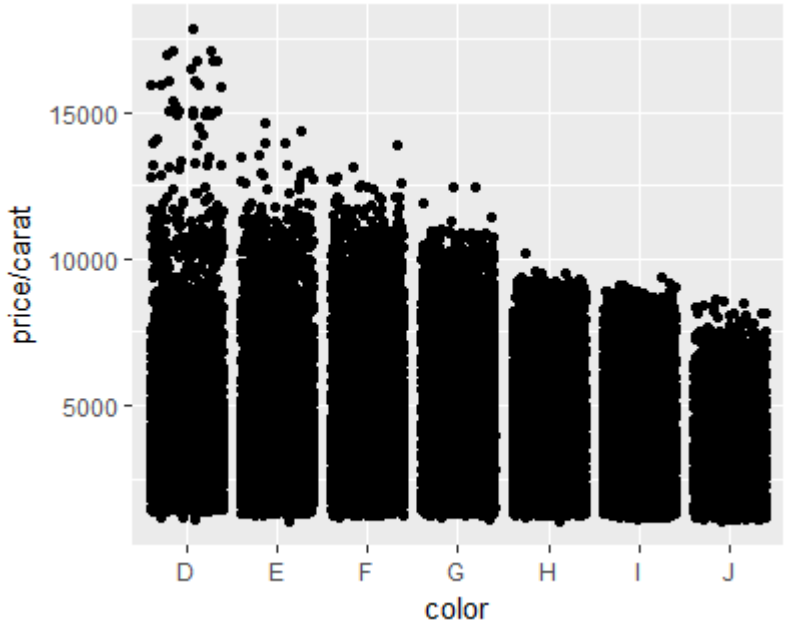
...

  |==
==================== | 83%
| Un posible valor para geom es "smooth", el cual ajusta una curva a los puntos que estas especificando,
| además de que te dibuja el error estándar de esa curva. Ingresa qplot(carat, price, data=diamonds, geom =
| c("point", "smooth")) en la línea de comandos.

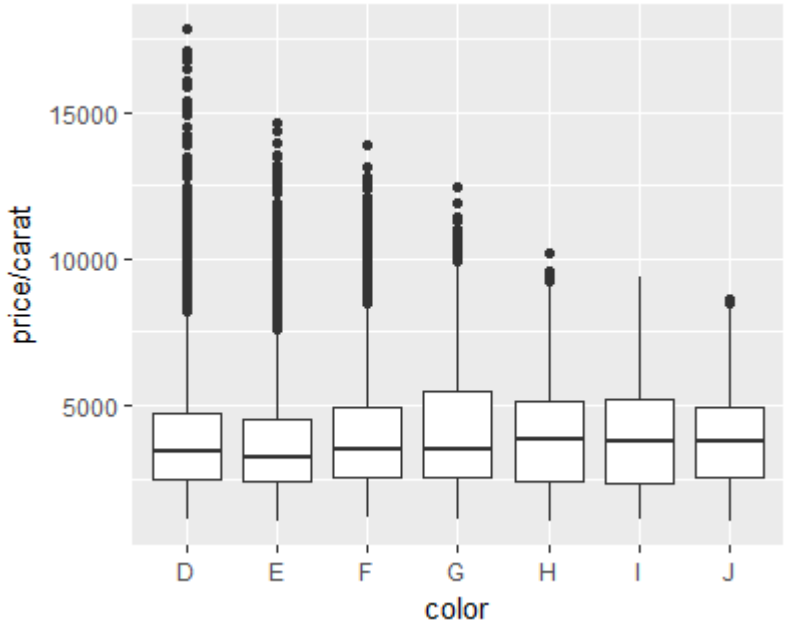
```
> qplot(carat, price, data=diamonds, geom = c("point","smooth"))
```




```
`geom_smooth()` using method = 'gam'
`geom_smooth()` using method = 'gam'

| ¡Buen trabajo!

  |======================================================================
=======================              |  86%
| Una característica de "smooth" es que puedes especificar el tipo
de ajuste que quieres para la curva.

...

  |======================================================================
=========================            |  89%
| Otro valor posible es "jitter", el cual permite analizar en caja
s cómo se están dispersando los puntos,
| dependiendo de las variables que quieras observar. Ingresa qplot
(color, price/carat, data=diamonds, geom =
| "jitter") en la línea de comandos.
> qplot(color, price/carat, data=diamonds, geom = "jitter")
```




```
| ¡Mantén este buen nivel!

  |======================================================================
============================          |  91%
| Si usas geom="boxplot" te graficará una gráfica de caja, donde p
uedes observar la distribución de tus datos
| por la categoría que quieras observar. Ingresa qplot(color, pric
e/carat, data=diamonds, geom = "boxplot")
| en la línea de comandos.
> qplot(color, price/carat, data=diamonds, geom = "boxplot")
```




```
| ¡Eres el mejor!

  |======================================================================
==============================        |  94%
```

```
| Otros dos valores importantes que puede tomar geom son "line" y
"path".

...

  |=================================================================
==================================    |  97%
| Actualmente, ggplot2 no puede ser usado para crear gráficas 3D.

...

  |=================================================================
==================================| 100%
```

Lección de swirl: 14. Simulación

```
Selection: 14
  |
|   0%

| Esta lección asume que tienes familiaridad con algunas distribuc
iones de probabilidad. Aun si no tienes
| ninguna experiencia con estos conceptos, serás capaz de completa
r esta lección y entender las ideas
| generales.

...

  |====
|   4%
| R contiene por defecto muchas distribuciones de probabilidad. Po
r ejemplo, si ingresas ?Normal, obtendrás
| información acerca de la distribución normal.

> ?Normal

| ¡Eres el mejor!

  |=======
|   7%
| Usualmente cada distribución tendrá 4 funciones de la forma pref
ijo + apodo_distribucion. Cada prefijo
| indica: r - la función para generar números aleatorios, p - la f
unción de distribución, d - la función de
| densidad, q - la función que te da cuantiles.

...

  |===========
|  11%
| A todas estas funciones les puedes especificar la media y la var
ianza de la distribución.

...

  |==============
|  14%
| En general, la primera letra de la función determina qué informa
ción obtendrás acerca de la distribución y
| luego el apodo_distribución sigue directamente.

...

  |==================
|  18%
| Por ejemplo, si quisieras la función de densidad de la distribuc
ión Poisson usas dpois(). A continuación se
| muestran algunas distribuciones útiles.

(Se ha copiado el archivo distributions.txt a la ruta C:/Users/Sou
rmak/Documents/swirl_temp/distributions.txt ).
```

...

  |=====================
| 21%
| Para obtener una lista completa de las distribuciones disponible
s en R, ingresa help(Distributions) en la
| línea de comandos.

> help(Distributions)

| ¡Acertaste!

  |=========================
| 25%
| Es importante que revises que las funciones definan la distribuc
ión de la manera que tú esperas.

...

  |============================
| 29%
| Ahora verás cómo estas funciones te pueden ayudar para resolver
un problema de simulación.

...

  |==============================
| 32%
| Supón que quisieras encontrar la probabilidad de tener al menos
5 caras al lanzar 7 veces una moneda justa.

...

  |===================================
| 36%
| Recuerda que el lanzamiento de una moneda puede ser modelado por
una variable aleatoria bernoulli.

...

  |=======================================
| 39%
| Es decir, donde p es la probabilidad de éxito; en este caso que
caiga una cara, por lo que 1-p es la
| probabilidad de fracaso, es decir, que no caiga cara.

...

  |==
| 43%
| Y ahora puedes usar una variable aleatoria binomial que te dice
el número de éxitos en n repeticiones de
| experimentos bernoulli. Esta variable aleatoria la puedes modela
r con la función rbinom(). No olvides que
| si n=1 el experimento que estarás observando es el lanzamiento d
e una moneda (el caso más simple).

...

  |===
| 46%
| En este caso puedes usar n=7 que son los 7 lanzamientos de la mo
neda.

...

  |==
| 50%
| Ahora vas a generar esos números aleatorios usando la función rb
inom(). Antes que nada ingresa ?rbinom en
| la línea de comandos para conocer su uso.

> ?rbinom

| ¡Tu dedicación es inspiradora!

  |==
| 54%
| Como te podrás dar cuenta rbinom() se encuentra definida así: rb
inom(n, size, prob), donde n indica el
| número de veces que se va a llevar a cabo el experimento. size d
ice el número de intentos; en este caso el
| número de lanzamientos que se van a llevar a cabo. Y prob indica
la probabilidad de éxito en cada intento;
| en este caso que caiga cara.

...

  |==
| 57%
| Si quisieras llevar a cabo este experimento una única vez, es de
cir, lanzar 7 veces una moneda con
| probabilidad 1/2 de éxito (es decir, que la moneda sea justa), d
eberás llamar a rbinom() de la siguiente
| manera: ingresa rbinom(1, 7, 0.5) en la línea de comandos.

> rbinom(1, 7, 0.5)
[1] 2

| ¡Buen trabajo!

  |==
| 61%
| El resultado que obtuviste sólo te indica el número de éxitos qu
e obtuviste en un único experimento, es
| decir, el número de caras que salieron.

...

  |==
= | 64%
| Si quisieras sacar la proporción de éxitos que tendrías, habría
que llevar a cabo este experimento una gran
| cantidad de veces.

```
...

  |=========================================================================
=====                                    |  68%
| Con R lo puedes justificar de manera intuitiva, aplicando este e
xperimento muchas veces. En este caso
| repite el experimento 100 000 veces y guárdalo en la variable 'r
esultado'.

>
> resultado <- rbinom(100000, 7, 0.5)

| ¡Eres el mejor!

  |=========================================================================
========                                 |  71%
| Como ya te podrás imaginar, 'resultado' contiene el número de ca
ras que salieron por cada experimento; en
| este caso 100 000. Ingresa tail(resultado) para conocer el resul
tado de los últimos 6 experimentos.

> tail(resultado)
[1] 5 5 2 5 5 4

| ¡Eres bastante bueno!

  |=========================================================================
============                             |  75%
| Una vez teniendo los resultados de aplicar este experimento 100
000 veces, puedes encontrar cuántos de esos
| experimentos tuvieron 5 veces o más éxito. Ingresa tail(resultad
o > 5) en la línea de comandos.

> tail(resultado > 5)
[1] FALSE FALSE FALSE FALSE FALSE FALSE

| ¡Muy bien!

  |=========================================================================
================                         |  79%
| Teniendo este vector ahora puedes encontrar la proporción de ver
daderos contra falsos sacando la media.
| Ingresa mean(resultado > 5) en la línea de comandos.

>
> mean(resultado > 5)
[1] 0.06281

| ¡Buen trabajo!

  |=========================================================================
====================                     |  82%
| Con esto has encontrado la probabilidad de tener al menos 5 cara
s al lanzar 7 veces una moneda justa.

...
```

```
  |======================================================================
======================                |  86%
| Te preguntarás cómo sacaste la media de un vector de valores lóg
icos; recuerda que esto es posible debido a
| la coerción.

...

  |======================================================================
=========================             |  89%
| Por último, es posible que desees que tus resultados sean replic
ables. La función set.seed() te permite
| establecer el punto de inicio en la generación de números aleato
rios. Ingresa ?set.seed en la línea de
| comandos.

> ?set.seed

| ¡Buen trabajo!

  |======================================================================
============================          |  93%
| Como verás, si estableces la misma semilla antes de generar núme
ros Aleatorios, siempre obtendrás los
| mismos números aleatorios.

...

  |======================================================================
================================      |  96%
| Ten en cuenta que no siempre es lo que puedas necesitar.

...

  |======================================================================
====================================| 100%
```