Reseña del Programa R- Doctorado en Economía. Universidad Nacional del Sur, basado en los libros de The Epidemiologist R Handbook y R for Data Science

Dr. Juan A. Dip- Facultad de Ciencias Económicas- UNaM-Noviembre 2022

Iniciándose en R

Lo bueno de R: es un software libre con una comunidad en constante desarrollo y aportando la modelización teórica con fines de la aplicación Empírica.

Estructura de R y terminología. RStudio como ayuda

RStudio es una interfaz gráfica de usuario (GUI) para facilitar el uso de R.

Objetos: todo lo que se almacena en R (conjuntos de datos, variables, una lista de nombres, un número de población total, incluso resultados como gráficos) son objetos a los que se les asigna un nombre y se puede hacer referencia a ellos en comandos posteriores.

Funciones: una función es una operación de código que acepta entradas y devuelve una salida transformada.

Paquetes: un paquete R es un paquete de funciones que se puede compartir.

Scripts: un script es el archivo de documento que contiene sus comandos.

Funciones - El Core de R

Una nota rápida sobre la sintaxis:las funciones están escritas en código de texto con paréntesis abiertos, así: `filter()`. Las funciones se descargan dentro de los paquetes. A veces, en el código de ejemplo, puede ver el nombre de la función vinculado explícitamente al nombre de su paquete con dos puntos (::) como este: `dplyr::filter()`.

Funciones simples

Una función es como una Maquina de hacer pan que recibe entradas (ingredientes), realiza alguna acción con esas entradas (mezclar o amazar) y produce una salida (masa para pan). La salida depende de la función que expresemos.

Las funciones normalmente operan sobre algún objeto colocado dentro de los paréntesis de la función. Por ejemplo, la función `sqrt()` calcula la raíz cuadrada de un número:

```
sqrt(64)
```

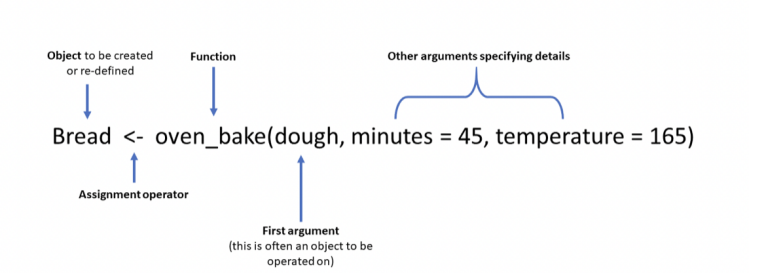
```
## [1] 8
```

Funciones con varios argumentos

Las funciones a menudo solicitan varias entradas, llamadas argumentos, ubicados dentro de los paréntesis de la función, generalmente separados por comas.

Algunos argumentos son necesarios para que la función funcione correctamente, otros son opcionales. Los argumentos opcionales tienen configuraciones predeterminadas Los argumentos pueden tomar caracteres, numéricos, lógicos (TRUE/FALSO) y otras entradas.

Por ejemplo, la imagen muestra el objeto que deseamos crear, la función utilizada para tal fin y los argumentos adicionales que especifican detalles.



Escribir Funciones

R es un lenguaje que está orientado en torno a las funciones, por lo que uno debería sentirse capacitado para escribir sus propias funciones.

Directorio de Trabajo

El directorio de trabajo es la ubicación de la carpeta raíz utilizada por R para su trabajo, donde R busca y guarda archivos de forma predeterminada.

De manera predeterminada, guardará nuevos archivos y salidas en esta ubicación, y buscará archivos para importar (por ejemplo, las bases de datos).

El directorio de trabajo aparece en texto gris en la parte superior del panel de RStudio- Console. También puede imprimir el directorio de trabajo actual ejecutando `getwd()` (dejar los paréntesis vacíos).

Recomendación Una forma común para administrar eldirectorio de trabajo y rutas de archivos, es combinar 3 elementos en un flujo de trabajo orientado a proyectos de R:

1) Un `R-project` para almacenar todos sus archivos

2)El paquete `here` para localizar archivos

3)El paquete `rio` para importar/exportar archivos.

Establecer el directorio con un comando

Muchas personas que aprendieron R se les enseña a comenzar sus scripts con un comando `setwd()`. No se recomienda este enfoque en la mayoría de las circunstancias. Sin embargo, puede usar el comando `setwd()` con la ruta del archivo de la carpeta deseada entre comillas, por ejemplo:

```
setwd("C:/Documentos/Archivos R/Mi análisis")
```

Para establecer el **directorio de trabajo manualmente** (el equivalente de señalar y hacer clic en setwd()), haga clic en el menú desplegable Sesión y vaya a “Establecer directorio de trabajo” y luego “Elegir directorio”. Esto establecerá el directorio de trabajo para esa sesión R específica. Nota: si usa este enfoque, tendrá que hacerlo manualmente cada vez que abra RStudio.

Dentro de un proyecto R

Si usa un proyecto R, el directorio de trabajo se establecerá de forma predeterminada en la carpeta raíz del proyecto R que contiene el archivo “.rproj”. Esto se aplicará si abre RStudio haciendo clic en abrir el Proyecto R (el archivo con la extensión “.rproj”).

Rutas de archivo

Quizás la fuente más común de frustración, es escribir una ruta de archivo para importar o exportar datos.

A continuación se muestra un ejemplo de una ruta de archivo "absoluta" o "dirección completa". Es probable que se rompan si los usa otra computadora. Una excepción es si está utilizando una unidad de red/compartida.

`C:/Usuarios/Nombre/Documento/Softwareanalítico/R/Proyectos/Analysis2019/data/March2019.csv`

Si escribe una ruta de archivo, tenga en cuenta la dirección de las barras. Use barras diagonales (/) para separar los componentes ("data/provincial.csv"). Para los usuarios de Windows, la forma predeterminada en que se muestran las rutas de los archivos es con barras invertidas (), por lo que deberá cambiar la dirección de cada barra. Si usa el paquete `here`, la dirección de la barra no es un problema.

Rutas de archivo relativas

En general, recomendamos proporcionar rutas de archivo "relativas" en su lugar, es decir, la ruta relativa a la raíz de su `Proyecto R`. Puede hacer esto usando el paquete aquí como se explica en la página de `proyectos R`. Una ruta de archivo relativa podría verse así: Ejemplo **Importe la cloacas csv desde las subcarpetas datosBA/Calles/Cloaca/ de un proyecto R**

`cloacas <- importar (here ("datosBA", "Cloacas", "Calles", "cloacas.csv"))` Incluso si usa rutas de archivo relativas dentro de un `proyecto R`, aún puede usar rutas absolutas para importar/exportar datos fuera de su `proyecto R`.

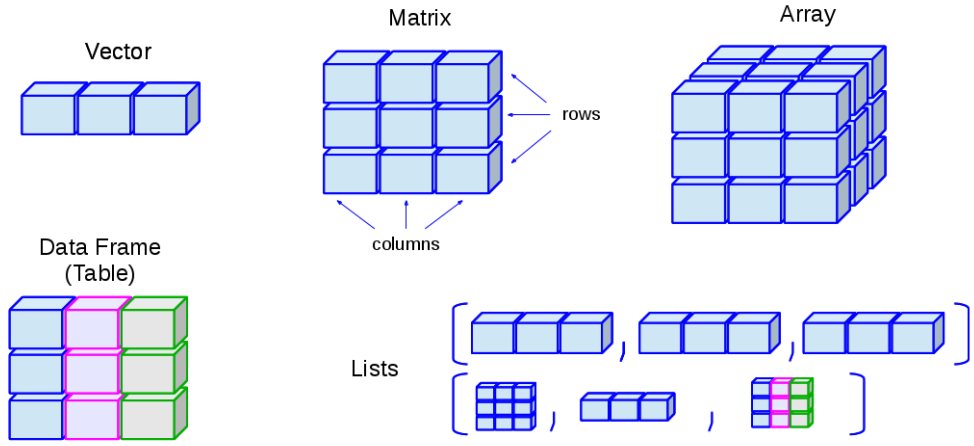
Comencemos a trabajar con algunas cuestiones básicas

Para usar las funciones disponibles en un paquete R, se deben implementar 2 pasos: El paquete debe instalarse (una vez), vemos `install` y El paquete debe cargarse (cada sesión de R) colocamos en la consola `library()`s

Ejemplo de uso de la sintaxis de un código

Para mayor claridad, las funciones a veces están precedidas por el nombre de su paquete usando el símbolo `::` de la siguiente manera: `nombre_paquete::nombre_función()`

Estructuras de los datos



Creamos vectores

```
a <- 1.7        #asignar un valor a un vector con un solo elemento
1.7 -> a
a = 1.7
assign("a", 1.7)
## Observamos el resultado
a
```

```
## [1] 1.7
```

```
print(a)
```

```
## [1] 1.7
```

```
## Generar secuencias de números
2:10
```

```
##  [1]  2  3  4  5  6  7  8  9 10
```

```
seq(from=1, to=10, by=3)
```

```
## [1]  1  4  7 10
```

```
seq(1, 10, 3)
```

```
## [1]  1  4  7 10
```

```
seq(length=10, from=1, by=3)
```

```
##  [1]  1  4  7 10 13 16 19 22 25 28
```

```
# generar repeticiones
a <- 1:3; b <- rep(a, times=3); c <- rep(a, each=3)
a
```

```
## [1] 1 2 3
```

```
b
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

```
c
```

```
## [1] 1 1 1 2 2 2 3 3 3
```

Creamos Matrices

```
a <- matrix(1:12, nrow=3, ncol=4)  ## matris de 3 filas y 4 columnas
print(a)
```

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

```
dim(a)
```

```
## [1] 3 4
```

```
#Los elementos de vectores y matrices se reciclan cuando las dimensiones involucradas lo requieren:

a <- matrix(1:8, nrow=4, ncol=4)
a
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    1    5
## [2,]    2    6    2    6
## [3,]    3    7    3    7
## [4,]    4    8    4    8
```

```
## Funcionamiento de la funcion ARRAY

z <- array(1:24, dim=c(2,3,4))
z
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

Data Frame

Un *data Frame* es un tipo especial de "lista" muy útil para el trabajo estadístico. Existen algunas restricciones para garantizar que puedan ser utilizados para hacer estadísticas

Así, un *data Frame* debe verificar que:

a) Los componentes de la lista deben ser vectores (vectores numéricos, de caracteres o lógicos), factores, matrices numéricas u otros *data Frames*. Los vectores, que son las variables en el *data Frame*, deben tener la misma longitud.

b) Básicamente, en un *data Frame* toda la información se muestra como una tabla donde las columnas tienen el mismo número de filas y pueden contener objetos de diferente tipo (números, caracteres, ...).

```
df <- data.frame(numeros=c(10,20,30,40),texto=c("a","b","c","a"),UNS=c("Maria", "MariaI", "LuKe", "Silv
df
```

```
##   numeros texto     UNS notexto
## 1      10     a   Maria      10
## 2      20     b  MariaI       4
## 3      30     c    LuKe      30
## 4      40     a SilviaL      20
```

```
df$texto
```

```
## [1] "a" "b" "c" "a"
```

```
df$numeros
```

```
## [1] 10 20 30 40
```

```
df$UNS
```

```
## [1] "Maria"   "MariaI"  "LuKe"    "SilviaL"
```

```r
mode(df) # modo  de almacenamiento del objeto
```

```
## [1] "list"
```

```r
typeof(df)  #modo de almacenamiento interno del objeto
```

```
## [1] "list"
```

```r
class(df) # clase o tipo del objeto
```

```
## [1] "data.frame"
```

Acceso/índice con corchetes ([ ])

```r
 my_vector <- c("a", "b", "c", "d", "e", "f")  # definimos el vector
my_vector[5] #elemento 5 del vector
```

```
## [1] "e"
```

```r
df[,2] # elementos que estan en la segunda columna del dataframe df
```

```
## [1] "a" "b" "c" "a"
```

```r
df[3,] # elementos que estan en la tercera fila del dataframe df
```

```
##   numeros texto  UNS notexto
## 3      30     c LuKe      30
```

```r
# ver las lineas de la 1 a la 2 y una columna especifica por ejemplo UNS
df[1:2, c("UNS")]
```

```
## [1] "Maria"  "MariaI"
```

```r
df[1:3, c( "texto","UNS")]
```

```
##   texto    UNS
## 1     a  Maria
## 2     b MariaI
## 3     c   LuKe
```

```r
# Ver filas y columnas según criterios
#
df[df$numeros > 20 , c("UNS")]
```

```
## [1] "LuKe"    "SilviaL"
```

Pipes %>%

Explicado de forma sencilla, el operador *pipe* (%>%) pasa de una salida intermedia de una función particular a la siguiente. Se puede pensar que dice "entonces". Muchas funciones se pueden vincular con %>%.

pipe enfatiza una secuencia de acciones, no el objeto en el que se realizan las acciones. Las canalizaciones son mejores cuando se debe realizar una secuencia de acciones en un objeto

pipe provienen del paquete `magrittr`, que se incluye automáticamente en los paquetes `dplyr` y `tidyverse`

pipe puede hacer que el código sea más limpio y más fácil de leer, más intuitivo

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

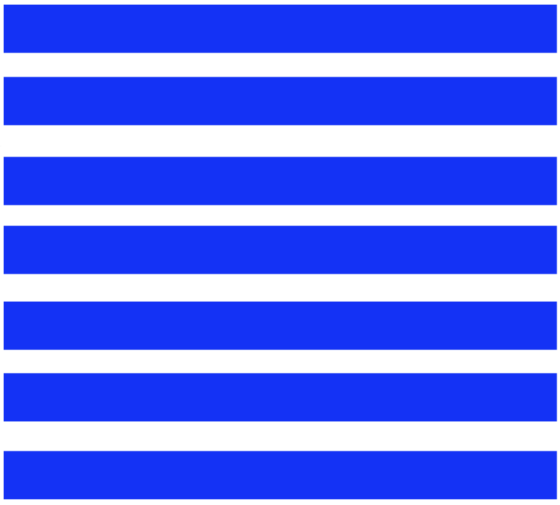
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
x_j <- df%>%select("numeros")
y_j <- df%>%select("notexto")
suma<- sum(x_j,y_j)
suma
```

```
## [1] 164
```

##Operadores##

Meaning	Operator	Example	Example Result
Equal to	`==`	`"A" == "a"`	`FALSE` (because R is case sensitive) *Note that == (double equals) is different from = (single equals), which acts like the assignment operator* `<-`
Not equal to	`!=`	`2 != 0`	`TRUE`
Greater than	`>`	`4 > 2`	`TRUE`
Less than	`<`	`4 < 2`	`FALSE`
Greater than or equal to	`>=`	`6 >= 4`	`TRUE`
Less than or equal to	`<=`	`6 <= 4`	`FALSE`
Value is missing	`is.na()`	`is.na(7)`	`FALSE`
Value is not missing	`!is.na()`	`!is.na(7)`	`TRUE`



Algunas Cuestiones entre R y Stata -

STATA	R
You can only view and manipulate one dataset at a time	You can view and manipulate multiple datasets at the same time, therefore you will frequently have to specify your dataset within the code
Online community available through https://www.statalist.org/	Online community available through RStudio, StackOverFlow, and R-bloggers
Point and click functionality as an option	Minimal point and click functionality
Help for commands available by `help [command]`	Help available by `[function]?` or search in the Help pane
Comment code using * or /// or /* TEXT */	Comment code using #
Almost all commands are built-in to Stata. New/user-written functions can be installed as **ado** files using **ssc install** [package]	R installs with **base** functions, but typical use involves installing other packages from CRAN (see page on R basics)
Analysis is usually written in a **do** file	Analysis written in an R script in the RStudio source pane. R markdown scripts are an alternative.

Basic data manipulation

STATA	R
Dataset columns are often referred to as "variables"	More often referred to as "columns" or sometimes as "vectors" or "variables"
No need to specify the dataset	In each of the below commands, you need to specify the dataset - see the page on Cleaning data and core functions for examples
New variables are created using the command **generate** *varname* =	Generate new variables using the function `mutate(varname = )`. See page on Cleaning data and core functions for details on all the below **dplyr** functions.
Variables are renamed using **rename** *old_name new_name*	Columns can be renamed using the function `rename(new_name = old_name)`
Variables are dropped using **drop** *varname*	Columns can be removed using the function `select()` with the column name in the parentheses following a minus sign
Factor variables can be labeled using a series of commands such as **label define**	Labeling values can done by converting the column to Factor class and specifying levels. See page on Factors. Column names are not typically labeled as they are in Stata.

Importing and viewing data

STATA	R
Specific commands per file type	Use `import()` from **rio** package for almost all filetypes. Specific functions exist as alternatives (see Import and export)
Reading in csv files is done by **import delimited** "filename.csv"	Use `import("filename.csv")`
Reading in xslx files is done by **import excel** "filename.xlsx"	Use `import("filename.xlsx")`
Browse your data in a new window using the command **browse**	View a dataset in the RStudio source pane using `View(dataset)`. *You need to specify your dataset name to the function in R because multiple datasets can be held at the same time. Note capital "V" in this function*
Get a high-level overview of your dataset using **summarize**, which provides the variable names and basic information	Get a high-level overview of your dataset using `summary(dataset)`

Descriptive analysis

STATA	R
Tabulate counts of a variable using **tab** *varname*	Provide the dataset and column name to `table()` such as `table(dataset$colname)`. Alternatively, use `count(varname)` from the **dplyr** package, as explained in Grouping data
Cross-tabulaton of two variables in a 2x2 table is done with **tab** *varname1 varname2*	Use `table(dataset$varname1, dataset$varname2` or `count(varname1, varname2)`

Working directory

STATA	R
Working directories involve absolute filepaths (e.g. "C:/usename/documents/projects/data/")	Working directories can be either absolute, or relative to a project root folder by using the **here** package (see Import and export)
See current working directory with **pwd**	Use `getwd()` or `here()` (if using the **here** package), with empty parentheses
Set working directory with **cd** "folder location"	Use `setwd("folder location")`, or `set_here("folder location)` (if using **here** package)

Algunos Ejercicios

Regresión Simple y logistica

```
# Generamos valores aleatorios para mi variable de inteligencia IQ
#con  media=30 y desvio estandar (sd) =2
IQ0 <- rnorm(40, 30, 2)

# Generar vector (distribuidos uniformenemten)
#con valores de notas de 40 estudiantes
 resultado0<-runif(40, min=0, max=100)

 # Correlacion
 cor(IQ0,resultado0)
```

```
## [1] -0.01407865
```

```
 # hacemos un data frame para trabajar
df0 <- as.data.frame(cbind(IQ0, resultado0))
 print(df0)
```

```
##         IQ0 resultado0
## 1  30.67008  46.308626
## 2  32.30052   4.175921
## 3  28.88280  57.369199
## 4  27.80315  17.142679
## 5  31.43072  80.301985
## 6  30.72317  57.402867
## 7  30.58437  27.293155
## 8  30.13473  53.565256
## 9  31.94444  36.472359
## 10 31.36897  42.090153
## 11 32.29834  71.539683
## 12 27.91754  38.983481
## 13 29.67116  25.011180
## 14 29.59801  83.781442
## 15 31.83573  51.635599
```

```
## 16 28.63030  66.074008
## 17 29.60884   6.923976
## 18 32.87028  97.214137
## 19 29.60067  14.410819
## 20 31.60614  24.026432
## 21 26.90534   2.559121
## 22 35.18833  32.812567
## 23 30.00159  68.022100
## 24 27.34036  85.912808
## 25 31.99136   8.599388
## 26 28.26093  36.339881
## 27 26.42539  15.652107
## 28 26.12033  12.416442
## 29 26.74852  90.148691
## 30 32.05147  59.328633
## 31 30.37822  82.469719
## 32 31.12869   3.557736
## 33 31.83052  43.960047
## 34 30.89577  22.390631
## 35 33.88748  20.716624
## 36 28.62204  25.530595
## 37 33.29314  32.464770
## 38 30.86967  43.864313
## 39 27.61699  79.759065
## 40 30.32706  82.798959
```

```
 # realizamos el modelo de regresion lineal
modelo0= lm(formula = resultado0 ~ IQ0,
                        data = df0)

summary(modelo0)
```

```
##
## Call:
## lm(formula = resultado0 ~ IQ0, data = df0)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -41.828 -21.545  -3.347  22.554  53.922
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  49.3268    64.1116   0.769    0.446
## IQ0          -0.1836     2.1154  -0.087    0.931
##
## Residual standard error: 28.19 on 38 degrees of freedom
## Multiple R-squared:  0.0001982,  Adjusted R-squared:  -0.02611
## F-statistic: 0.007533 on 1 and 38 DF,  p-value: 0.9313
```

```
# Generamos valores aleatorios para mi variable de inteligencia IQ
#con  media=30 y desvio estandar (sd) =2
IQ <- rnorm(40, 30, 2)

#  ordenamos IQ en orden ascendente
IQ <- sort(IQ)
```

```r
# Generar vector con valores de aprobado y reprobado de 40 estudiantes
resultado <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
1, 0, 0, 0, 1, 1, 0, 0, 1, 0,
0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
1, 1, 1, 0, 1, 1, 1, 1, 0, 1)

# hacemos un data frame para trabajar
df <- as.data.frame(cbind(IQ, resultado))

# Print data frame
print(df)
```

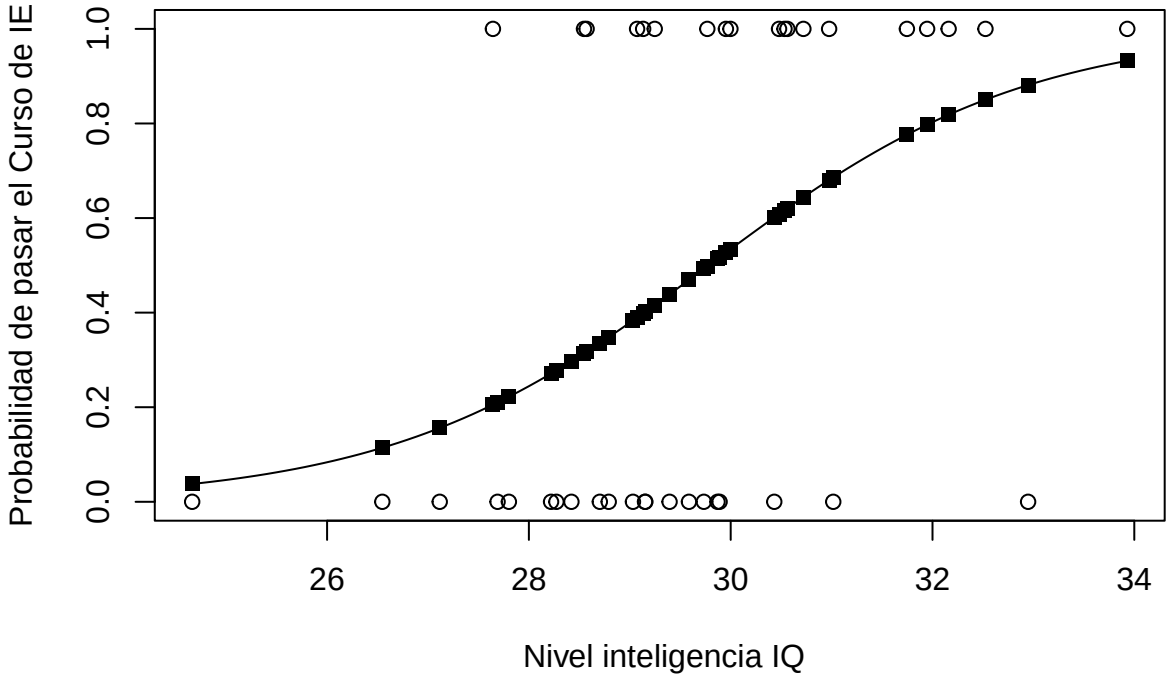
```
##          IQ resultado
## 1  24.66293         0
## 2  26.54749         0
## 3  27.11634         0
## 4  27.64363         1
## 5  27.68952         0
## 6  27.80180         0
## 7  28.21980         0
## 8  28.27263         0
## 9  28.42148         0
## 10 28.54621         1
## 11 28.57203         1
## 12 28.70309         0
## 13 28.79123         0
## 14 29.03145         0
## 15 29.07231         1
## 16 29.13226         1
## 17 29.15085         0
## 18 29.15386         0
## 19 29.24728         1
## 20 29.39420         0
## 21 29.58630         0
## 22 29.73364         0
## 23 29.76829         1
## 24 29.86935         0
## 25 29.89000         0
## 26 29.95266         1
## 27 29.99743         1
## 28 30.43183         0
## 29 30.47904         1
## 30 30.53331         1
## 31 30.56187         1
## 32 30.72040         1
## 33 30.97549         1
## 34 31.01635         0
## 35 31.74653         1
## 36 31.94674         1
## 37 32.16000         1
## 38 32.52489         1
## 39 32.94753         0
## 40 33.93055         1
```

```r
# hacemos un grafico tipo para observar la probabilidad de pasar el curso de IE
plot(IQ, resultado, xlab = " Nivel inteligencia IQ",
ylab = "Probabilidad de pasar el Curso de IE")

# Creamos un modelo logistico
g = glm(resultado~IQ, family=binomial, df)

# Creamos una curva basada en la predicción usando el modelo de regresión
curve(predict(g, data.frame(IQ=x), type="resp"), add=TRUE)

# Dibujamos un conjunto de puntos
# Basado en el ajuste al modelo de regresión
points(IQ, fitted(g), pch=15)
```




```r
# Resumen del modelo
summary(g)
```

```
##
## Call:
## glm(formula = resultado ~ IQ, family = binomial, data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0641  -0.9907  -0.3848   0.9867   1.7797
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -18.8542     7.7314  -2.439   0.0147 *
## IQ            0.6331     0.2608   2.428   0.0152 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 55.352  on 39  degrees of freedom
## Residual deviance: 46.986  on 38  degrees of freedom
## AIC: 50.986
##
## Number of Fisher Scoring iterations: 4
```