Resumen

Estas notas constituyen un resumen del Taller, *Visualizando grandes datos con Apache Zeppelin*, impartido en el II Curso de verano en la UPV: Transparencia a través de los datos celebrado en la Universitat Politècnica de València entre el 5 al 7 de septiembre de 2017. El taller consistió en la instalación y utilización del blog de notas web Apache Zeppelin para la visualización de datos de la plataforma AirBnb de la ciudad de Barcelona.

Valencia, Septiembre de 2017
Universitat Politècnica de València

## 1. Descarga e instalación del programa y de la base de datos

**a)** En primer lugar descargamos Apache Zeppelin de la dirección

`https://zeppelin.apache.org/download.html`

Elegimos el primer enlace —que es una versión que contiene diferentes intérpretes preconfigurados— que nos lleva a diferentes "mirrors" para la descarga. El archivo descargado viene comprimido —en formato tgz—. Si trabajas en Windows y no tienes descompresor para este tipo de archivos puedes usar `http://www.7-zip.org/download.html`.

Para poder utilizar el programa lo único que hay que hacer es descomprimir la carpeta (donde queramos, por ejemplo en el Escritorio). Para utilizar el programa accedemos a una ventana de terminal (en Windows se llama `cmd`) y montamos el servidor: para ello vamos al escritorio `cd Desktop` y luego a la carpeta `cd zeppelin-0.7.2-bin-all`. Si trabajas con MacOs o Linux simplemente escribe:

```
bin/zeppelin-daemon.sh start
```

Si trabajas en Windows todavía tienes que ir a la carpeta bin mediante `cd bin` y después escribir:

```
zeppelin.cmd start
```

Si todo ha ido bien podrás entrar al blog de notas escribiendo en cualquier navegador: `localhost:8080`.

**b)** Ahora vamos a descargar el archivo correspondiente a la base de datos relativa al AirBnb de Barcelona. Para ello accede a: `http://insideairbnb.com/get-the-data.html` Descarga el csv de Barcelona llamado `listings.csv` (puedes copiarlo directamente en el escritorio).

**c)** En tercer lugar vamos a instalar el software `R` que es un programa de carácter estadístico que nos va a permitir visualizar y tratar los datos. La instalación de este programa es standard. Lo podemos descargar a través del enlace `https://cloud.r-project.org/` y ejecutar el programa de instalación. Para este proyecto vamos a necesitar tres librerías que se descargan automáticamente abriendo `R` y escribiendo

```
install.packages("chron")
install.packages("ggplot2")
install.packages("googleVis")
```

## 2. Primer blog de notas usando R y SQL

Accedemos mediante el navegador y creamos un nueva nota (pulsa sobre **Create new note** y ponle un nombre —como intérprete puedes dejar el que aparece por defecto **spark**—).

**a)** Para comenzar puedes escribir algo de código usando **Markdown** que te permite escribir texto y obtener como salida el texto en formato **html** —puedes consultar la sintaxis de Markdown en **https://markdown.es/sintaxis-markdown/**—. Escribe el código que hay a continuación —la primera linea, **%md**, indica a Apache Zeppelin que el intérprete es Markdown.

```
%md
## Primera parte
```
Introducción y carga de datos con 'R'. Primeras búsquedas con 'R' y 'SQL'
```
```

Una vez escrito pincha sobre el icono ángulo (▷) para ejecutar. En el botón de ajustes de cada ventana puedes acceder al formato de la ventana (ancho, posición, título, ver o no el código, ver o no el resultados,...).

**b)** Ahora vamos a crear una nueva ventana en la que cargaremos los paquetes que hemos instalado de **R** y trataremos el archivo **csv** descargado.

```
%r
library(chron)
library(ggplot2)
setwd("~/Desktop/airbnbBarcelona")
# Leemos el archivo listings.csv
airbnb_Barcelona_data <- read.csv(file = "listings.csv")
# Los grabamos como  RData
save(airbnb_Barcelona_data, file="airbnb_Barcelona_data.rdata")
# Cargamos el RData
load("airbnb_Barcelona_data.rdata")
# Comprobamos que es correcto mirando, por ejemplo, el número de filas
   del archivo o las cabeceras
n_filas<-nrow(airbnb_Barcelona_data)
n_filas
```

Como de costumbre, una vez escrito el código, ejecutamos la ventana (pulsando en ▷). Observa que, en este caso, como la primera linea del código es **%r** entonces el intérprete usado por Apache Zeppelin será **R**. Para ver la estructura de los datos podemos crear una nueva ventana con el código:

```
%r
str(airbnb_Barcelona_data)
```

Las instrucciones **summary(airbnb_Barcelona_data)** ó **head(airbnb_Barcelona_data)** proporcionan oro tipo de información.

**c)** Las primeras visualizaciones las vamos a hacer utilizando **SQL**. En primer lugar creamos la base de datos escribiendo el siguiente código:

```
%spark.r
datos <- createDataFrame(sqlContext,airbnb_Barcelona_data)
registerTempTable(datos, "AIRBNB_BARC")
```

La primera búsqueda (y visualización) que vamos a hacer es saber qué tipos de habitaciones hay (eso está en la columna **room_type** del **csv** (que en la tabla tendrá la cabecera Tipo) y cuántas habitaciones hay (esto se cuenta con **count(*)** y a la columna de los resultados la llamamos Cantidad). Seleccionamos (con **SELECT** estos datos de nuestra base de datos (que se llama **AIRBNB_BARC**) y le decimos que las agrupo según la columna **room_type**:

```
%sql
SELECT room_type Tipo, count(*) Cantidad
FROM AIRBNB_BARC
GROUP BY room_type
```

Tras ejecutar el intérprete —en este caso `SQL`— tendremos la opción de ver la tabla de resultados de la búsqueda así como diferentes interpretaciones gráficas de los resultados como, por ejemplo, histogramas. Las gráficas presentadas son dinámicas en el sentido de que se pueden actualizar en función de los parámetros de búsqueda (en el caso de la búsqueda anterior Tipo y Cantidad). Para ver cómo hacer esto vamos a realizar una búsqueda con más parámetros.

Vamos ahora a utilizar tres columnas: con la de precio —llamada `price`— vamos a calcular el mínimo —columna a la que vamos a llamar `Precio_Min`—, el máximo —que será `Precio_Max`— y la media —que pondremos `Precio`—. Esto lo vamos a hacer para cada uno de los barrios de Barcelona —dato que está almacenado en la columna `neighbourhood_group`—. Finalmente vamos a contar el número de habitaciones en cada barrio mediante `count(*)` —llamando a dicha columna `Cantidad`—. Para hacer todo esto escribe y ejecuta el siguiente código en una nueva ventana:

```
%sql
SELECT Min(price) Precio_Min, MAX(price) Precio_Max, AVG(price) Precio,
    neighbourhood_group Barrio, count(*) Cantidad
FROM AIRBNB_BARC
GROUP BY neighbourhood_group
```

Como en el caso anterior podemos ver la tabla y visualizaciones pero en este caso tenemos cinco parámetros —uno por cada uno de las cinco columnas que tiene la tabla—. Vamos a ver diferentes cosas. Para ello pincha en el segundo icono —el del histograma— y después en `settings`. Verás como entonces aparecen las variables que puedes arrastrar a tres cuadros:

* `Keys`. Sería la variable que vas a poner en el eje de las `x`. Prueba a poner, por ejemplo el Barrio.
* `Groups`. Este te permite agrupar para que te aparezca una leyenda (o varias). Prueba a dejarlo en blanco o a poner la Cantidad.
* `Value`. Esto sería el valor del eje de la `y`. Prueba ahora a poner Precio —fíjate que puedes además hacer una operación sobre esta columna—.

Fíjate que si has seleccionado como grupo el Barrio puedes seleccionar para que en la gráfica veas los datos de uno o varios barrios. Puedes ver los otros formatos de visualización. En el caso de la nube de puntos en lugar de tres parámetros puedes seleccionar uno adicional —el campo `size`— que hace referencia al tamaño de cada punto. Prueba por ejemplo con

* `xAxis`: Barrio
* `yAxis`: Precio
* `group`: Barrio
* `size`: Precio

Uno de las opciones de Apache Zeppelin es la posibilidad de compartir. Para ello desde la visualización del histograma, por ejemplo, selecciona la opción `Link this paragraph` que aparece en el ajuste de la ventana. Verás como se ha abierto otra pestaña en el navegador dónde sólo se ve la parte gráfica. Observa qué pasa si ahora cambias —en la ventana original— la opción de visualización a , por ejemplo, la nube de puntos —pincha la nube se puntos y ahora regresa a la pestaña que se había abierto anteriormente—.

**c)** Las visualizaciones pueden depender también de campos dinámicos como, por ejemplo, selectores o campos de texto. Para ello puedes probar con el código:

```
%sql
SELECT ${Selección= neighbourhood_group, neighbourhood_group|
    neighbourhood|price|minimum_nights},
count(1) Cantidad
FROM AIRBNB_BARC
GROUP BY ${Selección= neighbourhood_group, neighbourhood_group|
    neighbourhood|price|minimum_nights}
```

o también con:

```
%sql
SELECT number_of_reviews Revisiones, price Precio, neighbourhood_group
   Barrio
FROM AIRBNB_BARC
WHERE number_of_reviews > "${Revisiones=0}"
    AND price < "${Precio_Max=100}"
```

## 3. Visualizaciones más avanzadas con googleVis de R

Vamos a crear un nuevo blog de notas (hazlo como al principio de la sección anterior). El paquete `googleVis` de `R` permite hacer otros tipos de visualizaciones —unas similares como tablas e histogramas y otras diferentes como mapas de geolocalización—. Muchas de estas visualizaciones las puedes encontrar en:

`https://cran.r-project.org/web/packages/googleVis/vignettes/googleVis_examples.html`

La primera observación es que para que estas visualizaciones funcionen es necesario tener conexión de Internet.

**a)** Como en el blog anterior vamos a crear una ventana con los paquetes y manipulaciones de nuestro archivo `csv`.

```
%r
library(chron)
library(ggplot2)
library(googleVis)
setwd("~/Desktop/airbnbBarcelona")
# Leemos el archivo listings.csv
airbnb_Barcelona_data <- read.csv(file = "listings.csv")
# Los grabamos como  RData
save(airbnb_Barcelona_data, file="airbnb_Barcelona_data.rdata")
# Cargamos el RData
load("airbnb_Barcelona_data.rdata")
# Comprobamos que es correcto mirando, por ejemplo, el número de filas
   del archivo o las cabeceras
n_filas<-nrow(airbnb_Barcelona_data)
n_filas
```

En primer lugar vamos a ver cómo imprimir una tabla con este paquete. El código de `R` sería:

```
%r
# Primero extraemos de nuestra base la columna neighbourhood_group
Barrio<-airbnb_Barcelona_data$neighbourhood_group
# Creamos ahora la tabla de frecuencia  de los barrios (cuántos pisos hay
    en cada barrio)
tabla_barrios<-table(Barrio)
# Y el correspondiente dataframe
tb<-data.frame(tabla_barrios)
# Y ahora vemos la tabla
LocTable <- gvisTable(tb, options=list(page='enable'))
print(LocTable)
```

Para crear la tabla usamos `gvisTable` y la visualizamos utilizando `print`.

Podemos ahora crear una nueva ventana donde veamos una línea —para esto se usa `gvisLineChart`—, el gráfico circular —para este `gvisPieChart` y el histograma —`gvisBarChart` en este último caso—. Para ello crea y ejecuta una nueva ventana con el código:

```
%r
Line<- gvisLineChart(tb,options=list(width=500, height=300))
Bar <- gvisBarChart(tb,options=list(width=805, height=400))
Pie <- gvisPieChart(tb,options=list(width=505, height=300))
PieBar <- gvisMerge(Line,Pie,horizontal=TRUE,tableOptions="bgcolor=\"#
    CCCCCC\"cellspacing=10")
print(PieBar)
print(Bar)
```

El comando `gvisMerge` permite poner las dos primeras visualizaciones en una sola fila.

**b)** Un tipo de visualizaciones distinto es de los mapas de geolocalización. Para esto se utiliza `gvisMap` que se utiliza sobre una base de datos —en este caso `db`— que hemos creado a partir de la original, `tb`, usada anteriormente. En primer lugar al nombre del barrio que había en la columna Barrio de `tb` —es decir `tb$Barrio`— le vamos a añadir "Barcelona, Spain"(esto es lo que almacenamos en la variable `dato_barrios`). Ahora vamos a crear una variable, que es lo que pondremos en la leyenda de cada banderita, que contendrá el nombre del barrio —que estaba en `tb$Barrio`—, la frecuencia —que está en `tb$Freq`— y le añadimos : entre medias como separador. Finalmente creamos una *data frame* que almacenamos en `db` formado por dos columnas la primera llamada `Postcode` y la segunda `Tip`. Estas dos variables son las que nos sirven para visualizar mediante la instrucción `gvisMap` con `locationvar="Postcode"` y `tipvar="Tip"`.

```
%r
# Unimos "Barcelona Spain" al nombre del barrio
dato_barrios<-paste(tb$Barrio,paste("Barcelona","Spain", sep='␣'))
# Unimos la frecuencia
frec_barrios<-paste(tb$Barrio,tb$Freq, sep=':')
# Hacemos la tabla
db<-data.frame(Postcode=dato_barrios,Tip=frec_barrios)
# Y finalmente imprimimos
M1 <- gvisMap(db, locationvar="Postcode", tipvar="Tip",
        options=list(showTip=TRUE, mapType='normal',enableScrollWheel=
            TRUE))
print(M1)
```

En algunas ocasiones no se ve el plano (prueba en este caso a actualizar el navegador).

**c)** Vamos a practicar algún otro tipo de visualización. Para ello vamos a crear una nueva tabla que corresponde sólo a los datos de abril de 2010 y que contiene sólo cinco columnas —vamos a verla con `gvisTable`—. En el `csv` descargado aparecen algunos campos vacíos —concretamente en el campo de las fechas—. Esto provoca que no se puedan hacer las visualizaciones con `googleVis`. Para evitar este problema una vez generado el *data frame* —que hemos almacenado en `minitabla`— creamos uno nuevo omitiendo las filas que tengan algún dato vacío mediante `na.omit`. Luego nos quedamos sólo con abril de 2017 ordenando por la columna de fechas y eligiendo las filas correspondientes.

```
%r
Id<-as.character(airbnb_Barcelona_data$id)
Tipo<-airbnb_Barcelona_data$room_type
Precio<-airbnb_Barcelona_data$price
Fecha<-airbnb_Barcelona_data$fecha
# Si construimos la tabla directamente hay fechas vacías
minitabla<-data.frame(Id,Barrio,Tipo,Precio,Fecha)
minitb<-na.omit(minitabla)
# Nos quedamos solo con abril de 2017
minitb <- minitb[order(minitb$Fecha,decreasing = TRUE),]
minit<-minitb[1:2525,]
# Y visualizamos
PopTable <- gvisTable(minit, options=list(page='enable'))
print(PopTable)
```

Con este nuevo *data frame* prueba las siguientes visualizaciones:

```
%r
Motion<-gvisMotionChart(minit,"Id", "Fecha")
print(Motion)
```

```
%r
Bubble <- gvisBubbleChart(minit, idvar="Id",
                          xvar="Precio", yvar="Barrio",
                          colorvar="Tipo", sizevar="Precio",
                          options=list(height=750,width=1100
                           ))
print(Bubble)
```

## 4. Y mucho más...

Además de los intérpretes predefinidos se pueden crear muchos otros. Por ejemplo se podría configurar un intérprete del programa de base de datos de grafos `Neo4j` que nos permitiría utilizar la sintaxis de Cypher para hacer nuestras búsquedas a partir de nuestro archivo `listings.csv`.

```
%neo4j
MATCH (n)-[r:PerteneceA]->(m)  RETURN n.zona as Zona,m.barrio as Barrio
   LIMIT 500
```

La versión actual de Apache Zeppelin es la 0.7.2. En breve saldrá la versión 0.8 que tiene implementada la opción `Helium` que proporcionará más visualizaciones así como la opción de crear otras.

Jose M. Calabuig Rodríguez
jmcalabu@mat.upv.es
Departamento de Matemática Aplicada
Instituto Universitario de Matemática Pura y Aplicada
Universitat Politècnica de València