Visualización de datos con R

Iñaki Inza, Borja Calvo

La visualización de datos es un campo en efervescencia dentro de la esfera del data science. Previo al aprendizaje de cualquier modelo (e.g. clustering, reglas de asociación, etc.), una apropiada y atractiva visualización puede ser suficiente para cubrir nuestras necesidades de análisis y descubrir nuevos patrones; y puede convencer a los directivos de la empresa a seguir apostando y que ganen confianza con las posibilidades del análisis de datos.

Con este objetivo, durante los últimos años se han desarrollado una notable cantidad de softwares y plataformas con posibilidades de visualización realmente atractivas. Me centraré en algunas que nos ofrece `R` software. Mientras que su distribución básica ya nos ofrece una serie de posibilidades para desarrollar gráficos standard, la comunidad nos está ofreciendo paquetes especializados. Por nombrar algunos destacados, y que serán en los que nos centraremos en este tutorial: `lattice`, `ggplot2`, `googleVis`.

1 R graphics. Fundamentos

Las siguientes líneas son un resumen personal del material de `R graphics` del libro [8]. Antes de empezar, tened en cuenta que las funciones gráficas en `R` se dividen en "high level functions" (que producen gráficos completos) y "low level functions" (que añaden objetos a gráficos existentes: líneas, texto...).

1.1 Principales parámetros gráficos

La función `plot()` nos ofrece el baseline de partida. La función `par()` nos ofrece actuar sobre parámetros del gráfico: si consultas su ayuda, comprobarás que la cantidad de parámetros para tunear el gráfico a tus requerimientos es mareante (también los tienes aquí). Por ejemplo, `mfcol` y `mfrow` nos permiten fijar la disposición en pantalla de los gráficos que vayamos creando.

```
f = factor(c("M", "M", "M", "M", "M", "F", "F","F"))
y = rnorm(8)
x = c(0, 2, 4, 8, 16, 32, 64, 128)
par(mfrow = c(2, 2))
plot(y)
plot(f)
plot(x, y)
plot(f, x)
```

El paquete `msos` nos ofrece una serie de datasets reales. Su dataset `states` recoge datos de 51 regiones de Estados Unidos. Los populares *scatterplot*, siempre tan útiles para mostrar distribuciones bivariadas. La función `pairs()` muestra una matriz con todos los posibles scatterplots bivariados.

```
library(msos)
data(states)
names(states) # lista de variables
```

Año: Master ICSI-KISA
Profesor(a): Iñaki Inza, Borja Calvo
Teléfono: 943 01 50 26
E-mail: inaki.inza,borja.calvo@ehu.es

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

```
str(states) # mostrando en forma compacta la estructura de los datos
plot(x= states$Income, y = states$Prisoners)
```

El tipo de representación gráfica se fija mediante el parámetro *type*: "p" para puntos (por defecto), "l" para líneas, "h" para histogramas, etc.). Los símbolos que representan los puntos pueden tunearse de muchas formas. Por ejemplo, los parámetros `pch`, `cex` y `col` se utilizan para fijar el tipo de símbolo, dimensión y color de cada punto, respectivamente.

```
plot(y,type="h") # tipo de representación gráfica
plot(x= states$Income, y = states$Prisoners, pch=18, cex=1.5, col="blue") # symbol tuning
```

El prefijo "states" para nombrar cada variable puede evitarse mediante la función `attach(states)`.

Diferentes parámetros nos permiten definir el *title* del gráfico y sus ejes. Mientras el parámetro `main` permite definir el título principal del gráfico, `xlab` y `ylab` acceden al título de ambos ejes. Los parámetros `xlim` y `ylim`, para los rangos en los ejes. Los ejes pueden fijarse en escala logarítmica mediante el parámetro `log`, e.g. `log="x"`, `log="xy"`. Lo anterior, reflejado en Figura 1.

```
# fíjate en la alternativa Y ~ X para fijar las variables en ambos ejes
plot(states$Prisoners ~ states$Income, main="Prisoners vs Income per USA state",
     xlab="Per capita income", ylab="Number of prisoners per 100,000 population")

plot(states$Prisoners ~ states$Income, main="Prisoners vs Income per USA state",
     xlab="Per capita income", ylab="Number of prisoners per 100,000 population",
     xlim=c(20000,50000), ylim=c(0,200), pch=16, col="blue", cex=2.0)
```

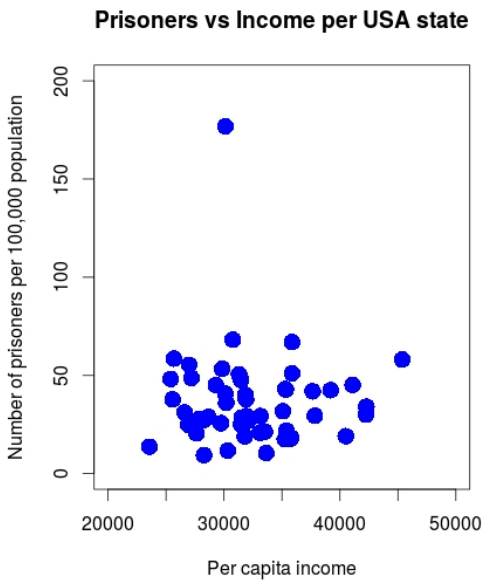


Figura 1: Prisoners vs Income per USA state

1.2 Funciones gráficas de low-level

Estas funciones de "bajo nivel" pueden mejorar el aspecto de los gráficos generados por las funciones de alto nivel(e.g. `plot()`). La función `text` permite introducir texto dentro del gráfico, mediante el parámetro `labels`. Los dos primeros parámetros de la función `text` representan las coordenadas donde ubicar el texto.

```
plot(states$Prisoners ~ states$Income, xlab="Per capita income",
     ylab="Number of prisoners") # creating the plot
# añadiendo el nombre del estado en cada punto
text(Prisoners ~ Income, data=states, labels=row.names(states), cex=1.0)
text(30000,150,labels="Prisoners vs Income per USA state", cex=2.0, col="royalblue")
```

La función `points()` controla el aspecto de los puntos dibujados. Podemos ajustar el diametro de cada punto respecto a la población del estado. La función `identify()` permite mostrar el índice de cada punto (después de clickar escape).

```
myCex = 4 * states$Population/max(states$Population)
plot(states$Prisoners ~ states$Income, xlab="Per capita income",
     ylab="Number of prisoners", main="Prisoners vs Income per USA state")
points(Prisoners ~ Income, data=states, pch=16, cex=myCex, col="black")
text(35000, 150, labels="Point size related to state population", cex=0.75)
# identificar el índice de cada punto clickado
identify(states$Prisoners ~ states$Income)
```

El gráfico se puede enriquecer mediante distintos tipos de *líneas*. La función `abline()` permite añadir líneas horizontales, verticales, oblicuas: en este caso, los valores medios de cada eje y un estimador-regresor de mínimos cuadrados. Hacemos también uso del parámetro *title*.

```
abline(h=mean(states$Prisoners), col="blue", lwd=2)
abline(v=mean(states$Poverty), col="red", lwd=2)
abline(lsfit(states$Poverty, states$Prisoners), col="green", lwd=3)
legend(x="topleft", legend=c("Average poverty", "Mean number of prisoners", "Linear regression"),
       col=c("red", "blue", "green"), lwd=3, cex=0.6)
title(main = "Poverty versus Number of prisoners in USA")
```

Aparte de la recurrent función `plot()`, `R` ofrece otros tipos de plots como *histogram, barplot* y boxplot. *Barplots* ofrece un formato atractivo para variables nominales-categóricas. *Boxplots* enriquece la información de una variable numérica: es posible segmentarla-condicionarla respecto a los valores de una variable categórica-nominal. El "box" muestra la mediana mediante una línea, así como los valores de primer y tercer cuartil. Los "whiskers"-"bigotes" muestran los valores mayores-menores que se encuentran a una distancia menor que 1.5 veces la longitud del "box"-"caja". Valores externos a los "whiskers" se muestran con círculos. Figura 2.

```
# histograma de frecuencias
hist(states$Crime, main="USA states' crime histogram", xlab="Crime degree",
     col="red", border="white")
data(SAheart) # mediciones sobre historias médicas coronarias
counts=table(SAheart$chd) # coronary heart disease (1=true, 0=false)
# barplot para una variable
barplot(counts, main="Diagnosed heart disease", col=c("blue", "red"))
counts=table(SAheart$chd, SAheart$famhist)
barplot(counts, xlab="Family history of heart disease", main="Diagnosed heart disease",
        col=c("blue", "red"), legend=T)
# beside display mode
```

```
barplot(counts, xlab="Family history of heart disease", main="Diagnosed heart disease",
        col=c("blue", "red"), beside=T, legend=T)
boxplot(SAheart$tobacco ~ SAheart$chd, main="Cumulative tobacco (kg)",
        xlab="Diagnosed heart diasease")
```

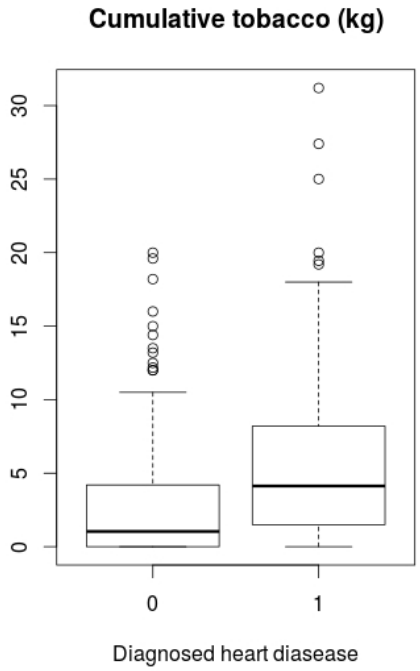


Figura 2: Cumulative tobacco in each group of patients (diagnosed heart disease negative or positive)

2 El paquete `Lattice`: mostrando una variable numérica condicionando a los valores de una nominal

En las siguientes líneas recojo un resumen personal acerca del material del paquete `lattice` en las siguientes referencias [4, 3].

El popular paquete `lattice` una notable mejora respecto a las capacidades gráficas básicas de `R` al mostrar relaciones entre variables: especialmente cuando una variable numérica es condicionada a una (o más) variables nominales-categóricas. Esta última variable es utilizada para "segmentar" la visualización de los valores de la variable numérica. Por ejemplo, mostrando la cantidad horaria de lluvia en siete subgráficos, uno por día de la semana. Así, estas variables, reconocidas como *factor* por `R` (e.g. día de la semana) son cruciales en los gráficos de `lattice`: codifican un conjunto finito de "categorías", "niveles", "factores" (e.g. "lunes", "martes"...). Ten en cuenta que `R` ofrece la `factor()` en caso de que dicha transformación en el tipo de la variable sea necesaria.

Esta relación entre variables numéricas y de tipo factor se puede codificar de distintas formas: mientras $\sim X$ muestra la variable numérica X en solitario, $\sim X|A$ muestra X por separado para cada valor-nivel de la variable factor A, $Y \sim X|A * B$ muestra la relación bivariada entre las dos numéricas Y y X de forma separada para combinación de los valores-niveles de las variables tipo factor A y B. Y así podríamos seguir...

Algunos ejemplos. Cargamos el data frame `mtcars` de nuestro paquete: recoge 11 variables de 32 vehículos antiguos. El número de marchas y cilindros son utilizados como variables nominales (tipo factor) para segmentar en los valores de las variables numéricas. Así, primero tenemos que convertir en variables tipo factor estos valores numéricos discretos (integer) de las variables "gears" y "cyl".

```
library(lattice)
attach(mtcars)
# crear dos variables nominales, tipo factor
gear.f<-factor(gear,levels=c(3,4,5), labels=c("3gears","4gears","5gears"))
cyl.f <-factor(cyl,levels=c(4,6,8), labels=c("4cyl","6cyl","8cyl"))
cyl.f
```

`lattice` ofrece una amplia gama de gráficos que puedes observar en la Figure 3, extraida del libro [4]).

Function	Default Display
`histogram()`	Histogram
`densityplot()`	Kernel Density Plot
`qqmath()`	Theoretical Quantile Plot
`qq()`	Two-sample Quantile Plot
`stripplot()`	Stripchart (Comparative 1-D Scatter Plots)
`bwplot()`	Comparative Box-and-Whisker Plots
`dotplot()`	Cleveland Dot Plot
`barchart()`	Bar Plot
`xyplot()`	Scatter Plot
`splom()`	Scatter-Plot Matrix
`contourplot()`	Contour Plot of Surfaces
`levelplot()`	False Color Level Plot of Surfaces
`wireframe()`	Three-dimensional Perspective Plot of Surfaces
`cloud()`	Three-dimensional Scatter Plot
`parallel()`	Parallel Coordinates Plot

Figura 3: Tipos de gráficos ofrecidos por `lattice`

Los *histogramas y los "kernel density plots"* (fijando una función de densidad de kernel en cada punto y sumando todas ellas para formar la función de densidad de la variables) se obtienen con la función `densityplot()`. Se mostrarán en la Figura 4 tres subgráficos, uno por cada valor de la variables "number of cylinders". La función `layout` permite dividir la pantalla en un número específico de columnas y filas.

```
# conditioned density plot
densityplot(~mpg|cyl.f, main="Density Plot by Number of Cylinders",
   xlab="Miles per Gallon", layout=c(3,1))
```

Vayamos con algunos tipos de gráficos más. Los *Stripplots* permiten ordenar los valores de una variables en un eje. Los anteriormente comentados *box and whisker* pueden multiplicarse al condicionar-segmentar la visualización de la variables numérica en varias variables tipo factor. Lo mismo se aplica para los *scatterplots*,

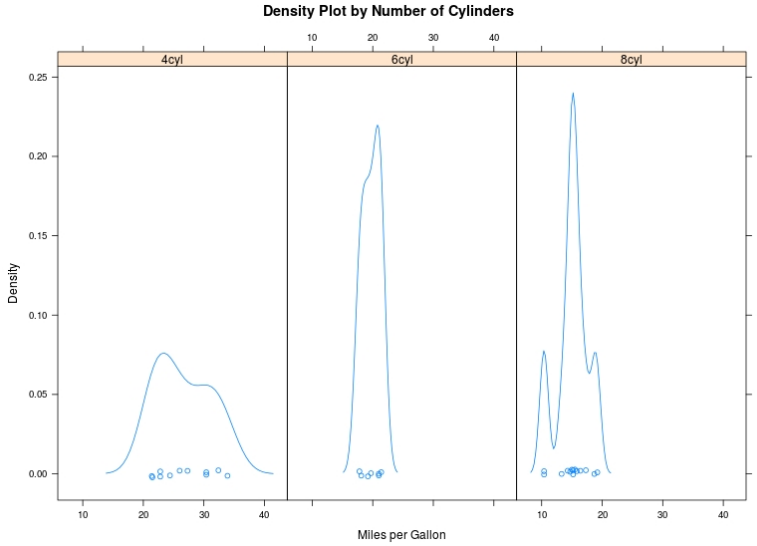


Figura 4: Density Plot by Number of Cylinders

que muestran la relación entre dos variables numéricas (Figura 5); la función `cloud()` los extiende a 3 variables numéricas. Los `levelplots` muestran esta relación tri-variada utilizando el color para generar la tercera dimensión-variable.

```
# stripplot: una variable factor condicionando-segmentando
stripplot(cyl.f~mpg, ylab="Cylinders", xlab="Miles per Gallon", main="Mileage per cylinder")
# stripplot: cambiando de eje la variable factor que condiciona-segmenta
stripplot(mpg~cyl.f, ylab="Miles per Gallon", xlab="Cylinders", main="Mileage per cylinder")
# box and whiskers plot: una variable factor condicionando-segmentando
bwplot(cyl.f~mpg, ylab="Cylinders", xlab="Miles per Gallon",
        main="Mileage by Cylinders")
# box and whiskers plot: dos variables factor condicionando-segmentando
bwplot(cyl.f~mpg|gear.f,ylab="Cylinders", xlab="Miles per Gallon",
        main="Mileage by Cylinders and Gears", layout=(c(1,3)))
# scatterplot: relación bivariada entre dos numéricas, condicionada a los valores
# de una tercera nominal
xyplot(mpg~wt|gear.f, main="Mileage and weight relationship by Gears", ylab="Miles per Gallon",
       xlab="Car Weight", layout=(c(3,1)))
# scatterplot: elación bivariada entre dos numéricas, condicionada a la
# combinación de valores de varias nominales
xyplot(mpg~wt|cyl.f*gear.f,main="Scatterplots by Cylinders and Gears",
   ylab="Miles per Gallon", xlab="Car Weight")
# 3-D scatterplot, segmentando por los valores de un único factor
cloud(mpg~wt*qsec|cyl.f, main="Mileage-Weigth-Acceleration Scatterplot by Cylinders",
      layout=(c(1,3)))
# Levelplot: utilizando colores para ganar una tercera dimensión
levelplot(mpg~wt*qsec, main="Mileage (by color scales) versus Weigth versus Acceleration")
# 3-D levelplot, condicionando a un factor
levelplot(mpg~wt*qsec | gear.f, layout=(c(3,1)),
          main="Mileage (by color scales) versus Weigth versus Acceleration by number of Gears")
```

Para conocer el interesante papel en las funciones de los símbolos $*$, $|$, $\sim$, consulta la página de ayuda de `lattice`.

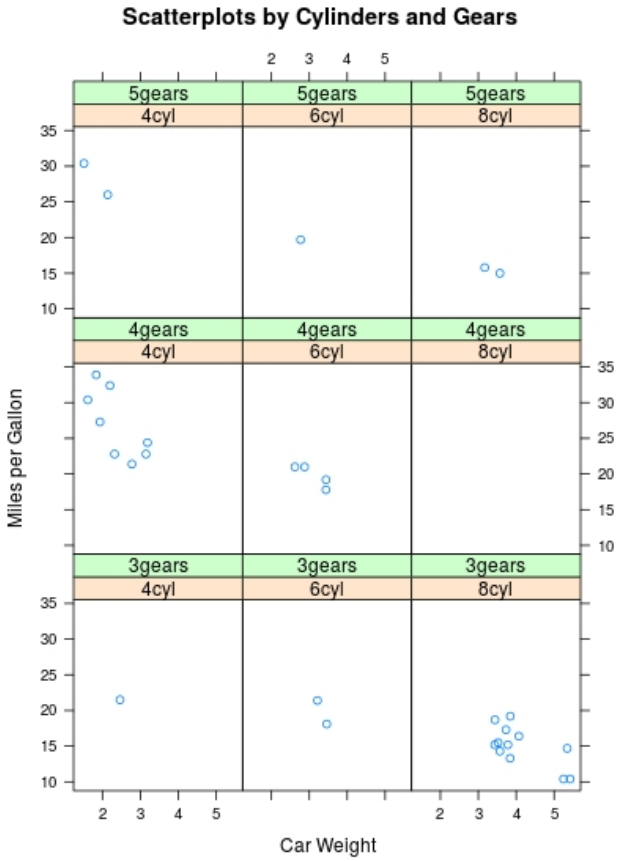


Figura 5: Scatterplots by Cylinders and Gears

3 `ggplot2`: la gramática de los gráficos

El popular paquete `ggplot2` se están convirtiendo en un "de facto" para los gráficos estáticos. Nos ofrece una batería de gráficos realmente atractivos. Eso sí, hay un "precio que pagar": se basa en una gramática, en un pequeño lenguaje, para llamar a las funciones gráficas. Los primeros pasos son farragosos, pero un poco de práctica nos hará tener a este paquete entre nuestros favoritos. Para empujaros en unos primeros pasos, me basaré en los primeros seis capítulos del libro [10].

La gramática nos va a permitir cambiar iterativamente distintas características de un gráfico ya creado. Estos llamados "building blocks" (o "piezas") de un gráfico pueden ser combinados para formar la visualización final: geometría, escalas, transformaciones estadísticas, etc... Y por otro, las llamadas "aesthetics", o características "que se pueden ver" de un gráfico: posiciones-ejes, color, relleno, posición, tamaño, etc.: se trata de que las variables de nuestros datos se mapeen a las "aesthetics" del gráfico.

El gráfico es posible crearlo con una única llamada a la función `qplot()`; o "pieza-a-pieza" con la función `ggplot()`, añadiéndo al gráfico "capas-layers" (elementos adicionales) de manera iterativa.

Cargaremos el dataset `diamonds` del propio paquete: características de una muestra de diamantes. Mostramos varios *gráficos de puntos* mediante la función `qplot()`: por defecto, para variables numéricas el objeto geométrico a mostrar son los puntos, `geom="point"`.

```
# describiendo las variables
summary(diamonds)
# seleccionamos 600 de sus casos al azar para las visualizaciones
dsmall=diamonds[sample(nrow(diamonds),600), ]
# ``carat" se relaciona con el peso del diamante
qplot(carat,price,data=dsmall)
# tenemos 4 ``aesthetics" en este llamada: ejes x e y, colour y shape del gráfico;
# los matcheamos con variables de nuestro dataset: ``carat", ``price", ``color", ``cut"
qplot(carat,price,data=dsmall,colour=color,shape=cut)
# añadiendo dos objetos geométricos (``geom") al gráfico: puntos y una regresión sobre ellos;
# al ser dos objetos geométricos, los concatenamos con la función ``c()"
qplot(carat,price,data=dsmall,geom=c("point","smooth"))
```

Vayamos con otros dos objetos geométricos ("geoms") de referencia: gráficos de histogramas y densidades.

```
# Histogramas
qplot(carat,data=diamonds, geom="histogram")
# visto el valor máximo en ``x", lo acotamos;
# y reducimos el tamaño del intervalo horizontal, ``binwidht": ¡cómo cambia!
qplot(carat,data=diamonds, geom="histogram",xlim=c(0,3),binwidth=0.01)
# Densidades
qplot(carat,data=diamonds, geom="density")
# aprovechando la variable ``color" para segmentar las densidades: ¡qué interesante!
qplot(carat,data=diamonds, geom="density",colour=color)
```

El llamado "faceting", de forma similar a la visto en el paquete `lattice`, nos permite segmentar-dividir el dataset en subconjuntos para luego mostrar figuras independientes para cada uno de ellos.

```
# histograma de ``carat" por valor de la variable ``color"
qplot(carat, data=diamonds, facets=color ~ ., geom="histogram", xlim=c(0,3))
# con ``..density.." mostramos en el eje Y la densidad en vez del conteo (count)
qplot(carat, ..density.., data=diamonds, facets=color ~ ., geom="histogram", xlim=c(0,3))
```

Todos estos mismos gráficos pueden ser creados mediante la función `ggplot()`. Ésta tiene dos argumentos: los datos, y un segundo que mediante la función `aes()` incluya los "aesthetics" al gráfico:

```
ggplot(diamonds, aes(carat, price, colour=color))
# observas que el gráfico está vacío, no muestra nada: en un momento entenderás porqué
```

Pasemos a continuación a hacer un uso intensivo de la gramática de `ggplot2`. Añadiremos capas-"layers" a gráficos creados (mediante `qplot()` o `ggplot()`), actualizándolos iterativamente. Con múltiples opciones, estas capas permitirán tunear el gráfico a nuestro gusto. Estas capas se especifican mediante "tipocapa_opcion".

```
# al gráfico anterior es necesario añadirle el tipo de objeto geométrico del gráfico;
# esto lo haremos mediante una capa ``geom", indicando que sea mediante puntos;
# ojo: ``carat", ``price" y ``cut" son variables del dataset
p = ggplot(diamonds, aes(carat, price, colour=cut))
p + geom_point()
# o creando el gráfico sin segmentar por colores los casos;
```

```
# y añadiendo una segunda capa de escalando el eje vertical;
# o añadiendo una segunda capa ``geom_stat()" con un modelo lineal ajustando los puntos;
# ya ves que cada capa añadida tiene opciones (e.g. el color)
q = ggplot(diamonds, aes(carat, price))
q + geom_point() + scale_y_log10()
q + geom_point() + geom_smooth(colour="red")
# histograma para una variable continua, variando la anchura del intervalo horizontal (bin)
# añadiendo un fondo en blanco-negro, títulos de ejes...
r = ggplot(diamonds,aes(depth))
r + geom_histogram(binwidth=10)
r + geom_histogram(binwidth=1) + theme_bw() + xlab("profundidad diamante") + ylab("núm. casos")
# un clásico box-plot: precio, según la "clarity" del diamante;
# el parámetro alpha añadiendo transparencia según el solapamiento de puntos: "overplot"
s = ggplot(diamonds, aes(clarity,price))
s + geom_boxplot(alpha=0.1)
# con el dataset reducido: "jittering", añadiendo ruido aleatorio para una mejor
# visualización y aliviar el "overplot": varios vectores "montados" sobre el mismo punto.
# El (aesthetic) color nos ofrece una tercera dimensión
t = ggplot(dsmall,aes(clarity,price))
t + geom_point(aes(colour=cut))
t + geom_jitter(aes(colour=cut))+ggtitle("Dataset Diamantes: clarity vs. price (jitter)")
# segmentando el scatterplot de dos variables numéricas (``carat", ``price")
# en base a los valores de una variable nominal (``clarity")
u = ggplot(dsmall, aes(carat, price))
u + geom_point() + facet_wrap(~clarity,nrow=2)
```

Posiblemente el tipo de capa-layer más popular es “geom_XXX” (tipo de objeto). Hay múltiples webs describiendo los tipos de capas y sus opciones. La siguiente es una, describiendo múltiples opciones para “añadidos”-capas al gráfico: geom, stat, scale...

4 El paquete `googleVis`: un interfaz en `R` hacia GoogleCharts

Las siguientes líneas son un resumen personal acerca del material de `googleVis` en [2, 1].

`googleVis` ofrece un interfaz entre `R` y la API de Google Charts, dándonos acceso a una interesante galería de gráficos interactivos ofrecidos por Google. Data frames que contienen los datos se convierten a objetos del popular formato JSON (JavaScript Object Notation), y los gráficos se muestran en navegadores de manera local. Así, el gráfico generado es código `html` que se podría embeber en webs como Google Sites y Blogger, en el popular paquete `shiny` de `R`, en presentaciones de PowerPoint o Impress, etc.

Tal y como admiten los autores del paquete [1], la configuración de parámetros en `googleVis` es algo “engorrosa”. Aún así, una serie de reglas generales son aplicadas:

- el nombre de la función de visualización comienza por “gvis”, seguido del tipo de gráfico, e.g. `gvisMap`, `gvisPieChart`, `gvisHistogram`;
- el primer parámetro en las funciones de visualización es el objeto data frame a mostrar;
- posteriormente, cada función de visualización tiene parámetros propios que deben ser instanciados. Por jemplo, en la función `gvisLineChart` se debe dar valor a los parámetros `xvar` y `yvar`, relacionados con los vectores que contienen las categorías-etiquetas y valores numéricos a plotear, respectivamente;

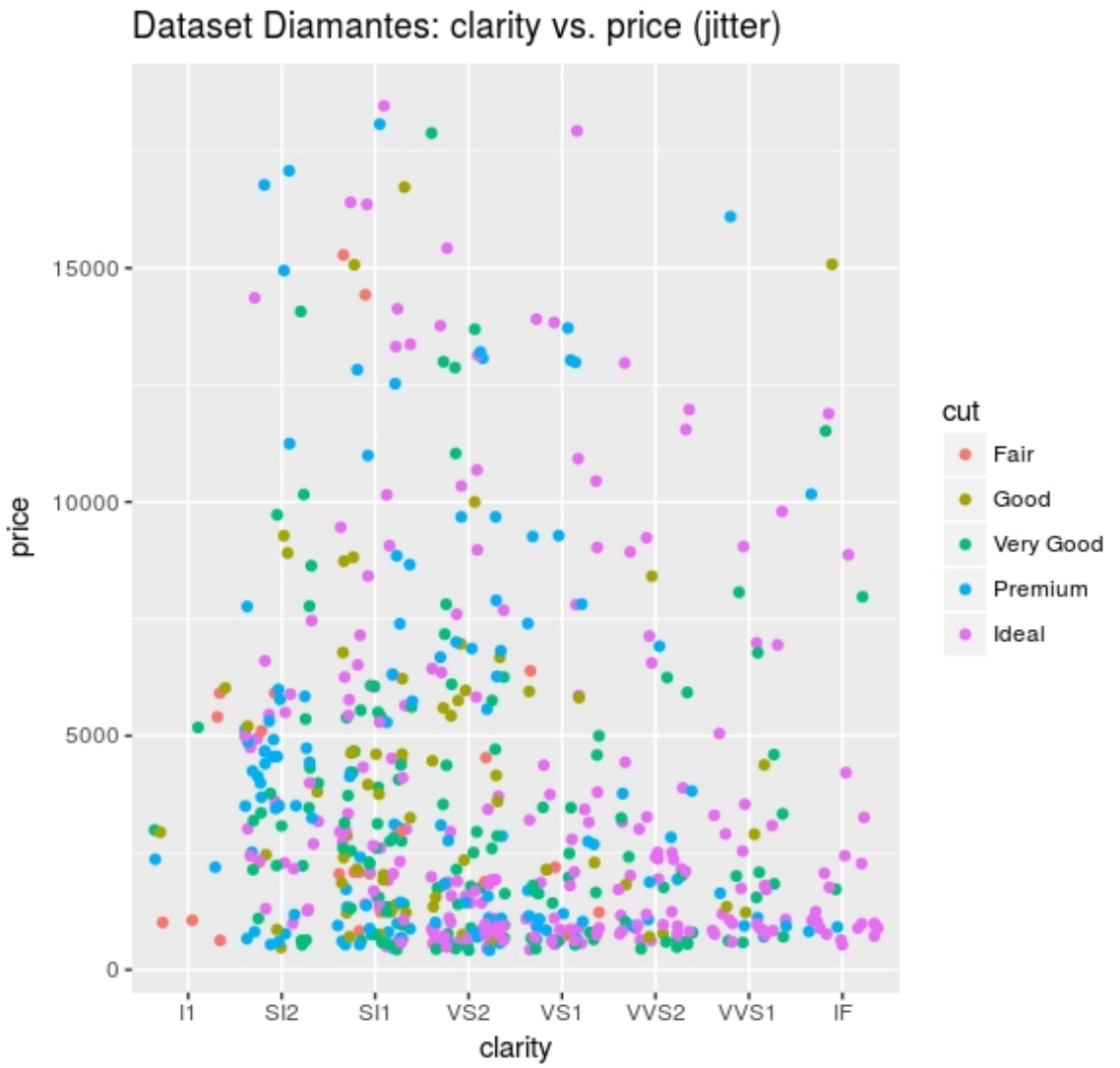


Figura 6: Dataset Diamantes: clarity vs. price. "Jittering" de puntos, añadiendo ruido aleatorio para una mejor visualización y aliviar el "overplot". El color nos ofrece una tercera dimensión

- el último parámetro en las funciones de visualización `options`, recoge en una lista opciones específicas para éste. Esta lista de opciones puede ser extensa y engorrosa. Las siguientes reglas son de aplicación en esta parametrización:
  - las opciones sencillas se fijan mediante `name=value`, e.g. `heigth=300`, `region='US'` or `title="Hello World"`;
  - parámetros que permiten varios valores se fijan mediante corchetes, e.g. `colors="['red','blue']"`;
  - parámetros compuestos por una serie de opciones sefijan mediante corchetes, e.g. `titleTextStyle="{color:'red',fontSize:16}"`;

Algunos de los tipos de gráficos ofrecidos por `googleVis` también es ofrecido, obviamente con otro "outlook", por otros paquetes de `R`. De la amplia gama de gráficos ofrecidos por `googleVis`, nos centraremos en aquellos genuinos-singulares del paquete:

La función `gvisGeoMap` crea un mapa de un país, continente o región. Aparte del data frame que contiene los datos a visualizar, la función tiene estos tres parámetros obligatorios:

- `locationvar` es el nombre de la columna con las localizaciones a mostrar. Aparte de los más comunes, países, regiones y nombres de ciudades, las localizaciones ISO-3166-1 standard codes pueden utilizarse. El formato 'latitude:longitude' también puede usarse;
- `numvar` es la columna con los valores numéricos a ser mostrados;
- la `hovervar` recoge un string adicional que es dinámicamente mostrado al pasar con el ratón sobre dicha región en el mapa.

Entre la larga lista de parámetros destacaremos `region` (la zona a mostrar, siguiendo el expuesto standard ISO-31661-1), `dataMode` (coloreando una región) y `colors` (un RGB number en la paleta de colores del formato 0xRRGGBB).

> If the map is not displayed in your default web-browser (e.g Chrome), copy the url-address and paste it in other web-browser (e.g. FireFox).

```
ciudad=c("Lisboa","Porto","Coimbra")
poblacion=c(2821697,2459045,142396)
atraccion=c("Lisboa: capital del pais", "Oporto: pulmon economico",
            "Coimbra: ciudad universitaria")
portugal=data.frame(ciudad, poblacion, atraccion)

G1=gvisGeoChart(data=portugal,locationvar='ciudad', colorvar='poblacion', hovervar='atraccion',
                options=list(region="PT", height=350, displayMode="regions",
                resolution="provinces"))

plot(G1)

G2=gvisGeoChart(data=portugal,locationvar='ciudad', colorvar='poblacion', hovervar='atraccion',
                options=list(region="PT", height=350, dataMode="markers",
                displayMode="regions", resolution="provinces",colors="[\'red', \'blue']"))

plot(G2)
```

La atractiva función `gvisMap` permite insertar la visualización de un data frame en los populares Google Maps. Esta función tiene dos parámetros obligatorios:

- `locationvar` recoge las geolocalizaciones en el formato 'latitude:longitude' o mediante la dirección en formato texto;
- `tipvar` se fija al nombre del vector que recoge los string que serán mostrados al pasar con el ratón por encima de las localizaciones en el mapa.

Entre su larga lista de parámetros opcionales destacaremos `mapType`, que recoge el tipo de mapa GoogleMap a utilizar.

```
# three touristic locations in Donostia
#descripcion=c("Paseo del Boulevard", "Paseo y Bahía de la Concha", "Peine del #Viento")
#direccion=c("Boulevard Zumardia, Donostia", "Paseo de la Concha, Donostia",
```

```
#            "Paseo Eduardo Chillida, Donostia")
#atraccionesDonostia=data.frame(descripcion,direccion)
#M1 <- gvisMap(atraccionesDonostia,locationvar="direccion",tipvar="descripcion"#,                    options
#plot(M1)

# tránsito del huracán "Andrew". Fíjate en las columnas del dataframe "Andrew"
names(Andrew)
head(Andrew)
M2 <- gvisMap(Andrew, "LatLong" , "Tip",options=list(mapType='hybrid'))

plot(M2)

# el paquete también nos ofrece visualizar un dataframe en el navegador
tableAndrew <- gvisTable(Andrew, options = list(width = 600, height = 650,
plot(tableAndrew)
```

La función `gvisBubbleChart`, a partir de los vectores-variables ofrecidos en un dataframe, crea la llamada "bubble chart". Su principal singularidad es que mediante ella es posible mostrar hasta 4 dimensiones. Mientras las dos primeras son las coordenadas (`xvar`, `yvar`), el color y el tamaño nos ofrecen otras dos dimensiones mediante los parámetros `colorvar` y `sizevar`, respectivamente.

```
G3 <- gvisBubbleChart(Fruits, idvar="Fruit", xvar="Sales", yvar="Expenses",
                           colorvar="Location", sizevar="Profit",
                           options=list(title="Fruit market in USA: bubble size by profit",
                                        hAxis='{title:"Sales"}', vAxis='{title:"Expenses"}'))
plot(G3)
```

5 Reducir dimensiones y visualizar en 2D: `PCA` versus `t-SNE`

La visualización de datos de alta dimensión, ubicando cada punto-vector en 2 dimensiones, es un reto que atrae la atención del mundo académico y empresarial. Una atractiva visualización sabemos que puede atraer la atención y aumentar la confianza de compañeros y directivos no-expertos en el análisis de datos.

Para ello, muchos de nosotros conocemos una herramienta como la `PCA, Principal Components Analysis`. Su enlace en Wikipedia ya nos ofrece un rico resumen. Se construyen unos nuevos ejes (i.e. principal components), combinaciones lineales de las variables originales, de tal manera que la varianza de mayor tamaño del conjunto de datos es capturada en el primer componente, la segunda varianza más grande es el segundo eje, y así sucesivamente. Las componentes, entre sí, no-correladas, ortogonales.

En caso de que las dos primeras componentes recojan una parte sustancial de la varianza original de la muestra (mayor del $90-95\%$), está 'validado' el poder visualizarlos en estas dos nuevas dimensaiones.

La principal limitación de la PCA es que sólo tiene en cuenta combinaciones lineales de las variables originales para los nuevos ejes creados-extraídos. Así, en múltiples dominios, el no poder considerar combinaciones no-lineales puede acarrear perder información esencial y no poder capturar la variabilidad de la muestra en las nuevas variables lineales y en la visualización correspondiente.

```
summary(iris)
# PCA sólo desde datos numéricos;
# centrado de variables en media 0, y varianza unitaria
```

```r
pca_iris <- prcomp(iris[1:4], scale = TRUE)
# coeficientes en la expresión lineal de cada PC
# e.g. PC1 = 0.521 SepalLength - 0.269 SepalWidht + ...
pca_iris$rotation
summary(pca_iris)
# "permitido" realizar el plot2D;
# los dos primeros PCs recogen más del 95% varianza total;
# plot 2D: PCs en ejes y dirección de variables originales sobre éstas
biplot(pca_iris, scale=0, col = c("blue", "red"))
# plots elegantes con paquete "ggfortify" -- basado en ggplot2
install.packages("ggfortify")
library(ggfortify)
library(ggplot2)
autoplot(pca_iris)
# "colorear" las clases
autoplot(pca_iris, data = iris, colour = 'Species')
# plot sin puntos y con índices casos
autoplot(pca_iris, data = iris, colour = 'Species', shape = FALSE, label.size = 3)
```

Como alternativa a PCA, durante los últimos años ha irrumpido con fuerza t-SNE, 't-Distributed Stochastic Neighbor Embedding' [9]. Mientras que PCA trata de preservar la estructura global de los datos mediante la captura de la varianza, t-SNE trata de preservar la estructura local de patrones-clusters en la muestra: Y en contraste a PCA, ésto lo realiza mediante una transformación no-lineal. En escenarios donde PCA no es capaz de capturar un nivel significativo de la varianza, t-SNE surge como una alternativa natural.

La intuición de su algorítmica, aún más compleja que la PCA, trataré de resumirla. Un espacio Euclideo de alta dimensión es transformado a *probabilidades condicionales que representan similitudes entre puntos*. El nivel de similaridad entre una pareja de puntos $\{\mathbf{x_i}, \mathbf{x_j}\}$ se representa mediante la probabilidad de que $\mathbf{x_i}$ escoja a $\mathbf{x_j}$ como vecino, siendo éstos escogidos en proporción a la probabilidad condicionada $p_{j|i}$ otorgada por una Gaussiana centrada en $\mathbf{x_i}$.

Pero nosotros estamos interesados en las 'versiones 2D' de nuestros puntos originales. Sean estas versiones $\{\mathbf{y_i}, \mathbf{y_j}\}$. Y entre ellos, la probabilidad de que uno escoja al otro como vecino, mediante la función de probabilidad condicionada $q_{j|i}$.

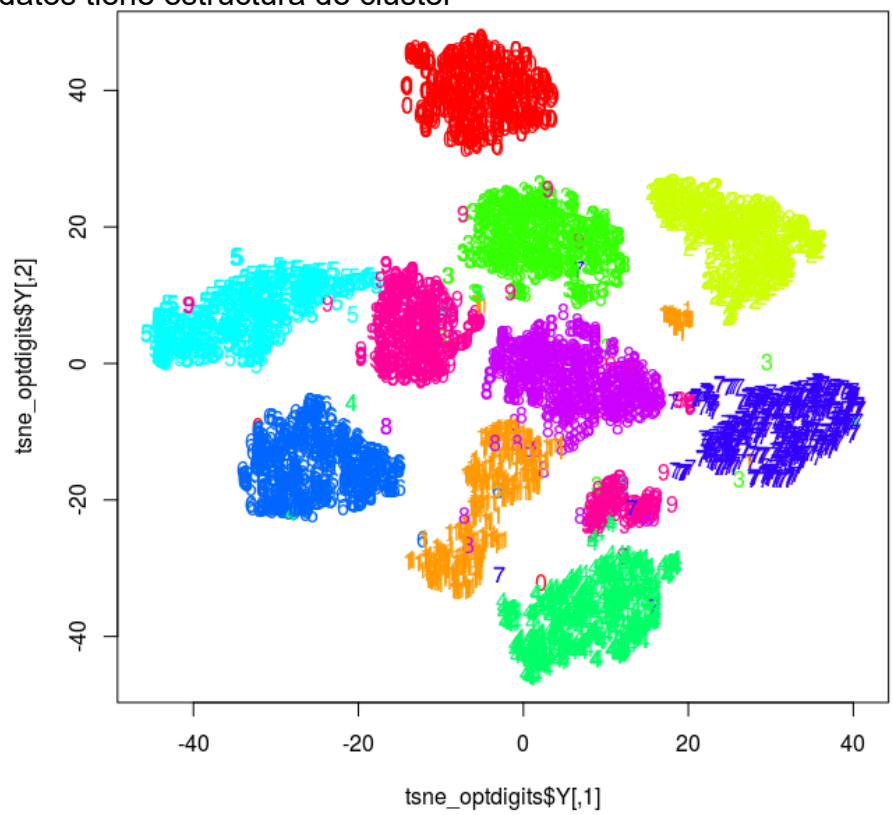
Así, t-SNE trata de encontrar, la representación en 2D de los datos que minimiza la distancia entre las expuestas $p_{j|i}$ y $q_{j|i}$. La divergencia de Kullback-Leibler es una manera natural y popular de medir el nivel de parecido entre dos distribuciones de probabilidad. t-SNE tiene el objetivo de minimizar la suma de las divergencias Kullback-Leibler sobre todos los puntos. La representación 2D que la propone, es la que nos interesa visualizar.

Practiquemos. El popular dataset 'optdigits', con 5620 observaciones-digitos escritos a mano, resumidos en 64 dimensiones y etiquetados con el valor del dígito. Un dominio no-lineal, complejo, en el que las dos primeras componentes no capturan más allá del 30% de la varianza global. Cálculos computacionalmente costosos.

```r
optdigits= read.csv("optdigits_csv.csv",header=TRUE,sep=",")
library(Rtsne)
# función Rtsne -- consulta su ayuda y parámetros en rdocumentation.com;
# https://www.rdocumentation.org/packages/Rtsne/versions/0.15/topics/Rtsne;
# los parámetros que utilizo -- indispensables para ejecución eficiente -- consúltalos
tsne_optdigits <- Rtsne(optdigits[,1:64], check_duplicates = FALSE, pca = FALSE, dims=2)
cols <- rainbow(10)
```

```
plot(tsne_optdigits$Y, t='n')
# coloreando los puntos con su dígito-etiqueta
text(tsne_optdigits$Y, labels=optdigits[,65], col=cols[optdigits[,65] +1])
# sorprendente resultado, compacta estructura de grupos;
# partiendo de 64D, meritorio, ¿no? Compara los grupos y los "colores"...
```

Para visualizar outliers despues de hacer outlier detection
O para ver si los datos tiene estructura de cluster



Los ejes no tienen representacion matematica

Figura 7: Optdigits dataset. Visualización en 2D tras reducción de la dimensionalidad mediante tSNE. Puntos coloreados con el dígito de la muestra.

Bibliografía

[1] M. Gesmann and D. Castillo. *Using the Google Chart Tools with R: googleVis-0.5.8 Package Vignette*, 2015.

[2] Markus Gesmann and Diego de Castillo. Using the Google visualisation API with R. *The R Journal*, 3(2):40–44, 2011.

[3] R.I. Kabacoff. *Quick-R. Lattice Graphs.* http://www.statmethods.net/advgraphs/trellis.html.

[4] D. Sarkar. *Lattice. Multivariate Data Visualization with R.* Use R! Springer.

[5] L.S. Shapley. A value for n-person games. Technical report, Princeton University, 1953. vol II of Contributions to the theory of games.

[6] H. Chen A. DeGrave J.M. Prutkin B. Nair R. Katz J. Himmelfarb. N. Bansal S.M. Lundberg, G. Enrion and S-I. Lee. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, 2:56–67, 2020.

[7] E. Strumbelj and I. Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41:647–665, 2014.

[8] N. Sturaro. *Rabbit. Introduction to R.* Quantide. http://www.quantide.com/R/r-training/r-web-books/rabbit-introduction-to-r/.

[9] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[10] H. Wickhman. *ggplot2. Elegant Graphics for Data Analysis.* Use R! Springer.