Programación de Aplicaciones para Internet y la Nube 2016-I

Informe de lectura: Dynamic Documents with R and knitr : Cross Reference "Documentos Dinámicos con R y knitr: Referencias Cruzadas"

Wilson Ricardo López Sánchez
Fecha de evaluación 1 de abril del 2016

Resumen:

El capítulo de referencias cruzadas habla sobre la utilización de códigos de R a través de un documento externo como es el caso de html, latex entre otros, por medio de paquete de R llamada Knitr, entre las ventajas que nos ofrece las referencias cruzadas están:

- la utilización de código a través de todo el documento, ejecutándolo y mostrando los resultados en el mismo
- reutilización de diferentes códigos a través de todo el documento.
- Utilización de códigos dentro de otros códigos por medio de etiquetas
- También es posible llamar códigos guardados que se encuentran dentro de la misma carpeta del documento principal.
- La asignación de diferentes etiquetas a diferentes partes de un código que se encuentra guardado, para ser utilizadas a lo largo de todo el código.
- La utilización de un documento principal y de códigos hijos para así tener una mejor organización del documento principal
- La utilización del encabezado del documento principal para compilar códigos que ni tienen encabezado o códigos que no fueron desarrollados para el documento principal si no que son totalmente apartes.

Para la demostración y utilización de las características mencionadas anteriormente se utilizará Rstudio tanto para la compilación de código R y del documento latex.

Contribución al desarrollo de la programación literaria: Referencias Cruzadas

A través de la programación literaria es posible reutilizar segmentos de códigos sin necesidad de volverlos a escribir, esto resulta muy útil para lograr un documento bien organizado sin redundancia de código. Como primer paso es necesario a ver instalado en el Rstudio el paquete Knitr y en Tools¿global option en la sección Sweave seleccionar en Weave Rnw files using Knitr, aceptamos y procedemos a abrir un archivo nuevo R Sweve el cual tendrá por defecto la siguiente estructura:

```
documentclass{article}
\begin{document}
Contenido del documento
\end{document}
```

Observamos que es la misma estructura básica que un documento latex una clase de articulo y el comienzo y fin del mismo.
Utilizando el paquete ggplot2 y el dataset de diamonds se realizara un ejemplo sencillo de la utilización de las referencias cruzadas:

Para ejecutar código R en un documento. Rnw es necesario utilizar la sintaxis:

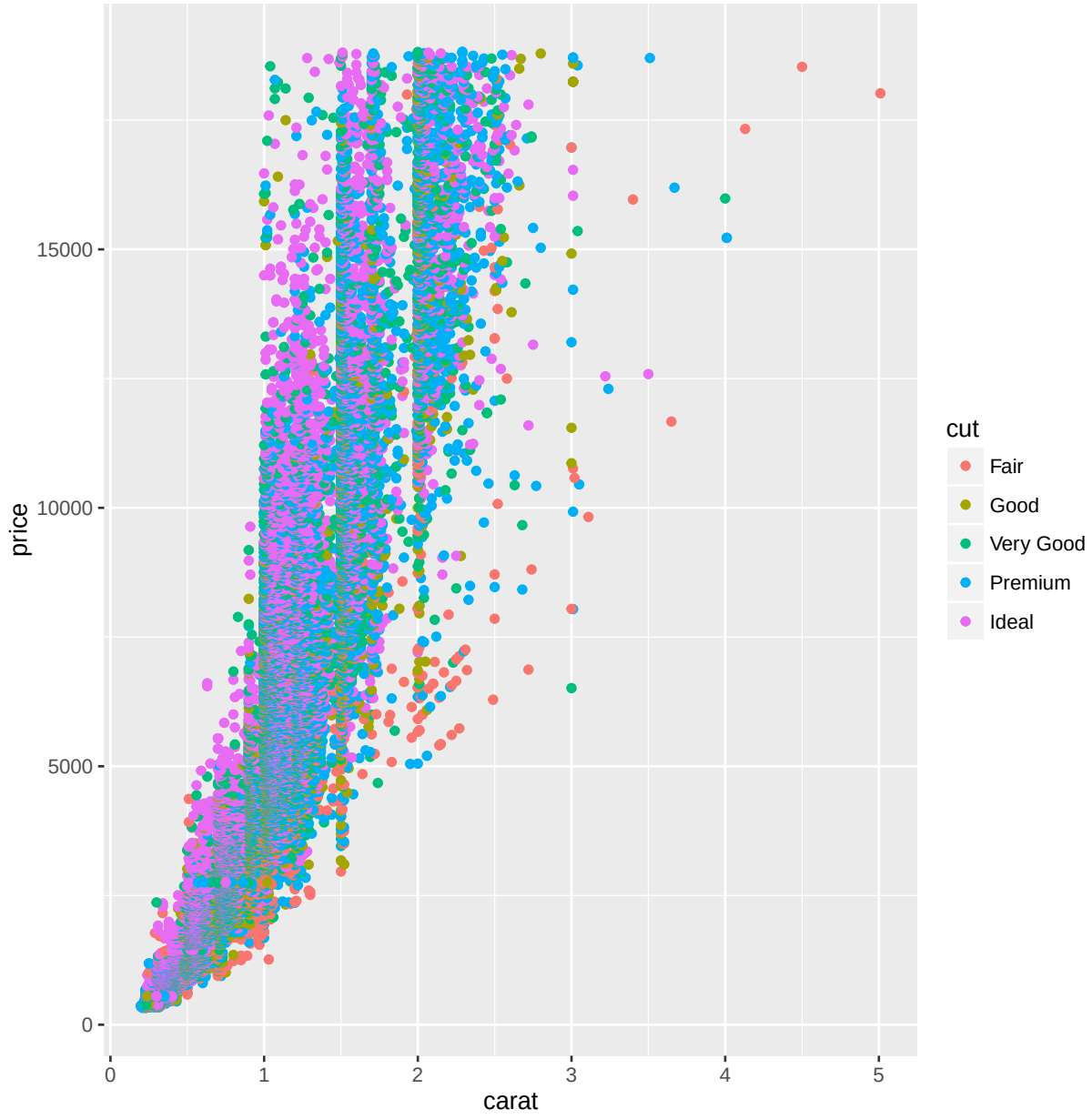
```
<<>>=
codigoR
@
```

Para mayor facilidad podemos usar la combinación de teclas *control+alt+i* para generar la sintaxis anterior
Después de instalar el paquete qqplot2 podemos cagar la librería y el siguiente ejemplo:

```
<<>>=
library('ggplot2')
qplot(carat, price, data = diamonds, color = cut)
@
```

Con lo que obtenemos:

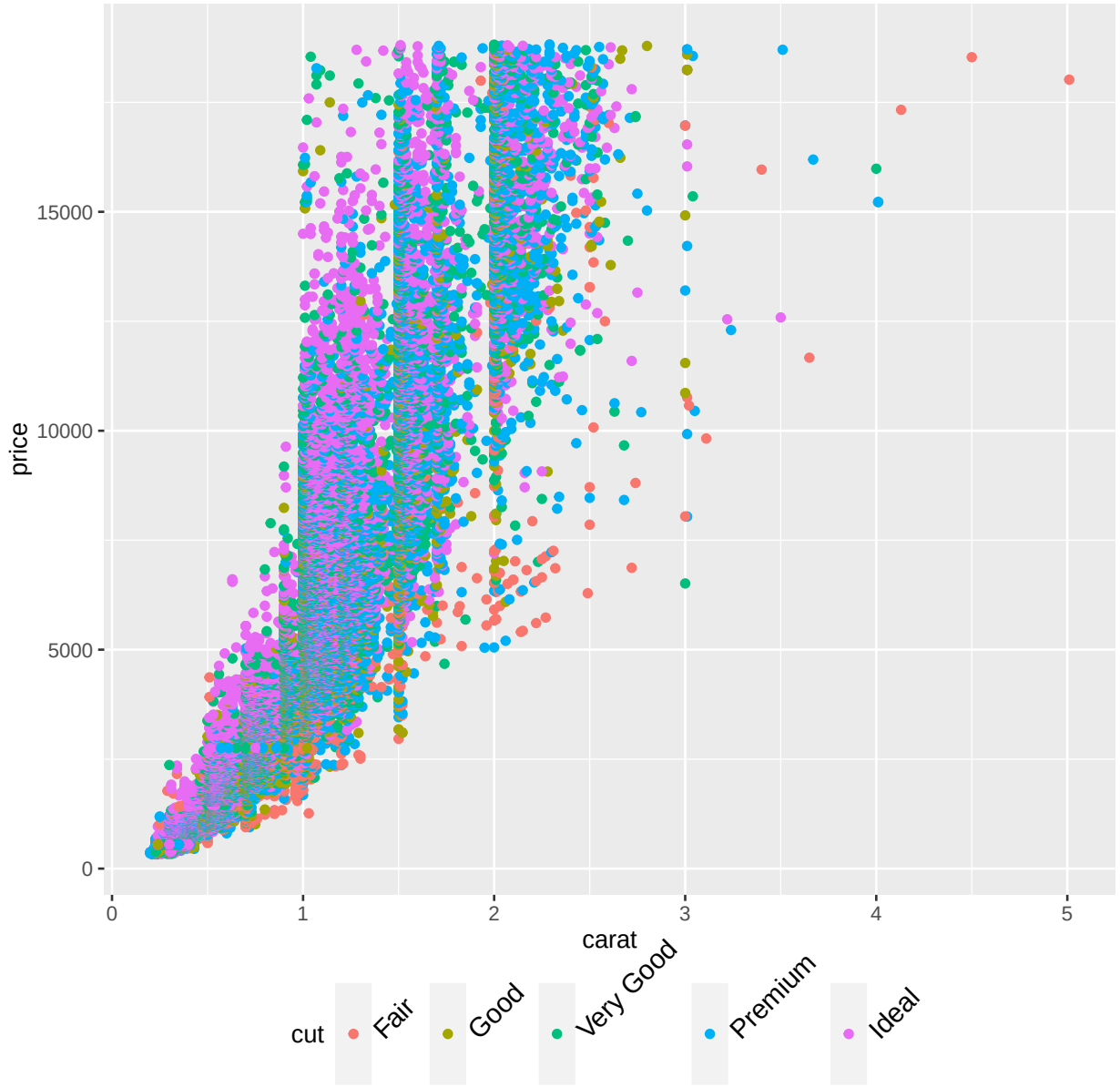
```
library('ggplot2')
qplot(carat, price, data = diamonds, color = cut)
```



Ahora le agregamos al código anterior un tema de ggplot2:

<<>>=
library('*ggplot*2')
qplot(*carat*, *price*, *data* = *diamonds*, *color* = *cut*)+
theme(*legend.text* = *element*$_t$*ext*(*size* = 12, *angle* = 45))+
theme(*legend.position* = "*bottom*")
@
Con lo que obtenemos

```
library('ggplot2')
qplot(carat, price, data = diamonds, color = cut) +
theme(legend.text = element_text(size = 12, angle = 45)) +
theme(legend.position = "bottom")
```



Se observa que la segunda grafica cambia con respecto a la primera, si se requiere que todas la graficas tengan el mismo tema se podría copiar el segmento de código que define el tema en todas la graficas pero en caso de querer cambiar alguna característica del tema sería necesario cambiar este parámetro en todas las gráficas con la ayuda de referencias cruzadas esto no es necesario. Para poder simplificar dicha tarea procedemos a darle un nombre o etiqueta a un segmento de código que contenga las características de la gráfica:

$<< temagraficas1, eval = FALSE >>=$
$theme(legend.text = element_text(size = 12, angle = 45))+$
$theme(legend.position = "bottom")$
@

Ahora podemos realizar nuestra grafica de la siguiente manera:

$<<>>=$
$qplot(carat, price, data = diamonds, color = cut)+$

$<< temagraficas1 >>$
@

Como se observa de esta forma R llama el segmento de código con la etiqueta «temagraficas1» y ejecuta todos los comandos que contiene dicho segmento.

1. **referencias por segmentos**
   Como se mencionó anteriormente es posible utilizar segmentos de códigos a través del documento las veces que se requiera por medio de etiquetas.

   a) referencias por segmentos embebidas
   Es posible llamar segmentos de códigos dentro de otros segmentos de código por medio de etiquetas, en el siguiente ejemplo utilizaremos la función runif la cual es una distribución uniforme es equiprobable para todos los valores incluidos en un intervalo dado: $<< A >>=$
   $x < -runif(1, 0, 10)$
   @
   $<< B >>=$
   x
   $<< A >>$
   x
   @
   Con lo que obtenemos:

```
x <- runif(1,0,10)
```

```
x

## [1] 0.9779362

x <- runif(1,0,10)
x

## [1] 6.574535
```

   Nos damos cuenta que al llamar el segmento A en el segmento B, es como si se copiara todo el contenido de A donde se encuentra la etiqueta $<< A >>$

   b) reutilización de todo un segmento de código
   En el caso de querer utilizar todo un segmento de código existen dos posibilidades:

   1) Usar la misma etiqueta dejándola vacía.

   $<< chunkA, eval = FALSE >>=$
   $x < -5 * 6$
   @
   $<< chunkA, eval = TRUE >>=$
   @
   Lo que da como resultado:

```
x <- 5*6
```

```
x <- 5*6
```

   El inconveniente con este código consiste en que no se pueden almacenar los dos segmentos en cache, en el ejemplo anterior el parámetro eval hace referencias que la segunda vez que utilizamos el segmento de código tiene en cuenta el resultado anterior del mismo.

2) Con la opción ref.label, la cual se utiliza en este caso para unir dos trozos de códigos diferentes en este caso generamos con código C apartir del os códigos A y B:

$<< A >>=$
$x < -rnorm(1)$
@
$<< B >>=$
$y < -x + 2$
@
$<< C, ref.label = c('A','B') >>=$
@
Lo que da como resultado:
codigo A1

```
x <- rnorm(1)
```

codigo B1

```
y <- x + 2
```

codigo C

```
x <- rnorm(1)
y <- x + 2
```

2. **Exteriorización de código**

Para obtener un documento mucho más ordenado y estructurado es mejor tener los segmentos de código de R en documentos aporte para esta tarea knitr por medio del script $read\_chunk()$ es posible. Este script se puede utilizar de dos formas usando etiquetas en el guion para separar los segmentos de código o especificar que código ejecutar tomando el número de la línea.

a) segmentos por etiqueta
Los comandos de R también utilizan etiquetas de la forma $\#\# ---- nombre ----$, es importante denotar que todos los documentos con los cuales se van a trabajar en el documento principal tiene que estar todos en la misma carpeta. se toma como ejemplo el siguiente código de R el cual guardaremos con el nombre de ejemplo1.R

$\#\# ----D----$
$m < -2$
$n < -5$
m
n

para llamar el código de R en nuestro documento lo primero es ejecutar el script read_chunk de la siguiente manera:

$<<>>=$
$read\_chunk("ejemplo1.R")$
@

```
read_chunk("ejemplo1.R")
```

Posteriormente ya podemos llamar el código por medio de su etiqueta $<< D >>=$
@

```
    m <- 2
    n <- 5
    m
```

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

```
    n
```

```
## [1] 5
```

b) segmentos de códigos basados en líneas

read_chunk() también permite asignar a líneas de código de un archivo de R diferentes etiquetas para ser utilizadas en el documento. Como ejemplo guardamos los numero del 1 al 12 en un archivo con el nombre ejemplo2.R

1
2
4
5
6
7
8
9
10
12

Vamos a tomar la línea 1-4,5-8 y 11-12 del archivo de R anterior con ellos formaremos los segmentos de codigo con las etiquetas E, F y G, dicho proceso se realiza de la siguiente manera:

$read\_chunk(\text{"}ejemplo2.R\text{"}, labels = c(\text{"}E\text{"}, \text{"}F\text{"}, \text{"}G\text{"}), from = c(1, 5, 11), to = c(4, 8, 12))$

```
read_chunk("ejemplo2.R", labels = c("E", "F", "G"), from = c(1,5, 11), to = c(4,8,12))
```

asi podemos llamar cualquiera de los tres segmentos de código en cualquier momento:

<< *E* >>=
@
<< *F* >>=
@
<< *G* >>=
@
para E:

```
1
```

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

```
2
```

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

```
3
```

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

```
4
```

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

para F:

```
5
```

```
## [1] 5
```

```
6
```

```
## [1] 6
```

```
7
```

```
## [1] 7
```

```
8
```

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

para G:

```
11
```

```
## [1] 11
```

```
12
```

```
## [1] 12
```

3. **Documentos hijos**

De Forma equivalente que se realiza en latex donde se puede tener un archivo principal y muchos documentos hijos que son llamados y compilados por el archivo principal, **knitr** permite este mismo tratamiento.

a) entradas de documentos hijos
Como ejemplo se toma como documento principal este mismo documento llamado referenciascruzadas.Rnw y como documento hijo crearemos un documento llamado cap1.Rnw con el siguiente código:

$<< H1 >>=$
$y < -sum(x^2)$
@

incluimos el documento hijo de la siguiente forma:
$<< H, child =' cap1.Rnw' >>=$
@

```
y <- sum(x^2)
```

Como se observa anteriormente el documento hijo utiliza una variable x para ello llamamos el segmento de codigo «A» que genera un numero aleatorio.

```
x <- runif(1,0,10)
```

Y si llamamos a H la cual utiliza la variable x, y da como resultado la variable y obtenemos como respuesta de y=1.668906

Cabe denotar que antes de llamar el segmento de codigo H ya tenemos la respuesta y esto debido que al momento de incluir el documento hijo este es compildo.
LaTeX

b) modo independiente

Cuando se trabaja con LaTeXlos documentos de LaTeXtiene un encabezado que les indica que tipo de documento es los paquetes que utiliza entre otras características generales que va tener el documento. Los documentos hijos y códigos de R que se trabajan en Rstudio muchas veces no tiene el encabezado de LaTeXpara poder ser compilados o que son códigos que se generaron de forma ajena al documento pero que queremos adjuntarlos, para estos casos es posible leer y compilar los códigos hijos con el elcabezado del documento principal.

La función set_parent() de knitr aplicada en el documeto hijo toma el encabezado del documento principal de esta forma es posible compilar el codigo hijo sin necesidad de compilar todo el documento principal. Tomando el documento cap1.Rnw si agregamos el siguiente comando al documento hijo:

$<< parent, include = FALSE >>=$
$set_parent("referenciascruzadas.Rnw")$
@
Con lo cual el nuevo documento hijo nos quedaría de la siguiente forma:

$<< parent, include = FALSE >>=$
$set_parent("referenciascruzadas.Rnw")$
@
$<< H1 >>=$
$y < -sum(x^2)$
@

Al compilar el documento hijo observamos que compila sin problemas y con las mismas características del documento principal.