



# K SCHOOL

## **MÁSTER DATA SCIENCE**

### **Introducción a R**

Pedro Poveda

¡Recuerda!

El material que verás en esta clase se ha elaborado específicamente para KSchool. En él han participado directores, profesores y personal de KSchool.

**La difusión parcial o total del contenido de esta presentación, así como de cualquier otro material de apoyo audiovisual de la clase está totalmente prohibido.**

KSchool se reserva el derecho a la propiedad intelectual de las clases grabadas, por lo que su utilización fuera del contexto académico de KSchool queda prohibida y se tomarán medidas legales ante su difusión pública, que supone un delito de infracción del copyright.

Además, queremos recordaros que, para hacer las clases mucho más interesantes, muchos de nuestros profesores utilizan datos de empresas reales. Estos datos son estrictamente confidenciales y pertenecen a la empresa. Se sancionará la utilización de estos fuera del contexto académico.

Y ahora, ¡a disfrutar de la clase!

# Sobre mí

## Pedro Poveda

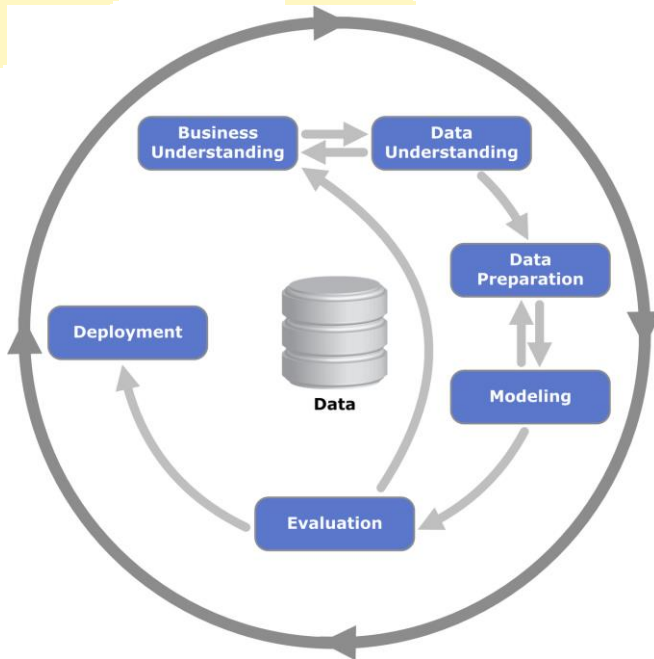
- *Manager Data & Analytics* en Amadeus
- Ing. Técnico Informática de Gestión en Universidad Politécnica de Madrid
- 15 años de experiencia en BI, Data & Analytics
- Profesor del Máster *Data Science* en **KSchool**





# *Data hacking* con R

## Objetivos



- ✓ Conocer el lenguaje R y sus posibilidades para *Data Science*
- ✓ Ser capaces de explorar conjuntos de datos, visualizarlos y aplicar transformaciones necesarias sobre datasets de R
- ✓ Conocer las principales técnicas de aprendizaje automático, tanto supervisado como no supervisado, usando librerías de R
- ✓ Introducción a las series temporales y a técnicas de predicción con R

# Agenda

- Introducción a R y entorno de trabajo
- Variables y tipos de datos en R
- Estructuras de datos principales
- Estructuras de control
- Funciones y paquetes
- Utilidades y recursos adicionales



# Introducción a R



# ¿Qué es R?



- R es un lenguaje de programación libre GNU y multi-paradigma, enfocado al análisis estadístico
- Fue creado en 1993 como una evolución del lenguaje S
- Es compatible con la mayoría de sistemas operativos (Windows, MacOS, Linux)
- Su desarrollo es responsabilidad del [R Core Team](#), y cuenta con +18k librerías (paquetes) para todo tipo de usos y aplicaciones

# Instalación de R



- Descargar la última versión del intérprete de R en <https://cloud.r-project.org/> y seguir instrucciones según sistema operativo
  - En *Windows*, es recomendable instalar en directorio C:\R
  - En *MacOS*, es recomendable instalar mediante [Homebrew](#)
- Una vez instalado R, procederemos a instalar el IDE **RStudio Desktop**, versión *Free*, en <https://www.rstudio.com/products/rstudio/download/>



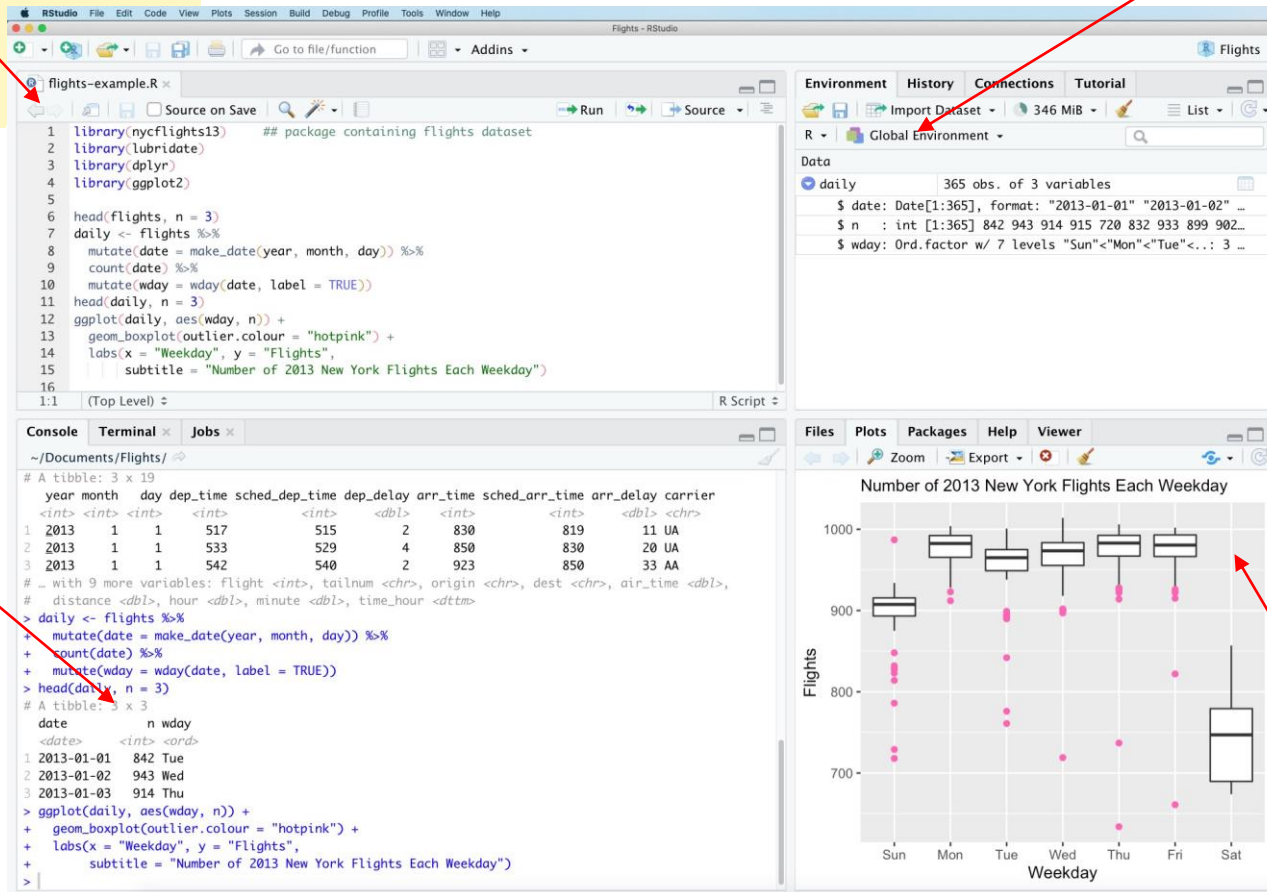


- Entorno de desarrollo integrado (IDE) para trabajar con R
- Es compatible con Windows, MacOS y Linux, y tiene versiones gratuitas y profesionales en sus versiones desktop y server
- Son los responsables de librerías de R muy extendidas como *tidyverse* o *ggplot2*

# RStudio

Scripts

Entorno (variables,  
conexiones, ...)



Consola

Output  
(gráficos,  
ayuda, ...)

# Entorno de trabajo en R







## Directorio de trabajo

- El directorio de trabajo (*working directory*) es el directorio de referencia para el código que ejecutamos en R
- Podemos consultar su valor mediante la función **`getwd()`**, y modificarlo mediante **`setwd("...")`**
- El separador de directorios será / incluso en Windows

# Entorno de trabajo en R

## Objetos de entorno

- Podemos listar los objetos del entorno de trabajo mediante la función *ls()*
- Para limpiar el entorno, ejecutaremos la siguiente sentencia: *rm(list=ls())*

Environment	History	Connections	Tutorial	
<div><div>   Import Dataset ▾</div><div> 95 MiB ▾</div><div></div></div>				
R ▾	 Global Environment ▾			
Data				
m	int [1:3, 1:3] 1 4 7 2 5 8 3 6 9			
Values				
d	7L			

# Entorno de trabajo en R

## Instalación de paquetes

- Una de las características principales de R es la cantidad de paquetes o librerías disponibles para todo tipo de propósito: procesamiento de datos, *machine learning*, ...
- Para instalar paquetes usaremos la función ***install.packages("package")***
- Para usar las funciones de un paquete previamente instalado, debemos cargar la librería mediante la sentencia ***library("package")***

# Variables y tipos de datos



# Asignación de variables, comentarios e impresión en pantalla

- En R podemos asignar valores a variables sin necesidad de declararlas mediante los operadores `<-` o `=`, siendo el primero el más utilizado
  - `v <- 5`
  - `v = 5`
- Los comentarios irán precedidos del carácter `#`
- Para imprimir un valor en pantalla, simplemente escribiremos la variable y ejecutaremos la sentencia

# Operadores aritméticos principales

- Suma  $+$
- Resta  $-$
- Multiplicación  $*$
- División  $/$
- Potencia  $^$
- Módulo  $%%$



# Operadores lógicos principales

- Menor que <
- Mayor que >
- Igual a ==
- Distinto de !=
- Y booleano &
- O booleano |
- No !

# Tipos de datos básicos

- **Character:** texto “c”, “hola”
- **Numeric:** números 4, 4.5
- **Logical:** booleanos TRUE, FALSE
- Podemos consultar el tipo de una variable mediante ***typeof()***
- En R toda variable es a su vez un objeto, del cual podemos consultar su clase con ***class()*** y sus atributos con ***attributes()***
- Existen objetos de clases más complejas, como veremos posteriormente

# *NA, NULL* y NaN

- Constantes que sirven para representar valores nulos, vacíos o inválidos
- **NA** se utiliza para representar valores no disponibles, inexistentes
- **NULL** es usado para representar objeto vacío
- **NaN** significa *not a number*
- Podemos comprobar si una variable corresponde a alguna de estas constantes mediante las funciones *is.na()*, *is.nul()* o *is.nan()*

```
> v <- c(1, 2, 3, NA, 4, NULL)
> v
[1] 1 2 3 NA 4
> sum(v)
[1] NA
> is.na(v)
[1] FALSE FALSE FALSE TRUE FALSE
> nulos <- is.na(v)
> sum(v[!nulos])
[1] 10
> nan <- 0 / 0
> nan
[1] NaN
> is.nan(nan)
[1] TRUE
> class(nan)
[1] "numeric"
> class(NULL)
[1] "NULL"
> class(NA)
[1] "logical"
>
```

# Estructuras de datos principales



# Vectores

- Conjunto de elementos unidimensionales ordenados de la misma clase (*arrays*)
- Se crean mediante la función `c()`
- Podemos acceder al elemento `n` (o al conjunto de elementos) indexando el vector con `[]`
- Mediante el operador `:` podemos crear una secuencia de enteros, tanto para crear el vector como para indexar sus elementos

`c(9, 5, 20, 3, 1, 100)`



9	5	20	3	1	100
---	---	----	---	---	-----

# Operaciones con vectores

- Podemos asignar valores a los índices del vector mediante la función **names()**, y usarlos posteriormente para indexar
- Las funciones de operaciones básicas como **sum()** o **mean()** aplicadas a un vector nos devuelve el resultado de la función aplicado a todos los elementos del vector
- Los operadores de comparación aplicados a un vector nos devuelve a su vez un vector de booleanos **TRUE** o **FALSE**, en función de la evaluación de dicha comparación en cada elemento del vector
- Este vector de booleanos se puede usar a su vez para indexar elementos del vector

```
V <- c(9, 5, 20, 3, 1, 100)
```

```
V * 2
```



18	10	40	6	2	200
----	----	----	---	---	-----

# Matrices

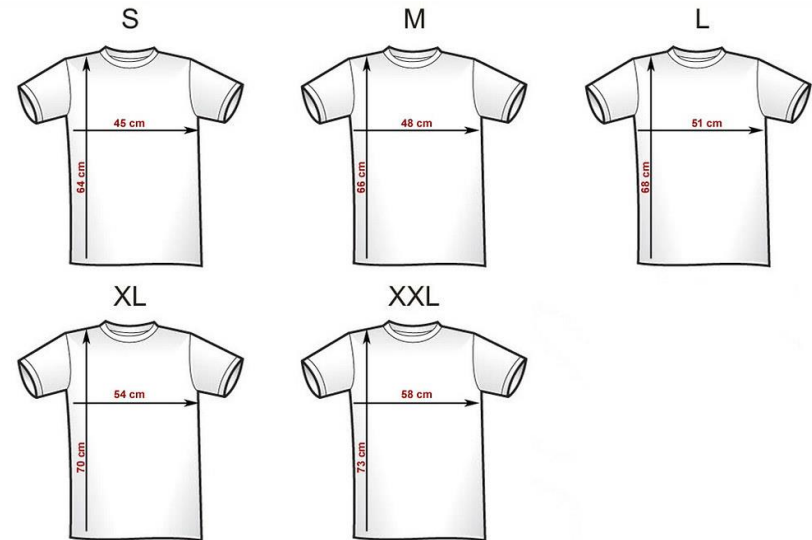
- Conjunto de elementos bidimensionales ordenados de la misma clase
- Se crean mediante la función ***matrix()***
- Podemos acceder al elemento de la fila *r* y columna *c* indexando la matriz con ***[r, c]*** (si no indicamos nada, devolveremos la fila o la columna entera)
- Podemos sumar, restar, multiplicar, etc. matrices
- Mediante ***cbind()*** y ***rbind()*** añadiremos columnas o filas respectivamente

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 7 \\ 4 & 9 & 2 \\ 6 & 0 & 5 \end{bmatrix}$$

[https://es.wikipedia.org/wiki/Matriz\\_\(matem%C3%A1ticas\)](https://es.wikipedia.org/wiki/Matriz_(matem%C3%A1ticas))

# Factores

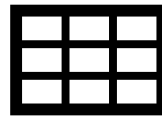
- Elementos que definen variables categóricas
- Se crean mediante la función ***factor()***
- Se pueden renombrar las variables mediante la función ***levels()***
- Los factores pueden ser ordenados o no, pudiendo comparar elementos en este último caso



[This Photo](#) by Unknown Author is licensed under [CC BY](#)



# Data frames



- Conjunto de elementos bidimensionales donde, a diferencia de las matrices, las columnas pueden ser de distintos tipos
- Elemento más común a la hora de trabajar con datos en R, es el equivalente a las tablas SQL o a las hojas de Excel
- Se crean mediante la función ***data.frame()*** mediante vectores como parámetros, que se convierten en columnas del data frame

# Operaciones con data frames

- **Indexación:** Se puede hacer mediante columna *c* y fila *f* con `[f, c]`, al igual que con las matrices. También se puede acceder a una columna mediante `$nombre_columna`
- **Extensión:** Podemos añadir columnas mediante `cbind()` y filas mediante `rbind()`
- **Filtrado:** Podemos crear un subconjunto del data frame en base a una condición lógica mediante la función `subset()`
- **Exploración:** `head()` nos devuelve los primeros elementos, `tail()` los últimos, `str()` nos devuelve su estructura, ...

Also see the **dplyr** package.

## Data Frames

```
df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))
```

A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

### List subsetting



### Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

### Matrix subsetting

`df[, 2]`



`df[2, ]`



`df[2, 2]`



`nrow(df)`  
Number of rows.

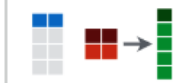
`ncol(df)`  
Number of columns.

`dim(df)`  
Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



<https://github.com/rstudio/cheatsheets/blob/main/base-r.pdf>

# Listas

- Permite almacenar de manera ordenada todo tipo de objetos, incluso de distinto tipo
- Se crean mediante la función **list()**, pudiendo dar nombre a cada uno de los elementos
- Para indexar, podemos usar el número de índice en doble corchete **[[i]]** o **\$nombre\_elemento**, al igual que en data frames

```
> v <- c(1:10)
> m <- matrix(1:9, byrow = TRUE, nrow = 3)
> df <- data.frame(c1 = c(1:5), c2 = c(6:10))
> l <- list(vector = v, matrix = m, data_frame = df)
> l
$vector
[1] 1 2 3 4 5 6 7 8 9 10

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

$data_frame
  c1 c2
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10

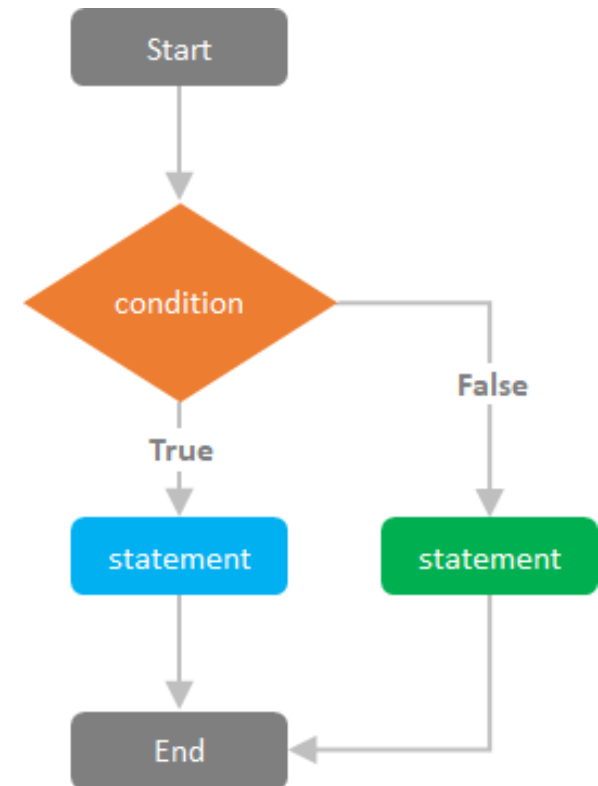
> l[[3]]
  c1 c2
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10
> l$data_frame
  c1 c2
1  1  6
2  2  7
3  3  8
4  4  9
5  5 10
> l$data_frame[1,2]
[1] 6
>
```

# Estructuras de control



# Sentencias condicionales IF ELSE

- Permiten crear bifurcaciones en la ejecución del código, en base a condiciones de tipo lógico booleano
- Se pueden crear bifurcaciones anidadas ***else if***, tantos niveles como necesitemos
- La última bifurcación ***else*** se ejecuta cuando no se ha cumplido ninguna otra condición explícitamente señalada



# Sentencias condicionales en R

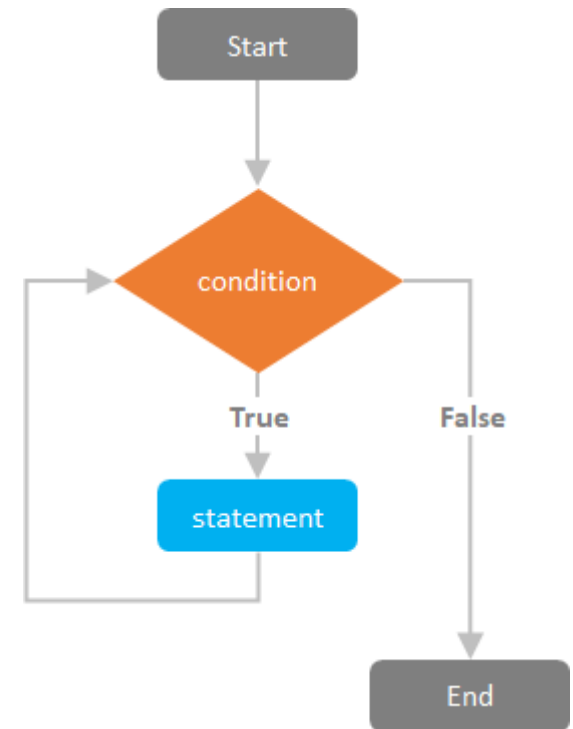
- Se construyen usando las sentencias *if* y *else*, con la condición a evaluar entre paréntesis y el código a ejecutar entre llaves
- La sintaxis es la misma en todas las bifurcaciones *else if*
- Existe la función *ifelse(condition, result\_if\_true, result\_if\_false)* para sentencias condicionales simples

```
# IF ELSE simple
if (condition) {
  code_when_condition
} else {
  code_if_not_condition
}

# IF ELSE IF ELSE anidado
if (condition_1) {
  code_when_condition_1
} else if (condition_2) {
  code_when_condition_2
} else {
  code_if_no_conditions
}
```

# Bucles WHILE

- Permiten ejecutar de manera repetida el mismo bloque de código mientras se cumpla la condición booleana del bucle
- El intérprete evalúa la condición en cada ejecución del bucle y solo ejecuta el código si esta se cumple
- Cuando la condición deja de cumplirse, el bucle es abandonado, pasando a ejecutarse el código restante del programa principal



# Bucles WHILE en R

- Se construyen mediante la sentencia **while**, la condición entre paréntesis y el código a ejecutar en bucle entre llaves
- Generalmente, en la condición se utiliza una variable numérica como control de iteraciones
- Es muy importante asegurarnos de que dicha condición llega a cumplirse, con el fin de evitar bucles infinitos

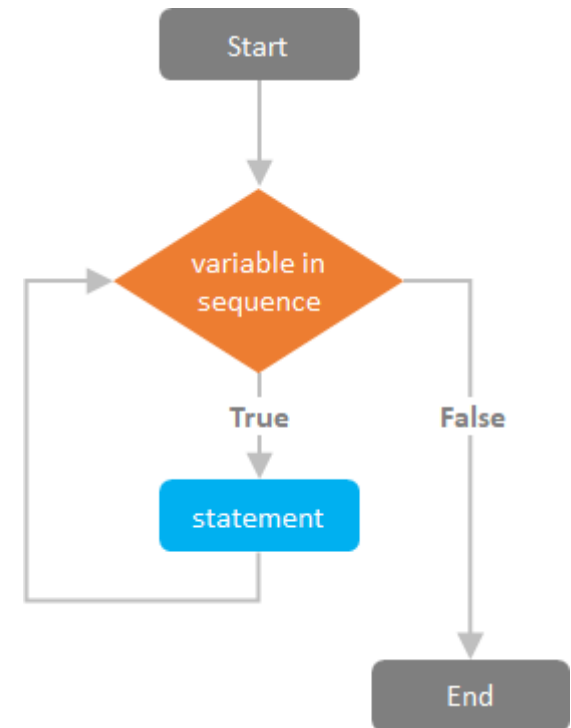
```
# Bucle WHILE
while (condition) {
  code
}
```

```
> i <- 0
> while (i < 10) {
+   print(paste("El valor de i es", i))
+   i <- i + 1
+ }
[1] "El valor de i es 0"
[1] "El valor de i es 1"
[1] "El valor de i es 2"
[1] "El valor de i es 3"
[1] "El valor de i es 4"
[1] "El valor de i es 5"
[1] "El valor de i es 6"
[1] "El valor de i es 7"
[1] "El valor de i es 8"
[1] "El valor de i es 9"
>
```



# Bucles FOR

- Permiten ejecutar de manera repetida el mismo bloque de código tantas veces como se defina en la secuencia de control
- La ejecución de código depende de la secuencia, no de ninguna condición booleana
- El código siempre se ejecutará al menos una vez, salvo secuencia vacía



# Bucles FOR en R

- Se construyen mediante la sentencia **for**, la condición de iteración entre paréntesis y el código a ejecutar en bucle entre llaves
- La condición de iteración tiene dos elementos, en orden inverso:
  - La secuencia de elementos **seq** sobre la que se itera, generalmente un vector o lista
  - El valor de la variable **var** evaluada en cada iteración

```
# Bucle FOR
for (var in seq) {
  code
}
```

```
# Bucle FOR con índices

v <- c(...)
for (i in 1:length(v)) {
  v[i]
}
```

# Funciones y paquetes



# Funciones

- Bloque de código que ejecuta una serie de tareas
- Puede ser invocado cuantas veces sea necesario en scripts o incluso en otras funciones
- Actúan como una caja negra que recibe parámetros de entrada y devuelve elementos de salida



# Funciones en R

- Las funciones en R son un objeto que puede ser creado y asignado en una variable para su posterior utilización mediante la función ***function()***
- Cuando creamos una función, debemos definir:
  - Parámetros de entrada, obligatorios y opcionales (estos últimos, con valores por defecto)
  - Nombre de la función
  - Código
- El resultado devuelto se realiza mediante la sentencia ***return***
- En las secciones anteriores ya hemos usado funciones: ***sum()***, ***matrix()***, ...

```
> my_function <- function(x, y = 1) {  
+   o <- x + y  
+   return (o)  
+ }  
>  
> z <- 3  
> my_function(z)  
[1] 4  
> my_function(z, 5)  
[1] 8  
> my_function()  
Error in my_function() : argument "x" is  
missing, with no default  
> print(z)  
[1] 3  
> print(x)  
Error in print(x) : object 'x' not found  
>
```

# Paquetes en R

- El ecosistema R nos proporciona de manera libre y gratuita acceso a +18k paquetes disponibles en **CRAN**: *Comprehensive R Archive Network*
- Cada paquete de R en CRAN nos proporciona una colección de funciones, *datasets* y documentación asociada, extendiendo o incluso añadiendo funcionalidades de R básico
- La instalación inicial de R adjunta de manera automática varios paquetes como *base* o *utils*



Top 10 R Packages for Data Science You Must Know in 2021, Analytics Vidhya

# Paquetes en R

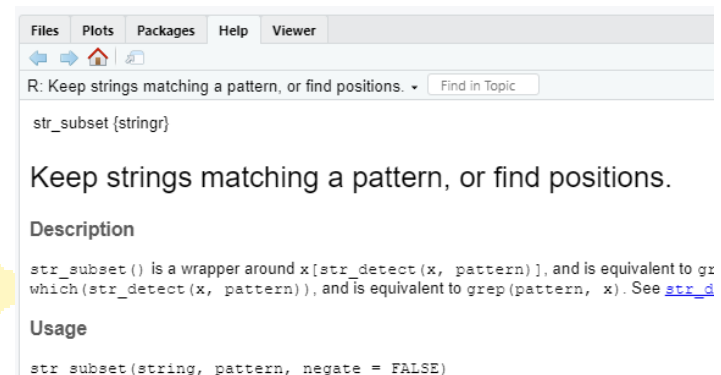
## Instalación y uso

- Para instalar un paquete usaremos la función ***install.packages***, disponible en el paquete ***utils***
- Para poder usar las funciones o ***datasets*** de un paquete, debemos adjuntarlo a la sesión de R mediante la función ***library***
- Podemos consultar los paquetes adjuntos a la sesión mediante la función ***search***
- La función ***help*** o ***?*** nos muestra la documentación de la función o del paquete seleccionado

```
> n <- c("Pedro", "Poveda", "Buedo")
> str_subset(n, "P")
Error in str_subset(n, "P") : could not find function "str_subset"
> search()
[1] ".GlobalEnv"          "tools:rstudio"
[3] "package:stats"       "package:graphics"
[5] "package:grDevices"   "package:utils"
[7] "package:datasets"    "package:methods"
[9] "Autoloads"           "package:base"
> install.packages("stringr")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/stringr_1.4.0.zip'
Content type 'application/zip' length 216753 bytes (211 KB)
downloaded 211 KB

package 'stringr' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\pedro\AppData\Local\Temp\Rtmp6dyuwn\downloaded_packages
> library("stringr")
> search()
[1] ".GlobalEnv"          "package:stringr"
[3] "tools:rstudio"       "package:stats"
[5] "package:graphics"    "package:grDevices"
[7] "package:utils"       "package:datasets"
[9] "package:methods"     "Autoloads"
[11] "package:base"
> str_subset(n, "P")
[1] "Pedro" "Poveda"
> ?str_subset
```



# Utilidades y recursos adicionales





# Manejo de fechas en R

- En R existen dos clases específicas para trabajar con objetos de tipo fecha / hora: **date** y **POSIXct**
- Internamente, estos objetos almacenan los días o segundos respectivamente desde el 1 de Enero de 1970
- Podemos convertir cadenas de texto en objetos **date** o **POSIXct** mediante las funciones **as.Date()** y **as.POSIXct()**, debiendo especificar el formato de entrada de los caracteres (**?strptime** para ver detalles)
- Estos objetos soportan operaciones aritméticas como la resta, obteniendo la diferencia en días o segundos respectivamente

```
Formatos ?strptime
-----

%Y: año en 4 dígitos (1982)
%y: año en 2 dígitos (82)
%m: mes en dos dígitos (01)
%d: día del mes en dos dígitos (15)
%A: día de la semana (Wednesday)
%a: día de la semana abreviado (Wed)
%B: mes (January)
%b: mes abreviado (Jan)

%H: hora en formato numérico (00-23)
%I: hora en formato numérico (01-12)
%M: minutos en formato numérico
%S: segundos en formato numérico
%T: atajo para formato de tiempo %H:%M:%S
%p: indicador AM/PM indicador
```



El paquete **lubridate** extiende las utilidades básicas de R para trabajar con fechas

<https://lubridate.tidyverse.org/>

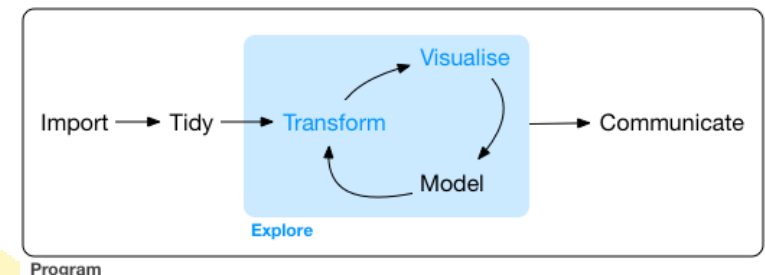
# La familia *apply*

- Funciones que permiten iterar sobre los elementos de un vector, lista o matriz, aplicando la misma función a cada elemento, simplificando los bucles
- Los parámetros de entrada son la estructura a evaluar y la función a aplicar con sus parámetros, pudiendo escribir directamente dicha función en la llamada (funciones anónimas)
- Las principales funciones son:
  - ***lapply***: itera sobre los elementos y devuelve una lista con el resultado de la función aplicada
  - ***sapply***: igual pero intenta simplificar, de manera automática, la estructura devuelta

```
lapply(X, FUN, ...)  
  
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)  
  
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)  
  
replicate(n, expr, simplify = "array")  
  
simplify2array(x, higher = TRUE)
```

# *Data Science en R con Tidyverse*

- **Tidyverse** es una colección de paquetes y utilidades para el ciclo de desarrollo del *data science*
- Está desarrollado por **RStudio** y disponible en formato *open source*
- Simplifican funciones de R base y facilitan un ciclo integrado de tratamiento de datos



Program

Fuente: <https://r4ds.had.co.nz/explore-intro.html>


# Documentación adicional



- Manuales de R en español
  - <https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf>
  - [https://cran.r-project.org/doc/contrib/rdebuts\\_es.pdf](https://cran.r-project.org/doc/contrib/rdebuts_es.pdf)
- Página principal de ***tidyverse***
  - <https://www.tidyverse.org/>
- Guía de estilo de R ***tidyverse***
  - <https://style.tidyverse.org/>
- RStudio Cheatsheets
  - <https://www.rstudio.com/resources/cheatsheets/>

# ¿Preguntas?





¡Gracias!