# Aprende R Haciendo

***Release 0.1***

**Francisco Palm**

23 de December de 2013

# Índice general

Contenidos:

CAPÍTULO 1

# Motivación

Hay un mito en la tecnociencia según la cual las herramientas "no son buenas ni malas, sino dependen de como se utilicen". Sin embargo, las herramientas en efecto condicionan el quehacer, e incluso el sentido de la investigación científica. Este curso pretende enriquecer el abanico de las alternativas disponibles.

Se podría intentar ser políticamente correcto y decir que queda a juicio de cada quién decir si R es mejor o peor que otras herramientas estadísticas. Sin embargo, desde el punto de vista de cualquier investigador riguroso el uso de R mejora con creces las funcionalidades que ofrece una hoja de cálculo, y en términos éticos y prácticos trabajar con software libre y de código abierto, basado en estándares abiertos, es sin duda una opción superior.

Muchos profesionales de la informática una vez graduados evaden las tareas de programación, de manera similar a como muchas personas evaden las matemáticas tanto como pueden. Esto se debe a que tanto una como la otra son, por lo general, muy mal enseñadas. En la actualidad, la programación es una destreza básica y fundamental al alcance de cualquier persona.

Estas notas buscan son una presentación de R basada en ejemplos, en lugar de hacer repaso pormenorizado de todos los conceptos y elementos del entorno R, se trabajará en base a demostraciones del flujo de trabajo usual para muchos usuarios de R con lo cuál se espera que cada persona gane comprensión de los retos y oportunidades que ofrece el uso de una herramienta de este tipo.

## 1.1 ¿Qué es R?

R es un lenguaje de programación, que se ejecuta sobre un entorno que ofrece capacidades de visualización y una gran cantidad de funciones y métodos de estadística computacional, y mecanismos para extender sus funcionalidades. Por lo tanto, el correcto manejo de R implica desarrollar destrezas de programación.

R cuenta con el conjunto más extenso de métodos estadísticos disponibles en cualquier software estadístico, ya sea libre o privativo, además de potentes y flexibles capacidades de graficación. Sin embargo, el diseño de R en tanto que lenguaje de programación, presenta debilidades que han motivado críticas incluso por sus usuarios más fieles, un ejemplo de esto es el libro The R inferno.

## 1.2 Descarga e instalación

### 1.2.1 Para Windows

La versión de R para Windows se puede conseguir desde: http://cran.r-project.org/bin/windows/base/

Descargar e instalar el archivo `R-X.X.X-win.exe` desde el enlace donde dice: *Download R X.X.X for Windows*, donde X.X.X es la versión de R más reciente.

### 1.2.2 Para Linux

Generalmente la versión disponible en los repositorios de las distintas distribuciones está algo desactualizada, esto no es conveniente porque se pueden tener incompatibilidad con los paquetes propios de R.

Para resolver esto se configuran los repositorios según la información que aparece en: http://cran.r-project.org/bin/linux/

Se generan paquetes de las últimas versiones de R para Debian, Redhat, SUSE y Ubuntu, si utiliza una distribución que no es compatible con ninguno de estos repositorios, puede compilar la última versión de R desde las fuentes http://cran.r-project.org/sources.html

R También está empaquetado para MacOSX Snow Leonard y superior

## 1.3 RStudio

RStudio es un entorno de desarrollo integrado para R, disponible para Linux (Debian/Ubuntu/Fedora/OpenSUSE), Windows y MacOSX.

Es un entorno diseñado para programación y en este sentido es lo mejor disponible para R, aparte de ser de código abierto por lo que está siendo continuamente mejorado.

Existen otros editores orientados a usuarios que quieren aplicar técnicas estadísticas a datos, este enfoque termina siendo limitado porque con mucha frecuencia se requiere modificar la estructura de los datos, experimentar con las técnicas, aplicar las técnicas de manera iterativa, generar reportes automáticos o configurar en detalle los graficos, en cualquiera de estos casos es preferible un entorno de programación como RStudio.

Desde el sitio: http://rstudio.org/download/desktop Descargar e instalar la versión más reciente

### 1.3.1 La Interfaz de Rstudio

Ofrece cuatro zonas personalizables: *Editor*, *Consola*, *Datos* y *Ayuda*.

Ilustración 1: Interfaz de RStudio

La primera vez que se ejecuta el programa la zona del Editor aparece desactivada, se activará en el momento que comience a editar un script de R.

### 1.3.2 Modos de trabajo

Se comienza con el modo interactivo (en la consola) haciendo pruebas y utilizándolo como manual de ayuda.

La serie de comandos que devuelven los resultados esperados se guardan y modifican (en el editor) como scripts para crear rutinas reproducibles y automatizables.

En el espacio Datos aparecen los objetos del espacio de trabajo actual, y el histórico de comandos utilizados.

En el espacio Ayuda se encuentran por defecto el explorador de archivos, el gestor de gráficos, el gestor de paquetes, y la interfaz de ayuda.

Todos los espacios están conectados y los cambios realizados en un espacio con frecuencia afectan a los demás.

### 1.3.3 Estructura de un proyecto

Se recomienda que cada proyecto tenga una estructura de archivos con las carpetas: (puede traducir los nombres si lo desea): `data`, `documents`, `graphics` , `images`, `notes`, `code`. También puede ser otra estructura adaptada a las preferencias del investigador, cualquiera es mejor que tener todos los archivos sueltos en una misma carpeta.

Ilustración 2: Estructura de un proyecto de análisis de datos

También es recomendable, si se utiliza RStudio, utilizar *proyectos* con el fin de gestionar los entornos de trabajo de manera que cada uno esté asociado a un espacio de trabajo, un historial, y un código bien diferenciado.

## 1.4 Un primer ejemplo

Empecemos por abrir en RStudio el archivo `simpleRprogram.R`.

A continuación se va a ejecutar el programa línea por línea usando el botón `Run` que está en la parte superior derecha del editor.

```
1  setwd("[ruta a]/myRfolder/")
2  mydata <- read.csv("mydata.csv")
3  mydata
4
5  mydata$workshop <- factor(mydata$workshop)
6  summary(mydata)
7
8  plot(mydata$q1, mydata$q4)
9
10 myModel <- lm(q4 ~ q1 + q2 + q3, data=mydata)
11 summary(myModel)
12 anova(myModel)
13 plot(myModel)
```

La función `setwd()` (establecer directorio de trabajo) indica a R cual va a ser el directorio que va a tomar como referencia para acceder a los archivos.

La expresión `mydata <- read.csv("mydata.csv")` utiliza la función `read.csv()` para leer el archivo separado por comas `mydata.csv`. El resultado que es un objeto *dataframe* lo asigna mediante el operador `<-` a la variable `mydata`.

Un objeto `dataframe` es una estructura de datos de tipo tabla en la que cada columna es un *campo* y cada fila un *registro* de forma similar a como se almacenan los datos en una base de datos o en una hoja de cálculo.

Como estamos ejecutando el archivo línea por línea, y por lo tanto estamos en el *modo interactivo* al ejecutar la línea 3 donde aparece `mydata` mostrará en la consola el contenido del objeto.

```
1  > mydata
2    workshop gender q1 q2 q3 q4
3  1        1      f  1  1  5  1
4  2        2      f  2  1  4  1
5  3        1      f  2  2  4  3
6  4        2         3  1 NA  3
7  5        1      m  4  5  2  4
8  6        2      m  5  4  5  5
9  7        1      m  5  3  4  4
10 8        2      m  4  5  5  5
```

Al leer del archivo el campo `workshop` se considera de tipo numérico. Para que R lo interprete como un campo categórico se utiliza la función `factor()`.

En la siguiente línea, la función `summary()` devuelve las estadísticas descriptivas básicas de cada uno de los campos. Nótese como el campo Workshop muestra un conteo de la ocurrencia de cada categoría "1" y "2".

La función `plot()` genera un gráfico de dispersión entre los valores en los campos `q1` y `q4`, para acceder a estos valores se utiliza la notación `mydata$q1` que hace referencia a los valores del campo `q1` como un vector numérico.

A continuación, se quiere construir un modelo lineal, en este caso como la respuesta `q4` se explica mediante las contribuciones de `q1`, `q2` y `q3`. Para lo cual se escribe la fórmula `q4 ~ q1 + q2 + q3`. El argumento `data=mydata` indica a R de cual conjunto de datos toma las variables.

El resultado se asigna a la variable `myModel` en este caso es un objeto de la clase `lm`. Esta es una estructura que contiene los componentes generados durante el ajuste: `coefficients`, `residuals`, `fitted.values`, entre otros.

A este modelo lineal se le puede aplicar un análisis de varianza mediante la función `anova()`, y finalmente se genera un conjunto de gráficos predefinidos para el modelo lineal con `plot(myModel)`. Hay muchas funciones como `plot()` y `summary()` que responden de forma distinta dependiendo del argumento de entrada.

CAPÍTULO 2

# Manejo de Datos

## 2.1 Guardar y cargar datos en R

### 2.1.1 Conjuntos de datos Rdata

Los datos en R se pueden guardar como archivos `.Rdata` usando la función `save()`. Después de lo cual pueden cargarse en R mediante la función `load()`. En el código a continuación `rm()` borra el objeto `a` de R.

```
1 a <- 1:10
2 save(a, file="./data/dumData.Rdata")
3 rm(a)
4 load("./data/dumData.Rdata")
5 print(a)
```

Cada vez que se dice *cargar en R* o *borrar de R*, se refiere al espacio de trabajo de R, donde residen los objetos a los que R tiene acceso para realizar operaciones.

En la línea 1, se crea un vector numérico con los valores *1, 2, ... 10* usando la notación resumida `1:10` y se asigna a la variable `a`.

Se utiliza la función `save()` para guardarlo en un archivo con el nombre `dumData.Rdata`, debe existir en este caso una carpeta `data` en el directorio de trabajo actual de lo contrario devolverá un error.

Para verificar cual es nuestro directorio de trabajo actual podemos utilizar la función `getwd()`. Si en necesario lo podemos cambiar con la función `setwd()` tal y como se hizo en *Un primer ejemplo*.

### 2.1.2 Importar y exportar desde archivos .CSV

Es posible crear archivos separados por comas, compatibles con las hojas de cálculo como EXCEL o LibreOffice Calc.

```
1 var1 <- 1:5
2 var2 <- (1:5) / 10
3 var3 <- c("R", "and", "Data Mining", "Examples", "Case Studies")
4 df1 <- data.frame(var1, var2, var3)
5 names(df1) <- c("VariableInt", "VariableReal", "VariableChar")
6 write.csv(df1, "./data/dummmyData.csv", row.names = FALSE)
7 df2 <- read.csv("./data/dummmyData.csv")
8 print(df2)
```

Se crea el vector `var1` de forma similar al ejemplo anterior, para el vector `var2` se crea un vector con los valores correlativos del 1 al 5 `1:5` y este valor se divide entre 10, es decir, se obtiene un nuevo vector en el que cada elemento es dividido por 10.

```
 > (1:5) / 10
[1] 0.1 0.2 0.3 0.4 0.5
```

En estos casos se dice que el operador `/` está *vectorizado* ya que aplica a todo el vector. Es el comportamiento usual de los operadores aritméticos cuando se utilizan con vectores y matrices.

El vector `var3` se contruye *combinando* valores distintos con la función `c()`. Esta es una de las formas más comunes de crear vectores en R. En este caso como los valores son cadenas de caracteres, el vector es de tipo `character`.

Y con `data.frame(var1, var2, var3)` se crea un dataframe en el que cada columna se corresponde con los vectores recién creados. Nótese que los tres vectores en este caso tienen la misma longitud, de lo contrario se repetirían los valores de los vectores más pequeños hasta completar un número de registros igual al del vector más grande.

```
> data.frame(1:2, 1:5, 1:10)
   X1.2 X1.5 X1.10
1     1    1     1
2     2    2     2
3     1    3     3
4     2    4     4
5     1    5     5
6     2    1     6
7     1    2     7
8     2    3     8
9     1    4     9
10    2    5    10
```

La sentencia `names(df1) <- c("VariableInt", "VariableReal", "VariableChar")` establece los nombres de las columnas de `df1` cada valor del vector creado con la función `c()` se corresponde con cada una de las columnas.

Al ejecutar `write.csv(df1, "./data/dummmyData.csv", row.names=FALSE)` se genera un archivo separado por comas de nombre `dummmyData.csv` en la carpeta `data`. La opción `row.names=FALSE` indica que el archivo no va a incluir los nombres de fila (que son los números de fila por defecto) como un campo adicional.

Finalmente, se cargan los datos del archivo recién creado en un nuevo dataframe `df2` con `df2 <- read.csv("./data/dummmyData.csv")` y se imprime su contenido para verificar que todo ha ido bien.

---

**Nota:** Archivos SAS, EXCEL y Bases de datos

Es posible trabajar de la misma forma con archivos de SAS, con archivos de EXCEL y con bases de datos, pero estas operaciones en general dependen del sistema operativo y del software instalado. Por ejemplo, en el caso de trabajar con los archivos de SAS hay que tener instalado SAS.

---

## 2.2 Acceder a los datos

### 2.2.1 El conjunto de datos iris

Es un conjunto incorporado a R de manera que no necesita ser cargado. Consiste en 50 muestras de tres clases de flores de iris.

Este conjunto de datos es clásico y cuenta incluso con su propia página web http://en.wikipedia.org/wiki/Iris_flower_data_set

Una clase de flores es linealmente separable de las otras dos, mientras que las dos últimas no son linealmente separables la una de la otra. El conjunto cuenta con cinco datos:

- longitud de los sépalos en cm,
- ancho de los sépalos en cm,
- longitud de los pétalos cm,
- ancho de los petalos en cm, y
- clase: Iris Setosa, Iris Versicolour, e Iris Virginica.

### 2.2.2 Revisar los datos

```
dim(iris)
names(iris)
str(iris)
attributes(iris)
```

Con las funciones dadas se obtiene las *dimensiones* del conjunto de datos `dim()`, los *nombres* de los campos `names()`, una descripción compacta de la *estructura* `str()`, y una versión extendida de los *atributos* `attributes()`.

```
iris[1:5, ]
head(iris)
tail(iris)
```

Los dataframe se pueden ver como matrices a los que se accede por filas y columnas de la forma `df[fil, col]`. Así, la sentencia `iris[1:5, ]` devuelve todas los campos (ya que después de la columna está vacío) para los primeros cinco registros (filas de la 1 a la 5).

Las funciones `head()` y `tail()` devuelven los primeros y los últimos registros de un conjunto de datos, por defecto 6, aunque se puede establecer con el parámetro `n=`.

```
> iris[1:10, "Sepal.Length"]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
> iris$Sepal.Length[1:10]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

También es posible los valores de una columna en particular, en el código anterior se muestran dos formas alternativas de obtener los valores del campo `Sepal.length` en los diez primeros registros. En el primer caso utilizando notación matricial `iris[1:10, "Sepal.Length"]`, y en el segundo accediendo primero al campo mediante la notación `objeto$campo` y luego extrayendo los primeros diez valores mediante `[1:10]`.

# Estadística descriptivas

## 3.1 Variables individuales

```
summary(iris)
```

```
quantile(iris$Sepal.Length)
quantile(iris$Sepal.Length, c(.1, .3, .65))
```

```
var(iris$Sepal.Length)
hist(iris$Sepal.Length)
```

```
plot(density(iris$Sepal.Length))
```

```
table(iris$Species)
pie(table(iris$Species))
```

```
barplot(table(iris$Species))
```

## 3.2 Múltiples variables

```
cov(iris$Sepal.Length, iris$Petal.Length)
cov(iris[,1:4])
cor(iris$Sepal.Length, iris$Petal.Length)
cor(iris[,1:4])
```

```
aggregate(Sepal.Length ~ Species, summary, data=iris)
```

```
boxplot(Sepal.Length~Species, data=iris)
```

```
with(iris, plot(Sepal.Length, Sepal.Width,
                col=Species, pch=as.numeric(Species)))
```

```
plot(jitter(iris$Sepal.Length), jitter(iris$Sepal.Width))
```

```
pairs(iris)
```

## 3.3 Y seguimos explorando

```
library(scatterplot3d)
scatterplot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

```
## library(rgl)
## plot3d(iris$Petal.Width, iris$Sepal.Length, iris$Sepal.Width)
```

```
distMatrix <- as.matrix dist iris[,1:4]))
heatmap(distMatrix)
```

```
library(lattice)
levelplot(Petal.Width~Sepal.Length*Sepal.Width, iris, cuts=9,
          col.regions=grey.colors(10)[10:1])
```

```
filled.contour(volcano, color=terrain.colors, asp=1,
               plot.axes=contour volcano, add=T))
```

```
persp(volcano, theta=25, phi=30, expand=0.5, col="lightblue")
```

```
library(MASS)
parcoord(iris[1:4], col=iris$Species)
```

```
library(lattice)
parallelplot(~iris[1:4] | Species, data=iris)
```

```
library(ggplot2)
qplot(Sepal.Length, Sepal.Width, data=iris, facets=Species ~.)
```

### 3.3.1 Guardar un gráfico en un archivo

```
## # save as a PDF file
## pdf("myPlot.pdf")
## x <- 1:50
## plot(x, log(x))
## graphics.off()
## #
## # Save as a postscript file
## postscript("myPlot2.ps")
## x <- -20:20
## plot(x, x^2)
## graphics.off()
```

# ggplot2

setwd("c:/myRfolder") load(file = "mydata100.Rdata")

# Get rid of missing values for facets mydata100 <- na.omit(mydata100) attach(mydata100) library("ggplot2")

## 4.1 Gráficos por defecto con qplot

```
# A. Bar Plot
myPlot <- qplot(workshop)

# B. Histogram
myPlot <- qplot(posttest)                    # Histogram

# C. Useless Scatter
myPlot <- qplot(workshop, gender)

# D. Strip Plot - Vertical
myPlot <- qplot(workshop, posttest)

# E. Strip Plot - Horizontal
myPlot <- qplot(posttest, workshop)

# F. Scatter Plot
myPlot <- qplot(pretest, posttest)

grid.newpage()  # Clear page.
```

### 4.1.1 Gráficos de barras

# Bar plot - vertical

qplot(workshop, geom = "bar")

**ggplot(mydata100, aes( workshop ) ) +** geom_bar()

# Bar plot - horizontal

qplot(workshop, geom = "bar") + coord_flip()

**ggplot(mydata100, aes(workshop) ) +** geom_bar() + coord_flip()

# Bar plot - single bar stacked

**qplot(factor(“”), fill = workshop,** geom = “bar”, xlab = “”) + scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100,** aes(factor(“”), fill = workshop) ) + geom_bar() + scale_x_discrete(“”) + scale_fill_grey(start = 0, end = 1)

# Pie charts, same as stacked bar but polar coordinates

**qplot(factor(“”), fill = workshop,** geom = “bar”, xlab = “”) + coord_polar(theta = “y”) + scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100,** aes( factor(“”), fill = workshop ) ) + geom_bar( width = 1 ) + scale_x_discrete(“”) + coord_polar(theta = “y”) + scale_fill_grey(start = 0, end = 1)

# Bar Plots - Grouped

**qplot(gender, geom = “bar”,**

fill = workshop, position = “stack”) +

scale_fill_grey(start = 0, end = 1)

**qplot(gender, geom = “bar”,**

fill = workshop, position = “fill”) +

scale_fill_grey(start = 0, end = 1)

**qplot(gender, geom = “bar”,**

fill = workshop, position = “dodge”) +

scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100, aes(gender, fill = workshop) ) +** geom_bar(position = “stack”) + scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100, aes(gender, fill = workshop) ) +** geom_bar(position = “fill”) + scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100, aes(gender, fill = workshop ) ) +** geom_bar(position = “dodge”) + scale_fill_grey(start = 0, end = 1)

# Bar Plots - Faceted

qplot(workshop, geom = “bar”, facets = gender ~ . )

**ggplot(mydata100, aes(workshop) ) +** geom_bar() + facet_grid(gender ~ . )

# Bar Plots - Pre-summarized data

**qplot( factor( c(1, 2) ), c(40, 60), geom = “bar”,** xlab = “myGroup”, ylab = “myMeasure”)

**myTemp <- data.frame(** myGroup = factor( c(1, 2) ), myMeasure = c(40, 60)

) myTemp ggplot(data = myTemp, aes(myGroup, myMeasure) ) +

geom_bar()

### 4.1.2 Gráficos de puntos

**qplot(workshop, stat = “bin”,**

facets = gender ~ . , geom = “point”, size = I(4) ) +

coord_flip()

# Same thing but suppressing legend a different way qplot(workshop, stat = “bin”,

facets = gender ~ . , geom = “point”, size = 4 ) +

opts(legend.position = “none”) + coord_flip()

**ggplot(mydata100,** aes(workshop, ..count.. ) ) + geom_point(stat = “bin”, size = 4) + coord_flip()+ facet_grid( gender ~ . )

# —Adding Titles and Labels—

**qplot(workshop, geom = “bar”,** main = “Workshop Attendance”, xlab = “Statistics Package nWorkshops”)

**ggplot(mydata100, aes(workshop, ..count..)) +** geom_bar() + opts( title = “Workshop Attendance” ) + scale_x_discrete(“Statistics Package nWorkshops”)

# Example not in text: labels of continuous scales. ggplot(mydata100, aes(pretest,posttest ) ) +

geom_point() + scale_x_continuous(“Test Score Before Training”) + scale_y_continuous(“Test Score After Training”) + opts(title = “The Relationship is Linear”)

### 4.1.3 Histogramas y gráficos de densidad

# Simle Histogram qplot(posttest, geom = “histogram”) qplot(posttest, geom = c(“histogram”, “rug”) ) #not shown

**ggplot(mydata100, aes(posttest) ) +** geom_histogram() + geom_rug() # not shown in text

# Histogram with more bars. qplot(posttest, geom = “histogram”, binwidth = 0.5)

**ggplot(mydata100, aes(posttest) ) +** geom_histogram(binwidth = 0.5)

# Density plot qplot(posttest, geom = “density”)

**ggplot(mydata100, aes(posttest)) +** geom_density()

# Histogram with density

**qplot(data = mydata100, posttest, ..density..,** geom = c(“histogram”, “density”) )

**ggplot(data=mydata100) +** geom_histogram( aes(posttest, ..density..) ) + geom_density( aes(posttest, ..density..) ) + geom_rug( aes(posttest) )

# Histogram - separate plots by group

qplot(posttest, geom = “histogram”, facets = gender ~ . )

**ggplot(mydata100, aes(posttest) ) +** geom_histogram() + facet_grid( gender ~ . )

# Histogram with Stacked Bars

**qplot(posttest, geom = “histogram”, fill = gender) +** scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100, aes(posttest, fill = gender) ) +** geom_bar() + scale_fill_grey(start = 0, end = 1)

### 4.1.4 Gráficos QQ

qplot(sample = posttest, stat = “qq”)

**ggplot(mydata100, aes(sample = posttest) ) +** stat_qq()

### 4.1.5 Gráficos de tiras

# Simple, but jitter too wide for our small data

qplot( factor(“”), posttest, geom = “jitter”, xlab = “”)

**ggplot(mydata100, aes(factor(“”), posttest) ) +** geom_jitter() + scale_x_discrete(“”)

# Again, with more narrow jitter

**qplot( factor(“”), posttest, data = mydata100, xlab = “”,** position = position_jitter(width = .02))

**ggplot(mydata100, aes(factor(“”), posttest) ) +** geom_jitter(position = position_jitter(width = .02)) + scale_x_discrete(“”)

# Strip plot by group. # First, the easy way, with too much jitter for our data:

qplot(workshop, posttest, geom = “jitter”)

**ggplot(mydata100, aes(workshop, posttest) ) +** geom_jitter()

# Again, limiting the jitter for our small data set:

**qplot(workshop, posttest, data = mydata100, xlab = “”,** position = position_jitter(width = .08) )

**ggplot(mydata100, aes(workshop, posttest) ) +** geom_jitter(position = position_jitter(width = .08) ) + scale_x_discrete(“”)

### 4.1.6 Gráficos de dispersión

# Simple scatter Plot

qplot(pretest, posttest) qplot(pretest, posttest, geom = “point”)

**ggplot(mydata100, aes(pretest, posttest) ) +** geom_point()

# Scatter plot connecting points sorted on x.

qplot(pretest, posttest, geom = “line”)

**ggplot(mydata100, aes(pretest, posttest) ) +** geom_line()

# Scatter plot connecting points in data set order.

qplot(pretest, posttest, geom = “path”)

**ggplot(mydata100, aes(pretest, posttest) ) +** geom_path()

# Scatter plot with skinny histogram-like bars to X axis.

**qplot(pretest,posttest,** xend = pretest, yend = 50, geom = “segment”)

**ggplot(mydata100, aes(pretest, posttest) ) +**

**geom_segment( aes( pretest, posttest,** xend = pretest, yend = 50) )

# Scatter plot with jitter

# qplot without: qplot(q1, q4)

# qplot with: qplot(q1, q4, position =

position_jitter(width = .3, height = .3) )

# ggplot without: ggplot(mydata100, aes(x = q1, y = q2) ) +

geom_point()

```
# ggplot with: ggplot(mydata100, aes(x = q1, y = q2) ) +
    geom_point(position = position_jitter(width = .3,height = .3) )
# Scatter plot on large data sets
pretest2 <- round( rnorm( n = 5000, mean = 80, sd = 5) ) posttest2 <- round( pretest2 +
    rnorm( n = 5000, mean = 3, sd = 3) )
pretest2[pretest2 > 100] <- 100 posttest2[posttest2 > 100] <- 100 temp <- data.frame(pretest2, posttest2)
# Small, jittered, transparent points.
qplot(pretest2, posttest2, data = temp, geom = "jitter", size = I(2), alpha = I(0.15), position = position_jitter(width
    = 2) )
ggplot(temp, aes(pretest2, posttest2), size = 2, position = position_jitter(x = 2, y = 2) ) + geom_jitter(colour =
    alpha("black", 0.15) )
# Hexbin plots
# In qplot using default colors. qplot(pretest2, posttest2, geom = "hex", bins = 30)
# This works too: ggplot(temp, aes(pretest2, posttest2) ) +
    stat_binhex(bins = 30) +
# In ggplot, switching to greyscale. ggplot(temp, aes(pretest2, posttest2) ) +
    geom_hex( bins = 30 ) + scale_fill_continuous(
        low = "grey80", high = "black")
# Using density contours and small points.
qplot(pretest2, posttest2, data = temp, size = I(1), geom = c("point","density2d"))
ggplot(temp, aes( x = pretest2, y = posttest2) ) + geom_point(size = 1) + geom_density2d()
# Density shading ggplot(temp, aes( x = pretest2, y = posttest2) ) +
    stat_density2d(geom = "tile", aes(fill = ..density..), contour = FALSE) +
    scale_fill_continuous( low = "grey80", high = "black")
rm(pretest2,posttest2,temp)
# Scatter plot with regression line, 95 % confidence intervals.
qplot(pretest, posttest, geom = c("point", "smooth"), method = lm )
ggplot(mydata100, aes(pretest, posttest) ) + geom_point() + geom_smooth(method = lm)
# Scatter plot with regression line but NO confidence intervals.
qplot(pretest, posttest, geom = c("point", "smooth"), method = lm, se = FALSE )
ggplot(mydata100, aes(pretest, posttest) ) + geom_point() + geom_smooth(method = lm, se = FALSE)
# Scatter with x = y line
qplot(pretest, posttest, geom = c("point", "abline"), intercept = 0, slope = 1 )
ggplot(mydata100, aes(pretest, posttest) ) + geom_point() + geom_abline(intercept = 0, slope = 1)
# Scatter plot with different point shapes for each group.
qplot(pretest, posttest, shape = gender)
ggplot(mydata100, aes(pretest, posttest) ) + geom_point( aes(shape = gender ) )
```

# Scatter plot with regressions fit for each group.

**qplot(pretest, posttest,** geom = c("smooth", "point"), method = "lm", shape = gender, linetype = gender)

**ggplot(mydata100,** aes(pretest, posttest, shape = gender, linetype = gender) ) + geom_smooth(method = "lm") + geom_point()

# Scatter plot faceted for groups

**qplot(pretest, posttest,** geom = c("smooth", "point"), method = "lm", shape = gender, facets = workshop ~ gender)

**ggplot(mydata100,** aes(pretest, posttest, shape = gender) ) + geom_smooth(method = "lm") + geom_point() + facet_grid(workshop ~ gender)

# Scatter plot with vertical or horizontal lines

**qplot(pretest, posttest,** geom = c("point", "vline", "hline"), xintercept = 75, yintercept = 75)

**ggplot(mydata100, aes(pretest, posttest)) +** geom_point() + geom_vline(intercept = 75) + geom_hline(intercept = 75)

# Scatter plot with a set of vertical lines

**qplot(pretest, posttest, type = "point") +** geom_vline(xintercept = seq(from = 70,to = 80,by = 2) )

**ggplot(mydata100, aes(pretest, posttest)) +** geom_point() + geom_vline(xintercept = seq(from = 70,to = 80,by = 2) )

**ggplot(mydata100, aes(pretest, posttest)) +** geom_point() + geom_vline(xintercept = 70:80)

# Scatter plotting text labels

**qplot(pretest, posttest, geom = "text",** label = rownames(mydata100) )

**ggplot(mydata100,** aes(pretest, posttest, label = rownames(mydata100) ) ) + geom_text()

# Scatter plot matrix

plotmatrix( mydata100[3:8] )

# Small points & lowess fit. plotmatrix( mydata100[3:8], aes( size = 1 ) ) +

geom_smooth()

# Shape and gender fits. plotmatrix( mydata100[3:8],

aes( shape = gender ) ) + geom_smooth(method = lm)

### 4.1.7 Diagramas de cajas

# Box plot of one variable

**qplot(factor(""), posttest,** geom = "boxplot", xlab = "")

**ggplot(mydata100,** aes(factor(""), posttest) ) + geom_boxplot() + scale_x_discrete("")

# Box plot by group

qplot(workshop, posttest, geom = "boxplot" )

**ggplot(mydata100,** aes(workshop, posttest) ) + geom_boxplot()

# Box plot by group with jitter

# Wide jitter

**qplot(workshop, posttest,** geom = c("boxplot", "jitter") )

**ggplot(mydata100,** aes(workshop, posttest )) + geom_boxplot() + geom_jitter()

# Narrow jitter

**ggplot(mydata100,** aes(workshop, posttest )) + geom_boxplot() + geom_jitter(position = position_jitter(width = .1))

# Box plot for two-way interaction.

**qplot(workshop, posttest,** geom = "boxplot", fill = gender ) + scale_fill_grey(start = 0, end = 1)

**ggplot(mydata100,** aes(workshop, posttest) ) + geom_boxplot( aes(fill = gender), colour = "grey50") + scale_fill_grey(start = 0, end = 1)

# Error bar plot

**ggplot(mydata100,** aes( as.numeric(workshop), posttest ) ) + geom_jitter(size = 1,

position = position_jitter(width = .1) ) +

**stat_summary(fun.y = "mean",** geom = "smooth", se = FALSE) +

**stat_summary(fun.data = "mean_cl_normal",** geom = "errorbar", width = .2, size = 1)

### 4.1.8 Mapas geográficos

library("maps") library("ggplot2") myStates <- map_data("state") head(myStates) myStates[ myStates$region == "new york", ]

**qplot(long, lat, data = myStates, group = group,** geom = "path", asp = 1)

**ggplot(data = myStates, aes(long, lat, group = group) )+** geom_path() + coord_map()

myArrests <- USArrests head(myArrests)

myArrests$region <- tolower( rownames(USArrests) ) head(myArrests)

**myBoth <- merge(** myStates, myArrests, by = "region")

myBoth[1:4, c(1:5,8)]

myBoth <- myBoth[order(myBoth$order), ] myBoth[1:4, c(1:5,8)]

**qplot(long, lat, data = myBoth, group = group,** fill = Assault, geom = "polygon", asp = 1) + scale_fill_continuous(low = "grey80", high = "black")

**ggplot(data = myBoth,**

aes(long, lat, fill = Assault, group = group) )+

geom_polygon() + coord_map() + scale_fill_continuous(low = "grey80", high = "black")

### 4.1.9 Ejes logarítmicos

# Change the variables qplot( log(pretest), log(posttest) )

**ggplot(mydata100,** aes( log(pretest), log(posttest) ) ) + geom_point()

# Change axis labels

qplot(pretest, posttest, log = "xy")

**ggplot(mydata100, aes( x = pretest, y = posttest) ) +** geom_point() + scale_x_log10() + scale_y_log10()

# Change axis scaling

**qplot(pretest, posttest, data = mydata100) +** coord_trans(x = "log10", y = "log10")

**ggplot(mydata100, aes( x = pretest, y = posttest) ) +** geom_point() + coord_trans(x = “log10”, y = “log10”)

### 4.1.10 Relación de aspecto

# This forces x and y to be equal. qplot(pretest, posttest) + coord_equal()

# This sets aspect ratio to height/width. qplot(pretest, posttest) + coord_equal(ratio = 1/4)

### 4.1.11 Gráfico de barras multicuadro

grid.newpage() # clear page

# Sets up a 2 by 2 grid to plot into. pushViewport( viewport(layout = grid.layout(2, 2) ) )

# Bar plot dodged in row 1, column 1. myPlot <- ggplot(mydata100,

aes(gender, fill = workshop) ) +

geom_bar(position = “stack”) + scale_fill_grey(start = 0, end = 1) + opts( title = “position = stack ” )

**print(myPlot, vp = viewport(** layout.pos.row = 1, layout.pos.col = 1) )

# Bar plot stacked, in row 1, column 2. myPlot <- ggplot(mydata100,

aes(gender, fill = workshop) ) +

geom_bar(position = “fill”) + scale_fill_grey(start = 0, end = 1) + opts( title = “position = fill” )

**print(myPlot, vp = viewport(** layout.pos.row = 1, layout.pos.col = 2) )

# Bar plot dodged, given frames, # in row 2, columns 1 and 2. myPlot <- ggplot(mydata100,

aes(gender, fill = workshop) ) +

geom_bar(position = “dodge”) + scale_fill_grey(start = 0, end = 1) + opts( title = “position = dodge” )

**print(myPlot, vp = viewport(** layout.pos.row = 2, layout.pos.col = 1:2) )

### 4.1.12 Gráfico de dispersión multicuadro

# Clears the page grid.newpage()

# Sets up a 2 by 2 grid to plot into. pushViewport( viewport(layout = grid.layout(2,2) ) )

# Scatter plot of points myPlot <- qplot(pretest, posttest,main = “geom = point”) print(myPlot, vp = viewport(

layout.pos.row = 1, layout.pos.col = 1) )

**myPlot <- qplot( pretest, posttest,** geom = “line”, main = “geom = line” )

**print(myPlot, vp = viewport(** layout.pos.row = 1, layout.pos.col = 2) )

**myPlot <- qplot( pretest, posttest,** geom = “path”, main = “geom = path” )

**print(myPlot, vp = viewport(** layout.pos.row = 2, layout.pos.col = 1) )

**myPlot <- ggplot( mydata100, aes(pretest, posttest) ) +**

**geom_segment( aes(x = pretest, y = posttest,** xend = pretest, yend = 58) ) +

opts( title = “geom_segment example” )

**print(myPlot,** vp = viewport(layout.pos.row = 2, layout.pos.col = 2) )

### 4.1.13 Gráfico de dispersión para fluctuación (jitter) multicuadro

grid.newpage() pushViewport( viewport(layout = grid.layout(1, 2) ) )

# Scatterplot without myPlot <- qplot(q1, q4,

main = "Likert Scale Without Jitter")

**print(myPlot, vp = viewport(** layout.pos.row = 1, layout.pos.col = 1) )

**myPlot <- qplot(q1, q4,** position = position_jitter(width = .3, height = .3), main = "Likert Scale With Jitter")

**print(myPlot, vp = viewport(** layout.pos.row = 1, layout.pos.col = 2) )

## 4.2 Comparación detallada qplot vs ggplot

**qplot(pretest, posttest,** geom = c("point", "smooth"), method = "lm" )

# Or ggplot with default settings:

**ggplot(mydata100, aes(x = pretest, y = posttest) ) +** geom_point() + geom_smooth(method = "lm")

# Or with all the defaults displayed: ggplot() + layer(

data = mydata100, mapping = aes(x = pretest, y = posttest), geom = "point", stat = "identity"

) + layer(

data = mydata100, mapping = aes(x = pretest, y = posttest), geom = "smooth", stat = "smooth", method = "lm"

) + coord_cartesian()

# Regresión

```r
# free memory
rm(list = ls())
gc()
```

```r
year <- rep(2008:2010, each=4)
quarter <- rep(1:4, 3)
cpi <- c(162.2, 164.6, 166.5, 166.0,
         166.2, 167.0, 168.6, 169.5,
         171.0, 172.1, 173.3, 174.0)
plot(cpi, xaxt="n", ylab="CPI", xlab="")
# draw x-axis
axis(1, labels=paste(year,quarter,sep="Q"), at=1:12, las=3)
```

```r
cor(year,cpi)
cor(quarter,cpi)
```

```r
fit <- lm(cpi ~ year + quarter)
fit
```

```r
 cpi2011 <- fit$coefficients[1] + fit$coefficients[2]*2011 +
         fit$coefficients[3] * 1:4)
```

```r
attributes(fit)
fit$coefficients
```

```r
# differences between observed values and fitted values
residuals(fit)
summary(fit)
```

```r
## plot(fit)
```

```r
## The chunk above simply output code to document, and the results are produced by the chunk below.
layout(matrix(c(1,2,3,4),2,2)) # 4 graphs per page
plot(fit)
layout(matrix(1)) # change back to one graph per page
```

```r
library(scatterplot3d)
s3d <- scatterplot3d(year, quarter, cpi, highlight.3d=T, type="h", lab=c(2,3))
s3d$plane3d(fit)
```

```
data2011 <- data.frame(year=2011, quarter=1:4)
cpi2011 <- predict(fit, newdata=data2011)
style <- c(rep(1,12), rep(2,4))
plot(c(cpi, cpi2011), xaxt="n", ylab="CPI", xlab="", pch=style, col=style)
axis(1, at=1:16, las=3,
     labels=c(paste(year,quarter,sep="Q"), "2011Q1", "2011Q2", "2011Q3", "2011Q4"))
```

```
data("bodyfat", package="mboost")
myFormula <- DEXfat ~ age + waistcirc + hipcirc + elbowbreadth + kneebreadth
bodyfat.glm <- glm(myFormula, family = gaussian("log"), data = bodyfat)
summary(bodyfat.glm)
pred <- predict(bodyfat.glm, type="response")
```

```
plot(bodyfat$DEXfat, pred, xlab="Observed Values", ylab="Predicted Values")
abline(a=0, b=1)
```

# Análisis Multivariante

```r
# free memory
rm(list = ls())
gc()
```

```r
set.seed(8953)
```

```r
iris2 <- iris
iris2$Species <- NULL
 kmeans.result <- kmeans(iris2, 3)
```

```r
table(iris$Species, kmeans.result$cluster)
```

```r
plot(iris2[c("Sepal.Length", "Sepal.Width")], col = kmeans.result$cluster)
# plot cluster centers
points(kmeans.result$centers[,c("Sepal.Length", "Sepal.Width")], col = 1:3,
       pch = 8, cex=2)
```

```r
library(fpc)
pamk.result <- pamk(iris2)
# number of clusters
pamk.result$nc
# check clustering against actual species
table pamk.result$pamobject$clustering, iris$Species)
```

```r
layout(matrix(c(1,2),1,2)) # 2 graphs per page
plot(pamk.result$pamobject)
layout(matrix(1)) # change back to one graph per page
```

```r
pam.result <- pam(iris2, 3)
table pam.result$clustering, iris$Species)
```

```r
layout(matrix(c(1,2),1,2)) # 2 graphs per page
plot(pam.result)
layout(matrix(1)) # change back to one graph per page
```

```r
set.seed(2835)
```

```
idx <- sample(1:dim(iris)[1], 40)
irisSample <- iris[idx,]
irisSample$Species <- NULL
hc <- hclust(dist(irisSample), method="ave")
```

```
plot(hc, hang = -1, labels=iris$Species[idx])
# cut tree into 3 clusters
rect.hclust(hc, k=3)
groups <- cutree(hc, k=3)
```

```
library(fpc)
iris2 <- iris[-5] # remove class tags
ds <- dbscan(iris2, eps=0.42, MinPts=5)
# compare clusters with original class labels
table(ds$cluster, iris$Species)
```

```
plot(ds, iris2)
```

```
plot(ds, iris2[c(1,4)])
```

```
plotcluster(iris2, ds$cluster)
```

```
# create a new dataset for labeling
set.seed(435)
idx <- sample(1:nrow(iris), 10)
newData <- iris[idx,-5]
newData <- newData + matrix(runif(10*4, min=0, max=0.2), nrow=10, ncol=4)
# label new data
myPred <- predict(ds, iris2, newData)
# plot result
plot(iris2[c(1,4)], col=1+ds$cluster)
points(newData[c(1,4)], pch="*", col=1+myPred, cex=3)
# check cluster labels
table(myPred, iris$Species[idx])
```

# Series de Tiempo

```
# free memory
rm(list = ls())
gc()
```

```
a <- ts(1:30, frequency=12, start=c(2011,3))
print(a)
str(a)
attributes(a)
```

```
plot(AirPassengers)
```

```
# decompose time series
apts <- ts(AirPassengers, frequency=12)
f <- decompose(apts)
# seasonal figures
f$figure
plot(f$figure, type="b", xaxt="n", xlab="")
# get names of 12 months in English words
monthNames <- months(ISOdate(2011,1:12,1))
# label x-axis with month names
# las is set to 2 for vertical label orientation
axis(1, at=1:12, labels=monthNames, las=2)
```

```
plot(f)
```

```
fit <- arima(AirPassengers, order=c(1,0,0), list(order=c(2,1,0), period=12))
fore <- predict(fit, n.ahead=24)
# error bounds at 95% confidence level
U <- fore$pred + 2*fore$se
L <- fore$pred - 2*fore$se
ts.plot(AirPassengers, fore$pred, U, L, col=c(1,2,4,4), lty = c(1,1,2,2))
legend("topleft", c("Actual", "Forecast", "Error Bounds (95% Confidence)"),
       col=c(1,2,4), lty=c(1,1,2))
```

```
library(dtw)
idx <- seq(0, 2*pi, len=100)
a <- sin(idx) + runif(100)/10
b <- cos(idx)
```

```
align <- dtw(a, b, step=asymmetricP1, keep=T)
dtwPlotTwoWay(align)
```

```
sc <- read.table("./data/synthetic_control.data", header=F, sep="")
# show one sample from each class
idx <- c(1,101,201,301,401,501)
sample1 <- t(sc[idx,])
plot.ts(sample1, main="")
```

```
set.seed(6218)
```

```
n <- 10
s <- sample(1:100, n)
idx <- c(s, 100+s, 200+s, 300+s, 400+s, 500+s)
sample2 <- sc[idx,]
observedLabels <- rep(1:6, each=n)
# hierarchical clustering with Euclidean distance
hc <- hclust(dist(sample2), method="average")
plot(hc, labels=observedLabels, main="")
# cut tree to get 6 clusters
rect.hclust(hc, k=6)
memb <- cutree(hc, k=6)
table(observedLabels, memb)
```

```
library(dtw)
distMatrix <- dist(sample2, method="DTW")
hc <- hclust(distMatrix, method="average")
plot(hc, labels=observedLabels, main="")
# cut tree to get 6 clusters
rect.hclust(hc, k=6)
memb <- cutree(hc, k=6)
table(observedLabels, memb)
```

```
classId <- rep(as.character(1:6), each=100)
newSc <- data.frame(cbind(classId, sc))
library(party)
ct <- ctree(classId ~ ., data=newSc,
            controls = ctree_control(minsplit=30, minbucket=10, maxdepth=5))
pClassId <- predict(ct)
table(classId, pClassId)
# accuracy
(sum(classId==pClassId)) / nrow(sc)
plot(ct, ip_args=list(pval=FALSE), ep_args=list(digits=0))
```

```
library(wavelets)
wtData <- NULL
for (i in 1:nrow(sc)) {
  a <- t(sc[i,])
  wt <- dwt(a, filter="haar", boundary="periodic")
  wtData <- rbind(wtData, unlist(c(wt@W, wt@V[[wt@level]])))
}
wtData <- as.data.frame(wtData)
wtSc <- data.frame(cbind(classId, wtData))
```

```
# build a decision tree with DWT coefficients
ct <- ctree(classId ~ ., data=wtSc,
            controls = ctree_control(minsplit=30, minbucket=10, maxdepth=5))
pClassId <- predict(ct)
```

```
table(classId, pClassId
 sum(classId==pClassId) / nrow wtSc)
plot(ct, ip_args=list(pval=FALSE), ep_args=list(digits=0))
```

```
# fix seed to get a fixed result in the chunk below
set.seed(100)
```

```
k <- 20
# create a new time series by adding noise to time series 501
newTS <- sc[501,] + runif(100)*15
distances <- dist(newTS, sc, method="DTW")
s <- sort(as.vector(distances), index.return=TRUE)
# class IDs of k nearest neighbors
table(classId[s$ix[1:k]])
```

CAPÍTULO 8

# Reportes dinámicos

This post examines the features of R Markdown using knitr in Rstudio 0.96. This combination of tools provides an exciting improvement in usability for reproducible analysis. Specifically, this post

1. discusses getting started with R Markdown and knitr in Rstudio 0.96;

#. provides a basic example of producing console output and plots using R Markdown; #. highlights several code chunk options such as caching and controlling how input and output is displayed; #. demonstrates use of standard Markdown notation as well as the extended features of formulas and tables; and #. discusses the implications of R Markdown.

This post was produced with R Markdown. The source code is available here as a gist. The post may be most useful if the source code and displayed post are viewed side by side. In some instances, I include a copy of the R Markdown in the displayed HTML, but most of the time I assume you are reading the source and post side by side.

## 8.1 Getting started

To work with R Markdown, if necessary:

> Install R Install the lastest version of RStudio (at time of posting, this is 0.96) Install the latest version of the knitr package: install.packages("knitr")

To run the basic working example that produced this blog post:

> Open R Studio, and go to File - New - R Markdown
>
> If necessary install ggplot2 and lattice packages: install.packages("ggplot2"); install.packages("lattice")
>
> Paste in the contents of this gist (which contains the R Markdown file used to produce this post) and save the file with an .rmd extension
>
> Click Knit HTML

## 8.2 Prepare for analyses

```
'{r } set.seed(1234) library(ggplot2) library(lattice) '
```

## Basic console output To insert an R code chunk, you can type it manually or just press *Chunks - Insert chunks* or use the shortcut key. This will produce the following code chunk:

```{r}
```

Pressing tab when inside the braces will bring up code chunk options.

The following R code chunk labelled *basicconsole* is as follows:

```
'{r } x <- 1:10 y <- round(rnorm(10, x, 1), 2) df <- data.frame(x, y)
df '
```

The code chunk input and output is then displayed as follows:

```
'{r basicconsole} x <- 1:10 y <- round(rnorm(10, x, 1), 2) df <-
data.frame(x, y) df '
```

## Plots Images generated by *knitr* are saved in a figures folder. However, they also appear to be represented in the HTML output using a [data URI scheme]( http://en.wikipedia.org/wiki/Data_URI_scheme). This means that you can paste the HTML into a blog post or discussion forum and you don't have to worry about finding a place to store the images; they're embedded in the HTML.

### Simple plot Here is a basic plot using base graphics:

```
'{r } plot(x) '
```

```
'{r simpleplot} plot(x) '
```

Note that unlike traditional Sweave, there is no need to write *fig=TRUE*.

### Multiple plots Also, unlike traditional Sweave, you can include multiple plots in one code chunk:

```
    '{r } boxplot(1:10~rep(1:2,5)) plot(x, y) '
```

```
'{r multipleplots} boxplot(1:10~rep(1:2,5)) plot(x, y) '
```

### *ggplot2* plot Ggplot2 plots work well:

```
'{r ggplot2ex} qplot(x, y, data=df) '
```

### *lattice* plot As do lattice plots:

```
'{r latticeex} xyplot(y~x) '
```

Note that unlike traditional Sweave, there is no need to print lattice plots directly.

## R Code chunk features ### Create Markdown code from R The following code hides the command input (i.e., *echo=FALSE*), and outputs the content directly as code (i.e., *results=asis*, which is similar to *results=tex* in Sweave).

```
    '{r , results='asis', echo=FALSE} cat("Here are some dot
    points\n\n") cat(paste("* The value of y[", 1:3, "] is ",
    y[1:3], sep=, collapse="\n")) '
```

```
'{r dotpointprint, results='asis', echo=FALSE} cat("Here are some
dot points\n\n") cat(paste("* The value of y[", 1:3, "] is ", y[1:3],
sep=, collapse="\n")) '
```

**### Create Markdown table code from R** `'{r , results='asis', echo=FALSE} cat("x | y",
-- | ---", sep="\n") cat(apply(df, 1, function(X) paste(X, collapse="|
")), sep = "\n") '`

```
'{r createtable, results='asis', echo=FALSE} cat("x | y", -- | ---",
sep="\n") cat(apply(df, 1, function(X) paste(X, collapse="| ")), sep =
"\n") '
```

### Control output display The folllowing code supresses display of R input commands (i.e., *echo=FALSE*) and removes any preceding text from console output (*comment=""*; the default is *comment="##"*).

```
`{r echo=FALSE, comment=, echo=FALSE} head(df) `
```

```
`{r echo=FALSE, comment=, echo=FALSE} head(df) `
```

### Control figure size The following is an example of a smaller figure using *fig.width* and *fig.height* options.

```
`{r , fig.width=3, fig.height=3} plot(x) `
```

```
`{r smallplot, fig.width=3, fig.height=3} plot(x) `
```

### Cache analysis Caching analyses is straightforward. Here's example code. On the first run on my computer, this took about 10 seconds. On subsequent runs, this code was not run.

If you want to rerun cached code chunks, just [delete the contents of the *cache* folder](http://stackoverflow.com/a/10629121/180892)

```
```{r , cache=TRUE} for (i in 1:5000) {
    lm((i+1)~i)
```

## Basic markdown functionality For those not familiar with standard [Markdown](http://daringfireball.net/projects/markdown/), the following may be useful. See the source code for how to produce such points. However, RStudio does include a Markdown quick reference button that adequatly covers this material.

### Dot Points Simple dot points:

- Point 1
- Point 2
- Point 3

and numeric dot points:

1. Number 1
2. Number 2
3. Number 3

and nested dot points:

- **A**
    - A.1
    - A.2
- **B**
    - B.1
    - B.2

### Equations Equations are included by using LaTeX notation and including them either between single dollar signs (inline equations) or double dollar signs (displayed equations). If you hang around the Q&A site [CrossValidated](http://stats.stackexchange.com) you'll be familiar with this idea.

There are inline equations such as $y_i = alpha + beta x_i + e_i$.

And displayed formulas:

$$frac{1}{1+exp(-x)}$$

knitr provides self-contained HTML code that calls a Mathjax script to display formulas. However, in order to include the script in my blog posts I [took the script](https://gist.github.com/2716053) and incorporated it into my blogger
```

template. If you are viewing this post through syndication or an RSS reader, this may not work. You may need to view this post on my website.

### Tables Tables can be included using the following notation

A | B | C — | — | — 1 | Male | Blue 2 | Female | Pink

### Hyperlinks

- If you like this post, you may wish to subscribe to [my RSS feed](http://feeds.feedburner.com/jeromyanglim).

### Images Here's an example image:

![image from redmond barry building unimelb](http://i.imgur.com/RVNmr.jpg)

### Code Here is Markdown R code chunk displayed as code:

```
    `{r} x <- 1:10 x `
```

And then there's inline code such as *x <- 1:10*.

CAPÍTULO 9

# Índices y tablas

- *genindex*
- *modindex*
- *search*