Capítulo 4

Aplicaciones de tratamiento de datos con R

4.1. Introducción

Cuando empezamos a trabajar con un conjunto de datos, es conveniente explorarlos para ver sus características. Existen varias formas de extraer información básica de un objeto con datos, ya sean listas, matrices, data.frames o la cualquier otra clase. Algunos comandos funcionan con todas ellas, particularmente la función str. Otros comocolnames o rownames sólo sirven para objetos de dos dimensiones(como matrices).

Muchas veces interesa saber si las tablas se han importado correctamente o puede interesar realizar un visualización rápida de como son las variables y sus valores. Para esto existen dos funciones muy útiles que nos dan a conocer las primeras o últimas filas de nuestra tabla. Estas son las funciones head y tail respectivamente.

La función View es similar head solo que tanto en R como en RStudio, abre otra ventana respectivamente, y muestra la tabla con un formato tipo hoja de cálculo y en una mayor cantidad de filas.

A continuación, se va a tratar, sin entrar en excesivos detalles, el tratamiento estadístico básico de un conjunto de datos.

4.2. Tweets geolocalizados en la ciudad de Murcia con R

Para desarrollar esta aplicación, utilizamos una tabla de datos que se ha recopilado sobre el número y la geolocalización de los tweets que se han publicado en el centro de la ciudad de Murcia, durante algunos días del mes de marzo. La tabla se encuentra en un fichero que hemos denominado **tweetM.csv**. Hemos modificado esta tabla para obtener en una tabla separada los datos referentes a días de fin de semana (sábado-domingo) que llamamos **tweetMfd.csv**, mientras que la tabla **tweetMlab.csv** recoge los datos de tweets en día laborable, de lunes a viernes. Trabajamos inicialmente con la tabla general **tweetM.csv**.

Se puede introducir la tabla mediante la opción en RStudio *Import Dataset* o también mediante la introducción directa en el teclado utilizando la función read.csv(ruta", sep=";").

En nuestro caso hemos introducido la tabla como:

R 1 (Introducción de la tabla *tweetM.csv*).

```
> tweetM <- read.csv("/Volumes/Trabajo/Matematicas/networks/MURCIA/Twitter/tweetM.csv"
 +, sep=";")
>   View(tweetM)
> str(tweetM)
'data.frame': 2323 obs. of  6 variables:
$ date    : int  22 22 22 22 22 22 22 22 22 22 ...
$ hour    : int  9 9 9 9 8 8 8 8 5 ...
$ minute  : int  45 40 19 7 5 47 38 17 16 15 ...
$ tweet_id: num  7.12e+17 7.12e+17 7.12e+17 7.12e+17 7.12e+17 ...
$ lon     : num  -1.12 -1.14 -1.15 -1.12 -1.13 ...
$ lat     : num  38 38 38 38 38 ...
```

Una vez introducidos los datos de la tabla en R procedemos a obtener las características más básicas de la tabla con la función str(tweetM). Comprobamos que hemos importado al espacio de trabajo una tabla con 2323 registros y 6 variables, que son *date*, *hour*, *minute*, *tweetid*, *lon* y *lat*. De estas seis variables, las tres primeras son de tipo entera (*int*) y las tres últimas son numéricas (*num*).

La tabla **tweetM.csv** nos muestra la siguiente información sobre cada tweet geolocalizado en la ciudad de Murcia, en el mes de marzo de 2016:

- *date*: el día del mes en que se produce el tweet.
- *hour*: la hora del día en que se ha hecho el tweet.
- *minute*: el minuto en que se ha producido.
- *tweetid*: el identificador del tweet.
- *lon*: la coordenada longitud del mismo.
- *lat*: la coordenada latitud del mismo.

De esta forma, tenemos la geolocalización de los tweets de la ciudad de Murcia durante unos días del mes de Marzo.

Nos interesa separar los datos que tenemos por días, de manera que dispongamos de un subconjunto de datos que nos muestre la misma información, separando la misma en días, con el fin de analizar la emisión de tweets en cada día concreto del mes.

Para eso vamos a crear unas nuevas variables que reflejen los tweets de cada día concreto. Esto podemos hacerlo fácilmente con R, aprovechando las facilidades que nos ofrece el software para extraer subconjuntos de datos de un conjunto general. Así, hacemos:

R 2 (Datos de tweets por día).

```
> dia1 <- tweetM[tweetM$date == '1',  ]
> dia2 <- tweetM[tweetM$date == '2',  ]
> dia3 <- tweetM[tweetM$date == '3',  ]
> dia4 <- tweetM[tweetM$date == '4',  ]
> dia5 <- tweetM[tweetM$date == '5',  ]
  ...      ...      ...      ...
> dia22 <- tweetM[tweetM$date == '22',  ]
```

De esta forma hemos creado de una manera sencilla una variable para cada día de marzo (desde el 1 al 22) en la que tenemos los datos de los tweets y su geolocalización por día.

Si, por alguna razón, necesitamos extraer datos a partir de dos condiciones, podemos utilizar perfectamente el operador lógico & para encadenar las condiciones. Así, por ejemplo, supongamos que nos interesa conocer los tweets que se han producido el día 18 de marzo a partir de las 8 de la tarde. Para obtener este conjunto de datos, bastaría con que escribamos en R la siguiente secuencia:

R 3 (Extracción de un subconjunto de datos.).

```
> dato1 <- tweetM[tweetM$date == 18 & tweetM$hour >20,  ]
```

La obtención de un subconjunto de datos, ya sea de un vector, matriz o data frame, podemos realizarla mediante una instrucción denominada subset. Los argumentos más típicos y utilizados de esta función se resumen en:

```
subset(x, subset, select, ...)
```

donde x representa el objeto del que se extraen los casos, *subset* representa la expresión lógica que indica los elementos que deseamos extraer y *select* es la expresión que indica las columnas que deben ser seleccionadas.

Teniendo esto en cuenta, los tweets que se han producido el día 18 de marzo a partir de las 8 de la tarde, también pueden obtenerse en R utilizando la expresión:

R 4 (Subconjunto de datos con subset).

```
> dato1 <- subset(tweetM, date==18 & hour >20)
```

Consideramos ahora el problema de extraer todas las filas de un *data frame* que contengan varios valores de una variable concreta. Así, siguiendo con nuestro ejemplo, supongamos ahora que queremos obtener de la base de datos **tweetM** los tweets correspondientes a los días del fin de semana, contando como fin de semana sábado y domingo. En nuestra base de datos tenemos la variable *date*, que es la que determina el día. Dentro del mes de marzo y de los días de los que disponemos datos, nos interesan los tweets de los días $5 - -6$ y $19 - -20$, correspondientes al fin de semana. Se trata de volver a obtener un subconjunto de datos de la base **tweetM**, pero ahora necesitamos extraer las filas en las que en la variable *date* aparece $5, 6, 19$ o 20.

Podemos hacer esto en R de varias formas, aunque vamos a estudiar dos de ellas. En ambas se utiliza el operador (|).

La primera forma consiste en introducir en R la siguiente instrucción para obtener los tweets que se han producido en fin de semana:

R 5 (Subconjunto de datos con el operador (|)).

```
> fd1 <- tweetM[tweetM$date == "5" | tweetM$date == "6" | tweetM$date =="19" |
+ tweetM$date == "20", ]
```

La segunda forma de obtener la información que deseamos es mediante el siguiente conjunto de instrucciones:

R 6 (Subconjunto de datos con el operador (|)).

```
> fd1 <- (tweetM$date=="5" | tweetM$date=="6" | tweetM$date=="19" |  tweetM$date=="20")
> tweetM[fd1, ]
```

⊙ **Histograma de tweets por día**

Lo primero que nos cuestionamos es hacer un estudio de la cantidad de datos que se han realizado cada uno de los días para ver si existen similitudes o discrepancias en los resultados, desde el punto de vista cuantitativo.

Para realizar un histograma con una determinada variable, se utiliza la instrucción

```
hist(x, breaks="Sturges", freq = NULL, probability=!freq, include.lowest = TRUE,
      right=TRUE, density=NULL, angle=45, col=NULL, border=NULL,
      main.paste("Title", xname), xlim=range(breaks), ylim=NULL, xlab=xname,
      ylab, axes=TRUE, plot=TRUE, labels=FALSE, ann =TRUE, ...)
```

donde figuran los argumentos principales de esta compleja función. Los más importantes son:

x Variable (vector de datos) para los que queremos hacer el histograma.

breaks Vector que nos da los puntos de corte de las barras del histograma.

freq Cuando es TRUE se representan las frecuencias, es decir el número de veces que se da el valor.

include.lowest Para incluir en las barras el primer valor.

right Las barras se cierran por la derecha.

density La densidad de las líneas interiores de las barras.

angle Ángulo de las líneas interiores de las barras.

col Un color para rellenar las barras.

border El color del borde de las barras.

main, xlab, ylab Argumentos del título.

xlim, ylim Rango de x e y con valores sensibles

axes Se dibujan o se omiten, con sus respectivas anotaciones.

plot Para dibujar el gráfico.

labels Dibuja los valores en la parte superior de cada barra.

ann Para las anotaciones de los ejes.

En nuestro ejemplo, lo primero que debemos construir son dos vectores: el primero está relacionado con las separaciones o *breaks* de las barras. Queremos que cada día nos muestre en una barra su valor de tweets alcanzado. El segundo está relacionado con la intensidad que queremos en las diferentes barras que compondrán el histograma. Para ello, podemos formar los vectores:

R 7.

```
> sep <- c(0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5,9.5,10.5,11.5,12.5,13.5,
    + 14.5,15.5,16.5,17.5,18.5,19.5,20.5,21.5,22.5)
> den <- c(5,10,15,20,25,30,5,10,15,20,25,30,5,10,15,20,25,30,5,10,15,20,25)
```

Los vectores **sep** y **den** definen, respectivamente, las rupturas de las barras y las densidades de líneas en las mismas.

Ahora estamos en condiciones de dibujar el histogramas, utilizando el comando:

R 8 (Histograma de tweets por día)**.**

```
> hist(tweetM$date, sep,  density=den, col="violet", border="black", xlim=c(0,23),
main=paste("Histogram of tweets per day"),  xlab="Days", ylab="Number of tweets",
axes=FALSE, labels=TRUE, ann =TRUE)
> axis(1, at=1:22, lab=c("1","2","3","4","5","6","7","8","9","10","11","12","13",
"14","15","16","17","18","19","20","21","22"))
```

Con la instrucción *hist* y los parámetros que la acompañan hemos construido un histograma de frecuencias de tweets por día. La variable es *date*, la separación de las barras nos la determina el vector *sep*, mientras que la densidad de las líneas viene marcada por el otro vector *den*. Establecemos los colores de las líneas en violeta y negro en los bordes, mientras que establecemos las anotaciones que queremos que aparezcan en cada eje. Finalmente, le indicamos que nos muestre el valor de la frecuencia de tweets de cada día en la parte superior de la barra, así como que elimine las marcas de cada eje.

Las marcas en el eje de las X las insertamos mediante la función **axis**.

```
axis(side, at=NULL, label = TRUE, tick=TRUE, line = NA, pos = NA, outer=FALSE,
font=NA, lty="solid", lwd=1, lwd.ticks = lwd. col=NULL, col.ticks =NIULL)
```

Son sus principales argumentos:

side El eje en cuestión. Eje 1:bajo, 2:,3,4.

at Los puntos donde se dibujan las marcas.

labels Se pueden incluir anotaciones.

tick Valor lógico especificando si las marcas de los ejes deben visualizarse.

line MNúmero de líneas en el margen en el que la línea del margen debe dibujarse.

pos Coordenadas para la línea del eje.

outer El color del borde de las barras.

font Fuente para el texto.

lty Tipo de línea para el eje.

lwd,lwd.ticks Anchura de las líneas.

col,col.ticks Diversos colores.

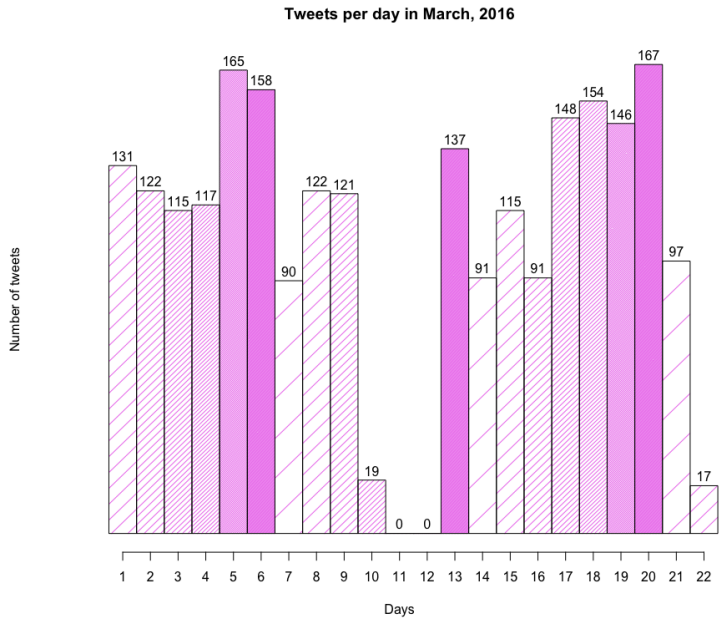

Figura 4.1: Histograma de frecuencias de los tweets por día (marzo) en la ciudad de Murcia.

El histograma resultante la vemos en la figura 4.1. Evidentemente se observa que no se disponen datos de los días 11 y 12, aunque el problema surgió el día anterior donde se observa un reducido número de tweets en comparación con el resto de datos registrados.

La razón para tomar el vector de densidades de la forma que lo hemos definido es para que podamos identificar visualmente el día de la semana que se corresponde con cada una de las barras del dibujo. Así, la barra con una menor

densidad de líneas corresponde al día lunes de la semana; la siguiente en densidad corresponde al martes y así sucesivamente, hasta que los días sábado y domingo presentan un relleno casi uniforme. De esta manera podemos observar más fácilmente las similitudes en los datos entre los mismos días de la semana.

A la vista del gráfico parece claro que los días con una mayor frecuencia de tweets son los sábados y los domingos, lo cual no resulta sorprendente. también resulta destacable que las frecuencias en los mismos días de la semana no sufren grandes variaciones y, en algunos casos, sus valores son similares.

⊙ Histograma de tweets por horas

Ahora nos interesa realizar el mismo estudio pero tomando como variable las horas. Queremos estudiar la frecuencia de los tweets en la ciudad por horas, independientemente del día en el que se producen.

Ahora la separación de las barras va a coincidir con cada una de las horas, por lo que podemos establecer el vector de separación **sephour** entre 0 y 23 horas. También descartamos el vector de densidades que hemos utilizado anteriormente, porque no vamos a incluir líneas en las barras, por lo que no especificamos la densidad de las mismas. Vamos a utilizar una escala de colores, de manera que cada barra quede dibujada de un color distinto. Para ello, utilizamos la escala dada por la función **rainbow(n)**, donde n es el número de colores. Así, tendremos los vectores dados por

R 9.

```
> sephour <- c(-0.5:23.5)
```

Ahora ya estamos en condiciones de representar el histograma, utilizando instrucciones similares a las utilizadas en el caso anterior.

R 10 (Histograma de tweets por horas)**.**

```
> hist(tweetM$hour, sephour,  density=30, col=rainbow(24), border="black", xlim =
c(0,24), main=paste("Histogram of tweets per hour"),  xlab="Hour", ylab="Number
of tweets", axes=FALSE, labels=TRUE, ann =TRUE)
> axis(1, at=0:23, lab=c("0","1","2","3","4","5","6","7","8","9","10","11","12","13","14",
"15","16","17","18","19","20","21","22","23"))
```

Con la instrucción *hist* y los parámetros que la acompañan hemos construido un histograma de frecuencias de tweets por hora. La variable es *hour*, la separación de las barras nos la determina el vector *sephour*. La diferencia con el caso anterior es que utilizamos una densidad constante para todas las barras y tenemos una escala cromática para representar la frecuencia de cada hora con un tono distinto.

La figura 4.2 nos muestra el histograma de frecuencias, en el que advertimos claramente la curva que se produce. Los valores mínimos se producen en las horas de madrugada, más concretamente entre las3 y las 7 de la mañana. También se advierten dos focos horarios de máxima actividad. El primero de estos focos se produce entre las 11 y las 14 horas, bajando las frecuencias durante las horas siguientes. Se vuelven a alcanzar valores máximos entre las 19 y las 22 horas, alcanzándose el máximo entre las 21 y las 22 horas.

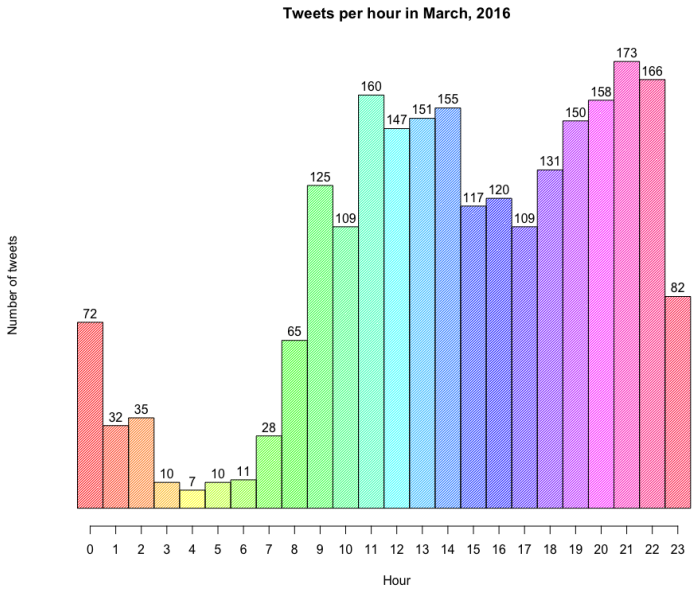

Figura 4.2: Histograma de frecuencias de los tweets por horas en la ciudad de Murcia.

Ejercicio 1.

Para este ejercicio debes cargar en R la base de datos relacionada con los tweets que se producen en fin de semana y en días laborables. Estas bases de datos se encuentran en los ficheros **tweetMfd.csv** y **tweetMlab.csv**.

1. Construye con R el histograma de tweets por hora, sólo de los días laborables, así como el histograma de tweets por hora durante los fines de semana.

2. ¿Observas diferencias o similitudes? Comenta los resultados.

Una vez estudiados los histogramas de los tweets, tanto por días como por horas, procedemos en las secciones siguientes a estudiar la forma de realizar un análisis gráfico de las geolocalizaciones de los tweets.

4.3. Mapas en R

R ha incluido desde sus inicios un package denominado *maps* para dibujar mapas geográficos. Como primera aproximación, está bien, pero para trabajar con mapas más precisos y atractivos, las grandes referencias en mapas son OpenStreetMap y Google Maps.

En una conferencia de usuarios de R en 2012 se presentó el package *ggmap*, que tiene como objetivo principal facilitar la tarea de importación de estos mapas en R. Así, es necesario instalar este package de la forma habitual:

R 11.

```
> install.packages("ggmap")
> library(ggmap)
```

A la hora de utilizar **ggmap**, debemos tener en cuenta que para obtener o cargar un mapa en pantalla, es necesario hacer los siguientes pasos:

1. Definir la localización o posición que deseamos visualizar.
2. Definir la fuente del mapa y el tipo.
3. Afinar la escala del mapa utilizando la opción *zoom*.

Definir la localización o posición que deseamos visualizar.
Tenemos tres formas distintas de definir una localización para ser usada en un mapa:

- *Localización/Dirección*. Basta con que demos una dirección conocida (entre comillas).
- *Longitud/Latitud*. Aquí debemos introducir la longitud y la latitud del lugar.
- *Rectángulo de selección*. Aquí debemos introducir cuatro parámetros: longitud inferior izquierda, latitud inferior izquierda, longitud superior derecha, latitud superior derecha. Lo que hacemos es delimitar el vértice inferior izquierda y superior derecha de un rectángulo que constituye el área que queremos visualizar.

Definir la fuente del mapa y el tipo.
La función que se utiliza para obtener mapas de un modo sencillo de múltiples fuentes es `get_map()`. Constituye una opción que nos permite explorar diferentes tipos de mapas.

```
get_map(location = c(lon = -95.3632715, lat = 29.7632836), zoom = "auto",
scale = "auto", maptype = c("terrain", "terrain-background", "satellite",
"roadmap", "hybrid", "toner", "watercolor", "terrain-labels", "terrain-lines",
"toner-2010", "toner-2011", "toner-background", "toner-hybrid",
"toner-labels", "toner-lines", "toner-lite"), source = c("google", "osm",
"stamen", "cloudmade"), force = ifelse(source == "google", TRUE, TRUE),
messaging = FALSE, urlonly = FALSE, filename = "ggmapTemp",
crop = TRUE, color = c("color", "bw"), language = "en-EN", api_key)
```

Analizamos brevemente las distintas posibilidades asociadas a los argumentos.

location Una de las tres posibilidades vistas anteriormente para dar la localización. La más cómoda es $c(lon =, lat =)$.

zoom Zoom del mapa, entre 3 (continentes) y 21 (edificios). Valor medio 10 (ciudad). Openstreetmaps presenta un límite de 18. .ªuto"determina automáticamente el zoom para el rectángulo definido como localización.

scale Argumento de escala referente a `get_googlemap()` o a `get_openstreetmap()`.

maptype Cadena de carácter que nos proporciona el tema del mapa. Las opciones posibles son "terrain", "terrain-background", "satellite", roadmap."nd "hybrid"(google maps), "terrain", "watercolor."nd "toner"(stamen maps) o un entero positivo para la fuente `cloudmap`.

source Goole maps ("google"), OpenStreetMap (."sm"), Stamen Maps ("stamen"), o CloudMade maps (çloudmade").

force Fuerza a crear un mapa nuevo.

messaging Mensajes on/off.

urlonly Devuelve solo la dirección web.

filename Fichero destino.

crop (Stamen and cloudmade) Corta títulos en los rectángulos de localización.

color Color o blanco y negro para el mapa.

language Idioma de google maps.

api-key Una api key para los mapas cloudmade.

Afinar la escala del mapa utilizando la opción *zoom*.

La función `get_map()` presenta diversos niveles de zoom, de acuerdo con el nivel de detalle que queremos visualizar. Así, este nivel de detalle lo controlamos con el parámetro *zoom*, que presenta un rango de 3 (nivel de los continentes) hasta 21 (nivel de edificios). El límite intermedio 10 corresponde al grado de detalle de ciudades. Notemos que OpenStreetMap tiene un límite de zoom de 18.

Hemos visto que hay cuatro fuentes para obtener mapas, cada una de ellas con unos tipos diferentes de mapas, como son

stamen maptype = c("terrain", "toner", "watercolor")

google maptype = c(roadmap", "terrain", "satellite", "hybrid")

osm open street map

cloudmade miles de mapas, aunque debe obtenerse una api key de la dirección http://cloudmade.com

A modo de ejemplo, vamos a obtener un mapa de google centrado en el punto $c(lon = -0.477159, lat = 38.349141)$ (castillo de Santa Bárbara, Alicante). Queremos obtener un mapa en blanco y negro de la zona próxima al punto geolocalizado, del tipo *satellite*. Para esto, basta introducir en R las siguientes instrucciones:

R 12 (Mapa de Google centrado en el castillo de Alicante)**.**

```
> miloc <- c(lon=-0.477159, lat=38.349141)
> mimapa <- get_map(miloc, zoom=16, source="google", maptype="satellite", color="bw")
> ggmap(mimapa)
```

Obtenemos el mismo mapa, tomando ahora como fuente OpenStreetMap, escribiendo

R 13 (Mapa de OpenStreetMap centrado en el castillo de Alicante)**.**

```
> miloc <- c(lon=-0.477159, lat=38.349141)
> mimapa2 <- get_map(miloc, zoom=16, source="osm", color="bw")
> ggmap(mimapa2)
```

Ejercicio 2.

Obténgase con R el mapa del campus de la universidad de Alicante, desde dos fuentes distintas como son google maps y OpenStreetMap, tanto en color como en blanco y negro. Escribe las instrucciones que utilizas de R.

Si por cualquier razón no podemos obtener el mapa que queremos, existen otras funciones definidas específicamente para las diferentes fuentes, como son: `get_googlemap()`, `get_openstreetmap()`, `get_stamenmap()`, `get_cloudmademap()`.

Una de las instrucciones esenciales de esta librería es `get_googlemap()`, que es la que nos permite descargar un mapa estático accediendo a la Google Static Maps API. Los argumentos principales que se asocian a la función específica de google maps son:

```
get_googlemap(center=c(lon= x, lat=y), zoom=k, size=c(640,640), scale=2,
format = c("png8","gif","jpg", "jpg-baseline", "png32"), maptype =
c("terrain","satellite","roadmap","hybrid"), language = "en-EN", sensor =FALSE,
urlonly=FALSE, filename = "ggmapTemp", color = c("color","bw"), force = FALSE,
where=tempdir(), archiving=FALSE, key="", region, markers, path, visible, style,...)
```

Analizamos brevemente las distintas posibilidades asociadas a los argumentos.

center El centro del mapa, en coordenadas lon/lat y como vector numérico.

zoom zoom del mapa, entre 3 (continentes) y 21 (edificios). Valor medio 10 (ciudad).

size dimensiones rectangulares en píxeles- hor x ver - con un máximo de $c(640, 640)$. La escala afecta a este factor.

scale Factor multiplicativo para el número de píxeles; este valor puede ser 1, 2 o 4. Por ejemplo, una imagen de $c(640, 640)$ y una escala igual a 2, produce una imagen de 1280×1280 píxeles. El valor 4 está reservado para negocios.

format Formato de imagen.

maptype Tipo de mapa.

language Idioma de las etiquetas.

sensor Especifica si se usa un sensor.

urlonly Devuelve solo la dirección web.

filename Fichero destino.

color Color o blanco y negro para el mapa.

Podemos ver en la ayuda una descripción más precisa de los argumentos, aunque aquí aparecen los más importantes.

Veamos un ejemplo. Escribimos en R,

R 14.

```
> map <- get_googlemap(center=c(lon= -1.1309, lat=37.9871), zoom=16, size=c(640,640),
scale=2, maptype = "roadmap", color="color")
> ggmap(map)
```

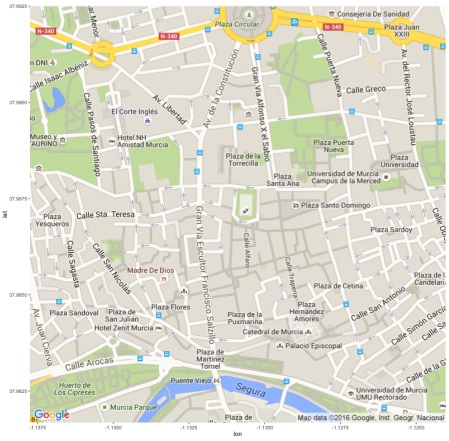

Figura 4.3: Mapa centrado en $center = c(lon = -1.1309, lat = 37.9871)$.

Con estas últimas instrucciones en R, hemos creado el mapa que aparece en la figura 4.3.

Ahora queremos avanzar un poco en la geolocalización de los datos. Para ello, utilizaremos una función llamada `geom_point()`, que es la que nos va a permitir dibujar un gráfico de dispersión de puntos. Al igual que en los casos anteriores, se trata de una función que tiene un gran número de posibilidades en forma de argumentos asociados a la misma. Comentamos los más destacados aunque recomendamos acudir a la ayuda para obtener una descripción más detallada del comando.

```
geom_point(mapping = NULL, data = NULL, alpha = , stat = "identity", position = "identity",
..., na.rm = FALSE, show.legend = NA, inherit.aes = TRUE)
```

Sus argumentos se refieren a:

mapping Es el conjunto de asignaciones estéticas creadas mediante la función *aes*.

data Los datos que se van a mostrar en esta capa.

alpha Controla el tema de la transparencia.

position Ajuste de la posición.

na.rm Borra los valores perdidos con un mensaje de advertencia.

Recordemos que los diagramas de dispersión se utilizan básicamente cuando queremos estudiar la relación entre dos variables, continuas o no. El inconveniente de los diagramas de dispersión es la superposición de puntos: siempre que tenga más de unos pocos puntos, éstos se pueden trazar unos sobre otros. Esto puede distorsionar seriamente el aspecto visual de la trama. No hay una solución a este problema, pero hay algunas técnicas que pueden ayudar. Se puede añadir información adicional utilizando las funciones `geom_smooth`, `geom_quantile` o `geom_density_2d`. Para pocos valores, `geom_boxplot` también puede ser útil. Alternativamente, se puede resumir el número de puntos en cada ubicación utilizando `stat_sum`.

Vamos a hacer un ejemplo de visualización de los tweets que se han producido en el centro histórico de la ciudad el día 5 de marzo, que fue sábado. Para eso, escribimos las siguientes instrucciones en R:

R 15.

```
> map <- get_googlemap(center=c(lon= -1.1309, lat=37.9871), zoom=16, size=c(640,640),
scale=2, maptype = "roadmap", color="color")
> ggmap(map)
> mur5 <- ggmap(map) + geom_point(aes(x = lon, y = lat), data = dia5, colour = 'red',
size = 3)
> mur5
```

Las dos primeras instrucciones son las utilizadas anteriormente para general el mapa de fondo. Superponemos los puntos geolocalizados de la variable *dia5*, que es donde están guardados los datos de los tweets del día 5. Obtenemos el mapa que vemos en la figura 4.4.

La función `geom_point()` podemos concatenarla varias veces de manera que vamos creando sucesivas capas sobre el gráfico. Podemos aprovechar esta característica para visualizar los tweets de varios días a la vez. Concretamente vamos a hacer un ejemplo que consiste en mostrar gráficamente los tweets que se han mostrado la primera semana del mes de marzo, desde el 1 de marzo (martes) hasta el 7 (lunes).

Además, vamos a discriminar por colores los tweets que se han producido en sábado o domingo frente a los que se han hecho el resto de la semana. Dibujamos en rojo los tweets del fin de semana, mientras que el resto se encuentran en azul. El gráfico se obtiene a partir de las instrucciones:

R 16.

```
> map <- get_googlemap(center=c(lon= -1.1309, lat=37.9871), zoom=16, size=c(640,640),
scale=2, maptype = "roadmap", color="bw")
murcia2 <- ggmap(map) + geom_point(aes(x = lon, y = lat), data=dia1, colour='blue',
size=2) +  geom_point(aes(x = lon, y = lat), data = dia2, colour='blue', size=2) +
geom_point(aes(x=lon, y=lat), data = dia3, colour = 'blue', size = 2) +
geom_point(aes(x=lon, y=lat), data = dia4, colour = 'blue', size = 2) +
geom_point(aes(x=lon, y=lat), data = dia5, colour = 'red', size = 3) +
geom_point(aes(x=lon, y=lat), data = dia6, colour = 'red', size = 3) +
geom_point(aes(x=lon, y=lat), data = dia7, colour = 'blue', size = 2)
```

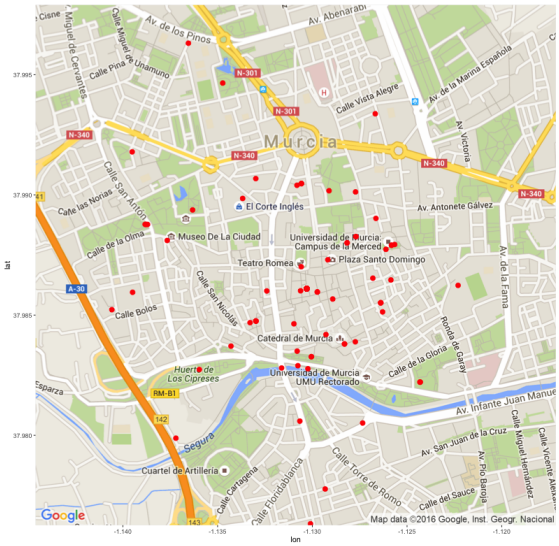

Figura 4.4: Mapa de los tweets en el centro de Murcia el día 5 de marzo de 2016 (sábado).

Notemos que la única diferencia en la construcción de este mapa es que ahora hemos generado el mapa de google en blanco y negro, con el fin de que se distingan mejor los puntos geolocalizados sobre el mapa.

En la figura 4.5 se ha representado la localización de los tweets recogidos en el centro de Murcia entre los días 1 al 7, ambos inclusive. En color rojo se han representado los tweets producidos en fin de semana, mientras que en color azul aparecen los tweets de los días laborables. Se observa claramente que la densidad de tweets en fin de semana es más alta que el resto de días, así como la mayor densidad de tweets en fin de semana en espacios públicos como plazas y jardines publicos. También notamos claramente una concentración de tweets mayor en torno al centro histórico (próximo a la zona de la catedral), durante el fin de semana. El resto de la semana se produce una dispersión mayor de los tweets.

La figura 4.6 refleja todos los tweets que aparecen en los datos registrados, independientemente de la hora y el día en el que se efectuaron. Este gráfico nos permite, con toda claridad, identificar zonas de alta concentración de tweets, así como realizar un análisis de las causas de los patrones que se encuentran en la repetición de datos.

4.4. Creando un mapa personalizado

El objetivo que nos proponemos ahora en esta sección es la de crear un mapa personalizado de un país que queramos vistar, mostrando las regiones que sean de nuestro interés para establecer la ruta y las comunicaciones entre ellas. Supongamos que queremos viajar a Uzbekistán y que vamos a marcar las regiones del mismo que tienen un mayor interés para nosotros, desde el punto

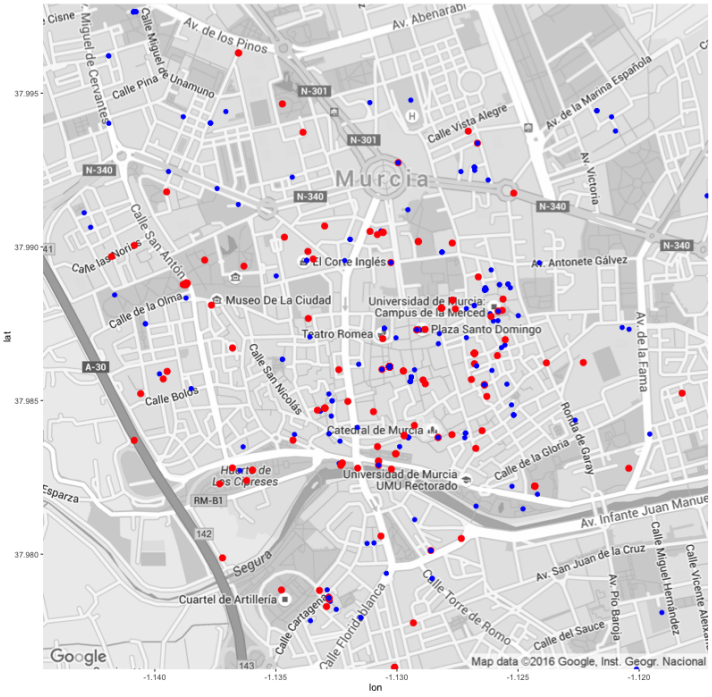

Figura 4.5: Mapa de los tweets geolocalizados en el centro de Murcia del 1 al 7 de marzo de 2016.

de vista turístico.

Procedemos realizando los siguientes pasos.

Paso 1. Descarga de los mapas.

Disponemos de múltiples lugares desde los que nos podemos descargar mapas de todos los países del mundo, así como del mundo en su globalidad. De entre todos ellos, queremos mencionar una base de datos que se denomina *Global Administrative Areas* (GADM) en http://gadm.org. GADM es una base de datos espaciales sobre la localización de las áreas o fronteras administrativas de todo el mundo para su uso en herramientas de GIS o cualquier otro software parecido. Cuando hablamos de áreas administrativas nos referimos a bases de datos donde aparecen países en su globalidad o subdivisiones tales como provincias, estados, condados, donde cada área tiene sus atributos con nombres, etc.

Los formatos en los que se encuentran los datos son: *shapefile*, *ESRI geodatabase*, *RData*, *Google Earth*. El formato *shapefile* resulta bastante interesante puesto que nos permite su uso por la mayoría de programas de GIS. El shapefile es un formato para sistemas de información geográfica. Es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos. Si bien la definición es básica, es increíble la cantidad de información descriptiva de un mapa que se puede almacenar en un *shapefile*.

Así pues, en nuestro caso, nos descargamos los mapas de Uzbekistán en for-

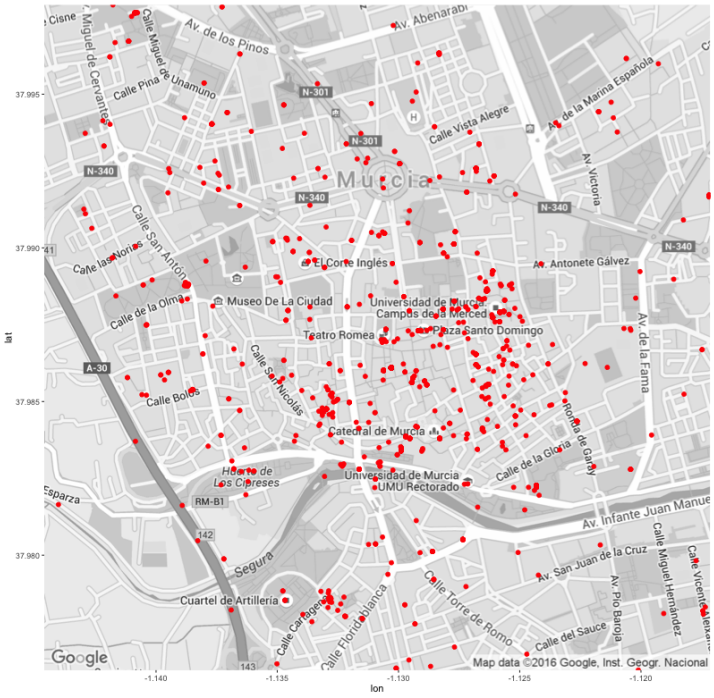

Figura 4.6: Mapa de los tweets de Murcia.

mato *shapefile* acudiendo a la zona de descargas de la página web oficial de GADM. Notamos que disponemos de varias bases de datos, **UZB_adm0.csv**, **UZB_adm1.csv** y **UZB_adm2.csv**. Las diferencias entre ellas es el grado de detalle de las áreas que se especifican. Así, por ejemplo, el fichero **UZB_adm1.csv** nos muestra las regiones principales de Uzbekistán, como si fuera un mapa nuestro de provincias.

Paso 2. **Descarga de los paquetes necesarios.**

Necesitamos instalar y cargar en R las bibliotecas *maptools*, *sp* y *RColorBrewer*. La librería o paquete *maptools* nos proporciona herramientas para leer y manejar los objetos espaciales. La librería o paquete *sp* implementa ciertas herramientas para manejar datos espaciales. La libreria *RcolorBrewer* nos proporciona una paleta de colores muy habitual en cartografía (véase http://colorbrewer2.org/).

Para instalar los paquetes hacemos lo siguiente

R 17.

```
> install.packages("maptools",lib="/directorio")
> library(maptools)
> install.packages("sp",lib="/directorio")
> library(sp)
> install.packages("RcolorBrewer",lib="/directorio")
> library(RcolorBrewer)
```

Paso 3. Convertir el fichero shapefile en un vector.

Leemos el fichero *shapefile* y lo convertimos en un nuevo vector que denominamos *UZ*. Escribimos

R 18.

```
> UZ <- readShapeSpatial("UZB_adm1.shp")
> plot(UZ)
```

Hemos cargado el mapa en formato shapefile y lo hemos convertido en un conjunto de datos espaciales. Para comprobar que se ha construido el mapa, basta con utilizar de una manera simple la función plot. Vemos que se dibuja el mapa político de Uzbekistán.

Ahora construimos una variable como el vector construido. Hacemos

R 19.

```
> mapa = UZ
```

Paso 4. Construir y abrir una tabla auxiliar.

Construimos una tabla auxiliar donde la primera columna está formada por las localidades y la segunda está formada por el criterio que utilicemos en cada caso. En nuestro caso, como queremos representar las zonas del país que deseamos visitar, debemos construir una columna donde indiquemos con un 1 aquéllas zonas de nuestro interés. El resultado final de la construcción de esta nueva tabla lo tenemos en el fichero *uzbe.csv*. Es importante señalar que las columnas deben tener el nombre "local"para la primera columna y "visita"para la segunda columna, siendo la codificación que se utiliza para la segunda columna de NA o 1 (lugar de interés).

Ahora debemos abrir el archivo de la tabla y crear otro vector, que llamaré üzbe". Así, escribimos

R 20.

```
> uzbe <- read.csv("uzbe.csv",header =TRUE)
> uzbe
local visita
1         Andijon     NA
2         Bukhoro      1
3        Ferghana     NA
4         Jizzakh     NA
5  Karakalpakstan     NA
6     Kashkadarya     NA
7         Khorezm     NA
8        Namangan     NA
9           Navoi     NA
10      Samarkand      1
11       Sirdaryo     NA
12   Surkhandarya     NA
13  Tashkent City      1
14       Tashkent     NA
```

Ahora nos queda reasignar la información de la variable *visita* del vector üzbe.[a]l *mapa* para que pueda ser representado correctamente. Para esto hacemos

R 21.

```
> mapa@data=data.frame(uzbe$visita)
```

Paso 5. Dibujo del mapa

Aunque hay múltiples paquetes para asignar colores en un mapa, en este caso me base en la paleta de colores de *colorBrewer*. El color pueden seleccionarlo guiándose con el siguiente link: http://colorbrewer2.org/

Ahora dibujamos el mapa con las localidades seleccionadas en "visitaz el color lo especificamos mediante col.regions haciendo *col.regions=brewer.pal()* como sigue a continuación:

R 22.

```
> spplot(mapa,c("uzbe.visita"),col.regions=brewer.pal(3, "RdYlBu"),
scales=list(draw = TRUE), main = "Regiones de interés de Uzbekistán")
```

La imagen que se obtiene la vemos en la figura 4.7.

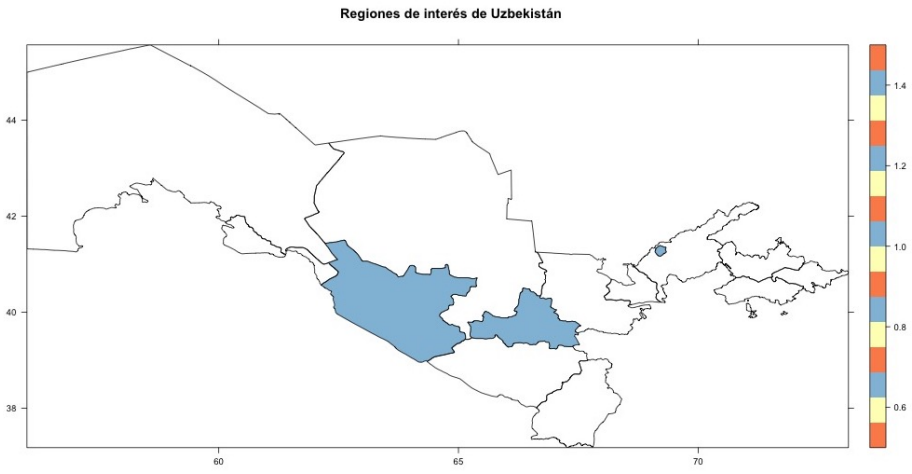

Figura 4.7: Mapa de las regiones de interés de Uzbekistán.

Otros links de interés donde puedes encontrar mapas son los siguientes:

- http://www.vdstech.com/map-data.aspx
- http://www.mapcruzin.com/free-asia-arcgis-maps-shapefiles.htm

- http://www.vdstech.com/usa-data.aspx

Ejercicio 3.

Descarga los mapas de España disponibles en GADM. Realiza sobre estos mapas las siguientes tareas:

1. Dibuja mapas de provincias, de comarcas y de pueblos.
2. Dibuja un mapa donde aparezcan las provincias que has visitados en tus viajes por España.
3. Intenta hacer lo mismo con un mapa del mundo, seleccionando los países que has visitado.